ESCUELA POLITÉCNICA NACIONAL

ESCUELA DE FORMACIÓN DE TECNÓLOGOS

DESARROLLO DE SISTEMA WEB Y APLICACIÓN MOVIL PARA COMANDAS EN LA CAFETERÍA EPN

DESARROLLO DE UN *BACKEND*

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

MIGUEL EDUARDO CUENCA CHAMBA

DIRECTOR: ING. YADIRA GUISSELA FRANCO ROCHA, Mg.

DMQ, marzo 2023

CERTIFICACIONES

Yo, Miguel Eduardo Cuenca Chamba 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.

MIGUEL EDUARDO CUENCA CHAMBA

miguel.cuenca01@epn.edu.ec

ecuenca4@gmail.com

Certifico que el presente trabajo de integración curricular fue desarrollado por Miguel Eduardo Cuenca Chamba, bajo mi supervisión.

Ing. Yadira Franco R Mg.

DIRECTOR

yadira.franco@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.

Cuenca Chamba Miguel Eduardo

DEDICATORIA

Este trabajo está dedicado a mi familia, donde se representa el resultado de años de esfuerzo, dedicación y perseverancia, y nada de esto habría sido posible sin el apoyo incondicional de mi familia. Ellos me han animado, motivado y han creído en mí en cada paso del camino. Siempre han estado ahí para mí, en las buenas y en las malas, y me han brindado el aliento y la fuerza que necesitaba para seguir adelante. Donde cada página de este trabajo refleja el sacrificio y la dedicación que han hecho por mí, y representa su inmenso amor y apoyo. Espero que este trabajo les haga sentir orgullosos y les muestre que su esfuerzo y su dedicación no han sido en vano. Agradezco a mi familia por ser mi mayor motivación y mi mayor logro en la vida. Los amo con todo mi corazón.

Cuenca Chamba Miguel Eduardo

AGRADECIMIENTO

Quiero expresar mi más profundo agradecimiento a los ingenieros de la Escuela Politécnica Nacional, quienes me han guiado, enseñado y motivado en este camino académico. Sus conocimientos, experiencia y dedicación han sido una fuente de inspiración para mí. Gracias por su paciencia, por su apoyo y por creer en mí. Este logro no habría sido posible sin su ayuda y su orientación. Me siento afortunado de haber tenido la oportunidad de aprender de profesionales tan excelentes.

También quiero agradecer a mi familia por su amor, apoyo y paciencia durante todo este proceso. Gracias por estar ahí para mí, por animarme cuando lo necesitaba y por comprender cuando estaba ocupado con mis estudios. Agradezco a mis padres por inculcarme valores y principios que me han guiado en todo momento. Gracias a mis abuelos y demás familiares por su amor y su apoyo incondicional. Este logro no habría sido posible sin su amor y su ayuda.

Por último, quiero agradecer a Dios por todas las bendiciones que ha derramado sobre mí durante este proceso. Gracias por guiarme, por fortalecerme en los momentos difíciles y por darme la perseverancia y la fe necesarias para lograr este objetivo. Sin tu amor y tu guía, nada de esto habría sido posible. Agradezco a Dios por sus bendiciones y por su protección en todo momento.

Cuenca Chamba Miguel Eduardo

ÍNDICE DE CONTENIDO

RESUMEN

El presente trabajo, de integración curricular, tiene como objetivo desarrollar e implementar un *backend,* este permite el manejo de un sistema de comandas para la cafetería de la Escuela Politécnica Nacional, con el fin de mejorar la eficiencia en la toma y entrega de pedidos y brindar una mejor experiencia a los clientes. Previamente al desarrollo, se llevó a cabo un análisis del funcionamiento actual de la cafetería y se identificaron las principales problemáticas en el proceso de toma de pedidos y preparación de estos. A partir de ello, se planteado el desarrollo de un sistema que permitiera optimizar estos procesos y reducir los tiempos de espera para los clientes.

El sistema de comandas permite el manejo de los pedidos, por lo tanto, en el presente trabajo se desarrolla el componente *backend*. A través de este componente, los clientes pueden: registrarse, realizar su pedido, seleccionar los productos de un menú previamente establecido y adjuntar un documento que certifique el pago de su pedido, esto permite que la información del *backend* sea usada por un *frontend* o una aplicación móvil, permitiendo que el manejo de pedidos se digitalice.

El presente documento se estructura de la siguiente forma: Primera sección: antecedentes, objetivos, alcance y marco metodológico. Segunda sección: implementación de la metodología ágil *SCRUM*, desarrollo de la base de datos, arquitectura utilizada y tecnologías empleadas para el desarrollo del componente. Para finalizar, la sección se especifica las actividades desarrolladas, resultados obtenidos y recomendaciones que se han generado en el desarrollo de este trabajo de integración.

PALABRAS CLAVE: Scrum, backend, mongoDB, endpoints, gestion.

ABSTRACT

The objective of this curricular integration work is to develop and implement a backend that will allow the management of an order system for the Escuela Politécnica Nacional cafeteria, to improve the efficiency in the taking and delivery of orders and provide a better experience to customers. Prior to the development, an analysis of the current operation of the cafeteria was carried out and the main problems in the order taking and preparation process were identified. Based on this, the development of a system that would optimize these processes and reduce waiting times for customers was proposed.

The order system developed allows us to manage customer orders, therefore, in this work the backend component was developed. Through this component, customers can register, place their order, select the products they want from a previously established menu and attach a document that certifies the payment of their order, this will allow us that all the backend information can be used by a frontend or a mobile application and allowing the management of orders are digitally with the use of modern technologies.

This document is structured as follows: In the first section we have the background, objectives, scope, and methodological framework of the project. The second section presents the implementation of the agile SCRUM methodology, the development of the database, the architecture used, and the technologies used for the development of the component. Finally, we have section three, which specifies the activities developed, the results obtained, conclusions and recommendations that have been generated in the development of this integration work.

KEYWORDS: Scrum, backend, mongoDB, endpoints, gestion.

1 DESCRIPCIÓN DEL COMPONENTE DESARROLLADO

En la actualidad, encontramos con tecnologías que permiten interactuar con los usuarios de una manera simple logrando facilitar la comunicación para poder brindar un buen servicio, es el caso de los restaurantes donde la toma de pedidos se realiza de una manera manual, donde se toma nota del pedido del cliente con ayuda de un mesero, un bolígrafo y una hoja de papel. Este sistema ha sido utilizado por un largo periodo de tiempo, pero el sector de los restaurantes está en constante cambio, sobre todo debido a la evolución de las preferencias de los clientes. Por ello, es fundamental que cualquier empresa de restauración próspera se mantenga al día de los avances más recientes en tecnología para restaurantes [1].

Actualmente, en el ámbito del desarrollo de software, es común que los sistemas sean interconectados entre la web o una aplicación móvil, lo cual facilita la interacción de los usuarios con el servicio. En efecto, múltiples plataformas que se usan hoy en día como: Facebook, Amazon, Tik Tok, etc., ejecutan varios servidores para poner en uso el *backend,* esto les permite consumir la información y facilitar la conexión con el *frontend* por medio de los *endpoints* de la API. Una API, en términos más sencillos, permite a una aplicación recuperar datos de varios servidores, lo que suele ser crucial cuando se crean aplicaciones a gran escala. Al utilizar una API, los desarrolladores pueden mejorar la visibilidad y fiabilidad de la utilización de sus recursos [2].

Como parte del proyecto de integración curricular, se ha desarrollado un *backend* basado en *JavaScript* para la cafetería de la EPN. Para ello se han creado una serie de *endpoints* que encapsulan los datos más importantes, lo que permite al *frontend* consumir información en función de los permisos del usuario, utilizando cualquier tecnología o *framework*. Esto facilita el pedido digital y la entrega de los productos y servicios de la cafetería, poniendo a disposición de los usuarios toda la información y los recursos pertinentes.

1.1 Objetivo general

Desarrollar un *backend* para un sistema web de comandas para la cafetería de la EPN.

1.2 Objetivos específicos

1. Establecer los requerimientos para el *backend*.
2. Diseñar la base de datos y la arquitectura para el *backend*
3. Implementar el *backend* en base a los requerimientos con la ayuda de un lenguaje de programación.
4. Ejecutar pruebas para comprobar que todas sus funciones estén implementadas.
5. Despliegue del *backend* a producción.

1.3 Alcance

La nueva era digital permite la implementación de un negocio a la red, facilitando la interacción con los usuarios, llegando a automatizar ciertas tareas que se las realiza manualmente. Por lo tanto, para el desarrollo de este componente se aplica el uso y elaboración de un conjunto de API's las cuales brindan un gran número de ventajas, facilitando la seguridad, escalabilidad, integración y consumo por las nuevas tecnologías del lado del cliente o *frontend.* Por lo tanto, existe una separación entre cliente/servidor, lo que permite que las API's realicen cambios en una parte de una aplicación sin afectar a otras áreas, es un aspecto importante del desarrollo de software. El uso de API permite a los desarrolladores modificar características y funcionalidades específicas sin afectar al estado general de la aplicación. Esto también permite realizar pruebas de API de forma independiente sin interrumpir otras operaciones [3].

Con la finalidad de emplear tecnologías modernas y aprovechando los beneficios que ofrecen las API's. En este proyecto se implementa un *backend* tomando en cuenta los requerimientos necesarios para que el funcionamiento de la cafetería de la EPN pueda ofrecer, recibir y despachar los pedidos, con el propósito de integrar los servicios de la cafetería en un medio tecnológico y digital.

Finalmente, al tiempo que mantiene la seguridad, integridad y coherencia de los datos introducidos por el *frontend* en la base de datos, este *backend* que se ha implementado provee de un conjunto de *endpoints* a continuación:

Usuarios del *backend*

- Administrador
- Cliente

- Cocinero

Usuario administrador

- Implementación de algunos *endpoints* para presentar los servicios disponibles (almuerzos, platos fuertes, guarnición, bebidas).
- Implementación de algunos *endpoints* para ingresar al sistema (iniciar sesión, cerrar sesión y recuperar contraseña).
- Implementación de algunos *endpoints* para administrar los menús.
- Implementación de algunos *endpoints* para administrar al usuario cocinero.
- Implementación de algunos *endpoints* para ver el pedido realizado por cliente y poder aprobar el pedido para que sea despachado.

Usuario cliente

- Implementación de algunos *endpoints* para interactuar con el sistema (iniciar sesión, cerrar sesión y recuperar contraseña).
- Implementación de algunos *endpoints* para modificar el perfil.
- Implementación de algunos *endpoints* para registro de usuario.
- Implementación de algunos *endpoints* para gestionar pedido.
- Implementación de algunos *endpoints* para cargar el comprobante de pago.
- Implementación de algunos *endpoints* para ver el estado del pedido.

Usuario cocinero

- Implementación de algunos *endpoints* para interactuar con el sistema (iniciar sesión, cerrar sesión y recuperar contraseña).
- Implementación de algunos *endpoints* para modificar el perfil.
- Implementación de algunos *endpoints* para visualizar los pedidos realizados por el usuario cliente.
- Implementación de varios *endpoints* para gestionar el estado del pedido.
- Implementación de varios *endpoints* para agregar los platos del menú.

1.4 Marco Teórico

Backend

En el desarrollo de páginas web la mayoría de ellas se fragmentan en dos partes importantes: el *frontend* dando la parte visible para el usuario y el *backend* que es la parte trasera que el usuario no puede ver. Estos dos elementos coexisten, donde cada uno depende del otro para su perfecto funcionamiento.

De esta manera, el *backend* se refiere a la configuración lógica de una página web o aplicación que no es visible para los usuarios finales. Esto incluye tareas como acceder a bases de datos, procesar datos introducidos por el usuario y ejecutar scripts. Algunas personas consideran que la programación *backend* abarca todos los aspectos de una página web o aplicación que no son directamente visibles para los usuarios finales. [4].

MongoDB

MongoDB es un tipo de sistema de gestión de bases de datos NoSQL o no relacional. A diferencia de los sistemas SQL tradicionales que utilizan tablas para almacenar datos, MongoDB emplea un modelo orientado a documentos, que se almacenan en BSON, una representación binaria de JSON. MongoDB no se basa en un esquema predefinido, lo que permite una mayor flexibilidad en el almacenamiento y la recuperación de datos. [5].

En MongoDB, los datos se organizan en bases de datos que contienen colecciones, que a su vez se componen de documentos individuales. Cada documento dentro de una colección puede tener un número variable de campos y el contenido y tamaño de cada documento puede diferir entre sí. Esta flexibilidad en la estructura de los documentos permite un almacenamiento de datos eficiente y versátil en MongoDB [6].

NoSQL

Los sistemas de almacenamiento NoSQL difieren de los sistemas de bases de datos relacionales en que no se basan en el lenguaje SQL para realizar consultas. En su lugar, los sistemas NoSQL utilizan una serie de lenguajes de consulta que se adaptan al modelo de datos específico y a la estructura de almacenamiento del sistema. Esto permite una mayor flexibilidad y eficiencia en el almacenamiento y recuperación de datos. Aunque no es que los sistemas NoSQL no puedan utilizar SQL, lo hacen de manera complementaria y no como herramienta principal para consultas. De ahí que estos sistemas se conozcan como NoSQL o no solo SQL [7].

Las bases de datos NoSQL están diseñadas para satisfacer las necesidades de las aplicaciones que requieren un manejo rápido y eficiente de grandes cantidades de datos. Esto se consigue mediante el uso de modelos de datos flexibles y un acceso a los datos

de baja latencia. Para facilitar este rendimiento, se relajan algunas de las restricciones de consistencia de datos que están presentes en otros tipos de bases de datos. Esto permite una mayor escalabilidad y adaptabilidad a las necesidades cambiantes de los datos [8].

API

Una API (*Application programming interface*) es un programa informático que facilita la comunicación y el intercambio de datos entre dos aplicaciones diferentes. Esto elimina la necesidad de recrear una aplicación entera para acceder a sus datos [9]. Las API suelen describirse en términos de arquitectura de cliente y servidor, donde el cliente es la aplicación que envía una solicitud y el servidor es la aplicación que envía una respuesta [10].

API REST

Las API REST (*Representational State Transfer*) se encuentran entre las más populares y versátiles que se utilizan actualmente en la web. Los clientes envían peticiones de datos al servidor, que los utiliza para realizar funciones internas y devolver los datos de salida al cliente. La característica que define a las API REST es que no tienen estado, lo que significa que los servidores no almacenan los datos del cliente entre una solicitud y otra. Las peticiones del cliente se estructuran de forma similar a las URL que se escriben en un navegador web, y las respuestas del servidor suelen consistir en datos simples en lugar de una representación gráfica de una página web [10].

Node.js

Node.js es un entorno de ejecución multiplataforma del lado del servidor basado en *JavaScript*. Está diseñado para crear aplicaciones escalables con una arquitectura basada en eventos que permite el establecimiento y la gestión de múltiples conexiones simultáneamente. Esta característica elimina las preocupaciones sobre el bloqueo de procesos y los bloqueos [11].

