ESCUELA POLITÉCNICA NACIONAL

ESCUELA DE FORMACIÓN DE TECNÓLOGOS

DESARROLLO DE *API REST* PARA AUTOMATIZAR PROCESO DE RECEPCIÓN, SEGUIMIENTO Y CALIFICACIÓN DE PROYECTOS DE TITULACIÓN-ESFOT.

TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE TECNÓLOGO EN ANÁLISIS DE SISTEMAS INFORMÁTICOS

CHANTAL ALEJANDRA MORALES ROJAS
chantal.morales@epn.edu.ec

JONATHAN ISRAEL VÁSQUEZ VIVAS
jonathan.vasquez01@epn.edu.ec

DIRECTOR: Ing. Edwin Gonzalo Salvador Pesantes, MSc.
edwin.salvador@epn.edu.ec

CODIRECTORA: Ing. Monica de Lourdes Vinueza Rhor, MSc.
monica.vinueza@epn.edu.ec

Quito, octubre 2021

CERTIFICACIÓN

Certificamos que el presente trabajo fue desarrollado por la Srta. Chantal Alejandra Morales Rojas y el Sr. Jonathan Israel Vásquez Vivas como requerimiento parcial a la obtención del título de TECNÓLOGO ANÁLISIS DE SISTEMAS INFORMÁTICOS, bajo nuestra supervisión:

Ing. Edwin Salvador MSc	**Ing. Mónica Vinueza MSc**
DIRECTOR DEL PROYECTO	CODIRECTORA DEL PROYECTO

DECLARACIÓN

Nosotros Morales Rojas Chantal Alejandra y Vásquez Vivas Jonathan Israel con CI: 1725211559 y 1723880512 respectivamente, declaramos bajo juramento que el trabajo aquí descrito es de nuestra autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y que hemos consultado las referencias bibliográficas que se incluyen en este documento.

Sin prejuicio de los derechos reconocidos en el primer párrafo del artículo 144 del Código Orgánico de la Economía Social de los Conocimientos, Creatividad e Innovación – COESC-, somos titulares de la obra en mención y otorgamos una licencia gratuita, intransferible y no exclusiva de uso con fines académicos a la Escuela Politécnica Nacional.

Entregamos toda la información técnica pertinente, en caso de que hubiese una explotación comercial de la obra por parte de la EPN, se negociará los porcentajes de los beneficios conforme lo establece la normativa nacional vigente

Chantal Alejandra Morales Rojas

Jonathan Israel Vásquez Vivas

DEDICATORIA

Quiero dedicar este proyecto, llena de felicidad, orgullo y amor, a las personas que de una u otra forma creyeron en mí y me vieron en este punto incluso antes de yo visualizarlo.

A mi hermana, mi compañera de vida que siempre me alentó, me motivo y hasta me presionó para cumplirlo, te amo. A la luz de mi vida, Dann que sin saberlo me acompañó durante este camino de principio a fin, noche y día.

A mi Papá y mi Abuelita, que lo dejaron ser, viendo cómo iba cumpliendo cada pequeño logro, cada esfuerzo, risa y lágrima, pero acá estoy y nunca han soltado mi mano.

A Janett, que quién sin saberlo, hasta este día, insertó en mí el amor por esta carrera y mi sueño de seguirla.

A mis Rayos, porque son una luz increíble que guía mi vida y me han ayudado a creer en mí y superar mis límites.

Finalmente, me la dedico a mí, por todos los minutos que pasé, desde que emprendí esta hermosa carrera, y por todo lo que me enseñé durante este trayecto.

Ustedes, han hecho esto posible sin siquiera saberlo, gracias.

MORALES ROJAS CHANTAL ALEJANDRA

AGRADECIMIENTO

A la vida, por permitirme vivir tantos momentos maravillosos en este camino, a la Escuela Politécnica Nacional que fue mi hogar estos años y tanto me enseñó, a mi familia por ser un pilar fundamental en todo esto. A todas las personas que aportaron en mi vida, me dieron ánimos, me apoyaron y me dieron un empujoncito cuando así lo sentían. A mis amigos, por cumplir con esto juntos y ser el mejor equipo que la vida me pudo poner para crecer como profesional y como persona, colegas, comenzaría mil proyectos más con ustedes. También a mis amigos que la vida me regaló de todas partes y hoy son como mi familia.

A mi director, Chalo, por todo lo que nos enseñó, ayudó y orientó en este proceso, a mis Profesores de la carrera por su aporte y su conocimiento para que hoy esto sea entregado.

Agradezco desde el fondo de mi corazón a todas las personas que fueron parte de mis días a través de esta aventura, sepan que tienen un lugar muy preciado en mi vida.

MORALES ROJAS CHANTAL ALEJANDRA

DEDICATORIA

Este trabajo te lo dedico a ti, Jonathan, que encontraste la fuerza de volverte a levantar pese a que arrastrabas ya varias caídas, que utilizaste tu tiempo en las aulas para nutrirte de conocimiento, pero también para rodearte de los mejores amigos y maestros, que siempre has sido un buen compañero, un buen alumno, un buen hijo y un gran ser humano.

Te agradezco por no haberte rendido, por haber tomado el mejor consejo, por dejarte guiar por los mejores y también por perseguir tus sueños. Nada de esto sería posible sin ti.

VÁSQUEZ VIVAS JONATHAN ISRAEL

AGRADECIMIENTO

Agradezco a la vida por haberme dado más de una oportunidad de intentarlo, por hacer que siempre encontrara la luz al final del túnel.

A mis padres por jamás perder la confianza en mí y porque siempre tuvieron una palabra de apoyo cuando más la necesitaba.

A mis hermanos Jorge y Alex, que fueron el soporte en el que me apoyé siempre.

A David y Gabi, porque sin su terquedad a dejarme ir, hoy esto no sería posible.

A mis profesores y en especial a Chalo, porque nos ha hecho y nos sigue haciendo mejores cada día.

A mis amigos, Nico, Sebas, Kevin, Esteban y Chanty, porque con ustedes el tiempo se pasó volando y no pude haber escogido mejores compañeros para esta aventura.

A Carolina, por amarme y acompañarme en mis éxitos y fracasos y por darle otro color a mi vida.

Finalmente, a la Escuela Politécnica Nacional y la Escuela de Formación de Tecnólogos por volverme un digno representante de tan prestigiosas instituciones.

VÁSQUEZ VIVAS JONATHAN ISRAEL

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN

El proceso de recepción seguimiento y calificación de proyectos de titulación en la ESFOT, siempre ha presentado algunas falencias que se pueden evidenciar en el largo procedimiento al que se enfrentan los estudiantes y las autoridades para que se complete este proceso, Además, en vista de la situación actual en la que se encuentra el mundo por el Covid-19, es necesario encontrar medidas que automaticen procesos como este y lo faciliten para todos los involucrados.

Tomando en cuenta los análisis realizados al proceso que se pretende automatizar, y en base a las directrices planteadas por la Universidad debido a la pandemia, se ha desarrollado una *API REST* que permite la interacción de los datos utilizadas junto con una plataforma para la automatización del proceso de recepción seguimiento y calificación de proyectos de titulación de la ESFOT.

Este proyecto está desarrollado con *Laravel* que a su vez permite interactuar con *PostgreSQL,* qué es el sistema gestor de base de datos, de una manera simple. Además, se utiliza una metodología ágil como *Scrum* que brinda beneficios en cuanto a la estructura, organización y desarrollo del proyecto. El uso de estas herramientas y metodologías hacen posible obtener un proyecto con calidad y planificación eficiente. En este documento se presenta el proceso del desarrollo junto con los resultados, también se menciona su despliegue a producción, así como la posibilidad de su consumo a través de cualquier cliente.

PALABRAS CLAVE: *Scrum*, *PostgreSQL, Laravel,* ESFOT*, API REST*.

ABSTRACT

The process of reception, monitoring and grading of degree projects at ESFOT, has always had some weaknesses that can be evidenced in the long process faced by students and authorities to complete this process. Also, due to the current situation in which we find ourselves globally by the Covid-19, it is necessary to find measures to automate processes like this and make it easier for all involved.

Considering the analyses made to this process, and based on the guidelines that the University raised due to the pandemic, a REST API has been developed to allow the interaction of the data that is used in conjunction with a platform for the automation of the process of receiving, monitoring and qualification of degree projects of the ESFOT.

This project is develop using Laravel, a framework that allows us to interact with PostgreSQL, the database management system, in a simple way. Furthermore, we use an agile methodology such as Scrum that gave us benefits in terms of structure, organization, and development of the system. The use of these tools and agile methodologies make it possible to obtain a project with quality and efficient planning. This document presents the development process and the of the project, the deployment to production is also mentioned, as well as the possibility of the consumption through any client.

KEYWORDS: *Scrum*, *PostgreSQL, Laravel,* ESFOT*, API REST*.

1 INTRODUCCIÓN

La Escuela de Formación de Tecnólogos (ESFOT) de la Escuela Politécnica Nacional (EPN), ha ofertado carreras en el ámbito tecnológico desde hace más de medio siglo. las ESFOT, que actualmente oferta cuatro carreras y está en proceso de aprobación de algunas más, evidencia algunas falencias en la ejecución del proceso de titulación. Se conoce que actualmente se gradúan 10 estudiantes aproximadamente en un semestre [1], es por eso por lo que se desea elevar esta cantidad de graduados. A través de la comisión de automatización de procesos, creada en el año 2019 mediante el memorando númeroo Nro. EPN-ESFOTDR-2019-1176-M. Se hace un análisis del proceso actual de la presentación del plan de planes, seguimiento y calificación de proyectos de titulación con el objetivo de mejorar el mismo.

El análisis presentado por la comisión [2] puso en evidencia diversas falencias en este proceso, que se mencionarán a continuación:

1. Problemas en la comunicación entre los tesistas y los directores, esto hace que exista una ausencia de seguimiento de avances en los proyectos.
2. Centralización del proceso en las autoridades y personal administrativo que desemboca en largos tiempos de respuesta a los tesistas.
3. Largos tiempos de espera en la corrección y devolución de los planes que la comisión revisa. Esto hace que el proceso de revisión se alargue y se pierda valioso tiempo. Además, se puede mencionar el desperdicio de papel que se evidencia entre una y otra versión del proyecto.

Por otro lado, teniendo en cuenta la situación actual en la que se encuentra el mundo y en este país que además ha enfrentado una emergencia sanitaria de la que aún quedan rezagos, se ha puesto en evidencia la necesidad de contar con un proceso digitalizado. Es por esto que la EPN ha creado las directrices para la realización de los procesos de graduación asociados a los trabajos de titulación y tesis mientras dure el estado de excepción lo cual permite que muchos de los documentos que antes se entregaban de forma física puedan ser entregados de forma digital y enviados mediante un medio electrónico [3]. Sin embargo, estas directrices no han solucionado los cuellos de botella y los tiempos de espera que afectan a este proceso más aún cuando mucha de la responsabilidad recae sobre las autoridades o personal administrativo.

Por todo lo anteriormente mencionado se ha desarrollado de manera importante y necesaria un sistema *web q*ue permite automatizar este proceso. El sistema ofrece

beneficios como son monitoreo de tiempos, registro de acciones, envío de notificaciones, registro de calificaciones y observaciones, entre otros. Además, el objetivo es que este sistema sea escalable y replicable por lo que el presente proyecto muestra la creación de una interfaz de programación de aplicaciones utilizando la transferencia de estado representacional o más conocido como *API REST* por sus siglas en inglés.

1.1 Objetivo general

Desarrollar una *API REST* para automatizar el proceso de recepción, seguimiento y calificación de proyectos de titulación de la ESFOT.

1.2 Objetivos específicos

- Levantar los requerimientos para el sistema.
- Diseñar la arquitectura y base de datos para el sistema.
- Programar el *API REST* de manera que pueda ser accedido por aplicaciones clientes independientes de la tecnología.
- Implementar pruebas unitarias del *API REST*.
- Desplegar el *API REST* en un servidor web público

1.3 Alcance

El uso de plataformas web dentro de distintos ámbitos en la actualidad ha ayudado a que muchos procesos se automaticen y se optimicen de una forma impresionante generando así un beneficio para los clientes o usuarios finales. Esto genera beneficios como la optimización del tiempo, comodidad e incluso un beneficio al impacto ambiental.

Teniendo en cuenta el contexto actual en el que se vive dentro de la pandemia y ante las recomendaciones del Ministerio de Salud Pública entre las cuales se promueve el uso de plataformas en las que se pueda automatizar procesos para evitar el contacto personal y hacerlo desde un ambiente seguro para el usuario [4].

El presente proyecto muestra la creación de un *API REST* para aprovechar las ventajas, entre las cuales se destaca la posibilidad de brindar a la interfaz una forma de trabajo independiente y un almacenamiento de datos mucho más simple. Esto brinda la posibilidad de ser transportado y replicado en otros sistemas. El cliente que acceda al *API REST* puede realizar peticiones para ver, guardar, editar y hasta eliminar los datos almacenados en las tablas de la BDD, también cuenta con roles de usuarios, registro de usuarios, así como la verificación y seguridad en las rutas para mantener controlado y seguro el acceso a la información.

El usuario con el Rol de Estudiante en el *API REST* puede:

- Acceder al inicio de sesión.
- Guardar un plan o proyecto de titulación.
- Editar un plan o proyecto de titulación.

- Ver las observaciones del plan y proyecto de titulación.

El usuario con el Rol de Profesor/director en el *API REST* puede:

- Acceder al inicio de sesión.
- Revisar un plan o proyecto de titulación (aplica para profesor, comisión y jurado).
- Editar un plan o proyecto de titulación.
- Subir las observaciones del plan y proyecto de titulación.
- Aprobar o rechazar proyectos.
- Subir ideas para proyectos de titulación.
- Guardar las calificaciones de los proyectos (aplica para jurado).

El usuario con el Rol de Secretaría en el *API REST* puede:

- Acceder al inicio de sesión.
- Guardar nuevos profesores.
- Guardar lista de estudiantes.
- Crear comisiones de las distintas carreras.
- Revisar y guardar el estado de la documentación de un estudiante.

El usuario con el Rol de Administrativo en el *API REST* puede:

- Acceder al inicio de sesión.
- Asignar un tribunal a un proyecto de titulación.
- Asignar fecha de defensa para un proyecto de titulación.
- Ver la lista de proyectos de titulación.

Esto se hace mediante rutas controladas para dicho acceso. Esto se replica a cada rol definido en el *API REST*, así cada usuario puede acceder y realizar cierto tipo de acciones definidos por el rol y las rutas.

El presente informe detalla el proceso de creación del *API REST* que se complementa con otro proyecto de titulación encargado de desarrollar la interfaz de usuarios para el consumo de los datos que se pretende ofrecer como parte del *Backend*, no ofrece la explicación de la creación y desarrollo de la parte del *frontend*.

2 METODOLOGÍA

Actualmente existen diferentes metodologías en el campo del desarrollo de *software*, las cuales permiten que el desarrollo de diferentes productos o servicios, el objetivo de las mismas es hacer que el trabajo del equipo de desarrollo sea lo óptimo y organizado posible. Es por eso que se determina que el desarrollo de este proyecto se lleva mediante una metodología ágil como lo es *Scrum.*

Scrum, es una metodología que se maneja mediante *sprints,* en los cuales se puede ir evidenciando el avance del proyecto y así ir completando los objetivos planteados en casa fase del desarrollo.

2.1 Metodología de Desarrollo

La metodología ágil *Scrum* agiliza el desarrollo de los sistemas ya que mantiene la participación y relación entre el cliente y el equipo de desarrollo. Se maneja a manera de *sprints* con una duración de aproximadamente un mes, manteniendo reuniones continuas entre el equipo, en las cuales se establecen los requerimientos e historias de usuario. Además, al terminar cada ciclo hay la posibilidad de acoplar los posibles cambios que puedan existir.

2.1.1 Roles

En *Scrum* se debe conformar un equipo, y determinar los roles y responsabilidades de cada miembro, procurando que la distribución del equipo se enfoque en sacar un *software d*e calidad y sobre todo que cumpla con los requerimientos del cliente. Es por eso que se deben asignar los siguientes roles de manera correcta para que esto a su vez desemboque en una correcta ejecución de parte del equipo de desarrollo.

Product Owner

Debe conocer y maximizar el valor del negocio, también se encarga de revisar el producto final al final de cada iteración, sugiriendo así cambios y aprobando las tareas implementadas ya que representa a las partes interesadas del proyecto. Este rol está representado por la ESFOT que necesita la automatización del proceso de titulaciónotorga la información al equipo.

