ESCUELA POLITÉCNICA NACIONAL

ESCUELA DE FORMACIÓN DE TECNÓLOGOS

SISTEMA DE GESTIÓN DE DELIVERY PARA RESTOBAR DE CERVEZA ARTESANAL KÖHL BEER

DESARROLLO DE UN *BACKEND*

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

ROBERTO LI FONG SHIAO MORALES

roberto.shiao@epn.edu.ec

DIRECTOR: IVONNE FERNANDA MALDONADO SOLIZ

ivonne.maldonadof@epn.edu.ec

DMQ, marzo 2024

CERTIFICACIONES

Yo, **ROBERTO LI FONG SHIAO MORALES** 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.

ROBERTO LI FONG SHIAO MORALES

roberto.shiao@epn.edu.ec

Shiao281@gmail.com

Certifico que el presente trabajo de integración curricular fue desarrollado por **ROBERTO LI FONG SHIAO MORALES**, bajo mi supervisión.

IVONNE FERNANDA MALDONADO SOLIZ

ivonne.maldonadof@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.

ROBERTO LI FONG SHIAO MORALES

DEDICATORIA

A mi madre, fuente inagotable de amor y sabiduría, gracias por tu apoyo constante, tus palabras alentadoras y tu incansable dedicación, este logro es también tuyo. A mi padre, cuyo ejemplo de perseverancia y sacrificio ha sido mi inspiración, gracias por ser mi guía, su sabiduría y amor han sido los cimientos de mi éxito.

A mi hermano tus palabras motivadoras y tu inquebrantable apoyo han sido cruciales en este viaje, compartir contigo cada paso de este camino ha sido un regalo invaluable.

A mis sobrinos, quienes han llenado mi vida de alegría y risas, este logro es un testimonio para ustedes mostrándoles que, con esfuerzo y determinación, pueden alcanzar cualquier meta que se propongan, que este trabajo inspire sus propios sueños y aspiraciones.

A cada uno de ustedes, mi familia, mi soporte emocional y mi mayor tesoro, dedico con amor y gratitud este trabajo gracias por ser la fuerza que impulsa mis logros.

Roberto Shiao.

AGRADECIMIENTO

A mis amigos del colegio, Alexis, Andrés, Mateo, mi mejor amigo David y Alejandra, quienes han compartido risas, desafíos y momentos inolvidables a lo largo de estos años, gracias por ser fuente constante de apoyo y amistad sincera, cada uno de ustedes ha dejado una huella imborrable en mi corazón.

A mi grupo de amigos de la universidad, Jared, Joseph, Alejandro, Elvis, Camila y a grandes amistades como Paulina que sus palabras de ánimo y su sinceridad han hecho que este viaje sea aún más significativo y Alexis quien colaboró conmigo en la realización del proyecto, su dedicación, ingenio y paciencia han sido fundamentales para el éxito de este proyecto. Juntos hemos superado desafíos y alcanzados metas, estoy agradecido por tenerlos a mi lado.

A los profesores de la ESFOT, ingenieros Marina, Byron, Vanessa, Juan Pablo, y a Ivonne, mi tutora de tesis, gracias por su dedicación a la enseñanza, su orientación experta y el inapreciable conocimiento que han compartido conmigo, cada uno de ustedes ha sido una fuente invaluable de inspiración y aprendizaje.

A todos ustedes, amigos y mentores, les agradezco sinceramente por su contribución a mi desarrollo académico y personal, sin su apoyo este logro no habría sido posible.

Roberto Shiao.

ÍNDICE DE CONTENIDO

CERTIFICACIONES ..... **I**
DECLARACIÓN DE AUTORÍA ..... **II**
DEDICATORIA ..... **III**
AGRADECIMIENTO ..... **IV**
RESUMEN ..... **I**
ABSTRACT ..... **II**
1 DESCRIPCIÓN DEL COMPONENTE DESARROLLADO **1**
1.1. OBJETIVO GENERAL 2
1.2. OBJETIVOS ESPECÍFICOS 2
1.3. ALCANCE 2
1.4. MARCO TEÓRICO 4
2. METODOLOGÍA **6**
2.1 METODOLOGÍA DE DESARROLLO 6
Roles *7*
Artefactos *8*
2.2 DISEÑO DE LA ARQUITECTURA 10
Arquitectura de datos *10*
Patrón arquitectónico *11*
2.3 HERRAMIENTAS DE DESARROLLO 11
3. RESULTADOS **13**
3.1. *SPRINT* 0. CONFIGURACIÓN DEL AMBIENTE DE DESARROLLO 13
Definición de roles *13*
Recopilación de requerimientos *14*
Configuración de herramientas necesarias para el desarrollo *15*
Diseño de la Base de Datos en PostgresQL *16*
3.2. *SPRINT* 1. AUTENTICACIÓN Y REGISTRO 16
Endpoint para el registro de usuario *16*
Endpoint para el inicio de sesión *17*
Endpoint para cerrar sesión *17*
Endpoint para editar perfil del usuario *18*
3.3. *SPRINT* 2. MODULO ADMINISTRADOR 20
Endpoints para que se puedan gestionar (añadir y visualizar) las categorías *20*
Endpoints para que se puedan gestionar (añadir y visualizar) los productos *21*
Endpoint que permita ver los pedidos por estados (pagados, despachados, en camino y entregados) *23*
Endpoint para que se permita asignar un delivery al tener ya un pedido pagado *24*
Endpoint para actualizar el estado del pedido como despachado *25*
3.4. *SPRINT* 3. MODULO CLIENTE 26

Endpoint para crear pedidos *27*
Endpoint para listar órdenes del cliente por status *27*
3.5. *SPRINT* 4. MODULO CLIENTE 28
Endpoints para gestionar (crear y listar) direcciones para entrega de pedidos *29*
Implementación socket.io para la ubicación en tiempo real *30*
3.6. *SPRINT* 5. MODULO *DELIVERY* 30
Endpoint para listar pedidos por entregar del delivery según status *31*
Endpoints para actualización del estado del pedido como (en camino y entregado) *32*
3.7. *SPRINT* 6. PRUEBAS Y DESPLIEGUE 34
Pruebas unitarias *34*
Prueba de carga *35*
Pruebas de estrés *36*
Despliegue Web Service en Render y StorageDB en Vercel *37*
4. CONCLUSIONES 38
5. RECOMENDACIONES 39
6. REFERENCIAS BIBLIOGRÁFICAS 40
7. ANEXOS 42
ANEXO I 43
ANEXO II 44
ANEXO III 68
ANEXO IV 69

RESUMEN

El RestoBar KÖHL Beer en Quito busca destacarse en un mercado competitivo, ganar visibilidad y fidelizar clientes. La estrategia actual de entrega a domicilio mediante plataformas de terceros enfrenta desafíos financieros debido a las elevadas comisiones. Para superar este obstáculo, se propone el desarrollo de un "Sistema de Gestión de *Delivery*" con el objetivo de mejorar la eficiencia operativa y ampliar los servicios de entrega directa a los hogares de los clientes. Teniendo como objetivo reducir los costos asociados con terceros y aumentar la visibilidad en un mercado competitivo.

La implementación exitosa del *backend* incluye JSON Web Tokens con lo que se refuerza la autenticación y seguridad, destacando el uso de Node.js y Vercel, asegurando eficacia y productividad. El proyecto abarca pruebas unitarias, de carga y estrés con Jest y Apache JMeter, concluyendo que estas son esenciales para garantizar el rendimiento optimo, este trabajo de integración curricular se centra en ofrecer una solución integral y eficaz para los desafíos específicos del RestoBar KÖHL Beer, contribuyendo al ámbito académico mediante la aplicación de tecnologías y prácticas contemporáneas en el desarrollo de sistemas.

El presente documento se ha organizado por secciones: sección 1, descripción del componente, objetivos y el marco teórico. Sección 2, metodología *Scrum*, arquitectura y herramientas de implementación del componente *backend*. Sección 3, resultados de cada *Sprint*. Sección 4 y 5, conclusiones y recomendaciones.

PALABRAS CLAVE: Enpoints, Node.js, JSON, *delivery*, backend, tokens, autenticación, Jest, petición.

ABSTRACT

The RestoBar KÖHL Beer in Quito seeks to stand out in a competitive market, gain visibility and build customer loyalty. The current home delivery strategy through third-party platforms faces financial challenges due to high commissions. To overcome this obstacle, the development of a "Delivery Management System" is proposed with the aim of improving operational efficiency and expanding direct delivery services to customers' homes. Aiming to reduce costs associated with third parties and increase visibility in a competitive market.

The successful implementation of the backend includes JSON Web Tokens, which reinforces authentication and security, highlighting the use of Node.js and Vercel, ensuring efficiency and productivity. The project covers unit, load and stress tests with Jest and Apache JMeter, concluding that these are essential to guarantee optimal performance, this curricular integration work focuses on offering a comprehensive and effective solution for the specific challenges of the RestoBar KÖHL Beer, contributing to academia through the application of contemporary technologies and practices in systems development.

This document has been organized by sections: section 1, description of the component, objectives and the theoretical framework. Section 2, Scrum methodology, architecture and implementation tools of the backend component. Section 3, results of each Sprint. Section 4 and 5, conclusions and recommendations.

KEYWORDS: Endpoints, Node.js, JSON, delivery, backend, tokens, authentication, Jest, request.

1 DESCRIPCIÓN DEL COMPONENTE DESARROLLADO

El RestoBar de cerveza artesanal Köhl Beer, es un bar-restaurante que actualmente cuenta venta de sus productos y promociones directo en el local [1], es decir no cuenta con un servicio *delivery* que posibilite obtener productos puerta a puerta. Reduciendo el alcance y por ende pocas ganancias del negocio. Por otro lado, el local solo cuenta con publicidad mediante redes sociales que, aunque ha resultado bueno hasta el momento, no es suficiente para cumplir con los objetivos planteados como negocio, lo que significa el no llegar al número esperado de clientes.

Hablando de las empresas dedicadas a la entrega a domicilio, en Quito actualmente se tienen tarifas de entre 15% y 35% por cada pedido, precio que resulta elevado para poder ser asumido por los pequeños negocios [2], pero que sigue siendo un servicio necesario para que los negocios emprendedores sigan creciendo, haciendo que el no contar con dicho servicio sea una desventaja frente a la competencia, dado que después de la pandemia del COVID-19 todo negocio, empresa u organización ha acudido a la digitalización de sus procesos.

Por otro lado, si bien la descarga de una App no implica su uso, es un indicador de la penetración que están teniendo las aplicaciones de *delivery*. Por ejemplo, Pedidos Ya, que tiene más de 1 millón de usuarios registrados en Ecuador, está evidenciando un crecimiento de 70% en sus pedidos mensuales, con relación a 2022. Se considera que este crecimiento tiene relación con que la utilización de aplicaciones de *delivery* se ha vuelto cada vez más frecuente, lo que ha permitido alcanzar a más personas. De ahí que sea innegable el crecimiento que están teniendo los negocios por el simple hecho de hacer parte de su comercio a las aplicaciones de *delivery*, pues involucran en el proceso innovaciones y mejoras en el servicio y eso pasa a ser la respuesta a la demanda de los consumidores [3].

Es así como, el presente Trabajo de Integración Curricular trata sobre desarrollar el *backend* para un sistema de gestión de *delivery* para RestoBar de cerveza artesanal köhl Beer, que permita ofrecer un servicio de entrega puerta a puerta y muestre los productos del RestoBar a potenciales clientes. El producto de este desarrollo es el contar con una API Rest, componente *backend* de fácil acceso, siempre y cuando se cuente con las credenciales de acceso autorizadas, para el

consumo del lado de la aplicación móvil. Asimismo, este componente se puede integrar con plataformas externas. Adicional, se cuenta con roles para la correcta gestión del servicio puerta a puerta, así como también para el ingreso de información sobre las categorías y productos del negocio.

1.1. Objetivo general

Desarrollar el *backend* para la gestión y *delivery* para RestoBar de cerveza artesanal Köhl Beer.

1.2. Objetivos específicos

1. Determinar los requerimientos para el *backend*.
2. Diseñar e implementar la estructura de la base de datos relacional para que cumpla con los requerimientos.
3. Diseñar la arquitectura del *backend* conforme los requerimientos.
4. Codificar los *endpoints* según los requerimientos.
5. Ejecutar pruebas a los *endpoints* para comprobar su funcionamiento.
6. Desplegar el *backend* hacia producción.

1.3. Alcance

La digitalización trata de agilizar funciones de procesamiento de datos y transacciones de un negocio. En cuanto a procesos, la digitalización ha otorgado no solo agilidad sino también mayor alcance, y con esto, mayores posibilidades de ganancias. Es por esto que muchas empresas y negocios ya tienen presencia digital y se trabaja fuertemente desde este frente. La explosión digital surgida a partir del COVID-19 ha generado que si un negocio no posee alcance digital este esté en desventaja frente a sus actuales competidores [4].

El *backend* es una parte esencial de cualquier aplicación, ya que maneja todas las operaciones detrás de escena que permiten que la aplicación funcione de manera efectiva y segura. Trabaja junto con la aplicación móvil, que interactúa directamente con el usuario, para dar una experiencia completa y funcional. Para el desarrollo y la codificación del *backend* se involucran una variedad de lenguajes como Java,

Python, PHP y JavaScript, junto con *frameworks* reconocidos como ExpressJS, Laravel y Django, los cuales son ampliamente utilizados en este contexto. La elección depende de los requerimientos del sistema [5].

Este proyecto codifica, prueba y despliega el *backend* para el consumo por parte de la aplicación móvil para el negocio RestoBar Köhl Beer, posee tres roles. El administrador, que también puede desempeñarse como repartidor y cliente, asume la gestión del negocio, incluyendo visualizar pedidos por estados, asignar repartidores, categorizar y gestionar productos. El repartidos, quien también pueden actuar como cliente, recibe los pedidos asignados por el administrador, accediendo así a la información detallada y al estado actual de cada pedido. Finalmente, el rol cliente, tiene acceso a la información detallada de los productos (cervezas, bebidas sin alcohol, cócteles y *shots*, pizzas, platos fuertes, hamburguesas y más que pueden ser agregados por el rol administrador) ofrecidos para realizar compras.

La implementación del componente *backend* ha seguido la guía de *Scrum*, una base de datos relacional, un patrón arquitectónico MVC, una serie de pruebas y un despliegue a producción en render para la *Web Service* y Versel para el almacenamiento de la base de datos, concluyendo su desarrollo de manera exitosa.

Roles del aplicativo móvil:

- Administrador
- *Delivery*
- Cliente

El administrador puede:

- Iniciar y cerrar sesión.
- Visualización de pedidos por estado (pagados, despachados, en camino y entregados).
- Visualización del detalle de los pedidos.
- Asignar repartidores.

- Categorizar productos.
- Gestionar productos.
- Seleccionar el rol.

El *delivery* puede:

- Iniciar y cerrar sesión.
- Escoger pedido a realizar.
- Visualización de pedidos por estado (despachados, en camino y entregados).
- Editar estado del pedido.
- Cancelar envío del pedido.

El cliente puede:

- Registrarse.
- Iniciar y cerrar sesión.
- Editar su perfil personal.
- Visualizar información sobre productos por categorías.
- Visualización de pedidos por estado (pagados, despachados, en camino y entregados).
- Gestionar pedidos.
- Crear una dirección de entrega.
- Cancelar mediante pasarela de pago.

1.4. Marco Teórico

Backend

En el contexto del desarrollo de aplicaciones, se refiere a la parte no visible, que trabaja en segundo plano para hacer que todo funcione de manera fluida. Es el lado

del servidor que almacena y gestiona los datos, realiza cálculos y garantiza la seguridad y autenticación de los usuarios. Es responsable de procesar solicitudes, recuperar información de bases de datos y enviarla al *frontend*, la parte visible para los usuarios. Es esencial para habilitar la funcionalidad completa de cualquier aplicación, y su rendimiento y eficiencia son vitales para una experiencia en línea satisfactoria [6].

Base de datos relacional

Es un tipo de base de datos que almacena y proporciona acceso a puntos de datos relacionados entre sí. Las bases de datos relacionales se basan en el modelo relacional, una forma intuitiva y directa de representar datos en tablas. En una base de datos relacional, cada fila en una tabla es un registro con un ID único, llamado clave. Las columnas de la tabla contienen los atributos de los datos y cada registro suele tener un valor para cada atributo, lo que simplifica la creación de relaciones entre los puntos de datos [7].

Node.js

Es un entorno de ejecución de JavaScript multiplataforma y de código abierto. Este se ejecuta en el motor JavaScript V8, el núcleo de Google Chrome, fuera del navegador. Esto permite que Node.js tenga un gran rendimiento. Una aplicación Node.js se ejecuta en un único proceso, sin crear un nuevo hilo para cada solicitud. Node.js proporciona un conjunto de primitivas de E/S (Entrada/Salida) asíncronas en su biblioteca estándar que evitan que el código JavaScript se bloquee y, en general, las bibliotecas en Node.js se escriben utilizando paradigmas sin bloqueo, lo que hace que el comportamiento de bloqueo sea la excepción y no la norma [8].

Postman

Es una herramienta de desarrollo API (interfaz de programación de aplicaciones) que ayuda a crear, probar y modificar APIs. Tiene la capacidad de realizar varios tipos de solicitudes HTTP (GET, POST, PUT, PATCH), guardar entornos para su uso posterior y convertir la API a código para varios lenguajes (como JavaScript, Python) [9].

2. METODOLOGÍA

El estudio de caso se refiere a un enfoque de investigación empírica que se centra en fenómenos contemporáneos, la esencia de esta metodología reside en la recopilación de datos, un elemento fundamental para cualquier investigación auténtica. Este método distingue su aplicación fuera del entorno controlado del laboratorio, abordando situaciones que se extraen del mundo real y abarcando tanto objetos concretos como procesos en análisis. Además, cabe resaltar la noción de fronteras difusas indicando que, en ciertas circunstancias, la delimitación de un caso puede depender de la elección razonada del investigador en lugar de criterios evidentes [10].

Este Proyecto de Titulación adopta un enfoque de estudio de caso para explorar los beneficios de la implementación del *backend* eficiente para una aplicación móvil para RestoBar KÖHL Beer. La investigación se centra en mejorar los procesos administrativos diarios, como la gestión de categorías y productos, la gestión de pedidos y el seguimiento de estos, además de la gestión de los pedidos por parte del *delivery*. El objetivo fundamental es identificar estrategias que optimicen cada uno de estos procesos, asegurando que el *backend* sea totalmente funcional para ser consumido por la aplicación móvil.

2.1 Metodología de Desarrollo

Las metodologías ágiles representan enfoques para el desarrollo de software que priorizan la flexibilidad, la colaboración y la adaptabilidad sobre procesos y planificaciones rígidas. Estas metodologías destacan la entrega constante de software funcional desde etapas tempranas, así como la capacidad de ajustarse ágilmente a cambios y retroalimentación del cliente.

Una mayor eficiencia en el desarrollo de un proyecto supone un trabajo altamente rápido y de calidad y la participación activa de los implicados maximiza la mejora continua entre desarrolladores y clientes, contribuyendo a generar un producto superior, incluso sobresaliendo entre la competencia. Esta metodología facilita el cumplimiento parcial de todas las fases del proyecto, asegurando que los tiempos de entrega se cumplan y permitiendo un control rápido del progreso para ofrecer un producto de calidad, incluso en la primera entrega, se puede obviar características

que el cliente prefiera omitir, evitando posibles descontentos y asegurando la confianza necesaria para ser un proveedor potencial [11].

Roles

En la metodología *Scrum*, los roles son funciones específicas que desempeñan los miembros del equipo para garantizar un desarrollo eficiente del proyecto. Los roles principales en *Scrum* para este proyecto son:

Product Owner

Asume la tarea de determinar las prioridades y maximizar el valor del producto en desarrollo, esta labor implica la gestión de presupuestos, la contratación del equipo de desarrollo y la comunicación efectiva sobre el valor que aporta el producto. Como representante del negocio, toma de decisiones es esencial, el *Product Owner* debe tener la autoridad necesaria, además en su papel de intraemprendedor, el *Product Owner* actúa como un ágil *Product Manager*, evaluando el valor generado y utilizando la flexibilidad de la entrega por *Sprint* para incrementar ese valor de manera efectiva [12]. La **Tabla 2.1** muestra quién es el que maneja este rol.

Scrum Master

Actúa como un líder orientado al servicio, facilita la implementación eficiente de la Metodología *Scrum*, su enfoque está en aspectos comerciales, siendo responsable del retorno de inversión (ROI) del proyecto. Transmite la visión del proyecto al equipo, formaliza las funcionalidades en historias para el *Product Backlog*, principalmente su rol se centra en respaldar al equipo y la organización para aprovechar al máximo *Scrum* [12]. La persona que se encuentra a cargo de desempeñar este rol se puede visualizar en la **Tabla 2.1**

Development Team

Grupo de profesionales altamente capacitados y multifuncionales que colabora de manera conjunta en la ejecución y construcción del proyecto, comprometidos con las historias acordadas al inicio de cada *Sprint*, este equipo juega un papel crucial en el éxito del proyecto, ya que cada *Sprint* se centra en construir y entregar un incremento del producto. **Tabla 2.1** muestra quién es el que maneja este rol.

Tabla 2.1: Roles designados

Rol	Integrantes
Product Owner	Ing. Adrián Huaca.
Scrum Master	Ing. Ivonne Maldonado, MSc.
Development Team	Sr. Roberto Shiao.

Artefactos

Son herramientas esenciales para la gestión de proyectos, permiten tener transparencia de lo realizado a lo largo del proyecto [11].

Recopilación de Requerimientos

Es esencial para la definición de roles, la identificación de responsabilidades y la planificación del equipo, esto aborda el "quién" y el "qué" del proceso, estableciendo claramente las funciones de cada miembro para el desarrollo, además, se vincula con el "cuándo" y el "dónde", que están representados por el *Sprint*, marcando el tiempo y el lugar en el que se llevan a cabo las actividades y en cuanto al "cómo" y el "por qué", se eligen herramientas que utilizan los miembros de *Scrum*, asegurando la eficiencia y justificando las decisiones tomadas en el proceso de desarrollo [11].

La lista de requerimientos se puede visualizar en **ANEXO II.**

Historias de Usuario

Tras el levantamiento de requerimientos se empiezan a crear las historias de usuario (HU), una historia de usuario se configura como una explicación general e informal de una función de software, narrada desde la perspectiva del usuario final, donde su intención es detallar cómo dicha función de software aporta valor al cliente [13].