Express

Express.js es un *framework* de Node.js para el *backend* que ofrece una amplia variedad de características y herramientas, para desarrollar aplicaciones escalables. Su sistema de enrutamiento y características simplificadas permiten adaptarlo a los requisitos específicos de la aplicación que se está desarrollando, incluso con componentes y partes más avanzadas. Debido a su gran popularidad, Express.js cuenta con un sólido soporte de la comunidad, además de recursos y paquetes que brindan soluciones para cualquier desarrollo. Además, cuenta con el respaldo de Google, es una elección popular entre los desarrolladores de Node.js. El entorno de código abierto de Express.js, también, permite

que los desarrolladores creen paquetes y recursos extensibles que puedan ser utilizados en este *framework* [12].

Typescript

Para el manejo de estas herramientas se hace uso del lenguaje de *JavaScript*, que se trata de un lenguaje dinámico, potente y flexible. Justamente la flexibilidad al momento de desarrollar puede complicar la ejecución y desarrollo de los proyectos, aquí es donde *TypeScript* se inclina hacia una dirección, siendo una extensión de *JavaScript* que incorpora tipado estático y objetos basados en clases, y en última instancia se traduce a *JavaScript* plano a través de la compilación [13].

TypeScript es conocido por su capacidad para gestionar diversos tipos de datos, lo que lo convierte en un lenguaje adaptable y seguro. Su código puede validarse antes de la ejecución, lo que se traduce en un alto grado de precisión y fiabilidad, con una incidencia mínima de errores [14].

2 METODOLOGÍA

El método de investigación, conocido como estudio de casos, implica la recopilación de información y su análisis detallado para comprender cómo funciona un caso específico en una situación determinada. Estos estudios se enfocan en un caso particular de investigación y se presentan de manera disciplinada y cualitativa, lo que permite la creación de hipótesis y teorías fundamentadas en una investigación exhaustiva.

Por lo tanto, en el desarrollo de software las organizaciones que se dedican plenamente a llevar a cabo una transformación digital integral suelen integrar, aplicar y construir metodologías ágiles en todas sus divisiones. Esto se hace para suministrar productos y/o servicios de alta calidad a costes reducidos, al tiempo que se entregan en plazos mucho más cortos [15].

2.1 Metodología de Desarrollo

Para ejecutar este proyecto se ha decidido emplear una metodología ágil conocida como *SCRUM*. Este enfoque consiste en realizar una serie de tareas de forma recurrente con el objetivo principal de promover el esfuerzo colaborativo, es decir, fomentar el trabajo en equipo [16]. De esta manera, se pretende lograr el mejor resultado del proyecto en el tiempo acordado.

Roles

Los roles de *Scrum* son responsables, por igual, en entregar un incremento de producto que sea valioso en cada Sprint. Este grupo pequeño de profesionales se distribuye entre

los siguientes tres roles *Scrum: Developers* o desarrolladores, *Scrum Product Owner y Scrum Máster* [17].

Estos tres perfiles claves:

- *Project Owner:* es quien ha encargado el proyecto al equipo de trabajo, su misión es establecer las prioridades y los objetivos [18].
- *Scrum Máster:* es el líder del equipo, su trabajo se centra en supervisar, controlar plazos y procedimientos, además, eliminar cualquier obstáculo [18].
- *Developers team*: es el equipo de trabajo que completa las tareas en cada Sprint [18].

Haciendo referencia a lo anteriormente mencionado la **TABLA I** muestra la asignación de los roles.

TABLA I: Asignación de roles

Rol	Integrantes
Product Owner	Ing. Yadira Franco, Mg.
Scrum Master	Ing. Yadira Franco, Mg.
Development Team	Miguel Cuenca

Artefactos

En el marco de *Scrum*, los artefactos se refieren a los productos generados como consecuencia de la aplicación de *Scrum*. Es de destacar que estos artefactos se elaboran deliberadamente para mejorar la claridad y la visibilidad de la información crucial. Es inevitable que todos tengan el mismo entendimiento de los artefactos [19]. Por esta razón, a continuación, se expone los artefactos para el desarrollo de este proyecto.

Recopilación de Requerimientos

Es el procedimiento de identificación de las necesidades precisas del proyecto de principio a fin. Aunque este proceso se lleva a cabo en la fase inicial del proyecto, continúa durante toda la duración de este [20]. En la **TABLA II**, se puede apreciar el modelo que se ha utilizado para la Recopilación de requerimientos, cabe detallar que la tabla completa se encuentra en el ANEXO II del presente documento.

TABLA II: Modelo para la Recopilación de Requerimientos

RECOPILACIÓN DE REQUERIMIENTOS		
TIPO DE SISTEMA	**ID RR**	**DESCRIPCIÓN**
BACKEND	**RR001**	Los tres tipos de usuario: administrador, cliente y cocinero, necesitan generar algunos *endpoints,* que serán para visualizar la información general de la cafetería.

Historias de Usuario

Una historia de usuario es una narración informal que describe una característica del software desde el punto de vista del usuario final. Es esencial redactar estas historias en un lenguaje no técnico para proporcionar contexto al equipo de desarrollo [21]. En la **TABLA III**, se muestra el modelo que se ha utilizado para la historia de usuarios. Cabe detallar que la tabla completa se encuentra en el presente documento.

TABLA III: Historia de usuario 01 - Información de la cafetería

HISTORIA DE USUARIO	
Identificador (ID): HU001	**Usuario**: Administrador, cliente y cocinero
Nombre Historia: Información de la cafetería	
Prioridad en negocio: Media	**Riesgo en desarrollo:** Media
Iteración Asignada: 1	
Responsable (es): Miguel Cuenca	
Descripción: En el *backend* se cuenta con tres perfiles los cuales son: administrador, cliente y cocinero, los cuales pueden visualizar la información general de la cafetería. En este apartado se tiene varios *endpoints* que se permiten consumir por aplicaciones del lado del cliente.	
Observación: Los *endpoints* que se mencionan son el de los módulos de iniciar sesión y registro.	

Product Backlog

El *Product Backlog* es una lista ordenada de lo que se necesita en el proyecto y es la fuente de requisitos para cualquier cambio que se realice en este. Él es un elemento dinámico y en constante evolución del proyecto, que se adapta y desarrolla continuamente para seguir siendo relevante, competitivo y valioso. Es importante tener en cuenta que mientras exista

el proyecto, el *backlog* del producto también seguirá existiendo y actualizándose. Los atributos del *Product Backlog* son: descripción, orden, estimación y valor [22]. En la **TABLA IV**, se puede apreciar el modelo que se utilizado para la *Product Backlog*, cabe detallar que la tabla completa se encuentra en el **ANEXO II** del presente documento.

TABLA IV: Modelo del *Product Backlog*

ID – HU	HISTORIA DE USUARIO	ITERACIÓN	PRIORIDAD	ESTADO
HU001	Información de la cafetería	1	Finalizado	Mediana
HU002	Gestión de usuario cocinero. Iniciar sesión, cerrar sesión y recuperar contraseña.	1	Finalizado	Alta

Sprint Backlog

El *Sprint Backlog*, también conocido como Pila, se refiere a la colección de elementos del *Product Backlog* que han sido elegidos para un *Sprint* específico. Por lo tanto, no puede existir sin un *Product Backlog* previamente priorizado. El Equipo de desarrollo utiliza el *Sprint Backlog* como una previsión de las funcionalidades que se han incluido en el próximo incremento, junto con el trabajo necesario para cumplir los requisitos definidos y satisfacer los criterios para ser considerado finalizado [23]. En la **TABLA V**, se puede apreciar el modelo que se han utilizado para la *Product Backlog*. Cabe detallar que la tabla completa se encuentra en el **ANEXO II** del presente documento.

TABLA V: Modelo del *Sprint Backlog*

ELABORACIÓN DE *SPRINT BACKLOG*						
ID – SB	**NOMBRE**	**MÓDULO**	**ID-HU**	**HISTORIA DE USUARIO**	**TAREA**	**TIEMPO ESTIMADO**
SB000	Configuración del ambiente de desarrollo.	----------	-------	----------	• Definición de requerimientos. • Estructura del proyecto. • Elaboración de la base de datos. • Definición de los respectivos usuarios.	20H

2.2 Diseño de la arquitectura

El diseño de una arquitectura es de gran importancia, se trata de las bases para el funcionamiento del proyecto, donde depende de esta para el correcto funcionamiento y poder dar mantenimiento luego de que salga a producción. La arquitectura de software de un sistema describe la disposición o composición del sistema y aclara su comportamiento previsto [24]. Por lo tanto, es necesario para este proyecto tener claro que arquitectura es la adecuada para poder codificar el *backend* con la ayuda de tecnologías actuales.

Patrón arquitectónico Modelo Vista Controlador (MVC)

Se trata de una estructura de software que se fundamenta en la segmentación del código en capas especializadas en tareas concretas, con el propósito de separar la interfaz de usuario, la lógica del proyecto y los datos en tres elementos independientes. A continuación, se detallan estas capas:

- **Modelo**: Se encarga de manejar los datos y la lógica del negocio. Aquí se realizan todas las operaciones relacionadas con la base de datos, la validación de datos, la implementación de reglas de negocio y demás procesos que no están directamente relacionados con la vista o la interacción del usuario.
- **Vista**: Esta capa se encarga de mostrar la información al usuario final. Aquí se definen todos los elementos visuales con los que el usuario interactúa, como botones, formularios e imágenes.
- **Controlador**: Esta capa se encarga de gestionar las interacciones del usuario, comunicarlas con el modelo y la vista correspondientes. Aquí se implementa la lógica que define cómo la aplicación responde a las acciones del usuario.

Por lo tanto, se muestra en la Fig. 1 el diseño del esquema arquitectónico, que se ha ejecutado de manera conjunta con las herramientas empleadas para su elaboración e implementación en el entorno de producción.

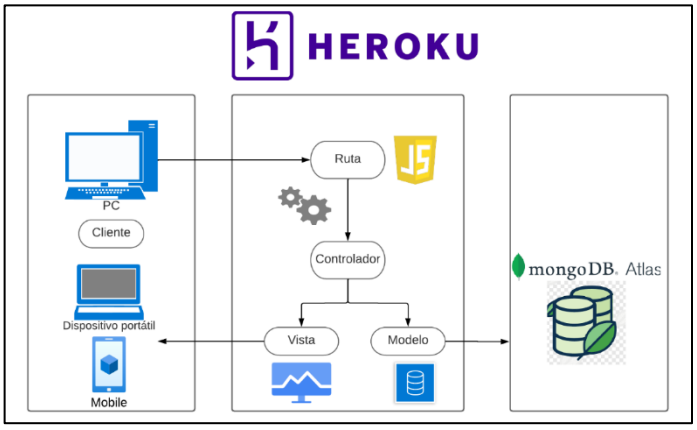


Fig. 1 Patrón arquitectónico

2.3 Herramientas de desarrollo

Luego de determinar que patrón arquitectónico se implementa, es de gran importancia la elección de las distintas herramientas para el desarrollo. El *backend* consta de varias etapas que dependen de la tarea específica que se esté realizando, y cada una de estas etapas implica distintas herramientas de desarrollo de software [25]. Por esta razón en la **TABLA VI**, se muestra las distintas herramientas que se utilizan para el desarrollo del *backend* y para que exista comunicación con la base de datos.

TABLA VI: Herramientas para el desarrollo del *backend*

HERRAMIENTA	JUSTIFICACIÓN
Node.js	Node.js es un entorno multiplataforma de código abierto que permite ejecutar código *JavaScript* fuera de un navegador web. Su versatilidad permite a los desarrolladores emplear *JavaScript* para crear aplicaciones del lado del servidor [26].
MongoDB	Se trata de una herramienta poderosa esta ofrece un recurso eficiente para el manejo de grandes volúmenes de información [27].
Express.js	Express es una estructura mínima y adaptable que opera en Node.js. Proporciona una amplia diversidad de funcionalidades para el desarrollo, manteniéndose flexible y adaptable [28].
Nodemailer	*Nodemailer* es una biblioteca para aplicaciones que facilita el envío de correos electrónicos de manera simple [29].
Mongoose	*Mongoose* es un paquete para Node.js que posibilita las consultas para una base de datos de MongoDB, y que cuenta con funciones adicionales como validaciones, creación de *queries*, middlewares, conversión de tipos y otras características que mejoran la funcionalidad de la base de datos [30].
Cloudinary	*Cloudinary* proporciona una interfaz sencilla de utilizar para transformar, optimizar y entregar imágenes, lo que facilita el almacenamiento de estas [31].
npm	*Npm* es un administrador de paquetes que permite gestionar los proyectos de Node.js disponibles para uso público. Los proyectos que se encuentran en el registro de *npm* son denominados paquetes [32].

Postman	Es una plataforma que simplifica la creación y uso de APIs, lo que resulta muy útil para la programación ya que permite realizar pruebas y verificar el correcto funcionamiento de los proyectos [33].

3 RESULTADOS

Esta sección describe los resultados logrados en la implementación de los *endpoints* en el *backend* y sus resultados de las pruebas de implementar el proceso a producción. Sin embargo, cada resultado se presenta a través de *Sprints*.

3.1 *Sprint* 0. Configuración del ambiente de desarrollo

De acuerdo con lo establecido en el *Sprint Backlog* las actividades a realizar son:

- Recopilación y definición de requerimientos.
- Elaboración de la Base de datos en MongoDB.
- Estructura del proyecto
- Roles de usuario

Recopilación y definición de requerimientos

Generar *endpoints* de información general

En el *backend* a través de varios métodos y su respectiva ruta acceso, permite obtener y gestionar la información de la cafetería para el usuario administrador, cliente y cocinero.

Generar *endpoints* para el registro de usuarios

Se desarrolla en el *backend* un método, el cual permite el registro de los usuarios de acuerdo con su rol, los campos solicitados se definen en la base de datos, cabe detallar que el administrador es el encargado de registrar al usuario cocinero.

Generar *endpoints* que permiten iniciar sesión, cerrar sesión y recuperar contraseña.

En el *backend* se desarrolla un método, donde los usuarios administradores, cliente y cocinero, pueden ingresar al sistema con su respectivas credenciales, acompañado de un método para poder cerrar sesión y, para finalizar, con un método para cambiar contraseña.

Generar *endpoints* para gestionar al usuario cocinero

En el *backend* se desarrolló varios métodos, donde el usuario administrador puede crear, eliminar y visualizar los datos del usuario cocinero, el cual luego de su creación será asignado a la elaboración de los pedidos asignados por el administrador.

Generar *endpoints* para el manejo de pedidos

En el *backend* se desarrolla varios métodos, donde el usuario administrador visualiza el estado de los pedidos, incluye información del cliente, fecha del pedido, comprobante de pago, total a pagar y que acción tomar referente al pedido. La acción que tomar puede ser aprobar el pedido y asignar a un cocinero o declinar el pedido.

Generar *endpoints* para crear el menú.

En el *backend* se desarrolla varios métodos, el usuario administrador puede agregar diferentes productos y los diferentes platos que ofrece. Luego del ingreso de la información de los platos, se puede crear un menú asignando los platos previamente ingresados de acuerdo con el menú que deseamos crear.

