PROYECTO DE GRADO

Presentado ante la ilustre UNIVERSIDAD DE LOS ANDES como requisito parcial para obtener el Título de INGENIERO DE SISTEMAS

# DISEÑO E IMPLEMENTACIÓN DE UNA APLICACIÓN WEB PARA EL CONTROL DE SUMINISTRO DE COMBUSTIBLE.



Por

Br. Gabriel Alejandro Quintero Uza

Tutor: Dr. Gerard Páez Monzón

Febrero 2021

# Diseño e Implementación de una Aplicación Web para el Control de Suministro de Combustible

Br. Gabriel Alejandro Quintero Uza

Proyecto de Grado — Sistemas Computacionales, 71 páginas
Escuela de Ingeniería de Sistemas, Universidad de Los Andes.

**Resumen:** Este trabajo presenta los procesos de diseño e implementación de una aplicación web, que permite servir como plan de contingencia ante la escasez de combustible que se presenta en Venezuela, dada la naturaleza del sistema, se busca minimizar los tiempos de aprendizaje y uso de la aplicación por parte de los usuarios, por ello se considera de gran importancia que el mismo posea una interfaz intuitiva y de fácil acceso. El sistema se realizó siguiendo la metodología White WATCH, y se implementó haciendo uso del framework Laravel de PHP para sustentar el desarrollo de las funciones. Esta aplicación web permite a los propietarios y conductores de automóviles realizar solicitudes de combustible desde cualquier ubicación con acceso a internet, y esperar a que la gestión automatizada de combustible le notifique cuando puede dirigirse a la estación de servicio de su preferencia, buscando mejorar así la calidad de vida de los usuarios y optimizar el proceso de suministro de combustible.



**Palabras clave:** Combustible, Aplicación Web, Laravel, Autogestión

# Índice

# Índice de Tablas

# Índice de Figuras

# Capítulo 1

# Introducción

La gasolina es una mezcla de hidrocarburos obtenida de la destilación fraccionada del petróleo, y se utiliza principalmente como combustible en motores de combustión interna. Estos hidrocarburos se pueden originar a partir de restos de plantas y microorganismos enterrados durante millones de años y sujetos a distintos procesos físicos y químicos. Alrededor del 80% de la energía utilizada para el transporte es proporcionada por combustibles provenientes del refinamiento petrolero y la demanda diaria global de gasolina supera los 4,8 billones de litros.



Venezuela es el país con las mayores reservas probadas de crudo pesado en el mundo, y sin embargo debido a múltiples factores políticos, sociales y económicos, se encuentra envuelta en una gran crisis que ha generado como una de sus consecuencias la escasez de combustible para cubrir la demanda interna. Pero aun en este contexto el estado venezolano mantiene el subsidio del combustible. Según el Fondo Monetario Internacional (FMI) los subsidios totales en el área energética le costaron al gobierno venezolano aproximadamente un 8,9 por ciento del PIB, con subsidios a la gasolina del 7,1 por ciento (Di Bella et al., 2015, p. 10). Esto permite que el precio de la gasolina en el país continúe siendo el más económico del mundo, y una reducción substancial del subsidio parece políticamente inviable, incluso en momentos de gran inestabilidad económica y fiscal.

A medida que la tecnología ha ido avanzando se han podido desarrollar diferentes alternativas buscando el beneficio de la sociedad, mediante aplicaciones que facilitan la administración de tareas específicas, permitiendo así economizar recursos y tiempo, a partir de esto surgen metodologías que orientan el desarrollo rápido de aplicaciones a través de Frameworks los cuales han mostrado ser herramientas útiles para dar apoyo al proceso de construcción de software, debido a que impulsan la

reutilización del código, al prescribir y soportar una arquitectura estandarizada que garantiza su mantenibilidad.

## 1.1 Antecedentes

Durante la última década el uso de las nuevas tecnologías informáticas se ha convertido en un aspecto imprescindible, ya que ofrecen una gran cantidad de beneficios en la gestión operativa y la automatización de procesos, tanto en las empresas privadas como en las públicas. Purón-Cid. G., Gil-García R., (2013), a través de su artículo: "Análisis de políticas públicas y tecnologías de información: oportunidades y retos para América Latina y el Caribe", realiza una síntesis donde se describe el incremento de usuarios y las grandes oportunidades para la implementación de iniciativas de gobierno electrónico, donde se logra entrever el aumento de la cantidad de usuarios y como este suceso aumenta las posibilidades de desarrollo de iniciativas tecnológicas en américa latina.

Tomando esta consideración, encontramos diferentes modelos de planificación para el abastecimiento de combustible, como es el caso de Santelices R. (2007) de la Universidad de Talca el cual nos describe en su tesis de grado: "Propuesta de un Sistema de Planificación para el Abastecimiento de Combustible.", un estudio donde se estiman las demandas para distintas plantas de aprovisionamiento, utilizando como metodología para el desarrollo de los pronósticos mensuales diferentes modelos estocásticos, dada la escasa información de la demanda.



Distintos trabajos han venido desarrollando aplicaciones web para control del abastecimiento de combustible. Tal es el caso de Domínguez L. (2014) en su trabajo titulado "Sistema de Administración de las Operaciones de Marcación de Combustible en Petroecuador E.P.", donde puede observarse el desarrollo de una aplicación web que traslada la información de los niveles de combustible desde las estaciones de servicio a través de cuadros estadísticos, permitiendo generar reportes sobre los niveles de marcación de los diferentes terminales, a fin de mejorar el control y la toma de decisiones.

En el año 2016, de la misma manera, Carabali, G., Moyano, C., en su proyecto: "Aplicación para la Gestión de Abastecimiento de Combustible que Brinda la Empresa distribuidora Levox a gasolineras Mobil.", desarrolla una aplicación que permite gestionar información relevante para los procesos de

logística y abastecimiento de combustible de dicha empresa, con la finalidad de contar con información veraz y actualizada al momento de gestionar la asignación de recursos, optimizando así los tiempos en desplazamiento y evitando posibles errores de estimación.

Este proyecto de grado tiene como finalidad desarrollar un software que permita la asignación y distribución de gasolina de manera eficiente, como un mecanismo de contingencia para la escasez del combustible.

## 1.2 Planteamiento del Problema

En Venezuela debido a la política de subsidios y la incapacidad de la estatal petrolera para incrementar la producción de gasolina, se presenta un escenario donde no se puede cubrir la demanda interna de combustible. Como una solución ante esta problemática surge la idea de diseñar e implementar una aplicación que permita la asignación de combustible de manera eficiente.



Mediante este sistema se podrá obtener información, controlar y tomar decisiones de manera eficiente sobre la distribución del combustible, además de permitirles a los usuarios realizar solicitudes para la asignación del mismo, lo que se presume disminuiría considerablemente los tiempos de espera en las estaciones de servicio.

Cabe destacar que el aumento en el uso de tecnologías y acceso a internet propone un escenario favorable para la implementación de dicha aplicación, que además de servir como mecanismo de contingencia ante la escasez de combustible, también deberá permitir que los datos registrados sirvan como fuente de información para su posterior análisis.

## 1.3 Alcance

Se realizará el diseño e implementación de una aplicación web cuyo propósito es servir como plan de contingencia ante la escasez de combustible, optimizando la asignación de recursos y evitando las prolongadas esperas en las estaciones de servicio. Además, se podrá acceder al mismo desde cualquier lugar mediante un dispositivo que cuente con conexión a internet.

A través del sistema los usuarios tendrán la facilidad de realizar solicitudes, verificar estatus de solicitudes, consultar el histórico de solicitudes, especificar información sobre sus vehículos, y recibir información sobre cualquier eventualidad en la distribución del combustible.

Por otra parte, la aplicación pondrá a disposición una serie de reportes donde se especifique información relevante para el análisis y la toma de decisiones con respecto a la distribución del combustible.

## 1.4 Objetivos

### 1.4.1 Objetivo General

El presente proyecto pretende desarrollar un software que permita el control de la distribución de gasolina para de esta manera poder asignar este recurso de manera eficiente, implementando así un mecanismo de contingencia para esta problemática social. Los resultados generados por este trabajo, pueden ser usados posteriormente para ampliar el enfoque del software o planificar el diseño de nuevos mecanismos de contingencia.



### 1.4.2 Objetivos Específicos

En la búsqueda de lograr el objetivo general, se han identificado los siguientes objetivos específicos:

- Identificar y clasificar los requisitos a partir del contexto.
- Implementar la aplicación utilizando un marco de trabajo para el desarrollo de aplicaciones web.
- Diseñar la aplicación que posea una interfaz gráfica intuitiva.
- Crear un sistema de reportes que permita el análisis de los datos de manera centralizada.
- Definir y realizar pruebas para comprobar el correcto funcionamiento del software desarrollado.

## 1.5 Metodología

Para el desarrollo de este proyecto se utilizará el método White WATCH (Montilva, J., Barrios, J., 2010) el cual es definido como: "Un modelo de procesos que balancea la producción de especificaciones de producto que se transforman en la medida que se avanza en el proceso, con la disponibilidad en corto tiempo de versiones parciales y operativas del producto." Además, agregan que "Esta manera de estructurar el marco metodológico permite que la ejecución de los procesos de desarrollo sea cíclica, iterativa y controlada."

Este método describe una serie de actividades para los procesos gerenciales y los procesos técnicos o de desarrollo, estos últimos incluyen la ingeniería de requisitos, el diseño del sistema de software, aprovisionamiento de componentes, implementación, pruebas del sistema, y la entrega del software, cada ciclo de procesos debe producir una nueva versión del sistema debido a su enfoque evolutivo y en cada uno de los ciclos se puede iterar entre los diferentes procesos técnicos a fin de corregir errores, producir nuevos requisitos o mejorar el producto en desarrollo.



Por lo general en las aplicaciones web los requisitos varían a lo largo del tiempo por lo cual la utilización de procesos iterativos y evolutivos podría ser beneficioso para este tipo de proyecto, y dado que se trata de un plan de contingencia, es necesario que el producto esté disponible lo más pronto posible, esta necesidad lleva a plantear los objetivos mediante el uso del método de desarrollo de software White WATCH.

# Capítulo 2

# Marco Teórico

Este capítulo realiza una revisión sobre los principales conceptos teóricos que sirven como base para el trabajo a realizar, partiendo desde los aspectos más generales, y mediante la recopilación de ideas y definiciones de diferentes autores culminar con una descripción detallada sobre los aspectos más relevantes del entorno en el cual se desarrollará el proyecto.

## 2.1 Generalidades y conceptos básicos.



En Venezuela existe un déficit de combustibles; como consecuencia no se logra satisfacer la demanda interna considerando la producción de las refinerías locales, sumado a la imposibilidad de importar gasolina para satisfacer la demanda. El precio de comercialización es subsidiado por el estado venezolano teniendo entonces la gasolina más económica del mundo.

### 2.1.1 Gasolina

La gasolina se obtiene a través de la refinación y tratamiento del crudo, también conocido como petróleo. La composición química del crudo permite que se puedan manipular sus átomos y moléculas para producir un sinnúmero de derivados, entre ellos la gasolina. Sin embargo, debe ser sometida a un proceso de tratamiento ya que, en estado natural, su nivel de calidad y octanaje es muy bajo.

Por ello, la ciencia ha desarrollado procesos de refinamiento y mejoramiento para poder comercializar una gasolina de calidad, Además, el desarrollo de nuevas tecnologías; como también el desarrollo automotriz, apuntan al mejoramiento de gasolinas que gocen de un mayor índice de octanaje y sean amigables con el medio ambiente. Es decir, libres de elementos contaminantes.

## 2.2 Bases de Datos

Una base de datos se puede considerar como un modelo de la realidad, es la representación integrada de los conjuntos de entidades del sistema de información y de sus interrelaciones. Esta representación informática o conjunto estructurado de datos debe poder ser utilizada de forma compartida por muchos usuarios de distintos tipos (Silberschatz, A., Korth, H., and S, S., 2002). Dicho de otro modo, una base de datos es un conjunto de información relacionada que se encuentra agrupada o estructurada.

### 2.2.1 Bases de Datos Relacionales

Según (Silberschatz, A., Korth, H., and S, S., 2002) Una base de datos relacional consiste en un conjunto de tablas o repositorio compartido de datos, donde a cada tabla se le asigna un nombre exclusivo. Cada tabla tiene una estructura donde cada fila de la tabla representa una relación entre un conjunto de valores.

Para hacer disponibles los datos de una base de datos relacional a los usuarios hay que considerar varios aspectos, como la forma en que los usuarios solicitan los datos, la integridad de datos y la seguridad. Es por ello que el modelo relacional se ha establecido actualmente como el principal modelo de datos para las aplicaciones de procesamiento de datos.



### 2.2.2 Sistemas de Gestión de Base de Datos

Los Sistemas de gestión de base de datos son un tipo de software muy específico, dedicado a servir de interfaz entre la base de datos, el usuario y las aplicaciones que la utilizan. Se compone de un lenguaje de definición de datos, un lenguaje de manipulación de datos y de un lenguaje de consulta.

El uso de sistemas de gestión de base de datos nos brinda una gran cantidad de beneficios como el control sobre la redundancia de los datos, lo que nos permite mejorar la consistencia e integridad de los mismos, además gestionan el acceso simultáneo y garantizan que no ocurran problemas de concurrencia. Pero una de las principales razones de usar sistemas de gestión de bases de datos es tener un control centralizado tanto de los datos como de los programas que acceden a ellos.

## 2.3 UML (Unified Modeling Language)

UML es un lenguaje de modelado visual de propósito general orientado a objetos para visualizar, especificar, construir y documentar partes de un sistema software desde distintos puntos de vista pueden usarse con cualquier proceso de desarrollo, a lo largo de todo el ciclo de vida y puede aplicarse a todos los dominios de aplicación y plataformas de implementación.

UML 2.0 define trece tipos de diagramas, divididos en tres categorías:

1. **Los diagramas de estructura:** incluyen el diagrama de clase, el diagrama de objetos, el diagrama de componentes, el diagrama de estructura compuesta, el diagrama de paquetes y el diagrama de implementación.
2. **Los diagramas de comportamiento**: incluyen el diagrama de casos de uso, diagrama de actividad y diagrama de máquina de estado.
3. **Los Diagramas de interacción**: incluyen el diagrama de secuencia, el diagrama de comunicación, el diagrama de tiempo y el diagrama general de interacción.



A continuación, definiremos los diagramas en los cuales haremos énfasis para la elaboración de este trabajo:

### 2.3.1 Diagrama de Clases

Es una colección de elementos de modelado estáticos como clases, tipos, sus contenidos y relaciones que describen la vista estática de un sistema. Son los diagramas más comunes en el análisis y diseño de un sistema ya que permiten explorar conceptos del dominio, analizar requisitos y describir el diseño detallado de un software.

### 2.3.2 Diagrama de Casos de Uso

Los diagramas de casos de uso describen el comportamiento de un software o componentes a partir del punto de vista del usuario, identificando roles de cada uno de los usuarios del sistema y los servicios o funciones provistas por el sistema para estos.

## 2.4 Método de desarrollo White WATCH

Es un marco de trabajo que describe el conjunto de actividades necesarias para desarrollar un producto de software, tomando como premisa la disminución en el detalle de especificaciones de apoyo parcial, lo que ofrece al equipo de desarrollo el cual no debe ser mayor a dos (02) personas más tiempo para dedicarle a las actividades de implementación de versiones operativas y evolutivas del producto.

El modelo de procesos del método inspirado en la metáfora del reloj de pulsera, organiza los procesos técnicos en forma circular y los procesos gerenciales en el centro lo que permite que la ejecución de los procesos de desarrollo se lleve a cabo de manera cíclica, iterativa y controlada. Cada uno de los ciclos produce una versión del sistema, sin embargo, en cada ciclo se puede iterar entre los procesos técnicos a fin de corregir errores, o mejorar el producto.

### 2.4.1 Procesos Gerenciales

Son los encargados de describir las actividades que el líder del proyecto debe realizar, como la planificación, organización y control del desarrollo del proyecto, además de asegurar la calidad del sistema mediante las validaciones pertinentes y gestionar los cambios en las especificaciones del proyecto.



### 2.4.2 Procesos Técnicos

Estos describen los pasos a seguir para la elaboración de un producto de software, contemplan actividades como la ingeniería de requisitos, el diseño de software, el aprovisionamiento de componentes, la implementación del software y las pruebas. Cada uno de los procesos de este modelo se asocia con pasos o subprocesos que indican de manera detallada el conjunto de acciones a ejecutar.

Teniendo presente el enfoque evolutivo del método, cada una de las versiones deberá ser el resultado del refinamiento de la versión previa, fundamentándose en la reutilización de componentes como medio para acortar el tiempo total del desarrollo, y el uso de la notación UML para la elaboración y mantenimiento de la documentación técnica del proyecto.


**Figura 2.1: Modelo de procesos del método White Watch**



## 2.5 Modelo Vista Controlador (MVC)

El patrón Modelo-Vista-Controlador (MVC) surge con el objetivo de reducir el esfuerzo de programación, necesario en la implementación de sistemas múltiples y sincronizados de los mismos datos, a partir de estandarizar el diseño de las aplicaciones. Fue descrito por primera vez en 1979 por Trygve Reenskaug. Una de sus características principales está dada por el hecho de que, el modelo, las vistas y los controladores se tratan como entidades separadas; esto hace que cualquier cambio producido en el modelo se refleje automáticamente en cada una de las vistas. MVC está demostrando ser un patrón de diseño bien elaborado pues las aplicaciones que lo implementan presentan una extensibilidad y una mantenibilidad únicas comparadas con otras aplicaciones basadas en otros patrones.

### 2.5.1 Modelo

Es un conjunto de clases que representan la información del mundo real que el sistema debe reflejar. Es la parte encargada de manejar los datos y controlar todas sus transformaciones. El modelo no tiene conocimiento específico de los controladores o de las vistas, ni siquiera contiene referencias a ellos. Es

el propio sistema el que tiene encomendada la responsabilidad de mantener enlaces entre el modelo y sus vistas, y notificar a las vistas cuando cambia el modelo.

### 2.5.2 Vista

Son las encargadas de la representación de los datos contenidos en el modelo al usuario. La relación entre las vistas y el modelo son de muchas a uno, es decir cada vista se asocia a un modelo, pero pueden existir muchas vistas asociadas al mismo modelo. Todo esto se realiza mediante la interacción con el controlador.

### 2.5.3 Controlador

Es el encargado de interpretar y dar sentido a las órdenes del usuario, realizando acciones sobre los datos representados por el modelo. Centra toda la interacción entre la vista y el modelo. Cuando se realiza algún cambio, entra en acción, bien sea por cambios en la información del modelo o por alteraciones de la vista.



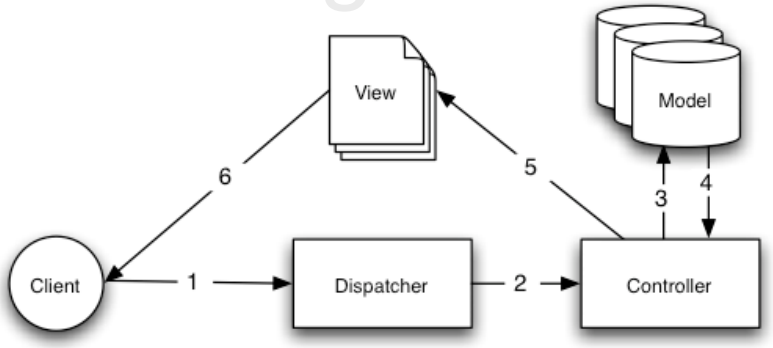



**Figura 2.2: Modelo Vista Controlador (MVC)**

## 2.6 Framework

Un framework, es el esquema o estructura que se establece y que se aprovecha para desarrollar y organizar un software determinado. Esta definición, podría explicarse como el entorno pensado para hacer más sencilla la programación de cualquier aplicación o herramienta. Este sistema plantea varias

ventajas para los programadores, ya que automatiza muchos procesos y además facilita el conjunto de la programación. Es útil, para evitar el tener que repetir código para realizar funciones habituales en un rango de herramientas. Un framework sirve para poder escribir código o desarrollar una aplicación de manera más sencilla. Es algo que permite una mejor organización y control de todo el código elaborado, así como una posible reutilización en el futuro. Debido a esto, garantiza una mayor productividad que los métodos más convencionales y una minimización del coste al agilizar las horas de trabajo volcadas en el desarrollo.

## 2.7 PHP: Hipertext Preprocessor

PHP es un lenguaje de script de propósito general adecuado para el desarrollo web, el cual es de código abierto y puede ser incrustado en HTML. En lugar de usar muchos comandos para mostrar HTML como en otros lenguajes de programación las páginas de PHP contienen HTML con código PHP incrustado el cual está encerrado entre las etiquetas especiales de comienzo y final. Lo que distingue a PHP es que el código es ejecutado del lado del servidor, generando HTML y enviándolo al cliente, por lo que el cliente recibe el resultado del script, pero no sabrá cuál era el código.



