PROYECTO DE GRADO

Presentado ante la ilustre Universidad de los Andes como requisito final para obtener el Título de INGENIERO DE SISTEMAS

DESARROLLO DEL MÓDULO TREN-DE-CLIENTES DE LA ACELERADORA VIRTUAL IGNIS GRAVITAS INC. EN RUBY ON RAILS

Por

Br. Paul David Barreto Torres

Tutor: Dr. Gérard Páez Monzón

Junio 2024

Desarrollo del Módulo Tren de Clientes de la Aceleradora Virtual Ignis Gravitas INC. en Ruby on Rails.

Br. Paúl David Barreto Torres.

Resumen

La presente investigación tuvo como objetivo principal el Diseño e Implementación del Módulo Tren de Clientes en la Aceleradora Virtual Ignis Gravitas Inc. en Ruby on Rails con el propósito de integrarlo en el área local (espacio de desarrollo) de la plataforma y así darle la funcionabilidad requerida para que los emprendedores/innovadores puedan desarrollar sus proyectos de manera inteligente y satisfacer a sus clientes. La investigación se desarrolló bajo la modalidad de proyecto factible por ser un modelo operativo viable para solucionar problemas, sustentado en una investigación de campo y un estudio documental. Se estructuró en tres fases y el desarrollo e implementación del módulo TOC se realizó bajo la estrategia ideada por los fundadores de la plataforma, plasmada en el documento de requisitos, en la línea de responsabilidades. Consiste en ocho estaciones por las cuales deben transitar los emprendedores hasta lograr consolidarse como empresa; el módulo fue desarrollado en Ruby on Rails framework para el desarrollo de aplicaciones web app con base en el patrón Modelo-Vista-Controlador. Se obtuvo como resultado el despliegue e implementación exitosa de cada una de las estaciones que conforman la estrategia, logrando de esta manera la integración del módulo TOC en plataforma dando cumplimiento a los objetivos planteados en la presente investigación.

Palabras Claves: módulo, aceleradora virtual**,** ruby on rails, emprendimiento, innovación.

INDICE

INDICE DE CUADROS

INDICE DE FIGURAS

INTRODUCCIÓN

El emprendimiento ha ido evolucionando de la mano de las nuevas tecnologías y el internet, revolucionando la sociedad en general y la forma en que las organizaciones deben orientarse en el mercado, frente a los cambios y en relación directa a las necesidades de los clientes. En la actualidad existen plataformas tecnológicas cuyo propósito es promover la cultura del movimiento emprendedor, incorporando cada día a más personas y por diversas razones. La utilización de la innovación tecnológica se ha vuelto fundamental en los procesos empresariales. Gracias a las nuevas tecnologías, el emprendimiento se presenta bajo el concepto de startups, propuesta de mercadeo a gran escala.

Las startups son empresas instauradas recientemente, suelen ser creadas y administradas por emprendedores a través de herramientas tecnológicas innovadoras; es decir, son empresas de nueva creación, escalables que buscan aumentar su magnitud de ingresos en periodos de tiempos cortos, sin que esto conlleve un aumento de sus gastos, su objetivo es crecer rápidamente. También se les conoce como aceleradoras virtuales. Entre sus principales características cabe destacar que no se encuentran limitadas geográficamente y desarrollan su modelo de negocios en la innovación. El emprendedor innova mediante la introducción de nuevas ideas útiles mediante estas tecnologías que buscan mejorar la vida de las personas, bien sea a través del desarrollo de un producto o el ofrecimiento de un servicio.

La innovación, entonces, es elemento clave de la actividad emprendedora dentro de las startups; sin embargo, así como estas pueden ser empresas exitosas, también conllevan un riesgo: la incertidumbre a perderlo todo y quedarse en el camino, desaparecer. Según estudios, (Bednar, R y Tariskova, N, año 2, número 5) y (Network, 2023), 9 de cada 10 startups terminan fracasando y entre el 75% - 90% de las startups desaparecen antes de completar el cuarto año de vida. Estos porcentajes deben mantenernos alertas e indican que no todos las startups, aun contando con la idea del emprendedor y las herramientas tecnológicas, sobreviven. Las razones del fracaso son varias, vale destacar: 1.- No existe mercado; es decir, lanzamiento de un producto sin existir una clara necesidad o demanda de este. 2,- Creación de un producto poco atractivo e ignorar a los clientes.

La plataforma Ignis Gravitas Inc. Es cimiento de la presente investigación. Emerge de la necesidad de presentar herramientas eficaces a emprendedores que sirvan para tener la organización adecuada en la construcción de productos de innovación como lo son las startups. Fue diseñada para nutrirlas desde la formación de la idea inicial pasando por el escalado de operaciones, hasta el asentamiento como empresa. El producto y modelo de negocio de la startups dentro de la plataforma de Ignis Gravitas Inc, se configura con pautas aprendidas de éxitos y fracasos de anteriores procedimientos. Plantea transformar la idea en soluciones directas de acuerdo a las necesidades y deseos reales de los clientes e inducir la colocación adecuada de productos de innovación. Para ello, crea dentro de sus espacios el panel Clientes y dentro de este se desarrolla el módulo tren de clientes utilizando el framework (Ruby on Rails).

De tal manera, el presente estudio tiene como propósito el desarrollo e implementación del módulo tren de clientes en la aceleradora virtual Ignis Gravitas Inc en Ruby on Rails, a través de la estrategia ideada por los fundadores en el documento de requisititos, en el cual se describe la estrategia a desplegarse e implementar: consta de ochos estaciones: observacional, enfoque de grupo, etapa primer cliente, primer clientes, primeros adoptantes, chasm (abismo), escalando y empresa; de tal manera que se integre en la plataforma en el espacio local Panel de Cliente mostrando al emprendedor una visión real del desarrollo de su producto y de su crecimiento en el mercado que este aplicando.

En consecuencia y para dar cumplimiento a los objetivos planteados, este proyecto se estructura de la siguiente manera:

Capítulo I. Planteamiento del problema. Se describe el problema objeto de estudio en su contexto de lo general a lo particular, formulando la pregunta de investigación, se delimitan los objetivos que se persiguen durante la ejecución de la investigación, así como la justificación, y alcances del estudio.

Capitulo II. Marco Teórico: Se narran los antecedentes de la investigación tomando como referencia estudios ya realizados por otros investigadores relacionados con el tema planteado y que tienen relevancia y aportes para la presente investigación; así como las aspectos referentes a las variables de investigación: Modulo Tren de Clientes, Aceleradora Virtual Ignis Gravitas y Ruby

on Rails, bases y conceptos que fundamenta el desarrollo de este trabajo producto de la revisión documental-bibliográfica y se definen términos básicos y basamento legal .

Capitulo III. Marco Metodológico, en este apartado se plantea el "cómo" se realizó el estudio, el camino transitado para responder a la problemática planteada y dar repuesta exitosa a la solución del problema. Se describen los procedimientos: nivel y tipo de investigación, diseño de investigación, población y muestra; técnicas e instrumentos de recolección de datos y técnicas de procesamiento y análisis de datos.

Capitulo IV. Se presenta el análisis y resultados de la investigación planteados en fases que son: Fase 1, exploración (mapeo del ecosistema de emprendimiento de Ignis Gravitas y sondeo de líneas de códigos y rutas del área local para levantar el proyecto); fase 2, desarrollo e implementación de cada una de las estaciones que conforman la estrategia del Tren de Clientes utilizando el framework Ruby on Rails, se detallan hallazgos y se muestran datos arrojados para su análisis y resultados.

Capitulo V. Implementación y Despliegue, se describe detalladamente el proceso para llevar a cabo el funcionamiento de cada una de las estaciones del Tren de Clientes, así como su demostración.

Conclusiones y Recomendaciones. Se narra las reflexiones y análisis derivados de los resultados obtenidos en función de la interrogante planeada y los objetivos de la investigación; así como se plantean sugerencias de acuerdo a los resultados obtenidos en la investigación.

Finalmente, se indican todas las Referencias utilizadas en el presente proyecto de investigación y anexos que incluyen información complementaria y relevante de la investigación.

CAPITULO I

PLANTEAMIENTO DEL PROBLEMA

1.1.El Problema

Hoy por hoy, se percibe que en muchas personas se ha despertado el espíritu de emprendimiento e innovación. Innovar es un elemento clave, pero es muy complejo, es un proceso creativo donde se involucran diversas actividades y la conversión de ideas para generar una ventaja competitiva o un beneficio que brinde el mejor servicio o producto a las partes interesadas, entre ellas, la más importante: el cliente (Diaz, 2018). En este sentido, se infiere que el cliente directa o indirectamente se ha relacionado con el uso de las tecnologías, en búsqueda de un proceso creativo garantizando que las empresas le ofrezcan un servicio o producto actualizado y único.

En tal sentido, se puede indicar que la innovación significa crear algo introduciendo una novedad, cambiar algo, cambiar una costumbre para ser exitoso; mientras, emprender significa comenzar con un proyecto donde se desarrolla una idea de producto o servicio a través de un negocio destinado a cubrir una necesidad. De allí, cuando los emprendedores emplean la innovación a sus emprendimientos, generan avances tecnológicos y nuevas formas de hacer las cosas.

En consecuencia, el emprendedor aborda la tecnología recurriendo al internet y a las plataformas tecnológicas de emprendimiento en startups, como un medio de crecer y darse a conocer dentro del campo de su producción, buscando la manera de mejorar sus procesos, servicios y satisfacer las necesidades del cliente. Un buen servicio al cliente es calificado óptimo/ impecable si solamente este logra satisfacer las necesidades y deseos del cliente.

Sin embargo, se observa la existencia de muchas plataformas virtuales de emprendimientos en las que las startups se desarrollan jugando un papel importante en el sistema económico y en el desarrollo tecnológico de cualquier país; no obstante, fracasan. Señala (Fernández Esteban, 2018) que en investigaciones realizadas por la consultora CB Insights entre los motivos por la cual fracasan son los siguientes: tiempo de lanzamiento del producto, ignorar a los clientes, mala comunicación, falta de modelo de negocio, crear un producto poco atractivo, mala relación costo

precio, ignorar a la competencia, no contar con el equipo adecuado. Agrega que algunos de estos patrones fueron motivos comunes por los cuales empresas de joven recorrido cerraron sus emprendimientos, siendo que una de las razones es "ignorar a los clientes" patrón que representa el 14% del fracaso de las startups. Ademá, indica que los esfuerzos del área correspondiente deben concentrarse hacia las necesidades del cliente, no dejar pasar mucho tiempo entre las primeras ideas y las entrevistas con los clientes a fin de que el servicio otorgado y recibido sea lo que el cliente realmente necesita, por lo que pierde la economía, el recurso y se cierran las soluciones.

Lo anteriormente descrito, actualmente representa un problema para emprendedores e innovadores, y es por ello que la plataforma de emprendimiento de Ignis Gravitas Inc, con base en la experiencia e investigaciones científicas que ha realizado, proyecta soluciones. La plataforma integra actividades de emprendimiento en forma organizada, planificada, útiles para los emprendedores, ofreciendo un módulo que contiene el paquete de requerimientos del producto: Tren de Cliente (TOC), en la línea de responsabilidades; el cual consta de ochos estaciones: observacional, enfoque de grupo, primer cliente, primer grupo de clientes, primeros adoptantes, chasm (abismo), escalando y empresa. Con la implementación del módulo lo que se busca es dar funcionabilidad a la plataforma de tal manera, que el emprendedor /innovador pueda moverse de estación en estación con responsabilidad y tome decisiones de forma inteligente, con base a las necesidades de los clientes.

Ahora bien, el problema radica en que actualmente está creado el espacio panel de clientes en la aceleradora virtual de Ignis Gravitas, pero no está desarrollado el TOC en el mismo. De modo que, el presente estudio tiene como propósito el desarrollo e implementación del Módulo Tren de Clientes/ Train of Clients en la aceleradora virtual IGNIS GRAVITAS, INC. Es de acotar que TOC se desarrolla en Ruby on Rails, nombre del framework que utiliza el lenguaje Ruby, permitiendo un desarrollo ágil y rápido de las aplicaciones para startups.

Como resultado de lo anteriormente planteado, surge la interrogante que orienta el desarrollo de la presente investigación: ¿Es posible, una vez desarrollado y presentado el módulo: Tren de Clientes aplicando la estrategia en espiral de ocho estaciones, en la aceleradora virtual de Ignis Gravitas Inc. con Ruby on Rails, ¿se logre dar la funcionabilidad requerida y hacer que los negocios de los emprendedores crezcan de manera inteligente?

1.2.Objetivos de la Investigación

1.2.1.-Objetivo General

Desarrollar e implementar un módulo del Tren de Clientes en la Aceleradora Virtual Ignis Gravitas en Ruby on Rails para fortalecer a los emprendedores / innovadores de forma inteligente e incrementar las probabilidades de éxito en su modelo de negocios.

1.2.2. Objetivos Específicos

- Realizar revisión bibliográfica y del estado del arte sobre las variables a investigar: Tren de Clientes en Ignis Gravitas Inc. y Ruby on Rails.
- Conocer el funcionamiento de la aceleradora virtual Ignis Gravitas Inc. con el propósito de relacionarse sobre el manejo de la misma.
- Investigar la literatura acerca de la utilización práctica de Ruby on Rails para el perfeccionamiento del módulo: Tren de Clientes.
- Definir y describir las ocho estaciones del TOC a implementar como definitivo.
- Desarrollar el módulo de Tren de Clientes en Ruby on Rails aplicando la estrategia en espiral ideada por Ignis Gravitas, Inc.
- Implementar el módulo en la Aceleradora Virtual de Ignis Gravitas; es decir, integrarla en la plataforma

1.3. Justificación

La presente investigación surge de la necesidad de implementar el Módulo Tren de Clientes en la Aceleradora Virtual Ignis Gravitas, Inc. en Ruby on Rails, pues posee una personalidad enfocada hacia el crecimiento de las distintas Startups que hacen vida en la plataforma con el objetivo de proporcionar una visión integral en el desarrollo del proceso emprendedor, que día a día enfrentan las startups. Actualmente dado el alto grado de incertidumbre con el cual subsisten las startups al lanzar nuevos productos y servicios innovadores al mercado, entre un 75% y 90% de ellas fracasan (Neil, 2017). Ignis Gravitas frente a esta situación, con base a la experiencia y estudios científicos, diseña los requerimientos del Tren de Clientes en la línea de responsabilidades, agregándole estaciones por donde tienen que transitar el modelo de negocios del startup. Ignis Gravitas ha desarrollado en gran parte esta estrategia, pero actualmente, no está incluida dentro de su plataforma y es lo que se realiza en esta investigación**,** ya que la plataforma no se encontraba equilibrada en cuanto a las funcionalidades de los clientes. Por lo tanto, hizo necesario la evaluación y desarrollo, para su integración a la plataforma.

La importancia de esta investigación radica en que permite analizar de manera detallada las necesidades de los clientes por estación, apoyando a los emprendedores con herramientas útiles e información experta para despertar y hacerlos accionar de manera tal que se obtenga un producto de innovación que logre engranarse al mercado.

En relación con este planteamiento, se hace necesario precisar que TOC se despliega en Ruby on Rails debido a que la plataforma Volition de Ignis Gravitas, en su espacio de trabajo denominado LAB está construida en Ruby on Rails, por tanto al respecto se precisa que Rails es un framework de desarrollo de aplicaciones web, escrito en el lenguaje de programación Ruby, diseñado para hacer que la programación de aplicaciones sea más fácil, permitiendo que cada desarrollador pueda escribir menos códigos, realizados más que muchos otros lenguajes y framework.

1.4.Alcances

- Formar en Ruby on Rails.
- Instruir en lo que es una plataforma tecnológica como producto de innovación.
- Fomentar la cultura de startups a nivel mundial en los jóvenes emprendedores e innovadores, así como en organizaciones
- Orientar a cualquier ente bien sea persona u organización empresarial acerca de Ignis Gravitas que es una aceleradora autónoma virtual para emprendimientos desde los distintos startups que hagan vida en su plataforma.
- Acompañar y orientar a los emprendedores de innovación en el crecimiento de clientes.
- Lograr que la aceleradora virtual Ignis Gravitas Inc, presente en la plataforma el módulo TOC la cual tiene como objetivo principal apoyar y ser útil a los innovadores / emprendedores en la producción de startups, sin importar el lugar donde se encuentre o el poco manejo en cuanto a los recursos de emprendimiento tengan.

1.5.Delimitación

- La investigación está centrada solo en el desarrollo e implementación del Módulo Tren de Clientes en Ruby on Rails, para dar funcionabilidad y equilibrio a la plataforma, partiendo de la estrategia ideada por Ignis Gravitas contentiva de ocho estaciones que inicia en una primera estación observacional hasta llegar a la octava estación: empresa (el emprendedor como usuario construye su idea y la levanta al punto de llevarla a empresa).

- Espacialmente, el trabajo de investigación se desarrolla en Ignis Gravitas Inc. es un startup inscrito en Delaware, Estados Unidos. Tiene como objetivo la expansión de la cultura emprendedora y el crecimiento de las startups a nivel mundial.

CAPITULO II

MARCO TEÓRICO

Al realizar una investigación, es necesario situar el marco de referencia que orienta el estudio en relación a sus variables y todos sus aspectos, de manera que establecen la visión del problema y muestran el interés del investigador al analizar la realidad objeto de estudio. De ahí que, se presentan los antecedentes de la investigación con afinidad al tema de investigación: Desarrollo del Módulo del Tren -de- Clientes en la aceleradora virtual de Ignis Gravitas Inc. en Ruby on Rails para lograr su implementación en la plataforma de Ignis Gravitas INC. Por otra parte, se exponen las teorías concernientes al tema y conceptos que guardan relación con el problema planteado, encausando las herramientas para conseguir el propósito principal de la presente investigación.

2.1. Antecedentes de la Investigación

Durante los últimos años, se ha desarrollado el movimiento emprendedor en el mundo y en Venezuela se ha propiciado aún más, quizás por la crisis económica, surgiendo la necesidad de reinventarse. Se ha hecho común oír el término "startup" para referirnos a empresas de nueva creación. El mundo de la empresa y emprendimiento han tenido la necesidad imprescindible de utilizar la innovación tecnológica en sus procesos empresariales. La tecnología ha tenido un papel disruptivo y ha marcado el inicio de la revolución digital; surgiendo las startups como empresas diseñadas para crecer rápido en busca de un negocio de modelo rentable, con ideas innovadoras que puede repetirse y hacerse más grande.

En el marco de estas innovaciones nace IGNIS GRAVITAS INC., buscando implementar herramientas eficaces para tener la organización adecuada en la construcción de productos de innovación como lo son las startups. Estos nuevos emprendimientos tienen muchas interrogantes las cuales la plataforma trata de responder y cubrir. Ha sido desarrollada con la finalidad de impulsar la creatividad de los emprendedores y ofrecer un conjunto de herramientas capaces de facilitar el trabajo a seguir para tener un producto de calidad en el mercado y de manera inteligente, planteándose estar en constante evolución para presentar nuevos modelos de negocio, tomando en cuentas todos las aspectos, entre ellos las necesidades de los clientes a través de estrategias, permitiendo a los emprendedores innovar y crecer, por lo que incluye en su modelo de negocios el

módulo tren-de-clientes, el cual se ejecuta dentro de su plataforma de emprendimiento utilizando la estrategia que consiste en ocho estaciones creadas por sus fundadores.

Partiendo de lo antes descrito, se tomaron en cuenta como antecedentes que aportan aspectos de relevancia al presente estudio las siguientes investigaciones:

Las Aceleradoras como Trampolín de "Startups". Trabajo de grado presentado en la Universidad politécnica de Valencia-España, cuyo objetivo consistió en analizar el sector emergente de las Aceleradoras de Startups, entendidas éstas como las plataformas de apoyo e incubación de nuevos startups. Con el fin de evidenciar, a través de la práctica de casos concretos, el modelo de negocio y los procesos concretos desarrollados por esta nueva tipología de empresas. Por otra parte, se analizan dos modelos de aceleradoras distintos aplicando el modelo de (NESTA, 2014). El método de investigación utilizado es descriptivo– explicativo. Concluyendo que: los programas de aceleración de startups generan riqueza al impulsar la creación de nuevas empresas. Igualmente, favorecen la velocidad e intensidad, los inversores reciben las propuestas de negocio o las ofertas de inversión y, en última instancia, impulsan la creación de nuevos puestos de trabajo. De igual modo, las aceleradoras de startups también favorecen la velocidad e intensidad a la que los inversores reciben las propuestas de negocio o las ofertas de inversión. Además, se puede decir que las aceleradoras de startups son un sector emergente. Este hecho complica la medición del impacto de las aceleradoras de startups, pero todavía falta mucho trabajo de recopilación de datos para evaluar el éxito de las aceleradoras y sus startups. (Gimeno, 2018).

El estudio antes descrito aportará a la presente investigación, conocimientos teóricos y bibliográficos sobre los programas de aceleración de startups y la velocidad e intensidad que los inversores reciben las propuestas de negocio o las ofertas de inversión, impulsando la creación de nuevos puestos de trabajo.

Estudio y Análisis de la Herramienta Opensource Ruby on Rails orientado al desarrollo de aplicaciones Web versus las demás Herramientas actuales. Tesis de grado presentado para optar al título de Ingeniero de Sistemas Computacionales en la Universidad de Guayaquil - Ecuador**.** Esta investigación tuvo como objetivo general, demostrar de manera teórica - práctica las ventajas y beneficios del uso de la herramienta para el desarrollo web, creando un portal web usando el framework Ruby on Rails, con la finalidad de que esta herramienta facilite de manera ágil la codificación y mejora los tiempos de desarrollo. Como método realizó comparativas

con algunos frameworks que están al mismo nivel, entre ellos, Zend, Django y Jsf, y cuadros estadísticos que permiten al lector tomar una decisión sobre cuán útil es la herramienta. Para esto, se realizó una investigación sobre las principales funciones y características de Ruby on Rails, además de listar las principales funciones de los frameworks, realizaron investigaciones de tipo bibliográficas y de comprobación de hipótesis. La modalidad de la investigación es bibliografía, presentando como conclusiones que se la herramienta Ruby on Rails, tiene mucho por ofrecer en el área del desarrollo, facilitando la agilidad en la codificación y en las interacciones con la base de datos. A nivel incremental se muestra cuán importante son las características que se describieron para el uso de un frameworks. En relación al análisis comparativo, mostraron cuáles son los pro y contras de los frameworks de acuerdo a los criterios de comparación, para que el futuro lector pueda tener una idea acerca de cuál herramienta utilizar. (Zambrano, 2010).

Este trabajo será de gran utilidad, tanto a nivel teórico como práctico, para la presente investigación. Muestra la practicidad de usar el framework Ruby on Rails, señalando cuáles son las fortalezas que permiten un óptimo rendimiento, y proveen una ayuda para la creación de aplicaciones sencillas y útiles. Además, promueven su estudio e indican las ventajas obtenidas al utilizar frameworks que disminuyan el tiempo de desarrollo, dando paso para que los proyectos no tomen más tiempo, garantizando seguridad, escalabilidad y concurrencia

La Tecnología, una Herramienta de Servicio al Cliente. Trabajo de Grado presentado en la Universidad de Nueva Granada-Bogotá, para optar al título de Especialista en Alta Gerencia. Este trabajo, tuvo como objetivo principal analizar la importancia del uso e implementación de la tecnología en las compañías para conocer si se están desarrollando correctamente, con el propósito de señalar que el uso de la tecnología debe ir siempre de la mano de la inteligencia del hombre; su aplicación como herramienta mejorará los procesos de atención y servicio al cliente, pero debe ser un apoyo a la inteligencia humana y no viceversa. El trabajo se desarrolló en una línea de investigación orientada en la calidad del servicio, la cual está enfocada en una evaluación de los procesos aplicados y orientados en la satisfacción del cliente, como ente central de toda compañía, motivado por el uso eficiente de la tecnología en las áreas de atención y servicio al cliente dentro de una organización. Esta investigación permitió concluir: la tecnología no está siendo utilizada con el fin para el que fue creada. Cuando se está frente de una compañía con una gran infraestructura tecnológica, con diferentes canales de atención, pero ninguno funciona, es porque

no existe una excelente estrategia de servicio al cliente. Esto se debe a que la tecnología como todo, tiene características positivas y negativas para los clientes, pero todo depende de la experiencia que cada uno tenga. Por consiguiente, lo importante no es solo estar a la vanguardia de las nuevas tecnologías, sino el uso eficiente y eficaz que estas nos puedan brindar con el fin de fidelizar los clientes de la compañía y, estos a su vez, generen nuevos clientes. (Bueno, 2021).

La contribución de este trabajo de investigación consiste en los conocimientos teóricos promovidos, destaca la importancia que tiene la tecnología hoy día para emprendimientos y los usuarios de la misma. Por otra parte, hace énfasis en la aplicación eficiente y eficaz de herramientas tecnológicas y puedan ser utilizadas de forma correcta y proporcionar una adecuada calidad de servicio al cliente.

Análisis y Desarrollo del Motor de Conocimientos de una Plataforma Tecnológica. Trabajo de Grado presentado en la Universidad de los Andes Mérida -Venezuela, para optar al título de Ingeniero de Sistemas. Este estudio tuvo como objetivo general: Analizar de manera detallada el sistema de la plataforma de emprendimiento IGNIS GRAVITAS INC., con la finalidad de identificar y proponer soluciones a las deficiencias que se presenten en la plataforma para lograr el desarrollo de un motor de aprendizaje con las especificaciones de todas las herramientas para su uso. Su delimitación fue la plataforma IGNIS GRAVITAS INC., con el análisis detallado de lo desarrollado hasta el momento en ella, el funcionamiento que tiene, dónde se encuentran fallas en los procesos que realiza, y qué falta por desarrollar, que beneficios y posibles deficiencias esta posee. Para el análisis se utilizó la guía para el diseño de plataformas "Platform Design Toolkit, el cual ayudo a desarrollar y ejecutar estrategias de plataformas que empoderen y movilicen ecosistemas. El método de investigación fue de tipo analítico e investigativo, obteniéndose las siguientes conclusiones: con la aplicación del kit de herramientas para el diseño de plataformas, se logró estructurar la situación actual de la problemática en cuanto a las deficiencias que existen en ella y sustentar para el planteamiento de posibles soluciones que se proponen en la investigación. Se comprobó que la principal deficiencia presentada en la plataforma está en la administración de la información y los conocimientos que deben tener los usuarios para el uso adecuado y entendimiento de todas sus herramientas. Por otra parte, se logró estructurar un modelo cultural de emprendimiento para todos los usuarios que hacen vida dentro de la plataforma, en donde se aplican tres espacios de importancia que debe tomar en cuenta el emprendedor: el espacio Creatividad,

espacio Startup y el espacio Empresa. De esta forma, podrán tener un conocimiento apropiado de lo que deben hacer antes, durante y al completar el desarrollo con su startup. Con la intensión de fortalecer el sistema de aprendizaje que esta aplicado en la plataforma, se obtuvo el planteamiento de un Motor de conocimiento más eficiente para dar la información de uso de las herramientas que están desarrolladas dentro de esta plataforma. Se realizó la documentación de un manual de usuario, con toda la información a detalle de cada una de las herramientas, construido de forma sinóptica de las vistas de los paneles de la plataforma, para poder aprender y entender la finalidad cultural representada en Ignis Gravitas. (Castillo, 2020).

El proyecto descrito significa uno de los mayores aportes a la presente investigación, sus planteamientos y hallazgos están involucrados directamente con la delimitación de este estudio sobre la plataforma Ignis Gravitas Inc. Sus aportes permitirán obtener una visión teórica de la formación y desarrollo de la misma; así como, revisar las estrategias desarrolladas en la plataforma y lo más valioso, su aporte información para la creación del manual de usuario, el cual contiene la información de las vistas de paneles que conforman la plataforma con información imprescindible para los usuarios que interactúan.

2.2. Contexto Organizacional.

2.2.1. Ignis Gravitas Inc.

Es un startup registrado en Delaware, Estados Unidos. Ha sido desarrollada con la finalidad de promover la creatividad de los emprendedores y ofrecer un conjunto de herramientas capaces de facilitar el trabajo a seguir para tener un producto de calidad en el mercado. Su objetivo es la expansión de la cultura emprendedora y el crecimiento de las startups en todo el mundo. Ignis Gravitas Inc., cumple su misión por medio de dos canales: el primero, la vinculación con el mundo emprendedor a través de la plataforma tecnológica y, el segundo, la estructuración de una propuesta de valor complementaria como es la Academia Ignis Gravitas, donde se forman todos los interesados en el emprendimiento con el objetivo de promover una nueva alternativa en la educación, adaptada a las nuevas demandas del mundo emprende.

2.2.2. Misión.

La plataforma Ignis Gravitas Inc., tiene como cometido lograr convertirse en una aceleradora virtual mundial para el emprendimiento y de esta forma apoyar de manera experta a cualquier miembro de este ecosistema, en especial a los emprendedores dedicados a la innovación en empresas de crecimiento rápidas, conocidos como Startup. Teniendo como responsabilidad llevar a estos emprendedores paso a paso por todos los procesos que se pueden realizar en ella y darles la motivación necesaria para que tomen acción dentro de la plataforma y sean capaces de manejar los emprendimientos que se desarrollen en ella como si fueran expertos, sin importar la experiencia que se tengan.

2.2.3. Visión.

Hacer que la plataforma "Ignis Gravitas Inc.", tenga accesibilidad a todos los emprendedores e innovadores a nivel mundial. Está compuesta de una zona "Ignis", que es el fuego que debe tener todo emprendedor o startup para accionar su emprendimiento como lo haría cualquier experto, y así lograr tener una propuesta real para la construcción de un producto o servicio. Se apoya a los emprendedores con herramientas útiles, para que obtengan un producto de innovación que pueda ser exitoso, enseñándoles que necesitan de un buen modelo de negocios adaptado al producto que quieren construir, un equipo de trabajo, una infraestructura, además que deben organizar la información importante que tengan para poder atraer inversionistas, obtener apoyo de especialistas y demás. La plataforma de emprendimiento Ignis Gravitas Inc. tiene el objetivo de escalar mundialmente y poder ofrecer herramientas eficaces para lograr todo esto dentro de ella. La zona "Gravitas" se encuentra en proceso de construcción y a través de ella se ofrecerá el apoyo de intercambio de valores entre todos los miembros de la comunidad Ignis Gravitas Inc.

2.2.4. Modelo Estratégico

Ignis Gravitas Inc. plantea como modelo estratégico tres espacios donde se desenvuelve la cultura de emprendimiento y donde los usuarios pueden definir una estrategia adecuada para la

creación, desarrollo y evolución de su idea de innovación. Estos espacios son: creatividad, startups y empresa.

Creatividad: en este espacio se muestra al emprendedor para obtener una idea de innovación debe pasar por algunas fases de estudio donde se busca tener un impulso adecuado para la construcción y desarrollo de la idea que pueda surgir, algunas de estas fases son: motivación, observación, reflexión y eureka.

Startups: este espacio tiene como finalidad que el emprendedor pueda realizar el desarrollo completo de la idea de innovación que ha tenido para afrontar la crisis que existe. Aquí se encuentran una serie de herramientas necesarias para construir la estrategia que le permitirán tener al final un producto novedoso, eficiente y de muy buena calidad, para el mercado donde se vaya a mostrar. Con las herramientas ofrecidas en la plataforma Ignis Gravitas, el emprendedor puede entender, que no solo basta con tener una buena idea, sino que, para lograr el éxito, debe darle la dedicación necesaria durante su desarrollo y experimentar al máximo, para mejorar y obtener un producto lo bastante novedoso y atractivo para todos sus usuarios.

Empresa: con este espacio el emprendedor ya logra concretar la idea de innovación y desarrolla un producto que es factible y eficiente para enfrentar la crisis que se presenta, ahora solo debe reproducir ese producto y llevarlo a un nivel de competencia en el mercado.

2.3. Bases teóricas

2.3.1. Emprendimiento/Innovación

Emprendimiento e innovación guardan una estrecha relación, estos dos elementos se encuentran sutilmente asociados. Sin embargo, un emprendimiento no significa que haya innovación y viceversa, para que exista innovación no es necesario hacer un emprendimiento. Por consiguiente, cuando se dan en conjunto y el proceso es bien llevado por la(s) personas o empresas, el éxito es absoluto. Al instaurar un negocio con base en la innovación, nos encontramos frente a un emprendimiento innovador. Desde esta perspectiva, se visualiza el proceso un tanto complejo, que aquel que no incluye innovación. Pero, si se inicia un negocio con base en la innovación, generará una mayor expansión en los niveles de producción y ventas. Por tal motivo, para entender qué es emprendimiento e innovación, es necesario desarrollar estos dos conceptos.

2.3.2. Emprendimiento

Conlleva a convertir una idea nueva en una innovación exitosa utilizando habilidades, visión, creatividad, persistencia y exposición al riesgo. Cabe decir, un emprendimiento es cualquier actividad nueva que hace una persona, empresa u organización y tiene como propósito lograr ganancias económicas o no.

Emprender es atreverse a dar un paso más y asumir los riesgos que a ello lleva, es hacer un sueño realidad. Un emprendimiento es llevado a cabo por una persona a la que se denomina emprendedor. La palabra emprendedor tiene su origen en el francés entrepreneur (pionero), y en un inicio se usó para denominar a aquellos que se lanzaban a la aventura de viajar hacia el Nuevo Mundo, sin tener ningún tipo de certeza de que iban a encontrar allí. (Formichella, 2004)

2.3.3. La innovación

Es uno de los elementos clave de la actividad emprendedora. El emprendedor innova mediante la introducción de ideas o la mejora de productos y servicios, mediante la incorporación de nuevas tecnologías, procesos productivos, prácticas de trabajo o formas de hacer negocios. Innovación, es la implementación de esa idea nueva y útil.

Es la realización efectiva que logra un cambio en el sistema, con el propósito de mejorar y perfeccionar algún aspecto de su estructura, contenido o funcionamiento. En consecuencia, como afirma (Adair, 1992) la innovación convierte las ideas en productos o servicios útiles, practicables y comerciales. Innovar, se refiere a la realización de una actividad asociada al cambio, no necesariamente debe ser muy significativo, basta con que involucre una muestra de renovación. Es decir, se renovó o cambió.

2.3.3.1. Clases de innovación

De acuerdo con (Duarte, 2007) se puede establecer una clasificación de innovación utilizando tres criterios básicos: el objeto, el grado de novedad y la finalidad estratégica de la innovación.

Según el objeto de la innovación:

- **Innovaciones tecnológicas**: es la invención o el desarrollo de tecnologías novedosas, lo cual suele traducirse en herramientas más sofisticadas, capacidades antes imposibles; es decir, es el cambio de índole técnico o científico donde se introduce el bien o servicio ofrecido por una empresa u organización, a los procesos que se desarrollan dentro de la misma.
- **Innovación en producto** (bienes y servicios): el máximo nivel corresponde a la introducción de un producto totalmente nuevo en el mercado, luego figuran las mejoras, sustanciales y leves.
- **Innovación en procesos:** denominándose innovaciones TPP (Innovaciones Tecnológicas en Productos y Procesos). Abarca actividades de naturaleza muy diversa (científica, tecnológica, organizacional, financiera y comercial). Cualquier actividad empresarial (producción, organización, gestión, marketing, etcétera).
- **Innovación en los mercados**: consiste en crear, ampliar o segmentar los mercados de ventas y en crear o mejorar las fuentes de aprovisionamiento de materias primas y productos.

Según el grado de novedad de la innovación:

- **Las innovaciones radicales**: también llamadas básicas, primarias o totales, hacen referencia a productos o procesos totalmente nuevos, ya que presentan diferencias significativas en cuanto a su finalidad, prestaciones, características, propiedades teóricas, materias primas o componentes utilizados en su fabricación.

- **Innovaciones incrementales**: parciales, progresivas o secundarias, son mejoras en productos o procesos ya existentes.

2.4. Plataforma digital

Las plataformas digitales son espacios en internet que permiten la ejecución de diversas aplicaciones o programas en un mismo lugar, para satisfacer distintas necesidades. Cada una cuenta con funciones diferentes y ayudan a los usuarios a resolver distintos tipos de problemas de manera automatizada, usando menos recursos. El principal objetivo de las plataformas digitales es facilitar

la ejecución de tareas a través de programas o aplicación en un mismo lugar en la web. (Guild, 2021)

2.4.1. Tipos de Plataformas digitales:

Refiere (Giraldo, 2019), existen diversas plataformas virtuales para todos los segmentos que podamos imaginar. Cada una de ella varía los objetivos de acuerdo a las necesidades de los usuarios. Siempre que haya una necesidad, podrá crearse una plataforma para satisfacerla independientemente del sector.

- **Plataformas Educativas:** se enfocan en la educación a distancia e intentan simular las mismas experiencias de aprendizaje que encontramos en un salón de clase. Sirven para complementar o sustituir el proceso de educación tradicional.
- **Plataformas Sociales:** conocidas también como redes sociales, son muy utilizadas actualmente por gran parte de nuestra sociedad. Estas plataformas guardan diversas informaciones relacionadas son las interacciones sociales.
- **Plataformas de comercio electrónico**: actualmente abundan en internet y gracias a ella es posible comprar diversos productos y servicios sin salir de casa, sin fronteras físicas.
- **Plataformas especializadas:** son creadas para satisfacer las necesidades de un grupo de usuarios.

2.5. Startups

Desde la posición de (Ries, 2012), uno de los padres del movimiento “Lean Startup”, defiende en su obra homónima que una startup es “una institución humana diseñada para crear un producto o servicio bajo condiciones de incertidumbre extrema” y (Blank, 2012) otro de los ideólogos del método “lean startup” define las compañías startup como “una organización formada para buscar un modelo de negocio repetible y escalable”, con un gran potencial de crecimiento gracias a la introducción de innovaciones en el mercado. (Rocha, 2019)

Desde el punto de vista de los autores antes mencionados, se desprende que una startup es una organización humana con gran capacidad de cambio que desarrolla productos o servicios de gran innovación, requeridos por el mercado donde su diseño y comercialización están orientados completamente al cliente.

2.5.1. Características principales de una aceleradora de startups

Haciendo uso del informe "The Startup Factories" (Miller y Bound , 2011) sostienen que poseen las siguientes características:

- **Tienen un proceso de ingreso que es abierto y altamente competitivo.** La intención es conocer mejor a los emprendedores, y puedan ellos preguntar todas las dudas que tengan acerca del proceso de aceleración. Existen distintos procesos de admisión en función de las bases de cada programa de aceleración.
- **Proveen de financiación pre-semilla**, normalmente en intercambio por acciones de las startups.
- **Se focalizan en pequeños equipos, y no en individuos**. Los programas de aceleración estarán interesados en acoger a un equipo de emprendedores en vez de a un único emprendedor debido a la excesiva carga de trabajo durante la aceleración.
- **El apoyo limitado en el tiempo,** incluye eventos programados y mentoria intensiva.
- **Las startups son admitidos en grupos ("cohortes"), durante períodos fijos de tiempo.** En lugar de albergar compañías individuales, los programas de aceleración acogen en cada edición a promociones o "clases" de startup. Los emprendedores pueden compartir ideas y obtener ayuda de los demás compañeros, aunque no formen parte de su equipo.

2.5.2. Importancia de las startups

Cada startup está resguardado por una idea de negocio online y trata de reducir procesos y trabajos complicados con la intención de que el mercado tenga una experiencia de uso simplificada y fácil. Habitualmente son negocios que quieren innovar, desarrollar tecnologías y diseñar procesos web, aliados con empresas de capital-riesgo que deciden apostar por las startups. No todo el mundo debe o tiene la oportunidad de trabajar en grandes empresas, y ese es el grado de importancia de un startup; además, se caracterizan por ser digitales, innovadoras, ágiles y propensas a asumir y gestionar riesgos. Mantienen un estrecho vínculo con la tecnología, permite a los emprendedores y emprendedoras desarrollar nuevas ideas y consolidarlas en el mercado. Impulsan nuevos modelos de negocio qué ofrecer al mercado, cómo hacerlo, quién va a ser el público objetivo, cómo vender un producto o servicio y cuál será el método para generar ingresos.

2.6. Tren de Clientes en Ignis Gravitas Inc.

Este espacio ofrece al emprendedor la capacidad de mover el tren, y llevarlos a emprender e innovar de manera inteligente y logrando un crecimiento de clientes de manera serena, que implementen, hagan más inversiones económicas estación tras estación hasta convertirse en empresa. Está compuesto por ocho estaciones: Observacional, Enfoque de Grupo, Etapa del Primer Cliente, Primer Grupo de Clientes, Primeros Adoptantes, Chasm (Abismo), Escalamiento y Empresa. Estas estaciones al implementarse y darle funcionalidad en la plataforma, permiten al emprendedor/innovador ir de estación en estación y tomar las decisiones con respecto a cómo se está alimentando esta responsabilidad, se visualizan e identifican a los clientes y usuarios capturados para el impulso del emprendimiento. (Ignis Gravitas INC).

2.6.1. El Cliente

El cliente es la persona o ente que requiere del servicio o producto ofrecido por una empresa, a partir de sus necesidades. Actualmente, el cliente como emprendedor va de la mano con la tecnología, Es necesario que las startups centren sus estrategias en los consumidores. Pero, para tener éxito en esta tarea es fundamental establecer un buen abordaje de servicio al cliente, pues es muy importante para un startup porque interfiere directamente en el desarrollo de la empresa.

2.6.2. Estaciones del Tren del Clientes (TOC) ideada por Ignis Gravitas Inc.

Estrategia conformada por ocho estaciones ideadas por los cofundadores de Ignis Gravitas que se desarrolla e implementa en la plataforma, dentro de la línea de responsabilidad del espacio Tren de Clientes según el documento de requisitos del producto, estaciones:

- **Observacional:** esta estación es responsabilidad de los cofundadores de startups, su objetivo es identificar en detalle la actividad central, las necesidades, molestias, situaciones problemáticas, deseos y búsquedas de crecimiento cultural, de los potenciales clientes; a través de entrevistas a fin de detectar todas sus necesidades en detalle, una vez que esto se logre, presentar la innovación: el producto o servicio y seguir indagando acerca de lo que les gustaría del producto, el estado observacional detecta las necesidades.

- **Enfoque de grupo:** una vez estudiado las necesidades del cliente a través de sus opiniones para ir mejorando el producto, se formará un focus group donde se muestra el producto que se está desarrollando, durante jornadas de participación por los usuarios, y tener presente el objetivo es captar cómo el producto es recibido en realidad, mostrando las mejoras y nuevas funcionalidades.
- **Etapa del Primer cliente:** en esta estación el usuario detectado va a estar usando el producto o servicio que se le presenta, que se le construye y a éste es el que hay que estudiar en detalle, en lo más mínimo para entender por qué quiere el producto o servicio; es decir, estudiar la situación problemática existente del usuario para plantear la solución. Este primer usuario después de estudiarlo hay que preguntarle ideas de lo que desea integrar al producto, que quiere agregarle. El primer usuario es un cliente clave pues dará al producto las primeras recomendaciones y se podrá observar si se cubren las necesidades para las cuales se ha desarrollado el producto o servicio que falta por agregarle y poder evaluar de esta manera lo que se tiene
- **Primer grupo de clientes:** una vez logrado el interés del primer usuario, es necesario llevarlo a la estación "primer grupo de clientes". Aquí se toma un grupo de usuarios enfocados en el uso del producto y la razón silenciosa de un grupo de clientes, la masa de clientes estaría ubicados por la zona donde se ubicó el primer cliente. Con este primer grupo de clientes se podrá obtener opiniones con respecto al producto o servicio y se podrá ir mejorando para adaptarlo a las necesidades de los usuarios. Este proceso se debe repetir con varios grupos de usuarios hasta obtener un producto, que cubra la problemática existente y se obtenga una solución satisfactoria a todos los clientes de la plataforma Ignis Gravitas donde se estará llevando el producto del emprendimiento.
- **Primeros adoptantes:** son un tipo único de clientes y son invaluables para un startup, son los que adoptan el producto, lo tienen como parte integral de su vida, entienden que habrá errores en las primeras versiones del producto, ofrecerán comentarios valiosos para que puedan construir la herramienta en la dirección correcta. Los primeros en adoptar son líderes de opinión, prestan atención a lo que los innovadores han descubierto y encuentran un uso práctico para la innovación, posteriormente, comunican a sus seguidores la utilidad del nuevo producto desempeñando un papel muy importante al influir en la actitud y cambiar el comportamiento de posteriores adoptantes.

- **Chasm (abismo):** es un momento clave, una vez que tenemos el grupo de adoptantes del producto, estos empiezan a extenderse por toda la población y el emprendedor - innovador comienza a emocionarse porque llegan clientes de todas partes y, de repente, el número de clientes comienza a decaer de mes en mes, a este periodo se le conoce como "abismo o precipicio" y es cuando el emprendimiento se queda o muere en el intento. Es el momento en que a pesar de seguir haciendo esfuerzo no obtienes resultados; es decir, se estanca, e incluso, empeora. Pero, es aquí en este momento en que el emprendedor debe salir del precipicio innovando con nuevas estrategias, cambiar el discurso para ganar clientes, promover la captura de clientes que mantengan los gastos; emprender de adentro hacia afuera buscando la forma de saltar el precipicio; después de esto de alguna manera comienza a ganar clientes nuevamente, pero el crecimiento es de manera exponencial y se pasa a la siguiente estación "escalando".
- **Escalamiento:** una vez que el emprendedor logra salir del precipicio, y cuenta con un número de clientes importantes, comienza a crecer su emprendimiento. Sus servicios y productos pueden ser entregados a un mayor número de clientes, sin que ello signifique más esfuerzos o incluso inversiones obligatorias. Para escalar el emprendimiento, lo puede hacer a través de diferentes canales de promoción y llevarlo al mercado o directamente al consumidor**.** La idea de escalamiento es llevar el producto al mundo y pasar a convertirse en empresa.
- **Empresa:** esta es la última estación en la cual el emprendedor logra concretar la idea de innovación y desarrolla un producto que es factible y eficiente para enfrentar la problemática presentada cubriendo las necesidades del usuario, por lo que debe reproducir el producto y llevarlo a un nivel de competencia en el mercado.

2.7. Módulo en Programación

En programación un módulo es un software que agrupa un conjunto de subprogramas y estructuras de datos; es decir, es un segmento de un programa de ordenador. Los módulos son unidades que pueden ser compiladas por separado y los hace reusables, permiten que múltiples programadores trabajen en diferentes módulos en forma simultánea, produciendo ahorro en los tiempos de desarrollo. Los módulos promueven la modularidad y el encapsulamiento, pudiendo generar programas complejos de fácil comprensión. De las varias tareas que debe realizar un

programa para cumplir su función u objetivos, un módulo realizará una o varias de dichas tareas en algún caso.

En el caso de la programación, los módulos suelen estar organizados jerárquicamente en niveles, de forma que hay un módulo principal que realiza las llamadas oportunas a los módulos de nivel inferior. Cuando un módulo es convocado, recibe como entrada los datos proporcionados por otro del mismo o superior nivel, el que ha hecho la llamada; luego realiza su tarea. A su vez este módulo convocado puede llamar a otro u otros módulos de nivel inferior si fuera necesario; cuando ellos finalizan sus tareas, devuelven la salida pertinente al módulo inmediato llamador, en secuencia inversa. Finalmente se continúa con la ejecución del módulo principal. Cada uno de los módulos de un programa idealmente debería cumplir las siguientes características: tamaño relativamente pequeño e Independencia modular

2.8. Ruby on Rails. R o R (framework)

Ruby on Rails es un framework web que nació en la empresa Basecamp, caracterizado por la simplicidad, se construyó sobre la base del lenguaje de programación Ruby que originalmente estaba destinado a ser una tecnología para el desarrollo rápido de software. Por eso a Ruby on Rails se le suele llamar "tecnología para startups", ya que se creó para realizar lanzamientos rápidos. Ruby on Rails depende de Ruby para existir y funcionar. Trabaja con el modelo MVC (modelo vista-controlador), un patrón de arquitectura bastante famoso. Un framework, es un conjunto de herramientas software combinadas de forma que simplifican o automatizan tareas que de otro modo serían demasiado complejas o tediosas para sus usuarios, dejando la parte más complicada lista para ser utilizada. Su desarrollo fue iniciado en 2004, y a finales de 2005 vio la luz la primera versión estable; en una época en la que se estaba generalizando el acceso a internet y la computación personal empezaba a trasladarse cada vez más a la red.

Ruby on Rails se distribuye al público bajo la licencia MIT, que permite su libre uso, modificación y distribución (The MIT, 2008), permite el uso de las herramientas de Rails en productos comerciales; esto ha permitido que Rails haya sido la solución elegida para el desarrollo de plataformas populares como Airbnb, Kickstarter o GitHub. Se lanzó como Opensource y fue todo un éxito; en aquel momento no había una herramienta con esas características para el

desarrollo web, su fin último es entregar lo más rápido posible el valor que el cliente o el producto necesita.

Ruby on Rails sigue algunos principios de desarrollo, como Don't Repeat Yourself (No te repitas a ti mismo), para evitar repetir código y se vale de Convención sobre Configuración; esto es un paradigma de programación de software que busca minimizar el número de decisiones que un desarrollador necesita hacer, ganando simplicidad, sin perder flexibilidad.

En pocas palabras, Ruby on Rails es una mejora o complemento de Ruby, lenguaje de scripting orientado a objetos, caracterizado por ser claro y conciso, donde el desarrollador puede expresar las ideas de forma limpia y natural. La sencillez a la hora de programar facilita retomar un código que fue escrito en tiempo pasado, dando legibilidad y claridad. Señala que el lenguaje Ruby permite la meta programación, que consiste en generar o manipular programas a partir de otro programa, es decir que un código tenga la capacidad de modificar o generar un programa. Esto permite al programador ahorrar tiempo en la producción de código, de la cual Rails hace uso, lo que resulta en una sintaxis que muchos de sus usuarios encuentran legible. (Acerca de Ruby, 2007).

2.8.1. Ventajas de RoR

Señala (Hans Baumann, 2021)**.**

- **Eficiencia de tiempo:** aunque es relativamente fácil aprender los fundamentos del marco de aplicaciones Ruby llevará un tiempo aprovechar verdaderamente su potencial. Sin embargo, es rápido y eficaz.
- **Ahorro en costos:** el desarrollo de aplicaciones con RoR no sólo es eficiente en cuanto a tiempo, sino también en cuanto a los costos. En el caso de las "startups", que necesitan ahorrar en todo lo posible, no tendrían que comprometer su calidad al utilizar Ruby.
- **Capacidad de herramientas:** hay muchos fans del marco de aplicaciones Ruby en todo el mundo. Estos aficionados son los desarrolladores de software que están constantemente creando "Rubygems" y bibliotecas de terceros para mejorar las características de Ruby, que se pueden utilizar como parte de tu propio software. Estar en "Rubygems" no influye en el contenido del código, sino en la experiencia en el diseño de software. Algunas de ellas pueden ayudar en la depuración; es decir, a identificar y corregir errores de programación. Por consiguiente, son una gran ayuda en la optimización de la fase de pruebas de cualquier trabajo de desarrollo. Las "gemas" pueden hacer casi cualquier tarea, agregándole

funciones a la aplicación en desarrollo haciendo que sea posible y fácil ayudar a los desarrolladores web a crear aplicaciones que satisfagan la mayoría de los requisitos de los usuarios.

- **Adaptabilidad a los estándares web:** una de las fuertes ventajas de RoR es el respaldo a los estándares web para cada parte de una aplicación desde la interfaz de usuario (UX) hasta la transferencia de datos.

2.8.2. Desventajas de Ruby on Rails.

- **Poca flexibilidad:** cuando se trata de tareas y funciones básicas RoR es casi invencible. Sin embargo, como hay muchos objetos predeterminados y establecidos no deja mucho espacio para la creatividad. Por lo tanto, antes de tomar la decisión sobre el marco de aplicaciones que se desea utilizar para un proyecto en particular, es necesario pensar en los elementos básicos (si son estándar o más bien únicos).
- **Evolución continua:** si bien los cambios continuos son un proceso natural, especialmente para un lenguaje con una comunidad tan grande como la de Ruby, puede resultar abrumador y muy difícil adaptarse, especialmente para los principiantes, dichos cambios se producen rápidamente no sólo en el propio framework, sino en las herramientas y bibliotecas desarrolladas por la comunidad de Ruby.
- **Tiempo de rendimiento: e**sta característica de RoR puede ser un tema discutible, pues en comparación con otros frameworks de aplicaciones web, el tiempo de arranque del framework es bastante largo, especialmente cuando se trabaja con un proyecto masivo. Comparado con el tiempo de espera de otros frameworks, hay una diferencia. Aunque muchos de sus usuarios afirman que no es una característica tan significativa, dadas todas las grandes ventajas que Ruby on Rails tiene que ofrecer.

2.9. Modelo Vista Controlador y Rails

El Patrón Modelo-Vista-Controlador (MVC) es una arquitectura de software que separa la lógica y los datos de su representación y las comunicaciones. Esta arquitectura está diseñada para facilitar el mantenimiento de las aplicaciones, la reutilización de código y la separación de tareas. Los elementos de MVC cumplen las siguientes funciones: **Modelo:** Define los datos del sistema,

gestionando el acceso a estos y su transformación. **Vista**: Presenta la información al usuario de la aplicación. Es posible definir múltiples vistas para la misma información**. Controlador**: Recibe entradas y las convierte a comandos para el modelo o la vista. Habitualmente, la entrada del usuario se transmite al controlador mediante elementos del interfaz gráfico, perteneciente a la vista

Rails ofrece una serie de clases y herramientas dedicadas exclusivamente a la implementación de una arquitectura MVC.

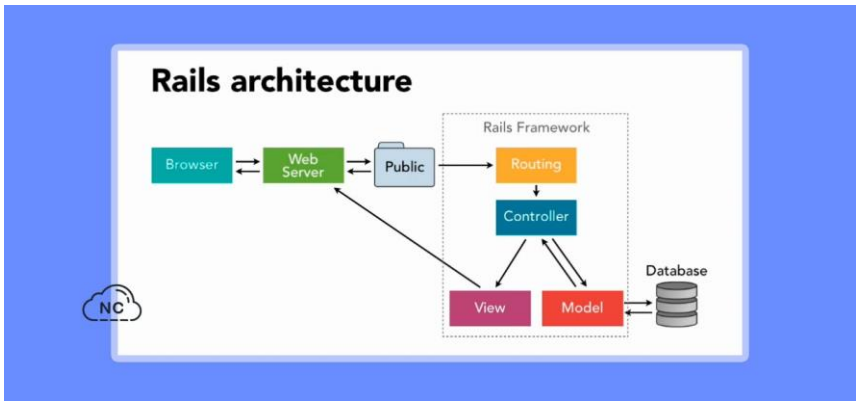
- **La clase Active Record**. Este módulo, que cumple el papel de Modelo en MVC, está diseñado para manejar todas las tareas de la aplicación relacionadas con la base de datos. Esto incluye la conexión, la recuperación y almacenamiento de información desde la base de datos. Este componente se basa en la abstracción de la base de datos, que permite que el desarrollador pueda realizar todas las tareas necesarias en la base de datos (incluyendo la creación de tablas) sin tener que utilizar el lenguaje de consultas del SGBD en cuestión. Para esto incluye la herramienta de migraciones.

- **La clase Action Controller**. El controlador gestiona la lógica del programa, actuando como unión entre el navegador, la capa de presentación y los datos de la aplicación. Se encarga de recibir peticiones del navegador y decidir cómo gestionarlas, además de recuperar información del modelo y transmitirla a la vista. Esta clase, además proporciona otras funcionalidades, como la autenticación HTTP, el manejo de Cookies y la gestión de excepciones. También aporta herramientas para mejorar la seguridad de la aplicación como son el forzado de SSL y la prevención de falsificación de peticiones.

- **La clase Action View.** Al igual que Action Controller, esta clase forma parte de la librería Action Pack. El módulo Action View, está pensado para incluir únicamente la lógica de presentación. Es decir, no debería incluir lógica de aplicación ni acceder directamente a la base de datos bajo ningún concepto. Es posible que una vista incluya únicamente HTML, sin usar Ruby, pero es altamente probable que sea necesario generar la vista en base a los resultados de una operación realizada por el controlador. Para ello, esta clase hace uso de ERB, un formato de plantillas que permite incluir código Ruby en medio de ficheros HTML. Estos ficheros (que reciben la extensión .html.erb) son procesados por la clase Action View, sustituyendo el código Ruby incrustado por el resultado de su ejecución. Un

ejemplo de HTML con ERB embebido es el siguiente: El código situado entre « %» sería evaluado por Ruby, y sería situado en el HTML resultante entre las etiquetas « ».

Figura 1. Rails Patrón Modelo Vista Controlador



2.10. Ruby on Rails y Base de Datos PostgreSQL

Una base de datos es un "almacén" que nos permite guardar grandes cantidades de información de forma organizada para que luego podamos encontrar y utilizar fácilmente. Se puede definir como un conjunto de información relacionada que se encuentra agrupada o estructurada. Desde el punto de vista informático, la base de datos es un sistema formado por un conjunto de datos almacenados en discos que permiten el acceso directo a ellos y un conjunto de programas que manipulen ese conjunto de datos. Cada base de datos se compone de una o más tablas que guarda un conjunto de datos. Cada tabla tiene una o más columnas y filas. Las columnas guardan una parte de la información sobre cada elemento que queramos guardar en la tabla, cada fila de la tabla conforma un registro. Entre sus características se pueden mencionar: independencia lógica y física de los datos, redundancia mínima, acceso concurrente por parte de múltiples usuarios, integridad de los datos, consultas complejas optimizadas, seguridad de acceso auditoría, respaldo y recuperación.

En lo referente a la fusión de las base de datos por defecto para cualquier aplicación o desarrolladores web se aplican las bases de datos que vienen por defecto en este caso son las de MySQL el propio SQL En este caso la fusión de Ruby on Rails con PostgreSQL relativamente fue

exitosa ya que postgreSQL presenta características acoplable y es considerada una base de datos de código abierto más avanzada del mundo que proporciona grandes características que normalmente solo se encuentran en bases de datos comerciales como DB2 y Oracle.

2.11. Definición de Términos Básicos

Emprendimiento: es el proceso que desarrolla e implementa un pionero para hacer realidad una idea, por lo general, de negocio.

Innovación: es el motor de todos las startups. Estos negocios tienen como característica esencial ofrecer soluciones nuevas o bien aprovechar los recursos tecnológicos para reinventar una oferta existente. En muchos casos, las startups más lucrativos son aquellas que incluso generan una nueva necesidad en el mercado.

Organización: es un grupo de personas orientadas a la consecución de objetivos comunes, destacando en su propuesta de valor la oferta de productos y servicios tradicionales adecuados para el desarrollo de las condiciones hacia el mundo empresarial.

Startup: organización emprendedora emergente, cuyo objetivo es la realización y ejecución de su propuesta de valor en el mercado. Basada en esquemas de alta especialización e innovación, generando una disrupción a lo establecido

Framework: En el desarrollo de software, es una estructura de soporte definida en la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, un framework puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros 148 softwares para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

Ruby on Rails: es un framework de aplicaciones web, con código abierto escrito en el lenguaje de programación Ruby, siguiendo el paradigma del patrón Modelo Vista Controlador (MVC).

Modelo de negocio en startups: consiste en una empresa joven enfocada principalmente a la innovación y tiene un importante potencial de desarrollo. El objetivo de un startup es ofrecer un producto o servicio que tenga un fuerte impacto en el comportamiento de consumo del cliente

2.12. Basamento Legal

Ignis Gravitas Inc, está inscrita en Delaware, Estados Unidos. Tiene como objetivo la expansión de la cultura emprendedora y el crecimiento acelerado de las startups y su dinámica en su plataforma a nivel mundial. Contiene un espacio donde pueden interactuar los distintos roles que hacen vida en la plataforma y por ende la aceleradora Ignis Gravitas, siendo la comunidad e-Volition un acelerador de startups en línea con orientación autónoma de expertos **y** un ecosistema empresarial exclusivo en crecimiento. Por tal razón, Ignis Gravitas Inc. y los miembros que forman parte del ecosistema Volition tienen su basamento legal en la legislación de Delaware en los Estados Unidos y a través de la suscripción del acuerdo redactado e implementado por Ignis Gravitas, los miembros que forman parte del ecosistema Volition se comprometen a cumplir con todos los aspectos relacionados a la legislación, constituyendo así el marco legal al que se encuentren sujetos.

