ESCUELA POLITÉCNICA NACIONAL

ESCUELA DE FORMACIÓN DE TECNÓLOGOS

DESARROLLO DE UN BACKEND PARA EL CONTROL DE INVENTARIO DEL LOCAL COMERCIAL “PARABRISAS UNIVERSAL”

TRABAJO DE INTEGRACIÓN CURRICULAR PRESENTADO COMO REQUISITO PARA LA OBTENCIÓN DEL TÍTULO DE TECNÓLOGO SUPERIOR EN DESARROLLO DE SOFTWARE

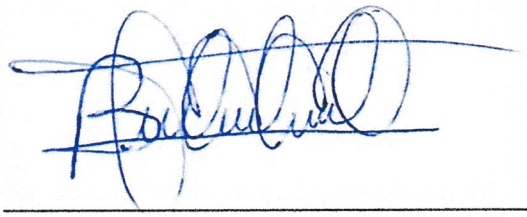
KEVIN FERNANDO BENAVIDES ROBALINO

DIRECTOR: ING. JUAN PABLO ZALDUMBIDE PROAÑO

Quito, septiembre 2022

CERTIFICACIONES

Yo, KEVIN FERNANDO BENAVIDES ROBALINO declaro que el trabajo de integración curricular aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

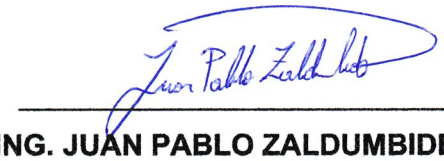


KEVIN FERNANDO BENAVIDES ROBALINO

kevin.benavides@epn.edu.ec

ferkevin71997@hotmail.com

Certifico que el presente trabajo de integración curricular fue desarrollado por KEVIN FERNANDO BENAVIDES ROBALINO, bajo mi supervisión.



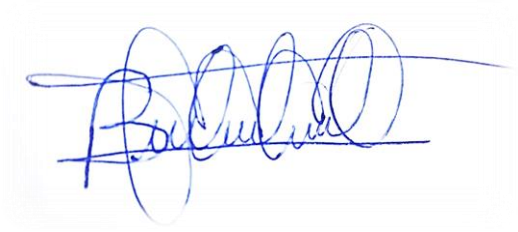
ING. JUAN PABLO ZALDUMBIDE PROAÑO

DIRECTOR

juan.zaldumbide@epn.edu.ec

DECLARACIÓN DE AUTORÍA

A través de la presente declaración, afirmamos que el trabajo de integración curricular aquí descrito, así como el (los) producto(s) resultante(s) del mismo, son públicos y estarán a disposición de la comunidad a través del repositorio institucional de la Escuela Politécnica Nacional; sin embargo, la titularidad de los derechos patrimoniales nos corresponde a los autores que hemos contribuido en el desarrollo del presente trabajo; observando para el efecto las disposiciones establecidas por el órgano competente en propiedad intelectual, la normativa interna y demás normas.



KEVIN FERNANDO BENAVIDES ROBALINO

DEDICATORIA

Quiero dedicar este proyecto a mí, por mantenerme constante mentalmente durante todo el proceso a pesar de que en etapas su renuncia era una opción, por mantener vivo el entusiasmo de saber que se podía lograr sin importar las circunstancias y por haber sabido llevar cada contratiempo de manera serena hasta solventarlo.

Esta dedicatoria también es por ser considerado frente a quienes me daban consejos y seguirlos, por pensar en el fracaso como algo lejano.

Apreciada valentía.

BENAVIDES ROBALINO KEVIN FERNANDO

AGRADECIMIENTO

Agradezco a la vida por brindarme las circunstancias oportunas para pensar en proponerme esta meta y aún más por brindarme las herramientas y capacidades necesarias para poder lograrlo.

A mi mama por exigirme sutilmente que ya era hora de culminarlo, por tener su esperanza en mi desde el inicio de esta etapa y nunca habérselo cuestionado.

A mi papa que desde el inicio me menciono la frase que describe a la Universidad desde mi perspectiva, “Es una carrera de resistencia mas no de velocidad”, y me sirvió para acondicionarme mentalmente a no decaer en ninguna dificultad.

A mi abuelita por ayudarme en lo que más ha podido y alegrarse al ir escuchando cada progreso.

A mis hermanos, Hubert y Steven que fueron esa pequeña motivación constante y por brindarme su ánimo cada vez que estaba en sus manos.

A mi sobrino Mateo, que llego de improviso para dar alegría y una razón para mantenerse firme en el camino.

A los docentes de la ESFOT por ser exigentes y firmes para formar profesionales de calidad, también por ser comprensivos y benevolentes durante etapa de la pandemia.

A mis amigos que con sus logros académicos y personales iban haciéndome afianzar más a mi meta hasta conseguirla.

Finalmente, a la Escuela Politécnica Nacional y la Escuela de Formación de Tecnólogos por considerarme para ser su alumno y ofrecerme sus instalaciones para llevar a cabo el todo este ciclo pacífico de aprendizaje.

BENAVIDES ROBALINO KEVIN FERNANDO

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN

El proyecto surge con la idea de automatizar el trato de la información, para hacerla automática, dado que el método regular adoptado por locales comerciales de este tipo son nada más escritos en cuadernos seccionados por entidades en este caso por pedidos, clientes, proveedores y productos, de tal manera que la búsqueda de un elemento en específico puede tardar varios minutos, causando molestias en los clientes que consideran la calidad del servicio según el tiempo de respuesta del local.

Este documento explica el proceso del desarrollo de un backend para el control del inventario del local comercial “Parabrisas Universal”, siguiendo la metodología SCRUM, misma que aporta beneficios tales como: obtención de requerimientos, flexibilidad entrono al cambio, roles definidos y reuniones con el usuario, esto permite que el desarrollo concluya en los tiempos determinados, además se utilizara el framework “Laravel”, el lenguaje “PHP” con plantillas “Blade”, un sistema gestor base de datos en “MySQL” (SQL), el sistema desarrollado tiene la posibilidad de hacer registros e inicios de sesión por partes de los administradores, vendedores y clientes, con la opción de poder recuperar su contraseña enviando un enlace a su correo registrado.

Por otro lado, tiene la capacidad de gestionar el inventario, pedidos, proveedores, comentarios siendo tratado esta información de forma automática en todo el entorno, es decir, permitirá el guardado y conservación de todos los registros ingresados permitiendo su actualización en cualquier momento y su eliminación, que será la acción de cambiar de estado a inactivo o indisponible, con todo esto podrá acceder a la información requerida con tan solo una búsqueda por elemento.

PALABRAS CLAVE: Scrum, inventario, backend, SQL, Laravel, PHP.

ABSTRACT

The project arises with the idea of automating the treatment of information, to make it automatic, given that the regular method adopted by commercial premises of this type is nothing more than writing in notebooks sectioned by entities, in this case by orders, customers, suppliers and products. , in such a way that the search for a specific item can take several minutes, causing inconvenience to customers who consider the quality of the service according to the response time of the premises.

This document explains the process of developing a backend to control the inventory of the “Universal Windshield” commercial premises, following the SCRUM methodology, which provides benefits such as: obtaining requirements, flexibility around change, defined roles and meetings with the user, this allows the development to conclude in the determined times, in addition, the "Laravel" framework will be used, the "PHP" language with "Blade" templates, a database management system in "MySQL" (SQL), the system developed You have the possibility of registration and login by administrators, sellers and customers, with the option of being able to recover your password by sending a link to your registered email.

On the other hand, it has the ability to manage inventory, orders, suppliers, comments, this information being treated automatically throughout the environment, that is, it will allow the saving and conservation of all the entered records, allowing them to be updated at any time and elimination, which will be the action of changing the status to inactive or unavailable, with all this you will be able to access the required information with just one search per item.

KEYWORDS: Scrum, inventory, backend, SQL, Laravel, PHP.

1 DESCRIPCIÓN DEL COMPONENTE DESARROLLADO

Hoy en día, con las tecnologías dispuestas para hacer un mundo más simple en cuanto a información automática se refiere, aún existen locales comerciales que prefieren la adopción de los estilos antiguos y algo normalizados, que muestra alguna forma de miedo a las tecnologías y a la evolución, dentro de esto está el trato de su información en cuadernos, carpetas o cubículos que puede llegar a ser estructurada y ordenada pero en sistemas medianos y grandes no garantiza una administración optima y veloz para brindar al cliente que por lo general suelen ser impacientes, y todo esto es justo lo que resuelve la informática, un conjunto de técnicas, programas, códigos e implementaciones que hacen el trato de la información automática.

En este sentido este componente desarrollado como trabajo de integración curricular solventa los problemas encontrados en la administración de los locales comerciales, habiendo desarrollado un backend desplegado en una herramienta web para que la interfaz de usuario pueda ser accedida a través de un navegador web y su uso sea solo mediante esta aplicación, con la herramienta Microsoft Visual Studio Code, con la arquitectura MVC, el framework Laravel, el lenguaje de programación PHP y el sistema gestor de bases de datos en (SQL) MySQL, con la finalidad de generar un ambiente para controlar y gestionar los registros de cada una de las entidades establecidas que permitirán mantener la integridad y la consistencia de los datos que se ingresen, se modifiquen, se visualicen o se cambien de estado, y a todo esto un seguimiento de control por parte de quienes lo administraran.

En otro apartado se encuentra la administración de usuarios que de manera concreta lo que permite hacer es registrar usuarios vendedores con sus credenciales, así mismo para clientes y con esto garantizar su inicio de sesión en la aplicación para que pueda administrar todo el flujo del negocio y como todo sistema de seguridad de acceso, permite la recuperación de contraseña con envíos de correo a sus correos registrados.

Con esto se planea llevar un mejor control de inventario, proveedores, clientes suscritos a todo el local comercial y ayudar con la toma de decisiones alrededor de la tónica de este local comercial como lo es en: determinar que productos se debe hacer una adquisición porque está próximo a agotarse, para lo cual también existe un apartado, determinar que proveedores brindan un aprovisionamiento más óptimo para el inventario, determinar que cliente tienen en proceso un pedido, y si está próximo a ser atendido, determinar el nivel de clientes ha tenido por temporada, incluso determinar la calidad de servicio brindado mediante los comentarios de sus clientes.

1.1 Objetivo general

Desarrollar un backend para automatizar el proceso de control y gestión de inventario del local comercial “Parabrisas universal”.

1.2 Objetivos específicos

1. Realizar la recopilación de requerimientos del sistema.
2. Diseñar la arquitectura y la base de datos para el sistema.
3. Programar una API REST funcional para que aplicaciones de desarrollo frontend o móvil puedan acceder de manera externa a este sistema.
4. Implementar la lógica de negocio en el backend con el framework Laravel.
5. Ejecutar pruebas unitarias al API REST para comprobar su funcionamiento.
6. Desplegar el backend en un servidor web público.

1.3 Alcance

El presente proyecto plantea el desarrollo de un backend para que el gerente y sus colaboradores puedan gestionar de manera adecuada el inventario que tiene disponible, optimizar la comunicación con el cliente y que puedan percibir los productos disponibles.

Por su parte, el perfil administrador tiene la posibilidad de gestionar el inventario, pedidos, clientes y proveedores. Además, que el perfil vendedor tiene la posibilidad de gestionar el inventario y pedidos para que el perfil administrador pueda revisar, mientras que el perfil usuario podrá realizar pedidos, programar un mantenimiento o servicio y obtener información de productos por la página web. Es por esta razón que en este proyecto de integración curricular se propone implementar un backend, capaz de brindar una solución que permite optimizar la interacción entre vendedor y cliente, con esto agilizar la comercialización de estos productos por medio de la tecnología.

Por otro lado, se garantizará la integridad, consistencia y seguridad de los datos con el backend para tener un mejor control de los datos que serán presentados por parte del Frontend y la utilización de herramientas de desarrollo actuales y modernas, teniendo en cuenta la escalabilidad y robustez.

La metodología para usar en este proyecto es la metodología ágil SCRUM, debido a que se trata de un equipo pequeño y para cumplir con los objetivos propuestos se considera el tiempo necesario, también se toma en cuenta las reuniones periódicas con el cliente

para avanzar de buena condición el proyecto, y de ser el caso realizar alguna modificación en el proyecto, se plantea un diseño simple y sencillo pero funcional [1].

Además, la integración de una metodología ágil de desarrollo denominada Scrum para el cumplimiento de los objetivos propuesto, un patrón arquitectónico, una serie de pruebas para verificar el correcto funcionamiento del backend y una etapa final para realizar el despliegue a producción respectivamente.

1.4 Marco Teórico

Sistema de gestión de inventario

El inventario es todo el conjunto de materiales o artículos que dentro de un negocio tiene el propósito de brindar a los clientes con fines lucrativos, por consecuente el tratado de este apartado debe ser una labor con su real importancia, denominado gestión de inventario, el cual se lleva a cabo desde su fabricación hasta los almacenes, y de estas instalaciones hasta el punto de venta. La finalidad de la gestión de inventario es en lo posible tener los productos correctos en el lugar adecuado y en el momento preciso y todo esto requiere un modo de visualizar que artículos se mantiene en bodega, un limite de productos para realizar pedidos y reabastecer, incluso clasificar el espacio por categorías para mantener un orden en el proceso. [1]

Dicho esto, el control de inventario en determinante para la eficacia de la logística de la empresa que va a implementarlo y una opción considerable para quienes lo tengan en mente.

Backend

Existe conceptos que van de la mano con el concepto de Backend, haciendo alusión al Frontend, ambos conceptos coexisten en este ámbito, dependiendo en gran medida el uno del otro, entonces la explicación del Frontend hará más comprensible el apartado de Backend.

Explicado de manera sencilla, el Frontend de una aplicación es el apartado que se dedica al diseño de todo lo que el usuario tiene a su vista, aquí se incluyen las imágenes, botones, menús, transiciones, etc. Se la puede denominar la capa más superficial de la aplicación.

En esta perspectiva, el Backend entra en juego para que el Frontend tenga sentido, esta toma acción sobre funcionalidades que el usuario final no puede ver, controlando permisos, enviando datos, y guardando datos en la base de datos establecida,

complementando la navegación y haciendo posible las interacciones del usuario con los datos disponibles. [2]

MySQL

MySQL es el sistema de gestión de bases de datos relacional más extendido en la actualidad al estar basada en código abierto. Desarrollado originalmente por MySQL AB, fue adquirida por Sun MicroSystems en 2008, entonces MySQL es un sistema de gestión de bases de datos que cuenta con una doble licencia. Por una parte, es de código abierto, pero por otra, cuenta con una versión comercial gestionada por la compañía Oracle. [3]

PHP

PHP es un lenguaje de programación destinado a desarrollar aplicaciones para la web y crear páginas web, favoreciendo la conexión entre los servidores y la interfaz de usuario, entre los factores que hicieron que PHP se volviera tan popular, se destaca el hecho de que es de código abierto, esto significa que cualquiera puede hacer cambios en su estructura. En la práctica, esto representa dos cosas importantes:

- es de código abierto, no hay restricciones de uso vinculadas a los derechos. El usuario puede usar PHP para programar en cualquier proyecto y comercializarlo sin problemas.
- está en constante perfeccionamiento, gracias a una comunidad de desarrolladores proactiva y comprometida. [4]

Heroku

Heroku es una plataforma como servicio (PaaS) de computación en la nube la cual inicialmente fue desarrollada para soportar aplicaciones en lenguaje Ruby, pero posteriormente ha extendido su soporte a numerosos lenguajes y aplicaciones que ofrecen una mayor flexibilidad y agilidad a la hora de construir aplicaciones en este entorno de desarrollo.

Heroku es una PaaS completamente gestionada por lo que toda la complejidad subyacente como la respuesta a fallos, monitorización de vulnerabilidades de seguridad, planificación de ampliaciones y ampliaciones reales están gestionados por la propia plataforma. [5]

phpMyAdmin

PhpMyAdmin es una aplicación web que sirve para administrar bases de datos MySQL de forma sencilla y con una interfaz amistosa. Se trata de un software muy popular basado en PHP. La ventaja de usar una aplicación web es que nos permite conectarnos con servidores remotos, a los cuales no siempre se puede acceder usando programas de interfaz gráfica.

Para usar phpMyAdmin simplemente necesitas subir el conjunto de archivos PHP que componen la aplicación a un servidor web, configurar con los datos de acceso a MySQL y empezar a administrar las bases de datos. Con phpMyAdmin puedes hacer todo tipo de operaciones, desde la creación y borrado de bases de datos a la administración de las tablas (crear, modificar y eliminar) y, por supuesto, de sus propios datos. [6]

Laravel

Laravel es un framework PHP. Es uno de los frameworks más utilizados y de mayor comunidad en el mundo de Internet, resulta bastante moderno y ofrece muchas utilidades potentes a los desarrolladores, que permiten agilizar el desarrollo de las aplicaciones web.

Laravel pone énfasis en la calidad del código, la facilidad de mantenimiento y escalabilidad, lo que permite realizar proyectos desde pequeños a grandes o muy grandes. Además, permite y facilita el trabajo en equipo y promueve las mejores prácticas. [7]

2 METODOLOGÍA

La metodología incluyendo la “ágil”, comúnmente se la confunde con una serie de indicaciones sobre qué hacer precisamente durante el desarrollo de software, pero en realidad la metodología se la debe manejar como una forma de pensar en la colaboración y los flujos de trabajo, y describe un conjunto de valores que guían nuestras decisiones con respecto al rol que tenemos y a la manera en la que lo ejecutamos.

Por ello, la metodología específicamente hablando de las agiles, buscan proporcionar en poco tiempo una reducida porción de software en funcionamiento para crear validez real frente al cliente, manteniendo así su enfoque flexible y armonía en el trabajo en equipo para brindar mejoras constantes. [8]

2.1 Metodología de Desarrollo

Scrum es una metodología ágil y flexible para gestionar el desarrollo de software, cuyo principal objetivo es maximizar el retorno de la inversión para su empresa. Se basa en construir primero la funcionalidad de mayor valor para el cliente y en los principios de inspección continua, adaptación, autogestión e innovación. [9]

Roles

En Scrum, el equipo se focaliza en construir software de calidad. La gestión de un proyecto Scrum se centra en definir cuáles son las características que debe tener el producto a construir (qué construir, qué no y en qué orden) y en vencer cualquier obstáculo que pudiera entorpecer la tarea del equipo de desarrollo.

