Proyecto de Grado

# IMPLEMENTACIÓN DE UNA INTERFAZ WEB CON CONEXIÓN SEGURA PARA LA EVALUACIÓN REMOTA DE CÓDIGO DE PROGRAMACIÓN



Por

Br. Jesús Humberto Mazzei Fontiveros

Tutor: Prof. MSc. Rodolfo Sumoza

Noviembre 2017

# IMPLEMENTACIÓN DE UNA INTERFAZ WEB CON CONEXIÓN SEGURA PARA LA EVALUACIÓN REMOTA DE CÓDIGO DE PROGRAMACIÓN

Br. Jesús Humberto Mazzei Fontiveros

Proyecto de Grado — Sistemas Computacionales, 74 páginas

**Resumen:** En la escuela de Ingeniería de Sistemas se requiere de una plataforma que permita implementar herramientas que hagan posible a los estudiantes presentar de manera segura y remota evaluaciones propias de la carrera. En este proyecto se plantea la integración de herramientas web existentes con el fin de implementar una interfaz que les permita realizar evaluaciones vinculadas a la compilación y ejecución de código.

**Palabras clave:** Interfaz Web, Seguridad, Evaluación Remota de Código

# Índice

# Índice de Figuras

# Capítulo 1

# Introducción

La emergencia de nuevas tecnologías de comunicación han permitido que cada día más actividades sean realizadas de manera remota a través del Internet, lo cual ha hecho factible que las instituciones educativas se logren adaptar a las nuevas exigencias en cuanto al tema de la enseñanza y el aprendizaje. Varias universidades a nivel mundial también han sabido aprovechar esta emergencia incorporando este tipo de herramientas para lograr ajustarse a los nuevos retos. En el caso nacional, específicamente en la Universidad de Los Andes, dada esta realidad, se tiene la necesidad de implementar plataformas que ofrezcan los recursos necesarios para adaptarse a los nuevos requisitos de la educación superior.



Con la emergencia de herramientas para la educación a distancia, se han presentado nuevos retos. Uno de estos es la evaluación. Para que un alumno pueda ser calificado sobre el conocimiento aprendido necesita ser evaluado de manera correcta. Esto requiere que exista una plataforma que permita al estudiante realizar dicha evaluación. de manera segura y estable. Aunado a esto la plataforma debe permitirle al profesor corroborar que la persona que presentó la evaluación sea quien dice ser.

En este trabajo se integraron herramientas web existentes para implementar una interfaz web que permite a estudiantes realizar evaluaciones de códigos de programación a distancia y con conexión segura, utilizando mecanismos de verificación de identidad.

## 1.1 Marco Teórico

Debido a la naturaleza de este proyecto y a sus requisitos, es necesario estar familiarizado con algunos términos que forman parte fundamental del contexto de este proyecto. Entre estos términos se encuentra la *World Wide Web (WWW)*, los protocolos de comunicación HTTP y HTTPS, la computación en la nube, específicamente el modelo de servicio SaaS, los lenguajes de diseño web HTML y CSS, el formato JSON, las APIs y los *frameworks*. Todos estas tecnologías, protocolos y lenguajes cumplen un papel fundamental pues la interfaz que se desarrolló en este proyecto hace uso de éstos.

Como bases para este proyecto se utilizaron tecnologías que ofrecen las prestaciones necesarias para cumplir con los requisitos establecidos. Uno de los fundamentales es el poder ofrecer acceso remoto. Para esto, se utiliza el acceso vía web. La *World Wide Web (WWW)*, comúnmente conocida como la web, fue inventada por Sir Tim Berners Lee como una propuesta en 1989 mientras trabajaba en el CERN [1]. La web hace uso de un conjunto de tecnologías y protocolos (HTML, HTTP, entre otros) que permiten compartir información y recursos por medio del Internet haciendo uso de navegadores web. Principalmente ésta fue utilizada para ofrecer páginas web sencillas con información sobre algún tema específico. Con el paso del tiempo la web pasó a ofrecer servicios más complejos como aplicaciones web. Hoy en día las aplicaciones web dependen de intercambios de información entre servidores y clientes, lo cual permite, entre otras cosas, realizar acciones cotidianas como registrarse en un servicio, autenticarse, realizar transacciones bancarias, búsquedas y publicación de contenido por sus usuarios [2]. Las aplicaciones web, esencialmente, son aplicaciones que se ofrecen por medio de un navegador.

Al hacer uso de la web, se aprovechan algunas virtudes de la computación en la nube. La computación en la nube consiste en ofrecer aplicaciones, plataformas o infraestructuras que se encuentran distribuidas en Internet a cambio de una subscripción paga o gratuita [3]. Entre sus virtudes se encuentra en modelo SaaS (*Software as a Service*) o software como servicio. SaaS es un modelo de computación en la nube que permite a proveedores ofrecer su software por medio de la web o de APIs para que éstos sean utilizados por los consumidores sin necesidad de instalar software

adicional en sus equipos [4]. Entre sus características principales se encuentran:

- Los usuarios no tienen que administrar, actualizar ni instalar software.
- La data se encuentra segura en la nube, la falla de el equipo donde se acceda a ésta no ocasiona su perdida.
- Las aplicaciones son accesibles casi desde cualquier dispositivo conectado a Internet.

La gran mayoría de los sistemas o aplicaciones web tienen como fundamento base la arquitectura Cliente-Servidor la cual es una arquitectura de aplicación en la cual siempre hay un *host* disponible, llamado servidor, el cual provee respuestas a solicitudes de otros *hosts*, llamados clientes. Un ejemplo clásico son las aplicaciones web. En éstas siempre hay un servidor disponible para responder las solicitudes de los navegadores ejecutados en los *hosts* de los clientes [5].

Debido a que el proyecto es una aplicación web, se hace uso de varias tecnologías utilizadas para el desarrollo web. Entre éstas se encuentra HTML (*HyperText Markup Language*), el cual es el lenguaje estructural estándar utilizado en el diseño web. Éste se encarga de describir la estructura de una página web abarcando desde cómo se ve en un navegador web, hasta cómo se comporta cuando el usuario interactúa con la página o aplicación web [6]. También se utiliza CSS (*Cascade Style Sheet*), lenguaje utilizado para describir la presentación de las páginas o aplicaciones web, abarcando desde estilos de letra, diseño y colores [6].



Las aplicaciones web se apoyan en protocolos estándar que hacen posible su funcionamiento, entre éstos se encuentra HTTP (*HyperText Transfer Protocol*), un protocolo de capa de aplicación para sistemas de información distribuidos, colaborativos e hiper-media. Es un protocolo que puede ser usado para muchas tareas más allá del uso común [7]. Es el estándar utilizado en la web para la transferencia de recursos a través de ésta. Para mejorar la seguridad de las aplicaciones web el protocolo HTTP se combina con el protocolo TLS (*Transfer Layer Security*), éste provee seguridad en la comunicación a través de la web. El protocolo permite que aplicaciones cliente-servidor se comuniquen en una forma diseñada para prevenir espionaje, alteración o falsificación de mensajes [8]. Al combinar estos protocolos se implementa HTTPS (*HTTP Secure*),

conocido comúnmente como HTTP seguro, un mecanismo que ofrece la seguridad no ofrecida por HTTP. Conceptualmente HTTPS es HTTP/TLS, es decir, utilizar HTTP con el protocolo TLS [9].

El uso de HTTPS permite aumentar la seguridad de un sistema web. La seguridad web forma parte de la seguridad informática. La seguridad informática se define como un conjunto de métodos y herramientas destinados a proteger la información y por ende los sistemas informáticos ante cualquier amenaza [10].

Entre los objetivos de la seguridad informática se encuentran:

- Confidencialidad: Los datos solo deben ser conocidos y accedidos por quienes estén debidamente autorizados durante su almacenamiento, procesamiento o transmisión [11].

- Integridad: Los datos solo pueden ser modificados y/o eliminados por quienes estén autorizados para ello y los sistemas y aplicaciones solo deben ser operados por personal autorizado. Esto incluye Autenticidad, No Repudio, Contabilidad [11].



- Autenticidad de la Información: Es la cualidad o estado de ser genuino u original, no una reproducción o fabricación. La información es considerada autentica cuando se encuentra en el mismo estado en el cual estaba al ser creada, almacenada o transferida [12].

- No Repudio: Termino asociado con la aceptación de comunicación entre emisor y receptor (cliente y servidor) normalmente a través del intercambio de certificados digitales de autenticación. El no repudio asegura que se cumplan todas las operaciones por ambas partes en una comunicacion [11].

- Autenticación o Control de Acceso: Proceso de validación de la supuesta identidad de un solicitante [12].

Este proyecto utiliza también el formato JSON (*JavaScript Object Notation*), un formato ligero de intercambio de datos, fácil de escribir y leer para humanos además de ser fácil de traducir y generar para computadores. Es un formato de texto independiente del lenguaje de programación que lo utilice [13].

Debido a la futura interacción de la interfaz web desarrollada con sistemas externos, es necesario contar con una API (*Application Programming Interface*) que se encargue de coordinar la comunicación de los sistemas ya mencionados con la interfaz web. Esencialmente una API es un contrato que especifica cómo se puede interactuar con otras aplicaciones [14]. Los sistemas externos que tendrán interacción con la interfaz web por medio de la API serán los *frameworks*. Un *framework* es un sistema que ofrece servicios y herramientas para realizar operaciones específicas, en la mayoría de los casos, operaciones comunes y repetitivas. Éste tiene una estructura específica y requiere del cumplimiento de una serie de protocolos establecidos para garantizar su correcto funcionamiento.

## 1.2 Antecedentes

En la actualidad existen muchas plataformas que ofrecen cursos a través del Internet, los cuales son presentados en distintas plataformas como EdX [15], Coursera [16], FutureLearn [17]. Estas plataformas prestan sus servicios a muchas universidades y les permiten ofrecer sus cursos a cualquier persona con una conexión a Internet. Dichas plataformas cuentan con herramientas básicas para crear evaluaciones: selección simple, selección múltiple, respuestas cortas, etc. pero la mayoría no cuenta con herramientas para hacer evaluaciones específicas, por ejemplo, para la evaluación de códigos de programación. Para evaluaciones de éste tipo las universidades deben tener su propia plataforma la cual se encargue de evaluar los códigos enviados por los estudiantes. Dichas evaluaciones deben incluir: compilación del código, ejecución, casos de pruebas, revisión del resultado de la ejecución, etc.



También se pueden encontrar plataformas como HackerRank [18] o Codeassess [19] que realizan evaluaciones o retos de programación con distintos propósitos, que van desde prácticas para el programador que quiera mantener sus habilidades al día, hasta evaluaciones para la selección de talento humano que consisten en distintos retos de programación realizados en estas plataformas. La plataforma Codeassess por ejemplo, ofrece sus servicios de pruebas por un cierto precio. Por otro lado la plataforma HackerRank permite a profesores evaluar tareas de programación a sus alumnos por

medio de su plataforma, a través de Internet, por supuesto, luego de una solicitud previamente realizada.

La mayoría de estas plataformas son implementadas utilizando conexiones seguras las cuales permiten la protección de hardware y software de ataques que amenacen la integridad, autenticidad, confidencialidad y estabilidad de dichas plataformas, de esta manera asegurando el servicio para los estudiantes que necesiten utilizarlo.

## 1.3 Planteamiento del Problema

El avance continuo de la tecnología ha ocasionado cambios drásticos en las formas de aprendizaje de los seres humanos. Muchas tecnologías han sido implementadas para facilitar el acceso a recursos de aprendizaje en otrora inaccesibles.

En la carrera de Ingeniería de Sistemas de la Universidad de Los Andes, específicamente en el área de Sistemas Computacionales se desea utilizar ampliamente el uso de los nuevos recursos tecnológicos para mejorar los procesos de aprendizaje. Un caso puntual es pretender mejorar la ejecución de prácticas y proyectos que involucren el desarrollo de programas computacionales en diferentes asignaturas, sin embargo aún no se ha logrado implementar una plataforma que facilite la evaluación de prácticas de laboratorios, proyectos u otras asignaciones similares, en los cuales se requiere compilar, interpretar, ejecutar y evaluar código de forma remota. Esta plataforma, estructurada a nivel de hardware y software, debe permitir llevar a cabo estas tareas de manera remota con el fin de mejorar su eficiencia y accesibilidad, para lo que se necesita el establecimiento de conexiones seguras que permitan la protección de la integridad, autenticidad, no repudio y confidencialidad de los datos (del código). Para esto debe contar con una interfaz que permita al usuario conectarse de manera remota y segura al centro de datos (servidores), realizar el envío seguro del código a evaluar y también recibir los resultados correspondientes a la evaluación.

## 1.4 Justificación

Aunque existen plataformas disponibles para realizar evaluaciones de prácticas de programación a distancia, éstas tienen un costo elevado, o no soportan todos los lenguajes necesitados para un evaluador. Además, desarrollar una plataforma propia para este propósito facilitaría la adaptación de esta a nuevas necesidades que se presenten y también se podrían realizar pruebas que le permitirían al profesor estar presente durante la evaluación y no sería necesario el acceso a Internet, pues al estar en la misma red local de la plataforma, el acceso a ésta sería más eficiente.

## 1.5 Alcance

Este proyecto finalizó con la creación de una interfaz web con conexión segura que permite a usuarios registrarse en ella, acceder utilizando métodos de autenticación y también interactuar con *frameworks* a los que se encuentren relacionados implementando una API que se encargue de especificar las reglas que se deben cumplir para ejecutar dicha interacción. Esta API se ofrece como un traductor que recibe una representación gráfica en formato JSON y convierte esta representación en código HTML que un navegador web puede interpretar y mostrar al usuario, así la API permite al *framework* mostrar al usuario lo que desee siempre y cuando cumpla con las reglas establecidas por ésta.



## 1.6 Objetivos

### 1.6.1 Objetivo General

Integrar herramientas y protocolos web que permitan implementar una interfaz con conexión segura a una plataforma de evaluación remota de códigos de programación.

### 1.6.2 Objetivos Específicos

- Buscar y evaluar herramientas y protocolos web vinculados con la evaluación de código remota y segura.

- Seleccionar las herramientas y protocolos web para su integración en una interfaz que permita el acceso a una plataforma de evaluación.
- Desarrollar una aplicación web que integre las herramientas y protocolos seleccionados.
- Aplicar pruebas de rendimiento, eficacia y seguridad de la plataforma.

# Capítulo 2

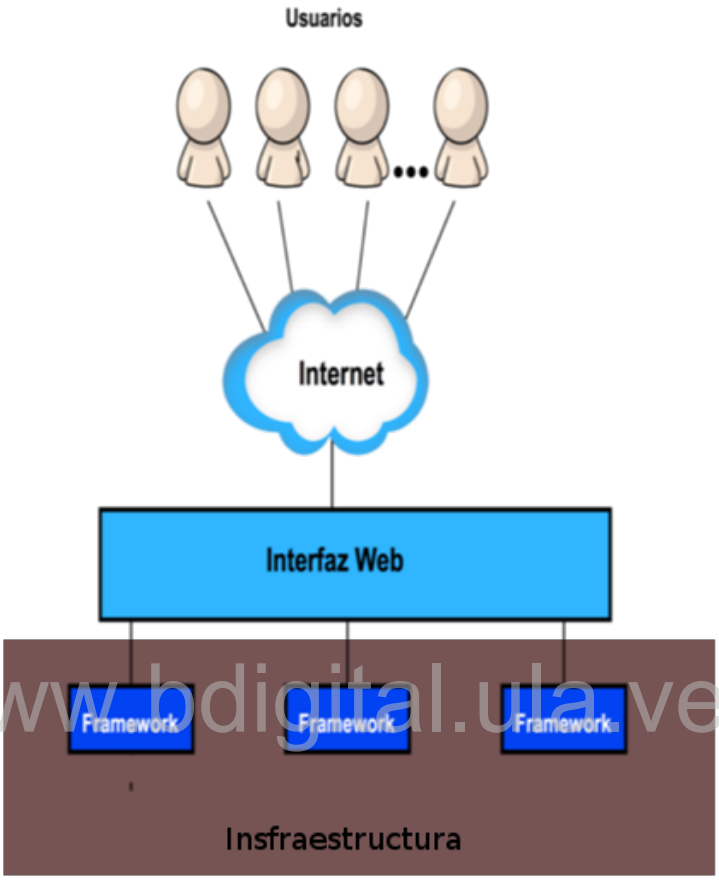
# Diseño de los componentes que conforman la interfaz web.

Antes de explicar la solución planteada al problema, es imperativo explicar el contexto en el que se encuentra esta solución. La interfaz web que se desarrolla en este proyecto no es un sistema aislado, al contrario, es un sistema que se encarga de intermediar la comunicación entre otros entes. Específicamente entre usuarios y *frameworks*. Los usuarios utilizan la interfaz para solicitar contenido a los *frameworks*, y éstos envían dicho contenido a través de la interfaz a los usuarios.



La siguiente figura ilustra este contexto.

Figura 2.1: Contexto en que se encuentra la interfaz web.






En este contexto los entes de interés para la interfaz web son los usuarios y los *frameworks*. Un usuario en este contexto es una persona que desee utilizar la interfaz web para interactuar con algún *framework*. Cabe destacar que en este contexto, los *frameworks* estarán relacionados con asignaturas a cursar y que éstos ofrecen servicios y herramientas relacionados con estas asignaturas a los usuarios.

Se muestran varios usuarios y varios *frameworks* pero una sola interfaz, esto quiere decir que la interfaz debe permitir la interacción de distintos tipos de usuarios con distintos tipos *frameworks*. Por ejemplo, la interfaz debe permitir a un profesor

interactuar con un *framework* de matemática o a un estudiante interactuar con un *framework* de compiladores de forma transparente y sin limitar a ninguno de los dos,dado que su trabajo es simplemente intermediar en la comunicación entre usuarios y *frameworks*.

Indudablemente este enfoque exige que la interfaz ofrezca prestaciones más avanzadas que las ofrecidas por una interfaz común, ya que requiere una generalización que le permita interpretar lo que envían distintos tipos de *frameworks* independientemente de lo que ofrezcan a los usuarios. Para lograr la generalización deseada es necesario crear una API que establezca las reglas de interacción entre *framework* e interfaz. Esta API será explicada más adelante en este capítulo.

La interfaz debe ofrecer acceso remoto, el cual se puede lograr vía web. La interfaz además se encarga de ejercer un control de acceso que exige a los usuarios una autenticación para poder acceder al sistema. Debido a que varios tipos de usuarios interactuarán con el sistema, es necesario crear la distinción entre éstos, pues no todos los usuarios tendrán los mismos privilegios. Para cumplir con los requerimientos necesarios existirán tres tipos de usuarios en la interfaz:



- Administrador de Sistema: Este tipo usuario se encarga de la gestión de la interfaz. Es éste el que debe autorizar a usuarios y *frameworks* para que puedan utilizar la interfaz.

- Administrador de *Frameworks*: Este tipo de usuario se encarga de agregar *frameworks* a la interfaz para que puedan ser utilizados por medio ésta. Estos *frameworks* deben ser aprobados por el Administrador de Sistema.

- Usuario Común: Cualquier otro tipo de usuario que use el sistema, solo puede interactuar con los *frameworks* a los que el esté relacionado. Esta relación la crean los *frameworks* por medio de la API, mando de los Administradores de *Frameworks*.

La interfaz también se ofrece utilizando una conexión segura, es decir, utilizando HTTPS como protocolo de comunicación. El objetivo buscado con este protocolo es garantizar que si un atacante intercepta la información intercambiada por la interfaz y el usuario éste no pueda interpretarla. Este protocolo es muy utilizado por sistemas

que intercambian información sensible (bancos, correo electrónico, entre otros) y cada día se acerca más a convertirse en un estándar utilizado en la web.

Este enfoque garantiza que la interfaz ofrezca los requisitos necesarios para lograr una interacción remota y segura con los usuarios.

