ESCUELA POLITÉCNICA NACIONAL

ESCUELA DE FORMACIÓN DE TECNÓLOGOS

DESARROLLO DE SISTEMA WEB DE UN ECOMMERCE CON FACTURACIÓN ELECTRÓNICA

DESARROLLO DEL BACK-END

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

JHOSEF ALEXANDER REA CHAMORRO

DIRECTOR: DR. RICHARD PAUL RIVERA GUEVARA

DMQ, septiembre 2022

CERTIFICACIONES

Yo, Jhosef Alexander Rea Chamorro 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.

JHOSEF ALEXANDER REA CHAMORRO

jhosef.rea@epn.edu.ec

jhosefrea0@gmail.com

Certifico que el presente trabajo de integración curricular fue desarrollado por Jhosef Alexander Rea Chamorro, bajo mi supervisión.

PhD. RICHARD PAUL RIVERA GUEVARA

DIRECTOR

Richard.rivera01@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.

REA CHAMORRO JHOSEF ALEXANDER

DEDICATORIA

Dedico este proyecto a mi madre Olga y mi padre Fernando por brindarme sus consejos, apoyo, paciencia y su confianza durante mi formación académica. A mis abuelitos por acompañarme en cada situación importante de mi vida, a mi hermana que me dio aliento de apoyo. De igual forma, a mis docentes por estar al pendiente en mis escritos junto con su guía que me ayudaron a mejorar mi proyecto.

AGRADECIMIENTO

Agradezco primero a Dios por ser el eje que guía mi vida, mi fortaleza que siempre está en los momentos más difíciles. A mi madre y a mi padre que nunca dudaron en el logro en este paso de mi vida. De hecho, siempre hubo un ambiente de confianza, cariño, respeto y apoyo incondicional para alcanzar mis objetivos.

A mis abuelitos, hermana y a mis familiares que me alentaron con sus buenas oraciones, consejos durante mi formación académica, a mis amigos por extender su mano en momentos difíciles alentándome con sus buenas vibras.

Finalmente, agradezco a mis docentes de la Escuela Politécnica Nacional, quiénes me han guiado con sus enseñanzas, me han motivado a estar al pendiente de mis tareas dentro de mi carrera. Gracias por ser tan honestos en su labor formando y corrigiendo al estudiante.

ÍNDICE DE CONTENIDO

RESUMEN

En la actualidad, Ecuador vive un aumento en transacciones electrónicas en entidades de *E-Commerce.* Sin embargo, aún se presenta el modelo tradicional compra-venta; ocasionado por el desconocimiento, el limitante acceso a Internet y desconfianza del vendedor/consumidor, así mismo, la legalidad y seguridad en transacciones “sin papel” en medios de pago electrónicos. No obstante, Ecuador muestra capacidad en el desarrollo de un ecosistema *E-Commerce;* muestra de ello es el negocio virtual, donde da a conocer su producto, creando confianza con el cliente. El vendedor se favorece con un control y privacidad en la emisión de una factura electrónica; pues, es fundamental en transacciones *E-Commerce*, por la simplificación de gestiones.

El propósito del proyecto es contribuir al sector *retail* de la ciudad de Quito a través de un *API REST* para el sistema web *E-Commerce* con facturación electrónica. Que permita, según el rol del usuario, validar la identificación, almacenar un catálogo de productos, crear una lista en el carrito de compras. Al efectuarse una compra, los usuarios reciben la factura electrónica; la cual se avala con la recepción vía *correo electrónico* de un documento electrónico de compra-venta.

Durante la construcción del *API REST* se adopta *Scrum* como metodología ágil, se integra el patrón MVC con el *framework NestJS*, que permite generar de forma ágil y eficiente un *API REST* e integrando como base de datos *MongoDB*. En el presente documento se da a conocer el procedimiento y resultados obtenidos del desarrollo del proyecto.

PALABRAS CLAVE: *API REST, Scrum, NestJS, MongoDB, factura*

ABSTRACT

Currently, Ecuador is experiencing an increase in electronic transactions in E-Commerce entities. However, the traditional purchase-sale model is still presented. This traditional model is caused by ignorance, limited Internet access and distrust of the seller/consumer. Another cause of this traditional model is the legality and security of "paperless" transactions in electronic means of payment.

For other way, Ecuador shows capacity in the development of an E-Commerce ecosystem. An example of this is the virtual business, where it makes its product known, creating trust with the client. The seller is favored with control and privacy in the issuance of an electronic invoice; therefore, it is fundamental in E-Commerce transactions, due to the simplification of procedures.

The purpose of the project is to contribute to the retail sector of the city of Quito through an API REST for the E-Commerce web system with electronic invoicing. That allows, depending on the user's role, to validate the identification, store a catalog of products or create a list in the shopping cart. When making a purchase, users receive an electronic invoice, which is endorsed with the receipt via email of an electronic purchase-sale document.

During the construction of the API REST, Scrum is adopted as an agile methodology. The MVC pattern is integrated with the NestJS framework, which allows the agile and efficient generation of an API REST, and the integration of MongoDB as a database. This document presents the procedures and results obtained, from the development of the project.

KEYWORDS: *API REST, Scrum, NestJS, MongoDB, invoice.*

.

1 DESCRIPCIÓN DEL COMPONENTE DESARROLLADO

Previo a la última década, Ecuador ha confrontado cifras alarmantes en acceso a las nuevas tecnologías en el sector rural y en zonas urbano marginales. En el área rural, apenas el 12% de habitantes ha usado Internet y el analfabetismo digital llegaba al 48% [1].

A partir del año 2012, el Estado, a través de política pública con enfoque de derechos humanos de acceso a Internet, lo que ha significado que el 46,4 % de la población acceda a la red desde su hogar. Para el año 2018, se incrementa al 66,2% de la población ecuatoriana respecto a zonas demográficas, el 71% de la población urbana accede a Internet desde su hogar. Mientras que, en el área rural, el 49% lo hace [1]. Es notoria la brecha de acceso a Internet que se ha mantenido entre el sector urbano y rural.

El sector comercial ecuatoriano se resiste al cambio a las nuevas Tecnologías de la información y comunicación. El temor se presenta al plantearse una inversión en una tienda online, ocasionado por desconocimiento, fraude, complejidad técnica, costos [2]. Asimismo, la falta de confianza del consumidor, la resistencia a procesos de compra contemporáneos, la legalidad de las transacciones y contratos “sin papel”; la privacidad y seguridad en medios de pago electrónicos [3]. Sin embargo, en Ecuador existe un ecosistema vivo y adecuado para el *E-Commerce,* después de la tendencia al uso de *smartphones, laptops y* computadoras de escritorio; como protagónico, el acceso a los servicios de Internet [222]. En conjunto con política pública; empresas ecuatorianas líderes en *E-Commerce*; realizan eventos de capacitación y *networking* como “El *Ecommerce Day*” organizado por *Ecommerce Institute regional* y coorganizado localmente por la Cámara Ecuatoriana de Comercio Electrónico CECE [4].

En la última década, Ecuador da testimonio del derecho humano al acceso a Internet, la oportunidad de crecimiento en negocio local e internacional de la mano del desarrollo del *E-Commerce.* Ambos factores, junto a la vanguardia tecnológica, las Tecnologías de la Información y la Comunicación (TIC), han propiciado la productividad laboral e inciden directamente sobre el crecimiento socioeconómico del Ecuador [5]. Esto, porque el ecosistema digital involucra un conjunto de fenómenos industriales con impacto económico que adoptan las Tecnologías de información y comunicación [2].

En este sentido, el propósito del proyecto es dar apoyo al sector retail de la ciudad de Quito a través de un *API REST* para el sistema web *E-Commerce* con facturación electrónica. Donde, un negocio puede dar muestra del producto y vender bajo la reformatoria de la Ley de Régimen Tributario. De hecho, la emisión de documentos digitales da un control al ciudadano y privacidad, pues una factura electrónica se envía al correo electrónico personal. Además, de ser el impulso que necesitan los emprendedores ecuatorianos; porque, es necesario un sitio donde se pueda potenciar y darse a conocer a miles de clientes cibernéticos.

1.1 Objetivo general

Desarrollar el *backend* de un sistema web de *E-Commerce* con facturación electrónica.

1.2 Objetivos específicos

1. Levantar requerimientos funcionales y no funcionales del sistema.
2. Diseñar la arquitectura y el modelo no relacional de la base de datos del sistema.
3. Codificar el backend con base en los módulos definidos.
4. Implementar pruebas en el *API REST*.

1.3 Alcance

Una plataforma digital *E-Commerce* necesita del desarrollo de un *API REST* que cuente con autenticación de usuario, un almacenamiento robusto; seguridad en el acceso y manipulación de los datos; para satisfacer las peticiones provenientes de la *GUI.* Más, la generación de documentos electrónicos que corroboren, bajo la ley tributaria del país, el modelo tradicional compra y venta entre vendedor y comprador.

El presente proyecto cuenta con las características de un sistema web *E-Commerce genérico;* sin embargo, no se consolidan procesos de transparencia ni participación ciudadana. Pues, el *API REST* no emite comprobantes electrónicos autorizados, bajo la normativa vigente del Sistema de Rentas Internas del Ecuador; es decir, no se incluye la facturación electrónica con el SRI; pero, sí se avala la recepción de un documento electrónico, vía correo electrónico, que consta del detalle de la compraventa, la información requirente del negocio y cliente. Para futuras versiones se podría considerar incluir la relación con el SRI en este proyecto, aclarando que para esta versión dicha relación esta fuera del alcance de este proyecto.

Para garantizar la experiencia del usuario en el sistema, se ha definido un sistema de roles basado en la validación de información de registro. El sistema de perfiles se encuentra descrito, a continuación:

Perfil administrador permite:

- Registrarse y autenticarse.
- Gestionar sobre usuarios registrados, productos y en la generación de la facturación electrónica.

Perfil comprador permite:

- Registrarse y autenticarse.
- Visualizar el detalle del producto.
- Filtrar el producto.
- Comprar productos mediante un carrito de compras.
- Recepción de la factura electrónica.

1.4 Marco Teórico

Metodología

Una metodología permite dirigir un determinado proceso de manera eficiente y ordenada para alcanzar los objetivos planteados con guía en estrategias durante el proceso de investigación [6] . Además, cabe mencionar que permite seleccionar temas de elementos significativos de un problema con el cual se puede estructurar una explicación causal. En efecto, la metodología conlleva el estudio de uno o más métodos, incluida una demostración y discusión de su lógica subyacente, un análisis de los diversos procedimientos específicos utilizados en el estudio y una discusión de sus características y propiedades. Por ello, la metodología tiene una gran importancia en la provisión de una serie de conceptos, principios y leyes para crear un correcto proceso de investigación; más, la adquisición de conocimientos [7].

Metodología ágil

Una metodología ágil es caracterizada por el desarrollo iterativo e incremental; es el conjunto de procedimientos flexibles, adaptativos y orientados a personas, pues se ajustan a la realidad y al cambio de cada equipo y proyecto. De hecho, esta propuesta metodológica, surge a principios de los años 90; logrando resultados más rápidos en el

marco del desarrollo de software sin comprometer la calidad del software y a su vez agilizando el esquema organizacional de una empresa [8].

Al caracterizarse por ser flexible; los proyectos ágiles son subdivididos en proyectos más pequeños y se tiene una simplicidad de la implementación; se adaptan mejor a los cambios, por ejemplo, en la priorización de los requerimientos; está orientado a personas porque es altamente colaborativo en el equipo de desarrollo e incluye comunicación constante con el cliente en entregas-avances y en la retroalimentación. Por consiguiente, tanto el producto como el proceso son mejorados frecuentemente [8].

Backend

Comprende la utilización en el lado del servidor, para realizar gestión de peticiones de información, además de la gestión de las bases de datos que se encuentran alojadas en los mismos [9]. El desarrollo de *backend* involucra los otros aspectos de la programación a los que se enfrenta el servidor/programador y son responsables de la respuesta a la acción efectuada por el usuario/cliente; es decir, el funcionamiento del sistema [10].

API REST

Es una arquitectura de desarrollo web basada en el protocolo *HTTP* para el transporte del tráfico de la información de un recurso específico sobre el sistema. Es decir, es el intermediario teniendo como función, el recibimiento de peticiones del cliente, devolviendo la respuesta en un formato de texto organizado; controlando las restricciones definidas [11].

E-Commerce

También denominado comercio electrónico tiene su origen a la par de la invención de los instrumentos electrónicos dirigidos a la transmisión remota de datos [12]. Pero, tiene su auge con la llegada de la web 1.0; a través de la invención de una red empresarial global con vista a nuevos modelos de negocio; sin embargo, bajo el formato tradicional de compra y venta [13].

El *e-commerce* consiste en un sistema de compra y venta de productos o servicios que se actúa entre un comprador y un vendedor, a través de Internet. Donde se efectúa la distribución, transacciones, marketing y suministro de información de estos [14]. En este tipo de plataformas online también se gestionan cobros y pagos emitiendo documentos en formato digital [15].

Facturación Electrónica

Marco legal

Junto a normativas del SRI y administración tributaria como, Ley No. 67. En el R.O. Suplemento 557 de 17 de abril del 2002; con reglamento emitido por el Dr. Gustavo Noboa Bejarano mediante Decreto No. 3496 del 12 de diciembre de 2002. Permiten establecer normas, requerimientos, desarrollo e implementación de facturación electrónica en Ecuador; manteniendo vigencia en Ley de comercio electrónico, firmas electrónicas; desde el año 2014 es obligatorio el uso de comprobantes electrónicos para contribuyentes (sujetos pasivos) bajo requisitos legales y reglamentarios exigibles por el SRI [16].

La facturación electrónica es un sistema de emisión de facturas en formato digital y recepción en formato electrónico. Las facturas generadas por un software utilizan un formato estandarizado, por consiguiente, pueden ser leídas por cualquier software; eliminando la necesidad de introducir nuevos datos o errores y asegurando la interoperabilidad de los datos [17] .

En Ecuador, una factura electrónica debe cumplir con los requisitos legales actualizados por el SRI garantizando así, la autenticidad e integridad del origen e información de la factura electrónica. Este método potencializa el modelo de negocio, al brindar soporte de todas las transacciones comerciales que este mismo realice [18].

2 METODOLOGÍA

La metodología de investigación que se ha empleado es un caso de estudio porque se busca resolver un aspecto de la vida real. Pues, permite un enfoque de investigación cualitativa, acoplándose de la utilización de múltiples métodos de recopilación de información para un examen detallado de un solo "caso" [19]. En este proyecto, el caso de estudio se enfoca en un grupo emprendedor, el sector retail de la ciudad de Quito; basada en el desarrollo de un *API REST E-Commerce*, proporcionando el acceso y manejo de la información; más, da otra perspectiva en la relación y experiencia entre cliente-vendedor con una automatización de procesos de compra-venta.