Scrum Master

Se encarga de agilizar la comunicación entre el equipo y el *product owner*, también debe gestionar que el proceso de *Scrum* se desarrolle de la manera correcta, así como manejar posibles problemas que se vayan dando durante la implementación y así generar un producto con un verdadero valor [5]. En este caso el *Scrum Master* también es parte del equipo de desarrollo rol es representado por el Ing. Edwin Salvador.

Development Team

Se conforma usualmente de 3 a 9 miembros, que se encargan de proveer los conocimientos técnicos para la implementación del sistema, tienen como objetivo cumplir con cada tarea del *sprint* y brindar incrementos de valor de principio a fin de cada *sprint* [5].

En la **TABLA *I*** presenta la distribución del equipo *scrum*, conformado por 4 miembros con sus respectivos roles.

TABLA I: Roles del Proyecto

Rol	Integrante
Product Owner	ESFOT.
Scrum Master	Ing. Edwin Salvador MSc.
Development Team	Sr. Israel Vásquez.
	Sra. Chantal Morales.

2.1.2 Artefactos

El uso de artefactos dentro de la metodología *Scrum*, asegura el registro de información importante, así como la transparencia de esta, permitiendo establecer las bases dentro de un proyecto [5].

Historias de Usuario

Usualmente una historia de usuario se escribe mediante una tarjeta en la cual se explica de una manera informal y de manera general una función de *software* viéndolo desde la perspectiva del cliente [6].

Todas las funcionalidades que se van a implementar en la plataforma se describen en una historia de usuario. La **TABLA II** presenta la estructura de las historias de usuario que se han utilizado. El resto de las historias de usuario se pueden encontrar en la sección de **ANEXOS** apartado **6.1** .

TABLA II: Historia de usuario Nro. 4: Ver el plan de titulación

Historia de Usuario	
Identificador: HU04	**Usuario:** Estudiante
Nombre Historia: Ver el plan de titulación	
Prioridad en Negocio (Alta/Medio/Baja): Alta	**Riesgo en Desarrollo (Alta/Medio/Baja):** Alta
Iteración asignada: 2	
Responsable (es): Chantal Morales, Israel Vásquez	
Descripción: El sistema deberá enviar los datos almacenados del plan de titulación.	
Observación: El usuario debe haber guardado los datos del plan de titulación previamente y estar con la sesión activa.	

Product Backlog

Es una lista de todo lo que el producto o plataforma debe cumplir en base a las necesidades especificadas por el cliente. El encargado de su manipulación y organización es el *Product Owner,* mismo que crea y edita las tareas como esta lista va creciendo a medida que se van cumpliendo con los *sprints,* con esto se obtiene un producto mucho más útil y competitivo. En el apartado ***6.2*** de la sección de anexos se encuentra la figura del *Product Backlog*.

Sprint Backlog

Esta es una lista de todas las tareas que se deben entregar en cada iteración, para que un *sprint* se dé por finalizado se debe demostrar que estas tareas han sido cumplidas formando todas estas un entregable. El equipo de desarrollo es el principal actor en cuánto al desarrollo de estas tareas, el *Scrum Master* es el que decide cuáles tareas van a pasar del *Product Backlog* al *Sprint Backlog* en base a lo expresado por el *Product Owner*. En el apartado **6.3** de la sección de **ANEXOS** se encuentra la figura del *Sprint Backlog.*

2.2 Diseño de la arquitectura

La elección del patrón de arquitectura es importante ya que así se determina cuál es la opción eficaz para un problema planteado. A continuación, se describe el modelo, la arquitectura que se utiliza en el desarrollo de la plataforma.

2.2.1 Patrón arquitectónico Modelo Vista Controlador (MVC)

Este proyecto cuenta con una arquitectura MVC, se ha seleccionado este tipo de patrón ya que así se puede manejar el sistema por capas las cuales permiten separar el funcionamiento en diferentes interfaces que son [7]:

Modelo: el trabajo con los datos usados por el sistema y la lógica en cómo funcionan los mismos.

Vista: es la capa visual que se presenta al usuario, también conocida como interfaz de usuario, aquí se muestra toda la información que se envía, así como la interacción de los datos.

Controlador: capa que nos sirve como enlace entre la vista y el modelo desde el flujo de información entre ambas capas.

La **Fig. 1** muestra la capa del modelo y el controlador, así como las herramientas que se utilizan y permiten el desarrollo de esta estructura.

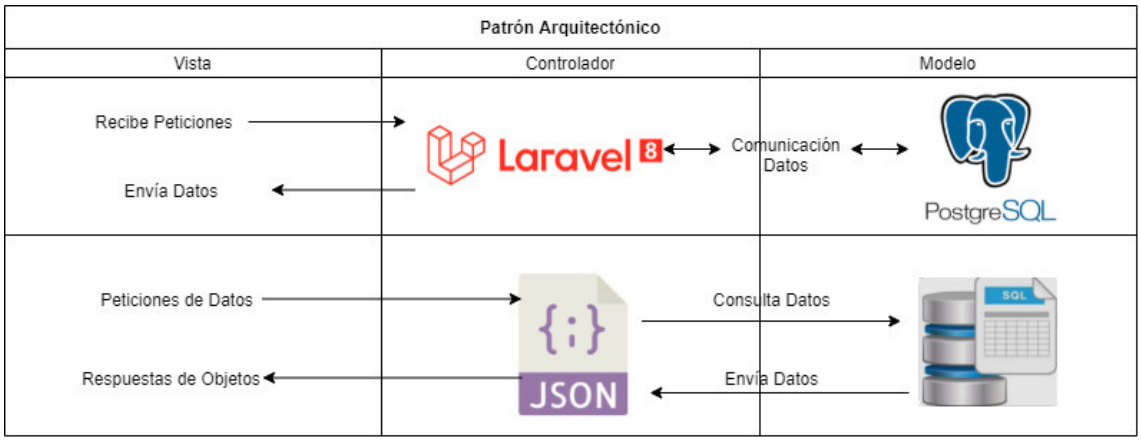


Fig. 1: Patrón arquitectónico de la *API REST*

2.3 Diseño de la base de datos

Laravel utiliza Eloquent ORM (Object Relational Mapping) el cual es un modelo de programación que transforma la interacción con la base de datos a través de un formato común y así evitamos el uso de consultas SQL, esto permite manejar la información y los datos a través de instancias y modelos [8].

Un *ORM* maneja la base de datos de una manera mucho más simple, e estableciendo una correspondencia a una tabla con un modelo, así interactúa con el modelo a través de instancias definidas por clases y funciones [9]. Para este proyecto se utilizan distintos comandos que brinda *Laravel* a través de *Eloquent* mismos que ayudan a crear las

distintas tablas que conforman la base de datos, así mismo se crean las relaciones entre las mismas.

2.4 Herramientas de desarrollo

Les herramientas que se utilizan se definieron en base a la arquitectura del proyecto. La **TABLA III** muestra dichas herramientas que son utilizadas para el desarrollo del sistema.

TABLA III: Herramientas para el desarrollo de la *API REST*

Herramienta		Justificación
Laravel	Es un *framework* de código abierto para construir aplicaciones web con PHP de manera simple. Cuenta con diferentes características como como el uso de *eloquent, routing*, middlewares, entre otros esto permite crear componentes de una forma más fácil y como una estructura más definida [10].	El uso de *laravel* permite controlar la *API* de manera sencilla y maneja los datos del lado del servidor de una forma óptima además que al presentar los datos en formato JSON hace que las consultas sean más eficientes.
PostgreSQL	Es un sistema de gestión de bases de datos relacionales, de código abierto, es uno de los gestores de datos más potente del mercado con gran accesibilidad y multiplataforma [11].	El uso de esta base de datos permite la consulta y almacenamiento de los datos de forma sencilla, ya que el manejo de sus relaciones se puede hacer a través de *Eloquent* con *Laravel*.
Composer	Es un gestor de dependencias de *PHP* que permite gestionar los paquetes de *software* de un proyecto, de modo que se encarga de mantener actualizadas todas estas dependencias [12].	Trabaja juntamente con *laravel* y su tarea principal es mantener actualizadas las principales dependencias que utiliza este *framework* para el desarrollo del proyecto.
Postman	Herramienta que se utiliza para realizar pruebas a una *API REST* mediante peticiones *http*, esto	Esta herramienta permite probar que los *endpoints* funcionen de manera correcta ya que a través

	permite crear pruebas, simulaciones y monitoreos. [13].	de esta se puede comprobar cómo llegan los datos y como se envían y almacenan a través de la *API*.
XAMPP	Es una distribución de Apache que contiene varios *softwares* libres como lo son el servidor web Apache, el sistema de administración de *BBDD MariaDB* y *MySQL* y los lenguajes de programación *Pearl* y *PHP* [14].	Con esta herramienta se puede probar la base de datos de manera local, así como mantener levantado el servidor de la *API* y probar su conexión con el *frontend*.
MySQL	Es un sistema de gestión de bases de datos relacional, cuenta con una versión de código abierto y otra versión comercial. Utiliza tablas múltiples que se interconectan y almacenan la información de forma organizada [15].	Este sistema gestor de base de datos permite gestionar la base de datos en el ambiente de desarrollo y de manera local, y así verificar que el almacenamiento de los datos y relaciones entre tablas funcionen de manera correcta.
Heroku	Es una plataforma como servicio (*PaaS*) que es basada en contenedores y en la nube. Se puede utilizar para implementar, administrar y alojar aplicaciones. Brinda una estructura de la aplicación controlada [16].	El uso de *Heroku* permite desplegar a la *API REST* a un ambiente de producción, a su vez hace posible crear un servidor en *PostgreSQL* y controlar la estructura del sistema web.
Git	Es un sistema de control de versiones de distribución libre y de código abierto, que permite manejar grandes y pequeños proyectos con rapidez [17].	Esta herramienta permite controlar las distintas versiones del proyecto, realizar cambios de manera aislada sin modificar el proyecto base y subir versiones actualizadas.
GitHub	Es una plataforma de desarrollo de *software* colaborativo, que permite el alojamiento de proyectos y	*GitHub* es esencial en el desarrollo del proyecto ahí se encuentra el repositorio de este

	repositorios, mejorando el trabajo en equipo a través de la gestión y control de versiones del código [18].	proyecto, además permite visualizar las diferentes versiones y controlar las solicitudes de cambios.
ZenHub	Es una plataforma usada para la gestión de proyectos, que a su vez está integrada con *GitHub*, se pueden conectar varios repositorios a un panel de tareas mediante el cual se puede tener un control y clasificación de estas [19].	A través de esta plataforma se controla y se manejan las distintas tareas que han sido creadas para el desarrollo del proyecto, también se vincula con el repositorio para mantener un control y gestión en las distintas versiones que se suben del proyecto.

3 RESULTADOS Y DISCUSIÓN

En esta sección, se encuentra un resumen detallado de las tareas que componen cada uno de los 5 *Sprints,* además, ilustraciones del armado y pruebas de la *API*.

3.1 *Sprint* 0. Planificación de proyecto

Se prepara un *Sprint* de planificación previa al inicio del desarrollo en el que se logra establecer las bases del proyecto, levantando todos los requerimientos para el sistema, tanto de parte de la dirección como de los estudiantes que usan el sistema. De esta forma se realiza una correcta planificación de las etapas de trabajo.

Los objetivos en este *Sprin*t son:

- Levantamiento de requerimientos.
- Planificación de tareas y funciones de los miembros para el desarrollo.
- Creación del proyecto de Laravel.

Además, se utiliza una herramienta llamada Miro, que es una pizarra colaborativa en la que se ofrecen múltiples platillas que se pueden seguir para desarrollar diagramas, mapas mentales y flujos de trabajo [20].

Para poder establecer un flujo de trabajo en el sistema es necesario conocer el flujo de titulación en la ESFOT es por esto que se utiliza Miro para crear un diagrama de flujo, que se muestra en la **Fig. 2** para tener claro todos los actores del proceso y poder darles un rol y acciones que deben desempeñar.

Fig. 2: Diagrama de Flujo Proceso para el Sistema de Titulación

A través de esta herramienta se diseña un *SiteMap*, mismo que ayuda a establecer el flujo del sistema y las acciones que se deben hacer por cada rol, así se diseña el flujo de trabajo de manera grupal y colaborativa entre todos los miembros del equipo de desarrollo. El *SiteMap* cuenta con distintas tarjetas que se diferencias por colores, mismos que representan un rol de usuario como se muestra en la **Fig. 3**

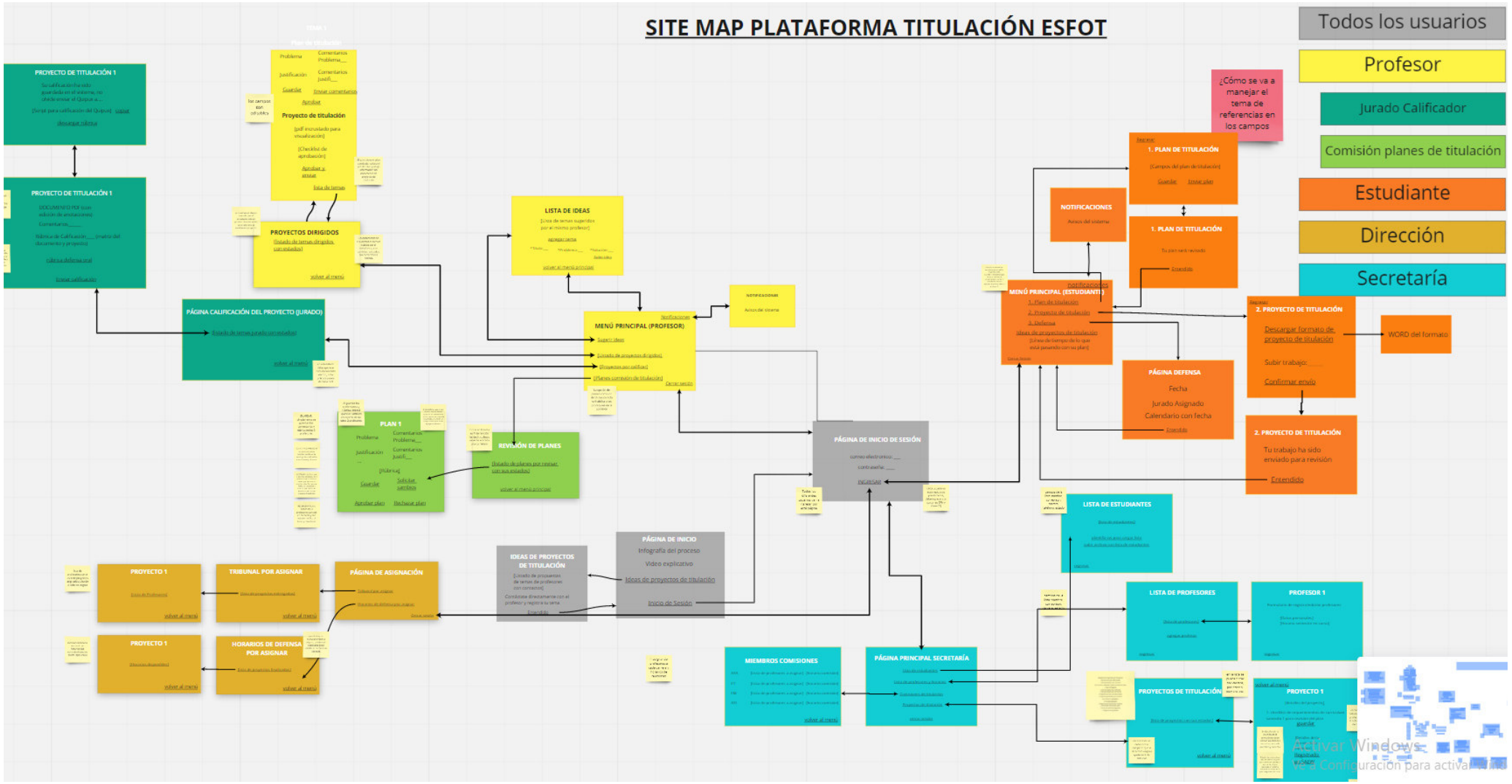


Fig. 3: *SiteMap* Sistema de Titulación

3.1.1 Levantamiento de requerimientos.

En el levantamiento de requerimientos de la *API REST* para el consumo en un sistema web para el proceso de titulación en la ESFOT, se organizan reuniones con representantes de la dirección de la ESFOT con el objetivo de comprender todo el flujo que se sigue en el proceso de titulación. Aquí, se establecieron actores, funcionalidades, requisitos y etapas desde la inscripción de un plan de tesis hasta la posterior defensa de grado.