El equipo Scrum está formado por los siguientes roles:

Scrum master: Persona que lidera al equipo guiándolo para que cumpla las reglas y procesos de la metodología. Gestiona la reducción de impedimentos del proyecto y trabaja con el Product Owner para maximizar el desempeño.

Product owner (PO): Representante de los accionistas y clientes que usan el software. Se focaliza en la parte de negocio y él es responsable del proyecto (entregar un valor superior al dinero invertido). Traslada la visión del proyecto al equipo, formaliza las prestaciones en historias a incorporar en el Product Backlog y las re-prioriza de forma regular.

Development Team: Grupo de profesionales con los conocimientos técnicos necesarios y que desarrollan el proyecto de manera conjunta llevando a cabo las historias a las que se comprometen al inicio de cada sprint. [10]

En la **TABLA I** presenta la distribución del equipo scrum, conformado por 3 miembros con sus respectivos roles.

TABLA I: Roles del proyecto

Rol	Integrante
Product Owner	Sr. Stalin Robalino
Scrum Master	Ing. Juan Pablo Zaldumbide
Development Team	Sr. Kevin Benavides

Artefactos

Dentro del marco de Scrum se denomina Artefactos a las herramientas funcionales que se producen como resultado de la adaptación de Scrum. Existen 3 artefactos o herramientas Scrum principales: Product Backlog, Sprint Backlog y el Incremento. [11]

Entonces tenemos estructuradas las bases para desplegar el proyecto, sin embargo, existen artefactos más allá de los principales que nos ayudaron con funciones específicas y de manera paulatina, en el desarrollo del proyecto.

Recopilación de Requerimientos

La recopilación de requisitos es una sucesión que radica en determinar las necesidades precisas del proyecto de principio a fin. Este proceso es dictado para la fase inicial del proyecto, pero se mantiene por todo el cronograma del proyecto. [12]

Las funcionalidades que se van a poner en efecto en el proyecto se describen inicialmente en una tabla de recopilación de requerimientos. La **TABLA II** presenta un fragmento de la estructura utilizada para la tabla de recopilación de requerimientos.

La tabla completa está en la sección ANEXOS

TABLA II: Fragmento de tabla de recopilación de requerimientos.

RECOPILACION DE REQUERIMIENTOS			
ID. Requisito	**Nombre del requisito**	**Descripción del requisito**	**Usuario**
RR-001	Ingreso al sistema	El sistema debe regir el acceso con las credenciales correctas de cada administrador y vendedor.	Administrador/ Vendedor
RR-002	Control de acceso	El sistema debe permitir ver el acceso de los usuarios "Vendedores".	Administrador

Historias de Usuario

Las historias de usuario son fracciones con descripciones de las funcionalidades que pide el cliente, en su redacción se debe concretar y describir su rol, la funcionalidad y el resultado esperado en una frase sencilla.

En ocasiones se debe acompañarla con criterios de aceptación, con un límite de 4 por historia, de debe redactar de tal manera que se brinde un contexto, el evento y el proceder esperado frente a este evento. [13]

De manera que este proceso es incremental, las historias de usuario es un paso más adelante de la recopilación de requerimientos para lo cual utilizamos la estructura presentada en la **TABLA III**.

El resto de las historias de usuario se pueden encontrar en la sección de **ANEXOS** apartado

TABLA III: Historia de usuario Nro. 3: Buscar artículo.

HISTORIA DE USUARIO	
Identificador (ID): HU-003	**Usuario:** Administrador/Vendedor
Nombre de historia: Buscar articulo	
Prioridad de negocio: Alta	**Riesgo en desarrollo:** Alto
Iteración asignada: 1	
Responsable: Kevin Benavides	

Descripción: Quiero poder buscar un artículo específico dentro de la lista de productos disponibles.
Observaciones: El Administrador/Vendedor podrá buscar un artículo específico dentro de la lista de productos disponibles.

Product Backlog

Dicho de una manera, es el catálogo que contiene todas las implementaciones y básicamente las funcionalidades que el proyecto debe efectuar creado a partir de la *recopilación de requerimientos* y las *historias de usuario*, se puede decir que es el paso más concreto de los *artefactos*, ya que va independizándose cada módulo hasta ser separada de forma independiente cada tarea, también se añade que el único capaz de la manipulación y organización de este artefacto es explícitamente el *Product Owner*. En el apartado de anexos se encuentra la tabla del *Product Backlog*.

Sprint Backlog

Llamado así a la lista de tareas que se debe realizar en un determinado *sprint* o iteración, que deben entregarse para ser considerada finalizada. El equipo de desarrollo es lógicamente el encargado de realizar su implementación correspondiente, el *Scrum Master* decide que tareas del *Product Backlog* estarán presentes en el *Sprint Backlog*, guiándose de las necesidades que priman para el *Product Owner*. En el apartado # de la sección Anexos se encuentra la tabla del Sprint Backlog.

2.2 Diseño de la arquitectura

Se debe considerar la opción más eficiente para que el problema planteado pueda controlarse y satisfacerse con patrón de arquitectura elido. A continuación, se especifica la arquitectura que se emplea en el proyecto.

Patrón arquitectónico Modelo Vista Controlador (MVC)

La arquitectura MVC propone, independientemente de las tecnologías o entornos en los que se base el sistema a desarrollar, la separación de los componentes de una aplicación en tres grupos (o capas) principales: el *modelo*, la *vista*, y el *controlador*, y describe cómo se relacionarán entre ellos para mantener una estructura organizada, limpia y con un acoplamiento mínimo entre las distintas capas. [14]

Modelo: es una representación de los datos y entidades usados por el sistema que de cierta manera se transformarán en clases y mantendrán un estado consistente e integro la toda la información.

Vista: es la capa visible para el usuario, también denominada interfaz de usuario y su función es mostrar la información solicitada y la interacción de datos.

Controlador: es el intermediario entre el usuario y el sistema, o así mismo entre vista y modelo, funciona detectando que acción realiza el usuario en la vista, y coordina que datos del modelo y bajo que circunstancias puede acceder a esta información.

La **Fig. 1** contiene la representación del patrón arquitectónico y las herramientas que permiten su desarrollo clasificándose según el patrón MVC en su respectiva categoría.

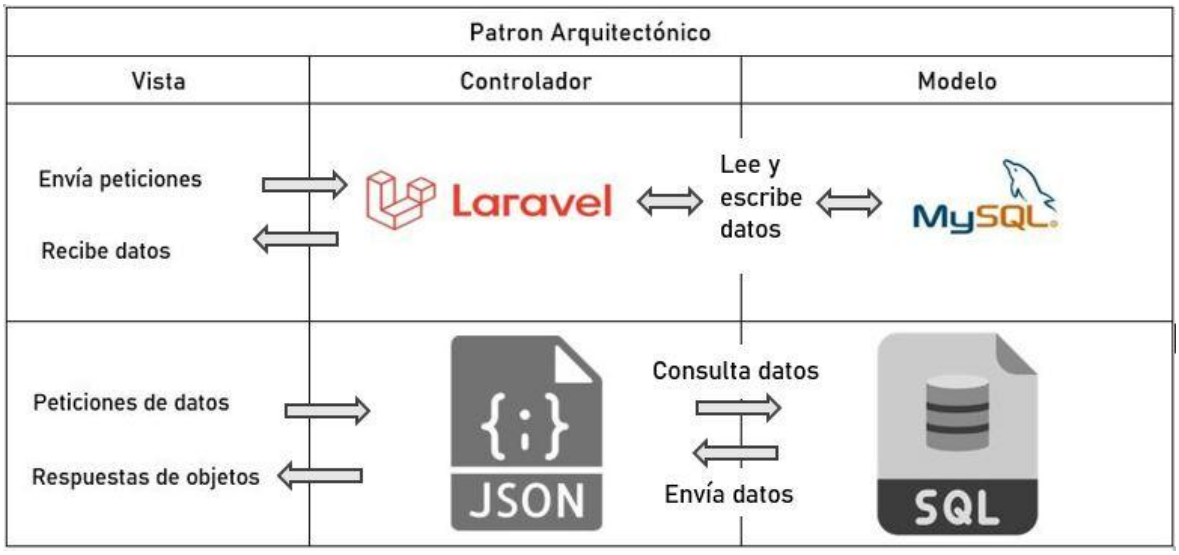


Fig. 1: Patrón arquitectónico de la API REST

2.3 Diseño de la base de datos

En *Laravel* podemos hacer uso de un *ORM* llamado *Eloquent*, un ORM es un *Mapeo Objeto-Relacional* por sus siglas en inglés (Object-Relational mapping), que es una forma de mapear los datos que se encuentran en la base de datos almacenados en un lenguaje de script SQL a objetos de PHP y viceversa, esto surge con la idea de tener un código portable con el que no tengamos la necesidad de usar lenguaje SQL dentro de nuestras clases de PHP. [15]

Eloquent hace uso de los Modelos para recibir o enviar la información a la *base de datos*. [15]

De esta manera se instauraron las tablas de la base de datos en *MySQL* y fueron creadas a través de configuraciones con *Eloquent*, llamándolas a estas migraciones, por otro lado, también se configuraron los *Seeder* y los *Factories* para poder trabajar con datos ficticios hasta su despliegue respectivo.

2.4 Herramientas de desarrollo

Las herramientas para el desarrollo del presente proyecto son en base a la arquitectura del proyecto. La tabla 3 muestra las herramientas utilizadas en el desarrollo.

TABLA I: Herramientas para el desarrollo del Backend.

Herramienta		Justificación
Laravel	*Laravel* es un framework popular que usa el lenguaje PHP, ayuda a la elaboración de aplicaciones web a un nivel que permite personalizaciones con buena calidad. Es un framework bastante moderno que ofrece muchas utilidades a los desarrolladores que agilizan su propio desarrollo.[7]	Laravel y su uso permiten que la capa de acceso y manejo de datos sea optimo del lado del servidor y ventajoso para el cliente porque recibirá sus consultas en el formato más ligero actualmente llamado JSON.
MySQL	MySQL es un sistema gestor de base de datos relacional basado en código abierto, muy popular en la actualidad. Una funcionalidad evidente es el uso de base de datos relaciones, que se interconectan entre si para mantener una armonía y consistencia de datos. [3]	Esta base de datos permite las opciones comunes de un SGBD como: creación de tablas, consultas y almacenamiento de datos y en trato con todas sus relaciones a través de Eloquent acorde a sus expectativas.
Composer	Composer es un gestor de paquetes para el lenguaje PHP que tiene su propio estándar para gestionar, descargar e instalar dependencias y librerías, funciona de igual manera que NPM en Node.js y Bandler en Ruby. Generalmente esta es la solución optima cuando se trabaja con un proyecto que requiere un buen número de fuentes externas. En si su función es descargar de manera automática todas estas dependencias. [16]	Este sistema de gestor de paquetes llega a ser muy útil, en proyectos de este tipo donde necesitamos algunas dependencias, nos permite instalar librerías de manera rápida y sencilla, para poder enfocarnos con rapidez en la lógica de negocio.

Postman	Postman es una herramienta que ayuda a chequear y analizar las peticiones de APIs de tipo REST propio o de terceros de manera sencilla y ahora tiene una aplicación de escritorio con utilidades más completas. [17]	Esta herramienta permite comprobar cómo se envían los datos, como llegan o almacenan a través de la API con los endpoints programados en el backend desarrollado y ver que su funcionamiento sea el correcto y acorde a su función adjudicada.
Git	Git es sistema de control de versiones, dedicado a mantener su control, ya sea para unir ramas, tratar cada rama por separado hasta lograr su propósito o eliminar ramas que no lograron su cometido. También ayuda su función cuando se trata de trabajo en equipo, permite ir comprobando el trabajo de cada participante y comparando con su versión anterior para consolidar su avance. [18]	Git nos brinda la posibilidad de controlar las versiones del proyecto, realizar cambios, permitir cambios o actualizaciones y administrar que ramas serán unificadas para el despliegue de los proyectos.
GitHub	GitHub es la funcionalidad de Git en un servicio en la nube, permite alojar código para su versionamiento, con esto permitir la colaboración de desarrolladores a traves de una aplicación web, facilitando el acceso a esta herramienta Git. [19]	Sin mucha competencia reconocida en el mercado, GitHub es la mejor interfaz gráfica propia de Git, es decir, las mismas funciones de Git se lo puede hacer en esta herramienta con una interfaz más intuitiva.
Heroku	Heroku es una plataforma como servicio en la nube, enfocada en el despliegue de aplicaciones web, que permite su alojamiento sin casi importar el lenguaje usado para la creación, tiene utilidades enlazadas a bases de datos propias o externas para no excluir proyecto este apartado y ampliando su nicho. [20]	Se considero sus bondades por mantener un plan gratuito con las funcionalidades suficientes para subir este proyecto a producción, a pesar de existir otras en el mercado, su función ser albergar el proyecto de manera permanente para brindar la información requerida por los administradores
Miro	Es una aplicación para desarrollar flujos de trabajo en equipo de forma remota a través de una pizarra virtual infinita. [21]	Esta herramienta nos da la posibilidad de crear los diagramas conceptuales más conocidos y populares en un espacio infinito, con colaboración de amigos o conocidos.

3 RESULTADOS

En la presente sección se encuentra de forma detallada un resumen de las actividades de cada uno de los Sprints adjunto con las ilustraciones y pruebas de la API.

3.1 Sprint 0. Planificación del proyecto

Como a cada proyecto le pertenece su etapa de inicio, este Sprint se realiza para la configuración del ambiente de desarrollo logrando establecer la estructura del proyecto, enfocándose en los requerimientos, principalmente de roles administradores y de forma estándar de usuarios cliente. De esta manera se crea un plan de las etapas para que se mantenga sistemático.

Los objetivos del Sprint inicial son:

- Estructuración del proyecto.
- Levantamiento de requerimientos.
- Elaboración del *Product Backlog*.
- Creación del proyecto de *Laravel*.

Estructuración del proyecto

Para poder generar una guía de flujo para el sistema es necesario separar por roles los procesos que tendrán sin profundizar los permisos de los recursos de la *API* del sistema de gestión de inventario, enfatizando en su manera de autenticarse, basado en los diagramas de flujo, creado por la herramienta Miro [21] que se muestra en la **Fig. 2** que concreta quienes son los actores del proceso y su flujo según su determinado rol.

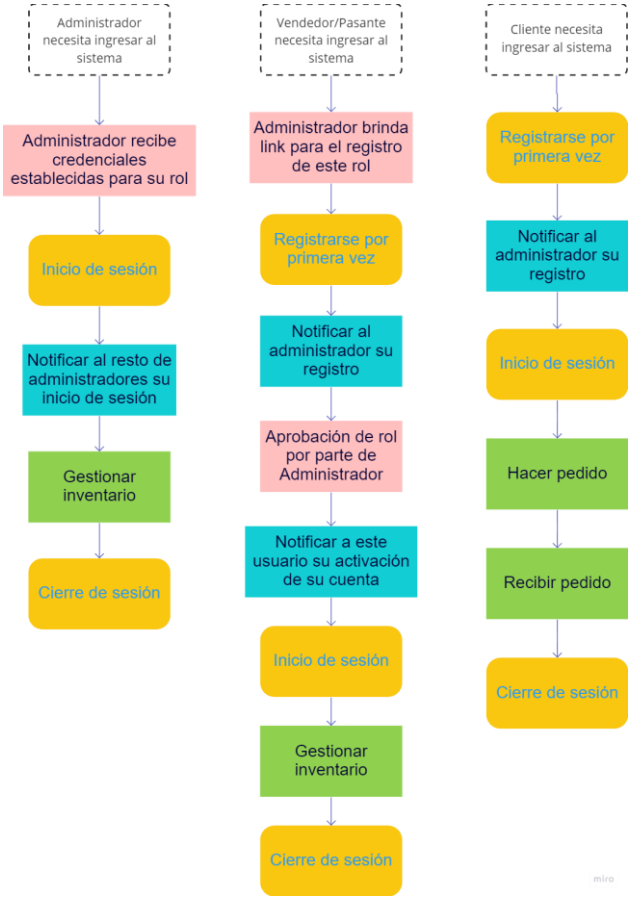


Fig. 2: Diagrama de flujo sobre el proceso por cada rol de usuario.

Levantamiento de requerimientos

Para el levantamiento de requerimientos del sistema de gestión de inventario del local comercial “Parabrisas Universal”, se tuvo que poner en contacto con el gerente propietario de dicho local, estableciendo pequeñas reuniones para detallar en conjunto necesidades que deben ser solventadas por parte de esta parte de sistema (backend) que se encarga de la lógica del negocio y por otra parte, generar historias de usuario para instaurar funcionalidades desde todas las perspectivas inmersas en el sistema de negocio, desde la necesidad de un cliente de buscar y adquirir un elemento hasta concretar la venta incluyendo la búsqueda del elemento por el inventario actual o encontrándolo en sus proveedores.

Elaboración del *Product Backlog*

Para la construcción de este *artefacto* guía se establecen etapas o Sprints que deben ser cumplidas en el tiempo definido acorde a su complejidad y su demanda, clasificándolos en módulos y funciones que deberán irse integrando al repositorio para llevar un control

sobre las versiones y los tiempos en los cuales se finaliza cada versión, y así controlar si se mantiene lo planificado.

Creación del proyecto de *Laravel*

Para la creación del proyecto y su configuración, se procedió a instalar o verificar su instalación de Visual Studio Code para la parte de desarrollo, Composer para la gestión de dependencias, MySQL Workbench para la gestión de base de datos, Postman para las pruebas de rutas, y con esto se inicializo el proyecto con los comandos respetivos siguiendo los parámetros del framework Laravel que nos brinda la base de directorios como muestra la **Fig. 3.**

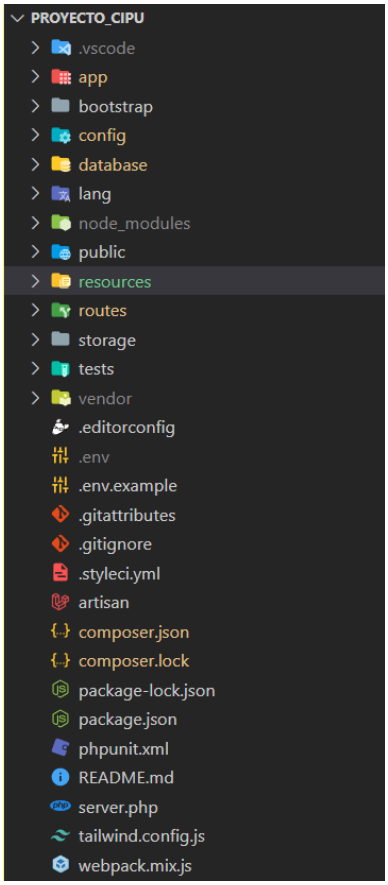


Fig. 3: Estructura del proyecto de Laravel.