En cuanto a *frameworks* la interfaz debe permitir que se pueda acceder a éstos, para esto es necesario que se conozca la dirección única o URL (*Uniform Resource Locator*) de cada *framework* con el que puede interactuar. También es necesario que ofrezca una identificación única al *framework* sobre el usuario que solicita la interacción con éste, debido a que, como ya se estableció, la interfaz no se encarga de limitar a los usuarios al momento de interactuar con los *frameworks*, eso le corresponde a los *frameworks*, por lo que un *framework* debe saber la identidad del usuario que solicita una interacción con él. Esta identificación se logra con un token de autenticación, asignado por cada *framework* a cada usuario con que se quiera relacionar. El Administrador de *Frameworks* asigna su propio token de autenticacion al momento de registrar el *framework* en la interfaz. Más adelante se explica como se logra la asignación por medio de la API. Con esta información (URL (*Uniform Resource Locator*) y token de autenticación) la interfaz puede establecer una interacción con cualquier *framework*.



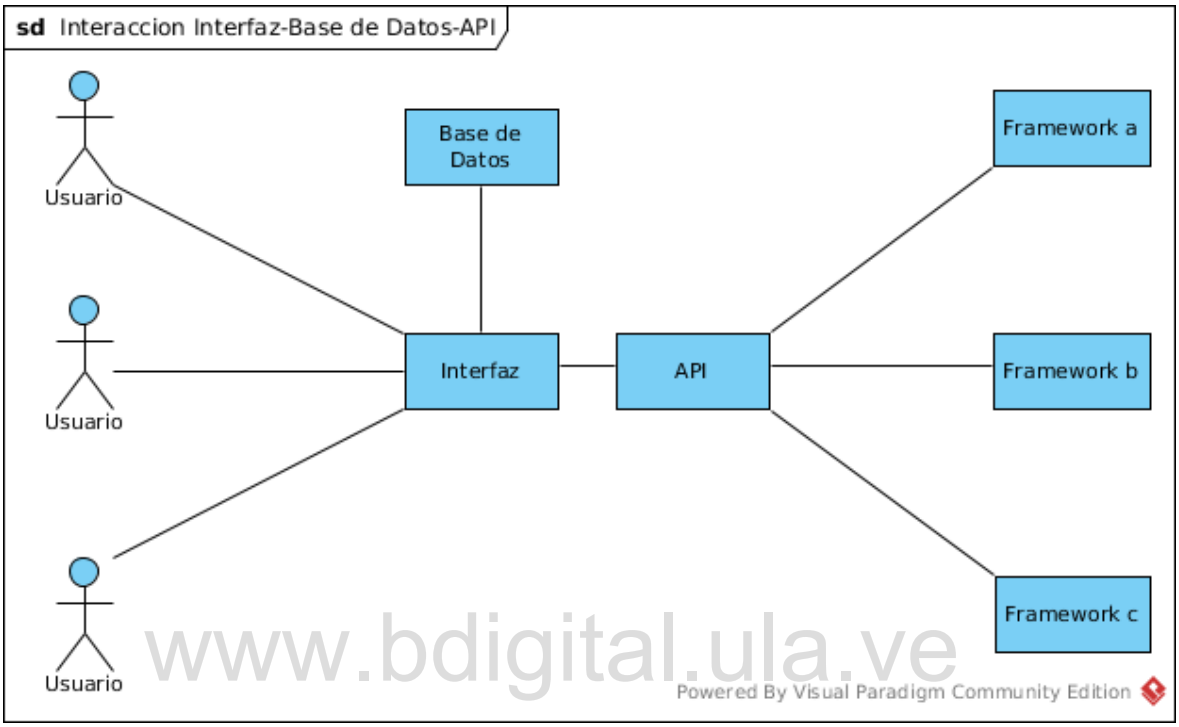
Por ultimo, el requerimiento más importante y complejo de la interfaz para con los *frameworks* es permitirle a éstos que ofrezcan acciones de a los usuarios de cualquier índole. Es para ésto que se requiere una API que especifique el protocolo que deben cumplir al enviar estas acciones a la interfaz para que ésta sea capaz de interpretarlas. El protocolo que se debe cumplir envuelve las reglas, la estructura y el formato que debe tener una acción al ser enviada a la interfaz. Cumplir con este protocolo permite a los *frameworks* explotar las prestaciones de la interfaz.

Una vez establecidos los requerimientos, se definen los componentes que se necesitan diseñar para cumplir con éstos. La mayoría de los componentes que se necesitan para realizar este proyecto son implementados con herramientas existentes y su implementación se muestra en el capitulo 4 de este documento.

Los componentes que se definen a continuación son la base de datos del sistema y la API. Con la inclusión de estos componentes en el contexto de la interfaz, se puede mostrar una ilustración de la interacción entre la interfaz, la base de datos y la API.

La siguiente figura muestra esta representación.

Figura 2.2: Interaccion Interfaz-Base de Datos-API

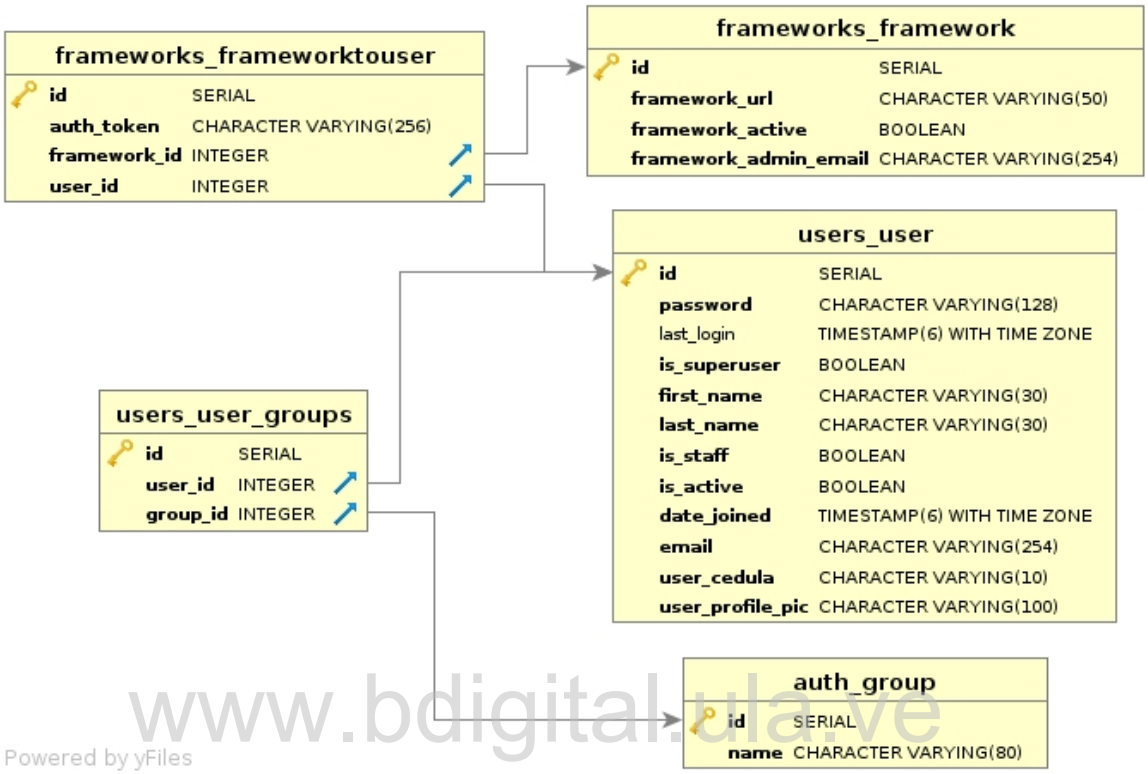





En esta figura se aprecia el papel que cumple tanto la base de datos como el API. La base de datos almacena la información necesaria para que la interfaz pueda funcionar y el API se encarga de intermediar la comunicación entre usuarios y *frameworks*.

Uno de los componentes fundamentales para desarrollar este proyecto es la base de datos. En la siguiente figura se muestra el modelo diseñado para la misma.

Figura 2.3: Diagrama ERE de la Base de Datos






Los nombres de las tablas han sido generados por la herramienta con la que se implementó la base de datos. A continuación se explica el diseño de cada tabla

- users_user: Esta tabla contiene la información de los usuarios. Entre esta información se encuentran datos personales como nombre, apellido, cédula, foto de perfil, correo y también se encuentra información necesaria para su interacción con la interfaz como la contraseña, si está activo o no, etc.

- frameworks_framework: Esta tabla contiene la información necesaria para la interacción del *framework* con la interfaz. Entre esta información se encuentra la URL (*Uniform Resource Locator*), el correo del administrador para solo permitirle a éste administrarlo y también una variable que indica si el *framework* está activo o no.

- auth_group: Esta tabla contiene los tipos de los usuarios en el sistema. Es aquí donde se especifican los tres tipos de usuarios mencionados previamente.

- users_user_groups: Esta tabla contiene las relaciones entre usuarios y grupos, es decir, es esta tabla la que indica de qué tipo es cada usuario. Esta relación es muchos a muchos, lo que permite que un usuario pueda ejercer distintos tipos de roles en la interfaz.

- frameworks_framerworktouser: Esta tabla contiene las relaciones entre usuarios y *frameworks*, es en esta tabla donde se indica si un usuario puede interactuar con un *framework* o no. Esto evita que a un usuario se le ofrezcan todos los *frameworks* existentes. Esta relación es muchos a muchos, lo que permite que muchos usuarios estén relacionados con muchos *frameworks*

En resumen, la interfaz utiliza toda la información almacenada en estas tablas para garantizar al usuario un acceso seguro e interacción con los *frameworks* a los que éste esté relacionado.

Para definir la API es necesario definir los participantes que interactúan con ésta, las actividades que se pueden realizar y los recursos que maneja. En cuanto a participantes, como ya se mostró en la figura 2, la interfaz y los *frameworks* son los hacen uso de la API. La interfaz se encarga de solicitar a la API una interacción con algún *framework* en específico y los *frameworks* envían representaciones gráficas a la interfaz por medio de la API para que ésta la muestre a el usuario.



Aunque el objetivo de la API es ofrecer un mecanismo genérico que permita a *frameworks* ofrecer acciones a los usuarios, es necesario que se definan un numero finito de actividades que determinen la funcionalidad de las interacciones entre *framework* y usuarios. Es decir, se debe definir un flujo de trabajo que defina cómo debe iniciar una interacción y que especifique cada paso intermedio hasta que esta interacción llegue a su final. Se han definido cuatro actividades posibles que permite ejecutar el API:

- Obtener información de *frameworks*.

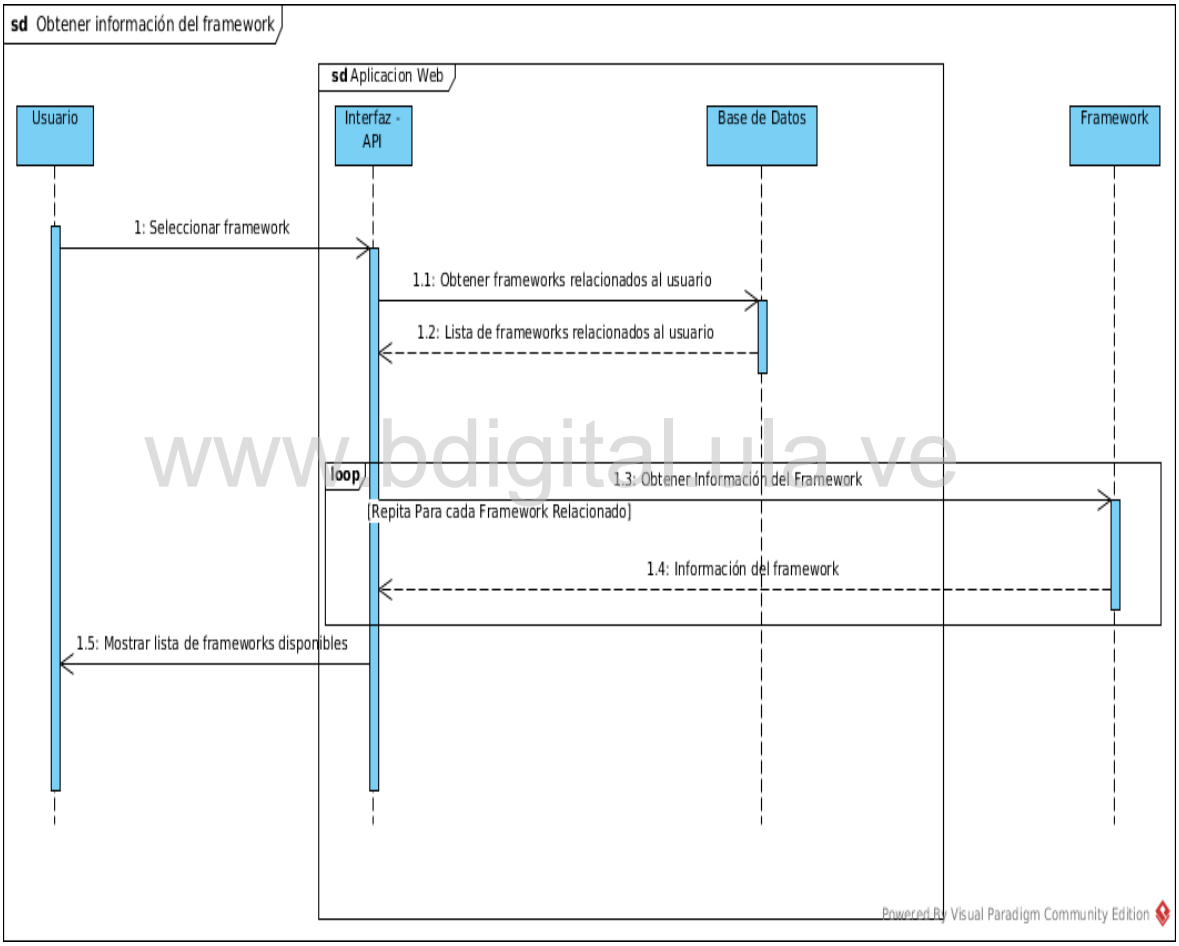
- Seleccionar un *framework* para ver las acciones que éste ofrece al usuario que solicita la interacción.

- Seleccionar una acción, realizarla, enviarla al *framework* y obtener un resultado final.

- Realizar una sincronización de usuarios que genere las relaciones entre los usuarios del sistema y el *framework*.

A continuación se muestra un diagrama de secuencia para cada actividad y se explica cada uno de estos.

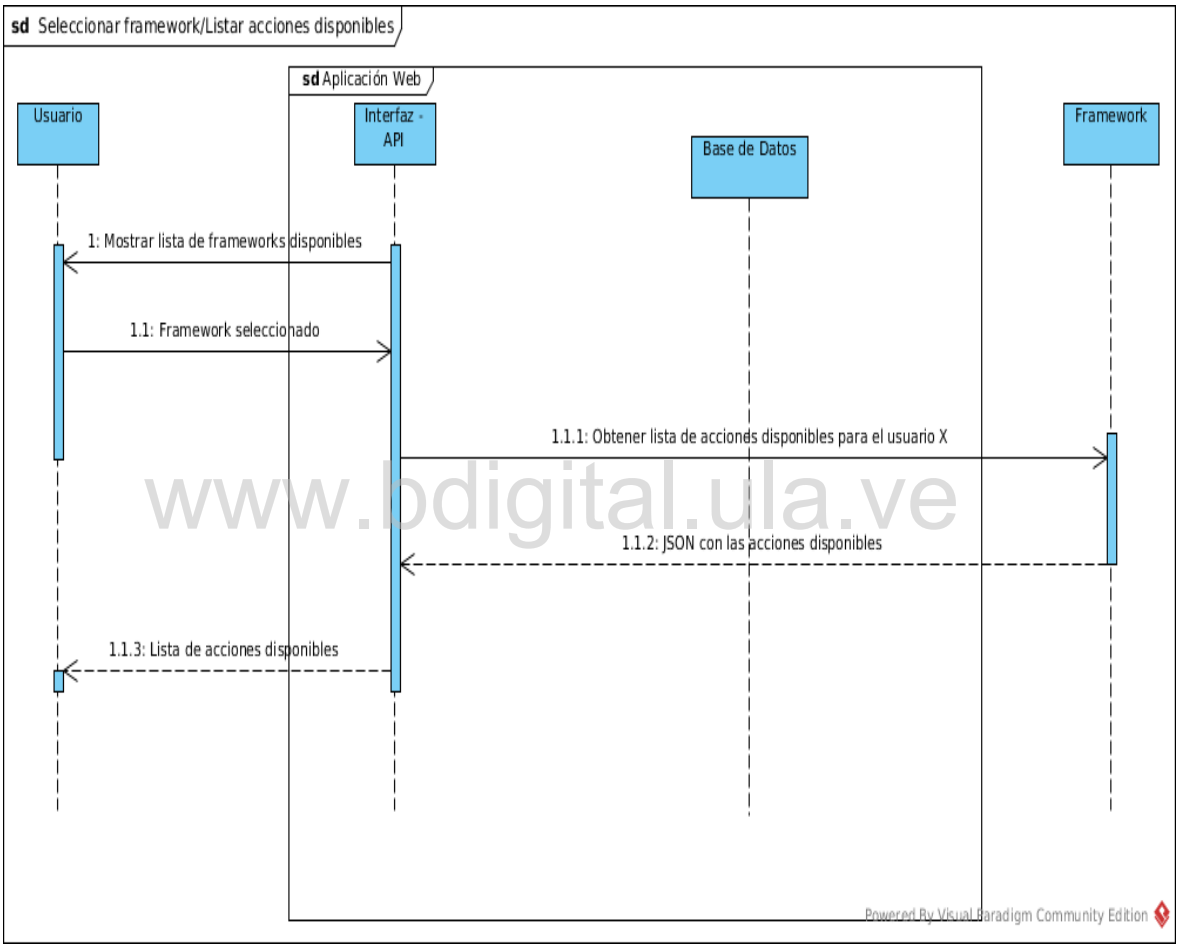
Figura 2.4: Diagrama de Secuencia - Obtener informacion de los *frameworks*




Este diagrama representa cómo se le muestran los *frameworks* disponibles al usuario. Un usuario puede estar relacionado con varios *frameworks*, por lo que la API debe solicitar información a cada uno de estos *frameworks*, esto se representa con el *loop* que se aprecia en el diagrama. En resumen, un usuario solicita la lista de *frameworks* disponibles, la interfaz revisa en la base de datos a qué *frameworks* el usuario está relacionado y luego el API pasa a solicitarle a cada uno de estos *frameworks* la

información que se le debe mostrar al usuario que solicitó la interacción. El resultado de esto es una lista de *frameworks* de los que el usuario puede seleccionar para pasar a la siguiente actividad, explicada en el siguiente diagrama.