2.1 Metodología de Desarrollo

Scrum es una metodología ágil, flexible con enfoque iterativo e incremental cuyo principal objetivo es optimizar el retorno en procesos controlando riesgos. Es decir, es un *framework* para el desarrollo y mantenimiento de un producto en el marco del desarrollo de software. De hecho, asegura entregables en iteraciones cortas de tiempo, pues permite generar iteraciones, conocidas como *Sprints*, que facilitan el desarrollo de todas las actividades propuestas durante el ciclo de vida del proyecto [20].

Un equipo Scrum se caracteriza por ser auto-organizados y multifuncionales. Por consiguiente, se garantiza la entrega de resultados de calidad que permitan cumplir los objetivos de negocio del cliente [21].

Roles

El desarrollo de software es una actividad que requiere trabajo colaborativo donde cada persona tiene un rol. Para cada rol, se definen sus objetivos, actividades, interacción con otros roles, herramientas a utilizar y un plan de trabajo. Para un desarrollo mejor de cada rol se necesita experiencia y capacidades personales para lograr objetivos claros y estratégicos de desarrollo de software. A continuación, se habla de 3 roles importantes para un equipo de *Scrum* [22].

Product Owner

Su principal objetivo es obtener la mayor funcionalidad posible del producto a ser desarrollado por un equipo de *Scrum*. Pues, se encarga de la exposición, manejo y priorización de las tareas del *Product Backlog* [20]. Además, es el único perfil que se comunica con el cliente; por consiguiente, se obtienen conocimientos sobre el negocio [21]. Este rol es representado por el Dr. Richard Rivera debido a su experiencia y compromiso.

Scrum Master

Es el manager de *Scrum*, un líder responsable de proteger y facilitar procesos, transmitir el conocimiento teórico y práctico, remover los impedimentos que surgen dentro del equipo *Scrum* e incluso capacita al *Product Owner*, en el desarrollo de un sprint [20]. El Dr. Richard Rivera representa el rol de *Scrum Master*.

Development Team

El equipo de desarrollo prioriza las tareas del *Product Owner* que estiman las tareas del *Product Backlog*; mostrando sus capacidades en el diseño e implementación del software. Para ello, el equipo debe identificar las vulnerabilidades y amenazas que pueden presentarse durante el ciclo de vida del producto de software [20].

En la **TABLA I** se presenta la manera en la que se han distribuido los roles para la realización del proyecto.

TABLA I: Asignación de roles

Rol	**Integrante**
Product Owner	Dr. Richard Rivera
Scrum Master	Dr. Richard Rivera
Development Team	Sr. Jhosef Rea

Artefactos

Se utiliza para transparentar el proyecto del equipo *Scrum*; forjándose una misma visión de lo que está ocurriendo en el ciclo de vida de Software. De tal manera que brinda información para inspección y adaptación [21].

Recopilación de Requerimientos

Son la pieza fundamental en un proyecto de desarrollo de software. Es un reflejo detallado de las necesidades de los clientes o usuarios del sistema; al cumplir un papel primordial en

la especificación de las funcionalidades acorde al modelo de negocio. Pues, es la base que permite verificar el alcance y cumplimiento de los objetivos establecidos [23]. Los requerimientos recopilados se encuentran en el **ANEXO II**.

Historias de Usuario

Empleadas como una herramienta de comunicación que combina las fortalezas de ambos medios: escrito y verbal, la seguridad, representan una funcionalidad de software desde el punto de vista del usuario; al enfocarse en las necesidades o requerimientos del negocio [24]. En la **TABLA II** se muestra un ejemplo de las historias de usuario que se utilizaran en el proyecto. Las historias de usuario restantes se detallan en el **ANEXO II**.

TABLA II: Historia de Usuario HU-01

HISTORIA DE USUARIO	
Identificador: HU-01	**Usuario:** Administrador y usuario final
Nombre de historia: Registrar Usuario	
Prioridad en Negocio (Alto/Medio/Baja): Alta	**Riesgo en Desarrollo (Alto/Medio/Baja):** Medio
Iteración asignada: 1	
Responsable: Jhosef Rea	
Descripción: La API debe proporcionar la funcionalidad de registro, para que el usuario administrador pueda registrarse.	
Observación: No se deben pedir muchos datos para registrarse. Basta con los campos, nombre/nickname, email y contraseña. Los usuarios podrán acceder al sistema a base de su rol.	

Product Backlog

Es una lista de tareas ordenadas de los requerimientos del proyecto definidas por el *Product Owner*; misma que mantiene una constante comunicación con el cliente para portar un orden de prioridad en las tareas con su respectivo tiempo estimado [21]. El detalle del *Product Backlog* se encuentra en el **ANEXO II**.

Sprint Backlog

Es un conjunto de tareas e ítems del *Product Backlog* diseñado para transparentar la información del trabajo realizado en cada entrega final de un sprint; además, de brindar

información para inspección y adaptación [20]. El equipo Scrum es el encargado de realizar estas tareas. El *Sprint Backlog* se encuentra en el **ANEXO II**.

2.2 Diseño de la arquitectura

A partir de los requerimientos se estructura el diseño de la solución del sistema que es definido por medio de un patrón arquitectónico; donde se presentan los componentes del sistema, las propiedades visibles a otros componentes, su relación y su comunicación; satisfaciendo las necesidades del negocio [25]. En esta siguiente sección se detalla el modelo arquitectónico usado para el desarrollo del proyecto.

Arquitectura de datos

NestJs es un framework de código abierto basado en Express y utiliza en gran medida la arquitectura de Angular; es utilizado para crear aplicaciones del lado del servidor de NodeJs; es decir, se ocupa del almacenamiento y la gestión de datos directamente en el backend para luego responder al cliente. Además, proporciona una fácil integración del API de Swagger, el lenguaje de programación de Typescript, lo que permite a los desarrolladores crear aplicaciones eficientes, mantenibles y altamente escalables. De hecho, al tener un marco extensible, permite crear de forma ágil y eficiente un *API* y microservicios compuestos por sistemas *backend* como bases de datos y servicios *SOAP* o *REST* [26].

En la **Fig. 1** se visualiza las herramientas usadas para el desarrollo del proyecto, así como también el flujo del Modelo-Vista-Controlador (MVC).

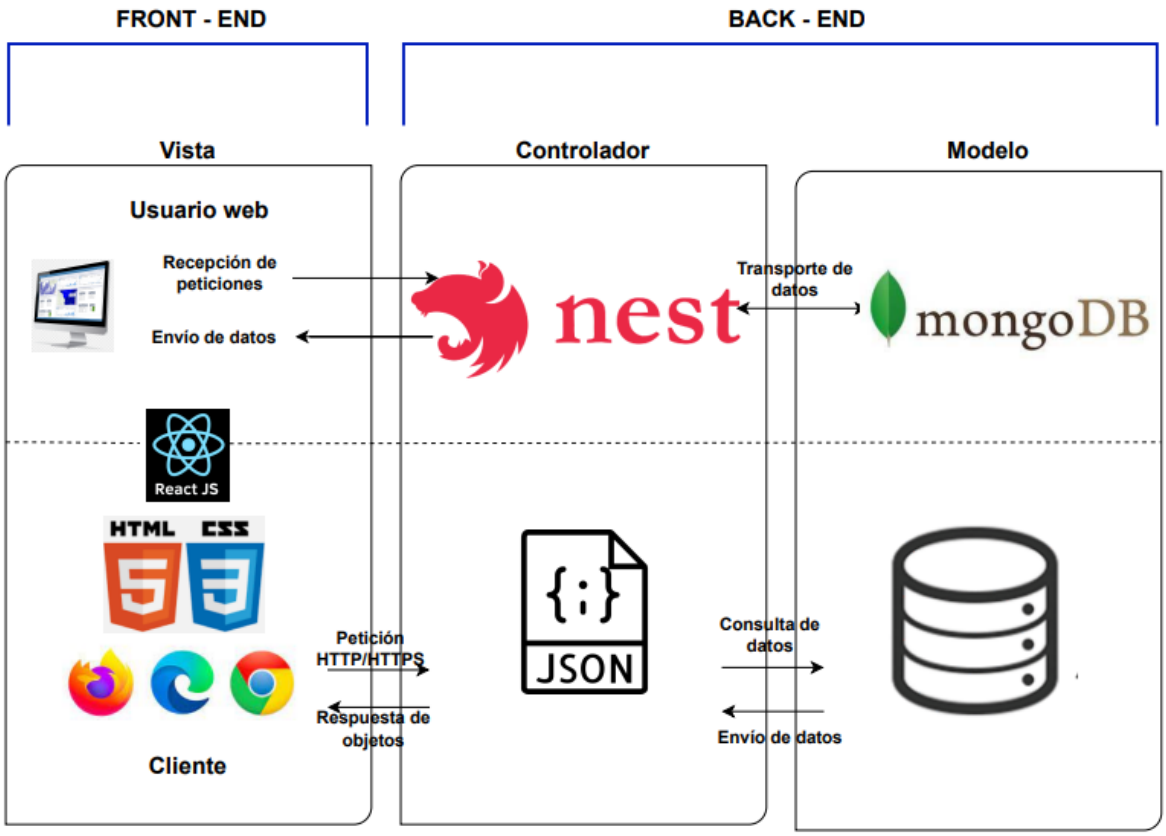


Fig. 1: Patrón Arquitectónico del API REST.

Patrón arquitectónico Modelo Vista Controlador (MVC)

MVC es una arquitectura de software utilizada en la estructuración del código; manteniendo distintas capas encargadas de una tarea concreta; en función de sus responsabilidades o conceptos. Este patrón arquitectónico se adapta a distintos tipos de aplicaciones; de hecho, satisface la entrega de un software más robusto, potencia la facilidad de mantenimiento, reutilización del código y la separación de conceptos; es decir, con un ciclo de vida del software, más adecuado [27].

Su fundamento es la separación del código en tres capas diferentes, mismas que se describen a continuación:

Modelo: Es la capa donde se encuentra la lógica del proyecto, además se trabaja directamente con los datos teniendo aquí las formas para su correcta manipulación [27].

Vista: Es la interfaz de usuario, la cual interactúa directamente con el usuario; atiende solicitudes y muestra resultados. Se trabaja realizando llamados a los datos sin tener un acceso directo a los mismos [27].

Controlador: Es la capa que funciona como enlace entre la capa de vista y modelo, la cual implementa mecanismos de respuesta a solicitudes que el usuario tenga dentro del sistema web [27].

Herramientas de desarrollo

Las herramientas de desarrollo que se usan son acordes al patrón arquitectónico definido para el Sistema Web. En la **TABLA III** se muestran las herramientas requeridas para la elaboración e implementación.

TABLA III Herramientas utilizadas para el desarrollo del Sistema Web.

Herramienta	Justificación
NestJS	Como *framework* se adopta *NestJS, porque,* da un punto de partida con metodología y estructura en el desarrollo de un *API REST* bajo el intérprete de Node.js; utilizando como lenguaje de *TypeScript*. En efecto, permite crear aplicaciones del lado del servidor de forma ágil, eficiente y escalable. [28]
MongoDB	Como base de datos se integra MongoDB es gestor de base de datos NoSQL, un modelo orientado a documentos que se almacenan en BSON. De hecho, aporta flexibilidad y rapidez en el tiempo de desarrollo con sistemas de trabajo más ágiles; pues, no necesita seguir un esquema. [29]
Git	Git es un sistema de control de versiones distribuido gratuito y de código abierto diseñado. Es usado durante la etapa de desarrollo, obteniendo un producto con rapidez, eficiencia y tolerante a fallos. [30]
GitHub	GitHub es una plataforma de desarrollo de software usado durante la etapa de desarrollo del proyecto para almacenar, rastrear y colaborar en proyectos, pues, permite a los desarrolladores crear sus propios archivos de código y colaborar en tiempo real con otros desarrolladores en proyectos de código abierto. [31]
Swagger	Como cliente HTTP se integra Swagger que, es un conjunto de

	herramientas de código abierto que simplifica el proceso de construir un API basadas en REST pues, ayuda a documentar, probar y consumir servicios web RESTful. Además, proporciona una interfaz de usuario amigable y las herramientas necesarias para escribir APIs seguras, de alto rendimiento y escalables [32].
Postman	Postman es una herramienta últi cuando se trata de direccionar las Api Restful e incluso para realizar pruebas de una Api Rest en construcción. En el proyecto se ha utilizado como segunda opción en clientes HTTP [33].

3 RESULTADOS

En esta sección se presentan los resultados obtenidos conforme a las actividades realizadas en cada *Sprint*.

Sprint 0. Configuración del ambiente de desarrollo

El *Sprint* 0 corresponde a la primera iteración del *Sprint Backlog* que involucra la preparación del equipo *Scrum* y configuraciones esenciales para dar inicio al desarrollo del proyecto. En este Sprint se desarrollan las siguientes tareas:

- Definición de requerimientos.
- Instalación del *IDE Visual Studio Code* y la *CLI* del *framework NestJs*.
- Modelar la base de datos *NoSQL.*
- Creación del proyecto con el *framework NestJs*.
- Creación del repositorio en *GitHub*.

Definición de requerimientos

Se definen los perfiles de usuario con base en lo estipulado en el *Sprint Backlog*. Se cuenta con dos roles de usuario: *"Admin"* y "Cliente" tal y como se presenta a continuación en las figuras **Fig. 2** y **Fig. 3.**

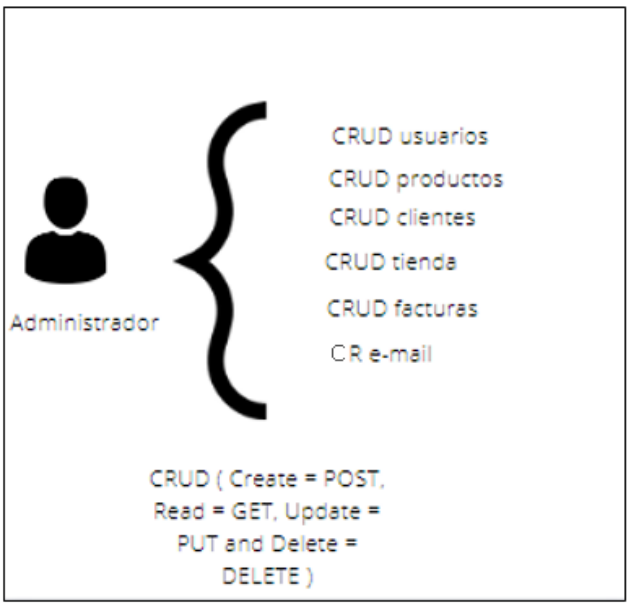


Fig. 2: Usuario con perfil administrador.