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

Con referencia al Product Backlog creado en el Sprint anterior, se constituyen tareas específicas por cada Sprint.

De manera que se quedaron establecidas las siguientes actividades:

- Diagrama de flujo.
- Modelo de base de datos.
- Creación del repositorio.

Diagrama de flujo

La **Fig. 4** muestra los recursos a los cuales tienen acceso según cada tipo de usuario que se encontrara activo y su variación de interacción, influenciados por los requerimientos recolectados según el rol de cada usuario, mostrando al *Frontend* de qué forma y en que instancias pueden consumir los recursos de la *API*.

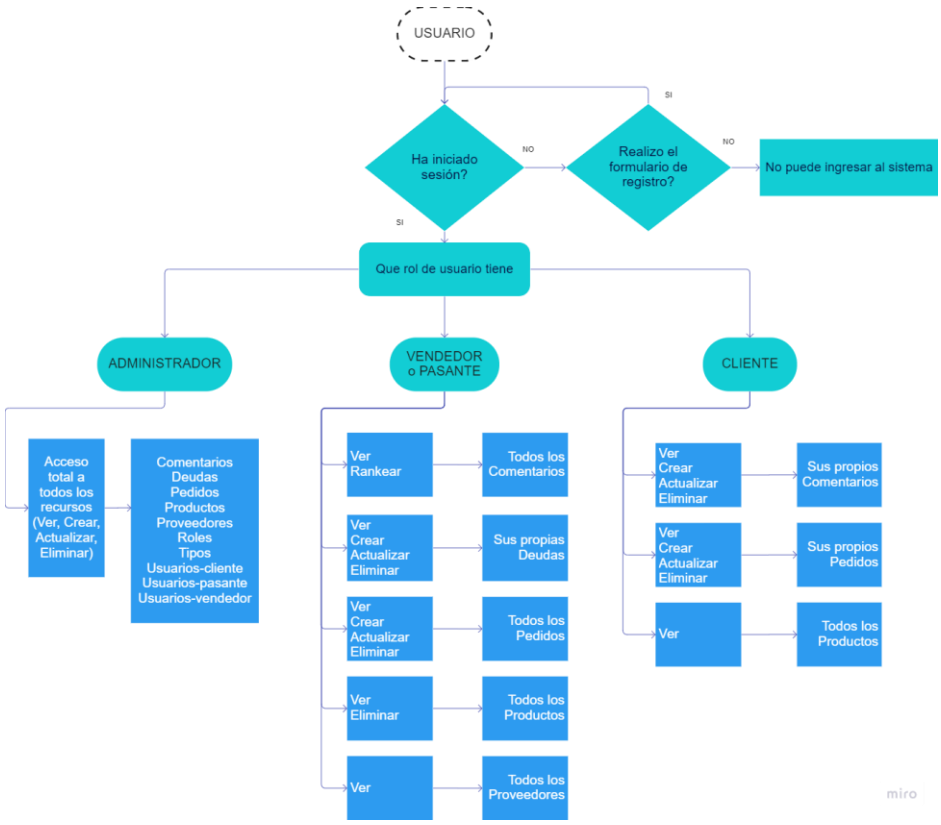


Fig. 4: Diagrama de flujo.

Modelo de base de datos

Para la construcción de la base de datos se utilizó el Sistemas Gestor de Base de Datos MySQL, por el hecho de que tiene disponible todas las herramientas para el modelado de datos e integración necesarias para el sistema. La base de datos es esencial ya que allí se almacenarán los datos para el sistema de gestión de inventario, para dar respuesta a las peticiones o consultas realizadas a la API, sin contar que su interfaz gráfica da la facilidad de administrar y generar diagramas, como el que se muestra en la **Fig. 5** denominado diagrama entidad - relación con sus claves y sus relaciones.

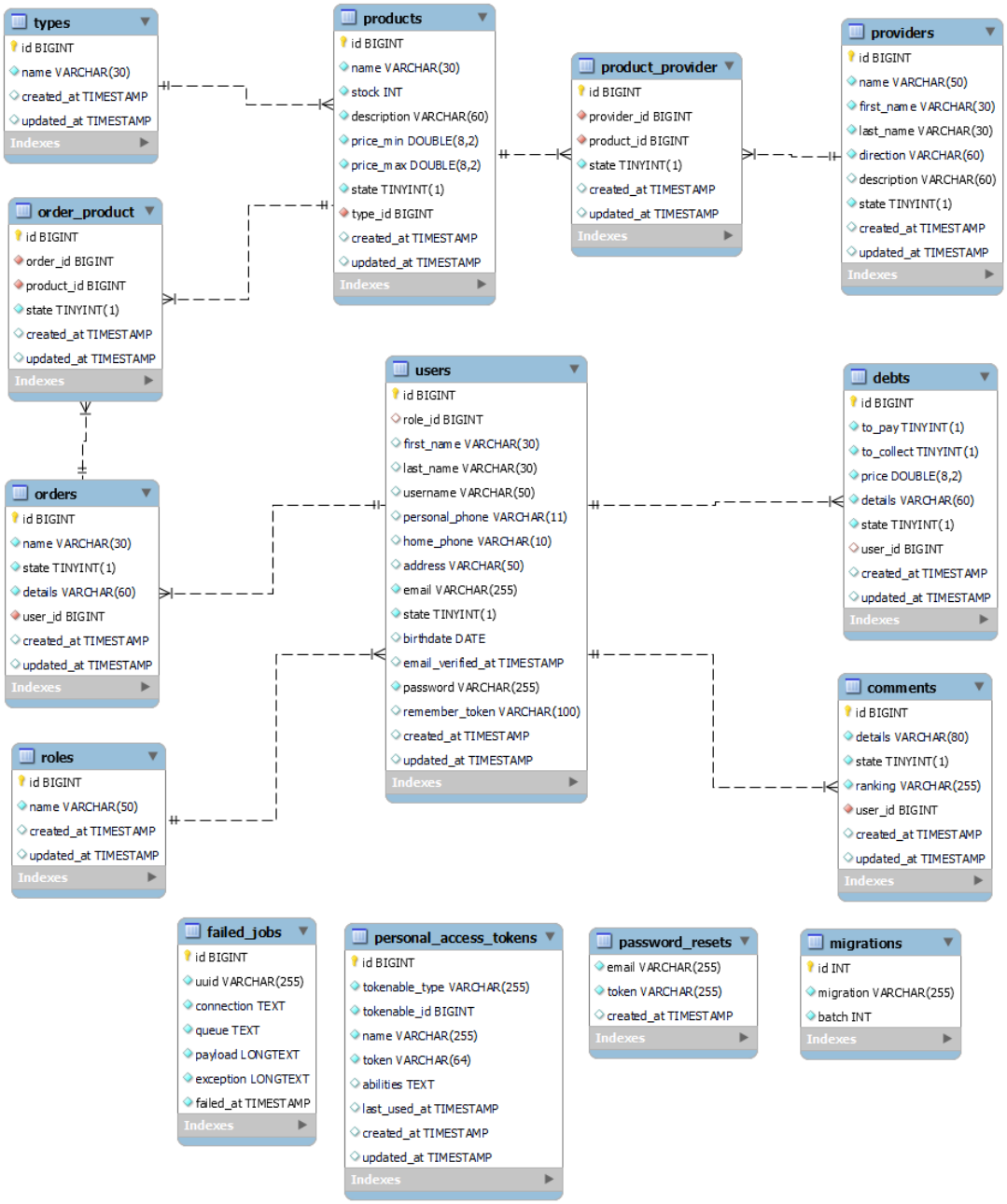


Fig. 5: Diagrama MER

Creación del repositorio

Con la herramienta detallada Git y su plataforma GitHub se creó el repositorio propio del proyecto de desarrollo para poder trabajar con ramas en cada tarea si se lo requiere, por otro lado, nos puede permitir llevar un control progresivo de todo el progreso y así mismo revisar los *Pull Request*, que hace referencia a la combinación de ramas secundarias con la principal para mantener la integridad del proyecto.

De manera simultánea con *ZenHub* se pudo crear un tablero Kanban para una mejor estructura de las tareas planificadas como los denominados Product Backlog y los Sprint Backlog, y ayudara a la gestión estableciendo etiquetas para categorizar, haciendo estimados para priorizar o clasificándolo dentro de cada tubería, refiriéndose a cada estado de la tarea dentro de su ciclo en el tablero Kanban como ilustra la ¡Error! No se ncuentra el origen de la referencia..

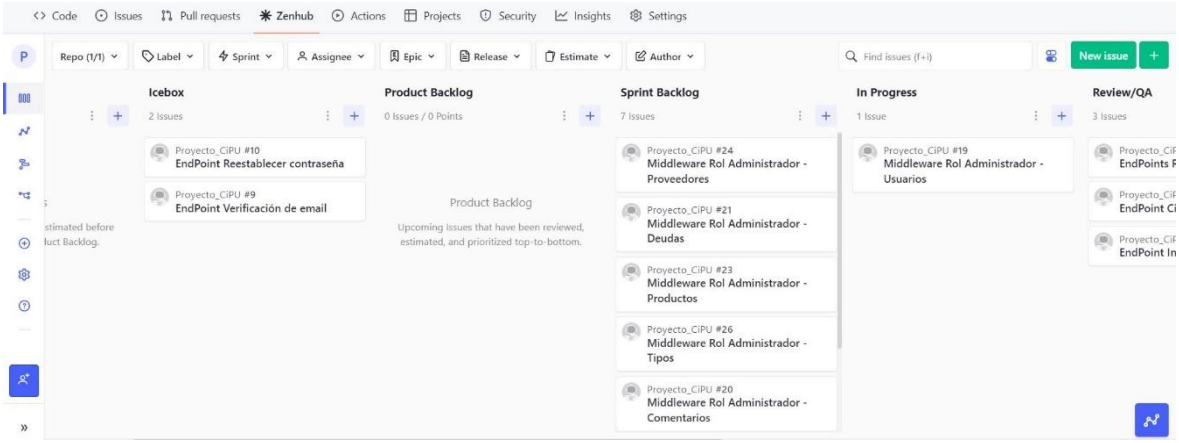


Fig. 6: Tablero Kanban de tareas del proyecto.

3.3 Sprint 2. Desarrollo de módulo de conexión entre base de base de datos y el ORM Eloquent de Laravel.

El sistema web de control de inventario con la base de datos establecida nos muestra aquellos modelos o tablas que tendrá a disponibilidad el Product Owner o Administrador del sistema, por lo que este Backend tendrá que adaptar estas tablas hacia el proyecto de Laravel para poder utilizar sus datos con el lenguaje de programación PHP, para ello existe un importante intermediario denominado ORM que significa Mapeo Objeto Relacional, explicado con anterioridad dentro del diseño de la base de datos.

Eloquent es el ORM de *Laravel* que implementa el patrón de arquitectura Active Record. Eso significa que cada tabla en la base de datos corresponde a una clase PHP (Modelo) que interactúa con ella. Cada instancia de dicha clase corresponde a un registro en la tabla y genera persistencia en la base de datos, con lo cual una actualización de los atributos de la clase puede alterar el registro correspondiente en la base de datos. [23]

Simplificado en otras palabras, cada tabla de la base de datos pasara a ser una clase o también llamado *Modelo* en nuestro proyecto pudiendo crear instancias de estas y utilizándolas para acceder a sus datos y realizar los CRUD sobre estos según la lógica de negocio lo requiera.

Precisando que, aunque todas estas tecnologías informáticas son útiles para el desarrollo web no llegan a ser automáticas por lo que se especifican acciones manuales que se deben hacer para establecer la conexión de la base de datos con el proyecto en Laravel, estas son: migraciones, creación de modelos y relaciones a nivel de base de datos y a nivel de Eloquent, también para fines de desarrollo los seeders y factories.

Que en si los Seeders acorde a su nombre son semillas para la creación de datos fijos que debe trabajar su base de datos como pueden ser los roles en la lógica de negocio de este proyecto o en los tipos de productos que se manejarán, haciendo alusión a los datos que probablemente se mantendrán así a lo largo del ciclo de vida del sistema informático.

Mientras que los Factories son herramientas para generar registros falsos con los lineamientos y relaciones que se le configuren, orientado a los entornos de desarrollo que busca generar datos rápidamente para optimizar el tiempo.

Detallado todos los elementos para la conexión establecida para este Sprint, queda establecidas las siguientes actividades:

- Configuración de migraciones y relaciones a nivel de base de datos.
- Creación de modelos y configuración de relaciones a nivel de Eloquent.
- Programación de seeders.
- Programación de factories.

Configuración de migraciones y relaciones a nivel de base de datos.

Esta es la primera interacción entre la base de datos y el framework Laravel, por lo que se configuro la manera en que se creara las tablas en la base de datos MySQL, el procedimiento de esta configuración es primero la creación de la migración con comandos directos en la terminal de VS Code que ya nos instancia este archivo PHP para establecer que campos debe tener cada tabla, que tipo de dato, su extensión y en ocasiones valores predeterminados como lo indica la **Fig. 8**, mientras que la **Fig. 7** nos muestra la lista de archivos de migraciones que se realizaran a la base de datos y será una por cada tabla, a excepción de la tabla usuarios que contiene migraciones por defecto para acciones de autenticación y cambios de contraseñas.

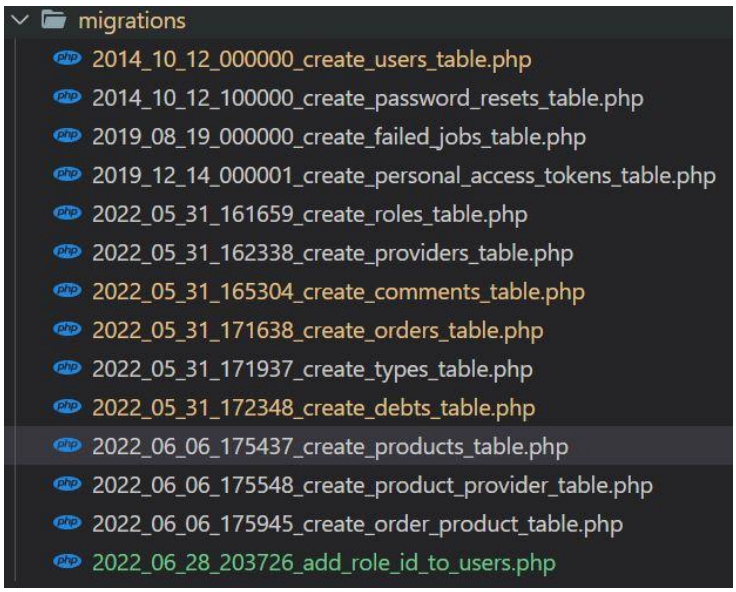


Fig. 7: Lista de migraciones.

```php
Schema::create('products', function (Blueprint $table) {
    //ID para la tabla BDD
    $table->id();
    //nombre producto
    $table->string('name',30);
    //stock producto
    $table->integer('stock');
    //descripcion producto
    $table->string('description',60);
    //precio minimo producto
    $table->float('price_min',8,2);
    //precio maximo producto
    $table->float('price_max',8,2);
    //estado del producto
    $table->boolean('state')->default(true);
```

Fig. 8: Propiedades de los campos en archivo PHP de migración.

Es importante tener en cuenta que las migraciones se ejecutan en el orden que fueron creadas por lo que si existe un campo en la migración de usuario que necesite la relación de otra migración pues evidentemente deben ser creadas están migraciones con anterioridad para luego crear una migración destinada a crear este campo en la tabla con su respectiva relación como lo ilustra la misma **Fig. 7** que al final tiene una migración con un nombre particular y distinto al resto, específicamente “add_rol_id_to_users”, que por tal razón mencionada se encuentra al final de la lista.

Ahora, con los atributos de las tablas establecidos en cada migración se crea las relaciones a nivel de base de datos especificando el campo en el cual se quiere establecer dicha relación, seguido de sus configuraciones esenciales como el nombre del campo foráneo, la referencia del campo foráneo, sobre que tabla y sus propiedades en caso de eliminación o actualización como lo ilustra la **Fig. 9**.

```php
//relacion
$table->unsignedBigInteger('type_id');
//un tipo puede tener uno o mas productos y un producto le pertenece a un tipo
$table->foreign('type_id')
    ->references('id')
    ->on('types')
    ->onDelete('cascade')
    ->onUpdate('cascade');
```

Fig. 9: Relación de producto con tipo a nivel de base de datos.