El lenguaje fue creado originalmente por Rasmus Lerdorf en el año 1995, y actualmente el lenguaje sigue siendo desarrollado con nuevas funciones por el grupo PHP. Es destacable el hecho de que PHP puede emplearse en casi todos los sistemas operativos, admitiendo la mayoría de los servidores web usados hoy en día, y brindado soporte para una gran variedad de bases de datos. Además de estas características también cuenta con la posibilidad de utilizar programación por procedimientos o programación orientada a objetos.

## 2.8 Laravel

Laravel es un framework de código abierto para desarrollar aplicaciones y servicios web con PHP, con sintaxis expresiva y elegante, el cual tiene como objetivo facilitar las tareas comunes utilizadas en la mayoría de los proyectos web. Fue creado en 2011 por Taylor Otwell, Está basado en el patrón de diseño Modelo Vista Controlador (MVC) desarrollado para incrementar la productividad del programador y reducir las barreras a la programación de aplicaciones web.

### 2.8.1 Características

El uso de un framework como Laravel nos ofrece además de integración, escalabilidad y facilidad de mantenimiento ciertas características que nos suponen una ventaja a la hora de desarrollar una aplicación web como lo son:

- Incluye un ORM (Mapeo Objeto-Relacional) llamado ***Eloquent***, el cual provee una forma de mapear los datos que se encuentran almacenados en la base de datos sin necesidad de usar lenguaje SQL dentro de las clases de PHP. Sin embargo, también se pueden hacer consultas directas a la base de datos mediante el uso de ***Query Builder***.
- Utiliza un motor de plantillas para las vistas llamado ***Blade***, el cual permite usar estructuras de control y variables de PHP de manera más simple y elegante, además de modularizar las vistas mediante la extensión y uso de plantillas o secciones creadas en otras vistas.
- Proporciona un método muy simple y expresivo para definir las rutas, las cuales permiten determinar que función de un controlador se desea ejecutar.
- Facilita el manejo de la base de datos mediante un sistema de migraciones, el cual permite un mayor control de las versiones de la base de datos y provee portabilidad para diferentes gestores usando el mismo código PHP.
- Incorpora una interfaz de línea de comandos llamada ***Artisan***, la cual es un medio de interacción que proporciona un conjunto de comandos los cuales facilitan la realización de diferentes tareas durante el desarrollo de la aplicación.

# Capítulo 3

# Ingeniería de Requisitos

Este capítulo contempla las primeras fases del proceso de desarrollo definidas por el método White WATCH como lo son el descubrimiento, análisis y especificación de los requisitos del sistema. Este conjunto de procesos técnicos detalla las características que debe tener la aplicación, proporcionando una base que será usada posteriormente en el desarrollo del sistema. Se identificaron las necesidades, y se definieron los requerimientos funcionales y no funcionales del sistema.



## 3.1 Requisitos No Funcionales

Los requerimientos no funcionales representan algunas propiedades generales que la aplicación debe tener, tales como la capacidad de uso, el desempeño, las restricciones de desarrollo, entre otros aspectos. Para este proyecto se definieron los siguientes:

- Se utilizará el lenguaje de script PHP, a través del framework Laravel.
- Los estilos visuales deben ser amigables e intuitivos al usuario, mostrando mensajes de error que sean informativos.
- El sistema debe asegurar que los datos estén protegidos del acceso no autorizado.
- El acceso a las diferentes funciones del sistema vendrá dado según el rol que cada uno de los usuarios tenga asignado.
- El uso de las funcionalidades del sistema deberá estar documentada mediante manuales que faciliten la operatividad por parte de los usuarios.

## 3.2 Requisitos Funcionales

Con el objetivo de generar especificaciones que describan con claridad y de manera consistente los servicios que el sistema debe proporcionar, Se observaron las actividades que realizaban los usuarios, y mediante su posterior análisis, se definieron una serie de requerimientos que luego fueron validados a través de las iteraciones y serán presentados en el presente proyecto como casos de uso.

Los casos de uso son una descripción de la forma particular en que el sistema es empleado por los usuarios para alcanzar sus objetivos, permitiendo así definir los límites del mismo y como se relaciona con su entorno. Surgen de la necesidad de observar el comportamiento del sistema desde el punto de vista del usuario, bajo la forma de acciones y reacciones.

### 3.2.1 Registrarse

Para ingresar al sistema por primera vez cualquier usuario deberá ingresar sus datos personales como nombre, apellido, correo y contraseña. Esto le permitirá a cualquier invitado registrarse en la base de datos. En la tabla 3.1 se define este requerimiento



**Tabla 3.1: Caso de uso: Registrarse**

| | |
|---|---|
| **Nombre:** | Registrarse |
| **Descripción:** Le permite a cualquier invitado registrarse en la base de datos | |
| **Actores:** | Invitado |
| **Precondiciones:** | Ninguna |
| **Flujo Normal:** | 1. El usuario selecciona la opción "Registrarse" en la página de inicio<br>2. El usuario rellena sus credenciales en el formulario de registro, y pulsa "Registrar" |
| **Flujo Alternativo:** | Si los datos no son correctos o son duplicados en la base de datos, se validará el formulario y se mostrará un mensaje de error. |
| **Postcondiciones:** | Un nuevo usuario con sus correspondientes campos es creado en la base de datos de la aplicación y este es redirigido a la pantalla principal. |
| **Notas:** | Ninguna |

### 3.2.2 Iniciar sesión

Luego de estar registrado en el sistema cada vez que se desea acceder al mismo, es necesario que el usuario del sistema ingrese las credenciales correspondientes para cambiar el rol de invitado por el de usuario registrado. En la tabla 3.2 se define este requerimiento y en la figura 3.1 se visualiza el diagrama de caso de uso.

**Tabla 3.2: Caso de uso: Iniciar sesión**

| | |
|---|---|
| **Nombre:** | Iniciar Sesión |
| **Descripción:** Le permite a un usuario no autenticado ingresar al sistema. | |
| **Actores:** | Invitado |
| **Precondiciones:** | 1. El usuario debe estar registrado en el sistema. |
| **Flujo Normal:** | 1. El usuario llena los campos con sus credenciales.<br>2. Pulsa el botón Iniciar Sesión. |
| **Flujo Alternativo:** | Si las credenciales no son correctas, el sistema devuelve error. |
| **Postcondiciones:** | El Invitado cambia de rol y pasa a ser usuario registrado en el sistema. Siendo es redirigido a la pantalla principal. |
| **Notas:** | Ninguna |



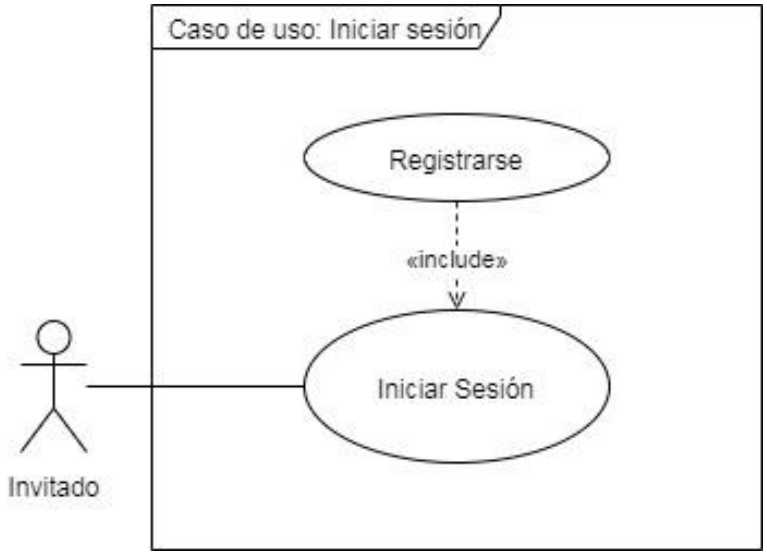



**Figura 3.1: Diagrama de caso de uso: Iniciar sesión**

### 3.2.3 Recuperar contraseña

Un usuario registrado en el sistema debe poder recuperar sus credenciales en caso de olvidar alguna de ellas, con la finalidad de mantener la seguridad e integridad de los datos en el sistema, la recuperación de contraseña debe realizarse mediante él envió de un correo electrónico. En la tabla 3.3 se define este requerimiento.

**Tabla 3.3: Caso de uso: Recuperar contraseña**

| | |
|---|---|
| **Nombre:** | Recuperar contraseña |
| **Descripción:** Le permite a un usuario registrado recobrar su contraseña. | |
| **Actores:** | Invitado. |
| **Precondiciones:** | 1. El usuario debe estar registrado en el sistema. |
| **Flujo Normal:** | 1. El usuario presiona el link "olvide mi contraseña" en la pantalla de inicio de sesión.<br>2. Introduce su dirección de e-mail y pulsa el botón "Enviar".<br>3. Se envía un correo con un enlace para restablecer la contraseña. |
| **Flujo Alternativo:** | Si el email es incorrecto, el sistema devuelve error. |
| **Postcondiciones:** | El usuario recupera su contraseña para acceder al sistema. |
| **Notas:** | Ninguna. |



### 3.2.4 Gestión del Perfil

Cada usuario del sistema debe poseer un perfil con sus datos personales, el cual debe completar al ingresar por primera vez en el sistema y el cual debe poder modificar en caso de ser necesario, con la finalidad de no comprometer la integridad de los datos, algunos datos personales como la cedula de identidad y el correo electrónico solo podrán ser modificados por un administrador, sin embargo es pertinente que el usuario posea de un módulo que le permita gestionar su contraseña de acceso por sí mismo. En la tabla 3.4 se define este requerimiento, donde el usuario previamente autenticado podrá gestionar su perfil de usuario según considere el mismo.

**Tabla 3.4: Caso de uso: Gestión de perfil**

| | |
|---|---|
| **Nombre:** | Gestión de perfil |
| **Descripción:** Un usuario puede gestionar el perfil con sus datos personales. | |
| **Actores:** | Usuario. |
| **Precondiciones:** | 1. El usuario debe estar autenticado. |
| **Flujo Normal:** | 1. El usuario selecciona "Perfil de Usuario" en la barra de navegación.<br>2. Se cargan los datos personales ingresados por el usuario en el sistema de manera que este puede observar una ficha con los datos actuales de su perfil donde dispondrá de dos (02) opciones.<br>2.1. Modificar los datos personales llenando el formulario correspondiente y pulsando el botón "Actualizar Datos".<br>2.2. Seleccionar la pestaña contraseña para cambiar de formulario y llenar los campos requeridos para realizar el cambio de contraseña y pulsar el botón "Actualizar Contraseña". |
| **Flujo Alternativo:** | 2A Si el usuario no ha completado su perfil se le mostrara un formulario donde pueda completar los datos correspondientes.<br>2.1A Si los datos no son correctos, se validará el formulario y se mostrará un mensaje de error.<br>2.2A Si los datos no son correctos, se validara el formulario y se le mostrara un mensaje de error. |
| **Postcondiciones:** | Los datos del usuario serán actualizados en el sistema. |
| **Notas:** | Ninguna. |



### 3.2.5 Gestión de usuarios

Un administrador debe tener la capacidad para editar y eliminar la información de los otros usuarios del sistema, además de administrar los roles de cada uno de los usuarios del sistema, todo esto sin comprometer la integridad de los datos. En la tabla 3.5 se define este requerimiento y en la figura 3.2 se puede apreciar el diagrama de caso de uso, donde el administrador previamente autenticado, podrá gestionar los usuarios registrados en el sistema según sea conveniente.

**Tabla 3.5: Caso de uso: Gestión de usuarios**

| | |
|---|---|
| **Nombre:** | Gestión de usuarios |
| **Descripción:** Un administrador puede gestionar los usuarios existentes en el sistema. | |
| **Actores:** | Administrador |
| **Precondiciones:** | 1. El administrador debe estar autenticado. |
| **Flujo Normal:** | 1. El usuario selecciona "Gestionar Usuarios" en la barra de navegación.<br>2. Se cargan todos los usuarios registrados en el sistema de manera que el administrador dispone de dos (02) opciones:<br>2.1. Eliminarlos pulsando el icono en forma de cubo de basura.<br>2.2. Modificar sus datos seleccionando el icono con forma de lápiz.<br>2.2.1. El administrador modifica los datos y el rol del usuario seleccionado, y pulsa "Aceptar". |
| **Flujo Alternativo:** | 2.2.1A Si los datos no son correctos, se validara el formulario y se mostrara un mensaje de error. |
| **Postcondiciones:** | Los usuarios correspondientes se verán actualizados o eliminados del sistema. |



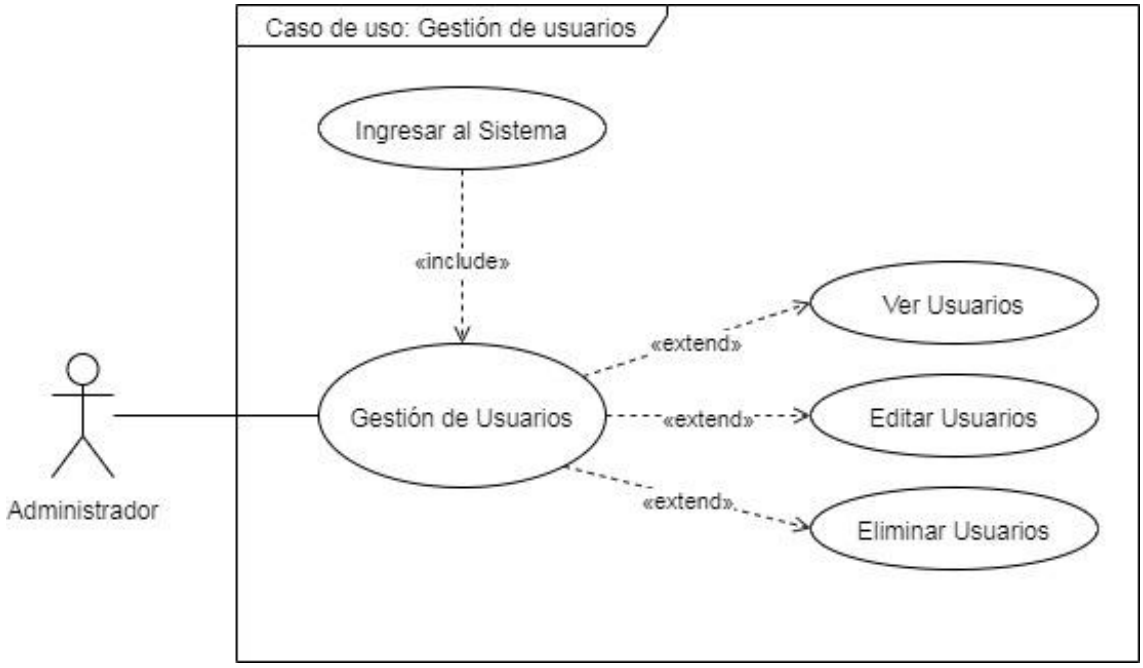



**Figura 3.2 Diagrama de caso de uso: Gestión de usuarios**

### 3.2.6 Gestión de Automóviles

Un usuario puede tener uno o más automóviles, por lo que es necesario un módulo de gestión que permita agregar, ver, editar y eliminar los datos de los mismos. En la tabla 3.6 se define este requerimiento y en la figura 3.3 se visualiza el diagrama de caso de uso donde el usuario luego de haber realizado el proceso de autenticación para ingresar al sistema, puede realizar cambios u obtener información sobre los automóviles que el previamente haya registrado.

**Tabla 3.6: Caso de uso: Gestión de automóviles**

| | |
|---|---|
| **Nombre:** | Gestión de automóviles |
| **Descripción:** Permite a un usuario ingresar, observar y modificar la información sobre sus automóviles. | |
| **Actores:** | Usuario |
| **Precondiciones:** | 1. El usuario debe estar autenticado en el sistema. |
| **Flujo Normal:** | 1. El usuario selecciona "Automóviles" en la barra de navegación.<br>2. Se cargan todos los automóviles registrados previamente por ese usuario de manera que este dispone de tres (03) opciones:<br>2.1. Eliminarlos pulsando el icono en forma de cubo de basura al lado de cada registro.<br>2.2. Modificar los datos, seleccionando el icono en forma de lápiz que se encuentra al lado de cada registro.<br>2.2.1. El usuario modifica los datos del automóvil seleccionado en el formulario que se muestra en pantalla y pulsa "Actualizar".<br>2.3. Agregar un nuevo automóvil pulsando el botón "Agregar Automóvil".<br>2.3.1. El usuario llena un formulario con los datos del automóvil a ingresar y pulsa "Agregar" |
| **Flujo Alternativo:** | 2.2.1A Si los datos no son correctos, se validará el formulario y el sistema devolverá un mensaje de error.<br><br>2.3.1A Si los datos no son correctos, se validara el formulario y se mostrara un |

| | |
|---|---|
| | mensaje de error. |
| **Postcondiciones:** | Los automóviles correspondientes se verán actualizados o eliminados del sistema. Los nuevos automóviles serán agregados a la base de datos. |
| **Notas** | Ninguna. |

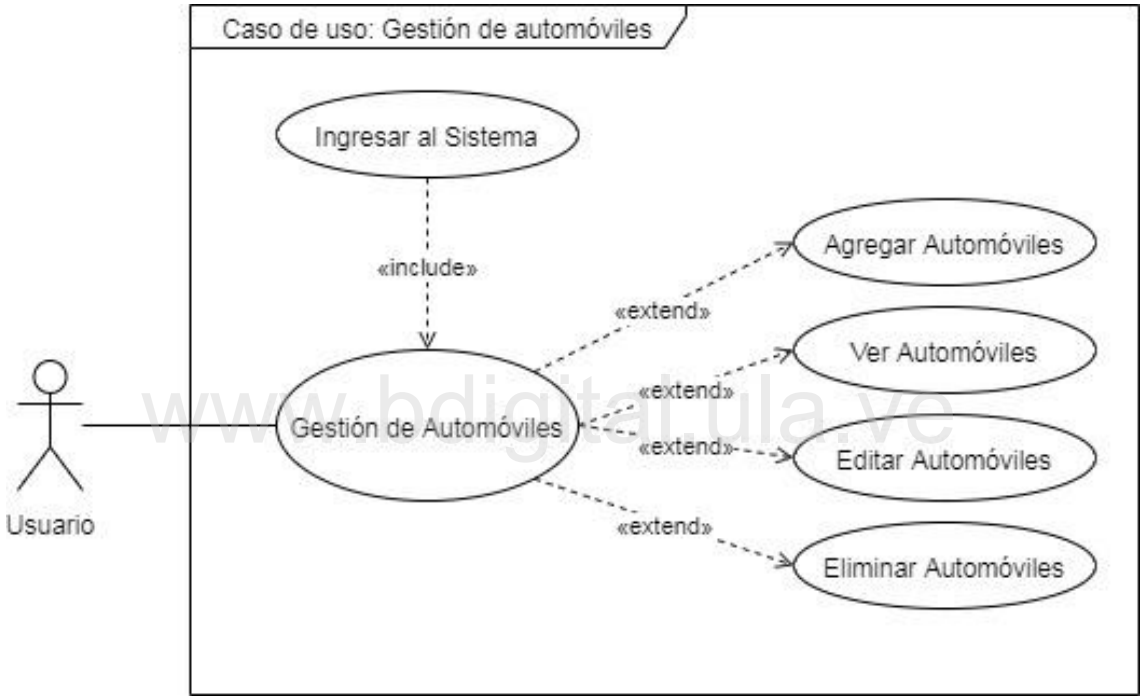



**Figura 3.3: Diagrama de caso de uso: Gestión de automóviles**

### 3.2.7 Consulta de Automóviles

Es fundamental para el manejo centralizado de la información que los administradores tengan la capacidad de consultar todos los automóviles registrados en el sistema a fin de estimar la demanda del combustible y facilitar de esta manera la toma de decisiones, en la tabla 3.7 se define este requerimiento y en la figura 3.4 se visualiza el diagrama de caso de uso.