3.1.2 Planificación de tareas y funciones de los miembros para el desarrollo.

Para el armado del cronograma de trabajo, se establen las etapas que deben ser cumplidas por el equipo de desarrollo y las tareas específicas que son trabajadas de forma individual y conjunta. Aquí también se detalla el tiempo y dificultad que representaba cada tarea y en función de estos valores, ajustar al tiempo planteado al inicio del desarrollo del proyecto de titulación.

3.1.3 Creación del proyecto en *Laravel*.

Para la configuración de la *API* y los componentes para su funcionamiento, se procede a instalar *PHP Storm*, *Composer*, *Postman* y *XAMPP*. Además, siguiendo los parámetros establecidos por *Laravel* y el patrón de arquitectura se procede a la creación del proyecto, que deja una estructura como se ilustra en la **Fig.** 4.

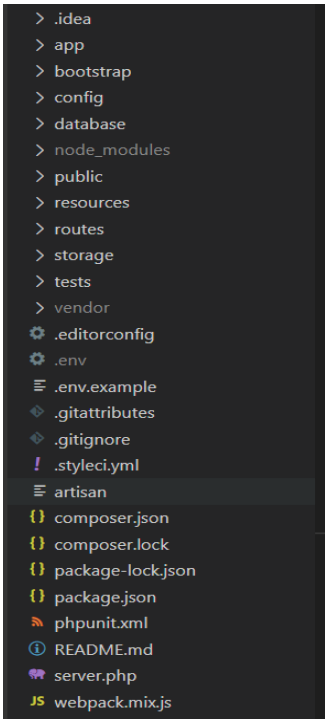


Fig. 4: Estructura del proyecto de *Laravel*

3.2 *Sprint* 1. Diseño de arquitectura y base de datos para el sistema

En función de la planificación realizada en el *Sprint* anterior y las tareas del *Sprint Backlog*, se determinan las tareas de armado arquitectónico de la *API* y cómo es su flujo de ejecución.

Quedando definidas las siguientes actividades:

- Diagrama de flujo
- Modelo de base de datos
- Estructura del repositorio
- Definición de arquitectura a ser utilizada para la *API*.

3.2.1 Diagrama de Flujo

A continuación, en la **Fig.** 5 se puede observar la forma de interacción que tiene el *FrontEnd* para el consumo del *API*. En este diagrama se puede destacar cómo varia la interacción y los datos manejados de acuerdo con el usuario que se encuentra activo. De esta forma la *API* se adapta a los requerimientos que pueda llegar a tener cada tipo de usuario y entrega o recibe la información pertinente.

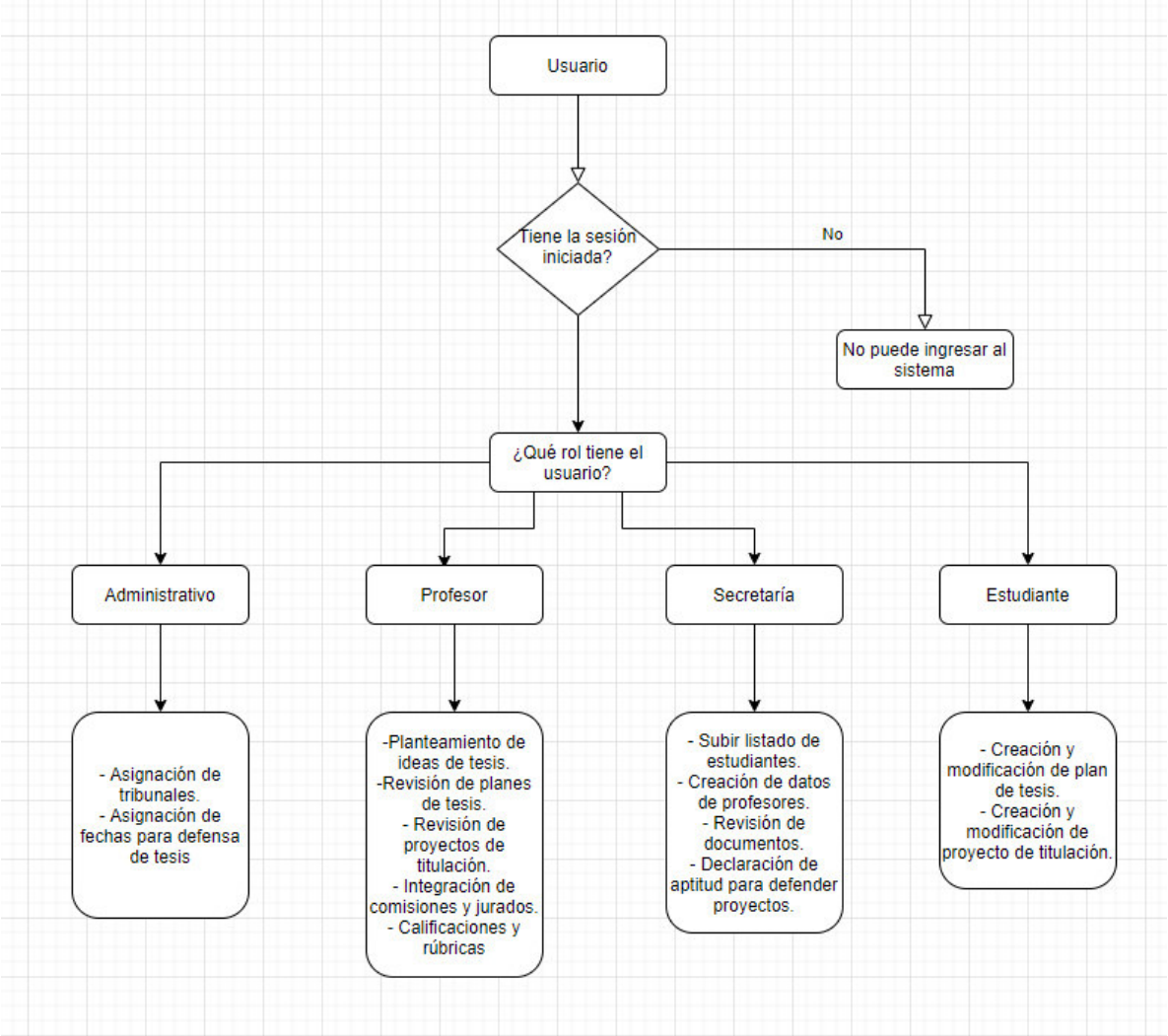


Fig. 5: Diagrama de flujo

3.2.2 Modelo de base de datos.

Para el armado de la base de datos se utiliza el Sistema Gestor de Base de Datos MySQL, debido a que cuenta con todas las herramientas solicitadas y cumple con todos los parámetros de seguridad e integración necesarios para el sistema [21]. La importancia de la base de datos es esencial ya que aquí se puede almacenar la información que luego responde a las distintas consultas realizadas a la *API*, además de la posibilidad de administrar la información de una forma fácil a través de su interfaz gráfica. Para su creación se toma como base el siguiente modelo entidad – relación que se muestra en la **Fig. 6** además del diseño de claves y relaciones. Para el diseño del modelo MER se utilizó la herramienta *MySQL Workbench*.

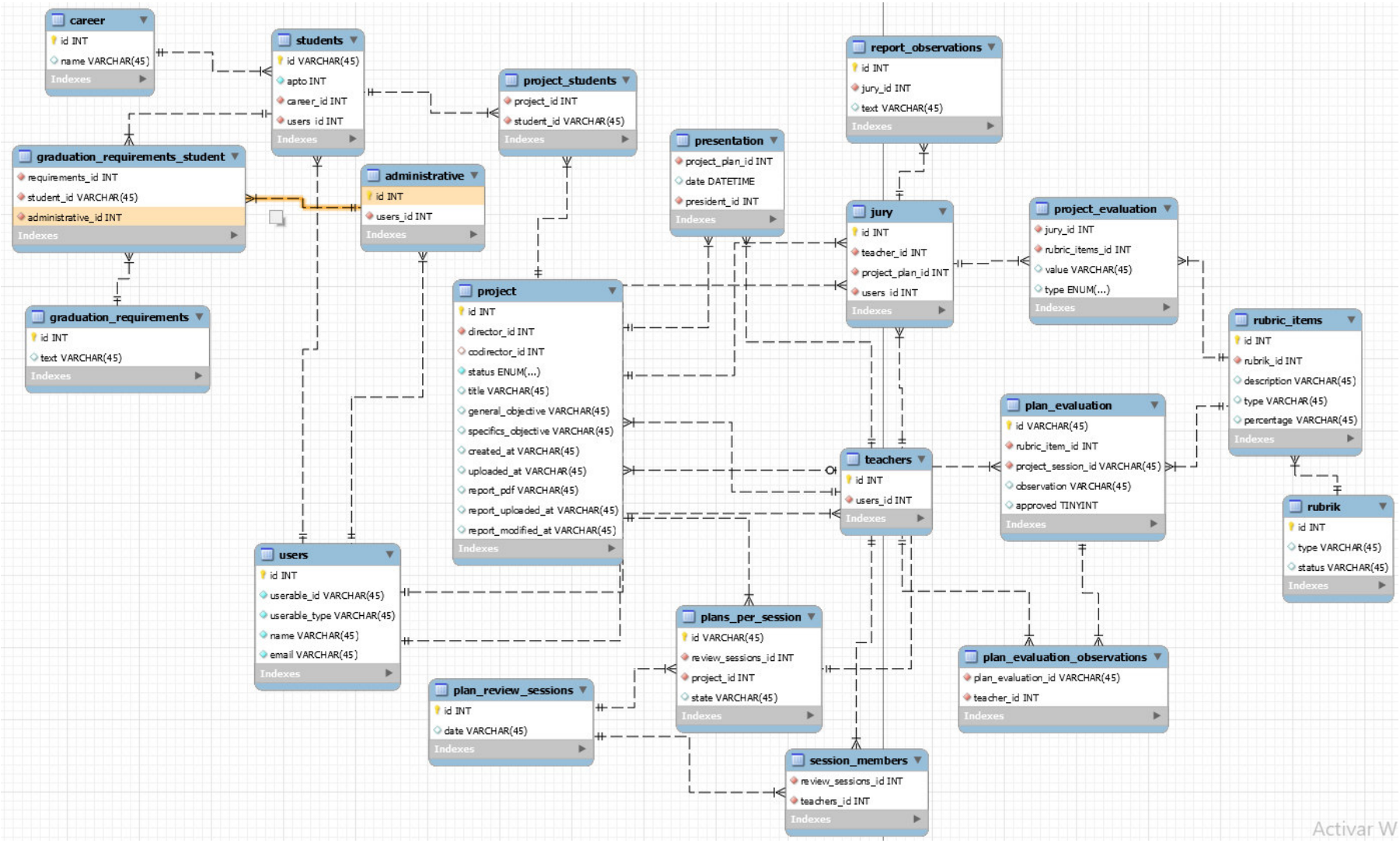


Fig. 6: Diagrama MER

3.2.3 Estructura del repositorio.

Git y GitHub son herramientas que permiten el trabajo en conjunto de un equipo de desarrollo, realizando un control progresivo de todos los aspectos referentes a proyectos grandes y pequeños. Entre las ventajas que poseen estas herramientas está un motor de trabajo muy poderoso que se complementa de una interfaz amigable, permitiendo un rápido aprendizaje de todas sus herramientas [22]. A través de *Git* se realiza la creación de un repositorio que contiene el proyecto base de desarrollo, donde se puede trabajar con ramas pertenecientes a cada tarea planificada. En *GitHub* se puede realizar la revisión de *Pull Requests*, que significa la combinación de las ramas secundarias con la rama principal donde se encuentra el proyecto base, manteniendo la integridad total del sistema.

Además, se utiliza la extensión *ZenHub* para la creación de un tablero con todas las tareas necesarias para el desarrollo del sistema, como se observa en la **Fig.** 7. Aquí, se puede controlar y categorizar las tareas para que su ejecución fuera realizada de forma ordenada y acorde a la planificación.

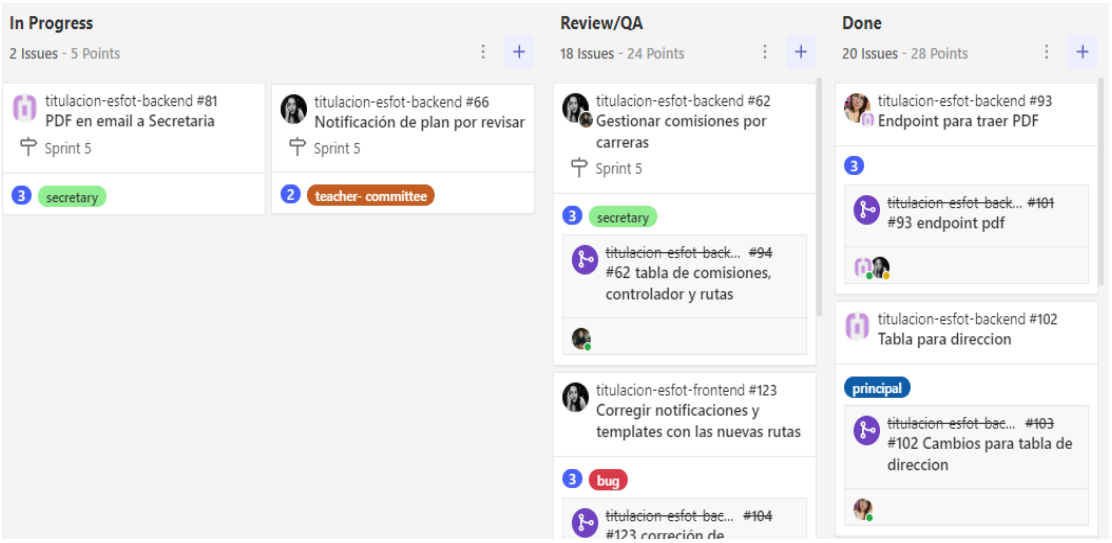


Fig. 7: Tablero de organización de tareas

3.2.4 Arquitectura utilizada para la *API*.

En esta sección se establece cómo es el flujo de la información, de acuerdo con el tipo de petición realizada. Los aspectos fundamentales que se cumplen en la arquitectura son las llamadas HTTP realizadas desde un cliente hacia la base de datos, pasando a

través de la *API*, encargada de resolver estas solicitudes en el formato solicitado en ambos sentidos. La *API* considera una serie de características para controlar la entrega de la información, siendo el tipo de usuario uno de los factores más importantes para poder realizar o no alguna acción específica [23]. En la **Fig. 8** se muestra cómo interactúan las peticiones de los clientes con la *API* y las respuestas que puede recibir de los servicios y base de datos.

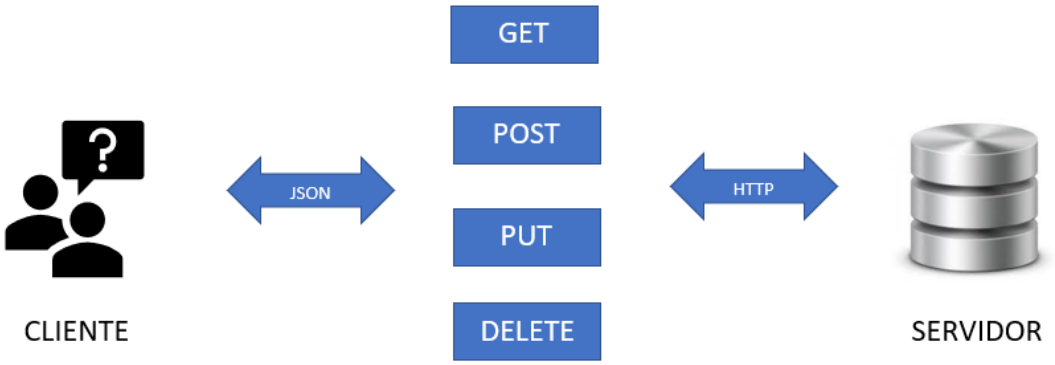


Fig. 8: Arquitectura de interacción de la *API*

3.3 *Sprint* 2. Desarrollo de los módulos de Estudiante y Profesor.

En base a la planificación del *Sprint Backlog*, en el *Sprint 2*, se realizan tareas comprendidas en los módulos para estudiante y profesor, abarcando desde el inicio de sesión hasta la subida de proyectos con la opción a emitir comentarios.

Como resultado de este *Sprint* se tiene:

- Inicio de sesión para todos los tipos de usuarios.
- Creación de ideas de proyecto por parte del profesor y visualización por parte del estudiante.
- Creación de plan de titulación por parte del estudiante y revisión del plan para su aprobación por parte del profesor.
- Subida de PDF del proyecto por parte del estudiante para su posterior revisión.
- Revisión y aprobación del proyecto por parte del profesor.
- Notificaciones.