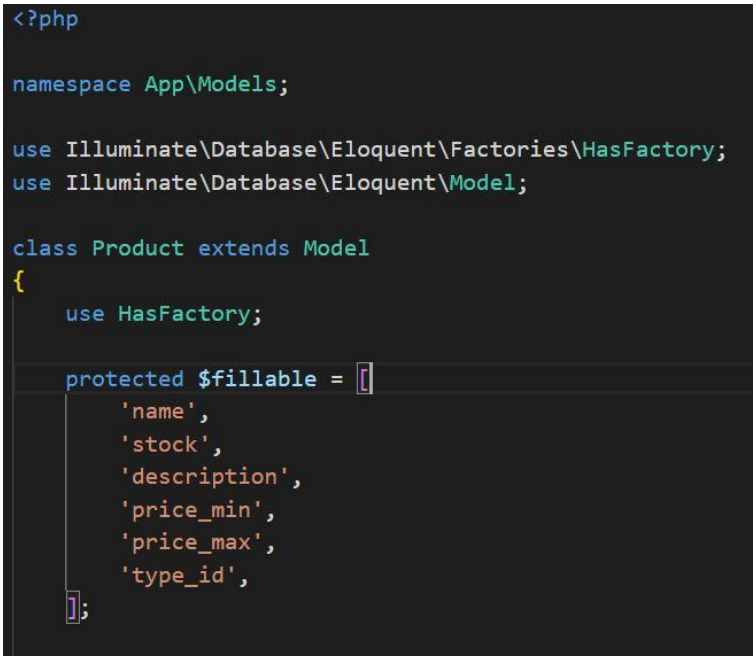
Creación de modelos y configuración de relaciones a nivel de Eloquent

El proceso de creación de modelos en Laravel tiene su procedimiento que sigue el estándar regular de los comandos del propio framework para una armonía en la búsqueda y aplicación de aquellos tomando en cuenta sus adicionales, haciendo alusión a las banderas que nos permiten crear y enlazar más archivos como factories y seeders según sea la necesidad, la **Fig. 10** presenta el comando regular para la creación de cualquier modelo en Laravel con su bandera para crear y enlazar al modelo, su factory y seeder de manera adicional.

```
PS C:\Users\HP\Documents\EPN\PROYECTO DE INTEGRACION CURRICULAR\TRABAJO DE
 INTEGRACION CURRICULAR\PROYECTO_CiPU>php artisan make:model Product -mfs
```

Fig. 10: Creación del modelo "Producto".

Mayormente la creación de los modelos se llevó a cabo de esta manera exceptuando el modelo “User” que es un modelo autenticable y viene por defecto cuando se usa la plantilla de Laravel Breeze, que nos brinda una interfaz gráfica como adicional, pero para fines de BackEnd no se lo utilizo, pero su modelo y sus tablas relacionadas a la autenticación serán útiles en el siguiente Sprint dedicado al módulo de autenticación. Este Sprint muestra cómo queda un modelo creado y que atributos debe tener, lo cual lo muestra la **Fig. 11.** El modelo “Producto” como el ORM detallaba que cada tabla creada con cada archivo de migración se convertirá en una clase, se presenta la estructura de un modelo, y los atributos mencionados son el arreglo protegido denominado “$fillable” que permitirá la asignación masiva y debe ir acorde a los campos creados en cada tabla de las migraciones dado que si el nombre de un campo no está establecido en este arreglo, no permitirá la creación hacia la base de datos por métodos externos como lo es el método POST de la solicitud HTTP.



```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Product extends Model
{
    use HasFactory;

    protected $fillable = [
        'name',
        'stock',
        'description',
        'price_min',
        'price_max',
        'type_id',
    ];
```

Fig. 11: Modelo "Producto".

Para completar la creación individual de cada modelo se estableció sus relaciones a nivel de Eloquent acorde a las creadas con su archivo de migración, es decir, las mismas relaciones que se creó en los archivos de migraciones, se creó en cada modelo de forma individual con su estructura decretada de manera que tenga consistencia la base de datos con el proyecto, la **Fig. 12** muestra las relaciones a nivel de Eloquent notando la diferencia entre niveles, pero ambas completamente imprescindibles.

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Product extends Model
{
    use HasFactory;

    protected $fillable = [
        'name',
        'stock',
        'description',
        'price_min',
        'price_max',
        'type_id',
    ];

    public function type()
    {
        return $this->belongsTo(Type::class);
    }

    public function orders()
    {
        return $this->belongsToMany(Order::class)->withTimestamps();
    }

    public function providers()
    {
        return $this->belongsToMany(Provider::class)->withTimestamps();
    }
}
```

Fig. 12: Relaciones a nivel ORM Eloquent.

Programación de Factories

Los factories son código de creación de registros con valores por default o valores ficticios para toda la tabla estipulada o campos estrictos, maneja varios tipos de datos que deben ir acorde a los establecidos en los atributos de cada tabla por razones de compatibilidad dentro de Eloquent y su enlace con la base de datos, mientras los factories creen registros con los tipos de datos de acuerdo a los atributos de las tablas de la base de datos no existirá ninguna contrariedad con su instauración. La **Fig. 11** ilustra las formas en las que se fueron creando los registros de manera ficticia de las tablas que tienen esta configuración.

```php
<?php

namespace Database\Factories;

use Illuminate\Database\Eloquent\Factories\Factory;
use App\Models\Product;

class ProductFactory extends Factory
{
    protected $model = Product::class;

    public function definition()
    {
        return[
            'name'=> $this->faker->title(),
            'stock'=> $this->faker->numberBetween($min=1, $max=50),
            'description' => $this->faker->text(40),
            'price_min'=>$this->faker->randomFloat(2, 0, 60),
            'price_max'=>$this->faker->randomFloat(2, 60, 120)
        ];
    }
}
```

Fig. 13: Factory “Producto”.

Cabe mencionar que esto fue útil para registros que no tienen registros estipulados de manera estricta como lo es la tabla “Roles” que ya tienen definido sus 4 registros de acuerdo a los requerimientos establecidos, sin embargo, a pesar de que las tablas no necesiten datos ficticios deben esperar su accionar a través de su seeder individual enlazado al modelo que lo creo.

Programación de Seeders

En esta sección, y lo que le diferencia a este código con los factories, es que los seeders son código de configuración para establecer registros, campos o incluso relaciones que son reales para la base de datos en producción, en otras palabras, si existe roles definidos u otra tabla con valores específicos para la lógica de negocio, se lo trata en este apartado para que permanezca durante todo el ciclo de vida del software. Los Seeders enlazados a un modelo son creados a la par con este último, su configuración

```php
<?php

namespace Database\Seeders;

use Illuminate\Database\Console\Seeds\WithoutModelEvents;
use Illuminate\Database\Seeder;
use App\Models\Product;
use App\Models\Type;

class ProductSeeder extends Seeder
{
    public function run()
    {
        $types=Type::all();

        $types->each(function($type)
        {
            Product::factory()->count(10)->for($type)->create();
        });
    }
}
```

Fig. 14: Seeder "Producto".

Los seeders necesitan de una variable que sea instancia del modelo de quien tenga relación con el modelo del seeder a configurar, llevado a la practica el modelo Producto tiene un relación con el modelo Tipo, la relación es “un producto le pertenece a un tipo, y un tipo tiene uno o varios productos”, por aquello el modelo Tipo debió ser creado inicialmente, con la ayuda de un bucle vamos iterando por cada tipo existente y por cada tipo se crea 10 registros de productos, quedando una tabla final de 10 registros por cada id de tipo, como lo indica la **Fig. 15.**

id	name	stock	description	price_min	price_max	state	type_id	created_at	updated_at
1	Dr.	3	Ratione id pariatur quia cum.	35.01	70.16	1	1	2022-08-02 03:53:54	2022-08-02 03:53:54
2	Mrs.	29	Et cupiditate rem sed cupiditate.	56.19	85.42	1	1	2022-08-02 03:53:54	2022-08-02 03:53:54
3	Mr.	13	Dolor explicabo qui et facere.	5.32	87.90	1	1	2022-08-02 03:53:54	2022-08-02 03:53:54
4	Miss	8	Dolor non pariatur neque quia.	59.15	61.39	1	1	2022-08-02 03:53:54	2022-08-02 03:53:54
5	Prof.	38	Qui maiores et quibusdam sit ex est.	11.12	81.05	1	1	2022-08-02 03:53:54	2022-08-02 03:53:54
6	Prof.	3	Non et enim soluta sequi ea.	22.36	91.84	1	1	2022-08-02 03:53:54	2022-08-02 03:53:54
7	Mr.	6	Nisi aut temporibus pariatur aut.	58.32	69.31	1	1	2022-08-02 03:53:54	2022-08-02 03:53:54
8	Prof.	24	Ab modi dolorem quia aperiam at.	9.83	87.47	1	1	2022-08-02 03:53:54	2022-08-02 03:53:54
9	Dr.	29	Omnis rem libero qui maxime placeat.	50.40	71.64	1	1	2022-08-02 03:53:54	2022-08-02 03:53:54
10	Dr.	19	Deleniti et non id iusto.	21.78	118.38	1	1	2022-08-02 03:53:54	2022-08-02 03:53:54
11	Dr.	29	Amet cum quis quod occaecati ut dol...	33.91	102.02	1	2	2022-08-02 03:53:54	2022-08-02 03:53:54
12	Miss	16	Aut enim aut dolor totam nobis.	57.77	92.83	1	2	2022-08-02 03:53:54	2022-08-02 03:53:54
13	Prof.	39	Velit magnam iste corporis dolor esse.	4.64	98.48	1	2	2022-08-02 03:53:54	2022-08-02 03:53:54
14	Prof.	34	Ea quia nulla id.	52.69	82.93	1	2	2022-08-02 03:53:54	2022-08-02 03:53:54
15	Dr.	13	Eum accusantium asperiores et.	37.81	104.66	1	2	2022-08-02 03:53:54	2022-08-02 03:53:54
16	Ms.	49	In est voluptatibus ea minima.	50.95	60.13	1	2	2022-08-02 03:53:54	2022-08-02 03:53:54

Fig. 15: Tabla "Productos" en la Base de datos.

A continuación, llevamos a la ejecución con comandos destinados a la implementación de las migraciones, seeders y factories. El último detalle de este apartado es el registro de forma ordenada de los seeders en el archivo DatabaseSeeder en la función de ejecución,

primero los seeder que no dependan de otros para que los seeder que si dependan ya tengan su base.

3.4 Sprint 3. Desarrollo los módulos de autenticación y registro usuarios.

Acorde a la planificación dentro del Product Backlog, establece que en el Sprint 2, conlleva las funcionalidades referentes desde los endpoints de autenticación de todos los usuarios con su respectivo rol, hasta el restablecimiento de contraseña.

Estableciendo las siguientes tareas para el Sprint:

- Inicio de sesión para todos los roles de usuarios.
- Registro de usuarios y verificación de email.

Inicio de sesión para todos los roles de usuarios.

El estándar utilizado para el módulo de autenticación de la API es el denominado JSON Web Token (JWT) el cual es un sistema abierto que crea Tokens temporales para permitir él envió de información entre aplicaciones en forma de objeto JSON, estos se forman utilizando algoritmos de encriptación lo cual los hace seguros y útiles para la arquitectura cliente-servidor por lo que los usuarios utilizar para hacer solicitudes a la API y acceder a sus recursos con el Token presente en cada solicitud.

Funciona en la práctica mandando al servidor los datos de inicio de sesión del usuario que sean válidos, es decir, que existan y coincidan en la base de datos, esto crea el JWT y lo envía a la aplicación cliente para establecer la autenticación luego en cada petición se debe enviar en la cabecera este Token para verificar la autenticación exitosa y a la vez saber quién es, pero hay que recalcar que este Token tiene un tiempo estimado de uso, por lo que tiene tiempo de expiración, situaciones en las que bastara realizar una nueva autenticación.

Como dato importante, las rutas deben ser protegidas con este middleware para que se exija la autenticación antes de poder acceder a los recursos, aquellas rutas que se encuentren fuera de este middleware se podrán acceder sin la necesidad de enviar el Token en la cabecera y sus recursos estarán disponibles sin restricción, en la ¡Error! No e encuentra el origen de la referencia. se ilustra las rutas que están dentro y fuera del middleware JWT destinado a la autenticación.

Fig. 16: Rutas con JWT.

El proceso de autenticación es el mismo para todos los roles de usuario, la diferencia será los recursos a los cuales pueden acceder con cada rol.

Registro de usuarios y verificación de email.

El registro de usuarios funciona con el estándar común utilizado, llena un formulario, se validan si los datos de los campos tienen el formato apropiado y el servidor aprueba sus credenciales, en servidores puntuales antes de autenticarse se debe confirmar el correo electrónico esto con fines de ver si el correo electrónico es real, para que en un futuro si necesita un restablecimiento de contraseña no tenga inconvenientes con el acceso a este último, por lo mencionado el backend tiene estas funcionalidades por un lado la ruta para enviar como parámetros datos del registro y por otro la verificación del correo a traves de un correo electrónico, valga la redundancia.

La **Fig. 17** muestra la función que comprueba los campos ingresados y guarda las credenciales en la base de datos para su posterior autenticación, mientras que la **Fig. 18** muestra el modelo del correo para verificar el correo electrónico del cliente que se registre.

```php
public function register(Request $request)
{
    $validator = Validator::make($request->all(),[
        'name' => 'required|string|max:255',
        'email' => 'required|string|email|max:255|unique:users',
        'password' => 'required|string|min:6|confirmed',
    ]);
    if($validator->fails()){
        return response()->json($validator->errors()->toJson(),400);
    }

    $user = User::create([
        'name'=>$request->get('name'),
        'email'=>$request->get('email'),
        'password'=>Hash::make($request->get('password'))
    ]);

    $token = JWTAuth::fromUser($user);

    return response()->json(compact('user','token'),201);
}
```

Fig. 17: Función de registro de usuario "cliente".

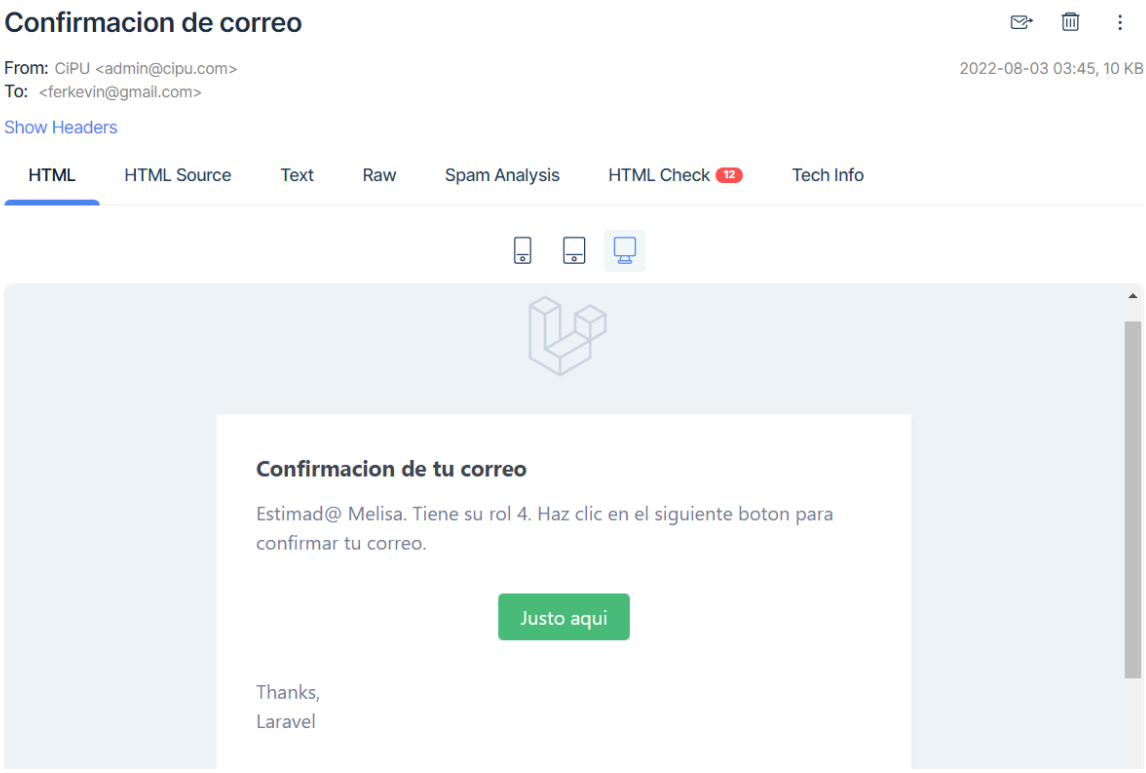


Fig. 18: Verificación del correo registrado por usuario.

3.5 Sprint 4. Desarrollo de módulos Endpoints de Usuarios, Productos, Pedidos, Proveedores, Roles, Tipos, Comentarios y Deudas.

Desde el módulo anterior los usuarios se pueden autenticar, ahora se creará para los usuarios las rutas para el acceso a los recursos, seguido del controlador que permitirá extraer los recursos, por motivo que el backend no se enfoca en las vistas hacia el usuario, quedara disponible en rutas para ser consumidas por el frontend, casi en su totalidad los recursos estarán disponibles de la misma manera. A raíz de los controladores específicos para API creados con el comando acorde a la armonía de Laravel, de aquí partirá la codificación de los controladores con el fin de dirigir los recursos hacia los usuarios correctos estipulados según los requerimientos.

Por lo tanto, el principio queda adoptado para poder realizar las siguientes actividades:

- Creación de controladores API.
- Codificación de métodos de controladores API.
- Creación de rutas API.

Creación de controladores API

Conforme al estándar de comandos de Laravel dicta, se realiza la creación de los controladores para todos los modelos, en su casi totalidad tendrán las mismas funciones, la documentación oficial de Laravel tiene guías de los comandos para atribuir al controlador capacidades para acelerar el desarrollo, como la bandera “-api” que muestra la **Fig. 19**, para indicarle que métodos debe tener personalizado.

```
PS C:\Users\HP\Documents\EPN\PROYECTO DE INTEGRACION CURRICULAR\TRABAJO DE INTEGRA
CION CURRICULAR\PROYECTO_CiPU> php artisan make:controller ProductController -api
```

Fig. 19: Comando para crear controlador API.

Codificación de métodos de controladores API

Los controladores API no manejaran vistas más bien manejaran respuestas de objetos JSON que son una notación sencilla que lleva una estructura basada en el tipo de datos diccionario que tiene clave y un valor, para facilitar el intercambio de información.

Esto le dará al desarrollador Frontend un estándar de datos flexible y óptimo para trabajar en su desarrollo manteniendo la armonía instaurada por el principio de los objetos JSON durante todo el ciclo de vida del sistema informático.

Continuando, para la manera de mostrar recursos tenemos métodos diferenciados como: 'index' para presentar todos los registros de su modelo, 'store' para crear un registro del modelo, 'show' para mostrar un registro especifico, 'update' para actualizar el registro en cuestión y 'destroy' para eliminar el registro. Construyendo la estructura se puede hacer personalizaciones a los controladores con métodos adicionales si los requerimientos lo necesitan, la **Fig. 20** muestra un método individual para el controlador de "Producto" que impide la eliminación de un producto si tiene un stock disponible y la **Fig. 21** muestra parte de las funciones requeridas para un controlador API, que fueron utilizados para la mayoría de los controladores del proyecto.