2.13. Variables de la Investigación

Tema de Estudio: Desarrollo del Módulo Tren- de –Clientes de la aceleradora virtual Ignis Gravitas Inc. en Ruby on Rails

Variables:

- **Variable independiente:** desarrollo del Módulo Tren- de –Clientes
- **Variable Interviniente:** aceleradora virtual Ignis Gravitas Inc
- **Variable Dependiente:** Ruby on Rails (framework)

CAPITULO III

MARCO METODOLÓGICO

3.1. Consideraciones Generales

En toda investigación es indispensable que en los procesos a estudiar, así como sus resultados y las diferentes evidencias de importancia que se presentan en el problema planteado, su relación y los objetivos a lograr, se establezca un debido marco metodológico para dar respuesta a la interrogante planteada, que en el presente trabajo se sitúa en la integración del módulo de tren de clientes en la aceleradora virtual Ignis Gravitas Inc., con base en la conceptualización de ocho estaciones estrategia ideada por Ignis Gravitas y por la cual tiene que transitar el modelo de negocios del startup. Por lo tanto, el marco metodológico describe los pasos necesarios que se emplearon durante la investigación, como fueron: los métodos, técnicas y procedimientos; es decir, esboza el momento técnico-operacional presente a lo largo de la investigación.

3.2. Nivel y Tipo de Investigación

De acuerdo a la problemática planteada, y en función de los objetivos, el nivel de la investigación se ubicó dentro de la investigación descriptiva. Esta se trabaja sobre realidades de hechos o fenómenos y su característica fundamental es la de presentar una interpretación correcta de la misma. El tipo de investigación se ubicó dentro del Proyecto Factible que según Manual de FUNDAUPEL (Libertador, 2011, pg.21) “consiste en la investigación, elaboración y desarrollo de una propuesta de un modelo operativo viable para solucionar problemas, requerimientos o necesidades de organización o grupos sociales; pueden referirse a la formulación de políticas, programas, tecnológicas, métodos y procesos. El proyecto debe tener apoyo en una investigación de tipo documental, de campo o de un diseño que incluya ambas modalidades”

De la definición anterior, se concluye como proyecto factible todo estudio en donde se desarrollan propuestas que brindan soluciones a problemas de diferente índole, ofreciendo recomendaciones que incluyen, no solo políticas, sino también tecnológicas con sus respectivos métodos, procesos y pasos a seguir, como es el caso de la presente investigación donde se

desarrolló e implemento el módulo tren de clientes y hacerlo funcional dentro de la plataforma de Ignis Gravitas.

3.3. Diseño de Investigación

Dentro de la investigación, se definió como diseño de investigación la estrategia que orientó el proceso que se desarrolló para llevar a cabo la implementación del módulo tren de clientes en la plataforma de Ignis Gravitas, a través del desarrollo de las estaciones que transitarán por el modelo de negocios planteado y, en función de los objetivos establecidos. Se orientó en un diseño combinado tanto de campo como documental y como se explicó en el apartado anterior, fue de un nivel de tipo descriptivo. En el diseño de campo se observan los hechos estudiados tal como se manifiestan en su ambiente natural.

Define (Tamayo,T, 2006.pg.110), el diseño de campo como "cuantos datos se recogen directamente de la realidad por lo cual, los denominados primarios su valor radica en que permiten cerciorarse de las verdaderas condiciones en que se han obtenido los datos, ello facilita su revisión o modificación en caso de surgir dudas". Este diseño permitió extraer, observar y recolectar datos directamente de la plataforma Ignis Gravitas para ser analizados y llevar a cabo el objetivo planteado.

Por otra parte, esta investigación se apoyó en la Investigación Documental**.** Según (Arias, Fidias, 2006.pg 27.) "es un proceso basado en la búsqueda, recuperación, análisis, critica e interpretación de datos secundarios, es decir, los obtenidos y registrados por otros investigadores en fuentes documentales1a: impresos, audiovisuales o electrónicos". Por lo tanto, en la presente investigación se llevó a cabo la revisión y selección de fuentes bibliográficas con la finalidad de recolectar información para luego analizarla y clasificarla, con la intención de indagar y reflexionar sobre la temática planteada.

Esta investigación se consolidó con base en los siguientes procedimientos, para la documentación de los diferentes procesos que se realizan dentro de la plataforma Ignis Gravitas Inc., se hizo una recopilación y análisis de información de fuentes directas e indirectas según su origen. La información de fuentes directas, se obtuvo a través de la consulta con las personas que intervienen en los procesos que se realizan dentro del sistema y constituye la plataforma de

emprendimiento, pues son los que conocen el funcionamiento completo de estas herramientas y aportan de manera veraz toda la información a documentar. La información de fuentes indirectas, se obtuvo de textos especializados para el diseño de plataformas y artículos de documentación de diferentes procesos de aprendizajes de sistemas.

3.4. Población y Muestra.

3.4.1. Población.

En la presente investigación el objeto de estudio fue el ecosistema que conforma la plataforma virtual Ignis Gravitas Inc, la cual comprende los propósitos establecidos en la misma**.** Expresa Arias Antes citada, se entiende por población un conjunto finito o infinito de elementos con características comunes para los cuales serán extensivas las conclusiones de la investigación. Esta se queda delimitada por el problema y por los objetivos del estudio.

3.4.2. Muestra

La muestra es una parte de la población u universo de estudio que se selecciona con el fin de realizar la investigación y determinar los aspectos de dicha población. En este caso el objeto de estudio fue el espacio del panel Tren de clientes que conforma la plataforma Ignis Gravitas Inc, y el tipo de muestra es no probabilística, pues no brinda a todas las personas, objetos o elementos la misma probabilidad de ser elegidos para realizar el estudio. La Muestra en este caso es también llamada intencional y es una técnica que utiliza el investigador con base en su propio juicio para elegir lo que formará parte del estudio. La muestra intencional es definida como un procedimiento que permite que los elementos sean escogidos con bases, criterios y juicios preestablecidos por el investigador (Balestrine, M, 2006).

3.5. Técnicas e Instrumentos de Recolección de Información

En la presente investigación se utilizaron para alcanzar el desarrollo del estudio en sus diferentes etapas, las siguientes técnicas e instrumentos para recolección de información y posterior análisis de datos:

- **La observación directa**: técnica que permitió mostrar una visión general y particular del área en estudio, en su ambiente real que es la plataforma Ignis Gravitas Inc.
- **El análisis de fuentes documentales con respecto a las variables a investigar**. La revisión documental, fue otra técnica en la que se apoyó la presente investigación en cuanto a la búsqueda, recuperación, análisis e interpretación de datos secundarios; es decir, los obtenidos y registrados por otros investigadores en fuentes documentales: impresas, audiovisuales o electrónicas.
- **Entrevistas con informantes claves:** esta técnica se empleó con personas que poseen información valiosa y determinante para esta investigación. La entrevista en esta investigación fue no estructurada, con base en la interacción pertinente entre el investigador y los fundadores de la plataforma Ignis Gravitas, con el fin de obtener información y datos relevantes para el cumplimiento de los objetivos.

Técnicas de Recolección de información e Instrumentos a utilizar

Tabla 1. Técnica de Recolección de información e Instrumentos
Fuente: Datos de Investigación

TÉCNICAS	INSTRUMENTO
Observación Directa	Lista de chequeo, cámara fotográfica, diario de campo
Entrevista con informantes claves	Entrevista no estructurada distribuida en varias sesiones (Reuniones)
Análisis documental	Cuaderno de notas, fichas, equipo, laptop, informes técnicos y CP

3.6. Sistema de Operacionalización de variables

Matriz de Operacionalización de Variables

Objetivo General: Desarrollar e implementar un módulo del Tren de Clientes en la Aceleradora Virtual Ignis Gravitas en Ruby on Rails para fortalecer a los emprendedores / innovadores de forma inteligente e incrementar las probabilidades de éxito en su modelo de negocios.

Tabla 2. Matriz de operación de Variables

VARIABLES	DIMENSIÓN	INDICADOR	TÉCNICA/INSTRUMENTO
Desarrollo del Módulo Tren- de –Clientes	*Emprendimiento *Innovación	*Toma de decisiones y responsabilidad *Segmento de clientes. *Ejecución de las estaciones a implementar	*Organización jerárquica en niveles. *Estructuración de datos *Observación y revisión de fuentes documentales (requisitos del producto: (Tren-de-Clientes)
Aceleradora virtual Ignis Gravitas Inc	*Startups	* Implementación del Módulo Tren-de-clientes *	*Supervisión y evaluación de la ejecución de cada una de las estaciones
Ruby on Rails (framework)	*Metodología agiles *Factibilidad de uso	*Sintaxis *Sencillez del código *Ejecución de la plataforma a nivel local *Creación automática de casos de datos y formularios.	*Actualizaciones a nivel de aplicación general *Ejemplos de tutoriales *Realización de demos * Revisión de funciones y módulos que acceden a la base de datos

3.7. Técnicas de Procesamiento y Análisis de Datos.

Para el desarrollo e implementación del módulo en la plataforma de emprendimiento Ignis Gravitas Inc., se utilizó el documento de requisitos del producto: TREN DE CLIENTES. En la línea de responsabilidades. Diseñado por los fundadores de la plataforma, el cual permitió la ejecución de la estrategia que consiste en el desarrollo de ocho estaciones y así darle funcionabilidad e impulsar la creatividad de los emprendedores y ofrecerles un espacio de crecimiento en el mercado de una manera inteligente.

3.8. Fases de la investigación.

La guía para el desarrollo de la estrategia de las estaciones se ubicó en la plataforma de Ignis Gravitas Inc., en el espacio Tren de clientes del ecosistema de emprendimiento. Para la ejecución de cada una de las estaciones se emplearon las indicaciones establecidas por los fundadores en el documento de requerimientos del producto (anexo), y se efectuaran en tres fases.

Fase 1. De exploración.

En esta fase, se realizó un mapeo del ecosistema de emprendimiento de la plataforma Ignis Gravitas Inc. a fin de indagar y conocer la situación real de la plataforma, en el contexto existente; así como estudiar la aplicabilidad de la estrategia a implementar en la plataforma que podría impactar e influir en el contexto actual.

Por otra parte, se realizará la instalación del framework con las herramientas básicas necesarias para iniciar el desarrollo de Ruby on Rails contentivo de los comandos necesarios para la ejecución del estudio; así mismo, se formalizará la configuración de la base de datos, definición del módulo inicial y la introducción de los datos de prueba.

Fase 2. Desarrollo de cada una de las estaciones.

En esta fase, se inició el proceso de desarrollo e implementación de forma integrada para posteriormente ser compactadas en el módulo tren- de -clientes utilizando el framework Ruby on Rails que son clave para su éxito, presentando la ventaja que en la plataforma ya estaba preparada para dicho framework.

Primera Estación:

- **Observacional:** la actividad de esta estación es responsabilidad de los Co-Fundadores. Su objetivo fundamental es identificar la actividad central, las necesidades, las molestias, los deseos, búsquedas de crecimientos cultural. Para esto, se ejecutará un juego de formularios para apoyar a los emprendedores. **Implementación:** botón Segmento de Clientes (SdeC), a través de un formulario que contendrá el nombre de los segmentos, así como dos casillas y dos tarjetas siemesis, una a la izquierda de la vista y otra a su derecha. Este Formulario

S de C. (Se debe actualizar con el dialogo **Actividad**, y sustitución de nombres). Cada Segmento de Clientes es único y cada SdeC forma un conjunto. Habrá una casilla **Circunstancia** que tendrá algunos componentes. Asimismo, se reflejará el reglón de la **Actividad**, Se refiere a la actividad central a realizar por el segmento de cliente identificado en la circunstancia descrita. Además, contendrán: **Ventana 'Planteamiento del Problema'** que estará compuesto por cuatro botones emparejados, botón trabajo, botón molestias, botón deseos y botón cultura actual. También contendrá Ventana '**soluciones al Problema'**. Tiene como propósito expresar la solución al planteamiento de la actividad en la circunstancia descrita y estará compuesta por cuatro botones emparejados con los cuatro botones de la ventana planteamiento. Estos botones serán botón propuesta, botón alivios, botón satisfacciones y botón nueva cultura.

Segunda Estación.

- **Enfoque de Grupo:** tiene como visión, la realización de experimentos, mediante eventos programables; los cuales serán ejecutados seleccionando clientes como invitados. El análisis de las pruebas del producto con los clientes, determinará las fortalezas y debilidades del producto, lo cual llevará a descubrir el camino del producto, la línea que se debe seguir. Los objetivos de esta estación son: encontrar mediante el análisis de resultados y conclusiones de los experimentos realizados, las molestias, situaciones y otras circunstancias (las cuales pueden ser contrastadas con las hipótesis planteadas en la estación observacional). Este contraste será clave para definir y listar los siguientes objetivos: encontrar fortalezas y debilidades del producto. Implementación: vista creación de evento (tarjeta de enfoque de grupo) y vista formulario de tarjeta de enfoque de grupo. **Implementación:** vista creación de evento (tarjeta de enfoque de grupo), vista formulario de tarjeta de enfoque de grupo, vista de notificación emergente y vista formulario de relleno de información de resultados y conclusiones.

Tercera Estación:

- **Etapa Primer Cliente**. aquí se tratará de orientar a los cofundadores a identificar un usuario quien sea real, sincero, en lo que vive alrededor del producto que se desarrolla. Esta persona debe conocerse profundamente. Su implementación está compuesta por cuatro niveles de formularios, los siguientes:

A. Formulario Datos Personales. Consiste simplemente en capturar los datos de contactos y datos personales que permitan perfilar aún mejor a la persona como Primer Usuario. - Nombre, Género, Edad, Dirección, Emails, Teléfonos.

B. Formulario contexto personal aquí se transcribirá el ambiente y actividad que vive el Primer Usuario: profesión, breve descripción de su personalidad, grado de necesidad por el producto, grado de experticia, contexto de uso y si el cliente paga o no. También se agregará el precio estimado por el potencial cliente de la Propuesta.

Cuarta Estación:

- **Primer Grupo de Clientes:** contiene un listado de los SdeC creados, permitiendo agregar una cantidad sustancial de clientes por segmento. Los objetivos de esta estación son: Implementación: vista para la lista de segmentos de cliente, vista para la lista de clientes, vista de la tarjeta de cliente (ficha de información del cliente), vista para agregar un cliente contendrá ticket de cliente: información personal: •Edad. •Correo electrónico. •Dirección. •Teléfono. •Nombres y apellidos; información de contexto: •Profesión. •De donde proviene. •Contexto de uso del cliente. •Grado de interés (0-100%). •Grado de Experticia (0-100%). •Paga: si o no. •Descripción. •Número de ventas obtenidas.

Quinta Estación:

- **Primeros adoptantes:** esta estación contendrá un conjunto de bloques de información que representaran, el estatus en cantidad de clientes que han adoptado y comprado el producto;

así como representar el estatus de marketing. La Implementación contendrá una cajetilla de información compuesta por los siguientes elementos: número de clientes que llegan (por día, mes y año), número de clientes que se retiran (por día, mes y año), número total de clientes que han llegado y se han retirado en el tiempo que se ha desarrollado el emprendimiento del producto y calibrador que represente el estatus de marketing digital.

Sexta Estación:

- **Chasm** (abismo): representa el espacio de tiempo y de responsabilidades ejecutadas durante el Chasm del emprendimiento. Los objetivos de esta estación son: diseñar una estrategia para resolver el abismo, con base en siguientes elementos: enumerar y clasificar fortalezas del producto, enumerar y clasificar debilidades del producto, crear una lista de la situación o circunstancia descrita de las fortalezas (antes) de ser glorificadas o mejoradas, crear una lista de situación o circunstancia descrita de las debilidades (antes) de ser corregidas o reparadas. Su Implementación contendrá: se tendrá un botón para agregar fortaleza, debilidad y ventas obtenidas. Esta vista tendrá el nombre de la fortaleza, su descripción, tiempo de glorificación unidad de tiempo (en horas, días o meses) y grado de libertad (%). También esa misma vista tendrá el nombre de la debilidad, descripción de la debilidad, tiempo de reparación unidad de tiempo (en horas, días o meses), grado de la debilidad (%). En ese mismo espacio se tendrá el número de ventas obtenidas para cada fortaleza/debilidad correspondiente. Antes: lista de fortalezas antes de ser glorificadas, contiene una ficha con una descripción, tiempo de glorificación, el grado de la fortaleza. Después: lista de fortalezas luego de ser glorificadas. Descripción de la solución. Botón de crear fortaleza, debilidad y venta obtenida. Cada fortaleza se podrá editar, mostrar y eliminar; si se elimina una fortaleza se eliminará su debilidad en caso de que tenga. Se marcará con una X la fortaleza/debilidad de no haberse completado todos sus campos, de lo contrario se marcará con un check de completar todos sus campos. Se mostrará la venta obtenida por cada fortaleza/debilidad y se podrá editar la venta.

Séptima Estación:

- **Escalamiento**: contendrá un conjunto de bloques de información que representaran el progreso del proyecto de emprendimiento. Implementación: una cajetilla de información compuesta por los siguientes elementos: número de ventas, número de clientes activos, número de clientes inactivos. Se tendrá una sola cajetilla de información por SdeC, la cajetilla estará equipada con dos botones: ver detalles y editar cajetilla de información.

Octava Estación

- **Empresa**: esta visualización, contendrá un boceo de diseño libre, el cual será una medalla, un premio o una felicitación al emprendedor como usuario, por haber construido su idea y levantarla al punto de llevarla a una empresa. Implementación; Imagen, información y texto distribuido netamente por diseño gráfico libre (que contraste y siga la trasformación de la LANDING PAGE.

Fase 3. Integración del módulo tren de clientes en la plataforma Ignis Gravitas Inc.

El objetivo de esta fase es dar funcionalidad al Módulo Tren de Clientes en el ecosistema de emprendimiento de la plataforma Ignis Gravitas Inc. Esta fase sería el punto de sincronización ente el desarrollo de cada una de las estaciones que conforman el módulo y la integración de cada una de ellas en la plataforma, para dar cumplimiento con los objetivos planteados en la investigación, ejecutando de esta manera la puesta en marcha de la idea planteada.

CAPITULO IV

ANALISIS Y RESULTADOS

Fase Inicial del Proyecto

El presente capitulo refiere al análisis y resultados de las fases de la investigación planteadas en el capítulo anterior. Se presentan de manera estructura en relación a los aspectos relacionados con la primera fase de exploración (mapeo del ecosistema de emprendimiento de Ignis Gravitas) y aspectos generales que involucran al patrón MVC tales como rutas, modelo base (Customer) y controlador que envuelven el espacio de desarrollo del proyecto, tomando como guía el documento de requisitos del producto, cuyo propósito fue dar repuestas a los objetivos planteados en la investigación.

4.1. Fase de Exploración

4.1.1. Mapeo del Ecosistema

El ecosistema de Ignis Gravitas está conformado por emprendedores y en este caso en particular se observaron las entidades que la conforman, entre las que podemos mencionar: al Fundador y CEO de la plataforma Dr. Gerard Páez Monzón, los cofundadores que son los principales interesados en la dinámica del sistema completo de la plataforma y quienes poseen la visión y aseguran que la estrategia para la cual ha sido creada exista y evolucione; así como el conjunto de programadores y colaboradores .

En relación a la investigación, al realizar el mapeo a nivel de códigos de programación de la plataforma Ignis Gravitas, se pudo observar que cuenta con un gran número de líneas de códigos, entre los que se pueden indicar: el MVC de Customer quien es el responsable de proporcionar el espacio adecuado para el desarrollo del módulo Tren de Clientes. Estos códigos se construyeron gracias a la astucia y conocimiento de diversos programadores que forman el equipo de desarrollo, contando con ingeniería de requisitos; ya que es una plataforma compleja, extensa y que utiliza distintas tecnologías para su buen funcionamiento.

Se pueden señalar, una gran cantidad de fortalezas que envuelven el apartado de Customer, entre ellas el área que se utilizó para desarrollar e implementar el módulo tren de clientes, esto

representó una ventaja al momento de desarrollar el producto, pues se percibió la presencia definida del modelo inicial (@Customer), con su controlador y su vista principal sin olvidar sus accesos a rutas prestablecidas (rutas iniciales al momento de comenzar a desarrollar el módulo) y las bases de datos predefinidas para lo que requiera la plataforma (PostgreSQL).

Es de resaltar, que Ruby on Rails utiliza SQLite por defecto, en Ignis Gravitas se estableció el uso de PostgreSQL para satisfacer las demandas de los productos que contiene. Por otra parte, se observó en el mapeo de la plataforma las estructuras de carpetas y archivos de códigos. Todo esto permitió, conocer la situación real de la plataforma tanto a nivel interno (codificación) como externo (interfaz).

4.1.2. Inicio del proyecto

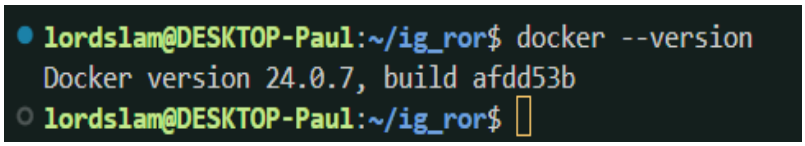
Una vez verificado que se encontraban instaladas las herramientas necesarias en el dispositivo donde se desarrolló el producto Tren de Clientes, se destacó como primera herramienta esencial: Docker, que es un software de código abierto que permite desplegar aplicaciones dentro de contenedores; dichos contenedores fungen como máquinas virtuales ligeras, requiriendo menos recursos de hardware de donde se ejecutó la plataforma local. Posteriormente a esto se procedió a la instalación del Docker en la estación de trabajo (PC).

Pasos para la instalación de Docker:

- Se actualizó la lista de paquetes existente con **sudo apt update**.
- Se instalaron algunos paquetes de requisitos previos que permitió al **apt** usar paquetes, a través de **HTTPS,** con el comando **sudo apt install apt-transport-https ca-certificates curl software-propretores-common.**
- Luego se añadió la clave de **GPG** para el repositorio oficial de Docker en el sistema, **curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add –**
- Se agregó el repositorio de Docker a las fuentes de **APT sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"**
- Posteriormente, se actualizo el paquete de datos con los paquetes de Docker **sudo apt update,** desde el repositorio de Docker en lugar del repositorio predeterminado de **Ubuntu apt-cache policy docker-ce**

- Finalmente, para instalarlo se utilizó el comando **sudo apt install docker-ce.** Con esto Docker se instaló en el sistema **(Ubuntu con WSL2).**
- Por último, se verifico la versión de Docker con **docker –versión**

Figura 2. Versión de Docker



Pasos para instalar el local de Ignis Gravitas:

- Para construir la imagen de Docker. **docker build -t ig_ror_app.**
- **docker-compose up** o **docker compose up** (Ejecute el servidor de desarrollo)
- **docker run --rm -it --entrypoint /bin/bash ig_ror-app** (para conectarse con el shell de Docker, si se requiere).

Con estos pasos se logró levantar localmente IG. Si, se requieren pasos adicionales se podrán solucionar según lo que muestre la consola de Ubuntu en WSL2. Obligatoriamente, para lograr tener un **USER** en la plataforma se necesitaron una serie de pasos adicionales que fueron los siguientes:

- Crear una cuenta en https://mailtrap.io/
- Tomar los valores de MAILTRAP_USERNAME y MAILTRAP_PASSWORD desde Mailtrap.
- En **docker-compose.yml** se remplazó los parámetros correspondientes por los obtenidos en Mailtrap, en la siguiente sección:

  environment:
  - PG_DB_NAME=PG_DB_NAME
  - PG_USERNAME=PG_USERNAME
  - PG_PASSWORD=PG_PASSWORD MAILTRAP_USERNAME=XXXX
  - MAILTRAP_PASSWORD=XXXX
  - MAIL_ACCOUNT="test1234@test.com"
  - MAIL_PASSWORD="12345678"

- STRIPE_PUBLISHABLE_KEY= XXXXXXXXXXXX
- STRIPE_SECRET_KEY=XXXXXXXXXXXX

- Descomentar el área comentada dentro de docker-compose.yml, luego de haber realizado los pasos para dejar operativo el local de Ignis Gravitas.
- Desde el directorio de ig_ror hacer Docker compose-up o docker compose up

En este orden de ideas, analizado el documento de requisitos del producto (TOC) y verificadas las herramientas necesarias para levantar localmente, se ubicó la rama de desarrollo para el Tren de Clientes en el apartado de códigos que contempla la plataforma. Lo primero que se vio por conveniencia fue la ubicación de la vista (View) de @Customer, ubicada en: **app/views/customers**; pero como conocemos un Modelo Vista Controlador (MVC) no contiene solo vistas, también contiene modelos y controladores, partes esenciales para la construcción del producto y se observó el comportamiento de las distintas pruebas a priori que se le realizaron.

Volviendo al punto de partida, tenemos el siguiente archivo el cual tiene el nombre de **index.html.erb**, es un archivo con compatibilidad de uso de plantillas tipo HTML (HyperText Markup Language) y erb (Embedded Ruby); el cual se encontró en **app/views/customers/index.html.erb,** de allí se desprende el desarrollo de cada una de las estaciones, siendo Customer la base de todas las estaciones ejecutadas. Una vez en el index se pudo experimentar un “HelloWorld”, para verificar que el apartado estuviese en orden.

Figura 3. Código Hola Mundo desde Ruby on Rails

```
<h1>Hola Mundo desde Ruby on Rails -HTML-</h1>
<div style="text-align: center;">
    <%= "Hola Mundo desde Ruby on Rails -erb-" %>
</div>
```

Resultando la aplicación Hola Mundo generada automáticamente por Rails;

Figura 4. Vista Hola Mundo

Siendo satisfactorio, la combinación dinámica de estos dos, dio como resultado un archivo HTML "puro" para su renderizado, y erb para el desarrollo.

4.1.3. Consideraciones de inicio:

Una vez configurado correctamente el inicio de Ignis Gravitas en forma local, se tomó en cuenta algunas configuraciones (en este caso particular) para el adecuado despliegue (desarrollo) del TOC, que son las siguientes:

- Detener el proceso de **PostgreSQL** en la consola de Ubuntu (WSL2), para ello se utilizó el siguiente comando : **sudo lsof -i tcp:5432**, obteniendo lo que se muestra en la ilustración

Figura 5. Comando sudo lsof-i tcp:5432

```
lordslam@DESKTOP-Paul:~/ig_ror$ sudo lsof -i tcp:5432
[sudo] password for lordslam:
COMMAND  PID     USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
postgres 286 postgres    5u  IPv4  19143      0t0  TCP localhost:postgresql (LISTEN)
lordslam@DESKTOP-Paul:~/ig_ror$ sudo kill 286
```

- Posteriormente, se usó el siguiente comando: **sudo kill [PID],** en este caso el PID fue 286

Si no liberamos el PID del postgres, saldrá algo como se muestra en la figura 2:

Figura 6. Comando docker compose up

```
lordslam@DESKTOP-Paul:~/ig_ror$ docker compose up
[+] Running 3/0
 ✓ Container ig_ror-app-1        Created                                    0.0s
 ✓ Container ig_ror-webpacker-1  Created                                    0.0s
 ✓ Container ig_ror-db-1         Created                                    0.0s
Attaching to ig_ror-app-1, ig_ror-db-1, ig_ror-webpacker-1
Error response from daemon: driver failed programming external connectivity on endpoint ig_ror-db
-1 (4c2cccf0cbd5c5118524d470a1467d8dec0ee295998d65845d333b3b403fbc3d): Error starting userland pr
oxy: listen tcp4 0.0.0.0:5432: bind: address already in use
```

De tal forma, que podría ejecutar con éxito el **docker compose up**, que sirve para inicializar los containers necesarios para poder levantar la plataforma a nivel local (localhost:3000).

4.1.4. Configuración de la Base de datos.

Como se señaló anteriormente, Ruby on Rails permite utilizar un motor de base de datos, que por defecto es SQLite, que es un motor de base de datos ligero e independiente. En este caso, se utiliza **PostgreSQL** que soporta tipo de datos relacionales y no relacionales, es robusto, fiable y posee soportes de estándares técnicos abiertos. La ubicación del archivo **database.yml** (la configuración de la base de datos) se encuentra en **config/database.yml**.

4.1.5. Definición del modelo inicial

El modelo inicial ya estaba presente en Ignis Gravitas, en la ubicación **app/models/customer.rb** cuyo nombre del archivo es **customer.rb** (archivo de Ruby), para este fin, el modelo contiene definiciones esenciales en el manejo de las ocho estaciones presentes en el tren de clientes a nivel relacional como se ilustra en la siguiente figura --

Figura 7. app/models/customer.rb

Las líneas de códigos que muestran la figura 7, son necesarias para el TOC, y se definen a continuación:

- **has_many: cardfgs, dependent: :destroy**: un cliente puede tener muchas cardfgs. Si se elimina un cliente, todas sus cardfgs también se eliminarán.
- **Has one: firstclient, dependent: destroy y accepts_nested_attributes_for :firstclient:** se refiere a que un cliente tiene un firstclient. Si se elimina un cliente, su firstclient también se eliminará. Además, se pueden aceptar atributos anidados para firstclient.
- **has_one: climber, dependent: destroy y accepts_nested_attributes_for :climber**: similar a firstclient, un cliente tiene un climber. Si se elimina un cliente, su climber también se eliminará. Además, se pueden aceptar atributos anidados para climber.
- **has_many :firstcgs, dependent: :destroy:** es similar a cardfgs, un cliente puede tener muchos firstcgs. Si se elimina un cliente, todos sus firstcgs también se eliminarán.
- **has_many :earlyadopters, dependent: :destroy:** parecido a cardfgs, un cliente puede tener muchos earlyadopters. Si se elimina un cliente, todos sus earlyadopters también se eliminarán.
- **has_many :chasm_productos, dependent: :destroy:** También muy similar a cardfgs, un cliente puede tener muchos chasm_productos. Si se elimina un cliente, todos sus chasm_productos también se eliminarán.

Validaciones dentro de customer.rb (Modelo de negocios):

En este caso tenemos presentes dos validaciones:

- **validates :nombre, presence: true, uniqueness: { message: "El SdeC ya existe, por favor usa otro nombre." }**: esto valida que el campo nombre esté presente y sea único. Si el nombre ya existe, se mostrará el mensaje de error “El SdeC ya existe, por favor usa otro nombre.”.
- **validates_each :nombre do |record, attr, value| record.errors.add(attr, 'must start with upper case') if value =~ /\A[[:lower:]]/ end**: esto valida que el campo nombre comience con una letra mayúscula. Si el nombre comienza con una letra minúscula, se agregará un error al registro. El cliente deberá usar una letra mayúscula en el siguiente intento para agregar el nombre del SdeC (Segmento de Cliente).

Igualmente la figura 7, muestra ejemplos de uso (comentados), estos son los siguientes:

- **Customer. Create(nombre: "John Doe").valid? # => true:** este es un ejemplo de cómo crear un nuevo cliente. En este caso, el nombre es “John Doe”, que es válido porque comienza con una letra mayúscula.
- **Customer.create(nombre: nil).valid? # => false:** este es un ejemplo de intentar crear un nuevo cliente sin un nombre, lo cual no es válido.

4.1.6. Definición del Controlador Inicial

El controlador inicial de nombre **customers_controller.rb** se ubica en **app/controllers/customers_controller.rb.** En las siguientes tres figuras ---, ---,-- se observa cómo se definió el controlador **customers_controller.rb**

Figura 8. app/controllers/customers_controller.rb (parte 1)

```ruby
class CustomersController < ApplicationController


def index
  authorize! :show, current_startup if current_startup

end

def index
  @customers = Customer.all
  if @customers.empty?
    flash[:notice] = "Crea tu SdeC "
    redirect_to new_customer1_path
  end
end

def new
  @customer = Customer.new
  @customer.build_firstclient
  #@climber = @customer.climber.build
end

#crea una nueva tarjetea de cardfg
def newcard
  @customer = Customer.find(params[:id])
  @cardfgs = @customer.cardfgs
  @cardfgs = Cardfg.new
end

def observational
  @customers = Customer.all
  @customer = Customer.find(params[:id])
  if @customers.empty?
    flash[:notice] = "Crea tu SdeC "
    redirect_to new_customer1_path
  end

end

def focusg
  @customers = Customer.all
  @customer = Customer.find(params[:id])

  # Si params[:cardfg_id] está presente, encuentra el Cardfg específico.
  # De lo contrario, obtén el primero.
  @cardfg = if params[:cardfg_id].present?
              @customer.cardfgs.find(params[:cardfg_id])
            end

  if @customers.empty?
    flash[:notice] = "Crea tu SdeC"
    redirect_to new_customer1_path and return
  end
end

def firstclient
  @customers = Customer.all
@customer = Customer.find(params[:id])
@firstclient = @customer.firstclient || @customer.build_firstclient
end
```

Figura 9. app/controllers/customers_controller.rb (parte 2)

```ruby
  #Escalamiento, anidado para que funcionen parametros de customer en la estacion respectiva
  def climber
   @customers = Customer.all
   @customer = Customer.find(params[:id])
   @climber = @customer.climber || @customer.build_climber
  end

 #debo asegurarme de esto para obtener el id y prepararlo desde la vista de customer a firstcg new
  def firstcg
    @customer = Customer.find(params[:id])
  end

  def earlyadopter
    @customer = Customer.find(params[:id])
    if @customer.nil?
      redirect_to customers_path, alert: 'Cliente no encontrado'
    else
      @earlyadopter = @customer.earlyadopters.order(created_at: :desc).first
      @earlyadopters = @customer.earlyadopters
      @total_ntotal = @customer.earlyadopters.sum(:ntotal)
    end
  end

  def earlyadopter
    @customer = Customer.find(params[:id])
    @earlyadopter = @customer.earlyadopters.first # Asegúrate de que esto no sea nil
    @earlyadopters = @customer.earlyadopters # Asegúrate de que esto no sea nil
  end

  def chasm
    @customer = Customer.find(params[:id])
    @chasm_producto = @customer.chasm_productos.find_by(id: params[:chasm_producto_id])
    @chasm_producto = @customer.chasm_productos.find_by(id: params[:chasm_producto_id])
    if @customer.nil?
      redirect_to customers_path, alert: 'Cliente no encontrado'
    else
      @chasm_productos = @customer.chasm_productos
  end
  end
```

Figura 10. app/controllers/customers_controller.rb (parte 3)


```ruby
  def create
    @customer = Customer.new(customer_params)
    if @customer.save
      flash[:notice] = "SdeC creado correctamente"
      redirect_to customers_path
    else
      flash[:alert] = @customer.errors.full_messages.join(",")
      render :new
    end
  end

  def edit
    @customer = Customer.find(params[:id])
  end

  def update
    @customer = Customer.find(params[:id])
    if @customer.update(customer_params)
      flash[:notice] = "SdeC actualizado correctamente"
      redirect_to customers_path
    else
      render 'edit'
    end
  end

  def destroy
    @customer = Customer.find(params[:id])
    @customer.destroy
    flash[:notice] = "SdeC eliminado correctamente junto con todos los registros referenciados."
    redirect_to customers_path
  rescue ActiveRecord::InvalidForeignKey => e
    flash[:alert] = "Hubo un problema al eliminar el SdeC y sus registros asociados."
    redirect_to customers_path
  end

  def set_customer_id
    session[:selected_customer_id] = params[:customer_id]
    Rails.logger.info "session[:selected_customer_id] is now #{session[:selected_customer_id]}"
    head :ok
  end



  private

  def customer_params
    params.require(:customer).permit(:nombre, :siempre_rara_vez, :descripcion_actividad, :barral, :descripcion_deseos, :descripcion_molestias, :descripcion_cultura, :descripcion_propuestas, :descripcion_alivios, :descripcion_satisfacciones, :descri
  end
end
```

Las figuras 8,9 y 10, muestran el controlador, maneja las acciones relacionadas con los objetos Customer.

Definiciones para este controlador son las siguientes:

- **index:** muestra todos los clientes. Si no hay clientes, redirige al usuario a la página de creación de un nuevo cliente.
- **new:** inicializa un nuevo objeto Customer y construye un objeto Firstclient asociado.
- **newcard:** inicializa un nuevo objeto Cardfg asociado a un Customer específico.
- **observational, focusg, firstclient, climber, firstcg, earlyadopter, chasm:** estas acciones buscan un Customer específico y realizan operaciones adicionales dependiendo de la acción. Por ejemplo, focusg busca un Cardfg específico si se proporciona params[:cardfg_id].
- **create:** crea un nuevo Customer con los parámetros proporcionados. Si la creación es exitosa, redirige al usuario a la página de índice de clientes. Si no, muestra los errores y vuelve a renderizar la página de creación.
- **edit:** busca un Customer específico para editar.
- **update:** Actualiza un Customer específico con los parámetros proporcionados. Si la actualización es exitosa, redirige al usuario a la página de índice de clientes. Si no, vuelve a renderizar la página de edición.
- **destroy:** Elimina un Customer específico y redirige al usuario a la página de índice de clientes. Si ocurre un error (por ejemplo, debido a una clave foránea inválida), muestra un mensaje de error.
- **set_customer_id:** establece session[:selected_customer_id] al customer_id proporcionado.
- **session[:selected_customer_id]** es una forma de almacenar datos específicos del usuario entre múltiples solicitudes en una aplicación Ruby on Rails. Cuando usas **session[:selected_customer_id],** estás utilizando la sesión del usuario para almacenar el id de un cliente seleccionado. Las sesiones son útiles porque HTTP, el protocolo en el que se basan las aplicaciones web, es sin estado, lo que significa que no mantiene información entre solicitudes. Las sesiones proporcionan una forma de mantener la información de un usuario de una solicitud a otra. En el caso de session[:selected_customer_id],

probablemente se está almacenando el id de un cliente que el usuario ha seleccionado de alguna manera, para que se pueda recordar esa selección en futuras solicitudes. Por ejemplo, se podría usar esto para recordar qué cliente está viendo actualmente el usuario, incluso si navega a diferentes páginas o realiza diferentes acciones. Es importante tener en presente que los datos de la sesión se almacenan en el lado del servidor y se envían al cliente como una cookie. Esto significa que, aunque puedes confiar en que los datos de la sesión no serán modificados por el usuario, hay que tener cuidado de no almacenar datos sensibles en la sesión, ya que la cookie podría ser interceptada y leída.

Método privado del controlador inicial customer.rb:

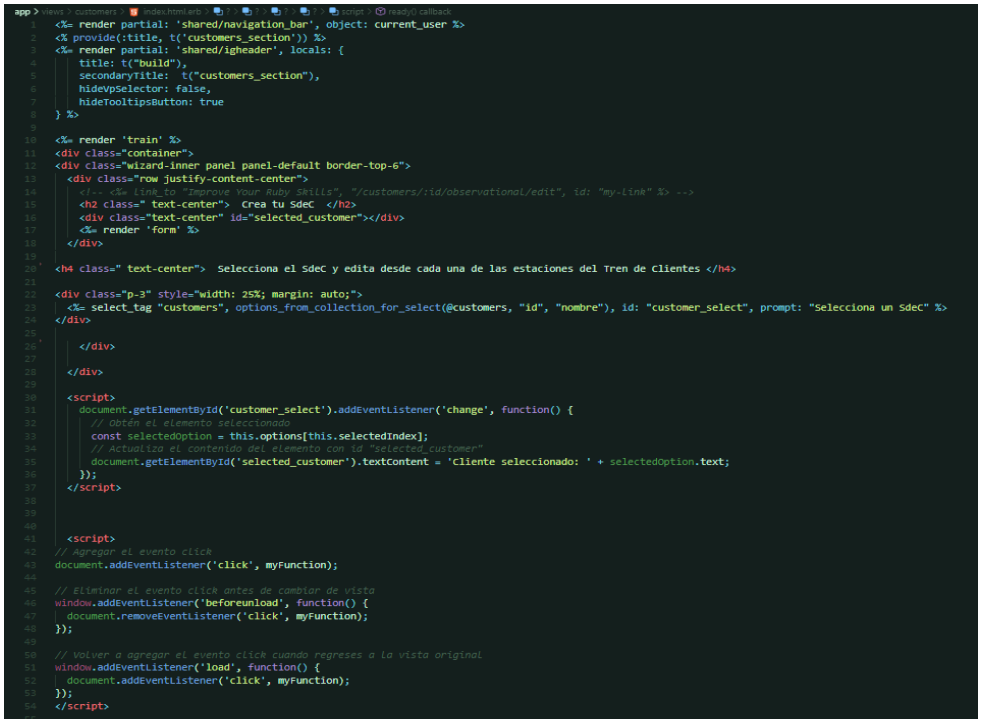
- **customer_params**: este método se utiliza para la seguridad de los parámetros. Permite a la aplicación aceptar ciertos parámetros del formulario de Customer (para la estación observacional) y rechazar otros. Esto ayuda a prevenir ataques de asignación masiva. Estos parámetros pertenecen a la primera estación: Observacional. Están en este espacio para aprovechar el **customer_params**, funciona de buena manera ya que el modelo inicial está relacionado a la primera estación, que carece de su propio controlador.

4.1.7. Vistas de customers

La vista de Customer está determinada por **index.html.erb (**ver anexo B1), ubicada en **app/views/customers**. En este apartado se construyó lo necesario para crear, editar y eliminar el SdeC (segmentos de clientes). A través de un seleccionador de etiqueta se puede elegir entre los SdeC creados y utilizar las distintas estaciones para ese segmento de cliente en específico. Desde acá, el usuario tiene todo lo necesario para manejar los distintos SdeC, igualmente, a través de un panel podrá desplazarse a través de las ocho estaciones que se incluyen en el TOC.

Es preciso definir que este panel de estaciones se llama a través de un render ubicado en **app/views/customers**, cuyo archivo exacto es **app/views/customers/_train.html.erb**, los archivos de tipo _archivo.html.erb tienen una gran ventaja, ya que permiten construir una respuesta a una solicitud, es decir, podemos llamar a este fragmento de código en todas las vistas que se requieran, _train.html.erb se llama a través de <%= render 'train' %> para que se pueda visualizar en la vista deseada, cumpliendo con un principio de no repitas código **DRY(**Don't Repeat Yourself), adicionalmente permitiendo ahorrar líneas de código, ventaja que siempre se agradece.

Figura 11. index.html.erb (parte uno)



```
app > views > customers > index.html.erb > ? > ? > ? > ? > ? > ? > script > ready() callback
<%= render partial: 'shared/navigation_bar', object: current_user %>
<% provide(:title, t('customers_section')) %>
<%= render partial: 'shared/igheader', locals: {
    title: t("build"),
    secondaryTitle:  t("customers_section"),
    hideVpSelector: false,
    hideTooltipsButton: true
} %>

<%= render 'train' %>
<div class="container">
<div class="wizard-inner panel panel-default border-top-6">
  <div class="row justify-content-center">
    <!-- <%= link_to "Improve Your Ruby Skills", "/customers/:id/observational/edit", id: "my-link" %> -->
    <h2 class=" text-center">  Crea tu SdeC  </h2>
    <div class="text-center" id="selected_customer"></div>
    <%= render 'form' %>
  </div>

<h4 class=" text-center">  Selecciona el SdeC y edita desde cada una de las estaciones del Tren de Clientes </h4>

<div class="p-3" style="width: 25%; margin: auto;">
  <%= select_tag "customers", options_from_collection_for_select(@customers, "id", "nombre"), id: "customer_select", prompt: "Selecciona un SdeC" %>
</div>

    </div>

  </div>

  <script>
    document.getElementById('customer_select').addEventListener('change', function() {
      // Obtén el elemento seleccionado
      const selectedOption = this.options[this.selectedIndex];
      // Actualiza el contenido del elemento con id "selected_customer"
      document.getElementById('selected_customer').textContent = 'Cliente seleccionado: ' + selectedOption.text;
    });
  </script>



  <script>
// Agregar el evento click
document.addEventListener('click', myFunction);

// Eliminar el evento click antes de cambiar de vista
window.addEventListener('beforeunload', function() {
  document.removeEventListener('click', myFunction);
});

// Volver a agregar el evento click cuando regreses a la vista original
window.addEventListener('load', function() {
  document.addEventListener('click', myFunction);
});
</script>
```


Figura 12. index.html.erb (parte dos)

```
<script>
  $(document).ready(function() {
    // Verificar si hay un customer_id guardado y seleccionarlo en el select_tag
    var savedCustomerId = localStorage.getItem('selectedCustomerId');
    if (savedCustomerId) {
      $('#customer_select').val(savedCustomerId);
      // Actualiza el contenido del div basado en la opción seleccionada
      var selectedOptionText = $('#customer_select option:selected').text();
      document.getElementById('selected_customer').textContent = 'Cliente seleccionado: ' + selectedOptionText;
    }

    $('#customer_select').change(function() {
      var customer_id = $(this).val();

      if (customer_id) {
        // Guardar el id del cliente en el almacenamiento local
        localStorage.setItem('selectedCustomerId', customer_id);

        // Guardar el id del cliente en la sesión
        $.ajax({
          type: 'POST',
          url: '/set_customer_id',
          data: { customer_id: customer_id },
          success: function() {
            // Recargar la página para actualizar los enlaces
            location.reload();
          }
        });
      } else {
        // Si se selecciona "Selecciona un SdeC" (valor vacío), redirigir a la lista de clientes
        window.location.href = '/customers';
      }
    });
  });
</script>
```

Definiciones de la vista principal index.html.erb, figuras 11 y 12:

- **Renderizado de Parciales:** los parciales son vistas que se pueden reutilizar en diferentes partes de la aplicación. En este caso, se está renderizando los parciales **shared/navigation_bar, shared/igheader, y train.**
- **Contenedor Principal:** Todo el contenido de la página está envuelto en un div con la clase container. Dentro de este contenedor, hay otro div con las clases wizard-inner panel panel-default border-top-6.
- **Formulario de Creación de Cliente:** dentro del contenedor principal, hay un formulario para crear un nuevo cliente. Este formulario se renderiza a través del parcial form (estación Observacional).
- **Selección de Cliente:** Debajo del formulario, hay un select_tag que permite al usuario seleccionar un cliente de una lista de clientes existentes. Cuando el usuario selecciona un cliente, se muestra el nombre del cliente seleccionado y se guarda el id del cliente en el almacenamiento local y en la sesión.
- **Scripts:** Hay varios scripts en la página que manejan cosas como la selección de clientes, la eliminación de mensajes flash después de un cierto tiempo, y la adición y eliminación de eventos de clic.

Método privado del controlador inicial customer.rb:

- **customer_params**: este método se utiliza para la seguridad de los parámetros. Permite a la aplicación aceptar ciertos parámetros del formulario de Customer (para la estación observacional) y rechazar otros. Esto ayuda a prevenir ataques de asignación masiva. Estos parámetros pertenecen a la primera estación: Observacional. Están en este espacio para aprovechar el **customer_params**, funciona de buena manera ya que el modelo inicial está relacionado a la primera estación, observacional. La primera estación carece de su propio controlador.

Definición de los scripts:

Primer script:

- **document.addEventListener('click', myFunction);:** Agrega un controlador de eventos al documento completo. Cada vez que se hace clic en el documento, se llama a la función **myFunction.**
- **window.addEventListener('beforeunload', function(){document.removeEventListener('click', myFunction); });:** Este código agrega un controlador de eventos que se activa justo antes de que la página se vaya a descargar (por ejemplo, cuando el usuario navega a otra página). Cuando esto sucede, se elimina el controlador de eventos de clic del documento.

Segundo script:

- **$(document).ready(function() {...});:** Se ejecuta cuando el documento HTML ha sido completamente cargado y el árbol DOM está construido.
- **var savedCustomerId = localStorage.getItem('selectedCustomerId');:** Recupera el valor de selectedCustomerId del almacenamiento local del navegador, si existe.
- **$('#customer_select').val(savedCustomerId);:** Este código establece el valor del elemento con id **customer_select** al **id** del cliente guardado.
- **document.getElementById('selected_customer').textContent = 'Cliente seleccionado: ' + selectedOptionText;:** Este código actualiza el contenido del elemento con id selected_customer para mostrar el nombre del cliente seleccionado.
- **$('#customer_select').change(function() {...});:** Este código agrega un controlador de eventos al elemento con id customer_select que se activa cuando el valor del elemento cambia.
- **localStorage.setItem('selectedCustomerId', customer_id);:** Guarda el id del cliente seleccionado en el almacenamiento local del navegador.
- **$.ajax({...});:** Realiza una solicitud AJAX al servidor para guardar el id del cliente en la sesión del servidor.
- **window.location.href = '/customers';:** Redirige al usuario a la página de lista de clientes si se selecciona la opción “Selecciona un SdeC” (que tiene un valor vacío).

Nota: los scripts presentes bien pueden situarse en **app/assets/javascripts**; sin embargo, por fines de desarrollo y pruebas a priori se mantienen en su ubicación actual, en un futuro se podrían ubicar respectivamente para dar una imagen más limpia al código, esto no afecta su funcionalidad y es práctico para hacer una modificación inmediata de los mismos <script>.

4.1.8. Vista de _train.html.erb

Esta vista está ubicada en **app/views/customers/_train.html.erb** (anexo B.1), cuyo archivo es _train.html.erb, esta sección de código contiene Ruby on Rails junto con HTML, CSS y JavaScript se define de la siguiente manera:

Figura 13. app/views/customers/_train.html.erb (parte uno)

Figura 14. app/views/customers/_train.html.erb (parte dos)

```
@keyframes spin {
  0% {
    transform: rotate(0deg);
  }
  100% {
    transform: rotate(360deg);
  }
}

.rc-steps-icon {
  color: #e0895a;
}
  /* Estilos generales aquí */
  /* Media query para dispositivos móviles */
  @media (max-width: 767px) {
    .row > div {
      overflow: auto; /* Añade barras de desplazamiento si es necesario */
    }
  }

  /* Media query para tablets */
  @media (min-width: 768px) and (max-width: 991px) {
    .row > div {
      overflow: auto; /* Añade barras de desplazamiento si es necesario */
    }
  }

</style>
```

Figura 15. app/views/customers/_train.html.erb (parte tres)

```
<div class="container">
  <div class="container-fluid customer">
    <div class="row">
      <!-- Control sidebar -->
      <div class="wizard product col-xs-12">

        <div class="connecting-line" style="width: 66.6667%;">
        </div>

        <ul role="tablist" class="nav-tabs nav">
        <li role="presentation" class="" style="width: 33.3333%;">
        <a title="Gestiona la sección de clientes de tu modelo de negocios actual" id="tabsss-tab-undefined" role="tab" aria-controls="tabsss-pane-undefined" aria-selected="false" href="#" tabindex="-1">
        <span class="round-tab" href="/es/customers/igcustomers">
        <h5 class="text-icon">Clientes</h5><i class="fa fa-user"></i></span></a></li>
        <li role="presentation" class="active" style="width: 33.3333%;">
        <a title="Establece el estado de tu startup con respecto al número de clientes y usuarios que tienes" id="tabsss-tab-undefined" role="tab" aria-controls="tabsss-pane-undefined" aria-selected="true" href="#">
        <span class="round-tab active" href="/es/customers/toc"><h5 class="text-icon">Tren de Clientes</h5>
        <i class="fa fa-train"></i></span></a></li><li role="presentation" class="" style="width: 33.3333%;">
        <a title="Evalúa tus hipótesis de negocio con experimentos" id="tabsss-tab-undefined" role="tab" aria-controls="tabsss-pane-undefined" aria-selected="false" href="#" tabindex="-1">
        <span class="round-tab" href="/es/customers/igbusiness">
        <h5 class="text-icon">Experimentos</h5>
        <i class="fa fa-compass">
        </i></span></a></li></ul>
        </div>
      </div>

      <div class="wizard-inner panel panel-default border-top-6">

        <div class="panel-body" title="Tren de Clientes" >

          <div class="container">
            <div class="row ">
              <div class="col-8"> <!-- Tamaño de columna, para ordenar elementos dentro de la misma. -->
                <div class="rc-steps rc-steps-horizontal rc-steps-label-vertical ">

                  <% steps = [
                    { title: 'Observacional', path: session[:selected_customer_id].present? ? edit_customer_observational_path(session[:selected_customer_id]) : nil },
                    { title: 'Enfoque en Grupo', path: session[:selected_customer_id].present? ? edit_customer_focusg_path(session[:selected_customer_id]) : nil },
                    { title: 'Primer Cliente', path: session[:selected_customer_id].present? ? edit_customer_firstclient1_path(session[:selected_customer_id]) : nil },
                    { title: 'Primer Grupo de Clientes', path: session[:selected_customer_id].present? ? edit_customer_firstg1_path(session[:selected_customer_id]) : nil },
                    { title: 'Primeros Adoptantes', path: session[:selected_customer_id].present? ? edit_customer_early&adopter1_path(session[:selected_customer_id]) : nil },
                    { title: 'Abismo', path: session[:selected_customer_id].present? ? edit_customer_chasm1_path(session[:selected_customer_id]) : nil },
                    { title: 'Escalamiento', path: session[:selected_customer_id].present? ? edit_customer_climber1_path(session[:selected_customer_id]) : nil },
                    { title: 'Empresa', path: session[:selected_customer_id].present? ? edit_customer_enterprise1_path(session[:selected_customer_id]) : nil  }
                    ] %>

                                <% steps.each_with_index do |step, index| %>
                    <div class="rc-steps-item <% index < 8 ? 'rc-steps-item-process' %> stage-item-<%= step[:title].parameterize.underscore %> step" style="animation-delay: <%= index * 1.50 %>s">
                      <div class="rc-steps-item-tail"></div>
                      <div class="rc-steps-item-icon">
                        <% if step[:path] %>
                          <%= link_to raw("<span class='rc-steps-icon'>#{index + 1}</span>"), step[:path], id: "customer_link#{index}", class: 'active-link' %>
                        <% else %>
                          <span class="rc-steps-icon rcicon rcicon-<%= step[:icon] %>"></span>
                        <% end %>
                      </div>
                      <div class="rc-steps-item-content">
                        <div class="rc-steps-item-title"><%= step[:title] %></div>
                        </div>
                        <div class="rc-steps-item-upper-content"></div>
                      </div>
                    <% end %>
                      </div>
                    </div>
                  </div>
                </div>
              </div>
            </div>

        </div> <!--fin inner-->
      </div>


</div>
```

Figura 16. app/views/customers/_train.html.erb (parte cuatro)

Las siguientes definiciones pertenecen a las figuras 13, 14, 15 y 16

Definiciones de _train.html.erb:

- ** ... :** Este es un contenedor principal que envuelve el contenido de la página.
- ** ... :** Este div contiene una barra de progreso o "wizard" que se utiliza para guiar al usuario a través de varios pasos o etapas.
- **<ul role="tablist" class="nav-tabs nav">...:** Esta es una lista desordenada que se utiliza para crear una serie de pestañas. Cada pestaña representa una sección diferente de la página.
- **<% steps = [...] %>:** Esta es una matriz de hashes en Ruby. Cada hash representa un paso en el proceso y contiene un título y una ruta.
- **<% steps.each_with_index do |step, index| %>...<% end %>:** Es un bucle que recorre cada paso en la matriz de pasos. Para cada paso, genera un elemento de la lista (<li>) que representa ese paso en la interfaz de usuario.

Definición del primer script de _train.html.erb:

Este script utiliza jQuery para agregar interactividad al entorno.

- **$(document).ready(function() {...}):** Este bloque de código se ejecuta cuando el documento HTML ha sido completamente cargado.
- **$('.rc-steps-item-icon a').on('click', function() {...}):** Agrega un controlador de eventos de clic a todos los enlaces (<a>) dentro de los elementos con la clase .rc-steps-item-icon. Cuando se hace clic en uno de estos enlaces, se agrega la clase 'loading' al elemento con la clase .wizard-inner.
- **$(window).on('load', function() {...}):** Se ejecuta cuando todos los recursos de la página (imágenes, scripts, etc.) han sido completamente cargados. Dentro de este bloque, se quita la clase 'loading' del elemento con la clase .wizard-inner.
- **window.onpopstate = function(event) {...}:** Se ejecuta cuando el usuario navega a través del historial del navegador. Si el elemento con la clase .wizard-inner tiene la clase 'loading', esta se quita después de un retraso de 20 milisegundos.

Definición del segundo script de _train.html.erb:

Este script utiliza JavaScript puro para agregar interactividad al entorno.

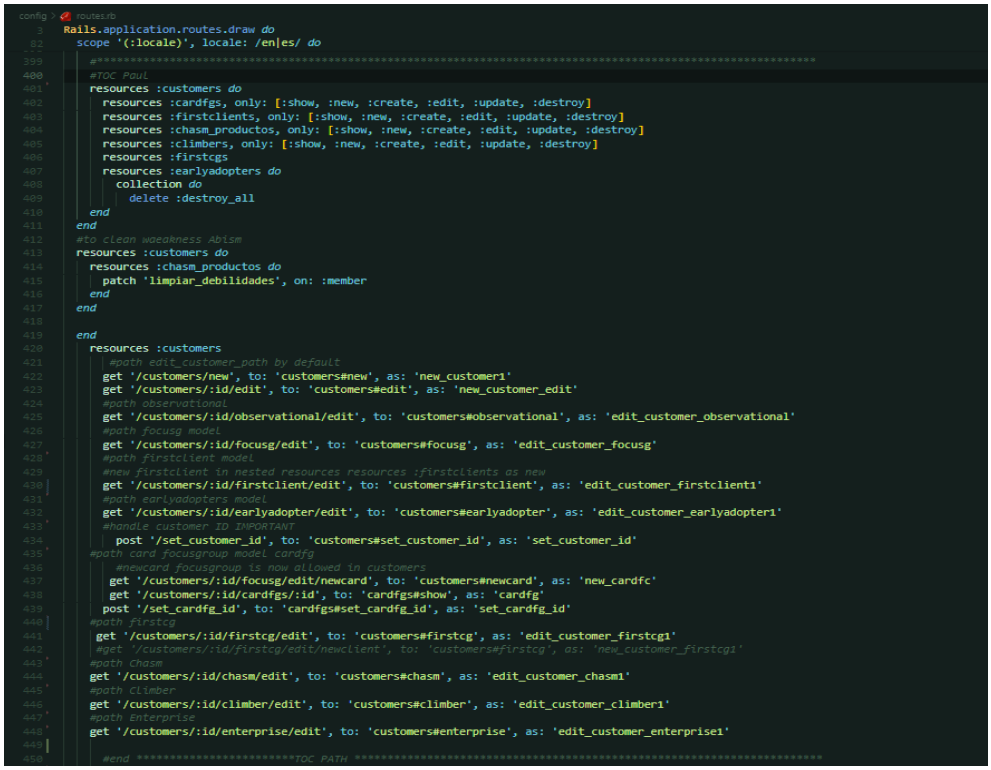
- **var steps = document.querySelectorAll('.rc-steps-item-icon'):** Esta línea selecciona todos los elementos con la clase .rc-steps-item-icon y los almacena en la variable steps.
- **steps.forEach(function(step, index) {...}):** Este bloque de código recorre cada elemento en la variable steps. Para cada elemento, verifica si el índice del elemento coincide con el valor almacenado en el almacenamiento local bajo la clave 'blinking'. Si es así, se agrega la clase 'blinking' al elemento.
- **step.addEventListener('click', function(event) {...}):** Agrega un controlador de eventos de clic a cada elemento en la variable steps. Cuando se hace clic en un elemento, se elimina la clase 'blinking' de todos los elementos y se agrega al elemento clickeado. Luego, se almacena el índice del elemento clickeado en el almacenamiento local bajo la clave 'blinking'.