La **Tabla 2.2** ejemplifica una Historia de Usuario, empleada para expresar la funcionalidad solicitada por el usuario. En el **Historias de Usuario** se pueden observar el resto de Historias de Usuario.

Tabla 2.2: Formato de Historias de Usuario – HUB002

HISTORIA DE USUARIO	
Identificador: HU002	**Usuario:** Cliente, *delivery,* administrador.
Nombre historia: Iniciar sesión, cerrar sesión.	
Prioridad en Negocio: Alta	**Riesgo en Negocio:** Alta
Iteración asignada: 1	
Responsable: Roberto Shiao	
Descripción: Como cliente, *delivery* o administrador, necesita generar varios *endpoints* que permitan acceder a la ampliación móvil. Para iniciar sesión, se valida las credenciales, una vez que el usuario este logeado tiene la posibilidad de cerrar sesión.	
Observación: Ninguna.	

Product Backlog

Constituye una detallada lista de tareas esenciales para la culminación del proyecto, facilitando la organización del equipo de desarrollo. Cada tarea cuenta con un tiempo asignado, garantizando que el equipo conozca los plazos para su inicio y finalización, manteniendo el proyecto en la línea temporal establecida, el *Product Backlog* es adaptable a contratiempos, conforme el proyecto progresa, la lista se reduce, pudiendo añadir o modificar tareas según las necesidades emergentes [14]. En el **Product Backlog** se detalla este elemento.

Sprint Backlog

Es una subdivisión del *Product Backlog*, comprende tareas específicas que deben completarse en el plazo estimado del *Sprint*, es una forma efectiva de desglosar las tareas del *Product Backlog*, especialmente en proyectos complejos, actuando como hitos intermedios hacia la meta final. Una vez definidas las tareas para un *Sprint Backlog* y su tiempo de ejecución, rara vez se modifican y si no se pueden completar todas las tareas, se trasladan al siguiente *Sprint* sin alterar la estructura original [14]. En el **Sprint Backlog** se detalla este elemento.

2.2 Diseño de la arquitectura

Diseñar la arquitectura de software es un proceso crucial en el desarrollo de sistemas, pues en este paso se definen las estructuras fundamentales y los componentes del software, además el diseño de la arquitectura también implica tomar decisiones estratégicas sobre la distribución de tareas y responsabilidades entre los diversos elementos, proporcionando una base sólida para el desarrollo y la evolución del software.

Arquitectura de datos

Es la estructura organizativa que rige la gestión completa de los datos, desde su adquisición y transformación hasta su distribución y utilización. Este enfoque detalla el diseño estratégico para los datos, delineando cómo circulan a través de los sistemas de almacenamiento [15]. Definiendo el camino para una gestión eficiente y efectiva de los datos, esta arquitectura proporciona la base para garantizar la integridad y la disponibilidad de la información en todo el ecosistema del sistema de información.

La **Figura 2.1** muestra la representación gráfica del diseño de la base de datos, mostrando de manera visual el modelado de las entidades y las relaciones implementadas para la administración de datos.

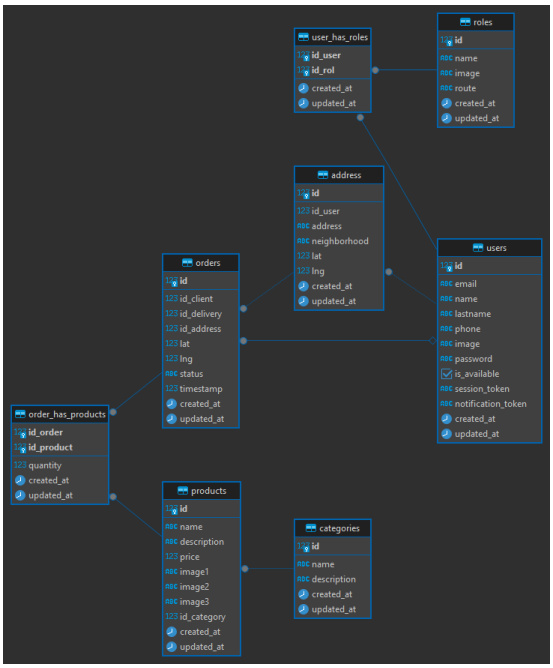


Figura 2.1: Modelo ER

Patrón arquitectónico

MVC (Modelo-Vista-Controlador) es un patrón de diseño comúnmente empleado en el desarrollo de software para organizar la interfaz de usuario, datos y lógica de control, enfocándose en la separación de la lógica empresarial y la presentación. Simplifica el mantenimiento y mejora la escalabilidad [16].

Los tres componentes del patrón de diseño de software MVC se pueden explicar de la siguiente manera:

- **Modelo**: Gestiona datos y lógica empresarial para garantizar la coherencia y el funcionamiento adecuado de la aplicación.
- **Vista**: Encargada del diseño y la presentación de la interfaz de usuario, proporcionando la experiencia visual para los usuarios.
- **Controlador**: Dirige y gestiona las interacciones del usuario, transmitiendo comandos a los modelos y vistas correspondientes en la aplicación.

La **Figura 2.2** exhibe el patrón arquitectónico integrado en la creación del *backend*, junto con las herramientas empleadas para el funcionamiento del componente.

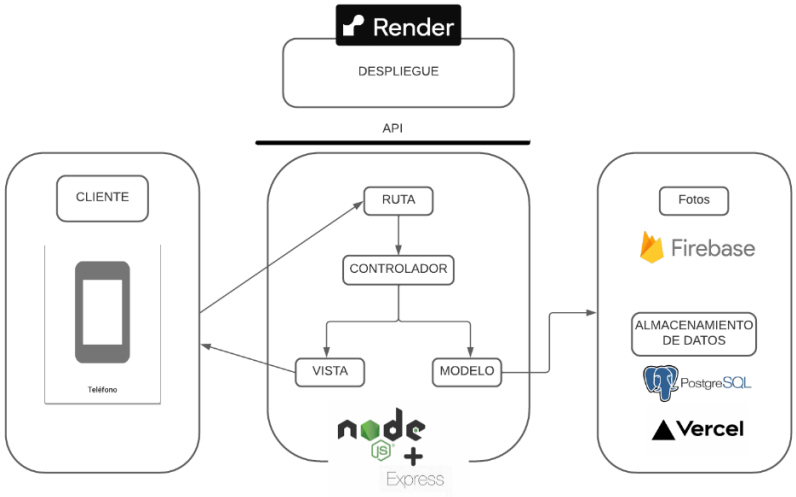


Figura 2.2 Modelo Arquitectónico – *Backend*

2.3 Herramientas de desarrollo

Después de definir el patrón arquitectónico, en esta sección se detallan las herramientas utilizadas en la creación de los *endpoints*, maximizando el potencial de las tecnologías disponibles para garantizar la eficiencia y productividad del

componente *backend*. La **Tabla 2.3** resume el conjunto de herramientas que han desempeñado un papel significativo en el desarrollo del *backend*.

Tabla 2.3: Herramientas para el desarrollo - *Backend*

Herramienta	Justificación
Visual Studio Code	Editor de código fuente ligero y potente, sus funciones avanzadas han permitido una programación eficiente.
Git y GitHub	Ha permitido tener un control y rastreo de los cambios en el código. La plataforma de alojamiento ha permitido gestionar el proyecto de manera colaborativa en línea.
Node js	Entorno de ejecución de JavaScript del lado del servidor, que permite ejecutar código JavaScript fuera del navegador, también es eficiente y escalable para aplicaciones web.
PostgreSQL	Sistema de gestión de bases de datos relacional de código abierto que ofrece confiabilidad, rendimiento y extensibilidad, ha sido utilizado para almacenar y gestionar los datos del proyecto.
Postman	Plataforma de colaboración para el desarrollo de APIs, ha permitido diseñar y probar las APIs de manera eficiente y simplificada.
Render	Servicio que ofrece una infraestructura de nube escalable para ejecutar aplicaciones web y móviles, ha permitido implementar y gestionar el proyecto de manera sencilla.
Vercel	Servicio de almacenamiento en la nube, ha permitido gestionar y distribuir archivos estáticos de manera eficiente.
Firebase	Servicio de almacenamiento en la nube, diseñado para almacenar y recuperar archivos de usuario, como imágenes o videos, ha facilitado la gestión de recursos multimedia.

3. RESULTADOS

Esta sección está dedicada a mostrar los resultados que se han obtenido en cada *Sprint*, demostrando así el cumplimiento de los objetivos planteados como parte del desarrollo del componente *backend.*

3.1. *Sprint* 0. Configuración del ambiente de desarrollo

Para este *Sprint* se incluyen las siguientes actividades:

- Definición de roles
- Recopilación de requerimientos
- Configuración de herramientas necesarias para el desarrollo
- Diseño de la Base de Datos en PostgresQL

Definición de roles

En la **Figura 3.1**, se representan los roles definidos. Además, se exhiben las funcionalidades activadas correspondientes a cada rol, proporcionando una visualización clara de la estructura operativa.

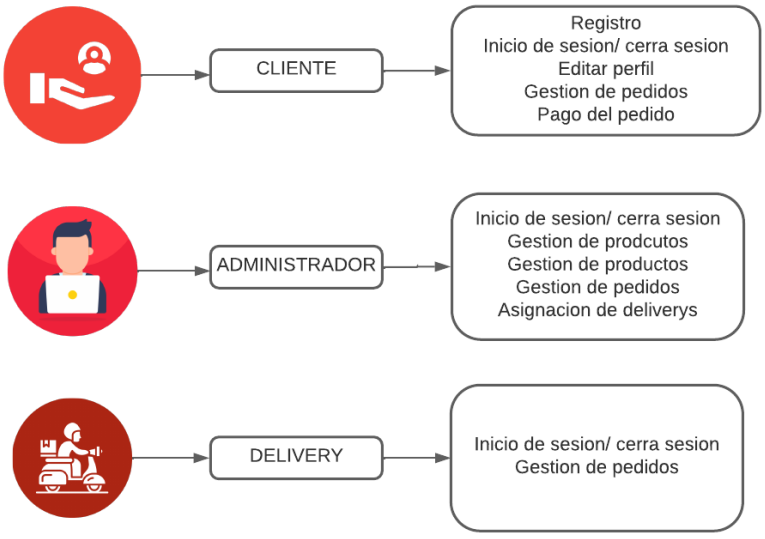


Figura 3.1: Roles de usuarios

Recopilación de requerimientos

Endpoint para el registro del usuario

Este *endpoint* facilita al usuario llevar a cabo un registro, con el rol predeterminado siendo principalmente cliente.

Endpoints de autenticación, inicio de sesión y cierre de sesión

El *endpoint* de autenticación permite proteger las rutas generando un token por medio de JWT para cada usuario al realizar el inicio de sesión. El *endpoint* de inicio de sesión valida las credenciales y da paso al *dashboard* según su rol. Y finalmente el *endpoint* de cierra sesión se borra el token generado al iniciar sesión.

Endpoint para listar usuarios por ID

El *endpoint* sirve para poder obtener los usuarios junto a los roles que se les ha otorgado.

Endpoints para listar y crear categorías

El *endpoint* de visualización de categorías lista todas las categorías agregadas por el rol administrador. Mientras que el *endpoint* de crear categorías sirve para la creación de las distintas categorías que agrupan los diferentes productos.

Endpoints para listar y crear productos

El *endpoint* de visualización de productos lista todos los productos agregados por el rol administrador junto a su categoría. Mientras que el *endpoint* de crear producto sirve para la creación del producto según la categoría.

Endpoints para listar y crear direcciones

El *endpoint* de visualización de direcciones lista todas las direcciones agregadas por el rol cliente para la entrega del pedido. Mientras que el *endpoint* de crear direcciones crea una nueva dirección con cierta *data* que permite al rol *delivery* conocer el punto exacto de la entrega.

Endpoint para edición de información personal del usuario

Endpoint que permite actualizar la información personal solicitada en el registro.

Endpoint para asignación del *delivery*

Este *endpoint* permite listar solo los usuarios que se les ha otorgado el rol de *delivery*.

Endpoints para listar ordenes por estados

Estos *endpoints* muestran los pedidos clasificados según sus estados actuales, permitiendo que cada rol pueda visualizar los pedidos en distintos estados.

Endpoints para actualizar estados de los pedidos

Los *endpoints* encargados de la actualización de los estados de las ordenes, ordenan los pedidos según los estados que se vayan marcando en cada rol.

Endpoint para actualizar la ubicación en tiempo real

En este *endpoint* se implementa con socket.io que ayuda a la actualización continua de la latitud y longitud para poder visualizar la ubicación en tiempo real.

Configuración de herramientas necesarias para el desarrollo

Para la configuración de las herramientas, se ha llevado a cabo la instalación de las herramientas para el desarrollo, incluyendo Visual Studio Code, Postman, Node.js y PostgreSQL. La **Figura 3.2** muestra la estructura del proyecto.

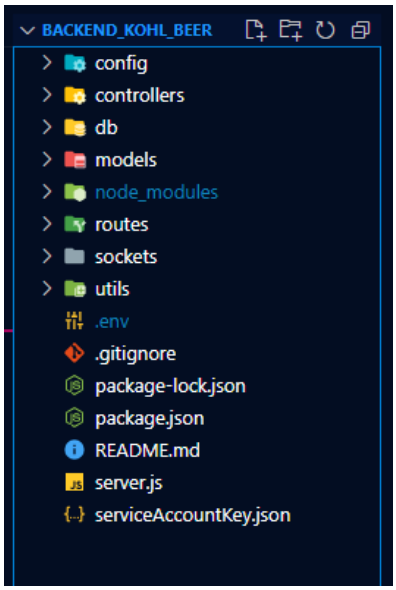


Figura 3.2: Estructura del proyecto

Diseño de la Base de Datos en PostgresQL

La base de datos relacional ha sido creada en PostgreSQL de manera local y con el respaldo de Vercel, esto facilita el almacenamiento de datos del sistema en la nube. El diseño de la base de datos se presenta en la sección de **Diseño de la arquitectura** dentro del apartado **Arquitectura de Datos** de este documento.

3.2. *Sprint* 1. Autenticación y registro

Para este *Sprint* se incluyen las siguientes actividades:

- *Endpoint* para el registro de usuario
- *Endpoint* para el inicio de sesión
- *Endpoint* para cerrar sesión
- *Endpoint* para editar perfil del usuario

Endpoint para el registro de usuario

La funcionalidad de registro de usuario genera un nuevo usuario con el rol predeterminado "cliente". Para consumir este *endpoint*, no es necesario validar el token, ya que este se genera al iniciar sesión, la solicitud de este *endpoint* se realiza mediante el método *POST*, ya que implica la inserción de datos.

La **Figura 3.3** a presenta el resultado exitoso del registro de un nuevo usuario.

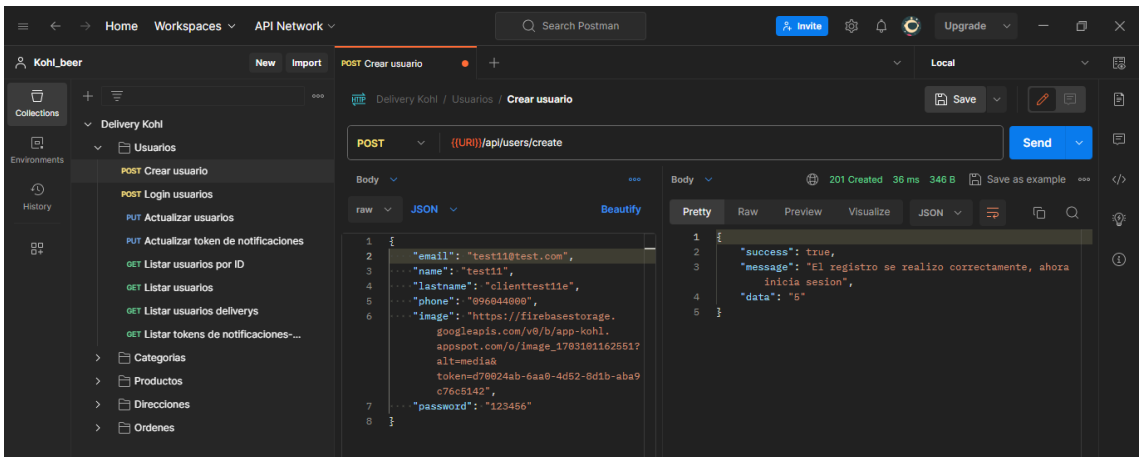


Figura 3.3: Método *POST* registro de usuario

Endpoint para el inicio de sesión

En la funcionalidad de inicio de sesión, el usuario proporciona las credenciales necesarias (correo electrónico y contraseña), una vez que el usuario inicia sesión, se genera un token mediante JWT. La solicitud de este *endpoint* es de tipo *GET*, ya que lee los datos del usuario.

En la **Figura 3.4**, se presenta un resultado exitoso del inicio de sesión, donde se puede visualizar el token generado y el rol predeterminado.

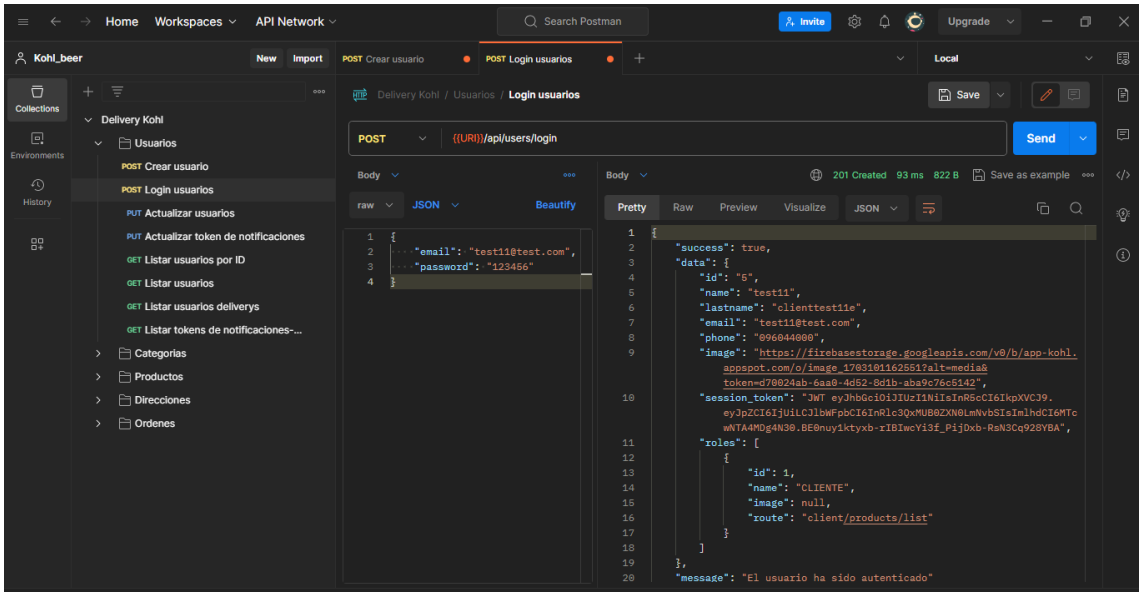


Figura 3.4: Método *POST* inicio de sesión

Endpoint para cerrar sesión

El *endpoint* destinado para cerrar sesión ofrece una forma segura y controlada para que los usuarios concluyan su sesión activa, al realizar una solicitud mediante el método *POST*, se invalida el token de autenticación asociado con la sesión actual, previniendo así cualquier acceso no autorizado posterior. Esta medida garantiza la protección de la privacidad y seguridad del usuario al cerrar la sesión de manera adecuada, ya que implica la eliminación del token.

En la **Figura 3.5**, se presenta el resultado positivo al cerrar sesión. Además, en la **Figura 3.6**, donde se realiza una solicitud de tipo *GET* para listar los usuarios, se puede verificar que el token generado al iniciar sesión ha sido eliminado.

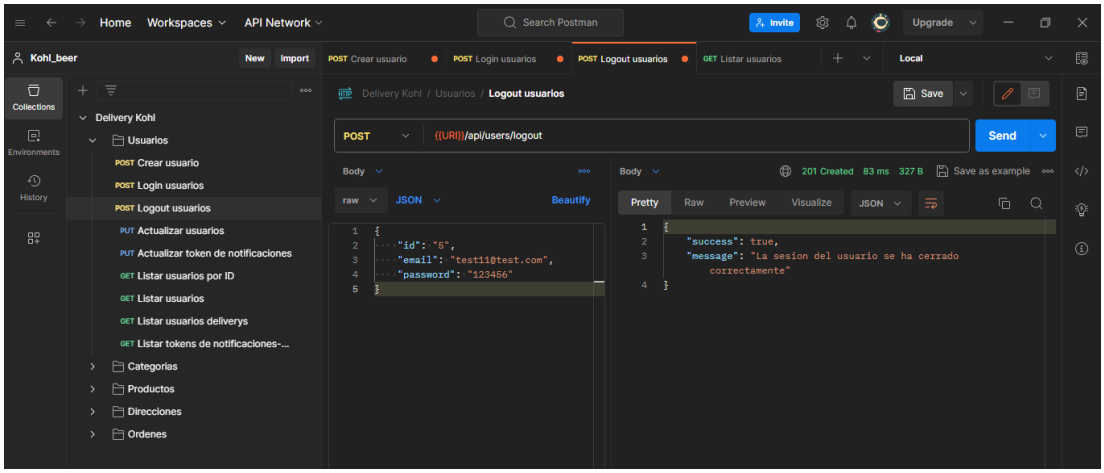


Figura 3.5: Método *POST* cerrar sesión

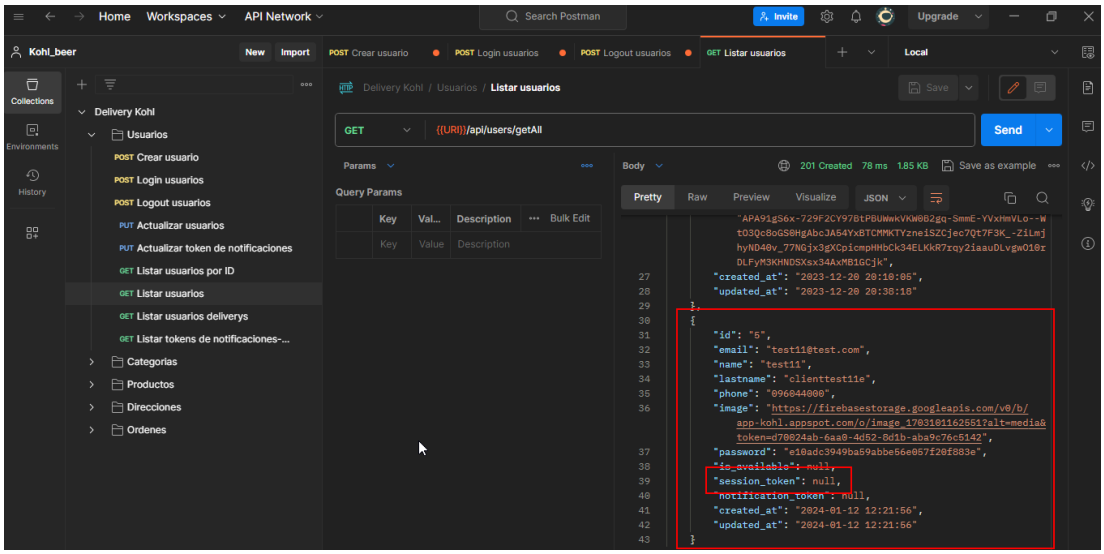


Figura 3.6: Método *GET* listar usuarios

Endpoint para editar perfil del usuario

El *endpoint* destinado a la edición del perfil del usuario desempeña una función crucial al posibilitar que los usuarios actualicen y personalicen la información de sus cuentas. Para llevar a cabo esta acción, se utiliza una solicitud de tipo *PUT*, la cual envía los datos modificados del usuario. Es importante destacar que, para validar la actualización de los datos, se incluye el token correspondiente.