Figura 2.5: Diagrama de Secuencia - Seleccionar *framework*/Listar actividades disponibles *frameworks*



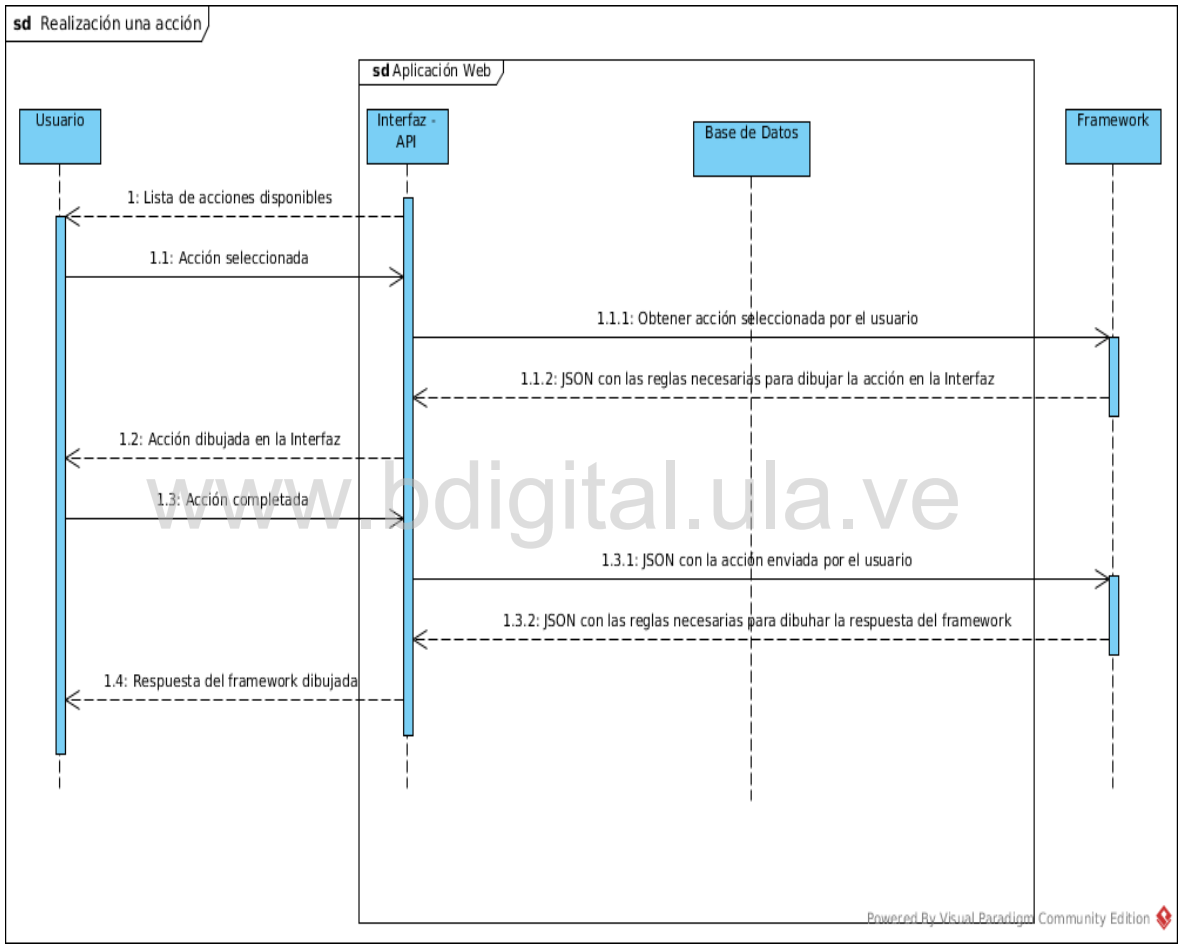



En este diagrama se representa cómo se le ofrece la lista de acciones disponibles al usuario. Una vez seleccionado un *framework* de la lista (figura anterior) la API solicita a dicho *framework* una lista de acciones para este usuario. El resultado de este paso es una lista de acciones disponibles para que el usuario pueda escoger una y pasar a ejecutarla. En resumen, una vez que el usuario escoja un *framework*, la API solicita a éste una lista de acciones y luego muestra esta lista al usuario. Como se mencionó previamente, el usuario cuenta con una identificación única que el *framework* debe

utilizar para decidir qué acciones ofrecer al usuario. Ni la interfaz ni la API deciden que acciones puede o no puede realizar un usuario en un *framework*.

Una vez se ofrezca una lista de acciones al usuario, se pasa a ejecutar la siguiente actividad, explicada en el siguiente diagrama.

Figura 2.6: Diagrama de Secuencia - Realización una acción






En esta actividad al usuario se le presenta una lista de acciones, resultado de la actividad anterior. De esta lista de acciones el usuario selecciona una lo cual ocasiona la secuencia de operaciones representada en el diagrama. Existen dos tipos de acciones que el *framework* puede enviar al usuario, estas son, acciones que requieren un envío de información y acciones que no requieren de un envío de información, por ejemplo, una prueba de programación, requiere que el usuario envié las respuestas al *framework* para que éste las evalúe y envíe un resultado. Una acción que no requiera envío de

información por parte del usuario puede consistir en una lista de enlaces web que le permitan al usuario observar material didáctico de un tema específico. En el siguiente párrafo se describe detalladamente la secuencia de operaciones que ocurren cuando el usuario selecciona una acción que requiera de envío de información al *framework*.

Una vez la acción haya sido seleccionada la API solicita una representación gráfica de la acción al *framework*, una vez el *framework* envíe dicha representación, ésta se muestra en la interfaz al usuario, y permite al usuario realizar la acción, por ejemplo, si es una prueba de programación formada por preguntas prácticas, la interfaz permite al usuario responder dichas preguntas y enviarlas al *framework*. Una vez realizada la acción se le envía al *framework* y la API pasa a procesar la respuesta, la cual también debe ser también una representación gráfica.

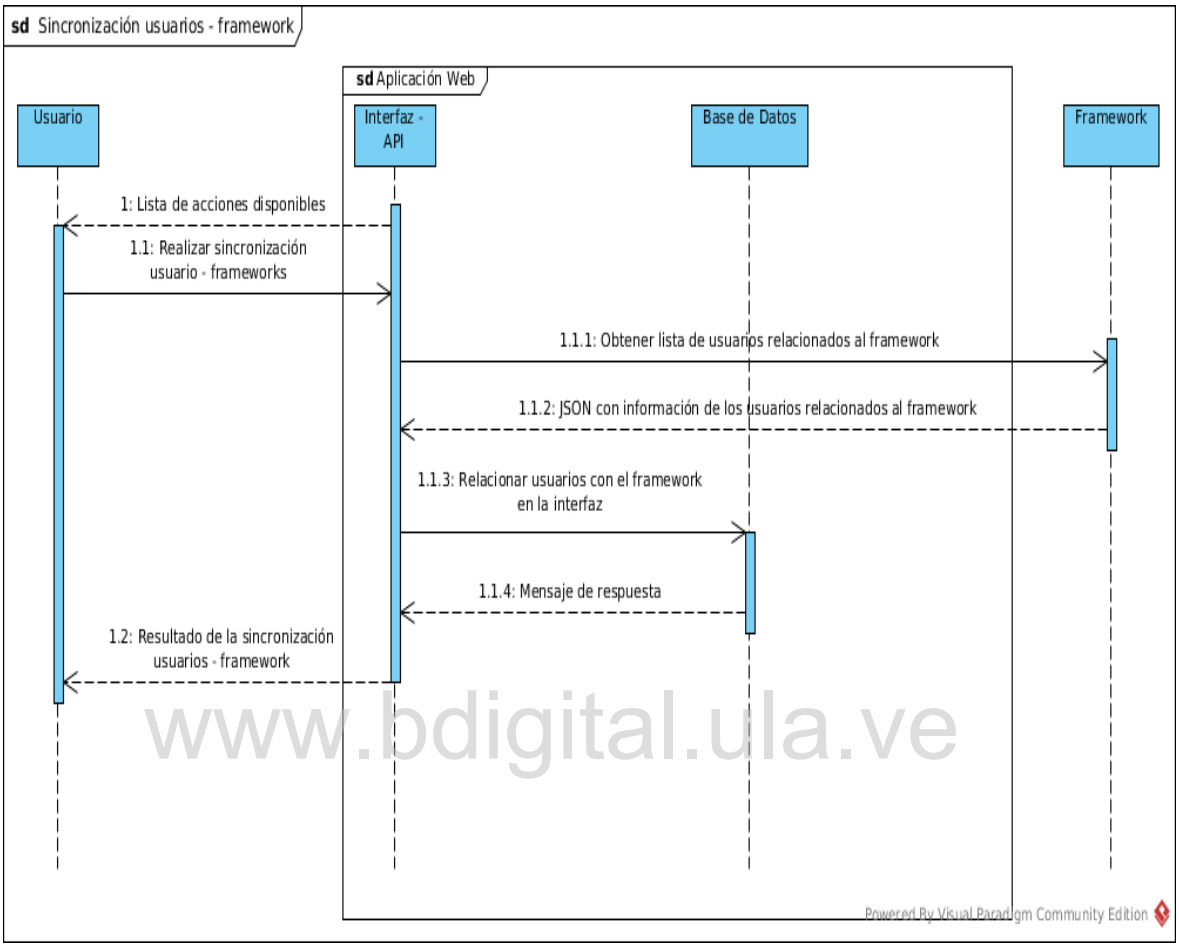
La ultima actividad disponible de la API es una acción especial que es necesaria para el funcionamiento requerido por la interfaz. Los usuarios tendrán acceso a los *frameworks* a los que se encuentran relacionados. Esta relación es generada por los *frameworks*, esto implica, que son los *frameworks* los que tienen que enviar la información necesaria para crear la relación. Esta información consiste de un identificador único del usuario en la interfaz, correo electrónico en este caso, y un token de autenticación que identifique inequívocamente al usuario en el *framework*. Si el usuario obtiene un token de autenticación erróneo, la interfaz no podrá establecer comunicación con el *framework* en representación de este usuario.



Cabe aclarar que aunque esta actividad es una acción especial, también se selecciona de una lista de acciones ofrecida por el *framework*, al igual que las acciones representadas en el diagrama anterior. Establecer qué usuario puede ejecutar esta acción queda de parte del *frameworks*. Esta acción debe estar disponible para un Administrador de *Framework*, pero la decisión final de quien la ejecuta le corresponde al *framework* en cuestión.

El siguiente diagrama representa esta última actividad.

Figura 2.7: Diagrama de Secuencia - Sincronización de Usuarios






En resumen, esta acción se ejecuta de la siguiente manera. La API solicita la sincronización de usuarios a los que el *framework* quiere relacionar consigo mismo. El *framework* responde con una lista de usuarios a los cuales se le genera la relación necesaria con éste y luego se muestra una respuesta al usuario que solicitó la sincronización en donde se especifiquen qué usuarios fueron relacionados exitosamente y cuáles no lo fueron. Por ejemplo si el *framework* trata de relacionar un usuario que no exista en el sistema, esto se le notifica al usuario que solicitó la sincronización.

Ya establecidas las actividades disponibles, se definen los recursos que maneja la API.

Los recursos que maneja son solicitudes y respuestas en formato JSON. Específicamente, las solicitudes o datos que envíe el usuario al *framework* y también las

respuestas del *framework* a dichos envíos realizados por el usuario. El formato JSON provee la flexibilidad necesaria para las representaciones gráficas que se desean enviar pero al mismo tiempo ofrece una estructura fácil de entender y útil para expresar estas representaciones.

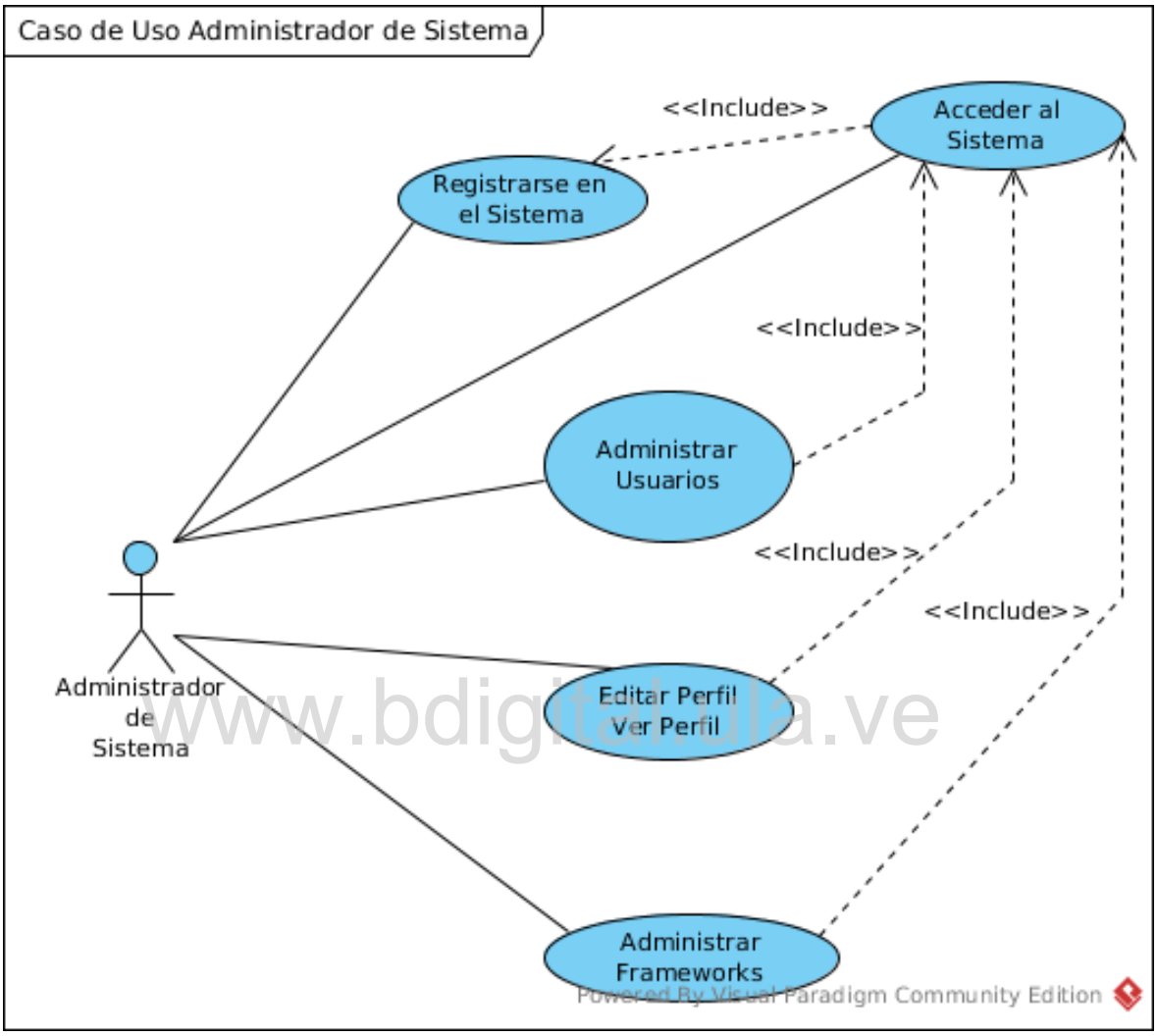
Se definen algunas reglas que regulan el funcionamiento de la interfaz y luego se especifican las posibilidades que la interfaz le ofrece a cada tipo de usuario que se registre en ella.

Inicialmente, todos los usuarios deben registrarse en la interfaz y ser activados por un Administrador de Sistema antes de poder acceder a ésta. Esta activación sucede después de que el Administrador corrobore los datos del usuario registrado con los datos de la Universidad de Los Andes y compruebe que dicho usuario sí puede hacer uso de la interfaz. Los Administradores de *Frameworks* deben registrar el *framework* que administrarán al momento de registrarse, estos además deben enviar la información del *framework* al Administrador de Sistema para que éste corrobore que el *framework* sí cumple con los requisitos para ser utilizado por medio de la interfaz. Tanto *frameworks* como usuarios pueden ser desactivados en cualquier momento si un Administrador de Sistema lo considera como una medida justificable para preservar la seguridad de la interfaz. Los Usuarios pueden tener varios roles, por ejemplo, pueden ser Administradores de *Framework* y Usuario Común en la interfaz. Esto es necesario ya que si un usuario administra un *framework* pero también está relacionado con otro que no administra, el sistema debe poder realizar la distinción para ofrecerle las capacidades requeridas con cada uno de estos *frameworks*. Por ejemplo dicho usuario podría editar información del *framework* que administra pero no debe poder hacerlo con los otros a los que esté relacionado pero no administre.



Las capacidades de cada tipo de usuario en la interfaz se representan por medio de diagramas de casos de uso. Todos los tipos de usuarios se deben registrar en el sistema, pueden acceder a éste (después de ser activados), pueden editar y ver su perfil de usuario en el sistema.

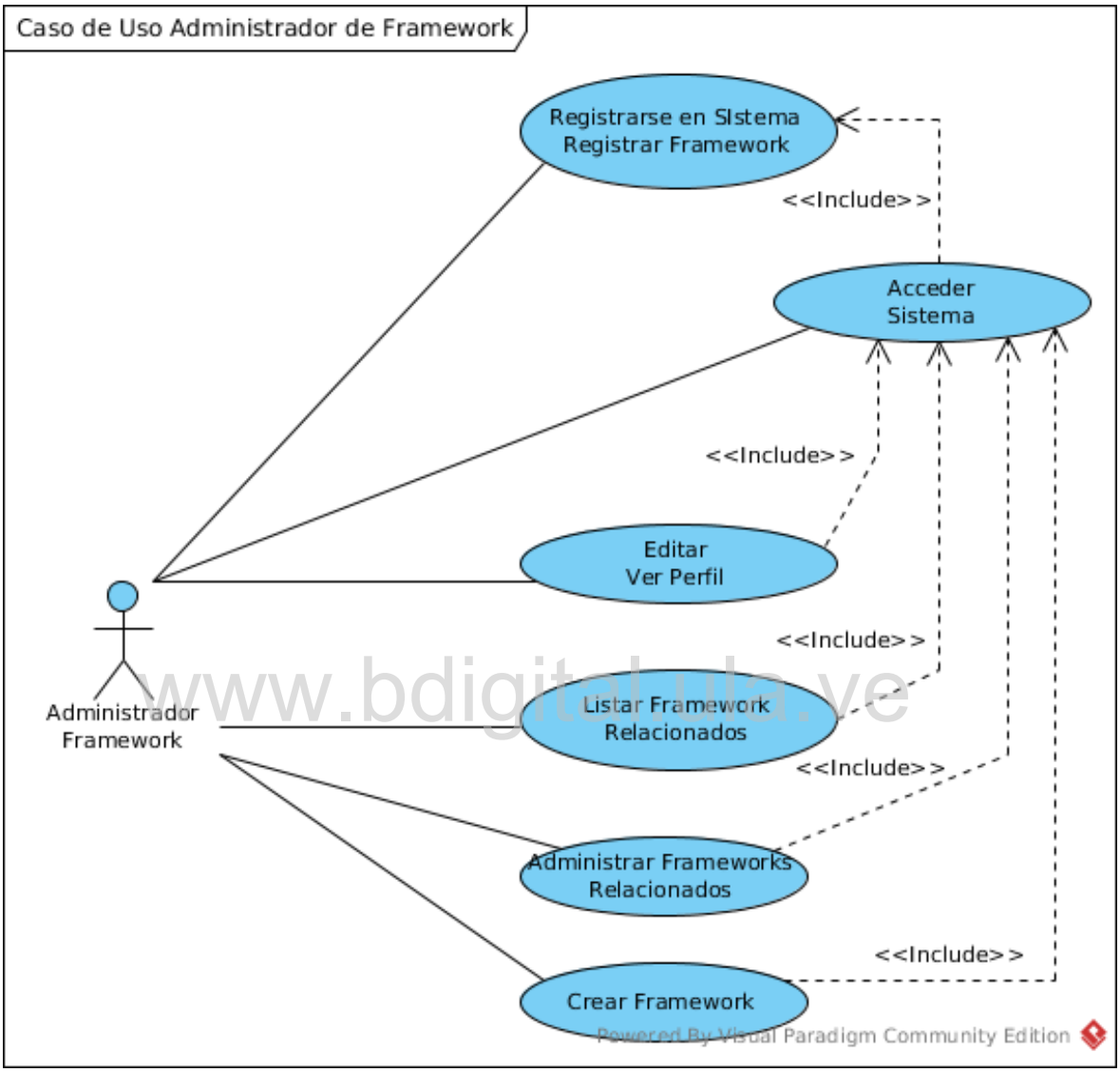
Figura 2.8: Diagrama de Caso de Uso - Administrador de Sistema