**Tabla 3.7: Caso de uso: Consulta de automóviles**

| | |
|---|---|
| **Nombre:** | Consulta de automóviles |
| **Descripción:** Un administrador puede consultar todos los automóviles que hayan sido ingresados por los usuarios al sistema. | |
| **Actores:** | Administrador |
| **Precondiciones:** | 1. El administrador debe estar autenticado.<br>2. Deben existir automóviles registrados. |
| **Flujo Normal:** | 3. El administrador selecciona "Automóviles" en la barra de navegación.<br>4. Se cargan una lista todos los datos de todos los automóviles registrados en el sistema. |
| **Flujo Alternativo:** | Ninguno. |
| **Postcondiciones:** | Ninguna. |
| **Notas:** | Ninguna. |



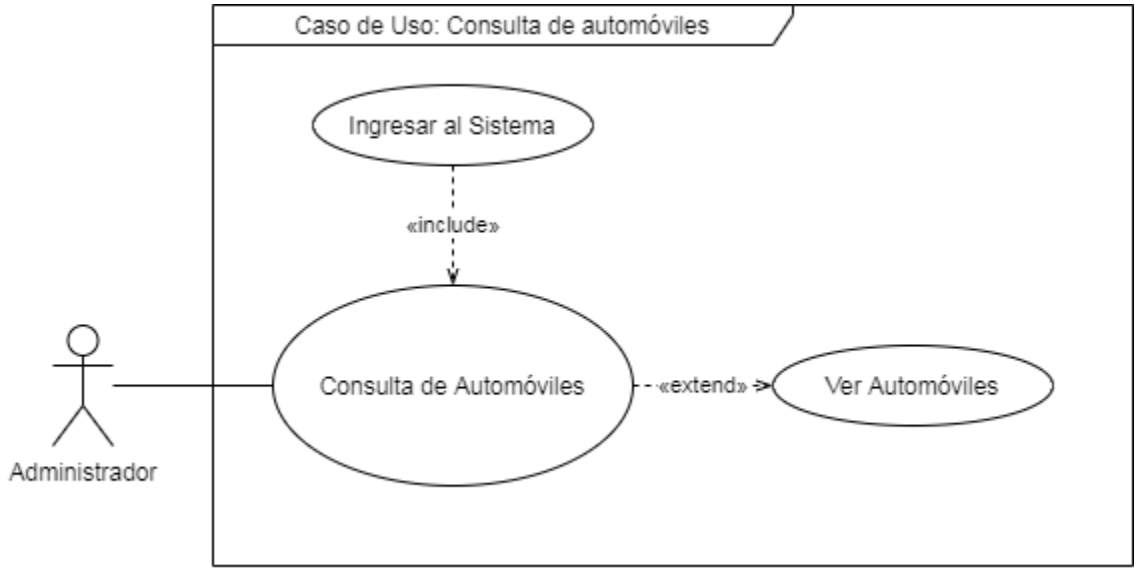



**Figura 3.4: Diagrama de caso de uso: Consulta de automóviles**

### 3.2.8 Gestión de Estaciones de Servicio

Con la finalidad de poner a disposición del usuario toda la información referente a las estaciones de servicio, los administradores podrán agregar, editar y eliminar la información de las estaciones disponibles para surtir combustible. En la tabla 3.8 se define este requerimiento y en la figura 3.5 se observa el diagrama de caso de uso correspondiente, donde se aprecia como los usuarios del sistema solo podrán visualizar las estaciones de servicio mientras que los administradores podrán gestionar las mismas según consideren necesario.

**Tabla 3.8: Caso de uso: Gestión de estaciones de servicio**

| | |
|---|---|
| **Nombre:** | Gestión de estaciones de servicio |
| **Descripción:** Permite a un administrador ingresar, observar y modificar la información sobre las estaciones de servicio, para posteriormente ser mostrada a los usuarios. | |
| **Actores:** | Administrador. |
| **Precondiciones:** | 1. El usuario debe estar autenticado en el sistema. |
| **Flujo Normal:** | 1. El administrador selecciona "Estaciones de Servicio" en la barra de navegación.<br>2. Se visualizan todas las estaciones de servicio registradas previamente, permitiendo gestionarlas mediante (03) opciones:<br>2.1. Eliminar las estaciones existentes pulsando el icono en forma de cubo de basura al lado de la ficha de cada estación.<br>2.2. Modificar los datos de la estación de servicio pulsando el icono con forma de lápiz.<br>2.2.1. El usuario modifica los datos de la estación de servicio seleccionada en el formulario que se muestra en pantalla y pulsa "Actualizar".<br>2.3. Agregar una nueva estación de servicio pulsando el botón "Agregar Estación"<br>2.3.1. El administrador llena un formulario con los datos de la estación de servicio a |

| | |
|---|---|
| | ingresar y pulsa “Agregar” |
| **Flujo Alternativo:** | 2.2.1A Si los datos no son correctos, se validará el formulario y el sistema devolverá un mensaje de error.<br><br>2.3.1A Si los datos no son correctos, se validara el formulario y se mostrara un mensaje de error debajo de los campos correspondientes. |
| **Postcondiciones:** | Las estaciones de servicio correspondientes se verán actualizadas o eliminadas del sistema. Las nuevas estaciones de servicio serán agregadas a la base de datos. |
| **Notas:** | Ninguna. |

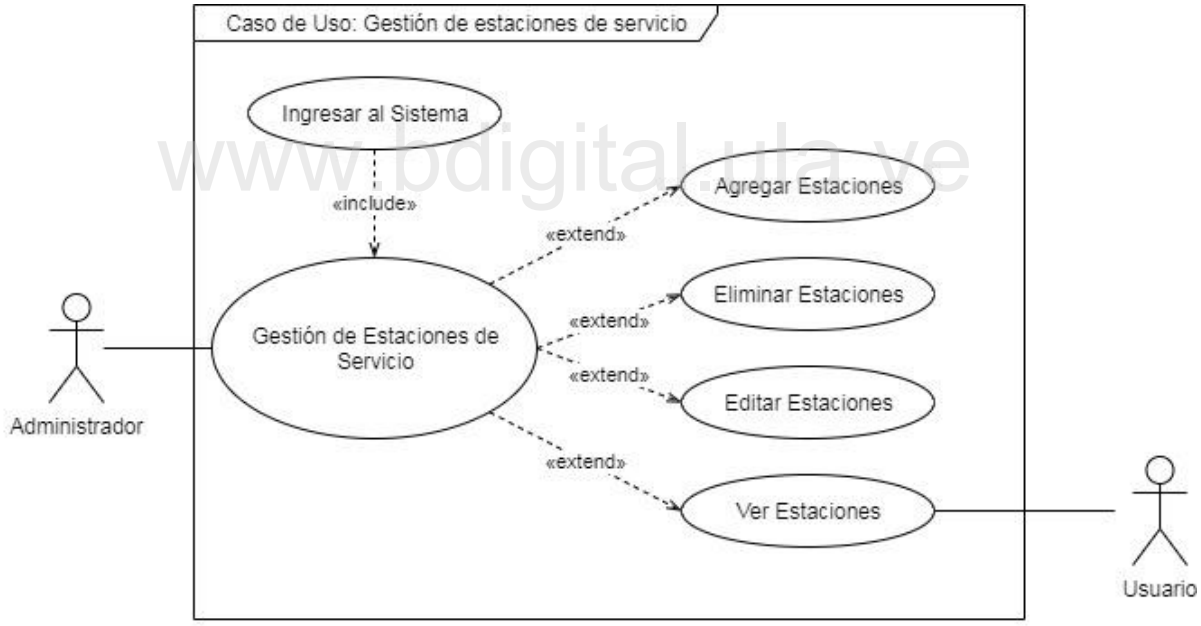



**Figura 3.5: Diagrama de caso de uso: Gestión de estaciones de servicio**

### 3.2.9 Gestión de Transporte

Con la finalidad de mantener un mayor control sobre los envíos de combustible se deberá crear un módulo para la gestión de transporte, donde los administradores, previamente autenticados, podrán ingresar al sistema toda la información referente a los camiones cisterna, los cuales se utilizan para llevar el combustibles desde las plantas de despacho, hasta las estaciones de servicio, estos pueden transportar aproximadamente 40.000 litros de combustible, contando con dispositivos electrónicos que miden permanentemente la carga. En la tabla 3.9 se define este requerimiento y en la figura 3.6 se visualiza el diagrama donde el administrador puede listar las unidades existentes, y gestionarlas, agregando nuevas unidades, editando la información de las ya existentes, o eliminando alguna de ellas según se considere pertinente.

**Tabla 3.9: Caso de uso: Gestión de transporte.**

| | |
|---|---|
| **Nombre:** | Gestión de transporte |
| **Descripción:** Permite a un administrador ingresar, observar y modificar la información sobre las unidades que transportaran el combustible. | |
| **Actores:** | Administrador |
| **Precondiciones:** | 1. El usuario debe estar autenticado en el sistema. |
| **Flujo Normal:** | 1. El administrador selecciona "Transporte" en la barra de navegación.<br>2. Se visualizan todos los transportes de combustible registrados previamente, permitiendo gestionarlas mediante (03) opciones:<br>2.1. Eliminar una unidad existente pulsando el icono en forma de cubo de basura.<br>2.2. Modificar los datos de la unidad pulsando el icono con forma de lápiz.<br>2.2.1. El usuario modifica los datos del transporte seleccionado en el formulario que se muestra en pantalla y pulsa "Actualizar".<br>2.3. Agregar una nueva unidad pulsando el botón "Agregar Transporte"<br>2.3.1. El administrador llena un formulario con |

| | |
|---|---|
| | los datos de la unidad a ingresar y pulsa "Agregar" |
| **Flujo Alternativo:** | 2.2.1A Si los datos no son correctos, se validará el formulario y el sistema devolverá un mensaje de error.<br><br>2.3.1A Si los datos no son correctos, se validara el formulario y se mostrara un mensaje de error debajo de los campos correspondientes. |
| **Postcondiciones:** | Las unidades de transporte correspondientes se verán actualizadas o eliminadas del sistema. Las nuevas unidades serán agregadas a la base de datos. |
| **Notas** | Ninguna |

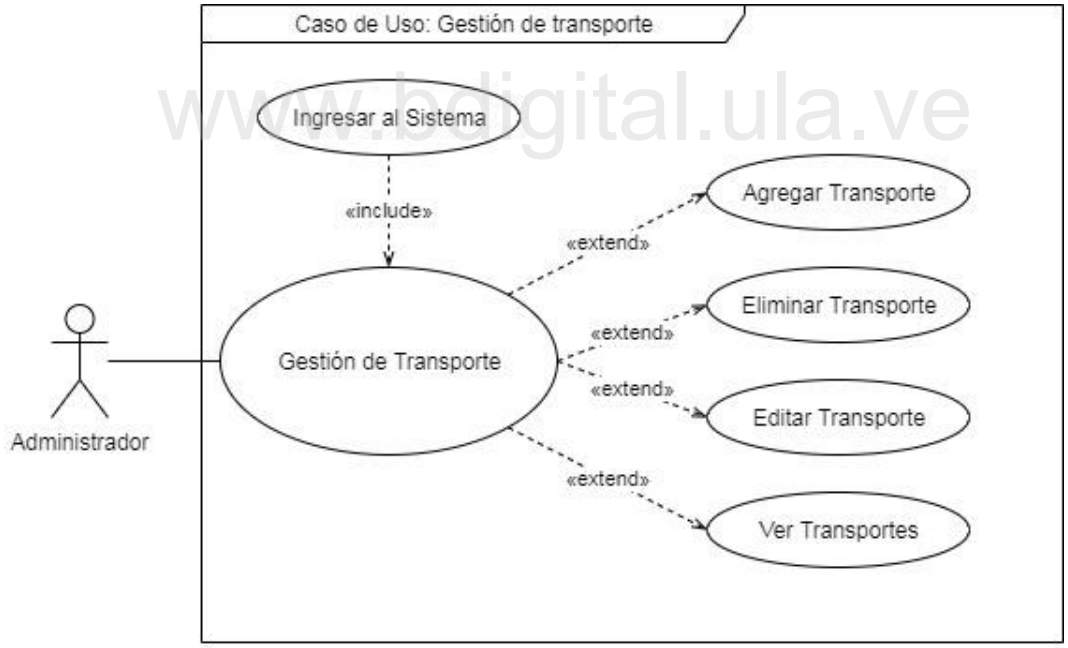





**Figura 3.6: Diagrama de caso de uso: Gestión de transporte**

### 3.2.10 Gestión de Solicitudes

Un usuario del sistema deberá poder realizar solicitudes de combustible cuando así lo considere necesario, es por ello que al acceder al sistema encontrará un módulo que le permitirá de manera rápida y sencilla realizar solicitudes de combustible para cualquiera de los automóviles previamente registrados. En estas solicitudes deberá escoger además del automóvil al cual desea surtir de combustible, la estación de servicio provista por el administrador que mejor se adapte a sus necesidades. En la tabla 3.10 se define este requerimiento y en la figura 3.7 se visualiza el diagrama donde el usuario puede agregar o eliminar solicitudes según su criterio.

**Tabla 3.10: Caso de uso: Gestión de solicitudes**

| | |
|---|---|
| **Nombre:** | Gestión de solicitudes. |
| **Descripción:** Un usuario podrá agregar o eliminar solicitudes de combustible. | |
| **Actores:** | Usuario. |
| **Precondiciones:** | 1. El usuario debe estar autenticado.<br>2. Deben existir automóviles y estaciones de servicio. |
| **Flujo Normal:** | 1. El usuario selecciona "Solicitudes" en la barra de navegación.<br>2. Se cargan todas las solicitudes de combustible registrados en el sistema por el usuario de manera que este dispone de dos (02) opciones:<br>2.1. Eliminar la solicitud.<br>2.1.1. El usuario pulsa el icono del cubo de basura al lado de la solicitud.<br>2.2. Agregar una nueva solicitud.<br>2.2.1. El usuario pulsa "Agregar Solicitud" en la barra de navegación.<br>2.2.2. Selecciona el automóvil que desea surtir de combustible y la estación de servicio, y pulsa "Agregar". |
| **Flujo Alternativo:** | 2.2.2A Si existe una solicitud en espera de asignación para el automóvil que se está seleccionando se arroja un mensaje de error. |
| **Postcondiciones:** | La solicitud de combustible se crea o elimina de la base de datos. |
| **Notas:** | Luego que la solicitud sea aprobada ya no podrá ser eliminada. |

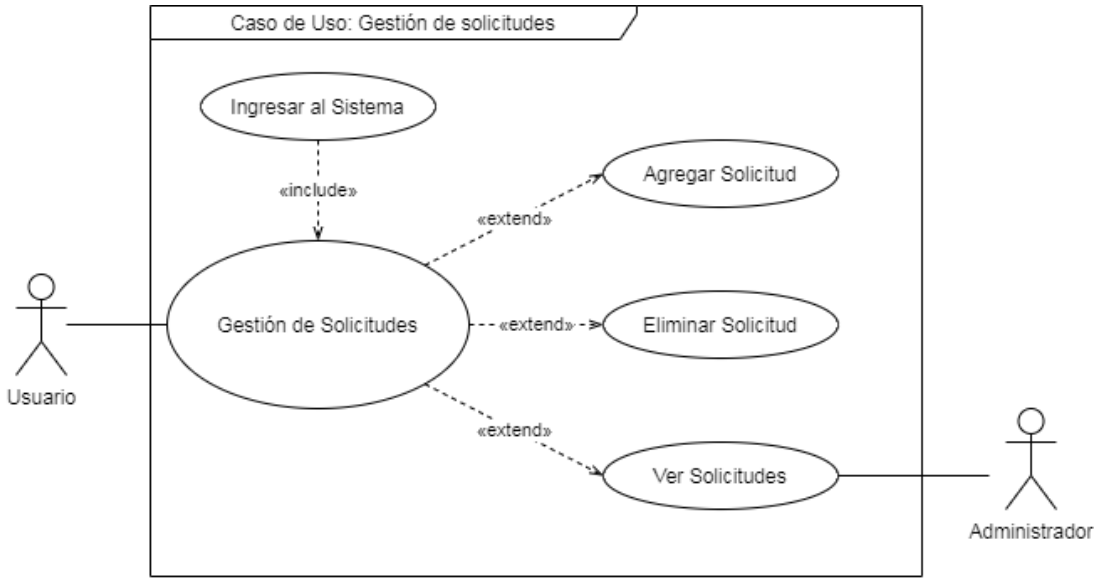



**Figura 3.7: Diagrama de caso de uso: Gestión de solicitudes**



### 3.2.11 Gestión de Combustible

Un administrador debe poder asignar envíos de combustible a las diferentes estaciones de servicio, para ello deberá indicar información relevante para el despacho como los datos del transporte, la cantidad de combustible asignada a la estación de servicio, y la fecha estimada de recepción. Estos envíos permitirán realizar cálculos a posteriori de la oferta de combustible. En la tabla 3.11 se describe este requerimiento y en la figura 3.8 se observa el diagrama de caso de uso correspondiente.

**Tabla 3.11: Caso de uso: Gestión de combustible**

| | |
|---|---|
| **Nombre:** | Gestión de Combustible. |
| **Descripción:** Un administrador podrá agregar información sobre un envió de combustible a una estación de servicio. | |
| **Actores:** | Administrador |
| **Precondiciones:** | 1. El usuario debe estar autenticado.<br>2. Deben existir unidades de transporte y estaciones de servicio. |
| **Flujo Normal:** | 1. El administrador selecciona “Combustible” en la barra de navegación en |

| | |
|---|---|
| | el cual tendrá dos (02) opciones.<br>1.1. Selecciona la opción "Nuevo Envió de Combustible"<br>1.1.1. Indica los datos que solicita el formulario y pulsa "Aceptar".<br>1.2. Selecciona la opción "Consulta de envíos", donde se desplegara una lista con la información detallada de todos los envíos. |
| **Flujo Alternativo:** | 1.1.1A Si los datos no son correctos, se validara el formulario y el sistema devolverá un mensaje de error. |
| **Postcondiciones:** | Los datos del envió de combustible son agregados a la base de datos. |
| **Notas:** | Ninguna. |

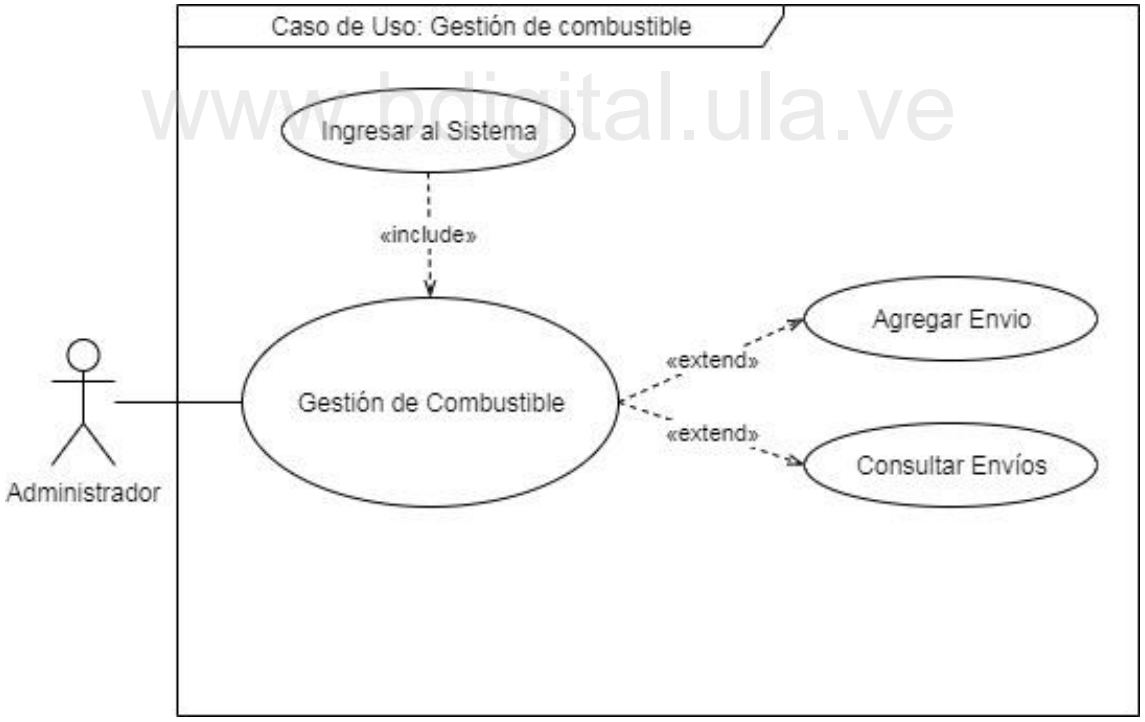



**Figura 3.8: Diagrama de caso de uso: Gestión de combustible**

### 3.2.12 Asignación de Combustible

Para que el sistema cumpla con su objetivo deberá existir un módulo que les permita a los administradores aprobar las solicitudes de gasolina realizadas por los usuarios en función de la cantidad enviada a cada estación de servicio, dado el propósito del sistema es pertinente que el proceso de asignación de combustible se realice de manera automatizada sucesivo a la aprobación del administrador. En la tabla 3.12 se describe este requerimiento y en la figura 3.9 se observa el diagrama de caso de uso correspondiente.