Generar *endpoints* para modificar perfil.

En el *backend* se desarrolla varios métodos, donde los usuarios pueden visualizar sus datos ingresados y actualizarlos.

Generar *endpoints* para generar pedidos

En el *backend* se desarrolla varios métodos, donde el usuario cliente puede realizar su pedido, verificar el valor a cancelar y poder subir su comprobante de pago. Adicionalmente se puede verificar el estado del pedido el cual tiene algunos estados posibles como: validando pedido, pedido en proceso y pedido finalizado. Cabe tener en cuenta que el cliente luego de realizar su pedido no puede ser modificado.

Generar *endpoints* para el usuario cocinero

En el *backend* se desarrolla varios métodos, el usuario cocinero puede visualizar los pedidos asignados, estos pedidos asignados son realizados por el usuario administrador, por lo tanto, el usuario cocinero puedo cambiar el estado de los pedidos como son: en proceso y finalizado.

Seguidamente en la **Fig. 2, Fig. 3, Fig. 4** se detalla los diferentes métodos, done cada usuario puede interactuar con el *backend.*

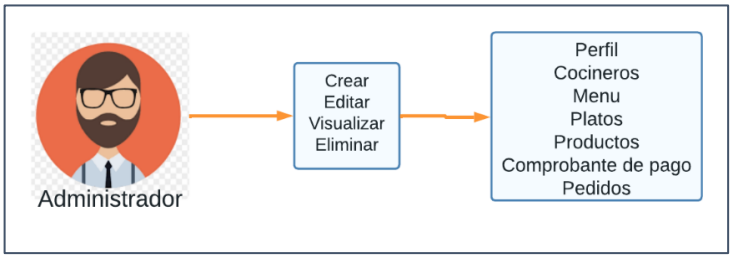


Fig. 2 Perfil administrador

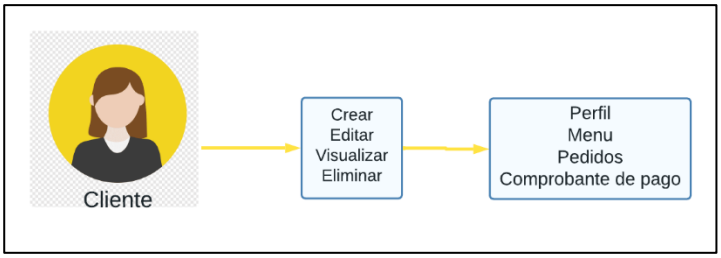


Fig. 3 Perfil cliente

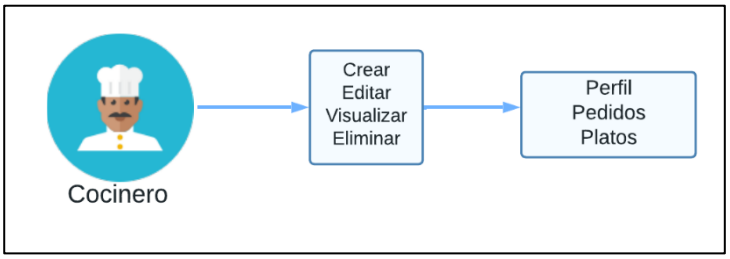


Fig. 4 Perfil cocinero

Elaboración de la Base de datos en MongoDB

Para el manejo de los datos de este proyecto, se ha optado por la plataforma de MongoDB que trata de un sistema local o en la nube para el manejo de datos NoSQL. A diferencia de las bases de datos relacionales como SQL, las bases de datos NoSQL no utilizan tablas ni registros, y no requieren una estructura rígida. Esto ofrece una mayor flexibilidad a la hora de diseñar el esquema y establecer relaciones entre los elementos de datos [34]. En la **Fig. 5** se representa la base de datos NoSQL, que se ha implementado para el manejo de los datos con el *backend,* cabe recalcar que la base completa se puede visualizar en el **ANEXO II** del presente documento.

Fig. 5 Base de datos No relacional

Estructura del proyecto

Se utiliza *Visual Estudio Code* como entorno de desarrollo para crear los *endpoints*. La estructura de cada directorio se ha establecido a través del panel de inicio, siguiendo el patrón arquitectónico elegido previamente. La **Fig. 6** ilustran la estructura actual del proyecto.

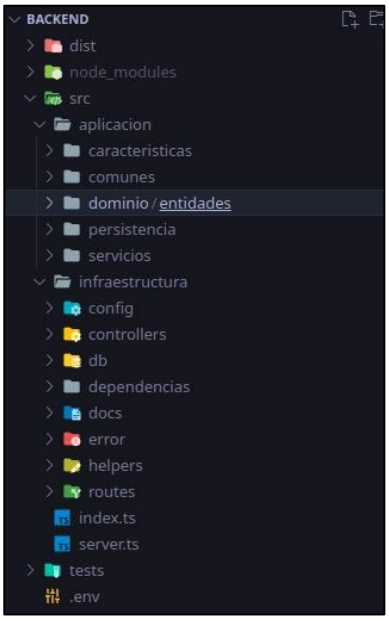


Fig. 6 Estructura del proyecto.

Roles de usuario

A continuación, la **Fig. 7,** puede visualizar los tres tipos de usuario junto a su respectivo módulo, que cada uno tiene acceso, la disponibilidad de cada módulo depende del rol del usuario.

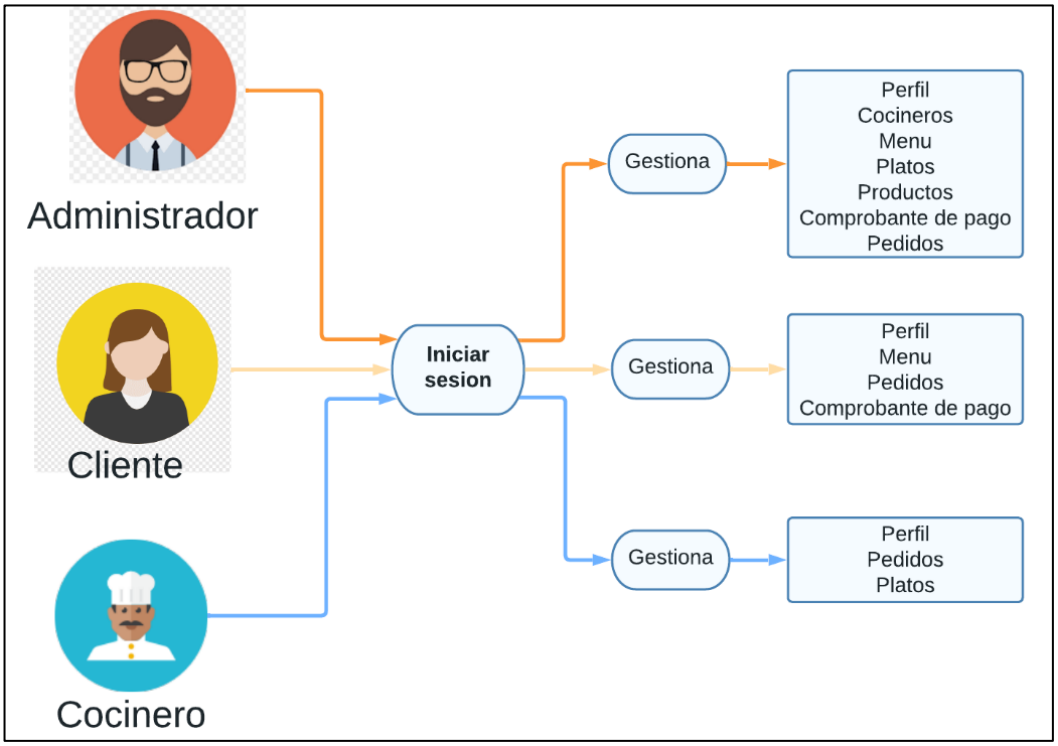


Fig. 7 Roles de usuario

3.2 *Sprint* 1. Implementación de los métodos para registro, actualización y autenticación de usuarios.

Tomando en cuenta, las tareas asignadas en el *Sprint backlog* se muestran los siguientes puntos:

- Generar *endpoint* para la información de la cafetería.
- Generar *endpoints* para registro de usuarios
- Generar *endpoints* para iniciar sesión y recuperar contraseña
- Generar *endpoints* para actualizar perfil de usuarios.
- Generar *endpoints* para crear, eliminar, obtener cocinero y obtener colaboradores (administrador y cocineros).

Generar *endpoint* para la información de la cafetería.

La información de la cafetería se encuentra en una colección NoSQL, los datos más importantes como el menú, los servicios que se ofrece en la cafetería, el menú del día, de la semana, promociones e información que sea de importancia para la comunidad. Adicionalmente, en el *backend* se ha asignado una ruta publica de tipo *GET*, el cual permite obtener la información que puede ser consumida por el lado del *frontend*.

Generar *endpoints* para registro de usuarios

Para que los usuarios puedan registrarse, en el sistema tienen que ingresar algunos campos, los cuales serán validados. Para el ingreso de estos campos se ha desarrollado varios métodos que permiten que el registro de los tres usuarios se realice de una manera segura, cómo se puede visualizar en la **Fig. 8**.

```
export class Registrar{
  private contexto:Contexto;
  private servicioArchivo:IServicioArchivo;

  constructor(dependencias:{contexto:Contexto, servicioArchivo: IServicioArchivo}){
    this.contexto = dependencias.contexto;
    this.servicioArchivo = dependencias.servicioArchivo;
  }

  async ejecutar(usuario:Comando): Promise<DTO>{
    await this.validar(usuario);

    const nuevoUsuario = new Usuario(usuario.nombre!, usuario.apellido!, usuario.telefono!, usuario.email!, usua
    nuevoUsuario.password = await this.encriptarPassword(nuevoUsuario.password);
    if(usuario.fotoBase64){
      nuevoUsuario.urlFoto = await this.servicioArchivo.subir(usuario.fotoBase64);
    }
    const usuarioCreado = await this.contexto.Usuario.create(nuevoUsuario);

    return this.getDTO(usuarioCreado);
  }

  public async encriptarPassword(password:string):Promise<string>{
    const salt = await genSalt(10);
    return await hash(password,salt);
  }

  private async validar(usuario:Comando){
    this.validarCampos(usuario);
    await this.validarExitenciaEmail(usuario.email!)
    if(usuario.rol == 'administrador') await this.validarQueNoExistaUnAdmin();
  }

  private validarCampos(usuario:Comando){
    const resultado = new ValidadorRegistro().validate(usuario)
    if(resultado.isInvalid()){
      throw new ErrorServer('Campos incorrectos!',400,resultado.getFailureMessages())
    }
  }
```

Fig. 8 Método registro de usuarios

Generar *endpoints* para iniciar sesión y recuperar contraseña

Para la interacción de los usuarios registrados se tiene un inicio de sesión y cambio de contraseña, para hacer uso de estas funciones se han desarrollado varios métodos que permiten a los distintos usuarios acceder a los módulos. De esta manera en el *backend* se ha desarrollado un método para poder ingresar las credenciales las cuales consta de correo y contraseña, como se puede visualizar en la **Fig. 9. E**l método, permite recuperar la contraseña el cual solicitara el correo que previamente registramos, como se puede visualizar en la **Fig. 10.**

```
export class Login{
  private contexto:Contexto;

  constructor(dependencias:{contexto:Contexto}){
    this.contexto = dependencias.contexto;
  }

  async ejecutar(request:Comando){
    this.validarCampos(request);

    const usuario = await this.contexto.Usuario.findOne({email: request.email});

    if(!usuario) throw new ErrorServer('Credenciales incorrectas!')

    const resultado = await compare(request.password!, usuario.password!)

    if (!resultado) {
      throw new ErrorServer('Credenciales incorrectas!')}


    return this.getDTO(usuario);
  }

  validarCampos(comando:Comando){
    const respuesta = new ValidadorLogin().validate(comando);
    if(respuesta.isInvalid()){
      throw new ErrorServer('Campos incorrectos!',400,respuesta.getFailureMessages()
    }
  }
```

Fig. 9 Método iniciar sesión

```
export class RecuperarPasswordConCodigo{
  private contexto:Contexto;

  constructor(dependencias: {contexto: Contexto}){
    this.contexto = dependencias.contexto;
  }

  async ejecutar(request:IRequest){
    if(!request.email){
      throw new ErrorServer('Debe enviar el email!');
    }
    if(!request.codigo){
      throw new ErrorServer('Debe enviar el codigo!');
    }
    if(!request.password){
      throw new ErrorServer('Debe enviar el password!');
    }

    if(request.password.length < 8){
      throw new ErrorServer('El password debe tener al menos 8 caracteres!');
    }

    const usuario = await this.contexto.Usuario.findOne({email: request.email});
    if(!usuario){
      throw new ErrorServer('El email no existe!');
    }

    if(usuario.codigo ≠ request.codigo){
      throw new ErrorServer('El código es incorrecto!');
    }

    usuario.codigo = this.generarCodigo();
    usuario.password = await this.encriptarPassword(request.password.toString());
    await usuario.save();
    return usuario;
  }

  private generarCodigo():string{
    let codigo:string = "";
    for (let i = 0; i < 6; i++) {
      codigo += Math.floor(Math.random()*9+1)
```

Fig. 10 Método recuperar contraseña

Generar *endpoints* para actualizar perfil de usuario.

Para el manejo de la información de los usuarios registrados se han desarrollado algunos métodos los cuales permiten al usuario actualizar sus datos personales. De esta manera en el *backend* se ha desarrollado un método para poder actualizar el perfil del usuario, como se puede visualizar en la **Fig. 11.**

```
export interface Comando{
  id?: string,
  telefono?:string,
  password?:string,
  fotoBase64?:string | null
}

export class ActualizarUsuario{
  private contexto:Contexto;
  private servicioArchivo:IServicioArchivo;

  constructor(dependencias:{contexto:Contexto, servicioArchivo: IServicioArchivo}){
    this.contexto = dependencias.contexto;
    this.servicioArchivo = dependencias.servicioArchivo;
  }

  async ejecutar(request:Comando): Promise<any>{
    const usuario = await this.contexto.Usuario.findById(request.id);
    if(!usuario){
      throw new ErrorServer('El id no existe!');
    }
    if(request.fotoBase64){
      usuario.urlFoto = await this.servicioArchivo.subir(request.fotoBase64) as any;
    }
    if(request.password){
      usuario.password = await this.encriptarPassword(request.password);
    }
    if(request.telefono){
      usuario.telefono = request.telefono;
    }
    usuario.save();

    return usuario;
  }

  public async encriptarPassword(password:string):Promise<string>{
    const salt = await genSalt(10);
    return await hash(password,salt);
  }
}
```

Fig. 11 Método actualizar usuario

Generar *endpoints* para crear, eliminar, obtener cocinero y obtener colaboradores (administradores y cocineros).