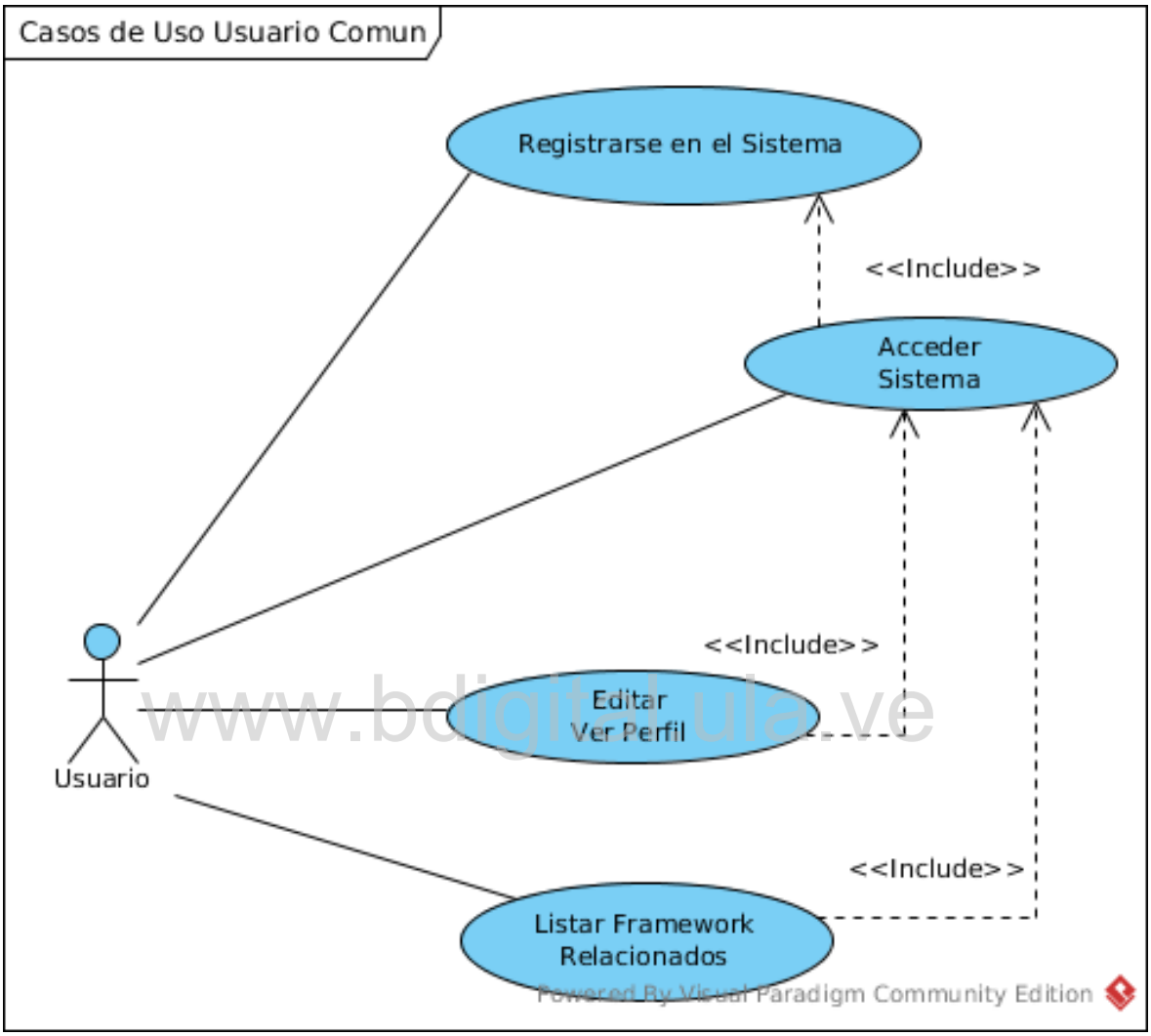

Las acciones especiales de un Administrador de Sistema se relacionan con administrar usuarios y *frameworks* en la interfaz. Esto permite activar o desactivar usuarios, activar o desactivar *frameworks*, editar usuarios, etc.

Figura 2.9: Diagrama de Caso de Uso - Administrador de Frameworks




Para el Administrador de *Frameworks* se permiten acciones especiales: administrar *frameworks* relacionados y crear *frameworks* nuevos. También se le permite listar los *frameworks* a los que se encuentra relacionado.

Figura 2.10: Diagrama de Caso de Uso - Usuario Comun






Al Usuario Común, además de las acciones básicas, solo se le permite listar los *frameworks* relacionados.

Al decir que se permite listar los *frameworks* relacionados, esto constituye todas las actividades posibles que fueron especificadas en el diseño de la API. Es por medio de esta acción que se inicia la interacción con los *frameworks*, interacción que sigue los pasos explicados en los diagramas de secuencias anteriores.

En este capítulo se estableció la solución planteada que cumple con todos los requerimientos exigidos por la interfaz que se desarrolló en este proyecto. En los capítulos siguientes se hace una breve descripción de las herramientas que se utilizaron

para implementar esta solución y cómo se utilizaron estas herramientas para lograr la implementación mencionada.

# Capítulo 3

# Herramientas Seleccionadas

En este capítulo se presenta una breve descripción de las herramientas que se utilizaron para implementar la solución planteada en el capítulo anterior además de una breve explicación de porqué se decidió utilizarlas.

Es fundamental que se mencione qué componentes conforman una interfaz web como la que se desarrolló en este proyecto. Una interfaz web en general contiene dos partes fundamentales, el front-end, termino asignado a la parte que visualiza el usuario, es decir, lo que le muestra el navegador, y el back-end, termino asignado a la parte que se encarga del funcionamiento interno de la interfaz, por ejemplo, hacer las operaciones de la base de datos que el usuario solicite desde el front-end, decidir qué mostrar en el front-end, etc. Para que dicho back-end pueda manejar una base de datos es necesario que se cuente con un sistema gestor de base de datos, el cual se encargue de realizar las operaciones solicitadas por el back-end. Con estos tres componentes ya se tiene, en teoría, una interfaz web. También es necesario contar con un software que se encargue de servir la interfaz a través de la web, para esto se utiliza un servidor web. Con este software se puede gestionar la interfaz web y ofrecerla a los usuarios que la soliciten. Y por ultimo, debido a los requerimientos de este proyecto, es necesario contar con un software que genere los certificados TLS que se utilizan para verificar la conexión y de ejercer el HTTPS (HTTP seguro).

A continuación se describen que herramientas se utilizaron para implementar cada una de las partes mencionadas previamente.

## 3.1 Servidor web: Apache

Para implementar el Servidor Web se utilizó la herramienta Apache HTTP Server, el cual es un esfuerzo colaborativo que tiene como objetivo desarrollar un software robusto, de nivel comercial y de libre uso [20] ofrecido bajo las directrices de la licencia Apache 2.0 [21]. El proyecto es mantenido por un grupo de voluntarios distribuidos por el mundo entero los cuales utilizan la web para comunicarse, planear y desarrollar el servidor y su documentación correspondiente.

Se decide utilizar el servidor web Apache debido a que, por ser un proyecto mantenido por un gran número de voluntarios, ofrece una documentación abundante lo cual permite realizar el proceso de instalación y configuración siguiendo las instrucciones recomendadas por sus mismos desarrolladores. Además Apache permite implementar HTTPS (HTTP seguro), el cual es uno de los requerimientos fundamentales de este proyecto.

La integración del servidor web Apache con las otras herramientas implementadas ha sido comprobada y está ampliamente documentada [22].



## 3.2 Certificados TLS: OpenSSL

OpenSSL es un conjunto de herramientas que ofrecen software que trabajan con los protocolos TLS(*Transport Layer Security*) y SSL(*Secure Sockets Layer*). También ofrece una librería criptográfica de propósito general la cual permite generar certificados que puedan ser utilizados para verificar una conexión HTTPS [23].

OpenSSL es ofrecida bajo una licencia estilo Apache [21] la cual básicamente permite utilizarla para fines comerciales y no comerciales, depende de cuál sea el caso.

## 3.3 Sistema gestor de base de datos (SGBD): PostgreSQL

Como SGBD se utilizó PostgreSQL. PostgreSQL es un sistema de base de datos objeto-relación de código abierto ofrecido bajo la licencia PostgreSQL [24] que ofrece la

libertad de usar, modificar y redistribuir el gestor como se desee. Entre los beneficios de este gestor se encuentran un amplio rango de tipos de datos aceptados para el almacenamiento, como datos numéricos, alfanuméricos, booleanos, etc. Cumple con el principio ACID (Atomicidad, Consistencia, Aislamiento y Durabilidad) y que se apega al estándar ANSI-SQL:2008 [25]. También es multiplataforma lo que lo hace adaptable a varias plataformas de desarrollo. Además por su naturaleza de código abierto y su permisiva licencia, cuenta con una documentación amplia y completa la cual ayuda con el proceso de instalación y su configuración.

La compatibilidad de PostgreSQL con las otras herramientas utilizadas en este proyecto ha sido comprobada y documentada lo cual lo convierte en una opción valida para este proyecto [26].

## 3.4 Back-end y Front-end: Django

Django es un *framework* web que fue creado pensando en facilitar y agilizar la construcción de un sitio web haciendo el desarrollo de tareas repetitivas algo fácil de implementar con unas pocas líneas de código. La Django Software Foundation es la organización encargada del desarrollo y mantenimiento de este *framework* web [27], es un proyecto código-abierto bajo su propia licencia [28], cuenta con una gran comunidad de usuarios y colaboradores que contribuyen con su desarrollo y avance y el lenguaje de programación utilizado para desarrollar con este *framework* es Python.



Django trabaja con una variedad de la arquitectura MVC (*Model View Controller*), conocida como MVT (*Model View Template* o Modelo Vista Plantilla). Siguiendo esta arquitectura, una aplicación desarrollada en Django está formada por Modelos, Vistas y Plantillas, los cuales definen la estructura de datos, el funcionamiento del sitio web y la visualización de éste en el navegador respectivamente.

Entre los beneficios que trae Django se encuentran la automatización de tareas fundamentales y comunes que se encuentran en la mayoría de los sitios web, como autenticarse, mostrar listas de contenidos, mostrar detalles de contenidos, ofrecer formularios a los usuarios con validación de entrada incluida, etc. Django también ofrece muchas medidas de seguridad como HTTPS y protección contra ataques de

tipo CSRF(*Cross Site Request Forgery*), inyección SQL, XSS(*Cross Site Scripting*), también cifra automáticamente las contraseñas para que no sean almacenadas en la base de datos en texto plano, lo cual evita que se puedan conocer éstas si alguien vulnera la base de datos.

## 3.5 Front-end: React

Como se mencionó en capítulos anteriores, la naturaleza de la interfaz que se desarrolla en este proyecto hace necesario que ésta se pueda adaptar a distintas representaciones gráficas enviadas por los *frameworks* para luego traducir estas representaciones a código HTML que pueda ser interpretable por un navegador web. Normalmente los sistemas web pueden mostrar una pagina u otra dependiendo de lo que decida o seleccione el usuario, pero esto se hace comúnmente con recursos que se encuentran en el servidor, es decir, la estructura de la interfaz se encuentra en el servidor y la navegación es dirigida por el usuario. Sin embargo, en nuestro contexto, el servidor no sabe qué se va a mostrar debido a que esto es decidido por el usuario y especificado por los *frameworks*. Esto exige un alto nivel de adaptabilidad por parte de la interfaz, que permita traducir distintas representaciones gráficas enviadas en formato JSON a código HTML que cumpla con las reglas sintácticas y semánticas del lenguaje.



Para adaptarse a este dinamismo en las páginas web, es importante mencionar el lenguaje de programación JavaScript, lenguaje creado precisamente para ofrecer esta característica a los sitios web. JavaScript ofrece dinamismo en el navegador, es decir, el sitio web puede cambiar sin necesidad de hacer una petición al servidor.

En sus inicios JavaScript fue creado como un lenguaje de programación que funcionaba en el cliente, es decir, en el navegador web, todo su procesamiento y sus operaciones ocurrían en el navegador web, el servidor solo enviaba el código o la ruta de un archivo JavaScript para que le navegador lo pudiese descargar y utilizar. Hoy en día JavaScript ha evolucionado y es utilizado en clientes, servidores, aplicaciones móviles, etc.

Para este proyecto se utilizó una librería JavaScript llamada React, la cual actúa en el navegador. React (también conocido como ReactJS) es una librería JavaScript de

código abierto desarrollada por Facebook Inc y mantenida por una amplia comunidad de colaboradores [29]. React es ofrecida bajo la licencia MIT [30] la cual permite el libre uso, redistribución y modificación y ésta libre de costo.

React a diferencia de Django no es un *framework*, sino una librería para crear interfaces de usuarios compuestas. Ésta exhorta la creación de componentes de UI (User Interface o Interfaz de Usuario) reusables los cuales pueden presenta cambios en sus datos o estado a través en el tiempo [29].

Entre los beneficios que ofrece el uso de React para este proyecto se encuentran las siguientes:

- Una de las principales ventajas de React es la posibilidad de crear componentes reusables, lo que quiere decir que permite crear elementos compuestos por varios componentes HTML que pueden ser reutilizados y agregados a la aplicación como un solo componente.

- Permite el uso de una sintaxis llamada JSX creada por Facebook Inc la cual permite incluir código HTML dentro de código JavaScript de manera directa sin necesidad de una conversión previa. Esto permite crear componentes de manera rápida porque no requiere el uso de cadenas de texto concatenadas que hagan la conversión de JavaScript a HTML.

- Normalmente cuando una interfaz web cambia de estado, es decir, su representación HTML cambia, el árbol DOM (*Document Object Model*) es renderizado completamente, independientemente de si el cambio en la interfaz sea en un solo elemento, todo el árbol DOM se renderiza de nuevo. React, al contrario, implementa una funcionalidad conocida como DOM virtual, al implementar esta función React modifica el árbol DOM virtual una vez que reciba un cambio de estado, luego compara este árbol DOM virtual con el árbol DOM del navegador y si encuentra algún elemento diferente, solo renderiza este elemento en el árbol DOM del navegador. Esto agiliza el proceso de renderizado y genera una navegación más rapida.

- React también ofrece herramientas de depuración como extensiones de

navegadores como Chrome y Firefox. Esto permite al desarrollador observar qué sucede cuando un sitio hecho con React funciona.

# Capítulo 4

# Implementación de la interfaz web

En el capítulo anterior se realizó una breve descripción de las herramientas utilizadas para implementar la solución planteada en el diseño de la interfaz. En este capítulo se procede a explicar como se utilizaron estas herramientas para implementar la solución planteada.



Para implementar la interfaz web se instaló y configuró un servidor web para su alojamiento y además se configuró la conexión HTTPS (HTTP seguro). Se procedió a instalar y configurar los ambientes de desarrollo que se utilizaron para las herramientas seleccionadas (Django, PostgreSQL, React). Se desarrolló la interfaz siguiendo el diseño planteado previamente y finalmente se alojó la aplicación en el servidor para que pueda ser accesible desde Internet.

La implementación del Sistema se realizó en un computador con el sistema operativo Ubuntu 16.04 LTS. Los pasos que aquí se explican han sido ejecutados con este sistema operativo.

## 4.1 Instalación y configuración del servidor web

Debido a que la instalación del servidor Apache es bastante común y directa, se hace un pequeño resumen y se hacen referencias a sitios web donde se encuentra la descripción detallada del proceso.

Primero se deben hacer algunas configuraciones iniciales para aumentar la seguridad

del servidor, como por ejemplo agregar un usuario con autenticación segura utilizando *ssh*. Para esto se genera un par de claves RSA, una pública y otra privada las cuales se utilizan para acceder al servidor utilizando *ssh*. También se desactiva la autenticación por par usuario-contraseña para evitar un acceso no deseado. Luego se configura el firewall para solo permitir conexiones a los puertos deseados, en [31] se encuentra una lista de pasos detallados para ejecutar estas configuraciones.

Una vez realizada la configuración inicial del servidor, se procedió a instalar los paquetes necesarios para el servidor web, en este caso, el servidor web Apache. Primero se instala el software Apache, éste software se encuentra en los repositorios de Ubuntu por lo que se puede instalar por consola ejecutando el siguiente comando:

```
$ apt-get install apache2
```



Una vez instalado el programa, se procede a añadir las reglas necesarias al firewall para que éste permita el acceso al servidor web en los puertos requeridos para la comunicación HTTP y HTTPS. Esto se logra utilizando UFW, un programa que permite administrar el firewall del sistema operativo. Al hacer esto, ya el servidor está funcionando. En el primer paso de la siguiente guía a la que se hace referencia en [32] se pueden encontrar los pasos detallados para lograr cumplir con esta instalación.

Luego se procedió a crear los certificados SSL que utiliza el servidor para implementar la conexión segura, es decir, implementar HTTPS. Estos certificados son utilizados solo con fines educativos pues no han sido certificados por una CA (*Certification Authority*) o Autoridad Certificadora, por lo que el navegador web lo identifica como una conexión no confiable.

Para crear los certificados se utilizó la herramienta libre OpenSSL, esta herramienta permite crear certificados propios que pueden ser utilizados localmente. Al ejecutarla, esta solicita la información básica con la que sera creado el certificado, esta información sera la que se muestre cuando un usuario solicite ver el certificado. Algunos de los datos solicitados son: País, Estado o Provincia, Organización etc. Se utilizaron los datos de la Universidad de Los Andes. Al completar la información el programa creara el par

de claves (privada y pública) que se utiliza para la conexión segura. Estas claves se utilizan al momento de configurar el sitio web en el servido Apache. En la guía [33] se realiza esta operación paso a paso.

## 4.2 Instalación y configuración de Django y PostgreSQL

Debido a que se está trabajando con Python y se necesitan instalar paquetes y módulos para éste, se trabara con Pip. Pip es un instalador alternativo para paquetes de Python [34]. Su instalación en Ubuntu se ejecuta así (para Python 3):

```
$ sudo apt-get install python3-pip
```

Para desarrollar con Django se trabajó con la herramienta virtualenv [35]. Esta herramienta es utilizada para crear ambientes de desarrollo aislados, es decir, se encarga de instalar todos los programas y dependencias en un mismo ambiente, aislado del ambiente general del sistema. Esto facilita el proceso de desarrollo y el mantenimiento de dependencias. Toda la instalación de Django, React y sus dependencias se hace en un entorno virtual creado con virtualenv. Este entorno virtual utiliza Python 3 pues es una dependencia para el funcionamiento correcto de la última version de Django.



Para la instalación de Django, PostgreSQL y su configuración inicial se puede seguir la siguiente guía (para Python 3) [26]. Esta utiliza virtualenv como se dijo previamente. Una vez instalado y configurados esos dos paquetes se procede a instalar las dependencias necesarias para la implementación del sitio web, estas fueron instaladas con el ya mencionado Pip:

- Pillow: Utilizada para el manejo de imágenes con Python.

  ```
  $ pip install Pillow
  ```

- Pep8 y Flake8: Utilizados para la revisión de errores en el código e implementar buenas prácticas al programar.

  ```
  $ pip install pep8 flake8
  ```

- Path: utilizada para facilitar el manejo de los archivos y directorios dentro de los archivos de configuración del proyecto.

  ```
  $ pip install Path
  ```

- Webpack Loader: Paquete utilizado para cargar los paquetes creados por Webpack cuando se desarrolla en React.

  ```
  $ pip install django-webpack-loader
  ```

## 4.3 Instalación y configuración de React

Una vez instalado Django es necesario crear el ambiente para desarrollar en React, para ésto es necesario instalar varias dependencias incluyendo un intérprete de React, el empaquetador de módulos llamado Webpack que se encarga de compilar toda la aplicación desarrollada en React y convertirlo en archivos JavaScript interpretables por el navegador. Es con estos módulos empaquetados que se utiliza el ya mencionado *django-webpack-loader*, éste se encarga de buscarlos y cargarlos en la interfaz web cuando sea requerido. Para lograr esto se siguió la siguiente guía mostrada en [36]



## 4.4 Desarrollo de la interfaz

Ya con las herramientas instaladas se inició el desarrollo de la aplicación. Primero se creó una jerarquía de carpetas que facilita el manejo y mantenimiento del proyecto. En esta jerarquía se decidió dividir el Sistema en aplicaciones que se encarguen cada una de una tarea específica. Se creó una aplicación que maneje a los usuarios (users), otra que maneje los *frameworks* (frameworks) y otra que se encargue de la API (api). A continuación se muestra dicha jerarquía y se explica brevemente qué función cumple cada componente (los directorios se encuentran en negrita):

**Sistema Web**
- **api**
- **config**
- **frameworks**
- *manage.py*
- **proyecto_de_grado**
- **users**

El **Sistema Web** es el directorio raíz, es donde se encuentra todo el sistema desarrollado. Dentro de este directorio se encuentra lo siguiente:

- **api**: Directorio donde se encuentran todos los archivos utilizados para implementar la API, la cual es la que se encarga establecer las reglas y relaciones para la interacción entre los usuarios y los *frameworks*. Por ejemplo es aquí donde se encuentran las vistas de Django que pasan la información que reciben por parte de los *frameworks* al fron-tend desarrollado en React.