Fig. 3: Usuario con perfil cliente.

Roles de Usuario

A continuación, la **Fig. 4** se ilustra a los dos perfiles de usuario y a los módulos a los cuales tienen acceso dependiendo del rol que tengan asignado.

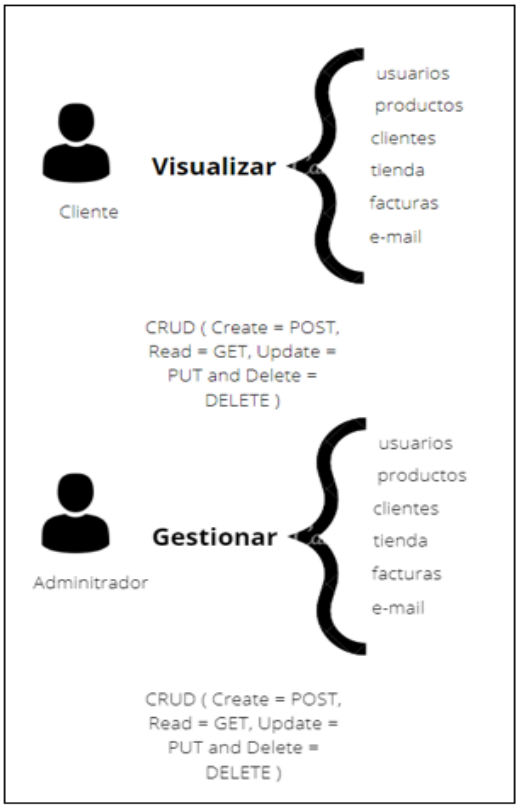


Fig. 4: Usuario y roles para el *backend*.

Instalación del *IDE Visual Studio Code y* la CLI del *framework NestJs.*

Para el desarrollo del proyecto se ha utilizado el entorno de desarrollo *Visual Studio Code* porque cuentan con un sinnúmero de extensiones desarrolladas por la comunidad que ayudan con la codificación en f*rameworks*, cuenta con *snippets* de varios lenguajes de programación e incluso extensiones de clientes.

El *backend* se desarrolla tomando de intérprete a *Node.js* y usando el lenguaje *TypeScript*. Por consiguiente, se instaló el paquete de Node.js y *npm.* Seguidamente, se instala la *CLI* de *NestJs*. En la **Fig. 5** se observa, las versiones del entorno de desarrollo utilizado, la *shell* instalada y el gestor de paquetes *npm* necesarios para crear un proyecto en *NestJs*.

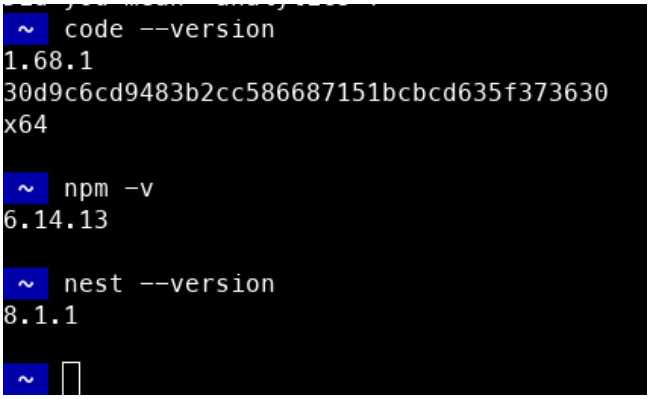


Fig. 5: Versiones de *IDE* y *CLI.*

Modelar la base de datos *NoSQL*.

Para el almacenamiento de los datos del sistema *E-Commerce* se ha optado por un sistema gestor de base de datos *NoSQL*, llamado *MongoDB*; su modelamiento fue establecido en *Microsoft Excel*. Además, se ha empleado *MongoDB Compass* como *GUI* para poder visualizar las colecciones almacenadas en la base de datos, como se ilustra en la **Fig. 6** la base de datos *NoSQL* que se ha utilizado para la comunicación con el *backend*; donde se establecieron las colecciones con sus respectivos campos y su tipo de dato; al igual, la referencia entre colecciones. Para un mejor entendimiento, el esquema completo se puede apreciar en el **ANEXO II** del presente documento.

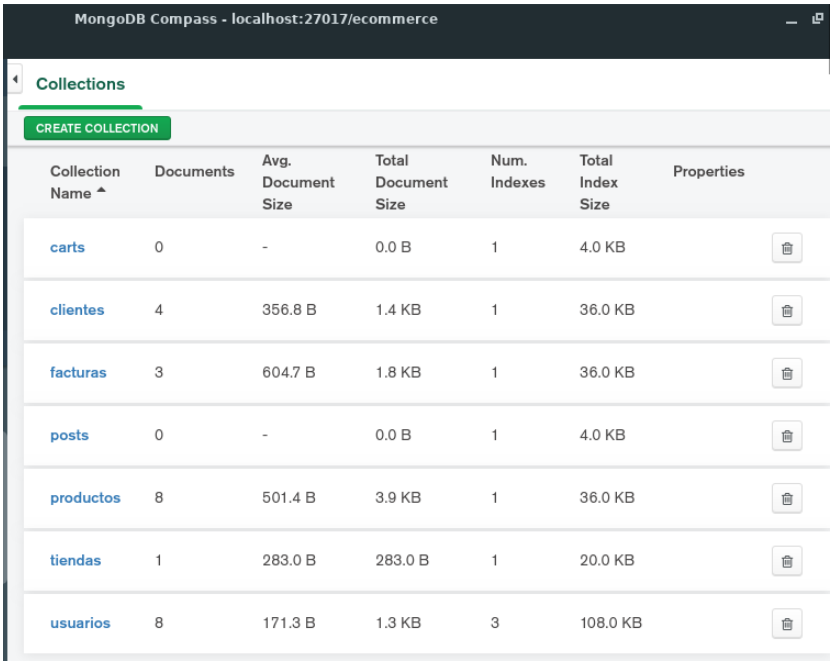


Fig. 6: Colecciones de la base de datos NoSQL.

Creación del proyecto con el framework NestJs.

En la **Fig. 7** se visualiza la arquitectura *REST* del proyecto generado a través de la *CLI* de *NestJs*. Con base en los requerimientos se han generado recursos que cuentan con su módulo, el controlador, el servicio y el *DTO*; esta arquitectura que nos proporciona *NestJs* es escalable y nos obliga a seguir su estructura.

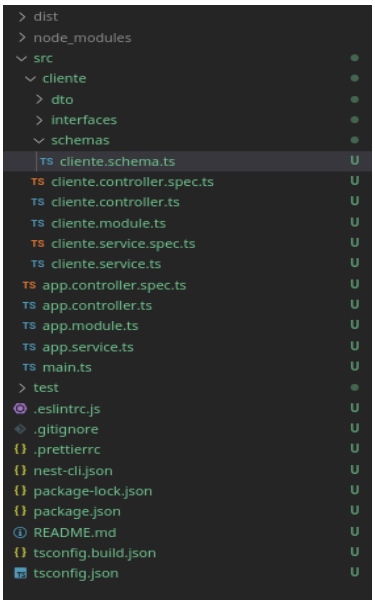


Fig. 7: Estructura inicial del *backend.*

Creación del repositorio en *GitHub*.

Se ha creado el repositorio en *GitHUb* donde se definen 2 ramas iniciales *"rama main"*, es la rama principal que albergará el proyecto al culminar la planificación del *Sprint Backlog*. La rama *"dev"* es donde se ha establecido para realizar *merge* siempre y cuando la tarea abordada cumpla con el requerimiento; es decir, se ha definido que una rama corresponda a una tarea. En la **Fig. 8** se observa la rama *main*; es decir, la estructura inicial del proyecto.

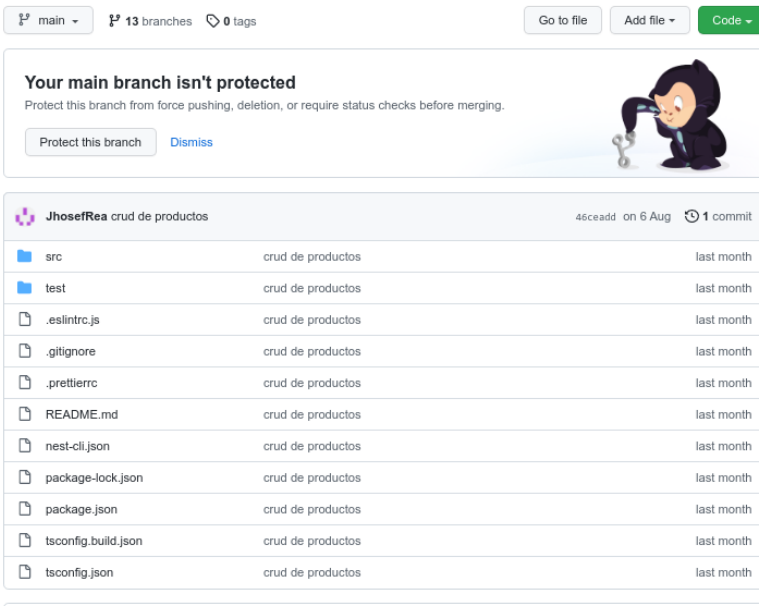


Fig. 8: Repositorios del backend.

Sprint 1. Diseño e implementación del *endpoint* para los usuarios con perfil cliente y administrador.

El *Sprint* 1 corresponde con el inicio de las actividades para la construcción de un *API REST*. Para comenzar, se definió los modelos de Usuarios según su rol. En este *Sprint* se desarrollan las siguientes tareas:

- Generar *endpoints* para el registro y autenticación de usuarios.
- Implementación de roles en el modelo de usuarios.
- Generar *endpoint* de perfil de usuario.

Generar *endpoints* para el registro y autenticación de usuarios.

Para el registro y autenticación de usuarios se han creado las rutas que están definidas en el controlador del recurso *auth*; haciendo uso de los métodos que están definidos en el

servicio de *auth*. Estos componentes permiten tener un control a las peticiones del lado del cliente.

En el controlador se ha creado una ruta pública que define un método *POST* correspondiente al registro; para ingresar la información a través de un formulario y su posterior registro en la base de datos *NoSQL*; específicamente en la colección Usuarios, como se ilustra en la **Fig. 9**. El sistema asigna el rol de "Cliente" a cualquier usuario que se registre; únicamente en la ruta *POST* de usuarios, el administrador puede crear un usuario con cualquier rol disponible.

Seguidamente, se creó el método *POST* para que un usuario, sin importar el rol, se autentique; en la **Fig. 10** se observa la ruta pública de autenticación.

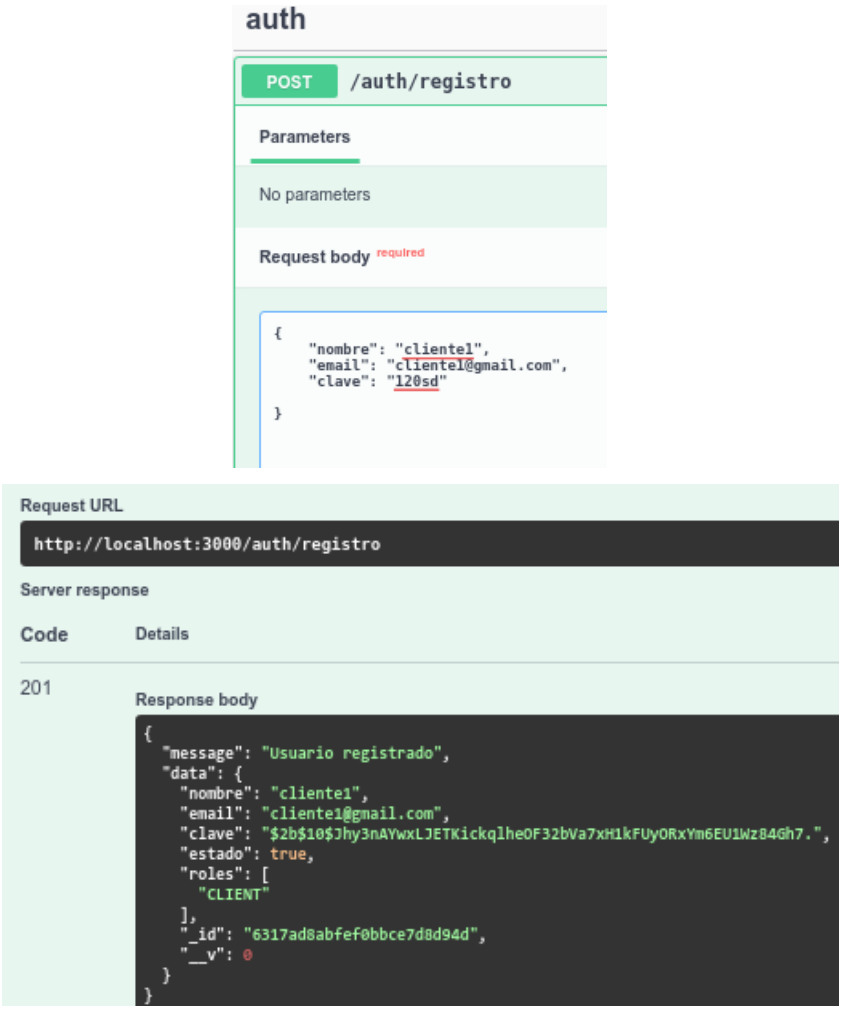


Fig. 9: Método *POST* para registrar usuario.