**Tabla 3.12: Caso de uso: Asignación de combustible**

| | |
|---|---|
| **Nombre:** | Asignación de Combustible. |
| **Descripción:** Un administrador podrá aprobar la asignación del combustible procedente de una gestión de despacho. | |
| **Actores:** | Administrador. |
| **Precondiciones:** | 1. Deben existir una gestión de combustible. |
| **Flujo Normal:** | 1. El usuario selecciona "Asignación de Combustible" en la barra de navegación.<br>2. Se cargan todas las gestiones de combustible registrados en el sistema por el usuario de manera que este dispone de dos (02) opciones:<br>2.1. Eliminar la gestión, el administrador pulsa el icono del cubo de basura al lado de la gestión, confirma pulsando "aceptar" en la modal de comprobación.<br>2.2. Aprobar la gestión, el administrador pulsa el icono con la forma de campana y aprueba las solicitudes correspondientes a la gestión de combustible seleccionada. |
| **Flujo Alternativo:** | Ninguno |
| **Postcondiciones:** | Las solicitudes correspondientes cambiaran su estatus a aprobadas y la cantidad de litros disponibles de la gestión se reducirá según la capacidad de los carros asignados. |
| **Notas:** | Una vez aprobada la gestión y notificados los usuarios esta no podrá ser eliminada. |

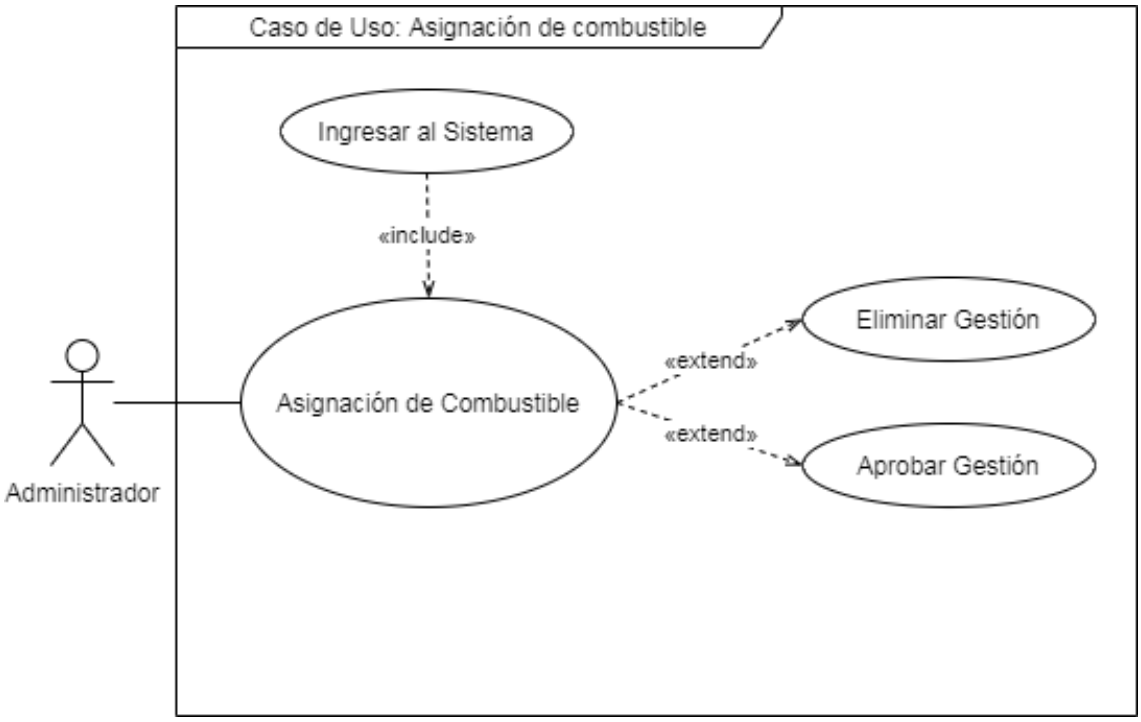



**Figura 3.9: Diagrama de caso de uso: Aprobación de combustible.**



Este capítulo definió los requerimientos necesarios para el desarrollo de la aplicación, lo que nos proporciona una base estable de cuáles deben ser las restricciones y el comportamiento que debe tener nuestro sistema. Se tiene una serie de requisitos funcionales que ofrecen una experiencia de uso completa, ya que abarcan todos los aspectos del proceso de suministro de combustible, desde las perspectivas del usuario y del administrador. Teniendo como premisa la facilidad y eficacia del sistema en la integración ambos roles encontramos operaciones básicas de gestión de usuarios como autenticación, registro de automóviles, necesarias para poder llevar a cabo las solicitudes del usuario. Además, se definieron especificaciones que deben ser llevadas a cabo por los administradores de la aplicación con la finalidad de permitir la correcta funcionalidad del sistema y brindar información que pueda ser utilizada a posteriori para toma de decisiones. Los requerimientos no funcionales definidos establecen reglas acerca de la implementación y gestión del proyecto, como lenguaje de programación, estilos visuales y control de versiones.

# Capítulo 4

# Diseño del Software

Continuando el proceso de desarrollo definido por el método White WATCH encontramos el diseño del software, el cual tiene como objetivo describir la arquitectura que tendrá el sistema, y el diseño de cada uno de los componentes que integren dicha arquitectura, en este capítulo se contemplan los procesos de identificación de subsistemas, diseño de interfaz de usuario, diseño de modelo de datos y despliegue del sistema. Los cuales fueron refinados en cada una de las iteraciones de la fase de desarrollo.



## 4.1 Identificación de Subsistemas

Con el objetivo de reducir la complejidad y facilitar el diseño del sistema, se realiza una descomposición del sistema en subsistemas, la identificación de estos se puede realizar mediante la continuidad directa de los modelos de análisis de requerimientos o aplicando nuevos criterios de diseño tales como la facilidad de mantenimiento, la reutilización de elementos del propio sistema y la optimización de recursos.

Adoptando criterios lógicos como la homogeneidad de los procesos y la afinidad de los requisitos podemos identificar los siguientes subsistemas para el sistema en desarrollo: Registro de un usuario, inicio de sesión y recuperación de contraseña, gestión de usuarios, gestión de automóviles, gestión de estaciones de servicio, gestión de transporte y gestión de solicitudes.

## 4.2 Diseño de Subsistemas

Con el propósito de definir comportamientos específicos para cada una de los subsistemas identificados en términos de colaboración entre elementos, Se identificarán y definirán las dependencias entre subsistemas, además, se asignarán los requisitos correspondientes a cada uno de los subsistemas.

### 4.2.1 Registro de usuarios

Este componente está relacionado con el caso de uso presentado en la tabla 3.1. Y comprende el registro de los usuarios en el sistema, incluye datos básicos para el uso del sistema como nombre completo del usuario, correo electrónico y contraseña con el cual realizara el proceso de autenticación posteriormente. Para ello es necesario el modelo de datos presentado en la figura 4.1.

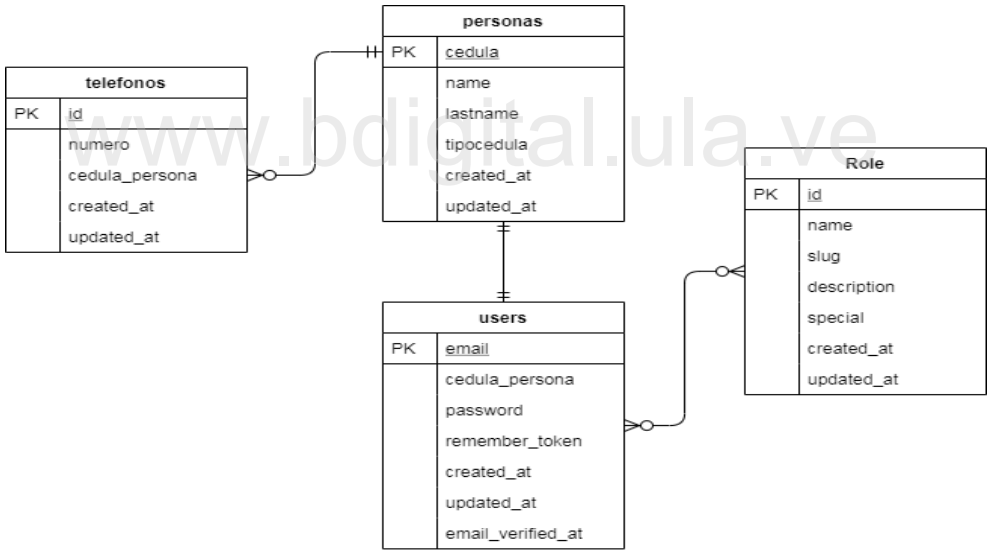





**Figura 4.1: Modelo de datos: Registro de usuarios**

### 4.2.2 Inicio de Sesión y Recuperación de Contraseña

Este subsistema se relaciona con los casos de uso presentados en las tablas 3.2 y 3.3, su funcionalidad se basa en el proceso de autenticación de los usuarios al ingresar al sistema, incluyendo el proceso de recuperación de contraseña en caso de perder el acceso al sistema. Para ello es pertinente el uso del modelo de datos presentado en la figura 4.2.

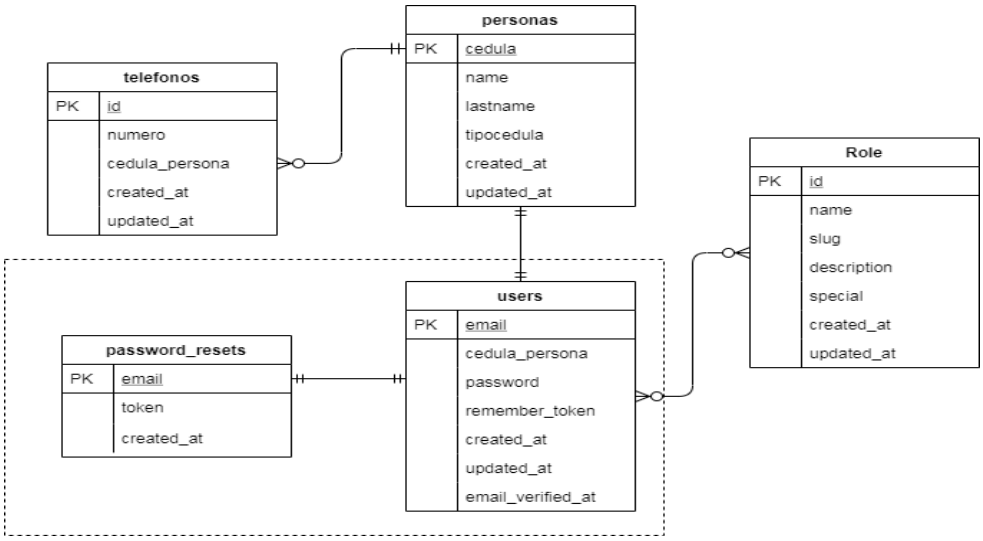



**Figura 4.2: Modelo de datos: Inicio de sesión y recuperación de contraseña**



### 4.2.3 Gestión de Usuarios

En este módulo se engloban las actividades mostradas en las tablas 3.4 y 3.5, en las cuales se definen las tareas de ver, editar y eliminar usuarios. Algunas de estas pueden ser ejecutadas por el mismo usuario, sin embargo, la mayor parte de ellas serán llevadas a cabo por los administradores del sistema; Entre las tareas del subsistema se adjunta la posibilidad de edición de roles de usuarios. El modelo de datos necesario ya fue desarrollado en la figura 4.1.

### 4.2.4 Gestión de Automóviles

Este subsistema contempla las actividades relacionadas con los casos de uso presentados en la tabla 3.6 y 3.7, las cuales incluyen la creación, edición, visualización y eliminación de automóviles, aunque estas tareas están reservadas en su mayoría para los usuarios, este módulo tiene la particularidad de englobar información de especial interés para los administradores, para ello es necesario el modelo de datos presentado en la figura 4.3.

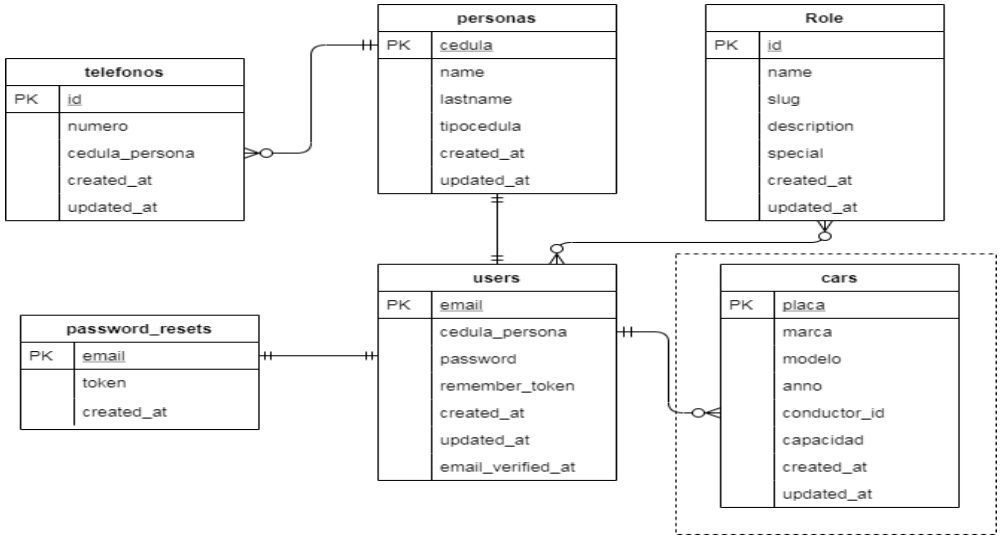



**Figura 4.3: Modelo de datos: Gestión de automóviles**

### 4.2.5 Gestión de Estaciones de Servicio



Este componente se relaciona con los casos de uso presentados en la tabla 3.8, donde se incluyen las actividades de crear, editar, ver y eliminar estaciones de servicio, las cuales se realizan con la finalidad de brindar al usuario información que le permita escoger la estación que mejor se adapte a sus necesidades, para llevarlo a cabo es necesario el modelo de datos que observamos en la figura 4.4.

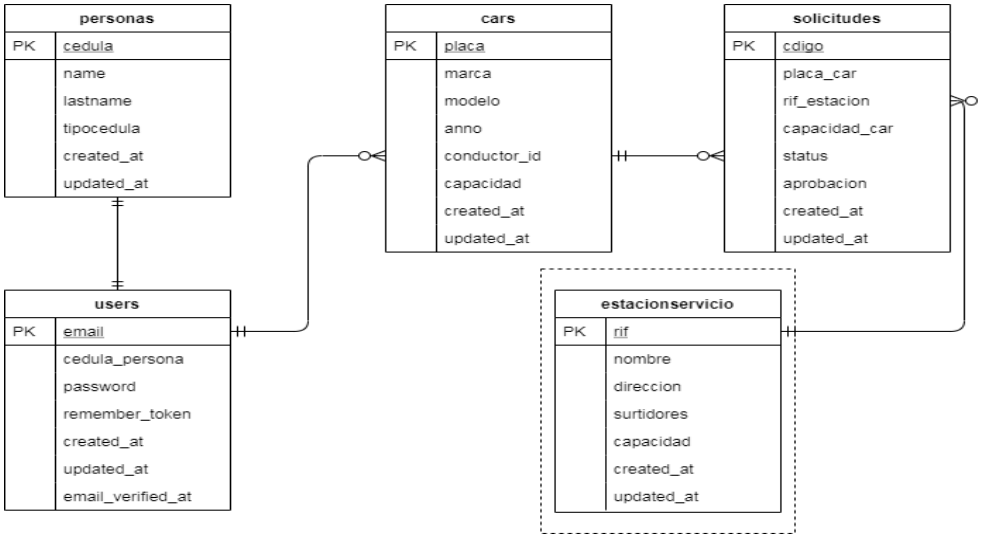



**Figura 4.4: Modelo de datos: Gestión de estaciones de servicio**

### 4.2.6 Gestión de Transporte

El módulo de gestión de transporte se relaciona con los casos de uso presentados en la tabla 3.9, en ellos se contemplan las actividades de crear, editar, ver y eliminar los datos de un camión cisterna los cuales son fundamentales para la asignación de combustible a las estaciones de servicio, es necesario el uso del modelo presentado en la figura 4.5 ya que nos ofrece datos de gran importancia para los envíos.

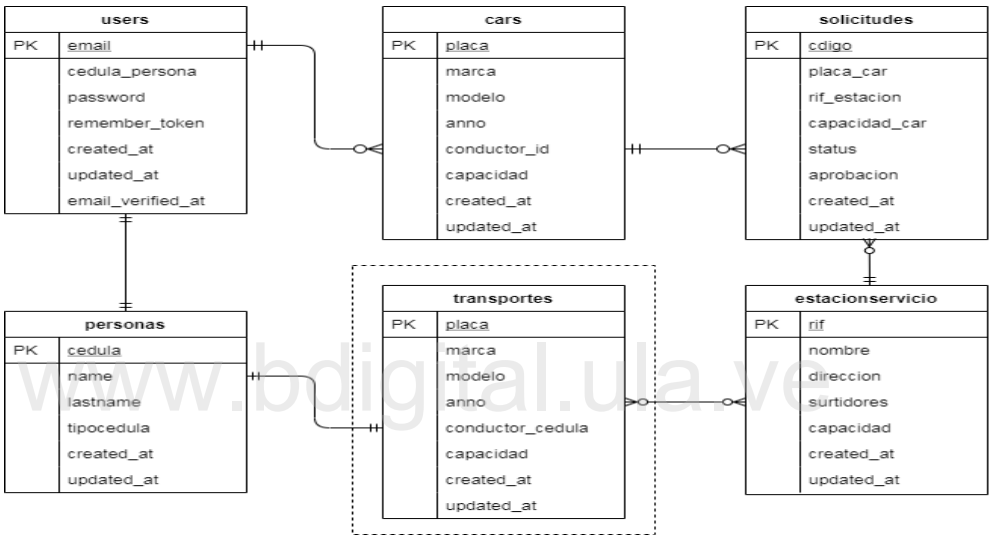



**Figura 4.5: Modelo de datos: Gestión de transporte**

### 4.2.7 Gestión de Solicitudes

El subsistema de solicitudes abarca los casos de uso presentados en la tabla 3.10, en la cual se incluyen las actividades de crear, ver y eliminar solicitudes, este módulo cuenta con una variable de estado, mediante la cual evalúa si la solicitud ya ha sido aprobada o sigue a la espera de asignación, este componente es de gran importancia ya que como se observa en la figura 4.6 funciona como tabla de unión o intermedia, en la relación de muchos a muchos que existe entre los automóviles y las estaciones de servicio.

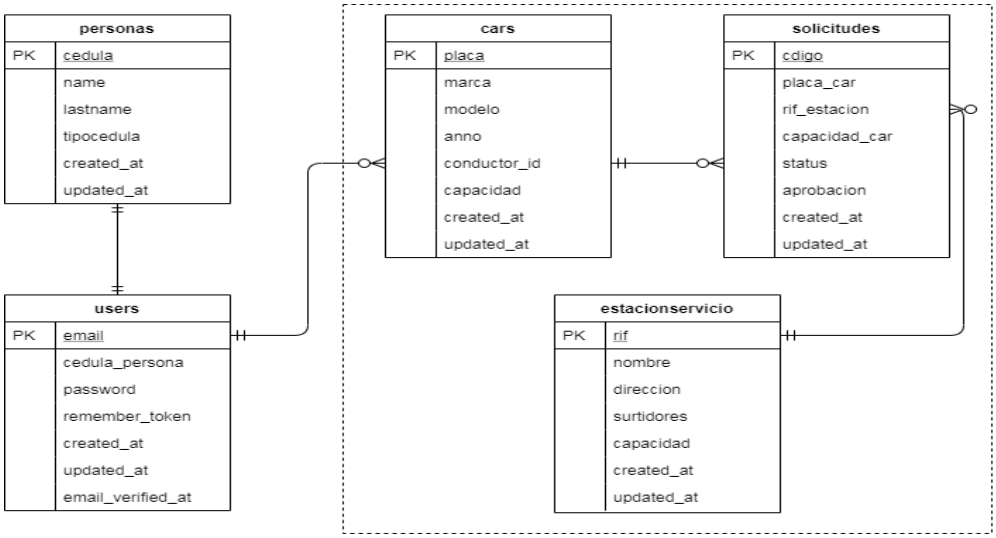



**Figura 4.6: Modelo de datos: Gestión de solicitudes**



## 4.3 Diseño de Interfaz

Se realizó el diseño de una interfaz que pretende ser intuitiva y amigable, mediante el uso de prototipos de interfaces los cuales no permiten hacernos una idea más precisa de las interfaces que proveerá el sistema, también se realizó la definición de un diagrama jerárquico de pantallas el cual nos permite apreciar cuáles serán las vistas del sistema y como está estructurada cada una de las opciones a fin de contribuir de forma positiva con la experiencia del usuario. Este diagrama también nos facilitará la tarea de describir los perfiles del sistema, mostrando los privilegios que tendrá cada uno de los roles asignados. En la figura 4.7 encontraremos el diagrama jerárquico de pantallas, el cual es utilizado para definir los perfiles del sistema en las figuras 4.8 y 4.9.