- **frameworks**: Directorio donde se encuentran los archivos con los que se implementa el manejo y administración de los *frameworks* en el sistema. Por ejemplo en este directorio se encuentran los modelos de Django que estructuran las tablas de las bases de datos que almacenan a los *frameworks*, también las vistas de Django que se encargan del funcionamiento de la aplicación y las *templates* de Django que muestran lo relacionado con los *frameworks* al usuario.

- **users**: Directorio donde se encuentran los archivos que se encargan del manejo de los usuarios. Al igual que en el directorio de los *frameworks*, en este directorio se almacenan los archivos necesarios para todo el funcionamiento que se relaciona con los usuarios en el sistema.

- **config**: Directorio donde se encuentran todos los archivos de configuración del sistema, es en éste donde se encuentra la configuración de la base de datos (usuario, contraseña, puerto, etc), además de otras configuraciones necesarias. Además se encuentran los archivos utilizados para desarrollar en React. Esto

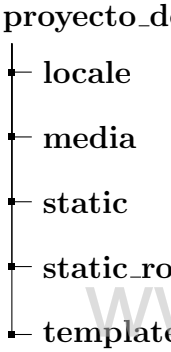
incluye archivos de configuración para Webpack en modo desarrollo y producción, los archivos JSX, etc.

- *manage.py*: Este archivo es utilizado por Django para ejecutar tareas de administración de sistema.

- **proyecto_de_grado**: En este directorio se encuentran todos los archivos generales del sistema, es decir, los que no son manejados por una aplicación.

A continuación se explica qué contiene cada uno de estos directorios.

En el directorio se **proyecto_de_grado** encuentran los siguientes componentes:





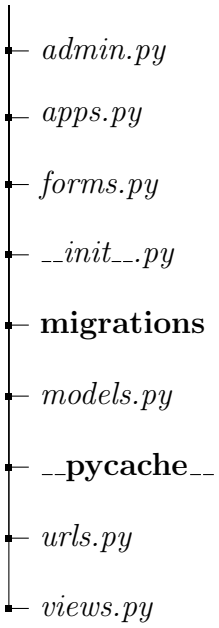

- **locale**: Directorio donde se encuentran los archivos que genera Django para el manejo de distintos idiomas en la interfaz web. Estos archivos hacen posible que el sistema se muestre en el idioma en que se encuentre configurado el navegador. media: Directorio donde se encuentran los archivos subidos por los usuarios, en este caso, solo las fotos de perfil que se muestran en el sistema.

- **static**: Directorio donde se encuentran los archivos estáticos que utiliza el sistema, las hojas de estilo, archivos JavaScript, etc.

- **static_root**: Directorio donde se acumulan los archivos estáticos que utiliza el sistema para que puedan ser introducidos al directorio static, que es donde los busca el servidor.

- **templates**: Directorio donde se encuentran todos los archivos HTML que Django utiliza para construir la interfaz que observa al usuario.

Los directorios **users**, **frameworks** y **api** comparten la siguiente estructura:




- **__pycache__** y **migrations**: Directorios utilizados por Django para mantenimiento.

- *__init__.py*, *admin.py*, *apps.py*: Archivos utilizados por Django para la configuración de cada una de las aplicaciones correspondientes (users, frameworks, api).

- *models.py*: Archivo donde se describe el modelo de datos de la aplicación, cada modelo se convierte en una tabla en la base de datos del sistema. Es aquí donde se declara el modelo de la base datos diseñado previamente.

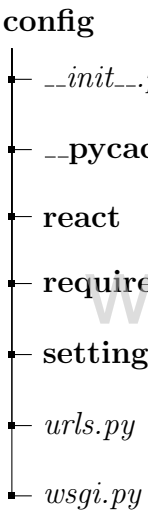
- *forms.py*: Archivo donde se especifican los formularios que utiliza la aplicación. En este archivo también se define la validación de todos los campos que el usuario introduce en cada uno de los formularios.

- *views.py*: Archivo donde se definen las clases de Django que reciben las peticiones del usuario y generan una respuesta a ésta. En su mayoría trabajan con respuestas a peticiones HTTP GET y HTTP POST, algunas solamente responden a peticiones HTTP GET. Es en este archivo donde se define la lógica de cada

aplicación, es decir, que debe responder a cada aplicación. Por ejemplo, si un usuario hace click en un botón y es dirigido a otra página, esta acción es realizada por una vista.

- *urls.py*: Archivo donde se declaran las URL de cada aplicación. Este archivo es el que especifica cómo se accede a cada vista. Cada URL (*Uniform Resource Locator*) está relacionada con una vista y es a ésta donde se envía la petición hecha a esa URL (*Uniform Resource Locator*). El archivo *urls.py* define el árbol de navegación de cada aplicación.

El directorio config, tiene la siguiente estructura:






- *__init__.py*: Archivo de configuración del Sistema.
- **__pycache__**: Directorio utilizado por Django para el mantenimiento del Sistema
- **react**: Directorio donde se encuentran los archivos utilizados para el desarrollo del API en React. Esto incluye archivos de configuración para el programa que compila los archivos de React (desarrollados en código JSX) y los convierte en archivos JavaScript interpretables por el navegador, además contiene los archivos que manejan la carga de los archivos JavaScript necesarios a la interfaz cuando esta así lo requiera.
- **requirements**: Directorio en donde se declaran las dependencias en archivos de texto que pueden ser leídos por un instalador de paquetes de Python, en este caso

PIP. Se crea un directorio para tener una mejor organización entre las distintas dependencias que se requieren en distintas fases del desarrollo.

- **settings**: Directorio en donde se encuentran los archivos de configuración necesarios para funcionamiento correcto de Django. En este directorio se encuentran varios archivos de configuración los cuales funcionan en distintas fases del desarrollo. Por ejemplo, en un ambiente de desarrollo y pruebas se utiliza una base de datos declarada en un archivo específico, mientras que en un ambiente de producción se utiliza otra base de datos diferente. Esto es posible de manera directa debido a la división de los archivos de configuración.

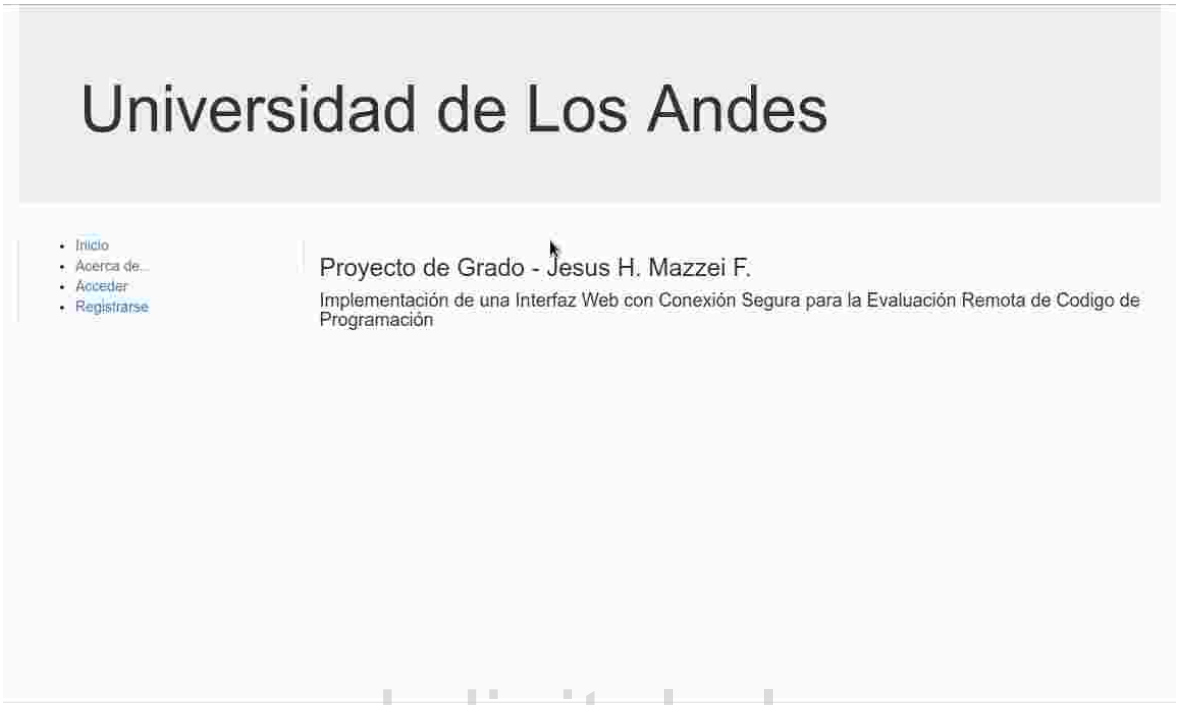
- *urls.py*: En este archivo se le especifica a la interfaz cómo acceder a las distintas aplicaciones creadas previamente (users, frameworks, api). En este archivo se hace referencia a los archivos *urls.py* creados en cada aplicación independiente y esto crea el árbol de navegación de todo la interfaz. Al incluir las URL de cada aplicación individualmente se otorga flexibilidad al momento de mantenimiento y depuración.



- *wsgi.py*: WSGI (*Web Server Gateway Interface*) es la especificación que describe cómo un servidor web se comunica con aplicaciones web en Python. Este es el archivo que se encarga de la comunicación con el servidor web y su funcionamiento con la interfaz web.

La página inicial de la interfaz web luce de la siguiente manera:

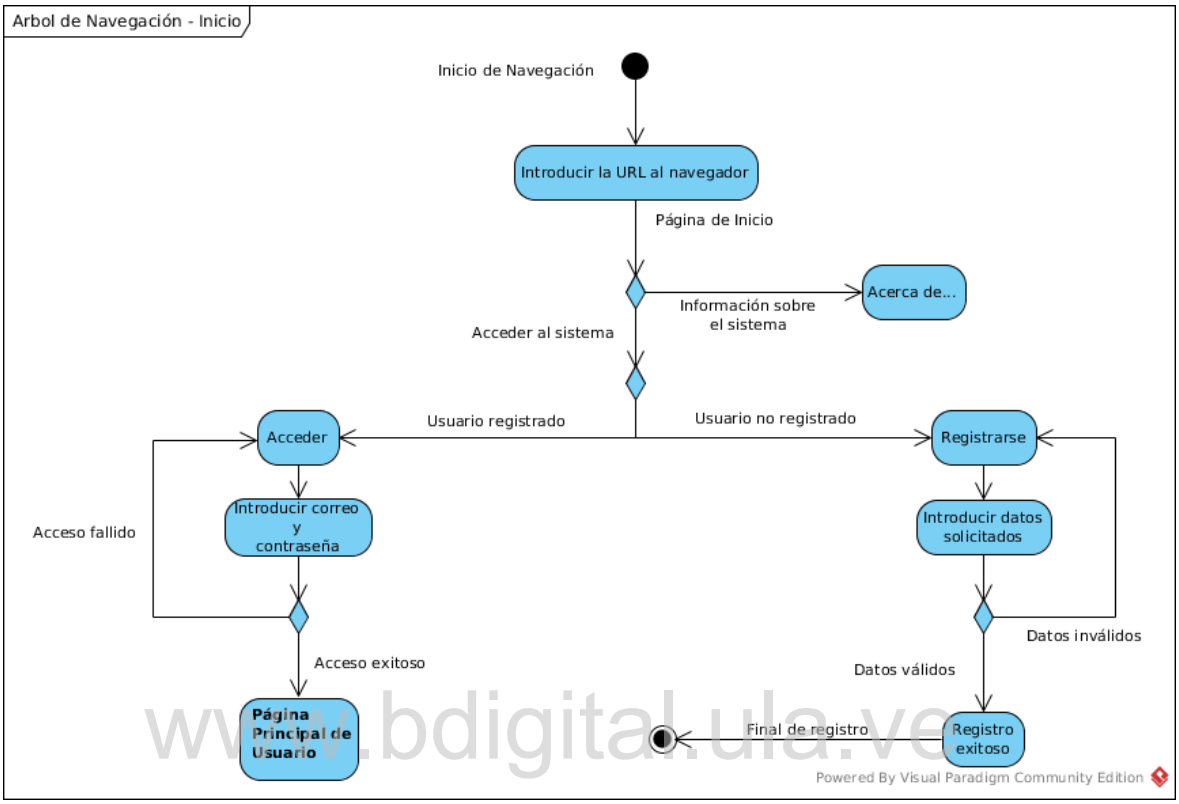
Figura 4.1: Captura de pantalla: Página principal de la interfaz web






La página principal de la interfaz ofrece varias opciones de navegación las cuales son representadas en el siguiente diagrama:

Figura 4.2: Diagrama de actividad: Arbol de navegacion - inicio





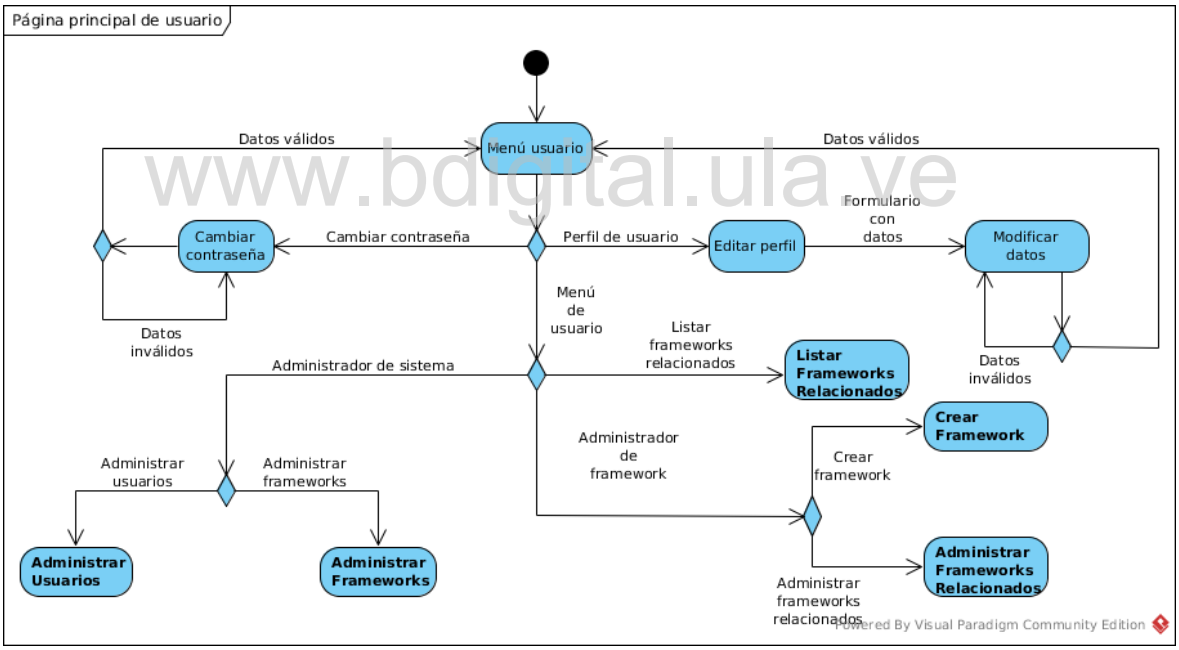

En este diagrama se muestra el flujo de navegación posible desde la página inicial de la interfaz. Como se puede apreciar, se pueden realizar tres acciones:

- Ver Información de la Interfaz en "Acerca de..."
- Acceder al Sistema
- Registrarse en el Sistema

Para egistrarse en el sistema se solicita información al Usuario la cual debe ser proporcionada de manera correcta, por ejemplo, la cédula debe tener el formato V000000000, el correo debe tener formato de correo y no debe haber sido registrado previamente, etc. Si el usuario introduce los datos correctamente éste es direccionado a una página donde se le informa que su registro ha sido exitoso y que podrá acceder al sistema una vez que un Administrador de Sistema haya verificado la información proporcionada y lo haya activado. Para acceder al sistema al usuario se le solicita el

correo y contraseña con que se registró, si estos datos son inválidos (mala combinación correo-contraseña o está inactivo en el Sistema), el mensaje de error se muestra en la misma página de acceso donde se encuentra, si los datos son correctos el usuario es direccionado a el menú principal del sistema para ese usuario, además, si el usuario olvidó su contraseña también puede solicitar una nueva desde la página de acceso. Para solicitar una contraseña nueva al usuario se le envía una dirección web única al correo que registró en el sistema, al acceder a este enlace puede introducir su clave nueva. Como ya se ha mencionado previamente los usuarios pueden tener distintos tipos de roles en el sistema. Dependiendo de su rol, un usuario tendrá acceso a distintos tipos de menú. En los siguientes diagramas se demuestra esto:

Figura 4.3: Diagrama de actividad: Pagina principal de usuario



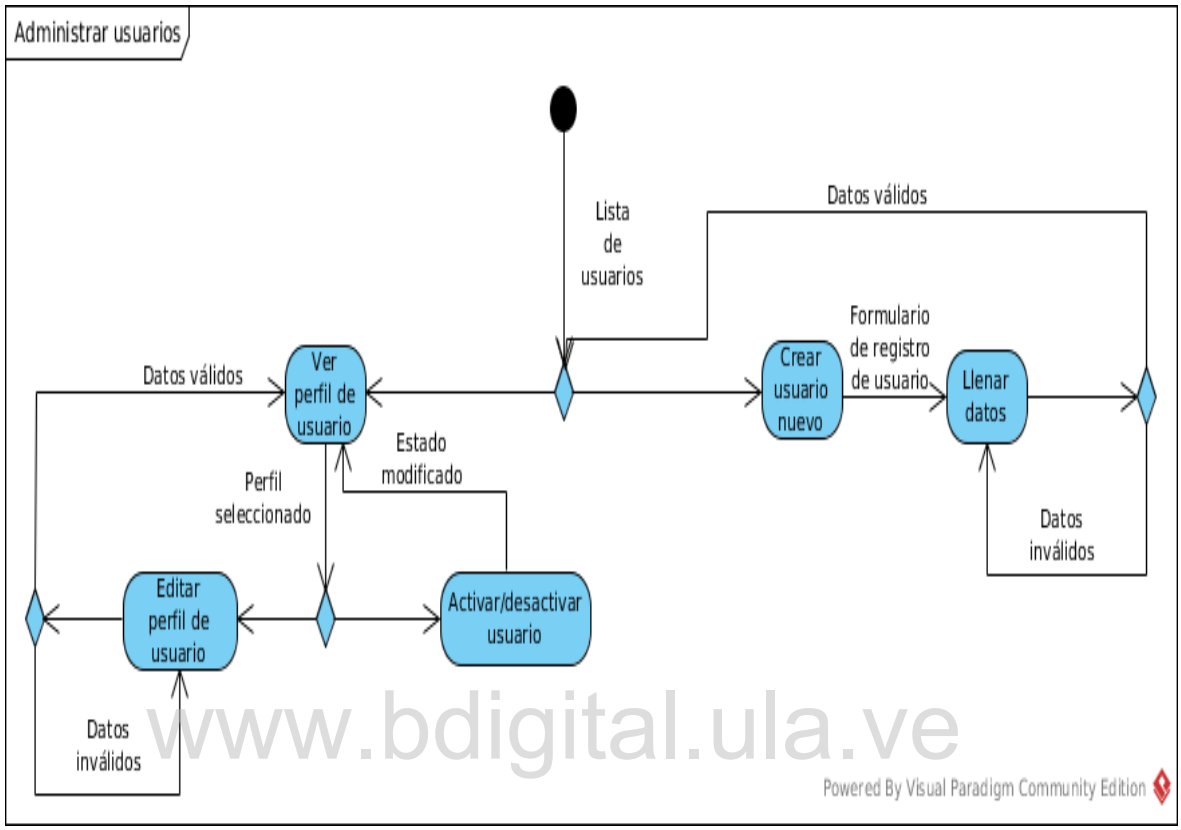

En la página principal del sistema como lo muestra este diagrama, se pueden ejecutar varias acciones. Todos los usuarios pueden editar su perfil y cambiar su contraseña sin importar su rol. Como fue especificado en el capítulo 2, cada tipo de usuario tiene ciertas actividades especiales que puede realizar. En la próxima lista se recuerdan cuales son estas actividades y a continuación se explican brevemente cómo se realiza cada una de éstas.