```php
public function destroy($id)
{
    $product= Product::find($id);
    $state= $product->state;

    if($this->verifyProductHasAssignedElements($product))
    {
        return response()->json([
            'name'=>"The product $product->name has assigned elements",
            'code'=>'400',
        ]);
    }
    $product->state=!$state;
    $product->save();
    return $product;
}

public function verifyProductHasAssignedElements(Product $product)
{
    return $product->stock > 0;
}
```

Fig. 20: Función de control de stock para el método 'destroy'.

```php
class ProductController extends Controller
{

    public function index()
    {
        return Product::all();
    }

    public function store(Request $request)
    {
        $request->validate([
            'name'=> ['required'],
            'stock'=> ['required'],
            'description' => ['required'],
            'price_min' => ['required'],
            'price_max' => ['required'],
        ]);

        return Product::create($request->all());
    }

    public function show($id)
    {
        return Product::find($id);
    }

    public function update(Request $request, $id)
    {
        $request->validate([
            'name'=> ['required'],
            'stock' => ['required'],
            'description' => ['required'],
            'price_min' => ['required'],
            'price_max' => ['required'],
        ]);
```

Fig. 21: Métodos controlador "Producto".

Creación de rutas API

Para la manera de entregar estos datos al Frontend existe el mecanismo por rutas, constituyéndose como solicitudes HTTP para el consumo de la API, el Frontend a través de solicitudes HTTP con la estructura instaurada en este Sprint recibirá objetos JSON, la estructura consta del nombre que se le atribuya, que por convención suele ser el nombre del recurso al cual queremos acceder y dependiendo el método, la variable que necesite como parámetro, se debe precisar el nombre del controlador y el método encargado de devolver los recursos, para su instrucción esta la **Fig. 22**.

```php
Route::middleware('jwt.verify')->group(function()
{
    //Obtener usuario autenticado
    Route::get('/perfil', [UserController::class, 'getAuthenticatedUser']);
    Route::get('/logout',[UserController::class, 'logout']);
    //Products
    Route::get('/products',[ProductController::class,'index']);
    Route::post('/products',[ProductController::class,'store']);
    Route::get('/products/{product}',[ProductController::class,'show']);
    Route::put('/products/{product}',[ProductController::class,'update']);
    Route::delete('/products/{product}',[ProductController::class,'destroy']);
    //Providers
    Route::get('/providers',[ProviderController::class,'index']);
    Route::post('/providers',[ProviderController::class,'store']);
    Route::get('/providers/{provider}',[ProviderController::class,'show']);
    Route::put('/providers/{provider}',[ProviderController::class,'update']);
    Route::delete('/providers/{provider}',[ProviderController::class,'destroy']);
    //Orders
    Route::get('/orders',[OrderController::class, 'index']);
    Route::post('/orders',[OrderController::class, 'store']);
    Route::get('/orders/{order}',[OrderController::class, 'show']);
    Route::put('/orders/{order}',[OrderController::class, 'update']);
    Route::delete('/orders/{order}',[OrderController::class, 'destroy']);
```

Fig. 22: Lista de rutas API.

3.6 Sprint 5. Desarrollo de módulo de Gates y Policies.

Una de las funciones y tal vez la más importante del Backend es la protección de sus recursos a personal no autorizado, lo que conlleva la utilización de mecanismos de gestión a través de métodos de autenticación y autorización, tratada la autenticación con su prudente importancia es el Sprint 3, queda por resolver los métodos de autorización que en el framework Laravel les atribuyen los nombres de: Gates y Policies.

Por lo tanto, gestionar que roles podrán acceder a que recursos es la misión de este Sprint y estos roles fueron sustrayéndose en base a la recopilación de requerimientos quedando fijados los roles de tipo: Rol Administrador, Rol Vendedor o Pasante y Rol Cliente, todos guardados en la tabla Usuario con el Id de Rol que servirá para atribuirles los permisos oportunos como ilustra la **Fig. 23**.

id	role_id	first_name	last_name	username
1	1	Jenifer	Spinka	Nedra Bailey
2	1	Larry	Koch	Dr. Delbert Wisoky
3	2	Marcelino	Hackett	Howell Reinger
4	2	Kimberly	Mayer	Retha Lakin
5	3	Gussie	Zboncak	Ezekiel Cassin
6	3	Willis	Baumbach	Eloy Stracke
7	4	Melisa	Schamberger	Carol Heller
8	4	Jalyn	Schimmel	Ron Borer IV
9	4	Ariane	Bruen	Ms. Layla Kutch Jr.
10	4	Margie	O'Kon	Leola Crist

Fig. 23: Tabla "Usuarios".

Entonces dentro del análisis, el nombre “Gate” literalmente significa “Puerta” pero técnicamente hace alusión a un permiso especifico, definido en su porción de código para permanecer como una función que devuelve verdadero o falso, acorde a las circunstancias y esto dependerá si el perfil o rol cumple con los requisitos, para autorizar la acción o negarla.

De este modo se define las “Policies” o Políticas, usan el mismo principio y lógica de los “Gates” pero están orientados a un modelo con métodos CRUD, dejando instaurada una clase propiamente para los métodos definidos por default, o para métodos personalizados simplemente agrupados de otra manera.

Por todo aquello plasmado, las tareas en este Script se han clasificado de esta manera:

- Construcción del diagrama de casos de uso.
- Codificación de “Gates” de los roles: Administrador, Vendedor o pasante y Cliente.
- Codificación de “Policies” de recursos destinados a modificar solamente su autor.

Construcción del diagrama de casos de uso

Inicialmente se preparó los diagramas de casos de uso con el fin de tener puntualmente representación visual de que acciones podría hacer cada tipo usuario dentro del sistema, para seguido de esto interpretar y adaptar en código en el proyecto de Laravel sin pasar por alto ningún detalle, para ello las figuras **Fig. 24, Fig. 25** y la **Fig. 26** muestra las acciones disponibles para el rol “Cliente”, “Vendedor” o “Pasante”, y el rol “Administrador”, respectivamente.

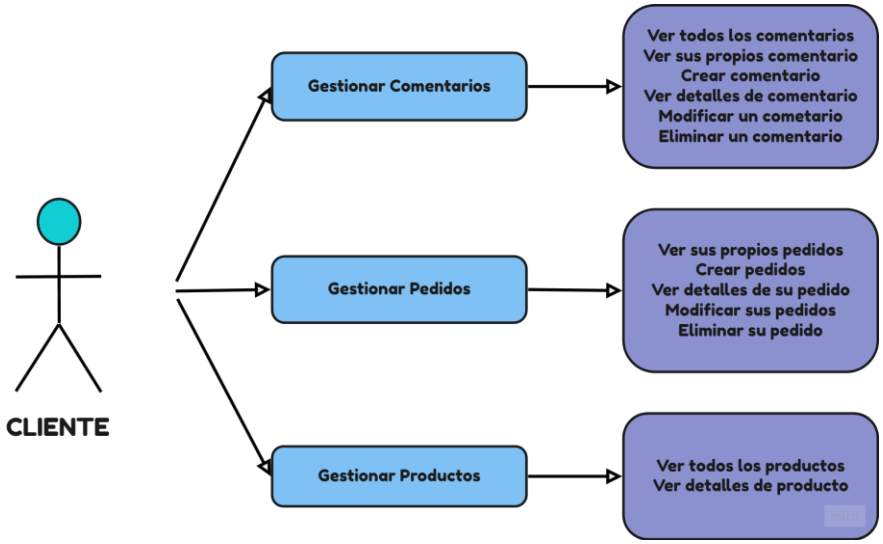


Fig. 24: Diagrama de Casos de uso de Rol "Cliente".

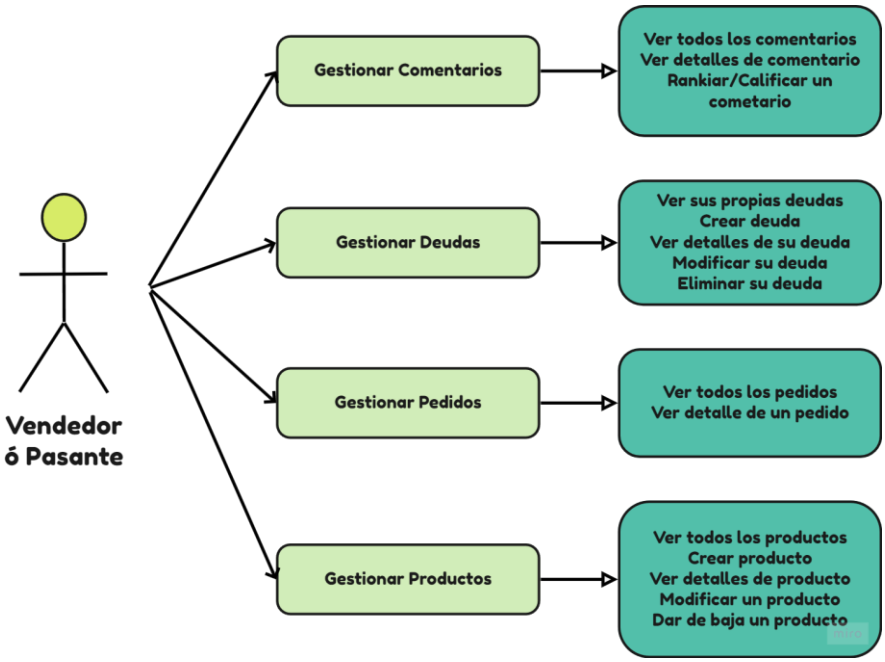


Fig. 25: Diagrama de Casos de uso de Rol "Vendedor/Pasante".

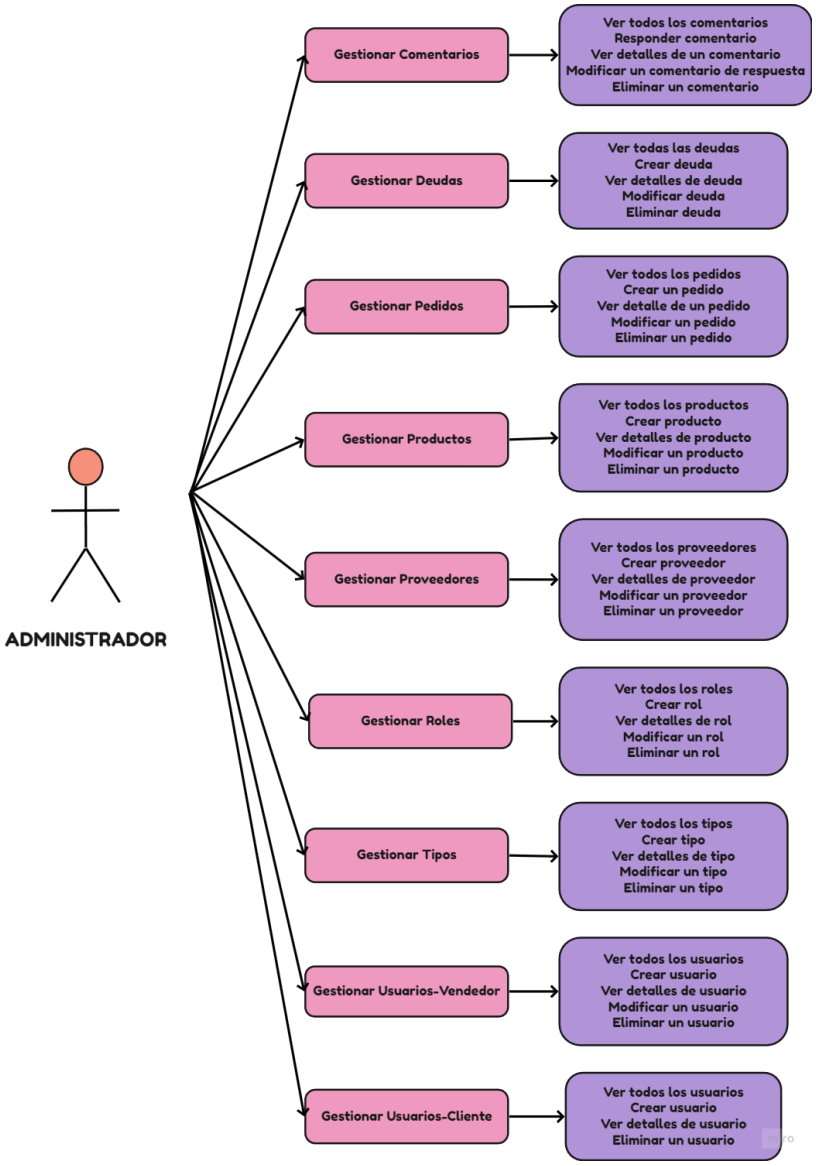


Fig. 26: Diagrama de Casos de uso de Rol "Administrador".

Codificación de “Gates” de los roles: Administrador, Vendedor o pasante y Cliente.

Como consecuencia se tiene una guía detallada de cuales “Gates” implementar y a que modelos permitir o denegar acciones, dependiendo del rol en uso, el principio da la oportunidad de instanciar un “Gate” por cada tipo de rol e implementarlo en cada constructor de cada controlador, atribuyéndole a este ultimo la posibilidad de permitir el acceso a sus funciones o denegarlas, para mejor estructura del proyecto se creó carpetas, una por cada rol para clasificar los controladores y mantener un orden sugerido, la **Fig. 27** como resultado muestra los “Gates” para cada rol, que se aplicó en cada controlador.

```php
public function boot()
{
    $this->registerPolicies();

    //Puertas ADMIN
    Gate::define('admin-manage-resources', function(User $user)
    {
        return $user->role->name === 'admin' ;
    });

    //Puertas CLIENT
    Gate::define('client-manage-resources', function(User $client)
    {
        return $client->role->name === 'client' ;
    });

    //Puertas SELLER & PASSANT
    Gate::define('manage-comments', function(User $user)
    {
        return $user->role->name === 'seller' || $user->role->name === 'passant';
    });
}
```

Fig. 27: Lista de "Gates".

Codificación de "Policies" de recursos destinados a modificar solamente por su autor.

El segundo tipo de este mecanismo mantiene la norma dicha, pero se las puede dedicar a modelos o a permisos personalizados, que puede ser las verificaciones de autoría para controlar recursos que primeramente hayan sido creados por los mismos usuarios, poniendo el caso que un usuario con rol "Cliente" pueda ver nada más que sus propios pedidos, así como modificarlos y/o eliminarlos, según sea sus necesidades, como ejemplo la ilustración de la **Fig. 28,** que verifica esta lógica con los parámetros recibidos.

```php
class OrderPolicy
{
    use HandlesAuthorization;

    public function isAuthor(User $user, Order $order)
    {
        return $order->user_id === $user->id;
    }
}
```

Fig. 28: Política de autoría sobre el modelo "Order".

3.7 Sprint 6. Desarrollo de módulo de notificaciones con correo electrónico.

A continuación, el módulo de notificaciones destinado a mantener informado al administrador y en ocasiones a los vendedores, de los eventos importantes o sobresalientes que sucedan en el ciclo del negocio ya sea para tomar acciones consecuentes o simplemente llevar un control de los eventos que generen sus alertas.

No obstante, el módulo de notificaciones no es exclusivo del lado de Backend, por lo que el Frontend lo podría asumir sin problemas, pero siendo estas alertas tan adheridas a la protección de recursos y a la necesidad de control de autenticación es imposible desintegrarlo del Backend, por lo tanto, se recurrió a su implementación con su procedimiento de Laravel que cuenta con una guía y plantillas útiles, como muestra la **Fig. 29**.

```
@component('mail::message')
# Confirmacion de tu correo

Estimad@
{{$user->first_name}}.
Tiene su rol {{$user->role_id}}.
Haz clic en el siguiente boton para confirmar tu correo.

@component('mail::button', ['url' => 'https:///www.google.com', 'color'=>'success'])
Justo aqui
@endcomponent

Thanks,<br>
{{ config('app.name') }}
@endcomponent
```

Fig. 29: Plantilla de correo de Laravel.

Por estas razones y por las peticiones de los requerimientos, su implementación necesito las historias de usuario para delimitar que alertas deben ser consideradas por el Backend, llegando a definir la lista concreta de las alertas estimadas, como lo ilustra la **Fig. 30.**

Modelo	Alerta
Comentario	• Aviso de nuevo comentario.
Deuda	• Aviso de nueva deuda creada.
Producto	• Aviso de mercancía a la mínima cantidad posible. • Aviso de nuevo producto ingresado.
Pedido	• Aviso de nuevo pedido creado.
Usuario - Cliente	• Aviso nuevo usuario registrado.
Usuario - Vendedor/pasante	• Aviso nuevo vendedor registrado.

Fig. 30: Lista de alertas del Backend.

Finalmente, en este módulo tenemos la implementación de la función de alertas por correo electrónico, ejecutándose con la estructura de Laravel, sus plantillas y sus clases, como muestra de la creación de los correos existe la **Fig. 31**.

```
//EMAILS
Route::get('/registro',function(){
    $correo = new RegistroMailable;
    Mail::to("direccion de correo electronico")->send($correo);
    return 'mensaje enviado';
});
```

Fig. 31: Ruta de creación de correo.

3.8 Sprint 7. Implementación de pruebas unitarias de la API REST.

Continuando con la planificación instaurada en el Sprint Backlog, en el Sprint 7, se precisan las tareas para la comprobación del funcionamiento de la API mediante pruebas unitarias.

Como resultado de este Sprint se presentan las tareas:

- Pruebas unitarias a rutas
- Comprobación de llamadas al servidor por medio de la API utilizando una aplicación cliente.
- Análisis de respuestas de cada componente del sistema que realice consulta.

Pruebas unitarias a rutas

Las pruebas unitarias son una práctica de verificación en este caso a rutas, que los desarrolladores sugieren para reducir el riesgo de posibles errores en el futuro y con ello mejorar la calidad de los productos desarrollados, dentro de las extensiones de Laravel se encuentra una herramienta para ejecutar pruebas de forma directa llamada PHPUnit.

El propósito del código del test en primera instancia es acceder a una ruta de recursos protegida, pero sin una previa autenticación, para examinar si la prueba es satisfactoria emitirá un código de estado '200', caso contrario mostrará el error, y según la **Fig. 32**.