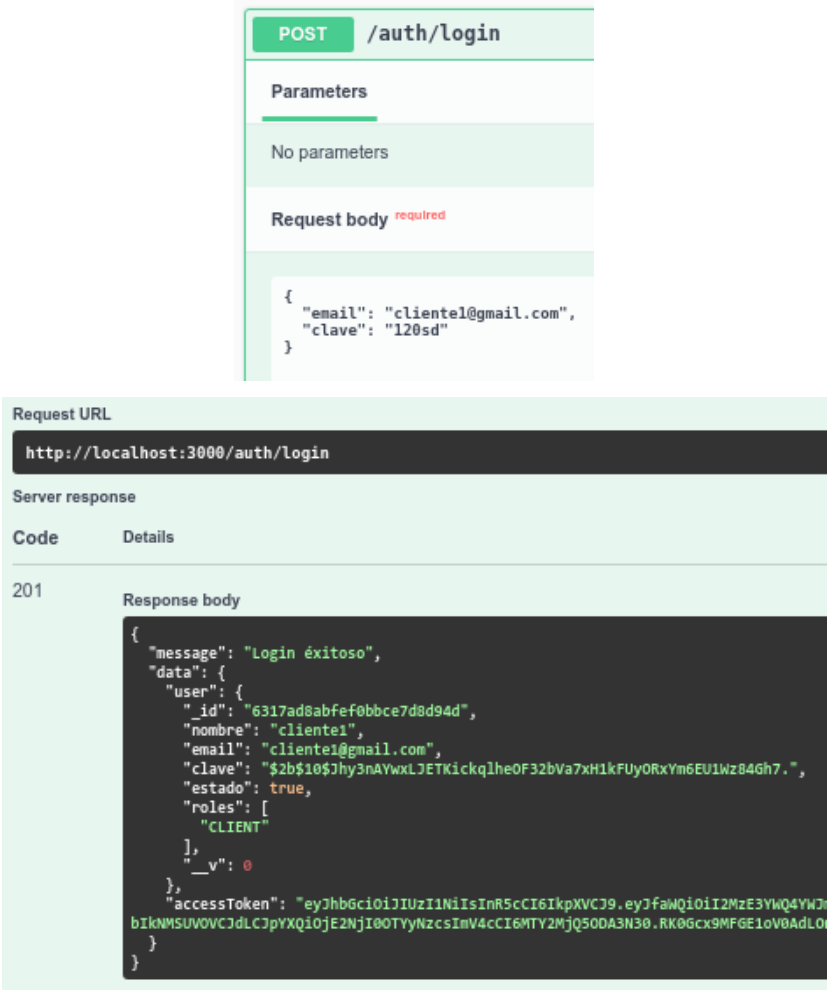


Fig. 10: Método *POST* para autenticar usuario.

Como medio de identificación de acceso de un usuario a los *endpoints* protegidos, cada vez que este se autentique genera su *access token*, mismo que tiene un tiempo establecido de duración. Si el tiempo llegase a ser corto, se ha creado una ruta protegida de tipo *GET* encargada de generar el *refresh token* siempre y cuando el *access token* sea válido. En la siguiente **Fig. 11** se observa el *endpoint* encargado de generar el *refresh token* con la única finalidad de que el usuario autenticado, no vuelva a iniciar sesión.

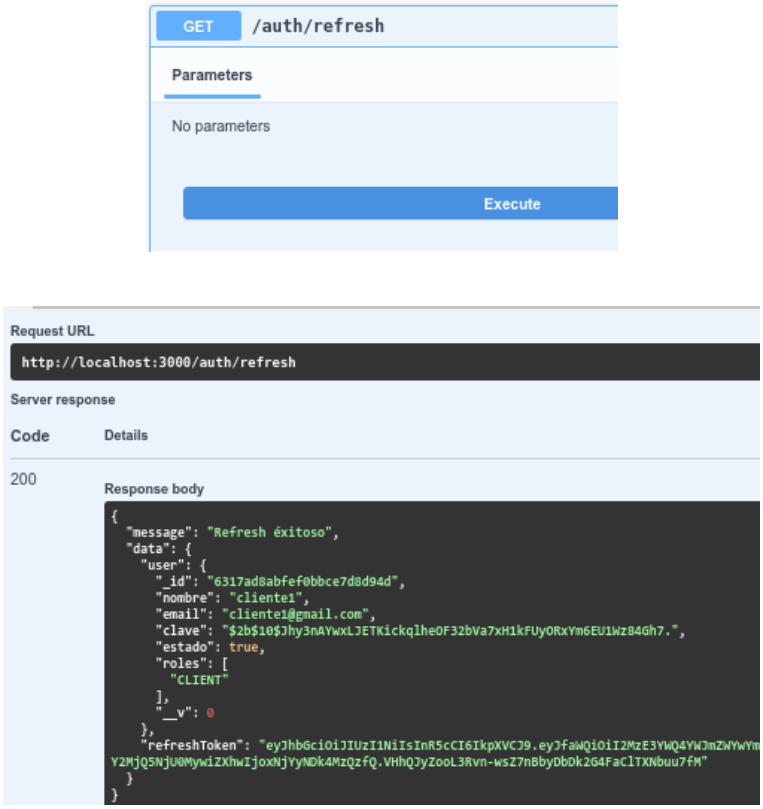


Fig. 11: Método *GET* para obtener un nuevo *token*.

Es importante recalcar que, en ambas rutas públicas, se validan los respectivos campos que envía el cliente, luego, se guarda el registro en *MongoDB*; como se detalla el **ANEXO III** del presente documento.

Generar *endpoint* de perfil de usuario

A través de un método *HTTP GET*, un usuario autenticado puede observar su perfil tal y como se observa en la **Fig. 12.**

Fig. 12: Método *GET* para visualizar perfil de usuario.

Implementación de roles en el modelo de usuarios

Para controlar las acciones de un usuario, es necesario incluir un sistema de roles. En este caso, se hizo uso de la librería *nest-access-control* para establecer los roles *"Admin"* y "Cliente" con sus respectivos permisos; el usuario con rol *"Admin"* o administrador cuenta con todos los privilegios de efectuar cualquier tipo de acción dentro del sistema; hay que mencionar que, a partir de los requerimientos se ha establecido un único usuario con rol *"Admin"*; y corresponde al dueño del negocio, mas, es generado una única vez al arrancar el servidor. Sin embargo, el usuario administrador puede generar otros perfiles con este mismo rol. En cambio, el rol de usuario "Cliente" o cliente posee restricciones en permisos; sin embargo, le permiten interactuar con el sistema web *E-Commerce*. En la **Fig. 13** se muestran los privilegios definidos para cada rol.

```
14  //permisos de usuarios
15  roles
16    .grant(AppRoles.CLIENT)
17    .updateOwn([AppResources.USERS])
18    .deleteOwn([AppResources.USERS])
19    .createOwn([AppResources.CLIENTS])
20
21    .grant(AppRoles.ADMIN)
22    .extend(AppRoles.CLIENT)
23    .createAny([AppResources.USERS, AppResources.PRODUCTS])
24    .updateAny([AppResources.USERS, AppResources.PRODUCTS])
25    .deleteAny([AppResources.USERS, AppResources.PRODUCTS])
26    .createAny([AppResources.CLIENTS]);
27
```

Fig. 13: Privilegios definidos para cada rol de usuario.

Además, se definió el campo roles en el modelo de usuario y de igual forma en su correspondiente *DTO*; como se muestra en la **Fig. 14** y **Fig. 15** respectivamente.

```
@Schema()
export class Usuarios {
  @Prop({ required: true, unique: true })
  nombre: string;

  @Prop({ required: true, unique: true })
  email: string;

  @Prop({ required: true })
  clave: string;

  @Prop({ required: false, default: true })
  estado: boolean;

  @Prop({ required: false })
  roles: string[];
}
```

Fig. 14: Modelo Usuarios.

```
export class CreateUserDto {
  @IsNotEmpty()
  nombre: string;

  @IsNotEmpty()
  @IsEmail()
  email: string;

  @IsNotEmpty()
  @MinLength(3)
  @MaxLength(10)
  clave: string;

  @IsArray()
  @IsEnum(AppRoles, {
    each: true,
    message: `Debe ser un rol válido, ${EnumToString(AppRoles)}`,
  })
  roles: string[];
}
```

Fig. 15: *DTO* del modelo de Usuarios.

Sprint 2. Diseño e Implementación de *endpoints* del modelo de usuarios y productos.

Con base en lo estipulado en el *Sprint Backlog*, se define la gestión en el modelo de usuarios, productos y clientes. Este *Sprint* contiene las siguientes tareas:

- Generar *endpoints* para gestionar los usuarios
- Generar *endpoints* para gestionar los productos
- Generar *endpoints* para gestionar los clientes

Generar un *endpoint* para gestionar los usuarios

Se han creado rutas públicas y protegidas con varios métodos *HTTP* que permiten a los usuarios interactuar con otros usuarios. Las rutas públicas correspondientes a la gestión de usuarios que son de tipo *GET* y corresponden a rutas estáticas y dinámicas; son necesarias para obtener toda la información de un usuario registrado o ya sea añadido por el administrador, como se observa en la **Fig. 16**.

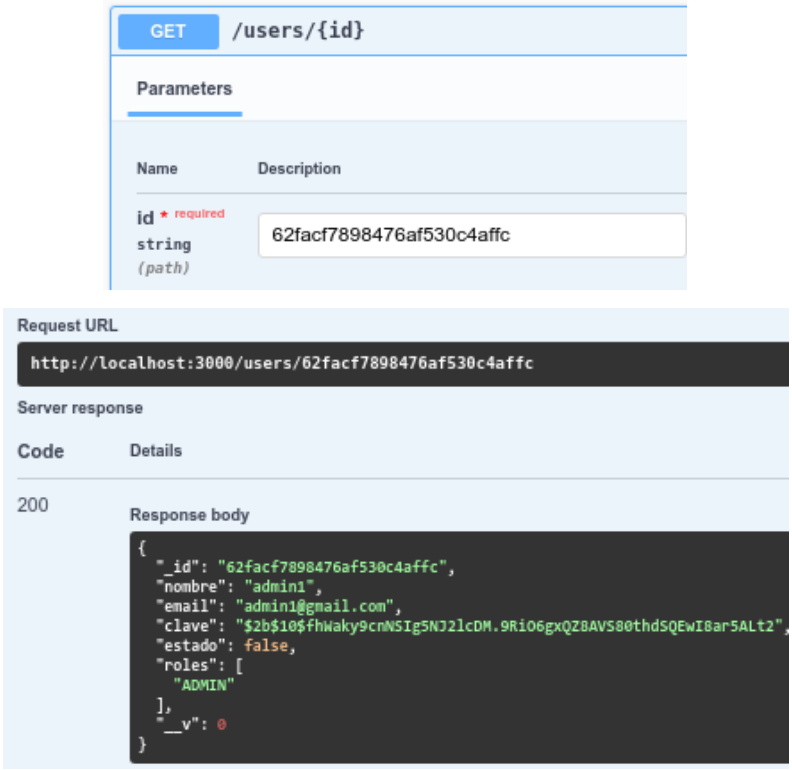


Fig. 16: Método dinámico *GET* de información del usuario.

Por otro lado, las rutas protegidas solo pueden ser accedidas con el rol *"Admin"* y son del tipo *POST* para crear un usuario y añadir el nuevo registro a la colección Usuarios de la base de datos. Los métodos que se explican a continuación, son llamados en rutas dinámicas, *PATCH* para actualizar cualquier usuario y guardar el cambio en el registro y *DELETE* para eliminar un usuario de la base de datos, como se ilustra en la **Fig. 17** y en **Fig. 18** se observa el intento de actualización de los datos de un usuario autenticado que no corresponde a su ID.

Se detalla que, cada campo ingresado es validado antes de ser guardado el registro, al ejecutar un método *HTTP* de tipo *POST* y *PATCH*; de igual forma, el proceso para consumir

la información en lado del cliente y de la base de datos, se puede visualizar en el **ANEXO III** del presente documento.

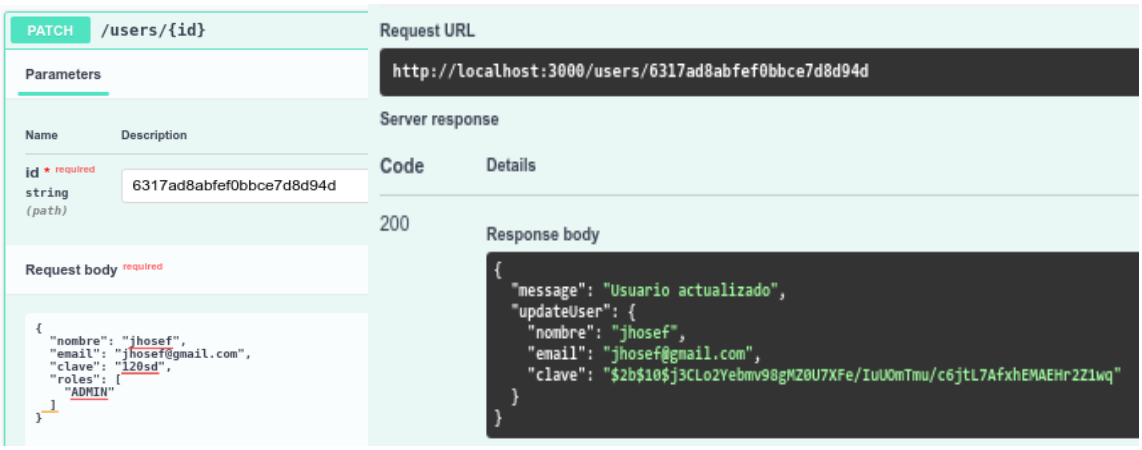


Fig. 17: Método *PATCH* actualizar información de usuario.

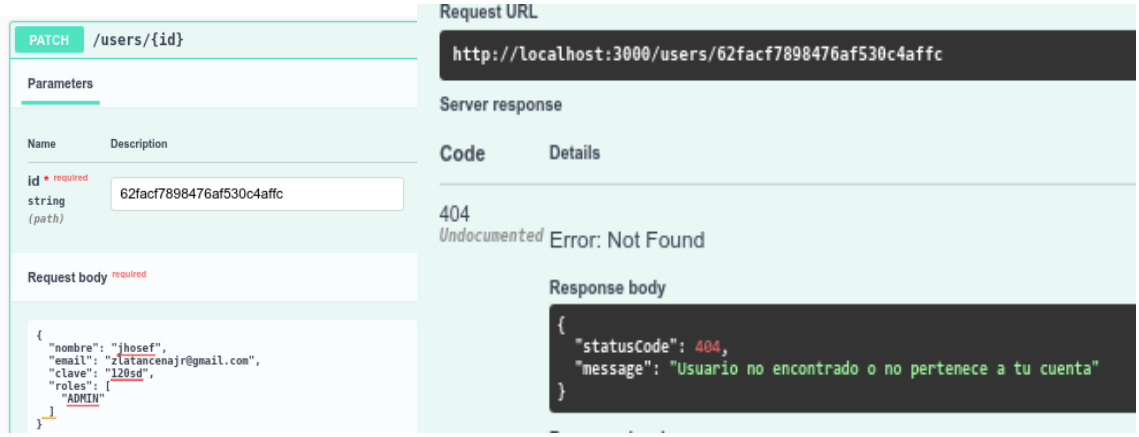


Fig. 18: Método *PATCH* 404.

Generar *endpoints* para gestionar los productos

Se han creado rutas públicas y protegidas con varios métodos *HTTP* que permiten a los usuarios interactuar con los productos. Las rutas públicas correspondientes a la gestión de productos que son de tipo *GET* y corresponden a rutas estáticas y dinámicas; son necesarias para obtener toda la información de un producto registrado por el administrador, como se observa en la **Fig. 19**.