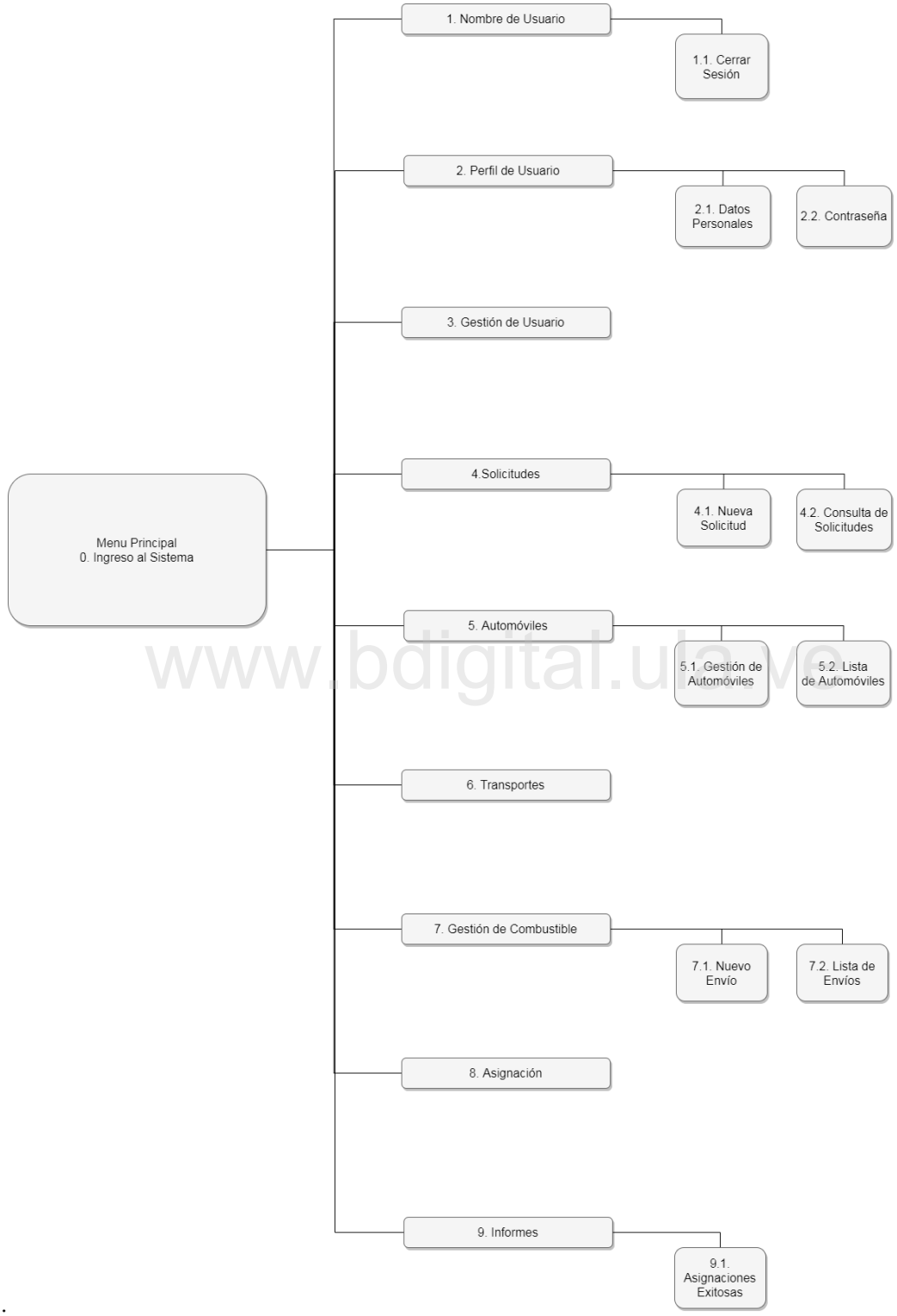
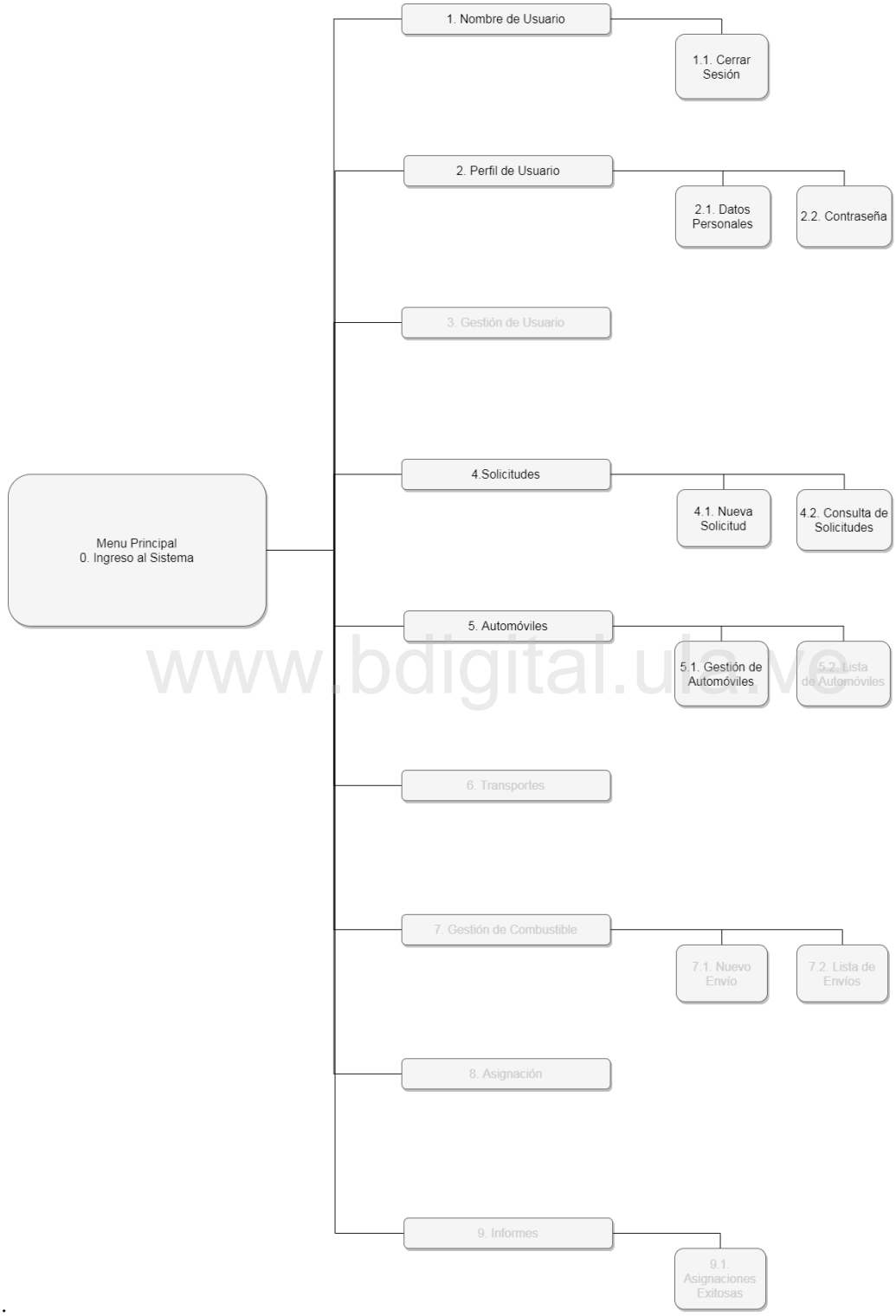
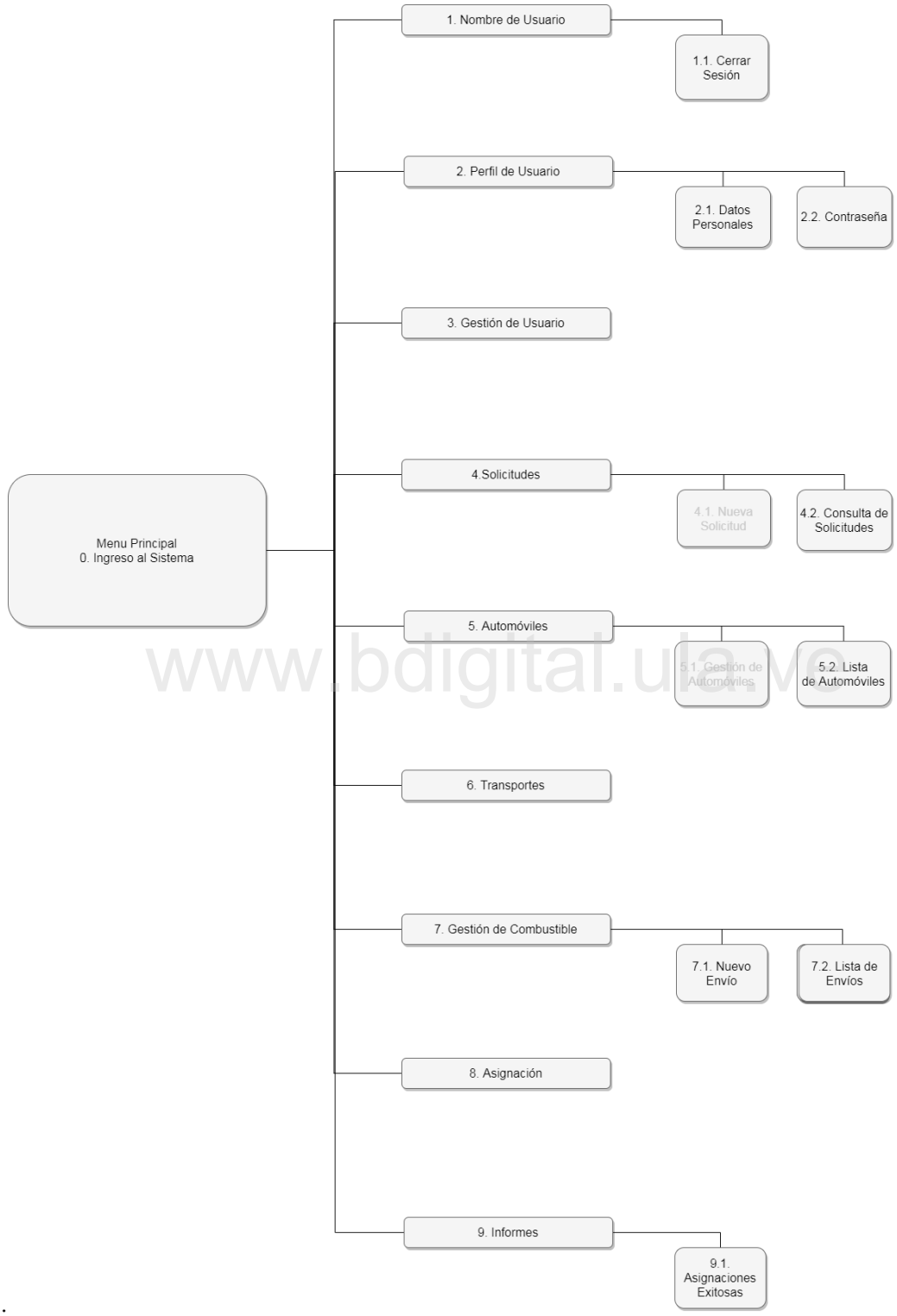



**Figura 4.7: Diagrama jerárquico de pantallas**




.

**Figura 4.8: Perfiles del sistema: Usuario**




.

**Figura 4.9: Perfiles del sistema: Administrador**

## 4.4 Diseño del Modelo de Datos

El diseño del modelo de datos o modelado de la base de datos, consiste en la estructuración de los datos con el fin de facilitar el análisis y la integración de los mismos, para ello se utilizó el modelo de entidad-relación y el modelo relacional.

### 4.4.1 Modelo Entidad-Relación

El modelo de entidad-relación posee como una de sus principales características un alto nivel de abstracción lo que permite ver con mayor claridad la información utilizada. En la figura 4.10 podemos observar el diagrama entidad-relación del sistema.

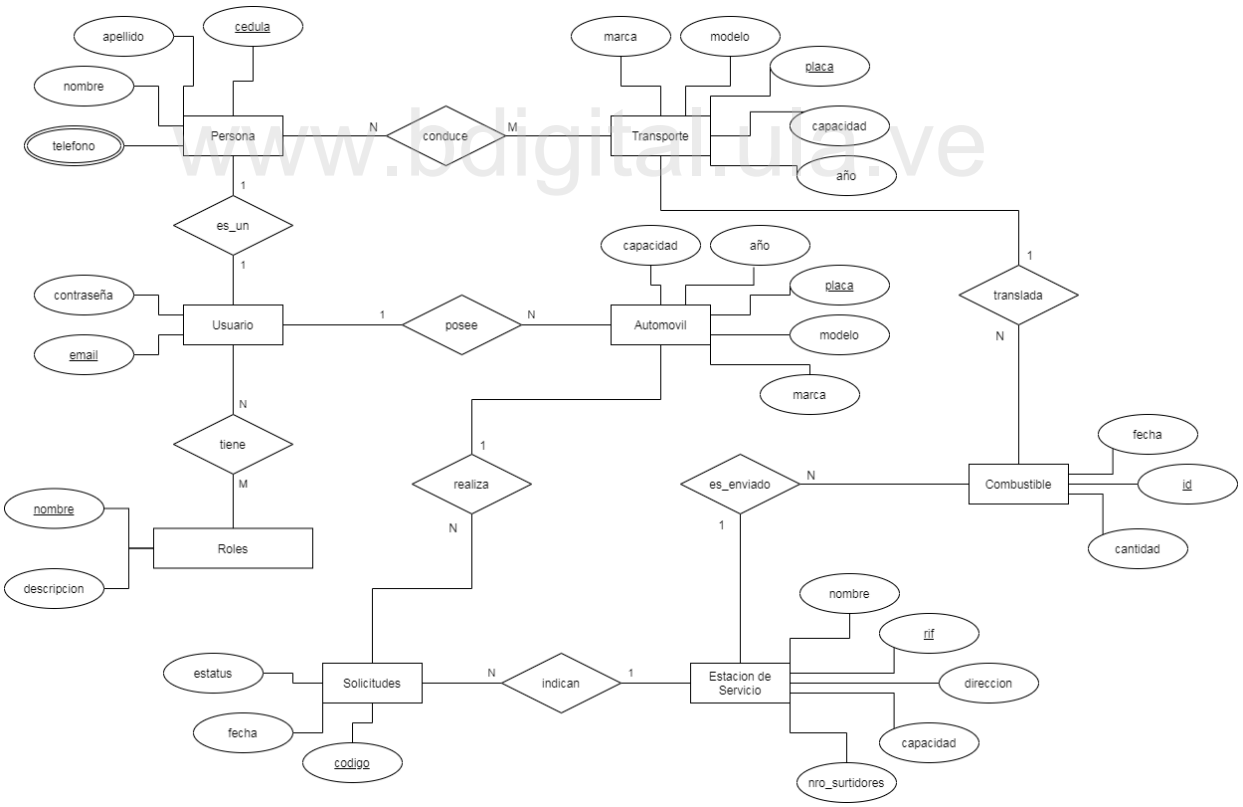



**Figura 4.10: Diagrama de Modelo Entidad-Relación.**

### 4.4.2 Modelo Relacional

El modelo relacional nos proporciona un modelo basado en la teoría de conjuntos, cuya idea es el uso de relaciones. En la figura 4.11 se observa el resultado de la integración de los modelos de datos.

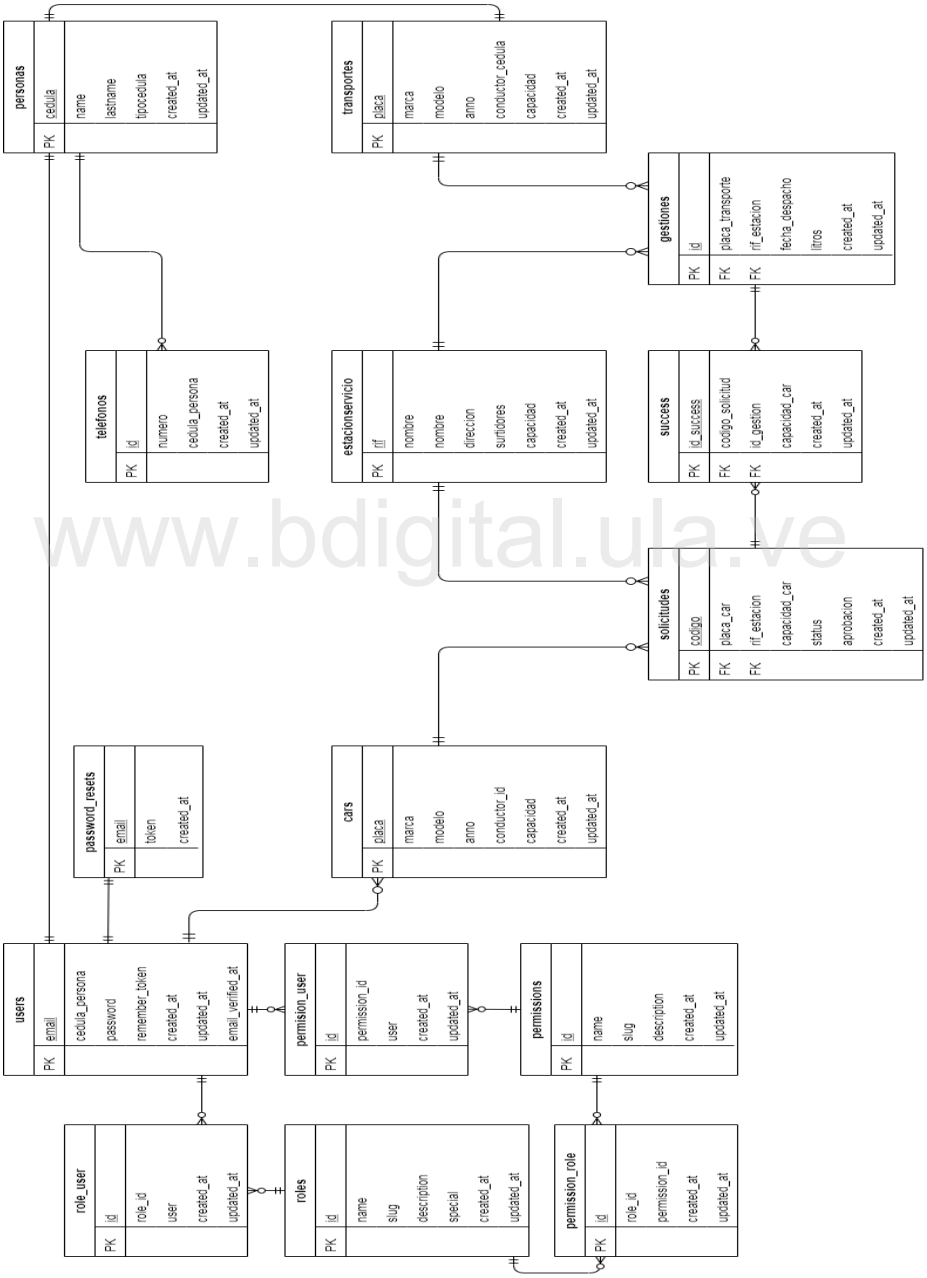



**Figura 4.11: Diagrama de Modelo Relacional**

## 4.5 Despliegue de la aplicación

En la figura 4.12 se muestra el diagrama de despliegue de la aplicación, resaltando una arquitectura de tres capas y dos niveles, donde del lado del cliente encontramos el navegador web y del lado del servidor se encuentra el servidor web ***HTTP apache*** principal encargado de soportar la aplicación.

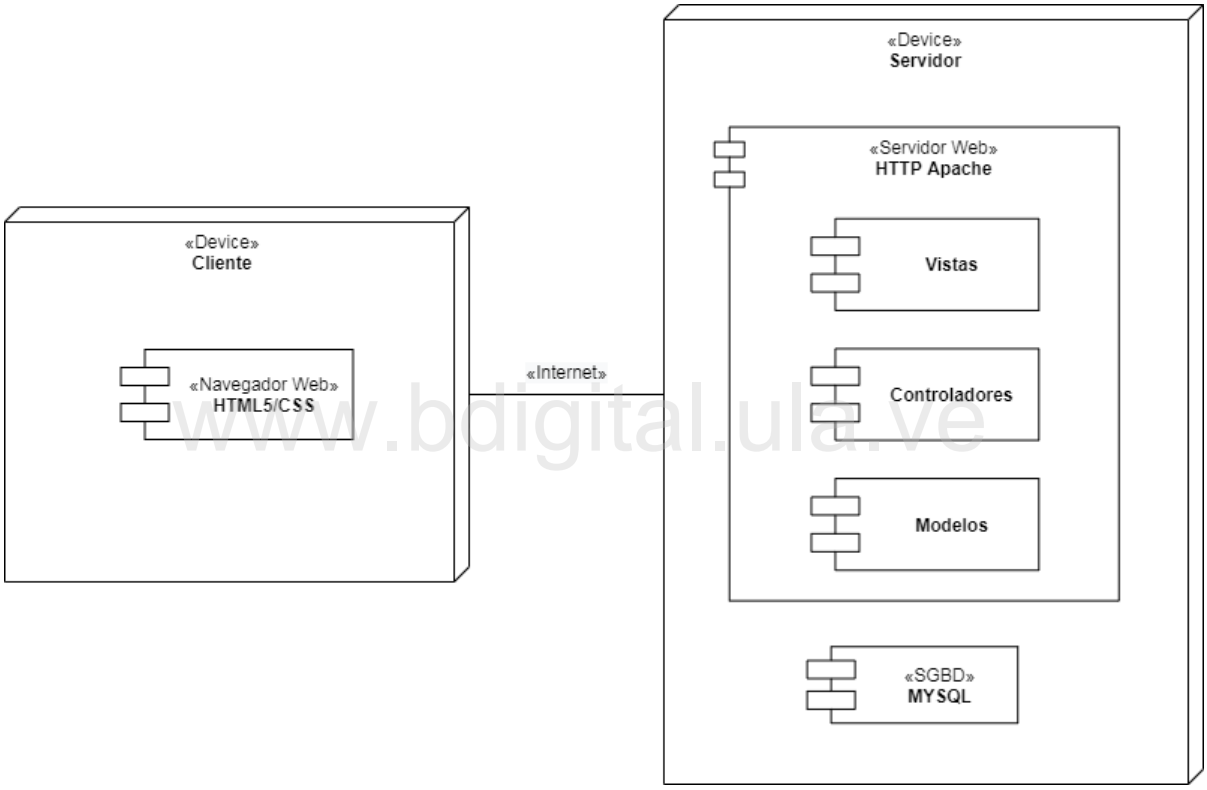





**Figura 4.12: Diagrama de Despliegue**

El ***Servidor*** aloja el sistema, incluyendo el servidor web el cual procesa y dirige las peticiones HTTP realizadas por el cliente, y el sistema gestor de bases de datos. Este diseño arquitectónico busca el correcto funcionamiento del sistema y a su vez permite que el mismo cumpla sus objetivos, siendo uno de ellos el ser accesible para cada uno de los usuarios desde cualquier dispositivo con acceso a internet.