- **definición de style de _train.html.erb:** @keyframes example {...} y @keyframes blink {...}: Estas son las definiciones de dos animaciones. La animación example cambia la opacidad de un elemento de 0 a 1, creando un efecto de desvanecimiento. La animación blink cambia la opacidad de un elemento de 1 a 0 y de nuevo a 1, creando un efecto de parpadeo.
- **.step {...}:** Aplica la animación example a los elementos con la clase step.
- **.blinking {...}:** Se aplica la animación blink a los elementos con la clase blinking. También añade una sombra de caja al elemento.
- **.active-link {...}:** Permite que los elementos con la clase active-link sean clicables y cambia el cursor a un puntero cuando se pasa el ratón sobre ellos.
- **.rc-steps-item-icon {...}:** Logra aplicar una transición y una transformación a los elementos con la clase rc-steps-item-icon. Cuando se pasa el ratón sobre estos elementos, su tamaño aumenta y su color cambia.
- **.wizard-inner.loading::before {...}:** Crea un pseudo-elemento antes de los elementos con las clases wizard-inner y loading. Este pseudo-elemento es un círculo que gira, creando un efecto de carga.
- **@media (max-width: 767px) {...} y @media (min-width: 768px) and (max-width: 991px) {...}:** Estos son los bloques de consulta de medios que aplican estilos específicos a los elementos dependiendo del ancho de la ventana del navegador. En este caso, añaden barras de desplazamiento a los elementos si es necesario.

4.2. Rutas

El archivo routes.rb se encuentra en **config/routes.rb**, este archivo gestiona todas las rutas de la aplicación (IG), figura como "mapa" de todas las URL disponibles en la aplicación e indica que controladores y acciones manejaran cada una de estas URL. En el caso del TOC, hay rutas definidas para el correcto manejo de las estaciones para cada uno de los SdeC que tengamos presentes.

Figura 17. config/routes.rb



```
config > routes.rb
Rails.application.routes.draw do
  scope '(:locale)', locale: /en|es/ do
    #*****************************************************************************************************
    #TOC_Paul
    resources :customers do
      resources :cardfgs, only: [:show, :new, :create, :edit, :update, :destroy]
      resources :firstclients, only: [:show, :new, :create, :edit, :update, :destroy]
      resources :chasm_productos, only: [:show, :new, :create, :edit, :update, :destroy]
      resources :climbers, only: [:show, :new, :create, :edit, :update, :destroy]
      resources :firstcgs
      resources :earlyadopters do
        collection do
          delete :destroy_all
    end
    end
    #to clean waeakness Abism
    resources :customers do
      resources :chasm_productos do
        patch 'limpiar_debilidades', on: :member
      end
    end

    end
      resources :customers
        #path edit_customer_path by default
        get '/customers/new', to: 'customers#new', as: 'new_customer1'
        get '/customers/:id/edit', to: 'customers#edit', as: 'new_customer_edit'
        #path observational
        get '/customers/:id/observational/edit', to: 'customers#observational', as: 'edit_customer_observational1'
        #path focusg model
        get '/customers/:id/focusg/edit', to: 'customers#focusg', as: 'edit_customer_focusg'
        #path firstclient model
        #new_firstclient_in_nested_resources :firstclients_as_new
        get '/customers/:id/firstclient/edit', to: 'customers#firstclient', as: 'edit_customer_firstclient1'
        #path earlyadopters_model
        get '/customers/:id/earlyadopter/edit', to: 'customers#earlyadopter', as: 'edit_customer_earlyadopter1'
        #handle customer_ID_IMPORTANT
          post '/set_customer_id', to: 'customers#set_customer_id', as: 'set_customer_id'
      #path card focusgroup model cardfg
        #newcard_focusgroup_is_now_allowed_in_customers
        get '/customers/:id/focusg/edit/newcard', to: 'customers#newcard', as: 'new_cardfc'
        get '/customers/:id/cardfgs/:id', to: 'cardfgs#show', as: 'cardfg'
      post '/set_cardfg_id', to: 'cardfgs#set_cardfg_id', as: 'set_cardfg_id'
    #path firstcg
      get '/customers/:id/firstcg/edit', to: 'customers#firstcg', as: 'edit_customer_firstcg1'
      #get '/customers/:id/firstcg/edit/newclient', to: 'customers#firstcg', as: 'new_customer_firstcg1'
    #path Chasm
    get '/customers/:id/chasm/edit', to: 'customers#chasm', as: 'edit_customer_chasm1'
    #path Climber
    get '/customers/:id/climber/edit', to: 'customers#climber', as: 'edit_customer_climber1'
    #path Enterprise
    get '/customers/:id/enterprise/edit', to: 'customers#enterprise', as: 'edit_customer_enterprise1'

      #end **********TOC_PATH ***********************************************************************
```


Las siguientes definiciones pertenecen a la figura 17

Definiciones de las rutas para el TOC:

- **Rutas Anidadas:**

Las rutas anidadas permiten modelar relaciones entre recursos. Por ejemplo, resources :customers do ... end, define un conjunto de rutas para el recurso customers. Dentro de este bloque, se pueden definir rutas para recursos relacionados, como cardfgs, firstclients, chasm_productos, climbers, firstcgs, y earlyadopters. Estas rutas permiten manejar URLs como /customers/:customer_id/cardfgs/: id.

- **Rutas de Recursos:**

resources :customers genera un conjunto de rutas que corresponden a las operaciones CRUD (crear, leer, actualizar, eliminar) en el recurso customers. Se pueden limitar las rutas generadas utilizando la opción only, como en resources :cardfgs, only: [:show, :new, :create, :edit, :update, :destroy].

- **Rutas Personalizadas:**

Además de las rutas de recursos, se pueden definir rutas personalizadas. Por ejemplo, get '/customers/:id/observational/edit', to: 'customers#observational', as: 'edit_customer_observational' define una ruta GET a /customers/:id/observational/edit que será manejada por la acción observational del controlador customers. La opción as permite nombrar la ruta para que se pueda diferenciar fácilmente en el código.

- **Rutas de Colección y Miembro:**

Las rutas de colección y miembro permiten añadir rutas adicionales que actúan en la colección o en los miembros individuales de un recurso. Por ejemplo, delete :destroy_all dentro del bloque collection do ... end define una ruta DELETE a /customers/:customer_id/earlyadopters/destroy_all que actúa en la colección de earlyadopters de un customer.

- **Rutas Anidadas Múltiples:**

También se pueden anidar rutas repetidas veces, como en resources :customers do resources :chasm_productos do patch 'limpiar_debilidades', on: :member end end. Esto define una ruta PATCH a /customers/:customer_id/chasm_productos/:id/limpiar_debilidades que será manejada por la acción limpiar_debilidades del controlador chasm_productos.

CAPÍTULO V

IMPLEMENTACIÓN Y DESPLIEGUE

Desarrollo y Ejecución de las Estaciones del Tren de clientes

En este apartado se describen y explican los bloques (líneas) de códigos de cada una de las estaciones, las cuales son producto del tratamiento de ellas, durante el proceso de investigación. En este caso se presenta el desarrollo e implementación de las estaciones que integran el Tren de Clientes (TOC); es de destacar que todas las estaciones se desarrollaron bajos los lineamientos descritos en el documento de requisitos del producto, en la línea de responsabilidades (anexo A), estrategia creada por los fundadores de Ignis Gravitas Inc.

5.1.- Primera estación: Observacional.

Su objetivo fundamental es identificar la actividad central, las necesidades, las molestias, los deseos, búsquedas de crecimientos cultural. Para esto, se construyeron una serie de formularios con el fin apoyar a los emprendedores.

5.1.1. Modelo de la Estación: Observacional

Es el mismo modelo de customer, ya que se aprovechó al máximo el espacio de customer.rb, dando a conocer a customer como un modelo que sostiene otros modelos; es decir, que observacional se sirve de customer. Customer solo proporciona sus espacios de desarrollo para dicha estación. Figura 18.

Figura 18. app/models/customer.rb.

```ruby
app > models > customer.rb
class Customer < ApplicationRecord

  has_many :cardfgs, dependent: :destroy
  has_one :firstclient, dependent: :destroy
  accepts_nested_attributes_for :firstclient
  has_one :climber, dependent: :destroy
  accepts_nested_attributes_for :climber
  has_many :firstcgs, dependent: :destroy
  has_many :earlyadopters, dependent: :destroy
  has_many :chasm_productos, dependent: :destroy


  #Valida que no este vacio el campo nombre.
  validates :nombre, presence: true, uniqueness: { message: "El SdeC ya existe, por favor usa otro nombre." }



  #Obliga al usuario a colocar la primera letra en mayuscula.
  validates_each :nombre do |record, attr, value|
    record.errors.add(attr, 'must start with upper case') if value =~ /\A[[:lower:]]/
  end
  #Customer.create(nombre: "John Doe").valid? # => true
  #Customer.create(nombre: nil).valid? # => false
end
```

5.1.2. Vista de la primera estación: Observacional

- **Primera vista _new_edit.html.erb:** es lo primero que se visualiza al entrar al Tren de Clientes, gracias a la condición codificada en el controlador que busca clientes existentes, si no existe al menos un cliente, redirige a la vista de _form.html.erb para crear el primer cliente. También entran en juego las validaciones codificadas en el modelo de customer. Esta vista además se usa para editar el nombre del SdeC seleccionado. Este archivo se encuentra en **app/views/customers/_new_edit.html.erb** (anexo B.2 y B.3).

Figura 19. app/views/customers/_new_edit.html.erb

- **Segunda vista _form.html.erb:** el archivo **_form.html.erb** (anexo B.1) es un parcial, contiene todo el apartado para la creación de un segmento de cliente (SdeC).

Figura 20. app/views/customers/_form.html.erb

Definicion de _form.html.erb:

- **.milist {...}:** Aplica estilos a los elementos con la clase milist. En este caso, oculta cualquier desbordamiento horizontal.
- **.milist li {...}:** Este bloque aplica estilos a los elementos <li> dentro de los elementos con la clase milist. Establece una transición en la propiedad transform, ajusta el origen de la transformación, establece la posición del elemento como relativa y establece su z-index en 1.
- **.milist li:hover {...}:** Se aplica estilos a los elementos <li> dentro de los elementos con la clase milist cuando se pasa el ratón sobre ellos. En este caso, aumenta su tamaño en un 20%.
- **Tercera vista observatinal.html.erb:** la tercera vista de la estación observacional es **app/views/customers/observational.html.erb** (anexo B.4), cuyo archivo es **observational.html.erb**. Contiene gran parte del código para poder generar una interfaz a observacional, en sus secciones (bloques) como circunstancias, planteamientos y soluciones.

Figura 21. app/views/customers/observational.html.erb (parte 1)

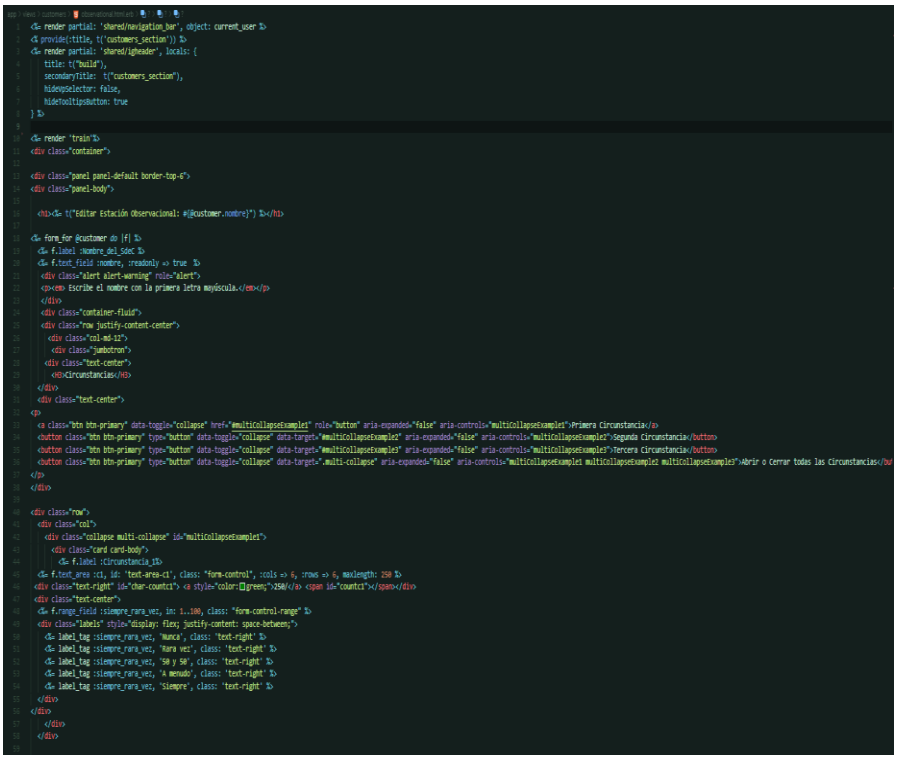


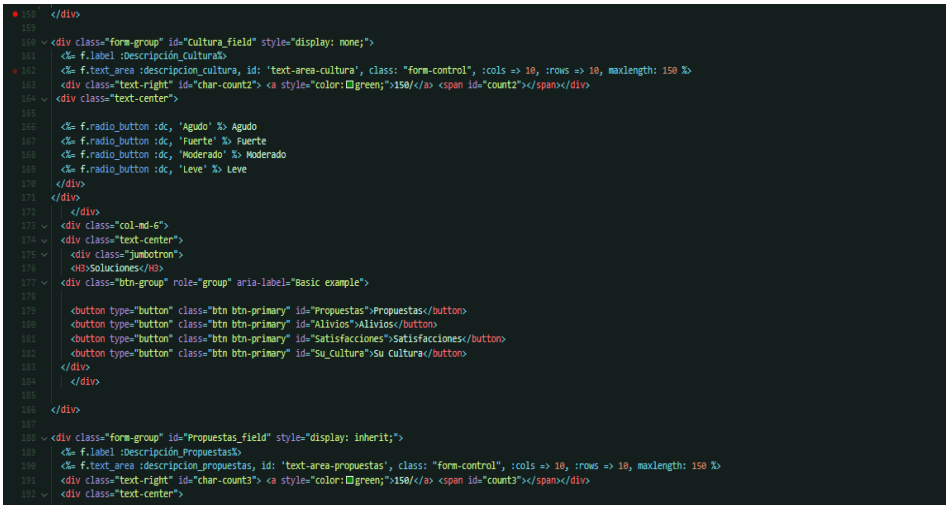
Figura 22. app/views/customers/observational.html.erb (parte 2)

```
<div class="col">
  <div class="collapse multi-collapse" id="multiCollapseExample2">
    <div class="card card-body">
      <%= f.label :circunstancia_2%>
 <%= f.text_area :c2, id: 'text-area-c2', class: "form-control", :cols => 6, :rows => 6, maxlength: 250 %>
<div class="text-right" id="char-countc2"> <a style="color:green;">250/</a> <span id="countc2"></span></div>
<div class="text-center">
 <%= f.range_field :barra2, in: 1..100, class: "form-control-range" %>
 <div class="labels" style="display: flex; justify-content: space-between;">
   <%= label_tag :barra2, 'Nunca', class: 'text-right' %>
   <%= label_tag :barra2, 'Rara vez', class: 'text-right' %>
   <%= label_tag :barra2, '50 y 50', class: 'text-right' %>
   <%= label_tag :barra2, 'A menudo', class: 'text-right' %>
   <%= label_tag :barra2, 'Siempre', class: 'text-right' %>
 </div>
</div>
    </div>
  </div>


  <div class="col">
    <div class="collapse multi-collapse" id="multiCollapseExample3">
      <div class="card card-body">
        <%= f.label :circunstancia_3%>
 <%= f.text_area :c3, id: 'text-area-c3', class: "form-control", :cols => 6, :rows => 6, maxlength: 250 %>
<div class="text-right" id="char-countc3"> <a style="color:green;">250/</a> <span id="countc3"></span></div>
<div class="text-center">
 <%= f.range_field :barra3, in: 1..100, class: "form-control-range" %>
 <div class="labels" style="display: flex; justify-content: space-between;">
   <%= label_tag :barra3, 'Nunca', class: 'text-right' %>
   <%= label_tag :barra3, 'Rara vez', class: 'text-right' %>
   <%= label_tag :barra3, '50 y 50', class: 'text-right' %>
   <%= label_tag :barra3, 'A menudo', class: 'text-right' %>
   <%= label_tag :barra3, 'Siempre', class: 'text-right' %>
 </div>
</div>
    </div>
  </div>
</div>
</div>
    </div>
  </div>
<!-- /.row -->
</div>
<!-- /.container -->

    <h2><div class="col text-center" id="selected_name"></div></h2>
    <div class="col-md-6">

<div class="text-center">
<div class="jumbotron">
<H3>Planteamientos</H3>
```

Figura 23. app/views/customers/observational.html.erb (parte 3)

```
<div class="btn-group" role="group" aria-label="Basic example">
  <button type="button" class="btn btn-primary" id="Actividad">Actividad</button>
  <button type="button" class="btn btn-primary" id="Molestias">Molestias</button>
  <button type="button" class="btn btn-primary" id="Deseos">Deseos</button>
  <button type="button" class="btn btn-primary" id="Cultura">Cultura</button>
</div>
</div>
</div>

<div class="form-group" id="Actividad_field" style="display: inherit;">
  <%= f.label :Descripción_Actividad%>
  <%= f.text_area :descripcion_actividad, id: 'text-area-actividad', class: "form-control",:cols => 10, :rows => 10, maxlength: 1500 %>
  <div class="text-right" id="char-count0"> <a style="color:green;">1500/</a> <span id="count0"></span></div>
<div class="text-center">

  <%= f.radio_button :barra1, 'Agudo' %> Agudo
  <%= f.radio_button :barra1, 'Fuerte' %> Fuerte
  <%= f.radio_button :barra1, 'Moderado' %> Moderado
  <%= f.radio_button :barra1, 'Leve' %> Leve
 </div>
</div>
<div class="form-group" id="Molestias_field" style="display: none;">
  <%= f.label :Descripción_Molestias%>
  <%= f.text_area :descripcion_molestias,id: 'text-area-deseos', class: "form-control", :cols => 10, :rows => 10, maxlength: 150 %>
  <div class="text-right" id="char-count1"> <a style="color:green;">150/</a> <span id="count1"></span></div>
<div class="text-center">

  <%= f.radio_button :dd,  'Agudo' %> Agudo
  <%= f.radio_button :dd,  'Fuerte' %> Fuerte
  <%= f.radio_button :dd,  'Moderado' %> Moderado
  <%= f.radio_button :dd,  'Leve' %> Leve
 </div>
</div>
<div class="form-group" id="Deseos_field" style="display: none;">
  <%= f.label :Descripción_Deseos%>
  <%= f.text_area :descripcion_deseos, id: 'text-area-molestias', class: "form-control", :cols => 10, :rows => 10, maxlength: 150 %>
 <div class="text-right" id="char-count"> <a style="color:green;">150/</a> <span id="count"></span></div>
 <div class="text-center">

  <%= f.radio_button :dm, 'Agudo' %> Agudo
  <%= f.radio_button :dm, 'Fuerte' %> Fuerte
  <%= f.radio_button :dm, 'Moderado' %> Moderado
  <%= f.radio_button :dm, 'Leve' %> Leve
 </div>
```

Figura 24. app/views/customers/observational.html.erb (parte 4)



```
</div>

<div class="form-group" id="Cultura_field" style="display: none;">
  <%= f.label :Descripción_Cultura%>
  <%= f.text_area :descripcion_cultura, id: 'text-area-cultura', class: "form-control", :cols => 10, :rows => 10, maxlength: 150 %>
  <div class="text-right" id="char-count2"> <a style="color:green;">150/</a> <span id="count2"></span></div>
<div class="text-center">

  <%= f.radio_button :dc, 'Agudo' %> Agudo
  <%= f.radio_button :dc, 'Fuerte' %> Fuerte
  <%= f.radio_button :dc, 'Moderado' %> Moderado
  <%= f.radio_button :dc, 'Leve' %> Leve
 </div>
</div>
    </div>
  <div class="col-md-6">
  <div class="text-center">
    <div class="jumbotron">
    <H3>Soluciones</H3>
  <div class="btn-group" role="group" aria-label="Basic example">

    <button type="button" class="btn btn-primary" id="Propuestas">Propuestas</button>
    <button type="button" class="btn btn-primary" id="Alivios">Alivios</button>
    <button type="button" class="btn btn-primary" id="Satisfacciones">Satisfacciones</button>
    <button type="button" class="btn btn-primary" id="Su_Cultura">Su Cultura</button>
  </div>
    </div>

</div>

<div class="form-group" id="Propuestas_field" style="display: inherit;">
  <%= f.label :Descripción_Propuestas%>
  <%= f.text_area :descripcion_propuestas, id: 'text-area-propuestas', class: "form-control", :cols => 10, :rows => 10, maxlength: 150 %>
  <div class="text-right" id="char-count3"> <a style="color:green;">150/</a> <span id="count3"></span></div>
  <div class="text-center">
```

Figura 25. app/views/customers/observational.html.erb (parte 5)

```
    <%= f.radio_button :dp, 'Agudo' %> Agudo
    <%= f.radio_button :dp, 'Fuerte' %> Fuerte
    <%= f.radio_button :dp, 'Moderado' %> Moderado
    <%= f.radio_button :dp, 'Leve' %> Leve
   </div>
</div>
<div class="form-group" id="Alivios_field" style="display: none;">
  <%= f.label :Descripción_Alivios%>
  <%= f.text_area :descripcion_alivios, id: 'text-area-alivios', class: "form-control", :cols => 10, :rows => 10, maxlength: 150 %>
  <div class="text-right" id="char-count4"> <a style="color:green;">150/</a> <span id="count4"></span></div>
  <div class="text-center">

    <%= f.radio_button :da, 'Agudo' %> Agudo
    <%= f.radio_button :da, 'Fuerte' %> Fuerte
    <%= f.radio_button :da, 'Moderado' %> Moderado
    <%= f.radio_button :da, 'Leve' %> Leve
   </div>
</div>
<div class="form-group" id="Satisfacciones_field" style="display: none;">
  <%= f.label :Descripción_Satisfacciones%>
  <%= f.text_area :descripcion_satisfacciones, id: 'text-area-satisfacciones', class: "form-control", :cols => 10, :rows => 10, maxlength: 150 %>
  <div class="text-right" id="char-count5"> <a style="color:green;">150/</a> <span id="count5"></span></div>
  <div class="text-center">

    <%= f.radio_button :ds, 'Agudo' %> Agudo
    <%= f.radio_button :ds, 'Fuerte' %> Fuerte
    <%= f.radio_button :ds, 'Moderado' %> Moderado
    <%= f.radio_button :ds, 'Leve' %> Leve
   </div>
</div>
<div class="form-group" id="Su_Cultura_field" style="display: none;">
  <%= f.label :Descripción_Su_Cultura%>
  <%= f.text_area :descripcion_su_cultura, id: 'text-area-sucultura', class: "form-control", :cols => 10, :rows => 10, maxlength: 150 %>
  <div class="text-right" id="char-count6"> <a style="color:green;">150/</a> <span id="count6"></span></div>
  <div class="text-center">

    <%= f.radio_button :dsc, 'Agudo' %> Agudo
    <%= f.radio_button :dsc, 'Fuerte' %> Fuerte
    <%= f.radio_button :dsc, 'Moderado' %> Moderado
    <%= f.radio_button :dsc, 'Leve' %> Leve
   </div>
</div>
</div>

</div>
</div>
  <%= f.submit "Guardar" %>
<% end %>
<%= link_to '« Volver', customers_path, class: 'btn btn-light', data: { confirm: '¿Seguro que quieres volver? Asegurate siempre de guardar' } %>

</div>
```

Figura 26. app/views/customers/observational.html.erb (parte 6)

```
<script>
  //script esencial para gestionar botones por medio de JQuery
  $(document).ready(function() {
    const campos = ['Actividad', 'Deseos', 'Molestias', 'Cultura', 'Propuestas', 'Alivios', 'Satisfacciones', 'Su_Cultura'];

    function ocultarTodosLosCampos() {
      campos.forEach(campo => $('#' + campo + '_field').hide());
    }

    campos.forEach(campo => {
      $('#' + campo).click(function() {
        ocultarTodosLosCampos();
        $('#' + campo + '_field').show();
      });
    });

    ocultarTodosLosCampos();
  });
 </script>


<script>
//SCRIPT de contador de actividad

document.addEventListener("turbolinks:load", function() {
  var textArea = document.getElementById("text-area-actividad");
  var charCount = document.getElementById("count0");
  var maxLength = parseInt(textArea.getAttribute("maxlength"));

  textArea.addEventListener("input", function() {
    var currentLength = textArea.value.length;
    var remaining = maxLength - currentLength;

    charCount.textContent = remaining;

    if (remaining == 0) {
      charCount.style.color = "red"; // Cambia el color a rojo si se excede el límite
    } else {
      charCount.style.color = "green"; // Restablece el color si está dentro del límite
    }
  });

  // Inicializa el contador en la carga de la página
  charCount.textContent = maxLength;
});

  </script>
```


Figura 27. app/views/customers/observational.html.erb (parte 7)

```
  <script>
  //SCRIPT de contador de molestias

  document.addEventListener("turbolinks:load", function() {
    var textArea = document.getElementById("text-area-molestias");
    var charCount = document.getElementById("count");
    var maxLength = parseInt(textArea.getAttribute("maxlength"));

    textArea.addEventListener("input", function() {
      var currentLength = textArea.value.length;
      var remaining = maxLength - currentLength;

      charCount.textContent = remaining;

      if (remaining == 0) {
        charCount.style.color = "red"; // Cambia el color a rojo si se excede el límite
      } else {
        charCount.style.color = "green"; // Restablece el color si está dentro del límite
      }
    });

    // Inicializa el contador en la carga de la página
    charCount.textContent = maxLength;
  });
  </script>




  <!--deseos script-->

  <script>
  //SCRIPT de contador de deseos 150

  document.addEventListener("turbolinks:load", function() {
    var textArea = document.getElementById("text-area-deseos");
    var charCount = document.getElementById("count1");
    var maxLength = parseInt(textArea.getAttribute("maxlength"));

    textArea.addEventListener("input", function() {
      var currentLength = textArea.value.length;
      var remaining = maxLength - currentLength;

      charCount.textContent = remaining;

      if (remaining == 0) {
        charCount.style.color = "red"; // Cambia el color a rojo si se excede el límite
      } else {
        charCount.style.color = "green"; // Restablece el color si está dentro del límite
      }
    });

    // Inicializa el contador en la carga de la página
    charCount.textContent = maxLength;
  });

    </script>
```

Figura 28. app/views/customers/observational.html.erb (parte 8)

```
<script>
//SCRIPT de contador de cultura 150

document.addEventListener("turbolinks:load", function() {
  var textArea = document.getElementById("text-area-cultura");
  var charCount = document.getElementById("count2");
  var maxLength = parseInt(textArea.getAttribute("maxlength"));

  textArea.addEventListener("input", function() {
    var currentLength = textArea.value.length;
    var remaining = maxLength - currentLength;

    charCount.textContent = remaining;

    if (remaining == 0) {
      charCount.style.color = "red"; // Cambia el color a rojo si se excede el límite
    } else {
      charCount.style.color = "green"; // Restablece el color si está dentro del límite
    }
  });

  // Inicializa el contador en la carga de la página
  charCount.textContent = maxLength;
});

  </script>




  <script>
    //SCRIPT de contador de propuestas 150

    document.addEventListener("turbolinks:load", function() {
      var textArea = document.getElementById("text-area-propuestas");
      var charCount = document.getElementById("count3");
      var maxLength = parseInt(textArea.getAttribute("maxlength"));

      textArea.addEventListener("input", function() {
        var currentLength = textArea.value.length;
        var remaining = maxLength - currentLength;

        charCount.textContent = remaining;

        if (remaining == 0) {
          charCount.style.color = "red"; // Cambia el color a rojo si se excede el límite
        } else {
          charCount.style.color = "green"; // Restablece el color si está dentro del límite
        }
      });

      // Inicializa el contador en la carga de la página
      charCount.textContent = maxLength;
    });

      </script>
```


Figura 29. app/views/customers/observational.html.erb (parte 9)

```
                <script>
                  //SCRIPT de contador de su cultura 150

                  document.addEventListener("turbolinks:load", function() {
                    var textArea = document.getElementById("text-area-sucultura");
                    var charCount = document.getElementById("count6");
                    var maxLength = parseInt(textArea.getAttribute("maxlength"));

                    textArea.addEventListener("input", function() {
                      var currentLength = textArea.value.length;
                      var remaining = maxLength - currentLength;

                      charCount.textContent = remaining;

                      if (remaining == 0) {
                        charCount.style.color = "red"; // Cambia el color a rojo si se excede el límite
                      } else {
                        charCount.style.color = "green"; // Restablece el color si está dentro del límite
                      }
                    });

                    // Inicializa el contador en la carga de la página
                    charCount.textContent = maxLength;
                  });

                    </script>

<script>
//SCRIPT de contador de circunstancia 1

document.addEventListener("turbolinks:load", function() {
  var textArea = document.getElementById("text-area-c1");
  var charCount = document.getElementById("countc1");
  var maxLength = parseInt(textArea.getAttribute("maxlength"));

  textArea.addEventListener("input", function() {
    var currentLength = textArea.value.length;
    var remaining = maxLength - currentLength;

    charCount.textContent = remaining;

    if (remaining == 0) {
      charCount.style.color = "red"; // Cambia el color a rojo si se excede el límite
    } else {
      charCount.style.color = "green"; // Restablece el color si está dentro del límite
    }
  });

  // Inicializa el contador en la carga de la página
  charCount.textContent = maxLength;
});

  </script>
```

Figura 30. app/views/customers/observational.html.erb (parte 10)

```
<script>
  //SCRIPT de contador de alivios 150

  document.addEventListener("turbolinks:load", function() {
    var textArea = document.getElementById("text-area-alivios");
    var charCount = document.getElementById("count4");
    var maxLength = parseInt(textArea.getAttribute("maxlength"));

    textArea.addEventListener("input", function() {
      var currentLength = textArea.value.length;
      var remaining = maxLength - currentLength;

      charCount.textContent = remaining;

      if (remaining == 0) {
        charCount.style.color = "red"; // Cambia el color a rojo si se excede el límite
      } else {
        charCount.style.color = "green"; // Restablece el color si está dentro del límite
      }
    });

    // Inicializa el contador en la carga de la página
    charCount.textContent = maxLength;
  });

  </script>

  <script>
    //SCRIPT de contador de satisfacciones 150

    document.addEventListener("turbolinks:load", function() {
      var textArea = document.getElementById("text-area-satisfacciones");
      var charCount = document.getElementById("count5");
      var maxLength = parseInt(textArea.getAttribute("maxlength"));

      textArea.addEventListener("input", function() {
        var currentLength = textArea.value.length;
        var remaining = maxLength - currentLength;

        charCount.textContent = remaining;

        if (remaining == 0) {
          charCount.style.color = "red"; // Cambia el color a rojo si se excede el límite
        } else {
          charCount.style.color = "green"; // Restablece el color si está dentro del límite
        }
      });

      // Inicializa el contador en la carga de la página
      charCount.textContent = maxLength;
    });

    </script>
```


Figura 31. app/views/customers/observational.html.erb (parte 11)

```
  <script>
//SCRIPT de contador de circunstancia 1

document.addEventListener("turbolinks:load", function() {
  var textArea = document.getElementById("text-area-c2");
  var charCount = document.getElementById("countc2");
  var maxLength = parseInt(textArea.getAttribute("maxlength"));

  textArea.addEventListener("input", function() {
    var currentLength = textArea.value.length;
    var remaining = maxLength - currentLength;

    charCount.textContent = remaining;

    if (remaining == 0) {
      charCount.style.color = "red"; // Cambia el color a rojo si se excede el límite
    } else {
      charCount.style.color = "green"; // Restablece el color si está dentro del límite
    }
  });

  // Inicializa el contador en la carga de la página
  charCount.textContent = maxLength;
});

  </script>

<script>
//SCRIPT de contador de circunstancia 1

document.addEventListener("turbolinks:load", function() {
  var textArea = document.getElementById("text-area-c3");
  var charCount = document.getElementById("countc3");
  var maxLength = parseInt(textArea.getAttribute("maxlength"));

  textArea.addEventListener("input", function() {
    var currentLength = textArea.value.length;
    var remaining = maxLength - currentLength;

    charCount.textContent = remaining;

    if (remaining == 0) {
      charCount.style.color = "red"; // Cambia el color a rojo si se excede el límite
    } else {
      charCount.style.color = "green"; // Restablece el color si está dentro del límite
    }
  });

  // Inicializa el contador en la carga de la página
  charCount.textContent = maxLength;
});

  </script>
```

Las siguientes definiciones pertenecen a las figuras (del 21 al 31)

Definicion de observational.html.erb:

- **<%= render partial: 'shared/navigation_bar', object: current_user %>: E**sto renderiza una barra de navegación que es compartida entre varias vistas. El objeto current_user se pasa a la vista parcial como una variable local.
- **<% provide (:title, t('customers_section')) %>: E**stablece el título de la página a la traducción del string 'customers_section'.
- **<%= render partial: 'shared/igheader', locals: {...} %>:** Renderiza otra vista parcial llamada ‘igheader’. Se pasan varias variables locales a esta vista.
- **<%= render 'train'%>:** Renderiza una vista parcial llamada ‘train’.
- ** ... :** Es el contenedor principal que envuelve el contenido de la página.
- **<%= form_for @customer do |f| %>:** Esto comienza un formulario para el objeto @customer. Dentro de este bloque, se definen los campos del formulario. Se usa @customer ya que la estación observacional se construyó en la base de la misma.
- **<%= f.text_field :nombre, :readonly => true %>:** Crea un campo de texto para el atributo nombre del objeto @customer. Este campo es de solo lectura.
- **<%= f.text_area :c1, id: 'text-area-c1', class: "form-control", :cols => 6, :rows => 6, maxlength: 250 %>:** Crea un área de texto para el atributo c1 del objeto @customer. Tiene un límite de longitud de 250 caracteres.
- **<%= f.range_field :siempre_rara_vez, in: 1..100, class: "form-control-range" %>:** Genera un campo de rango para el atributo siempre_rara_vez del objeto @customer. El rango va de 1 a 100.
- **<%= f.radio_button :barra1, 'Agudo' %> Agudo:** Esto crea un botón de radio para el atributo barra1 del objeto @customer. El valor del botón de radio es ‘Agudo’.
- ** :** Este es un grupo de formulario que inicialmente está oculto (display: none). Contiene un campo de texto y botones de radio relacionados con la “Cultura”.

- **<%= f.text_area :descripcion_cultura, id: 'text-area-cultura', class: "form-control", :cols => 10, :rows => 10, maxlength: 150 %>:** Esto crea un área de texto para el atributo descripcion_cultura del objeto @customer. Tiene un límite de longitud de 150 caracteres.
- E**<%= f.radio_button :dc, 'Agudo' %> Agudo:** Esto crea un botón de radio para el atributo dc del objeto @customer. El valor del botón de radio es 'Agudo'. Hay botones de radio similares para 'Fuerte', 'Moderado' y 'Leve'.
- ** :** Este es un contenedor que ocupa 6 columnas en una fila de Bootstrap. Contiene grupos de formulario para "Propuestas", "Alivios", "Satisfacciones" y "Su Cultura". Cada uno de estos grupos de formulario es similar al grupo de formulario "Cultura" que expliqué anteriormente.
- **<%= f.submit "Guardar" %>:** Esto crea un botón de envío para el formulario. Cuando se hace clic en este botón, se envía el formulario.
- **<%= link_to '« Volver', customers_path, class: 'btn btn-light', data: { confirm: '¿Seguro que quieres volver? Asegurate siempre de guardar' } %>**: Crea un enlace para volver a la ruta customers_path. Cuando se hace clic en este enlace, se muestra un mensaje de confirmación para asegurarse de que el usuario ha guardado sus cambios.

Definición del primer script de observational.html.erb:

- **$(document).ready(function() { ... })**; Asegura que el código se ejecute una vez que el DOM (modelo de objetos del documento) esté completamente cargado. Esto garantiza que los elementos HTML estén disponibles para manipulación antes de que se ejecute el script.
- **const campos = ['Actividad', 'Deseos', 'Molestias', 'Cultura', 'Propuestas', 'Alivios', 'Satisfacciones', 'Su_Cultura']**; Define un arreglo llamado campos que contiene los nombres de los campos.
- **function ocultarTodosLosCampos() { ... }**: Es una función que oculta todos los campos. Utiliza el método .hide() de jQuery para ocultar los elementos con los IDs correspondientes a los nombres de los campos.
- **campos.forEach(campo => { ... })**; itera sobre cada campo en el arreglo.
- **(.click(function() { ... }))** : Para cada campo, se agrega un evento de clic que ejecuta el siguiente código: llama a **ocultarTodosLosCampos()** para ocultar todos los campos,

muestra el campo específico (por ejemplo, Actividad) utilizando el método .show(). **ocultarTodosLosCampos()**; se llama al principio para asegurarse de que todos los campos estén ocultos inicialmente.

Definición del segundo script de observational.html.erb:

El script se ejecuta cuando la página se carga o cuando se navega a través de Turbolinks (un sistema de navegación AJAX utilizado en algunas aplicaciones web).

- **var textArea = document.getElementById("text-area-actividad")**; Busca el elemento del DOM con el ID "text-area-actividad". Esto probablemente se refiere a un área de texto donde los usuarios pueden escribir su actividad.
- **var charCount = document.getElementById("count0")**; Busca el elemento del DOM con el ID "count0". Esto probablemente es donde se mostrará el contador de caracteres.
- **var maxLength = parseInt(textArea.getAttribute("maxlength"))**; Obtiene la longitud máxima permitida para el área de texto (definida por el atributo maxlength).
- **textArea.addEventListener("input", function() { ... })**; Agrega un evento de escucha para cuando el usuario escriba en el área de texto. Dentro de esta función, se calcula la longitud actual del texto en el área de texto (currentLength) y la cantidad de caracteres restantes (remaining). Se actualiza el contenido del elemento con ID "count0" para mostrar la cantidad de caracteres restantes. Si no quedan caracteres disponibles (es decir, remaining es igual a 0), el color del contador se cambia a rojo. De lo contrario, se restablece a verde.
- **Inicialización del Contador**: **charCount.textContent = maxLength**; Establece el valor inicial del contador en la longitud máxima permitida.

Definición del tercer script de observational.html.erb:

Exactamente que en el script anterior, este evento asegura que el código se ejecute cuando la página se carga o cuando se navega a través de Turbolinks.

- **var textArea = document.getElementById("text-area-molestias")**; Busca el elemento del DOM con el ID "text-area-molestias". Esto se refiere a un área de texto donde los usuarios pueden describir sus molestias.

- **var charCount = document.getElementById("count")**; Busca el elemento del DOM con el ID “count”. Aquí es donde se mostrará el contador de caracteres.
- **var maxLength = parseInt(textArea.getAttribute("maxlength"))**; Obtiene la longitud máxima permitida para el área de texto (definida por el atributo maxlength).
- **textArea.addEventListener("input", function() { ... })**; Agrega un evento de escucha para cuando el usuario escriba en el área de texto.Dentro de esta función, se calcula la longitud actual del texto en el área de texto (currentLength) y la cantidad de caracteres restantes (remaining). Se actualiza el contenido del elemento con ID “count” para mostrar la cantidad de caracteres restantes. Si no quedan caracteres disponibles (es decir, remaining es igual a 0), el color del contador se cambia a rojo. De lo contrario, se restablece a verde.
- **charCount.textContent = maxLength**; Establece el valor inicial del contador en la longitud máxima permitida.

Definición del cuarto script de observational.html.erb:

Al igual que en los scripts anteriores, dicho evento asegura que el código se ejecute cuando la página se carga o cuando se navega a través de Turbolinks.

- **var textArea = document.getElementById("text-area-deseos")**; busca el elemento del DOM con el ID “text-area-deseos”. Se refiere a un área de texto donde los usuarios pueden escribir sus deseos.
- **var charCount = document.getElementById("count1")**; busca el elemento del DOM con el ID “count1”. Aquí es donde se mostrará el contador de caracteres.
- **var maxLength = parseInt(textArea.getAttribute("maxlength"))**; obtiene la longitud máxima permitida para el área de texto (definida por el atributo maxlength).
- **textArea.addEventListener("input", function() { ... })**; agrega un evento de escucha para cuando el usuario escriba en el área de texto. Dentro de esta función, se calcula la longitud actual del texto en el área de texto (currentLength) y la cantidad de caracteres restantes (remaining). Se actualiza el contenido del elemento con ID “count1” para mostrar la cantidad de caracteres restantes. Si no quedan caracteres disponibles (es decir, remaining es igual a 0), el color del contador se cambia a rojo. De lo contrario, se restablece a verde.

- **Inicialización del Contador: charCount.textContent = maxLength**; establece el valor inicial del contador en la longitud máxima permitida

Definición del quinto script de observational.html.erb:

Al igual que en los scripts previos, este evento asegura que el código se ejecute cuando la página se carga o cuando se navega a través de Turbolinks.

- **var textArea = document.getElementById("text-area-cultura")**; busca el elemento del DOM con el ID "text-area-cultura". Se refiere a un área de texto donde los usuarios pueden escribir sobre su cultura.
- **var charCount = document.getElementById("count2")**; busca el elemento del DOM con el ID "count2". Aquí es donde se mostrará el contador de caracteres.
- **var maxLength = parseInt(textArea.getAttribute("maxlength"))**; obtiene la longitud máxima permitida para el área de texto (definida por el atributo maxlength).
- **textArea.addEventListener("input", function() { ... })**; agrega un evento de escucha para cuando el usuario escriba en el área de texto. Dentro de esta función, se calcula la longitud actual del texto en el área de texto (currentLength) y la cantidad de caracteres restantes (remaining). Se actualiza el contenido del elemento con ID "count2" para mostrar la cantidad de caracteres restantes. Si no quedan caracteres disponibles (es decir, remaining es igual a 0), el color del contador se cambia a rojo. De lo contrario, se restablece a verde.
- **charCount.textContent = maxLength**; establece el valor inicial del contador en la longitud máxima permitida.

Definición del sexto script de observational.html.erb:

Como en los scripts anteriores, este evento asegura que el código se ejecute cuando la página se carga o cuando se navega a través de Turbolinks.

- **var textArea = document.getElementById("text-area-propuestas")**; busca el elemento del DOM con el ID "text-area-propuestas". Se refiere a un área de texto donde los usuarios pueden escribir sus propuestas.

- **var charCount = document.getElementById("count3")**; busca el elemento del DOM con el ID “count3”. Aquí es donde se mostrará el contador de caracteres.
- **var maxLength = parseInt(textArea.getAttribute("maxlength"))**; obtiene la longitud máxima permitida para el área de texto (definida por el atributo maxlength).
- **textArea.addEventListener("input", function() { ... })**; agrega un evento de escucha para cuando el usuario escriba en el área de texto. Dentro de esta función, se calcula la longitud actual del texto en el área de texto (currentLength) y la cantidad de caracteres restantes (remaining). Se actualiza el contenido del elemento con ID “count3” para mostrar la cantidad de caracteres restantes. Si no quedan caracteres disponibles (es decir, remaining es igual a 0), el color del contador se cambia a rojo. De lo contrario, se restablece a verde.
- **charCount.textContent = maxLength**; establece el valor inicial del contador en la longitud máxima permitida.

Definición del septimo script de observational.html.erb:

Este evento asegura que el código se ejecute cuando la página se carga o cuando se navega a través de Turbolinks.

- **var textArea = document.getElementById("text-area-alivios")**; busca el elemento del DOM con el ID “text-area-alivios”. Esto se refiere a un área de texto donde los usuarios pueden escribir sobre sus alivios.
- **var charCount = document.getElementById("count4")**; busca el elemento del DOM con el ID “count4”. Aquí es donde se mostrará el contador de caracteres.
- **var maxLength = parseInt(textArea.getAttribute("maxlength"))**; obtiene la longitud máxima permitida para el área de texto (definida por el atributo maxlength).
- **textArea.addEventListener("input", function() { ... })**; agrega un evento de escucha para cuando el usuario escriba en el área de texto. Dentro de esta función, se calcula la longitud actual del texto en el área de texto (currentLength) y la cantidad de caracteres restantes (remaining). Se actualiza el contenido del elemento con ID “count4” para mostrar la cantidad de caracteres restantes. Si no quedan caracteres disponibles (es decir, remaining es igual a 0), el color del contador se cambia a rojo. De lo contrario, se restablece a verde.

- **charCount.textContent = maxLength**; establece el valor inicial del contador en la longitud máxima permitida.

Definición del octavo script de observational.html.erb:

Este evento asegura que el código se ejecute cuando la página se carga o cuando se navega a través de Turbolinks.

- **var textArea = document.getElementById("text-area-satisfacciones")**; busca el elemento del DOM con el ID "text-area-satisfacciones". Se refiere a un área de texto donde los usuarios pueden escribir sobre sus satisfacciones.
- **var charCount = document.getElementById("count5")**; busca el elemento del DOM con el ID "count5". Aquí es donde se mostrará el contador de caracteres.
- **var maxLength = parseInt(textArea.getAttribute("maxlength"))**; obtiene la longitud máxima permitida para el área de texto (definida por el atributo maxlength).
- **textArea.addEventListener("input", function() { ... })**; agrega un evento de escucha para cuando el usuario escriba en el área de texto. Dentro de esta función, se calcula la longitud actual del texto en el área de texto (currentLength) y la cantidad de caracteres restantes (remaining). Se actualiza el contenido del elemento con ID "count5" para mostrar la cantidad de caracteres restantes. Si no quedan caracteres disponibles (es decir, remaining es igual a 0), el color del contador se cambia a rojo. De lo contrario, se restablece a verde.
- **charCount.textContent = maxLength**; establece el valor inicial del contador en la longitud máxima permitida.

Definición del noveno script de observational.html.erb:

Como en scripts anteriores, este evento asegura que el código se ejecute cuando la página se carga o cuando se navega a través de Turbolinks.

- **var textArea = document.getElementById("text-area-sucultura")**; busca el elemento del DOM con el ID "text-area-sucultura". Esto afecta a un área de texto donde los usuarios pueden escribir sobre su cultura.

- **var charCount = document.getElementById("count6")**; busca el elemento del DOM con el ID “count6”. Aquí es donde se mostrará el contador de caracteres.
- **var maxLength = parseInt(textArea.getAttribute("maxlength"))**; obtiene la longitud máxima permitida para el área de texto (definida por el atributo maxlength).
- **textArea.addEventListener("input", function() { ... })**; agrega un evento de escucha para cuando el usuario escriba en el área de texto. Dentro de esta función, se calcula la longitud actual del texto en el área de texto (currentLength) y la cantidad de caracteres restantes (remaining). Se actualiza el contenido del elemento con ID “count6” para mostrar la cantidad de caracteres restantes. Si no quedan caracteres disponibles (es decir, remaining es igual a 0), el color del contador se cambia a rojo. De lo contrario, se restablece a verde.
- **charCount.textContent = maxLength**; establece el valor inicial del contador en la longitud máxima permitida.

Definición del décimo script(dos scripts) de observational.html.erb:

Este script se ejecuta cuando la página se carga o cuando se navega a través de Turbolinks (evento turbolinks:load).

- Obtiene el elemento del DOM con el ID “text-area-c2”, que probablemente se refiere a un área de texto relacionada con “circunstancia 1”.También obtiene el elemento con el ID “countc2”, donde se mostrará el contador de caracteres.
- Calcula la longitud actual del texto en el área de texto y la cantidad de caracteres restantes y actualiza el contenido del contador para mostrar la cantidad de caracteres restantes.

Si no quedan caracteres disponibles (es decir, se ha alcanzado el límite), cambia el color del contador a rojo; de lo contrario, lo restablece a verde.

Script de Contador de Circunstancia 2:

- Este script es idéntico al primero, pero se aplica al área de texto con el ID “text-area-c3” y muestra el contador en el elemento con el ID “countc3”.

5.1.3. Controlador de la primera estación: Observacional

El controlador de la estación observacional se encuentra ubicado en **app/controllers/customers_controller.rb**, tal como se había comentado anteriormente, la estación observacional es controlada a través de **customers_controller.rb**

Base de datos (db migrate) del modelo customer.rb

Esta base de datos se generó al momento de crear el modelo de la primera estación customer.rb, cada uno con los atributos necesarios y requeridos por el documento de requisitos del producto.

La tabla customers contiene múltiples atributos necesarios para el despliegue de la estación.

Figura 32. db/migrate/20230916161155_create_customers.rb

```
db > migrate > 20230916161155_create_customers.rb
class CreateCustomers < ActiveRecord::Migration[5.1]
  def change
    create_table :customers do |t|

      t.integer :siempre_rara_vez
      t.text :descripcion_actividad
      t.string :barra1
      t.text :descripcion_deseos
      t.text :descripcion_molestias
      t.text :descripcion_cultura
      t.text :descripcion_propuestas
      t.text :descripcion_alivios
      t.text :descripcion_satisfacciones
      t.text :descripcion_su_cultura

      t.timestamps
    end
  end
end
```


Figura 33. db/migrate/20230916234849_add_nombre_to_customers.rb

```
db > migrate > 20230916234849_add_nombre_to_customers.rb
class AddNombreToCustomers < ActiveRecord::Migration[5.1]
  def change
    add_column :customers, :nombre, :string
  end
end
```

Figura 34. db/migrate/20230924193246_add_dc_dm_dd_to_customers.rb

db > migrate > 20230924193246_add_dc_dm_dd_to_customers.rb

```ruby
class AddDcDmDdToCustomers < ActiveRecord::Migration[5.1]
  def change
    add_column :customers, :dc, :string
    add_column :customers, :dm, :string
    add_column :customers, :dd, :string
  end
end
```

Figura 35. db/migrate/20231001195601_add_c1_c2_c3_barra1_barra2_to_customers.rb

db > migrate > 20231001195601_add_c1_c2_c3_barra1_barra2_to_customers.rb

```ruby
class AddC1C2C3Barra1Barra2ToCustomers < ActiveRecord::Migration[5.1]
  def change
    add_column :customers, :c1, :text
    add_column :customers, :c2, :text
    add_column :customers, :c3, :text
    add_column :customers, :barra2, :integer
    add_column :customers, :barra3, :integer
  end
end

```

Figura 36. db/migrate/20240203171008_add_dp_to_customers.rb

db > migrate > 20240203171008_add_dp_to_customers.rb

```ruby
class AddDpToCustomers < ActiveRecord::Migration[5.1]
  def change
    add_column :customers, :dp, :string
  end
end

```

Figura 37. db/migrate/20240217014348_add_da_ds_dsc_to_customers.rb

db > migrate > 20240217014348_add_da_ds_dsc_to_customers.rb

```ruby
class AddDaDsDscToCustomers < ActiveRecord::Migration[5.1]
  def change
    add_column :customers, :da, :string
    add_column :customers, :ds, :string
    add_column :customers, :dsc, :string
  end
end
```

En Ruby on Rails no se puede agregar directamente nuevos atributos a la tabla, para ello debemos utilizar la consola de Rails, escribir el comando correspondiente y hacer rails **db: migrate.**

Tabla resumen: Estación Observacional

Tabla 3. Tabla resumen: Estación Observacional

	Ubicación
Modelo	app/models/customer.rb
Vista	app/views/customers/_new_edit.html.erb, app/views/customers/_form.html.erb, app/views/customers/observational.html.erb
Controlador	app/controllers/customers_controller.rb

Análisis: el conjunto de códigos utilizados tanto en el MVC de la estación permitió que se alcanzara el objetivo propuesto, dando los espacios necesarios (gracias a customer) para el despliegue de los formularios, permitiendo poder guardar los datos de cada campo en la respectiva base de datos de customer. Cada uno de sus bloques está en funcionamiento, incluyendo los respectivos calibradores, las validaciones reflejadas en el documento de requisito de producto también están presentes y funcionales.

5.2. Segunda estación: Enfoque de grupo

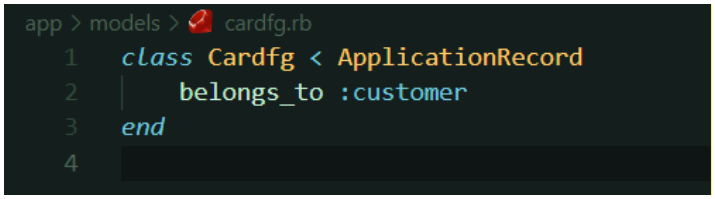
El propósito esta estación fue encontrar las molestias, situaciones y otras circunstancias a través del análisis de resultados y conclusiones de los experimentos realizados, las cuales pueden ser contrastadas con las hipótesis planteadas en la primera estación, el contraste fue clave para definir y listar los siguientes objetivos: Encontrar las fortalezas y debilidades del producto.

Se implementó una tarjeta de enfoque de grupo, fundamental para esta estación.

5.2.1. Modelo de la segunda estación: Enfoque de grupo

El modelo de esta estación se encuentra en **app/models/cardfg.rb**, cuyo archivo tiene el nombre de **cardfg.rb**.

Figura 38. app/models/cardfg.rb



Definición del modelo Cardfg:

- **class Cardfg < ApplicationRecord**: esto define una clase llamada Cardfg que hereda de ApplicationRecord. En Ruby on Rails, ApplicationRecord es la clase base para todos los modelos de la base de datos. Por lo tanto, Cardfg es un modelo asociado de base de datos.
- **belongs_to :customer: e**stablece una relación entre el modelo Cardfg y el modelo Customer. La relación es de "pertenencia" (belongs_to), lo que significa que un registro de Cardfg está asociado con un registro de Customer. En términos de base de datos, esto generalmente se traduce en una clave foránea en la tabla Cardfg que apunta a la tabla Customer.

Figura 39. db/migrate/20231004043447_add_customer_ref_to_cardfgs.rb

Esto se logró gracias al siguiente comando en la Rails Console: **rails generate migration AddCustomerRefToCardfgs customer:references** y luego en la Rails Console**: rails db:migrate**.

Esta migración cumplió con la función de agregar una nueva columna llamada customer_id a la tabla cardfgs (que hace referencia a la columna id en la tabla customers). Además, también

agregó un índice a la nueva columna. Todas las estaciones excepto observacional poseen esta relación.

5.2.2. Vistas de la segunda estación: Enfoque de grupo

Definicion de focusg.html.erb:

Esta vista se encuentra ubicada en **app/views/customers/focusg.html.erb** (anexo B.5), su archivo tiene el nombre de **firstcg.html.erb** y es la vista principal de Enfoque de grupo.

Figura 40. app/views/customers/focusg.html.erb (parte 1)

```
<%= render partial: 'shared/navigation_bar', object: current_user %>
<% provide(:title, t('customers_section')) %>
<%= render partial: 'shared/igheader', locals: {
    title: t("build"),
    secondaryTitle:  t("customers_section"),
    hideVpSelector: false,
    hideTooltipsButton: true
} %>

<style>


  .milist li {
    transition: transform .2s;

  }
  .milist li:hover {
    transform: scale(1.2);
  }
  </style>


<%= render 'train' %>

<div class="container">
<div class="panel panel-default border-top-6">
<div class="panel-body">
<h1 class="text-center"> Editar Estación Enfoque de Grupo: <%= @customer.nombre %> </h1>
<div class="container">
<div class="col text-center">
  <button type="button" class="btn btn-light">
    <% if @cardfg.present? %>
      <% path = edit_customer_cardfg_path(@customer, @cardfg) %>
    <% else %>
      <% path = new_customer_cardfg_path(@customer) %>
    <% end %>
    <%= link_to '+ Agregar Tarjeta enfoque de Grupo', path, class: 'btn btn-light' %>
  </button>
</div>

<ul>

<div class="col text-center">
<div class="table-responsive" style="width: 100%; margin: auto;">
<table class="table table-bordered ">
    <thead>
    <tr>
        <th><h3 class=" text-center"> Tarjeta enfoque de Grupo</h3></th>

    </tr>
    </thead>
    <tbody>
     <tr>
        <td> <% @customer.cardfgs.each do |cardfg| %>
        <div class="milist">
    <li class="p-3" style="list-style-type: none; width: 50%; margin: auto;">
     <%= link_to cardfg.titulo_tarjeta, edit_customer_cardfg_path(@customer, cardfg), data: { confirm: '¿Vas a editar?' }, title: 'Haz clic aquí para editar' %>
     <%= "|" %>
    <%= link_to 'Eliminar', customer_cardfg_path(@customer, cardfg), method: :delete, data: { confirm: '¿Estás seguro?' } %>
     <%= "|" %>

    </li>
    </div>
  <% end %></td>
```

Figura 41. app/views/customers/focusg.html.erb (parte 2)

```
    </tr>

    </tbody>
</table>
</div>
</div>


</ul>


<h4 class=" text-center">  Selecciona la Tarjeta enfoque de Grupo del SdeC: <%= @customer.nombre %> </h4>


<% if @customer.cardfgs.any? %>
<div style="list-style-type: none; width: 50%; margin: auto;">
  <%= select_tag "cardfgs", options_from_collection_for_select(@customer.cardfgs, "id", "titulo_tarjeta"), id: "cardfg_select", prompt: "Selecciona la Tarjeta enfoque de Grupo" %>

  <%= link_to '« Volver', customers_path, class: 'btn btn-light', data: { confirm: '¿Seguro que quieres volver? Asegurate siempre de guardar' } %>

  <%= link_to 'Ver detalles', '#', id: 'details_link', class: 'btn btn-primary' %>
  <%= link_to 'Editar', '#', id: 'edit_link', class: 'btn btn-primary' %>
</div>
<% end %>
</div>
</div>
</div>
</div>
```

Figura 42. app/views/customers/focusg.html.erb (parte 3)


```
<script>
  $('#cardfg_select').change(function() {
    var cardfg_id = $(this).val();

    // Actualizar el enlace de 'Ver detalles'
    var urlBase = '<%= customer_cardfg_path(@customer, 'ID') %>';
    var detailsLink = urlBase.replace('ID', cardfg_id);
    $('#details_link').attr('href', detailsLink);

    // Actualizar el enlace de 'Editar'
    var editLink = "/customers/" + <%= @customer.id %> + "/cardfgs/" + cardfg_id + "/edit";
    $('#edit_link').attr('href', editLink);
  });
</script>
<script>
//script sombrero seleccionador magico
$(document).ready(function() {
  $('#cardfg_select').change(function() {
    var cardfg_id = $(this).val();

    // Guardar el id del cliente en la sesión
    $.ajax({
      type: 'POST',
      url: '/set_cardfg_id',  // ruta que coincide con el post en routes.rb
      data: { cardfg_id: cardfg_id }
    });

    //
    //var link_observational = "/customers/" + customer_id + "/observational/edit";
    //$('#customer_link').attr('href', link_observational);
    var link_focusg = "/customers/" + customer_id + "/focusg/edit" + cardfg_id;
    $('#customer_link1').attr('href', link_focusg);
  });
});
</script>
<script>
//si se selecciona Selecciona un SdeC y decide ir a una estacion
$('#customer_select').change(function() {
  if ($(this).val() == "") { // Asume que el valor del prompt es una cadena vacía
    window.location = 'redirect_to customers_path'; // Reemplaza esto con la ruta correcta a customers_path
  }
});
</script>
  <script>
// Agregar el evento click
document.addEventListener('click', myFunction);

// Eliminar el evento click antes de cambiar de vista
window.addEventListener('beforeunload', function() {
  document.removeEventListener('click', myFunction);
});

// Volver a agregar el evento click cuando regreses a la vista original
window.addEventListener('load', function() {
  document.addEventListener('click', myFunction);
});
</script>
```

Las figuras 40, 41 y 42 muestran el conjunto de codigos que conmponen el modelo **focusg.html.erb**.