En las **Figura 3.7** y **Figura 3.8** se presenta un resultado positivo tras realizar la solicitud de edición del perfil, acompañado del token agregado para la validación del usuario, este proceso confirma la efectividad de la implementación y asegura la actualización exitosa de la información del perfil del usuario. Se puede observar en

la **Figura 3.9** la actualización de los datos con el método *GET* para listar los usuarios.

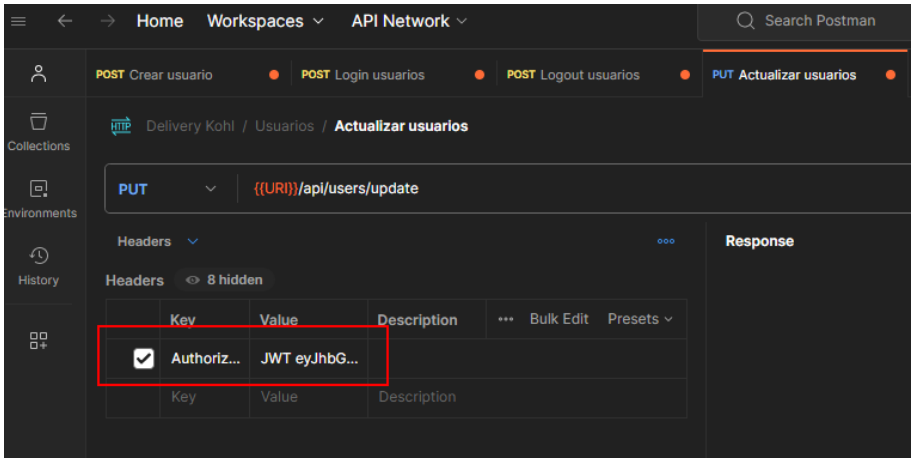


Figura 3.7: Token

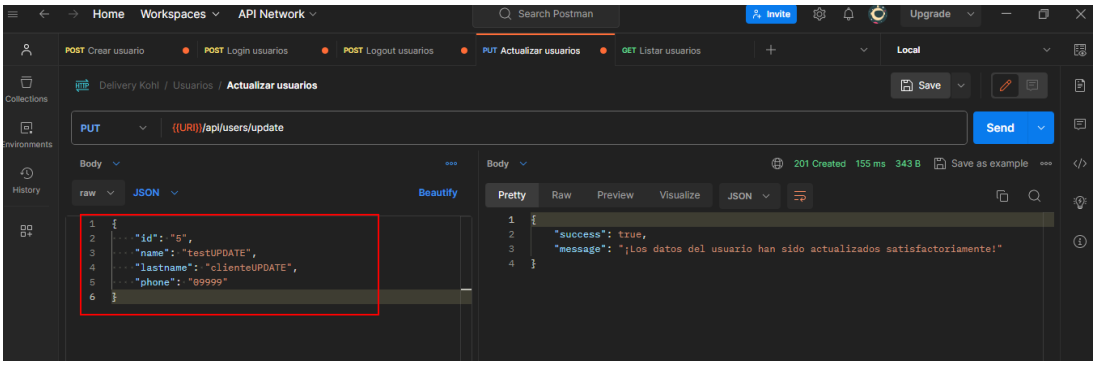


Figura 3.8: Método *PUT* editar perfil del usuario

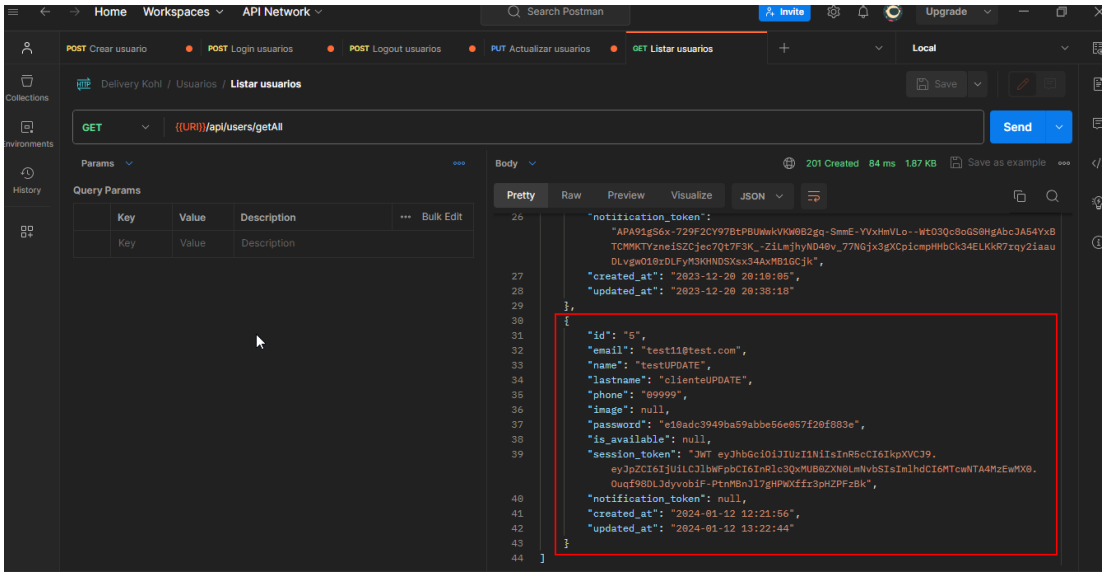


Figura 3.9: Método *GET* listar usuarios

3.3. *Sprint* 2. Modulo administrador

Para este *Sprint* se incluyen las siguientes actividades:

- *Endpoints* para que se puedan gestionar (añadir y visualizar) las categorías
- *Endpoints* para que se puedan gestionar (añadir y visualizar) los productos
- *Endpoint* que permita ver los pedidos por estados (pagados, despachados, en camino y entregados)
- *Endpoint* para que se permita asignar un *delivery* al tener ya un pedido pagado
- *Endpoint* para actualizar el estado del pedido como despachado

Endpoints para que se puedan gestionar (añadir y visualizar) las categorías

En los siguientes *endpoints*, se evidencia la funcionalidad de añadir y visualizar categorías mediante las cuales los productos son clasificados, tal como se muestra en la **Figura 3.10**. Para la creación de una categoría, se requiere el uso del *token* JWT asignado a cada usuario al iniciar sesión, garantizando así la autenticación y seguridad del proceso.

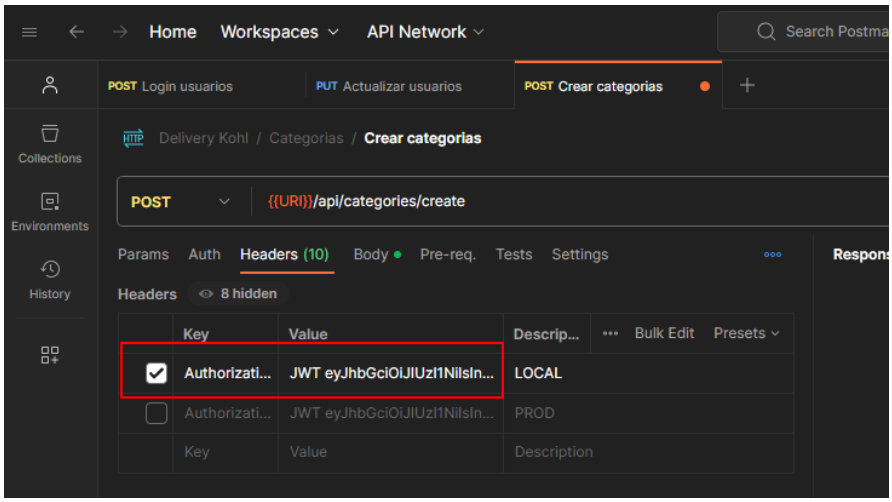


Figura 3.10: Autenticación

Para la creación de una categoría, se emplea el método *POST* en la solicitud del *endpoint*, ya que este método se utiliza para la inserción de los datos asociados a la categoría.

En la **Figura 3.11** se muestra el resultado con éxito al crear una categoría.

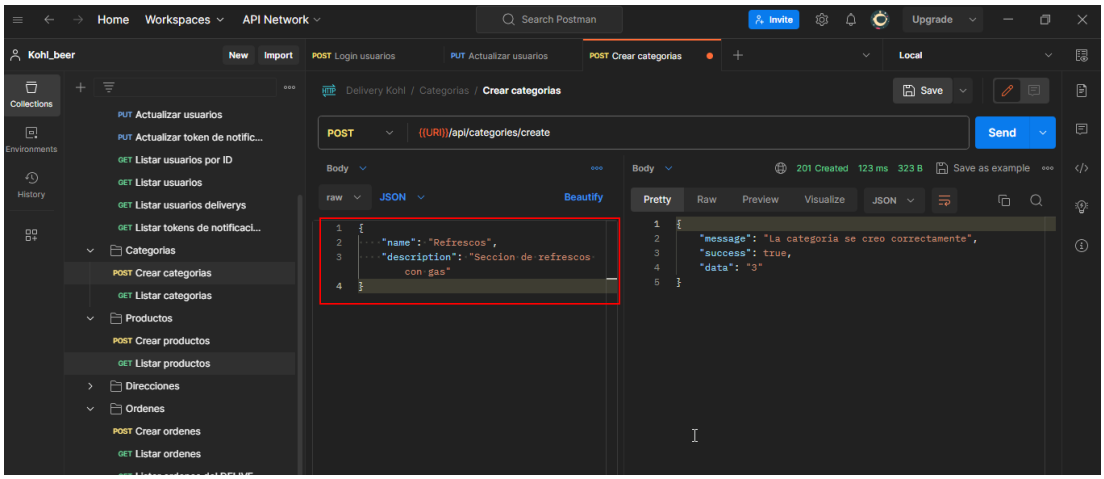


Figura 3.11: Método *POST* crear categoría

Para visualizar las categorías agregadas por el administrador, se efectúa una solicitud al *endpoint* utilizando el método *GET*, permitiendo la lectura de las categorías que ya han sido almacenadas. En la **Figura 3.12** se observa la solicitud con éxito al leer las categorías agregadas.

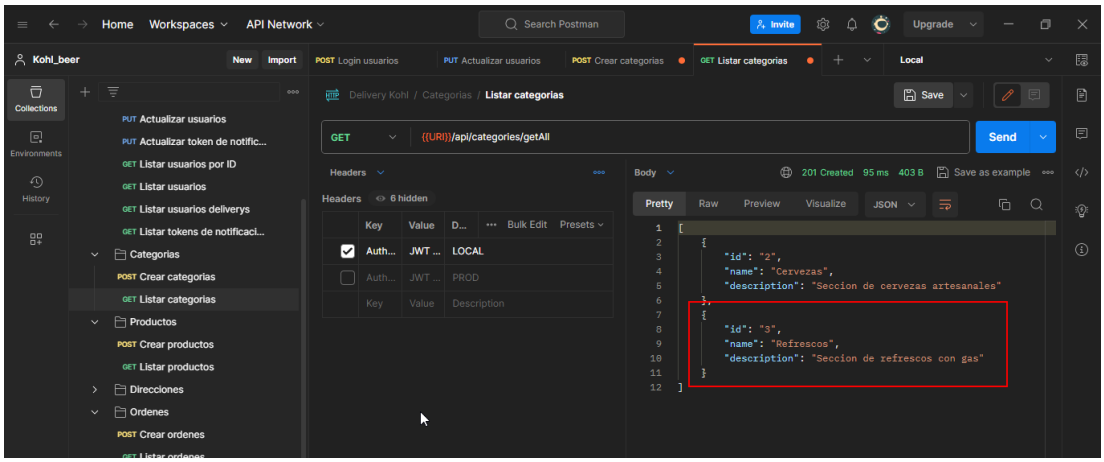


Figura 3.12: Método *GET* listar categorías

Endpoints para que se puedan gestionar (añadir y visualizar) los productos

En los siguientes *endpoints*, se evidencia la funcionalidad de añadir y visualizar los productos los cuales son clasificados por las categorías, tal como se muestra en la **Figura 3.13.** Para la creación de un producto, se requiere el uso del token JWT asignado a cada usuario al iniciar sesión, garantizando así la autenticación y seguridad del proceso.

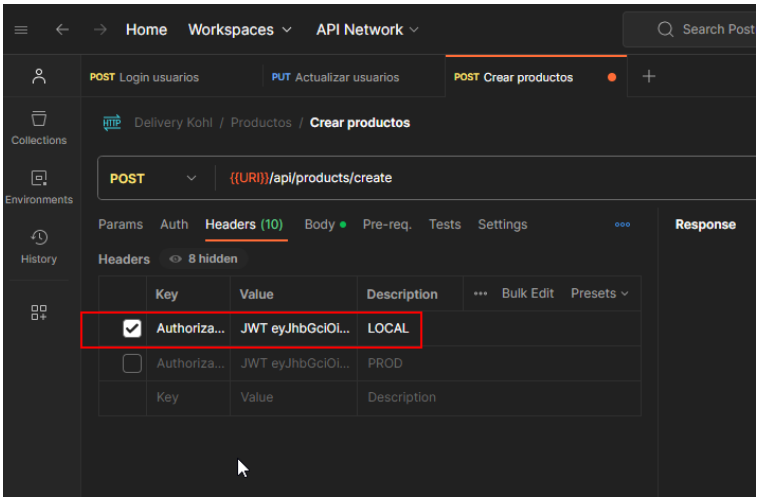


Figura 3.13: Autenticación

Para la creación de un producto, se emplea el método *POST* en la solicitud del *endpoint*, ya que este método se utiliza para la inserción de los datos asociados a la categoría. En **Figura 3.14** la se muestra el resultado con éxito al crear una categoría.

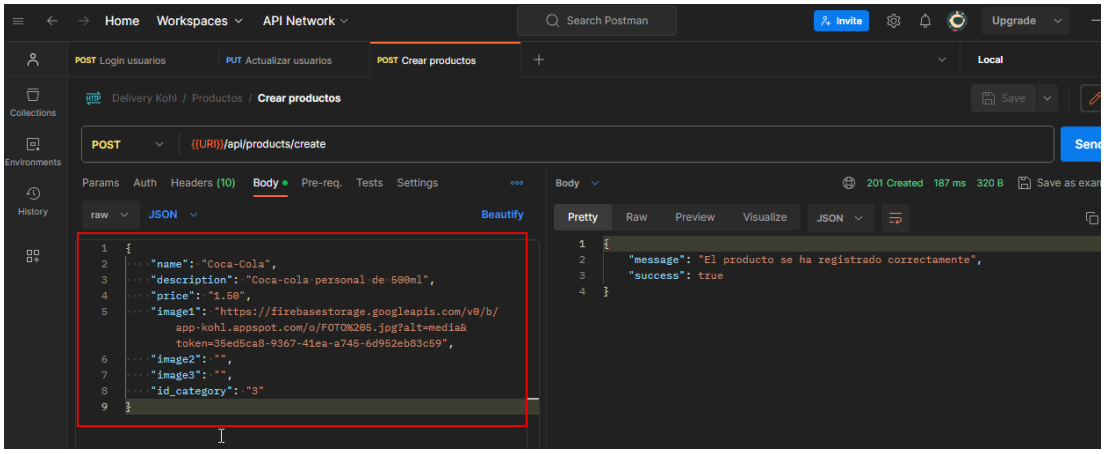


Figura 3.14: Método *POST* crear producto

Para visualizar los productos agregados por el administrador, se efectúa una solicitud al *endpoint* utilizando el método *GET*, permitiendo la lectura de los productos por ID de categoría que se han almacenado. En **Figura 3.15** la se observa la solicitud con éxito al leer las categorías agregadas.

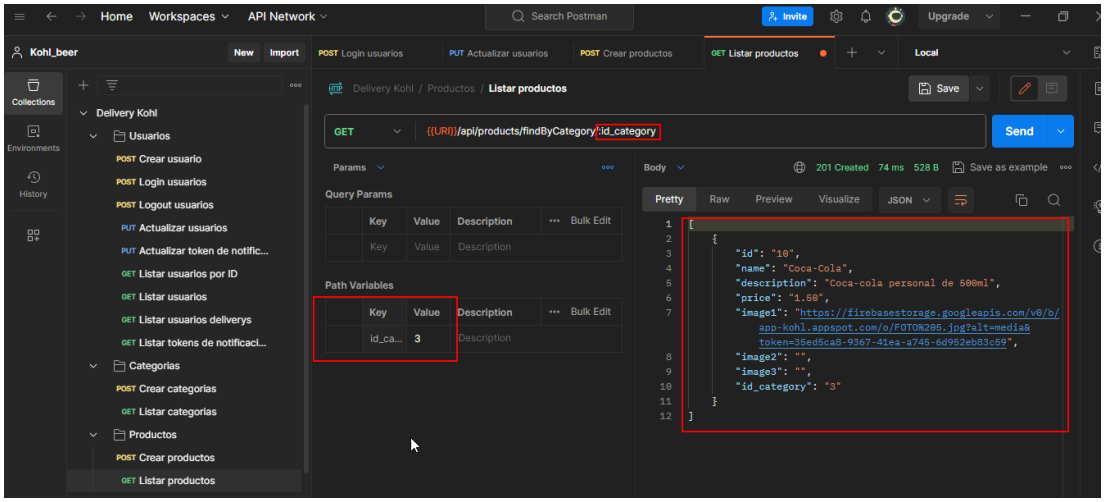


Figura 3.15: Método *GET* listar productos por id de categorías

***Endpoint* que permita ver los pedidos por estados (pagados, despachados, en camino y entregados)**

Esta funcionalidad resulta útil para listar los pedidos de acuerdo con su estado actual. En el *endpoint*, se especifica el estado, ya sea "pagado", "despachado", "en camino" o "entregado", y se visualizan todos los pedidos asociados a dicho estado.

Para acceder a la visualización de los pedidos por estado, es necesario autenticarse utilizando el token JWT generado al iniciar sesión, como se ilustra en la **Figura 3.16**.

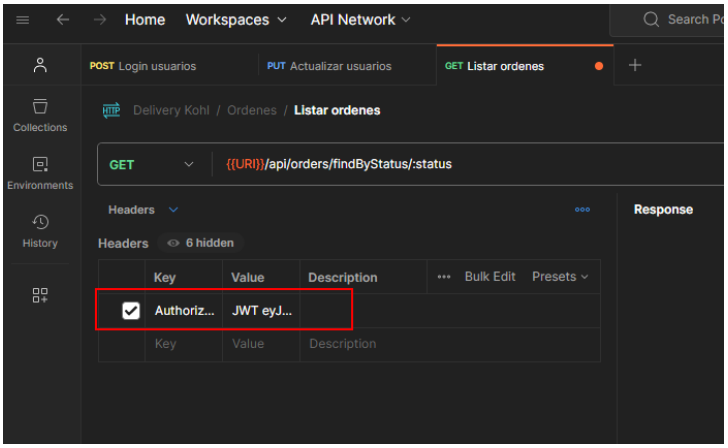


Figura 3.16: Autenticación

En la **Figura 3.17**, se emplea el método *GET* para acceder a la información sobre los pedidos almacenados, clasificados por estados. Además, se observa que la solicitud se realiza con éxito.

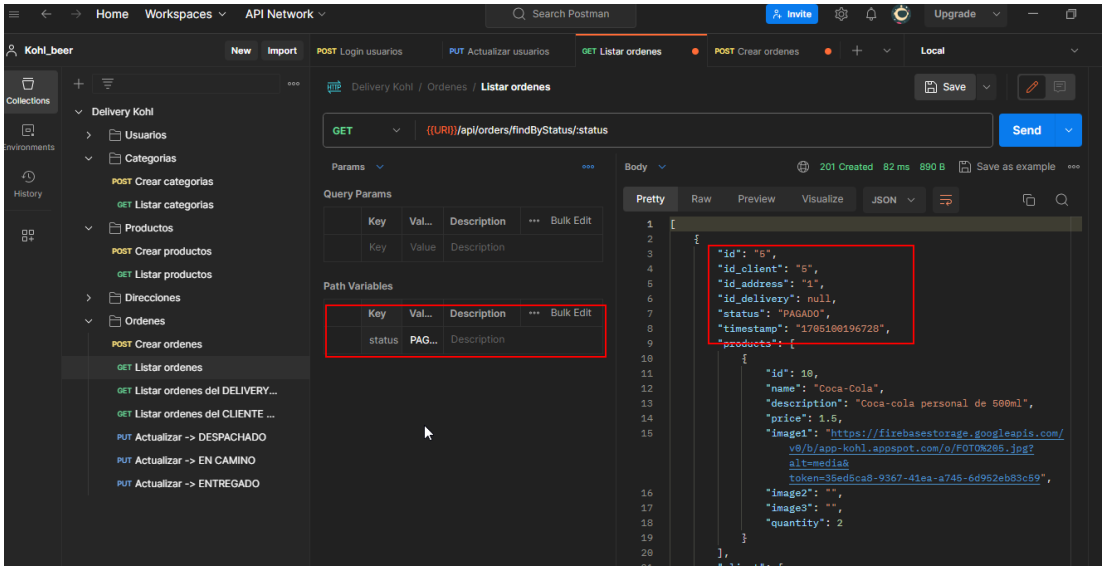


Figura 3.17: Método *GET* listar pedidos por estados

Endpoint para que se permita asignar un *delivery* al tener ya un pedido pagado

La funcionalidad siguiente es esencial para la asignación de un repartidor al tener un pedido en estado "pagado", el *endpoint* enumera los usuarios con el rol de *delivery*, lo que facilita la selección de usuarios con este rol durante la asignación de un repartidor en la interfaz correspondiente. En la **Figura 3.18**, se autentica al usuario mediante la transmisión del token generado por JWT.