```php
/** @test Login */
public function check_auth()
{
    $response = $this->get('http://localhost:8000/api/products');

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

Fig. 32: Prueba unitaria sin autenticación.

Como estaba planeado, la prueba emitió un error por la falta de autenticación previa y la **Fig. 33** presenta más detalles sobre el error.

```
C:\Users\HP\Documents\EPN\PROYECTO DE INTEGRACION CURRICULAR\TRABAJO DE INTEGRACION CURRICULAR\PROYECTO_CiPU>php vendor
/bin/phpunit --filter check_auth
PHPUnit 9.5.20 #StandWithUkraine

F                                                                   1 / 1 (100%)

Time: 00:00.474, Memory: 24.00 MB

There was 1 failure:

1) Tests\Feature\LoginTest::check_auth
Expected response status code [200] but received 401.
Failed asserting that 200 is identical to 401.

C:\Users\HP\Documents\EPN\PROYECTO DE INTEGRACION CURRICULAR\TRABAJO DE INTEGRACION CURRICULAR\PROYECTO_CiPU\vendor\laravel\
framework\src\Illuminate\Testing\TestResponse.php:179
C:\Users\HP\Documents\EPN\PROYECTO DE INTEGRACION CURRICULAR\TRABAJO DE INTEGRACION CURRICULAR\PROYECTO_CiPU\tests\Feature\L
oginTest.php:28
C:\Users\HP\Documents\EPN\PROYECTO DE INTEGRACION CURRICULAR\TRABAJO DE INTEGRACION CURRICULAR\PROYECTO_CiPU\vendor\phpunit\
phpunit\phpunit:98

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

Fig. 33: Resultado de la prueba sin autenticación.

Pero dentro del principio de hacer pruebas esta la comprobación satisfactoria enviando los datos correctos, con lo que se espera recibir una respuesta positiva, para dicha acción

se pasa como parámetros el usuario y la contraseña a través de un método POST en la ruta Login, como ilustra la **Fig. 34** y que nos expone la **Fig. 35** la respuesta satisfactoria como se la proyectaba.

```php
/** @test login 2*/
public function test_example()
{
    $credentials = [
        "email" => "josefina57@example.net",
        "password" => "secreto"
    ];
    $response = $this->post('http://localhost:8000/api/login',$credentials);
    $response->assertStatus(200)->assertJsonStructure(['token']);
}
```

Fig. 34: Prueba unitaria de Login.

```
C:\Users\HP\Documents\EPN\PROYECTO DE INTEGRACION CURRICULAR\TRABAJO DE INTEGRACION CURRICULAR\PROYECTO_CiPU>
php vendor/bin/phpunit --filter test_example
PHPUnit 9.5.20 #StandWithUkraine

..                                                                  2 / 2 (100%)

Time: 00:00.299, Memory: 24.00 MB

OK (2 tests, 2 assertions)
```

Fig. 35: Resultado de prueba unitaria de Login.

Posteriormente las solicitudes requerirán como parte de la cabecera, el token generado en la ruta de autenticación para acceder al resto de recursos en el rol administrador. Como consecuencia la **Fig. 36** exhibe como está estructurada la solicitud con el método GET para verificar que el producto con ID igual a 5, sea el mismo que se le postea en la respuesta JSON de la prueba unitaria, lo cual la **Fig. 37** lo consolida.

```php
/** @test login 3*/
public function test_products()
{
    $response = $this->json('GET', 'http://localhost:8000/api/products/5', [], ['Authorization'=>'eyJ0eXAiOiJ'] );
    $response->assertStatus(200)
            ->assertExactJson([
                'id'=>5,
                'name'=>'Prof.',
                'stock'=>32,
                'description'=>'Ea autem cum numquam eum excepturi eum.',
                'price_min'=>16.61,
                'price_max'=>62.10,
                'state'=>true,
                'type_id'=>1,
                'created_at'=>'2022-08-10 01:48:18',
                'updated_at'=>'2022-08-10 01:48:18'
            ]);
}
```

Fig. 36: Prueba unitaria método GET de Productos.

```
C:\Users\HP\Documents\EPN\PROYECTO DE INTEGRACION CURRICULAR\TRABAJO DE INTEGRACION CURRICULAR\PROYECTO_CiPp>
hp vendor/bin/phpunit --filter test_products
PHPUnit 9.5.20 #StandWithUkraine

.                                                                   1 / 1 (100%)

Time: 00:00.284, Memory: 24.00 MB

OK (1 test, 1 assertion)
```

Fig. 37: Resultado de prueba unitaria de método GET.

Chequeo de llamadas al servidor utilizando una aplicación cliente por medio de la API.

Mediante la aplicación cliente Postman se mantuvo realizando pruebas puntuales durante todo el desarrollo, precisando el tipo de método HTTP, acorde a la función del controlador, al igual que las rutas que fueron compuestas acorde el árbol de direcciones URL que necesitáramos, todo esto en el servidor en línea en el ambiente de producción, para una ilustración la **Fig. 38**, indica el método POST de la solicitud HTTP, la estructura de la ruta respectiva a la URL de desarrollo y los parámetros para ejecutar el registro de manera correcta.

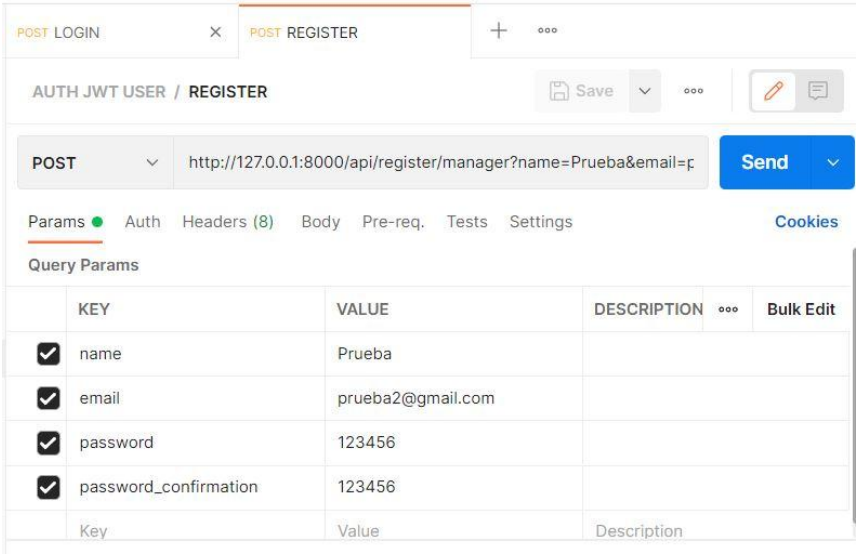


Fig. 38: Estructura de consumo de la API a través de Postman.

En este sentido, las rutas que se pueden revisar sin la creación del Token son lógicamente las de autenticación y registro, de allí en adelante en cada cabecera es necesario enviar el Token generado previamente, para muestra de un ejemplo la **Fig. 39**. Con este elemento fijo tras cada autenticación se puede acceder a los recursos y continuar con las pruebas para el módulo de permisos y autorizaciones, como manifiesta

la **Fig. 40** un ejemplo particular de la autorización denegada como resultado de una solicitud de acceso a un recurso donde el rol no es el apropiado para su permiso.


```
{
    "token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.
        eyJpc3MiOiJodHRwOi8vMTI3LjAuMC4xOjgwMDAvYXBpL2xvZ2luL21hbmFnZXIiLCJp
        YXQiOjE2NjAwOTkzMjAsImV4cCI6MTY2MDEwMjkyMCwibmJmIjoxNjYwMDk5MzIwLCJq
        dGkiOiJPd3c1cDJMd25qcTJiY0RyIiwic3ViIjoiMSIsInBydiI6IjIzYmQ1Yzg5NDlm
        NjAwYWRiMzllNzAxYzQwMDg3MmRiN2E1OTc2ZjcifQ.
        8rV4aArxnUsxgaDkYz2z1csKt-OCjks6ksZZhgHZmBw",
    "user": {
        "id": 1,
        "role_id": 1,
        "first_name": "Garret",
        "last_name": "Welch",
        "username": "Lee Ondricka",
        "personal_phone": "0991623438",
        "home_phone": "02383901",
        "address": "7433 Mohr Summit",
        "email": "abner74@example.net",
        "state": 1,
        "birthdate": "28/01/2018",
        "email_verified_at": "2022-08-10T01:48:16.000000Z",
        "created_at": "2022-08-10T01:48:16.000000Z",
        "updated_at": "2022-08-10T01:48:16.000000Z"
    }
```


Fig. 39: Inicio de sesión a través de Postman.