El perfil administrador tiente la función disponible de poder administrar a los colaboradores. Donde eliminara, creará y obtendrá su información. Por lo tanto, se ha desarrollado algunos métodos, el de agregar colaborador donde este será el cocinero, el cual tendrá su respectiva información de registro, acompañado de sus respectivas validaciones, como se puede visualizar en la **Fig. 12.** El método que se ha utilizado para obtener los colaboradores que se tiene, se puede apreciar en la **Fig. 13.** El método que se ha utilizado para obtener los cocineros que se tiene, se puede apreciar en la **Fig. 14**. Tenemos un método que nos permite eliminar colaborades, como se puede visualizar en la **Fig. 15.**

```
export class CrearColaborador{
  private contexto;
  private servicioArchivo:IServicioArchivo;
  private servicioEmail: IServicioEmail;

  constructor(dependencias:{contexto:Contexto, servicioArchivo: IServicioArchivo, servi
    this.contexto = dependencias.contexto;
    this.servicioArchivo = dependencias.servicioArchivo;
    this.servicioEmail = dependencias.servicioEmail;
  }

  async ejecutar(request:Comando): Promise<DTO>{
    await this.validar(request);
    const codigo = this.generarCodigo();
    const nuevoUsuario = new Usuario(request.nombre!, request.apellido!, request.telefo
    nuevoUsuario.password = await this.encriptarPassword(nuevoUsuario.password);
    if(request.fotoBase64){
      nuevoUsuario.urlFoto = await this.servicioArchivo.subir(request.fotoBase64);
    }

    const usuarioCreado = await this.contexto.Usuario.create(nuevoUsuario);

    await this.servicioEmail.enviarEmail(
      nuevoUsuario.email!,
      `<h2>Credenciales</h2>
      <br>
      <p>Email: <strong>${nuevoUsuario.email}</strong></p>
      <p>Contraseña: <strong>${codigo}</strong></p>`,
      'Credenciales del comedor EPN')

    return this.getDTO(usuarioCreado);
  }

  public async encriptarPassword(password:string):Promise<string>{
    const salt = await genSalt(10);
    return await hash(password,salt);
  }
```

Fig. 12 Método crear colaborador

```
import {Contexto} from '../../persistencia/contexto'

export class ObtenerColaboradores{
  private contexto:Contexto;

  constructor(dependencias:{contexto:Contexto}){
    this.contexto = dependencias.contexto;
  }

  public async ejecutar():Promise<any[]>{
    return await this.contexto.Usuario.find({
      rol:{
        '$in':['administrador','cocinero']
      }
    })
  }
}
```

Fig. 13 Método obtener colaboradores (administrador y cocineros)

```
import {Contexto} from '../../persistencia/contexto'

export class ObtenerCocineros{
  private contexto:Contexto;

  constructor(dependencias:{contexto:Contexto}){
    this.contexto = dependencias.contexto;
  }

  public async ejecutar():Promise<any[]>{
    const cocineros =  await this.contexto.Usuario.find({
      rol:{
        '$in':['cocinero']
      }
    });
    let data:any[] = [];

    for (let i = 0; i < cocineros.length; i++) {
      let pedidosPendientes = 0;
      let pedidos = await this.contexto.Pedido.find({_cocinero:cocineros[i]._id});
      pedidos.forEach(x => {
        if(x.estado[x.estado.length-1] == 'verificado'){
          pedidosPendientes++;
        }
      })
      const {_id, nombre, apellido} = cocineros[i];
      data.push({
        pedidosPendientes,
        _id,
        nombre,
        apellido
      })
    }

    return data;
  }
}
```

Fig. 14 Método obtener cocinero

```
import { Contexto } from '../../persistencia/contexto';
import {ErrorServer} from '../../comunes/expeciones/error';

interface IRequest{
  id: string
}

export class EliminarColaborador{
  private contexto:Contexto;

  constructor(dependencias:{contexto:Contexto}){
    this.contexto = dependencias.contexto;
  }

  async ejecutar(request: IRequest){
    const usuario = await this.contexto.Usuario.findByIdAndDelete(request.id);
    if(!usuario){
      throw new ErrorServer('El id no existe!');
    }
    return usuario;
  }
}
```

Fig. 15 Método eliminar colaboradores

3.3 *Sprint* 2. Implementación de los métodos para manejo productos, platos, menú y pedidos.

Tomando en cuenta, las tareas asignadas en el Sprint backlog tenemos lo siguientes puntos:

- Generar *endpoints* para el manejo de productos.
- Generar *endpoints* para el manejo de platos.

- Generar *endpoints* para el manejo de menú.
- Generar *endpoints* para el manejo de pedidos.

Generar *endpoints* para el manejo de los productos

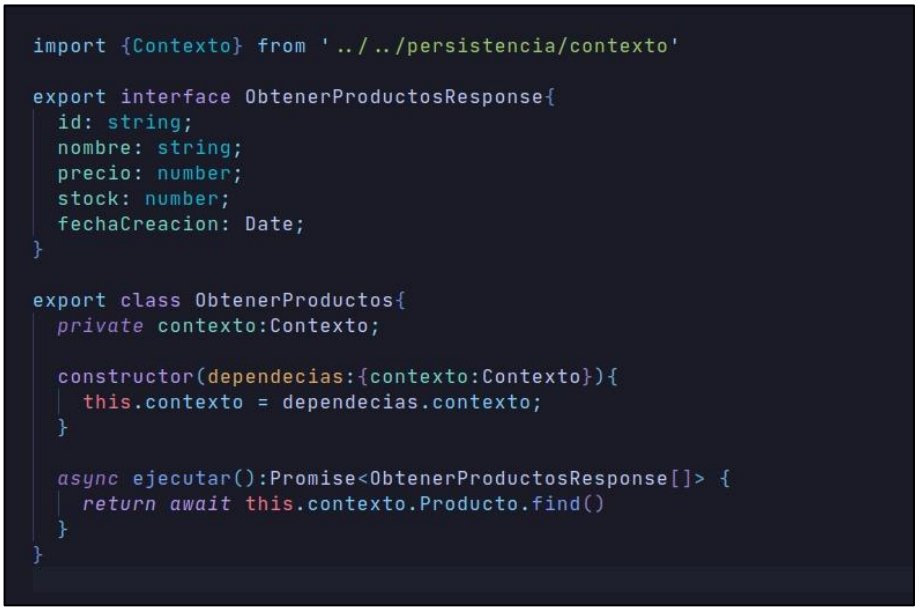
El manejo de los productos el perfil administrador puede, agregar, eliminar, actualizar y obtener la información de los productos, por lo tanto, en el *backend* se ha desarrollado una serie de métodos para poder ingresar la información, los datos ingresados cuentan con su respectiva validación. La interacción con los productos se tiene 4 métodos:

El método que se ha implementado para agregar los productos se muestra en la **Fig. 16.**

```
export class AgregarProducto{
  private contexto:Contexto;

  constructor(dependecias:{contexto:Contexto}){
    this.contexto = dependecias.contexto;
  }

  async ejecutar(request:AgregarProductoRequest):Promise<AgregarProductoResponse>{
    this.validarCampos(request);

    const producto = await this.contexto.Producto.create(new Producto(request.nombre, request.precio

    return{
      id: producto.id,
      nombre: producto.nombre!,
      precio: producto.precio!,
      stock: producto.stock!,
      fechaCreacion: producto.fechaCreacion!
    }
  }

  private validarCampos(request:AgregarProductoRequest){
    const resultado = new ValidadorAgregarProducto().validate(request)
    if(resultado.isInvalid()){
      throw new ErrorServer('Campos incorrectos!',400,resultado.getFailureMessages())
    }
  }
}
```

Fig. 16 Método agregar producto

El método que se ha implementado para obtener los productos se muestra en la **Fig. 17.**



```
import {Contexto} from '../../persistencia/contexto'

export interface ObtenerProductosResponse{
  id: string;
  nombre: string;
  precio: number;
  stock: number;
  fechaCreacion: Date;
}

export class ObtenerProductos{
  private contexto:Contexto;

  constructor(dependecias:{contexto:Contexto}){
    this.contexto = dependecias.contexto;
  }

  async ejecutar():Promise<ObtenerProductosResponse[]> {
    return await this.contexto.Producto.find()
  }
}
```

Fig. 17 Método obtener productos

El método que se ha implementado para actualizar los productos se muestra en la **Fig. 18.**

```
import {Contexto} from '../../persistencia/contexto'

export class ActualizarProducto{
  private contexto:Contexto;

  constructor(dependecias:{contexto:Contexto}){
    this.contexto = dependecias.contexto;
  }

  async ejecutar(request:any){

    const producto = await this.contexto.Producto.findByIdAndUpdate(request.id,{
      nombre: request.nombre,
      precio: request.precio,
      stock: request.stock
    });

    return producto;
  }
}
```

Fig. 18 Método actualizar productos

El método que se ha implementado para eliminar los productos se muestra en la **Fig. 19.**

```
import { ErrorServer } from '../../comunes/expeciones/error';
import {Contexto} from '../../persistencia/contexto'

export interface EliminarProductoResponse{
  id: string;
  nombre: string;
  precio: number;
  stock: number;
  fechaCreacion: Date;
}

export class EliminarProducto{
  private contexto:Contexto;

  constructor(dependencias:{contexto:Contexto}){
    this.contexto = dependencias.contexto;
  }

  async ejecutar(id:string):Promise<EliminarProductoResponse>{

    const producto = await this.contexto.Producto.findOneAndDelete({id});

    if(!producto) throw new ErrorServer("El id no existe!");

    return{
      id: producto.id,
      nombre: producto.nombre!,
      precio: producto.precio!,
      stock: producto.stock!,
      fechaCreacion: producto.fechaCreacion!
    }
  }
}
```

Fig. 19 Método eliminar producto

Generar *endpoint*s para el manejo de los platos

El manejo de los platos el perfil administrador puede agregar, eliminar, actualizar y obtener la información de los platos que componen el menú, por lo tanto, en el *backend* se ha desarrollado una serie de métodos que nos permite interactuar con los platos para poder ingresar la información, los valores ingresados contaran con su respectiva validación.

El método que se ha implementado para agregar los platos se muestra en **Fig. 20.**

```
export class AgregarComponente{
  private contexto:Contexto;
  private servicioArchivo:IServicioArchivo;

  constructor(dependencias:{contexto:Contexto, servicioArchivo: IServicioArchivo}){
    this.contexto = dependencias.contexto;
    this.servicioArchivo = dependencias.servicioArchivo;
  }

  async ejecutar(request:AgregarComponenteRequest):Promise<AgregarComponenteResponse>{
    this.validarCampos(request);
    const {nombre, descripcion, precio, base64Foto, medida, tipo} = request;
    const componente = new Componente(nombre, descripcion,precio,medida,tipo);

    if(base64Foto){
      componente.urlFoto = await this.servicioArchivo.subir(base64Foto);
    }

    const componenteNuevo = await this.contexto.Componente.create(componente);

    return{
      id: componenteNuevo.id,
      nombre: componenteNuevo.nombre!,
      descripcion: componenteNuevo.descripcion!,
      precio: componenteNuevo.precio!,
      medida: componenteNuevo.medida!,
      tipo: componenteNuevo.tipo!,
      urlFoto: componenteNuevo.urlFoto!,
      fechaCreacion: componenteNuevo.fechaCreacion!
    }
  }

  private validarCampos(request:AgregarComponenteRequest){
    const resultado = new ValidadorAgregarComponente().validate(request)
    if(resultado.isInvalid()){
      throw new ErrorServer('Campos incorrectos!',400,resultado.getFailureMessages())
    }
  }
}
```

Fig. 20 Método agregar plato

El método que se ha implementado para obtener los platos se muestra en la **Fig. 21.**

```
import {Contexto} from '../../persistencia/contexto';

export interface ObtenerComponentesResponse{
  id: string;
  nombre: string;
  descripcion:string;
  precio: number;
  medida: string;
  tipo: string;
  urlFoto: string | null;
  fechaCreacion: Date;
}

export class ObtenerComponentes{
  private contexto:Contexto;

  constructor(dependencias:{contexto:Contexto}){
    this.contexto = dependencias.contexto;
  }

  async ejecutar():Promise<ObtenerComponentesResponse[]>{
    return await this.contexto.Componente.find();
  }
}
```

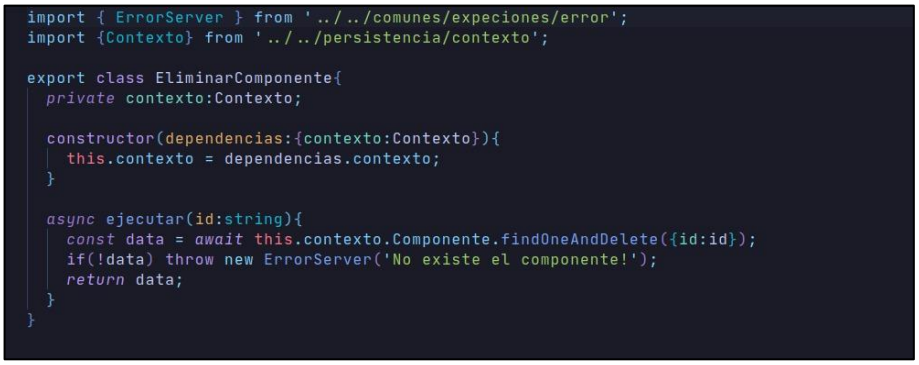
Fig. 21 Método obtener plato

El método que se ha implementado para actualizar los productos se muestra en la **Fig. 22.**

```
import {Contexto} from '../../persistencia/contexto';

export interface ActualizarComponenteRequest{
  id:string,
  nombre: string,
  descripcion: string,
  precio: number,
  medida: string,
  tipo: string,
}

export class ActualizarComponente{
  private contexto:Contexto;

  constructor(dependencias:{contexto:Contexto}){
    this.contexto = dependencias.contexto;
  }

  async ejecutar(request:ActualizarComponenteRequest){
    const {nombre, descripcion, precio, medida, tipo} = request;

    const componente = await this.contexto.Componente.findOneAndUpdate({id:request.id},{nc

    return componente;
  }
}
```

Fig. 22 Método actualizar plato

El método que se ha implementado para eliminar los productos se muestra en la **Fig. 23.**



```
import { ErrorServer } from '../../comunes/expeciones/error';
import {Contexto} from '../../persistencia/contexto';

export class EliminarComponente{
  private contexto:Contexto;

  constructor(dependencias:{contexto:Contexto}){
    this.contexto = dependencias.contexto;
  }

  async ejecutar(id:string){
    const data = await this.contexto.Componente.findOneAndDelete({id:id});
    if(!data) throw new ErrorServer('No existe el componente!');
    return data;
  }
}
```

Fig. 23 Método eliminar plato

Generar *endpoint* para el manejo del menú.