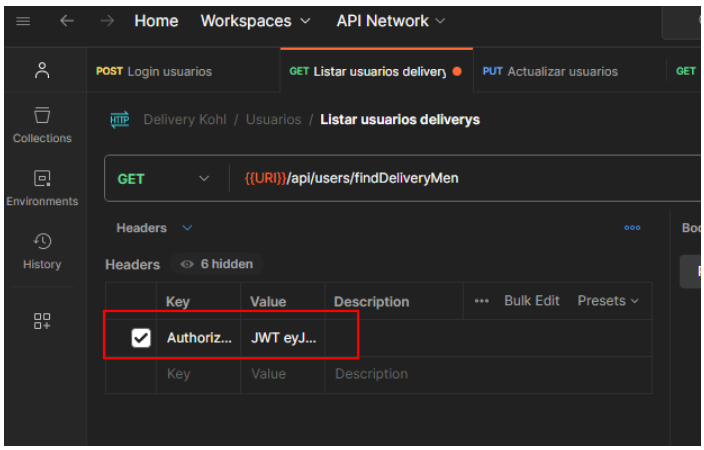


Figura 3.18: Autenticación

En la **Figura 3.19**, se observa la exitosa solicitud del método *GET* para la visualización del listado de usuarios con el rol de repartidor.

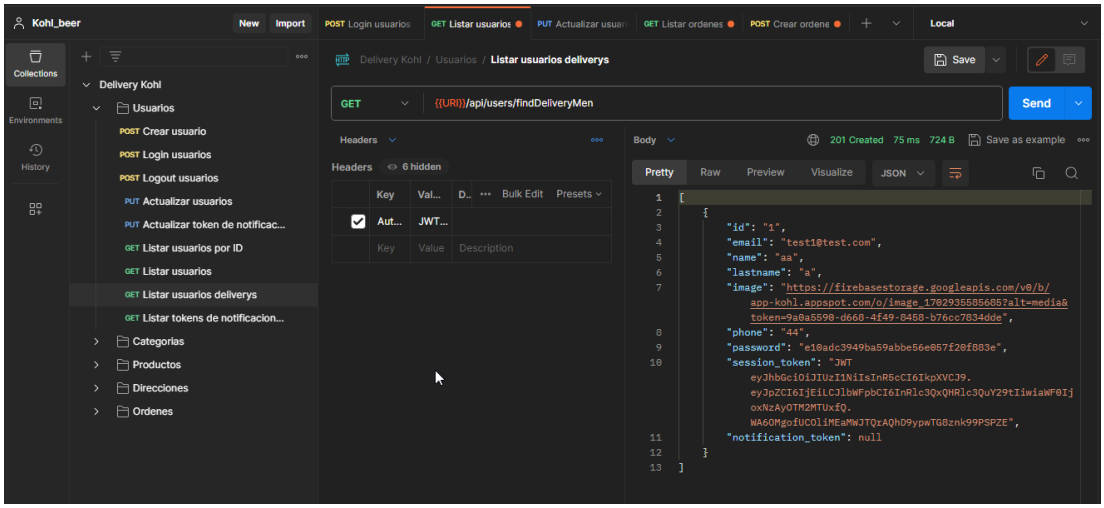


Figura 3.19: Método *GET* usuarios *delivery*

Endpoint para actualizar el estado del pedido como despachado

Esta funcionalidad es crucial para marcar el pedido como "despachado" una vez que se ha asignado un *delivery*, permitiendo así comenzar a visualizar las órdenes pendientes de entrega. Para realizar la solicitud, es necesario enviar el token generado por JWT y autenticar al usuario, como se detalla en la **Figura 3.20**.

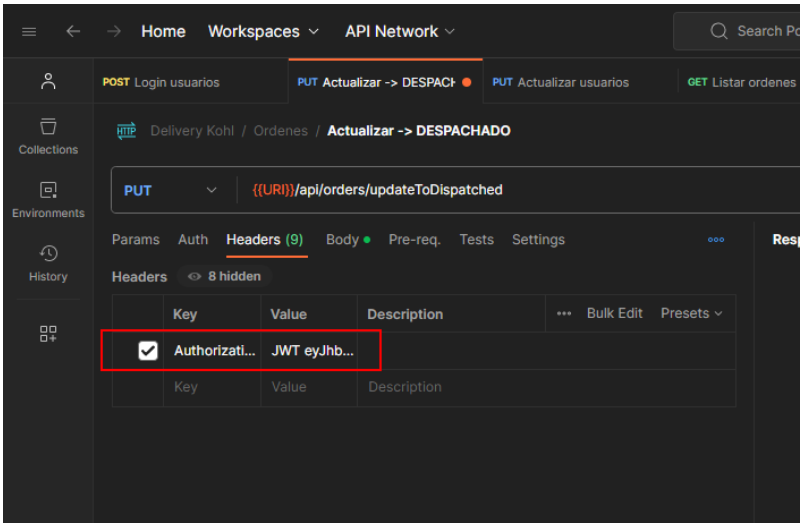


Figura 3.20: Autenticación

La solicitud de actualización se realiza mediante el método *PUT* para cambiar el estado de "pagado" a "despachado". La **Figura 3.21** exhibe la ejecución exitosa de la solicitud, y en la **Figura 3.22** se verifica que el pedido ahora figura como despachado.

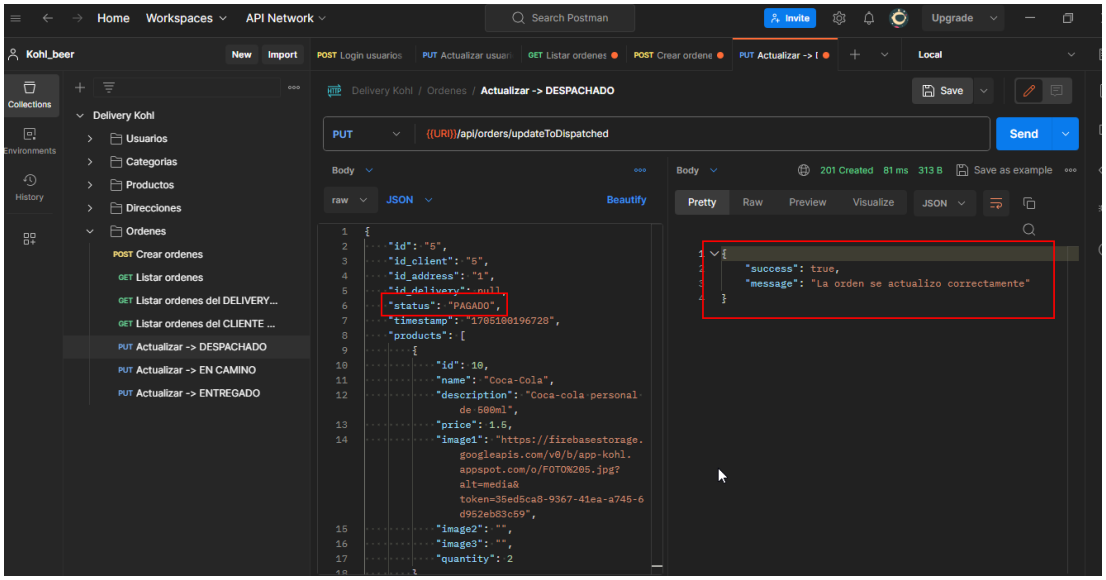


Figura 3.21: Método *PUT* para actualización a despachado

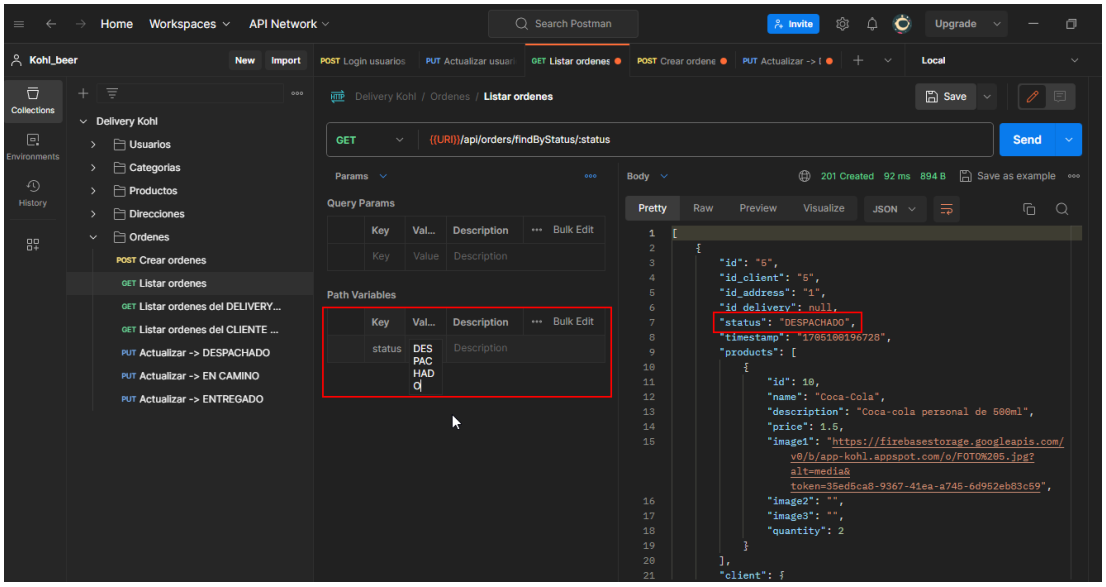


Figura 3.22: Verificación actualización de estado ha despachado

3.4. *Sprint* 3. Modulo cliente

Para este *Sprint* se incluyen las siguientes actividades:

- *Endpoint* para crear pedidos
- *Endpoint* para listar órdenes del cliente por *status*

Endpoint para crear pedidos

Esta funcionalidad permite al cliente realizar un pedido con varios productos que haya elegio del listado de productos, al realizar un pedido se marca por defecto como pagado. Como se muestra en la **Figura 3.23** se debe pasar el token generado por JWT para la autenticación del usuario.

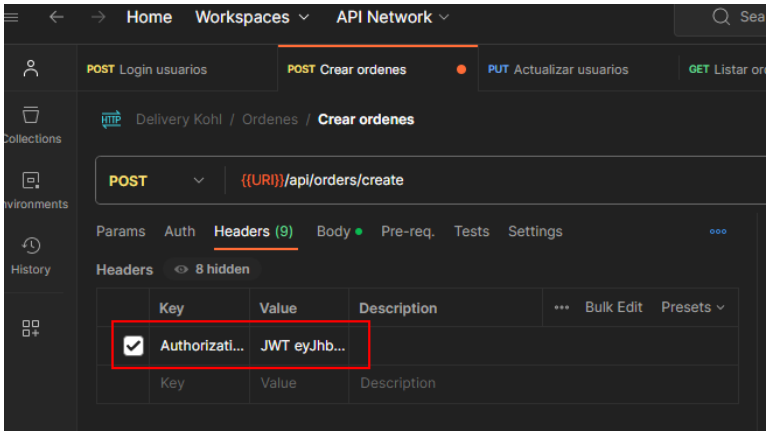


Figura 3.23: Autenticación

En la **Figura 3.24**, se evidencia la exitosa ejecución de la solicitud *POST* para la creación del pedido, en esta solicitud, se incorporan varios parámetros esenciales, tales como el ID del cliente, la dirección y un conjunto que incluye los IDs de los productos junto con sus respectivas cantidades.

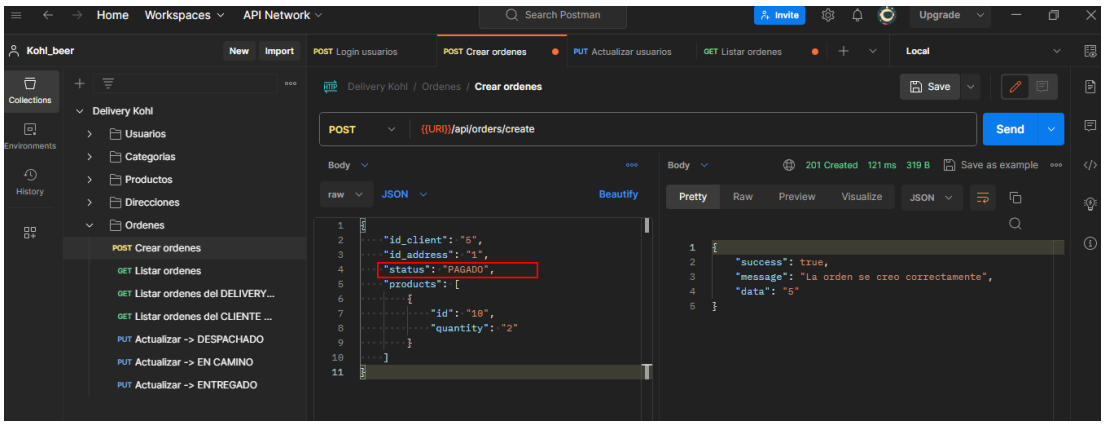


Figura 3.24: Método *POST* crear pedidos

Endpoint para listar órdenes del cliente por *status*

Esta funcionalidad permite listar los pedidos realizados por el cliente según su estado, brindando la posibilidad de revisar los detalles de cada uno.

Como se evidencia en la **Figura 3.25**, es necesario proporcionar el token generado por JWT para autenticar al usuario y acceder a esta característica.

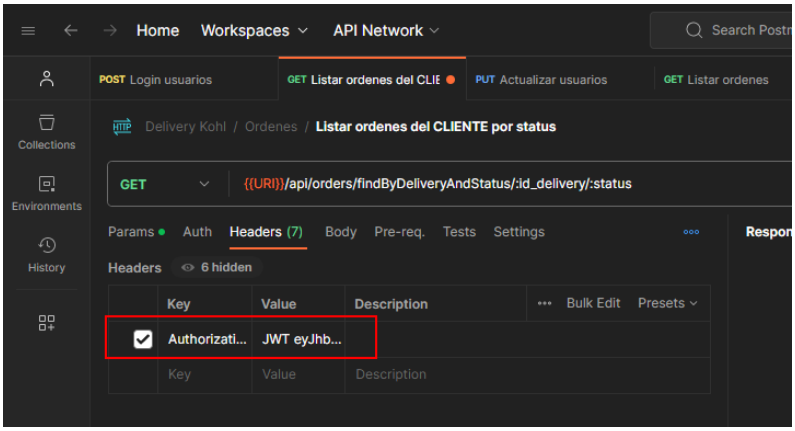


Figura 3.25: Autenticación

Para visualizar las órdenes realizadas por el cliente, se efectúa una solicitud utilizando el método *GET* para obtener los pedidos registrados. La exitosa ejecución de esta solicitud se evidencia en la **Figura 3.26**.

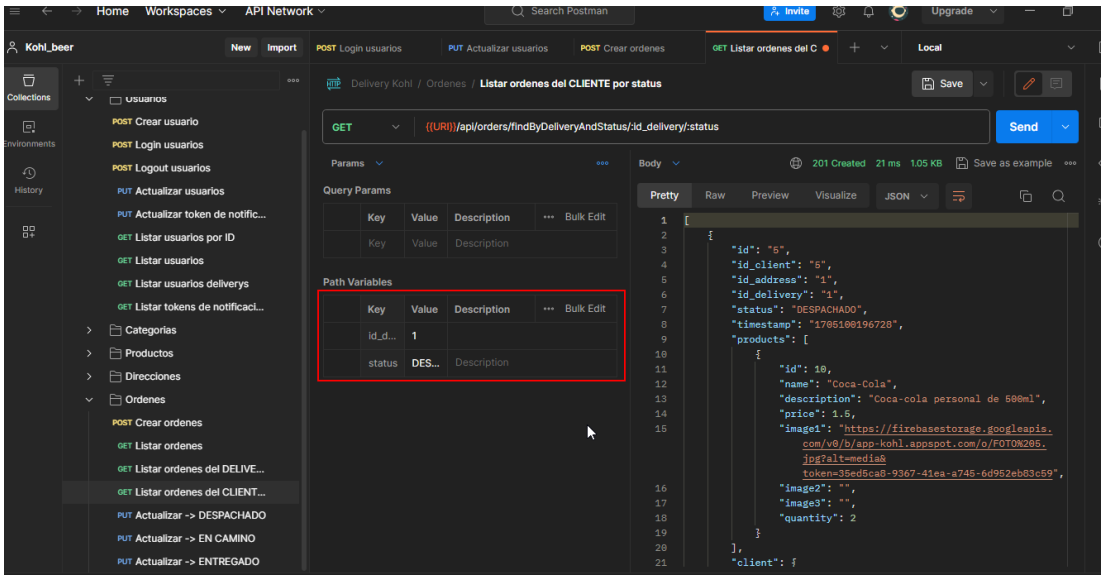


Figura 3.26: Método *GET* listar pedidos del usuario

3.5. *Sprint* 4. Modulo cliente

Para este *Sprint* se incluyen las siguientes actividades:

- *Endpoints* para gestionar (crear y listar) direcciones para entrega de pedidos
- Implementación socket.io para la ubicación en tiempo real

Endpoints para gestionar (crear y listar) direcciones para entrega de pedidos

Esta funcionalidad habilita a los clientes para añadir una dirección con parámetros clave, como la ubicación y el vecindario, mientras guarda la longitud y latitud en el mapa para asegurar una precisión más exacta en la dirección de entrega. Para gestionar tanto la creación como la visualización de una dirección, es necesario proporcionar el Token generado por JWT para autenticar al usuario, como se muestra en la **Figura 3.27**.

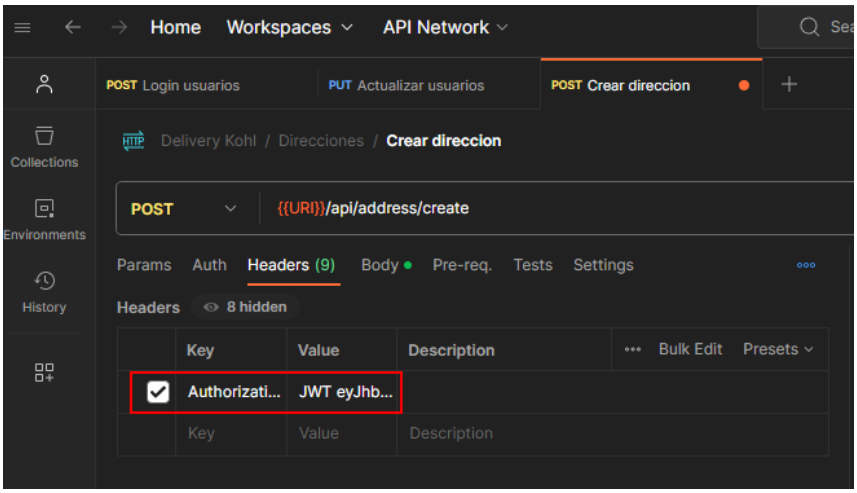


Figura 3.27: Autenticación

Para crear una dirección, se efectúa una solicitud mediante el método *POST* que se encarga de almacenar la información proporcionada. En la **Figura 3.28**, se observa que la petición es exitosa y se crea una nueva dirección para el usuario.

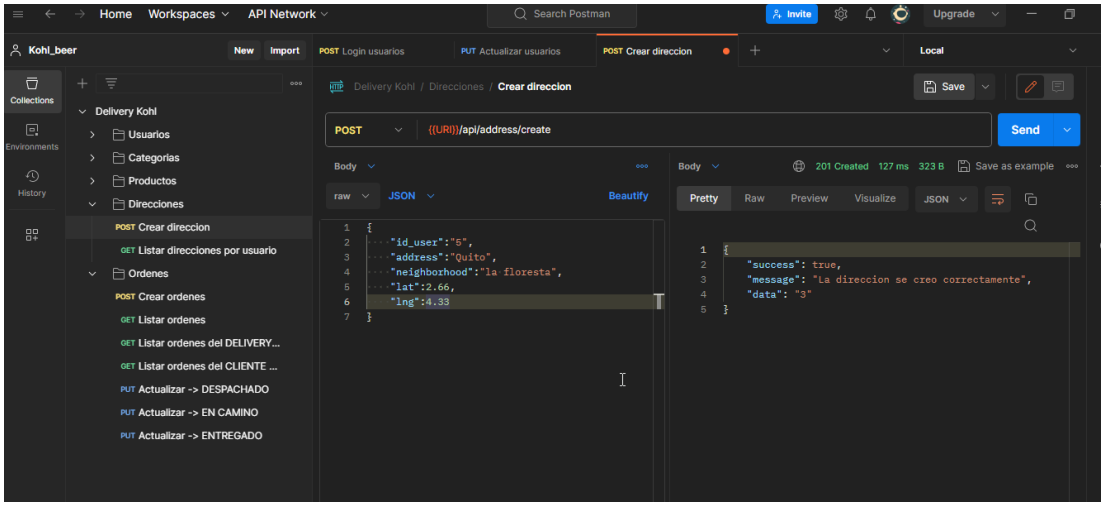


Figura 3.28: Método *POST* creación de direcciones

Para visualizar las direcciones almacenadas, se utiliza el método *GET*, que lee las direcciones almacenadas del usuario, como se muestra en la **Figura 3.29**.