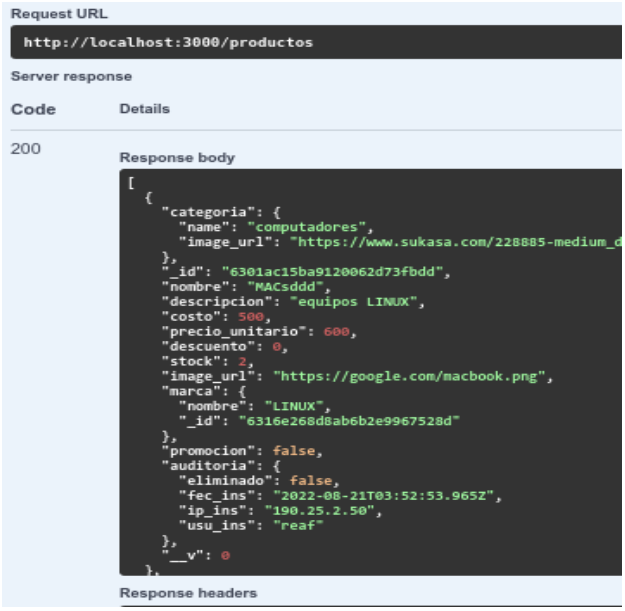
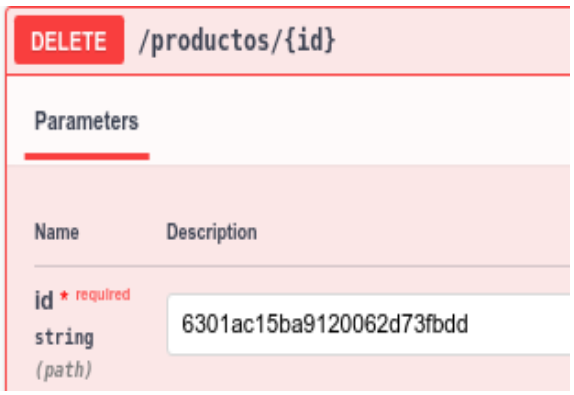


Fig. 19: Método *GET* para visualizar productos.

Por otro lado, las rutas protegidas solo pueden ser accedidas con el rol *"Admin"* y son del tipo *POST* para crear un producto, a su vez se añade el nuevo registro en la colección Productos de la base de datos. Los métodos que se explican a continuación son llamados en rutas dinámicas, *PATCH* para editar un producto y actualizar el registro; *DELETE* para eliminar un producto de la base de datos, como se ilustra en la **Fig. 20**.

Se detalla que, cada campo ingresado es validado antes de ser guardado el registro, al ejecutar un método *HTTP* de tipo *POST* y *PATCH*; de igual forma, el proceso para consumir la información en lado del cliente y de la base de datos, se puede visualizar en el **ANEXO III** del presente documento.



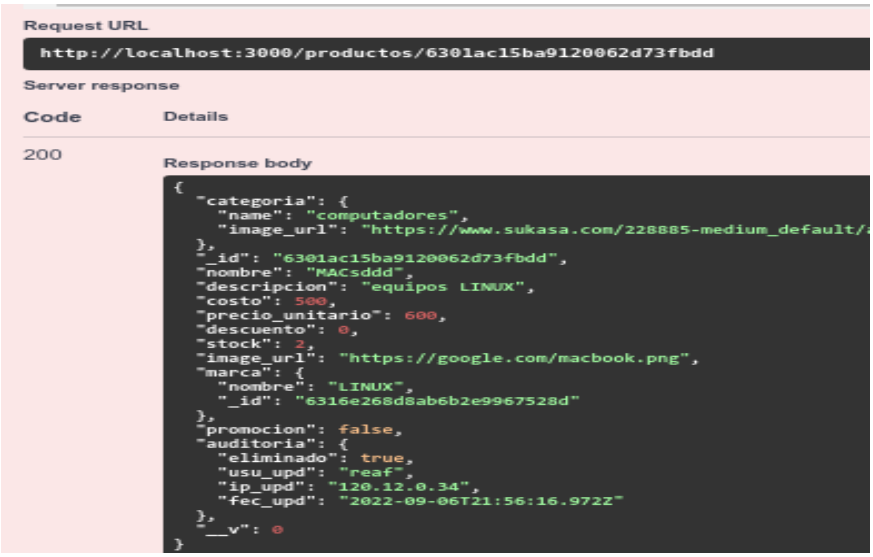


Fig. 20: Método *DELETE* de un producto

Generar *endpoints* para gestionar la información personal del cliente

Un usuario autenticado es llamado cliente y su registro se almacena en la colección Usuarios. Cuando un usuario, sin importar su rol, quiere generar un carrito de compras, primero tiene que existir en la colección Clientes, pues, es necesario identificar quién quiere añadir productos y para su posterior efectúo de una compra. Para ello, el usuario tiene que ingresar sus datos personales, pues, son necesarios para generar una factura electrónica; por esta razón, se ha definido el *endpoint* de clientes previo al endpoint de carrito de compras.

Se han creado rutas protegidas con varios métodos *HTTP*, como la ruta estática *POST* que permite al usuario con perfil *"Admin"* ingresar los datos personales del usuario que quiere interactuar con el carrito de compras, esta ruta es accesible únicamente por el usuario administrador. Es importante destacar que, el registro de datos personales del usuario se da una única vez; como en esta colección se almacena el carrito de compras, entonces, una vez generada la factura, automáticamente se vacía este campo de tipo *array* de objetos; sin embargo, se mantiene la información personal del cliente. En cambio, las rutas dinámicas protegidas con métodos *GET*, *PATCH* y *DELETE* son accesibles por el administrador y son necesarias para obtener la información correspondiente a datos personales del usuario que quiere generar un carrito de compras. A continuación, se observa en la **Fig. 21** el resultado de que un usuario registre sus datos personales.

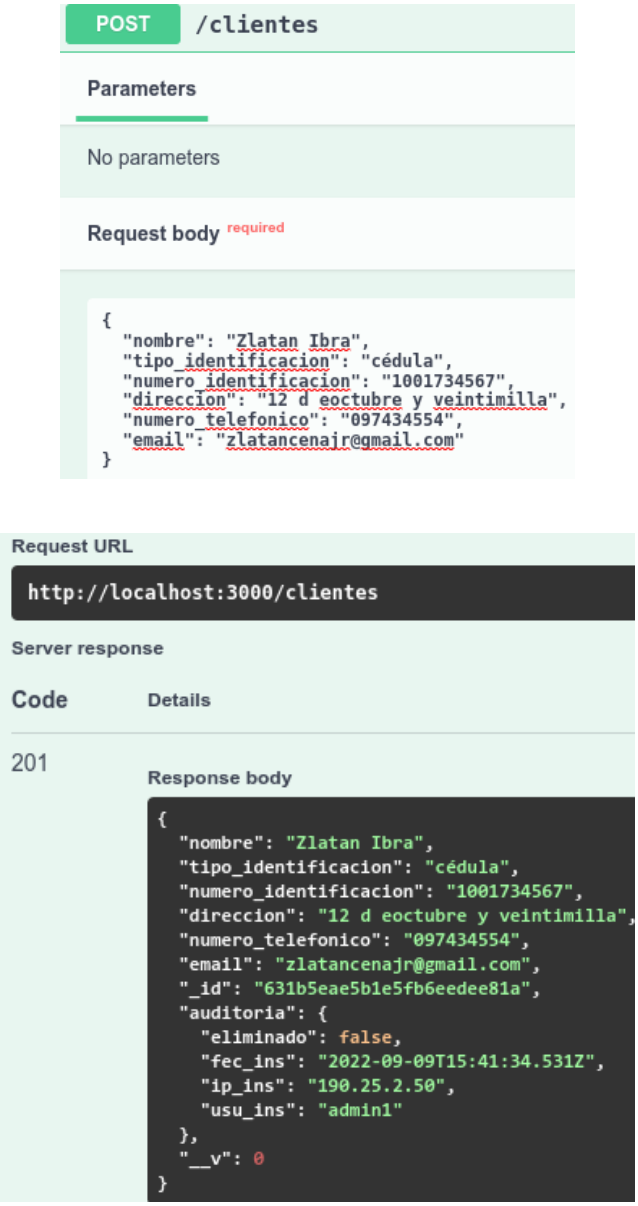


Fig. 21: Método *POST* registro datos personales del usuario.

Además, al ejecutar un método *HTTP* de tipo *POST* y *PATCH* cada campo ingresado es validado antes de ser guardado el registro; de igual forma, el proceso para consumir la información en lado del cliente y de la base de datos, se puede visualizar en el **ANEXO III** del presente documento.

Sprint 3. Diseño e Implementación de *endpoints* del modelo de carrito de compras y modelo de facturación electrónica.

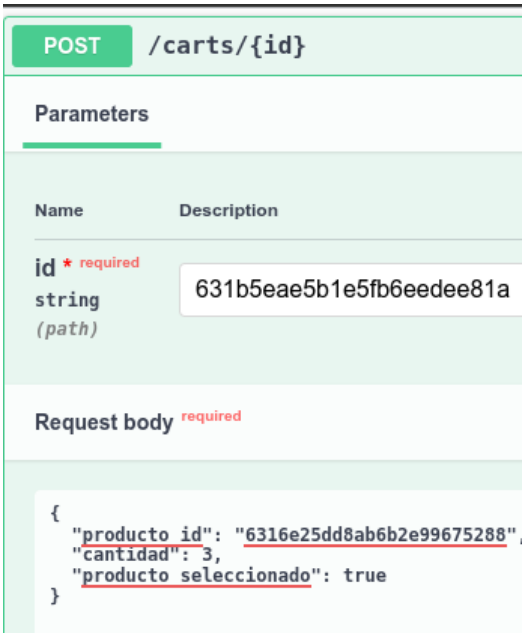
Con base en lo estipulado en el *Sprint Backlog*, este *Sprint* contiene las siguientes tareas:

- Generar *endpoint* para operatividad del carrito de compras
- Generar *endpoints* para gestionar la tienda
- Generar *endpoints* para gestionar la facturación electrónica

Generar *endpoint* para operatividad del carrito de compras

Una vez que los registros estén en la colección Clientes entonces, el usuario puede utilizar el carrito de compras; para ello el usuario administrador o dueño de la tienda, es quién registra al usuario que desee interactuar con el carrito de compras; como se explica en el ítem anterior, este registro se da una única vez, pues, queda almacenado en la colección Clientes y el campo correspondiente al carrito de compras queda vacío siempre y cuando el usuario genere la factura electrónica o si no la llega a generar, pues se vacía una vez que el *access token* o el *refresh token* es inválido; es decir, ha caducado el tiempo de duración del token.

Por medio del *backend* se ha creado una ruta pública que permite al usuario con perfil "Cliente" y *"Admin"* interactuar con el carrito de compras. Un usuario puede añadir productos; estos productos son almacenados en un campo de tipo *array* de objetos, de una colección de tipo Clientes. A continuación, se observa en la **Fig. 22** el resultado de que un usuario añada productos a su carrito de compras, a través de método *POST*.



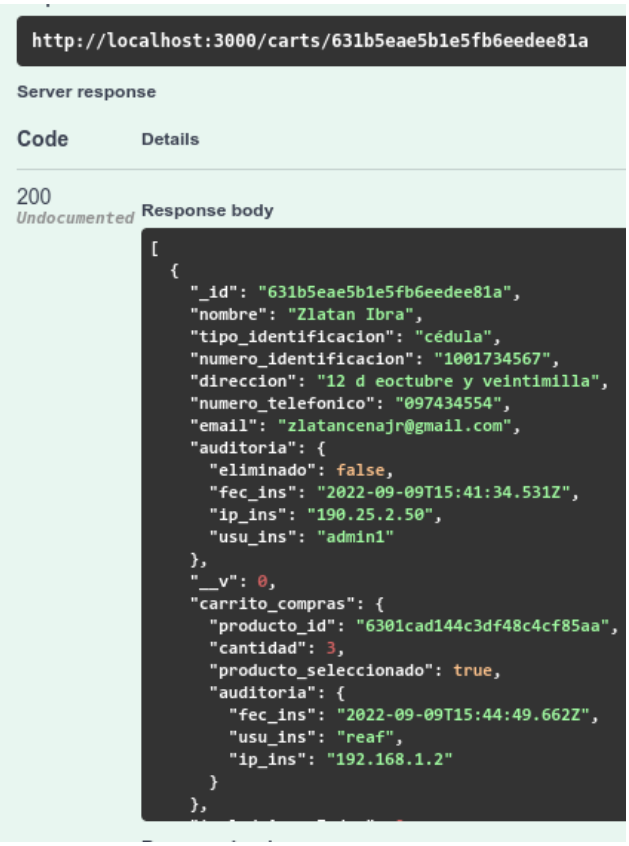


Fig. 22: Método *POST* registro de carrito de compras.

Además, cada campo es validado conforme al tipo de dato que debe tener la entrada, de igual forma, se validan que sean campos que existan en el *eschema* de la colección. Tal y como se ha hecho con previos ejemplos de peticiones de tipo *POST* y *PATCH* expuestos en el presente documento. Las validaciones respectivas se pueden apreciar en el **ANEXO III**.

Generar *endpoints* para gestionar la tienda

Basándose en el *Sprint Backlog*, se definen las rutas protegidas para registrar la información del negocio; en efecto, las rutas de tipo *POST*, *PATCH* y *DELETE* son accedidas únicamente por el administrador. Estos datos son necesarios para generar la factura electrónica. A continuación, se observa el resultado del ingreso de la información de la tienda en **Fig. 23**.

Fig. 23: Método *POST* registro de tienda.

Además, el proceso para realizar tanto el consumo de la información de una tienda como sus respectivas validaciones, se puede apreciar en el **ANEXO III** del presente documento.

Generar *endpoints* para gestionar la facturación electrónica

Se han creado tres rutas protegidas y una ruta pública, esta última corresponde al *endpoint* de tipo *POST* porque, puede ser accedido con cualquier perfil, pues, cualquier usuario para generar la factura electrónica que se corresponde al cliente que genera el carrito de compras. En este sentido, un usuario que no esté registrado en la colección Clientes, no puede generar la factura; pues, en la ruta *POST* se recibe el ID del Cliente en los *params* del *endpoint* y este es buscado en la colección Clientes. En *POST*, también se permite a los usuarios establecer su compra bajo la intervención del administrador.

La factura electrónica generada consta de la información de la tienda, los datos personales del cliente y un *array* de objetos de los productos seleccionados; como se observa en **Fig. 24**.