```
{
    "message": "This action is unauthorized.",
    "exception": "Symfony\\Component\\HttpKernel\\Exception\\AccessDeniedHttpException",
    "file": "C:\\Users\\HP\\Documents\\EPN\\PROYECTO DE INTEGRACION CURRICULAR\\TRABAJO DE INTEGRACION
        CURRICULAR\\PROYECTO_CiPU\\vendor\\laravel\\framework\\src\\Illuminate\\Foundation\\Exceptions\\Handler.php",
    "line": 379,
```


Fig. 40: Resultado de intento de acceso a un recurso no permitido**.**

Análisis de respuestas de cada componente del sistema que realice consulta.

En toda consulta que el sistema consumidor de la API haga uso, esta recibirá un respuesta por parte de la API, y estará construida por un orden, el primer elemento será la información requerida por la solicitud http, en métodos ‘get’, excluyendo los métodos ‘post’, ‘put’ y ‘delete’, que contaran con el segundo elemento que presenta el estado de la solicitud, derivados de las respuestas de solicitudes http y su clasificación, entonces directamente esta prueba busca probar todas las rutas y determinar si contienen ambos o de ser el caso al menos el elemento de estado de la solicitud, prueba de esto la se muestra un ejemplo de un mensaje de un estado exitoso.

Fig. 41: Estado de solicitud.

3.9 Sprint 8. Despliegue del proyecto de API REST en un servidor web público.

Como último, con la planificación instaurada en el Sprint Backlog, en el Sprint 8, se precisan las tareas finalmente el despliegue en un servidor público virtual y las pruebas en el ambiente de producción.

Como resultado del último Sprint se presentan las tareas:

- Configuración del servidor web apache.
- Verificación del rendimiento del sistema en el servidor.
- Comprobación de la seguridad del sistema.
- Prueba del consumo de la API desde otro sistema.

Configuración del servidor web Apache

Para el despliegue de la API REST se necesita un host público que pueda albergar el proyecto y la base de datos, pero lo más importante es que sean gratuitas sus herramientas, con lo detallado la plataforma Heroku se adapta a nuestras necesidades y a los requerimientos de la API.

Por lo tanto, dentro de los requisitos de Heroku dicta tener una cuenta creada como primero, como adicional instalar Heroku CLI para permitirnos usar los comandos propios de la plataforma y con esto subir al sistema. El siguiente requisito es la configuración de la carpeta que se subirá a producción, esto con la creación de un archivo Procfile dentro de la raíz del proyecto y la fi contiene la configuración del archivo mencionado.

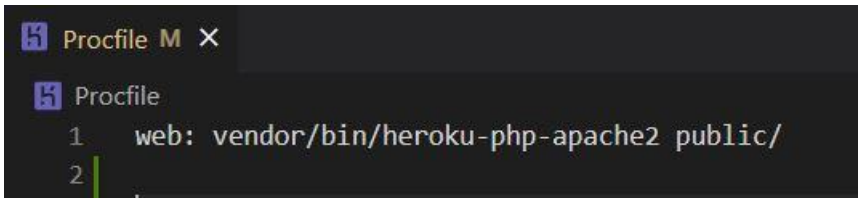


Fig. 42: Archivo Procfile que detalla la carpeta que sube a producción.

Como siguiente requisito esta la configuración de las variables de entorno para que el sistema tenga la guía completa de nuestra App, antes de esto debemos añadir un recurso a nuestra App, denominado Heroku Postgres, elegida por que mantiene su valor gratuito para el alojamiento de nuestra base de datos y con ella la API tenga inyectada ya los recursos del negocio, esto nos brinda los nombres para utilizar en las variables de

entorno y que su configuración este completa. Para terminar, se ejecuta comandos para subir el proyecto arrojando la URL del proyecto para construir los endpoints y con estas realizar solicitudes a la API, signo de que el proyecto está en línea la **Fig. 43** muestra el proyecto en el Dashboard de Heroku a un clic de su visualización.

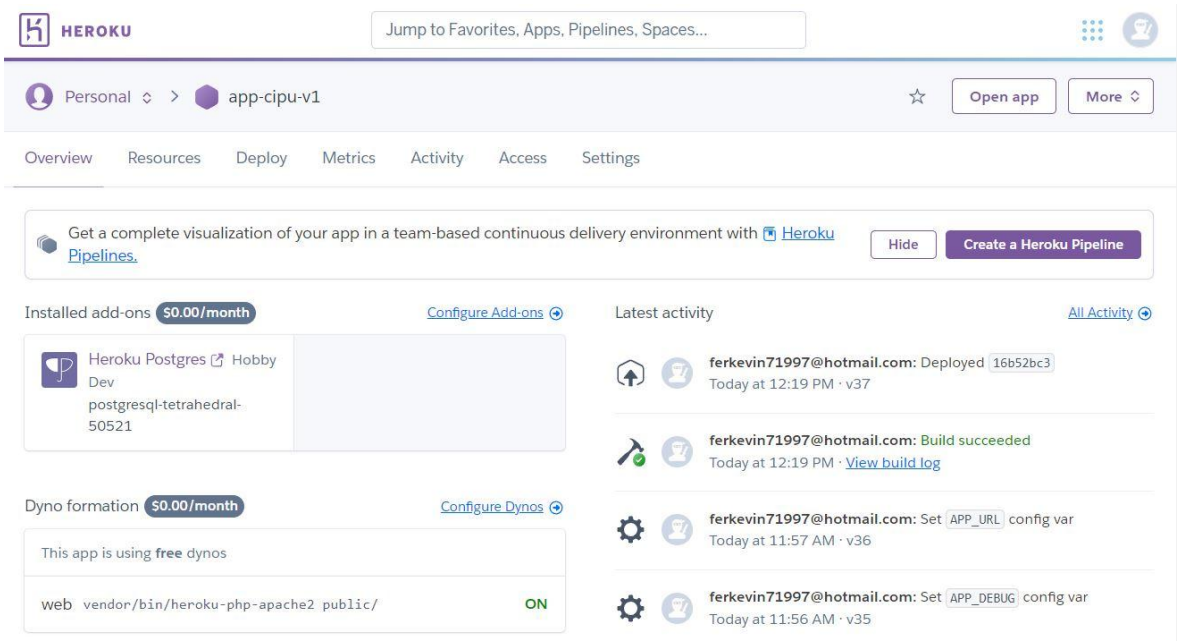


Fig. 43: Aplicación hostiada en Heroku.

Verificación del rendimiento del sistema en el servidor

Heroku es una plataforma que se encarga de la administración del servidor que se despliegue y por administración se refiere al control de rendimiento, presentación de estadísticas y estados de servicios, con ello facilita la gestión de la disponibilidad del sistema. A primera vista las métricas visibles son: su estado de disponibilidad, versión, número de conexiones, registros disponibles, tamaño de la base de datos, incluso el número de tablas, detallado en la **Fig. 44**.

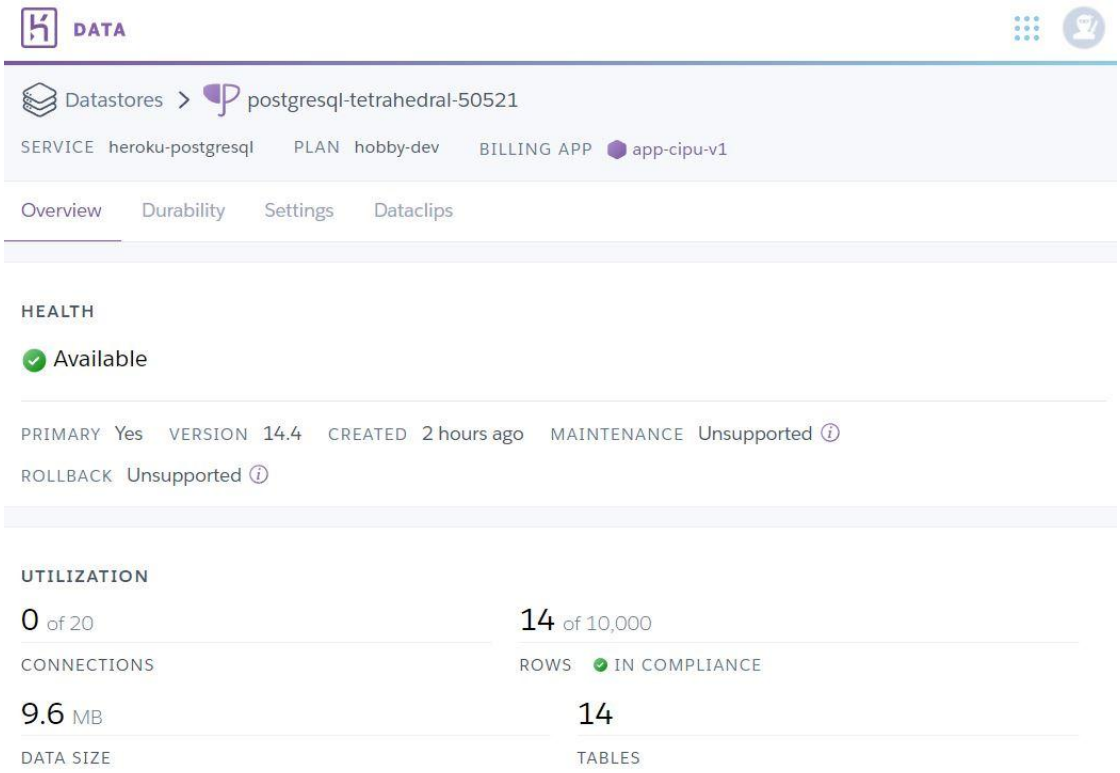


Fig. 44: Administración de datos en Heroku.

Pese a la gratuidad de la plataforma, el número de conexiones a la base de datos o el tamaño de la base de datos, son métricas que tienen un límite que no precisamente afectan a funcionamiento del sistema, pero de debe llevar un control para tomar decisiones en cuanto se supere el máximo.

Comprobación de la seguridad del sistema

Otra de las ventajas de la plataforma de Heroku es la seguridad en los despliegues, trabaja con seguridades SSL en todas las rutas del servidor, ya sea de acceso a la aplicación o de consultas que se realicen, incluso en la versión gratuita.

En retrospectiva, la seguridad de la aplicación está basado en la autenticación por Token, exactamente en JWT, que genera un token con cada inicio de sesión que está disponible durante un periodo de tiempo mientras el usuario activo requiera de los recursos de la API, reiterando, a más de las credenciales de usuario y contraseña verificados en la base de datos, con cada solicitud se deberá enviar este token único e intransferible, garantizando que en todas las peticiones sea el propio usuario activo el responsable de cada petición. Como ilustración de este proceso en la siguiente sección de muestra la autenticación y una solicitud con el token puesto en la cabecera.

Prueba del consumo de la API desde otro sistema

En conclusión, el proyecto esta desplegado en el servidor virtual de Heroku y disponible con la URL respectiva para construir los endpoints para cada función de nuestro Backend con el fin de recibir la información necesitada, esto lo diferencia del modo de desarrollo donde el servidor era la maquina local, por ende, disponible solo localmente, ya en producción las rutas son accesibles desde cualquier ordenador con internet y conocimiento de nuestras URL. No obstante, la autenticación es imprescindible por la importancia ya antes mencionada y como muestra de aquello, la **Fig. 45** indica la autenticación de un usuario registrado en la base de datos, con el método POST, enviando los parámetros de usuario y contraseña y con ellos esperar recibir el token para acceder a los recursos.

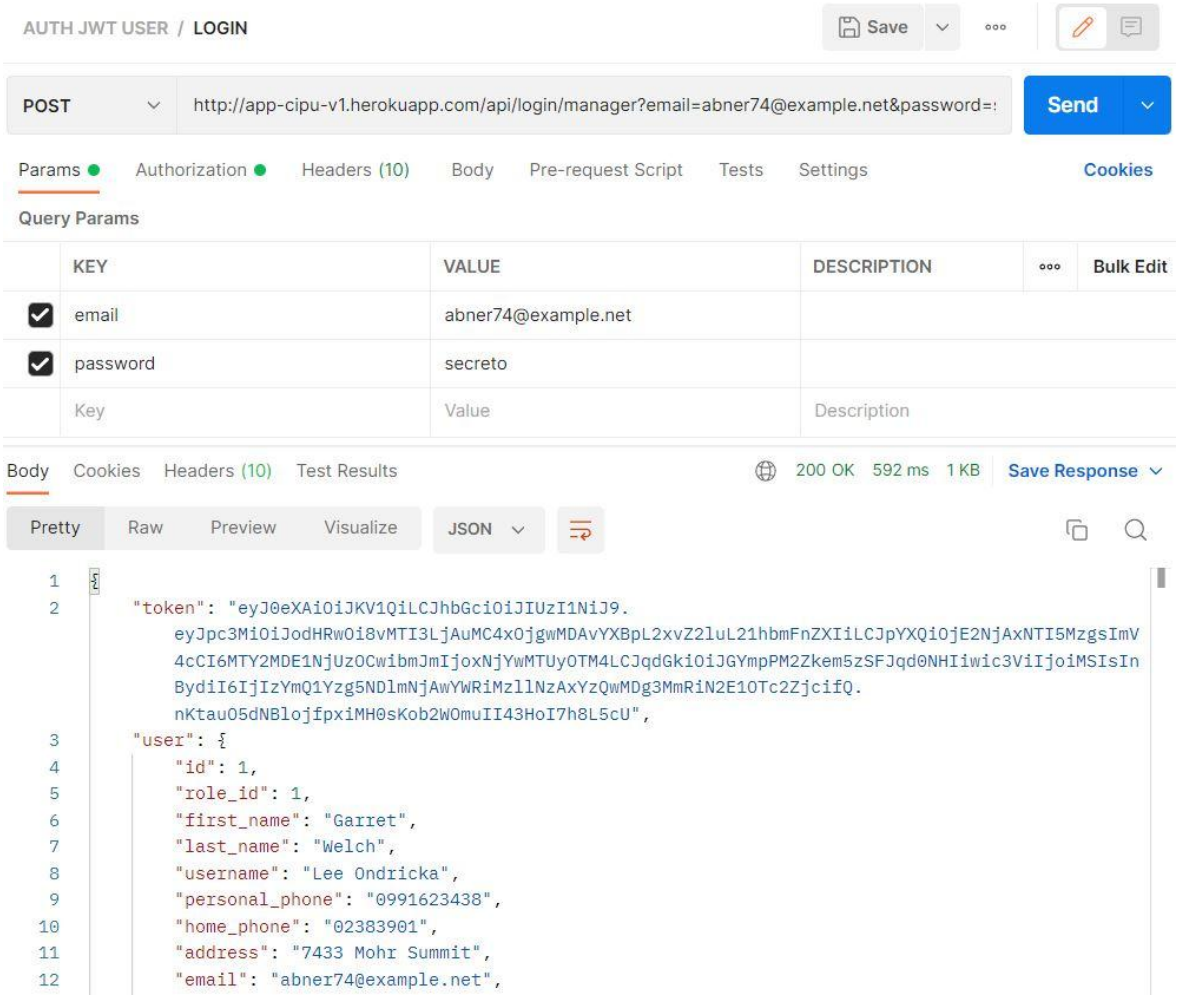


Fig. 45: Inicio de sesión en Postman con URL en producción.

Para terminar, con la obtención del token se tiene el requisito único para inyectarlo en la cabecera de la solicitud, para que el sistema nos permita acceder a todos y cada uno de

los recursos si el rol brindado lo permite. Como ultima ilustración la **Fig. 46** exhibe la manera en la que debe ir inyectado el token en el apartado de autorización en cada solicitud para poder recibir un permiso habilitado hacia la API, la solicitud en este caso quiere recibir información de todos los productos disponibles a través de un método GET y la URL del árbol de direcciones atribuidos a estos recursos en particular.

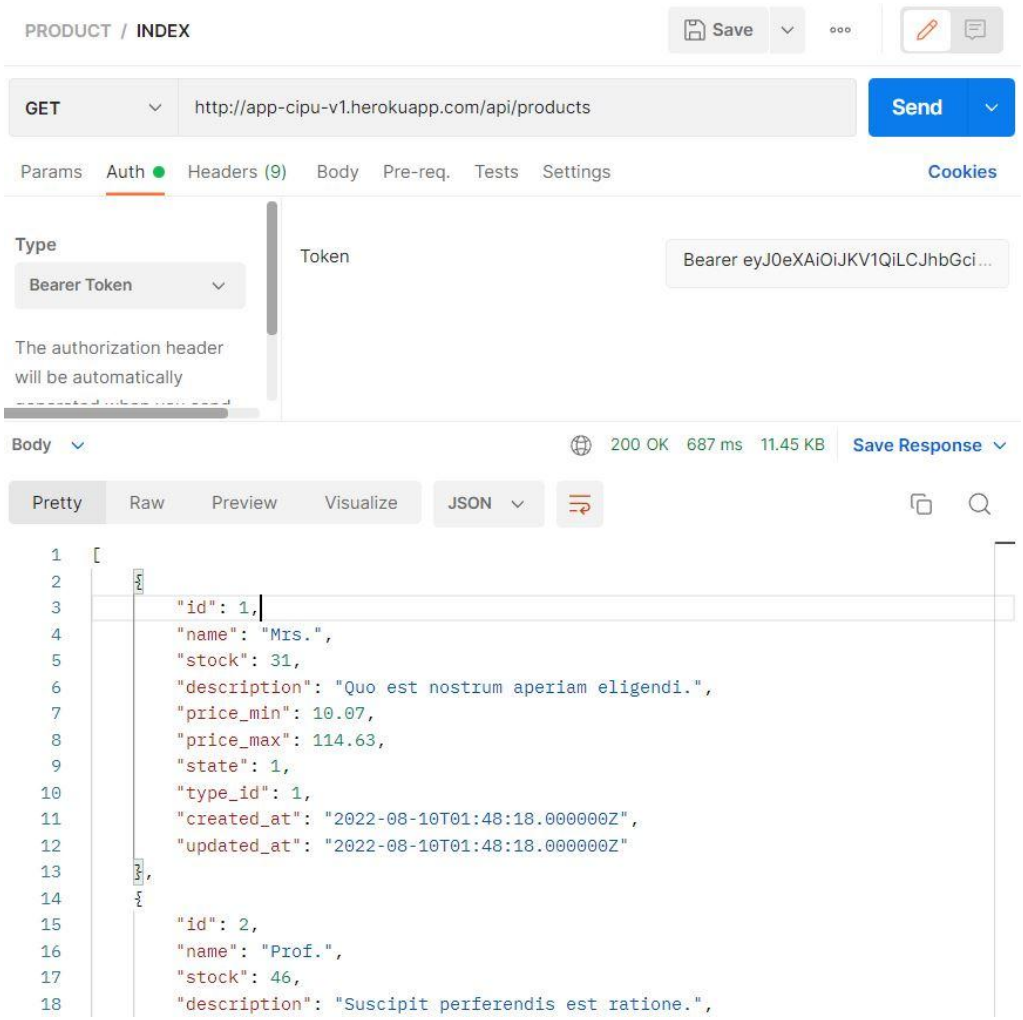


Fig. 46: Solicitud GET con token en produccion.

4 CONCLUSIONES

- El backend de gestión de inventario cumple con los objetivos propuestos y con el alcance definido, su intención de establecer una conexión con el frontend ya sea para aplicación web o aplicación móvil, según sean las necesidades esta solventada con el despliegue en el servidor virtual de Heroku, el desarrollador Frontend tendrá a su alcance, todas las URL para consumir los recursos.

- Al planificar y estructurar los Sprints acorde al modelo de la metodología Scrum nos posibilita distribuir de manera homogénea las funcionalidades para ser manejadas como un conjunto en su totalidad, es decir, se logró clasificar las funcionalidades que requieran los mismos principios o funciones para acelerar el proceso de codificación dentro de todo su Sprint, enfocándose en el primer elemento y replicándolo sobre resto de elementos que necesiten su similitud.

- El framework Laravel goza de muchas herramientas muy útiles y prácticas para todo el ciclo de desarrollo, lo que es digno de un framework, pero igual se valora la intención de agrupar herramientas para complementar este entorno y simplificar los archivos configurables para enfocarse en la implementación de la lógica de negocio sin la larga tareas de inicialización del proyecto desde cero y sin rumbo definido.

- El framework Laravel como consecuencia de un sistema armónico no permite la creación de más de un modelo autenticable, haciendo alusión al modelo Usuario con configuraciones de autenticación lo cual fue un deseo del Product Owner transferido y puesto a manera de reto, pero la compactación del framework no concedió otra solución que la más sencilla inicialmente, que era mantener todos los tipos de usuarios dentro de un mismo modelo, separado con su simple id de rol.

- La deducción acerca del patrón arquitectónico del backend fue acertada al elegir un patrón MVC que permite una escalabilidad y un mantenimiento muy intuitivo y practico, incluso para quienes deseen replicarlo en proyectos futuros.

- La plataforma Heroku tiene lo que se puede considerar como su principal desventaja de al superar el límite de tamaño requerir un plan con mejores prestaciones, como su principal ventaja, el desarrollador podrá migrar al plan que más se adapte a las necesidades y tamaño del sistema que requiera gestionar.

5 RECOMENDACIONES

- Durante la etapa de recopilación de requerimientos es importante mantener un contacto dinámico con el Product Owner, establecer canales de comunicación eficientes, dado que en cualquier momento se le puede ocurrir una idea discutible para el proyecto.

- Se recomienda usar extensiones del IDE del lenguaje de programación que se esté utilizando para facilitar las importaciones de clases, precisar los argumentos y la forma en la cual trabaja cada función y evitar pérdida de tiempo errores de estos motivos.

- Es importante tener un criterio de selección frente a las tecnologías disponibles en la red, existen las cuales apenas están emergiendo y consagrándose en el mercado recibiendo elogios de ser muy útiles y practicas sin mucha documentación y escasa comunidad, frente a las tecnologías clásicas, pero con estructuras consolidadas, soporte al día, documentación amplia y una comunidad muy amplia.

- Es adecuado y optimo distribuir los Sprints en funciones homogéneas para optimizar el tiempo al replicar los componentes que lo necesiten y que estos Sprints tengan como una de sus finalidades presentar algo palpable o visible para que el usuario pueda ir constatando el progreso de forma muy aproximada al rango real de su desarrollo.

- Es recomendable tener a disposición herramientas de comprobación, para ir corrigiendo errores de manera continua durante el ciclo de desarrollo, evitando adjudicar que el código de las funcionalidades está en el estado deseado por seguir al pie de la letra tutoriales, porque la tecnología o herramienta pude haberse actualizado y cambiado ligeramente su implementación, y esto evitara dejar todo el trabajo al módulo de pruebas y mantener un seguimiento

6 REFERENCIAS BIBLIOGRÁFICAS

[1] IBM, «¿Que es la gestion del inventario?,» [En línea]. Available: https://www.ibm.com/es-es/topics/inventory-management.

[2] M. M. Association, «Todo lo que necesitas saber sobre backend,» MMA, 6 Enero 2016. [En línea]. Available: https://www.mmaglobal.com/news/todo-lo-que-necesitas-saber-sobre-backend-all-you-need-know-regarding-backend#:~:text=El%20Backend%2C%20tambi%C3%A9n%20conocido%20como,y%20devolverla%20al%20usuario%20final..

[3] A. Robledano, «Que es MySQL: Caracteristicas y ventajas,» 24 Septiembre 2019. [En línea]. Available: https://openwebinars.net/blog/que-es-mysql/.

[4] I. d. Souza, «Descubre qué es el lenguaje de programación PHP y en qué situaciones se hace útil,» rockcontent, 9 Marzo 2022. [En línea]. Available: https://rockcontent.com/es/blog/php/.

[5] A. B. Navarro, «Que és Heroku y sus principales características,» ifGeekThen, 08 Septiembre 2020. [En línea]. Available: https://ifgeekthen.nttdata.com/es/Que%20es%20Heroku%20y%20principales%20caracter%C3%ADsticas.

[6] F. G. d. Zuñiga, «¿Qué es phpMyAdmin y cómo usarlo?,» Arsys, 25 Noviembre 2021. [En línea]. Available: https://www.arsys.es/blog/phpmyadmin.

[7] D. web, «Laravel,» 2021. [En línea]. Available: https://desarrolloweb.com/home/laravel.

[8] R. Hat, «¿Qué es la metodología ágil?,» 20 Enero 2020. [En línea]. Available: https://www.redhat.com/es/devops/what-is-agile-methodology.

[9] SOFTENG, «Metodologia Scrum para desarrollo de software,» 2021. [En línea]. Available: https://www.softeng.es/es-es/empresa/metodologias-de-trabajo/metodologia-scrum.html.

[10] SOFTENG, «The internet development company,» 2021. [En línea]. Available: https://www.softeng.es/es-es/empresa/metodologias-de-trabajo/metodologia-scrum/proceso-roles-de-scrum.html.

[11] Deloitte., «Artefactos Scrum: las 3 herramientas clave de gestión,» 2022. [En línea]. Available: https://www2.deloitte.com/es/es/pages/technology/articles/artefactos-scrum.html.

[12] T. Asana, «Guía de 6 pasos para la recopilación de requisitos para asegurar el éxito de tu proyecto,» Asana, 15 Noviembre 2021. [En línea]. Available: https://asana.com/es/resources/requirements-gathering.

[13] E. Ledesma, «Proyectum,» 3 Septiembre 2020. [En línea]. Available: https://www.proyectum.com/sistema/blog/scrum-como-escribir-historias-de-usuarios-sin-morir-en-el-intento/#:~:text=Las%20historias%20de%20usuario%2C%20son,esperado%20en%20una%20f

rase%20corta..

[14] J. M. Aguilar, «¿Qué es el patrón MVC en programación y por qué es útil?,» 15 Octubre 2019. [En línea]. Available: https://www.campusmvp.es/recursos/post/que-es-el-patron-mvc-en-programacion-y-por-que-es-util.aspx.

[15] G. Book, «Modelos y uso de Eloquent,» Git Book, 2021. [En línea]. Available: https://richos.gitbooks.io/laravel-5/content/capitulos/chapter7.html.

[16] Yair, «¿Qué es composer y como usarlo?,» Styde, 12 Diciembre 2019. [En línea]. Available: https://styde.net/que-es-composer-y-como-usarlo/.

[17] Y. Muradas, «Que es postman y primeros pasos,» OpenWebinars, 12 Mayo 2022. [En línea]. Available: https://openwebinars.net/blog/que-es-postman/.

[18] «Git,» Desarrollo web, 2021. [En línea]. Available: https://desarrolloweb.com/home/git.

[19] G. B., «Que es GitHub y como usarlo,» Hostinger Tutoriales, 22 Julio 2022. [En línea]. Available: https://www.hostinger.es/tutoriales/que-es-github.

[20] R. celis, «Que es Heroku, Como funciona la plataforma y para que sirve,» Platzi, 2017. [En línea]. Available: https://platzi.com/blog/que-es-heroku/.

[21] P. V. Alcantud, «¿Qué es miro?,» Universidad de Alicante, Alicante.

[22] C. D. Alcolea, «Qué es un ORM,» OpenWebinars, 10 Septiembre 2021. [En línea]. Available: https://openwebinars.net/blog/que-es-un-orm/.

[23] D. Rivera, «Qué es Eloquent en Laravel,» Pleets Blog, 13 Julio 2019. [En línea]. Available: https://blog.pleets.org/article/eloquent-orm-en-laravel.

[24] SCRUMstudy, «¿Cuándo usar SCRUM en tus proyectos?,» tenstep, 2016. [En línea]. Available: https://www.tenstep.ec/portal/articulos-boletin-tenstep/41-scrum/334-cuando-usar-scrum-en-tus-proyectos.

7 ANEXOS

A continuación, se presentan todos los Anexos que se han utilizado en el desarrollo de la backend, los cuales se encuentran divididos de la siguiente manera:

ANEXO I. Resultado del programa anti-plagio Turnitin

ANEXO II. Manual técnico

ANEXO III. Manual de usuario

ANEXO I

Incluir el contenido del Anexo I.

ANEXO II

Manual técnico

Recopilación de requerimientos

A continuación, la **Tabla IV** muestra los requerimientos obtenidos a través de entrevistas y reuniones con el Product Owner durante la etapa de planificación.

Tabla IV: Recopilación de requerimientos.

RECOPILACION DE REQUERIMIENTOS			