El manejo de los menús cuenta con algunos métodos los cuales nos permite crear, eliminar, actualizar el estado y obtener la información del menú. Donde, el usuario administrador puede integrar los platos que se agregó previamente y armar un menú, en el *backend* para la creación del menú tiene que ingresar algunos campos, con su respectiva validación, como se puede visualizar en la **Fig. 24.** También, se tiene un método el cual permite obtener el menú, como se muestra en la **Fig. 25.** También, se tiene un método el cual permite actualizar el estado del menú, como se muestra en la **Fig. 26.** También, se tiene un método el cual permite eliminar el menú, como se muestra en la **Fig. 27.**

```
import {Contexto} from '../../persistencia/contexto';
import {Menu} from '../../dominio/entidades/menu';

export interface Comando{
  titulo: string,
  fecha: string,
  componentes: any[]
}

export  class AgregarMenu{
  private contexto:Contexto;

  constructor(dependencias:{contexto:Contexto}){
    this.contexto = dependencias.contexto;
  }

  async ejecutar(request:Comando){
    const menu = this.contexto.Menu.create(new Menu(request.titulo, request.fecha, request
    return menu;
  }
}
```

Fig. 24 Método agregar menú

```
import {Contexto} from '../../persistencia/contexto';

export class ObtenerMenus{
  private contexto:Contexto;

  constructor(dependencias:{contexto:Contexto}){
    this.contexto = dependencias.contexto;
  }

  async ejecutar():Promise<any[]>{
    return await this.contexto.Menu.find().populate('componentes._componente');
  }
}
```

Fig. 25 Método obtener menú

```
import {Contexto} from '../../persistencia/contexto';

interface Comando{
  id: string,
  _componente: string
}

export class ActualizarEstadoMenu{
  private contexto:Contexto;

  constructor(dependencias:{contexto:Contexto}){
    this.contexto = dependencias.contexto;
  }

  async ejecutar(request:Comando){
    const menu = await this.contexto.Menu.findById(request.id)
    menu?.componentes.map((x:any)=>{
      if(x._componente._id == request._componente){
        x.estado = !x.estado;
      }
      return x;
    })
    await menu?.save();
    return menu;
  }
}
```

Fig. 26 Método actualizar estado de menú

```
import {Contexto} from '../../persistencia/contexto';
import {ErrorServer} from '../../comunes/expeciones/error';

export class EliminarMenu{
  private contexto:Contexto;

  constructor(dependencias:{contexto:Contexto}){
    this.contexto = dependencias.contexto;
  }

  async ejecutar(id:string){
    const menu = await this.contexto.Menu.findOneAndDelete({id});
    if(!menu) throw new ErrorServer('El id no existe!');
    return menu;
  }
}
```

Fig. 27 Método eliminar menú

Generar *endpoints* para el manejo de pedidos.

El manejo de los pedidos tiene algunos métodos los cuales permite agregar, actualizar el estado y obtener la información del pedido, donde, los usuarios pueden ver los pedidos realizados. En el *backend* para agregar un pedido se tiene un método para agregar pedido, en este se tiene que agregar un comprobante para poder validar el pago del pedido, como se puede visualizar en la **Fig. 28.** También, tiene un método el cual permite actualizar el pedido, como se muestra en la **Fig. 29.** También, tiene un método el cual permite listar los pedidos realizados, como se muestra en la **Fig. 30.**

```
import { Pedido } from "../../dominio/entidades/pedido";
import { Contexto } from '../../persistencia/contexto';
import {ErrorServer} from '../../comunes/expeciones/error';
import { IServicioArchivo } from '../../comunes/interfaces/IServicioArchivo';

export class AgregarPedido{
  private contexto:Contexto;
  private servicioArchivo:IServicioArchivo;

  constructor(dependencias:{contexto:Contexto, servicioArchivo: IServicioArchivo}){
    this.contexto = dependencias.contexto;
    this.servicioArchivo = dependencias.servicioArchivo;
  }

  async ejecutar(request:any){
    const urlComprobante = await this.servicioArchivo.subir(request.base64Comprobante);
    if(!urlComprobante){
      throw new ErrorServer('Debe enviar la foto del comprobante')
    }

    const pedido = new Pedido(request.total, urlComprobante, request._cliente, request.co

    pedido.productos.forEach(async(x:any) => {
      const producto = await this.contexto.Producto.findById(x._producto);
      producto!.stock! -= x.cantidad;
      await producto!.save();
    });

    const nuevoPedido = await this.contexto.Pedido.create(pedido);
    return nuevoPedido;
  }
}
```

Fig. 28 Método agregar pedido

```
import { Contexto } from '../../persistencia/contexto';

export class ActualizarEstadoPedido{
  private contexto:Contexto;

  constructor(dependencias:{contexto:Contexto}){
    this.contexto = dependencias.contexto;
  }

  async ejecutar(request:any){
    const pedido = await this.contexto.Pedido.findById(request.id);
    pedido?.estado.push(request.estado);
    if(request.estado == 'verificado'){
      pedido!._cocinero = request.cocinero;
    }
    pedido?.save();
    return pedido;
  }
}
```

Fig. 29 Método actualizar estado de pedido

```
import { Contexto } from '../../persistencia/contexto';

export class ObtenerPedidos{
  private contexto:Contexto;

  constructor(dependencias:{contexto:Contexto}){
    this.contexto = dependencias.contexto;
  }

  async ejecutar(request:any){
    if(request.rol == 'cliente'){
      return await this.contexto.Pedido.find({_cliente:request.id})
        .populate('_cliente')
        .populate('productos._producto')
        .populate('componentes._componente')
    }
    if(request.rol == 'cocinero'){
      return await this.contexto.Pedido.find({_cocinero:request.id})
        .populate('_cliente')
        .populate('productos._producto')
        .populate('componentes._componente')
    }
    return await this.contexto.Pedido.find()
      .populate('_cliente')
      .populate('productos._producto')
      .populate('componentes._componente')
  }
}
```

Fig. 30 Método obtener pedidos

3.4 *Sprint* 3. Pruebas

Basándose en lo mencionado en el *Sprint Backlog* y luego de haber terminado la codificación de cada uno de los *endpoints*, se tiene presente los siguientes puntos a realizar:

- Ejecución de pruebas unitarias y resultados.
- Ejecución de pruebas de rendimiento y resultados.
- Ejecución de pruebas de compatibilidad y resultados.

Ejecución de pruebas unitarias y resultados

Al terminar la codificación del *backend* y siguiendo lo planificado en el *Sprint Backlog,* en esta parte se desarrolla las pruebas unitarias al componente del *backend*. Las pruebas unitarias son pruebas que se realizan sobre las unidades individuales más pequeñas de los programas informáticos. Su objetivo principal es comprobar el correcto funcionamiento de estas pequeñas piezas en entornos aislados [35].En ese sentido, para comenzar con el *testing* del *backend* donde usaremos Mocha que nos proporciona una forma de escribir un código estructurado para probar las aplicaciones a fondo, clasificándolas en conjuntos de pruebas y casos de prueba [36].

La **Fig. 31** se visualiza una parte del código que se ha ejecutado para el inicio de sesión de los usuarios, en la siguiente figura **Fig. 32** se visualiza el resultado generado después de haber realizado las prueba. En tanto que, el detalle completo de la ejecución y resultados de las demás pruebas, se encuentran en el **ANEXO II** del presente documento.

```
it('Logeo exitoso | retorna usuario', async () => {
  const comando: Comando = { email: 'luisjoel@gmail.com', password: '12345678' }
  const usuario = {
    nombre: 'Luis TEST',
    apellido: 'Méndez Loor',
    telefono: '0983336558',
    email: 'luisjoel@gmail.com',
    password: '12345678',
    rol: 'cliente'
  }
  usuario.password = await registrar.encriptarPassword(usuario.password);
  await contexto.Usuario.deleteMany({});
  await contexto.Usuario.create(usuario);
  const sut = await login.ejecutar(comando);
  expect(sut).to.all.keys(['id', 'nombre', 'apellido', 'telefono', 'email', 'rol',
})
```

Fig. 31 Código prueba para el inicio de sesión

```
Login usuario
  ✔Logeo exitoso | retorna usuario (1039ms)
  ✔Logearse con email inexistente | retorna error (307ms)
  ✔Logearse con password incorrecto | retorna error (595ms)
  ✔Login con campos vacíos | retorna error

Obtener Colaboradores
  ✔Obtener Colaboradores debería retornar un array de usuarios (2226ms)
  ✔Obtener Colaboradores debería retornar un array de usuarios con rol administrador y conicero (1403ms)
```

Fig. 32 Resultado de la prueba

Después de la ejecución de la prueba y en base a los resultados generados, se establece que uno de los módulos del *backend* no expone ningún fallo en su funcionalidad o validación.

Ejecución de pruebas rendimiento y resultados

Las pruebas de rendimiento se refieren a la evaluación del rendimiento del sistema en diversos escenarios de prueba para evaluar la funcionalidad de los componentes de la aplicación. Este tipo de pruebas es esencial para garantizar la calidad y funcionalidad de un sistema determinado. [37], por consiguiente, en este proyecto las pruebas han sido

desarrolladas por la aplicación *web pagespeed,* la cual muestra datos que califican el rendimiento de nuestro *backend*. En ese sentido, la **Fig. 33 ,** se obtiene los resultados que se han generado después de aplicar la prueba.

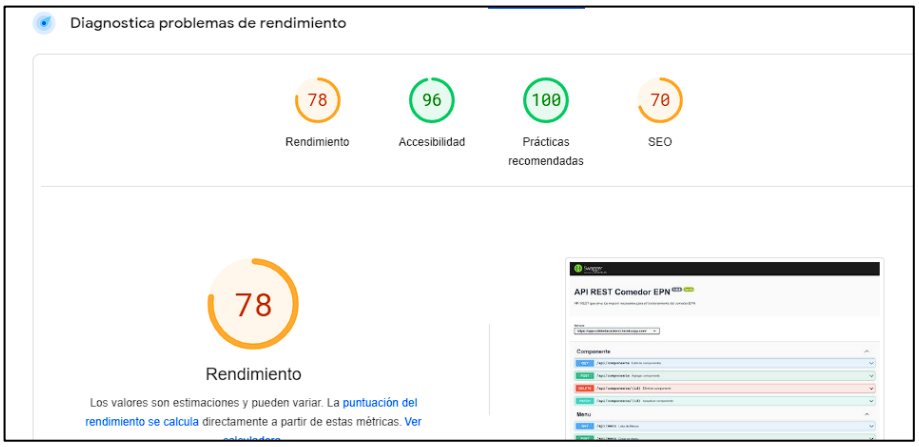


Fig. 33 Prueba de rendimiento en *PageSpeed*.

Luego de visualizar los datos de esta prueba, se establece que el *backend* no presenta problemas que pongan en riesgo el funcionamiento y el consumo por parte del *frontend*.

Ejecución de pruebas compatibilidad y resultados

Las pruebas de compatibilidad aseguran la ejecución del software en una configuración diferente, distintas aplicaciones y sus versiones [38]. Para la ejecución del componente desarrollado en este caso el *backend*, las pruebas han sido realizadas por clientes HTTP, los cuales nos permite interactuar con los *endpoints*, En este caso usaremos: *Postman* y *Thunder Client,* estas dos herramientas nos permite verificar de forma correcta la información de cada uno de los *endpoints*. De tal manera podemos visualizar en la siguiente **TABLA VII ,** las herramientas HTTP que sean realizado las pruebas, mientras que la ejecución y los resultados se puede apreciar en el **ANEXO II** del presente documento.

TABLA VII: Herramientas HTTP para la prueba de compatibilidad

NOMBRE	VERSION
Postman	V10.9.0
Thunder Client	V2.3.3.

Al finalizar las pruebas y en base a los resultados que se han generado, se llega a la conclusión que el *backend* no tiene ningún fallo para obtener las respuestas de las *endpoints*.

3.5 *Sprint* 4. Despliegue del *backend*

Luego de haber terminado la codificación y tomando en cuentas los resultados obtenidos en las pruebas, en este apartado se desarrollará:

- Despliegue en MongoDB Atlas de la Base de datos.
- Despliegue del *backend* en *Heroku*.

Despliegue MongoDB Atlas de la Base de datos.

Se realiza el despliegue de la base de datos NoSQL en la plataforma de MongoDB Atlas, como se muestra en la **Fig. 34,** cade recalcar que en el **ANEXO IV** se encuentra detallado el procedimiento del despliegue.

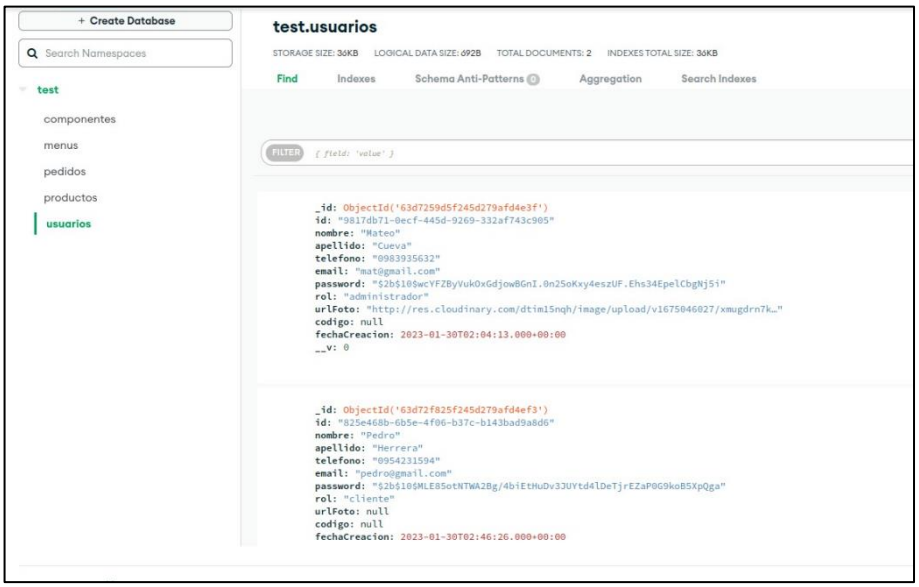


Fig. 34 Base de datos MongoDB Atlas

Despliegue del *backend* en *Heroku*.

Se realiza el despliegue del *backend* en la plataforma de *Heroku*, como se muestra en la **Fig. 35,** cade recalcar que en el **ANEXO IV** se encuentra detallado el procedimiento del despliegue. Para finalizar, para acceder al *backend*, se debe ingresar a la siguiente url:

https://app-cafeteriacackend.herokuapp.com/

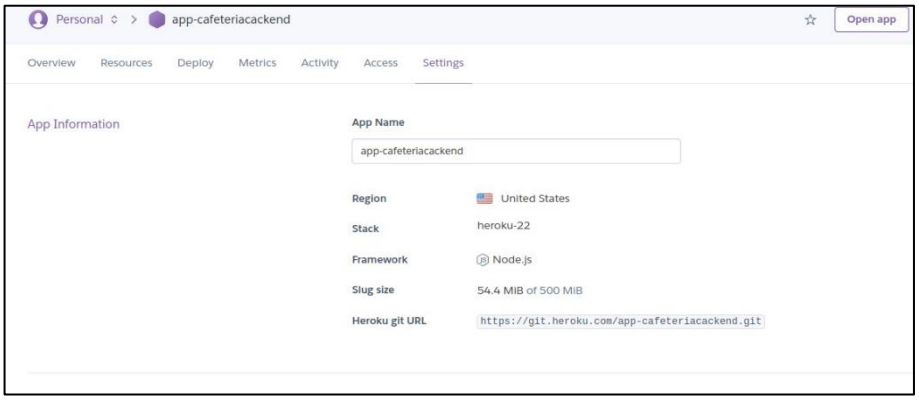


Fig. 35 *Backend* en *Heroku*

4 CONCLUSIONES

Las conclusiones que se han obtenido en el desarrollo de este trabajo de integración curricular.