- **.milist:** Cuando el usuario pasa el cursor sobre uno de estos elementos, se aplica una escala de 1.2 para crear un efecto de zoom.
- **<%= render 'train' %>:** Pertenece al parcial de ‘train’, el panel de estaciones.
- **<button type="button" class="btn btn-light"> .. </button>:** Verifica si @cardfg está presente (es decir, si existe). Si es así, se asigna una ruta para editar la tarjeta enfoque de grupo (edit_customer_cardfg_path(@customer, @cardfg)). De lo contrario, se asigna una ruta para crear una nueva tarjeta enfoque de grupo (new_customer_cardfg_path(@customer)). Finalmente, se muestra un enlace dentro del botón con el texto “+ Agregar Tarjeta enfoque de Grupo” y la ruta correspondiente.
- ** … :** Se crea una lista no ordenada (<ul>).Dentro de un contenedor con clase CSS "col text-center", se coloca una tabla responsiva. La tabla tiene un encabezado “Tarjeta enfoque de Grupo” y una sola fila de datos. En la celda de la tabla (<td>), se itera sobre la colección @customer.cardfgs.
- **Para cada tarjeta enfoque de grupo (cardfg):** Se crea un elemento de lista (<li>) con estilo personalizado (ancho del 50% y centrado). Se muestra un enlace con el título de la tarjeta (cardfg.titulo_tarjeta). Al hacer clic en este enlace, se redirige a la página de edición de la tarjeta enfoque de grupo (edit_customer_cardfg_path(@customer, cardfg)).Se usa un separador "|" para mejorar la legibilidad. De igual modo, se muestra un enlace para eliminar la tarjeta enfoque de grupo. Al hacer clic en este enlace, se solicita confirmación antes de eliminar la tarjeta (customer_cardfg_path(@customer, cardfg)).

Definición del primer script de focusg.html.erb:

- Este script escucha los cambios en el menú desplegable #cardfg_select. Cuando se selecciona una nueva opción, obtiene el cardfg_id seleccionado. Luego, construye y actualiza las URL para los enlaces “Ver detalles” y “Editar” según el cardfg_id seleccionado.

Definición del segundo script de focusg.html.erb:

- Este script también escucha los cambios en el menú desplegable #cardfg_select. Cuando se selecciona una nueva opción, obtiene el cardfg_id seleccionado, luego, guarda el ID del cliente en la sesión mediante una solicitud AJAX. Finalmente, construye y actualiza el enlace para la página de edición del "Grupo de Enfoque" según el cardfg_id seleccionado.

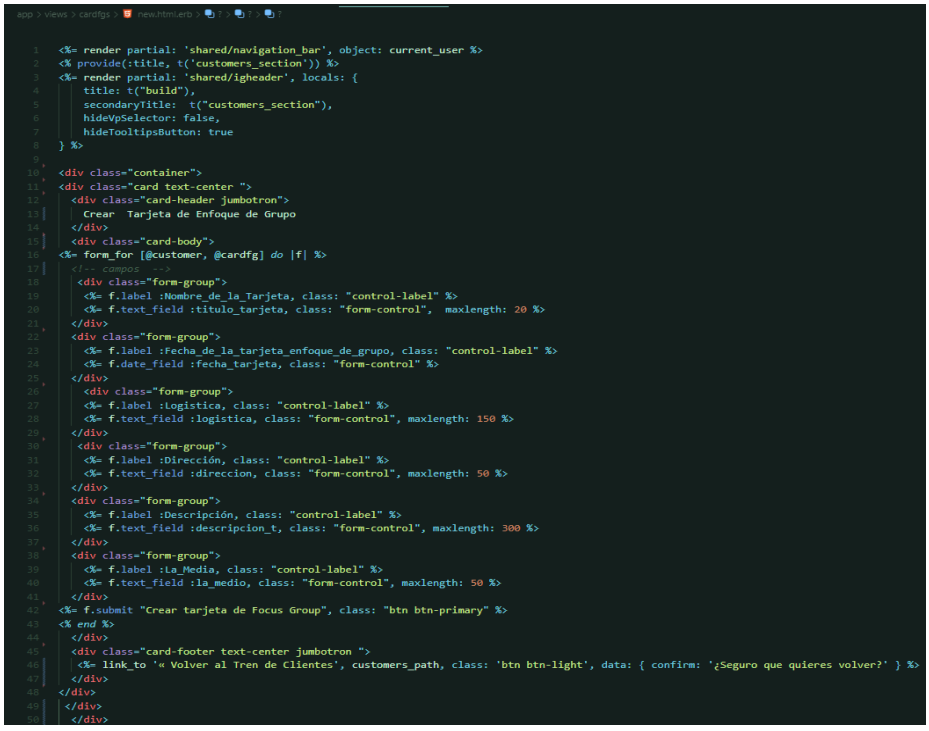
Definición del tercer script de focusg.html.erb:

- Este script configura un escucha de eventos para clics en todo el documento. Cuando el usuario hace clic, se invoca la función myFunction. Antes de que el usuario abandone la página (por ejemplo, cambie de vista o cierre la pestaña), se elimina el escucha de eventos de clic. Cuando el usuario regresa a la vista original (por ejemplo, navega hacia atrás), se vuelve a agregar el escucha de eventos de clic.

Definición de new.html.erb:

Esta vista se encuentra en **app/views/cardfgs/new.html.erb** (anexo B.6), tiene por archivo new.html.erb, esta encargada de crear el formulario para la tarjeta de enfoque de grupo.

Figura 43. app/views/cardfgs/new.html.erb



```
<%= render partial: 'shared/navigation_bar', object: current_user %>
<% provide(:title, t('customers_section')) %>
<%= render partial: 'shared/igheader', locals: {
    title: t("build"),
    secondaryTitle:  t("customers_section"),
    hideVpSelector: false,
    hideTooltipsButton: true
} %>

<div class="container">
<div class="card text-center ">
  <div class="card-header jumbotron">
    Crear  Tarjeta de Enfoque de Grupo
  </div>
  <div class="card-body">
<%= form_for [@customer, @cardfg] do |f| %>
  <!-- Campos -->
   <div class="form-group">
    <%= f.label :Nombre_de_la_Tarjeta, class: "control-label" %>
    <%= f.text_field :titulo_tarjeta, class: "form-control",  maxlength: 20 %>
  </div>
  <div class="form-group">
    <%= f.label :Fecha_de_la_tarjeta_enfoque_de_grupo, class: "control-label" %>
    <%= f.date_field :fecha_tarjeta, class: "form-control" %>
  </div>
   <div class="form-group">
    <%= f.label :Logistica, class: "control-label" %>
    <%= f.text_field :logistica, class: "form-control", maxlength: 150 %>
  </div>
   <div class="form-group">
    <%= f.label :Dirección, class: "control-label" %>
    <%= f.text_field :direccion, class: "form-control", maxlength: 50 %>
  </div>
  <div class="form-group">
    <%= f.label :Descripción, class: "control-label" %>
    <%= f.text_field :descripcion_t, class: "form-control", maxlength: 300 %>
  </div>
  <div class="form-group">
    <%= f.label :La_Media, class: "control-label" %>
    <%= f.text_field :la_medio, class: "form-control", maxlength: 50 %>
  </div>
<%= f.submit "Crear tarjeta de Focus Group", class: "btn btn-primary" %>
<% end %>
  </div>
  <div class="card-footer text-center jumbotron ">
   <%= link_to '« Volver al Tren de Clientes', customers_path, class: 'btn btn-light', data: { confirm: '¿Seguro que quieres volver?' } %>
  </div>
</div>
 </div>
  </div>
```

El formulario que muestra la figura 43 de **new.html.erb**, se crea utilizando la función form_for de Ruby on Rails.

- **El objeto @customer y @cardfg**: se utilizan para construir la ruta del formulario. Aca se observa la asociacion que tiene @customer con la segunda estación.
- Los campos del formulario están dentro de etiquetas ** .**
- Cada campo tiene una etiqueta (**<%= f.label ... %>**) y un campo de entrada (**<%= f.text_field ... %>** ó **<%= f.date_field ... %>**).
- Los atributos class se utilizan para aplicar estilos CSS a los campos.
- El atributo maxlength limita la longitud máxima de entrada para algunos campos.
- El botón "Crear tarjeta de Focus Group" envía el formulario.
- El enlace "« Volver al Tren de Clientes" redirige al usuario a la página de clientes (customers_path).

Definición de show.html.erb:

La vista de **show.html.erb** ubicada en **app/views/cardfgs/show.html.erb** (anexo B.7) se utiliza para mostrar el formulario que completó el usuario.

Figura 44. app/views/cardfgs/show.html.erb

```
app > views > cardfgs > show.html.erb > ? > ? > ?
<%= render partial: 'shared/navigation_bar', object: current_user %>
<% provide(:title, t('customers_section')) %>
<%= render partial: 'shared/igheader', locals: {
    title: t("build"),
    secondaryTitle:  t("customers_section"),
    hideVpSelector: false,
    hideTooltipsButton: true
} %>

<div class="container">
<div class="card text-center ">
<div class="card-header jumbotron">
  Tarjeta Creada
</div>
<div class="card-body">

    <p>
    <strong>Titulo de la tarjeta:</strong>
      <%= @cardfg.titulo_tarjeta %>
    </p>
    <p>
    <strong>Número de tarjeta:</strong>
      <%= @cardfg.id %>
    </p>
    <p>
    <strong>Fecha de la Tarjeta:</strong>
      <%= @cardfg.fecha_tarjeta %>
    </p>
    <p>
    <strong>Logística:</strong>
      <%= @cardfg.logistica %>
    </p>
    <p>
      <strong>Dirección:</strong>
        <%= @cardfg.direccion %>
      </p>
    <p>
    <strong>Descripción:</strong>
      <%= @cardfg.descripcion_t %>
    </p>
    <p>
    <strong>La Media:</strong>
      <%= @cardfg.la_medio %>
    </p>
</div>

<div class="card-footer text-center jumbotron ">
 <%= link_to '« Volver', customers_path, class: 'btn btn-light', data: { confirm: '¿Seguro que quieres volver?' } %>
</div>
</div>
</div>
 </div>
```

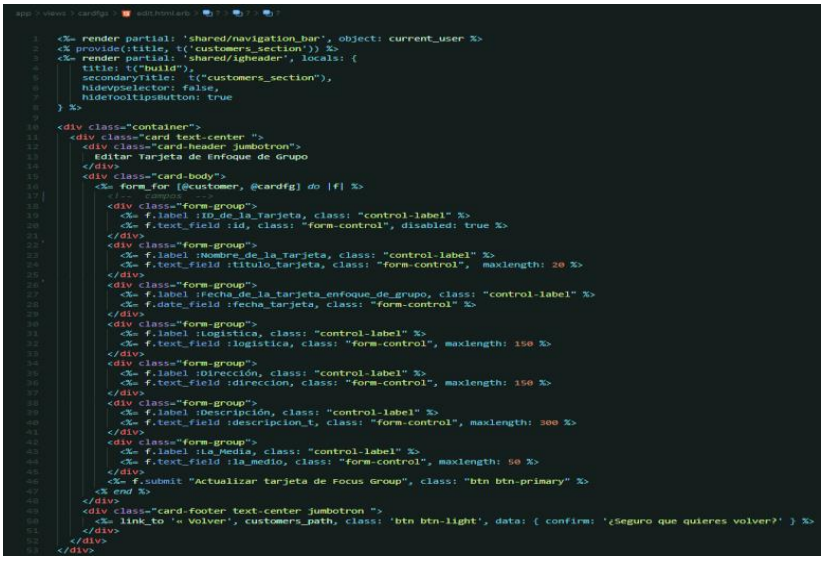
EL conjunto de codigos mostrados en la figura 44, se definieron de la siguiente forma:

- ** **: El código está estructurado en un contenedor para mantener el diseño ordenado.
- ** **: Se utiliza una tarjeta para mostrar la información de la tarjeta creada.
- ** **: La cabecera de la tarjeta muestra el título "Tarjeta Creada".
- ** **: El cuerpo de la tarjeta contiene los detalles de la tarjeta, como el título, número, fecha, logística, dirección, descripción y la media.
- ** **: El pie de la tarjeta contiene un enlace para volver a la página de clientes.
- **<%= ... %>**: Se utiliza para insertar los valores de las variables de Ruby (@cardfg.titulo_tarjeta, @cardfg.id, etc.) en el HTML.
- El enlace "« Volver" Redirige al usuario a la página de clientes (customers_path). Muestra un mensaje de confirmación "¿Seguro que quieres volver?" cuando el usuario hace clic en el enlace.

Definición de edit.html.erb:

EL archivo edit.htmnl.erb esta ubicado en **app/views/cardfgs/edit.html.erb** (anexo B.8), cumple con la funcion de editar el formulario destinado para la segunda estacion.

Figura 45. app/views/cardfgs/edit.html.erb



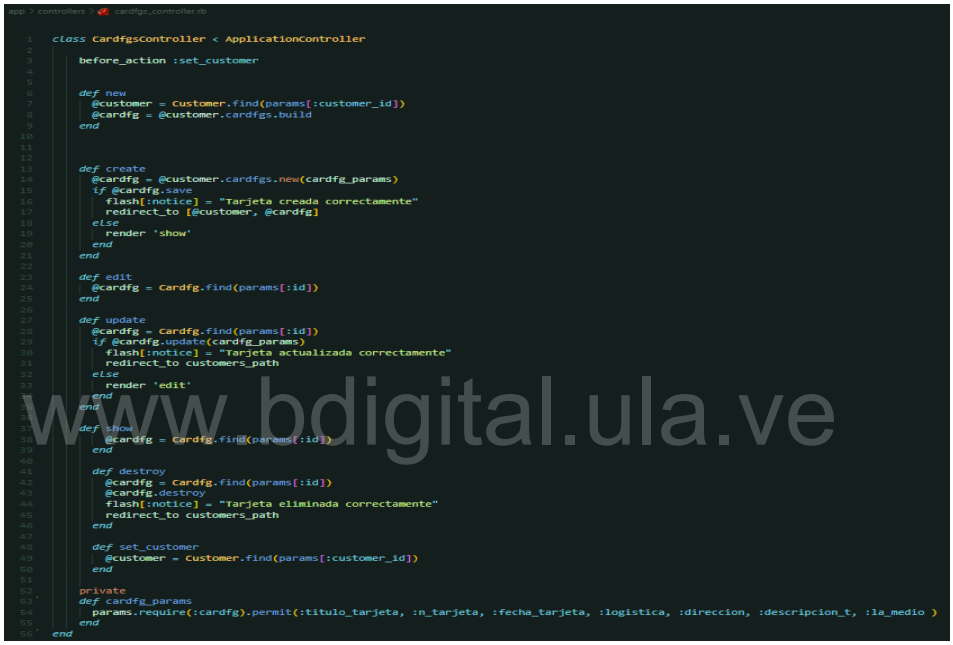
```
<%= render partial: 'shared/navigation_bar', object: current_user %>
<% provide(:title, t('customers_section')) %>
<%= render partial: 'shared/igheader', locals: {
    title: t("build"),
    secondaryTitle:  t("customers_section"),
    hideVpSelector: false,
    hideTooltipsButton: true
} %>

<div class="container">
  <div class="card text-center ">
    <div class="card-header jumbotron">
      Editar Tarjeta de Enfoque de Grupo
    </div>
    <div class="card-body">
      <%= form_for [@customer, @cardfg] do |f| %>
        <!-- campos -->
        <div class="form-group">
          <%= f.label :ID_de_la_Tarjeta, class: "control-label" %>
          <%= f.text_field :id, class: "form-control", disabled: true %>
        </div>
        <div class="form-group">
          <%= f.label :Nombre_de_la_Tarjeta, class: "control-label" %>
          <%= f.text_field :titulo_tarjeta, class: "form-control",  maxlength: 20 %>
        </div>
        <div class="form-group">
          <%= f.label :Fecha_de_la_tarjeta_enfoque_de_grupo, class: "control-label" %>
          <%= f.date_field :fecha_tarjeta, class: "form-control" %>
        </div>
        <div class="form-group">
          <%= f.label :Logistica, class: "control-label" %>
          <%= f.text_field :logistica, class: "form-control", maxlength: 150 %>
        </div>
        <div class="form-group">
          <%= f.label :Dirección, class: "control-label" %>
          <%= f.text_field :direccion, class: "form-control", maxlength: 150 %>
        </div>
        <div class="form-group">
          <%= f.label :Descripción, class: "control-label" %>
          <%= f.text_field :descripcion_t, class: "form-control", maxlength: 300 %>
        </div>
        <div class="form-group">
          <%= f.label :La_Media, class: "control-label" %>
          <%= f.text_field :la_medio, class: "form-control", maxlength: 50 %>
        </div>
        <%= f.submit "Actualizar tarjeta de Focus Group", class: "btn btn-primary" %>
      <% end %>
    </div>
    <div class="card-footer text-center jumbotron ">
      <%= link_to '« Volver', customers_path, class: 'btn btn-light', data: { confirm: '¿Seguro que quieres volver?' } %>
    </div>
  </div>
</div>
```

La figura 45, tiene una explicacion similar a **new.html.erb**, solo que acá se utiliza la función del controlador para editar sobre un formulario ya existente del ID seleccionado.

5.2.3. Controlador de la segunda estación: Enfoque de grupo

El controlador se ubica en **app/controllers/cardfgs_controller.rb**, el archivo **cardfgs_controller.rb** gestiona todas las funciones (acciones) y alberga todos los parámetros necesarios para operar los formularios.

Figura 46. app/controllers/cardfgs_controller.rb



```
app > controllers > cardfgs_controller.rb

class CardfgsController < ApplicationController

    before_action :set_customer


    def new
      @customer = Customer.find(params[:customer_id])
      @cardfg = @customer.cardfgs.build
    end



    def create
      @cardfg = @customer.cardfgs.new(cardfg_params)
      if @cardfg.save
        flash[:notice] = "Tarjeta creada correctamente"
        redirect_to [@customer, @cardfg]
      else
        render 'show'
      end
    end

    def edit
      @cardfg = Cardfg.find(params[:id])
    end

    def update
      @cardfg = Cardfg.find(params[:id])
      if @cardfg.update(cardfg_params)
        flash[:notice] = "Tarjeta actualizada correctamente"
        redirect_to customers_path
      else
        render 'edit'
      end
    end

    def show
        @cardfg = Cardfg.find(params[:id])
      end

      def destroy
        @cardfg = Cardfg.find(params[:id])
        @cardfg.destroy
        flash[:notice] = "Tarjeta eliminada correctamente"
        redirect_to customers_path
      end

      def set_customer
        @customer = Customer.find(params[:customer_id])
      end

    private
    def cardfg_params
      params.require(:cardfg).permit(:titulo_tarjeta, :n_tarjeta, :fecha_tarjeta, :logistica, :direccion, :descripcion_t, :la_medio )
    end
end
```


Definición del controlador cardfgs_controller.rb:

- **before_action :set_customer:** Esto es un filtro que se ejecuta antes de las acciones definidas en el controlador. El método set_customer se llama antes de las acciones new, create, edit, update, show y destroy. Lo que se busca es establecer la variable @customer basada en el parámetro customer_id proporcionado en la URL.
- **def new:** Esta acción se utiliza para mostrar el formulario para crear una nueva tarjeta de enfoque de grupo.
- Se crea una instancia de Cardfg asociada al cliente (@customer) para construir el formulario.

- **def create:** Esta acción se ejecuta cuando se envía el formulario para crear una nueva tarjeta.
- Se crea una nueva instancia de Cardfg con los parámetros proporcionados (cardfg_params). Si la tarjeta se guarda correctamente, se muestra un mensaje de éxito y se redirige al usuario a la página de detalles de la tarjeta.
- Si hay errores en la validación, se muestra nuevamente la página de detalles de la tarjeta.
- **def edit:** Esta acción se utiliza para mostrar el formulario de edición de una tarjeta existente.
- Se busca la tarjeta específica (Cardfg) basada en el ID proporcionado en la URL.
- **def update:** Esta acción se ejecuta cuando se envía el formulario para actualizar una tarjeta existente. se busca la tarjeta específica (Cardfg) basada en el ID proporcionado en la URL. Si la actualización es exitosa, se muestra un mensaje de éxito y se redirige al usuario a la lista de clientes. Si hay errores en la validación, se muestra nuevamente la página de edición de la tarjeta.
- **def show:** Esta acción se utiliza para mostrar los detalles de una tarjeta específica. Se busca la tarjeta específica (Cardfg) basada en el ID proporcionado en la URL.
- **def destroy:** Esta acción se ejecuta cuando se elimina una tarjeta. Se ubica la tarjeta específica (Cardfg) basada en el ID proporcionado en la URL. La tarjeta se destruye y se muestra un mensaje de éxito, luego se redirige al usuario a la lista de clientes.
- **private y cardfg_params**: cardfg_params es un método privado que permite especificar qué parámetros son permitidos para crear o actualizar una tarjeta. Esto ayuda a prevenir ataques de asignación masiva (mass assignment) y garantiza que solo los campos específicos sean modificables.

Base de datos (db migrate) del modelo cardfg.rb

Esta base de datos se generó al momento de crear el modelo de la segunda estación cardfg.rb, cada uno con los atributos necesarios y requeridos por el documento de requisitos del producto. La tabla cardfg.rb contiene múltiples atributos necesarios para el despliegue de dicha estación.

Figura 47. db/migrate/20231003191914_create_cardfgs.rb

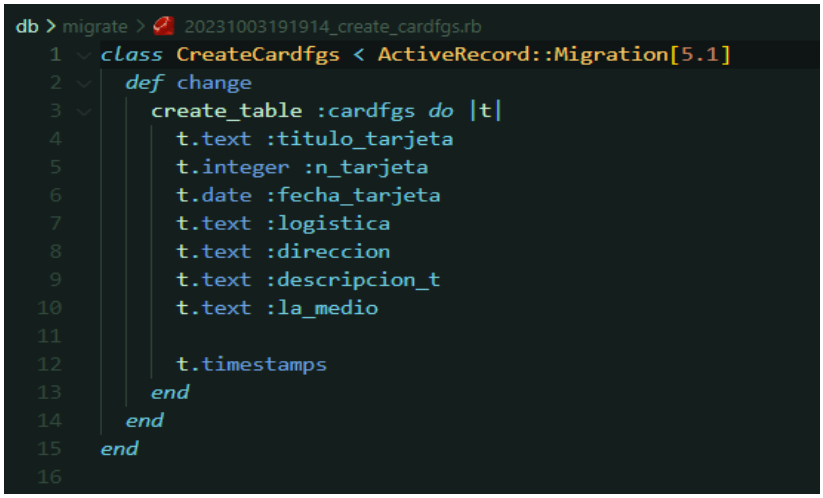


Figura 48. db/migrate/20231014231447_add_customer_ref_to_firstclients.rb

Es importante destacar la asociacion de modelo cardfg con el modelo customer, explicada previamente en el modelo de cardfg.rb

Tabla resumen de la segunda estación: Enfoque de grupo

Tabla 4. Tabla de resumen Segunda Estación: Enfoque de grupo

	Ubicación
Modelo	app/models/cardfg.rb
Vista	app/views/customers/focusg.html.erb, app/views/cardfgs/new.html.erb, app/views/cardfgs/show.html.erb, app/views/cardfgs/edit.html.erb
Controlador	app/controllers/cardfgs_controller.rb

Análisis: Con este conjunto de códigos siguiendo el patrón MVC, se logró que dicha permita la creación de la tarjeta enfoque de grupo, apoyándose de CRUD. La tarjeta que se muestra para cada SdeC contiene los campos requeridos.

5.3. Tercera estación: Primer Cliente

Esta estación del TOC, tiene por finalidad apoyar y orientar a los Cofundadores para que aprendan a identificar un usuario que sea real y sincero, en lo que vive alrededor del producto que se desarrolla. Esta persona debe conocerse profundamente. Es un cliente muy importante.

La estación en cuestión cuenta con dos formularios: Formulario de datos personales y Formulario contexto personal.

5.3.1. Modelo de la Tercera estación: Primer Cliente

Su único archivo **firstclient.rb** se encuentra en **app/models/firstclient.rb** al igual que la estación anterior, mantiene una estrecha relación a customer, previamente explicada.

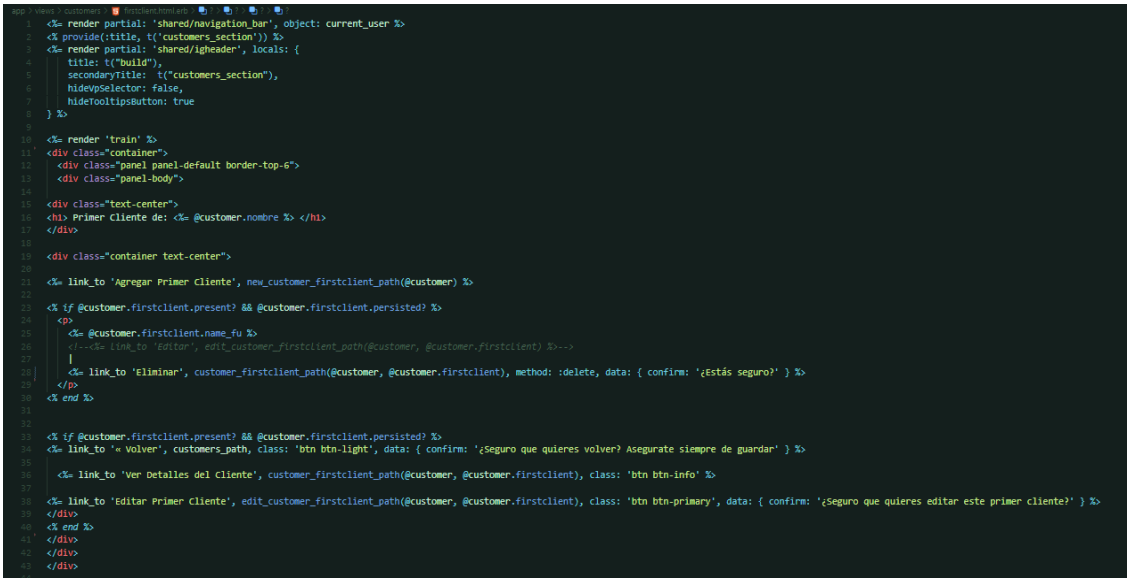
Figura 49. app/models/firstclient.rb


```
app > models > firstclient.rb
1  class Firstclient < ApplicationRecord
2      belongs_to :customer
3  end
```

5.3.2. Vista de la Tercera estación: Primer Cliente

Se encuentra en **app/views/customers/firstclient.html.erb** (anexo B.9) y su archivo se identifica como **firstclient.html.erb**, es la vista principal de la tercera estación, desde acá el usuario podrá agregar su primer cliente, solo podrá agregar uno por vez.

Figura 50. app/views/customers/firstclient.html.erb



```
<%= render partial: 'shared/navigation_bar', object: current_user %>
<% provide(:title, t('customers_section')) %>
<%= render partial: 'shared/igheader', locals: {
    title: t("build"),
    secondaryTitle:  t("customers_section"),
    hideVpSelector: false,
    hideTooltipsButton: true
} %>

<%= render 'train' %>
<div class="container">
  <div class="panel panel-default border-top-6">
  <div class="panel-body">

<div class="text-center">
<h1> Primer Cliente de: <%= @customer.nombre %> </h1>
</div>

<div class="container text-center">

<%= link_to 'Agregar Primer Cliente', new_customer_firstclient_path(@customer) %>

<% if @customer.firstclient.present? && @customer.firstclient.persisted? %>
  <p>
    <%= @customer.firstclient.name_fu %>
    <!--<%= link_to 'Editar', edit_customer_firstclient_path(@customer, @customer.firstclient) %>-->
    |
    <%= link_to 'Eliminar', customer_firstclient_path(@customer, @customer.firstclient), method: :delete, data: { confirm: '¿Estás seguro?' } %>
  </p>
<% end %>


<% if @customer.firstclient.present? && @customer.firstclient.persisted? %>
<%= link_to '« Volver', customers_path, class: 'btn btn-light', data: { confirm: '¿Seguro que quieres volver? Asegurate siempre de guardar' } %>

  <%= link_to 'Ver Detalles del Cliente', customer_firstclient_path(@customer, @customer.firstclient), class: 'btn btn-info' %>

<%= link_to 'Editar Primer Cliente', edit_customer_firstclient_path(@customer, @customer.firstclient), class: 'btn btn-primary', data: { confirm: '¿Seguro que quieres editar este primer cliente?' } %>
</div>
<% end %>
</div>
</div>
</div>
```

Las siguientes definiciones pertenecen a la figura 50

Definición de la vista firstclient.html.erb

- **<%= render 'train' %>**: Es el parcial que se invoca desde _train.html.erb. Renderiza un componente llamado “train”:
- La sección ** ... **: muestra el nombre del primer cliente del objeto @customer. Si existe un primer cliente y está almacenado en la base de datos, se muestra su nombre y se proporcionan enlaces para editar o eliminar. Los enlaces “Agregar Primer Cliente”, “Ver Detalles del Cliente” y “Editar Primer Cliente” permiten al usuario interactuar con los datos del cliente.
- **<%= link_to '« Volver al Tren de Clientes', … %>**: permite volver a la vista principal del Tren de Clientes.

Definición de la vista new.html.erb

Esta vista muestra los formularios de la tercera estación, permite al usuario crear y completar los formularios correspondientes reflejados en el documento de requisitos del producto Tren de Clientes. Su archivo es **new.html.erb** y la su ubicación **app/views/firstclients/new.html.erb** (anexo B.10).

Figura 51. app/views/firstclients/new.html.erb (parte 1)

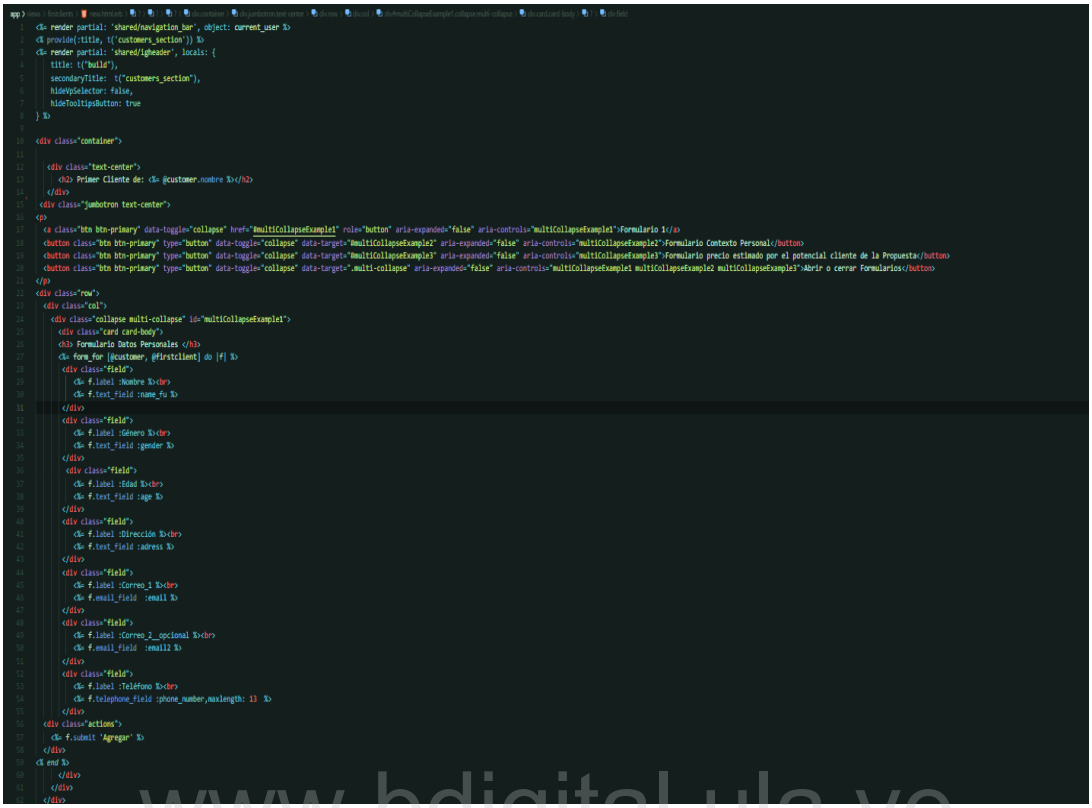
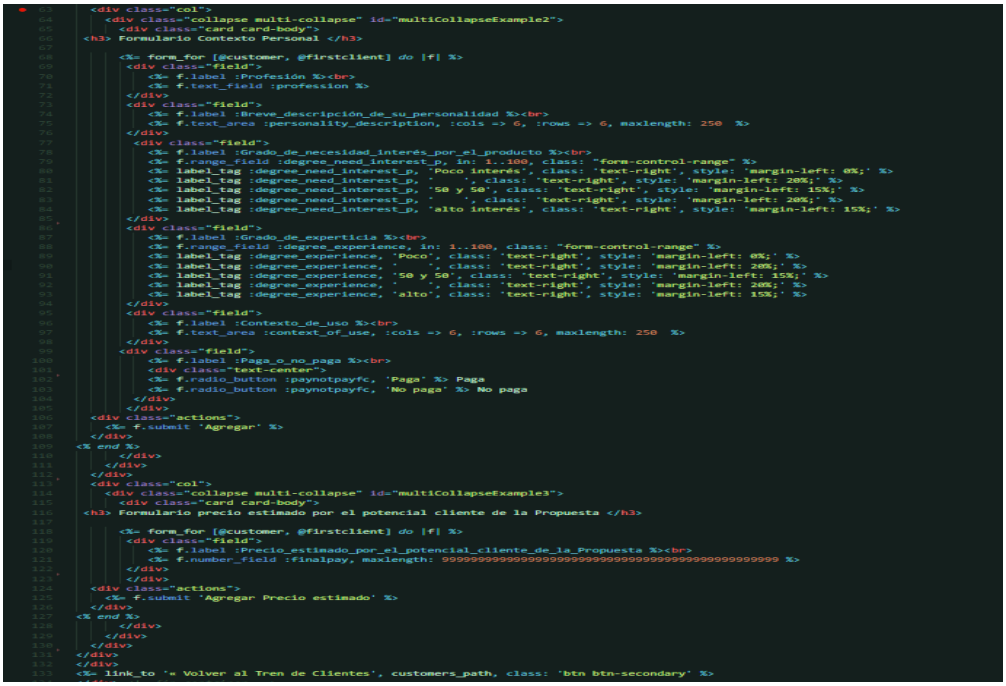


Figura 52. app/views/firstclients/new.html.erb (parte 2)



Las siguientes definiciones pertenecen a las figuras 51 y 52:

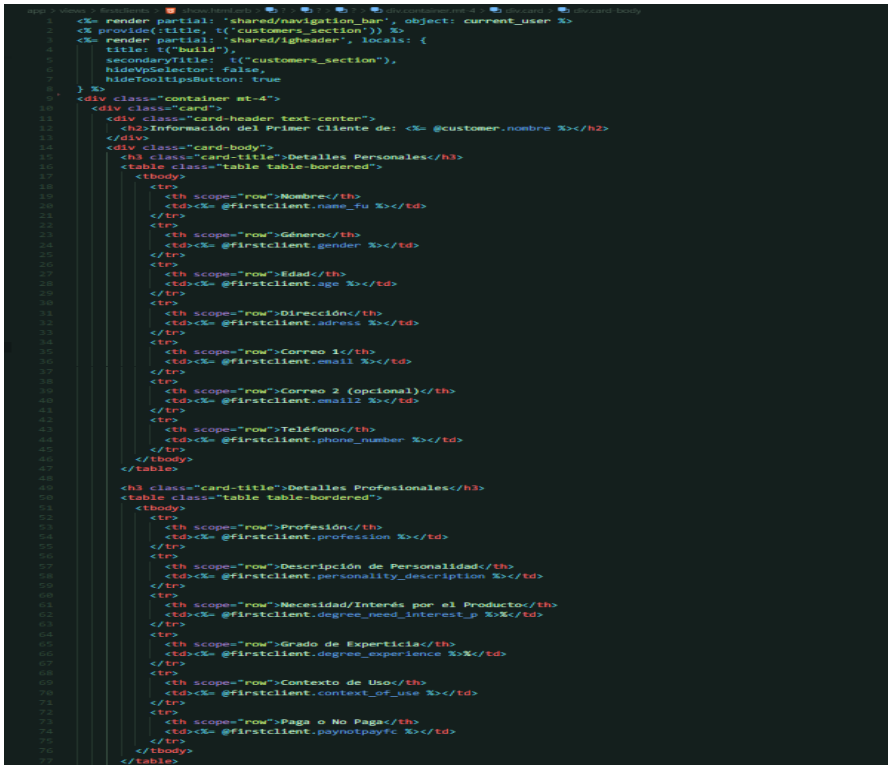
Los botones “Formulario 1”, “Formulario Contexto Personal” y “Formulario precio estimado por el potencial cliente de la Propuesta” permiten al usuario abrir o cerrar diferentes secciones de formularios. El último botón, “Abrir o cerrar Formularios”, controla la visibilidad de todos los formularios a la vez.

- **Formulario Datos Personales**: este formulario recopila información personal sobre el primer cliente. Los campos contienen: Nombre, Género, Edad, Dirección, Correo 1, Correo 2 (opcional) y Teléfono (con una longitud máxima de 13 caracteres).
- **Botón “Agregar”**: al hacer clic en este botón, se enviarán los datos ingresados en el formulario para agregar al primer cliente.

Definición de la vista show.html.erb

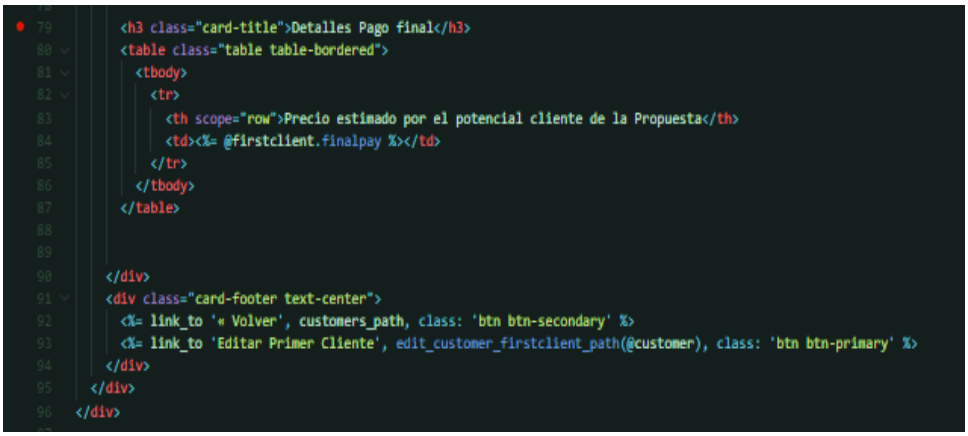
Esta vista se encuentra en **app/views/firstclients/show.html.erb** (anexo B.11), su archivo es **show.html.erb**, una vez creado el primer cliente esta vista se encarga de mostrar los formularios completados por el usuario.

Figura 53. app/views/firstclients/new.html.erb (parte 1)



```
<%= render partial: 'shared/navigation_bar', object: current_user %>
<% provide(:title, t("customers_section")) %>
<%= render partial: 'shared/igheader', locals: {
    title: t("build"),
    secondaryTitle:  t("customers_section"),
    hideVpSelector: false,
    hideTooltipsButton: true
} %>
<div class="container mt-4">
  <div class="card">
    <div class="card-header text-center">
      <h2>Información del Primer Cliente de: <%= @customer.nombre %></h2>
    </div>
    <div class="card-body">
      <h3 class="card-title">Detalles Personales</h3>
      <table class="table table-bordered">
        <tbody>
          <tr>
            <th scope="row">Nombre</th>
            <td><%= @firstclient.name_fu %></td>
          </tr>
          <tr>
            <th scope="row">Género</th>
            <td><%= @firstclient.gender %></td>
          </tr>
          <tr>
            <th scope="row">Edad</th>
            <td><%= @firstclient.age %></td>
          </tr>
          <tr>
            <th scope="row">Dirección</th>
            <td><%= @firstclient.adress %></td>
          </tr>
          <tr>
            <th scope="row">Correo 1</th>
            <td><%= @firstclient.email %></td>
          </tr>
          <tr>
            <th scope="row">Correo 2 (opcional)</th>
            <td><%= @firstclient.email2 %></td>
          </tr>
          <tr>
            <th scope="row">Teléfono</th>
            <td><%= @firstclient.phone_number %></td>
          </tr>
        </tbody>
      </table>

      <h3 class="card-title">Detalles Profesionales</h3>
      <table class="table table-bordered">
        <tbody>
          <tr>
            <th scope="row">Profesión</th>
            <td><%= @firstclient.profession %></td>
          </tr>
          <tr>
            <th scope="row">Descripción de Personalidad</th>
            <td><%= @firstclient.personality_description %></td>
          </tr>
          <tr>
            <th scope="row">Necesidad/Interés por el Producto</th>
            <td><%= @firstclient.degree_need_interest_p %>%</td>
          </tr>
          <tr>
            <th scope="row">Grado de Experticia</th>
            <td><%= @firstclient.degree_experience %>%</td>
          </tr>
          <tr>
            <th scope="row">Contexto de Uso</th>
            <td><%= @firstclient.context_of_use %></td>
          </tr>
          <tr>
            <th scope="row">Paga o No Paga</th>
            <td><%= @firstclient.paynotpayfc %></td>
          </tr>
        </tbody>
      </table>
```

Figura 54. app/views/firstclients/new.html.erb (parte 2)



```
    <h3 class="card-title">Detalles Pago final</h3>
    <table class="table table-bordered">
      <tbody>
        <tr>
          <th scope="row">Precio estimado por el potencial cliente de la Propuesta</th>
          <td><%= @firstclient.finalpay %></td>
        </tr>
      </tbody>
    </table>


  </div>
  <div class="card-footer text-center">
    <%= link_to '« Volver', customers_path, class: 'btn btn-secondary' %>
    <%= link_to 'Editar Primer Cliente', edit_customer_firstclient_path(@customer), class: 'btn btn-primary' %>
  </div>
 </div>
</div>
```

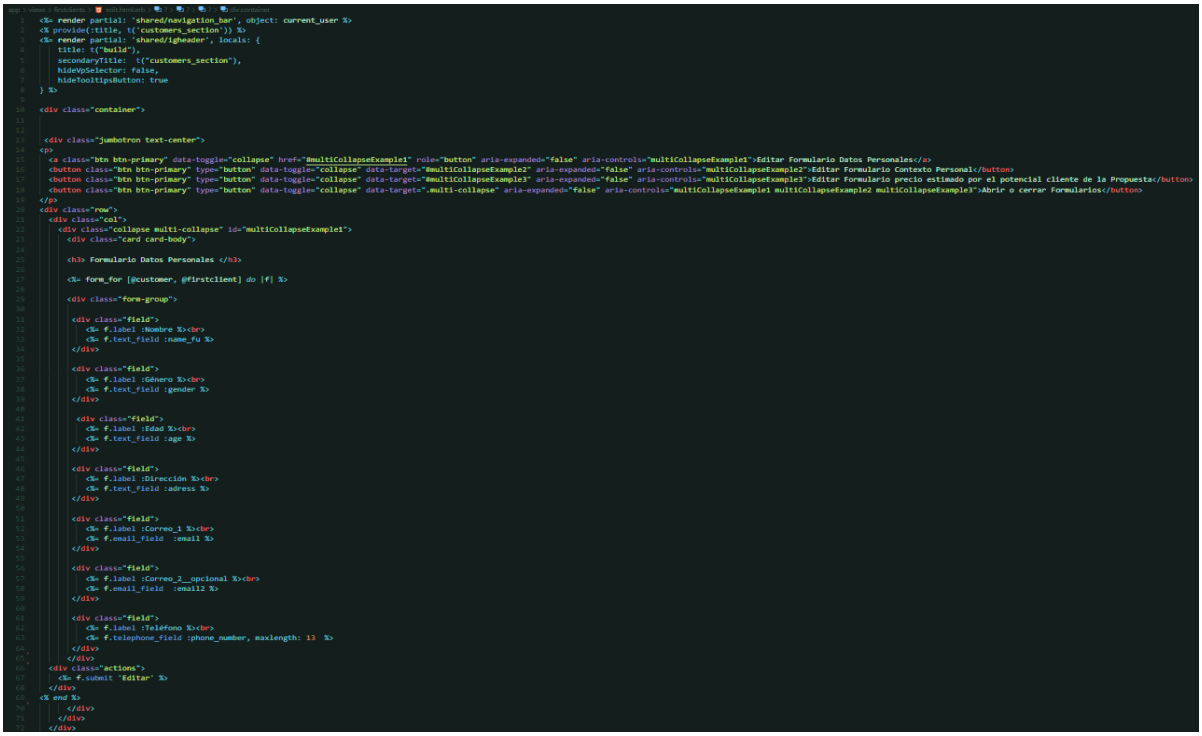
Las siguientes definiciones pertenecen a las figuras 53 y 54:

- **Encabezado y Detalles Personales:** el encabezado refleja el nombre del primer cliente. La tabla "Detalles Personales" contiene información como: Nombre, Género, Edad, Dirección, Correo 1, Correo 2 (opcional) y Teléfono
- **Detalles Profesionales**: la tabla "Detalles Profesionales" incluye información como: Profesión, Descripción de Personalidad, Necesidad/Interés por el Producto(expresado como un porcentaje), Grado de Experticia, Contexto de Uso, Paga o No Paga
- **Detalles Pago Final:** la tabla "Detalles Pago final" muestra el "Precio estimado por el potencial cliente de la Propuesta".
- **Botones de Navegación:** el botón "« Volver" permite regresar al Tren de Clientes. El botón "Editar Primer Cliente" lleva al formulario de edición para este cliente.

Definición de la vista edit.html.erb

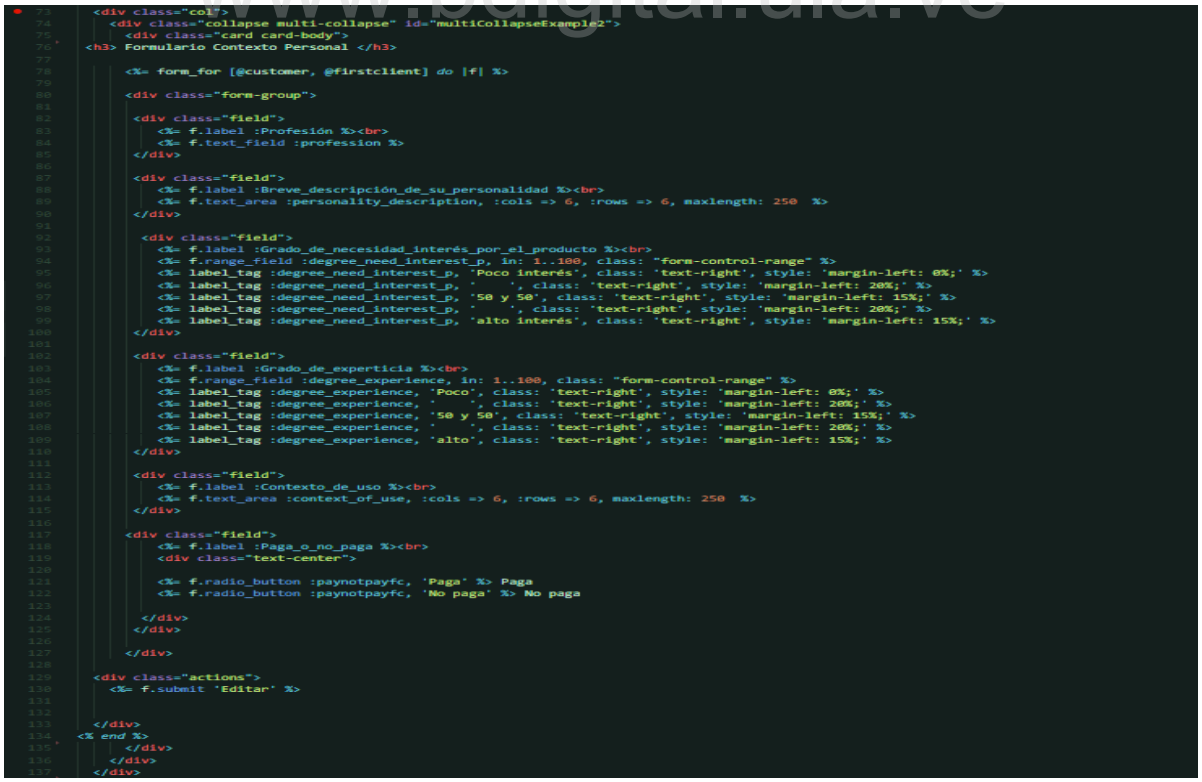
Ubicada en **app/views/firstclients/edit.html.erb** (anexo B.12), con su archivo **edit.html.erb**, permite editar los formularios correspondientes de la tercera estación.

Figura 55. app/views/firstclients/edit.html.erb (parte 1)



```
<%= render partial: 'shared/navigation_bar', object: current_user %>
<% provide(:title, t('customers_section')) %>
<%= render partial: 'shared/igheader', locals: {
    title: t("build"),
    secondaryTitle: t("customers_section"),
    hideVpSelector: false,
    hideTooltipsButton: true
} %>

<div class="container">


 <div class="jumbotron text-center">
 <p>
  <a class="btn btn-primary" data-toggle="collapse" href="#multiCollapseExample1" role="button" aria-expanded="false" aria-controls="multiCollapseExample1">Editar Formulario Datos Personales</a>
  <button class="btn btn-primary" type="button" data-toggle="collapse" data-target="#multiCollapseExample2" aria-expanded="false" aria-controls="multiCollapseExample2">Editar Formulario Contexto Personal</button>
  <button class="btn btn-primary" type="button" data-toggle="collapse" data-target="#multiCollapseExample3" aria-expanded="false" aria-controls="multiCollapseExample3">Editar Formulario precio estimado por el potencial cliente de la Propuesta</button>
  <button class="btn btn-primary" type="button" data-toggle="collapse" data-target=".multi-collapse" aria-expanded="false" aria-controls="multiCollapseExample1 multiCollapseExample2 multiCollapseExample3">Abrir o cerrar Formularios</button>
 </p>
 <div class="row">
  <div class="col">
    <div class="collapse multi-collapse" id="multiCollapseExample1">
      <div class="card card-body">

      <h3> Formulario Datos Personales </h3>

      <%= form_for [@customer, @firstclient] do |f| %>

      <div class="form-group">

       <div class="field">
          <%= f.label :Nombre %><br>
          <%= f.text_field :name_fu %>
        </div>

        <div class="field">
          <%= f.label :Género %><br>
          <%= f.text_field :gender %>
        </div>

         <div class="field">
          <%= f.label :Edad %><br>
          <%= f.text_field :age %>
        </div>

        <div class="field">
          <%= f.label :Dirección %><br>
          <%= f.text_field :adress %>
        </div>

        <div class="field">
          <%= f.label :Correo_1 %><br>
          <%= f.email_field  :email %>
        </div>

        <div class="field">
          <%= f.label :Correo_2__opcional %><br>
          <%= f.email_field  :email2 %>
        </div>

        <div class="field">
          <%= f.label :Teléfono %><br>
          <%= f.telephone_field :phone_number, maxlength: 13  %>
        </div>
       </div>
  <div class="actions">
    <%= f.submit 'Editar' %>
  </div>
<% end %>
    </div>
   </div>
  </div>
```

Figura 56. app/views/firstclients/edit.html.erb (parte 2)



```
    <div class="col">
      <div class="collapse multi-collapse" id="multiCollapseExample2">
        <div class="card card-body">
    <h3> Formulario Contexto Personal </h3>

          <%= form_for [@customer, @firstclient] do |f| %>

          <div class="form-group">

           <div class="field">
              <%= f.label :Profesión %><br>
              <%= f.text_field :profession %>
            </div>

            <div class="field">
              <%= f.label :Breve_descripción_de_su_personalidad %><br>
              <%= f.text_area :personality_description, :cols => 6, :rows => 6, maxlength: 250  %>
            </div>

             <div class="field">
               <%= f.label :Grado_de_necesidad_interés_por_el_producto %><br>
               <%= f.range_field :degree_need_interest_p, in: 1..100, class: "form-control-range" %>
               <%= label_tag :degree_need_interest_p, 'Poco interés', class: 'text-right', style: 'margin-left: 0%;' %>
               <%= label_tag :degree_need_interest_p, '     ', class: 'text-right', style: 'margin-left: 20%;' %>
               <%= label_tag :degree_need_interest_p, '50 y 50', class: 'text-right', style: 'margin-left: 15%;' %>
               <%= label_tag :degree_need_interest_p, '     ', class: 'text-right', style: 'margin-left: 20%;' %>
               <%= label_tag :degree_need_interest_p, 'alto interés', class: 'text-right', style: 'margin-left: 15%;' %>
            </div>

            <div class="field">
              <%= f.label :Grado_de_experticia %><br>
              <%= f.range_field :degree_experience, in: 1..100, class: "form-control-range" %>
              <%= label_tag :degree_experience, 'Poco', class: 'text-right', style: 'margin-left: 0%;' %>
              <%= label_tag :degree_experience, '     ', class: 'text-right', style: 'margin-left: 20%;' %>
              <%= label_tag :degree_experience, '50 y 50', class: 'text-right', style: 'margin-left: 15%;' %>
              <%= label_tag :degree_experience, '     ', class: 'text-right', style: 'margin-left: 20%;' %>
              <%= label_tag :degree_experience, 'alto', class: 'text-right', style: 'margin-left: 15%;' %>
            </div>

            <div class="field">
              <%= f.label :Contexto_de_uso %><br>
              <%= f.text_area :context_of_use, :cols => 6, :rows => 6, maxlength: 250  %>
            </div>

            <div class="field">
              <%= f.label :Paga_o_no_paga %><br>
              <div class="text-center">

              <%= f.radio_button :paynotpayfc, 'Paga' %> Paga
              <%= f.radio_button :paynotpayfc, 'No paga' %> No paga

             </div>
            </div>

          </div>

      <div class="actions">
        <%= f.submit 'Editar' %>


      </div>
    <% end %>
        </div>
      </div>
    </div>
```

Figura 57. app/views/firstclients/edit.html.erb (parte 3)

```
  <div class="col">
    <div class="collapse multi-collapse" id="multiCollapseExample3">
      <div class="card card-body">
<h3> Formulario precio estimado por el potencial cliente de la Propuesta </h3>

      <%= form_for [@customer, @firstclient] do |p| %>

      <div class="form-group">

       <div class="field">
          <%= p.label :Precio_estimado_por_el_potencial_cliente_de_la_Propuesta %><br>
          <%= p.number_field :finalpay, maxlength: 99999999999999999999999999999999999999999999999999 %>
       </div>


       </div>

      </div>

  <div class="actions">
    <%= p.submit 'Editar' %>

  </div>
<% end %>

      </div>
    </div>
  </div>

</div> <!--fin row-->
</div><!--fin container -->
<div class="text-center">
   <%= link_to '« Volver al Tren de Clientes', customers_path, class: 'btn btn-secondary' %>
</div>
```

Las siguientes definiciones pertenecen a las figuras 55, 56 y 57:

- **Formulario Datos Personales:** Sirve para recopilar información personal de un cliente. Los campos incluidos son: Nombre, Género, Edad, Dirección, Correo_1, Correo_2__opcional, y Teléfono. El formulario utiliza la sintaxis de Ruby on Rails para definir los campos y las etiquetas correspondientes.
- El método **form_for** Se utiliza para generar el formulario HTML con los campos especificados.
- **Acciones:** El botón “Editar” dentro del formulario envía los datos ingresados por el cliente al servidor para su procesamiento o almacenamiento.
- **Formulario Contexto Personal:** Este formulario se relaciona con el contexto personal del cliente o usuario. Los campos incluidos son: Profesión, Breve descripción de su personalidad, grado de necesidad/interés por el producto: un control deslizante (rango) que permite al cliente especificar su nivel de interés en el producto. Va desde “Poco interés” hasta “Alto interés”. Grado de experticia: otro control deslizante para indicar el nivel de experiencia o conocimiento del cliente. Va desde “Poco” hasta “Alto” y contexto de uso;

paga o no paga. El formulario utiliza la sintaxis de Ruby on Rails para definir los campos y las etiquetas correspondientes.

- **Formulario Precio Estimado por el Potencial Cliente de la Propuesta:** Este formulario tiene que ver con el precio estimado que el cliente está dispuesto a pagar por la propuesta. El campo incluido es: precio estimado por el potencial cliente de la propuesta, El formulario también utiliza la sintaxis de Ruby on Rails.
- **Volver al Tren de Clientes:** Enlace para volver al "Tren de Clientes".

5.3.3. Controlador de la Tercera estación: Primer Cliente

El controlador de la estación Primer Cliente está ubicado en **app/controllers/firstclients_controller.rb**, cuyo archivo tiene de nombre **firstclients_controller.rb**, gestiona todas las acciones correspondientes para el modelo, llevando al usuario a la vista correspondiente según la acción seleccionada.

Figura 58. app/controllers/firstclients_controller.rb


```ruby
class FirstclientsController < ApplicationController

  before_action :set_customer, only: [:new, :create, :edit, :update, :destroy]
  before_action :set_firstclient, only: [:show, :edit, :update, :destroy]
  def new
    @customer = Customer.find(params[:customer_id])
    @firstclient = @customer.build_firstclient
  end

  def create
    @customer = Customer.find(params[:customer_id])
    @firstclient = @customer.build_firstclient(firstclient_params)
    if @firstclient.save
      flash[:notice] = "Primer Cliente creado correctamente"
      redirect_to customer_firstclient_path(@customer, @firstclient)
    else
      render :new
    end
  end
  #para editar con la relacion 1:1 En este caso, no necesitas usar find(params[:id]) después de @customer.firstclient porque @customer.firstclient ya te dará el firstclient asociado con ese customer específico.
  def edit
    @customer = Customer.find(params[:customer_id])
  @firstclient = @customer.firstclient
  end

  def update
    @firstclient = Firstclient.find(params[:id])
    if @firstclient.update(firstclient_params)
      flash[:notice] = "Primer Cliente actualizado correctamente"
      redirect_to customers_path
    else
      render 'edit'
    end
  end

  def show
    @firstclient = Firstclient.find_by(id: params[:id])
    if @firstclient.nil?
      redirect_to firstclients_path, alert: "Cliente no encontrado"
    end
  end


  def destroy
    @firstclient = Firstclient.find(params[:id])
    @firstclient.destroy
    flash[:notice] = "Primer Cliente actualizado correctamente"
    redirect_to customers_path
  end
  private
  def set_customer
    @customer = Customer.find(params[:customer_id])
  end

  def set_firstclient
    @customer = Customer.find(params[:customer_id])
    @firstclient = @customer.firstclient
  end
  def firstclient_params
    params.require(:firstclient).permit(:name_fu, :gender, :age, :adress, :email, :email2, :phone_number, :profession, :personality_description, :degree_need_interest_p, :degree_experience, :context_of_use, :paynotpayfc, :finalpay )
  end
end
```

Las siguientes definiciones pertenecen a la figura 58:

Acciones del Controlador:

- **El FirstclientsController:** Define varias acciones que manejan diferentes aspectos de la gestión de los primeros clientes asociados con los clientes. Estas acciones incluyen new, create, edit, update, show y destroy.
- **Before Actions (Acciones Previas):** Los callbacks before_action se utilizan para ejecutar métodos específicos antes de ciertas acciones.
- **before_action** :set_customer: Asegura que el método set_customer se llame antes de las acciones especificadas (new, create, edit, update, destroy).
- **before_action :set_firstclient:** De manera similar, este método se llama antes de las acciones show, edit, update y destroy.
- **Acción New (Nuevo):** La acción new inicializa un nuevo objeto Firstclient asociado con un cliente específico y prepara el formulario para crear un nuevo primer cliente.
- **Acción Create (Crear**): La acción create maneja el envío de los datos del formulario al crear un nuevo primer cliente. Construye un nuevo objeto Firstclient basado en los parámetros recibidos del formulario. Si el cliente se guarda correctamente, redirige a la customer_firstclient_path (página de detalles del cliente recién creado). De lo contrario, renderiza nuevamente la plantilla new.
- **Acción Edit (Editar):** La acción edit recupera un Firstclient existente asociado con un cliente específico. Prepara el formulario para editar los detalles del cliente.
- **Acción Update (Actualizar):** La acción update maneja el envío de los datos del formulario al actualizar un primer cliente existente. Encuentra el cliente por su ID, actualiza sus atributos según los parámetros del formulario y guarda los cambios. Si la actualización es exitosa, redirige a customers_path. De lo contrario, renderiza nuevamente la plantilla edit.
- **Acción Show (Mostrar):** La acción show recupera un Firstclient existente por su ID. Si se encuentra el cliente, muestra los detalles del cliente. De lo contrario, redirige a firstclients_path con un mensaje de alerta.
- **Acción Destroy (Eliminar):** La acción destroy elimina un Firstclient existente por su ID. Después de la eliminación, redirige a customers_path con un mensaje de aviso.