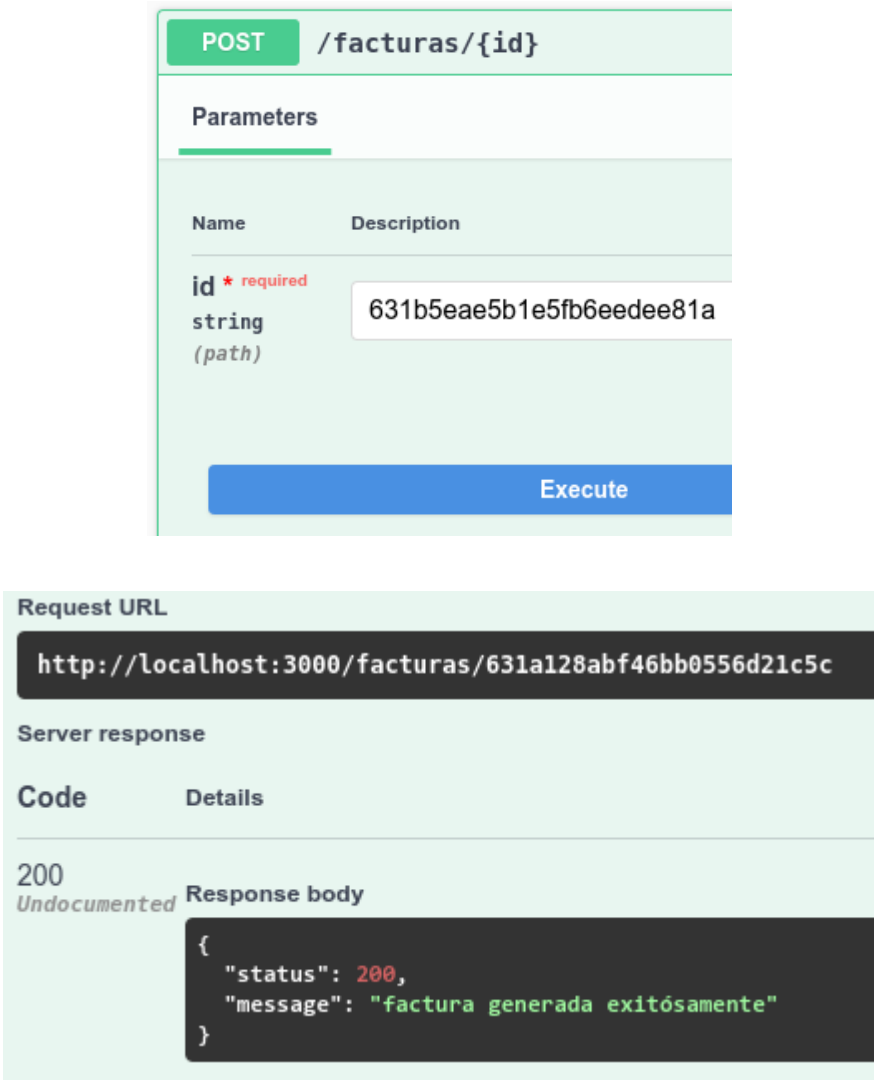


Fig. 24: Método *POST* registro de factura.

Las dos rutas protegidas de tipo *GET* son, una estática y una dinámica, únicamente accedida por el usuario con rol *"Admin"* y corresponden a la consulta de facturas; pues, es un requisito consultar toda la información de la/s factura/s generadas. De igual forma, la ruta protegida con método *HTTP DELETE* es accedida con perfil *"Admin"* y es la encargada de anular una factura electrónica. Es importante aclarar que, una factura no debe ser actualizada pues, una factura emitida, no puede ser modificada; tampoco se ha definido un

endpoint para eliminar una factura electrónica porque el sistema se ajusta a la normativa del SRI.

En la **Fig. 25** se observa la consulta de una factura electrónica donde se recibe el número de factura del *header*. Además, el proceso para realizar tanto el consumo de la información de una tienda como sus respectivas validaciones, se puede apreciar en el **ANEXO III** del presente documento.

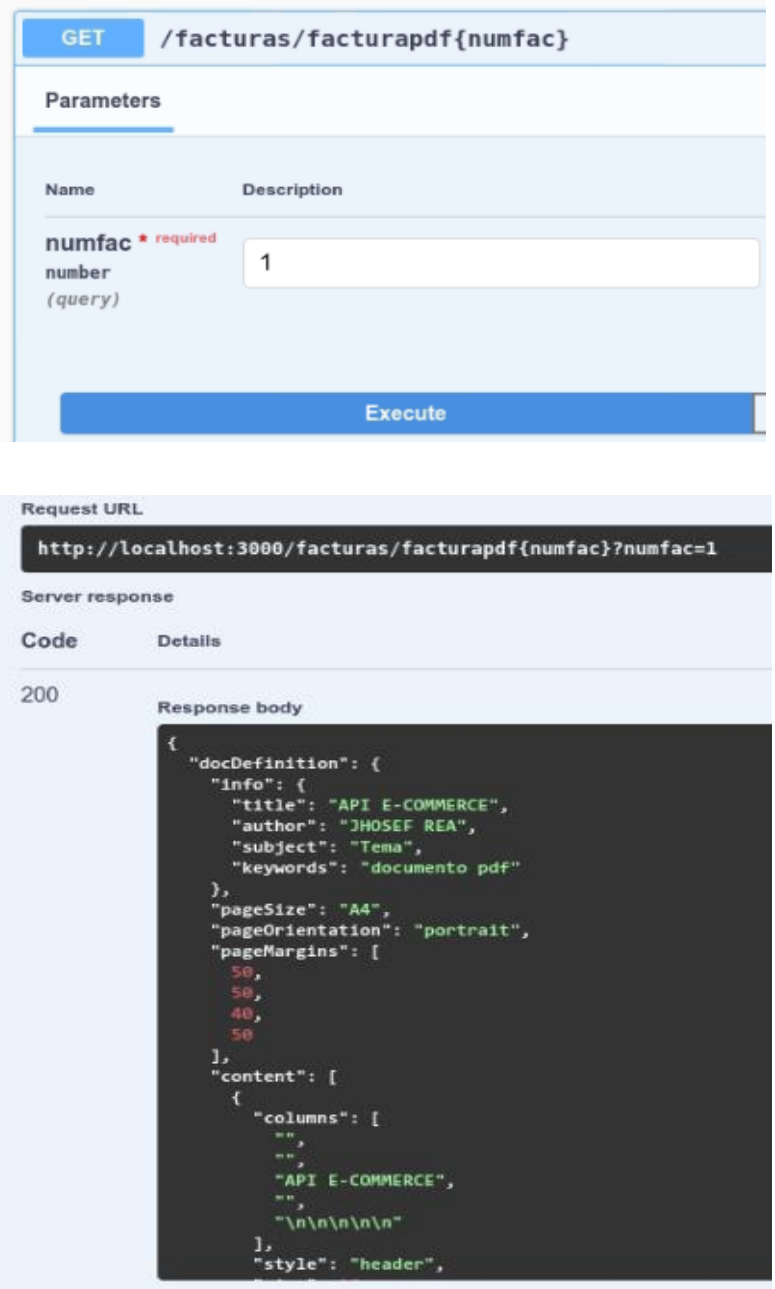


Fig. 25: Método *GET* para obtener una factura.

En la presente imagen, se presenta la colección factura y cuenta con el objeto que corresponde a la factura electrónica que genera el usuario administrador o el usuario con perfil cliente, como se observa en la **Fig. 26**.

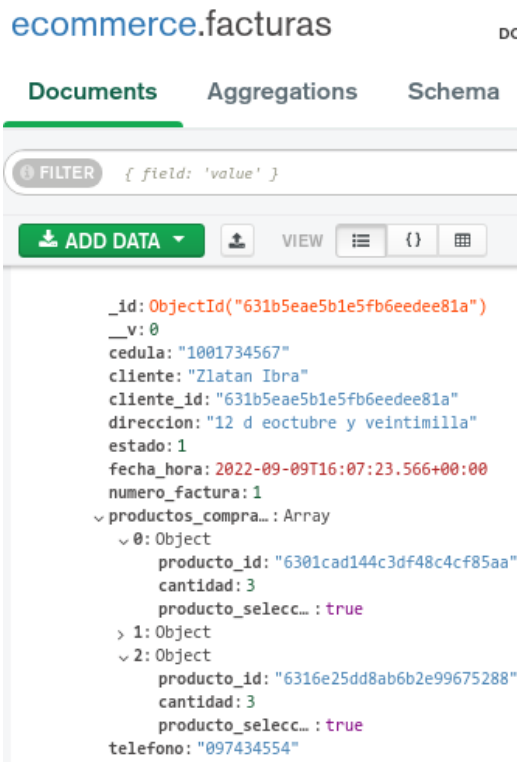


Fig. 26: Colección de factura electrónica.

Para poder visualizar el documento de facturación electrónica, se construyó un pequeño *frontend* con *Ionic* y *Angular;* estos *frameworks* no forman parte de las herramientas de desarrollo; pues, únicamente se adoptaron para poder visualizar la estructura del documento PDF generado. El *frontend* mantiene comunicación directa con el *API REST* y este a su vez con el *backend*; pues, la estructura del PDF es generado desde el *backend*. A continuación, en la **Fig. 27** se muestra la vista del *endpoint* encargado de descargar la factura electrónica, previamente generada en el *backend* por el usuario. La ruta es protegida porque únicamente el usuario con perfil administrador puede consultar las facturas para que sean enviadas al respectivo correo electrónico del cliente. Para ello, se ejecuta a través de un método *GET* que envía en los *headers* el número de factura que es ingresado en el input.

Además, en la sección **ANEXO II** se puede observar la estructura MVC definida en *Ionic* con angular; utilizada para tomar la respuesta del *backend* frente a la petición del *frontend* en la URL "*localhost:8100/home*".

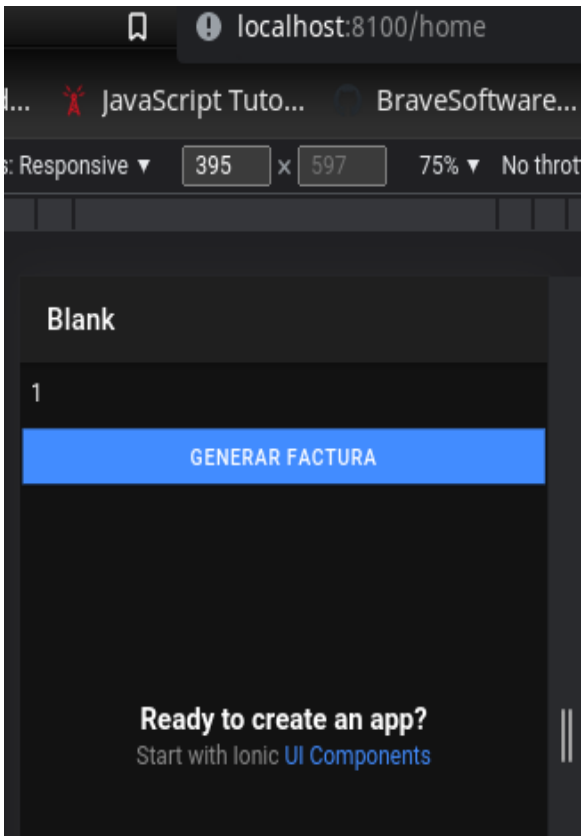


Fig. 27: *Frontend* de generar factura

En la presente imagen, **Fig. 28**, se observa la estructura *HTML* del PDF obtenido desde el *frontend*; donde se destaca, el código QR que se encarga de dar seriedad, autenticidad y validez al documento emitido por el sistema del negocio; los datos del negocio; los datos del cliente y por último el detalle de la compra. A continuación, en la **Fig. 29** se muestra un ejemplo del código QR generado para que pueda ser escaneado por el lector.

Por último, se ha escaneado el código QR desde una aplicación de tercero para Android. En la **Fig. 30** se observa los datos que se han establecido desde el recurso factura dónde consta: del nombre de la institución de tercer nivel, el nombre del proyecto, el número de la factura, la fecha, el RUC de la tienda y el correo electrónico del dueño de la tienda.

Una explicación más detallada es expuesta en el **ANEXO III** del presente documento.

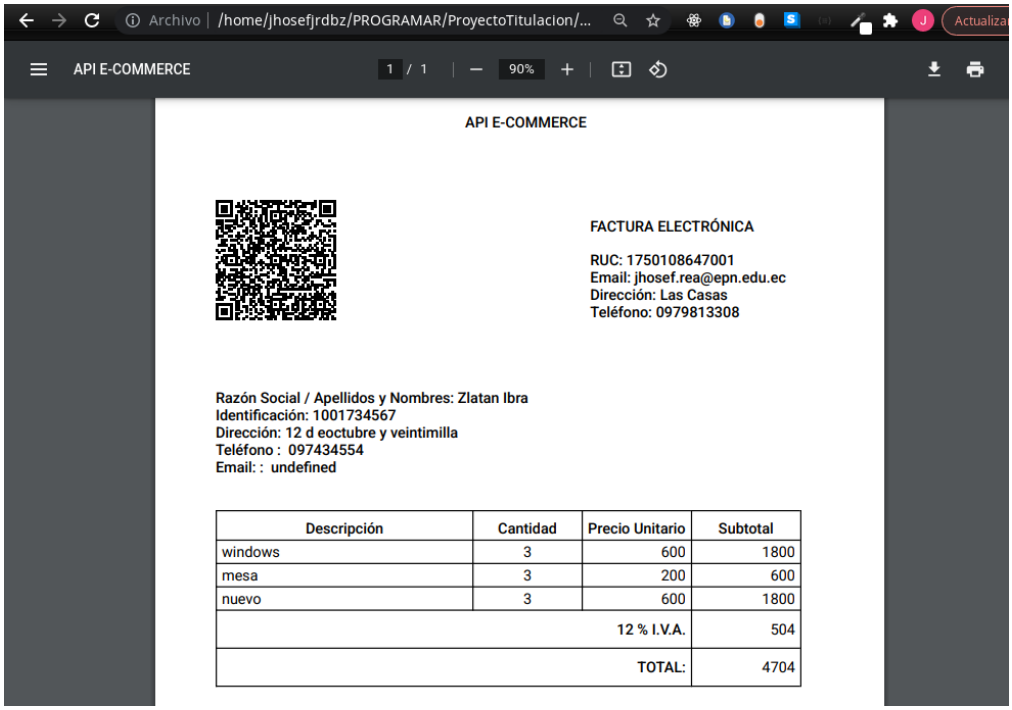


Fig. 28: Vista de la factura electrónica generada.

Fig. 29: Código QR de la factura electrónica generada.