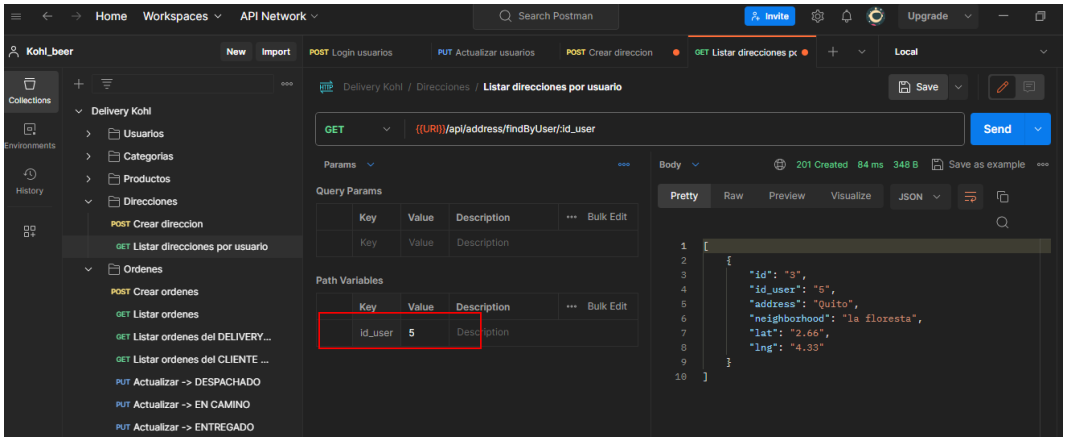


Figura 3.29: Método *GET* listar direcciones del usuario

Implementación socket.io para la ubicación en tiempo real

Esta funcionalidad se integra con la gestión de direcciones, permitiendo actualizar en tiempo real la latitud y longitud mediante la implementación de socket.io. La **Figura 3.30**, muestra la conexión con el socket encargado del almacenamiento en tiempo real.

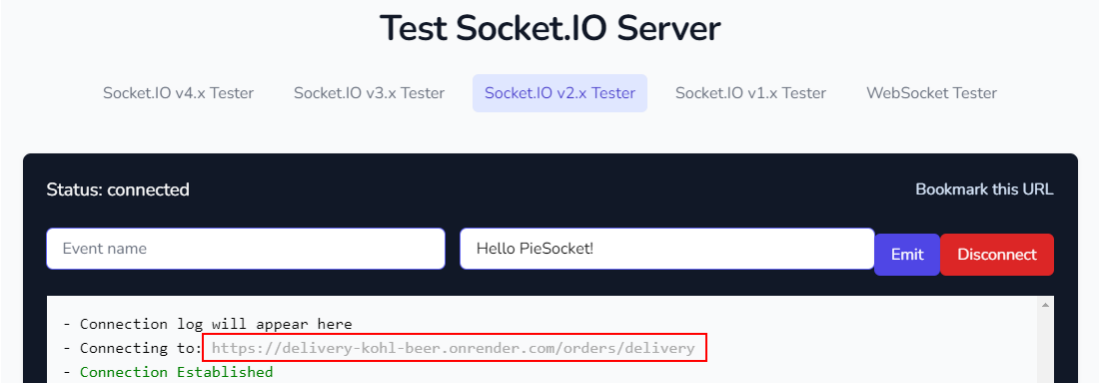


Figura 3.30: Socket.io

3.6. *Sprint* 5. Modulo *delivery*

Para este *Sprint* se incluyen las siguientes actividades:

- *Endpoint* para listar pedidos por entregar del *delivery* según *status*
- *Endpoints* para actualización del estado del pedido como (en camino y entregado)

Endpoint para listar pedidos por entregar del *delivery* según *status*

Esta funcionalidad posibilita al *delivery* listar las órdenes asignadas y comenzar la entrega en las direcciones correspondientes. Como se muestra en la **Figura 3.31**, es esencial proporcionar el token generado por JWT para autenticar al usuario y acceder a esta característica.

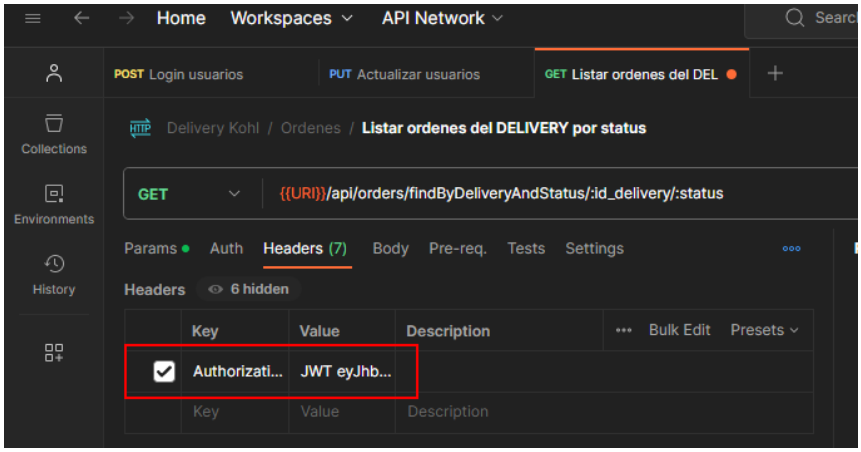


Figura 3.31: Autenticación

Para visualizar las órdenes asignadas a un *delivery*, es necesario proporcionar el ID del repartidor y especificar el estado "despachado", que indica que el repartidor puede comenzar las entregas. La solicitud se realiza mediante el método *GET*, y se puede observar en la **Figura 3.32** una respuesta positiva, ya que se muestran los pedidos asignados al repartidor.

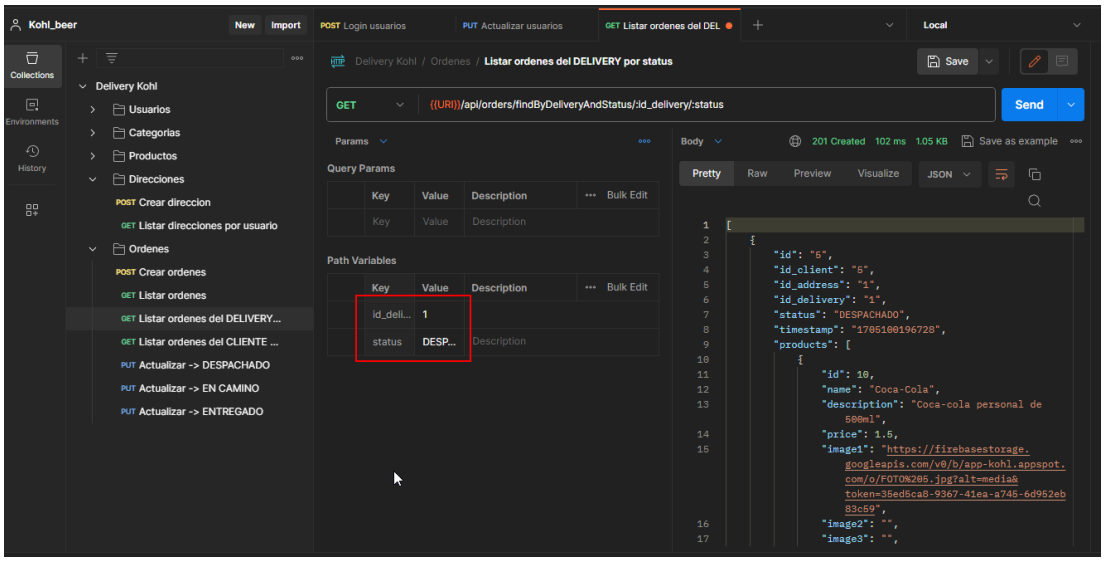


Figura 3.32: Método *GET* listar pedidos por entregar

Endpoints para actualización del estado del pedido como (en camino y entregado)

Esta funcionalidad permite al *delivery* actualizar los estados de los pedidos. Una vez que acepta el pedido, lo marca como "en camino", y al entregarlo a la persona correspondiente, lo marca como "entregado". Para llevar a cabo estas solicitudes de actualización, es necesario autenticarse utilizando el token generado por JWT, como se indica en la **Figura 3.33**.

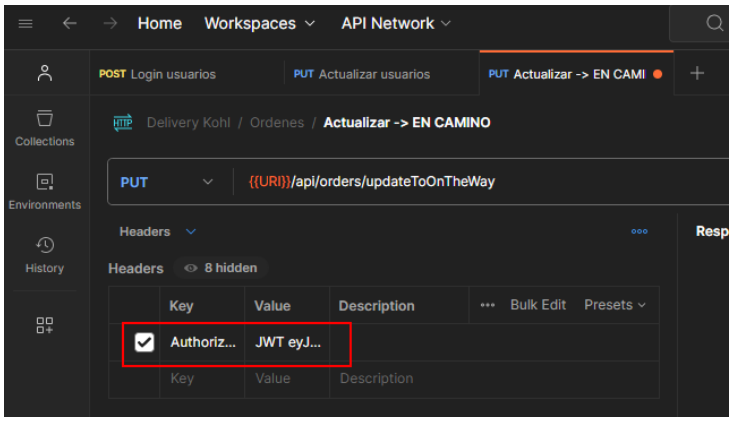


Figura 3.33: Autenticación

En este *endpoint* para actualizar el estado del pedido a "en camino", se emplea el método *PUT* para modificar los datos con el nuevo estado. En la **Figura 3.34**, se visualiza con éxito la actualización, y en la **Figura 3.35** se confirma la verificación mediante el método *GET* que lista los pedidos, mostrando el nuevo estado del pedido.

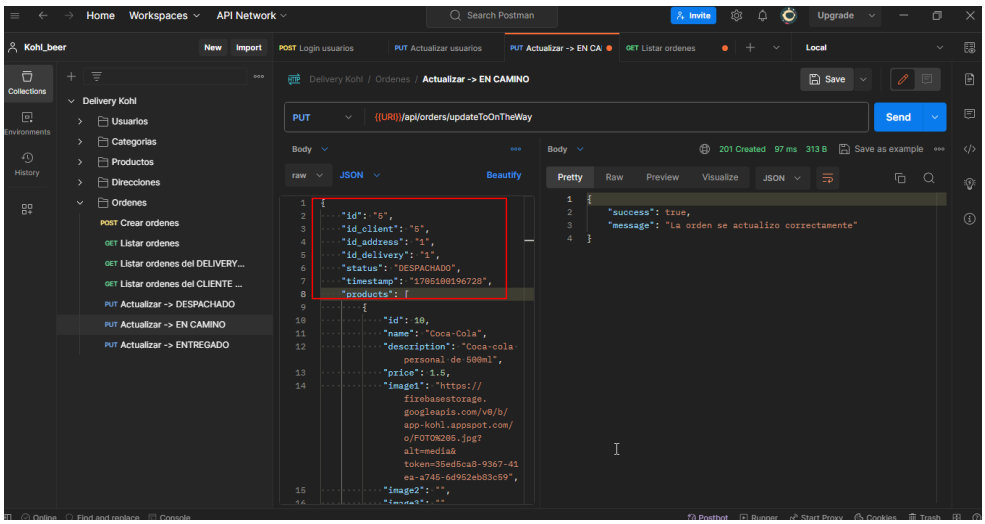


Figura 3.34: Método *PUT* actualización de estado en camino

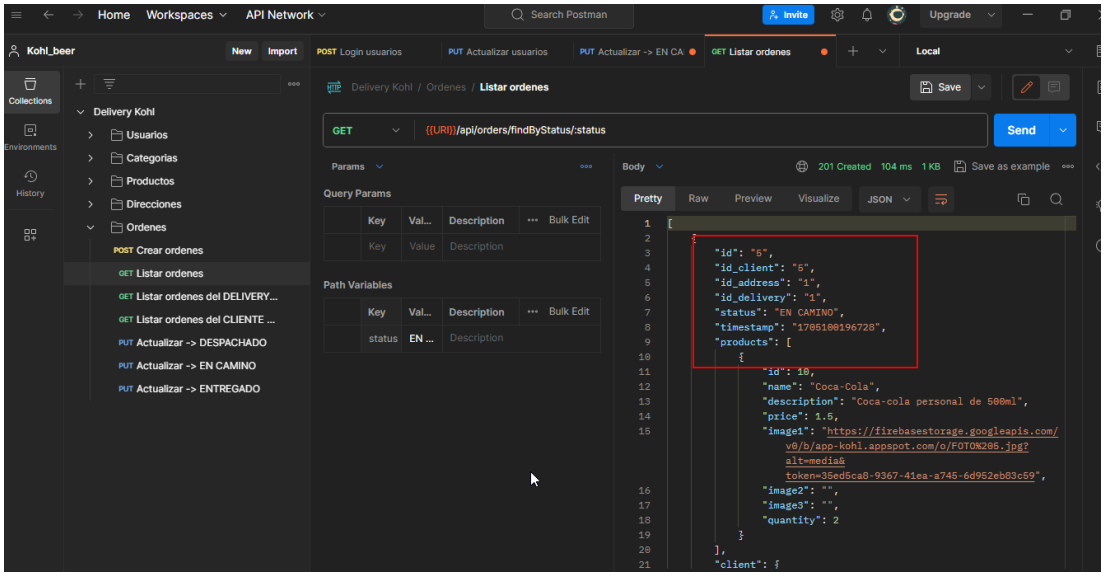


Figura 3.35: Verificación actualización de estado en camino

Para el *endpoint* que actualizar el estado del pedido a "entregado", se emplea el método *PUT* para modificar los datos con el nuevo estado. En la **Figura 3.36** se visualiza con éxito la actualización, y en la **Figura 3.37** se confirma la verificación mediante el método *GET* que lista los pedidos, mostrando el nuevo estado del pedido.

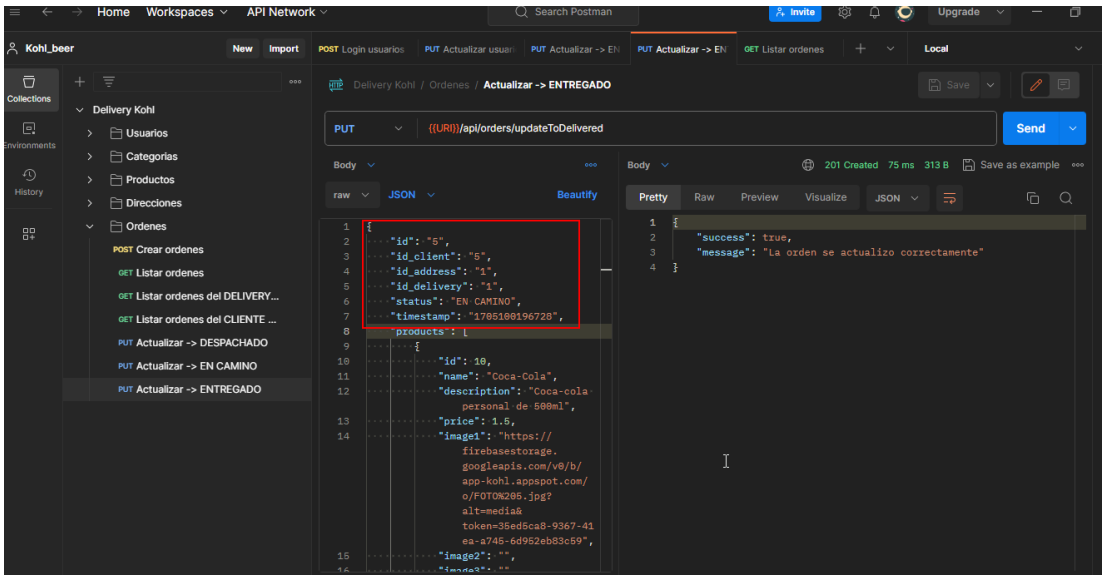


Figura 3.36: Método *PUT* actualización de estado entregado

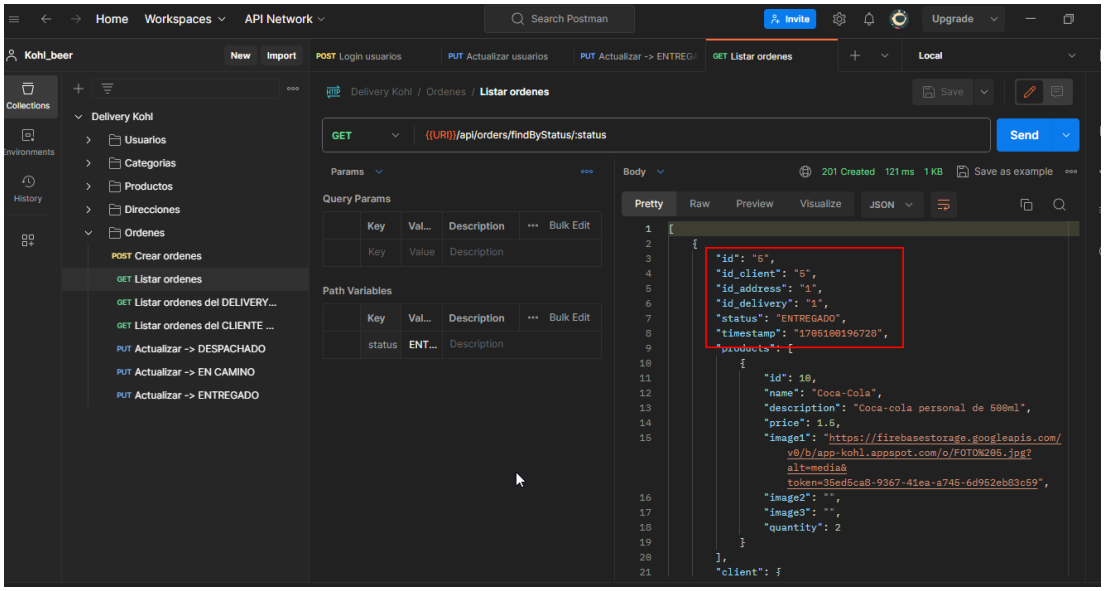


Figura 3.37: Verificación actualización de estado entregado

3.7. *Sprint* 6. Pruebas y Despliegue

Para este *Sprint* se incluyen las siguientes actividades:

- Pruebas unitarias
- Prueba de carga
- Prueba de estrés
- Despliegue Web Service en Render y StorageDB en Vercel

Pruebas unitarias

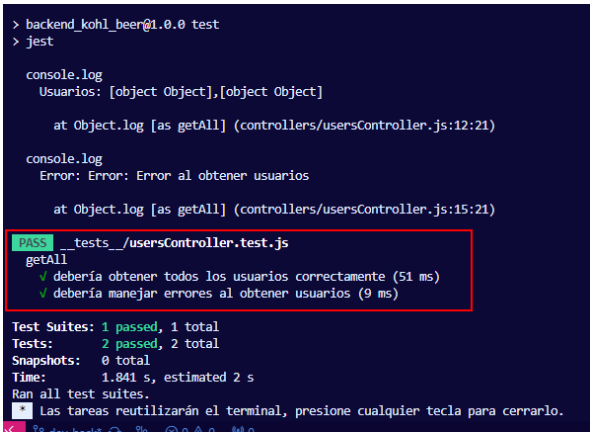
Una prueba unitaria se define como un segmento de código que evalúa la exactitud de una porción más pequeña y aislada del código de la aplicación, típicamente una función o un método. Su propósito es asegurar que este bloque de código se ejecute de acuerdo con las expectativas, conforme a la lógica prevista por el desarrollador, en este tipo de prueba, la interacción se limita a las entradas y salidas del bloque de código, validando afirmaciones capturadas de verdadero o falso [17].

Asimismo, con el propósito de verificar que cada fragmento de código funcione de manera correcta y genere los resultados anticipados, se llevan a cabo pruebas unitarias específicas para aquellos requisitos que gestionan la esencia central del negocio (CORE de negocio).

En la siguiente **Figura 3.38** se presenta la prueba unitaria correspondiente a la función "listar usuarios", mientras que la **Figura 3.39** exhibe los resultados obtenidos de dicha prueba. Para acceder al resto de las pruebas unitarias realizadas, se puede consultar en el Pruebas unitarias.

```
test('debería obtener todos los usuarios correctamente', async () => {
  const req = {};
  const res = {
    status: jest.fn().mockReturnThis(),
    json: jest.fn(),
  };
  const next = jest.fn();

  // Simula el comportamiento de la función User.getAll
  User.getAll = jest.fn().mockResolvedValue([{ id: 1, name: 'Usuario1' }, { id: 2, name: 'Usuario2' }]);

  await userController.getAll(req, res, next);

  // Verifica que las funciones y métodos esperados se hayan llamado con los argumentos correctos
  expect(res.status).toHaveBeenCalledWith(201);
  expect(res.json).toHaveBeenCalledWith([{ id: 1, name: 'Usuario1' }, { id: 2, name: 'Usuario2' }]);

  // Verifica que next no haya sido llamado (no se esperan errores)
  expect(next).not.toHaveBeenCalled();
});
```

Figura 3.38: Prueba unitaria listar usuarios



```
> backend_kohl_beer@1.0.0 test
> jest

  console.log
    Usuarios: [object Object],[object Object]

      at Object.log [as getAll] (controllers/usersController.js:12:21)

  console.log
    Error: Error: Error al obtener usuarios

      at Object.log [as getAll] (controllers/usersController.js:15:21)

 PASS  __tests__/usersController.test.js
  getAll
    √ debería obtener todos los usuarios correctamente (51 ms)
    √ debería manejar errores al obtener usuarios (9 ms)

Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   0 total
Time:        1.841 s, estimated 2 s
Ran all test suites.
 *  Las tareas reutilizarán el terminal, presione cualquier tecla para cerrarlo.
```

Figura 3.39: Resultado de la prueba unitaria listar usuarios

Prueba de carga

Las pruebas de carga evalúan el comportamiento de una aplicación ante diferentes cargas es decir usuario y tráfico en la red, en esencia permiten comprender el rendimiento de un programa con usuarios simultáneos. Identifican errores en el desarrollo que causan bajo rendimiento o fallas, permitiendo a los desarrolladores corregir problemas y garantizar un funcionamiento fluido, es importante destacar que las pruebas de carga pueden requerir un equipo profesional y experimentado para obtener resultados precisos [18].

En el *backend*, se ha empleado Apache JMeter versión 5.6.3 para realizar pruebas de carga, esta herramienta robusta genera usuarios concurrentes, cargando recursos de prueba, tanto dinámicos como estáticos, facilitando la creación de entornos de prueba realistas y la detección de posibles cuellos de botella [18]. La **Tabla 3.1** muestra la capacidad resultante para manejar peticiones HTTPS sin alcanzar sus límites de respuesta. Detalles sobre la ejecución y los resultados de la prueba de carga se encuentran en Prueba de carga del documento.