Base de datos (db migrate) del modelo firstclient.rb

Esta base de datos se generó al momento de crear el modelo de la tercera estación firstclient.rb, cada uno con los atributos necesarios y requeridos por el documento de requisitos del producto. La tabla firstclient.rb contiene múltiples atributos necesarios para el despliegue de dicha estación.

Figura 59. db/migrate/20231014230244_create_firstclients.rb

```ruby
class CreateFirstclients < ActiveRecord::Migration[5.1]
  def change
    create_table :firstclients do |t|
      t.string :name_fu
      t.string :gender
      t.integer :age
      t.string :adress
      t.string :email
      t.string :email2
      t.integer :phone_number
      t.string :profession
      t.text :personality_description
      t.integer :degree_need_interest_p
      t.integer :degree_experience
      t.text :context_of_use

      t.timestamps
    end
  end
end
```

Figura 60. db/migrate/20231014231447_add_customer_ref_to_firstclients.rb


```ruby
class AddCustomerRefToFirstclients < ActiveRecord::Migration[5.1]
  def change
    add_reference :firstclients, :customer, foreign_key: true
  end
end
```

Figura 61. db/migrate/20240217015305_add_finalpay_to_firstclients.rb

```ruby
class AddFinalpayToFirstclients < ActiveRecord::Migration[5.1]
  def change
    add_column :firstclients, :finalpay, :bigint
  end
end
```

Tabla resumen de la Tercera estación: Primer Cliente

Tabla 5. Tabla resumen de la Tercera estación: Primer Cliente

	Ubicación
Modelo	app/models/firstclient.rb
Vista	app/views/customers/firstclient.html.erb, app/views/firstclients/new.html.erb, app/views/firstclients/show.html.erb, app/views/firstclients/edit.html.erb
Controlador	app/controllers/firstclients_controller.rb

Análisis: Con este conjunto de códigos se logró levantar el espacio de la tercera estación, que comprenden los dos formularios: formularios de datos personales y formulario de contexto personal. De tal manera de que el cliente pueda introducir los datos requeridos de forma exitosa y esos datos queden contenidos de forma segura en la base de datos. Se facilita poder ver el formulario creado y poderlo editarlo, cada uno con su respectivo botón.

5.4. Cuarta estación: Primer Grupo de Clientes

Esta estación ubica al SdeC seleccionado y le permite agregar al primer cliente del grupo de clientes, puede crear una buena cantidad de clientes según sus necesidades. Cada Segmento tiene una lista con todos los clientes agregados, podrá ver detalles de cliente, editar y eliminar individualmente cada cliente previamente creado.

Cuando se crea el primer cliente del grupo de clientes, el usuario podrá completar dos formularios: Información personal e Información de contexto.

5.4.1. Modelo de la Cuarta estación: Primer Grupo de Clientes

El modelo de la cuarta estación está ubicado en **app/models/firstcg.rb**, su archivo es **firstcg.rb**

Figura 62. app/models/firstcg.rb

```ruby
class Firstcg < ApplicationRecord
    belongs_to :customer

    validates :phonecg, presence: true, format: { with: /\A\d+\z/, message: "Debe ser un número de teléfono válido." }, uniqueness: true


  end
```

Este modelo contiene una validación para el atributo phonecg, el número de teléfono no puede estar vacío al momento de crear o editar, si el usuario llena el campo con letras o caracteres especiales se guardará el campo con un 0.

5.4.2. Vista de la Cuarta estación: Primer Grupo de Clientes

Definición de la vista firstcg.html.erb: es la primera vista de la cuarta estación, se encuentra en **app/views/customers/firstcg.html.erb** (anexo B.13), su archivo **firstcg.html.erb** contiene todo lo necesario para que el usuario gestione múltiples clientes. Con la capacidad de crear, editar, mostrar y eliminar clientes.

Figura 63. app/views/customers/firstcg.html.erb (parte 1)

```erb
<%= render partial: 'shared/navigation_bar', object: current_user %>
<% provide(:title, t('customers_section')) %>
<%= render partial: 'shared/igheader', locals: {
    title: t("build"),
    secondaryTitle:  t("customers_section"),
    hideVpSelector: false,
    hideTooltipsButton: true
} %>
<style>
  .milist li {
    transition: transform .2s;
  }
  .milist li:hover {
    transform: scale(1.2);
  }
  </style>
<%= render 'train' %>
<div class="container">
  <div class="panel panel-default border-top-6">
  <div class="panel-body">
<h1 class="text-center"> Editar Estación: Primer Grupo de Clientes: <%= @customer.nombre %></h1>
<div class="col text-center">
<button type="button" class="btn btn-light  " ><a> + <%= link_to 'Agregar Primer Cliente de Grupo de Clientes ', new_customer_firstcg_path(@customer) %> </a></button>
</div>
<% if @customer.firstcgs.any? %>
  <div class="col text-center">
    <table class="table table-bordered ">
        <thead>
        <tr>
            <th><h3 class=" text-center"> Tarjeta Primer Grupo de Clientes de: <%= @customer.nombre %></h3></th>
        </tr>
        </thead>
        <tbody>
         <tr>
            <td>
            <% @customer.firstcgs.each do |firstcg| %>
            <% if firstcg.valid? %>
            <div class="milist">
                <li  style="list-style-type: none;">
                  <%= link_to firstcg.namecg1, edit_customer_firstcg_path(@customer, firstcg), data: { confirm: '¿Vas a editar?' } %>
                  <%= "|" %>
                 <%= link_to 'Eliminar', customer_firstcg_path(@customer, firstcg), method: :delete, data: { confirm: '¿Estás seguro?' } %>
                  <%= "|" %>
                </li>
            </div>
            <% end %>
            <% end %>
            </td>
        </tr>
        </tbody>
    </table>
  </div>
<% end %>
  </ul>
  <% if @customer.firstcgs.present? %>
  <div style="list-style-type: none; width: 50%; margin: auto;">
  <h4 class=" text-center">  Selecciona el Primer Grupo de clientes: <%= @customer.nombre %> </h4>
  <%= select_tag "firstcgs", options_from_collection_for_select(@customer.firstcgs, "id", "namecg1"), id: "firstcg_select", prompt: "Selecciona el Primer Cliente" %>
  <%= link_to '« Volver', customers_path, class: 'btn btn-light', data: { confirm: '¿Seguro que quieres volver? Asegurate siempre de guardar' } %>
  <%= link_to 'Ver detalles del cliente', '#', id: 'details_link', class: 'btn btn-primary' %>
  <%= link_to 'Editar', '#', id: 'edit_link', class: 'btn btn-primary' %>
  </div>
<% end %>
</div></div></div>
```

Figura 64. app/views/customers/firstcg.html.erb (parte 2)

Las siguientes definiciones pertenecen a las figuras 63 y 64:

- **Estilos CSS:** Se definen algunos estilos CSS, estos estilos afectan a los elementos HTML con la clase .milist.
- **El estilo transition: transform .2s**; Establece una transición suave de 0.2 segundos cuando se aplica una transformación (como el escalado) a los elementos.
- **El estilo :hover**: Se aplica cuando el cursor está sobre un elemento. En este caso, cuando el cursor está sobre un elemento con la clase .milist, se escala el elemento al 120% de su tamaño original.
- **Renderización de una vista parcial:** La línea <%= render 'train' %> renderiza una vista parcial llamada 'train'. Recordando que las vistas parciales son fragmentos reutilizables de código HTML/ERB que se pueden incluir en otras vistas.
- **Título y botón para agregar cliente:** Se muestra un título centrado que indica “Editar Estación: Primer Grupo de Clientes” seguido del nombre del cliente (<%= @customer.nombre %>). También se muestra un botón con un enlace para agregar un nuevo cliente al grupo de clientes (new_customer_firstcg_path(@customer)).
- **Tabla de clientes existentes:** Si hay clientes en el grupo (@customer.firstcgs.any?), se muestra una tabla con los clientes existentes. El encabezado de la tabla contiene el nombre del cliente (@customer.nombre). Cada cliente válido se muestra en una lista (<li>) con

enlaces para editar y eliminar. Los enlaces están protegidos con confirmaciones (data: { confirm: ... }).

- **Selección de cliente existente:** Si hay clientes presentes (@customer.firstcgs.present?), se muestra una sección para seleccionar un cliente existente. Se utiliza un campo de selección (select_tag) con opciones generadas a partir de la colección de clientes (@customer.firstcgs), también se muestran enlaces para volver atrás, ver detalles del cliente y editar.

Definición de la vista new.html.erb

Una vez pulsado el botón de: Agregar Primer Cliente de Grupo de Clientes, se accede a una nueva vista, está ubicada en **app/views/firstcgs/new.html.erb** (anexo B.14) y su archivo es new.html.erb, desde acá se completan dos formularios por parte del usuario: información personal e información de contexto, adicional de puede completar el campo de número de ventas obtenidas.

Figura 65. app/views/firstcgs/new.html.erb (parte 1)


```
<%= render partial: 'shared/navigation_bar', object: current_user %>
<% provide(:title, t('customers_section')) %>
<%= render partial: 'shared/igheader', locals: {
    title: t("build"),
    secondaryTitle:  t("customers_section"),
    hideVpSelector: false,
    hideTooltipsButton: true
} %>

<style>
  /* Media query para pantallas medianas */
  @media only screen and (max-width: 1024px) {
    .text-right {
      font-size: 0.8em; /* Tamaño de fuente para pantallas medianas */
    }
  }
  /* Media query para pantallas pequeñas */
  @media only screen and (max-width: 600px) {
    .text-right {
      font-size: 0.4em; /* Tamaño de fuente para pantallas pequeñas */
    }
  }
```

Figura 66. app/views/firstcgs/new.html.erb (parte 2)

Las siguientes definiciones pertenecen a las figuras 65 y 66:

- **Estilos CSS:** (la parte entre <style> y </style>), Esta sección define algunos estilos con media queries para ajustar el tamaño de la fuente del texto alineado a la derecha dependiendo del tamaño de la pantalla.
- **@media only screen and (max-width: 1024px):** Esta media query aplica los estilos solo para pantallas con un ancho máximo de 1024px (pantallas medianas).
- **font-size: 0.8em;:** Reduce el tamaño de la fuente a 0.8 veces su tamaño original.
- **@media only screen and (max-width: 600px):** esta media query aplica los estilos solo para pantallas con un ancho máximo de 600px (pantallas pequeñas).
- **font-size: 0.4em**;: Reduce el tamaño de la fuente a 0.4 veces su tamaño original.
- **Formulario HTML (la parte entre y):** El código crea un contenedor con la clase "container". Dentro del contenedor, se crean dos secciones tituladas "Información Personal" y "Información de Contexto" usando la etiqueta <h2>.
- **Formulario Ruby (la parte entre <%= form_for [@customer, @firstcg] do |f| %> y <% end %>):** Este bloque crea un formulario Ruby usando el helper form_for. El

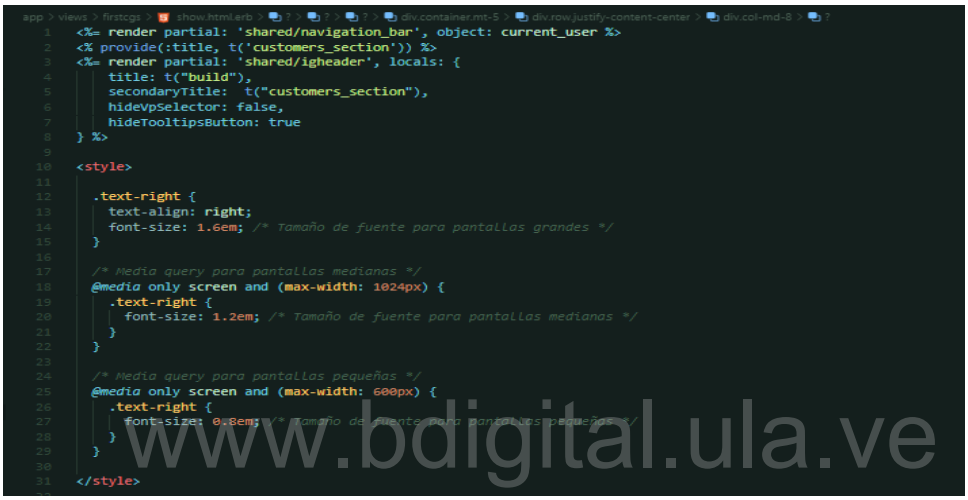
formulario se envía a la acción create del controlador que maneja los clientes (@customer). El bloque itera sobre un objeto @firstcg que presumiblemente contiene la información del cliente a crear o editar.

- **Manejo de errores (la parte entre <% if flash[:alert] %> y <% end %>):** Este bloque verifica si existe un mensaje de error en la variable flash[:alert]. Si existe un error, se crea una alerta de Bootstrap con la clase alert-danger para mostrar el mensaje.
- **Creación de campos del formulario (la parte dentro del ciclo <% f.do |f| %>):** Dentro del ciclo del formulario se generan dinámicamente varios campos para capturar información del cliente.
- **<%= f.label :Edad %>
:** Crea una etiqueta para el campo "Edad".
- **<%= f.number_field :age %>:** Crea un campo de texto para ingresar la edad. Solo acepta números.
- **Similar a los códigos anteriores se crean campos para:** Correo electrónico, Dirección, Teléfono, Primer nombre, Segundo nombre, Primer apellido, Segundo apellido, Profesión, De donde proviene, etc.
- **<%= f.text_area :context, rows: 5 %>:** Crea un campo de texto multi línea para ingresar el contexto de uso del cliente. El atributo rows: 5 especifica que tendrá 5 filas por defecto.
- **<%= f.range_field :degree_of_interestcg, in: 1..100, class:** "form-control-range" %>: Crea un campo tipo slider para indicar el grado de interés (de 1 a 100).
- **<%= label_tag :degree_of_interestcg, 'Poco interés', class: 'text-right' %>:** Crea etiquetas para el slider que indican el valor mínimo, medio y máximo. Se crea otro slider similar para el grado de experiencia.
- **<%= f.radio_button :pay_or_not_pay, 'paga' %> <%= f.label :pay_or_not_pay, 'Paga' %>:**crea botones de radio para seleccionar si el cliente paga o no paga.
- **<%= f.text_area :descriptioncg, rows: 5 %>:** Crea un campo de texto multi línea para ingresar una descripción del cliente.
- **<%= f.number_field :salesfcg, maxlength: 99999999999999999999999999 %>:** Crea un campo de texto para ingresar el número de ventas obtenidas. Permite ingresar un número muy grande.
- **<%= f.submit 'Agregar' %>:** Crea un botón de envío para el formulario con la etiqueta "Agregar".

Definición de la vista show.html.erb

La vista de show.html.erb muestra el resultado de haber completado los formularios de: información personal, información de contexto y el campo de número de ventas obtenidas. Su ubicación es **app/views/firstcgs/show.html.erb** (anexo B.15) y su archivo tiene de nombre **show.html.erb**.

Figura 67. app/views/firstcgs/show.html.erb (parte 1)



```
<%= render partial: 'shared/navigation_bar', object: current_user %>
<% provide(:title, t('customers_section')) %>
<%= render partial: 'shared/igheader', locals: {
    title: t("build"),
    secondaryTitle:  t("customers_section"),
    hideVpSelector: false,
    hideTooltipsButton: true
} %>

<style>

  .text-right {
    text-align: right;
    font-size: 1.6em; /* Tamaño de fuente para pantallas grandes */
  }

  /* Media query para pantallas medianas */
  @media only screen and (max-width: 1024px) {
    .text-right {
      font-size: 1.2em; /* Tamaño de fuente para pantallas medianas */
    }
  }

  /* Media query para pantallas pequeñas */
  @media only screen and (max-width: 600px) {
    .text-right {
      font-size: 0.8em; /* Tamaño de fuente para pantallas pequeñas */
    }
  }

</style>
```


Figura 68. app/views/firstcgs/show.html.erb (parte 2)

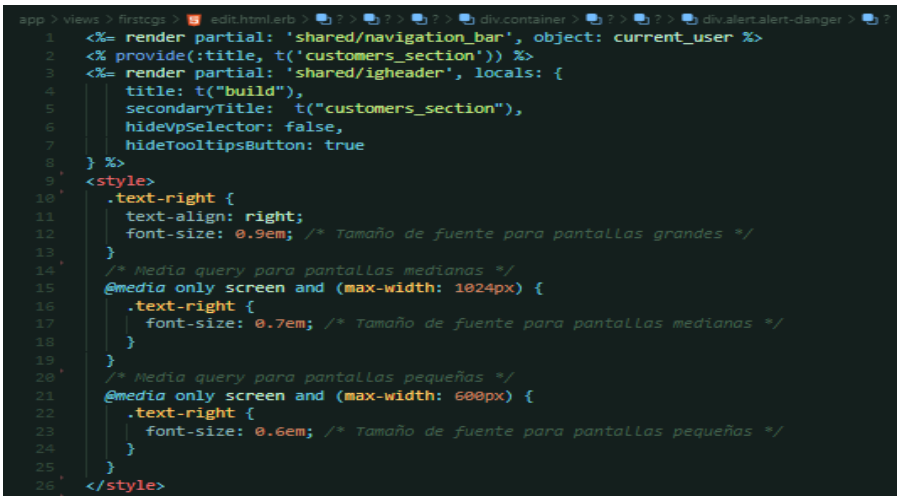
Las siguientes definiciones pertenecen a las figuras 67 y 68:

- **Estilos CSS (la parte entre <style> y </style>):** Esta sección define una clase llamada .text-right que se utiliza para dar estilo a los elementos de texto:
- **text-align: right;:** Alinea el texto a la derecha del contenedor.
- **font-size: 1.6em;:** Establece el tamaño de fuente predeterminado a 1.6 veces el tamaño de fuente base para pantallas grandes.
- **Consultas de medios se utilizan para ajustar el tamaño de fuente para diferentes tamaños de pantalla:** @media only screen and (max-width: 1024px): Esta consulta de medios aplica estilos solo para pantallas con un ancho máximo de 1024px (pantallas medianas). En este caso, reduce el tamaño de fuente a 1.2em.
- **@media only screen and (max-width: 600px):** Esta consulta de medios aplica estilos solo para pantallas con un ancho máximo de 600px (pantallas pequeñas). En este caso, reduce el tamaño de fuente a 0.8em.
- **Estructura HTML (la parte entre `` y):** El código crea un contenedor con la clase container y un margen superior (mt-5) utilizando clases de Bootstrap. Dentro del contenedor, hay una fila con la clase justify-content-center para centrar el contenido horizontalmente y una columna con la clase col-md-8 que ocupa 8 columnas en pantallas medianas (potencialmente utilizando el sistema de cuadrícula de Bootstrap). Se utiliza un elemento <h2> con la clase text-center y un margen inferior (mb-4) para mostrar el título "Información del Cliente" centrado y con algo de espacio debajo.
- **Tabla de información del cliente (la parte entre <table> y </table>):** se crea una tabla con las clases table y table-bordered para mostrar la información del cliente. El cuerpo de la tabla (<tbody>) contiene filas de tabla (<tr>). Cada fila contiene dos celdas de tabla (<td> y </td>):la primera celda (<th>) muestra la etiqueta del campo (por ejemplo, Edad - Edad), la segunda celda (<td>) muestra el valor correspondiente del objeto @firstcg (por ejemplo, <%= @firstcg.age %>). Este objeto contiene la información del cliente.
- **Enlaces y botones (la parte entre la tabla y el cierre):** Se crean dos enlaces utilizando el helper link_to: el primer enlace permite al usuario volver a la "Lista de clientes" con un mensaje de confirmación. El segundo enlace permite al usuario editar el "Primer grupo de clientes" actual con un mensaje de confirmación.

Definición de la vista edit.html.erb

La vista **edit.html.erb** ubicada en **app/views/firstcgs/edit.html.erb** (anexo B.16) permite editar los formularios existentes en la cuarta estación, también permite editar el número de ventas obtenidas.

Figura 69. app/views/firstcgs/edit.html.erb (parte 1)



```
<%= render partial: 'shared/navigation_bar', object: current_user %>
<% provide(:title, t('customers_section')) %>
<%= render partial: 'shared/igheader', locals: {
    title: t("build"),
    secondaryTitle:  t("customers_section"),
    hideVpSelector: false,
    hideTooltipsButton: true
} %>
<style>
  .text-right {
    text-align: right;
    font-size: 0.9em; /* Tamaño de fuente para pantallas grandes */
  }
  /* Media query para pantallas medianas */
  @media only screen and (max-width: 1024px) {
    .text-right {
      font-size: 0.7em; /* Tamaño de fuente para pantallas medianas */
    }
  }
  /* Media query para pantallas pequeñas */
  @media only screen and (max-width: 600px) {
    .text-right {
      font-size: 0.6em; /* Tamaño de fuente para pantallas pequeñas */
    }
  }
</style>
```


Figura 70. app/views/firstcgs/edit.html.erb (parte 2)

Las siguientes definiciones pertenecen a las figuras 69 y 70:

- **Estilos CSS:** esta sección define una clase llamada .text-right que se utiliza para alinear el texto a la derecha en la pantalla. Además, utiliza media queries para ajustar el tamaño de la fuente de este texto dependiendo del tamaño de la pantalla: para pantallas grandes (ancho máximo de 1024px) el tamaño de fuente es 0.9em, para pantallas medianas (ancho máximo de 600px) el tamaño de fuente es 0.7em. Para pantallas pequeñas (ancho máximo de 600px) el tamaño de fuente es 0.6em.
- **Código HTML:** Título: Se muestra un título " Información Personal " usando la etiqueta <h2>.
- **Formulario Ruby:** Se crea un formulario Ruby usando form_for para actualizar la información del cliente (@customer y @firstcg). Si existe un mensaje de error en la variable flash[:alert], se muestra una alerta roja con el mensaje usando Bootstrap.
- **Sección de información personal:** Se crea un grupo con la clase form-group que contiene varios campos para ingresar información personal del cliente: edad, correo electrónico, dirección, teléfono (campo de teléfono con máximo 13 caracteres), primer nombre, segundo nombre, primer apellido, segundo apellido, título.
- **Sección de información contextual:** Otro grupo con la clase form-group contiene campos para la información de contexto del cliente: profesión, de dónde proviene, contexto de uso del cliente (campo de texto multi línea con 5 filas por defecto), grado de interés (slider de 1 a 100), grado de experiencia (slider de 1 a 100 – exactamente como el grado de interés), paga o no paga (dos botones de radio para seleccionar si el cliente paga o no paga), descripción, número de ventas obtenidas.
- **Botones:** Se crea un grupo con la clase actions que contiene dos botones: un botón de envío (f.submit 'Actualizar') para guardar los cambios realizados en el formulario. Un enlace (link_to) que permite volver a la lista de clientes (customers_path). Este enlace tiene una confirmación para evitar que el usuario lo pulse accidentalmente.

5.4.3. Controlador de la Cuarta estación: Primer Grupo de Clientes

El controlador firstcgs_controller.rb se encuentra en **app/controllers/firstcgs_controller.rb,** maneja las solicitudes web relacionadas con "Primer Grupo de Clientes"

Figura 71. app/controllers/firstcgs_controller.rb

```ruby
class FirstcgsController < ApplicationController
  before_action :set_customer
  def index
  end
  def show
    @firstcg = Firstcg.find_by(id: params[:id])
    if @firstcg.nil?
      redirect_to firstcgs_path, alert: "Cliente no encontrado"
    end
  end
  def new
    @customer = Customer.find(params[:customer_id])
    @firstcg = @customer.firstcgs.new
  end
  def edit
    @firstcg = Firstcg.find(params[:id])
  end
  def create
    @customer = Customer.find(params[:customer_id])
    @firstcg = @customer.firstcgs.build(firstcg_params)
    if @firstcg.save
      redirect_to [@customer, @firstcg], notice: "Primer Cliente fue creado exitosamente."
    else
      render :new
    end
  end
  def update
    @firstcg = Firstcg.find(params[:id])
      if @firstcg.update(firstcg_params)
        flash[:notice] = "Primer Cliente actualizado correctamente"
        redirect_to customers_path
      else
        render 'edit'
      end
  end
  def destroy
    @firstcg = Firstcg.find(params[:id])
    @firstcg.destroy
    flash[:notice] = "Cliente eliminado correctamente"
    redirect_to customers_path
  end
  private

  def set_customer
    @customer = Customer.find(params[:customer_id])
  end
  def firstcg_params
    params.require(:firstcg).permit(:age, :emailcg, :addresscg, :phonecg, :namecg1, :namecg2, :surnamecg1, :surnamecg2, :professioncg, :where_it_comes_fromcg, :context, :degree_of_interestcg, :degree_of_expertisecg, :pay_or_not_pay, :descriptioncg, :salesfcg )
  end
end
```


Las siguientes definiciones pertenecen a la figura 71:

Acción previa:

- **before_action :set_customer:** Asegura que el método set_customer se ejecute antes de cualquier otra acción en este controlador. Este método probablemente recupera el objeto del cliente en función del parámetro customer_id.

Acciones CRUD para grupos de primeros clientes:

- **index**: Esta definición de método está vacía (def index; end). Podría ser de utilidad en el futuro.
- **show**: Este método encuentra un "Grupo de Primeros Clientes" específico en función del parámetro id en la URL (params[:id]). Si no se encuentra el grupo (@firstcg.nil?), redirige al usuario a la ruta firstcgs_path (potencialmente una ruta para enumerar todos los grupos de primeros clientes) con un mensaje de error que indica que no se encontró el cliente.

- **new:** Este método se utiliza para crear un nuevo "Grupo de Primeros Clientes". Encuentra el objeto del cliente utilizando el parámetro customer_id y luego crea un nuevo objeto Firstcg asociado con ese cliente utilizando @customer.firstcgs.new. Este nuevo objeto se asigna a la variable de instancia @firstcg.
- **edit:** Este método encuentra un "Grupo de Primeros Clientes" específico en función del parámetro id, similar al método show. Edita la información existente del grupo de clientes.
- **create:** este método se utiliza para guardar un nuevo "Grupo de Primeros Clientes". Encuentra el objeto del cliente utilizando el parámetro customer_id. Crea un nuevo objeto Firstcg asociado con el cliente utilizando @customer.firstcgs.build(firstcg_params). El método firstcg_params recupera y limpia los datos enviados por el usuario desde el formulario. Si el nuevo objeto Firstcg se guarda correctamente utilizando @firstcg.save, redirige al usuario a la página de visualización del grupo de clientes recién creado con un mensaje de éxito. Si falla el guardado, vuelve a renderizar la plantilla new, para mostrar cualquier error de validación encontrado al guardar el nuevo grupo de clientes.
- **update:** Este método se utiliza para actualizar un "Grupo de Primeros Clientes" existente. Encuentra el grupo de clientes específico en función del parámetro id. Intenta actualizar el objeto Firstcg existente utilizando @firstcg.update(firstcg_params). Similar a create, firstcg_params proporciona una entrada de usuario desinfectada. Si la actualización es exitosa, establece un mensaje de aviso flash que indica una actualización exitosa y redirige al usuario a la página de lista de clientes (customers_path). Si la actualización falla, vuelve a renderizar la plantilla edit, probablemente para mostrar cualquier error de validación que haya ocurrido durante el intento de actualización.
- **destroy:** Este método elimina un "Grupo de Primeros Clientes". Encuentra el grupo de clientes específico en función del parámetro id. Destruye el grupo de clientes encontrado utilizando @firstcg.destroy. Después de la eliminación, establece un mensaje de aviso flash que indica una eliminación exitosa y redirige al usuario a la página de lista de clientes (customers_path).

Métodos privados:

- **set_customer:** Como se mencionó anteriormente, este método recupera el objeto del cliente en función del parámetro customer_id y lo asigna a la variable de instancia

@customer. Este objeto de cliente se utiliza en los métodos del controlador para gestionar los grupos de primeros clientes asociados con ese cliente.

- **firstcg_params:** Este método define una lista blanca de parámetros que se pueden enviar a través del formulario para crear o actualizar un "Primer Grupo de Clientes" Utiliza params.require(:firstcg) para acceder al hash anidado que contiene los datos del formulario relacionados con el grupo de primeros clientes y luego permite atributos específicos como :age, :emailcg y otros, ayuda a evitar que se envíen datos no autorizados a través del formulario.

Base de datos (db migrate) del modelo firstcg.rb

Esta base de datos se generó al momento de crear el modelo de la cuarta estación firstcg.rb, cada uno con los atributos necesarios y requeridos por el documento de requisitos del producto. La tabla firstcg.rb contiene múltiples atributos necesarios para el despliegue de la respectiva estación.

Figura 72. db/migrate/20231014230244_create_firstclients.rb

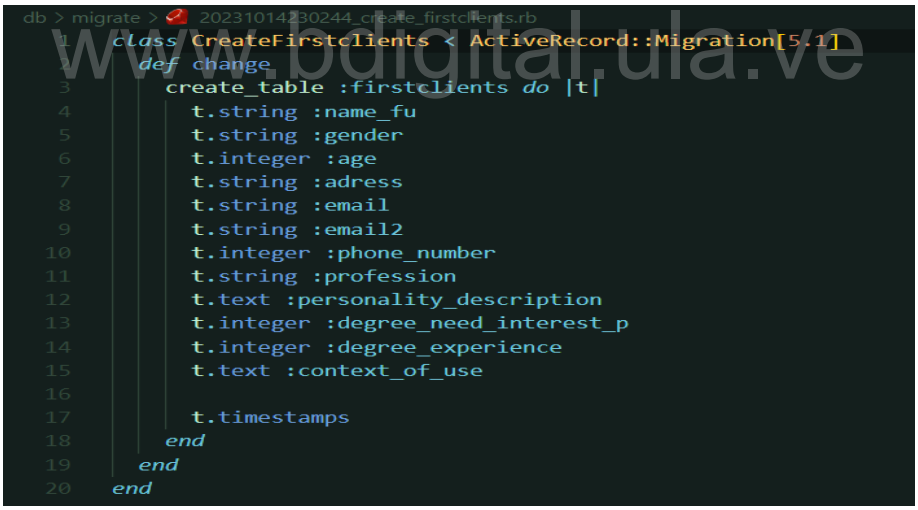


Figura 73. db/migrate/20231014231447_add_customer_ref_to_firstclients.rb

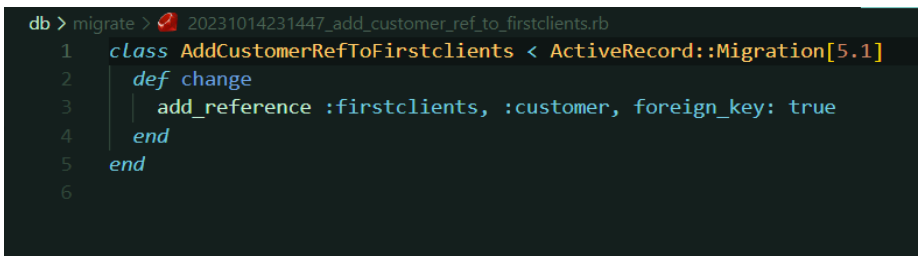


Figura 74. db/migrate/20231105211926_add_paynotpayfc_to_firstclients.rb

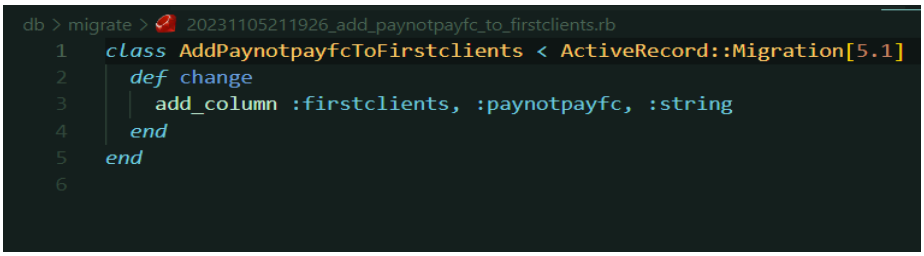


Figura 75. db/migrate/20240217015305_add_finalpay_to_firstclients.rb

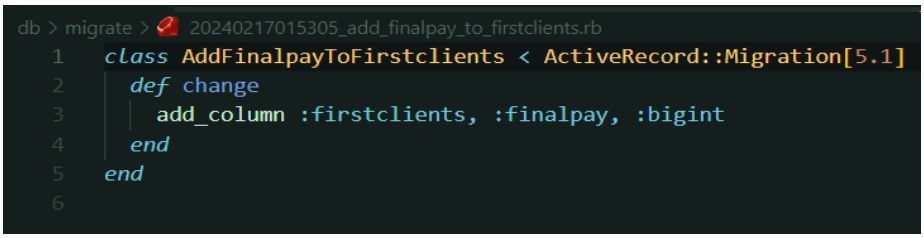


Tabla resumen de la Cuarta estación: Primer Grupo de Clientes

Tabla 6. Tabla resumen de la Cuarta estación: Primer Grupo de Clientes

	Ubicación
Modelo	app/models/firstcg.rb
Vista	app/views/customers/firstcg.html.erb, app/views/firstcgs/new.html.erb, app/views/firstcgs/show.html.erb, app/views/firstcgs/edit.html.erb
Controlador	app/controllers/firstcgs_controller.rb

Análisis: esta estación funciona de manera correcta, permitiendo al usuario crear varios clientes, con el botón "+ Agregar Primer Cliente de Grupo de Clientes", para completar los dos formularios, una vez creados, poder seleccionar uno con facilidad y poder ver los detalles del cliente o editarlo. Como en todas las estaciones, el CRUD hace presencia.

5. 5. Quinta estación: Primeros Adoptantes

Esta estación contiene un conjunto de bloques de información que representaran, el estatus en cantidad de clientes que han adoptado y comprado el producto, así como representar el estatus de marketing.

Contiene una cajetilla con información con: número de clientes que llegan (por día, mes y año), número de clientes que se retiran (por día, mes y año), número total de clientes que han llegado y se han retirado en el tiempo que se ha desarrollado el emprendimiento del producto y un calibrador que representa el estatus de marketing digital.

5.5.1. Modelo de la Quinta estación: Primeros Adoptantes

El modelo de la quinta estación está ubicado **app/models/earlyadopter.rb** y su archivo es **earlyadopter.rb**

Figura 76. app/models/earlyadopter.rb

Las siguientes definiciones pertenecen a la figura 76:

Herencia:

- **Earlyadopter < ApplicationRecord:** la clase Earlyadopter hereda de la clase ApplicationRecord. Esto significa que Earlyadopter obtiene automáticamente todas las funcionalidades básicas de un modelo de Rails, como validaciones, relaciones con otras tablas y persistencia de datos en la base de datos.

Relación con la clase Customer:

- **belongs_to :customer:** define una relación del tipo "pertenece a" entre Earlyadopter y customer. Esto significa que un Earlyadopter está asociado con un cliente en particular.

Callbacks:

- **before_validation :set_ntotal, :set_date:** Esta línea define callbacks que se ejecutan antes de la validación de un objeto Earlyadopter. Los callbacks son métodos especiales en Rails que se ejecutan en momentos específicos del ciclo de vida de un objeto del modelo. En este caso, se llaman a los métodos set_ntotal y set_date antes de que se guarde el objeto Earlyadopter en la base de datos.

Métodos callback:

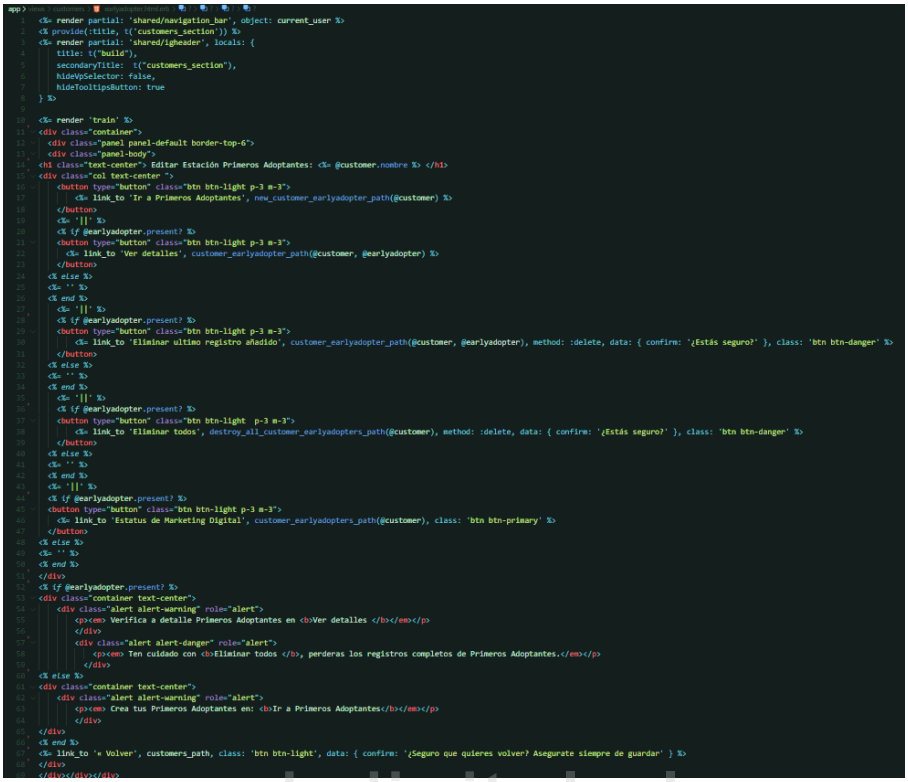
- **set_ntotal:** este método calcula la diferencia entre el número de clientes que entran (nclientsin) y el número de clientes que salen (nclientsout), self.ntotal = (nclientsin || 0) - (nclientsout || 0):
- **self.ntotal:** Esta parte asigna el resultado del cálculo a la propiedad ntotal del objeto Earlyadopter actual, nclientsin || 0: Si el valor de nclientsin no está definido o es nulo (nil), se usa 0 en su lugar, nclientsout || 0: Similar al anterior, si nclientsout no está definido o es nulo, se usa 0 en su lugar.
- **set_date:** Este método simplemente asigna la fecha actual (Date.today) a la propiedad datein del objeto Earlyadopter.

5.5.2. Vista de la Quinta estación: Primeros Adoptantes

Definición de la vista earlyadopter. Html.erb

Se encuentra **app/views/customers/earlyadopter.html.erb** (anexo B.17), el archivo **earlyadopter.html.erb** lleva a la vista principal de esta estación, desde acá se podrán crear los primeros adoptantes, con su cajetilla de información necesaria para el despliegue de los clientes que entran, los clientes que salen y los clientes totales, con una buena organización, gracias a que se lleva un registro en fecha de cuando ocurre esto. Apoyados por un estatus de marketing digital, para que el emprendedor tenga buenas nociones de lo que está sucediendo.

Figura 77. app/views/customers/earlyadopter.html.erb



Las siguientes definiciones pertenecen a la figura 77:

- **Inclusión de un parcial:** la primera línea <%= render 'train' %> Incluye un parcial llamado "train". contiene código HTML reutilizable para el diseño de la página.
- **Contenedor principal:** El código está envuelto por un contenedor principal con la clase container. Este contenedor define el ancho y el margen de la sección principal de la página.
- **Panel de edición:** Dentro del contenedor principal, hay un panel con la clase panel panel-default border-top-6. Este panel contiene el formulario y la información relacionada con la edición.
- **Título:** Un título <h2> con la clase text-center se muestra en la parte superior del panel. El título indica que se trata de la edición de la "Estación Primeros Adoptantes" para un cliente específico, cuyo nombre se obtiene de la variable @customer.nombre.
- **Botones de acción:** Una sección con la clase col text-center contiene varios botones que permiten realizar distintas acciones:

- el primer botón, con texto "Ir a Primeros Adoptantes", utiliza link_to para crear un enlace a la ruta new_customer_earlyadopter_path(@customer). Esto permite crear un nuevo registro de "Primeros Adoptantes" asociado al cliente (SdeC) actual.
- El segundo botón condicional (if @earlyadopter.present?) aparece solo si existe un registro de "Earlyadopter" asociado al cliente (@earlyadopter). Este botón utiliza link_to para crear un enlace a la ruta customer_earlyadopter_path(@customer, @earlyadopter), permitiendo ver los detalles del registro existente.
- El tercer botón condicional (if @earlyadopter.present?) también depende de la existencia de un registro. En este caso, el botón permite eliminar el último registro añadido. Utiliza link_to con el método :delete para enviar una petición DELETE a la ruta customer_earlyadopter_path(@customer, @earlyadopter). Además, incluye un mensaje de confirmación (data: { confirm: '¿Estás seguro?' }) para evitar eliminaciones accidentales.
- El cuarto botón condicional (if @earlyadopter.present?) permite eliminar todos los registros de "Primeros Adoptantes" asociados al cliente. Similar al anterior, utiliza link_to con el método :delete para enviar una petición DELETE a la ruta destroy_all_customer_earlyadopters_path(@customer). También incluye un mensaje de confirmación para advertir del borrado masivo.
- El quinto botón condicional (if @earlyadopter.present?) muestra un enlace a la ruta customer_earlyadopters_path(@customer). El texto del botón es "Estatus de Marketing Digital" y utiliza la clase btn-primary para darle un estilo distintivo, este enlace lleve a una sección que muestra el estado del marketing digital relacionado con los primeros adoptantes del cliente.

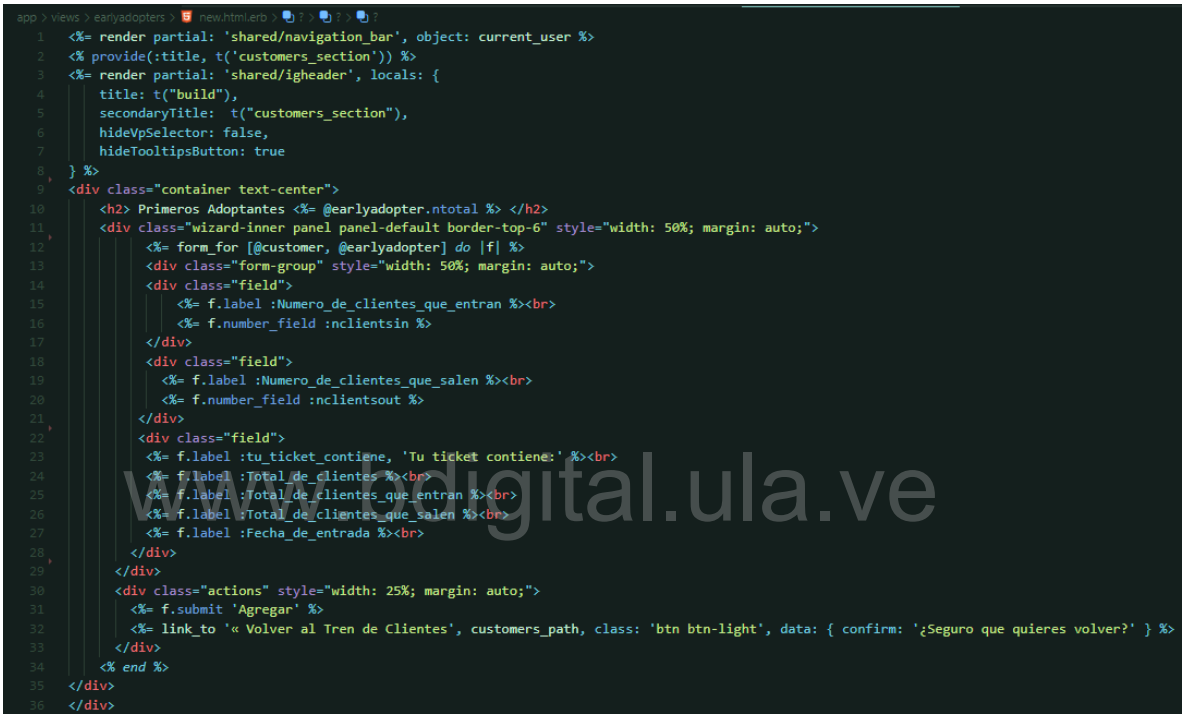
- **Mensajes de alerta condicionales:** Dependiendo de si existe un registro de "Earlyadopter" (@earlyadopter.present?), se muestran mensajes de alerta con información o advertencias: si existe un registro, se muestran dos alertas: una amarilla que indica que se pueden ver los detalles del registro y otra roja que advierte sobre el cuidado al usar la opción de eliminar todos los registros. Si no existe un registro, se muestra una única alerta amarilla que indica cómo crear nuevos registros de "Primeros Adoptantes".
- **Botón de retroceso:** Un enlace con texto "« Volver" utiliza link_to para crear un enlace a la ruta customers_path. Este botón permite volver a la lista de clientes. Incluye un mensaje

de confirmación para evitar que el usuario abandone accidentalmente la edición sin guardar.

Definición de la vista new.html.erb

Esta es la vista para crear un nuevo ticket, su archivo está ubicado en **app/views/earlyadopters/new.html.erb** (anexo B.18) y recibe el nombre de **new.html.erb**

Figura 78. app/views/earlyadopters/new.html.erb



```
app > views > earlyadopters > new.html.erb
<%= render partial: 'shared/navigation_bar', object: current_user %>
<% provide(:title, t('customers_section')) %>
<%= render partial: 'shared/igheader', locals: {
    title: t("build"),
    secondaryTitle:  t("customers_section"),
    hideVpSelector: false,
    hideTooltipsButton: true
} %>
<div class="container text-center">
    <h2> Primeros Adoptantes <%= @earlyadopter.ntotal %> </h2>
    <div class="wizard-inner panel panel-default border-top-6" style="width: 50%; margin: auto;">
        <%= form_for [@customer, @earlyadopter] do |f| %>
        <div class="form-group" style="width: 50%; margin: auto;">
        <div class="field">
            <%= f.label :Numero_de_clientes_que_entran %><br>
            <%= f.number_field :nclientsin %>
        </div>
        <div class="field">
          <%= f.label :Numero_de_clientes_que_salen %><br>
          <%= f.number_field :nclientsout %>
        </div>
        <div class="field">
          <%= f.label :tu_ticket_contiene, 'Tu ticket contiene:' %><br>
          <%= f.label :Total_de_clientes %><br>
          <%= f.label :Total_de_clientes_que_entran %><br>
          <%= f.label :Total_de_clientes_que_salen %><br>
          <%= f.label :Fecha_de_entrada %><br>
        </div>
      </div>
      <div class="actions" style="width: 25%; margin: auto;">
        <%= f.submit 'Agregar' %>
        <%= link_to '« Volver al Tren de Clientes', customers_path, class: 'btn btn-light', data: { confirm: '¿Seguro que quieres volver?' } %>
      </div>
    <% end %>
</div>
</div>
```


Las siguientes definiciones pertenecen a la figuras 78:

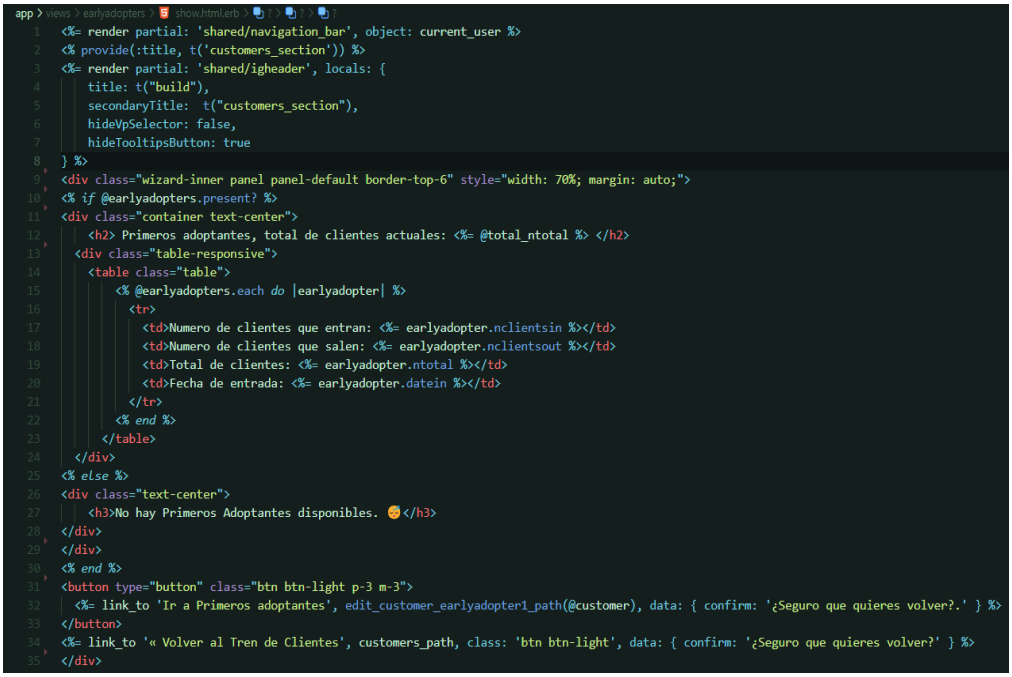
- **Contenedor y Título:** el código está envuelto por un contenedor con la clase container text-center. Esto centra el contenido del formulario horizontalmente. Dentro del contenedor, se muestra un título <h2> que indica "Primeros Adoptantes" seguido del valor de la propiedad @earlyadopter.ntotal. Este valor representa el total de clientes (número de clientes que entran menos el número de clientes que salen).
- **Formulario:** un formulario se crea utilizando form_for para crear la información del registro @earlyadopter asociado al cliente @customer. El formulario está dentro de un contenedor con la clase wizard-inner panel panel-default border-top-6, estándar para la sección del TOC. La propiedad style="width: 50%; margin: auto;" del contenedor lo posiciona en el centro de la página con un ancho del 50%.

- **Campos del formulario:** dos campos de tipo number_field se utilizan para ingresar la cantidad de clientes que entran (nclientsin) y la cantidad de clientes que salen (nclientsout). Cada campo tiene una etiqueta asociada creada con f.label.
- **Etiquetas informativas:** un grupo de etiquetas creadas con f.label proporciona información al usuario sobre el contenido del ticket (la información que se guardará en el formulario). Estas etiquetas no están asociadas a ningún campo de entrada de datos.
- **Botones de acción:** el formulario contiene dos botones dentro de un contenedor con la clase actions: un botón f.submit 'Agregar' permite guardar los cambios realizados en el formulario. El texto del botón es "Agregar". Un enlace creado con link_to permite volver a la lista de clientes (customers_path). El texto del enlace es "« Volver al Tren de Clientes" e incluye un mensaje de confirmación para evitar que el usuario abandone accidentalmente el formulario sin guardar.

Definición de la vista show.html.erb

El archivo show.html.erb se encuentra en **app/views/earlyadopters/show.html.erb** (anexo B.19), es la vista para mostrar el contenido una vez que se haya creado con **new.html.erb**.

Figura 79. app/views/earlyadopters/show.html.erb



```
app > views > earlyadopters > show.html.erb
<%= render partial: 'shared/navigation_bar', object: current_user %>
<% provide(:title, t('customers_section')) %>
<%= render partial: 'shared/igheader', locals: {
    title: t("build"),
    secondaryTitle:  t("customers_section"),
    hideVpSelector: false,
    hideTooltipsButton: true
} %>
<div class="wizard-inner panel panel-default border-top-6" style="width: 70%; margin: auto;">
<% if @earlyadopters.present? %>
<div class="container text-center">
    <h2> Primeros adoptantes, total de clientes actuales: <%= @total_ntotal %> </h2>
  <div class="table-responsive">
    <table class="table">
        <% @earlyadopters.each do |earlyadopter| %>
          <tr>
            <td>Numero de clientes que entran: <%= earlyadopter.nclientsin %></td>
            <td>Numero de clientes que salen: <%= earlyadopter.nclientsout %></td>
            <td>Total de clientes: <%= earlyadopter.ntotal %></td>
            <td>Fecha de entrada: <%= earlyadopter.datein %></td>
          </tr>
        <% end %>
      </table>
  </div>
<% else %>
<div class="text-center">
    <h3>No hay Primeros Adoptantes disponibles. 😴</h3>
</div>
</div>
<% end %>
<button type="button" class="btn btn-light p-3 m-3">
  <%= link_to 'Ir a Primeros adoptantes', edit_customer_earlyadopter1_path(@customer), data: { confirm: '¿Seguro que quieres volver?.' } %>
</button>
<%= link_to '« Volver al Tren de Clientes', customers_path, class: 'btn btn-light', data: { confirm: '¿Seguro que quieres volver?' } %>
</div>
```

Las siguientes definiciones pertenecen a la figura 79:

- **Contenedor principal:** el código está envuelto por un contenedor con la clase wizard-inner panel panel-default border-top-6 que define el estilo visual de la sección. La propiedad style="width: 70%; margin: auto;" lo posiciona en el centro de la página con un ancho del 70%.
- **Sección condicional para mostrar registros:** un bloque condicional <% if @earlyadopters.present? %> verifica si hay registros de "Early Adopters" asociados al cliente (@earlyadopters). Si hay registros (@earlyadopters.present? es verdadero). Un contenedor con la clase container text-center centra el contenido siguiente: un título <h2> muestra "Primeros adoptantes, total de clientes actuales, seguido del valor de la variable @total_ntotal. Este valor representa el total combinado de clientes de todos los registros de "Early Adopters". Un contenedor con la clase table-responsive permite que la tabla se ajuste responsivamente en diferentes tamaños de pantalla. Dentro del contenedor table-responsive, se crea una tabla HTML con la clase table. Un ciclo <% @earlyadopters.each do |earlyadopter| %> itera sobre cada registro de "Early Adopters" (@earlyadopters) almacenado en una variable temporal earlyadopter.
- **Para cada registro, se crea una fila <tr> en la tabla. Dentro de la fila, se crean celdas <td> que muestran los valores de las propiedades del registro earlyadopter:** número de clientes que entran (nclientsin)**,** número de clientes que salen (nclientsout)**,** total de clientes (ntotal), fecha de entrada (datein).
- **Si no hay registros (@earlyadopters.present? es falso):** un contenedor con la clase text-center centra el siguiente elemento. Se muestra un mensaje <h3> que indica "No hay Primeros Adoptantes disponibles. "
- **Botones de acción:** fuera del bloque condicional, se muestran dos botones: el primer botón utiliza link_to para crear un enlace a la ruta edit_customer_earlyadopter1_path(@customer). El texto del botón es "Ir a Primeros Adoptantes" e incluye un mensaje de confirmación para evitar que el usuario abandone accidentalmente la página. El segundo botón, similar al anterior, utiliza link_to para crear un enlace a la ruta customers_path que permite volver a la lista de clientes (SdeC). El texto del botón es "« Volver al Tren de Clientes" e incluye un mensaje de confirmación.

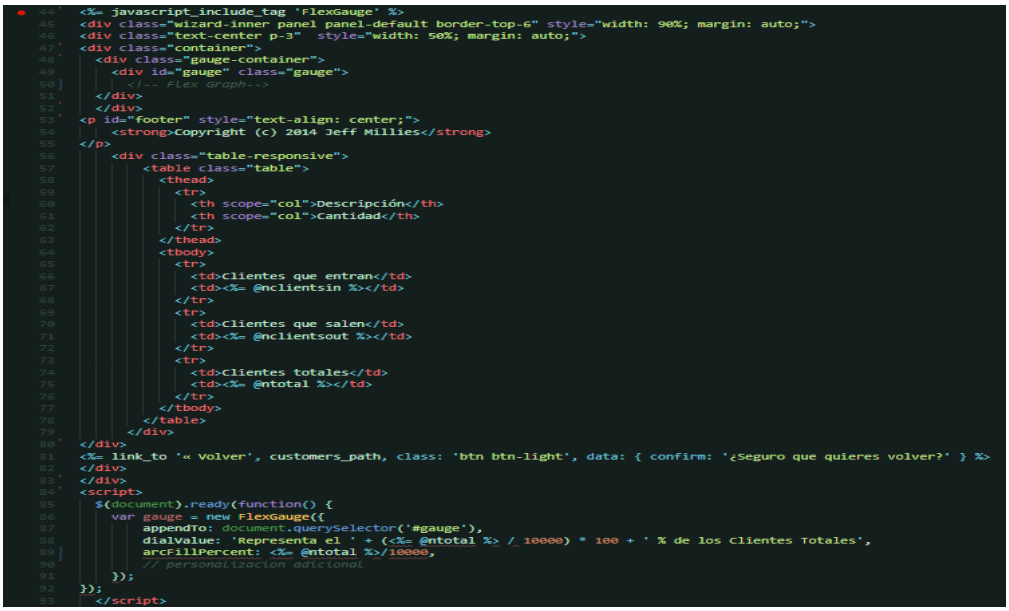
Definición de la vista index.html.erb

Esta vista contiene el apartado de estatus del marketing, contiene un gráfico que muestra los clientes totales entre todos los registros que se vayan creando, esta creado de forma simple pero precisa, ideal para que el emprendedor entienda rápidamente la situación de los clientes que entran y los clientes que salen. El archivo **index.html.erb** se encuentra en **app/views/earlyadopters/index.html.erb** (Anexos B.20).

Figura 80. app/views/earlyadopters/index.html.erb (parte 1)

```
<%= render partial: 'shared/navigation_bar', object: current_user %>
<% provide(:title, t('customers_section')) %>
<%= render partial: 'shared/igheader', locals: {
    title: t("build"),
    secondaryTitle:  t("customers_section"),
    hidevpSelector: false,
    hideTooltipsButton: true
} %>
<style>
  .gauge-container {
    display: flex;
    justify-content: center;
    align-items: center;
    height: 40vh; /* Ajusta esto según tus necesidades */
}
  /* Estilos por defecto para la gráfica */
  #gauge {
      width: 100%;
      height: auto;
  }
  /* Estilos específicos para pantallas con una anchura máxima de 600px */
  @media only screen and (max-width: 600px) {
      #gauge {
          width: 100%;
          height: auto;
      }
  }

  /* Estilos específicos para pantallas con una anchura mínima de 601px y una anchura máxima de 1024px */
  @media only screen and (min-width: 601px) and (max-width: 1024px) {
      #gauge {
          width: 75%;
          height: auto;
      }
  }
  /* Estilos específicos para pantallas con una anchura mínima de 1025px */
  @media only screen and (min-width: 1025px) {
      #gauge {
          width: 50%;
          height: auto;
      }
  }
</style>
```


Figura 81. app/views/earlyadopters/index.html.erb (parte 2)



```
<%= javascript_include_tag 'FlexGauge' %>
<div class="wizard-inner panel panel-default border-top-6" style="width: 90%; margin: auto;">
<div class="text-center p-3"  style="width: 50%; margin: auto;">
<div class="container">
  <div class="gauge-container">
    <div id="gauge" class="gauge">
      <!-- Flex Graph-->
  </div>
  </div>
<p id="footer" style="text-align: center;">
    <strong>Copyright (c) 2014 Jeff Millies</strong>
</p>
    <div class="table-responsive">
        <table class="table">
          <thead>
            <tr>
              <th scope="col">Descripción</th>
              <th scope="col">Cantidad</th>
            </tr>
          </thead>
          <tbody>
            <tr>
              <td>Clientes que entran</td>
              <td><%= @nclientsin %></td>
            </tr>
            <tr>
              <td>Clientes que salen</td>
              <td><%= @nclientsout %></td>
            </tr>
            <tr>
              <td>Clientes totales</td>
              <td><%= @ntotal %></td>
            </tr>
          </tbody>
        </table>
    </div>
</div>
<%= link_to '« Volver', customers_path, class: 'btn btn-light', data: { confirm: '¿Seguro que quieres volver?' } %>
</div>
</div>
<script>
  $(document).ready(function() {
    var gauge = new FlexGauge({
        appendTo: document.querySelector('#gauge'),
        dialValue: 'Representa el ' + (<%= @ntotal %> / 10000) * 100 + ' % de los Clientes Totales',
        arcFillPercent: <%= @ntotal %>/10000,
        // personalizacion adicional
    });
});
  </script>
```

Las siguientes definiciones pertenecen a las figuras 80 y 81:

Estilos CSS:

- **la sección style define estilos para el dashboard**: el contenedor principal .gauge-container utiliza flexbox para centrar su contenido horizontal y verticalmente. También se define su altura (height: 40vh;).
- **Estilos por defecto para la gráfica con id #gauge:** ocupa el ancho completo (width: 100%) y el alto se ajusta automáticamente (height: auto;).
- **Estilos responsivos para diferentes tamaños de pantalla:** pantallas con ancho máximo de 600px: mantienen el ancho y alto por defecto para la gráfica. Pantallas con ancho entre 601px y 1024px: la gráfica ocupa el 75% del ancho. Pantallas con ancho mínimo de 1025px: la gráfica ocupa el 50% del ancho.