En este capítulo se especificaron todos los aspectos del diseño arquitectónico del sistema. Se definieron cada uno de los subsistemas mediante la continuidad directa de los modelos de análisis de requerimientos, indicando los modelos de datos necesarios para el correcto funcionamiento, encontrando comportamientos específicos en términos de colaboración entre elementos. Se realizó el diseño de la interfaz y se definieron los accesos a cada una de las vistas del sistema para los perfiles de usuarios correspondientes partiendo de un diagrama jerárquico de pantallas. También se formalizó el modelado para una base de datos relacional buscando facilitar el análisis y la integración de la información suministrada por los usuarios y, por último, se planteó un diseño que cuenta con arquitectura compuesta por tres capas y dos niveles.

# Capítulo 5

# Desarrollo de Versiones

Según el método White WATCH, el desarrollo del producto debe realizarse de manera iterativa. Usando el enfoque progresivo o evolutivo, al final de cada ciclo se debe obtener una versión funcional del sistema. En este proyecto se realizaron seis (06) iteraciones que abarcaron todos los aspectos definidos en la fase de ingeniería de requisitos.

## 5.1 Aprovisionamiento de Componentes



Antes de comenzar a detallar las fases de desarrollo de la aplicación, es necesario mencionar los elementos que forman parte de la infraestructura del software requerida para desarrollar el sistema. La plataforma de desarrollo cuenta con las siguientes herramientas:

- Servidor de Base de Datos MySQL, versión 5.7.29.
- ***MySQL Workbench,*** versión 6.3.8**.**
- Framework Laravel, versión 6.18.0.
- Navegador Web ***Google Chrome***, versión 80.0.
- Editor de texto ***Sublime Text***, versión 3.2.2.
- ***Composer***, versión 1.9.1.

## 5.2 Primera Iteración

Durante esta primera fase de desarrollo se realizó un módulo de registro y autenticación de usuarios, donde las personas que deseen ingresar al sistema podrán registrar sus datos teniendo como premisa la simplicidad y amigabilidad de la interfaz, por esta razón se toma la decisión de utilizar los componentes provistos por el framework Laravel para el desarrollo de los módulos de registro, autenticación y recuperación de contraseña, estos componentes fueron modificados para adaptarlos a las necesidades de la aplicación, Además, se desarrolló una página de bienvenida para los usuarios autenticados donde resalta la implementación de una barra de navegación lateral que servirá como una plantilla de diseño para toda la aplicación.

Se destaca de esta versión, el manejo de las sesiones de usuarios, la creación y diseño de los formularios para guardar los datos y el almacenamiento de los mismos en la base de datos. En la figura 5.1 se muestra una visualización de la interfaz de acceso al sistema, donde encontramos ciertos elementos que son recurrentes en la mayoría de las páginas de autenticación, esto con la finalidad de hacerlo lo más intuitivo posible a los usuarios que apenas ingresan al sistema.



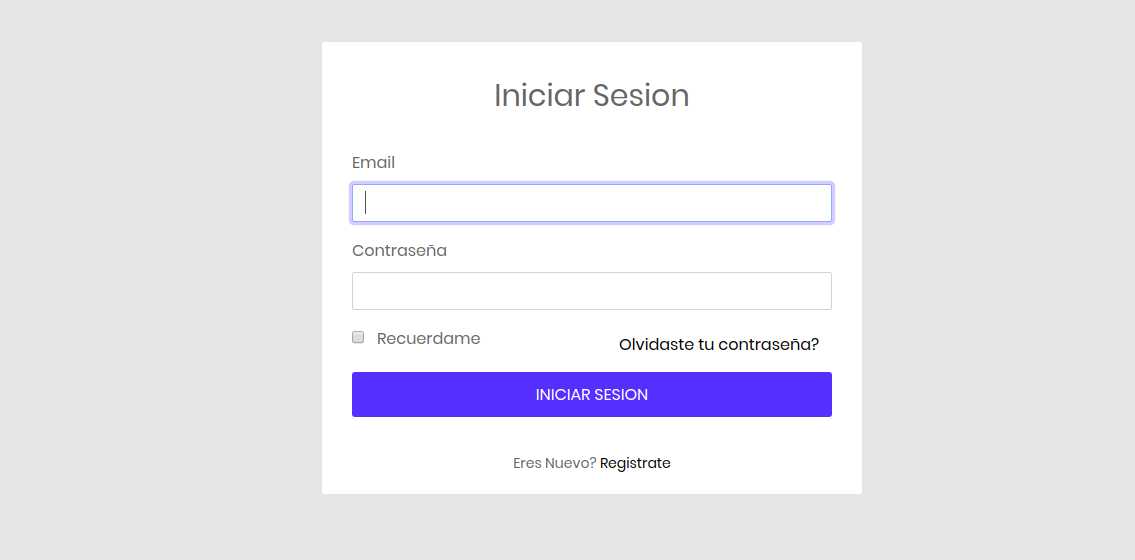



**Figura 5.1: Visualización de la interfaz de autenticación**

## 5.3 Segunda Iteración

Durante esta segunda iteración del desarrollo incremental, se realizó el módulo de gestión de usuarios para cumplir con los casos de uso especificados en las tablas 3.4 y 3.5, Se crearon formularios para la edición de usuarios y se realizaron las validaciones para cada uno de los campos del formulario en el controlador usando el método **validate**. Además, se adquirió el paquete ***shinobi*** para el manejo de los roles y permisos del sistema. Lo que permitirá a partir de esta iteración limitar los accesos al sistema según los roles que posea cada usuario mediante el uso de middlewares, e igualmente de manera visual haciendo uso de la función **can**. En la figura 5.2 se muestra una visualización general del módulo de gestión de usuarios, donde se aprecia como un administrador puede consultar, editar y eliminar cualquiera de los demás usuarios del sistema.

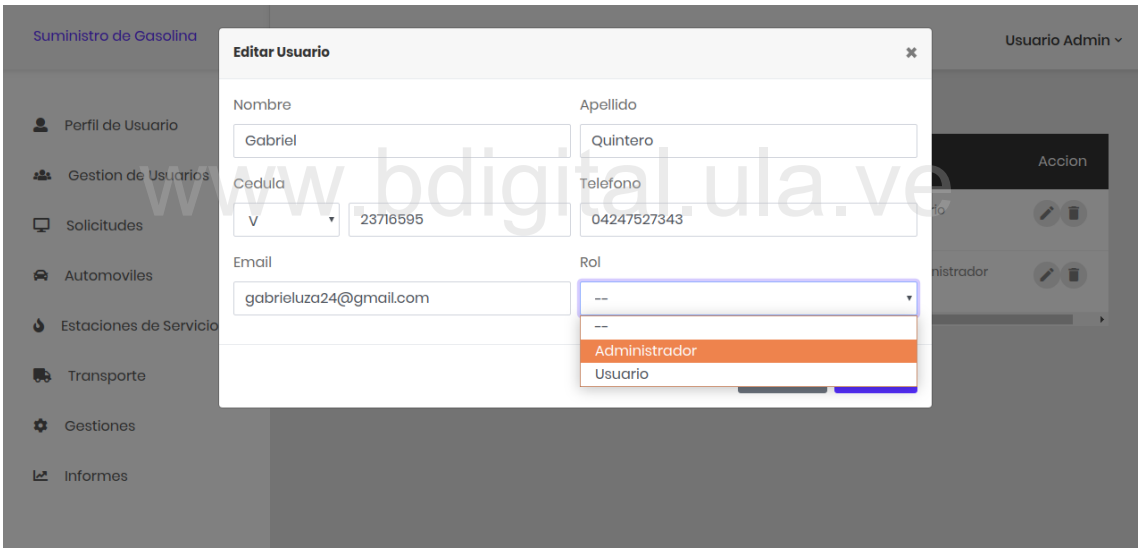



**Figura 5.2: Visualización de la interfaz de gestión de usuarios**

## 5.4 Tercera Iteración

En esta iteración del desarrollo incremental se realizó el módulo de gestión de automóviles, usando el patrón MVC del framework Laravel se construyeron controladores RESTful, que corresponden a las dependencias GET, POST, PUT, DELETE, PATCH. Los controladores se almacenan en el directorio de la aplicación App/Http/Controllers/, y para este módulo la estructura del controlador correspondiente a los automóviles se creó de la siguiente manera:

- App/Http/Controllers/cars/index
- App/Http/Controllers/cars/create
- App/Http/Controllers/cars/edit
- App/Http/Controllers/cars/show
- App/Http/Controllers/cars/store
- App/Http/Controllers/cars/update
- App/Http/Controllers/cars/destroy

Se empleó la función **__construct,** la cual funcionó como un contendor de inyección de dependencias ya que resuelve automáticamente todas las dependencias y las inyecta a instancias del controlador, esta se utilizó para emplear los middlewares y limitar los permisos a cada una de las funciones del controlador. También se realizaron las validaciones para cada uno de los campos del formulario en el controlador usando el método **validate**, una función provista por el framework la cual evalúa si el formulario cumple con las reglas de validación, en ese caso el código continuara ejecutándose normalmente, de lo contrario se lanza una excepción y se retorna un mensaje con el error. Este método nos permitió definir las reglas de validación para cada formulario y personalizar los mensajes de error.



En la figura 5.3 observamos el módulo de automóviles, donde se nos presenta una tabla con la información de cada uno de los automóviles registrados por el usuario, y botones con las acciones posibles para cada automóvil, correspondientes con su respectiva dependencia del controlador antes mencionada. Así se cumplió con los casos de uso especificados en las tablas 3.6 y 3.7 del análisis de requisitos.

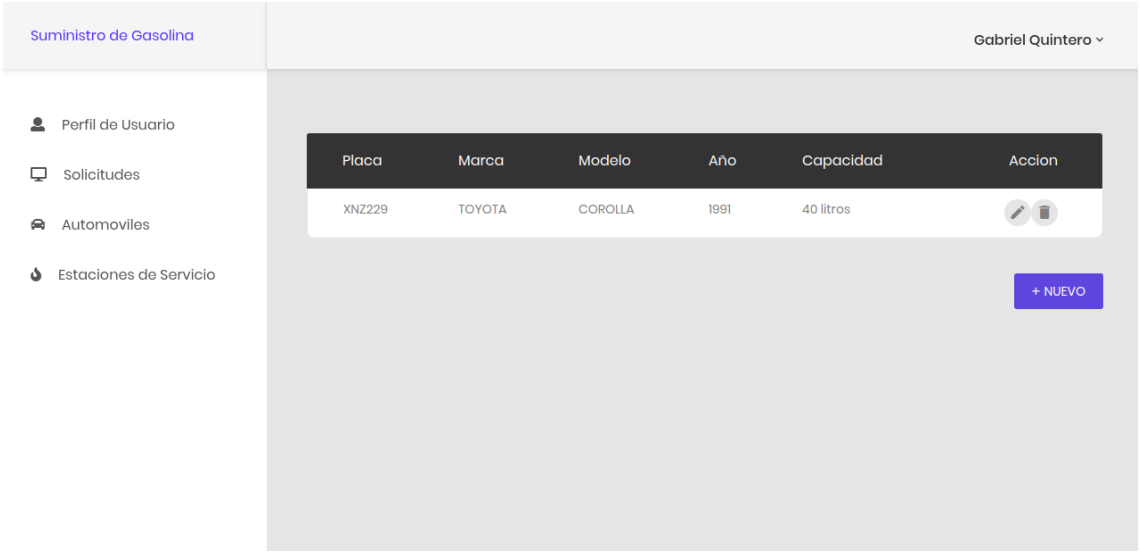



**Figura 5.3: Visualización de la interfaz de gestión de automóviles**

## 5.5 Cuarta Iteración



En esta iteración se desarrolló el módulo de gestión de estaciones de servicio y gestión de transporte, con el fin de brindarle al usuario toda la información que este requiera al momento de realizar su solicitud de combustible. Las estaciones de servicio juegan un rol fundamental para la solución del problema, es por ello que se determinó realizar una ficha de cada una de las estaciones disponibles, donde el administrador introduce información de cada una de las estaciones de servicio existentes para que los usuarios puedan observar estas fichas y determinar cuál se adapta mejor a sus necesidades, esta información también es importante para la toma de decisiones a posteriori. La estructura RESTful del controlador correspondiente quedó de la siguiente manera:

- App/Http/Controllers/EstacionesServicio/index
- App/Http/Controllers/EstacionesServicio/create
- App/Http/Controllers/EstacionesServicio/edit
- App/Http/Controllers/EstacionesServicio/show
- App/Http/Controllers/EstacionesServicio/store
- App/Http/Controllers/EstacionesServicio/update
- App/Http/Controllers/EstacionesServicio/destroy

Para el módulo de transporte, se desarrolló un formulario el cual permite ingresar toda la información relevante de un camión cisterna, incluyendo los datos del conductor. La estructura RESTful del controlador se creó de la siguiente manera:

- App/Http/Controllers/Transportes/index
- App/Http/Controllers/Transportes/create
- App/Http/Controllers/Transportes/edit
- App/Http/Controllers/Transportes/show
- App/Http/Controllers/Transportes/store
- App/Http/Controllers/Transportes/update
- App/Http/Controllers/Transportes/destroy

En ambos módulos, las reglas de validación respectivas para cada campo del formulario se realizaron en el controlador, usando el método **validate** del objeto Iluminate/Http/Request. Y se emplearon middleware para definir los permisos de acuerdo a cada uno de los roles del sistema. En la figura 5.4 se muestra la interfaz del módulo de estaciones de servicio para un usuario.



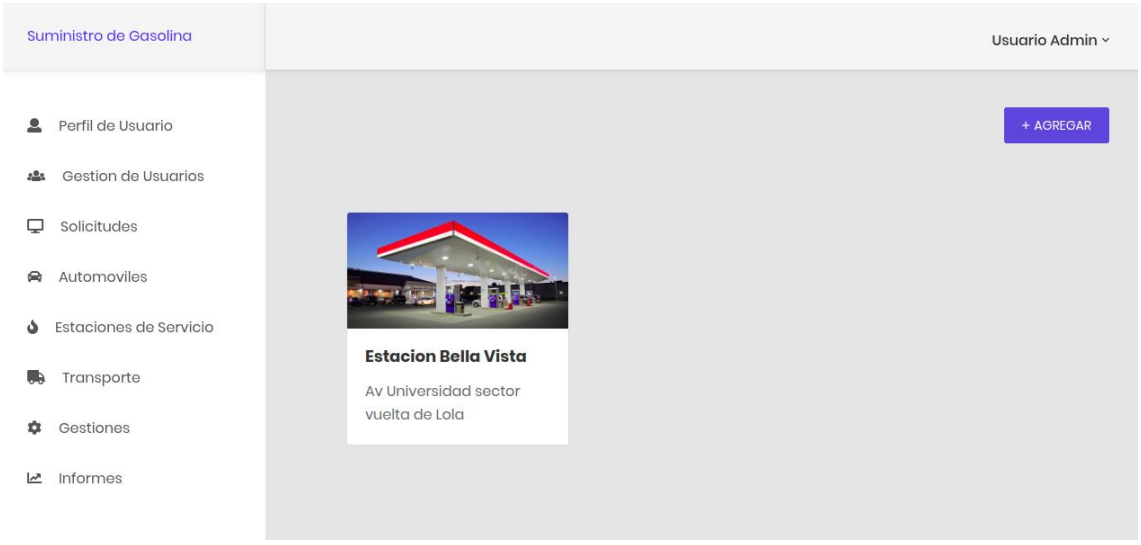



**Figura 5.4: Visualización de la interfaz de gestión de estaciones de servicio**

## 5.6 Quinta Iteración

Durante la quinta iteración se realizó el módulo de gestión de solicitudes. Partiendo de la premisa de obtener un sistema completo, pero de uso simple, se desarrolló el modulo mediante dos interfaces, la primera encargada de la creación de nuevas solicitudes, se realizó tomando las funcionalidades obtenidas en las iteraciones anteriores. Obteniendo un formulario en el que el usuario debe indicar cuál de los automóviles registrados en el sistema desea cargar de combustible y en cuál de las estaciones de servicio puestas a disposición. En la figura 5.5 se visualiza la interfaz mediante la cual el usuario realiza las solicitudes.


**Figura 5.5: Visualización de la interfaz para generar nuevas solicitudes**

Es importante resaltar que, aunque el formulario sea de selección se deben realizar las validaciones correspondientes para mantener la seguridad y la integridad de la aplicación en todo momento, también se valida que, si un automóvil tiene una solicitud en espera para una estación de servicio, no debe poder realizar más solicitudes hasta tanto no elimine la solicitud en proceso o la misma sea aprobada. Las reglas de validación se realizaron en el controlador, usando el método **validate** del objeto Iluminate/Http/Request.

La segunda interfaz del módulo de solicitudes, pone a disposición de todos los usuarios del sistema un mecanismo de consulta de las solicitudes, indicándoles el estatus de la misma y en caso de estar aprobada la fecha en la que deberá asistir a la estación de servicio. También permite eliminar alguna solicitud si esta aún no ha sido aprobada. Los administradores podrán observar una lista de todas las solicitudes del sistema. En la figura 5.6 observamos una visualización de la interfaz para el módulo de gestión de solicitudes.

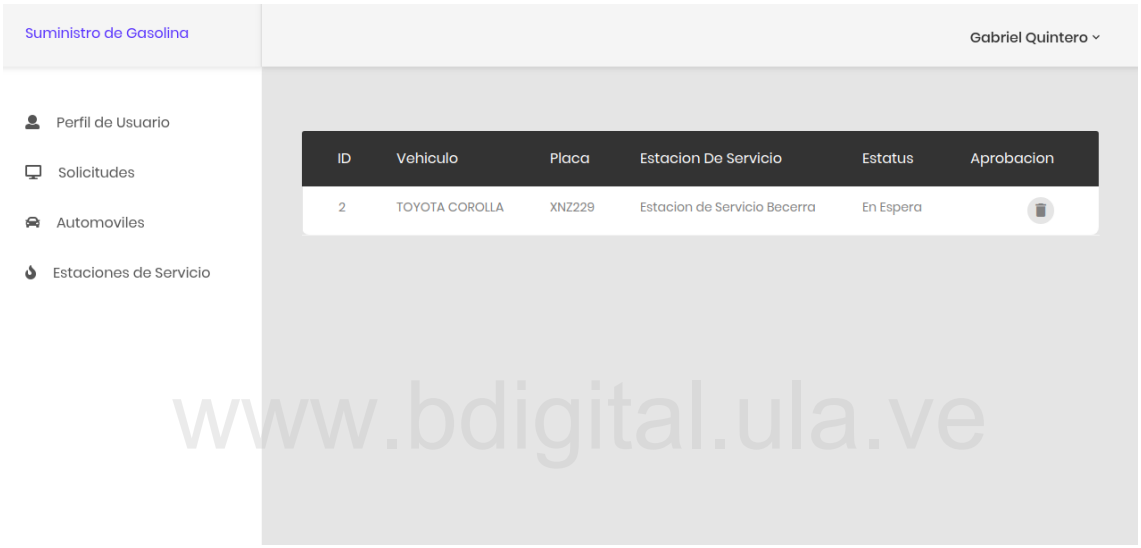





**Figura 5.6: Visualización de la interfaz de consulta de solicitudes**

## 5.7 Sexta Iteración

En esta iteración del proceso de desarrollo se lograron concluir todos los casos de uso definidos en los requerimientos del proyecto mediante la realización del módulo para la gestión y asignación de combustible. El cual consistió en un formulario que establece envíos de combustible a las estaciones de servicio indicando una fecha estimada de arribo y el transporte que realiza él envió. Al igual que en todas las demás iteraciones las validaciones para cada campo del formulario se realizaron en el controlador, usando el método **validate**. En la figura 5.7 observamos el formulario para la creación de un nuevo envió de combustible.