3.3.1 Inicio de sesión para todos los tipos de usuarios.

La autenticación en la *API* se la realiza a través de *JSON Web Token (JWT)*, que es un estándar de tipo abierto que se basa en la arquitectura *JSON* para la creación de *Tokens* temporales que permiten el envío de información entre distintas aplicaciones en forma de un objeto *JSON*, asegurando que estos datos en cuestión sean seguros gracias a que se pueden verificar y determinar su firma digital. Los *JWT* son formados usando algoritmos de encriptación, haciéndolos muy seguros y útiles para la integración en las solicitudes que los usuarios puedan hacer a la *API*, ya que éstas deben contener en su cabecera un *JWT* válido, caso contrario, son rechazadas.

La generación del *JWT* se la realiza en el inicio de sesión, para lo cual se debe proporcionar un usuario y contraseña válidos, estos datos se guardan en el *token* y se mantienen durante todo el tiempo de validez de éste. Es importante que no se incluyan datos secretos en el *Token*, ya que este se encuentra expuesto a otros usuarios, aunque no puedan ser modificados.

Es importante tener en cuenta a las rutas que deben ser verificadas con *JWT*, ya que todas las que se encuentren fuera de su control pueden ser accedidas sin la necesidad de un rol o inicio de sesión previo. A continuación, en la **Fig. 9** se visualizan las rutas que se encuentran fuera de la verificación de *JWT* y otras que si deben contener el *token* en su cabecera.

```php
Route::post('register', [UserController::class, 'register']);
Route::post('login', [UserController::class, 'authenticate']);
Route::get('projects/{project}/cronogram', [ProjectController::class, 'cronogram']);
Route::get('teachers-ideas', [TeacherPlanController::class, 'index']);

Route::group(['middleware' => ['jwt.verify']], function () {
    //users
    Route::get('user', [UserController::class, 'getAuthenticatedUser']);
    Route::post('logout', [UserController::class, 'logout']);
```

Fig. 9: Rutas con JWT

Otro aspecto importante en la autenticación son los Roles de los usuarios, ya que estos determinan a que información puede o no acceder un usuario en concreto. Estos Roles son declarados dentro del modelo de usuario y utilizados más adelante. En la **Fig. 10** se observan los diferentes roles presentes en la *API*:

```
const ROLE_SUPERADMIN = 'ROLE_SUPERADMIN';
const ROLE_ADMIN = 'ROLE_ADMIN';
const ROLE_STUDENT = 'ROLE_STUDENT';
const ROLE_TEACHER = 'ROLE_TEACHER';
const ROLE_SECRETARY = 'ROLE_SECRETARY';

private const ROLE_HIERARCHY = [
    self::ROLE_SUPERADMIN => [self::ROLE_ADMIN],
    self::ROLE_ADMIN => [self::ROLE_STUDENT, self::ROLE_TEACHER],
    self::ROLE_TEACHER => [],
    self::ROLE_STUDENT => [],
    self::ROLE_SECRETARY => []
];
```

Fig. 10: Roles de usuario

En las políticas de la *API* se encuentran especificadas las acciones permitidas para cada rol de usuario, siendo importante que se tengan en cuenta todos los posibles casos de solicitudes que se puedan realizar y si ese usuario en particular está autorizado o no.

El proceso de autenticación es el mismo para todos los tipos de usuario, la única diferencia es la información a la que éstos tienen acceso.

3.3.2 Creación de ideas de proyecto por parte del profesor y visualización por parte del estudiante.

El profesor tiene la posibilidad de subir ideas de proyectos de titulación que puedan ser tomadas por los estudiantes y llevadas a cabo junto con el profesor que las postula. Estas ideas se componen de un título, una problemática y una solución, que deben ir acorde con la carrera a la que los estudiantes pertenecen y deben darles la posibilidad de aplicar sus conocimientos en la realización del proyecto.

Para el guardado de estas ideas se crea una ruta tipo *POST*, a la que se le pasa como datos las secciones antes mencionadas. En esta solicitud también debe incluirse el token de verificación que genera el profesor en su inicio de sesión y estas ideas están relacionadas con el profesor postulante mediante su id.

Se crea un modelo en el que se declaran los valores de la idea y también se le asigna un estado inicial de no asignado, el cual cambia en el momento que un estudiante solicite que se le asigne ese proyecto.

Es necesario, además, crear la migración para que se genere la tabla correspondiente en la base de datos y ahí se puedan realizar las operaciones de escritura y consulta de estas ideas.

Finalmente, en el controlador de ideas de profesor, se establecen las funciones de lectura y escritura para que puedan ser realizadas a través de las rutas correspondientes.

En el caso del estudiante, este puede consultar desde su rol, las ideas que se hayan propuesto y el estado de cada una, todo a través del método *GET* que se observa en la **Fig. 11.**

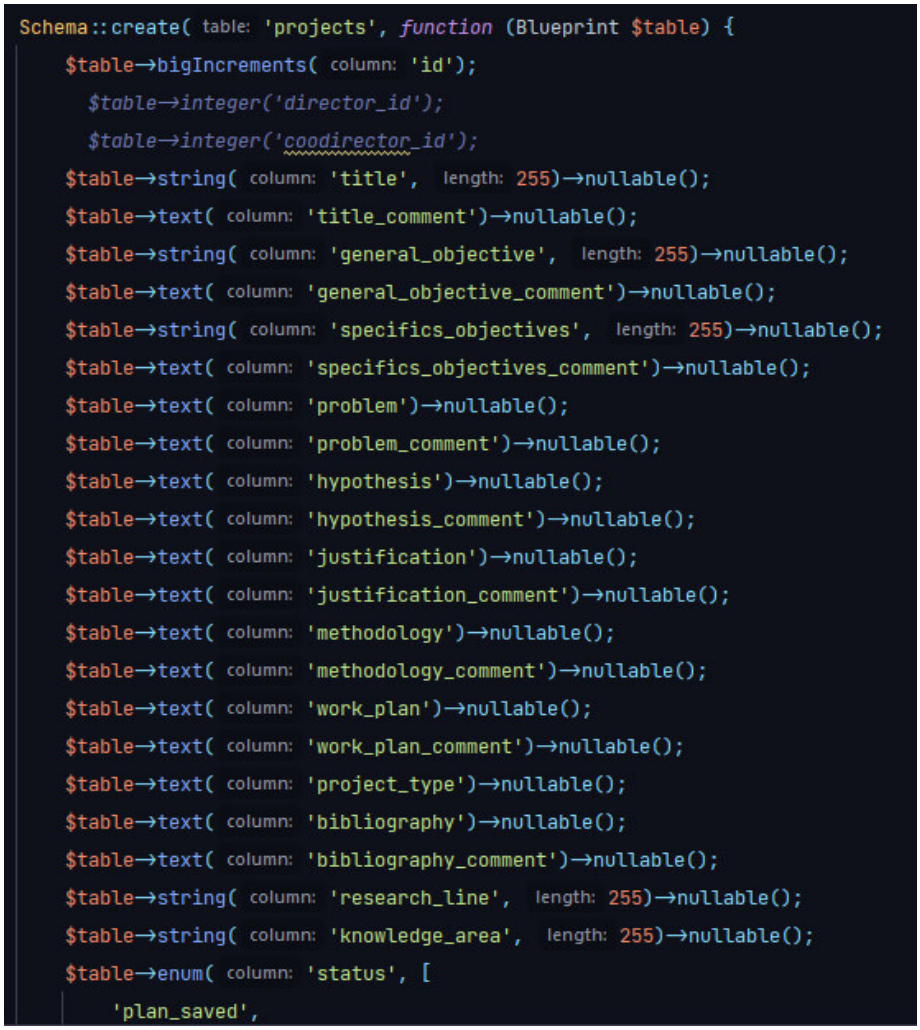
```
public function show(TeacherPlan $teacherplan)
{
    return response()->json(new TeacherPlanResource($teacherplan),  status: 200);
}
```

Fig. 11: Método *GET* de Ideas del profesor

3.3.3 Creación de plan de titulación por parte del estudiante y revisión del plan para su aprobación por parte del profesor.

El estudiante puede llenar el formato de plan de titulación y guardarlo en la base de datos a través de la ruta especificada para esta función. El proceso es similar a cualquier método *POST* y debe constar de los parámetros obligatorios y el *token* de verificación para que sea almacenado correctamente. Este proyecto tiene varios campos con particularidades, como el cronograma que es una imagen que se guardará en la base de datos o los estados que irán cambiando de acuerdo con el progreso del estudiante en las distintas etapas. Se crea el modelo para la especificación de los campos para la tabla y las relaciones que tienen las llaves foráneas con las tablas de estudiante y profesor. Un estudiante posee un proyecto de titulación y en un proyecto pueden estar involucrados más de un estudiante, por lo que se definen los parámetros para poder añadir un compañero de proyecto en caso de ser necesario.

Además, se crea la migración para generar la tabla en la base de datos con la especificación de cada columna y el tipo de dato que allí se guarda, tal y como se muestra en la **Fig. 12.**



```
Schema::create( table: 'projects', function (Blueprint $table) {
    $table->bigIncrements( column: 'id');
      $table->integer('director_id');
      $table->integer('coodirector_id');
    $table->string( column: 'title',  length: 255)->nullable();
    $table->text( column: 'title_comment')->nullable();
    $table->string( column: 'general_objective',  length: 255)->nullable();
    $table->text( column: 'general_objective_comment')->nullable();
    $table->string( column: 'specifics_objectives',  length: 255)->nullable();
    $table->text( column: 'specifics_objectives_comment')->nullable();
    $table->text( column: 'problem')->nullable();
    $table->text( column: 'problem_comment')->nullable();
    $table->text( column: 'hypothesis')->nullable();
    $table->text( column: 'hypothesis_comment')->nullable();
    $table->text( column: 'justification')->nullable();
    $table->text( column: 'justification_comment')->nullable();
    $table->text( column: 'methodology')->nullable();
    $table->text( column: 'methodology_comment')->nullable();
    $table->text( column: 'work_plan')->nullable();
    $table->text( column: 'work_plan_comment')->nullable();
    $table->text( column: 'project_type')->nullable();
    $table->text( column: 'bibliography')->nullable();
    $table->text( column: 'bibliography_comment')->nullable();
    $table->string( column: 'research_line',  length: 255)->nullable();
    $table->string( column: 'knowledge_area',  length: 255)->nullable();
    $table->enum( column: 'status', [
        'plan_saved',
```

Fig. 12: Migración de tabla Proyectos

Se desarrollan también las funciones correspondientes para el guardado, actualización y lectura de los datos del proyecto dentro del controlador. Finalmente, se especifican las rutas correspondientes para cada método, como se observa en la **Fig. 13.**

```php
//project
    Route::get('projects', [ProjectController::class, 'index']);
    Route::get('projects/{project}', [ProjectController::class, 'show']);
    Route::get('project/getPDF/{project}', [ProjectController::class,'getProjectPDFFile']);
    Route::post('students/projects', [ProjectController::class, 'store']);
    Route::post('projects/{project}', [ProjectController::class, 'update']);
    Route::post('projects/{project}/plan-sent',[ProjectController::class, 'planSent']);
    Route::post('projects/{project}/pdf', [ProjectController::class, 'updatePdf']);
    Route::delete('projects/{project}', [ProjectController::class, 'delete']);
    Route::post('projects/{project}/plan-review-teacher',[ProjectController::class, 'planReviewTeacher']);
    Route::post('projects/{project}/plan-corrections-done',[ProjectController::class, 'planCorrectionsDone']);
    Route::post('projects/{project}/plan-approved-director',[ProjectController::class, 'planApprovedDirector']);
    Route::post('projects/{project}/san-curriculum-1',[ProjectController::class, 'sanCurriculum1']);
    Route::post('projects/{project}/plan-review-commission',[ProjectController::class, 'planReviewCommission']);
    Route::post('projects/{project}/plan-corrections-done-2',[ProjectController::class, 'planCorrectionsDone2']);
    Route::post('projects/{project}/plan-approved-commission',[ProjectController::class, 'planApprovedCommission']);
    Route::post('projects/{project}/plan-rejected',[ProjectController::class, 'planRejected']);
    Route::post('projects/{project}/project-uploaded',[ProjectController::class, 'projectUploaded']);
    Route::post('projects/{project}/project-review-teacher',[ProjectController::class, 'projectReviewTeacher']);
    Route::post('projects/{project}/project-corrections-done',[ProjectController::class, 'projectCorrectionsDone']);
    Route::post('projects/{project}/project-approved-director',[ProjectController::class, 'projectApprovedDirector']);
    Route::post('projects/{project}/san-curriculum-2',[ProjectController::class, 'sanCurriculum2']);
    Route::post('projects/{project}/test-defense-apt',[ProjectController::class, 'testDefenseApt']);
    Route::post('projects/{project}/tribunal-assigned',[ProjectController::class, 'tribunalAssigned']);
    Route::post('projects/{project}/date-defense-assigned',[ProjectController::class, 'dateDefenseAssigned']);
    Route::post('projects/{project}/project-graded',[ProjectController::class, 'projectGraded']);
```

Fig. 13: Rutas de Proyecto

Los cambios de estado son una parte importante de la sección de proyecto, ya que esta define la etapa en la que se encuentra el estudiante en su proceso de titulación. Inicialmente, el estudiante sube su plan de titulación y está sujeto a los cambios que se realicen por parte del director del proyecto y con cada interacción y completado de requisitos se accede a la ruta correspondiente para realizar el cambio de estado. El profesor o director del proyecto, puede consultar los datos de un proyecto en el que éste cumpla el rol de director y si éste se encuentra en un estado de pendiente de revisión, puede añadir los comentarios necesarios, los cuales son guardados en su columna individual en la base de datos. En caso de que el proyecto cumpla con todos los parámetros requeridos, el profesor puede aprobar éste y así cambiar su estado para que el estudiante pueda continuar con su proceso.

3.3.4 Subida del PDF del proyecto por parte del estudiante para su posterior revisión.

En esta sección, el estudiante puede realizar la subida del archivo PDF que contenga todas las secciones de su proyecto de titulación y que a continuación recibirá la retroalimentación por parte del profesor y demás actores del proceso. Desde el sistema

que consume la *API* se realiza el proceso de captura y envío del archivo PDF hacia el *Endpoint* correspondiente en la *API*. En esta ruta se ejecuta la función para realizar el guardado del archivo en la carpeta *public* del proyecto y desde la cual podremos generar la ruta con la que podremos acceder a este archivo más adelante. En la base de datos se guarda la ruta de este archivo y en caso de que este sea actualizado o se necesite subir una nueva versión, se reemplaza el archivo original de la ruta por el nuevo y de esta forma se logra optimizar espacio y recursos. A continuación, en la **Fig. 14** se visualiza la función que guarda el archivo del proyecto.

```php
public function updatePdf(Request $request, Project $project)
{
    $user = Auth::user();
    $date = new DateTime();
    $student_id = $user->userable->id;
    $fileNameToStore = "project.pdf";
    $request->report_pdf->storeAs("public/reports/{$student_id}", $fileNameToStore);
    $project->report_pdf = "storage/reports/{$student_id}/{$fileNameToStore}";
    $project->save();
    return response()->json($project, status: 200);
}
```

Fig. 14: Guardado de PDF de proyecto

3.3.5 Revisión y aprobación del proyecto por parte del profesor

El sistema consumidor tiene la posibilidad de obtener el PDF del proyecto subido por el estudiante y presentarlo de forma que éste pueda agregar sus comentarios. Desde la *API* está configurada la función que nos permite leer los datos de este archivo. En la **Fig. 15** se presenta la función tipo *GET* que nos ayuda a acceder a la ruta de la carpeta *public* utilizada.

```php
public function getProjectPDFFile(Project $project){
    return response()->file(public_path($project->report_pdf));
}
```

Fig. 15: Obtención de archivo PDF de proyecto

3.3.6 Notificaciones.

Cada etapa, avance y cambio de estado del proyecto durante todo el proceso de titulación, genera una notificación a los actores involucrados para que estos puedan estar al tanto de estos cambios. Para esto, se realiza la configuración de un proveedor de correo que pueda ser ejecutado en el momento que se realiza estos cambios de estado. Este componente nos permite personalizar los mensajes que le llegan a los usuarios y que estos vayan acordes con la etapa que acaba de ser cumplida. Se envía como una variable los correos de los usuarios que recibirán la notificación y también el estado que se está cumpliendo para que el texto se adapte en consecuencia. A continuación, en las **Fig. 16** se puede visualizar la ejecución del componente de mail cuando se realiza un cambio de estado y el correo que le llega al usuario respectivamente.

```
public function projectUploaded(Project $project)
{
    $mail = new NewPdfUpload($project);
    return $this->changeStatus($project->id, $mail, $project->teacher->user,  newStatus: "project_uploaded",  prevStatus: "plan_approved_commission");
}
```

Fig. 16: Envío de email en cambio de estado

3.4 *Sprint* 3. Desarrollo de los módulos de Secretaría y Dirección

Continuando con la planificación hecha en el *Sprint Backlog*, en el *Sprint 3*, se ejecutan las tareas correspondientes a los módulos de secretaría y dirección.