Tabla 3.1: Ejecución de la prueba de carga para 20 *endponits* del *backend*

Cantidad de peticiones	Tiempo de respuesta	Detalle
1200	00:00:26	Completamente funcional

La obtención de los resultados de las pruebas de carga evidencia que el componente *backend* responde de manera adecuada a las peticiones HTTPS para los usuarios finales. Además, no se observa ninguna falla en los tiempos de ejecución, ya que se procesa cada petición de manera correcta.

Pruebas de estrés

Para las pruebas de estrés, se ha utilizado Apache JMeter versión 5.6.3, ya que con esta herramienta es posible simular una carga que supera el pico más alto de usuarios que ha experimentado el negocio. El propósito de estas pruebas es evaluar los límites y la capacidad de la aplicación para gestionar condiciones de alto estrés, como un elevado número de usuarios concurrentes, un volumen significativo de datos o transacciones.

La cantidad con la que se puede procesar peticiones HTTPS, estando sujeto a los límites de respuesta, es detallada en la **Tabla 3.2**. La explicación de la ejecución y los resultados de la prueba de estrés están disponibles en Prueba de estrés de este documento.

Tabla 3.2: Ejecución de la prueba de estrés para 20 *endponits* del *backend*

Cantidad de peticiones	Tiempo de respuesta	Detalle
1600	00:00:44	Completamente funcional

Los resultados de las pruebas de estrés revelan que al añadir un incremento de 20 usuarios más durante un pico alto, el componente *backend* muestra adaptabilidad y ejecución sin ninguna falla.

Despliegue Web Service en Render y StorageDB en Vercel

Una fase fundamenta en el desarrollo de software es el despliegue, esto debido a que se facilita el consumo exitoso del componente *backend* por parte de la aplicación móvil. En la presente tarea se ha completado la implementación del servicio web en Render y el almacenamiento en la nube para PostgreSQL en Vercel. La **Figura 3.40** muestra el despliegue del servicio web, conectado a la rama principal del repositorio en GitHub, mientras que la **Figura 3.41** ilustra el uso de Vercel para el almacenamiento de la base de datos.

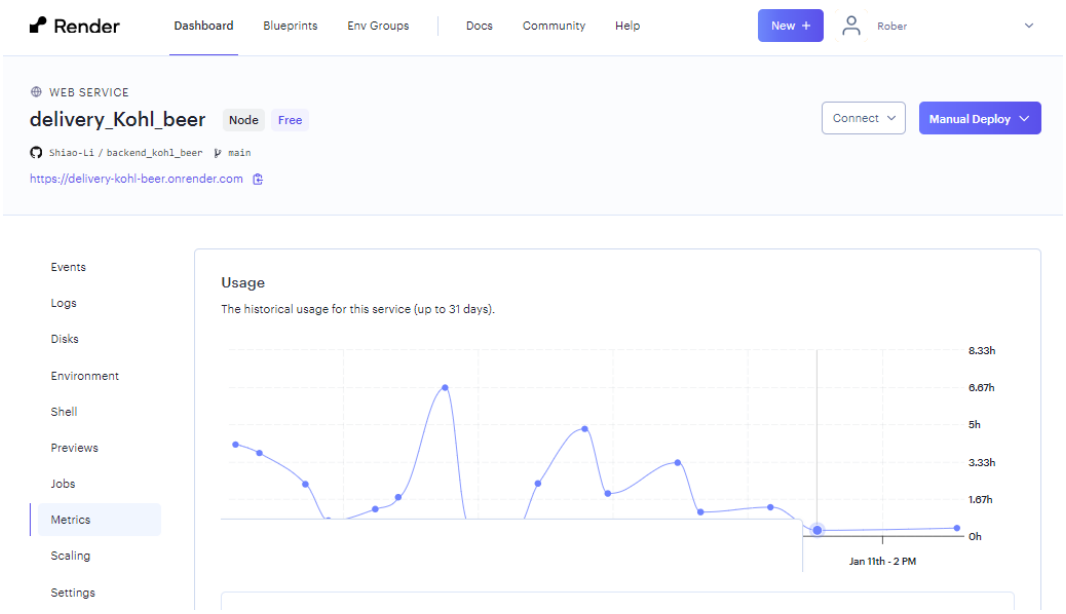


Figura 3.40: Despliegue del Web Service en Render

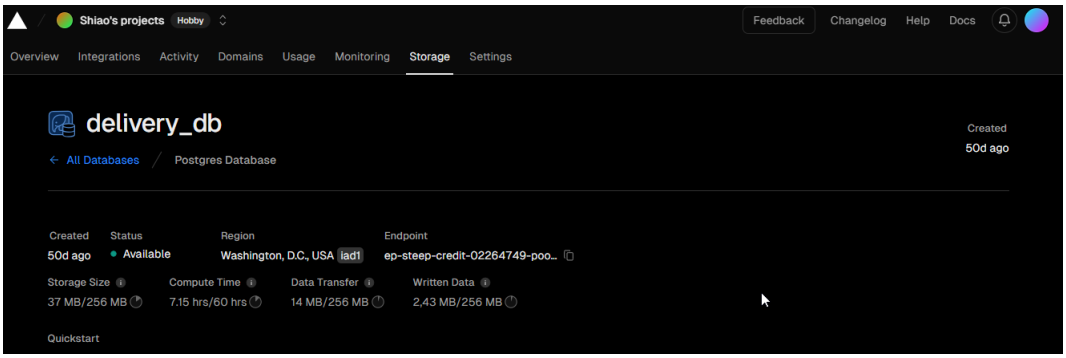


Figura 3.41: Base de datos PostgreSQL en Vercel

4. CONCLUSIONES

- La implementación de metodologías ágiles, específicamente *Scrum*, se reveló como esencial al proporcionar una planificación flexible y una capacidad de respuesta rápida a las cambiantes dinámicas del proyecto, así como al tener claro cada uno de los requerimientos, conduciendo a mejoras en la eficiencia del desarrollo y a un mayor nivel de satisfacción por parte del administrador del RestoBar (*Product Owner*) y el *Scrum Master*.

- La eficiente gestión de la información se logró a través de la aplicación de la arquitectura de datos y el diseño del patrón MVC, esta estrategia garantizó una separación transparente entre la lógica del negocio, la presentación y el control, facilitando significativamente el mantenimiento a futuro del sistema.

- La elección de Noje.js como plataforma de desarrollo para el *backend* demostró ser fundamentar para lograr un rendimiento eficiente y una escalabilidad óptima.

- La exitosa integración y uso de JWT en la autenticación de usuarios refuerza la seguridad del sistema, proporcionando un mecanismo efectivo para gestionar la autenticación y validar el acceso a las funcionalidades, garantizando la integridad de la información transmitida.

- Optar por Vercel para el alojamiento de la base de datos y Render para el despliegue resulta esencial, asegurando eficiencia y confiabilidad. La integración fluida de Vercel con servicios de almacenamiento y la escalabilidad de Render optimizan recursos, brindando accesibilidad y una experiencia de usuario constante.

- Las pruebas extensivas realizadas mediante Jest para pruebas unitarias y Apache JMeter para pruebas de carga y estrés son esenciales ya que validan y aseguran el rendimiento sólido y la funcionalidad ininterrumpida del sistema.

5. RECOMENDACIONES

- Se sugiere tratar las credenciales proporcionadas en el documento para las pruebas, como información confidencial y restringida, mantener la privacidad de estas credenciales es crucial para preservar la integridad y proteger contra posibles amenazas de seguridad.
- Se sugiere explorar e implementar una pasarela de pagos más adecuada para el contexto ecuatoriano, considerando que PayPal puede no ser utilizado por todos los usuarios.
- Se aconseja que cualquier nueva implementación o asignación de roles sea coordinada y discutida con el equipo de desarrollo.

6. REFERENCIAS BIBLIOGRÁFICAS

[1] «Facebook,» 15 Noviembre 2023. [En línea]. Available: https://www.facebook.com/kohl.beer.ec. [Último acceso: 15 Noviembre 2023].

[2] S. Ortiz, «El Comerio,» 05 Mayo 2021. [En línea]. Available: https://www.elcomercio.com/actualidad/negocios/confinamiento-impulsa-delivery-ecuador-pandemia.html. [Último acceso: 15 Noviembre 2023].

[3] «Primicias,» 31 Julio 2023. [En línea]. Available: https://www.primicias.ec/noticias/economia/apps-delivery-comida-descargas/. [Último acceso: 15 Noviembre 2023].

[4] M. Sánchez, «Linkedin,» 06 Enero 2022. [En línea]. Available: https://www.linkedin.com/pulse/negocios-en-la-era-digital-maibel-sánchez/?originalSubdomain=es. [Último acceso: 15 Noviembre 2023].

[5] K. Arizbé, «Gluo,» 03 Octubre 2023. [En línea]. Available: https://www.gluo.mx/blog/backend-que-es-y-para-que-sirve. [Último acceso: 15 Noviembre 2023].

[6] D. Borovskoy, «Linkedin,» 27 Septiembre 2023. [En línea]. Available: https://www.linkedin.com/pulse/que-es-backend-características-2023-denis-borovskoy/?originalSubdomain=es. [Último acceso: 15 Noviembre 2023].

[7] «Oracle,» 2023. [En línea]. [Último acceso: 15 Noviembre 2023].

[8] «Nodejs,» 2023. [En línea]. Available: https://nodejs.org/en/docs/guides/getting-started-guide. [Último acceso: 2023 Noviembre 15].

[9] «Geeksforgeeks,» 08 Julio 2022. [En línea]. Available: https://www.geeksforgeeks.org/introduction-postman-api-development/. [Último acceso: 2023 Noviembre 15].

[10] L. Codina, «Estudios de caso: características, tipología y bibliografía comentada,» 19 junio 2023. [En línea]. Available: https://www.lluiscodina.com/estudios-de-caso/. [Último acceso: 01 enero 2024].

[11] Trycore, «Trycore,» 18 octubre 2018. [En línea]. Available: https://trycore.co/buenas-practicas-ti/importancia-de-metodologias-agiles/. [Último acceso: 01 enero 2024].

[12] integrait, «Conoce los principales roles de Scrum y sus responsabilidades,» 28 abril 2019. [En línea]. Available: https://integrait.com.mx/blog/roles-de-scrum/. [Último acceso: 11 enero 2024].

[13] atlassian, «atlassian,» [En línea]. Available: https://www.atlassian.com/es/agile/project-management/user-stories. [Último acceso: 11 enero 2024].

[14] E. P. Camús, «Product Backlog y Sprint Backlog. Claves del Scrum,» 20 febrero 2023. [En línea]. Available: https://bloo.media/blog/product-backlog-sprint-backlog/. [Último acceso: 11 enero 2024].

[15] Ibm.com, «Ibm.com,» [En línea]. Available: https://www.ibm.com/es-es/topics/data-architecture. [Último acceso: 11 enero 2024].

[16] MDN Web Docs, «MDN Web Docs,» [En línea]. Available: https://developer.mozilla.org/es/docs/Glossary/MVC. [Último acceso: 11 enero 2024].

[17] Amazon.com, «Amazon.com,» [En línea]. Available: https://aws.amazon.com/es/what-is/unit-testing/#:~:text=Una%20prueba%20unitaria%20es%20un,la%20l%C3%B3gica%20te%C3%B3rica%20del%20desarrollador.. [Último acceso: 12 enero 2024].

[18] G. Lee, «LoadView,» LoadView by Dotcom-Monitor, 11 noviembre 2020. [En línea]. Available: https://www.loadview-testing.com/es/blog/5-ejemplos-de-pruebas-de-carga-de-jmeter/. [Último acceso: 13 enero 2024].

7. ANEXOS

Los anexos comprenden toda la información pertinente que expone de manera clara el progreso de este proyecto de integración curricular.

- **ANEXO I.** Certificado de Originalidad
- **ANEXO II.** Manual Técnico
- **ANEXO III.** Manual de Usuario
- **ANEXO IV.** Manual de Instalación

ANEXO I

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

CERTIFICADO DE ORIGINALIDAD

Quito, D.M. 15 de febrero de 2024

De mi consideración:

Yo, IVONNE FERNANDA MALDONADO SOLIZ, en calidad de Director del Trabajo de Integración Curricular titulado DESARROLLO DE UN BACKEND asociado al SISTEMA DE GESTIÓN Y DELIVERY PARA RESTOBAR DE CERVEZA ARTESANAL KÖHL BEER elaborado por el estudiante ROBERTO LI FONG SHIAO MORALES 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 secciones: Descripción del componente desarrollado, Metodología, Resultados, Conclusiones y Recomendaciones (sin anexos), producto del Trabajo de Integración Curricular indicado.

El documento escrito tiene un índice de similitud del 10%.

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,

Ivonne Maldonado
Docente Ocasional a Tiempo Completo
ESFOT

ANEXO II

En este apartado se encuentra la información que respalda la implementación del presente proyecto.

Levantamiento de requerimientos

En la **Tabla 1**, se observa la Recopilación de Requerimientos obtenida para el desarrollo del *backend*.

Tabla 1: Levantamiento de requerimiento

Recopilación de requerimientos		
Tipo de sistema	**ID-RR**	**Enunciado del Ítem**
Backend	**RR001**	Independientemente del rol asignado (cliente, *delivery* o administrador), el usuario necesita generar *endpoints* para: • Registrarse.
	RR002	El usuario con rol cliente, *delivery* o administrador necesita generar *endpoints* para: • Iniciar sesión y cerrar sesión.
	RR003	El usuario con rol cliente necesita generar *endpoints* para: • Crear y listar pedidos por estados.
	RR004	El usuario con rol cliente necesita generar *endpoints* para: • Visualizar detalle de productos por categorías.
	RR005	El usuario con rol cliente necesita generar *endpoints* para: • Ver categorías de los productos
	RR006	El usuario con rol cliente necesita generar *endpoints* para: • Crear, visualizar direcciones de entrega de pedidos.
	RR007	El usuario con rol cliente necesita generar *endpoints* para: • Editar perfil personal.
	RR008	El usuario con rol *delivery* necesita generar *endpoints* para: • Listar por estados.

	RR009	El usuario con rol *delivery* necesita generar *endpoints* para: • Actualizar el pedido como “En camino”
	RR010	El usuario con rol *delivery* necesita generar *endpoints* para: • Actualizar el pedido como “Entregado”
	RR011	El usuario con rol *delivery* necesita generar *endpoints* para: • Ver ubicación de la entrega.
	RR012	El usuario con rol administrador necesita generar *endpoints* para: • Gestionar (crear y listar) categorías de productos.
	RR013	El usuario con rol administrador necesita generar *endpoints* para: • Gestionar (crear y listar) productos.
	RR014	El usuario con rol cliente, delivery y administrador necesita generar *endpoints* para: • Listar por estados.
	RR015	El usuario con rol administrador necesita generar *endpoints* para: • Asignar un *delivery* al pedido.
	RR016	El usuario con rol administrador necesita generar *endpoints* para: • Actualizar el pedido como “Despachado”

Historias de Usuario

Después de finalizar la fase de recolección de requerimiento, se inicia la creación de las historias de Usuario para el *backend*. A continuación, se muestran las 15 historias de usuario que se han generado a partir de los requerimientos recopilados anteriormente, las cuales van desde la **Tabla 2** hasta la **Tabla 9**.

Tabla 2: Historia de usuario 001 – Registro.

HISTORIA DE USUARIO	
Identificador: HU001	**Usuario:** cliente, *delivery* y administrador.
Nombre historia: Registrar nueva cuenta de usuario	
Prioridad en Negocio: Alta	**Riesgo en Negocio:** Alta
Iteración asignada: 1	
Responsable: Roberto Shiao	

Descripción: Independientemente del rol asignado (cliente, *delivery* o administrador) necesita generar un *endpoint* que permitan registrarse a nuevos usuarios a la aplicación móvil. • Correo electrónico • Nombre • Apellido • Teléfono • Contraseña
Observación: El correo y teléfono serán datos únicos para el registro de nuevos usuarios. En el caso de los repartidores y administradores, es el equipo de desarrollo quien asigna este rol una vez que el cliente “repartidor” o “administrador” se ha registrado.

Tabla 3: Historia de usuario 003 – Gestión de pedidos.

HISTORIA DE USUARIO	
Identificador: HU003	**Usuario:** cliente
Nombre historia: Gestión de pedidos	
Prioridad en Negocio: Media	**Riesgo en Negocio:** Media
Iteración asignada: 1	
Responsable: Roberto Shiao	
Descripción: Como rol cliente necesita generar varios *endpoints* que permitan crear y listar los pedidos realizados. • Visualizar y crear pedidos	
Observación: Ninguna.	

Tabla 4: Historia de usuario 004 – Visualizar productos.

HISTORIA DE USUARIO	
Identificador: HU004	**Usuario:** cliente
Nombre historia: Visualizar productos	
Prioridad en Negocio: Media	**Riesgo en Negocio:** Media
Iteración asignada: 1	
Responsable: Roberto Shiao	
Descripción: Como rol cliente necesita generar varios *endpoints* que permitan visualizar los productos. • Visualizar detalle de productos por categorías	
Observación: El cliente podrá visualizar los productos según su categoría.	

Tabla 5: Historia de usuario 005 – Categorías de productos.

HISTORIA DE USUARIO	
Identificador: HU005	**Usuario:** cliente

Nombre historia: Categorías de productos	
Prioridad en Negocio: Media	**Riesgo en Negocio:** Media
Iteración asignada: 1	
Responsable: Roberto Shiao	
Descripción: Como rol cliente necesita generar varios *endpoints* que permitan ver las categorías de los productos. • Ver categorías de los productos	
Observación: Ninguna.	

Tabla 6: Historia de usuario 006 – Gestión de entrega.

HISTORIA DE USUARIO	
Identificador: HU006	**Usuario:** cliente
Nombre historia: Gestión de ubicaciones	
Prioridad en Negocio: Media	**Riesgo en Negocio:** Alta
Iteración asignada: 2	
Responsable: Roberto Shiao	
Descripción: Como rol cliente necesita generar varios *endpoints* que permitan agregar o seleccionar entre algunas una dirección de entrega del pedido. • Crear y ver direcciones de entrega.	
Observación: Ninguna.	

Tabla 7: Historia de usuario 007 – Editar perfil personal.

HISTORIA DE USUARIO	
Identificador: HU007	**Usuario:** Cliente
Nombre historia: Editar perfil personal	
Prioridad en Negocio: Baja	**Riesgo en Negocio:** Baja
Iteración asignada: 2	
Responsable: Roberto Shiao	
Descripción: Como rol cliente necesita generar varios *endpoints* que permita editar la información personal del perfil del cliente. • Correo electrónico • Nombre • Apellido • Teléfono	
Observación: El administrador, al tener el rol de cliente y el *delivery*, también con el rol de cliente, pueden editar sus perfiles desde dicha función del cliente.	

Tabla 8: Historia de usuario 008 – Listado de pedidos.

HISTORIA DE USUARIO	
Identificador: HU008	**Usuario:** *Delivery*
Nombre historia: Listar pedidos	
Prioridad en Negocio: Alta	**Riesgo en Negocio:** Alta
Iteración asignada: 2	
Responsable: Roberto Shiao	
Descripción: Como rol *delivery* necesita generar varios *endpoints* que permita listar por estados los pedidos por entregar, estados: • Lista de pedidos despachados • Lista de pedidos en camino • Lista de pedidos entregados	
Observación: Ninguna.	

Tabla 9: Historia de usuario 009 – Actualizar estado para entregar.

HISTORIA DE USUARIO	
Identificador: HU009	**Usuario:** *Delivery*
Nombre historia: Actualizar estado para entregar	
Prioridad en Negocio: Alta	**Riesgo en Negocio:** Alta
Iteración asignada: 2	
Responsable: Roberto Shiao	
Descripción: Como rol *delivery* necesita generar varios *endpoints* que permita al *delivery* actualizar el pedido como “En camino” para iniciar con la entrega del pedido. • Actualizar el pedido como “En camino”	
Observación: Ninguna.	

Tabla 3: Historia de usuario 010 – Actualizar estado del pedido entregado.

HISTORIA DE USUARIO	
Identificador: HU10	**Usuario:** *Delivery*
Nombre historia: Actualizar estado del pedido entregado	
Prioridad en Negocio: Media	**Riesgo en Negocio:** Media
Iteración asignada: 3	
Responsable: Roberto Shiao	
Descripción: Como rol *delivery* necesita generar varios *endpoints* que permita al *delivery* actualizar el pedido como “Entregado” una vez entregado al cliente el pedido: • Actualizar el pedido como “Entregado”	

Observación: Ninguno.

Tabla 4: Historia de usuario 011 – Ver ubicación de la entrega.

HISTORIA DE USUARIO	
Identificador: HU011	**Usuario:** *Delivery*
Nombre historia: Ver ubicación de la entrega	
Prioridad en Negocio: Media	**Riesgo en Negocio:** Media
Iteración asignada: 3	
Responsable: Roberto Shiao	
Descripción: Como rol *delivery* necesita generar varios *endpoints* que permita al *delivery* ver la ubicación de entrega agregada por el cliente.	
Observación: Obtención de la ubicación por medio de Google maps y almacenamiento de la ubicación en tiempo real con socket.io.	

Tabla 5: Historia de usuario 012 – Gestión categorías.

HISTORIA DE USUARIO	
Identificador: HU012	**Usuario:** Administrador
Nombre historia: Gestión categorías	
Prioridad en Negocio: Baja	**Riesgo en Negocio:** Media
Iteración asignada: 3	
Responsable: Roberto Shiao	
Descripción: Como rol Administrador necesita generar varios *endpoints* que permita gestionar las categorías de los productos. • Añadir • Visualizar	
Observación: Se podrá añadir una breve descripción de la categoría añadida.	

Tabla 6: Historia de usuario 013 – Gestión de productos.

HISTORIA DE USUARIO	
Identificador: HU013	**Usuario:** Administrador
Nombre historia: Gestión de productos	
Prioridad en Negocio: Alta	**Riesgo en Negocio:** Alta
Iteración asignada: 4	
Responsable: Roberto Shiao	
Descripción: Como rol Administrador necesita generar varios *endpoints* que permita gestionar los productos.	

• Añadir • Visualizar
Observación: Al crear el producto se podrá elegir 3 fotos y la categoría al que pertenece.

Tabla 7: Historia de usuario 014 – Listar pedidos.