El código también incluye un pie de página con **copyright** utilizando estilos en línea.

- **Librería Javascript y Contenedor principal:** la línea <%= javascript_include_tag 'FlexGauge' %> incluye una librería Javascript llamada FlexGauge, la cual se encarga de renderizar el gráfico circular, ubicado en **app/assets/javascripts/FlexGauge.js**. Un contenedor principal con la clase wizard-inner panel panel-default border-top-6 define el estilo visual de la sección y tiene un ancho del 90% con margen automático.
- **Contenedor del gráfico y tabla:** dentro del contenedor principal, hay otro contenedor con la clase text-center p-3 centrado horizontalmente y con un ancho del 50%. Este contenedor contiene un elemento div con la clase container que probablemente agrupa el gráfico y la tabla. Un contenedor con la clase gauge-container definido anteriormente centra el siguiente elemento. Dentro del contenedor gauge-container se encuentra un elemento div con el id "gauge" y la clase "gauge". Este elemento es donde se renderizará el gráfico circular utilizando la librería FlexGauge. Debajo del gráfico, se muestra un pie de página con el copyright del autor.
- **Copyright (c) 2014 Jeff Millies.** Un contenedor con la clase table-responsive permite que la tabla se ajuste responsivamente en diferentes tamaños de pantalla. Dentro del contenedor table-responsive, se crea una tabla HTML con la clase table, la tabla tiene un encabezado (<thead>) con dos columnas: "Descripción" y "Cantidad".

- **El cuerpo de la tabla (<tbody>) muestra tres filas:** clientes que entran (@nclientsin)**,** clientes que salen (@nclientsout) ,clientes totales (@ntotal)
- **Botón de Volver**: fuera del contenedor con la tabla, se muestra un botón creado con link_to. El botón permite volver a la lista de clientes (customers_path). El texto del botón es "« Volver" e incluye un mensaje de confirmación.

Javascript para el gráfico:

- El código Javascript se ejecuta cuando el documento esté listo ($(document).ready (function() { ... })). Se crea una nueva instancia del objeto FlexGauge y se configura:

- **appendTo:** indica el elemento del DOM donde se renderizará el gráfico (en este caso, el elemento con id "gauge").
- **dialValue:** define el texto que se muestra en el centro del gráfico. Utiliza el valor de @ntotal para calcular un porcentaje del total de clientes (configurado para un máximo de 10.000, la cantidad se podrá ajustar si es necesario).
- **arcFillPercent:** define el porcentaje que llena el arco del gráfico circular. Utiliza el valor de @ntotal para calcular el porcentaje.

5.5.3. Controlador de la Quinta estación: Primeros Adoptantes

El controlador de la estación Primeros Adoptantes está ubicado en **app/controllers/earlyadopters_controller.rb**, cuyo archivo tiene de nombre **earlyadopters_controller.rb**, gestiona todas las acciones correspondientes para el modelo, llevando al usuario a la vista correspondiente según la acción seleccionada. También gestionando operaciones como número de clientes que entran menos el número de clientes que salen.

Figura 82. app/controllers/earlyadopters_controller.rb

```ruby
app > controllers > earlyadopters_controller.rb
class EarlyadoptersController < ApplicationController
  before_action :set_customer
  def index
    @customer = Customer.find(params[:customer_id])  #ok
    @earlyadopters = @customer.earlyadopters  # cliente actual
    # Calculo de metricas
    @nclientsin = @earlyadopters.sum(:nclientsin)
    @nclientsout = @earlyadopters.sum(:nclientsout)
    @ntotal = @nclientsin - @nclientsout
  end
  def show
    @customer = Customer.find(params[:customer_id])
    @latest_earlyadopter = @customer.earlyadopters.order(created_at: :desc).first
    @earlyadopter = @customer.earlyadopters.find(params[:id])
    @earlyadopters = @customer.earlyadopters

    # Calcular valores solo para el cliente SdeC actual
    @nclientsin = @earlyadopter.nclientsin
    @nclientsout = @earlyadopter.nclientsout
    @ntotal = @nclientsin - @nclientsout

    if @earlyadopter.nil?
      redirect_to customer_earlyadopters_path(@customer), alert: "Cliente no encontrado"
    end
  end
  def new
    @customer = Customer.find(params[:customer_id])
    @earlyadopter = @customer.earlyadopters.new
  end
  def edit
    @earlyadopter = Earlyadopter.find(params[:id])
  end
  def create
    @customer = Customer.find(params[:customer_id])
    @earlyadopter = @customer.earlyadopters.build(earlyadopter_params)
    if @earlyadopter.save
      redirect_to [@customer, @earlyadopter], notice: 'Primeros Adoptantes! .'
    else
      render :new
    end
  end
  def update
    @earlyadopter = Earlyadopter.find(params[:id])
    if @earlyadopter.update(earlyadopter_params)
      flash[:notice] = "Primeros adoptantes actualizados correctamente"
      redirect_to customers_path
    else
      render 'edit'
    end
  end
  def destroy
    @earlyadopter = Earlyadopter.order(created_at: :desc).first
    @earlyadopter.destroy
    redirect_to customers_path, notice: 'Ultimo registro de Primeros Adoptantes eliminado correctamente.'
  end
  def destroy_all
    @customer = Customer.find(params[:customer_id])
    @customer.earlyadopters.destroy_all
    redirect_to customers_path, notice: 'Todos los Primeros Adoptantes han sido eliminados exitosamente.'
  end
  private
  def set_customer
    @customer = Customer.find(params[:customer_id])
  end
  def earlyadopter
    @customer = Customer.find(params[:customer_id])
    @earlyadopter = @customer.earlyadopters.find(params[:id])
  end
  def earlyadopter_params
    params.require(:earlyadopter).permit(:nclientsin, :nclientsout, :ntotal, :datein, :dateout, :oaux )
  end
end
```


Las siguientes definiciones pertenecen a la figura 82:

Filtro before_action:

La línea before_action :set_customer: indica que el método set_customer se ejecutará antes de cada acción del controlador. Este método se encarga de buscar el cliente actual basado en el customer_id pasado en los parámetros de la solicitud.

Acciones del controlador:

- **index:** esta acción se activa cuando se accede a la ruta del índice de Early Adopters para un cliente específico. Busca el cliente actual usando @customer = Customer.find(params[:customer_id]). Obtiene todos los Early Adopters asociados al cliente actual usando @earlyadopters = @customer.earlyadopters. Calcula las métricas totales (nclientsin, nclientsout, ntotal) para los Early Adopters del cliente actual.
- **show:** esta acción se activa cuando se accede a la vista de detalles de un Early Adopter específico. Busca el cliente actual usando @customer = Customer.find(params[:customer_id]). Busca el Early Adopter más reciente del cliente actual usando @latest_earlyadopter = @customer.earlyadopters.order(created_at: :desc).first. Busca el Early Adopter específico basado en el id del parámetro (@earlyadopter = @customer.earlyadopters.find(params[:id])). Obtiene todos los Early Adopters del cliente actual usando @earlyadopters = @customer.earlyadopters. Calcula las métricas (nclientsin, nclientsout, ntotal) solo para el Early Adopter actual usando sus atributos. Si el Early Adopter no se encuentra (@earlyadopter.nil?), redirige a la página del índice de Early Adopters con un mensaje de alerta.
- **new, edit, create, update:** estas acciones se utilizan para crear, editar y actualizar registros de Early Adopters. Obtienen el cliente actual y crean/encuentran un nuevo/existente Early Adopter asociado a ese cliente. Permiten guardar o actualizar el Early Adopter y redirigen a la página correspondiente según el éxito o fracaso de la operación.
- **destroy:** elimina el registro del Early Adopter más reciente del cliente actual. Redirige a la página del índice de Early Adopters con un mensaje de confirmación.
- **destroy_all:** elimina todos los registros de Early Adopters asociados al cliente actual. Redirige a la página del índice de Early Adopters con un mensaje de confirmación.

Métodos privados:

- **set_customer:** busca y establece el cliente actual basado en el customer_id del parámetro.
- **earlyadopter_params:** define los parámetros permitidos al crear o actualizar un Early Adopter (nclientsin, nclientsout, ntotal, datein, dateout, oaux).

Base de datos (db migrate) del modelo earlyadopter.rb

Esta base de datos se generó al momento de crear el modelo de la quinta estación earlyadopter.rb, cada uno con los atributos necesarios y requeridos por el documento de requisitos del producto. La tabla earlyadopters contiene múltiples atributos necesarios para el despliegue de la respectiva estación.

Figura 83. db/migrate/20231204195959_create_earlyadopters.rb

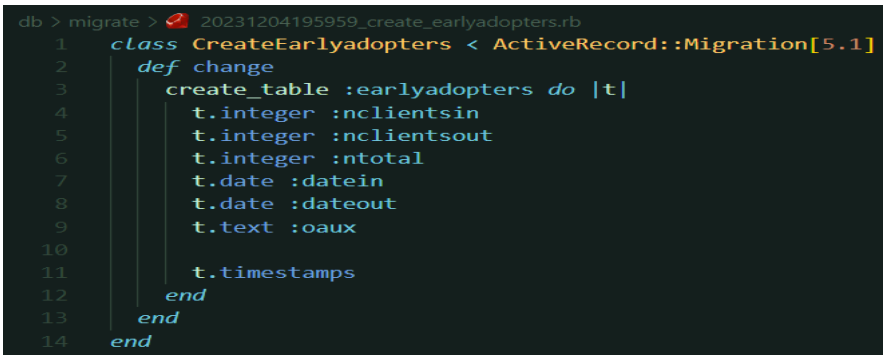


Figura 84. db/migrate/20231204200953_add_customer_ref_to_earlyadopters.rb

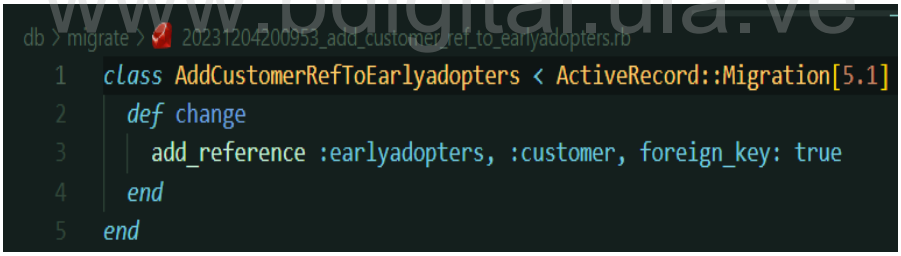


Tabla resumen de la Quinta estación: Primeros Adoptantes

Tabla 7. Tabla resumen de la Quinta estación: Primeros Adoptantes

	Ubicación
Modelo	app/models/earlyadopter.rb
Vista	app/views/customers/earlyadopter.html.erb, app/views/earlyadopters/new.html.erb, app/views/earlyadopters/show.html.erb, app/views/earlyadopters/index.html.erb
Controlador	app/controllers/earlyadopters_controller.rb

Análisis: la estación cuenta con una serie de validaciones que hacen ameno su uso, comenzando por el botón de ir a primeros adoptantes, el cual permite la creación de un ticket con el número de clientes que entran y el número de clientes que salen, con una fecha de entrada que permite dejar un buen registro de cada ticket generado, se usan cálculos para determinar la cantidad de clientes en total. Esta estación esta potenciada con un "Estatus de Marketing Digital", la cual permite un fácil entendimiento en cuanto a los clientes que entran versus los clientes que salen, cuenta con una gráfica para representar el total de clientes actuales en porcentaje, sobre la máxima capacidad de clientes totales. También se cuenta con botones como: ver detalles, eliminar ultimo registro añadido y eliminar todos.

5.6. Sexta estación: Abismo

Esta estación muestra el espacio de tiempo y de responsabilidades ejecutadas durante el CHASM (Abismo) del emprendimiento.

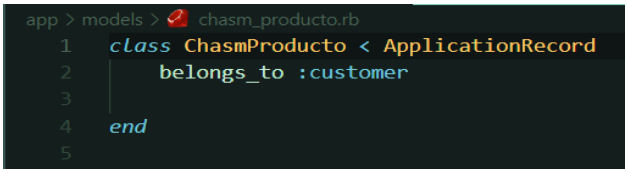
Se creó una estrategia para resolver el CHASM, con en base los siguientes objetivos:

- Enumerar y clasificar Fortalezas del producto. (Deben haber sido encontradas mediante análisis en la estación anterior).
- Enumerar y clasificar Debilidades del producto. (Deben haber sido encontradas mediante análisis en la estación anterior).
- Crear una lista de situación o circunstancia descrita de las fortalezas (antes) de ser glorificadas o mejoradas.
- Crear una lista de situación o circunstancia descrita de las debilidades (antes) de ser corregidas o reparadas.

5.6.1. Modelo de la Sexta estación: Abismo

Este controlador se ubica en **app/models/chasm_producto.rb**, el archivo **chasm_producto.rb**. Muestra una pertenencia al modelo customer, todas las estaciones cuentan con la misma pertenencia.

Figura 85. app/models/chasm_producto.rb



Las siguientes definiciones pertenecen a la figura 85:

- **Herencia:** La clase ChasmProducto hereda de la clase ApplicationRecord. ApplicationRecord es una clase base proporcionada por Rails que ofrece funcionalidades comunes a la mayoría de los modelos de la aplicación.
- **Asociación:** La línea belongs_to :customer define una relación del tipo "pertenece a" entre los modelos ChasmProducto y Customer. Esto significa que un registro de ChasmProducto está asociado con un único registro de Customer. En la base de datos, existe una columna de clave foránea en la tabla chasm_productos que referencie la tabla customers. Esta columna permite vincular cada producto a un cliente específico (SdeC).

5.6.2. Vista de la Sexta estación: Abismo

Definición de la vista chasm.html.erb

Es la vista principal de la sexta estación, la ubicación de **chasm.html.erb** es **app/views/customers/chasm.html.erb** (anexo B.21), en esta vista se podrán crear fortalezas, debilidades y ventas obtenidas por cada fortaleza, de manera intuitiva.

Figura 86. app/views/customers/chasm.html.erb

Las siguientes definiciones pertenecen a la figura 86:

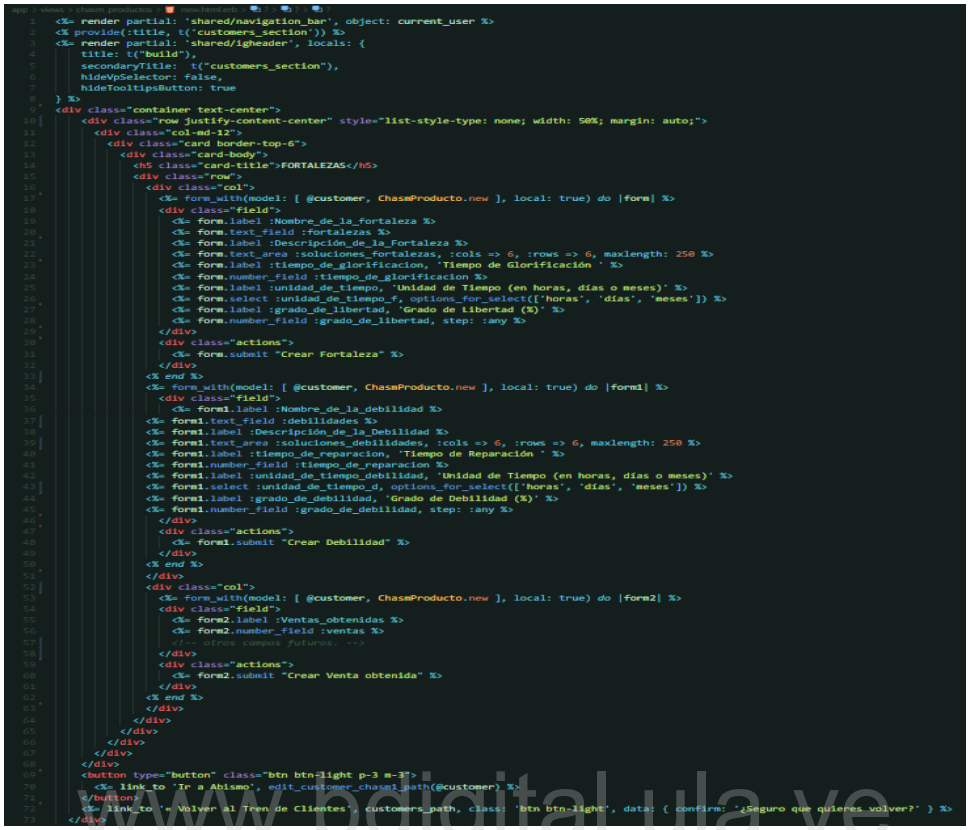
- **Contenedor principal:** Un contenedor div con clase container text-center que centra el contenido de la página.
- **Panel de edición:** Un contenedor div con clase wizard-inner panel panel-default border-top-6 que sirve como panel principal para la edición.
- **Título:** Un encabezado h1 con clase text-center que muestra el nombre del cliente y el título "Editar Estación Abismo".
- **Sección de Fortalezas:** Una etiqueta b que indica "Estas son tus Fortalezas". Una lista sin viñetas (ul) con ID fortalezas-list que itera sobre los productos asociados al cliente (@customer.chasm_productos).
- **Para cada producto:** Se muestra la fortaleza del producto (chasm_producto.fortalezas). Se verifica si se han completado todos los campos de la fortaleza, la debilidad y las ventas. Si es así, se muestra un símbolo "✓". Si no, se muestra una "X". Se incluyen enlaces para editar, mostrar y eliminar el producto.

- **Sección de Debilidades:** Estructura similar a la sección de Fortalezas, pero mostrando las debilidades (chasm_producto.debilidades) y verificando si se han completado los campos de la debilidad y la reparación.
- **Sección de Ventas:** Estructura similar a las secciones anteriores, pero mostrando las ventas obtenidas por cada fortaleza (chasm_producto.ventas) y un enlace para editar las ventas.
- **Alertas:** dos alertas con clases alert alert-danger y alert alert-warning que contienen mensajes informativos. La primera alerta advierte que eliminar una debilidad también eliminará su fortaleza asociada. La segunda alerta explica el significado de los símbolos "✓" y "X" que se muestran junto a cada fortaleza/debilidad.
- **Botón para crear:** Un botón button con clase btn btn-light p-3 m-3 que permite crear una nueva fortaleza, debilidad y agregar ventas.
- **Enlace para volver:** Un enlace link_to con clase btn btn-light y data-confirm que permite volver a la lista de clientes.
- **Funcionalidad:** La vista muestra una lista de productos asociados al cliente actual (@customer). Para cada producto, se muestran sus fortalezas, debilidades, ventas y enlaces para editar, mostrar o eliminar. Se indican visualmente si los campos de cada fortaleza/debilidad están completos. Se incluyen alertas para informar al usuario sobre posibles consecuencias al eliminar debilidades. Se proporcionan botones para crear nuevos productos y volver a la lista de clientes.
- **Detalles adicionales:** El código utiliza la gema **link_to** para crear enlaces a las acciones de edición, visualización y eliminación. Se utiliza la gema **data-confirm** para agregar un mensaje de confirmación al enlace de retorno. Las clases CSS se utilizan para el estilo visual de los elementos.

Definición de la vista new.html.erb

Es una vista esencial para la creación de los formularios recomendados para la estación. La ubicación de **new.html.erb** es **app/views/chasm_productos/new.html.erb** (anexo B.22).

Figura 87. app/views/chasm_productos/new.html.erb



Las siguientes definiciones pertenecen a la figura 87:

- **Contenedor principal:** Un contenedor div con clase container text-center que centra el contenido de la página.
- **Fila de formularios:** Un contenedor div con clase row justify-content-center que organiza los formularios en una fila horizontal centrada. Cada formulario se encuentra dentro de un contenedor div con clase col-md-12.

Formularios:

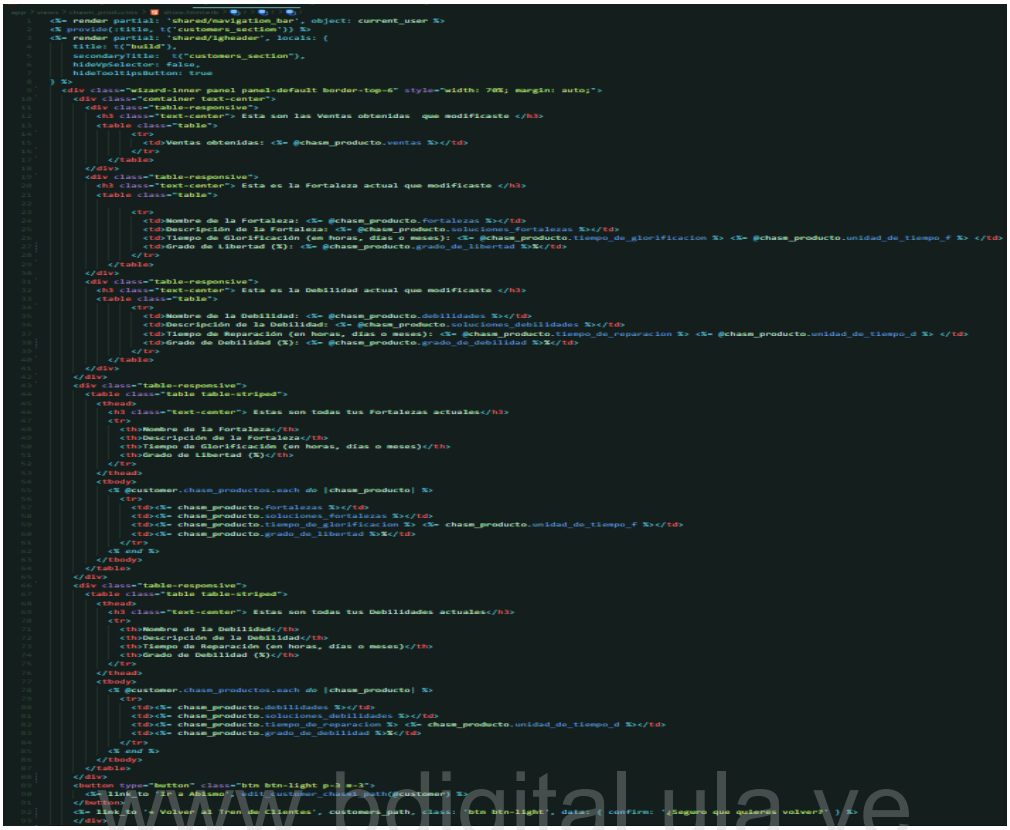
- **Fortalezas:** Un formulario para crear una nueva fortaleza. Incluye campos para: Nombre_de_la_fortaleza (campo de texto), Descripción_de_la_Fortaleza (área de texto)**,** tiempo_de_glorificacion (campo numérico)**,** unidad_de_tiempo_f (lista desplegable con opciones "horas", "días" o "meses")**,** grado_de_libertad (campo numérico con paso decimal) y un botón de envío con la etiqueta "Crear Fortaleza".

- **Debilidades:** Un formulario similar al de fortalezas, pero para crear una nueva debilidad. Incluye campos para: Nombre_de_la_debilidad (campo de texto)**,** Descripción_de_la_Debilidad (área de texto), tiempo_de_reparacion (campo numérico)**,** unidad_de_tiempo_debilidad (lista desplegable con opciones "horas", "días" o "meses"), grado_de_debilidad (campo numérico con paso decimal)**.** Un botón de envío con la etiqueta "Crear Debilidad".
- **Ventas:** Un formulario para crear una nueva venta, incluye campos para: Ventas_obtenidas (campo numérico), un botón de envío con la etiqueta "Crear Venta obtenida".
- **Botones de acción:** Dos botones button con clase btn btn-light p-3 m-3: el primero tiene la etiqueta "Ir a Abismo" y un enlace para la ruta edit_customer_chasm1_path(@customer). El segundo tiene la etiqueta "« Volver al Tren de Clientes" y un enlace para la ruta customers_path con un mensaje de confirmación.
- **Funcionalidad:** Esta vista permite al usuario crear nuevas fortalezas, debilidades y ventas para el cliente actual (@customer). Cada formulario utiliza el método form_with para generar los campos de entrada y el botón de envío. Los campos de entrada se asocian con los atributos correspondientes del modelo ChasmProducto. Los botones de envío envían los datos del formulario al servidor para crear los nuevos registros. Los botones de acción permiten al usuario navegar a otras vistas de la aplicación.
- **Detalles adicionales:** El código utiliza la gema **form_with** para simplificar la creación de formularios. Se incluyen etiquetas label para cada campo de entrada, mejorando la accesibilidad y la experiencia del usuario. Los campos numéricos tienen configurados el paso decimal y las unidades de tiempo para facilitar la entrada de datos. Se utilizan mensajes de confirmación para los botones de acción que podrían generar cambios irreversibles.

Definición de la vista show.html.erb

Es la vista encargada de mostrar el contenido que creó desde **new.html.erb**, muestra un resumen de las ventas obtenidas por la fortaleza actual, la fortaleza actual, la debilidad actual, el conjunto de fortalezas actuales y el conjunto de las debilidades actuales. Su ubicación es **app/views/chasm_productos/show.html.erb** (anexo B,23) y su archivo tiene el nombre de **show.html.erb**.

Figura 88. app/views/chasm_productos/show.html.erb



Las siguientes definiciones pertenecen a la figura 88:

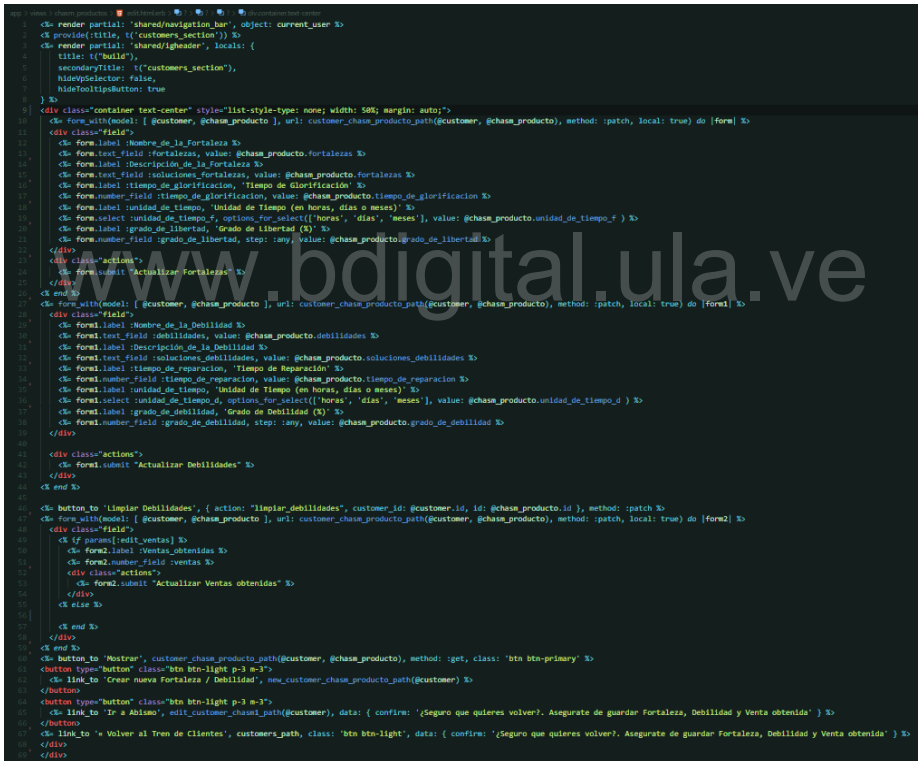
- **Sección de resultados:** esta sección muestra los resultados de la modificación de las "Ventas obtenidas", la "Fortaleza" y la "Debilidad" de un producto. Utiliza variables como @chasm_producto, @customer, @chasm_producto.ventas, @chasm_producto.fortalezas, etc., para acceder a los datos del producto. Muestra los datos en tablas con encabezados y filas.
- **Secciones de Fortalezas y Debilidades:** estas secciones muestran listas de todas las "Fortalezas" y "Debilidades" del cliente actual. Cada fila en la lista muestra el nombre, la descripción, el tiempo de glorificación/reparación y el grado de la fortaleza/debilidad. Se utiliza un bucle each para iterar sobre la colección de productos del cliente (@customer.chasm_productos) y mostrar cada producto en una fila.
- **Botones de acción:** el código incluye dos botones: "Ir a Abismo": este botón lleva al usuario a una página para editar el producto actual ("Abismo") y "« Volver al Tren de

Clientes": este botón lleva al usuario de vuelta a la página del listado de clientes. El botón "Volver al Tren de Clientes" tiene un atributo data-confirm que contiene un mensaje de confirmación que se muestra al usuario antes de redirigirlo.

Definición de la vista edit.html.erb

La vista de edit.html.erb se encarga de gestionar la edición de todos los respectivos formularios de la estación, incluyendo el apartado (individual) de ventas obtenidas por cada una de las fortalezas creadas. Se ubica en **app/views/chasm_productos/edit.html.erb** (anexo B.24) y su archivo tiene de **nombre edit.html.erb**.

Figura 89. app/views/chasm_productos/edit.html.erb



Las siguientes definiciones pertenecen a la figura 89:

- **Contenedor principal:** Un contenedor div con clase container text-center que centra el contenido de la página.
- **Formularios de edición:** se utilizan dos formularios form_with para editar: Fortalezas: un formulario para editar la fortaleza del producto actual (@chasm_producto). Incluye

campos para: Nombre_de_la_Fortaleza (campo de texto con valor inicial @chasm_producto.fortalezas), Descripción_de_la_Fortaleza (campo de texto con valor inicial @chasm_producto.soluciones_fortalezas)**,** tiempo_de_glorificacion (campo numérico con valor inicial @chasm_producto.tiempo_de_glorificacion), unidad_de_tiempo_f (lista desplegable con opciones "horas", "días" o "meses" y valor inicial @chasm_producto.unidad_de_tiempo_f)**,** grado_de_libertad (campo numérico con paso decimal y valor inicial @chasm_producto. Grado_de_libertad), un botón de envío con la etiqueta "Actualizar Fortalezas".

- **Debilidades:** un formulario similar al de fortalezas, pero para editar la debilidad del producto actual (@chasm_producto). Incluye campos para: Nombre_de_la_Debilidad (campo de texto con valor inicial @chasm_producto.debilidades), Descripción_de_la_Debilidad (campo de texto con valor inicial @chasm_producto.soluciones_debilidades), tiempo_de_reparacion (campo numérico con valor inicial @chasm_producto.tiempo_de_reparacion), unidad_de_tiempo_d (lista desplegable con opciones "horas", "días" o "meses" y valor inicial @chasm_producto.unidad_de_tiempo_d), grado_de_debilidad (campo numérico con paso decimal y valor inicial @chasm_producto.grado_de_debilidad), un botón de envío con la etiqueta "Actualizar Debilidades".
- **Botón para limpiar debilidades:** un botón button_to con la etiqueta "Limpiar Debilidades" que utiliza el método patch y llama a la acción limpiar_debilidades.
- **Formulario de edición de ventas (opcional):** un formulario form_with condicional para editar las ventas del producto. Solo se muestra si el parámetro params[:edit_ventas] está presente. Incluye campos para: Ventas_obtenidas (campo numérico), Un botón de envío con la etiqueta "Actualizar Ventas obtenidas".
- **Botones de acción**: dos botones button_to con clase btn btn-primary: el primero tiene la etiqueta "Mostrar" y utiliza el método get para mostrar la página actual. El segundo tiene la etiqueta "Crear nueva Fortaleza / Debilidad" y un enlace para la ruta new_customer_chasm_producto_path(@customer).
- **Dos botones button con clase btn btn-light p-3 m-3:** el primero tiene la etiqueta "Ir a Abismo" y un enlace para la ruta edit_customer_chasm1_path(@customer) con un mensaje

de confirmación. El segundo tiene la etiqueta "« Volver al Tren de Clientes" y un enlace para la ruta customers_path con un mensaje de confirmación.

- **Funcionalidad:** esta vista permite al usuario editar las fortalezas, debilidades y ventas de un producto específico (@chasm_producto) asociado a un cliente (@customer). cada formulario utiliza el método form_with para generar los campos de entrada, los valores iniciales y el botón de envío. Los valores iniciales de los campos se obtienen de las propiedades del producto actual (@chasm_producto). Los botones de envío envían los datos del formulario al servidor para actualizar los registros correspondientes. El botón "Limpiar Debilidades" llama a la acción limpiar_debilidades para limpiar las propiedades de debilidad del producto.

5.6.3. Controlador de la Sexta estación: Abismo

El controlador de la sexta estación es **chasm_productos_controller.rb**, está ubicado en **app/controllers/chasm_productos_controller.rb**, contiene todas las acciones necesarias para crear, mostrar y editar.

Figura 90. app/controllers/chasm_productos_controller.rb

```
# app/controllers/chasm_productos_controller.rb
class ChasmProductosController < ApplicationController
  before_action :set_customer, only: [:new, :create, :edit, :update, :destroy]

  def index
    @customer = Customer.find(params[:customer_id])
    @chasm_productos = @customer.chasm_productos
  end
  def create
    @customer = Customer.find(params[:customer_id])
    @chasm_producto = @customer.chasm_productos.build(chasm_producto_params)

    if @chasm_producto.save
      redirect_to edit_customer_chasm_producto_path(@customer, @chasm_producto), flash: { notice: "El producto fue creado exitosamente." }
    else
      redirect_to customers_path
    end
  end
  def destroy
    @chasm_producto = ChasmProducto.find_by(id: params[:id])

    if @chasm_producto
      @chasm_producto.destroy
      redirect_to customers_path, notice: 'Item eliminado correctamente.'
    else
      redirect_to customers_path, alert: 'Item no encontrado.'
    end
  end
  def edit
    @chasm_producto = ChasmProducto.find(params[:id])
  end
  def update
    @chasm_producto = ChasmProducto.find(params[:id])

    if @chasm_producto.update(chasm_producto_params)
      redirect_to edit_customer_chasm_producto_path(@customer, @chasm_producto), flash: { notice: "El producto fue actualizado exitosamente." }
    else
      render :edit
    end
  end
  def show
    @customer = Customer.find(params[:customer_id])
    @chasm_producto = @customer.chasm_productos.find_by(id: params[:id])

    if @chasm_producto.nil?
      # Maneja el caso donde el producto no se encuentra, por ejemplo, redirigiendo a una página de error o mostrando un mensaje de error
      redirect_to customers_path
    else
      render :show
    end
  end
  def limpiar_debilidades
    @customer = Customer.find(params[:customer_id])
    @chasm_producto = @customer.chasm_productos.find(params[:id])
    @chasm_producto.update(debilidades: nil, soluciones_debilidades: nil, tiempo_de_reparacion: nil, unidad_de_tiempo_d: nil, grado_de_debilidad: nil)
    redirect_to edit_customer_chasm_producto_path(@customer, @chasm_producto)
  end
  private
    def set_customer
      @customer = Customer.find(params[:customer_id])
    end
    def chasm_producto_params
      params.require(:chasm_producto).permit(:ventas, :fortalezas, :soluciones_fortalezas, :debilidades, :soluciones_debilidades, :tiempo_de_glorificacion, :grado_de_libertad, :tiempo_de_reparacion, :grado_de_debilidad, :unidad_de_tiempo_f, :unidad_de_tiempo_d, )
    end
end
```

Las siguientes definiciones pertenecen a la figura 90:

Definición de la clase:

- **class ChasmProductosController < ApplicationController:** Esta línea define una clase llamada ChasmProductosController que hereda de la clase base ApplicationController.
- **Filtro before_action:** before_action :set_customer, only: [:new, :create, :edit, :update, :destroy]: Este filtro ejecuta el método set_customer antes de las acciones new, create, edit, update y destroy. Su función es obtener el objeto Customer asociado al producto actual.

Acciones:

- **index:** Obtiene el cliente actual (@customer) usando el ID del parámetro customer_id. Obtiene todos los productos (@chasm_productos) asociados al cliente actual y renderiza la vista index para mostrar la lista de productos.
- **create:** Obtiene el cliente actual (@customer). Crea un nuevo objeto ChasmProducto asociado al cliente (@chasm_producto) usando los parámetros del formulario (chasm_producto_params). Si el producto se guarda correctamente: redirige a la vista de edición del producto recién creado (edit_customer_chasm_producto_path) con un mensaje de éxito. Si el producto no se guarda: redirige a la vista del listado de clientes (customers_path).
- **destroy:** Busca el producto (@chasm_producto) por su ID (params[:id]). Si el producto se encuentra: elimina el producto (@chasm_producto.destroy), redirige a la vista del listado de clientes (customers_path) con un mensaje de éxito. Si el producto no se encuentra: redirige a la vista del listado de clientes (customers_path) con un mensaje de alerta.
- **edit:** Busca el producto (@chasm_producto) por su ID (params[:id]). Renderiza la vista edit para mostrar el formulario de edición del producto.
- **update:** Busca el producto (@chasm_producto) por su ID (params[:id]). Actualiza el producto (@chasm_producto) con los parámetros del formulario (chasm_producto_params). Si el producto se actualiza correctamente: redirige a la vista de edición del producto actualizado (edit_customer_chasm_producto_path) con un mensaje de éxito. Si el producto no se actualiza: renderiza de nuevo la vista edit (mostrando los errores de validación, si los hay).

- **show:** Obtiene el cliente actual (@customer). Busca el producto (@chasm_producto) por su ID (params[:id]) asociado al cliente actual. Si el producto se encuentra: renderiza la vista show para mostrar los detalles del producto. Si el producto no se encuentra: redirige a la vista del listado de clientes (customers_path).
- **limpiar_debilidades:** Obtiene el cliente actual (@customer). Busca el producto (@chasm_producto) por su ID (params[:id]) asociado al cliente actual. Actualiza el producto para limpiar las propiedades de debilidad (debilidades, soluciones_debilidades, tiempo_de_reparacion, unidad_de_tiempo_d, grado_de_debilidad). Redirige a la vista de edición del producto (edit_customer_chasm_producto_path) con un mensaje de éxito.

Métodos privados:

- **set_customer:** Obtiene el objeto Customer asociado al producto actual usando el ID del parámetro customer_id. Lo asigna a la variable de instancia @customer.
- **chasm_producto_params:** Define los parámetros permitidos del formulario para la creación y actualización de productos. Incluye parámetros para: ventas, fortalezas, soluciones_fortalezas, debilidades, soluciones_debilidades, tiempo_de_glorificacion, grado_de_libertad, tiempo_de_reparacion .. entre otros.

Base de datos (db migrate) del modelo chasm_producto.rb

Esta base de datos se generó al momento de crear el modelo de la sexta estación chasm_producto.rb, cada uno con los atributos necesarios y requeridos por el documento de requisitos del producto. La tabla chasmproductos contiene múltiples atributos necesarios para el despliegue de la respectiva estación.

Figura 91. db/migrate/20240117215417_create_chasm_productos.rb

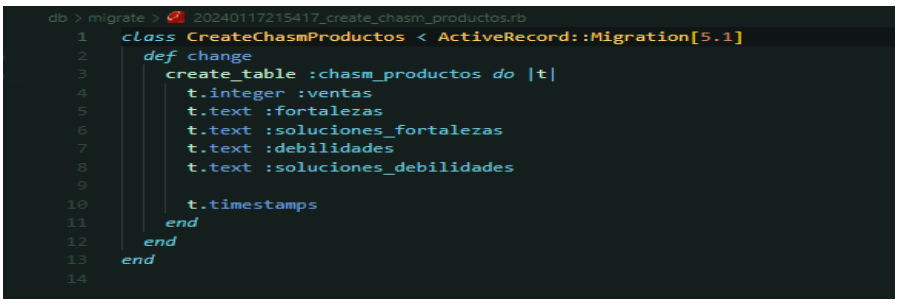


Figura 92. db/migrate/20240119222327_add_customer_ref_to_chasm_productos.rb

Figura 93. db/migrate/20240124221924_add_fields_to_chasm_productos.rb

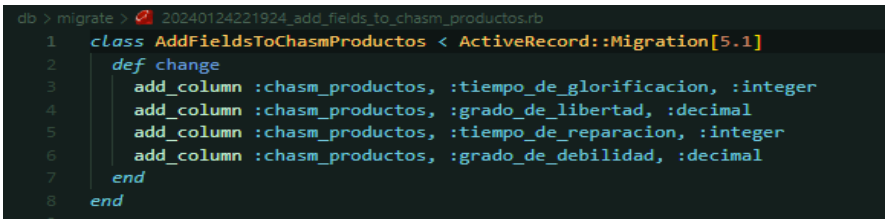


Tabla resumen de la Sexta estación: Abismo

Tabla 8. Tabla resumen de la Sexta estación: Abismo

	Ubicación
Modelo	app/models/chasm_producto.rb
Vista	app/views/customers/chasm.html.erb, app/views/chasm_productos/new.html.erb, app/views/chasm_productos/show.html.erb, app/views/chasm_productos/edit.html.erb
Controlador	app/controllers/chasm_productos_controller.rb

Análisis: esta estación funciona de forma adecuada, abismo (chasm), cuenta con una serie de validaciones para hacer más fácil su uso, se puede agregar fortaleces y debilidades, opcionalmente se puede editar las ventas obtenidas por cada fortaleza creada. Cuenta con operaciones CRUD. La vista mostrar está bien detallada con las ventas obtenidas que se modificaron, la fortaleza actual que se modificó, la debilidad actual que se modificó y un resumen de todas las fortalezas y debilidades actuales con sus respectivos campos.

5.7. Séptima estación: Escalamiento

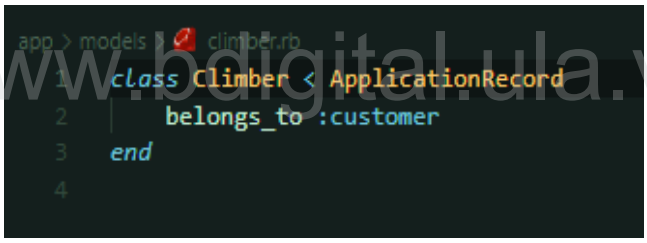
Esta estación contiene un conjunto de bloques de información que representaran, el progreso del proyecto de emprendimiento.

Su Implementación fue una cajetilla de información compuesta por: Número de ventas, Analítica de ventas, Número de clientes activos, Número de clientes inactivos (esta cajetilla de información será única, quiere decir que cada SdeC tendrá una sola cajetilla, se equipó con dos botones: Ver detalles y Editar cajetilla de información).

5.7.1. Modelo de la Séptima estación: Escalamiento

El modelo de esta estación se encuentra en **app/models/climber.rb** cuyo archivo tiene el nombre de **climber.rb**, pertenece al modelo base customer.

Figura 94. app/models/climber.rb



Las siguientes definiciones pertenecen a la figura 94:

- **Climber < ApplicationRecord:** esta línea indica que la clase Climber hereda de la clase base ApplicationRecord proporcionada por Rails.
- **belongs_to :customer:** esta línea define una relación de asociación del tipo "pertenece a" (belongs_to) entre la clase Climber y la clase Customer. Esto significa que un objeto Climber pertenece a un único objeto Customer. En la base de datos, habrá una columna de clave foránea en la tabla climbers que referencie la clave primaria de la tabla customers.

5.7.2. Vista de la Séptima estación: Escalamiento

Definición de la vista climber.html.erb

La ubicación de esta primera vista de la séptima estación es **app/views/customers/climber.html.erb** (anexo B.25), el archivo **climber.html.erb**

Figura 95. app/views/customers/climber.html.erb

Las siguientes definiciones pertenecen a la figura 95:

- **Renderizado de parcial <%= render 'train' %>:** se renderiza un parcial llamado train.
- **Contenedor principal:** un contenedor div con clase container, un div con clase panel panel-default border-top-6. Un contenedor div con clase panel-body. También un contenedor div con clase text-center. Un encabezado <h1> que muestra el nombre del cliente: Escalamiento de: <%= @customer.nombre %>. Un contenedor div con clase container text-center. Un enlace link_to con la etiqueta "Agregar Cajetilla de Información" para crear un nuevo "escalamiento" asociado al cliente actual (SdeC). Se incluye un mensaje de confirmación para evitar acciones no deseadas. Se comprueba si el cliente tiene un " escalamiento " asociado y si este "escalamiento " está guardado en la base de datos (@customer.climber.present? && @customer.climber.persisted?).
- **Si se cumplen estas condiciones:** se muestra un párrafo con el nombre del cliente y un enlace para eliminar el "escalador" asociado. Se incluye un mensaje de confirmación para

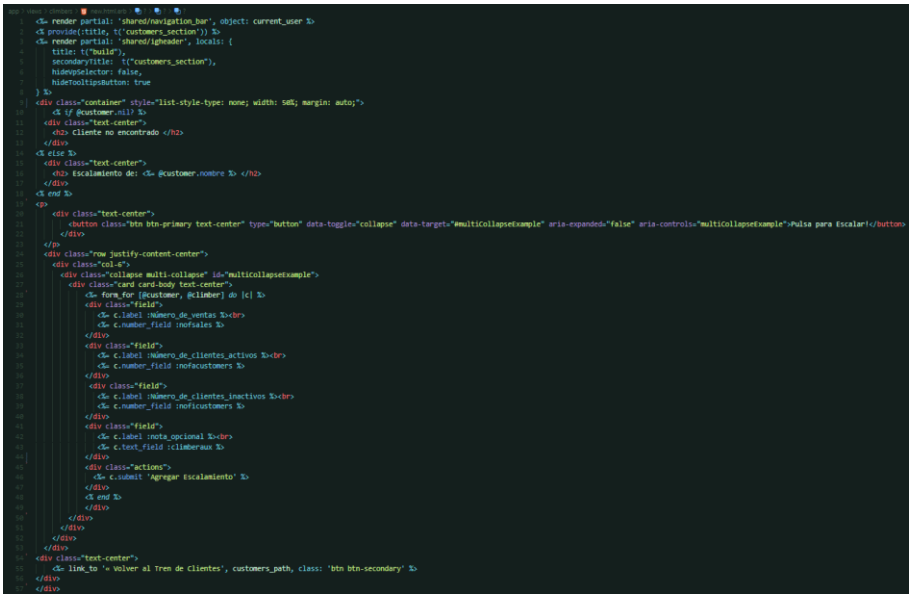
evitar acciones no deseadas. Se muestra un enlace link_to con la etiqueta "« Volver" para regresar a la lista de clientes. Se incluye un mensaje de confirmación para evitar acciones no deseadas. Se muestra un enlace link_to con la etiqueta "Ver Detalles" para ver los detalles del " escalamiento " asociado. Se muestra un enlace link_to con la etiqueta "Editar Cajetilla de Información" para editar el "escalador" asociado. Se incluye un mensaje de confirmación para evitar acciones no deseadas.

- **Funcionalidad:** esta vista muestra el nombre del cliente y la información relacionada con su "escalamiento" (posiblemente un proceso o progreso). Permite al usuario: agregar un nuevo " escalamiento " asociado al cliente, eliminar el " escalamiento " asociado al cliente (si existe), ver los detalles del " escalamiento " asociado (si existe), editar la información del " escalamiento " asociado (si existe) y regresar a la lista de clientes (SdeC).
- **Dependencias:** se utilizan algunas variables de instancia: @customer: representa el objeto Customer actual, @customer.climber: representa el objeto Climber asociado al cliente actual (si existe).
- **Consideraciones adicionales:** los mensajes de confirmación se utilizan para evitar acciones no deseadas por parte del usuario. La vista utiliza enlaces **link_to** para navegar entre diferentes acciones y vistas.

Definición de la vista new.html.erb

La vista **new.html.erb** permite al emprendedor completar los campos correspondientes de la cajetilla de información, las cuales son: Número de ventas, número de clientes activos, número de clientes inactivos y nota opcional. La ubicación de esta vista es **app/views/climbers/new.html.erb** (anexo B.26).

Figura 96. app/views/climbers/new.html.erb



Las siguientes definiciones pertenecen a la figura 96:

Comprobación de cliente:

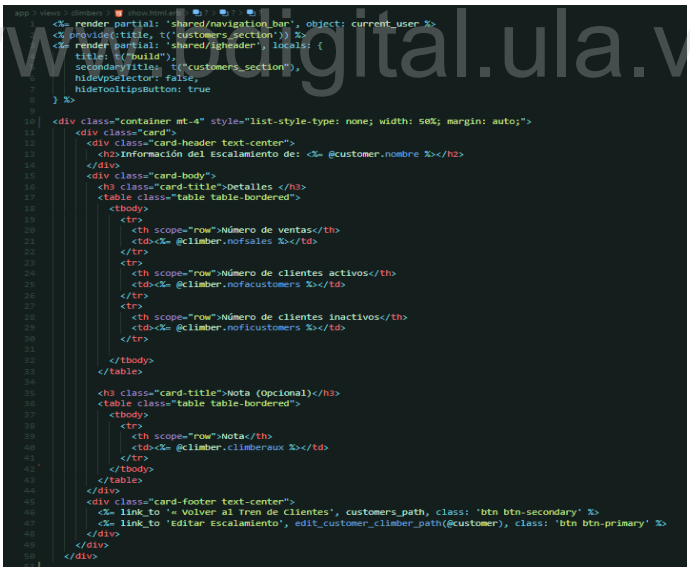
- **<% if @customer.nil? %>:** Se verifica si la variable de instancia @customer está vacía (es nil). Si el cliente está vacío: se muestra un mensaje indicando que "Cliente no encontrado". Si el cliente existe: se muestra un encabezado con el nombre del cliente: <h2> Escalamiento de: <%= @customer.nombre %> </h2>.
- **Botón de escalada:** un botón con la etiqueta "Pulsa para Escalar!" que despliega un acordeón (collapse).
- **Formulario de Escalamiento:** el acordeón (collapse) revela un formulario form_for para crear un nuevo registro de "escalamiento" asociado al cliente actual (@customer). El formulario incluye campos para: Número_de_ventas (campo numérico con la etiqueta y el nombre nofsales), Número_de_clientes_activos (campo numérico con la etiqueta y el nombre nofacustomers), Número_de_clientes_inactivos (campo numérico con la etiqueta y el nombre noficustomers), nota_opcional (campo de texto con la etiqueta y el nombre climberaux) y un botón de envío con la etiqueta "Agregar Escalamiento".
- **Enlace de regreso:** un enlace link_to con la etiqueta "« Volver al Tren de Clientes" para regresar a la lista de clientes (SdeC).

- **Funcionalidad:** esta vista permite crear un nuevo registro de "escalamiento" asociado a un cliente existente. El botón "Pulsa para Escalar!" muestra u oculta el formulario de escalamiento. Se evitan errores mostrando un mensaje si el cliente no se encuentra. El formulario utiliza el método **form_for** para generar los campos de entrada y el botón de envío. Al enviar el formulario, se crea un nuevo registro de "escalamiento" en la base de datos. El enlace "« Volver al Tren de Clientes" permite regresar a la lista de clientes.
- **Mejoras:** se maneja el caso en que el cliente no se encuentre, se organiza el código HTML para mejorar la claridad y se utilizan etiquetas apropiadas para los campos del formulario.

Definición de la vista show.html.erb

Luego de crear el escalamiento, esta vista muestra los detalles del numero de venta, numero de clientes activos, inactivos y una nota (opcional). Su ubicación es **app/views/climbers/show.html.erb** (anexo B.27) y su archivo es **show.html.erb.**

Figura 97. app/views/climbers/show.html.erb



Las siguientes definiciones pertenecen a la figura 97:

- **Estructura HTML:** El código utiliza etiquetas HTML para crear una estructura visual. El contenedor principal tiene la clase "container mt-4", lo que aplica estilos como márgenes y espaciado superior. Dentro del contenedor, se crea una tarjeta usando la clase "card".

- **Sección de encabezado:** El encabezado de la tarjeta (card-header) muestra un título <h2> que menciona "Información del Escalamiento de:". Luego, usando <%= @customer.nombre %>, se inserta dinámicamente el nombre del cliente obtenido de la variable @customer.

- **Sección del cuerpo:** El cuerpo de la tarjeta (card-body) contiene dos secciones principales: Detalles y Nota (Opcional). La sección Detalles utiliza una tabla HTML (table) con bordes (table-bordered) para mostrar tres filas (<tr>) de información, cada fila contiene una celda para el encabezado (<th>) y otra para el valor (<td>). Los valores se insertan dinámicamente usando variables como @climber.nofsales (número de ventas), @climber.nofacustomers (número de clientes activos) y @climber.noficustomers (número de clientes inactivos). La sección Nota (Opcional) también usa una tabla similar para mostrar la nota (@climber.climberaux) asociada al escalamiento.

- **Sección del pie de tarjeta:** El pie de la tarjeta (card-footer) contiene dos botones creados con link_to. El primer botón permite volver al "Tren de Clientes" (customers_path) y tiene la clase "btn btn-secondary" para el estilo. El segundo botón sirve para "Editar Escalamiento" (edit_customer_climber_path(@customer)) y tiene la clase "btn btn-primary" para el estilo.

Definición de la vista edit.html.erb

Esta vista cumple con la función de poder editar la cajetilla de información correspondiente de la séptima estación, su ubicación es **app/views/climbers/edit.html.erb** (anexo B.28) y su archivo tienen de nombre **edit.html.erb**

Figura 98. app/views/climbers/edit.html.erb

```
<%= render partial: 'shared/navigation_bar', object: current_user %>
<% provide(:title, t('customers_section')) %>
<%= render partial: 'shared/igheader', locals: {
    title: t("build"),
    secondaryTitle:  t("customers_section"),
    hideVpSelector: false,
    hideTooltipsButton: true
} %>
<div class="container" style="list-style-type: none; width: 50%; margin: auto;">
    <p>
        <div class="text-center">
          <button class="btn btn-primary text-center" type="button" data-toggle="collapse" data-target="#multiCollapseExample" aria-expanded="false" aria-controls="multiCollapseExample">Pulsa para Editar!</button>
        </div>
      </p>
      <div class="row justify-content-center">
        <div class="col-6">
          <div class="collapse multi-collapse" id="multiCollapseExample">
            <div class="card card-body text-center">
              <%= form_for [@customer, @climber] do |c| %>
                <div class="field">
                  <%= c.label :Número_de_ventas %><br>
                  <%= c.number_field :nofsales %>
                </div>
                <div class="field">
                  <%= c.label :Número_de_clientes_activos %><br>
                  <%= c.number_field :nofacustomers %>
                </div>
                <div class="field">
                  <%= c.label :Número_de_clientes_inactivos %><br>
                  <%= c.number_field :noficustomers %>
                </div>
                <div class="field">
                  <%= c.label :nota_opcional %><br>
                  <%= c.text_field :climberaux %>
                </div>
                <div class="actions">
                  <%= c.submit 'Actualizar Escalamiento' %>
                </div>
              <% end %>
            </div>
          </div>
        </div>
      </div>
    <div class="text-center">
        <%= link_to '« Volver al Tren de Clientes', customers_path, class: 'btn btn-secondary' %>
    </div>
    </div>
</div>
```

Las siguientes definiciones pertenecen a la figura 98:

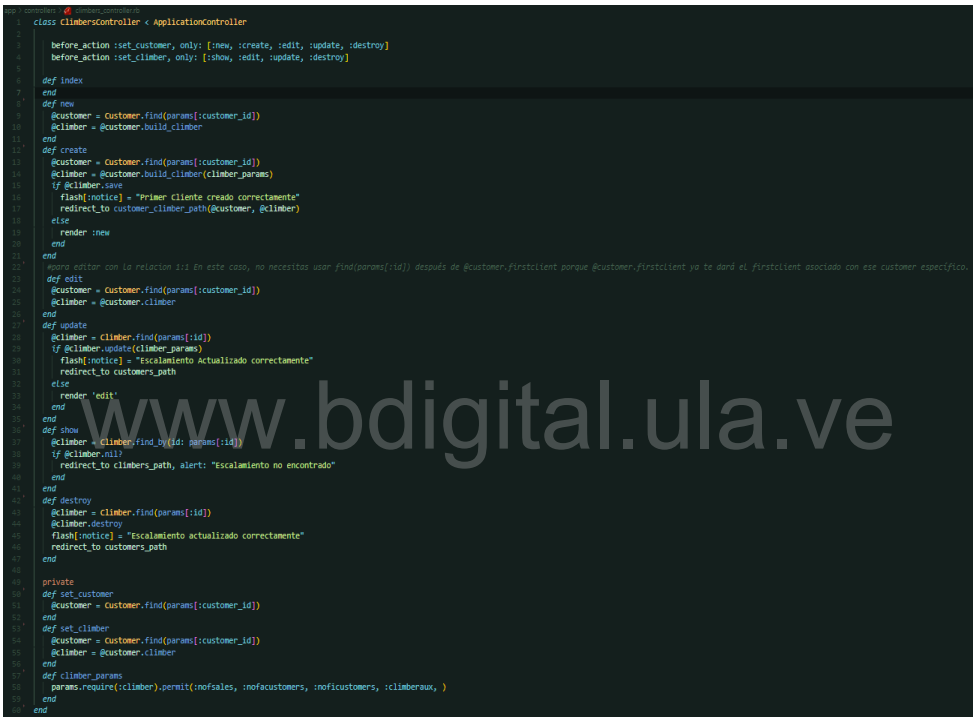
- **Botón de edición:** un botón con la etiqueta "Pulsa para Editar!" que despliega un acordeón (collapse).
- **Formulario de Escalamiento:** el acordeón (collapse) revela un formulario form_for para editar un registro de "escalamiento" existente asociado al cliente actual (@customer). El formulario incluye campos para: Número_de_ventas (campo numérico con la etiqueta y el nombre nofsales), Número_de_clientes_activos (campo numérico con la etiqueta y el nombre nofacustomers), Número_de_clientes_inactivos (campo numérico con la etiqueta y el nombre noficustomers), nota_opcional (campo de texto con la etiqueta y el nombre climberaux) y un botón de envío con la etiqueta "Actualizar Escalamiento".
- **Enlace de regreso:** un enlace link_to con la etiqueta "« Volver al Tren de Clientes" para regresar a la lista de clientes.
- **Funcionalidad:** esta vista permite editar un registro de "escalamiento" existente asociado a un cliente. El botón "Pulsa para Editar!" muestra u oculta el formulario de edición. El formulario utiliza el método form_for para generar los campos de entrada y el botón de

envío. Al enviar el formulario, se actualizan los datos del registro de "escalamiento" en la base de datos. El enlace "« Volver al Tren de Clientes" permite regresar a la lista de clientes.

5.7.3. Controlador de la Séptima estación: Escalamiento

El controlador se ubica en **app/controllers/climbers_controller.rb**, su archivo **climbers_controller.rb** controla todas las acciones necesarias para el correcto funcionamiento de la séptima estación.