Como resultado de este *Sprint* se obtiene:

- Ingreso de datos de estudiantes y profesores (Secretaría).
- Creación de comisiones por carreras (Secretaría).
- Listado de proyectos de titulación y sus estados (Secretaría).
- *Checklist* para currículum saneado I y II (Secretaría).
- Asignación de tribunales (Dirección).
- Asignación y modificación de horarios de defensa (Dirección).

3.4.1 Ingreso de datos de estudiantes y profesores (Secretaría)

Desde el módulo de secretaría, el usuario autenticado tiene la posibilidad de ingresar los datos requeridos para estudiantes y profesores y guardarlos en la base de datos, para luego poder ser consultados mediante la ruta correspondiente.

El proceso de creación es distinto en los dos usuarios, siendo necesario en el caso de los estudiantes la subida de un archivo de Excel con los datos hacia el *API* y el guardado en la base de datos. Con los Profesores, en cambio, se hace el ingreso de los datos directamente uno por uno.

Para la subida del archivo de Excel se deben cumplir varios pasos de configuración, tras la creación del modelo *Students,* donde se encuentran las columnas que corresponden a esa tabla y las relaciones que tiene con otras tablas, es necesario generar la migración para que se pueda crear la tabla en la base de datos. Así, está listo para configurar la función de la ruta de guardado. Es necesario una librería adicional para poder realizar el proceso, esta librería es *maatwebsite/Excel*, que es la encargada de realizar el proceso de importación y exportación de datos desde el archivo de Excel hacia la base de datos a través de la *API* y viceversa [24].

Ahora, se crea la función que realiza el mapeo de los datos del archivo de Excel, el cual se guarda en la carpeta *public* del proyecto. En esta función se especifica cómo se recorre cada celda del archivo y se lo relaciona con la tabla en la que se guarda de la base de datos. Al final, los datos son colocados en sus posiciones correspondientes, siempre y cuando estos coincidan completamente desde su origen hasta su destino. En las figuras **Fig. 17, Fig. 18, Fig. 19** se visualiza la función de importación, la ruta de acceso y el archivo de Excel de ejemplo respectivamente.

```php
public function uploadImportFile(Request $request)
{
    $fileName = 'estudiantes.xlsx';
    $request->file( key: 'file')->move(public_path( path: '/files'), $fileName);

    try {
        Excel::import(new StudentImport(),  filePath: 'files/estudiantes.xlsx');
    } catch (\Maatwebsite\Excel\Validators\ValidationException $e) {
        $failures = $e->failures();
        $collectionFailures = collect();

        foreach ($failures as $failure) {
            $collectionFailures = $collectionFailures->push(
                [
                    'row' => $failure->row(),
                    'attribute' => $failure->attribute(),
                    'errors' => $failure->errors(),
                    'values' => $failure->values()
                ]);
        }

        $collectionFailures = $collectionFailures->groupBy( groupBy: 'row');
        $collectionByRows = collect();

        foreach ($collectionFailures as $dataByRowsKey => $dataByRows) {
            $collectionByRow = collect();
            foreach ($dataByRows as $dataByRow) {
                foreach ($dataByRow as $key => $data) {
                    (is_null($collectionByRow->get( key: '' . $key))) ? $dataArray = collect() : $dataArray = $collectionByRow->get( key: '' . $key);
                    $dataArray->push($data);
                    $dataArray = $dataArray->unique();
                    $collectionByRow->put('' . $key, $dataArray);
                }
            }
            $collectionByRows->put('' . $dataByRowsKey, $collectionByRow);
```

Fig. 17: Función de importación de datos

```php
Route::post('students/uploadImportFile', [StudentController::class, 'uploadImportFile']);
```

Fig. 18: Ruta para guardado de archivo

A	B	C	D
name	unique_number	email	career_id
Jonathan Vasquez	201410413	jonathan.vasquez01@epn.edu.ec	1
Chantal Morales	201559876	chantal.morales@epn.edu.ec	1
Nicole Zambrano	201654983	nicole.zambrano@epn.edu.ec	1
Kevin Segovia	201443258	kevin.segovia@epn.edu.ec	1

Fig. 19: Archivo de ejemplo

3.4.2 Creación de comisiones de carreras (Secretaría)

Desde el módulo de secretaría es posible realizar la asignación de profesores a las diferentes comisiones que representan a las carreras en el proceso de revisión de

planes de titulación. Para este proceso se establecen las bases para el guardado de los datos en la base y las rutas de escritura y lectura.

Se crea el modelo *Commision* que contiene los datos correspondientes a la tabla y la relación de ésta con *Teachers* y *Career*, siendo así que, para la creación de una comisión, esta deberá pertenecer a una carrera en particular y será conformada por los profesores que corresponden a ésta. En la **Fig. 20** se puede visualizar el modelo y las relaciones.

```php
class Commission extends Model
{
    protected $fillable = [
        'commission_schedule',
        'career_id'
    ];


    public function teachers(){
        return $this->hasMany( related: 'App\Models\Teacher');
    }

    public function career(){
        return $this->belongsTo( related: 'App\Models\Career');
    }
}
```

Fig. 20: Modelo de Comisión

A continuación, se establece la migración para crear la tabla en la base de datos. En esta tabla se establecen las columnas que contienen los datos de los profesores y carreras de las comisiones. Finalmente, se configura el controlador para crear las funciones de las rutas que permiten el consumo de los datos a través de éstas. Algunas consideraciones importantes en esta sección es que la creación de las comisiones se realiza de forma que solo exista una comisión por carrera, ya que, bajo los parámetros actuales de trabajo, esta comisión actuará en sus obligaciones durante el semestre completo. También, se realiza la condición de que los profesores disponibles para integrar la comisión sean pertenecientes a la carrera correspondiente y no existan

cruces erróneos. En la **Fig. 21** se visualiza la función de guardado y consumo de los datos de comisiones.

```php
public function show(Commission $commissions)
{
    return response()->json(new CommissionResource($commissions), status: 200);
}
public function store(Request $request)
{
    $commissions= new Commission($request->except(['members']));
    $commissions->save();
    foreach ($request->members as $member){
        $teacher_commission = Teacher::find($member);
        $teacher_commission->commission()->associate($commissions);
        $teacher_commission->save();
    }
    return response()->json(new CommissionResource($commissions),  status: 201);
}
public function update(Request $request, Commission $commissions)
{
    $commissions->update($request->all());
    return response()->json(new CommissionResource($commissions),  status: 200);
}
```

Fig. 21: Controlador de Comisión

3.4.3 Listado de proyectos de titulación y sus estados (Secretaría)

La entidad que sirve como conexión de casi todos los componentes de la *API* es el proyecto de titulación y el avance que éste tenga en su estado determina también el avance del estudiante en su proceso de titulación. Esta entidad cuenta con su controlador en el que se establecen las relaciones que mantiene con Estudiantes, Profesores y Tribunal calificador. En la **Fig. 22** se puede ver la relación de uno a varios y uno a uno que posee el proyecto con las demás entidades.

```php
public function students()
{
    return $this->belongsToMany( related: 'App\Models\Student')->withTimestamps();
}

public function teacher()
{
    return $this->belongsTo( related: 'App\Models\Teacher',  foreignKey: 'teacher_id');
}

public function jury()
{
    return $this->hasOne( related: 'App\Models\Jury');
}
```

Fig. 22: Relaciones del proyecto

Al igual que otras entidades, es necesario realizar la migración para que se cree la tabla donde se alojan los datos de los proyectos. Cuando se crea el proyecto se establece un estado inicial para el proceso de titulación que irá variando de acuerdo con el avance de éste.

Pasando a la sección del controlador y las funciones de creación, actualización y guardado de los proyectos, cada uno de los accesos a estas rutas que correspondan a un paso cumplido del proceso cambia los estados del proyecto en alguno de los que se puede visualizar en la **Fig. 23**.

```
$table->enum( column: 'status', [
    'plan_saved',
    'plan_sent',
    'plan_review_teacher',
    'plan_corrections_done',
    'plan_approved_director',
    'san_curriculum_1',
    'plan_review_commission',
    'plan_corrections_done2',
    'plan_approved_commission',
    'plan_rejected',
    'project_uploaded',
    'project_review_teacher',
    'project_corrections_done',
    'project_approved_director',
    'san_curriculum_2',
    'tribunal_assigned',
    'project_graded',
    'test_defense_apt',
    'date_defense_assigned',
    'project_completed',
    'project_rejected']);
```

Fig. 23: Estados del proyecto

Hay que considerar algunos aspectos en la consulta de los datos de proyectos a la API, como son las validaciones de los campos, los cambios de estados y si el proyecto es realizado por uno o más estudiantes.

A continuación, en la **Fig. 24** se visualiza la función de guardado de un nuevo proyecto y los datos que intervienen.

```php
public function store(Request $request)
{
    $this->authorize( ability: 'create',  arguments: Project::class);
    $request->validate([
        'title' => 'string|unique:projects|max:255',
        'teacher_id' => 'required|exists:teachers,id',
        'schedule' => 'nullable',
        'student_id_2' => 'nullable|exists:users,id'
    ], self::$messages);

    $project = new Project($request->except(['student_id_2']));

    $user = Auth::user();
    $students[] = Auth::user();
    $project->save();
    if ($request->student_id_2 != null) {
        $project->students()->sync([$user->userable->id, $request->student_id_2]);
        $students[] = Student::find($request->student_id_2)->user;
    } else {
        $project->students()->sync([$user->userable->id]);
    }

    if ($request->status == 'plan_sent') {
        Mail::to($project->teacher->user)->send(new NewProjectUploadTeacher($project));
        Mail::to($students)->send(new NewProjectStudent($project));
    }


    return response()->json(new ProjectResource($project),  status: 201);
}
```

Fig. 24: Función de guardado del proyecto

3.4.4 *Checklis*t para currículum saneado I y II (Secretaría)

Los *checklis*t de currículum saneado son una parte importante de aprobación en el proceso de titulación y aunque en la actuación de la *API* no se realice más que un cambio de estado, es importante mencionarlo ya que de este cambio depende el avance a otras etapas del proceso. Desde el sistema consumidor de la *API* se puede realizar una llamada a la *API* para realizar el cambio de estado al proyecto una vez que el estudiante haya completado una serie de requisitos administrativos y académicos que

permitan habilitarlo para las próximas etapas. La forma en la que se realiza el cambio de estado es una representación de como ocurre este proceso en varias etapas del proceso. Para esto se utiliza una función asignada a una ruta específica para este propósito, asegurando así que la acción realizada no modifique nada más de lo establecido. En la **Fig. 25** se puede visualizar la función de cambio de estado para aprobar el Currículum Saneado II.

```php
public function sanCurriculum2(Project $project)
{
    $mail = new NewPlanUploadCommission($project); // TODO cambiar la estructura del correo
    return $this->changeStatus($project->id, $mail, $project->teacher->user,  newStatus: "san_curriculum_2",  prevStatus: "project_approved_director");
}
```

Fig. 25: Función de cambio de estado de currículum

3.4.5 Asignación de tribunales (Dirección)

Para poder realizar la asignación de tribunales para la revisión de proyectos de titulación se deben cumplir algunos parámetros previos, cómo son que el usuario que va a realizar esta acción es un miembro de la dirección autenticado como tal en el sistema y que el proyecto se encuentra en el estado de pendiente de asignación de tribunal. Si estos aspectos no son cumplidos no se puede realizar la asignación del tribunal. A partir de estas consideraciones, este proceso comparte muchas similitudes con la asignación de miembros para la comisión de carrera, con la diferencia clara de que estos tribunales van variando de acuerdo con el proyecto de titulación en cuestión, siendo así que pueden existir muchos tribunales durante el semestre. Una vez que se escogen los profesores que conformarán el tribunal se realiza el llamado a la ruta de guardado como se muestra en la **Fig. 26**.

```php
public function store ( Request $request)
{
    $juries = new Jury($request->except(['members']));
    $juries->save();
    $juries->teachers()->sync($request->members);
    return response()->json($juries,  status: 201);
}
```

Fig. 26: Función de guardado de tribunal

3.4.6 Asignación y modificación de horarios de defensa (Dirección)

El usuario autenticado como dirección puede realizar la asignación de horarios para las defensas de grado. Para esto será necesario que el proyecto se encuentre en el estado de Apto para defensa oral, así el estudiante ha cumplido con todos los requisitos previos para esta última etapa. Cada profesor cuenta con los datos de su horario de trabajo, los cuales pueden ser solicitados a la *API* para que se realice el cruce de horarios desde el sistema y se encuentre el día y hora propicio para formalizar la defensa.

3.5 Sprint 4. Implementación de pruebas unitarias de la *API REST*

Siguiendo con la planificación hecha en el *Sprint Backlog*, en el *Sprint 4*, se definen las tareas para la verificación del funcionamiento de la *API* mediante pruebas unitarias.

Como resultado de este *Sprint* se obtiene:

- Pruebas unitarias a rutas.
- Mejoramiento del código frente a la simulación de la *API*.
- Comprobación de llamadas al servidor por medio de la *API*.
- Analizar el funcionamiento de cada componente del sistema que realice consultas.

3.5.1 Pruebas unitarias a rutas

Las pruebas unitarias son una buena práctica de desarrollo que permite asegurar la calidad de los productos desarrollados y ayuda a reducir el riesgo de posibles errores o fallos futuros. En el caso de *Laravel*, permite realizar pruebas a través de mecanismos incluidos en el *Framework*, como es *PHPUnit*, una herramienta muy útil para probar funciones o rutas.

En la se realiza una prueba simple de consulta al *Endpoint* para obtener los datos de estudiantes, pero sin realizar la autenticación previa. Se espera que si **Fig. 27** el resultado de la prueba es satisfactorio, devuelva el código de estado 200, caso contrario, muestre el código de error.

```php
public function test_example()
{
    $response = $this->get( uri: 'http://localhost:8000/api/students/');

    $response->assertStatus( status: 200);
}
```

Fig. 27: Prueba unitaria de consumo sin autorización

Dado que no se puede acceder a la información sin realizar autenticación previa el error mostrado en la **Fig. 28** es 401 por falta de autorización.

```
$ vendor/bin/phpunit --filter apiTest
PHPUnit 9.5.5 by Sebastian Bergmann and contributors.

F                                                                   1 / 1 (100%)

Time: 00:01.933, Memory: 22.00 MB

There was 1 failure:

1) Tests\Feature\apiTest::test_example
Expected status code 200 but received 401.
Failed asserting that 200 is identical to 401.

C:\Users\Home\Desktop\TESIS-JUNIO\BACKEND\vendor\laravel\framework\src\Illuminate\Testing\TestResponse.php:187
C:\Users\Home\Desktop\TESIS-JUNIO\BACKEND\tests\Feature\apiTest.php:20

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.
```

Fig. 28: Resultado de la prueba sin autorización

Para poder probar que el inicio de sesión funciona correctamente, se realiza una prueba de función para un método *POST* en la ruta de *Login*. Es necesario pasar como parámetros un email y contraseña correctos para que devuelva una respuesta de proceso completado correctamente. En la **Fig. 29** se visualiza la función para inicio de sesión de un profesor y en la **Fig. 30** la respuesta satisfactoria en consola.

```php
public function test_example()
{

    $body = [
        'email' => 'profesor0@epn.edu.ec',
        'password' => '123456'
    ];
    $this->json( method: 'POST', uri: 'http://localhost:8000/api/login',$body,['Accept' => 'application/json'])
        ->assertStatus( status: 200)
        ->assertJsonStructure(['token']);
}
```

Fig. 29: Prueba unitaria de inicio de sesión

```
$ vendor/bin/phpunit --filter apiTest
PHPUnit 9.5.5 by Sebastian Bergmann and contributors.

.                                                                   1 / 1 (100%)

Time: 00:00.294, Memory: 24.00 MB

OK (1 test, 2 assertions)
```

Fig. 30: Resultado prueba unitaria de autenticación

En el caso de las solicitudes que ya poseen un *token* de autorización, este debe ser enviado en las cabeceras de la solicitud. A continuación, en la **Fig. 31** se observa la prueba unitaria que busca verificar que los datos de carrera en la posición 1, correspondan a los especificados. Para esto es necesario especificar el método que se usará, la ruta y las cabeceras. Al final, en la **Fig. 32** se puede observar la respuesta satisfactoria en la prueba.

```
$response = $this->json( method: 'GET',  uri: 'http://localhost:8000/api/careers/1',[],['Authorization' => 'Bearer eyJ0eXAi
$response->assertExactJson([
    'id'=> 1,
    'name'=> 'ASI',
    'created_at'=>'2021-08-11T01:12:57.000000Z',
    'updated_at'=>'2021-08-11T01:12:57.000000Z'
]);
```