HISTORIA DE USUARIO	
Identificador: HU014	**Usuario:** Administrador y cliente
Nombre historia: Listar pedidos	
Prioridad en Negocio: Media	**Riesgo en Negocio:** Media
Iteración asignada: 4	
Responsable: Roberto Shiao	
Descripción: Como rol Administrador necesita generar varios *endpoints* que permita listar por estados los pedidos por entregar, estados: • Lista de pedidos pagados • Lista de pedidos despachados • Lista de pedidos en camino • Lista de pedidos entregados	
Observación: Ninguna.	

Tabla 8: Historia de usuario 015 – Asignación de *deliverys*.

HISTORIA DE USUARIO	
Identificador: HU015	**Usuario:** Administrador
Nombre historia: Asignar *deliverys*	
Prioridad en Negocio: Alta	**Riesgo en Negocio:** Alta
Iteración asignada: 4	
Responsable: Roberto Shiao	
Descripción: Como rol Administrador necesita generar varios *endpoints* que permita asignar un *delivery* listando todos los usuarios que posean el rol de *delivery*.	
Observación: Ninguna.	

Tabla 9: Historia de usuario 016 – Actualizar estado con *delivery* asignado.

HISTORIA DE USUARIO	
Identificador: HU016	**Usuario:** Administrador
Nombre historia: Actualizar estado con *delivery* asignado	
Prioridad en Negocio: Alta	**Riesgo en Negocio:** Alta
Iteración asignada: 4	
Responsable: Roberto Shiao	

Descripción: Como rol administrador necesita generar varios *endpoints* que permita al administrador actualizar el pedido como “Despachado” asignando ya el *delivery*. • Actualizar el pedido como “Despachado”
Observación: Ninguna.

Product Backlog

La **Tabla 10** clasifica los requisitos mediante un orden establecido según las necesidades del *Product Owner* y la complejidad asociada a cada uno de ellos.

Tabla 10: *Product Backlog.*

ELABORACIÓN DEL *PRODUCT BACKLOG*				
ID – HU	**HISTORIA DE USUARIO**	**ITERACIÓN**	**ESTADO**	**PRIORIDAD**
HU001	Generar un *endpoints* para el registrar nueva cuenta de usuario.	1	Finalizado	Alta
HU002	Generar varios *endpoints* para iniciar sesión, cerrar sesión.	1	Finalizado	Alta
HU003	Generar varios *endpoints* para la Gestión de pedidos.	1	Finalizado	Media
HU004	Generar un *endpoints* para visualizar productos	1	Finalizado	Media
HU005	Generar varios *endpoints* para visualizar las categorías.	1	Finalizado	Media
HU006	Generar un *endpoint* para gestión de ubicaciones.	2	Finalizado	Alta
HU007	Generar un *endpoint* para editar perfil personal.	2	Finalizado	Media
HU008	Generar un *endpoint* para listado de pedidos del *delivery*.	2	Finalizado	Alta
HU009	Generar un *endpoint* para que el *delivery* pueda actualizar el estado del pedido para entregar.	2	Finalizado	Alta
HU010	Generar un *endpoint* para que el *delivery* pueda actualizar el estado del pedido entregado.	3	Finalizado	Media
HU011	Generar un *endpoint* para ver	3	Finalizado	Media

	ubicación de la entrega.			
HU012	Generar varios *endpoints* para gestionar categorías.	3	Finalizado	Baja
HU013	Generar varios *endpoints* para gestionar productos.	4	Finalizado	Alta
HU014	Generar un *endpoint* para listado de pedidos del cliente y del administrador.	4	Finalizado	Media
HU015	Generar un *endpoints* para la asignación de *deliverys.*	4	Finalizado	Alta
HU015	Generar un *endpoint* para que el *delivery* pueda actualizar estado con *delivery* asignado.	4	Finalizado	Alta

Sprint Backlog

En la **Tabla 11** muestra los 6 *Sprints* planeados para llevar a cabo el desarrollo del *backend*, detallando las tareas y el tiempo estimado necesario para cada una, con el objetivo de alcanzar los objetivos establecidos por el *Product Owner* en términos de los entregables requeridos.

Tabla 11: *Sprint Backlog*

ID-SB	NOMBRE	HISTORIA DE USUARIO	TAREAS	TIEMPO ESTIMADO
S0	Configuración del ambiente del desarrollo	N/A	• Definición de roles. • Recopilación de requerimientos • Configuración de herramientas necesarias para el desarrollo. • Diseño de la Base de Datos en PostgresQL.	20 h
S1	Autenticación y registro	HUB001: Registrar cuenta de usuario. HUB002: Iniciar sesión, cerrar sesión. HUB007: Editar perfil personal del cliente.	• Crear *endpoint* para el registro de usuario. • Crear *endpoint* para el inicio de sesión. • Crear *endpoint* para cerrar sesión. • *Crear endpoint* para editar perfil del usuario.	20 h
S2	Modulo Administrador	HUB012: Gestión categorías HUB013: Gestión de productos	• Crear *endpoints* para que se puedan gestionar (añadir y visualizar) las categorías.	50 h

<table>
<tr><td rowspan="3"></td><td rowspan="3"></td><td>HUB014: Listar pedidos</td><td rowspan="3">Crear endpoints para que se puedan gestionar (añadir y visualizar) los productos.
Crear endpoint que permita ver los pedidos por estados (pagados, despachados, en camino y entregados).
Crear endpoint para que se permita asignar un delivery al tener ya un pedido pagado.
Crear endpoint para actualizar el estado del pedido como despachado.</td><td rowspan="3"></td></tr>
<tr><td>HUB015: Asignar deliverys</td></tr>
<tr><td>HUB016: Actualizar estado con delivery asignado</td></tr>
<tr><td rowspan="2">S3</td><td rowspan="4">Modulo cliente</td><td>HUB003: Gestión de pedidos</td><td rowspan="4">Crear endpoint para crear pedidos.
Crear endpoint listar órdenes del cliente por status.
Crear endpoints para que pueda visualizar las categorías.
Crear endpoints para que pueda visualizar los productos.
Crear endpoints para gestionar (crear y listar) direcciones para entrega de pedidos
Implementación socket.io para la ubicación en tiempo real.</td><td rowspan="2">30h</td></tr>
<tr><td rowspan="3">HUB004: Visualizar productos</td></tr>
<tr><td rowspan="2">S4</td><td rowspan="2">35h</td></tr>
<tr><td>HUB005: Categorías de productos</td></tr>
<tr><td>S5</td><td>Modulo Delivery</td><td>HUB006: Gestión de ubicaciones</td><td>HUB008: Listar de pedidos</td><td>Crear endpoint listar pedidos por entregar del delivery según su status.</td><td>60 h</td></tr>
</table>

		HUB009: Actualizar estado para entregar	• Crear *endpoints* para actualización de del estado del pedido como (en camino y entregado). • Crear *endpoints* para listar direcciones para entrega de pedidos.	
		HUB0010: Actualizar estado para entregar		
		HUB011: Ver ubicación de la entrega		
S6	Pruebas y Despliegue	N/A	• Pruebas unitarias. • Prueba de carga. • Prueba de performance. • Despliegue Web service en Render y StorageDB en Vercel	5 h
Documentación		N/A	• Trabajo de Integración Curricular. • Anexos.	20 h
			Total =	240 h

Pruebas

A continuación, se describen las diversas pruebas llevadas a cabo, enfocadas en las funcionalidades fundamentales (CORE) del negocio.

Pruebas unitarias

La representación gráfica ilustra la ejecución de los puntos de conexión desarrollados desde la **Figura 1** hasta la **Figura 25**, mostrando las pruebas realizadas en las funciones de los controladores junto con sus resultados correspondientes.

```
test('debería obtener todos los usuarios correctamente', async () => {
  const req = {};
  const res = {
    status: jest.fn().mockReturnThis(),
    json: jest.fn(),
  };
  const next = jest.fn();

  // Simula el comportamiento de la función User.getAll
  User.getAll = jest.fn().mockResolvedValue([{ id: 1, name: 'Usuario1' }, { id: 2, name: 'Usuario2' }]);

  await userController.getAll(req, res, next);

  // Verifica que las funciones y métodos esperados se hayan llamado con los argumentos correctos
  expect(res.status).toHaveBeenCalledWith(201);
  expect(res.json).toHaveBeenCalledWith([{ id: 1, name: 'Usuario1' }, { id: 2, name: 'Usuario2' }]);

  // Verifica que next no haya sido llamado (no se esperan errores)
  expect(next).not.toHaveBeenCalled();
});
```

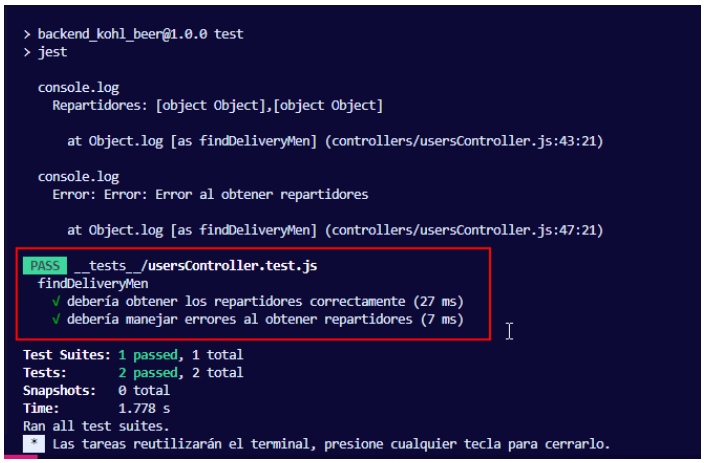
Figura 1: Prueba unitaria #1 Listar usuarios

```
> backend_kohl_beer@1.0.0 test
> jest

  console.log
    Usuarios: [object Object],[object Object]

      at Object.log [as getAll] (controllers/usersController.js:12:21)

  console.log
    Error: Error: Error al obtener usuarios

      at Object.log [as getAll] (controllers/usersController.js:15:21)

 PASS  __tests__/usersController.test.js
  getAll
    √ debería obtener todos los usuarios correctamente (51 ms)
    √ debería manejar errores al obtener usuarios (9 ms)

Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   0 total
Time:        1.841 s, estimated 2 s
Ran all test suites.
 *  Las tareas reutilizarán el terminal, presione cualquier tecla para cerrarlo.
```

Figura 2: Resultado prueba unitaria #1

```
test('debería obtener los repartidores correctamente', async () => {
  const req = {};
  const res = {
    status: jest.fn().mockReturnThis(),
    json: jest.fn(),
  };
  const next = jest.fn();

  // Simula el comportamiento de la función User.findDeliveryMen
  User.findDeliveryMen = jest.fn().mockResolvedValue([
    { id: 1, name: 'Repartidor1' },
    { id: 2, name: 'Repartidor2' },
  ]);

  await userController.findDeliveryMen(req, res, next);

  // Verifica que las funciones y métodos esperados se hayan llamado con los argumentos correctos
  expect(res.status).toHaveBeenCalledWith(201);
  expect(res.json).toHaveBeenCalledWith([
    { id: 1, name: 'Repartidor1' },
    { id: 2, name: 'Repartidor2' },
  ]);

  // Verifica que next no haya sido llamado (no se esperan errores)
  expect(next).not.toHaveBeenCalled();
});
```

Figura 3: Prueba unitaria #2 Listar *deliverys*



```
> backend_kohl_beer@1.0.0 test
> jest

  console.log
    Repartidores: [object Object],[object Object]

      at Object.log [as findDeliveryMen] (controllers/usersController.js:43:21)

  console.log
    Error: Error: Error al obtener repartidores

      at Object.log [as findDeliveryMen] (controllers/usersController.js:47:21)

 PASS  __tests__/usersController.test.js
  findDeliveryMen
    √ debería obtener los repartidores correctamente (27 ms)
    √ debería manejar errores al obtener repartidores (7 ms)

Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   0 total
Time:        1.778 s
Ran all test suites.
 *  Las tareas reutilizarán el terminal, presione cualquier tecla para cerrarlo.
```

Figura 4: Resultado prueba unitaria #2

```
test('debería registrar un usuario correctamente', async () => {
  const req = {
    body: {
      email: 'test@example.com',
      name: 'TestUser',
      lastname: 'TestLastName',
      phone: '123456789',
      password: 'password123',
    },
  };

  const res = {
    status: jest.fn().mockReturnThis(),
    json: jest.fn(),
  };

  const next = jest.fn();

  // Simula el comportamiento de las funciones de modelo User y Rol
  User.create = jest.fn().mockResolvedValue({ id: 123 }); // Simula la creación de un usuario
  Rol.create = jest.fn().mockResolvedValue(); // Simula la creación de un rol

  await userController.register(req, res, next);

  // Verifica que las funciones y métodos esperados se hayan llamado con los argumentos correctos
  expect(User.create).toHaveBeenCalledWith(req.body);
  expect(Rol.create).toHaveBeenCalledWith(123, 1); // Simula un rol por defecto
  expect(res.status).toHaveBeenCalledWith(201);
  expect(res.json).toHaveBeenCalledWith({
    success: true,
    message: 'El registro se realizo correctamente, ahora inicia sesion',
    data: 123, // El ID del usuario simulado
  });

  // Verifica que next no haya sido llamado (no se esperan errores)
  expect(next).not.toHaveBeenCalled();
});
```

Figura 5: Prueba unitaria #3 Registro de usuario

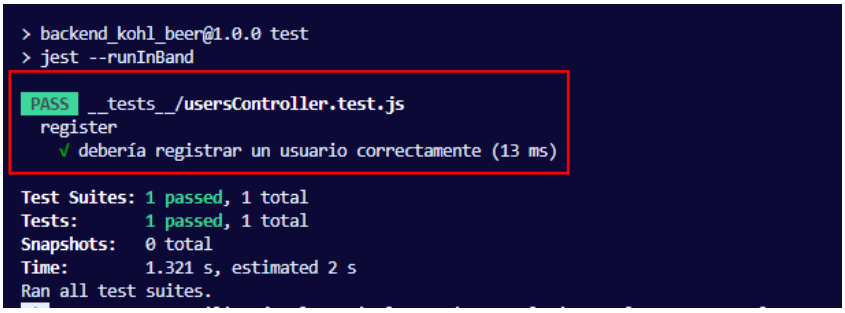


Figura 6: Resultado prueba unitaria #3

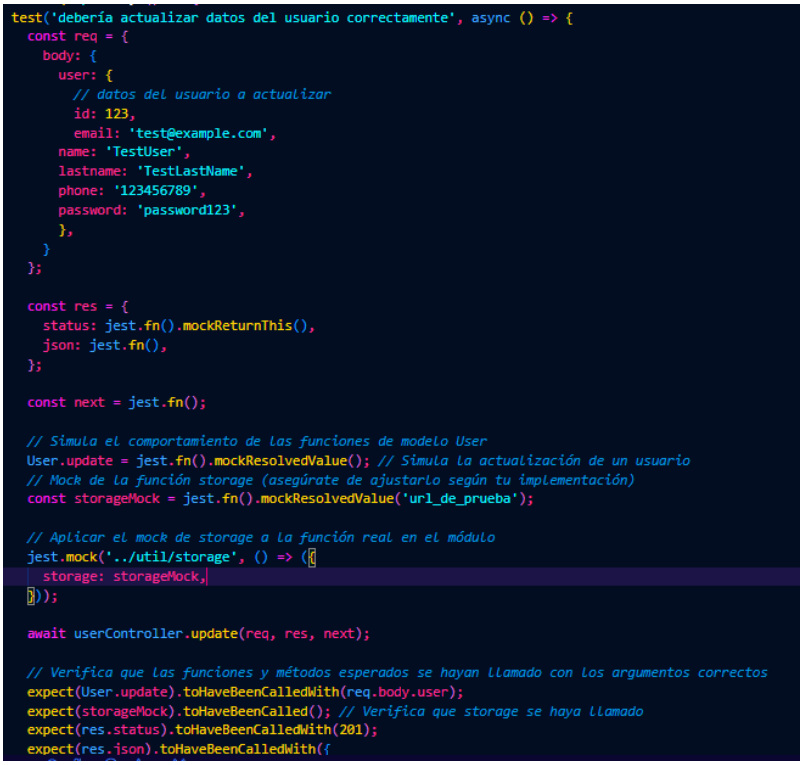


Figura 7: Prueba unitaria #4 Actualización datos del usuario

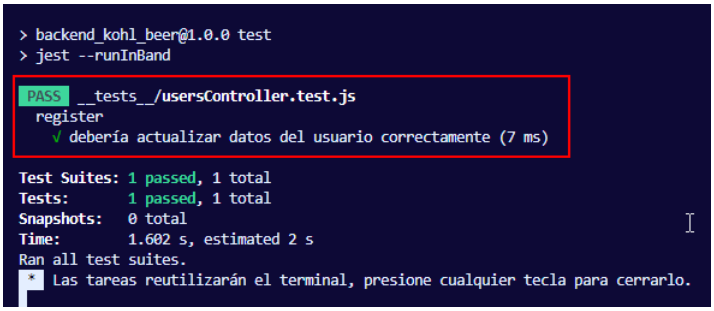


Figura 8: Resultado prueba unitaria #4

```
test('debería autenticar al usuario correctamente', async () => {
  const req = {
    body: {
      email: 'test@example.com',
      password: 'password123',
    },
  };

  const res = {
    status: jest.fn().mockReturnThis(),
    json: jest.fn(),
  };

  const next = jest.fn();

  // Simula el comportamiento de la función de modelo User
  User.findByEmail = jest.fn().mockResolvedValue({
    id: 123,
    name: 'TestUser',
    lastname: 'TestLastName',
    email: 'test@example.com',
    phone: '123456789',
    image: 'test_image.jpg',
    roles: ['CLIENTE'],
    password: 'hashedPassword', // Simula una contraseña ya hasheada almacenada en la base de datos
  });

  User.isPasswordMatched = jest.fn().mockReturnValue(true); // Simula que la contraseña coincide

  User.updateToken = jest.fn().mockResolvedValue(); // Simula la actualización del token

  await userController.login(req, res, next);

  // Verifica que las funciones y métodos esperados se hayan llamado con los argumentos correctos
  expect(User.findByEmail).toHaveBeenCalledWith('test@example.com');
  expect(User.isPasswordMatched).toHaveBeenCalledWith('password123', 'hashedPassword');
  expect(User.updateToken).toHaveBeenCalledWith(123, 'JWT fakeToken');
```

Figura 9: Prueba unitaria #5 Inicio de sesión

```
> backend_kohl_beer@1.0.0 test
> jest --runInBand

  console.log
    USUARIO ENVIADO [object Object]

      at Object.log [as login] (controllers/usersController.js:208:25)

 PASS  __tests__/usersController.test.js
  login
    √ debería autenticar al usuario correctamente (55 ms)
    √ debería manejar la autenticación fallida debido a una contraseña incorrecta (2 ms)
    √ debería manejar la autenticación fallida debido a un usuario no encontrado (1 ms)

Test Suites: 1 passed, 1 total
Tests:       3 passed, 3 total
Snapshots:   0 total
Time:        1.54 s, estimated 2 s
Ran all test suites.
*  Las tareas reutilizarán el terminal, presione cualquier tecla para cerrarlo.
```

Figura 10: Resultado prueba unitaria #5

```
test('debería cerrar sesión correctamente', async () => {
  const req = {
    body: {
      id: 123,
    },
  };

  const res = {
    status: jest.fn().mockReturnThis(),
    json: jest.fn(),
  };

  const next = jest.fn();

  // Simula el comportamiento de la función de modelo User
  User.updateToken = jest.fn().mockResolvedValue(); // Simula la actualización del token

  await userController.logout(req, res, next);

  // Verifica que la función y método esperados se hayan llamado con los argumentos correctos
  expect(User.updateToken).toHaveBeenCalledWith(123, null);

  // Verifica la respuesta enviada
  expect(res.status).toHaveBeenCalledWith(201);
  expect(res.json).toHaveBeenCalledWith({
    success: true,
    message: 'La sesion del usuario se ha cerrado correctamente',
  });

  // Verifica que next no haya sido llamado (no se esperan errores)
  expect(next).not.toHaveBeenCalled();
});
});
```

Figura 11: Prueba unitaria #6 Cerrar sesión

```
> backend_kohl_beer@1.0.0 test
> jest --runInBand

 PASS  __tests__/usersController.test.js
  logout
    √ debería cerrar sesión correctamente (13 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        1.658 s, estimated 2 s
Ran all test suites.
*  Las tareas reutilizarán el terminal, presione cualquier tecla para cerrarlo.
```

Figura 12: Resultado prueba unitaria #6

```
test('debería obtener todas las categorías correctamente', async () => {
  const req = {};
  const res = {
    status: jest.fn().mockReturnThis(),
    json: jest.fn(),
  };
  const next = jest.fn();

  // Simula el comportamiento de la función de modelo Category
  const categoriasMock = [{ id: 1, nombre: 'Categoria1' }, { id: 2, nombre: 'Categoria2' }];
  Category.getAll = jest.fn().mockResolvedValue(categoriasMock);

  await categoryController.getAll(req, res, next);

  // Verifica que la función esperada se haya llamado
  expect(Category.getAll).toHaveBeenCalled();

  // Verifica la respuesta enviada
  expect(res.status).toHaveBeenCalledWith(201);
  expect(res.json).toHaveBeenCalledWith(categoriasMock);

  // Verifica que next no haya sido llamado (no se esperan errores)
  expect(next).not.toHaveBeenCalled();
});
```

Figura 13: Prueba unitaria #7 Listar categorías

```
test('debería crear una categoría correctamente', async () => {
  const req = {
    body: {
      nombre: 'NuevaCategoria',
    },
  };
  const res = {
    status: jest.fn().mockReturnThis(),
    json: jest.fn(),
  };
  const next = jest.fn();

  // Simula el comportamiento de la función de modelo Category
  const categoriaCreadaMock = { id: 3, nombre: 'NuevaCategoria' };
  Category.create = jest.fn().mockResolvedValue(categoriaCreadaMock);

  await categoryController.create(req, res, next);

  // Verifica que la función esperada se haya llamado
  expect(Category.create).toHaveBeenCalledWith(req.body);

  // Verifica la respuesta enviada
  expect(res.status).toHaveBeenCalledWith(201);
  expect(res.json).toHaveBeenCalledWith({
    message: 'La categoria se creo correctamente',
    success: true,
    data: categoriaCreadaMock.id,
  });

  // Verifica que next no haya sido llamado (no se esperan errores)
  expect(next).not.toHaveBeenCalled();
});
```