Figura 99. app/controllers/climbers_controller.rb



```ruby
class ClimbersController < ApplicationController

  before_action :set_customer, only: [:new, :create, :edit, :update, :destroy]
  before_action :set_climber, only: [:show, :edit, :update, :destroy]

 def index
 end
 def new
   @customer = Customer.find(params[:customer_id])
   @climber = @customer.build_climber
 end
 def create
   @customer = Customer.find(params[:customer_id])
   @climber = @customer.build_climber(climber_params)
   if @climber.save
     flash[:notice] = "Primer Cliente creado correctamente"
     redirect_to customer_climber_path(@customer, @climber)
   else
     render :new
   end
 end
 #para editar con la relacion 1:1 En este caso, no necesitas usar find(params[:id]) después de @customer.firstclient porque @customer.firstclient ya te dará el firstclient asociado con ese customer específico.
  def edit
   @customer = Customer.find(params[:customer_id])
   @climber = @customer.climber
 end
 def update
   @climber = Climber.find(params[:id])
   if @climber.update(climber_params)
     flash[:notice] = "Escalamiento Actualizado correctamente"
     redirect_to customers_path
   else
     render 'edit'
   end
 end
 def show
   @climber = Climber.find_by(id: params[:id])
   if @climber.nil?
     redirect_to climbers_path, alert: "Escalamiento no encontrado"
   end
 end
 def destroy
   @climber = Climber.find(params[:id])
   @climber.destroy
   flash[:notice] = "Escalamiento actualizado correctamente"
   redirect_to customers_path
 end

 private
 def set_customer
   @customer = Customer.find(params[:customer_id])
 end
 def set_climber
   @customer = Customer.find(params[:customer_id])
   @climber = @customer.climber
 end
 def climber_params
   params.require(:climber).permit(:nofsales, :nofacustomers, :noficustomers, :climberaux, )
 end
end
```


Las siguientes definiciones pertenecen a la figura 99:

before_action:

- **set_customer:** Busca un cliente en la base de datos antes de ejecutar las acciones new, create, edit, update, y destroy.
- **set_climber:** Busca un registro de "escalamiento" en la base de datos antes de ejecutar las acciones show, edit, update, y destroy.
- **index:** Método vacío (no se utiliza).

- **new:** Busca el cliente con el ID especificado en los parámetros (params[:customer_id]). Crea un nuevo objeto Climber asociado al cliente encontrado. Renderiza la vista new para crear un nuevo registro de "escalamiento".
- **create:** Busca el cliente con el ID especificado en los parámetros (params[:customer_id]). Construye un nuevo objeto Climber asociado al cliente usando el método build_climber y los parámetros del formulario (climber_params). Si el objeto Climber se guarda correctamente: muestra un mensaje flash de aviso indicando que el "Primer Cliente" se ha creado (el mensaje parece incorrecto, debería mencionar "escalamiento" en su lugar). Redirecciona a la página de detalles del "escalamiento" creado (customer_climber_path). Si hay errores al guardar el objeto Climber: vuelve a renderizar la vista new para que el usuario corrija los errores.
- **edit:** Busca el cliente con el ID especificado en los parámetros (params[:customer_id]). Obtiene el registro de "escalamiento" asociado al cliente (@customer.climber). Renderiza la vista edit para editar el registro de "escalamiento".
- **update:** Busca el registro de "escalamiento" con el ID especificado en los parámetros (params[:id]). Actualiza el registro de "escalamiento" con los nuevos valores del formulario (climber_params). Si la actualización se realiza correctamente: muestra un mensaje flash de aviso indicando que el "Escalamiento" se ha actualizado. Redirecciona a la lista de clientes (customers_path). Si hay errores al actualizar el registro de "escalamiento": vuelve a renderizar la vista edit para que el usuario corrija los errores.
- **show:** Busca el registro de "escalamiento" con el ID especificado en los parámetros (params[:id]). Si no se encuentra el registro de "escalamiento": redirecciona a la lista de registros de "escalamiento" (climbers_path) y muestra un mensaje de alerta indicando que el "Escalamiento no encontrado". Si se encuentra el registro de "escalamiento": lo asigna a la variable de instancia @climber para ser utilizado en la vista. Se renderiza la vista show para mostrar los detalles del "escalamiento".
- **destroy:** Busca el registro de "escalamiento" con el ID especificado en los parámetros (params[:id]). Elimina el registro de "escalamiento" de la base de datos. Muestra un mensaje flash de aviso indicando que el "Escalamiento se ha eliminado". Redirecciona a la lista de clientes (customers_path).

Métodos privados:

- **set_customer:** Busca un cliente en la base de datos y lo asigna a la variable de instancia @customer.
- **set_climber:** Busca un registro de "escalamiento" asociado al cliente actual y lo asigna a la variable de instancia @climber.
- **climber_params:** Filtra los parámetros permitidos del formulario para la creación o actualización de un registro de "escalamiento".

Base de datos (db migrate) climber.rb

Esta base de datos se generó al momento de crear el modelo de la séptima estación climber.rb, cada uno con los atributos necesarios y requeridos por el documento de requisitos del producto. La tabla climbers contiene múltiples atributos necesarios para el despliegue de la respectiva estación.

Figura 100. db/migrate/20240126225454_create_climbers.rb


```ruby
class CreateClimbers < ActiveRecord::Migration[5.1]
  def change
    create_table :climbers do |t|
      t.integer :nofsales
      t.integer :nofacustomers
      t.integer :noficustomers
      t.text :climberaux

      t.timestamps
    end
  end
end
```

Figura 101. db/migrate/20240126230109_add_customer_ref_to_climbers.rb

```ruby
class AddCustomerRefToClimbers < ActiveRecord::Migration[5.1]
  def change
    add_reference :climbers, :customer, foreign_key: true
  end
end
```

Tabla resumen de la Séptima estación: Escalamiento

Tabla 9. Tabla resumen de la Séptima estación: Escalamiento

	Ubicación
Modelo	app/models/climber.rb
Vista	app/views/customers/climber.html.erb, app/views/climbers/new.html.erb, app/views/climbers/edit.html.erb
Controlador	app/controllers/climbers_controller.rb

Análisis: permite agregar una cajetilla de información de forma sencilla, tiene en la misma vista principal botones para ver detalles y editar la cajetilla de información, solo es posible tener una cajetilla de información, si el usuario desea crear una nueva cajetilla, la actual cajetilla será remplazada. Se cuenta con un formulario sencillo y funcional.

5.8. Octava estación: Empresa

Esta visualización, contiene un boceto de diseño libre, una felicitación de parte de **IGNIS GRAVITAS INC**, contiene una animación de un tren desplazándose mañana, tarde y noche, simbolizando el gran esfuerzo que es emprender, por haber construido la idea propuesta y levantarla al punto de llevarla a una empresa. La ubicación de esta estación es **app/views/customers/enterprise.html.erb** y su único archivo es **enterprise.html.erb.**

5.9. Despliegue del Módulo Tren de Clientes en la Plataforma de Ignis Gravitas Inc.

Para integrar módulo del Tren de Clientes al repositorio privado de Ignis Gravitas, se utilizó GitHub, ya que el repositorio de la plataforma se encuentra ahí. Como se trabajó desde un git clone con SSH, la ubicación por defecto en la que se trabajo fue en master*, se creó una nueva rama llamada TOC_IG* para que luego de ser revisado el código por el equipo de Ignis Gravitas se pueda llevar producción.

Los pasos para subir el Tren de Clientes al repositorio privado de Ignis Gravitas fueron los siguientes:

Para pasar de una rama a otra (de Master* -> TOC_IG*)

1. **git add .**
2. **git commit -m "TOC Changes"**
3. **git checkout -b TOC_IG**
4. **git branch**
5. **git commit**
6. **git pos origin TOC_IG**

CONCLUSIONES

- Se implementó con éxito el módulo Tren de Clientes dentro de la plataforma Ignis Gravitas, utilizando el popular framework Ruby on Rails, en la cual también se utilizaron lenguajes de programación como JavaScript, apoyándose en el patrón Modelo Vista Controlador (MVC), cuyos beneficios lograron llevar a otro nivel al Tren de Clientes, estos beneficios permitieron una mayor modularidad, lo que implica que los componentes MVC están bien definidos y separados, facilitando su desarrollo y mantenimiento. El patrón ofreció una gran flexibilidad permitiendo una fácil adaptación, sin olvidar la claridad, ya que cada componente tiene una responsabilidad específica lo que ofreció la manipulación del código y su nivel de depuración.

- La curva de aprendizaje jugó un papel fundamental en el desarrollo de cada una de las estaciones que componen el Tren de Clientes, permitiendo un rápido desarrollo a medida que se iban construyendo cada una de las estaciones; es decir, con cada estación completada la siguiente perdía un grado de dificultad gracias al aprendizaje que se iba obteniendo.

- Al integrar el Tren de Clientes a la plataforma Ignis Gravitas, se crea un impacto positivo en el cual se beneficiaron los responsables de la plataforma, ya que les permite ofrecer una herramienta adicional de alto nivel tecnológico, incrementa el atractivo de la plataforma para los actuales y futuros emprendedores. Así estos podrán recorrer el camino del emprendimiento con un mayor de conocimientos científicos, alcanzar una mayor probabilidad de éxito y alcanzar esa meta deseada llamada empresa.

- El rol de Ruby on Rails fue fundamental para el desarrollo de la investigación, permitiendo acceder a una modularidad veloz, gracias a su flexibilidad y facilidad para la organización de códigos.

RECOMENDACIONES

- En un futuro cercano Ignis Gravitas Inc, podria mejorar el módulo de Tren de Clientes, realizando una revisión a profundidad de los códigos, para mejorar la interfaz y la experiencia de usuario en general, también aplicar una función para visualizar el TOC en el idioma inglés. También estudiar próximas funcionalidades que beneficien a los emprendedores y permitan acelerar aún más para alcanzar la meta de la empresa.

- La curva de aprendizaje debería plantearse como refuerzo de enseñanza, ya que incentiva a mejorar el desarrollo de productos, permitiéndoles consumir menos tiempo para su elaboración, es decir, promover en un futuro este estilo de aprendizaje como refuerzo para la creación de productos en Ignis Gravitas Inc. Se recomienda incentivar a los estudiantes de Ingeniería de Sistemas de la Universidad de los Andes; de otras universidades, así como personas interesadas en este campo para que realicen proyectos relacionados a esta línea de investigación.

- Promover en el futuro la plataforma como atractivo para atraer emprendedores y que estos se beneficien no solo del TOC, sino de toda la plataforma aceleradora. Se recomienda el uso de campañas publicitarias y estrategias de marketing modernas para reforzar el valor atractivo del Ignis Gravitas Inc.

REFERENCIAS

Miller Paul y Bound Kirsten. (2011). Informe "The Startups Factories".

Acerca de Ruby. (2007). Obtenido de (Pág web en Línea). Disponible: http://www.ruby-lang.

Adair, J. (1992). El Reto Gerencial de la Innovación. LEGIS, 10. Obtenido de LEGIS.

Arias, Fidias . (2006). El proyecto de Investigación: Introducción a la metodología cientïfica. Caracas: 6° Edicción, Editorial Episteme.

Balestrine, Miriam. (2006). Como se elabora el Proyecto de Investigación. Caracas: 6° Edición , BL Consultores Asociados, Servicio Editorial.

Bednar, R y Tariskova, N. (año 2, número 5). "Indicators of startups failure". Internacional Scientific Journal, pp 238-240.

Bednar, R. y. (2017 numero 5). Indicators of Startups Failure. Internacional Scientific Jaurnal.

Blank, S. (2012). El Manual del propietario de startups. K&S. Ranch Publishing.

Bueno R, A. L. (2021). La Tecnología: una alternativa de servicio al cliente. Bogotá.: Universidad de Nueva Granada.

Castillo B, J. A. (2020). Análisis y Desarrollo del Motor de conocimientosde una Plataforma Tecnológica. Mérida-Venezuela.: Universidad de los Andes.ULA.

Diaz, M. M. (28 de noviembre de 2018). La Innovación: baluarte fundamental para las organizaciones,. (Journa) Recuperado el 10 de febrero de 2023, de INNOVA: Researcha Jocernal, 3(101), 212-229.: https://doi.org/10.33890/innova.v3.n10.1.2018.843

Duarte C, F. (2007). Emprendimiento, empresa y crecimiento empresarial. Revista Contabilidad y Negocios.

Fernández Esteban, C. (06 de septiembre de 2018). Las 10 razones principales por las que fracasan las startups. Obtenido de Bussines Insider España.: https://www.businessinsider.es/

Formichella, M. M. (mayo de 2004). El Concepto de Emprendimiento y su Relación con la Educación, el Empleo y el Desarrollo Local. Obtenido de https://www.researchgate.net/publicatión/281465619

Gimeno C, G. (2018). Las Aceleradoras como Trampolin de "Startups". Valenca.España.

Gimeno C, G. (2018). Las Aceleradoras como Trampolín de Startups. Valencia-España: Universidad Politécnica de Valencia. Facultad de Administración y Dirección de Empresas.

Giraldo, V. (14 de febrero de 2019). Plataformas Digitales:¿Qué son y qué tipos existen? Obtenido de rockcontent: https://rockcontent.com/es/blog/plataformas-digitales

Guild,H. (18 de febrero de 2021). 16 Tipos de Plataforma Tecnológicas. Obtenido de Hippocatresguild: https://hippocratesguild.com.es

Hans Baumann. (o5 de octubre de 2021). Qué es y para qué sirve Ruby on Rails. Obtenido de Crehana.com.

Ignis Gravitas INC, C. (s.f.). Documento de Requisitos del Producto: tren de Clientes.

Neil, P. (16 de Enero de 2017). El 90% de las startups fracasan, esto es lo que necesitan saber el 10%. Obtenido de Fobers-Empresarios: https://www.fobers.com/sistes/neilpatel/

Network, E. B. (2023). La Inversión en Startups Actividad y Tendencias.

Ries, E. (2012). El Método Lean Startup. Cómo crear empresas de éxito utilizando la innovación. barcelona: Deusto.S.A.Edixiones.

Rocha. (2019). Estrategias de Innovación para Empresas Startups. Obtenido de Revista Pensamiento Contemporaneo en Administración.: https://doiorg/10.12712/rpca.v13i1.27394

Tamayo,T. (2006). Proceso de l aInvestigación Científica. Caracas.: Editorial. Limusa.

The MIT, L. (2008). Obtenido de https://opensourse.org/licenses/MIT enlazadoen htttp//www.rubyonrails.org/

Zambrano T, C. A. (2010). Estudio y Analis de la Herramienta Opensourse Ruby on Rails,Orientado al Desarrollo de Aplicaciones Web versus las demás Herramientas Actuales. Guayaquil-Ecuador: Universidad de Guayaquil.

GLOSARIO

HTML (HyperText Markup Language): Lenguaje de marcado estándar para crear páginas web. Define la estructura y contenido de una página web utilizando etiquetas y atributos.

ERB (Embedded Ruby): Plantillas HTML que permiten embeber código Ruby dentro del código HTML. Esto permite generar contenido dinámico en páginas web.

WSL2 (Windows Subsystem for Linux 2): Capa de compatibilidad que permite ejecutar aplicaciones Linux nativas en Windows 10 y 11.

PID (Process ID): Identificador único asignado a cada proceso en ejecución en un sistema operativo.

Docker: Plataforma de Dockerización (contenedores) que permite ejecutar aplicaciones en entornos aislados y portátiles.

Local Host: Término que hace referencia a la computadora en la que se está ejecutando actualmente el código.

Customer: Cliente o usuario de un servicio o producto.

PostgreSQL: Sistema de gestión de bases de datos relacionales de código abierto.

Apt: Herramienta de administración de paquetes para sistemas operativos basados en Debian/Ubuntu.

USER: Variable de entorno que almacena el nombre del usuario actual.

View: Vista de base de datos, que proporciona una perspectiva específica de los datos almacenados en la base de datos.

Index: Estructura de datos que permite acceder a elementos de una colección de manera eficiente.

ID: Identificador único asignado a un elemento en una base de datos o colección.

URL (Uniform Resource Locator): Dirección única que identifica un recurso en internet, como una página web, imagen o archivo.

ANEXOS

ANEXO A

Documento de Requisitos del Producto: TREN DE CLIENTES
En la línea de responsabilidades, el apartado TREN DE CLIENTES o TRAIN OF CLIENTS.
Documento desarrollado por: **Ignis Gravitas Inc**.

1.0. Prefacio. IDEA y TOC.

Se hace necesario resaltar las diferencias entre IDEA y TOC en este momento.
IDEA es una sección de aprendizaje sobre lo que queremos construir como emprendedor, mientras que TOC es una sección de desarrollo. TOC es una herramienta que manejaremos ya con un aprendizaje obtenido, gracias a que en TOC se podrá definir lo necesario. IDEA y TOC trabajaran de forma i**ndependiente** (TOC V2).

IDEA es un estudio detallado sobre la idea o crisis que hayamos ideado o detectado para solucionar respectivamente. TOC es un estudio evolutivo sobre los potenciales clientes del producto. En otras palabras, IDEA se ocupa de orientar a los emprendedores en el chocolate, en el producto, mientras que TOC se ocupa de orientar a los emprendedores en los amantes del chocolate. Es importante entonces comprender esta diferencia cultural existente entre las dos herramientas, una, IDEA se centra en la idea, la otra, TOC se centra en los clientes de esa idea.
Para completar este señalamiento, veremos que la unión entre la mirada IDEA y la mirada TOC que hemos indicado se encuentra en la Primera Estación del Tren de Clientes, La Estación Observacional. En esta estación se podrán definir los Segmentos de Clientes a través de TOC.

1.1. Decisiones de uso de IDEA y TOC.

a. IDEA será una herramienta para analizar la idea y orientarla hacia un mercado real con la cristalización de una Idea-de-Producto.
b. Los potenciales Segmentos de Clientes podrán ser definidos en TOC, Segmentos de Clientes (SdeC).
c. En TOC se reflejará cada SdeC creado.
d. En TOC se podrán definir nuevas circunstancias pertenecientes a los SdeC, si fuera necesario.
e. TOC podrá evolucionar los SdeC experimentando con los Clientes para saber exactamente qué es lo que realmente quieren.

2.0. Primera Estación: Observacional.

La actividad de esta estación es responsabilidad de los Co-Fundadores. El emprendimiento exige que los cofundadores se conviertan en verdaderos científicos al aplicar el método científico sobre lo que se desea crear como producto o servicio. Su objetivo fundamental es identificar la

actividad central, las necesidades, las molestias, los deseos, búsquedas de crecimientos cultural. Para esto, apoyamos a los emprendedores con un juego de formularios.

2.1. Implementación.

2.2. Botón Segmento de Clientes.

Para desplegar los Segmentos de Cliente (SdeC) creados en **TOC** sobre el cual al seleccionar a uno de ellos debe aparecer **un formulario.**
a. Un botón Menú (>) para seleccionar el SdeC a trabajar.

2.3. Formulario: Segmento de Clientes:

El formulario, ver abajo Fig. 1.0, consiste del **nombre del SdeC, dos casillas y dos tarjetas siemesis**, una a la izquierda de la vista y otra a su derecha. Pasamos a detallar los datos a **2.3.1. Nombre del Segmento.**
Nombre definido en **TOC** representando una categoría de clientes.
☆Notas Bene.
a. Cada Segmento de Clientes es único y cada SdeC forma un conjunto.
b. Al entrar a la Estación Observacional, se desplegará el menú de SdeC posicionado en el último SdeC y su última Circunstancia trabajada.

2.3.2. Circunstancia.

Una vez seleccionado un SdeC se podrá ver la última circunstancia creada con todas las casillas de información relacionadas a ella. Se podrá editar su información, así como también se podría agregar una nueva circunstancia con presionar el botón de suma a su lado extremo del título 'Circunstancia'.
Esta casilla Circunstancia tendrá como componentes,
a.- Un recuadro completo para contener 150 caracteres,
b.- Un contador decreciente de caracteres.
c.- Un botón Menú (>) para desplegar todos las Circunstancias. (Abrir o cerrar todas las circunstancias.)
d.- Tres botones para desplegar (circunstancia 1, circunstancia 2 y circunstancia 3).
e.- Una línea calibradora entre dos extremos {"Me ahoga", "Sobrevivo"} (buscar mejores categorías). *Cada circunstancia* tendrá su línea calibradora (nunca, rara vez, 50 y 50, a menudo y siempre).

☆Nota Bene.
Pueden existir varias circunstancias bajo un mismo SdeC. Cada Circunstancia nueva se compone de todo lo que se define a continuación.

2.3.2.1. La Actividad.

Se refiere a la actividad central a realizar por el segmento de cliente identificado en la circunstancia descrita. Por ejemplo, en esta circunstancia se juega al fútbol, se imparte clases, se hace transporte público, etc.
a.- Con un recuadro completo para contener 1500 caracteres y

b.- Un contador decreciente de caracteres.
c.- Cuatro botones selectivos sobre la **intensidad de la actividad** identificada **{Agudo, Fuerte, Moderado, Leve}.**
☆**Nota Bene.** En caso que puedan existir varias actividades **-No deberían**, lo que se recomendará es escribir de primero, la actividad que aquí llamaremos como, **La Actividad Central**, y a continuación seguir con las siguientes actividades en orden de importancia. Cada actividad extra creada bajo una circunstancia podría los emprendedores a pensar en sus extra propuestas, igualmente. Todo esto debe ser expresado en la única casilla disponible y bajo el número de caracteres posibles (150)

La Actividad Central tendrá las siguientes ventanas para su estudio y análisis.

2.3.2.2. Ventana 'Planteamiento del Problema' {'Problem Statement'}.

Está compuesta por cuatro botones emparejados con los cuatros botones de la Ventana Derecha (ver abajo Ventana Soluciones). Cada botón al estar seleccionado, se selecciona automáticamente el botón correspondiente en la Ventana Derecha, y muestra la misma configuración que se indica abajo para cualquiera de los botones seleccionados.

2.3.2.2.1. Botón Trabajo {Actividad}(en IDEA: Trabajo*).
Aquí es muy importante, es el objetivo, de expresar cual es la **actividad** a realizar. Por ejemplo, dentro de la actividad del Fútbol podría existir según la circunstancia el trabajo de entrenamiento, de arbitraje, de dirigente, de transportista, de animador, y como se ejecuta la actividad en la circunstancia descrita aquí. Por ejemplo, se juega al futbol en el patio de atrás de la casa, se asiste a clases en salones sin techo, etc.

El recuadro compone de,
a.- Con su recuadro completo para contener 1500 caracteres y
b.- Un contador decreciente de caracteres.
c.- Cuatro sub-botones radio selectivos sobre la **intensidad de la Actividad Central** identificada {Agudo, Fuerte, Moderado, Leve}

☆**Nota Bene.** En caso que puedan existir varias Ejecuciones, simplemente especificarlas de mayor importancia a menor dentro de la casilla.

2.3.2.2.2. Botón Molestias {Pains}
Tal vez las molestias sean las primeras que vienen en mente y de manera clara. De alguna manera lo incomodo se percibe inmediatamente. Lo importante que estas molestias son faros de luces para mejorar una circunstancia de vida.
Se compone de,
a.- Con su recuadro completo para contener 150 caracteres y
b.- Un contador decreciente de caracteres.
c.- Cuatro sub-botones radio sobre la **intensidad de la Molestia Central** identificada {Agudo, Fuerte, Moderado, Leve}

☆**Nota Bene.** En caso que puedan existir varias Molestias, simplemente especificarlas de mayor importancia a menor dentro de la casilla.

2.3.2.2.3. Botón Deseos {Gains}
Siempre existen deseos, sueños, ansiedades que aparecen al vivir una circunstancia. Por ejemplo, el deseo de un padre que su hijo llegue a Primera División a través de la enseñanza y practica del futbol en una escuela. El sueño de lograr construir algo como experto, etc.
Se compone de,
a.- Con su recuadro completo para contener 150 caracteres y
b.- Un contador decreciente de caracteres.
c.- Cuatro sub-botones radio sobre la **intensidad del Deseo Central** identificado {Agudo, Fuerte, Moderado, Leve}
d.- Un botón Menú (>) para desplegar todas las **Propuestas** creadas.

2.3.2.2.4. Botón Cultura Actual {**Current Culture}** {En IDEA será **Cultura Actual/Current Culture**}
Trata de identificar conscientemente la cultura actual que se vive alrededor de la manera que se ejecuta la actividad con sus molestias y deseos existentes.

Por ejemplo, la cultura que se produce al jugar al futbol en el patio pequeño de tierra atrás de la casa es:
1. Sin uniforme, 2. sin árbitros, 3. Sin obligaciones de ropa o zapatos, 4. usando las paredes para jugar, 5. Sin tiempo limitados, 6. Número de goles limita el tiempo de juego, etc.

Características de la casilla.
Se compone de,

a.- Con su recuadro completo para contener 150 caracteres y
b.- Un contador decreciente de caracteres.
c.- Cuatro sub-botones radio sobre la **intensidad de la Cultura Actual** identificada {Agudo, Fuerte, Moderado, Leve}
☆**Nota Bene.** En caso que puedan existir varias Culturas Actuales, simplemente especificarlas de mayor importancia a menor dentro de la casilla.

2.3.2.3. Ventana 'Soluciones al Problema' {'Problem Solutions'}

En esta ventana su propósito es expresar la solución al planteamiento de la actividad en la circunstancia descrita. Para eso, descomponemos la búsqueda de la solución en los cuatros aspectos correspondientes a los cuatros aspectos identificados en el planteamiento del problema. Se trata de dar respuestas a cada uno de esos aspectos.
Entonces, está compuesta por cuatro botones emparejados con los cuatros botones de la Ventana Planteamiento. Cada botón de esta Ventana Solución al estar seleccionado, se selecciona automáticamente el botón correspondiente en la Ventana Planteamiento (Propuesta con Ejecución, Alivios con Molestias, Satisfacciones con Deseos, Cultura de Mañana con Cultura de Hoy) mostrando la misma configuración que se indica abajo para cualquiera de los botones seleccionados.

2.3.2.3.1. Botón Propuesta {Proposition}
Aquí los emprendedores deben describir como sería entonces la ejecución de la actividad según sus hipótesis, sería la Ejecución Ideal.

Se compone de,
a.- Un recuadro completo para contener 150 caracteres
b.- Un contador decreciente de caracteres.
c.- Cuatro sub-botones radio sobre la intensidad de la **Propuesta** identificada {Agudo, Fuerte, Moderado, Leve}

☆**Nota Bene.** En caso que puedan existir varias Propuestas, simplemente especificar la denominada Propuesta Central, luego especificarlas de mayor importancia a menor dentro de la casilla. También podrían emparejar las diferentes Ejecuciones expresadas en el Botón Ejecución de la Ventana Planteamientos.

2.3.2.3.2. Botón: Alivios {Relieves}
Esta casilla debe ir completándose según el orden descrito en la casilla del botón Molestias. Responder a esas molestias con sus alivios correspondientes que los cofundadores estimen que deben ser.
Se compone de,
a.- Con su recuadro completo para contener 150 caracteres y
b.- Un contador decreciente de caracteres.
c.- Cuatro sub-botones radio sobre la intensidad del **Alivio** identificado {Agudo, Fuerte, Moderado, Leve}
☆**Nota Bene.** En caso que puedan existir varios Alivios, simplemente especificar el denominado Alivio Central, luego especificarlos de mayor importancia a menor dentro de la casilla. También podrían emparejar las diferentes Molestias expresadas en el Botón Molestias de la Ventana Planteamientos.

2.3.2.3.3. Botón: Satisfacciones {Satisfactions}
Esta casilla debe ir completándose según el orden descrito en la casilla del botón Deseos. Responder a esos deseos con sus satisfacciones correspondientes que los cofundadores estimen que deben ser.
Se compone de,
a.- Con su recuadro completo para contener 150 caracteres y
b.- Un contador decreciente de caracteres.
c.- Cuatro sub-botones radio sobre la intensidad de la **Satisfacción** identificado {Agudo, Fuerte, Moderado, Leve}
☆**Nota Bene.** En caso que puedan existir varias Satisfacciones, simplemente especificar la denominada Satisfacción Central, luego especificarlas de mayor importancia a menor dentro de la casilla. También podrían emparejar los diferentes Deseos expresados en el Botón Deseos de la Ventana Planteamientos.

2.3.2.3.4. Nueva Cultura {Su cultura TOC} {En IDEA será **New Culture}**

Trata de identificar conscientemente la nueva cultura que se vivirá alrededor de la manera que se ejecuta la actividad de manera ideal con sus alivios y satisfacciones logradas. Por ejemplo, la cultura que se produce al jugar al futbol en unas canchas de futbol oficiales. Se marcarían, se buscarían árbitros, se entraría uniformado, habría bancas y cuerpos técnicos, etc.

Se compone de,
a.- Con su recuadro completo para contener 150 caracteres y
b.- Un contador decreciente de caracteres.
c.- Cuatro sub-botones radio sobre la intensidad de la **Nueva Cultura** identificada {Agudo, Fuerte, Moderado, Leve}

☆**Nota Bene.** En caso que puedan existir varias culturas tanto en actuales como en nuevas, simplemente especificarlas con la dominante cultura central, luego especificarlas de mayor importancia a menor dentro de la casilla.

Planteamientos.
NOTAS IMPORTANTES: Cada uno de los formularios debe estar referido a un segmento de clientes especifico. Además, estos formularios pueden solamente crearse y llenarse desde TOC.

PREREQUISITOS: Se debe tener construido el Modelo de Negocios Visión (BMC_Vision) para poder realizar la etapa observacional.

REQUISITOS DE FUNCIONALIDAD: Debe guardarse la fecha, hora, etapa y la herramienta desde la cual se procesa una modificación de los datos, de un segmento de cliente o de un cliente especifico, así como su eliminación, o creación de uno nuevo. Debe registrarse que usuario realizo esta modificación.

_____

Acerca de las tarjetas o tickets de clientes y segmento de clientes:

- **Ticket de segmento de cliente:** Este ticket debe tener las siguientes características:
  - Nombre del segmento de cliente.
  - Cantidad de clientes asignados a este segmento. (máximo 20 clientes por segmento).
  - Descripción.

Ticket de cliente:

- **INFORMACION PERSONAL:**
  - Edad.
  - Correo electrónico.
  - Dirección.
  - Teléfono.
  - Nombres y apellidos.
- **INFORMACION DE CONTEXTO:**
  - Profesión.
  - De donde proviene.
  - Contexto de uso del cliente.
  - Grado de interés (0-100%).
  - Grado de Experticia (0-100%).
  - ¿Paga? si o no.
  - Descripción.
- **Segmento de cliente:** Este se desplegará con un selector que mostrará los segmentos de clientes existentes para agregar.

Funcionalidades de crear, modificar o eliminar un segmento de cliente:

Para acceder a esta funcionalidad en la vista de tren de clientes, se debe dar "clic" en un botón con el título **editar segmento de cliente.** Este botón desplegara tres botones:

- **Crear un segmento de cliente nuevo:** Para crear un nuevo segmento de cliente se dará clic en este botón. Se desplegará un formulario con los siguientes datos para rellenar:
  - Nombre del segmento de cliente.
  - Agregar primer cliente al segmento.

Este formulario tendrá un botón "enviar", para culminar el proceso se notifica al usuario mediante una notificación emergente, que se ha creado el Segmento de cliente de manera exitosa.

- **Modificar un segmento de cliente existente:** Para modificar un segmento de cliente se dará clic en este botón. Se desplegará un formulario con los siguientes datos para modificar:
  - Nombre del segmento de cliente.
  - Descripción del segmento de cliente.
  - Agregar un cliente (s) al segmento.
  - Eliminar un cliente del segmento.

  Este formulario tendrá un botón "enviar", para culminar el proceso se notifica al usuario mediante una notificación emergente, que se ha modificado el Segmento de cliente de manera exitosa.
- **Eliminar un segmento de cliente:** Para eliminar un segmento de cliente, se mostrará al hacer clic en este botón, en un listado, todos los segmentos de clientes que existen, en ventanas que mostraran la información de cada uno, con un botón "X" para eliminar. Al presionar este botón, se mostrará una alerta que dice: "Esta seguro que desea eliminar este segmento de cliente. Un botón de aceptar para completar la tarea de eliminar.

Una vez completado se mostrará la notificación: "Se ha borrado exitosamente el segmento de cliente, y se ha actualizado IDEA".

NOTAS IMPORTANTES: Manejo de excepciones y errores: si el segmento de cliente introducido coincide con uno existente, deberá notificarse que el segmento ya existe al usuario, y que debe modificar el nombre elegido.
Se debe establecer un numero de caracteres máximo o de palabras para la descripción, de manera que esta no exceda, se sugiere 80 o 100 caracteres máximo para esto; de la misma forma la cantidad de caracteres para le nombre del segmento, sugerencia máximo 30 caracteres para el nombre.
En cuanto a la eliminación de segmento de cliente, sino existe ningún segmento hasta el momento de presionar este botón, se mostrará una alerta que diga: "Aun no ha creado ningún segmento de cliente".

Funcionalidades de crear, modificar o eliminar un cliente:

Para acceder a esta funcionalidad en la vista de tren de clientes, se debe dar "clic" en un botón con el título **editar cliente.** Este botón desplegara tres botones:

- **Crear cliente nuevo:** Para crear un nuevo segmento de cliente se dará clic en este botón. Se desplegará un formulario con los siguientes datos para rellenar:
  - Nombres y apellidos.
  - Profesión.
  - De donde proviene.
  - Descripción del cliente.

- Información de contacto. (correo electrónico, **opcional teléfono o red social sugerencia).**
- **Segmento de cliente:** Este se desplegará con un selector que mostrará los segmentos de clientes existentes para agregar.

Este formulario tendrá un botón "enviar", para culminar el proceso se notifica al usuario mediante una notificación emergente, que se ha creado el cliente de manera exitosa.

- **Modificar cliente existente:** Para modificar un cliente se dará clic en este botón. Se desplegará un formulario con los siguientes datos para modificar:
  - Nombres y apellidos.
  - Profesión.
  - De donde proviene.
  - Descripción del cliente.
  - Información de contacto. (correo electrónico, **opcional teléfono o red social sugerencia).**
  - **Segmento de cliente:** Este se desplegará con un selector que mostrará los segmentos de clientes existentes para agregar.

Este formulario tendrá un botón "enviar", para culminar el proceso se notifica al usuario mediante una notificación emergente, que se ha modificado el cliente de manera exitosa.

- **Eliminar cliente:** Este botón desplegará una lista con los clientes existentes, se mostrará la información de cada cliente en una ventana, y aparecerá un botón "X" para eliminar. Tras presionar el botón, se notificará mediante una alerta: "Esta seguro que desea eliminar este cliente ". En un botón aceptar, se confirma el proceso. Tras realizarse se notifica mediante un mensaje: " Se ha eliminado de manera exitosa el cliente".

NOTAS IMPORTANTES: Manejo de excepciones y errores: si el cliente introducido coincide con uno existente, deberá notificarse que el cliente ya existe al usuario, y que debe modificar el nombre elegido.
Se debe establecer un numero de caracteres máximo o de palabras para la descripción, de manera que esta no exceda, se sugiere 80 o 100 caracteres máximo para esto; de la misma forma la cantidad de caracteres para le nombre y apellidos del cliente, sugerencia máximo 30 caracteres para el nombre, de la misma manera para el apartado de profesión, de donde proviene e información de contacto.

2nd Estación: Enfoque de Grupo.

Descripción: Esta estación tiene como visión, la realización de experimentos, mediante eventos programables; los cuales serán ejecutados seleccionando clientes como invitados. El análisis de las pruebas del producto con los clientes, determinara las fortalezas y debilidades del producto, lo cual llevara a descubrir el camino del producto, la línea que se debe seguir.

Objetivos: Los objetivos de esta estación son:

- Encontrar mediante el análisis de resultados y conclusiones, de los experimentos realizados, las molestias situaciones y otras circunstancias (las cuales pueden ser

contrastadas con las hipótesis planteadas en la **estación 1: Observacional.** Este contraste será clave para definir y listar los siguientes objetivos:

- Encontrar Fortalezas del producto.
- Encontrar Debilidades del producto.

Implementación.

- **Vista: creación de evento (tarjeta de enfoque de grupo):**
  - Contendrá un botón en el cual se añade una nueva tarjeta con un título, se escribe el título en un campo de texto de no más de 20 caracteres.
  - Un elemento **Select** para seleccionar las tarjetas ya creadas
- **Vista: formulario de tarjeta de enfoque de grupo:**
  - Contiene un numero automático (del 1- número de tarjeta actual, esta se encontrará en el botón ver detalles.
  - Contiene un Nombre o título de máximo 20 caracteres.
  - Contiene un campo de texto para indicar la fecha de la tarjeta.
  - Contiene un campo de texto para indicar la logística (150 caracteres máximo).
  - Contiene un campo de texto para indicar la dirección (50 caracteres máximo).
  - Contiene un campo de texto para indicar la Descripción (300 caracteres máximo).
  - Contiene un campo de texto para indicar la media (50 caracteres máximo)

NOTAS IMPORTANTES: Las tarjetas son modificables si se seleccionan, solo antes de ser rellenado el campo de resultados y conclusiones, durante el evento de notificación emergente de ese componente. La muestra de la tarjeta de información de enfoque de grupo por eventos de tarjeta contendrá todos los elementos de la implementación.

- **Vista de notificación emergente, mostrada en la estación como punto de notificación**
  - Este punto de notificación aparecerá, cada vez que se edite con éxito la Tarjeta enfoque de Grupo. (Seleccionando previamente el nombre de la tarjeta de grupo).

3rd Estación: Etapa Primer Cliente.

Aquí tratamos de orientar a los Cofundadores a identificar un usuario quien sea real, sincero, en lo que vive alrededor del producto que se desarrolla. Esta persona debe conocerse profundamente.
Implementación.
Está compuesto por dos niveles de formularios, los siguientes:

A. Formulario Datos Personales.
Consiste simplemente en capturar los datos de contactos y datos personales que permitan perfilar aún mejor a la persona como Primer Usuario.
-Nombre, Género, Edad, Dirección, Emails, Teléfonos.

B. Formulario Contexto Personal.
Consiste en transcribir el ambiente y actividad que vive el Primer Usuario.
-Profesión, Breve descripción de su personalidad, Grado de necesidad/interés por el producto, Grado de experticia, Contexto de uso y si el cliente paga o no.
También se agregará el **precio estimado por el potencial cliente de la Propuesta.**

4th Estación: Primer Grupo de Clientes:

Descripción: Esta estación contendrá un listado de los segmentos de cliente creados hasta ahora, pero permitirá, añadir una cantidad sustancial por segmento. Estos segmentos contendrán las visualizaciones de los clientes con las respectivas tarjetas de información de ellos y su segmento correspondiente. Se permite añadir clientes acá.
Objetivos: Los objetivos de esta estación son:
Implementación.

- **Vista para la lista de clientes: (nota importante):** Se tendrá un botón **añadir cliente** abajo (+Agregar Primer Cliente de Grupo de Clientes), centrado para el segmento de Cliente seleccionado.
  - Lista de clientes (Selecciona el Primer Grupo de clientes:), el cual al seleccionar un cliente (ya añadido) se podrá ver dos botones: (Ver detalles del cliente) y (Editar).
- **Vista de la tarjeta de cliente:**
  - Ficha de información del cliente (ver detalles del cliente):
- **Vista para agregar un cliente:**
- **Ticket de cliente:**
- **INFORMACION PERSONAL:**
  - Edad.
  - Correo electrónico.
  - Dirección.
  - Teléfono.
  - Nombres y apellidos.
- **INFORMACION DE CONTEXTO:**
  - Profesión.
  - De donde proviene.
  - Contexto de uso del cliente.
  - Grado de interés (0-100%).
  - Grado de Experticia (0-100%).
  - ¿Paga? si o no.
  - Descripción
  - Numero de ventas obtenidas

5th Estación: Primeros adoptantes:

Descripción: Esta estación contendrá un conjunto de bloques de información que representaran, el estatus en cantidad de clientes que han adoptado y comprado el producto, así como representar el estatus de marketing.

Implementación.

- **Una cajetilla de información compuesta por los siguientes elementos:**
  - Número de clientes que llegan (por día, mes y año).
  - Número de clientes que se retiran (por día, mes y año).
  - Número total de clientes que han llegado y se han retirado en el tiempo que se ha desarrollado el emprendimiento del producto.

- Calibrador que represente el estatus de marketing digital.

6th Estación: CHASM (Abismo):

Descripción: Esta estación representa el espacio de tiempo y de responsabilidades ejecutadas durante el CHASM del emprendimiento.

Objetivos: Los objetivos de esta estación son:

- **Diseñar una estrategia para resolver el CHASM, en base a los objetivos siguientes:**
  1. Enumerar y clasificar **Fortalezas del producto.** (Deben haber sido encontradas mediante análisis en la estación anterior).
  2. Enumerar y clasificar Debilidades **del producto**. (Deben haber sido encontradas mediante análisis en la estación anterior).
  3. Crear una lista de situación o circunstancia descrita de las fortalezas (antes) de ser glorificadas o mejoradas.
  4. Crear una lista de situación o circunstancia descrita de las debilidades (antes) de ser corregidas o reparadas.

Implementación.

- En el apartado de Abismo se tendrá un botón para agregar Fortaleza, debilidad y ventas obtenidas
- Esta vista tendrá el nombre de la fortaleza, su descripción, tiempo de glorificación Unidad de tiempo (en horas, días o meses) y grado de libertad (%).
- También esa misma vista tendrá el nombre de la debilidad, descripción de la debilidad, tiempo de reparación Unidad de Tiempo (en horas, días o meses), grado de la debilidad (%).
- En ese mismo espacio se tendrá el número de ventas obtenidas para cada fortaleza/debilidad correspondiente.
-
- Antes: lista de fortalezas antes de ser glorificadas
  - Contiene una ficha con una descripción, tiempo de glorificación, el grado de la fortaleza.
- Despues: lista de fortalezas luego de ser glorificadas. Descripción de la solución.
- Botón de crear fortaleza, debilidad y venta obtenida
- Cada fortaleza se podrá editar, mostrar y eliminar
- Si se elimina una fortaleza se eliminará su debilidad en caso de que tenga.
- Se marcará con una X la fortaleza/debilidad de no haberse completado todos sus campos, de lo contrario se marcará con un check de completar todos sus campos
- Se mostrará la venta obtenida por cada fortaleza/debilidad
- Se podrá editar la venta.

- **Calibrador que refleje estatus de marketing en porcentaje (opcional).**

7th Estación: Escalando:

Descripción: Esta estación contendrá un conjunto de bloques de información que representaran, el progreso del proyecto de emprendimiento.

Implementación.

- **Una cajetilla de información compuesta por los siguientes elementos:**
  - Numero de ventas.
  - Analítica de ventas (Para desarrollar en un próximo MVP). Se supone que serán graficas estadísticas del progreso en la herramienta (**opcional**).
  - Número de clientes activos
  - Número de clientes inactivos.
  - Nota opcional.
    - **Nota**: Se tendrá una sola cajetilla de información por SdeC.
    - La cajetilla estará equipada con dos botones: Ver detalles y Editar cajetilla de información.

8ta Estación: Empresa:

Esta visualización, contendrá un boceto de diseño libre, el cual será una medalla, un premio o una felicitación al emprendedor como usuario, por haber construido su idea y levantarla al punto de llevarla a una empresa.

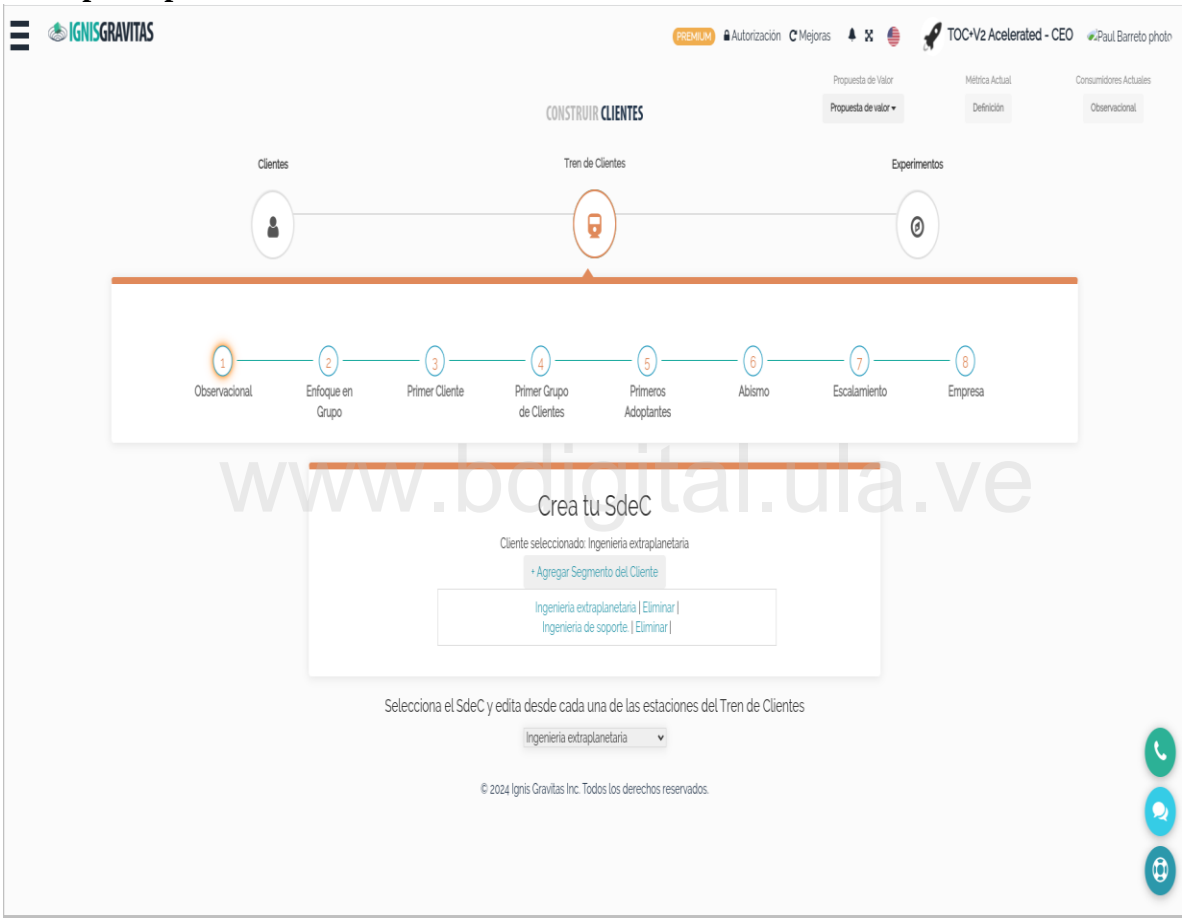
Implementación.

Imagen, información y texto distribuido netamente por diseño gráfico libre (que contraste y siga la transformación de la LANDING PAGE.

Anexo B

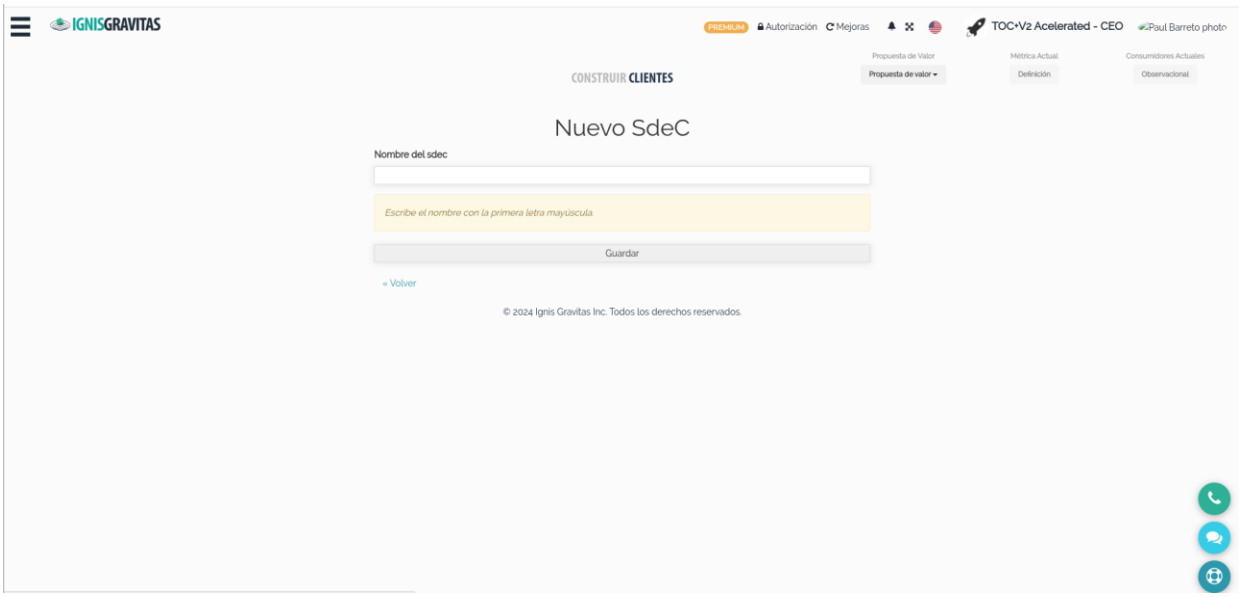
Los siguientes anexos, hacen referencia a información complementaria referente a las diferentes vistas (interfaz) de cada una de las estaciones desarrolladas, incluyendo la vista principal del TOC.

Anexo B.1
Vista principal del Tren de Clientes



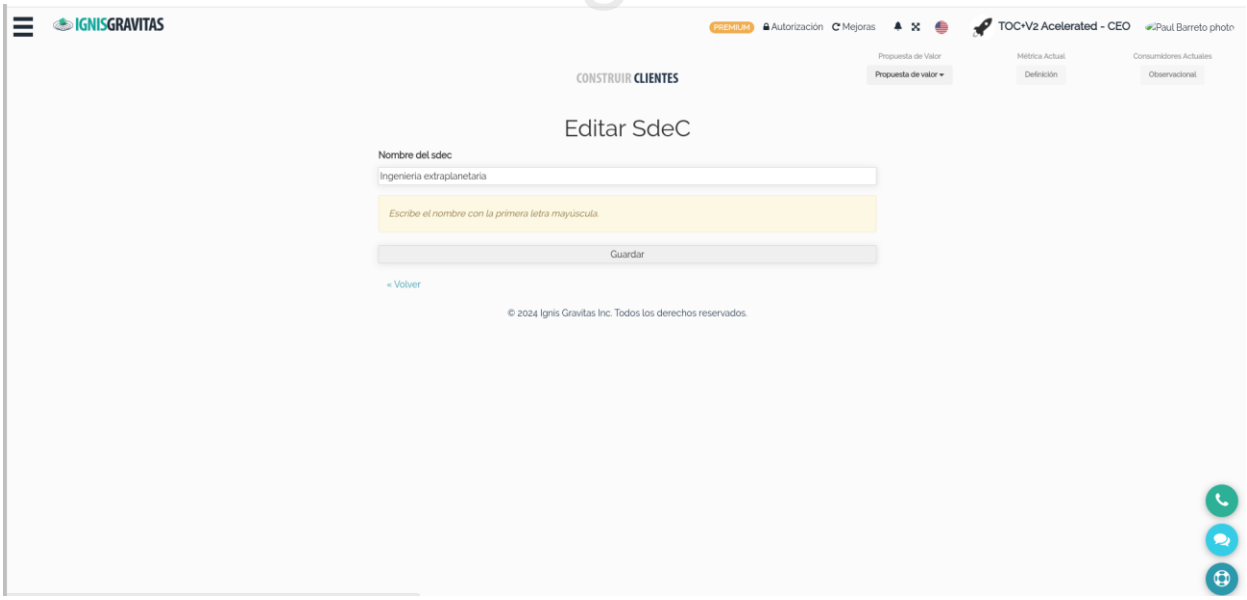
Anexo B.2

Estación Observacional: Nuevo SdeC



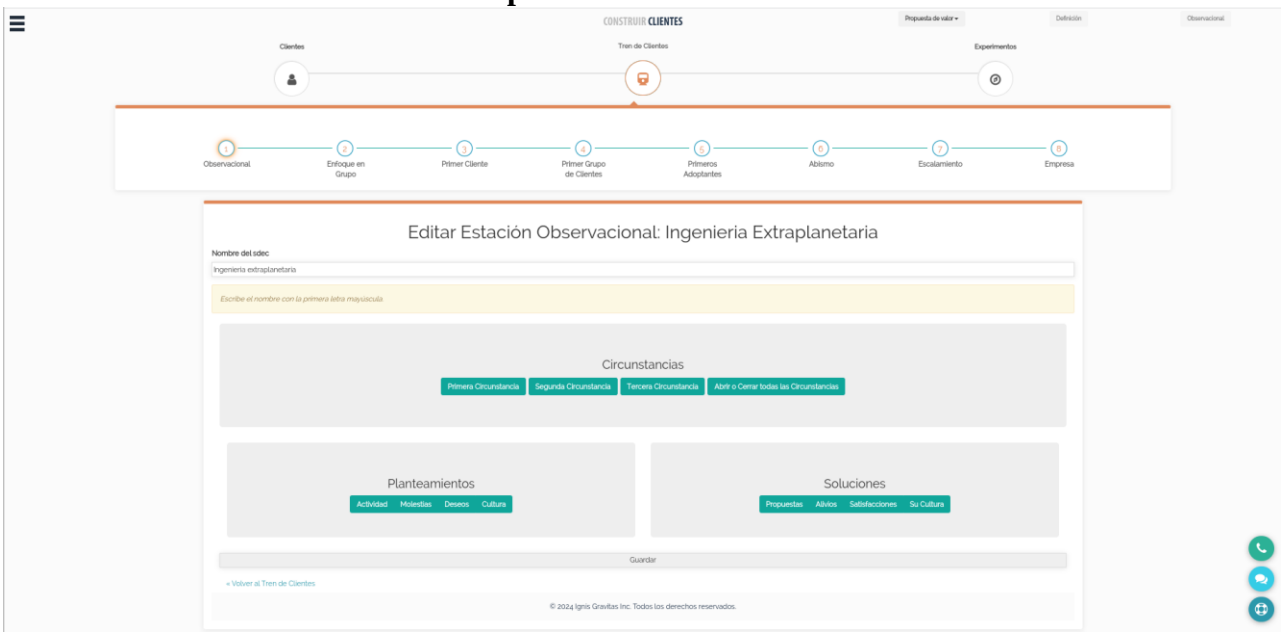
Anexo B.3

Estación Observacional: Editar SdeC



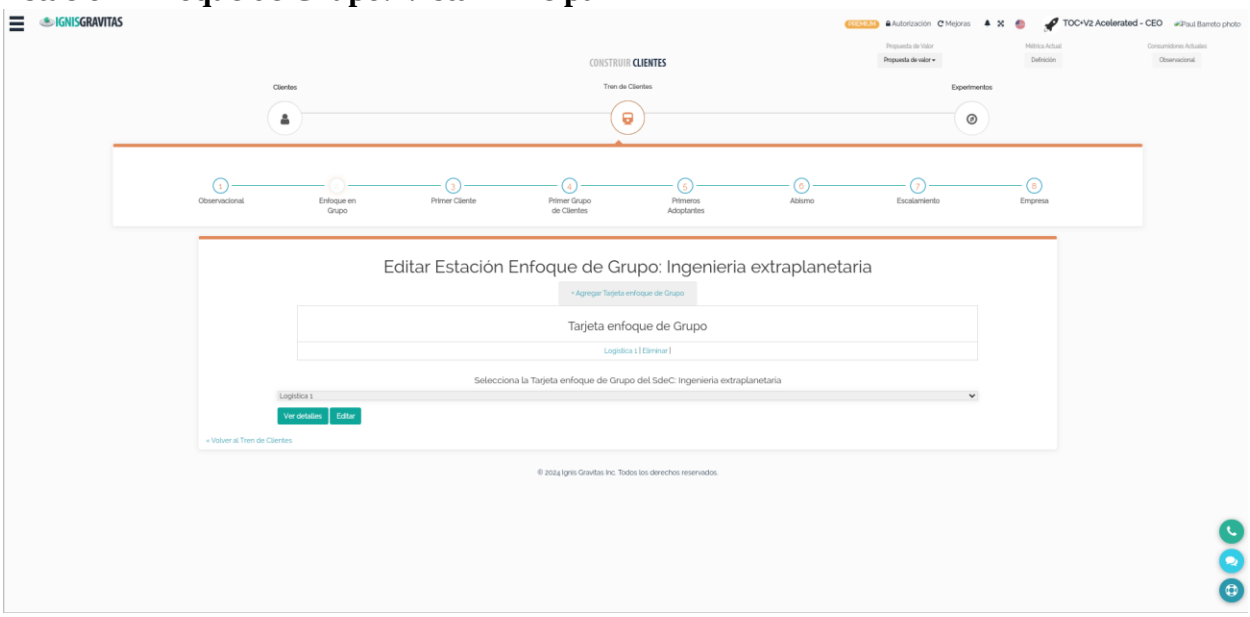
Anexo B.4

Estación Observacional: Vista Principal



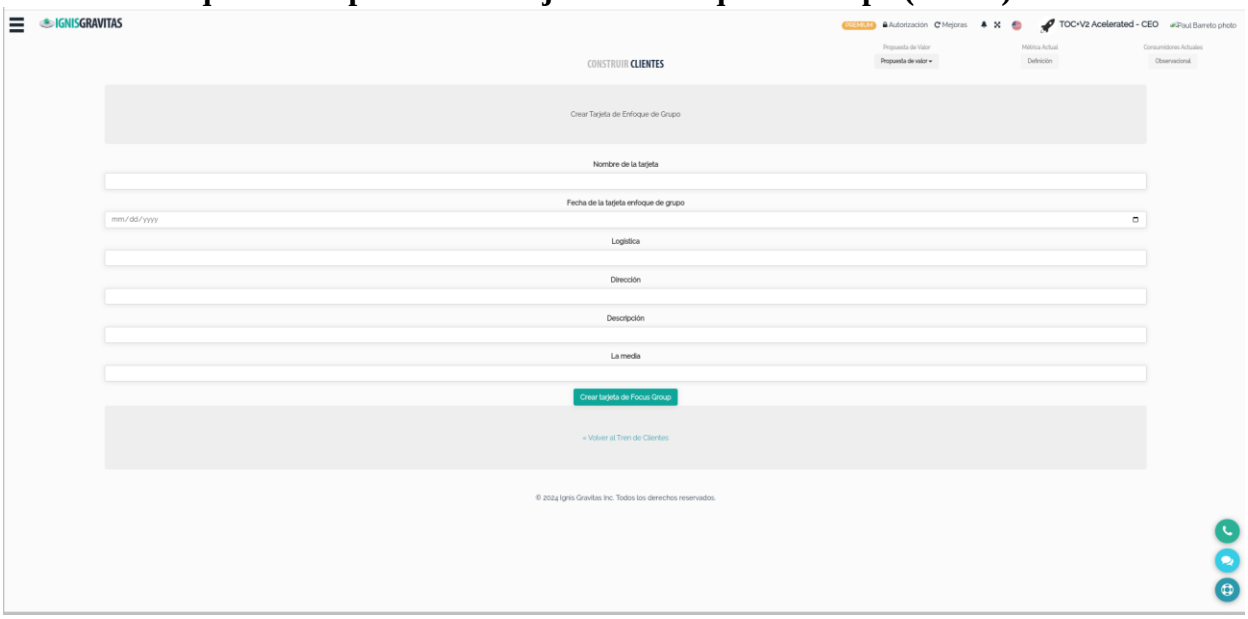
Anexo B.5

Estación Enfoque de Grupo: Vista Principal



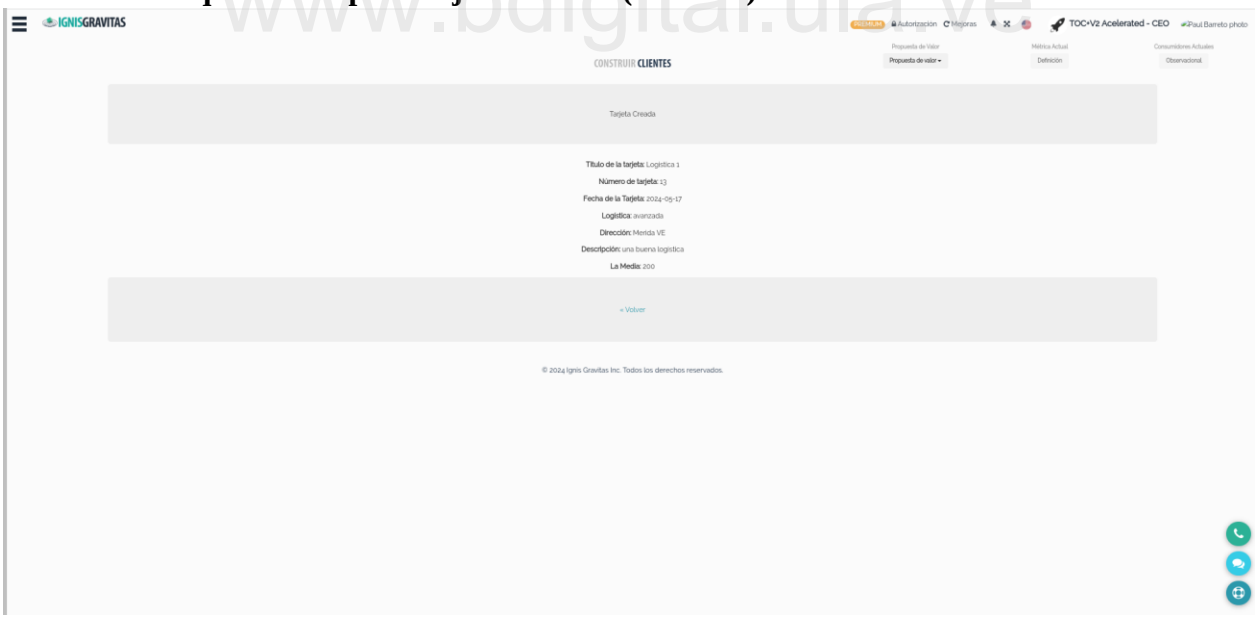
Anexo B.6

Estación Enfoque de Grupo: Crear Tarjeta de Enfoque de Grupo (Nuevo)