Fig. 31: Prueba unitaria método *get* de carreras

```
$ vendor/bin/phpunit --filter apiTest
PHPUnit 9.5.5 by Sebastian Bergmann and contributors.

.                                                                   1 / 1 (100%)

Time: 00:00.274, Memory: 22.00 MB

OK (1 test, 1 assertion)
```

Fig. 32: Resultado prueba unitaria método *get*

3.5.2 Mejoramiento del código frente a la simulación de la *API*

Las simulaciones se las realiza mediante los *endpoints*, comprobando que los datos devueltos por la API sean los correctos. De forma que, los *endpoints* deben estar correctamente formulados y distribuidos. En este sentido, se realiza un mejoramiento en la forma de acceso a estas rutas, para que éstas apunten a la función correspondiente que se encuentra en el controlador. Así, separamos de una forma más clara y reutilizable a las funciones y a las rutas de acceso. A continuación, en la **Fig. 33** podemos observar la sección de rutas y en la **Fig. 34** las funciones en el controlador de Profesores.

```php
public function show()
{
    $user=Auth::user();
    $teacher = $user->userable;
    return response()->json(ProjectResource::collection($teacher->projects), 200);
}

public function store(Request $request)
{
    $faker = \Faker\Factory::create();
    $password = Hash::make('123456');
    $teacher = Teacher::create([
        'titular'=> "1",
        'committee'=>"0",
        "career_id"=> $request->career_id,
        "schedule" => $request->schedule
    ]);
    $teacher->user()->create([
        'name' => $request->name,
        'email' => $request->email,
        'password' => $password,
        'role' => User::ROLE_TEACHER
    ]);
    return response()->json(new TeacherResource($teacher), 201);
}
```

Fig. 33: Funciones en controlador

```php
//teacher
    Route::get('teachers', [TeacherController::class, 'index']);
    Route::get('teacher-projects', [TeacherController::class, 'show']);
    Route::post('teachers/', [TeacherController::class, 'store']);
    Route::put('teachers/{teacher}', [TeacherController::class, 'update']);
    Route::delete('teachers/{teacher}', [TeacherController::class, 'delete']);
    Route::get('teachers/projects', [TeacherController::class, 'projects']);
    Route::get('teachers/{teacher}/projects/{project}', [TeacherController::class, 'project']);
```

Fig. 34: Llamado de funciones en rutas

3.5.3 Comprobación de llamadas al servidor por medio de la *API*

A través de *Postman* se realizan las llamadas al servidor local durante el desarrollo, especificando el tipo de solicitud correspondiente a la ruta y el *URL* de acceso y al servidor en línea en el ambiente de producción, donde el armado del *URL* es el mismo diferenciándose en la dirección del servidor. Para poder hacer las llamadas también es importante tener iniciada la sesión para que nuestro usuario activo posea el *token* de acceso y la autorización para hacer las consultas correspondientes.

El t*oken* de acceso se lo obtiene realizando una solicitud de tipo post, a la ruta *Login*, enviando como cuerpo de la solicitud el usuario y contraseña. La respuesta de esta solicitud es el *token* que deberemos incluir en nuestras otras consultas. La respuesta del *Login* se puede observar en la **Fig. 35**.


```
{
    "token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.
        eyJpc3MiOiJodHRwOlwvXC9sb2NhbGhvc3Q6ODAwMFwvYXBpXC9sb2dpbiIsImlhdCI6MTYzMDI1NjUxNCwiZXhwIjoxNjMwMjYwMTE0L
        YmQ1Yzg5NDlmNjAwYWRiMzllNzAxYzQwMDg3MmRiN2E1OTc2ZjcifQ.fnOHgCaB-02ce6kjofSSzkegNdnboM-k1KSmFt-zxb8",
    "user": {
        "id": 3,
        "name": "Betsy Brown Sr.",
        "email": "secretaria@epn.edu.ec",
        "email_verified_at": null,
        "role": "ROLE_SECRETARY",
        "created_at": "2021-08-11T01:12:58.000000Z",
        "updated_at": "2021-08-11T01:12:58.000000Z",
        "userable_id": 1,
        "userable_type": "App\\Models\\Secretary"
    }
}
```


Fig. 35: Inicio de sesión

En caso de que exista algún error en las llamadas al servidor, se recibe el código de estado de esa consulta para tener una idea más clara de lo que ha salido mal.

3.5.4 Analizar el funcionamiento de cada componente del sistema que realice consultas

El sistema consumidor de la *API* recibe una respuesta por parte de ésta, en caso de que la solicitud haya sido exitosa o si se tiene algún error. Por lo cual, desde el *Frontend* se puede obtener estos mensajes y mostrarlos de una forma más explícita al usuario, haciendo que la reacción de cada componente del sistema a las respuestas de la *API* deba ser definidas en su programación. Con las pruebas unitarias a las llamadas al servidor se logra la comprobación de que los mensajes de estado se muestren correctamente y así garantice que el comportamiento en los componentes del *Fronten*d

sean los correctos. En la **Fig. 36** se muestra un ejemplo de un mensaje de estado exitoso.

Fig. 36: Estado de la solicitud

3.6 *Sprint* 5. Despliegue de *la API REST* en un servidor web público

En base a la planificación del *Sprint Backlog*, en el *Sprint 4*, se definen las tareas para el despliegue de la *API REST* a un servidor web público y las pruebas finales de funcionamiento.

Al completar este *Sprint* se obtiene:

- Configuración del servidor web Apache.
- Chequeo del rendimiento del sistema en el servidor.
- Comprobación de la seguridad del sistema.
- Prueba del consumo de la *API* desde otro sistema.

3.6.1 Configuración del servidor *web* Apache

A continuación, se describe el proceso de despliegue de la *API REST* en ambiente de producción. Para esto, se configura la plataforma *Heroku* para que pueda alojar todo el sistema y la base de datos. Esta plataforma ofrece varias herramientas en su versión gratuita lo que la convierte en la mejor alternativa para cumplir los requerimientos iniciales de la *API*.

Es necesario tener una cuenta creada en la plataforma *Heroku* para poder usar sus herramientas, además de la instalación de *Heroku CLI* para usar comandos propios de la plataforma y subir al sistema. Es necesario especificar la carpeta que se sube a producción antes de comenzar el proceso, la carpeta que se exporta en este caso es *public*, como se puede observar en la **Fig. 37.**

Fig. 37: Carpeta que sube a producción

Para que el proceso se complete correctamente, es necesario configurar las variables de entorno en nuestra *App de Heroku* y así tenga acceso a todas las funciones de la *API*. Para el despliegue de la base de datos en producción es necesario que ésta se encuentre alojada en el *SGBD PostgreSQL*, ya que el proceso de subida con las herramientas gratuitas de la plataforma exige que la vinculación se la realice de esa forma. Una vez que se completa el proceso de despliegue a producción podemos obtener nuestro *URL* para armar los *endpoints* y realizar las consultas a la *API*. El estado de nuestra aplicación en producción está disponible en el dashboard de *Heroku*, como se muestra en la **Fig. 38.**

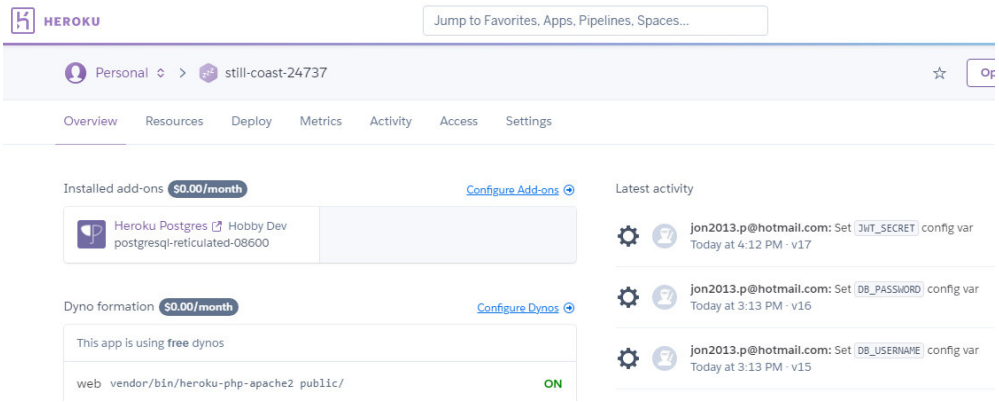


Fig. 38: Aplicación alojada en *Heroku*

3.6.2 Chequeo del rendimiento del sistema en el servidor

Una de las mayores ventajas de *Heroku* en el control de rendimiento de aplicaciones es la presentación de estadísticas y estados de servicio, facilitando de gran manera que se pueda llevar una revisión permanente de la disponibilidad del sistema. Además, aquí podremos comprobar el número de usuarios, sistemas a los que se encuentra vinculada la aplicación, carga de datos hecha al servidor y costos generados en estas

operaciones. En la **Fig. 39** se puede ver la pantalla de control que existe en *Heroku* para llevar un registro del rendimiento y estado de los datos de la *API*.

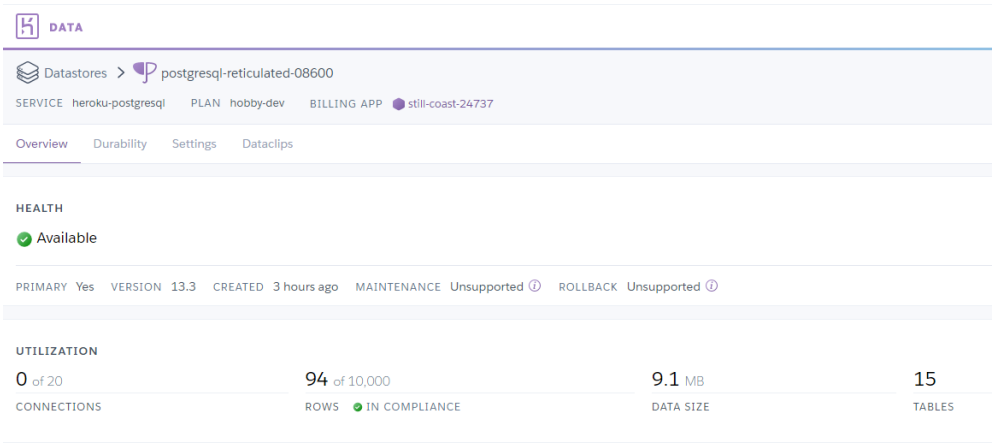


Fig. 39: Estado de los datos en *Heroku*

Algunas consideraciones que hay que tomar en cuenta son las limitaciones que existen en la plataforma debido al plan al que se accede para poder desplegar nuestra aplicación, las métricas extendidas y el número de datos que se pueden guardar en el servidor tienen un límite, pero no impiden el correcto funcionamiento y control de todas las funcionalidades del sistema.

3.6.3 Comprobación de la seguridad del sistema.

La plataforma *Heroku* brinda seguridad en los despliegues, brindando seguridad *SSL* a nuestras direcciones de acceso y para todas las consultas que se realicen a través de éstas. La herramienta mantiene activo permanentemente la seguridad a las consultas y accesos de la aplicación, incluso en la versión gratuita.

En cuanto a la seguridad que posee la *API* de cara a posibles intentos de acceso no autorizados, el código se encuentra hecho de forma que las consultas no puedan ser realizadas si estas no poseen en su cabecera de petición, el *token* generado por *JWT* al momento de iniciar sesión como un usuario registrado. En caso de que el usuario no posea las credenciales correctas de acceso, se le responde con un mensaje de error e impidiendo el acceso a cualquier característica de la *API*. En la siguiente sección, referente a pruebas de consumo en producción se puede visualizar cómo se realiza una consulta correcta a la *API*.

3.6.4 Prueba del consumo de la *API* desde otro sistema

Con el proceso de despliegue a producción completado, el servidor está listo para poder resolver las peticiones a través de los *Endpoints*. Para esto, a diferencia de la etapa de desarrollo en la cual la dirección fue el servidor local de los equipos, ahora se debe utilizar la dirección brindada por *Heroku* y así armar nuevos *endpoints* para recibir las respuestas correspondientes. El primer paso siempre es obtener el *token* de acceso, de tal forma que cualquier sistema que vaya a consumir la *API* debe implementar el inicio de sesión correspondiente para la generación de este token. A continuación, en la **Fig. 40** se puede observar el inicio de sesión en el servidor a través de una solicitud de tipo POST a la ruta correspondiente y así obtener como respuesta el token necesario.

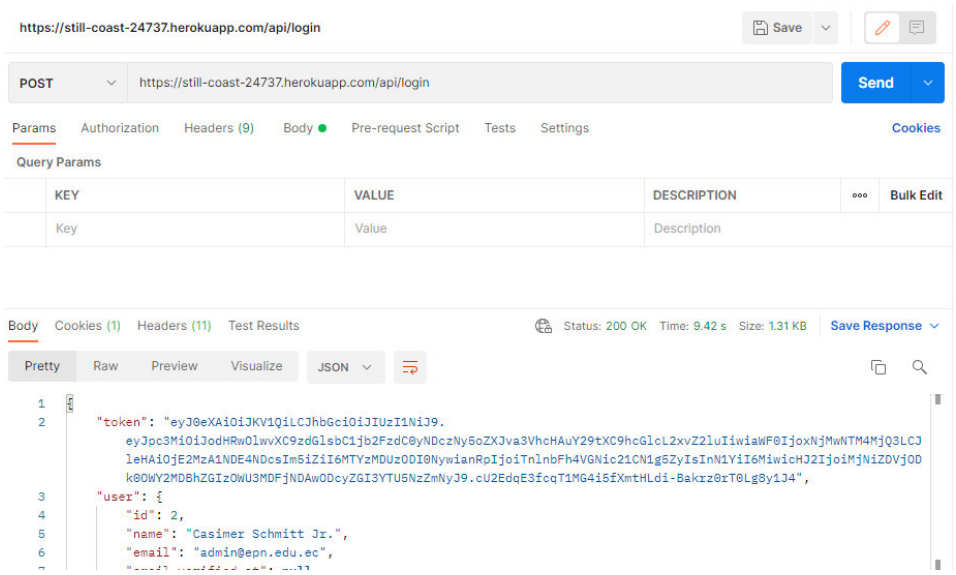


Fig. 40: Inicio de sesión en producción

Con el *token* generado, se puede incluir en la cabecera de las peticiones para que así tengan la autorización necesaria y no recibamos el mensaje de error. Finalmente, en la **Fig. 41** se puede ver el resultado de la petición tipo *GET* para obtener los datos de todos los profesores registrados a través del *Endpoint* correspondiente.

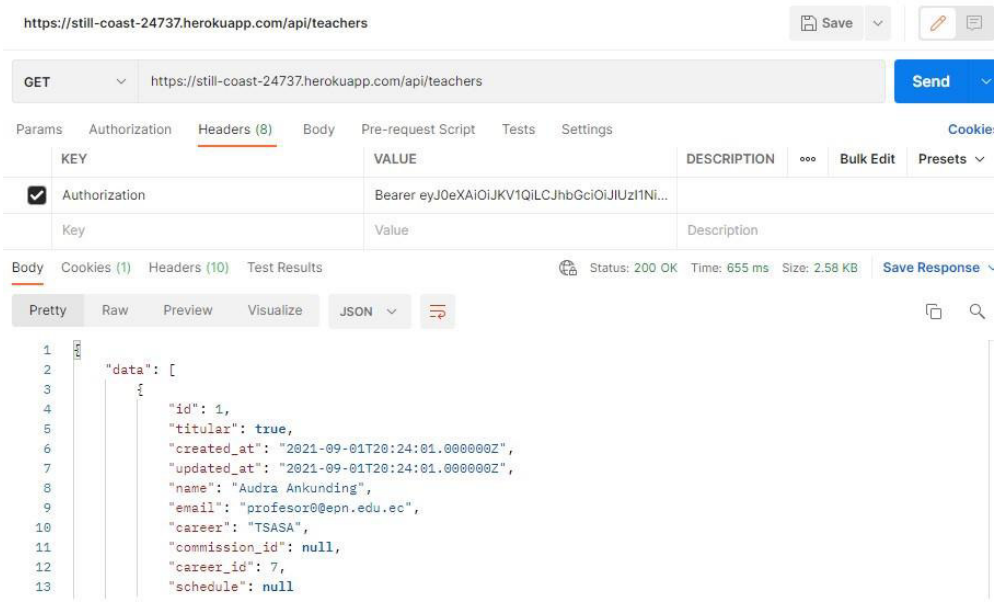


Fig. 41: Consulta a producción

4 CONCLUSIONES Y RECOMENDACIONES

En la siguiente sección se presentan las conclusiones y recomendaciones que se obtuvieron durante el desarrollo de este proyecto.