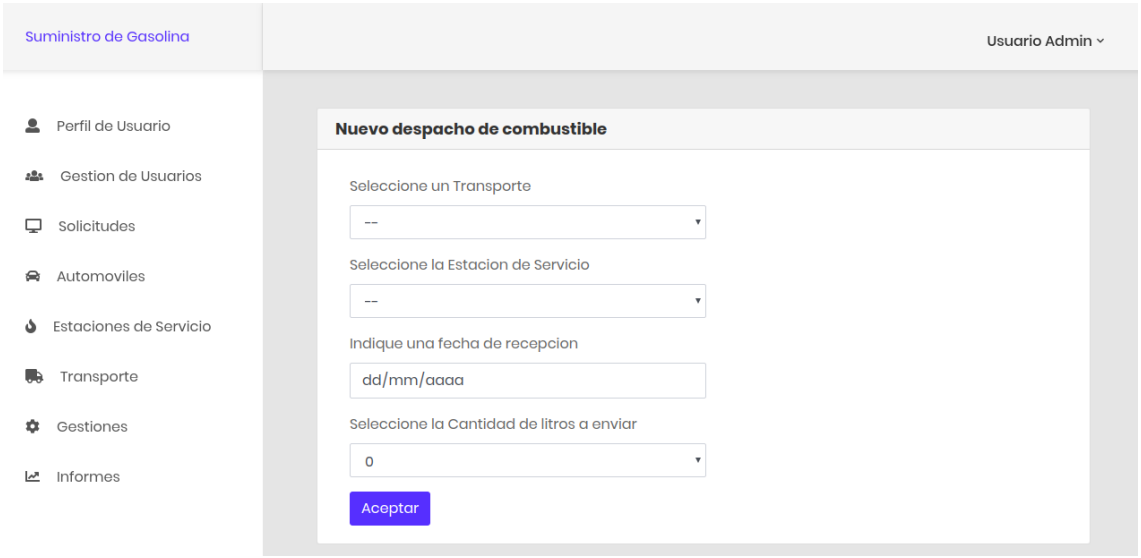



**Figura 5.7 Visualización de la interfaz del módulo de gestión de combustible**



Durante el desarrollo de esta iteración también se desarrolló un módulo de informes que les permite a los administradores visualizar todos los envíos de combustible realizados, con la finalidad de obtener información verídica sobre la oferta de combustible.

Una vez aprobado él envió del combustible a una estación de servicio, el sistema debe asignar de manera automática esa cantidad de combustible a los automóviles que posean una solicitud en espera para la dicha estación, en caso de ser insuficiente el combustible para la cantidad de solicitudes, se deben comparar ciertos de criterios como la antigüedad de la fecha de la solicitud para realizar la asignación. Laravel permite la creación de comandos personalizados que pueden ser ejecutados desde la línea de comandos de una terminal del sistema, se creó un comando que realiza la asignación de combustible, dicha tarea debe ejecutarse de manera automática. En la figura 5.8 observamos una lista con los envíos de combustible que aun poseen combustible sin asignar cuyo valor ira disminuyendo a medida que surjan nuevas solicitudes.

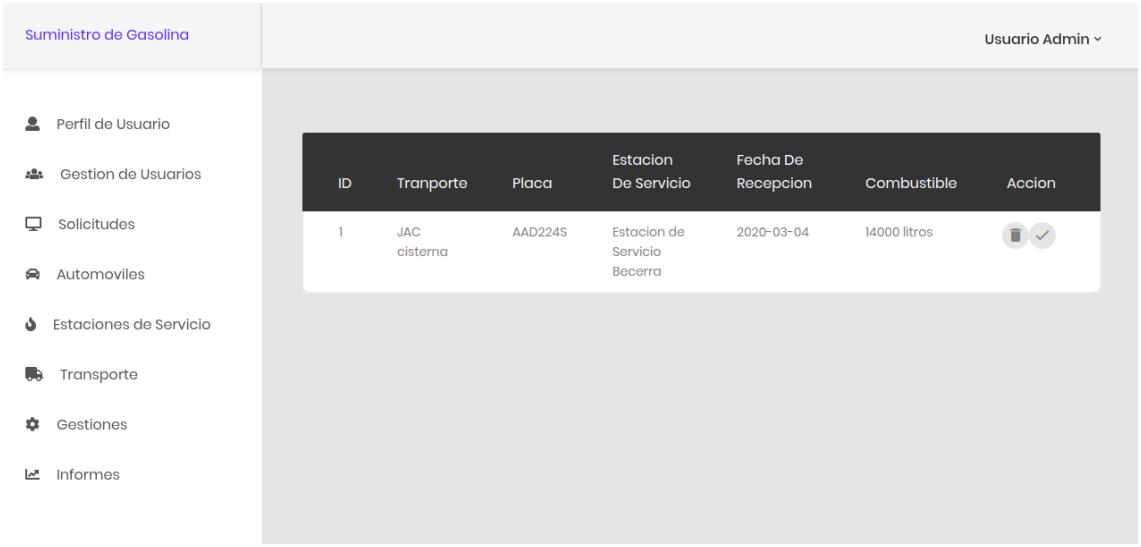



**Figura 5.8: Visualización de la interfaz de asignación de combustible**

De esta manera se cumplieron todos los requerimientos funcionales definidos para este proyecto, aplicando el enfoque evolutivo el cual consistió de seis (06) iteraciones, las cuales comenzaron con un módulo de autenticación de usuarios, que permitió ingresar los datos de los usuarios al sistema dándole paso a un módulo de gestión de usuarios donde se le permitió a los administradores asignar o quitar permisos a ciertos usuarios de acuerdo al rol que estos ocupasen. Se avanzó con las siguientes iteraciones aplicadas de manera metódica, realizando primero aquellos módulos que eran necesarios para la elaboración de los demás, tal es el caso del módulo de solicitudes el cual requirió de la existencia de los componentes de gestión de automóviles y el de gestión de estaciones de servicio antes de poder ser creado. La última iteración se enfocó más en el desarrollo de mecanismos para atender a las solicitudes de los usuarios, teniendo presente en cada una de las iteraciones la validación de los formularios y los permisos correspondientes para cada uno de los perfiles de usuarios. Cumpliendo así con los objetivos planteados por los casos de uso definidos en el desarrollo de requerimientos.

# Capítulo 6

# Pruebas del Sistema

El método White WATCH, contempla la realización de pruebas al sistema en cada una de sus versiones con el objetivo detectar errores en el código e implementación y corregirlos, reduciendo así la cantidad de fallos y comprobando que el sistema cumple con los requisitos.

## 6.1 Mecanismos de prueba

El framework Laravel permite la inclusión de un paquete llamado ***Laravel Dusk*** el cual consiste un sistema de pruebas cuyo objetivo es proporcionar una forma de lograr pruebas de interacción dentro del navegador, tales como hacer clic, interactuar con formularios y modificar la base de datos tal como lo haría un usuario, es por esta razón que se utilizó este sistema para crear pruebas funcionales automatizadas.



## 6.2 Casos de prueba

Se realizaron una serie de casos de pruebas funcionales, en los cuales se tomaron algunas de las funcionalidades del sistema con un resultado esperado, si la prueba concluye con el resultado esperado, se tomará como una prueba exitosa y se determinará como correcta, de lo contrario, debe evaluarse el sistema hasta encontrar el origen de la falla, corregirse y probarse nuevamente. Las pruebas se darán por finalizadas cuando se garantice la funcionalidad esperada. En la tabla 6.1 se muestra el plan de pruebas.

**Tabla 6.1: Plan de pruebas funcionales**

| Subsistema | Parámetros de Entrada | Código | Salida Esperada |
|---|---|---|---|
| Inicio de sesión | Datos correctos | Código 6.1 | Usuario valido. Retorna a la página principal. |
| Inicio de sesión | Usuario o contraseña inválidos | Código 6.1 | Datos incorrectos. Regreso al formulario, mensaje de error en el campo incorrecto. |
| Inicio de sesión | Usuario o contraseña en blanco | Código 6.1 | Regreso al formulario, mensaje de campo obligatorio en el campo correspondiente. |
| Gestión de Automóviles | Datos correctos | Código 6.2 | Mensaje de éxito, retorna a la lista de automóviles. |
| Gestión de Automóviles | Número de placa existente | Código 6.2 | Mensaje de error. |
| Gestión de Estaciones de Servicio | Datos correctos | Código 6.3 | Mensaje de éxito, retorna a vista de estaciones. |
| Gestión de Estaciones de Servicio | RIF o nombre en blanco | Código 6.3 | Mensaje de error. |
| Gestión de Solicitudes | Datos correctos | Código 6.4 | Mensaje de éxito, retorna a la lista de solicitudes. |
| Gestión de Solicitudes | Automóvil con solicitud en espera | Código 6.4 | Mensaje de error, retorna a formulario de solicitudes |

## 6.3 Pruebas Realizadas

A continuación, se detalla cada una de las pruebas realizadas, siguiendo el plan de pruebas se desarrolló un archivo de prueba para cada uno de los subsistemas, en las cuales se consideró cada uno de los parámetros de entrada propuestos, los cuales arrojaron una captura de pantalla del resultado obtenido. Este resultado se comparó con la salida esperada.

### 6.3.1 Inicio de Sesión

Se probó el componente de inicio de sesión para un usuario mediante tres pruebas, En el código 6.1, podemos observar como en la prueba primero se direcciono el navegador a la página de inicio de sesión, luego se introdujeron datos inválidos, posteriormente se regresó a la página de inicio de sesión nuevamente esta vez dejando los campos en blanco, y finalmente se ingresó al sistema haciendo uso de las credenciales del usuario por defecto que posee el sistema.



```
1. <?php
2.
3. namespace Tests\Browser;
4.
5. use Illuminate\Foundation\Testing\DatabaseMigrations;
6. use Laravel\Dusk\Browser;
7. use Tests\DuskTestCase;
8.
9. class LoginTest extends DuskTestCase
10.      {
11.          /**
12.           * A Dusk test example.
13.           *
14.           * @return void
15.           */
16.          public function testLogin()
17.          {
18.              $this->browse(function (Browser $browser) {
19.                  $browser->visit('/login')
20.                      ->type('email','nodata@gmail.com')
21.                      ->type('password','holamundo')
22.                      ->press('INICIAR SESION')
23.                      ->pause('400')
24.                      ->screenshot('login1');
25.              });
```

```
26.
27.             $this->browse(function (Browser $browser) {
28.                 $browser->visit('/login')
29.                     ->press('INICIAR SESION')
30.                     ->pause('400')
31.                     ->screenshot('login2');
32.             });
33.
34.             $this->browse(function (Browser $browser) {
35.                 $browser->visit('/login')
36.                     ->type('email','admin@gmail.com')
37.                     ->type('password','00000000')
38.                     ->press('INICIAR SESION')
39.                     ->pause('500')
40.                     ->screenshot('login3')
41.                     ->assertAuthenticated();
42.             });
43.
44.         }
45.     }
```

**Código 6.1: Prueba de inicio de sesión.**

El resultado de esta prueba fue capturado empleando el método **screenshot**. Se observa en las figuras 6.1, 6.2 y 6.3 los resultados correspondientes para cada uno de los parámetros de entrada.



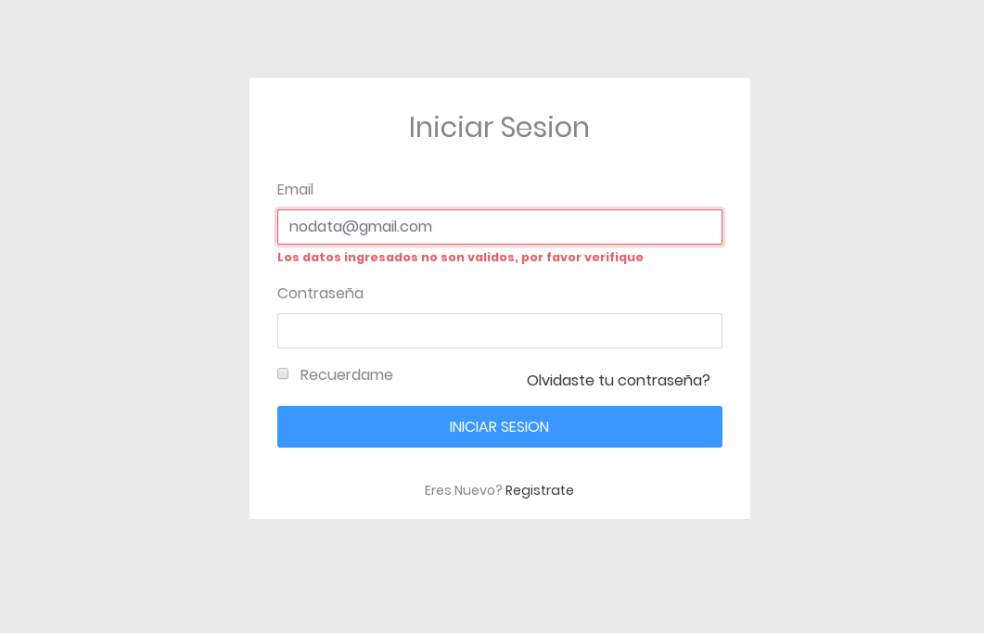



**Figura 6.1: Resultado de la prueba de inicio de sesión con datos inválidos**

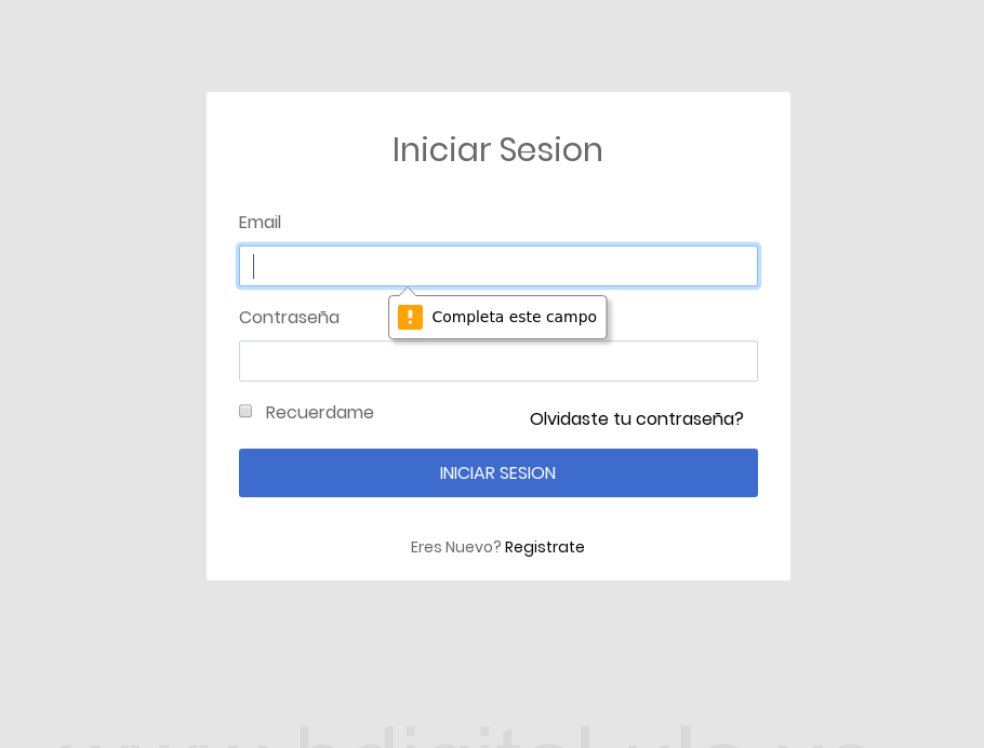





**Figura 6.2: Resultado de la prueba de inicio de sesión con datos en blanco**

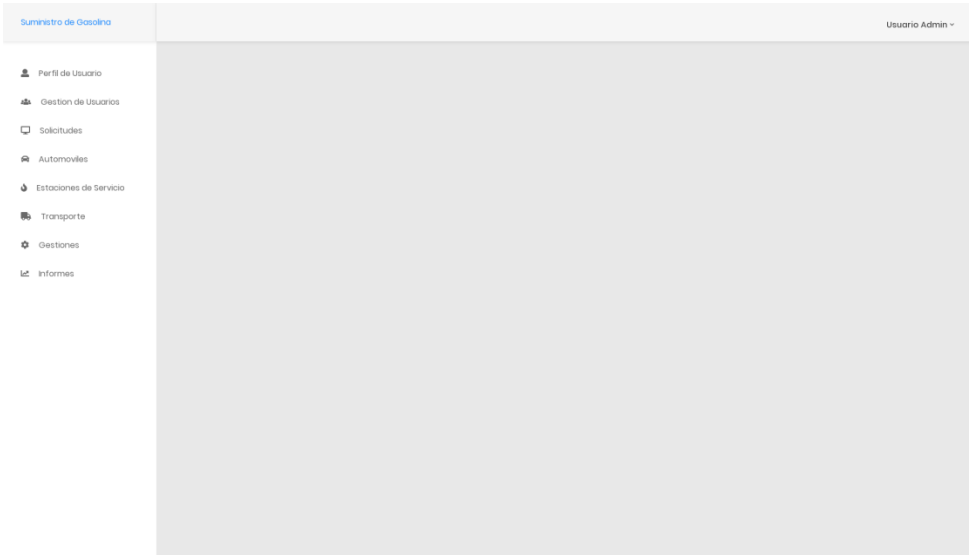



**Figura 6.3: Resultado de la prueba de inicio de sesión con datos correctos**

### 6.3.2 Gestión de Automóviles

Para el subsistema de gestión de automóviles, se realizaron dos pruebas. En el código 6.2 se muestra como primero se autentica el usuario empleando el método **LoginAs**. Luego se procedió a direccionar el navegador al módulo de automóviles, para finalmente ingresar datos correctos de un automóvil mediante el uso de la ventana modal, posteriormente se procede a tratar de ingresar los mismos datos nuevamente esperando una respuesta diferente en esta oportunidad ya que se estaría repitiendo la misma placa para dos automóviles.

```
1. <?php
2.
3. namespace Tests\Browser;
4.
5. use Illuminate\Foundation\Testing\DatabaseMigrations;
6. use Laravel\Dusk\Browser;
7. use Tests\DuskTestCase;
8.
9. class AutomovilesTest extends DuskTestCase
10.      {
11.          /**
12.           * A Dusk test example.
13.           *
14.           * @return void
15.           */
16.          public function testAutomoviles()
17.          {
18.
19.              $this->browse(function (Browser $browser) {
20.                  $user = \App\User::FindOrFail(2);
21.                  $browser->loginAs($user);
22.                  $browser->visit('/cars')
23.                      ->press('NUEVO')
24.                      ->waitFor('#crearAuto','1')
25.                      ->whenAvailable('#crearAuto', function ($modal)
26.                  {
27.                          $modal
28.                          ->type('placa','XNZ229')
29.                          ->type('marca','Toyota')
30.                          ->type('modelo','Corolla')
31.                          ->select('anno','1991')
32.                          ->select('capacidad','40')
33.                          ->press('Guardar');
34.
35.                  })
36.                      ->pause('400')
37.                      ->screenshot('Auto1')
38.                      ->assertSee('El Automovil Fue Agregado');
```

```
39.                 });
40.
41.                 $this->browse(function (Browser $browser) {
42.                      $browser->visit('/cars')
43.                         ->press('NUEVO')
44.                         ->waitFor('#crearAuto','1')
45.                         ->whenAvailable('#crearAuto', function ($modal)
46.                     {
47.                             $modal
48.                             ->type('placa','XNZ229')
49.                             ->type('marca','Ford')
50.                             ->type('modelo','Focus')
51.                             ->select('anno','2006')
52.                             ->select('capacidad','50')
53.                             ->press('Guardar');
54.                     })
55.                         ->pause('400')
56.                         ->screenshot('Auto2')
57.                         ->assertSee('Este automovil ya ha sido
    registrado');
58.                 });
59.             }
60.         }
```

**Código 6.2: Prueba de gestión de automóviles.**



Se utilizó el método **screenshot**, para tomar registro de los resultados obtenidos y en la figura 6.4 y 6.5 se muestra como los resultados de la prueba fueron exitosos