- Administrador de Sistema: Administrar Usuarios y Administrar Frameworks.
- Administrador de Framework: Listar Frameworks Relacionados, Administrar Frameworks Relacionados y Crear Framework.
- Usuario Común: Listar Frameworks Relacionados.

A continuación se presentan los diagramas para cada acción:

1. Administrador de Sistema
    - Administrar Usuarios: Como se muestra en el diagrama, al Administrador de Sistema se le presenta una lista de usuarios desde la cual puede seleccionar "Ver Perfil de Usuario" para ver la información de un usuario particular. Desde éste perfil el Administrador de Sistema puede activar o desactivar al usuario o editar el perfil de éste. Si el Administrador decide editar el perfil, los datos que modifique estarán sujetos a validaciones las cuales deben ser cumplidas, por ejemplo, la cédula debe tener el formato correcto, el correo electrónico no debe existir en el sistema etc.

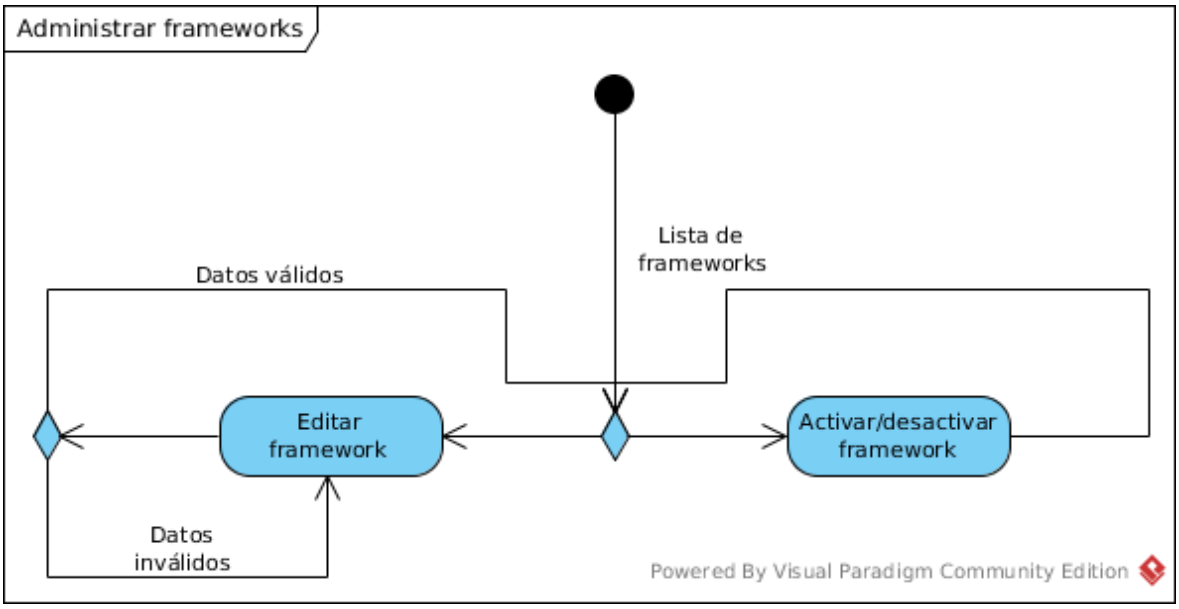
Figura 4.4: Diagrama de actividad: Administrar usuarios






- Administrar Frameworks: El Administrador obtiene una lista de *frameworks* desde la cual puede "Activar/Desactivar Frameworks" o editar la información sobre el Administrador de dicho *Framework*.

Figura 4.5: Diagrama de actividad: Administrar frameworks

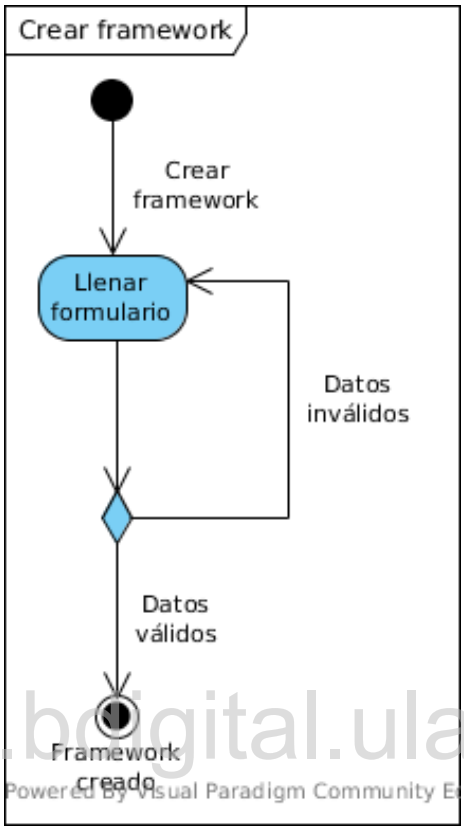



2. Administrador de Frameworks



   - Crear Framework: Esto permite al Administrador de Frameworks agregar nuevos frameworks a la interfaz y administrar más de un framework.

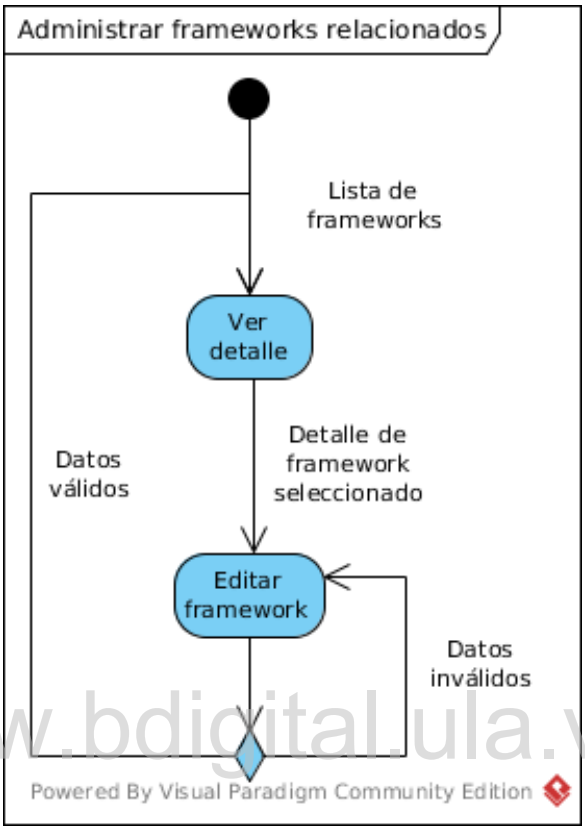
Figura 4.6: Diagrama de actividad: Crear framework






- Administrar Frameworks Relacionados: Esto permite al Administrador de Frameworks ver información sobre los *frameworks* que administra en la interfaz y también editar información sobre éstos.

Figura 4.7: Diagrama de actividad: Administrar frameworks relacionados

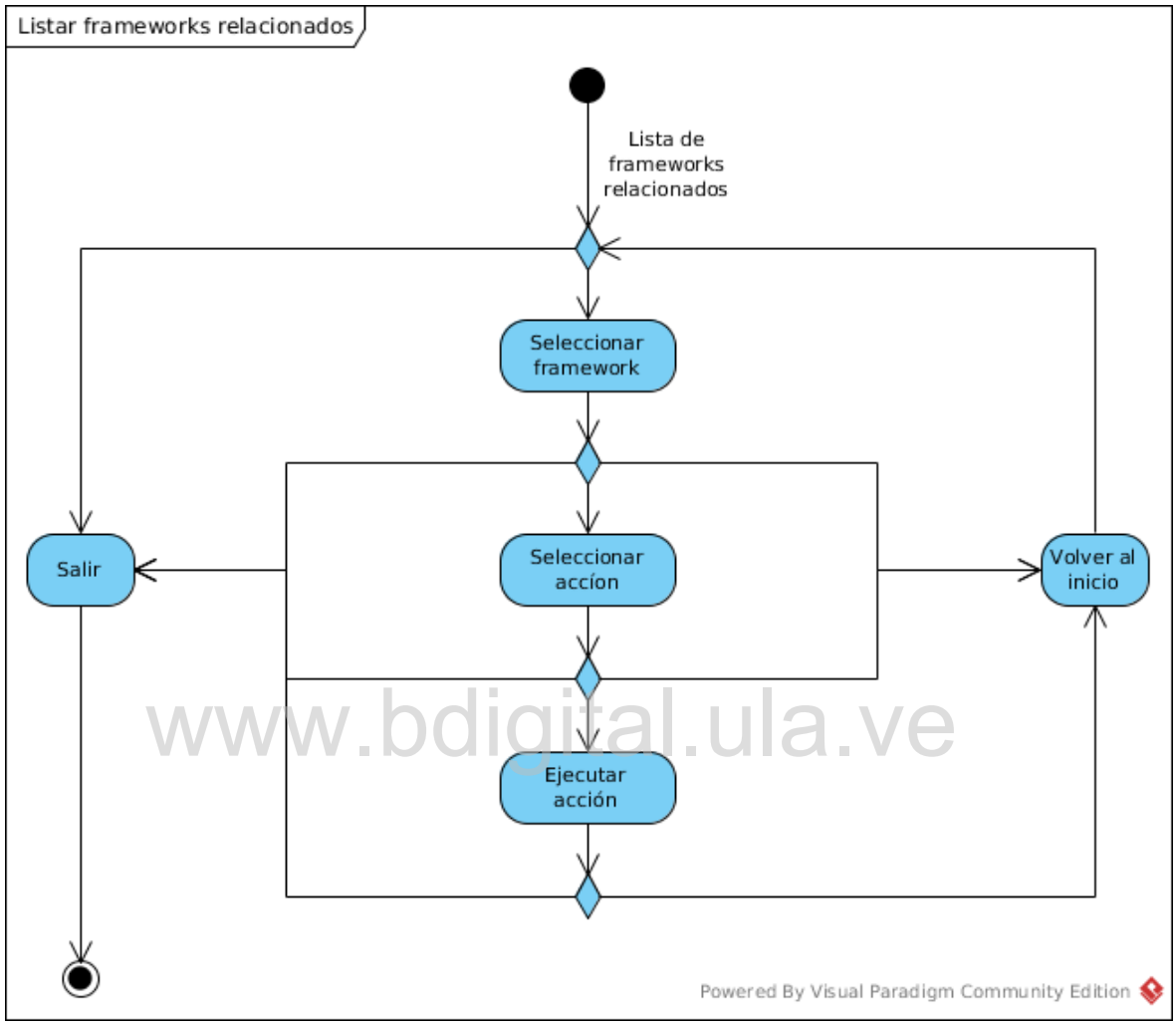





3. Administrador de Frameworks y Usuario Común:

   - Listar Frameworks Relacionados: Esto permite a un Administrador de Frameworks o Usuario Común interactuar con los *frameworks* a los que se encuentra relacionado. Es aquí donde la API se encarga de entablar una conversación con los *frameworks* externos correspondientes. Inicialmente el usuario en cuestión (Administrador o Usuario Común) obtiene una lista de los *frameworks* con los que puede interactuar, al seleccionar uno, obtendrá una lista de acciones disponibles ofrecidas por dicho *framework* y al seleccionar una acción. Se le presentara ésta para que la ejecute.

Figura 4.8: Diagrama de actividad: Listar frameworks relacionados

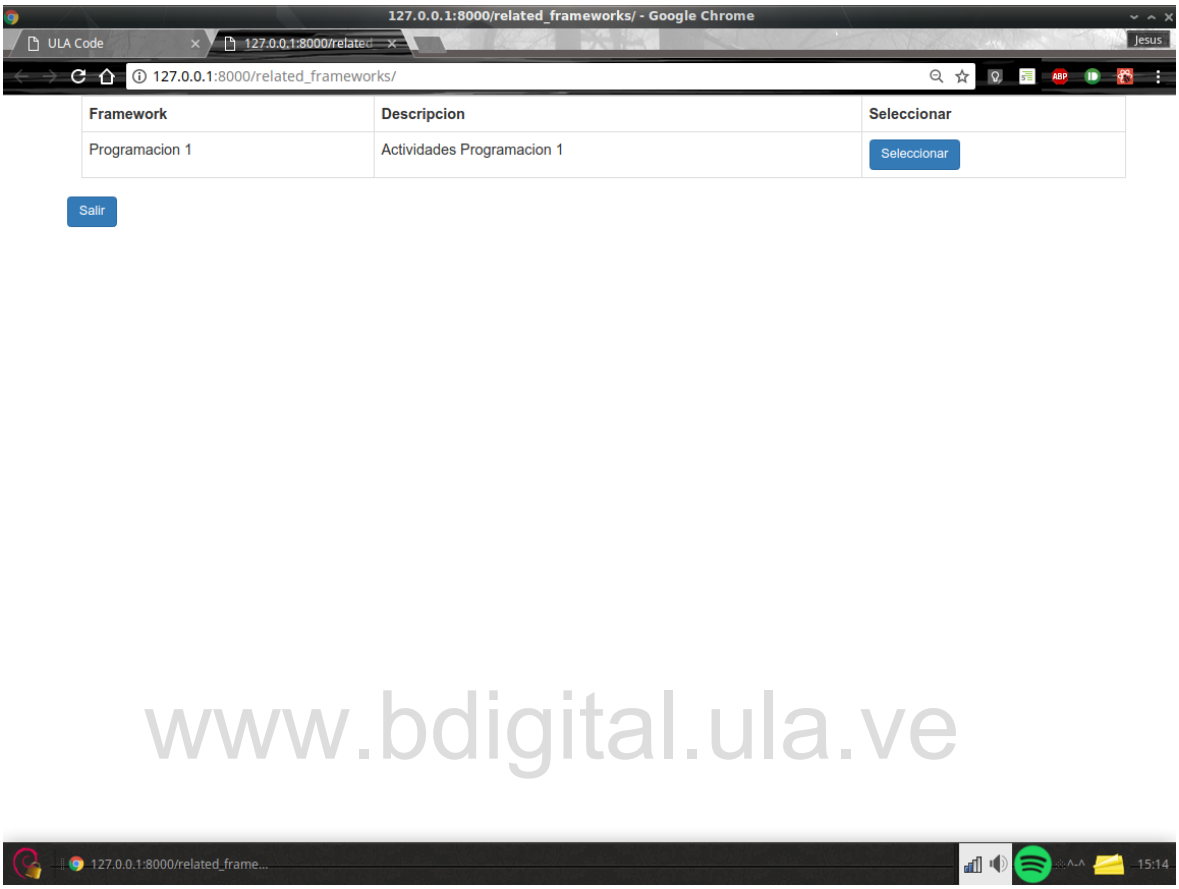





Como se puede apreciar, la interfaz ejecuta todas las operaciones mencionadas previamente pero también ofrece la posibilidad de salir de éstas en cualquier momento si el usuario así lo desea. También ofrece la posibilidad de volver al inicio de la interacción, lo que mostraría al usuario de nuevo la lista de *frameworks* relacionados.

En la siguiente captura de pantalla se muestra cómo a un usuario se le ofrece la lista de *frameworks* a la que se encuentre relacionado, en este caso, un solo *framework* de programación.

Figura 4.9: Captura de pantalla: Listar frameworks relacionados

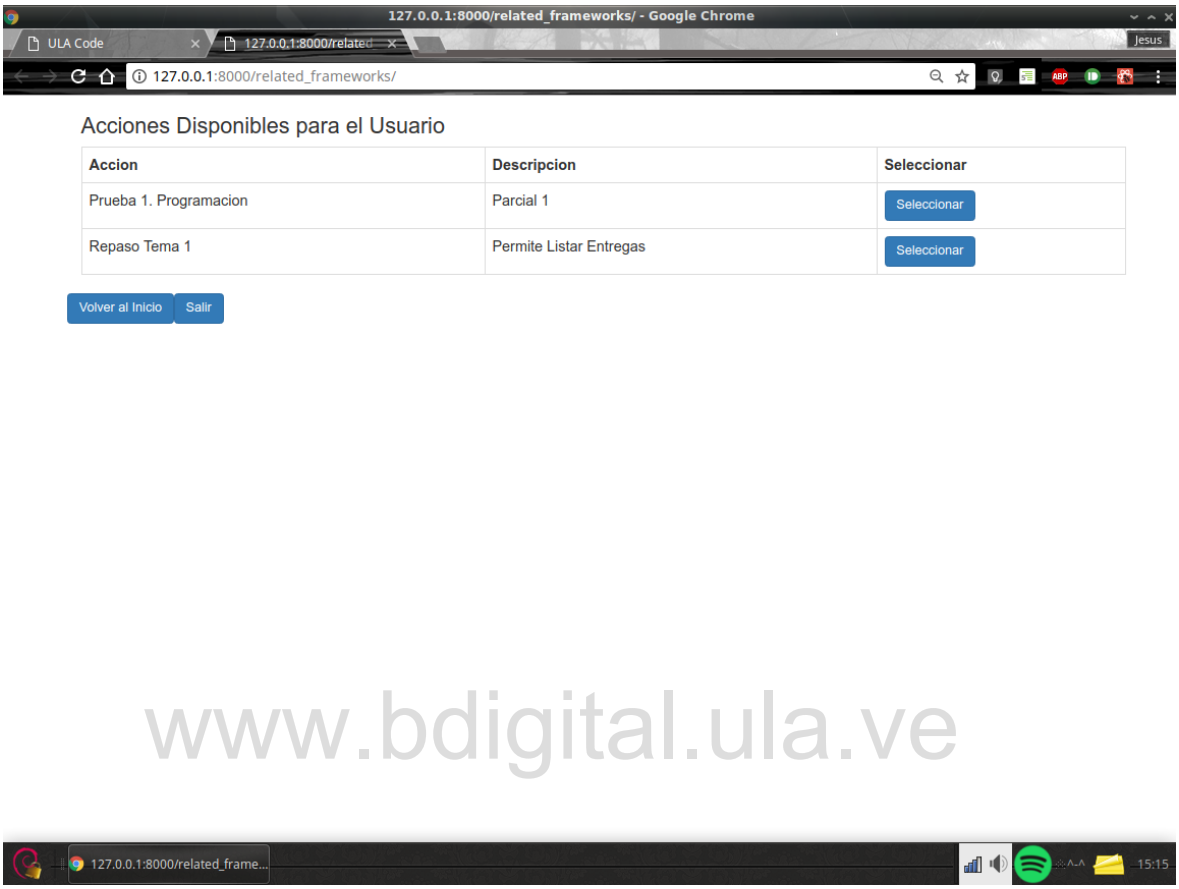





Como se puede apreciar, dicha lista de *frameworks* consiste de un nombre para cada uno y también una descripción, además de un botón que permite al usuario seleccionar el *framework* con el que quiera interactuar.

En la siguiente captura se muestra una lista de actividades ofrecidas por el *framework* seleccionado, en este ejemplo el usuario seleccionó el único *framework* que tenía disponible, "Programación 1". A continuación se le muestra una lista con las acciones disponibles para el usuario.