4.1 Conclusiones

- La *API REST* cumple con cada uno de los objetivos planteados y además con el alcance definido, puede ser accedida desde cualquier cliente y funciona de manera conjunta con la parte del *frontend*, cumpliendo además con las directrices impuestas por la pandemia.
- La correcta definición de los requerimientos y funcionalidades que el sistema tiene previo al comienzo de la etapa de desarrollo ha hecho que el proceso de implementación de la *API REST* sea mucho más estructurada, limpia y eficiente.
- El uso de *Scrum* como principal metodología y como metodología ágil permite obtener un producto de valor en cada *sprint*, así como correcciones en cada iteración para que el resultado final cumpla con todos los requisitos requeridos. Además, ayuda a que el tiempo de desarrollo se cumpla y se maneje en periodos establecidos.
- Al establecer la estructura correcta como lo es MVC permite obtener un sistema escalable y replicable para otros sistemas de manera que puede ser utilizado por otros clientes.
- El uso de *Laravel* como principal herramienta para el desarrollo de este sistema hace que la interacción con la base de datos a través de *ORM* sea mucho más simple y que la interacción con la misma sea de manera eficiente y rápida.
- Con la etapa de pruebas unitarias y las iteraciones que se hicieron se pudo comprobar cómo funcionaría esta *API REST* en un ambiente de producción los resultados han sido favorables ya que las respuestas a las peticiones son los datos que se solicitaron, una vez que se comprueba esta información se pudo también corregir algunas falencias que se pudieran producir.
- La plataforma *Heroku* ofrece una serie de herramientas y utilidades que permiten el despliegue fácil y seguro de aplicaciones con objetivos escalables como los de la *API REST*. Los desarrolladores tienen la oportunidad de migrar entre el plan gratuito al *premium* que brinda muchas más características y hace que el crecimiento de la aplicación y terceros sea exponencial.

4.2 Recomendaciones

- Se recomienda planificar cada *sprint* con las tareas necesarias para que en cada iteración podemos obtener un producto de valor, asimismo es importante que las asignaciones y las estimaciones sean hechas de manera óptima.
- Se recomienda utilizar herramientas que mantengan las librerías o f*rameworks* que se estén utilizando actualizados automáticamente ya que si podemos evitar errores.
- Es importante que cada componente cumpla con la estructura o patrón arquitectónico que se define para que pueda ser integrado al sistema de una manera limpia y sin complejidad.
- Es importante que se mantenga reuniones con el cliente o dueño del producto y acá si podemos verificar que no haya cambios en los requerimientos y que se siga el desarrollo de acuerdo con lo planificado y establecido previamente.
- Durante el proceso de desarrollo y en el despliegue a producción de una *API* y en general de la mayoría de las aplicaciones, es importante que los datos que corresponden a las variables de entorno se los maneje de forma muy cautelosa, realizando acciones de protección propios de los buenos estándares de desarrollo. De esta forma, se mantiene a salvo a la aplicación de cualquier acceso no autorizado o modificación maliciosa.

5 REFERENCIAS BIBLIOGRÁFICAS

[1] ESFOT-EPN, «ESFOT,» 2021. [En línea]. Available: https://esfot.epn.edu.ec/#. [Último acceso: 26 Agosto 2021].

[2] L. p. Edwin Salvador, «Informe Actividades Comisión de Automatización de Procesos 2019B,» Quito, 202.

[3] Escuela de Formación de Tecnólogos- Escuela Politécnica Nacional, «Proceso para entrega de Anillados - Emergencia Sanitaria: ESFOT,» 2020. [En línea]. Available: https://esfot.epn.edu.ec/index.php/unidad-titulacion/entrega-de-anillados-emergencia-sanitaria. [Último acceso: 26 Agosto 2021].

[4] Ambientum, «Tecnología móvil para controlar el COVID-19,» Ambientum - Tecnología, 6 Abril 2020. [En línea]. Available: https://www.ambientum.com/ambientum/tecnologia/tecnologia-movil-para-controlar-el-covid-19.asp. [Último acceso: 16 Julio 2020].

[5] M. Bara, «Roles, Eventos y Artefactos en la metodología Scrum: OBS Business School,» 5 Septiembre 2017. [En línea]. Available: https://www.obsbusiness.school/blog/roles-eventos-y-artefactos-en-la-metodologia-scrum. [Último acceso: 23 Agosto 2021].

[6] M. Rehkopf, «Agile - Historias de usuario,» ATLASSIAN Agile Couch, [En línea]. Available: https://www.atlassian.com/es/agile/project-management/user-stories. [Último acceso: 25 noviembre 2020].

[7] M. Á. Álvarez, «Qué es MVC: desarrollo web,» 28 julio 2020. [En línea]. Available: https://desarrolloweb.com/articulos/que-es-mvc.html. [Último acceso: 26 agosto 2021].

[8] ESIC Business & Marketing School., «El ORM como herramienta eficiente de trabajo: ESIC,» enero 2018. [En línea]. Available: https://www.esic.edu/rethink/tecnologia/el-orm-como-herramienta-eficiente-de-trabajo. [Último acceso: 25 agosto 2021].

[9] V. Peña, «Modelos y Eloquent ORM: Norvic,» 30 abril 2021. [En línea]. Available: https://norvicsoftware.com/modelos-y-eloquent-orm-en-laravel-8/. [Último acceso: 25 agosto 2021].

[10] K. Palomares, «Que es Laravel,» Kiko Palomares, 22 Octubre 2019. [En línea]. Available: https://www.kikopalomares.com/blog/que-es-laravel-y-para-que-sirve-frameworks-de-php. [Último acceso: 22 agosto 2020].

[11] HostingPedia, «Alogamiento Web - PostgreSQL,» HostingPedia, 07 Febrero 2019. [En línea]. Available: https://hostingpedia.net/postgresql.html. [Último acceso: 01 Mayo 2021].

[12] Desarrolloweb.com, «Manuales - Tutorial de Composer,» desarrolloweb.com, 24 enero 2020. [En línea]. Available: https://desarrolloweb.com/articulos/composer-gestor-dependencias-para-php.html. [Último acceso: 25 agosto 2021].

[13] A. López, «Desarrollo Web - Que es Postman y para que sirve,» Open Webinars, 03 junio 2019. [En línea]. Available: https://openwebinars.net/blog/que-es-postman/. [Último acceso: 25 Agosto 2021].

[14] IONOS, «Herramientas - Instala tu servidor local XAMPP,» Digital Guide IONOS by 1&1, 03 Septiembre 2019. [En línea]. Available: https://www.ionos.es/digitalguide/servidores/herramientas/instala-tu-servidor-local-xampp-en-unos-pocos-pasos/. [Último acceso: 25 agosto 2021].

[15] A. M. Robledano, «Qué es MySQL: Características y ventajas: OpenWebinars,» 24 Septiembre 2019. [En línea]. Available: https://openwebinars.net/blog/que-es-mysql/. [Último acceso: 22 Septiembre 2021].

[16] Capterra, «Heroku: Capterra,» 2007. [En línea]. Available: https://www.capterra.ec/software/158191/heroku. [Último acceso: 22 Septiembre 2021].

[17] git, «Git,» [En línea]. Available: https://git-scm.com/. [Último acceso: 22 septiembre 2021].

[18] L. Castillo, «Documentación: Conociendo GitHub,» 20212. [En línea]. Available: https://conociendogithub.readthedocs.io/en/latest/data/introduccion/. [Último acceso: 22 Septiembre 2021].

[19] J. D. Polo, «ZENHUB, PLATAFORMA DE GESTIÓN DE PROYECTOS INTEGRADA EN GITHUB, AHORA GRATIS PARA ESTUDIANTES: wwwhatsnew,» 24 septiembre 2015. [En línea]. Available: https://wwwhatsnew.com/2015/09/24/zenhub-plataforma-de-gestion-de-proyectos-integrada-en-github-ahora-gratis-para-estudiantes/. [Último acceso: 22 septiembre 2021].

[20] G. González, «Miro: una plataforma colaborativa para dibujar en pizarras en tiempo real y con videoconferencias : Genbeta,» 27 Abril 2020. [En línea]. Available: https://www.genbeta.com/herramientas/miro-plataforma-colaborativa-para-dibujar-pizarras-tiempo-real-videoconferencias. [Último acceso: 28 Septiembre 2021].

[21] A. Robledano, «Open Webinars,» 24 Septiembre 2019. [En línea]. Available: https://openwebinars.net/blog/que-es-mysql/. [Último acceso: 17 Julio 2021].

[22] SIVSA, «Git triunfa como el sistema de control de versiones más popular,» *Sivsa,* vol. 1, p. 3, 2019.

[23] C. A. Dorantes, «Platzi,» 2 Agosto 2016. [En línea]. Available: https://platzi.com/blog/arquitectura-laravel/. [Último acceso: 14 Julio 2021].

[24] Laravel - Excel, «laravel-excel,» 2020. [En línea]. Available: https://docs.laravel-excel.com/3.1/getting-started/. [Último acceso: 6 Agosto 2021].

[25] PMOinformatica.com, «7 Técnicas de levantamiento de requerimientos software,» PMOinformatica.com - La oficina de proyectos de informática, 3 agosto 2016. [En línea]. Available: http://www.pmoinformatica.com/2016/08/tecnicas-levantamiento-requerimientos.html. [Último acceso: 23 Noviembre 2020].

[26] M. Civantos, «Tribalyte,» 24 Mayo 2021. [En línea]. Available: https://tech.tribalyte.eu/blog-que-es-una-api-rest. [Último acceso: 20 Julio 2021].

[27] BBVA API Market, «BBVA,» 23 Marzo 2016. [En línea]. Available: https://www.bbvaapimarket.com/es/mundo-api/api-rest-que-es-y-cuales-son-sus-ventajas-en-el-desarrollo-de-proyectos/. [Último acceso: 2 Agosto 2021].

[28] CTES, «Arquitectura REST para el desarrollo de aplicaciones web,» *Revista Electrónica sobre Ciencia, Tecnología y Sociedad,* vol. 8, nº 15, p. 14, Junio 2021.

[29] I. Landa, «Geeks,» 21 Abril 2009. [En línea]. Available: https://geeks.ms/ilanda/2009/04/21/pruebas-unitarias-mocks/. [Último acceso: 3 Agosto 2021].

6 ANEXOS

6.1 Historias de usuario

A continuación, se presentan las historias de usuario que se hicieron para el desarrollo y planificación del proyecto.

	Iniciar sesión
Identificador: HU01	**Usuario:** Todos
Nombre Historia: Iniciar sesión.	
Prioridad en Negocio (Alta/Medio/Baja): Alta	**Riesgo en Desarrollo (Alta/Medio/Baja):** Alta
Iteración asignada: 2	
Responsable (es): Chantal Morales, Israel Vásquez	
Descripción: El usuario podrá iniciar sesión a través de un correo y una contraseña.	
Observación: El estudiante debe estar previamente registrado en la tabla de usuarios dentro de la base de datos.	

	Registrar el plan de titulación
Identificador: HU02	**Usuario:** Estudiante
Nombre Historia: Registrar el plan de titulación.	
Prioridad en Negocio (Alta/Medio/Baja): Alta	**Riesgo en Desarrollo (Alta/Medio/Baja):** Alta
Iteración asignada: 5	
Responsable (es): Chantal Morales, Israel Vásquez	
Descripción: El sistema deberá guardar los datos del plan de titulación que llenó la estudiante.	
Observación: El usuario debe estar previamente registrado y con su sesión activa. El usuario debe haber seleccionado un producto previamente.	

	Editar el plan de titulación
Identificador: HU03	**Usuario:** Estudiante
Nombre Historia: Editar el plan de titulación.	
Prioridad en Negocio (Alta/Medio/Baja): Alta	**Riesgo en Desarrollo (Alta/Medio/Baja):** Alta

Iteración asignada: 3	
Responsable (es): Chantal Morales, Israel Vásquez	
Descripción: El el sistema podrá modificar los datos guardados del plan de titulación.	
Observación: El usuario debe haber guardado los datos del plan de titulación previamente y estar con la sesión activa.	

	Ver el plan de titulación
Identificador: HU04	**Usuario:** Estudiante
Nombre Historia: Ver el plan de titulación	
Prioridad en Negocio (Alta/Medio/Baja): Alta	**Riesgo en Desarrollo (Alta/Medio/Baja):** Alta
Iteración asignada: 2	
Responsable (es): Chantal Morales, Israel Vásquez	
Descripción: El sistema deberá enviar los datos almacenados del plan de titulación.	
Observación: El usuario debe haber guardado los datos del plan de titulación previamente y estar con la sesión activa.	

	Cambiar estado del proyecto
Identificador: HU05	**Usuario:** Todos
Nombre Historia: Cambiar estado del proyecto	
Prioridad en Negocio (Alta/Medio/Baja): Alta	**Riesgo en Desarrollo (Alta/Medio/Baja):** Alta
Iteración asignada: 3	
Responsable (es): Chantal Morales, Israel Vásquez	
Descripción: El sistema deberá cambiar el estado del proyecto de acuerdo con la ruta a la que se apunte.	
Observación: Los usuarios debe estar previamente registrado y con su sesión activa. El usuario debe haber seleccionado un producto previamente.	

	Subir el proyecto de titulación
Identificador: HU06	**Usuario:** Estudiante
Nombre Historia: Subir el proyecto de titulación.	
Prioridad en Negocio (Alta/Medio/Baja): Alta	**Riesgo en Desarrollo (Alta/Medio/Baja):** Alta
Iteración asignada: 5	
Responsable (es): Chantal Morales, Israel Vásquez	

Descripción: el sistema deberá guardar el documento que suba el estudiante.
Observación: El usuario debe haber pasado el estado de aprobación del plan de titulación.

	Editar el proyecto de titulación
Identificador: HU07	**Usuario:** Estudiante
Nombre Historia: Editar el proyecto de titulación	
Prioridad en Negocio (Alta/Medio/Baja): Alta	**Riesgo en Desarrollo (Alta/Medio/Baja):** Alta
Iteración asignada: 3	
Responsable (es): Chantal Morales, Israel Vásquez	
Descripción: El sistema podrá actualizar el campo del proyecto de titulación.	
Observación: El usuario debe haber almacenado un documento antes y estar con la sección activa.	

	Ver observaciones del proyecto
Identificador: HU08	**Usuario:** Estudiante
Nombre Historia: Ver observaciones del proyecto	
Prioridad en Negocio (Alta/Medio/Baja): Alta	**Riesgo en Desarrollo (Alta/Medio/Baja):** Alta
Iteración asignada: 5	
Responsable (es): Chantal Morales, Israel Vásquez	
Descripción: El sistema deberá enviar los datos correspondientes a esos campos de acuerdo con la petición.	
Observación: El usuario debe haber ingresado previamente un documento y éste debe contar con un estado previo.	

	Recibir notificaciones
Identificador: HU09	**Usuario:** Todos
Nombre Historia: Recibir notificaciones	
Prioridad en Negocio (Alta/Medio/Baja): Alta	**Riesgo en Desarrollo (Alta/Medio/Baja):** Media
Iteración asignada: 3	
Responsable (es): Chantal Morales, Israel Vásquez	

Descripción: El sistema deberá enviar notificaciones a modo de correos electrónicos de acuerdo con los cambios de estado.
Observación: El proyecto deberá validar el estado previo y el estado a cambiarse.

	Crear idea de proyecto
Identificador: HU10	**Usuario:** Profesor
Nombre Historia: Crear idea de proyecto	
Prioridad en Negocio (Alta/Medio/Baja): Medio	**Riesgo en Desarrollo (Alta/Medio/Baja):** Medio
Iteración asignada: 3	
Responsable (es): Chantal Morales, Israel Vásquez	
Descripción: El sistema deberá guardar los datos de nuevas ideas de proyecto por parte de los profesores.	
Observación: Las ideas deberán contener el título de la idea, la problemática y la solución.	

	Revisar plan de titulación asignado
Identificador: HU11	**Usuario:** Profesor
Nombre Historia: Revisar plan de titulación asignado	
Prioridad en Negocio (Alta/Medio/Baja): Alta	**Riesgo en Desarrollo (Alta/Medio/Baja):** Media
Iteración asignada: 3	
Responsable (es): Chantal Morales, Israel Vásquez	
Descripción: El sistema deberá devolver la información correspondiente a los planes de titulación asignados al profesor para realizar su revisión.	
Observación: Los proyectos devueltos serán aquellos en los que el profesor participe como director de proyecto.	