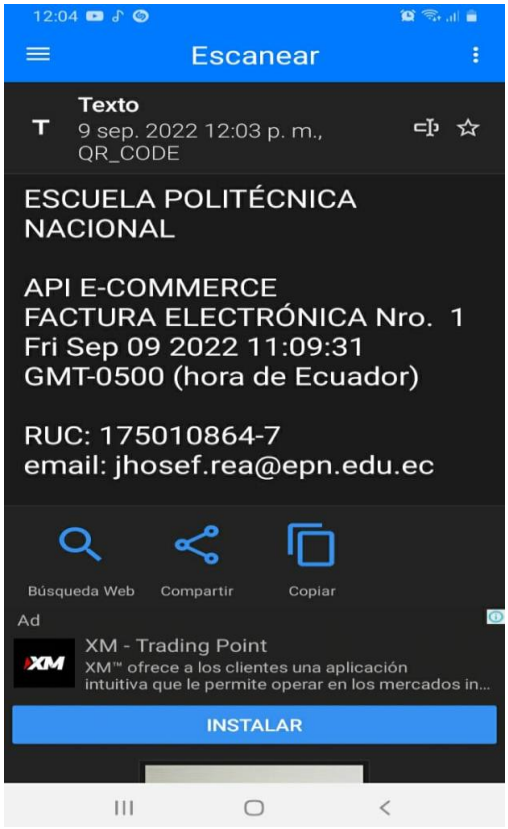


Fig. 30: Resultado en app de lector de códigos QR.

Sprint 4. Diseño e implementación de un *endpoint/s* de envío de correo electrónico y de Herramienta de búsqueda.

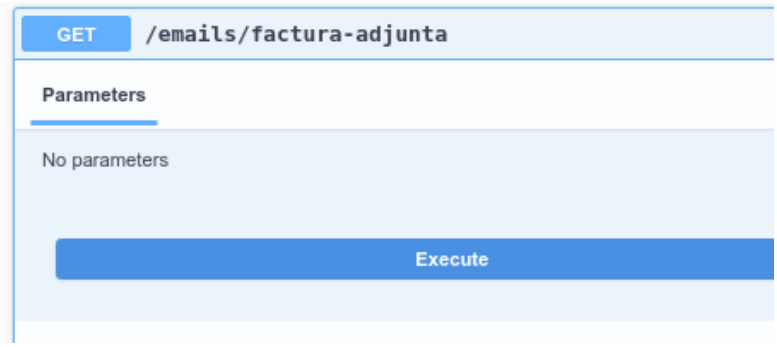
Con base a lo estipulado en el *Sprint Backlog* este *Sprint* contiene las siguientes tareas:

- Generar un *endpoint* de envío de correo electrónico.
- Generar *endpoints* para interactuar con la herramienta de búsqueda.

Generar un *endpoint* de envío de correo electrónico.

Como se especifica en los requerimientos, se necesita de un servicio de correo electrónico para poder adjuntar el documento electrónico y ser emitido al correo del usuario que ha finalizado una compra. En este sentido, se define una ruta protegida de tipo *GET* que es accedida únicamente por el usuario con rol *"Admin"*; por lo expuesto, esta ruta se encarga de enviar la factura electrónica al correo que ha sido registrado dentro de los datos personales del usuario; es decir, los datos de la colección Clientes.

A continuación, se observa la respuesta que envía el servidor al cliente al efectuarse el envío de correo electrónico **Fig. 31**.



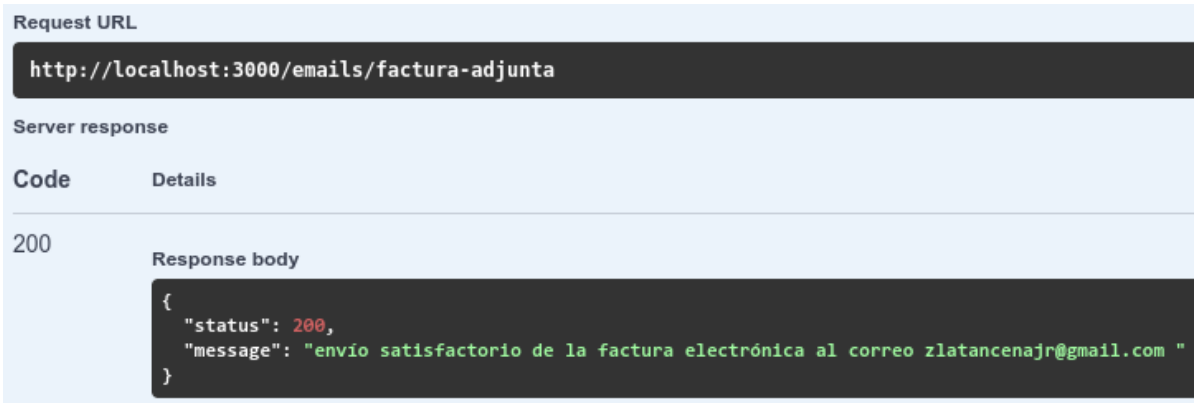


Fig. 31: Método *GET* envió de factura electrónica vía e-mail.

Seguidamente, se verifica la recepción del correo electrónico adjuntado la factura electrónica, tal y como se observa en las figuras **Fig. 32** y **Fig. 33**. Cabe destacar que, si un correo ingresado en la colección Clientes resulta ser inválido, entonces se da a conocer dicha advertencia en el correo electrónico del usuario administrador; lo expuesto es visualizado en la **Fig. 35**.

Por otra parte, el proceso para de envío y recepción de correo electrónico es explicado de mejor manera en el **ANEXO III** del presente documento.

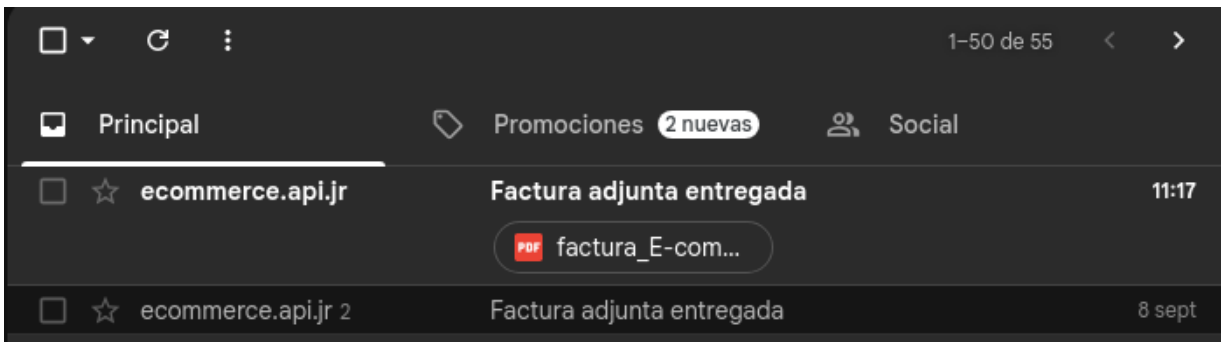


Fig. 32: Verificación de recepción de factura al correo electrónico.

Fig. 33: Detalle de la recepción del correo electrónico.

Finalmente, en la **Fig. 34** se verifica que una vez generada la factura electrónica, el sistema vacía la propiedad “carrito_compras” del cliente; pues, se adopta la lógica de negocio de un sistema *E-Commerce*.

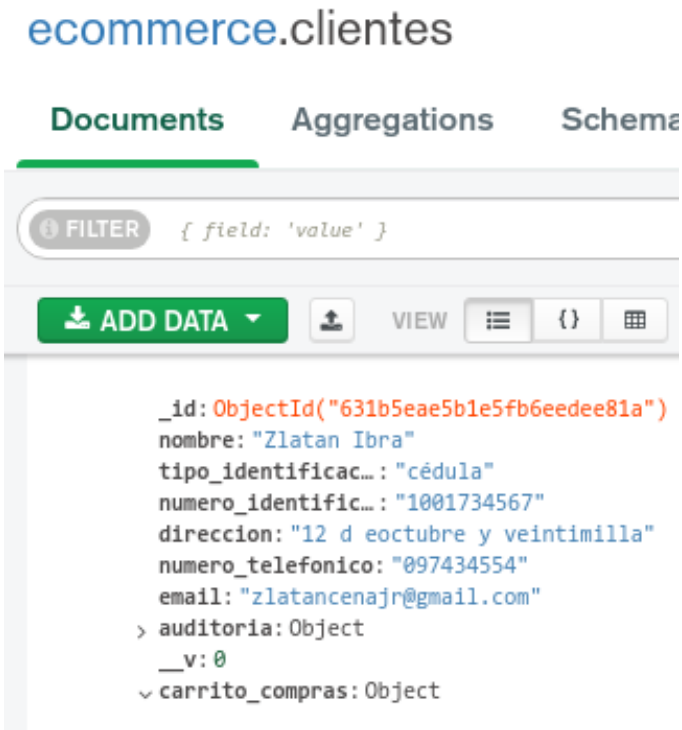


Fig. 34: Colección Clientes luego de efectuar una factura electrónica.

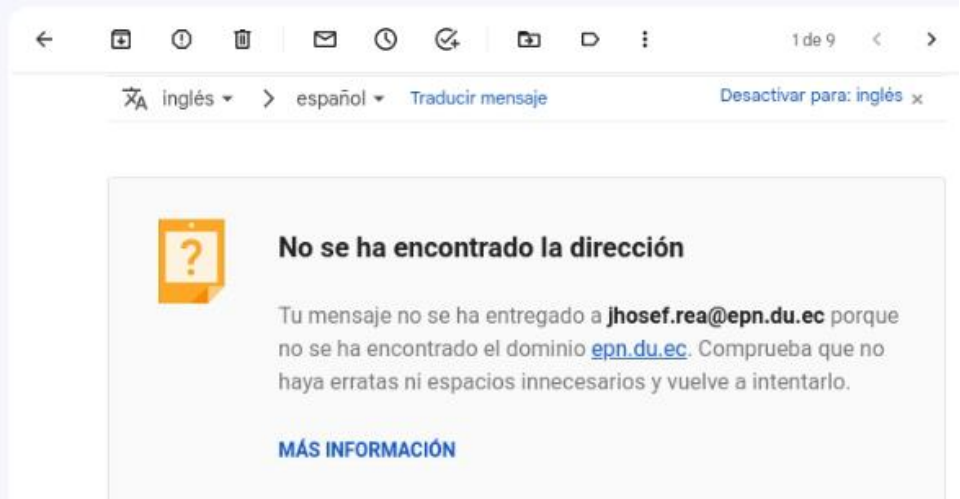


Fig. 35: Advertencia de correo inválido.

Generar *endpoints* para interactuar con la herramienta de búsqueda.

De acuerdo, al análisis de los requerimientos, se necesita la operatividad de una herramienta de búsqueda sobre los productos del negocio. Para dar funcionalidad, se han creado tres rutas públicas de tipo *GET*; donde una de las rutas recibe los *params* que corresponde a un número para obtener el número total de productos que existen en la colección Productos. Otra de las rutas, recibe el *query* y el *value* del *endpoint*; realizando una búsqueda por clasificación que corresponde a: búsqueda por promoción, precio, nombre y categoría del producto. El último *endpoint* realiza una búsqueda general; es decir, toma el *param* de la cabecera y como respuesta al cliente se obtienen todos los productos posibles, cuyo valor de cualquier propiedad, se asemejan al parámetro tal y como se observa en la **Fig. 36**.

Por otra parte, la operatividad de la herramienta de búsqueda se puede apreciar de mejor manera en el **ANEXO III** del presente documento.

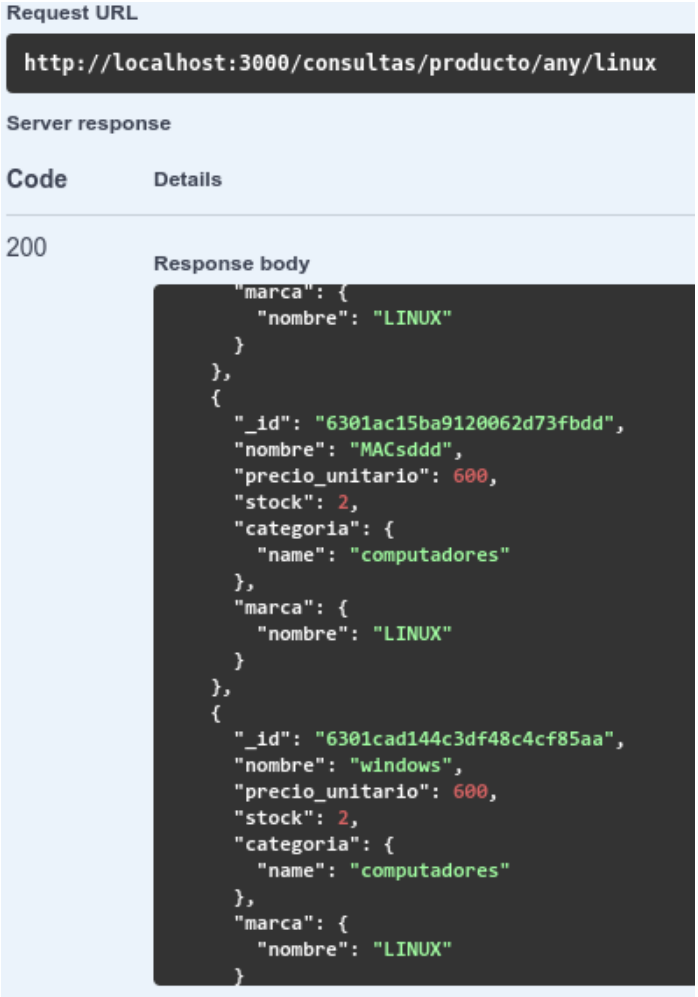


Fig. 36: Método GET herramienta de búsqueda.

Sprint 5. Pruebas en el *backend*.

Luego de haber concluido cada *Sprint* correspondiente a codificación de cada *endpoint* en el *backend*; se realizan las actividades de pruebas unitarias y de rendimiento junto a sus resultados tal y como se estipula en el *Sprint Backlog.* Este *Sprint* contiene las siguientes tareas:

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

Ejecución de pruebas unitarias y resultados.