ID. Requisito	Nombre del requisito	Descripcion del requisito	Usuario
RR-001	Ingreso al sistema	El sistema debe regir el acceso con las credenciales correctas de cada administrador y vendedor.	Administrador/ Vendedor
RR-002	Control de acceso	El sistema debe permitir ver el acceso de los usuarios "Vendedores".	Administrador
RR-003	Buscar articulo	El sistema debe permitir buscar un articulo especifico dentro de los productos disponibles.	Administrador/ Vendedor
RR-004	Aviso de mercancia	El sistema debe dar una alerta cada vez que un articulo este por debajo de un stock considerable.	Administrador/ Vendedor
RR-005	Revisar inventario	El sistema debe permitir ver que articulos fueron vendidos y/o comprados en una linea de tiempo.	Administrador
RR-006	Gestionar vendedores	El sistema debe permitir gestionar la lista de vendedores.	Administrador
RR-007	Gestionar clientes	El sistema debe permitir ver, añadir, borrar y actualizar la lista de clientes.	Administrador/ Vendedor
RR-008	Gestionar pedidos	El sistema debe permitir ver, añadir, borrar y actualizar la lista de pedidos.	Administrador/ Vendedor
RR-009	Gestionar proveedores	El sistema debe permitir ver, añadir, borrar y actualizar la lista de proveedores.	Administrador/ Vendedor
RR-010	Gestionar comentarios	El sistema debe permitir ver, añadir, borrar y actualizar la lista de comentarios.	Administrador/ Vendedor
RR-011	Gestionar inventario	El sistema debe permitir ver, añadir, borrar y actualizar la lista de inventario.	Administrador/ Vendedor
RR-012	Actualizacion de inventario	El sistema debe permitir ver la actualizacion automaticamente del numero real disponible del inventario	Administrador/ Vendedor
RR-013	Ver pedido	El sistema debe permitir ver el pedido y aceptarlo o denegarlo.	Administrador/ Vendedor
RR-014	Crear usuario	El sistema debe permitir registrar un nuevo usuario con sus credenciales.	Cliente
RR-015	Ver estado de productos	El sistema debe permitir ver el estado de cada producto y su precio dentro de la lista de productos.	Cliente
RR-016	Hacer pedidos	El sistema debe permitir hacer pedidos de productos o servicios.	Cliente
RR-017	Modificar pedidos	El sistema debe permitir modificar pedido hasta antes de que se encuentre aceptado.	Cliente
RR-018	Postear comentarios	El sistema debe permitir ver, añadir, actualizar y/o borrar comentarios	Cliente

Historias de usuario

Finalizada la etapa de recopilación de requerimientos, empieza la elaboración de las Historias de Usuario para el backend. Como consecuencia, se presentan las 18 Historias de usuario creadas a partir de la recopilación de requerimientos, su extensión va desde la **Tabla V** hasta la **Tabla XXII.**

Tabla V: Historia de usuario 1 - Ingreso al sistema.

HISTORIA DE USUARIO	
Identificador (ID): HU-001	**Usuario:** Administrador/Vendedor
Nombre de historia: Ingreso al sistema	
Prioridad de negocio: Alta	**Riesgo en desarrollo:** Alto
Iteración asignada: 1	
Responsable: Kevin Benavides	
Descripción: Quiero poder ingresar al sistema con mis credenciales y recuperar mi contraseña.	
Observaciones: El Administrador/Vendedor puede ingresar al sistema con sus credenciales establecidas según el administrador y también podría recuperar su contraseña con el correo establecido en el registro.	

Tabla VI: Historia de usuario 2 - Control de acceso.

HISTORIA DE USUARIO	
Identificador (ID): HU-002	**Usuario:** Administrador
Nombre de historia: Control de acceso	
Prioridad de negocio: Media	**Riesgo en desarrollo:** Alto
iteración asignada: 1	
Responsable: Kevin Benavides	
Descripción: Quiero poder ver si el usuario "Vendedor" ha iniciado sesión de manera diaria.	
Observaciones: El Administrador podrá ver la fecha y hora del ingreso al sistema por parte del usuario "Vendedor".	

Tabla VII: Historia de usuario 3 - Buscar artículo.

HISTORIA DE USUARIO	
Identificador (ID): HU-003	**Usuario:** Administrador/Vendedor
Nombre de historia: Buscar articulo	
Prioridad de negocio: Alta	**Riesgo en desarrollo:** Alto
iteración asignada: 1	
Responsable: Kevin Benavides	
Descripción: Quiero poder buscar un artículo especifico dentro de la lista de productos disponibles.	
Observaciones: El Administrador/Vendedor podrá buscar un artículo especifico dentro de la lista de productos disponibles.	

Tabla VIII: Historia de usuario 4 - Aviso de mercancía.

HISTORIA DE USUARIO	
Identificador (ID): HU-004	**Usuario:** Administrador/Vendedor
Nombre de historia: Aviso de mercancía	
Prioridad de negocio: Media	**Riesgo en desarrollo:** Alto
Iteración asignada: 1	
Responsable: Kevin Benavides	
Descripción: Quiero poder recibir una alerta cada vez que un artículo este por debajo de un numero establecido para cada producto.	
Observaciones: El Administrador/Vendedor podrá recibir una alerta cada vez que un artículo este en con el numero disponibles por debajo del número designado para ese producto específico.	

Tabla IX: Historia de usuario 5 - Revisar inventario.

HISTORIA DE USUARIO	
Identificador (ID): HU-005	**Usuario:** Administrador
Nombre de historia: Revisar inventario	
Prioridad de negocio: Media	**Riesgo en desarrollo:** Medio
iteración asignada: 1	

Responsable: Kevin Benavides
Descripción: Quiero poder ver que artículos fueron vendidos y/o comprados en una línea de tiempo.
Observaciones: El Administrador podrá ver la fecha y hora de cada producto que fue vendido y comprado para el inventario.

Tabla X: Historia de usuario - Gestionar vendedores.

HISTORIA DE USUARIO	
Identificador (ID): HU-006	**Usuario:** Administrador
Nombre de historia: Gestionar vendedores	
Prioridad de negocio: Media	**Riesgo en desarrollo:** Bajo
iteración asignada: 1	
Responsable: Kevin Benavides	
Descripción: Quiero poder gestionar la lista de vendedores.	
Observaciones: El Administrador podrá ver, añadir, actualizar y/o eliminar cada registro de la lista de vendedores.	

Tabla XI: Historio de usuario 7 - Gestionar clientes.

HISTORIA DE USUARIO	
Identificador (ID): HU-007	**Usuario:** Administrador/Vendedor
Nombre de historia: Gestionar clientes	
Prioridad de negocio: Alta	**Riesgo en desarrollo:** Bajo
iteración asignada: 1	
Responsable: Kevin Benavides	
Descripción: Quiero poder gestionar la lista de clientes.	
Observaciones: El Administrador/Vendedor podrá ver, añadir, actualizar y/o eliminar cada registro de la lista de clientes, por eliminar se refiere a poner el estado en inactivo.	

Tabla XII: Historia de usuario 8 - Gestionar pedidos.

HISTORIA DE USUARIO	
Identificador (ID): HU-008	**Usuario:** Administrador/Vendedor
Nombre de historia: Gestionar pedidos	
Prioridad de negocio: Alta	**Riesgo en desarrollo:** Bajo
iteración asignada: 1	
Responsable: Kevin Benavides	
Descripción: Quiero poder gestionar la lista de pedidos.	
Observaciones: El Administrador/Vendedor podrá ver, añadir, actualizar y/o eliminar cada registro de la lista de pedidos, por eliminar se refiere a poner el estado en inactivo.	

Tabla XIII: Historia de usuario 9 - Gestionar proveedores.

HISTORIA DE USUARIO	
Identificador (ID): HU-009	**Usuario:** Administrador/Vendedor
Nombre de historia: Gestionar proveedores	
Prioridad de negocio: Alta	**Riesgo en desarrollo:** Bajo
iteración asignada: 1	
Responsable: Kevin Benavides	
Descripción: Quiero poder gestionar la lista de proveedores.	
Observaciones: El Administrador/Vendedor podrá ver, añadir, actualizar y/o eliminar cada registro de la lista de proveedores, por eliminar se refiere a poner el estado en inactivo.	

Tabla XIV: Historia de usuario 10 - Gestionar comentarios.

HISTORIA DE USUARIO	
Identificador (ID): HU-010	**Usuario:** Administrador/Vendedor
Nombre de historia: Gestionar comentarios	
Prioridad de negocio: Alta	**Riesgo en desarrollo:** Bajo
iteración asignada: 1	

Responsable: Kevin Benavides
Descripción: Quiero poder gestionar la lista de comentarios.
Observaciones: El Administrador/Vendedor podrá ver, añadir, actualizar y/o eliminar cada registro de la lista de comentarios recibidos por los clientes, por eliminar se refiere a poner el estado en inactivo.

Tabla XV: Historia de usuario 11 - Gestionar inventario.

HISTORIA DE USUARIO	
Identificador (ID): HU-011	**Usuario:** Administrador/Vendedor
Nombre de historia: Gestionar inventario	
Prioridad de negocio: Alta	**Riesgo en desarrollo:** Bajo
iteración asignada: 1	
Responsable: Kevin Benavides	
Descripción: Quiero poder gestionar la lista de inventario.	
Observaciones: El Administrador/Vendedor podrá ver, añadir, actualizar y/o eliminar cada registro de la lista de inventario, por eliminar se refiere a poner el estado en inactivo.	

Tabla XVI: Historia de usuario 12 - Actualización de inventario.

HISTORIA DE USUARIO	
Identificador (ID): HU-012	**Usuario:** Administrador/Vendedor
Nombre de historia: Actualización de inventario	
Prioridad de negocio: Alta	**Riesgo en desarrollo:** Medio
iteración asignada: 1	
Responsable: Kevin Benavides	
Descripción: Quiero poder ver la actualización automáticamente del número real disponible de cada producto del inventario.	
Observaciones: El Administrador/Vendedor podrá ver de manera automática el número real disponible cada producto del inventario, siendo estas afectadas por las acciones de ventas y compras.	

Tabla XVII: Historia de usuario 13 - Ver pedido.

HISTORIA DE USUARIO	
Identificador (ID): HU-013	**Usuario:** Administrador/Vendedor
Nombre de historia: Ver pedido	
Prioridad de negocio: Media	**Riesgo en desarrollo:** Medio
iteración asignada: 1	
Responsable: Kevin Benavides	
Descripción: Quiero poder ver el pedido y aceptarlo o denegarlo.	
Observaciones: El Administrador/Vendedor podrá ver el pedido, sus atributos y aceptarlo o denegarlo, con esto quiere decir que abra otro estado dentro de los atributos de pedidos que indicará este detalle.	

Tabla XVIII: Historia de usuario 14 - Crear usuario.

HISTORIA DE USUARIO	
Identificador (ID): HU-014	**Usuario:** Cliente
Nombre de historia: Crear usuario	
Prioridad de negocio: Media	**Riesgo en desarrollo:** Medio
iteración asignada: 1	
Responsable: Kevin Benavides	
Descripción: Quiero poder registrarme como un nuevo usuario con mis credenciales.	
Observaciones: El Cliente para registrarse e iniciar sesión con sus credenciales, así como recuperar su contraseña.	

Tabla XIX: Historia de usuario 15 - Ver estado de productos.

HISTORIA DE USUARIO	
Identificador (ID): HU-015	**Usuario:** Cliente
Nombre de historia: Ver estado de productos	
Prioridad de negocio: Media	**Riesgo en desarrollo:** Medio
iteración asignada: 1	
Responsable: Kevin Benavides	
Descripción:	

Quiero poder ver la disponibilidad de cada producto y su precio dentro de la lista de productos.
Observaciones: El Cliente podrá ver la disponibilidad, es decir, si existe en inventario, además de su precio de cada producto dentro del inventario.

Tabla XX: Historia de usuario 16 - Hacer pedidos.

HISTORIA DE USUARIO	
Identificador (ID): HU-016	**Usuario:** Cliente
Nombre de historia: Hacer pedidos	
Prioridad de negocio: Alta	**Riesgo en desarrollo:** Bajo
iteración asignada: 1	
Responsable: Kevin Benavides	
Descripción: Quiero permitir hacer pedidos de productos o servicios.	
Observaciones: El Cliente podrá realizar pedidos de productos o servicios, también ver sus pedidos realizados.	

Tabla XXI: Historia de usuario 17 - Modificar pedidos.

HISTORIA DE USUARIO	
Identificador (ID): HU-017	**Usuario:** Cliente
Nombre de historia: Modificar pedidos	
Prioridad de negocio: Media	**Riesgo en desarrollo:** Medio
iteración asignada: 1	
Responsable: Kevin Benavides	
Descripción: Quiero poder modificar pedido.	
Observaciones: El Cliente podrá realizar actualizaciones o eliminar sus pedidos, pero antes de que el usuario "Administrador/Vendedor" haya aceptado este pedido.	

Tabla XXII: Historia de usuario 18 - Postear comentarios.

HISTORIA DE USUARIO	
Identificador (ID): HU-018	**Usuario:** Cliente
Nombre de historia: Postear comentarios	
Prioridad de negocio: Media	**Riesgo en desarrollo:** Bajo
iteración asignada: 1	
Responsable: Kevin Benavides	
Descripción: Quiero poder gestionar mis comentarios.	
Observaciones: El Cliente podrá ver, añadir, actualizar y/o eliminar mis comentarios acera del servicio.	

Product Backlog

La ¡Error! No se encuentra el origen de la referencia.**,** enumera la prioridad de cada equisito que se ha implementado en el backend. Estos requisitos se clasifican de acuerdo con las necesidades del cliente y la complejidad del desarrollo.

ELABORACION DEL PRODUCT BACKLOG				
ID - HU	**HISTORIA DE USUARIO**	**ITERACION**	**ESTADO**	**PRIORIDAD**
HU-001	Ingreso al sistema	1	Enlistado	Alta
HU-002	Control de acceso		Enlistado	Media
HU-003	Buscar articulo	3	Enlistado	Alta
HU-004	Aviso de mercancia	5	Enlistado	Media
HU-005	Revisar inventario	3	Enlistado	Media
HU-006	Gestionar vendedores	2	Enlistado	Media
HU-007	Gestionar clientes	2	Enlistado	Alta
HU-008	Gestionar pedidos	2	Enlistado	Alta
HU-009	Gestionar proveedores	2	Enlistado	Alta
HU-010	Gestionar comentarios	3	Enlistado	Alta
HU-011	Gestionar inventario	2	Enlistado	Alta
HU-012	Actualizacion de inventario	2	Enlistado	Alta
HU-013	Ver pedido	3	Enlistado	Media
HU-014	Crear usuario	1	Enlistado	Media
HU-015	Ver estado de productos		Enlistado	Media
HU-016	Hacer pedidos	4	Enlistado	Alta
HU-017	Modificar pedidos	4	Enlistado	Media
HU-018	Postear comentarios	4	Enlistado	Media

ELABORACION DE SPRINT BACKLOG					
ID - SB	**NOMBRE**	**ID - HU**	**HISTORIA DE USUARIO**	**TAREAS**	**TIEMPO ESTIMADO**
SB - 000	Configuración del entorno de desarrollo	N/A	N/A	Creación del proyecto en VS Code Creación de la base de datos MySQL Diagrama de flujo del sistema	20H
SB - 001	Módulo de conexión entre base de base de datos y el ORM Eloquent de Laravel.	N/A	N/A	Migraciones, Seeders y Factories, Modelos Relaciones a nivel de base de datos y a nivel de Eloquent	30H
SB - 002	Módulos de autenticación y registro usuarios.	HU - 001 HU - 014	Ingreso al sistema Registro de usuarios	Inicio de sesión para todos los roles de usuario. Registro de usuarios y verificación de email. Reestablecimiento de contraseñas.	40H
SB - 003	Módulos Endpoints de Usuarios, Productos, Pedidos, Proveedores, Roles, Tipos, Comentarios y Deudas.	HU - 006 HU - 009 HU - 003 HU - 005 HU - 013	Gestionar clientes, vendedores, pedidos, inventario. actualización de inventario. Buscar articulo Revisar y gestionar inventario Ver pedido	Creacion de controladores API Codificación de métodos de controladores API. Creación de rutas API. Para los modelos Usuario, Productos, Pedidos, Proveedores, Roles, Tipos, Comentarios y Deudas	60H
SB - 004	Módulo de Gates y Policies.	HU - 016 HU - 017 HU - 018	Hacer y modificar pedidos Postear comentarios	Construcción del diagrama de casos de uso. Codificación de “Gates” de los roles: Administrador, Vendedor o pasante y Cliente. Codificación de “Policies” de recursos destinados a modificar solamente su autor.	30H
SB - 005	Módulo de notificaciones con correo electrónico.	HU - 004	Aviso de mercadería.	Módulo de alerta de mercadería mínima.	20H
SB - 006	Módulos de Pruebas	N/A	N/A	Pruebas unitarias a rutas.	20H

				comprobación de llamadas al servidor por media de la API utilizando una aplicación cliente. Análisis de respuestas de cada componente del sistema que realice consultas.	
SB - 007	Despliegue del proyecto de API REST en un servidor web público.	N/A	N/A	Configuración del servidor web apache. Verificación del rendimiento del sistema en el servidor. Comprobación de la seguridad del sistema. Prueba del consumo de la API desde otro sistema.	20H
TOTAL					240H

ANEXO III

Manual de usuario

A continuación, para acceder al Manual de Usuario se debe ingresar a la URL del repositorio de GitHub, donde se encuentra el código fuente del proyecto y luego ubicarse en el README, aquí estará albergado el enlace directo hacia el Manual de Usuario.

URL del repositorio de GitHub.

https://github.com/Ferkevin07/Proyecto_CiPU.git