Figura 4.10: Captura de pantalla: Listar de acciones

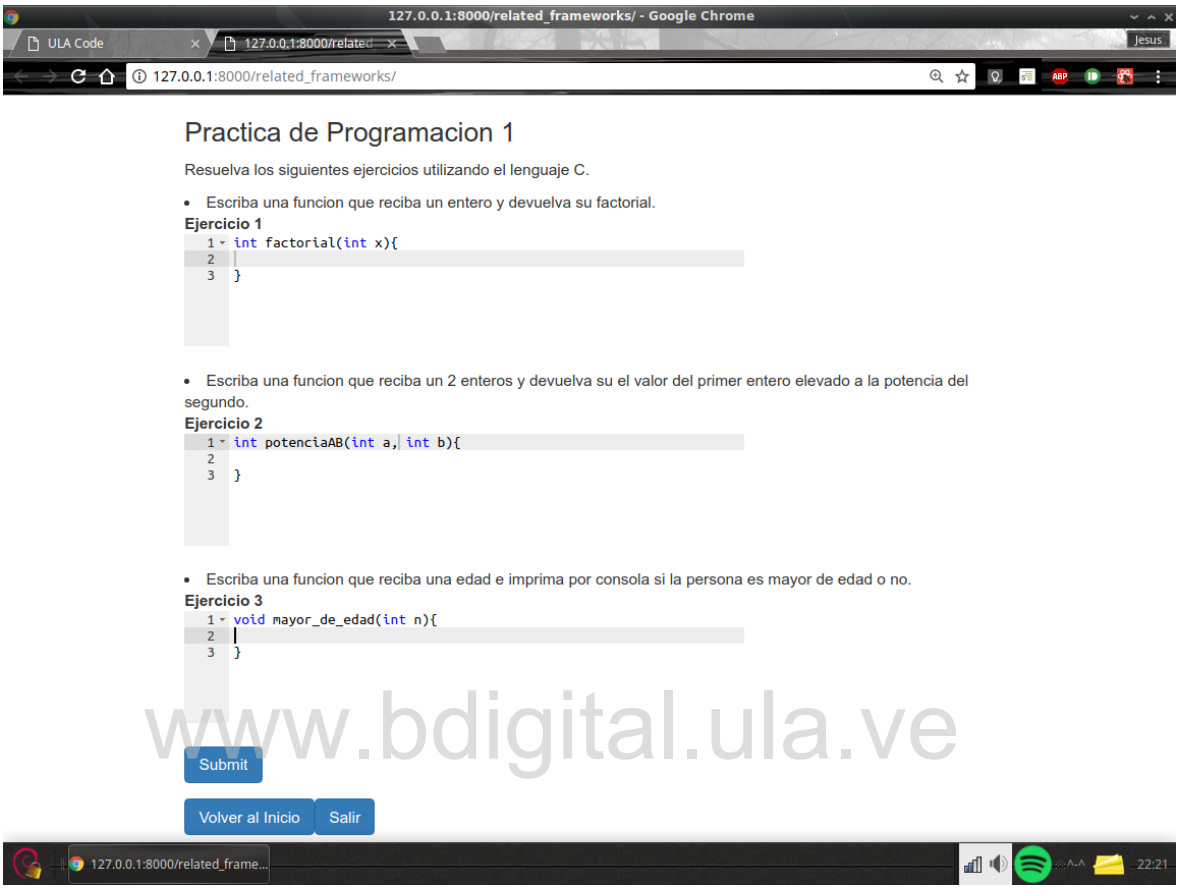





En esta captura se puede apreciar que la lista de actividades ofrece un nombre y una descripción para cada una de éstas y cada una de estas acciones tiene un botón asignado que permite seleccionarla. También se puede apreciar los botones "Volver al Inicio" y "Salir" los cuales permiten al usuario volver a la lista de *frameworks* relacionados o salir de la página lo que cierra la pestaña del navegador en la que se muestra esta lista.

Continuando con este ejemplo, en la próxima captura de pantalla se ofrece una muestra de la acción "Prueba 1. Programación", acción disponible en la lista de acciones mostradas en la captura anterior. Esto muestra una prueba de tres ejercicios prácticos de programación. Esta acción requiere que el usuario envíe información al *framework* a través de la interfaz, en este caso, la información son las respuestas a los ejercicios.

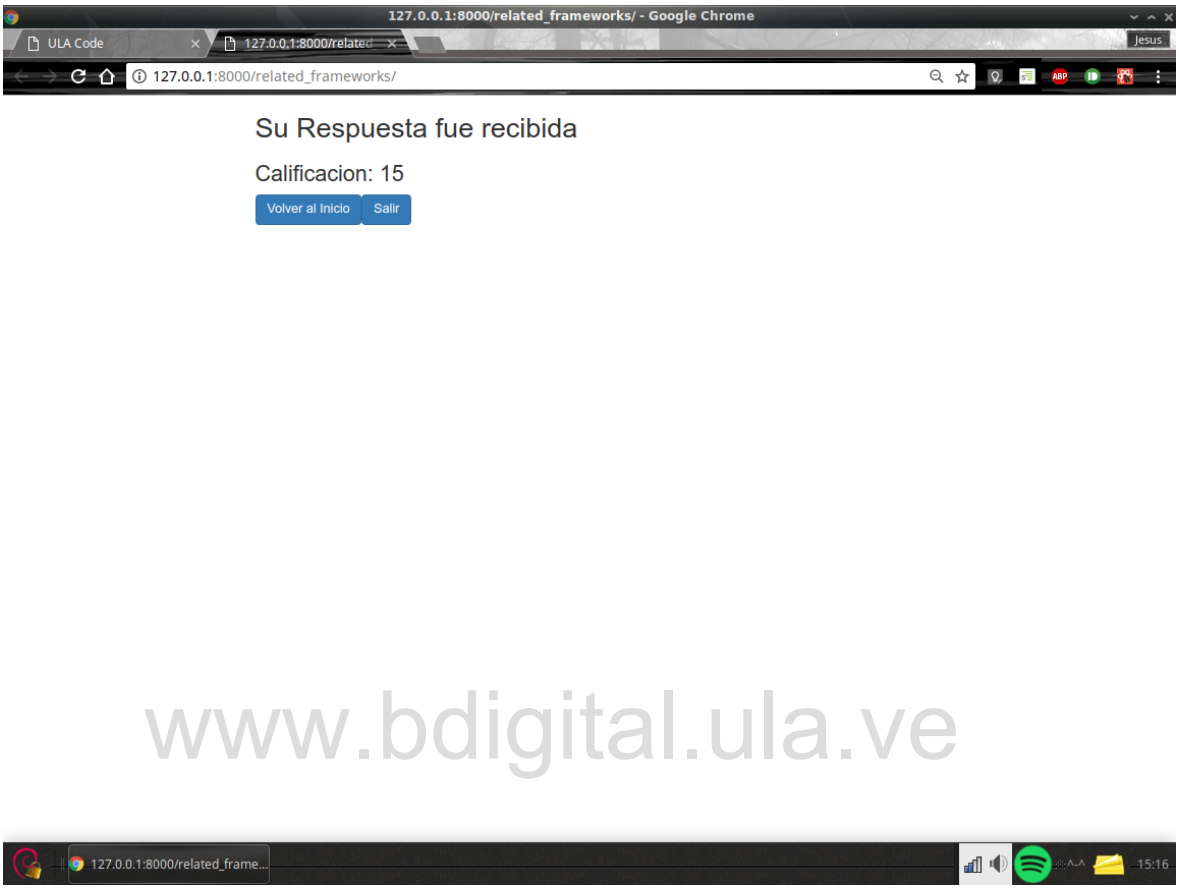
Figura 4.11: Captura de pantalla: Accion




En una acción de este tipo, es decir, que requiera envío de información por parte del usuario, se muestra un botón llamado "Submit" (Enviar) el cual envía lo introducido por el usuario al *framework* correspondiente. También muestra los botones mencionados previamente que permiten salir de la pestaña del navegador.

Para concluir este ejemplo, se muestra la respuesta enviada por el *framework* una vez recibe la información enviada por el usuario. Esta respuesta es un ejemplo de una acción que no requiere información por parte del usuario. Es por esto que las únicas opciones posibles son volver al inicio o salir.

Figura 4.12: Captura de pantalla: Resultado






Como se explicó en cada captura de pantalla anterior, este ejemplo demuestra cómo un estudiante de programación 1 podría presentar una prueba práctica ofrecida por un *framework* a través de la interfaz. La idea de este ejemplo es mostrar cómo trabaja la API cuando un usuario solicita interactuar con *frameworks* a través de ella. Esta secuencia de interacciones no es más que la implementación del diseño explicado previamente en este documento visto desde la perspectiva del usuario, es decir, se aprecia el funcionamiento de la API como se plantéo. A continuación se explica más a fondo cómo se implemento la API internamente, explicando sus reglas y cómo se usaron las herramientas para lograr el funcionamiento requerido.

En principio, se especifica cuáles son las reglas que exige la API para interactuar con *frameworks*.

- La API trabaja con solicitudes y respuestas en formato JSON.

- Cabe acotar que en todas las solicitudes van como contenido de la cabecera el token de autenticación del usuario para con el *framework* con el que va a interactuar, esto permitirá al *framework* identificar al usuario y enviarle la respuesta específica a éste.

- Para iniciar la comunicación con los *frameworks* la interfaz envia una solicitud HTTP GET al URL (*Uniform Resource Locator*) de dicho *framework* que está almacenado en la base de datos. La respuesta a esta solicitud debe ser una lista de materias disponibles para el que hizo la solicitud Usuario. Se dice "materias" porque cabe la posibilidad de que un *frameworks* maneje varias materias por lo que pueda ofrecer a un usuario más de una. Esta respuesta debe tener formato JSON, en la cual debe haber un objeto padre llamado "info", el valor de este elemento debe ser un arreglo de objetos, cada uno de los cuales debe ser una materia con el nombre, descripción y el enlace al que se debe hacer la solicitud para obtener la lista de acciones de esta materia. Este enlace se anexa a la URL (*Uniform Resource Locator*) base del *framework* que se encuentra en la base de datos, es decir, si un *framework* tiene como URL (*Uniform Resource Locator*) http://170.23.10.9 y una materia tiene como enlace /compiladores, esto quiere decir que la solicitud para obtener las acciones disponibles para esta materia se hace a la URL (*Uniform Resource Locator*) http://170.23.10.9/compiladores. En la siguiente figura se muestra un ejemplo de un *framework* que ofrece dos materias al Usuario.

Figura 4.13: Lista de materias

```
{
    "info":
        [
            {
            "nombre": "Compiladores",
            "link": "/compiladores",
            "descripcion": "Asignatura Compiladores, 8voSemestre"
            },
            {
            "nombre": "Redes",
            "link": "/redes",
            "descripcion": "Redes"
            }
        ]
}
```

- Una vez seleccionada la materia, se hará una solicitud HTTP GET a la URL (*Uniform Resource Locator*) mencionada previamente. La respuesta a esta solicitud debe ser una lista de acciones que se le ofrezcan al usuario (identificado por el token de autenticación). Esta respuesta debe tener formato JSON y debe tener un objeto padre llamado "actions_list", este objeto debe tener como valor un arreglo de objetos, cada uno de los cuales debe, al igual que los objetos de la lista de materias, tener nombre, link (enlace) y descripción. En la siguiente figura se muestra esto.



Figura 4.14: Lista de acciones

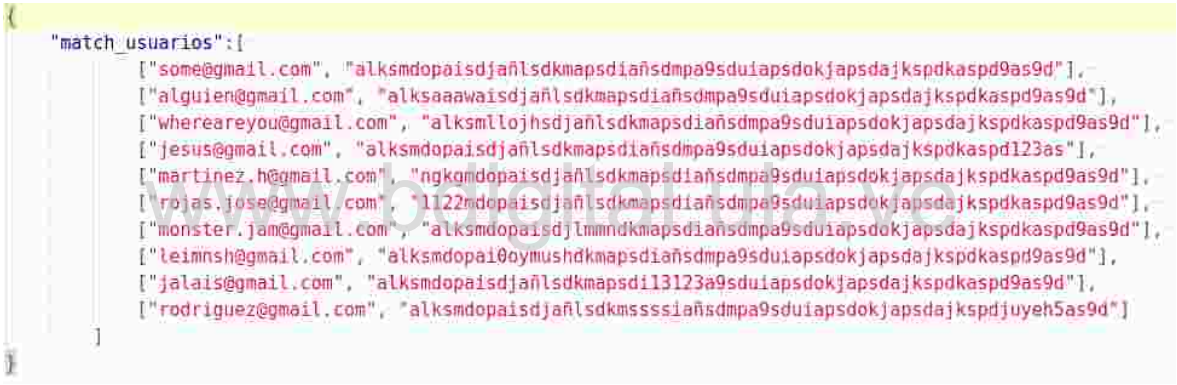
```
{
    "actions_list": [{
            "nombre": "Prueba 1. Programacion",
            "link": "/pr1/prueba/1",
            "descripcion": "Parcial 1"
        },
        {
            "nombre": "Repaso Tema 1",
            "link": "/pr1/teoria/1",
            "descripcion": "Permite Listar Entregas"
        }
    ]
}
```

- Al seleccionar la acción, se hará la solicitud al enlace proporcionado, a esta solicitud el framework puede responder con una de las siguientes tres acciones

posibles.

- match_usuarios: Esta acción se encarga de realizar una sincronización entre los usuarios registrados en el *framework* y los usuarios relacionados con dicho *framework* en la interfaz. La respuesta debe ser enviada en formato JSON y debe tener un objeto padre de nombre "match_usuarios" el cual como valor debe tener un arreglo cuyos elementos deben ser un arreglo para cada usuario, en cada arreglo de un usuario debe ir como primer elemento el correo del usuario y como segundo elemento el token de autenticación de este usuario para con este *framework*. La siguiente figura ilustra esto.

Figura 4.15: Sincronizacion de usuarios




- view_action: Esta acción debe llevar la descripción gráfica de los elementos HTML que representan la acción seleccionada, la cual no requiere envío de información por parte del usuario. La respuesta debe ser enviada en formato JSON y el objeto padre se debe llamar "view_action", su valor debe ser un objeto JSON que debe llevar el campo "elements" el cual debe tener como valor un arreglo de objetos, cada uno de los cuales debe describir la representación de un elemento HTML Estos objetos comparten los siguientes campos:
  - etiqueta: La etiqueta del elemento HTML.
  - id: El identificador único del elemento HTML en todo el documento.lxv

  Dependiendo del elemento HTML que se quiera renderizar, se deben agregar

otros campos. A continuación se especifican cuáles son estos campos y cómo funcionan.

* Elemento HTML básico: Estos elementos se describen de la misma manera sus atributos mínimos son iguales. Entre estos elementos se encuentran p, h1, h2, h3, h4, h5, h6 y li. Cada elemento debe llevar los campos previamente mencionados, además del siguiente:
  - content: Texto que va envuelto entre las etiquetas. A continuación un ejemplo:

Figura 4.16: Representación gráfica de elemento HTML p

```
{
    "etiqueta": "p",
    "id": "Enunciado",
    "content" : "Resuelva los siguientes ejercicios utilizando el lenguaje C."
},
```

Esto se traduce en el siguiente elemento HTML:



Figura 4.17: Traducción de representación gráfica de elemento p

```
<p id="Enunciado">Resuelva los siguientes ejercicios utilizando el lenguaje C.</p>
```

* Imagen en HTML: Este elemento se debe describir de manera especial pues tiene atributos necesarios que no comparte con los elementos HTML previamente mencionados. Las imágenes deben tener los siguientes campos extras:
  - src: Enlace donde se busca la imagen para cargarla al navegador. Cabe acotar que es recomendable que la imagen que se desea mostrar se encuentre en un servidor externo al *framework* pues este enlace debe ser absoluto, es decir, si está alojada en el *framework*, el usuario podrá ver la URL (*Uniform Resource Locator*) de éste.
  - alt: Descripción breve de la imagen.

  A continuación un ejemplo:

Figura 4.18: Representación gráfica de elemento HTML img

```
{
    "etiqueta": "img",
    "id": "img_1",
    "src":
    "https://www.google.com/images/branding/googleg/1x/googleg_standard_color_128dp.pn
    g",
    "alt": "Imagen cualquiera de Google"
},
```

Esto traduce en el siguiente elemento HTML:

Figura 4.19: Traducción de representación gráfica de elemento img

```
<img key="3" id="img_1" src="https://www.google.com/images/branding/googleg/1x/…" alt="Imagen cualquiera de Google" />
```

* Imagen en HTML: Este elemento se debe describir de manera especial pues tiene atributos necesarios que no comparte con los elementos HTML previamente mencionados. Las imágenes deben tener los siguientes campos extras:
  - href: Enlace que se abrirá al hacer click en éste. Estos enlaces serán abiertos en una nueva ventana del navegador. Al igual que con la imagen, es recomendable que el enlace que se desea mostrar se encuentre en un servidor externo al *framework*.
  - content: Texto que va entre las etiquetas y se le muestra al usuario como descripción del enlace.

  A continuación un ejemplo:



Figura 4.20: Representación gráfica de elemento HTML a

```
{
    "etiqueta": "link",
    "id": "archivo a descargar",
    "content" : "Modulo 0 - Seguridad Informatica",
    "href":
    "https://www.nebrija.es/~cmalagon/seguridad_informatica/transparencias/Modulo_0.pd
    f"
}
```

Esto traduce en el siguiente elemento HTML:

Figura 4.21: Traducción de representación gráfica de elemento a

```
<a key="4" id="archivo a descargar" href="https://www.nebrija.es/~cmalagon/seguridad_informa…">Modulo 0 - Seguridad Informatica</a>
```

Un view_action completo se describire así.

Figura 4.22: View action completo

```
{
    "view_action": {
        "elements": [
            {
                "etiqueta": "h3",
                "id": "pregunta_1",
                "atributos": "",
                "content" : "Cual es el nombre del inventor del lenguaje C++?"
            },
                {
                "etiqueta": "img",
                "id": "img_1",
                "src":
                "https://www.google.com/images/branding/googleg/1x/googleg_standard_color_128dp.png",
                "alt": "Imagen cualquiera de Google"
            },
            {
                "etiqueta": "link",
                "id": "archivo a descargar",
                "content" : "Modulo 0 - Seguridad Informatica",
                "href":
                "https://www.nebrija.es/~cmalagon/seguridad_informatica/transparencias/Modulo_0.pdf"
            }
        ]
    }
}
```



– form_action: Esta acción permite lo mismo que view_action con el valor agregado que puede representar elementos que permitan al usuario enviar información al *framework* por medio del uso de formularios HTML, es decir, permite representar elementos de entrada. Para representar un form_action la respuesta en formato JSON debe tener un objeto padre llamado "form_action", su valor debe ser un objeto que tenga un campo llamado "elements" cuyo valor sea un arreglo de objetos, cada uno representando el elemento HTML a mostrar y también un campo llamado "extra_info", el cual debe ser un objeto donde se almacene todos los campos extras que

son necesarios para enviar este formulario al *framework* pero que no son introducidos por el usuario, por ejemplo, si el *framework* necesita que le envíen un "id" al que está relacionado el formulario, es en "extra_info" donde debe ir especificado este "id". También en este campo "extra_info" debe ir la URL (*Uniform Resource Locator*) a la que se debe enviar este formulario en el *framework*. Esta URL (*Uniform Resource Locator*) se anexa a la URL (*Uniform Resource Locator*) base del *framework*. Todos los elementos de entrada comparten los siguientes campos:

* etiqueta: La etiqueta HTML del elemento. En todos los casos es "input" excepto en el caso del editor de código, pues se usa un programa llamado Ace Editor que no funciona con la etiqueta "input", para este editor se utiliza la etiqueta "code".
* label: Texto que se coloca precedente al campo de entrada, puede ser utilizado para hacer una pregunta y que su respuesta sea introducida en el campo de entrada.