Figura 14: Prueba unitaria #8 Crear categorías

```
    Error: Error: Error al crear categoría

      at Object.log [as create] (controllers/categoriesController.js:40:21)

PASS __tests__/categoriesController.test.js
  Category Controller
    getAll
      √ debería obtener todas las categorías correctamente (38 ms)
      √ debería manejar errores al obtener categorías (5 ms)
    create
      √ debería crear una categoría correctamente (10 ms)
      √ debería manejar errores al crear una categoría (10 ms)

Test Suites: 1 passed, 1 total
Tests:       4 passed, 4 total
Snapshots:   0 total
Time:        0.94 s, estimated 1 s
Ran all test suites.
```

Figura 15: Resultado prueba unitaria #7 y #8

```
test('debería obtener productos por categoría correctamente', async () => {
    const expectedData = [{ id: 1, name: 'Product1' }];
    Product.findByCategory.mockResolvedValue(expectedData);

    req.params.id_category = 1;

    await ProductController.findByCategory(req, res, next);

    expect(res.status).toHaveBeenCalledWith(201);
    expect(res.json).toHaveBeenCalledWith(expectedData);
    expect(next).not.toHaveBeenCalled();
});
```

Figura 16: Prueba unitaria #9 Listar productos por categorías

```
test('debería crear un producto correctamente', async () => {
    const expectedProduct = { id: 1, name: 'Product1' };
    const expectedFiles = [
        { /* imagen 1 */ },
        { /* imagen 2 */ },
    ];

    Product.create.mockResolvedValue({ id: 1 });
    Product.update.mockResolvedValue();

    req.body.product = expectedProduct;
    req.files = expectedFiles;

    await ProductController.create(req, res, next);

    // Espera a que todas las funciones asíncronas se completen
    await new Promise(resolve => setImmediate(resolve));

    expect(Product.create).toHaveBeenCalledWith(expectedProduct);
    expect(Product.update).toHaveBeenCalledWith(expectedProduct);
    expect(res.status).toHaveBeenCalledWith(201);
    expect(res.json).toHaveBeenCalledWith({
        success: true,
        message: 'El producto se ha registrado correctamente',
    });
    expect(next).not.toHaveBeenCalled();
});
```

Figura 17: Prueba unitaria #10 Crear productos

```
  console.log
    Error: Error: Error al registrar el producto

      at Object.log [as create] (controllers/productsController.js:99:25)

 PASS  __tests__/productsController.test.js
  Product Controller
    findByCategory
      √ debería obtener productos por categoría correctamente (5 ms)
      √ debería manejar errores al obtener productos por categoría (21 ms)
    create
      √ debería crear un producto correctamente (7 ms)
      √ debería manejar errores al crear un producto (61 ms)

Test Suites: 1 passed, 1 total
Tests:       4 passed, 4 total
Snapshots:   0 total
Time:        1.634 s, estimated 2 s
Ran all test suites.
```

Figura 18: Resultado prueba unitaria #9 y #10

```
test('debería obtener direcciones por usuario correctamente', async () => {
  const expectedData = [{ id: 1, address: 'Dirección1' }];
  Address.findByUser.mockResolvedValue(expectedData);

  req.params.id_user = 1;

  await AddressController.findByUser(req, res, next);

  expect(res.status).toHaveBeenCalledWith(201);
  expect(res.json).toHaveBeenCalledWith(expectedData);
  expect(next).not.toHaveBeenCalled();
});
```

Figura 19: Prueba unitaria #11 Crear productos

```
test('debería crear una nueva dirección correctamente', async () => {
  const expectedAddress = { id: 1, address: 'Dirección1' };

  Address.create.mockResolvedValue(expectedAddress);

  req.body = expectedAddress;

  await AddressController.create(req, res, next);

  expect(Address.create).toHaveBeenCalledWith(expectedAddress);
  expect(res.status).toHaveBeenCalledWith(201);
  expect(res.json).toHaveBeenCalledWith({
    success: true,
    message: 'La direccion se creo correctamente',
    data: expectedAddress.id,
  });
  expect(next).not.toHaveBeenCalled();
});
```

Figura 20: Prueba unitaria #12 Crear productos

```
  console.log
    Error Error: Hubo un error creando la direccion

      at Object.log [as create] (controllers/addressController.js:40:21)

 PASS  __tests__/addressController.test.js
  Address Controller
    findByUser
      √ debería obtener direcciones por usuario correctamente (28 ms)
      √ debería manejar errores al obtener direcciones por usuario (5 ms)
    create
      √ debería crear una nueva dirección correctamente (1 ms)
      √ debería manejar errores al crear una nueva dirección (8 ms)

Test Suites: 1 passed, 1 total
Tests:       4 passed, 4 total
Snapshots:   0 total
Time:        1.09 s, estimated 2 s
Ran all test suites.
```

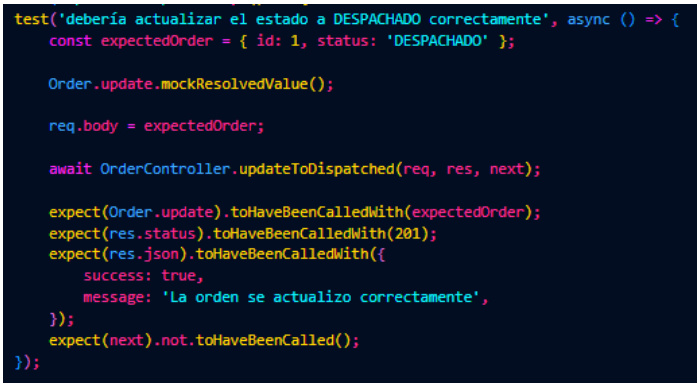
Figura 21: Resultado prueba unitaria #11 y #12

```
test('debería obtener órdenes por estado correctamente', async () => {
    const expectedData = [{ id: 1, status: 'PAGADO' }];
    Order.findByStatus.mockResolvedValue(expectedData);

    req.params.status = 'PAGADO';

    await OrderController.findByStatus(req, res, next);

    expect(res.status).toHaveBeenCalledWith(201);
    expect(res.json).toHaveBeenCalledWith(expectedData);
    expect(next).not.toHaveBeenCalled();
});
```

Figura 22: Prueba unitaria #13 Listar pedido por estado

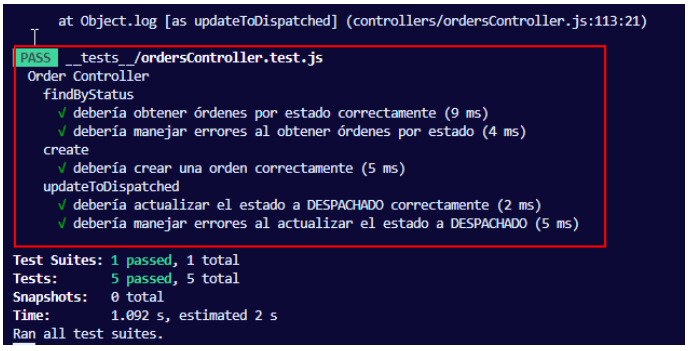
```
test('debería crear una orden correctamente', async () => {
    const expectedOrder = { id: 1, status: 'PAGADO', products: [] };

    Order.create.mockResolvedValue(expectedOrder);
    OrderHasProduct.create.mockResolvedValue();

    req.body = expectedOrder;

    await OrderController.create(req, res, next);

    expect(Order.create).toHaveBeenCalledWith(expectedOrder);
    expect(OrderHasProduct.create).toHaveBeenCalledWith(
        expectedOrder.id,
        expect.any(Number),
        expect.any(Number)
    );

    expect(res.status).toHaveBeenCalledWith(201);
    expect(res.json).toHaveBeenCalledWith({
        success: true,
        message: 'La orden se creo correctamente',
        data: expectedOrder.id,
    });

    expect(next).not.toHaveBeenCalled();
});
```

Figura 23: Prueba unitaria #14 Crear pedido



```
test('debería actualizar el estado a DESPACHADO correctamente', async () => {
    const expectedOrder = { id: 1, status: 'DESPACHADO' };

    Order.update.mockResolvedValue();

    req.body = expectedOrder;

    await OrderController.updateToDispatched(req, res, next);

    expect(Order.update).toHaveBeenCalledWith(expectedOrder);
    expect(res.status).toHaveBeenCalledWith(201);
    expect(res.json).toHaveBeenCalledWith({
        success: true,
        message: 'La orden se actualizo correctamente',
    });
    expect(next).not.toHaveBeenCalled();
});
```

Figura 24: Prueba unitaria #15 Actualizar estado del pedido



```
    at Object.log [as updateToDispatched] (controllers/ordersController.js:113:21)

PASS __tests__/ordersController.test.js
  Order Controller
    findByStatus
      √ debería obtener órdenes por estado correctamente (9 ms)
      √ debería manejar errores al obtener órdenes por estado (4 ms)
    create
      √ debería crear una orden correctamente (5 ms)
    updateToDispatched
      √ debería actualizar el estado a DESPACHADO correctamente (2 ms)
      √ debería manejar errores al actualizar el estado a DESPACHADO (5 ms)

Test Suites: 1 passed, 1 total
Tests:       5 passed, 5 total
Snapshots:   0 total
Time:        1.092 s, estimated 2 s
Ran all test suites.
```

Figura 25: Resultado prueba unitaria #13, #14 y #15

Prueba de carga

Se han realizado pruebas de carga en el *backend* mediante Apache JMeter para evaluar su desempeño, es importante señalar que cada ruta ha sido sometida a la carga de 60 usuarios, equivalente al máximo pico diario que el RestoBar ha

experimentado. Se recomienda un procesamiento en 1 segundo, resultando en un total de 1200 peticiones para las 20 rutas disponibles. Las configuraciones y resultados obtenidos se presentan en las **Figura 26** a la **Figura 29** para su revisión.

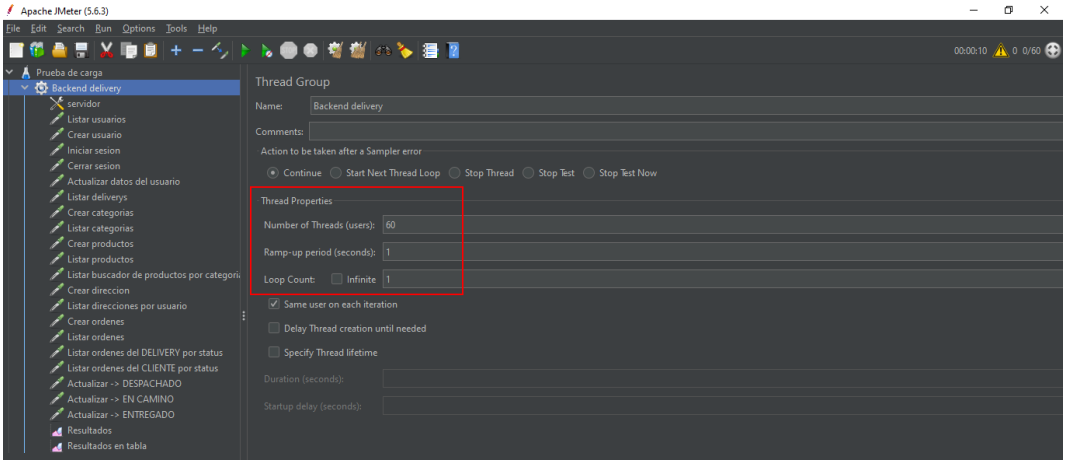


Figura 26: Configuración del grupo de hilos en JMeter

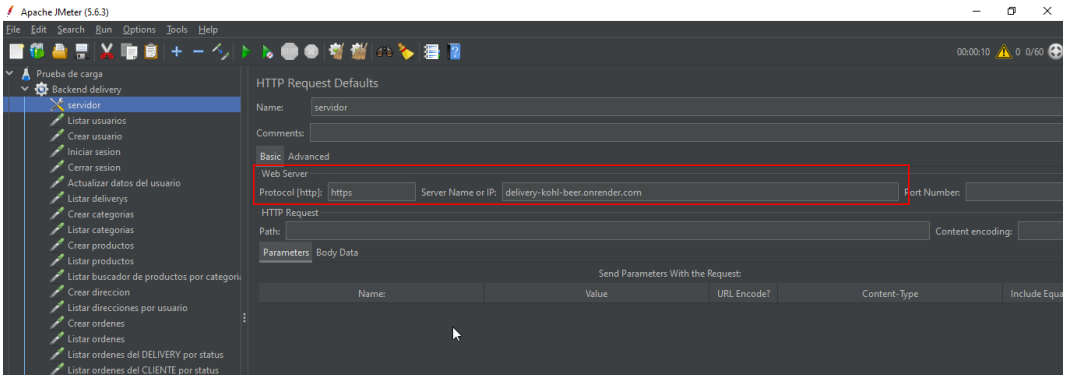


Figura 27: Configuración del servidor

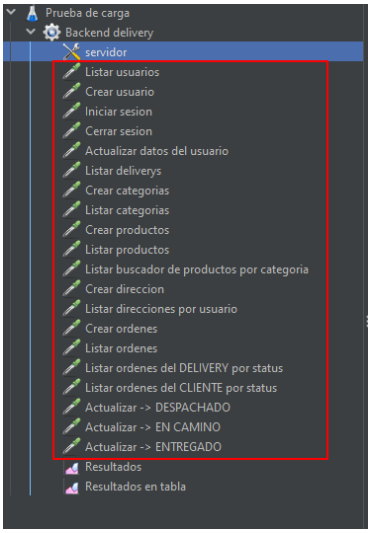


Figura 28: Configuraciones de las solicitudes HTTP

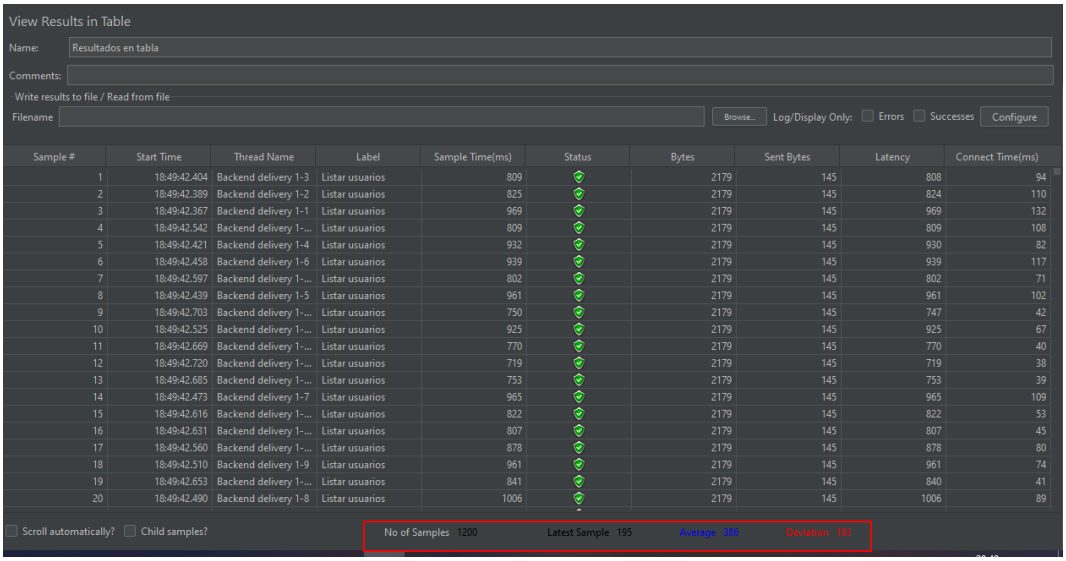


Figura 29: Resultados solicitudes HTTP

Prueba de estrés

Para las pruebas de estrés, se ha utilizado la herramienta JMeter para evaluar los resultados al someter las peticiones HTTPS a sus límites. Consideramos un aumento de 20 usuarios, llegando a un total de 80 usuarios realizando peticiones. Dado que hay 20 *endpoints* en total, se generan 1600 peticiones en total. Las configuraciones y resultados obtenidos se presentan en las **Figura 30** a la **Figura 32** para su revisión.

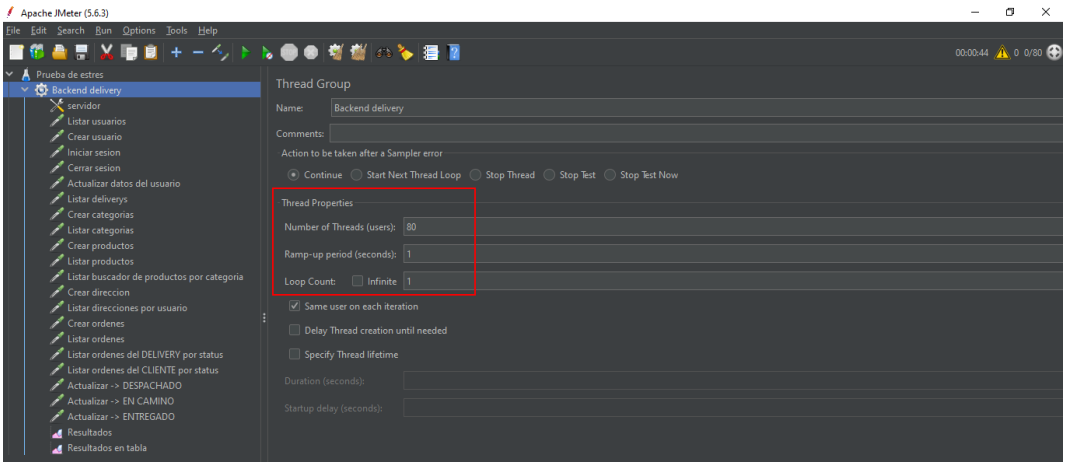


Figura 30: Configuración del grupo de hilos en JMeter para la prueba de estrés

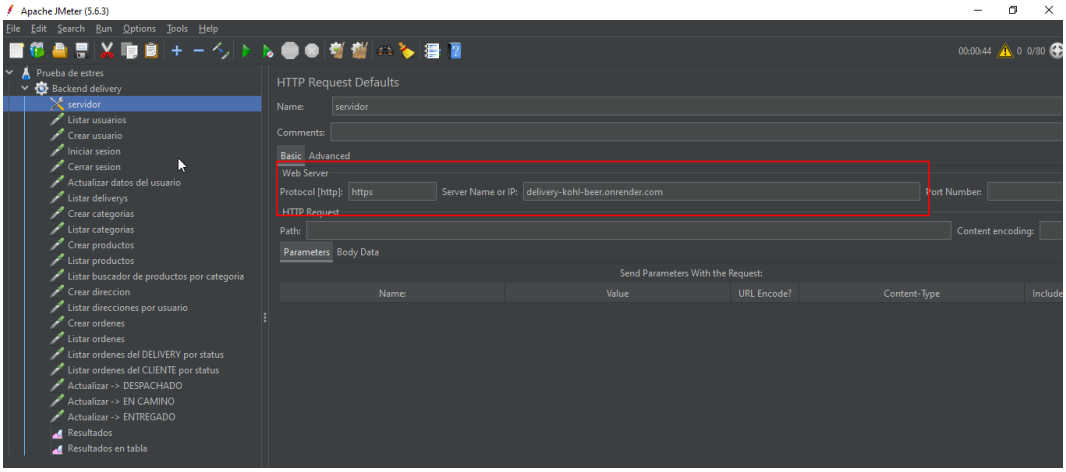


Figura 31: Configuración del servidor

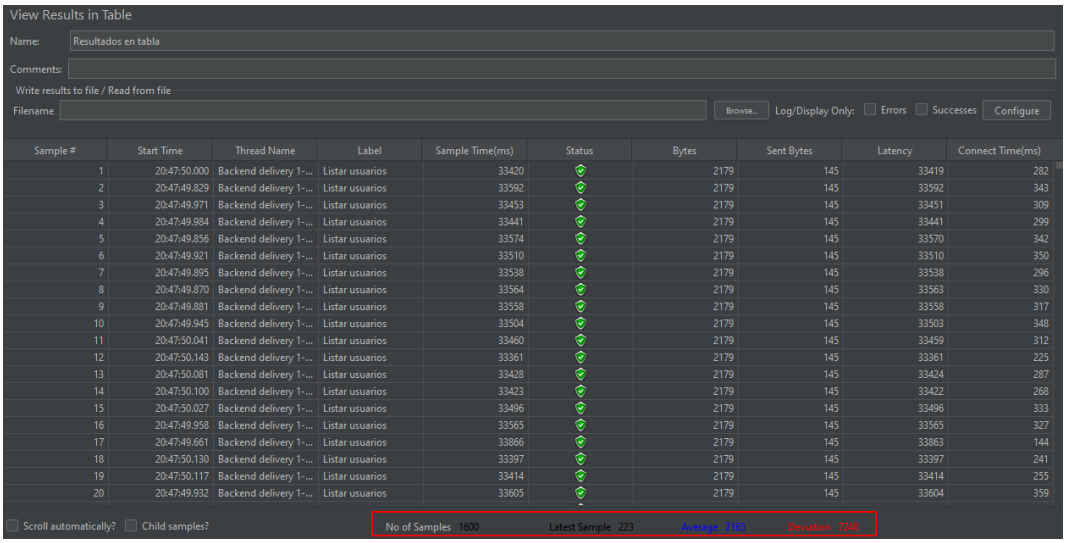


Figura 32: Resultados solicitudes HTTP para la prueba de estrés

ANEXO III

A continuación, el enlace dirige al video del manual del componente *backend* desarrollado.

https://www.youtube.com/watch?v=4mHrVp8VUR8&feature=youtu.be

ANEXO IV

A continuación, se presentan las credenciales “tipo prueba” de cada rol en el *backend*, el enlace al código fuente (repositorio *GitHub*, donde se encuentra el README que indica los pasos para desplegar de manera efectiva el *backend*).

Documentación de las APIs

El enlace siguiente contiene la evidencia en el consumo de cada *endpoint* en la herramienta *POSTMAN.*

https://app.getpostman.com/join-team?invite_code=94237e64ff495853ad38311cd8630c3f&target_code=7d32c778172d7d1d90a026fad446391d

Finalmente, el enlace al repositorio del código fuente alojado en *Github.*

https://github.com/Shiao-Li/backend_kohl_beer.git

Credenciales

Credenciales con 2 roles (Administrador y cliente)

Correo: admin@gmail.com

Contraseña: beer123

Delivery

Correo: juan@gmail.com

Contraseña: beer123

Cliente

Correo: jared@gmail.com

Contraseña: beer123