	Añadir observaciones al plan de titulación
Identificador: HU12	**Usuario:** Profesor
Nombre Historia: Añadir observaciones al plan de titulación	

Prioridad en Negocio (Alta/Medio/Baja): Alta	**Riesgo en Desarrollo (Alta/Medio/Baja):** Media
Iteración asignada: 5	
Responsable (es): Chantal Morales, Israel Vásquez	
Descripción: El sistema deberá guardar los comentarios realizados por el profesor en cada sección del plan de titulación.	
Observación: En el caso de existir alguna observación, ésta se relaciona con la sección del plan correspondiente.	

	Aprobar el plan de titulación
Identificador: HU13	**Usuario:** Profesor
Nombre Historia: Aprobar el plan de titulación	
Prioridad en Negocio (Alta/Medio/Baja): Alta	**Riesgo en Desarrollo (Alta/Medio/Baja):** Media
Iteración asignada: 2	
Responsable (es): Chantal Morales, Israel Vásquez	
Descripción: El sistema deberá cambiar el estado del proyecto en caso de que éste sea aprobado por el director.	
Observación: El proyecto deberá ser aprobado en caso de que no existan observaciones.	

	Revisar proyecto de titulación asignado
Identificador: HU14	**Usuario:** Profesor
Nombre Historia: Revisar proyecto de titulación asignado	
Prioridad en Negocio (Alta/Medio/Baja): Alta	**Riesgo en Desarrollo (Alta/Medio/Baja):** Media
Iteración asignada: 3	
Responsable (es): Chantal Morales, Israel Vásquez	
Descripción: El sistema deberá devolver la información correspondiente a los proyectos de titulación asignados al profesor para realizar su revisión.	
Observación: Los proyectos presentados deberán estar asignados al profesor y en el estado correcto para su revisión.	

	Añadir observaciones al plan de titulación
Identificador: HU15	**Usuario:** Profesor

Nombre Historia: Añadir observaciones al proyecto de titulación	
Prioridad en Negocio (Alta/Medio/Baja): Alta	**Riesgo en Desarrollo (Alta/Medio/Baja):** Media
Iteración asignada: 5	
Responsable (es): Chantal Morales, Israel Vásquez	
Descripción: El sistema deberá guardar los comentarios realizados por el profesor en cada sección del proyecto de titulación.	
Observación: Los comentarios se guardarán de forma independiente para luego ser consultados por el estudiante.	

	Aprobar el proyecto de titulación
Identificador: HU16	**Usuario:** Profesor
Nombre Historia: Aprobar el proyecto de titulación	
Prioridad en Negocio (Alta/Medio/Baja): Alta	**Riesgo en Desarrollo (Alta/Medio/Baja):** Media
Iteración asignada: 2	
Responsable (es): Chantal Morales, Israel Vásquez	
Descripción: El sistema deberá cambiar el estado del proyecto de titulación cuando el director lo apruebe.	
Observación: El proyecto deberá ser aprobado cuando no existan observaciones.	

	Ver proyectos asignados como parte de la comisión
Identificador: HU17	**Usuario:** Profesor
Nombre Historia: Ver proyectos asignados como parte de la comisión	
Prioridad en Negocio (Alta/Medio/Baja): Alta	**Riesgo en Desarrollo (Alta/Medio/Baja):** Media
Iteración asignada: 5	
Responsable (es): Chantal Morales, Israel Vásquez	
Descripción: El sistema deberá mostrar los proyectos que deberán ser revisados por la comisión.	
Observación: Los proyectos serán mostrados sólo en caso de que el profesor sea parte de la comisión.	

	Calificar proyectos como parte del jurado
Identificador: HU18	**Usuario:** Profesor

Nombre Historia: Calificar proyectos como parte de jurado	
Prioridad en Negocio (Alta/Medio/Baja): Alta	**Riesgo en Desarrollo (Alta/Medio/Baja):** Media
Iteración asignada: 3	
Responsable (es): Chantal Morales, Israel Vásquez	
Descripción: El sistema deberá guardar los comentarios y calificaciones realizados por parte del jurado al proyecto de titulación.	
Observación: El profesor podrá realizar esta acción si está asignado como jurado a ese proyecto.	

	Guardar lista de estudiantes
Identificador: HU19	**Usuario:** Secretaría
Nombre Historia: Guardar lista de estudiantes.	
Prioridad en Negocio (Alta/Medio/Baja): Media	**Riesgo en Desarrollo (Alta/Medio/Baja):** Media
Iteración asignada: 3	
Responsable (es): Chantal Morales, Israel Vásquez	
Descripción: El sistema deberá guardar los datos obtenidos de la lista de estudiantes en la tabla correspondiente.	
Observación: el usuario de estar con la sección activa.	

	Guardar profesores
Identificador: HU20	**Usuario:** Secretaría
Nombre Historia: Guardar Profesores.	
Prioridad en Negocio (Alta/Medio/Baja): Media	**Riesgo en Desarrollo (Alta/Medio/Baja):** Media
Iteración asignada: 3	
Responsable (es): Chantal Morales, Israel Vásquez	
Descripción: El sistema deberá guardar los datos obtenidos los profesores que se hayan ingresado en la tabla correspondiente	
Observación: el usuario de estar con la sección activa.	

	Declarar a un estudiante apto
Identificador: HU21	**Usuario:** Secretaría
Nombre Historia: Declarar a un estudiante apto.	

Prioridad en Negocio (Alta/Medio/Baja): Alta.	**Riesgo en Desarrollo (Alta/Medio/Baja):** Media
Iteración asignada: 2	
Responsable (es): Chantal Morales, Israel Vásquez	
Descripción: El sistema deberá actualizar los campos correspondientes del usuario.	
Observación: el usuario de estar con la sección activa y verificar los estados previos.	

	Asignar profesores a una comisión
Identificador: HU22	**Usuario:** Secretaría
Nombre Historia: Asignar profesores a una comisión.	
Prioridad en Negocio (Alta/Medio/Baja): Alta.	**Riesgo en Desarrollo (Alta/Medio/Baja):** Alta.
Iteración asignada: 5	
Responsable (es): Chantal Morales, Israel Vásquez	
Descripción: El sistema deberá enviar los datos de los profesores registrados y guardar la información de la nueva comisión en la tabla correspondiente.	
Observación: el usuario de estar con la sección activa y poder ver el listado de profesores.	

	Ver la lista de proyectos
Identificador: HU23	**Usuario:** Secretaría
Nombre Historia: Ver la lista de proyectos.	
Prioridad en Negocio (Alta/Medio/Baja): Alta.	**Riesgo en Desarrollo (Alta/Medio/Baja):** Baja.
Iteración asignada: 3	
Responsable (es): Chantal Morales, Israel Vásquez	
Descripción: El sistema deberá enviar toda la lista de proyectos registrados.	
Observación: El usuario de estar con la sección activa.	

	Ver la lista de proyectos
Identificador: HU24	**Usuario:** Dirección
Nombre Historia: Ver la lista de proyectos.	
Prioridad en Negocio (Alta/Medio/Baja): Alta.	**Riesgo en Desarrollo (Alta/Medio/Baja):** Baja.
Iteración asignada: 2	
Responsable (es): Chantal Morales, Israel Vásquez	

Descripción: El sistema deberá enviar toda la lista de proyectos registrados con toda la información.
Observación: El usuario de estar con la sección activa.

	Asignar tribunal a un proyecto
Identificador: HU25	**Usuario:** Dirección
Nombre Historia: Asignar tribunal a un proyecto.	
Prioridad en Negocio (Alta/Medio/Baja): Alta.	**Riesgo en Desarrollo (Alta/Medio/Baja):** Alta.
Iteración asignada: 5	
Responsable (es): Chantal Morales, Israel Vásquez	
Descripción: El sistema deberá guardar la información que se registró al formar un tribunal.	
Observación: el usuario de estar con la sección activa y podrá ver la lista de profesores.	

	Asignar fecha de defensa a un proyecto
Identificador: HU26	**Usuario:** Dirección
Nombre Historia: Asignar fecha de defensa a un proyecto.	
Prioridad en Negocio (Alta/Medio/Baja): Alta.	**Riesgo en Desarrollo (Alta/Medio/Baja):** Alta.
Iteración asignada: 3	
Responsable (es): Chantal Morales, Israel Vásquez	
Descripción: El sistema deberá guardar la información que se registró al encontrar una fecha de defensa con el cruce de horarios de profesores.	
Observación: el usuario de estar con la sección activa y podrá ver la lista de profesores y sus horarios.	

6.2 Product Backlog

En el *Product Backlog* se establecieron todas las tareas correspondientes a cada etapa del desarrollo. Éstas, se etiquetaron de acuerdo con el tipo de usuario al que pertenecía. Se realiza una estimación de tiempo y esfuerzo en cada una y se las planifica en cada iteración del *Sprint Backlog*. A continuación, en la **Fig. 42** se puede observar una sección del tablero de *Product Backlog*.

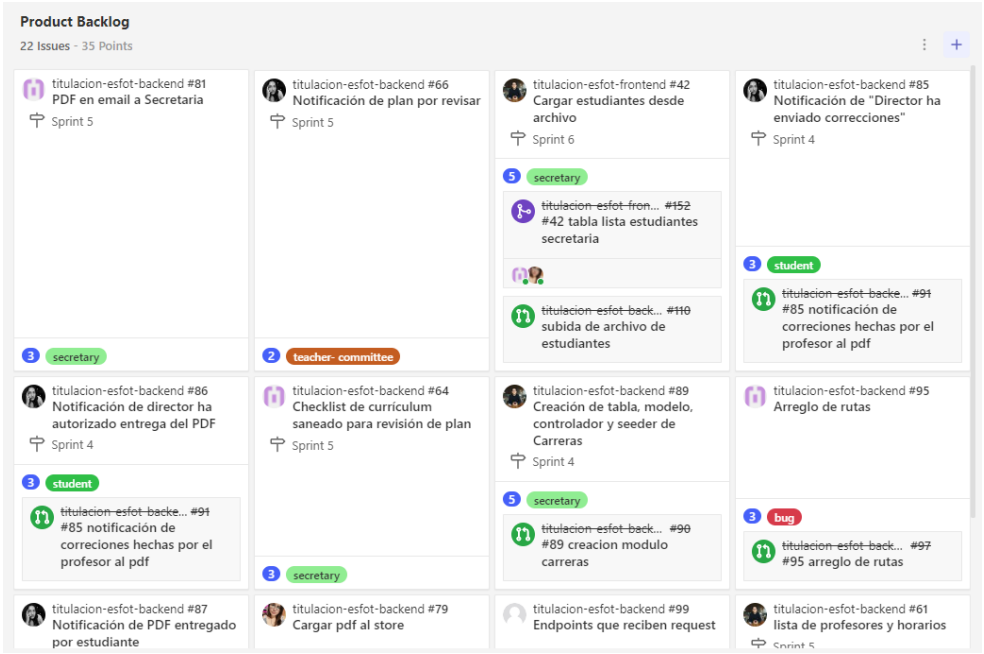


Fig. 42: *Product Backlog*

6.3 *Sprint Backlog*

El *Sprint Backlog* corresponde a la planificación semanal o quincenal que incluye las tareas que continúan en el flujo de trabajo. Los *Sprints* tienen una estimación de tiempo de acuerdo con la complejidad de las tareas y la carga previa de trabajo que se pueda tener. Además, se toman en cuenta los puntos de historia, que vienen a ser los puntos de estimación de complejidad de la tarea, para que cada *Sprint* posea un número similar de puntos de historia y así no se pierda el ritmo de trabajo. A continuación, en las figuras **Fig. 43** y **Fig. 44** se muestra un ejemplo de planificación de *Sprints*, correspondientes al *Sprint* 4 y 5 de desarrollo.

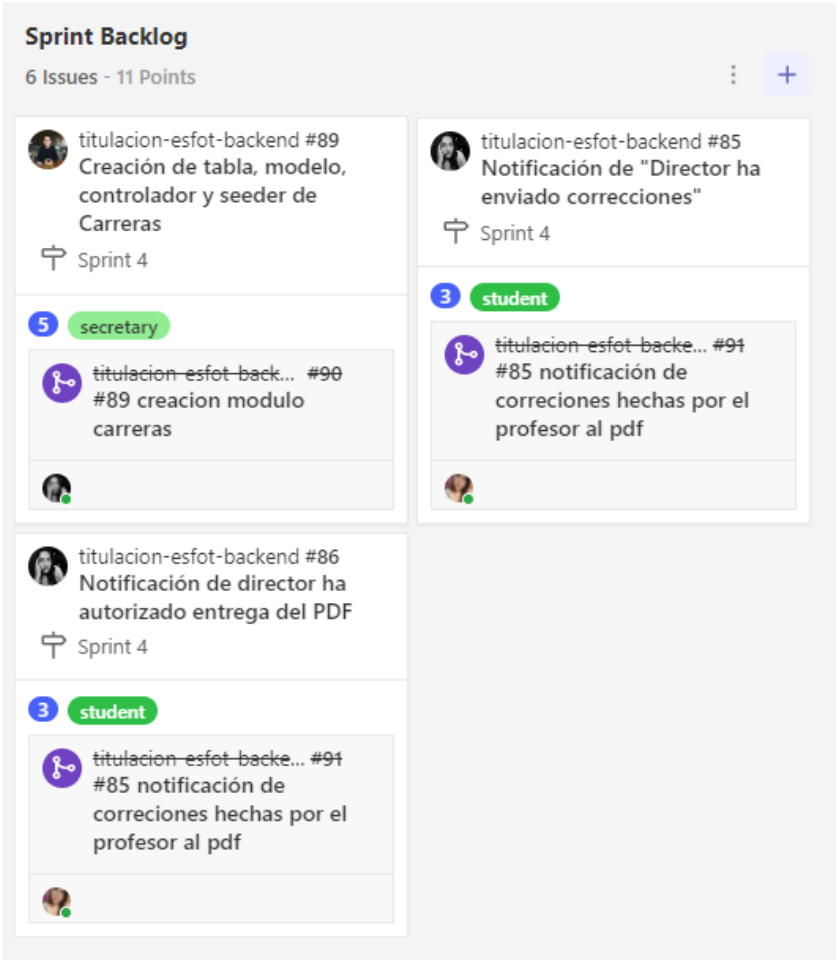


Fig. 43: Tablero *Sprint* 4

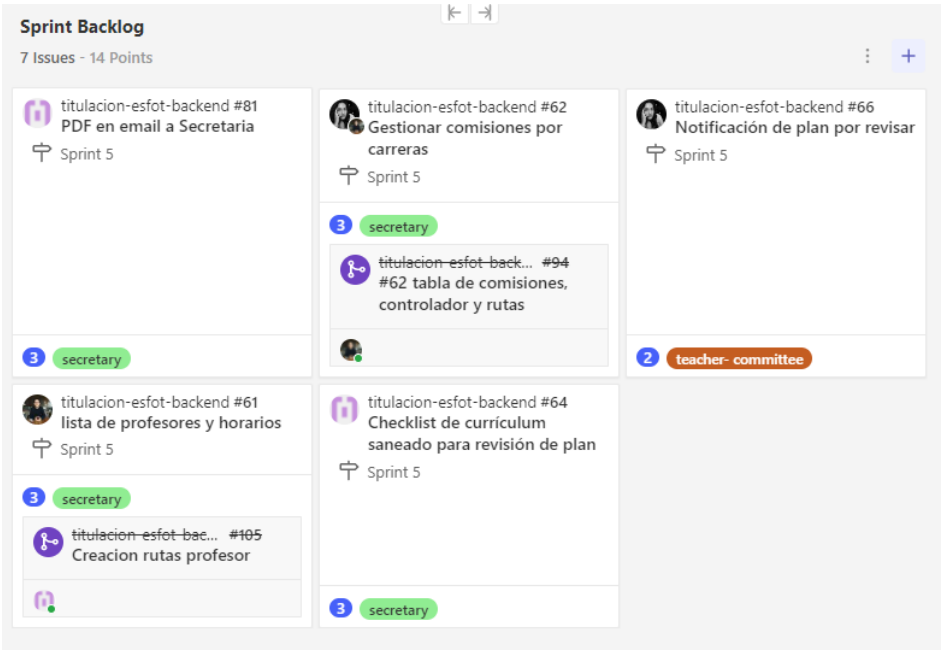


Fig. 44: Tablero *Sprint* 5

6.4 Repositorio del código fuente

El código fuente del proyecto, esta almacenado en un repositorio de *GitHub,* se puede encontrar el mismo a través del siguiente enlace:

https://github.com/chalosalvador/titulacion-esfot-backend.git