* name: Nombre único que se le da al campo de entrada. Éste es el identificador que utiliza HTML para enviar el valor introducido por el usuario, es decir, si un elemento tiene como "name" el valor "pregunta_1", la entrada introducida en este elemento será identificada como "pregunta_1" al momento de enviar el formulario al *framework*.
* value: Valor previo del campo de entrada. Por ejemplo si en un ejercicio práctico de programación al usuario se le desea dar un la cabecera de una función, es en este campo donde se debe agregar.
* type: Tipo de entrada, entre estos tipos se encuentran text, radio, checkbox, code y file. Éstos tipos se explican a continuación
  - text: Esta entrada se utiliza texto simple. Este tipo no tiene campos extras.

    Un ejemplo de este tipo:

Figura 4.23: Representación gráfica de elemento HTML input tipo "text"

```
{
    "etiqueta": "input",
    "label" : "Cual es el nombre del inventor del lenguaje Java?",
    "name": "pregunta_1",
    "type": "text",
    "value":""
},
```

Esto traduce en el siguiente elemento HTML:

Figura 4.24: Traducción de representación gráfica de elemento input tipo "text"

```
▼<label>
  "Cual es el nombre del inventor del lenguaje Java?"
 ▼<div>
    <input type="text" value="" name="pregunta_1"… />
  </div>
 </label>
```



- radio: Este tipo de entrada permite mostrar un elemento de selección simple al Usuario. Para este tipo se debe agregar un campo extra llamado "opciones" el cual tenga como valor un arreglo de objetos. Cada objeto debe tener dos campos, uno llamado "value" con el valor que se envía por el formulario si esta opción es seleccionada y otro llamado "contenido" cuyo valor se muestra en el navegador para esta opción.
  Un ejemplo de este tipo:

Figura 4.25: Representación gráfica de elemento HTML input tipo "radio"

```
{
    "etiqueta": "input",
    "label" : "En que lenguaje esta escrito el kernel de Linux?",
    "name": "pregunta_3",
    "type": "radio",
    "opciones":[{
        "value": "c",
        "contenido": "Lenguaje C"
    },
    {
        "value": "c++",
        "contenido": "Lenguaje C++"
    },
    {
        "value": "java",
        "contenido": "Java"
    }
    ],
    "value":""
},
```



Esto traduce en el siguiente elemento HTML:

Figura 4.26: Traducción de representación gráfica de elemento input tipo "radio"

```
▼<label>
  "En que lenguaje esta escrito el kernel de Linux?"
 ▼<div>
    <input type="radio" value="c" name="pregunta_3"… />
    " "
    "Lenguaje C"
  </div>
 ▼<div>
    <input type="radio" value="c++" name="pregunta_3"… />
    " "
    "Lenguaje C++"
  </div>
 ▼<div>
    <input type="radio" value="java" name="pregunta_3"… />
    " "
    "Java"
  </div>
</label>
```

· checkbox: Esta entrada permite mostrar un elemento de selección

múltiple al Usuario. Para este tipo también se debe agregar el campo extra llamado "opciones" explicado para el tipo radio.
Un ejemplo de este tipo:

Figura 4.27: Representación gráfica de elemento HTML input tipo "checkbox"

```
{
    "etiqueta": "input",
    "label" : "Cuales de estas palabras reservadas se usan para implementar bloques de
    repeticion en el lenguaje C",
    "name": "pregunta_8",
    "type": "checkbox",
    "opciones":[{
        "value": "if",
        "contenido": "if"
    },
    {
        "value": "for",
        "contenido": "for"
    },
    {
        "value": "while",
        "contenido": "while"
    },
    {
        "value": "switch",
        "contenido": "switch"
    }
    ],
    "value":[]
},
```



Esto traduce en el siguiente elemento HTML:

Figura 4.28: Traducción de representación gráfica de elemento input tipo "checkbox"

```
▼<label>
  "Cuales de estas palabras reservadas se usan para implementar bloques de repeticion en el lenguaje C"
 ▼<div>
    <input type="checkbox" value="if" name="pregunta_8"… />
    " "
    "if"
  </div>
 ▼<div>
    <input type="checkbox" value="for" name="pregunta_8"… />
    " "
    "for"
  </div>
 ▼<div>
    <input type="checkbox" value="while" name="pregunta_8"… />
    " "
    "while"
  </div>
 ▼<div>
    <input type="checkbox" value="switch" name="pregunta_8"… />
    " "
    "switch"
  </div>
</label>
```

- codeEditor: Permite al usuario enviar código de programación utilizando el editor de texto Ace Editor. Este tipo no tiene campos extras.

  Un ejemplo de este tipo:

Figura 4.29: Representación gráfica de editor de código

```
{
    "etiqueta": "code",
    "type": "codeEditor",
    "label" : "Ejercicio 1",
    "name": "ejercicio_1",
    "value": "int factorial(int x){\n\n}"
},
```

Esto traduce en el siguiente elemento HTML:

Figura 4.30: Traducción de representación gráfica de editor de código

```
▼<label>
   "Ejercicio 1"
  ▼<div>
   ▼<ReactAce ref=ref() mode="java" theme="textmate" height="100px"…>
      <div ref=bound updateRef() id="ejercicio_1" style={width: "500px", height: "100px"} />
    </ReactAce>
   </div>
  </label>
```

· file: Permite al Usuario enviar un archivo al *framework*. Este tipo de entrada no tiene el campo "value" que tienen los otros tipos de elementos "input".
Un ejemplo de este tipo:

Figura 4.31: Representación gráfica de elemento HTML input tipo "file"

```
{
    "etiqueta": "input",
    "label" : "Suba un archivo comprimido aqui.",
    "name": "file_1",
    "type": "file"
}
```



Esto traduce en el siguiente elemento HTML:

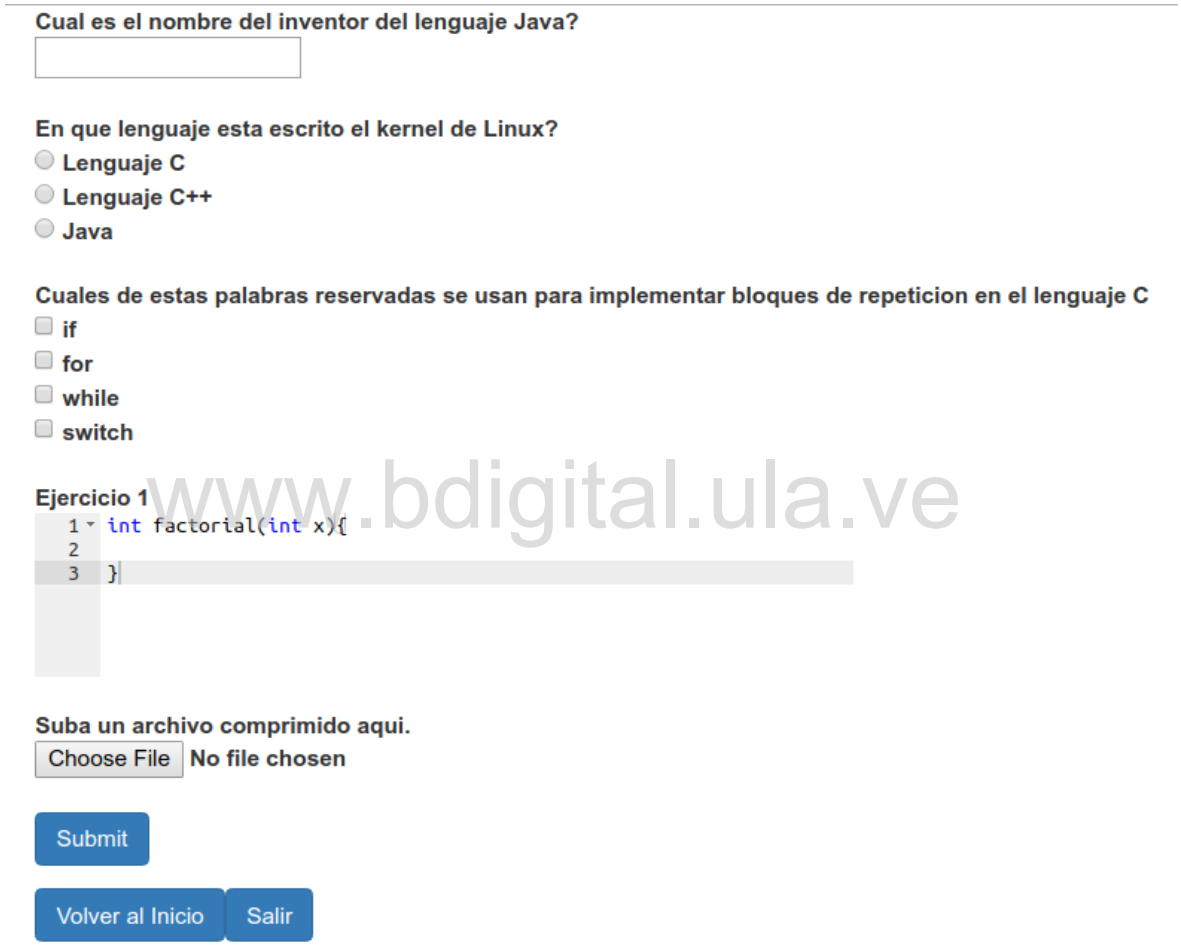
Figura 4.32: Traducción de representación gráfica de elemento input tipo "file"

```
▼<label>
   "Suba un archivo comprimido aqui."
  ▼<div>
     <input type="file" name="file_1" onChange=bound () />
   </div>
  </label>
```

En la siguiente captura de pantalla se puede apreciar cómo se mostraría a un usuario cada elemento de entrada que permite representar la API. Cada uno de los elementos que se muestran en la captura siguiente son los mismos elementos que se especificaron para mostrar el ejemplo de cada tipo de

entrada. Esto permite observar sus tres representaciones. La representación en JSON, la representación en código HTML y la representación gráfica que se le muestra al usuario en la interfaz.

Figura 4.33: Captura de pantalla de un form_action con todos los tipos de entradas posibles






## 4.5 Alojamiento de la interfaz desarrollada en el servidor web

Por último, se configura el servidor Apache para que pueda servir la interfaz desarrollada con Django y React. Esta configuración consiste en hacer que Apache

trabaje como interfaz con el modulo mod_wsgi, el cual traduce peticiones HTTP en un formato entendible por WSGI. En el paso "Configure Apache" de la guía [22] se detalla este proceso.

Esto concluye la implementación de la interfaz que cumple con los requisitos de este proyecto. En este capítulo se explicó de manera detallada la implementación de las partes más relevantes, tomando como base los capítulos anteriores en los cuales se planteó teóricamente la solución planteada para el problema afrontado y también la breve explicación de las herramientas que fueron utilizadas. Es relevante mencionar que, aunque la interfaz web está compuesta por varias partes, lo que la diferencia de una interfaz web común es la API que ésta contiene. Esta API permite afrontar el problema que presenta la falta de una estandarización entre *frameworks* especificando su propio conjunto de reglas, las cuales tienen como objetivo generalizar la interacción entre la interfaz y los *frameworks* manteniendo como prioridad limitar lo menos posible a los *frameworks* que deseen formar parte de la interacción. Específicamente el objetivo principal de este API es estandarizar la interacción entre las partes para no tener que estandarizar a los *frameworks*.

# Capítulo 5

# Conclusiones y Recomendaciones

## 5.1 Conclusiones

Se investigó sobre herramientas y protocolos web que cumplieran con los requerimientos de este proyecto. Estas herramientas fueron seleccionadas tomando en cuenta sus capacidades, su documentación y su compatibilidad al momento de trabajar juntas.



Se desarrolló una interfaz web implementando las herramientas seleccionadas previamente. Esta interfaz permite a usuarios registrarse en ella, acceder y utilizarla para interactuar con posibles *frameworks* a los que el usuario se encuentra relacionado. Esta interacción es coordinada por una API integrada en la interfaz web la cual define un conjunto de reglas las cuales permiten a *frameworks* distintos ofrecer acciones para ejecutar a sus usuarios por medio de la interfaz.

La interfaz fue desarrollada tomando en cuenta medidas de seguridad con HTTPS el cual ofrece conexión segura y evita que entes no autorizados puedan interpretar la información enviada por el usuario a la interfaz.

Se aplicaron pruebas de rendimiento y funcionalidad utilizando archivos JSON diseñados de acuerdo a las reglas establecidas por la API. Estos archivos simulan las respuestas enviadas por *frameworks* a la interfaz, debido a que estos *frameworks* todavía se encuentran en desarrollo. Aunque dichos *frameworks* todavía no interactúan con la interfaz, la API establece una estructura que permite una integración casi inmediata.

## 5.2 Recomendaciones

Se recomienda implementar autenticación de dos factores utilizando herramientas de autenticación biométrica para ofrecer un mayor nivel seguridad.

Aunque en este proyecto se implemento conexión segura con HTTPS, los certificados generados no son certificados para uso legal ya que son generados localmente. Los certificados utilizados comercialmente son certificados aprobados por una autoridad certificadora. Para implementar certificados autorizados sin costo se puede utilizar la Autoridad Certificadora Let's Encrypt [37]. Esta autoridad es un servicio ofrecido por el Internet Security Research Group que está siendo apoyada por grandes compañías como Mozilla, Cisco, Facebook, entre otras. Además de poder ser utilizada para este proyecto, puede ser utilizada para renovar certificados vencidos en los sitios web de la Universidad de Los Andes.

# Bibliografía


[1] W. W. W. Foundation, "History of the web," urlhttp://webfoundation.org/about/vision/history-of-the-web/, accedido 24-10-2017.

[2] M. P. D. Stuttard and J. Pauli, *The web application hacker's handbook.*, 2nd ed. Indianapolis, Indiana: Wiley Publishing, Inc., 2011.

[3] IBM, "Ibm cloud computing: ¿qué es cloud computing?" urlhttps://www.ibm.com/cloud-computing/mx/es/what-is-cloud-computing.html, accedido 24-10-2017.

[4] ——, "Ibm - iaas paas saas cloud service models," urlhttps://www.ibm.com/cloud-computing/learn-more/iaas-paas-saas/, accedido 24-10-2017.

[5] J. Kurose and K. Ross, *Computer networking: a top-down approach.*, 6th ed. Boston, MA: Pearson., 2013.

[6] W. W. W. Consortium, "Html and css," urlhttps://www.w3.org/standards/webdesign/htmlcss, accedido 24-10-2017.

[7] T. I. E. T. Force, "Hypertext transfer protocol," urlhttps://www.ietf.org/rfc/rfc2616.txt, 1999, accedido 24-10-2017.

[8] ——, "The tls protocol," urlhttps://www.ietf.org/rfc/rfc2246.txt, 1999, accedido 24-10-2017.

[9] ——, "Rfc 2818 - http over tls," urlhttps://tools.ietf.org/html/rfc2818, 2000, accedido 24-10-2017.

[10] J. R. Aguirre, *Libro Electrónico de Seguridad Informática y Criptografia*, 1st ed. Madrid: Departamento de Publicaciones de la Escuela Universitaria de Informa?tica de la UPM, 2006.

[11] A. Stolk, "Tecnicas de seguridad informatica con software libre," 2013, accedido 24-10-2017.

[12] M. Whitman and H. Mattord, *Principles of information security*, 4th ed. Boston, MA: Course Technology, 2012.

[13] T. I. E. T. Force, "Rfc 7159 - the javascript object notation (json) data interchange format," urlhttps://tools.ietf.org/html/rfc7159, 2014, accedido 24-10-2017.

[14] G. B. D. Jacobson and D. Woods, *APIs - A Strategy Guide*, 1st ed. Beijing: O'Reilly, 2012.

[15] "Edx," urlhttps://www.edx.org/, accedido 24-10-2017.

[16] "Coursera," urlhttps://www.coursera.org/, accedido 24-10-2017.



[17] "Futurelearn," urlhttps://www.futurelearn.com/, accedido 24-10-2017.

[18] "Hackerrank," urlhttps://www.hackerrank.com/, accedido 24-10-2017.

[19] "Codeassess," urlhttps://www.codeassess.com/, accedido 24-10-2017.

[20] T. A. S. Foundation, "Apache http server project," urlhttp://httpd.apache.org/, accedido 24-10-2017.

[21] ——, "Licenses of distribution," urlhttps://www.apache.org/licenses/, accedido 24-10-2017.

[22] J. Ellingwood, "How to serve django applications with apache and mod_wsgi on ubuntu 16.04," urlhttps://www.digitalocean.com/community/tutorials/how-to-serve-django-applications-with-apache-and-mod_wsgi-on-ubuntu-16-04, accedido 24-10-2017.

[23] O. S. Foundation, "Welcome to openssl," urlhttps://www.openssl.org/, accedido 24-10-2017.

[24] T. P. G. D. Group, "Postgresql: About," urlhttps://www.postgresql.org/about/, accedido 24-10-2017.

[25] ——, "The postgresql license," urlhttps://opensource.org/licenses/postgresql, accedido 24-10-2017.

[26] J. Ellingwood, "How to use postgresql with your django application on ubuntu 16.04," urlhttps://www.digitalocean.com/community/tutorials/how-to-use-postgresql-with-your-django-application-on-ubuntu-16-04, accedido 24-10-2017.

[27] T. D. S. Foundation, "Django, the web framework for perfectionists with deadlines," urlhttps://www.djangoproject.com/, accedido 24-10-2017.

[28] ——, "Django license," urlhttps://github.com/django/django/blob/master/LICENSE, accedido 24-10-2017.



[29] F. Inc., "React, a javascript library for building user interfaces," urlhttps://reactjs.org/, accedido 24-10-2017.

[30] ——, "React license," urlhttps://github.com/facebook/react/blob/master/LICENSE, accedido 24-10-2017.

[31] M. Anicas, "Initial server setup with ubuntu 16.04," urlhttps://www.digitalocean.com/community/tutorials/initial-server-setup-with-ubuntu-16-04, accedido 24-10-2017.

[32] B. Bearnes, "How to install linux, apache, mysql, php (lamp) stack on ubuntu 16.04," urlhttps://www.digitalocean.com/community/tutorials/how-to-install-linux-apache-mysql-php-lamp-stack-on-ubuntu-16-04, accedido 24-10-2017.

[33] J. Ellingwood, "How to create a self-signed ssl certificate for apache in ubuntu 16.04," urlhttps://www.digitalocean.com/community/tutorials/how-to-create-a-self-signed-ssl-certificate-for-apache-in-ubuntu-16-04, accedido 24-10-2017.

[34] P. S. Foundation, "Installing python modules," urlhttps://docs.python.org/3/installing/index.html, accedido 24-10-2017.

[35] I. Bicking, "Virtualenv," urlhttps://virtualenv.pypa.io/en/stable/, accedido 24-10-2017.

[36] M. Brochhaus, "Django reactjs boilerplate," urlhttps://github.com/mbrochh/django-reactjs-boilerplate, accedido 24-10-2017.

[37] "Let's encrypt - free ssl/tls certificates," urlhttps://letsencrypt.org/ , accedido 24-10-2017.

# Glosario

**URL (*Uniform Resource Locator*)** Es una dirección única que permite ubicar y localizar inequívocamente un recurso. Una dirección URL identifica un solo recurso universalmente.. 12, 14, 39, 54, 55, 57, 59

**DOM (*Document Object Model*)** API para documentos HTML y XML que proporciona una representación estructural del documento, sea HTML o XML. Este permite la modificación de una página web a través de sus elementos.. 30



**firewall** Software que se encarga de filtrar conexiones al sistema en que se encuentra instalado, basado en reglas especificadas en su configuración.. 33

**CA (*Certification Authority*)** Organización que se encarga de autorizar certificados criptográficos que se utilizan para ejercer una conexión HTTP sobre TLS. Estas autoridades le ofrecen una base de datos a los navegadores donde estos pueden verificar un certificado que ofrezca un sitio web y así comprobar su veracidad.. 33

**HTTP GET** Operación HTTP que permite obtener datos de un recurso específico. Para solicitar estos datos se utiliza una dirección URL.. 38, 54, 55

**HTTP POST** Operación HTTP que permite enviar datos acerca de un recurso específico. Esta operación también se envía a una dirección URL.. 38