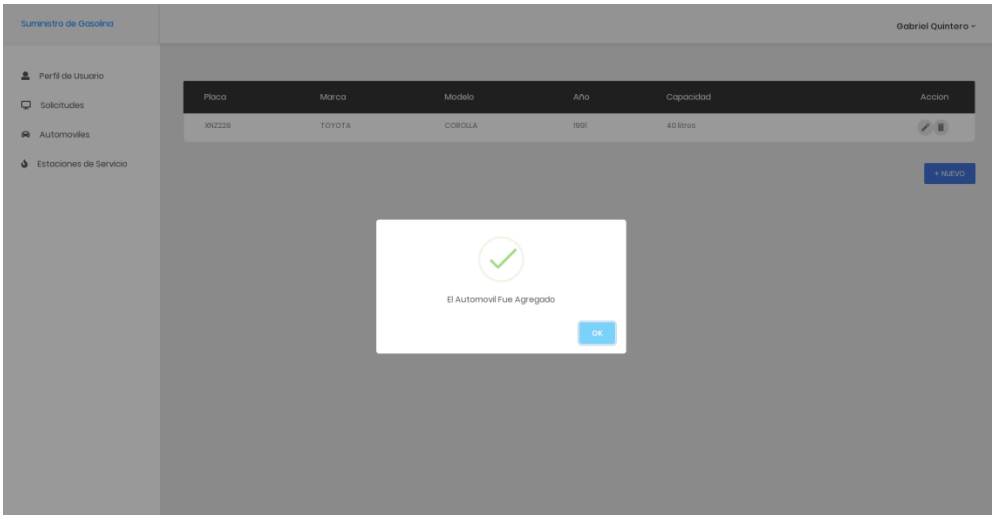



**Figura 6.4: Resultado de la prueba de gestión de automóviles con datos correctos**

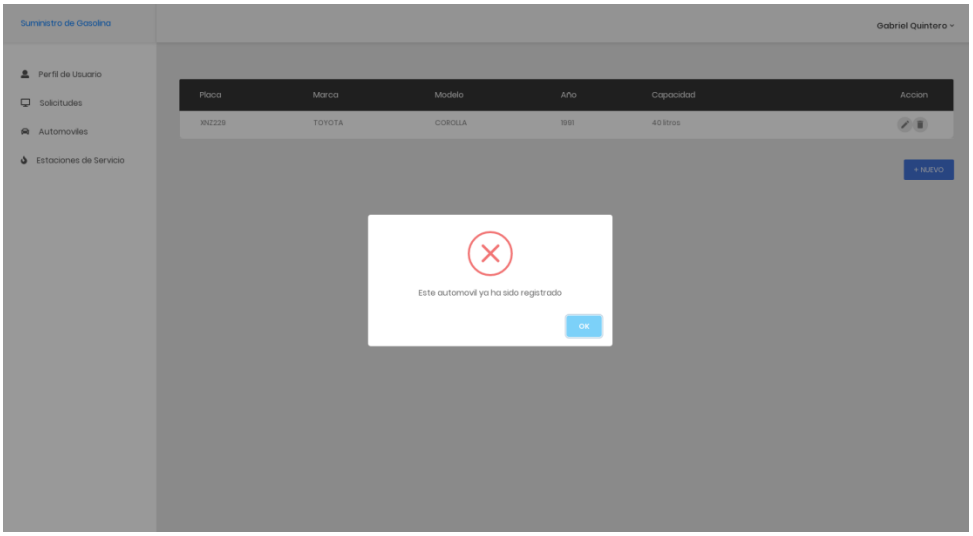



**Figura 6.5: Resultado de la prueba de gestión de automóviles con datos inválidos**

### 6.3.3 Gestión de Estaciones de Servicio



En el subsistema de gestión de estaciones de servicio también se realizaron dos pruebas. En el código 6.3 se puede observar como para esta prueba se ingresó al sistema haciendo uso del método **LoginAs**, para luego ser direccionados a la vista de estaciones, una vez allí se intentó registrar una nueva estación de servicio dejando el campo RIF en blanco, esperando un error por parte del sistema y posteriormente se procedió a completar todos los campos del formulario.

```
1. <?php
2.
3. namespace Tests\Browser;
4.
5. use Illuminate\Foundation\Testing\DatabaseMigrations;
6. use Laravel\Dusk\Browser;
7. use Tests\DuskTestCase;
8.
9. class EstacionesTest extends DuskTestCase
10.      {
11.          /**
12.           * A Dusk test example.
13.           * @return void
14.           */
15.          public function testExample()
16.          {
```

```
17.             $this->browse(function (Browser $browser) {
18.                 $user = \App\User::FindOrFail(1);
19.                 $browser->loginAs($user);
20.                 $browser->visit('/estaciones')
21.                     ->press('AGREGAR')
22.                     ->waitFor('#crearEstacion','1')
23.                     -
   >whenAvailable('#crearEstacion', function ($modal)
24.                 {
25.                         $modal
26.                         ->type('rif','')
27.                         ->type('nombre','Estacion Bella Vista')
28.                         ->type('direccion','Av Universidad sector
   vuelta de Lola')
29.                         ->select('surtidores','2')
30.                         ->select('capacidad','144000')
31.                         ->press('Guardar');
32.
33.                 })
34.                     ->pause('400')
35.                     ->screenshot('Estacion1')
36.                     ->assertSee('El campo rif es requerido.');
37.             });
38.
39.             $this->browse(function (Browser $browser) {
40.                 $browser->visit('/estaciones')
41.                     ->press('AGREGAR')
42.                     ->waitFor('#crearEstacion','1')
43.                     -
   >whenAvailable('#crearEstacion', function ($modal)
44.                 {
45.                         $modal
46.                         ->type('rif','J-00304500-6')
47.                         ->type('nombre','Estacion Bella Vista')
48.                         ->type('direccion','Av Universidad sector
   vuelta de Lola')
49.                         ->select('surtidores','2')
50.                         ->select('capacidad','144000')
51.                         ->press('Guardar');
52.
53.                 })
54.                     ->pause('400')
55.                     ->screenshot('Estacion2')
56.                     ->assertSee('La estacion de servicio Fue
   Agregada.');
57.             });
58.
59.         }
60.     }
```

**Código 6.3: Prueba de gestión de estaciones de servicio.**

En las figuras 6.6 y 6.7 observamos los resultados exitosos para los diferentes parámetros de entrada, obtenidos mediante el método ***screenshot***.

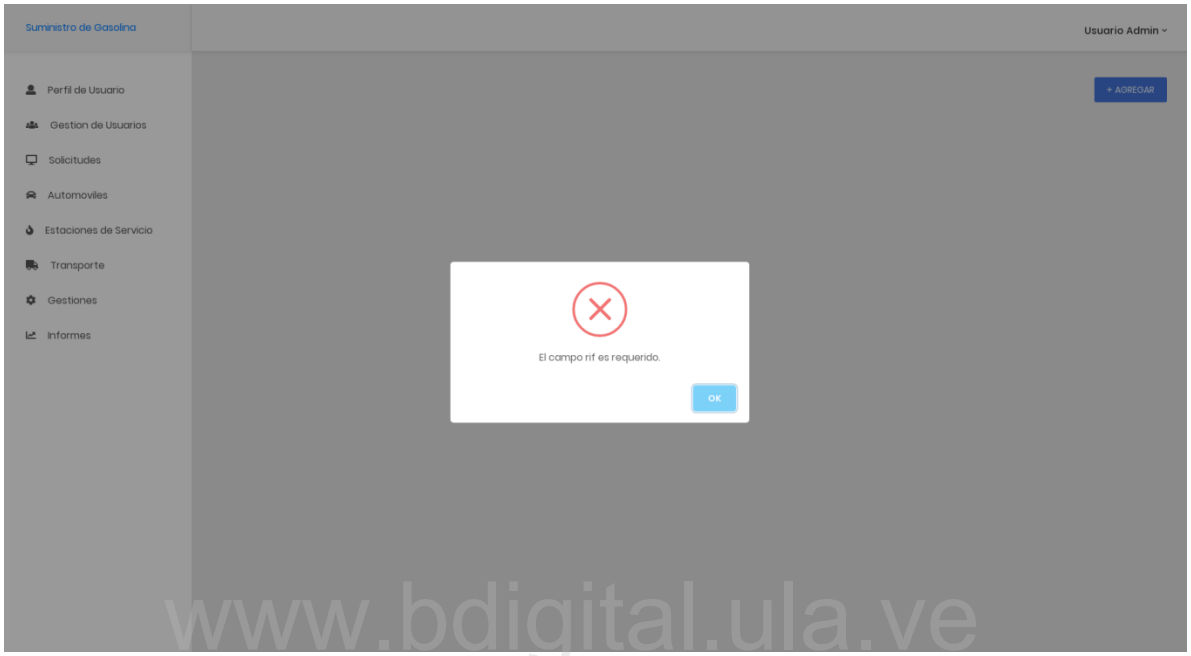





**Figura 6.6: Resultado de la prueba de gestión de estaciones de servicio con campos vacíos**

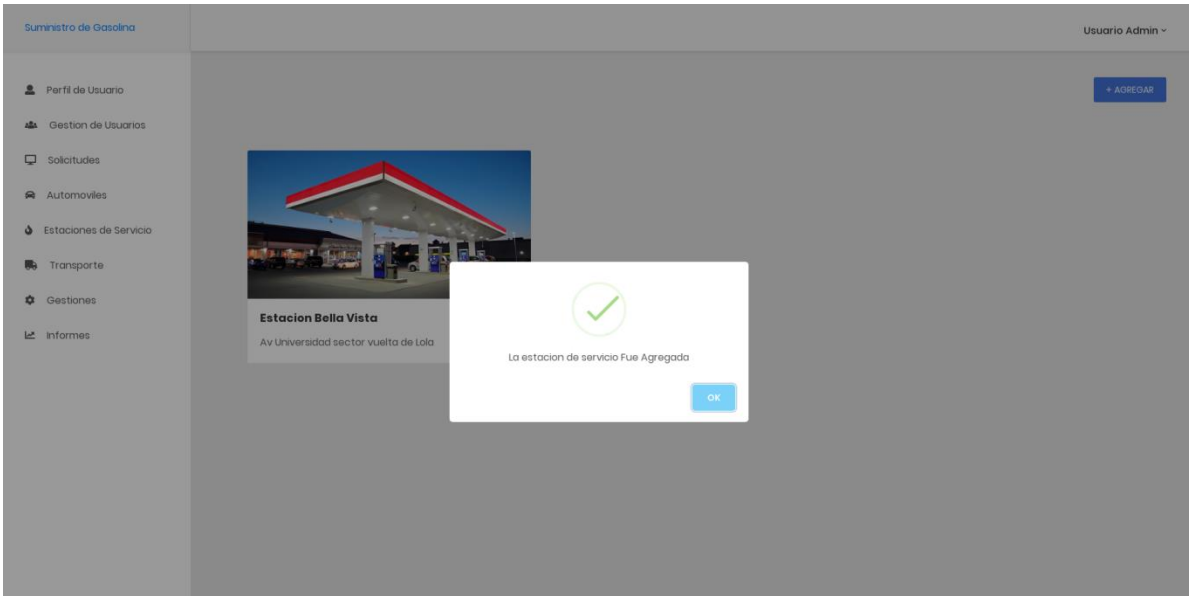



**Figura 6.7: Resultado de la prueba de gestión de estaciones de servicio con datos correctos**

### 6.3.4 Gestión de Solicitudes

Para la gestión de solicitudes se plantean los mismos parámetros de entrada, esperando resultados diferentes en cada ejecución, esto ocurre porque un automóvil solo puede tener una solicitud en espera al mismo tiempo. En el código 6.4 se observa como utilizando los parámetros creados en las pruebas anteriores se procede a crear una nueva solicitud de combustible esperando ser retornados a diferentes rutas mediante los métodos **AssertPathIs** y **AssertPathIsNot**.

```php
<?php

namespace Tests\Browser;
use Illuminate\Foundation\Testing\DatabaseMigrations;
use Laravel\Dusk\Browser;
use Tests\DuskTestCase;

class SolicitudesTest extends DuskTestCase
{
        /**
         * A Dusk test example
         */
        public function testExample()
        {
            $this->browse(function (Browser $browser) {
                $user = \App\User::FindOrFail(2);
                $browser->loginAs($user);
                $browser->visit('/solicitudes/create')
                    ->select('car','XNZ229')
                    ->select('estacion','J-00304500-6')
                    ->press('Aceptar')
                    ->pause('400')
                    ->screenshot('Solicitud1')
                    ->assertPathIs('/solicitudes');
            });
            $this->browse(function (Browser $browser) {
                $browser->visit('/solicitudes/create')
                    ->select('car','XNZ229')
                    ->select('estacion','J-00304500-6')
                    ->press('Aceptar')
                    ->pause('400')
                    ->screenshot('Solicitud2')
                    ->assertPathIsNot('/solicitudes');
            });
        }
    }
```

**Código 6.4: Prueba de gestión de solicitudes.**

En las figuras 6.8 y 6.9 observamos los resultados exitosos de la prueba, obtenidos mediante el método **screenshot**. Destacando los mensajes de éxito y error para la misma solicitud hasta tanto no se elimine o apruebe la anterior.

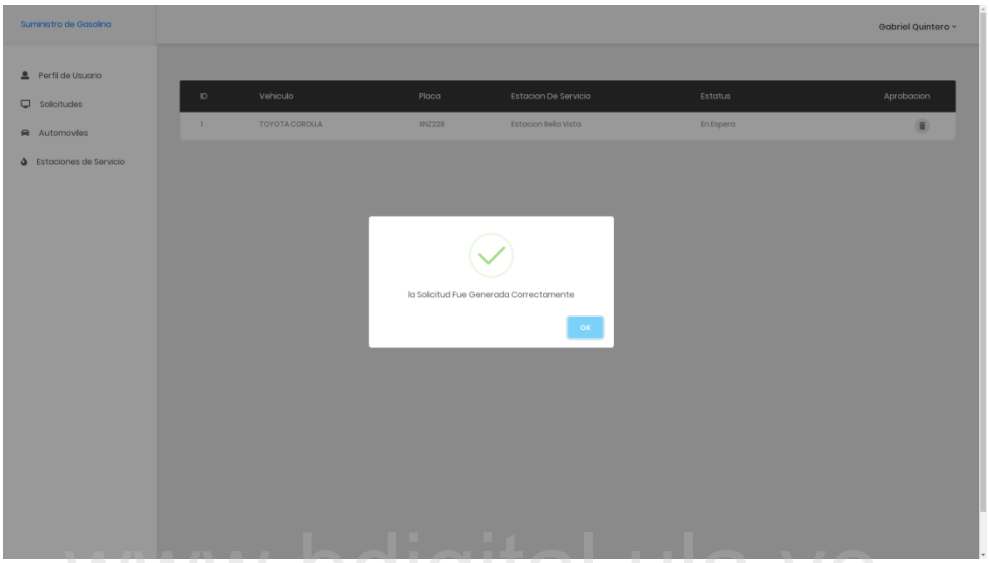





**Figura 6.8: Resultado de prueba de gestión de solicitudes**

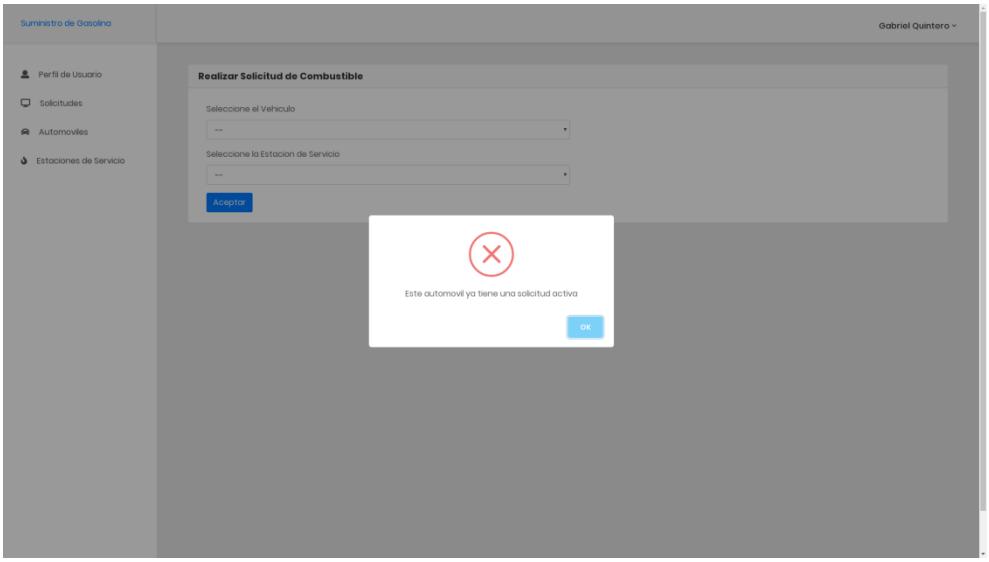



**Figura 6.9: Resultado de prueba de gestión de solicitudes con datos repetidos**

# Capítulo 7

# Conclusiones y Recomendaciones

## 7.1 Conclusiones

Venezuela, un país sumergido en una profunda crisis económica, social y política, sin embargo aún mantiene el precio del combustible más económico del mundo lo cual nos presenta un escenario donde los conductores de automóviles deben esperar largos periodos de tiempo a las afueras de las estaciones de servicio, sin ninguna seguridad de obtener el combustible necesario para realizar sus actividades diarias, por esta razón surge la idea de desarrollar una aplicación web con la finalidad de optimizar y controlar los procesos de suministro de combustible. Para ello se usó el método de desarrollo White WATCH, el cual nos provee una metodología de trabajo bien definida donde se incluye una serie de procesos técnicos para el desarrollo de software como la ingeniería de requisitos, el diseño de la arquitectura, la implementación del sistema y las pruebas del mismo, este modelo de desarrollo balancea la producción de especificaciones en la medida que se avanza en el desarrollo, por lo que cada uno de los procesos fueron refinados a lo largo de las versiones de desarrollo.



La ingeniería de requisitos permitió tanto identificar y analizar los problemas de información del contexto, como especificar mediante la elaboración de casos de uso cada uno de los aspectos funcionales de la aplicación. Para el diseño del sistema se definió una estructura inicial mediante la continuidad directa de los modelos de análisis de requerimientos donde se relacionaron cada uno de los requisitos obtenidos con la arquitectura del sistema. Para facilitar la integración de la arquitectura del software con el diseño de la interfaz y los modelos de datos, se dividió el sistema en subsistemas, obteniendo así una especificación completa del diseño.

El desarrollo de las versiones de software y la ejecución de las pruebas de validación automatizadas permitieron ir refinando los requerimientos del sistema, sin salirse de los parámetros o alcance definido para el proyecto. El uso del framework Laravel para la implementación de esta aplicación fue clave ya que el uso de sus componentes o métodos ***helpers*** facilitó el trabajo de integración y simplificó el proceso de realizar consultas complejas a la base de datos a través de su ORM. El uso de Composer, nos permitió gestionar las dependencias del proyecto e instalar paquetes que fueron de gran ayuda para el desarrollo del mismo.

Los objetivos del proyecto fueron abarcados y cumplidos, bien sea durante la fase de estudio o durante el desarrollo del software, logrando diseñar e implementar una aplicación web para el control de suministro de combustible.

## 7.2 Recomendaciones

Según Peter Drunkel "Innovar es encontrar nuevos o mejorados usos a los recursos de que ya disponemos". En la búsqueda de continuar innovando se hacen una serie de recomendaciones orientadas a la mejora de la calidad del software y de las funciones que ofrece:

- Promover el desarrollo de la metodología White WATCH como marco metodológico para la elaboración de software en equipos de desarrollo pequeños.

- Incluir un módulo de noticias en la página principal de la aplicación que le permita a todos los usuarios del sistema estar al tanto de cualquier eventualidad que pueda ocurrir con respecto a sus asignaciones.

- Ampliar la creación de estadísticas en el sistema, con la finalidad de facilitar el análisis y encontrar soluciones a largo plazo para el suministro de combustible.



- Agregar la funcionalidad de descarga de datos, para su aprovechamiento fuera de la plataforma.

- Elaborar un manual de usuario para proporcionar la información adecuada para el correcto uso de esta aplicación.

# Bibliografía

The World Factbook, Country comparison crude oil-proved reserves, (2018), ubicación: https://www.cia.gov/library/publications/the-world-factbook/fields/264rank.html#VE

Peters, S., (2019), Sociedades rentistas: Claves para entender la crisis venezolana, Revista Europea de Estudios Latinoamericanos y del Caribe. No 108

Purón-Cid, G., Gil-García, J., (2013), Análisis de políticas públicas y tecnologías de información: oportunidades y retos para América Latina y el Caribe, Revista del CLAD Reforma y Democracia. No. 55.



Carabali, G., Moyano, C., (2016), Aplicación para la Gestión de Abastecimiento de Combustible que Brinda la Empresa distribuidora Levox a gasolineras Mobil, Universidad Politécnica Salesiana, Guayaquil, Ecuador.

Santelices R., (2007), Propuesta de un Sistema de Planificación para el Abastecimiento de Combustible, Universidad de Talca, Chile

Domínguez L., (2014), Sistema de Administración de las Operaciones de Marcación de Combustible en Petroecuador E.P., Universidad Israel, Quito, Ecuador.

Barrios, J., Montilva, J., White-WATCH: Método para desarrollo de proyectos pequeños y poco complejos, Versión 1.2, Informe técnico, Proyecto Methodius, 2010, Recuperado desde http://www.methodius.info.ve

Silberschatz, A., Korth, H., and S, S., (2002), Fundamentos de Bases de Datos, Madrid, España.

Unified Modeling Language, Introduction to omg's Unified Modeling Language, https://www.uml.org/what-is-uml.htm

Junta de Andalucía, (2014), Patrón Modelo Vista Controlador, URL: http://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/122

García, F., Moreno, M., García, A., (2018), UML. Unified Modeling Language, Universidad de Salamanca, España.

Gutierrez, J., (2012), ¿Qué es un Framework Web?, URL: http://www.cssblog.es/guias/Framework.pdf



CakePhp, Entendiendo Modelo-Vista-Controlador, URL:https://book.cakephp.org/1.3/es/The-Manual/Beginning-With-CakePHP/Understanding-Model-View-Controller.html

The PHP Group, Manual de PHP, URL: www.php.net/manual/es

Read the Docs, (2019), Laravel Guide, URL:https://laravel-guide.readthedocs.io/en/latest/

Gitbook, Laravel 5, URL:https://richos.gitbooks.io/laravel-5/

Flowchart Maker & Online Diagram Software, URL:https//www.draw.io