- La obtención de los requisitos al inicio del proyecto permitió que el desarrollo del *backend* y las API *RESTful* se realizara de manera estructurada y escalable, en caso de que se necesite agregar más perfiles y módulos en el futuro.
- La implementación de la metodología ágil *Scrum* ha permitido el avance y corrección del proyecto en cada *Sprint*, lo que ha resultado en la obtención de un producto de alta calidad dentro de los plazos establecidos al inicio del proyecto.
- El uso de *NoSQL* y *MongoDB* en el desarrollo de aplicaciones ofrece una serie de beneficios. Donde, las bases de datos *NoSQL* permiten una mayor escalabilidad y flexibilidad en el almacenamiento de datos, lo que es especialmente útil en aplicaciones que manejan grandes volúmenes de información o que requieren una rápida adaptación a cambios en los requisitos de almacenamiento.
- El uso de *NoSQL* y *MongoDB* también puede ofrecer beneficios en términos de rendimiento, ya que estas bases de datos no tienen las mismas restricciones de coherencia que las bases de datos relacionales, lo que les permite manejar cargas de trabajo más intensivas.
- El *backend* está actualmente desplegado y listo para ser utilizado por cualquier aplicación cliente que desee consumir las rutas generadas por el *backend*.
- La ejecución de pruebas es una buena forma de asegurar que todas las funcionalidades del código funcionen de manera precisa, al probar cada porción de código en distintas situaciones.

5 RECOMENDACIONES

Aquí tenemos las recomendaciones que se han obtenido en el desarrollo de este trabajo de integración curricular.

- Es fundamental asegurarse de utilizar versiones estables o con soporte de Node.js, ya que esto puede tener un impacto significativo en el momento del despliegue en cualquier plataforma de la nube.

- Se sugiere que el administrador defina nuevas políticas de privacidad de información para cualquier nuevo *endpoint* que sea desarrollado.

- Se recomienda emplear *TypeScript* como lenguaje de programación para aprovechar su capacidad de verificación de tipos estática y mejorar la calidad del código y la detección de errores en tiempo de compilación.

- Implementa técnicas de seguridad, como la encriptación de contraseñas y el uso de tokens JWT para la autenticación, con el fin de proteger la información confidencial y evitar vulnerabilidades de seguridad.

- Asegúrate de utilizar un sistema de control de versiones, como Git, para controlar y gestionar el código fuente de tu proyecto. Esto te permitirá tener un historial de cambios y colaborar con otros desarrolladores de manera eficiente.

- Se recomienda corregir la documentación de *Swagger* validando los datos y mejorando la interacción con los *endpoints*, para mejorar la interacción con usuarios que deseen usar el *backend*.

6 REFERENCIAS BIBLIOGRAFICAS

[1] Revfine.com, «Revfine,» 21 Febrero 2022. [En línea]. Available: https://www.revfine.com/es/restaurante-tecnologia/. [Último acceso: 19 Noviembre 2022].

[2] I. d. Souza, «Rockcontent,» 17 Marzo 2020. [En línea]. Available: https://rockcontent.com/es/blog/api-rest/. [Último acceso: 27 11 2022].

[3] I. Gomez, «crehanarehana,» 17 Marzo 2022. [En línea]. Available: https://www.crehana.com/blog/transformacion-digital/que-es-api-rest/. [Último acceso: 2022 Noviembre 29].

[4] F. Machuca, «Crehana,» 18 mayo 2022. [En línea]. Available: https://www.crehana.com/blog/transformacion-digital/que-es-el-backend-y-como-usarlo/. [Último acceso: 2022 Diciembre 08].

[5] Inesdi, «Inesdi,» 27 Enero 2022. [En línea]. Available: https://www.inesdi.com/blog/mongodb-que-es-caracteristicas-para-que-sirve/. [Último acceso: 08 Diciembre 2022].

[6] DataScientest, «DataScientest,» 07 Abril 2022. [En línea]. Available: https://datascientest.com/es/mongodb-todo-sobre-la-base-de-datos-nosql-orientada-a-documentos. [Último acceso: 08 Diciembre 2022].

[7] ayudaley, «ayudaley,» [En línea]. Available: https://ayudaleyprotecciondatos.es/bases-de-datos/no-relacional/#Que_es_una_base_de_datos_no_relacional_Definicion. [Último acceso: 08 Diciembre 2022].

[8] AWS, «Amazon,» [En línea]. Available: https://aws.amazon.com/es/nosql/. [Último acceso: 08 Diciembre 2022].

[9] DataScientest, «DataScientest,» 15 Abril 2022. [En línea]. Available: https://datascientest.com/es/api-que-es-y-para-que-sirve. [Último acceso: 08 Diciembre 2022].

[10] AWS, «AWS,» [En línea]. Available: https://aws.amazon.com/es/what-is/api/. [Último acceso: 09 Diciembre 2022].

[11] C. Simões, «ITDO,» 27 Julio 2021. [En línea]. Available: https://www.itdo.com/blog/que-es-node-js-y-para-que-sirve/. [Último acceso: 10 Diciembre 2022].

[12] Kinsta, «Kinsta,» 25 Julio 2022. [En línea]. Available: https://kinsta.com/es/base-de-conocimiento/que-es-express/. [Último acceso: 10 Diciembre 2022].

[13] d. guevara, «medium,» 16 marzo 2018. [En línea]. Available: https://medium.com/@diegoguevaraco/typescriptconnodejs-dbaa83b2c3f3. [Último acceso: 10 Diciembre 2022].

[14] «IMMUNE TECHNOLOGY INSTITUTE,» 04 03 2022. [En línea]. Available: https://immune.institute/blog/typescript-que-es-como-se-diferencia-javascript/. [Último acceso: 10 Diciembre 2022].

[15] S. G. Sotomayor, «IEBS,» [En línea]. Available: https://www.iebschool.com/blog/que-son-metodologias-agiles-agile-scrum/. [Último acceso: 11 Diciembre 2022].

[16] R. APD, «APD,» 01 Enero 2022. [En línea]. Available: https://www.apd.es/metodologia-scrum-que-es. [Último acceso: 11 Diciembre 2022].

[17] A. LABS, «ALAIMO LABS,» [En línea]. Available: https://alaimolabs.com/es/self-learning/scrum/roles-del-equipo-scrum-desarrolladores-scrum-master-y-product-owner. [Último acceso: 11 Diciembre 2022].

[18] M. Á. D. DIOS, «Wearemarketing,» 19 Octubre 2022. [En línea]. Available: https://www.wearemarketing.com/es/blog/roles-de-un-equipo-scrum.html. [Último acceso: 11 Diciembre 2022].

[19] D. Esteban, «Medium,» 21 Junio 2022. [En línea]. Available: https://medium.com/@dcortes.net/fundamentos-de-scrum-pilares-equipo-y-artefactos-c268384ff0d9. [Último acceso: 12 Diciembre 2022].

[20] T. Asana, «asana,» 14 Junio 2022. [En línea]. Available: https://asana.com/es/resources/requirements-gathering. [Último acceso: 12 Diciembre 2022].

[21] T. Asana, «Asana,» 21 Enero 2022. [En línea]. Available: https://asana.com/es/resources/user-stories. [Último acceso: 12 Diciembre 2022].

[22] Donetonic, «Donetonic,» [En línea]. Available: https://donetonic.com/what-is-scrum/#Product_Backlog. [Último acceso: 12 Diciembre 2022].

[23] Donetonic, «Donetonic,» [En línea]. Available: https://donetonic.com/es/product-backlog-y-sprint-backlog/#Definicion_de_Sprint_Backlog. [Último acceso: 12 Diciembre 2022].

[24] j. freeman, «Dedraw,» 07 Enero 2021. [En línea]. Available: https://www.edrawsoft.com/software-architecture.html. [Último acceso: 12 Diciembre 2022].

[25] OkHosting, «OkHosting,» [En línea]. Available: https://okhosting.com/blog/herramientas-de-desarrollo-de-software/. [Último acceso: 22 Diciembre 2022].

[26] Serquo, «Serquo,» 28 Abril 2022. [En línea]. Available: https://serquo.com/blog/node-js/. [Último acceso: 13 Diciembre 2022].

[27] D. P. Acharya, «Kinsta,» 10 Febrero 2022. [En línea]. Available: https://kinsta.com/es/blog/mongodb-vs-mysql/. [Último acceso: 13 Diciembre 2022].

[28] E. C. Navone, «freecodecamp,» 07 Agosto 2022. [En línea]. Available: https://www.freecodecamp.org/espanol/news/aprende-node-js-y-express-curso-desde-cero/. [Último acceso: 13 Diciembre 2022].

[29] Nodemailer, «Nodemailer,» [En línea]. Available: https://nodemailer.com/about/. [Último acceso: 20 Enero 2023].

[30] codigofacilito, «codigofacilito,» [En línea]. Available: https://codigofacilito.com/articulos/que-es-mongoose. [Último acceso: 20 Enero 2023].

[31] D. P. Acharya, «geekflare,» geekflare, 16 Enero 2023. [En línea]. Available: https://geekflare.com/es/image-processing-and-optimization-api/. [Último acceso: 21 Enero 2023].

[32] Kinsta, «Kinsta,» 19 Diciembre 2022. [En línea]. Available: https://kinsta.com/es/base-de-conocimiento/que-es-npm/#:~:text=Aunque%20puedes%20ver%20diferentes%20variaciones,npm%20se%20llaman%20%C2%ABpaquetes%C2%BB.. [Último acceso: 21 Enero 2023].

[33] assemblerinstitute, «assemblerinstitute,» [En línea]. Available: https://assemblerinstitute.com/blog/que-es-postman/. [Último acceso: 21 Enero 2023].

[34] Datademia, «Datademia,» [En línea]. Available: https://datademia.es/blog/que-es-mongodb. [Último acceso: 26 12 2022].

[35] D. Esteban, «Medium,» 12 Septiembre 2022. [En línea]. Available: https://medium.com/@dcortes.net/test-unitarios-en-javascript-con-jest-4f120f5e7124. [Último acceso: 25 Enero 2023].

[36] lambdatest, «lambdatest,» [En línea]. Available: https://www.lambdatest.com/mocha-js. [Último acceso: 26 Enero 2022].

[37] R. KeepCoding, «KeepCoding,» 08 Agosto 2022. [En línea]. Available: https://keepcoding.io/blog/que-son-las-pruebas-de-rendimiento/#:~:text=Las%20pruebas%20de%20rendimiento%20de,los%20componentes%20de%20la%20aplicaci%C3%B3n.. [Último acceso: 27 Enero 2023].

[38] I. Montes, 12 01 2022. [En línea]. Available: https://www.hiberus.com/crecemos-contigo/tipos-de-pruebas-de-software-segun-la-piramide-de-cohn/. [Último acceso: 28 Enero 2023].

7 ANEXOS

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

- ANEXO I**.** Resultado del programa antiplagio *Turnitin*.
- ANEXO II. Manual Técnico.
- ANEXO III**.** Manual de Usuario.
- ANEXO IV. Manual de Instalación

ANEXO I

A continuación, se presenta el certificado que el director de Tesis ha emitido y en donde se evidencia el resultado que se ha obtenido en la herramienta antiplagio *Turnitin.*

ESCUELA POLITÉCNICA NACIONAL
ESCUELA DE FORMACIÓN DE TECNÓLOGOS
CAMPUS POLITÉCNICO "ING. JOSÉ RUBÉN ORELLANA"

CERTIFICADO DE ORIGINALIDAD

Quito, D.M. 1 de marzo de 2023

De mi consideración:

Yo, Franco Rocha Yadira Guissela, en calidad de Director del Trabajo de Integración Curricular titulado DESARROLLO DE UN BACKEND asociado DESARROLLO DE SISTEMA WEB Y APLICACIÓN MÓVIL PARA COMANDAS EN LA CAFETERÍA EPN elaborado por el estudiante **Cuenca Chamba Miguel Eduardo** de la carrera en Tecnología Superior en Desarrollo de Software, certifico que he empleado la herramienta Turnitin para la revisión de originalidad del documento escrito en las secciones: Descripción del componente desarrollado, Metodología, Resultados, Conclusiones y Recomendaciones, producto del Trabajo de Integración Curricular indicado.

El documento escrito tiene un índice de similitud del 11%.

Es todo cuanto puedo certificar en honor a la verdad, pudiendo el interesado hacer uso del presente documento para los trámites de titulación.

NOTA: Se adjunta el informe generado por la herramienta Turnitin.

Atentamente,

YADIRA GUISSELA FRANCO ROCHA

Franco Rocha Yadira Guissela
Profesor Ocasional a Tiempo Completo
Escuela de Formación de Tecnólogos

ANEXO II

Recopilación de Requerimientos

En la **TABLA VIII** se muestra los requerimientos que se han obtenido al inicio del proyecto de acuerdo con lo solicitado.

TABLA VIII: Recopilación de requerimientos

RECOPILACIÓN DE REQUERIMIENTOS		
TIPO DEL SISTEMA	**ID - RR**	**ENUNCIADO DEL ÍTEM**
BACKEND	**RR002**	Como usuarios administradores, cliente y cocinero necesitan desarrollar varios *endpoints* para: Iniciar sesión. Cerrar sesión. Recuperar contraseña.
	RR003	Como usuario administrador y clientes necesita generar varios *endpoints* para: Registrar los datos del usuario administrador y cliente Registrar usuario cocinero
	RR004	Como usuario administrador necesita generar a varios *endpoints* para: Visualizar los pedios y revisar el estado de pago del pedido
	RR005	Como usuario administrador necesita generar varios *endpoints* para: Asignar el pedido a un cocinero
	RR006	Como usuario administrador necesita generar varios *endpoints para:* Manejo de platos y productos
	RR007	Como usuario administrador necesita generar varios *endpoints para:* Manejo del menú
	RR008	Como usuario cliente y cocinero necesita generar varios endpoints para: Modificar su perfil

	RR009	Como usuario cliente necesita generar varios *endpoints* para: Manejo de pedidos.
	RR010	Como usuario cliente necesita generar varios *endpoints* para: Confirmar su pedido Subir comprobante de pago.
	RR011	Como cliente necesita generar varios *endpoints* para: Visualizar el estado del pedido.
	RR012	Como usuario cocinero necesita generar varios *endpoints* para: Visualizar los pedidos asignados.
	RR013	Como usuario cocinero necesita generar varios *endpoints* para: Cambiar el estado de los pedidos.