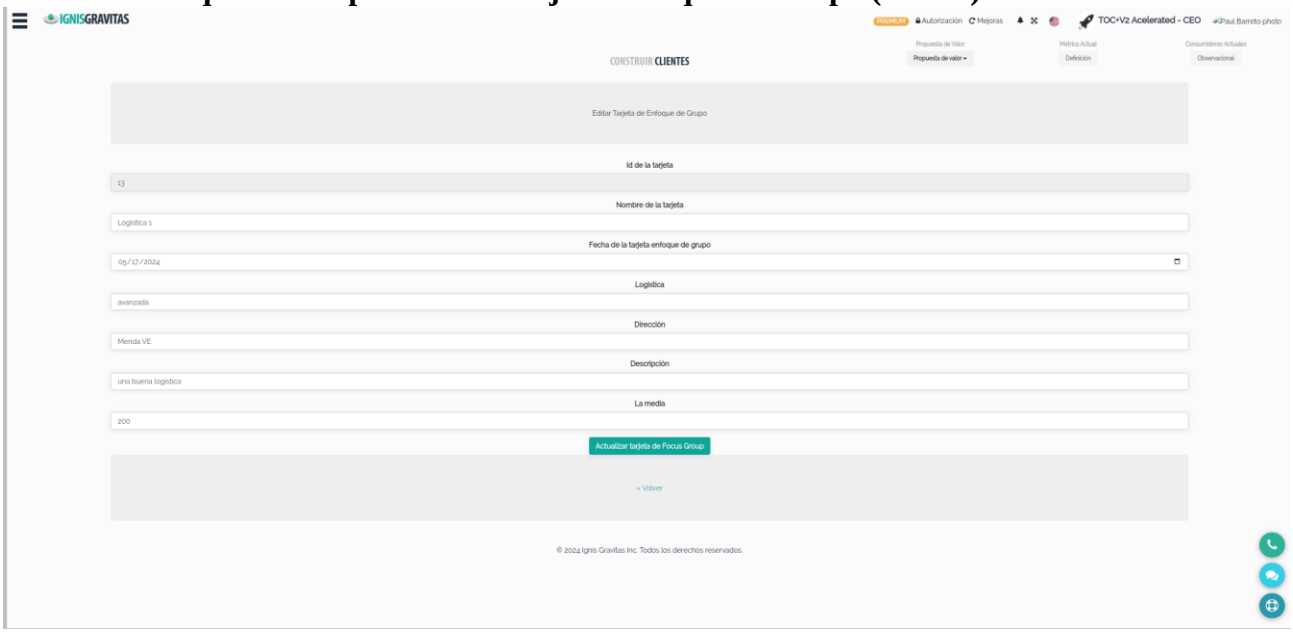
Anexo B.7

Estación Enfoque de Grupo: Tarjeta Creada (Mostrar)



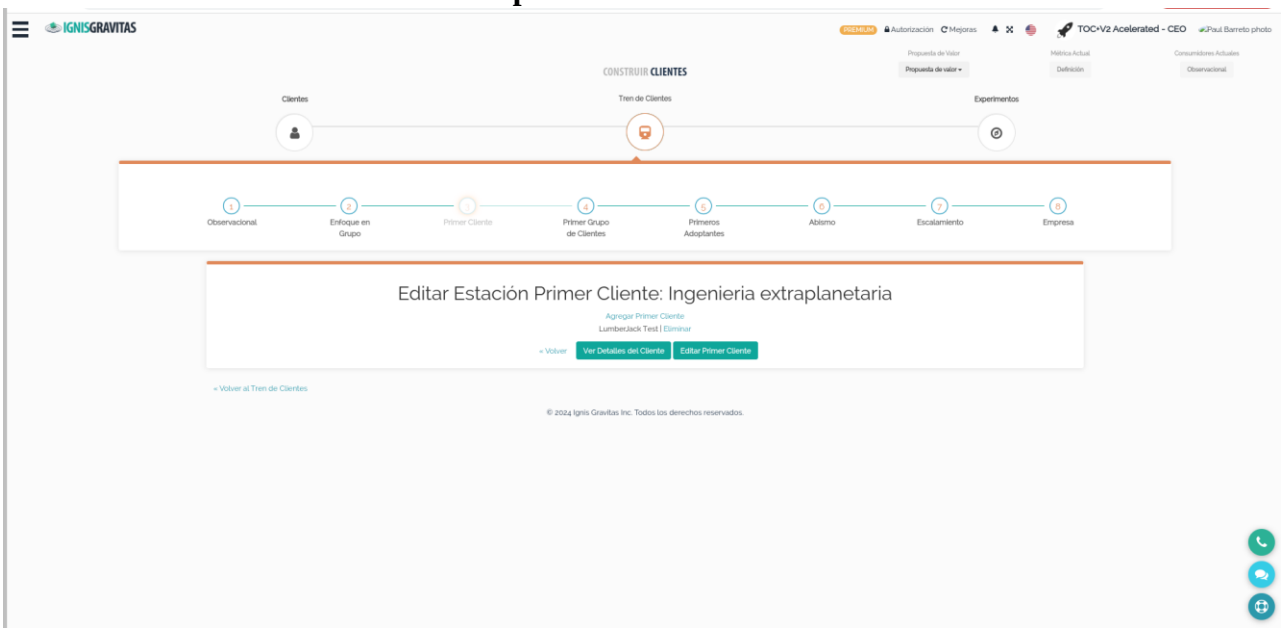
Anexo B.8

Estación Enfoque de Grupo: Editar Tarjeta Enfoque de Grupo (Editar)



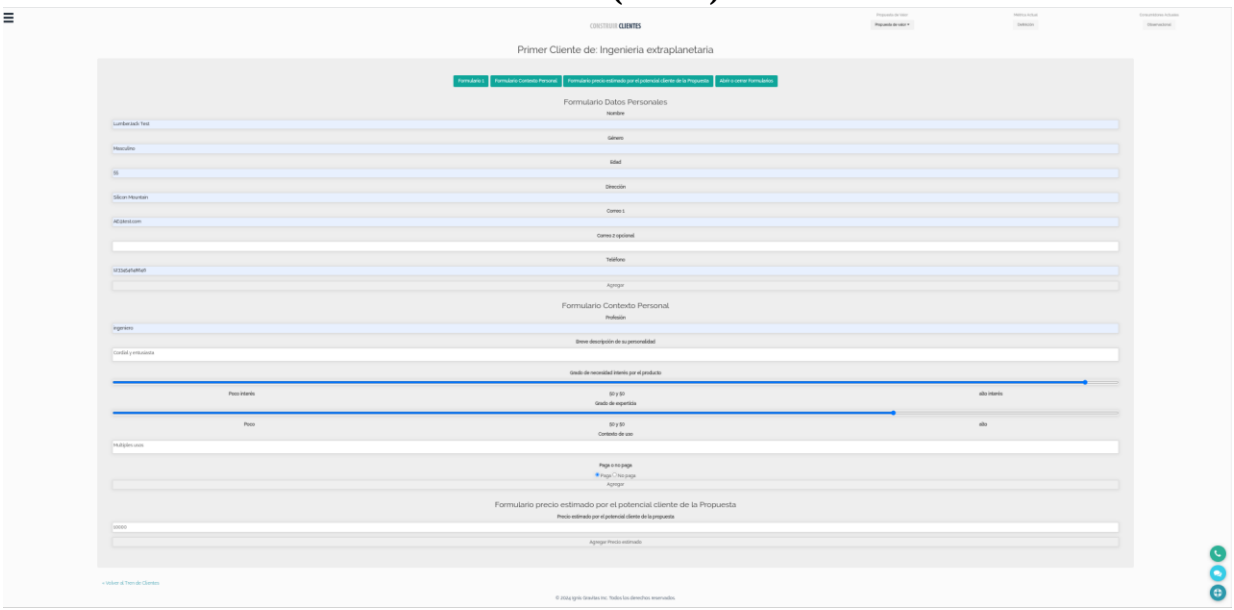
Anexo B.9

Estación Primer Cliente: Vista Principal



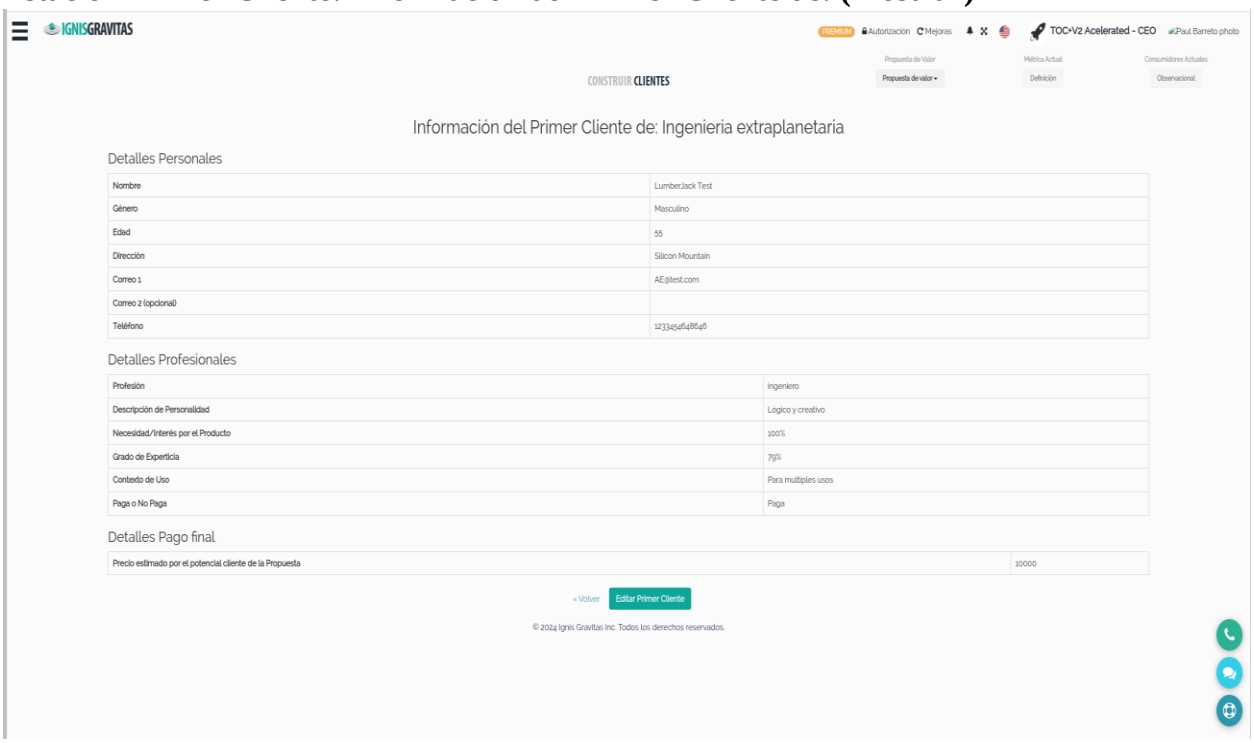
Anexo B.10

Estación Primer Cliente: Primer Cliente de: (Nuevo)



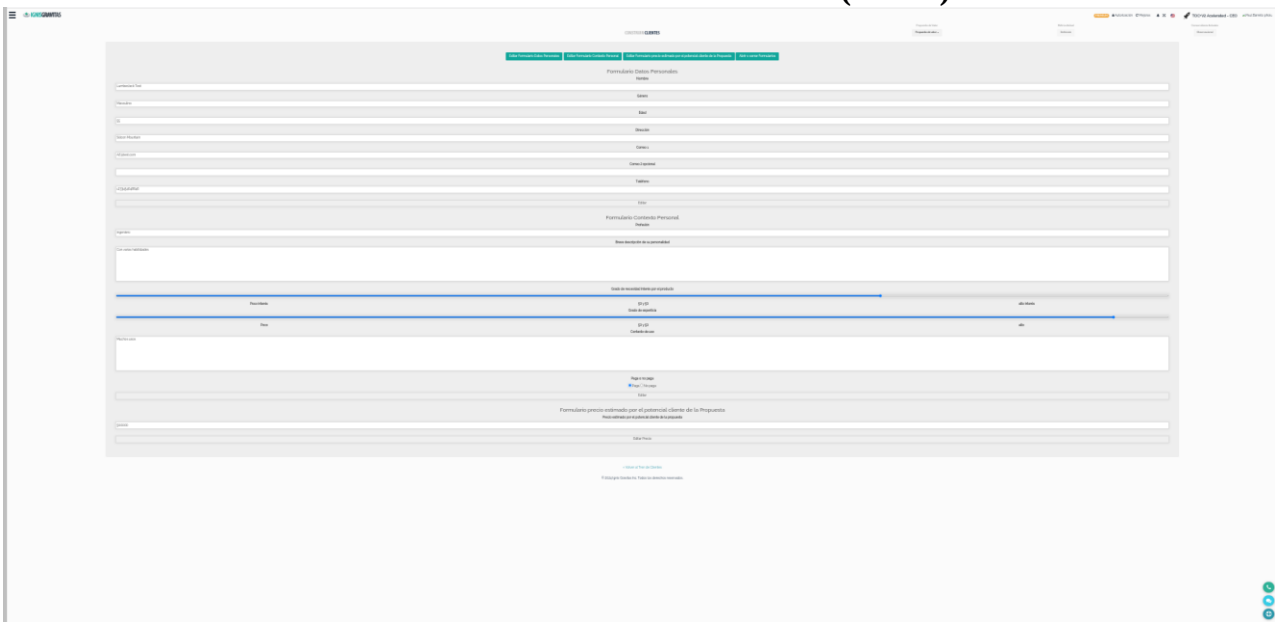
Anexo B.11

Estación Primer Cliente: Información del Primer Cliente de: (Mostrar)



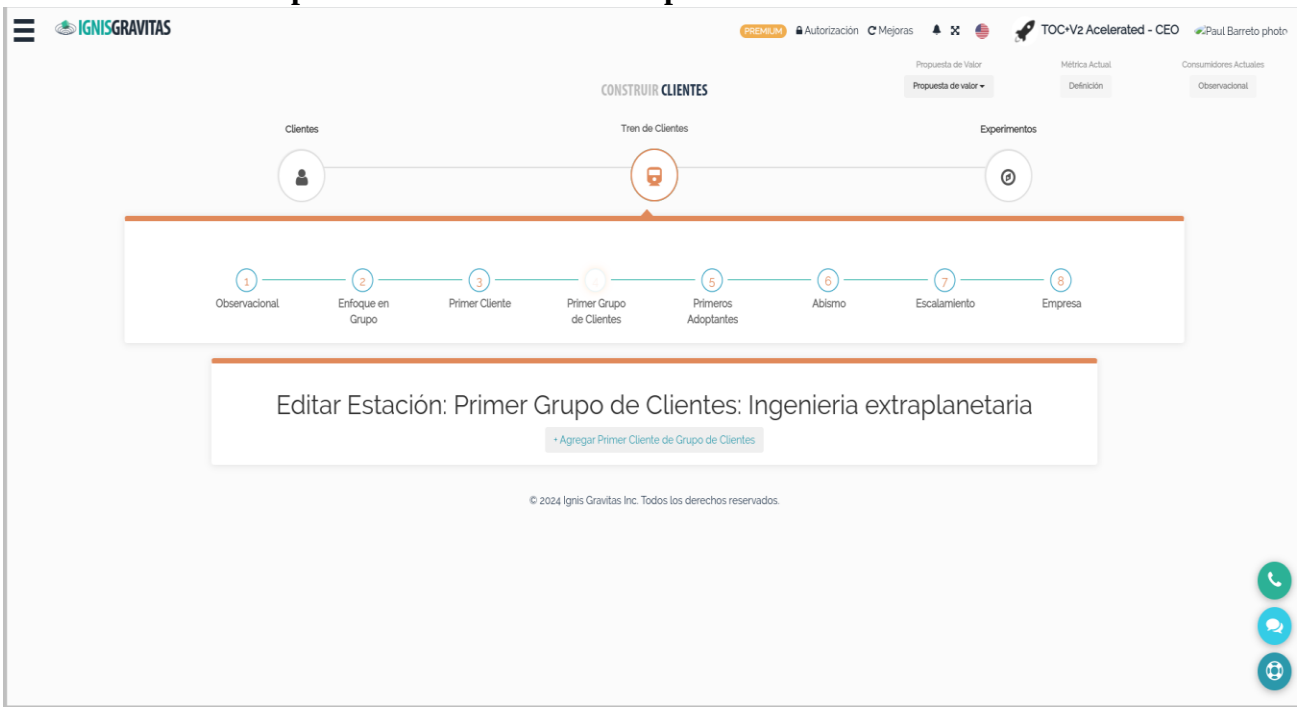
Anexo B.12

Estación Primer Cliente: Información del Primer Cliente de: (Editar)



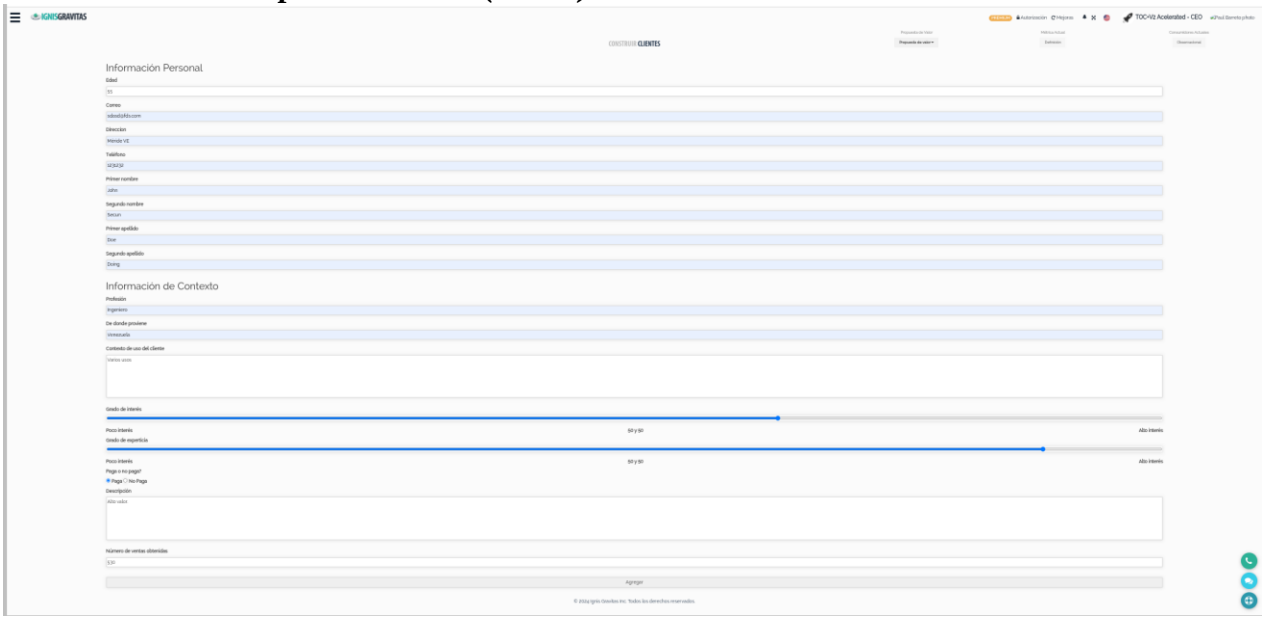
Anexo B.13

Estación Primer Grupo de Clientes: Vista Principal



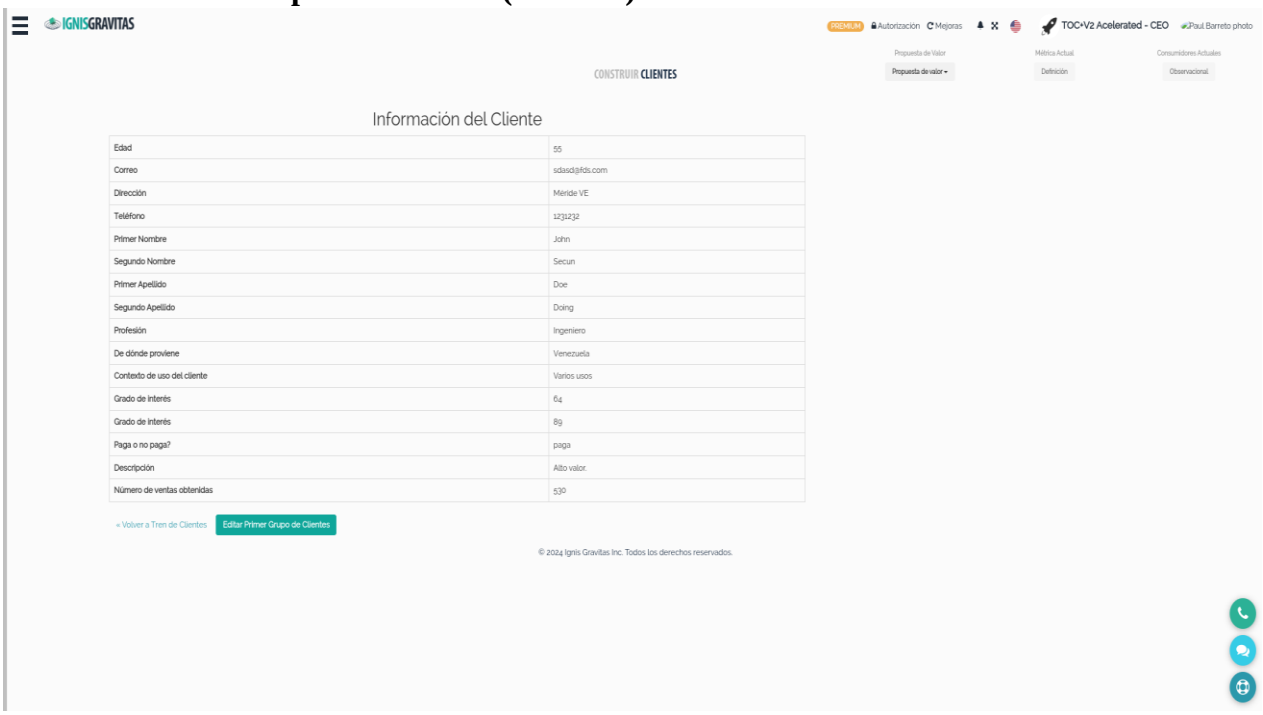
Anexo B.14

Estación Primer Grupo de Clientes: (Nuevo)



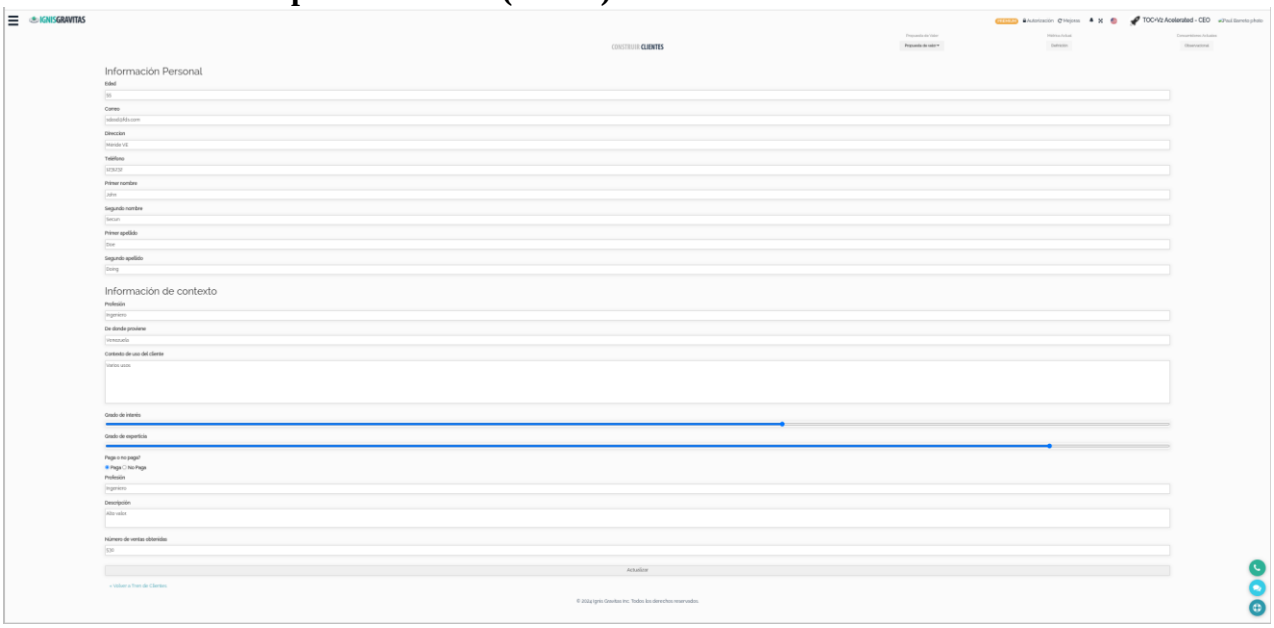
Anexo B.15

Estación Primer Grupo de Clientes: (Mostrar)



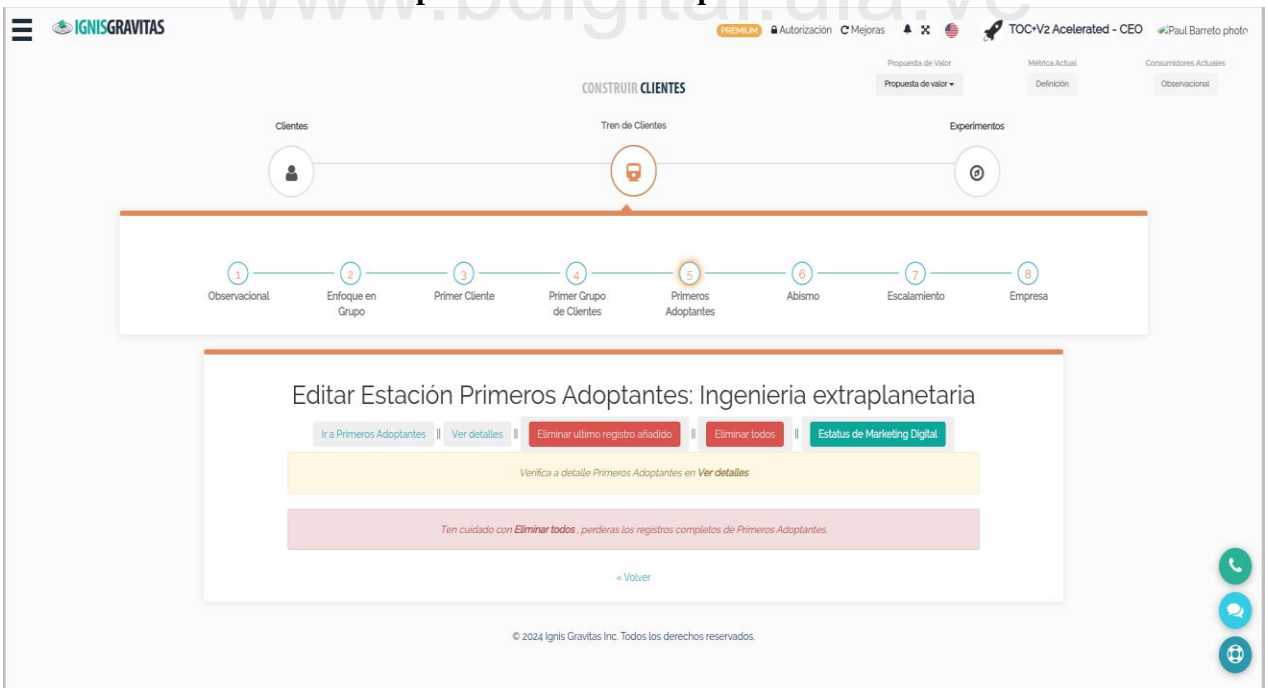
Anexo B.16

Estación Primer Grupo de Clientes: (Editar)



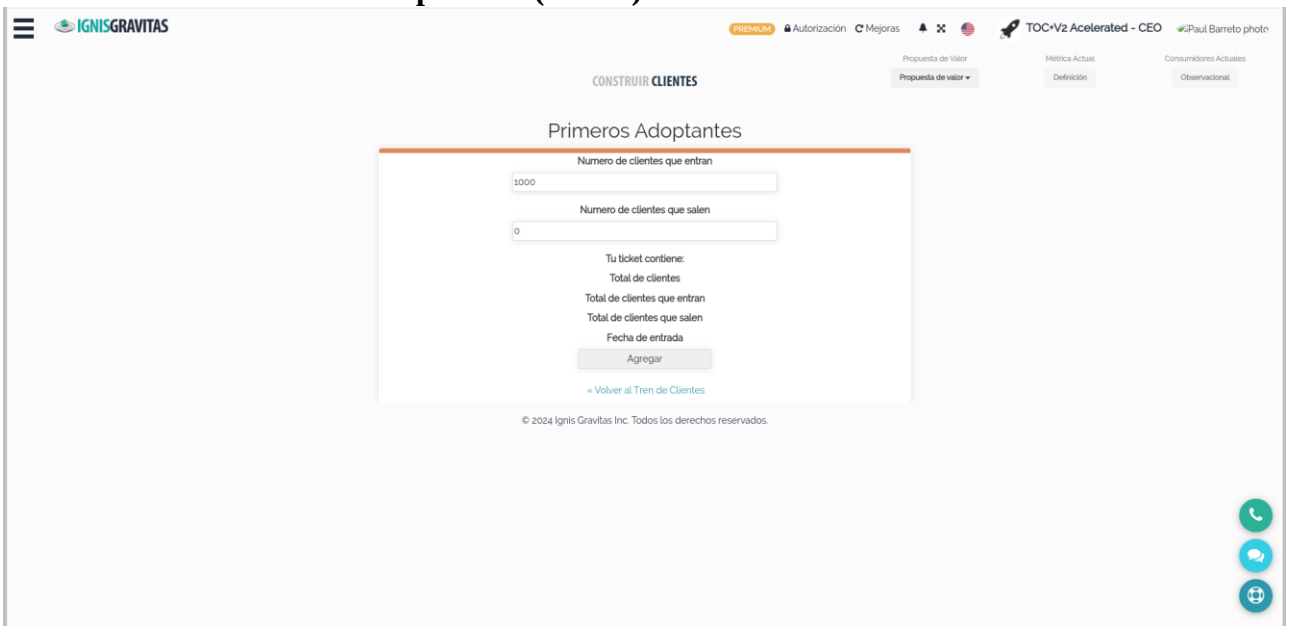
Anexo B.17

Editar Estación Primeros Adoptantes: Vista Principal



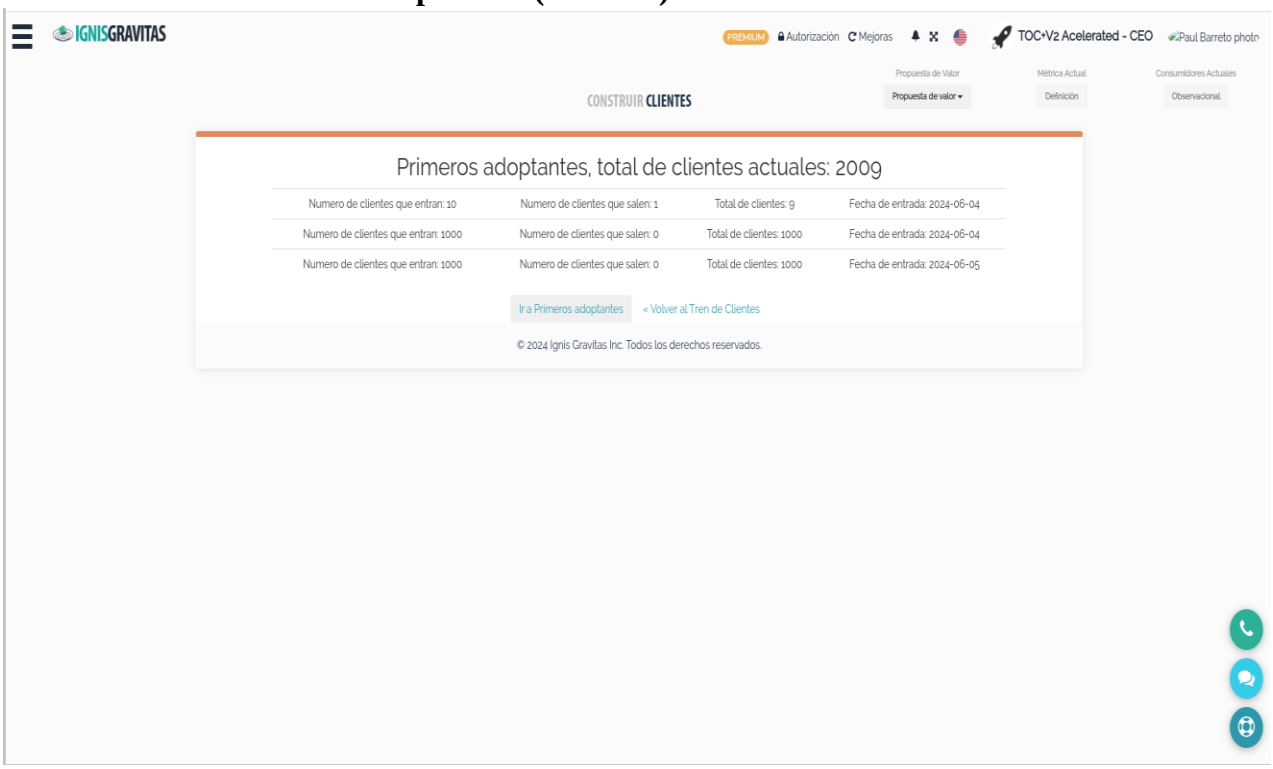
Anexo B.18

Editar Estación Primeros Adoptantes: (Nuevo)



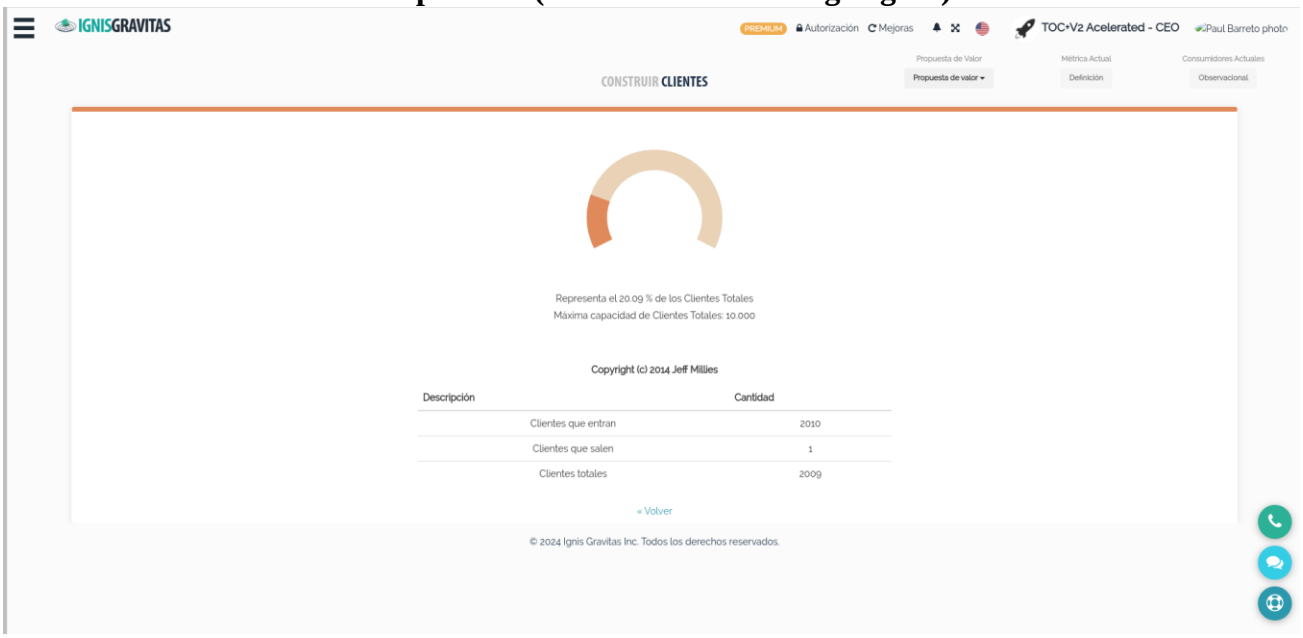
Anexo B.19

Editar Estación Primeros Adoptantes: (Mostrar)



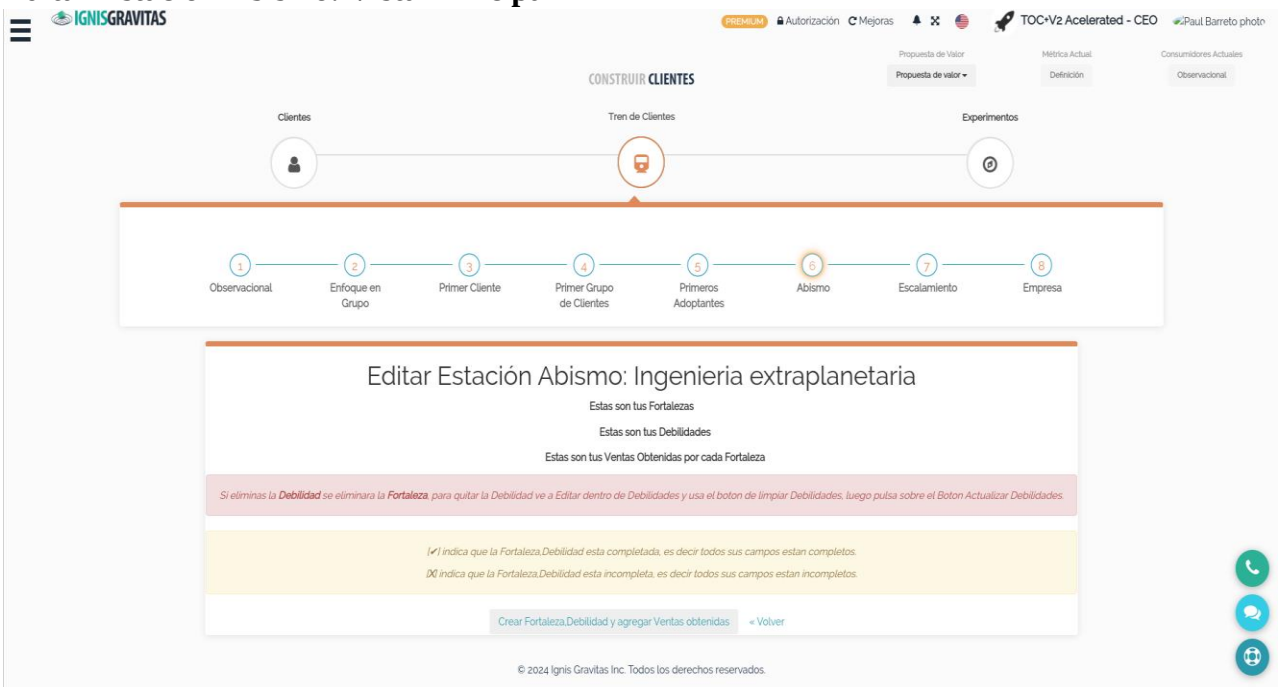
Anexo B.20

Editar Estación Primeros Adoptantes: (Estatus de Marketing Digital)



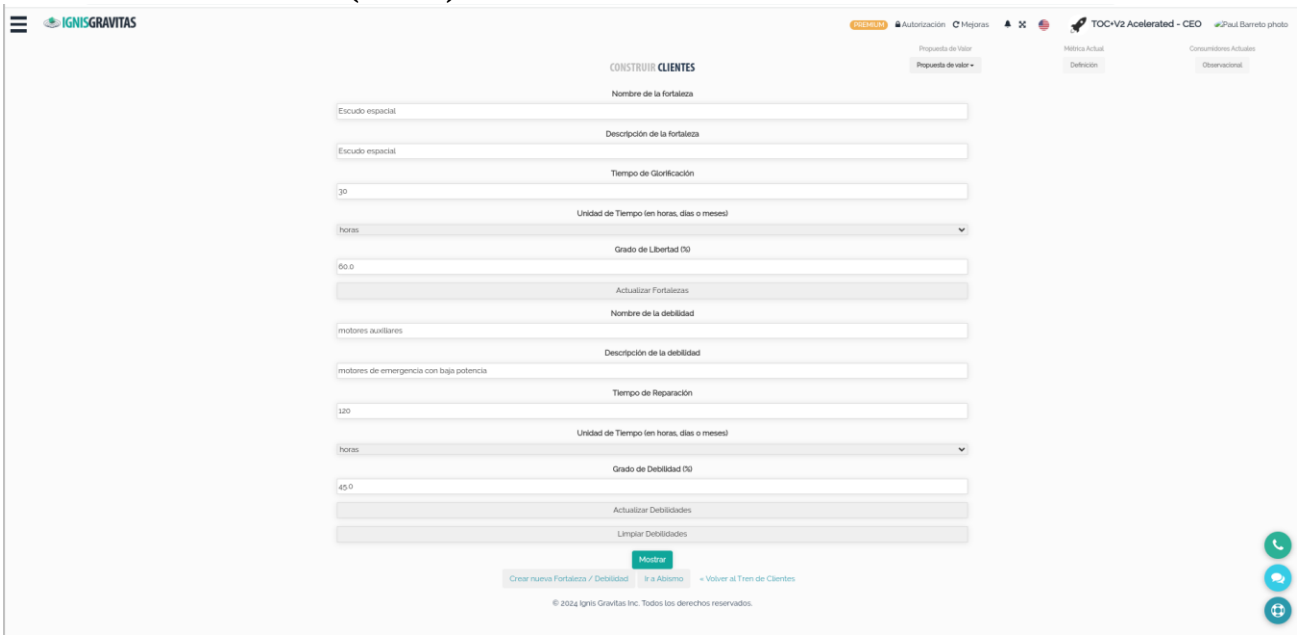
Anexo B.21

Editar Estación Abismo: Vista Principal



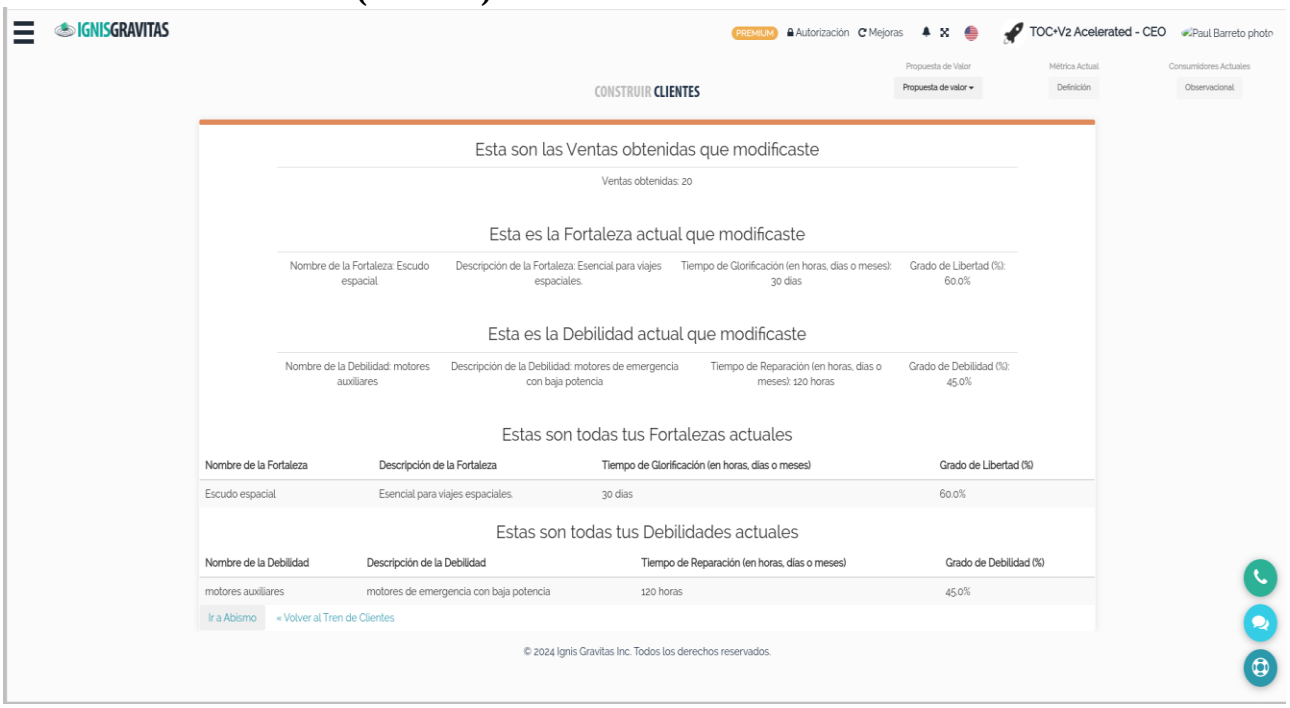
Anexo B.22

Editar Estación Abismo: (Nuevo)



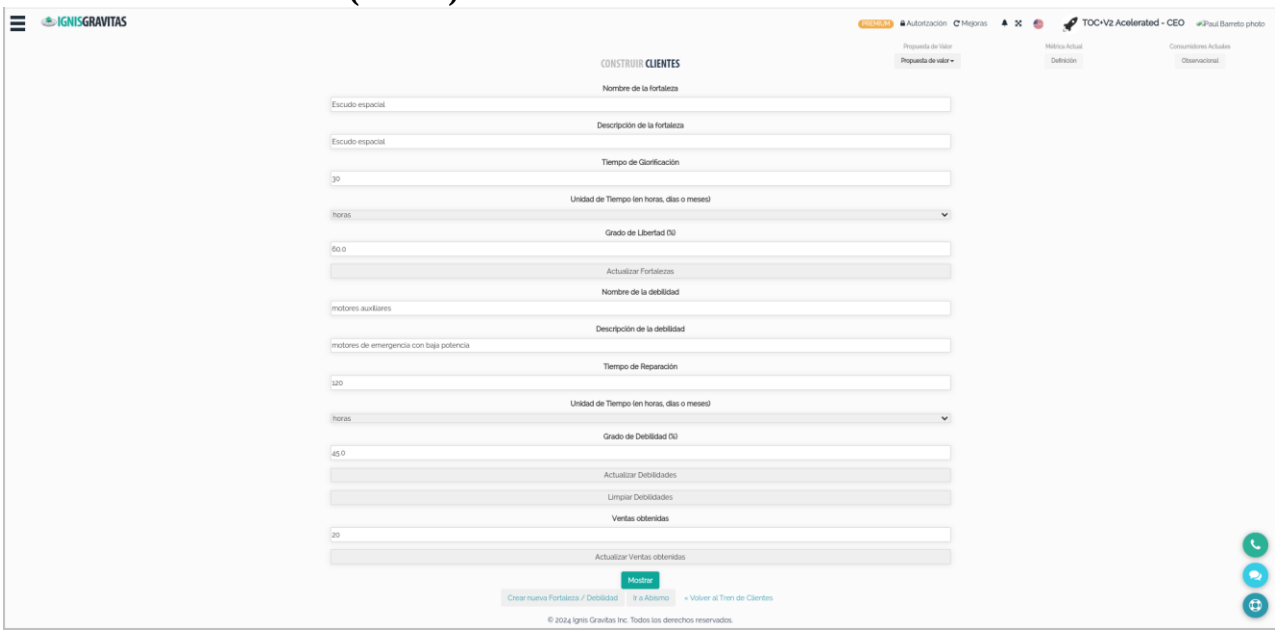
Anexo B.23

Editar Estación Abismo: (Mostrar)



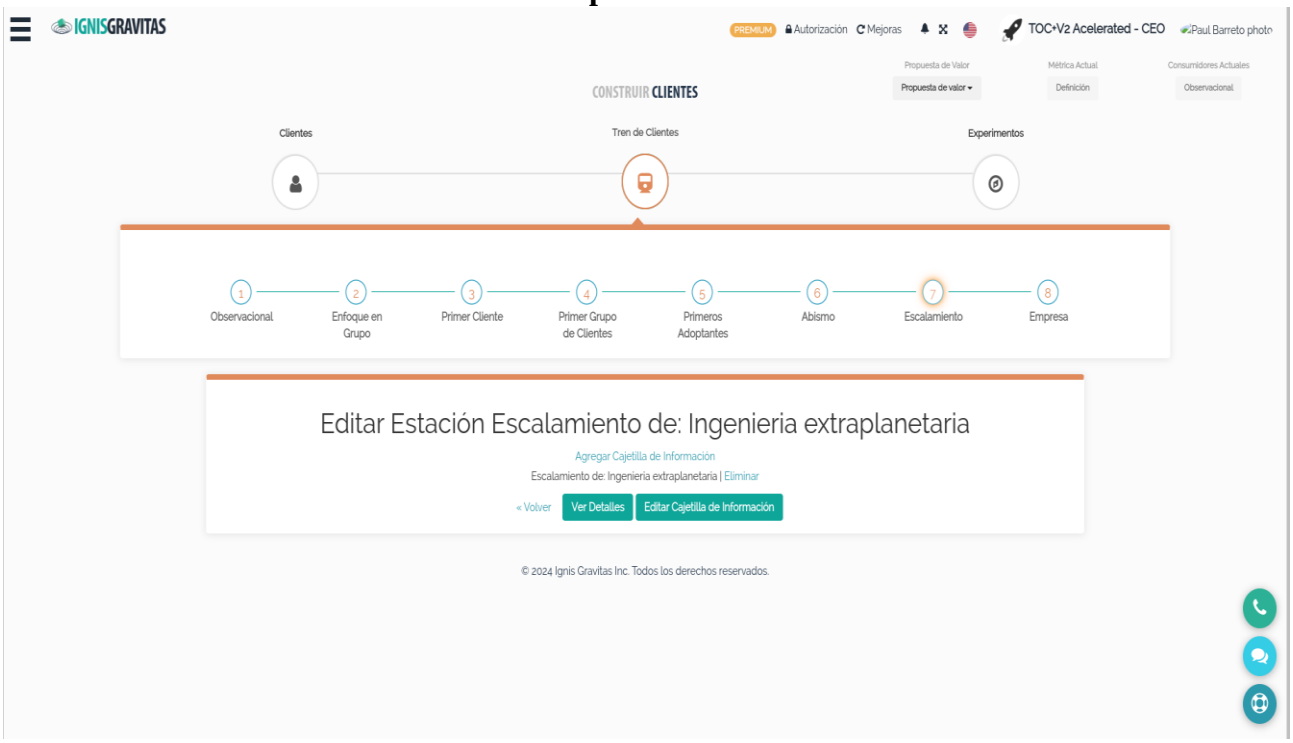
Anexo B.24

Editar Estación Abismo: (Editar)



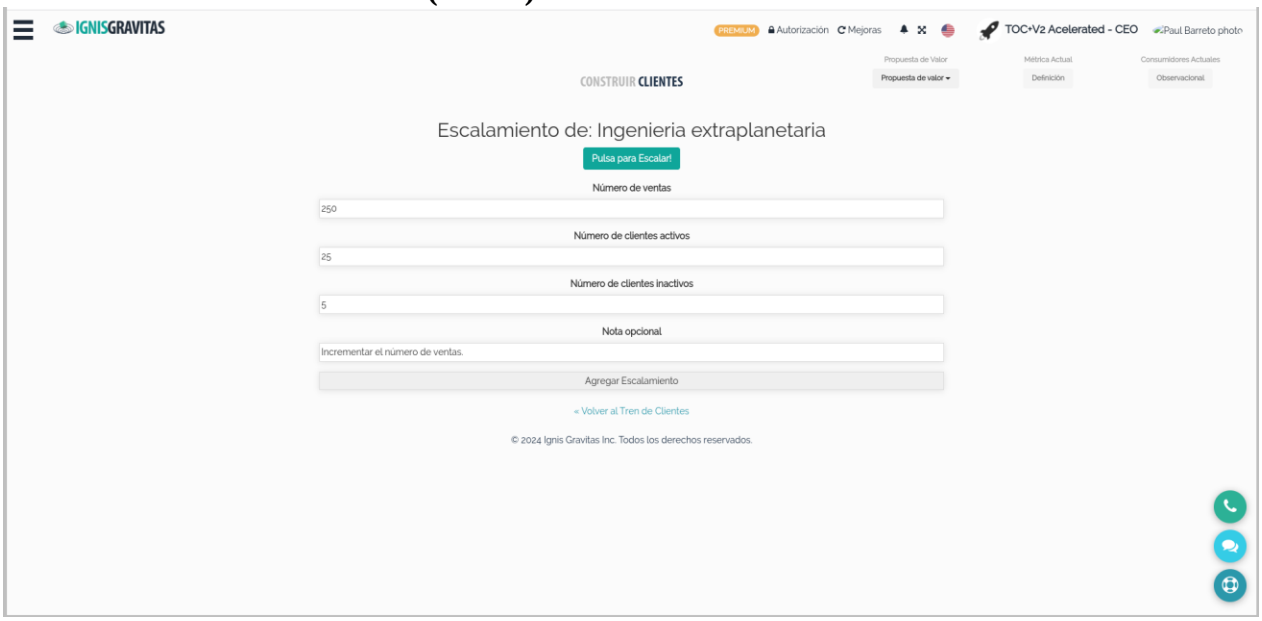
Anexo B.25

Editar Estación Escalamiento: Vista Principal



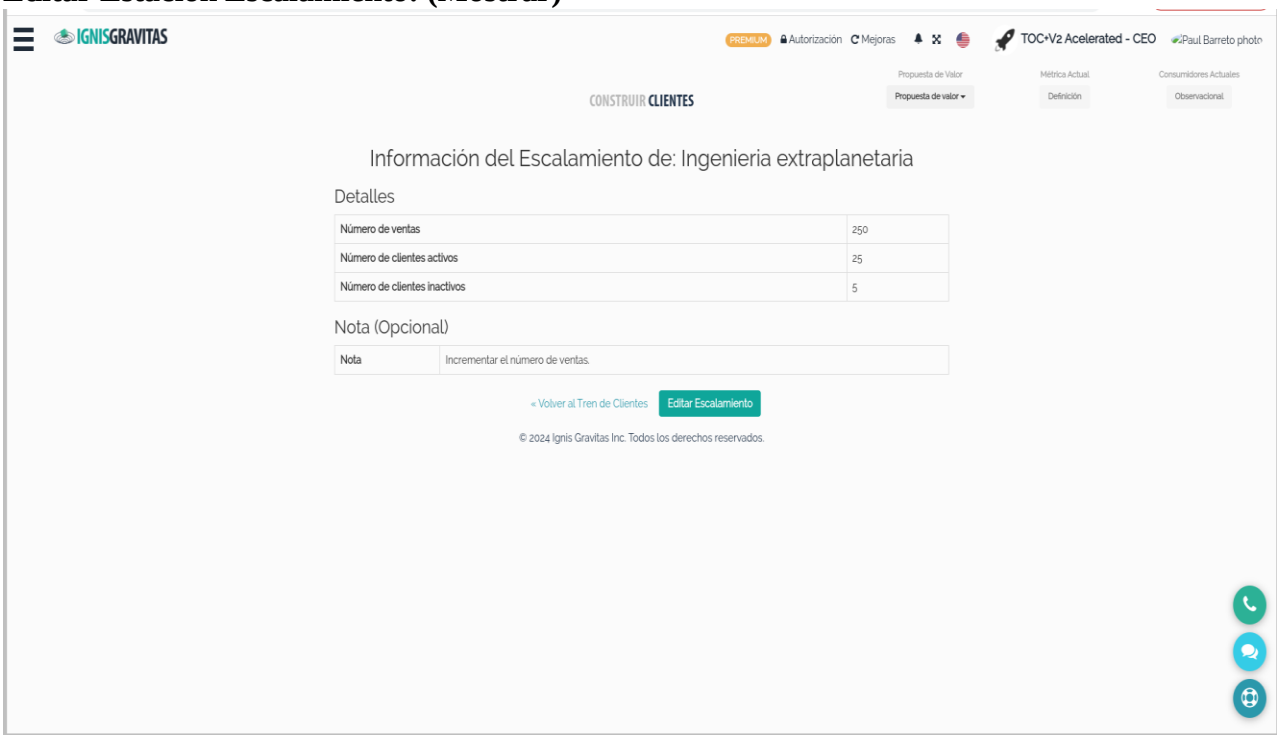
Anexo B.26

Editar Estación Escalamiento: (Nuevo)



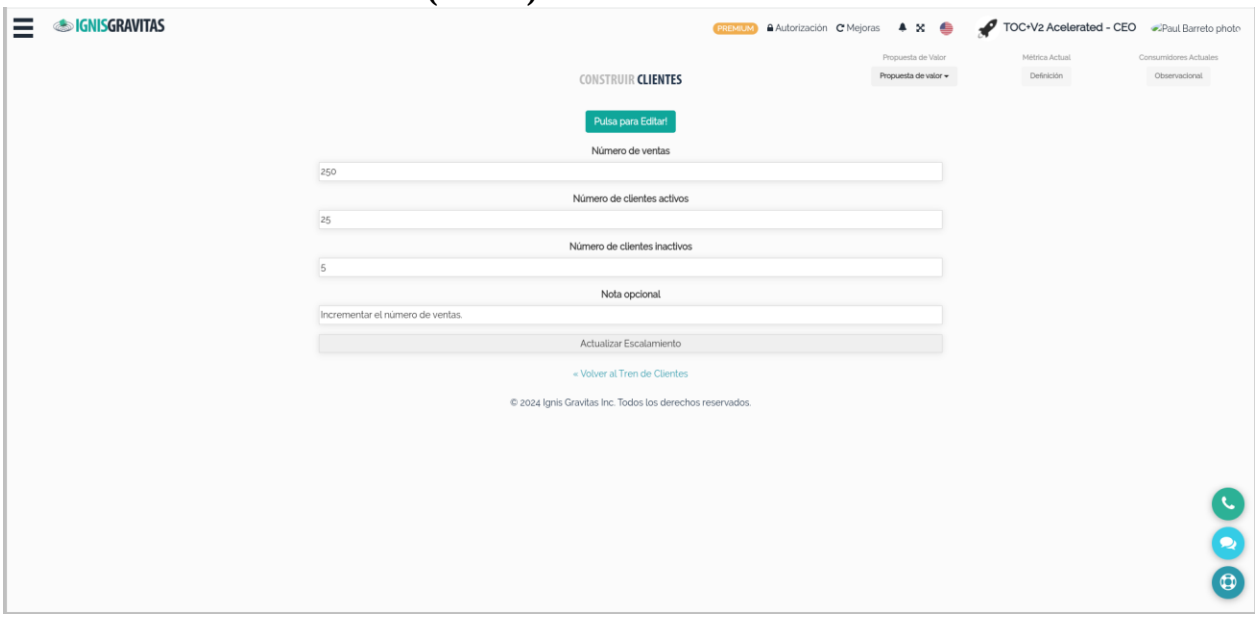
Anexo B.27

Editar Estación Escalamiento: (Mostrar)



Anexo B.28

Editar Estación Escalamiento: (Editar)



Anexo B.29

¡Gracias por usar el Tren de Clientes! (Estación Empresa)