Historias de Usuario

Culminada la etapa de recopilación de requerimientos, se procede a desarrollar las Historias de Usuario, para el *backend*. A continuación, se presentan las 13 Historias de Usuario escritas en base a los requerimientos del proyecto que va desde la **TABLA IX** a la TABLA XX **.**

TABLA IX: Historia de usuario 02 – Desarrollar varios *endpoints* para Iniciar sesión, cerrar sesión y recuperar contraseña

HISTORIA DE USUARIO	
Identificador (ID): HU002	**Usuario:** Administrador, cliente y cocinero
Nombre Historia: Crear varios *endpoints* para iniciar sesión, cerrar sesión y recuperar contraseña	
Prioridad en negocio: Alta	**Riesgo en desarrollo**: Alta
Iteración Asignada: 1	
Responsable: Miguel Cuenca	

Descripción: El usuario cocinero, cliente, administrador se tiene que desarrollar varios *endpoints* para poder: • Iniciar sesión • Cerrar sesión • Recuperar contraseña
Observación: Cuando el usuario cocinero, cliente, administrador ingresen usuario y contraseña de manera incorrecta se presenta un mensaje de error, y si desean restablecer su contraseña se lo realiza enviando un código al correo al email registrado.

TABLA X: Historia de usuario 03 – Desarrollar varios *endpoints* para registrarse

HISTORIA DE USUARIO	
Identificador (ID): HU003	**Usuario:** Administrador, cliente, cocinero
Nombre Historia: Desarrollar varios *endpoints* para registro de usuarios.	
Prioridad en negocio: Alta	**Riesgo en desarrollo**: Alta
Iteración Asignada: 1	
Responsable: Miguel Cuenca	
Descripción: El *backend* genera varios *endpoints* que permite a los usuarios poder registrarse. para el registro se realizar por medio de un formulario llenando los siguientes campos: • Nombre • Apellido • Teléfono • Email • Contraseña • Foto • Rol del usuario	
Observación: El cocinero puede acceder a su modulo solamente con los datos promocionados por el administrador. El administrador es el que registra al usuario cocinero.	

TABLA XI: Historia de usuario 04 – Desarrollar varios *endpoints* para para visualizar el pedido y la factura cancelada

HISTORIA DE USUARIO	
Identificador (ID): HU004	**Usuario:** Administrador
Nombre Historia: *Endpoints* para visualizar el pedido y la factura cancelada	
Prioridad en negocio: Alta	**Riesgo en desarrollo**: Alta
Iteración Asignada: 1	
Responsable: Miguel Cuenca	
Descripción: Para el usuario administrador se tiene que desarrollar varios *endpoints* donde el podrá obtener datos sobre el pedido realizado, donde una parte importante es el comprobante de pago, el cual debe ser validado por el administrador.	
Observación: El administrador puede cambiar el estado del pedido dependiendo del comprobante de pago, donde él puede aprobar el pedido o anularlo	

TABLA XII: Historia de usuario 05 – Desarrollar varios *endpoints* para asignar pedidos

HISTORIA DE USUARIO	
Identificador (ID): HU005	**Usuario:** Administrador
Nombre Historia: Desarrollar varios endpoints para asignar pedidos	
Prioridad en negocio: Alta	**Riesgo en desarrollo**: Alta
Iteración Asignada: 1	
Responsable: Miguel Cuenca	
Descripción: Para el usuario administrador se tiene que desarrollar varios *endpoints* para asignar el pedido a un cocinero que se encuentre libre. Por medio de métodos desarrollados se podrá ver el pedio y ver que cocinero está disponible para asignarle una orden.	
Observación: Se tiene que verificar la disponibilidad del cocinero y debe haber un usuario cocinero previamente registrado.	

TABLA XIII: Historia de usuario 06 – Desarrollar varios *endpoints* para los platos y productos.

HISTORIA DE USUARIO	
Identificador (ID): HU006	**Usuario:** Administrador
Nombre Historia: *Edpoints* para gestionar los platos y productos	
Prioridad en negocio: Alta	**Riesgo en desarrollo**: Alta
Iteración Asignada: 1	
Responsable: Miguel Cuenca	
Descripción: Para el usuario administrador se tiene que desarrollar varios *endpoints* para que pueda agregar platos (sopa, fuerte y bebida) y productos varios (colas, snacks, agua) que ofrezca la cafetería.	
Observación: Para poder agregar platos y productos, se manejará dos colecciones, ya que los platos son productos perecederos y los productos no perecederos.	

TABLA XIV: Historia de usuario 07 – Desarrollar varios *endpoints* para manejo del menú

HISTORIA DE USUARIO	
Identificador (ID): HU007	**Usuario: Administrador**
Nombre Historia: Desarrollar varios *endpoints* para manejo del menú	
Prioridad en negocio: Media	**Riesgo en desarrollo**: Alta
Iteración Asignada: 2	
Responsable: Miguel Cuenca	
Descripción: Para la creación del menú se debe desarrollar varios *endpoints* que nos permita ingresar ciertos datos, estos datos deben contar con sus respectivas validaciones.	
Observación: Para la creación de un menú es importante haber agregado platos para así poder armar el menú.	

TABLA XV: Historia de usuario 08 – Desarrollar varios *endpoints* para actualizar perfil.

HISTORIA DE USUARIO	
Identificador (ID): HU008	**Usuario:** Cliente, administrador, cocinero
Nombre Historia: Desarrollar varios *endpoints* para actualizar perfil.	
Prioridad en negocio: Alta	**Riesgo en desarrollo**: Alta
Iteración Asignada: 1	
Responsable: Miguel Cuenca	
Descripción: Para los usuarios se debe desarrollar varios *endpoints* para actualizar perfil cuando el cliente y cocinero lo requiera. La lectura y actualización del perfil se realiza mediante un formulario donde se ingresa la siguiente información: • Nombre • Avatar. • Teléfono	
Observación: Los datos que podamos actualizar dependerán de cada usuario.	

TABLA XVI: Historia de usuario 09 – Desarrollar varios *endpoints* para manejo de pedidos

HISTORIA DE USUARIO	
Identificador (ID): HU009	**Usuario:** Cliente
Nombre Historia: E*ndpoint* para manejo pedidos	
Prioridad en negocio: Alta	**Riesgo en desarrollo**: Alta
Iteración Asignada: 2	
Responsable: Miguel Cuenca	
Descripción: Para el cliente, en el sistema web puede consumir varios *endpoints* previamente desarrollados para crear pedidos cuando el cliente lo requiera. La creación de un pedido se realiza por un formulario, se selecciona la siguiente información. • Sopa • Proteína	

• Ensalada • Bebida
Observación: Los usuarios pueden consumir el *endpoint* para crear un dónde puede cambiar el estado del pedido. Los usuarios administrador y cocinero pueden ver los pedidos, administrador para aprobar y cocinero para despachar.

TABLA XVII: Historia de usuario 10 – Desarrollar varios *endpoints* para confirmar y cargar el comprobante de pago del pedido

HISTORIA DE USUARIO	
Identificador (ID): HU010	**Usuario:** Cliente
Nombre Historia: E*ndpoints* para confirmar y cargar el comprobante de pago del pedido.	
Prioridad en negocio: Alta	**Riesgo en desarrollo**: Alta
Iteración Asignada: 2	
Responsable: Miguel Cuenca	
Descripción: Para cliente, en el sistema web se debe desarrollar un *endpoint* para realizar la confirmación y el pago del pedido a través de un sistema en el que cargara el comprobante de pago.	
Observación: Si se carga un documento que no garantice el pago del pedido, dicho pedido no será aprobado para su realización.	

TABLA XVIII: Historia de usuario 11 – Desarrollar varios *endpoints* para visualizar el estado del pedido

HISTORIA DE USUARIO	
Identificador (ID): HU011	**Usuario:** Cliente
Nombre Historia: Desarrollar varios *endpoints* visualizar el estado del pedido.	
Prioridad en negocio: Alta	**Riesgo en desarrollo**: Alta
Iteración Asignada: 2	
Responsable: Miguel Cuenca	

Descripción: Para el cliente, en el sistema web se debe desarrollar varios *endpoints* para visualizar el estado en el que se encuentra su pedido, lo que se presenta en su vista es si el pedido está en observación, en preparación o finalizado.
Observación: El usuario cliente puede consumir los *endpoints* para listar un pedido a través de su ingreso al sistema web donde puede gestionar la actividad ya mencionada.

TABLA XIX: Historia de usuario 12 – Desarrollar varios *endpoints* para visualizar los pedidos asignados

HISTORIA DE USUARIO	
Identificador (ID): HU012	**Usuario:** Cocinero
Nombre Historia: E*ndpoints* para visualizar los pedidos asignados.	
Prioridad en negocio: Alta	**Riesgo en desarrollo**: Alta
Iteración Asignada: 1	
Responsable: Miguel Cuenca	
Descripción: El usuario cocinero visualizará en su vista principal el pedido que se le ha sido asignado.	
Observación: Ninguna.	

TABLA XX: Historia de usuario 13 – Desarrollar varios *endpoints* para cambiar el estado de los pedidos

HISTORIA DE USUARIO	
Identificador (ID): HU013	**Usuario:** Cocinero
Nombre Historia: Desarrollar varios *endpoints* para cambiar el estado de los pedidos	
Prioridad en negocio: Alta	**Riesgo en desarrollo**: Alta
Iteración Asignada: 1	
Responsable: Miguel Cuenca	

Descripción: EL usuario cocinero puede cambiar el estado del pedido por medio de un *endpoints* donde el podrá notificar el estado que se encuentra el pedido si está en proceso o se encuentra terminado.
Observación: Es importante que el pedido sea visible para los tres usuarios con sus respectivas restricciones.

Product Backlog

La **TABLA XXI** se enumera la prioridad que presenta cada requisito que se ha implementado en el *backend*. Estos requisitos tienen una clasificación de acuerdo con las necesidades de la cafetería.

TABLA XXI: *Product Backlog*

ELABORACIÓN DEL *PRODUCT BACKLOG*				
ID – HU	**HISTORIA DE USUARIO**	**ITERACIÓN**	**ESTADO**	**PRIORIDAD**
HU003	Desarrollar varios *endpoints* para registro de usuarios.	1	Finalizado	Alta
HU004	Desarrollar varios *endpoints* para para visualizar el pedido y la factura cancelada.	1	Finalizado	Alta
HU005	Desarrollar varios *endpoints* para asignar pedidos	1	Finalizado	Alta
HU006	Desarrollar varios *endpoints* para los platos y productos	1	Finalizado	Alta
HU007	Desarrollar varios *endpoints* para manejo del menú	1	Finalizado	Media
HU008	Desarrollar varios *endpoints* para actualizar perfil.	1	Finalizado	Alta
HU009	Desarrollar varios *endpoints* para manejo de pedidos	2	Finalizado	Alta

HU010	Desarrollar varios *endpoints* para confirmar y cargar el comprobante de pago del pedido	2	Finalizado	Alta
HU011	Desarrollar varios *endpoints* para visualizar el estado del pedido	2	Finalizado	Alta
HU012	Desarrollar varios *endpoints* para visualizar los pedidos asignados	*1*	Finalizado	*Alta*
HU013	Desarrollar varios *endpoints* para cambiar el estado de los pedidos	*1*	Finalizado	*Alta*

Sprint Backlog

La **TABLA XXII** se presenta los *Sprints*, los cuales ha permitido el desarrollo el *backend,* donde se describe las actividades y el tiempo determinado para los entregables que se han establecido.

TABLA XXII: *Sprint Backlog*

ELABORACIÓN DEL *SPRINT BACKLOG*						
ID – SB	**NOMBRE**	**MÓDULO**	**ID-HU**	**HISTORIA DEUSUARIO**	**TAREAS**	**TIEMPO ESTIMADO**
SB001	Implementación de los métodos para registro, actualización y autenticación de usuarios.	Sección informativa	HU001	Información de la cafetería	• Implementación de tipo público que devuelven la información importante para la comunidad de la politécnica.	70H
		Inicio de sesión	HU002	Iniciar sesión, cerra sesión y recuperar contraseña	• Diseño e implementación del *endpoint* para el inicio de sesión. • Diseño e implementación del *endpoint* para recuperar contraseña. • Consulta a la base de datos y autorización. • Cargar los módulos asignados	

		Registro de usuarios	HU003	Desarrollar varios *endpoints* para registro de usuarios.	• Diseño e implementación de *endpoint* para el registro de usuario. • Validación de los datos requeridos. • Verificación que el registro sea único. • Consulta en la Base de datos	
		Modulo pedido	HU004	Desarrollar varios *endpoints* para para visualizar el pedido y la factura cancelada.	• Diseño e implementación de *endpoints* para visualizar pedidos • Validación de datos. • Consulta en la Base de datos.	
		Modulo pedido	HU005	Desarrollar varios *endpoints* para asignar pedidos	• Diseño e implementación de *endpoints* para asignar pedidos del cliente. • Consulta en la Base de datos. • Validación de los datos requeridos	
		Módulo platos y productos	HU006	Desarrollar varios *endpoints* para los platos y productos	• Diseño e implementación de *endpoints* para para agregar, actualizar, visualizar y eliminar platos y productos. • Validación de datos.	

		Módulo menú	HU007	Desarrollar varios *endpoints* para manejo del menú	• Diseño e implementación de *endpoints* para modificar, agregar, actualizar y eliminar menú • Consulta en la base de datos. • Validaciones de los datos requeridos	
		Modulo perfil	HU008	Desarrollar varios *endpoints* para actualizar perfil.	• Diseño e implementación de *endpoints* para actualizar perfil. • Consulta en la base de datos. • Validaciones de los datos requeridos.	
		Módulos pedidos	HU009	Desarrollar varios *endpoints* para manejo de pedidos	• Diseño e implementación de *endpoints* para generar un pedido. • Consulta en la base de datos. • Validaciones de los datos requeridos	
		Módulo pedidos	HU010	Desarrollar varios *endpoints* para confirmar y cargar el comprobante de pago del pedido	• Diseño e implementación de *endpoints* para confirmar y cargar comprobante del pago. • Validación de los datos requeridos. • Consulta en la base de datos	

SB002	Implementación de los métodos para manejo productos, platos, menú y pedidos	Modulo pedidos	HU0011	Desarrollar varios *endpoints* para visualizar el estado del pedido.	• Diseño e implementación de *endpoints* para verificar el estado del pedido. • Validación de los datos requeridos. • Consulta en la base de datos	70H
		Modulo pedidos asignados	HU0012	*Endpoints* para visualizar los pedidos asignados	• Diseño e implementación de *endpoints* para visualizar los pedidos asignados. • Validación de los datos requeridos. • Consulta en la base de datos	
		Módulos pedidos	HU0013	*endpoints* para cambiar el estado de los pedidos	• Diseño e implementación de *endpoints* para cambiar el estado de los pedidos • Validación de los datos requeridos. • Consulta en la base de datos	
SB003	Pruebas en el *backend*	• Pruebas unitarias • Pruebas de rendimiento. • Prueba de aceptación				20H
SB004	Despliegue del sistema *web*	Despliegue del *backend* y Base de Datos NoSQL				10H
Documentación		• Informe Técnico. • Anexos.				50H
TOTAL						**240 H**

Diseño de la base de datos NoSQL

La **Fig. 36**, se presenta 5 colecciones que han permitido el desarrollo del *backend* con sus respectivos *endpoints*, facilitando el manejo de la información para las consultas.

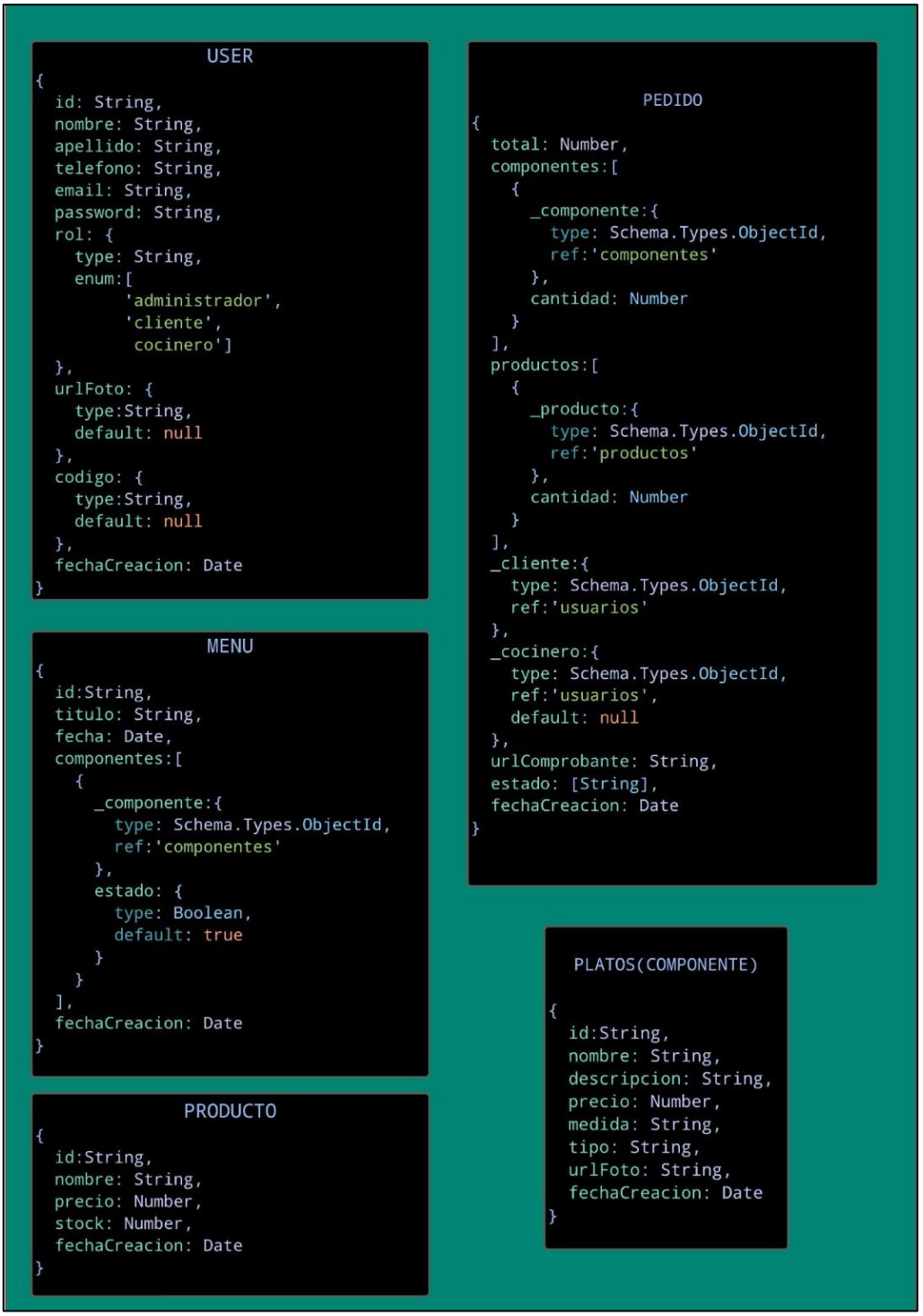


Fig. 36 Diseño de la Base de datos no relacional

Luego de haber culminado la fase de codificación se ha implementado una serie de pruebas, lo cual nos permite comprobar la calidad del código y el funcionamiento de cada uno de los módulos.

Pruebas unitarias

Se puede visualizar, en la **Fig. 37,** se muestra el código implementado para agregar un producto. A continuación, en la **Fig. 38** se puede verificar el resultado de la prueba. Adicional, en la **Fig. 39**, se muestra el código implementado para registrar usuario. A continuación, en la **Fig. 40** se puede verificar el resultado de la prueba.

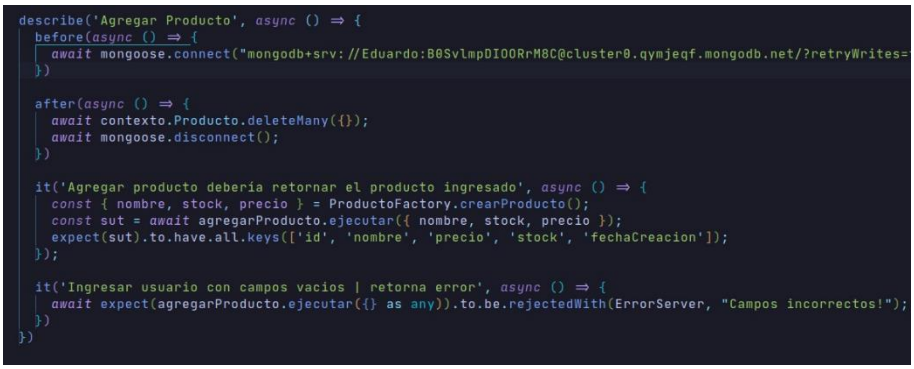


Fig. 37 Código agregar producto

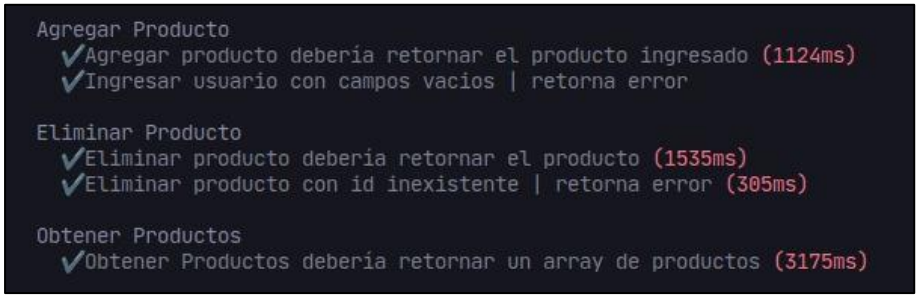


Fig. 38 Resultado de agregar producto

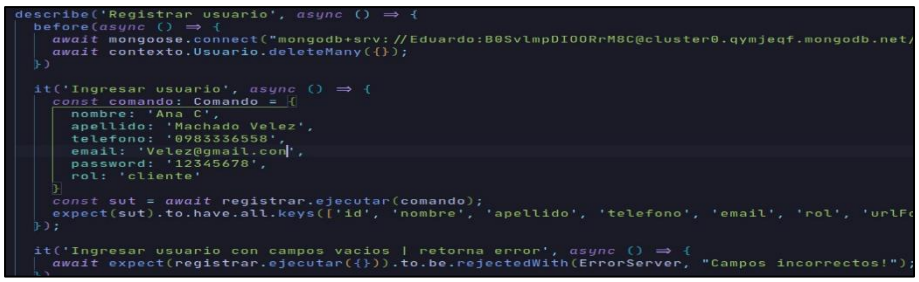


Fig. 39 Código registrar usuario

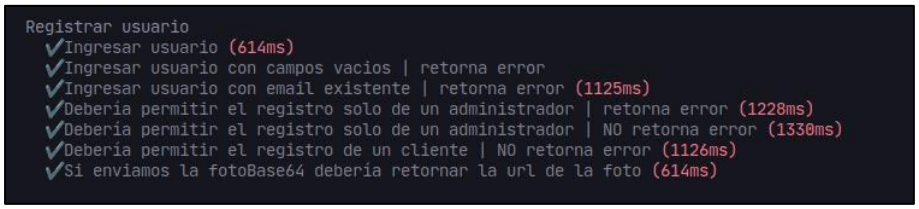


Fig. 40 Resultado de registrar usuario

Prueba de Compatibilidad

En esta parte, se presentan las pruebas de compatibilidad, para esta prueba se realizó con clientes HTTP que son *Postman* y *Thunder Client*. Es importante resaltar que en el presente proyecto las pruebas y testeo se ha realizado con *Postman*. Para la prueba, *Thunder Client* se ha utilizado la herramienta de *visual studio code* en específicamente su extensión, en la **Fig. 41** se van a apreciar el llamado con la herramienta de *Postman*, y en la **Fig. 42** con la herramienta de *Thunder client.*

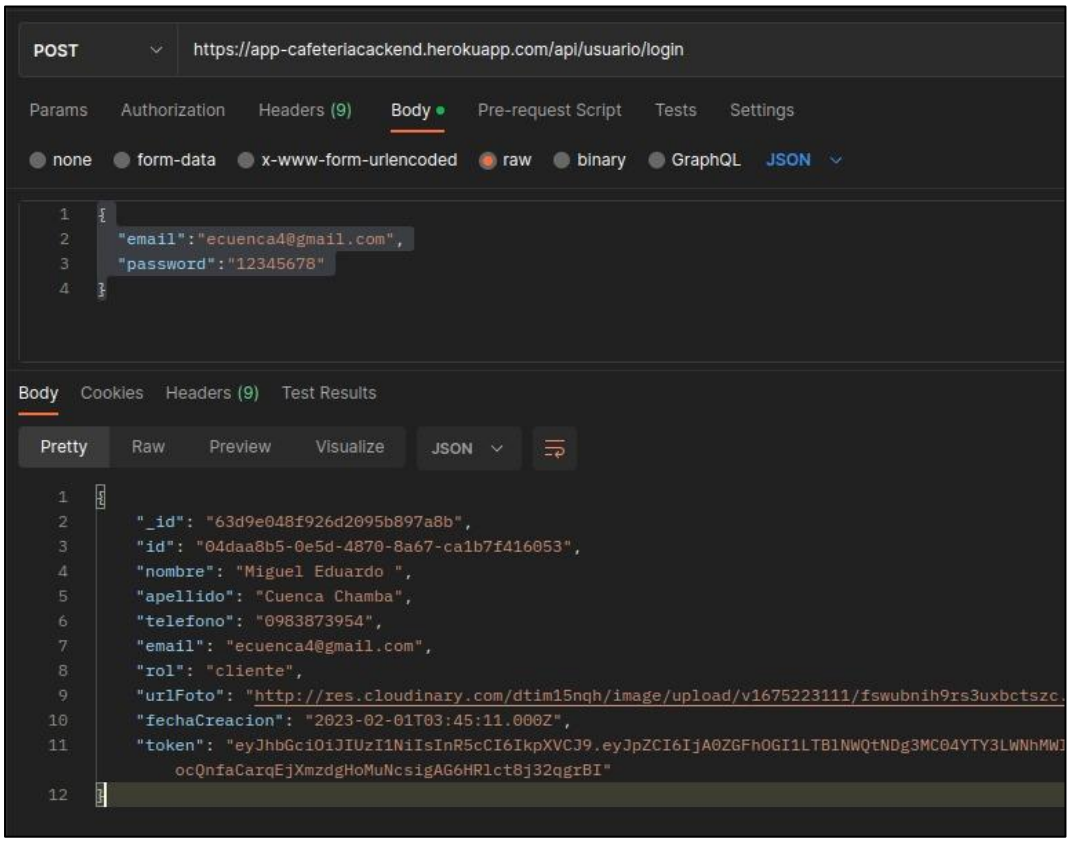


Fig. 41 *Postman*

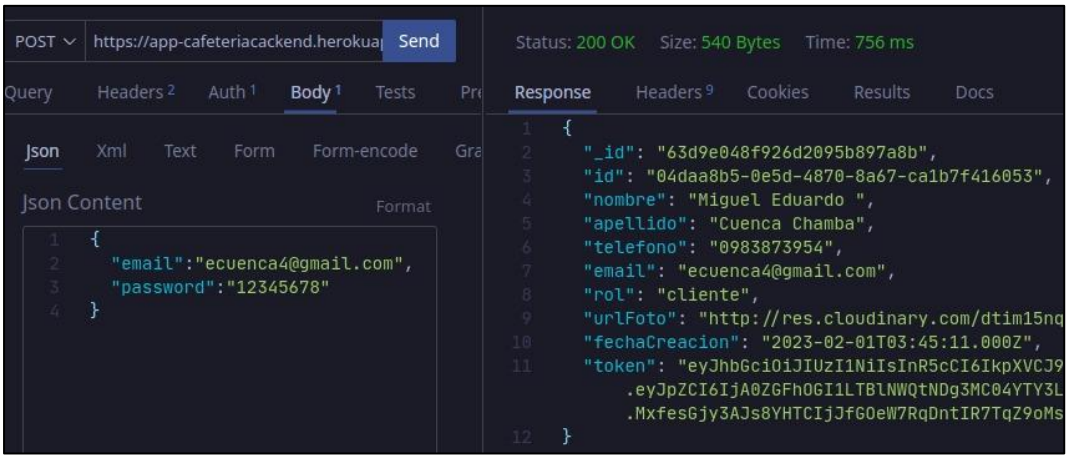


Fig. 42 *Thunder Client*

ANEXO III

A continuación, para acceder al enlace del Manual de Usuario se debe ingresar a la siguiente URL:

https://youtu.be/Oh3jnjLdypo

En el que se menciona de forma clara y detallada cada una de las funcionalidades del *backend* así como los perfiles que intervienen en la misma.

ANEXO IV

A continuación, se detallan las credenciales necesarias para acceder a los *endpoints* privados, así como el enlace al repositorio de GitHub donde se encuentra el código fuente. Además, en la sección de README del repositorio se describen los pasos necesarios para realizar la instalación del software de forma local.

Credenciales de acceso al *backend*

Para acceder a los *endpoints* que se encuentran en producción, ingresar a la siguiente dirección URL:

https://app-cafeteriacackend.herokuapp.com/docs/

Credenciales para generar un token de acceso a los *endpoints* privados y para acceder al usuario con perfil administrador en el sistema de escritorio son las siguientes:

- **Usuario:** admin@gmail.com
- **Contraseña:** 12345678

Las credenciales de acceso para un usuario con perfil de cocinero en el *backend* son las siguientes:

- **Usuario:** ecuenca4@gmail.com
- **Contraseña:** 123456789

Repositorio del código fuente del *backend*

Todo el código fuente del proyecto se encuentra alojado en el repositorio de GitHub, el cual puede ser accedido a través de la siguiente dirección URL:

https://github.com/Elsnight/CafeteriaBackend