Con la culminación de la codificación cumplida gracias al cronograma establecido en el Sprint *Backlog* y siguiendo la planificación establecida; en esta sección se procede a realizar las pruebas unitarias al *backend*; es decir, se evalúa los métodos definidos en los

endpoints del *API REST*. En ese sentido, se hace uso de *nestjs/Testing*, misma que es una herramienta que permite realizar pruebas unitarias completas. En la **Fig. 37** se muestra un parte del código que corresponde al inicio de sesión del usuario y en la **Fig. 38** se puede apreciar el resultado tras haber realizado la respectiva prueba. Las pruebas unitarias son detalladas en ejecución y resultados en el **ANEXO II** del presente documento.

```
describe('UsersController', () => {
  let controller: UsersController;

  beforeEach(async () => {
    const module: TestingModule = await Test.createTestingModule({
      imports: [UsersModule, AccessControlModule.forRoles(roles)],
      controllers: [UsersController],
      providers: [
        UsersService,
        {
          provide: getModelToken(Usuarios.name),
          useValue: { Symbol: jest.fn() },
        },
      ],
    })
      .overrideProvider(getModelToken(Usuarios.name))
      .useValue(jest.fn())
      .compile();

    controller = module.get<UsersController>(UsersController);
  });

  it('should be defined', () => {
    expect(controller).toBeDefined();
  });
});
```

Fig. 37: Código de prueba al controlador de usuarios.

```
OUTPUT   TERMINAL   COMMENTS   PROBLEMS 1

 PASS  src/users/__tests__/users.controller.spec.ts (10.188 s)
  UsersController
    ✓ should be defined (29 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        10.828 s
Ran all test suites matching /users.controller.spec.ts/i.

Watch Usage: Press w to show more.
```

Fig. 38: Resultado exitoso de la prueba.

Ejecución de pruebas de rendimiento y resultados.

Se ha efectuado pruebas de compatibilidad en clientes *HTTP*, los cuales son: *Swagger* y un *Postman* han permitido verificar cada respuesta que envía el *backend* hacia el cliente como, validaciones, estado de la respuesta e incluso el contenido se presenta de forma estructurada y correcta en la invocación en los *endpoints* ya sean públicos o protegidos. En ese sentido, en la **TABLA IV** se visualiza los clientes *HTTP* usados para la ejecución de las respectivas pruebas; en el **ANEXO II** del presente documento se da a conocer el detalle completo de la ejecución con resultados.

Con base a los resultados obtenidos, se determina que el backend E-Commerce no presenta fallos en tiempos de respuesta y presentación de la información antes las peticiones respectivas al cliente HTTP.

TABLA IV: Clientes HTTP para la prueba de compatibilidad

NOMBRE	VERSIÓN
Swagger	V 1.0
Postman	V 9.0.8

4 CONCLUSIONES

- Se concluye que, previo a la codificación, se requiere de un análisis de requerimientos obtenidos del problema planteado. Hay que tener en cuenta que, basarse en una arquitectura estructurada y escalable solventa los nuevos requerimientos que pueden surgir a futuro e incluso de forma inesperada, he ahí el hecho de optar por el *framework Nest.Js*.
- El uso de la metodología ágil *Scrum* permite que el proyecto presente un avance organizado en las tareas definidas. Durante el desarrollo se presentaron varios cambios que fueron incluidos, sin alterar el flujo de procesos; de hecho, el valor del producto entregado, es la evidencia de lo expuesto.
- Los objetivos propuestos definieron el progreso en los *sprints*, lo que conlleva, a las características que describen el *API REST Ecommerce*; de esta forma se proporciona la lógica que necesita el *frontend*.
- Una arquitectura Modelo-Vista-Controlador es necesaria en el desarrollo del *backend*; pues, permite estructurar la lógica en cada uno de los *endpoints*, siendo de fácil entendimiento para cualquier desarrollador; es decir tanto para los que codificaron como para los lectores.
- Un *Ecommerce* necesita de una arquitectura basada en microservicios. Se considera que, el uso de una base de datos *NoSQL*, ha permitido que la gestión de la información sea de una forma más organizada al igual que su respectivo acceso a la data; ya que no existe “relaciones” ni una “estructura inicial” en el modelamiento o definición de la base de datos ni en los esquemas de una colección.

5 RECOMENDACIONES

- Se debe tener en cuenta, un conocimiento y concepto totalmente claro de una base datos NoSQL en definición y validación, de campos, tipos de datos, referencias entre colecciones; pues, junto a la lógica de programación, se debe comprobar que dichos modelos trabajen de forma eficaz para que el API REST cumpla con los requerimientos del cliente.
- Se recomienda definir en los primeros Sprints los roles de usuarios, privilegios y acceso a endpoints para priorizar la seguridad que debe proporcionar un API REST, ya que debe haber un control cuando estos ingresen a la base de datos.
- Debe realizarse controles parciales programados, se recomienda tener respaldos de la base de datos, con el objetivo de salvaguardar la información necesaria para no tener problemas futuros.
- En un proyecto de desarrollo de software es importante basarse en la documentación oficial de cada herramienta usada, en este caso, para la construcción de un API REST se ha solventado los errores o dificultades que surgieron durante los Sprints.

6 REFERENCIAS BIBLIOGRÁFICAS

[1] M. A. Jaramillo Paredes, « El derecho humano al acceso a Internet,» Universidad Andina Simón Bolívar (pp. 65-69). Quito, 2020. [En línea]. Available: http://hdl.handle.net/10644/7563 .

[2] P. Tello, L. Pineda, « Análisis del Comercio Electrónico en Ecuador,» Universidad Internacional del Ecuador (pp. 26-35). Quito, 2017. [En línea]. Available: https://repositorio.uide.edu.ec/handle/37000/2476 .

[3] J. Maldonado, « Comercio Electrónico Ideas fundamentales,» Gestiopolis. Quito, 24 Marzo 2017. [En línea]. Available: https://www.gestiopolis.com/comercio-electronico-ideas-fundamentales/

[4] M. Cordero, « El comercio electrónico *e-commerce*, análisis actual desde la perspectiva del consumidor en la ciudad de Guayaquil, provincia del Guayas y estrategias efectivas para su desarrollo,» Universidad Católica de Santiago de Guayaquil (pp. 39). Guayaquil, 19 Noviembre 2019. [En línea]. Available: http://repositorio.ucsg.edu.ec/handle/3317/14064 .

[5] S. Vinueza, V. Simbaña, « Tecnologías de Información y Comunicación (TIC) en la matriz productiva,» Universidad Central del Ecuador (pp. 413). Quito, 2017. [En línea]. Available: https://dialnet.unirioja.es/ejemplar/468070 .

[6] M. Cortés, M. Iglesia, « Generalidades sobre Metodología de la Investigación,» Universidad Autónoma del Carmen (pp. 8). México, 2004. [En línea]. Available: http://www.unacar.mx/contenido/gaceta/ediciones/ediciones.html .

[7] E. Gallardo , « Metodología de la Investigación,» Universidad Continental. Huancayo-Perú, 2017. [En línea]. Available: https://hdl.handle.net/20.500.12394/4278 .

[8] A. Navarro, J. Fernández, J. Morales, « Revisión de metodologías ágiles para el desarrollo de software,» Universidad Autónoma del Caribe Colombia (pp. 30-31). Colombia, 2013. [En línea]. Available: https://www.redalyc.org/articulo.oa?id=496250736004 .

[9] J. Aranda, « Fortalecimiento del *Frontend* y *Backend* del sitio web,» Universidad Distrital Francisco José De Caldas (pp. 12). Bogotá-Colombia , 2018. [En línea]. Available: http://hdl.handle.net/11349/13876 .

[10] A. Sankaran, « *What Should You Know Before Developing Backend for Your E-commerce Website?,» Startup Talky*. 8 Abril 2022. [En línea]. Available: https://startuptalky.com/ecommerce-website-backend-development/ .

[11] Y. Tápanes, « ¿Qué es una API REST?,» Saasradar. 10 Mayo 2022. [En línea]. Available: https://saasradar.net/desarrollo-api-rest/#Que_es_una_API_REST .

[12] L. Carrillo, « EL COMERCIO ELECTRÓNICO EN PAÍSES EN DESARROLLO: LA PLANEACIÓN PARA EL ÉXITO DEL NEGOCIO,» Universidad de Santiago de Compostela (pp. 112). España, 2010. [En línea]. Available: https://www.usc.gal/economet/.

[13] « La evolución del comercio electrónico,» Universidad Estatal a Distancia. Costa Rica, 2018. [En línea]. Available: https://multimedia.uned.ac.cr/pem/mercadeo_digital/contenidos/ii-la-evolucion-del-comercio-electronico/ .

[14] « ¿Qué es *e-commerce* o Comercio Electrónico?,». 10 Abril 2014. [En línea]. Available: https://www.visa.com.pe/dirija-su-negocio/pequenas-medianas-empresas/notas-y-recursos/tecnologia/que-es-ecommerce-o-comercio-electronico.html .

[15] « ¿QUÉ ES EL E-COMMERCE O COMERCIO ELECTRÓNICO? ESERP Business School: Escuela de Negocios. Barcelona, (s.f.) [En línea]. Available: https://es.eserp.com/articulos/e-commerce-o-comercio-electronico/ .

[16] P. Saltos, « Desarrollo de un sistema de facturación electrónica para la empresa TRISTAR utilizando el *framework* Microsoft .Net.,» Escuela Superior Politécnica de Chimborazo (pp. 22-23). Riobamba, 2010. [En línea]. Available: http://dspace.espoch.edu.ec/handle/123456789/6306 .

[17] « E-*Invoicing* FAQs – *Frequently Asked Questions*, » *Cleartax*. 17 Julio 2022. [En línea]. Available: https://cleartax.in/s/e-invoicing-faqs .

[18] « FACTURACIÓN ELECTRÓNICA EN ECUADOR,» Contifico (pp. 4). Ecuador, 2013. [En línea]. Available: https://f.hubspotusercontent10.net/hubfs/2353964/%5BOfertas%20de%20Contenido%5D/%5BECUADOR%5D/EBOOK%20-%20FACTURACI%C3%93N%20ELECTR%C3%93NICA%20EN%20EL%20ECUADOR.pdf .

[19] « *Case study*,» 4 Abril 2022. [En línea]. Available: https://www.drcath.net/toolkit/case-study .

[20] W. Panjón, « PROPUESTA DE MODELO DE SCRUM SEGURO APLICADO A UN CASO DE ESTUDIO,» Escuela Politécnica Nacional (pp. 5-66). Quito, 2019. [En línea]. Available: http://bibdigital.epn.edu.ec/handle/15000/21368 .

[21] M. de Dios, « *Scrum:* qué es y cómo funciona este marco de trabajo,» *We are marketing*. 9 Mayo 2022. [En línea]. Available: https://www.wearemarketing.com/es/blog/metodologia-scrum-que-es-y-como-funciona.html .

[22] D. Fuller, « Apuntes de Taller de Ingeniería de Software / Capítulo 4: Roles en el desarrollo de software,». 2003. [En línea]. Available: http://profayadira.yolasite.com/resources/Roles_desarrollo_software.pdf .

[23] M. Arias, « La ingeniería de requerimientos y su importancia en el desarrollo de proyectos de software,» Universidad de Costa Rica (pp. 1-2). Costa Rica, 2005. [En línea]. Available: https://www.redalyc.org/articulo.oa?id=66612870011 .

[24] A. Menzinsky, G. López, J. Palacio, M. Sobrino, R. Álvarez, V. Rivas , «Historias de Usuario Ingeniería de Requisitos Ágil,» Scrum Manager (pp. 5). 2020. [En línea]. Available: https://www.scrummanager.net/bok/index.php?title=Scrum_Manager_BoK .

[25] « Modelo de Arquitectura de Software,» Universidad Autónoma Metropolitana (s.f.) [En línea]. Available: https://academicos.azc.uam.mx/jfg/pags/adsi.html .

[26] A. Dua, « What Is Nest.JS? Why Should You Use It?, » Turing. 30 Mayo 2022. [En línea]. Available: https://www.turing.com/blog/what-is-nest-js-why-use-it-in-2022/ .

[27] M. Álvarez, « Qué es MVC,» 28 Julio 2020. [En línea]. Available: https://desarrolloweb.com/articulos/que-es-mvc.html .

[28] NestJS, « NestJS, Overview». 2022 [En línea]. Available: https://docs.nestjs.com .

[29] THREEPOINTS, « MongoDB: qué es, características y para qué sirve,» THREEPOINTS , 2022. [En línea]. Available: https://www.threepoints.com/blog/mongodb-que-es-caracteristicas-para-que-sirve .

[30] Git, « Git, Documentation,» 2022. [En línea]. Available: https://git-scm.com/ .

[31] J. Juviler, « What Is GitHub? (And What Is It Used For?)». HubSpot, 2021 [En línea]. Available: https://blog.hubspot.com/website/what-is-github-used-for .

[32] Shivang, « What is Swagger? And Why Do You Need it for your Project?,» Scaleyourapp. [En línea]. Available: https://scaleyourapp.com/what-is-swagger-and-why-do-you-need-it-for-your-project/ .

[33] « What is Postman, and Why Should I Use It?,» DigitalCrafts. [En línea]. Available: https://www.digitalcrafts.com/blog/student-blog-what-postman-and-why-use-it .

7 ANEXOS

ANEXO I. Certificado de originalidad

ANEXO II. Manual técnico

ANEXO III. Manual de usuario

ANEXO IV. Manual de instalación

ANEXO I

A continuación, se presenta el certificado de originalidad que el Director de Proyecto de Titulación ha emitido.

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

CERTIFICADO DE ORIGINALIDAD

Quito, D.M. 12 de septiembre de 2022

De mi consideración:

Yo, Richard Paúl Rivera Guevara, en calidad de Director del Trabajo de Integración Curricular titulado DESARROLLO DEL BACKEND asociado al DESARROLLO DE SISTEMA WEB DE UN E-COMMERCE CON FACTURACIÓN ELECTRÓNICA elaborado por el estudiante JHOSEF ALEXANDER REA CHAMORRO 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 completo, producto del Trabajo de Integración Curricular indicado.

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

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

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

Atentamente,

RICHARD PAUL RIVERA GUEVARA

Firmado digitalmente por RICHARD PAUL RIVERA GUEVARA
Fecha: 2022.09.12 00:09:48 -05'00'

Richard Rivera
Profesor Ocasional a Tiempo Completo
ESFOT

ANEXO II Manual técnico

En la siguiente tabla se detallan los elementos del manual técnico del proyecto.

1. RECOPILACIÓN DE REQUERIMIENTOS

A continuación, se presenta la sección de recopilación de requerimientos del negocio.

TABLA V: Recopilación de requerimientos

RECOPILACIÓN DE REQUERIMIENTOS	
TIPO DE SISTEMA	Sistema Web
ID - RR	**ENUNCIADO DEL ÍTEM**
RR001	Como API, necesito registrar usuarios
RR002	Como API necesito autenticar usuarios
RR003	Como API necesito consultar, editar, eliminar usuarios finales.
RR004	Como API necesito registrar productos.
RR005	Como API necesito consultar, editar, eliminar productos.
RR006	Como API necesito generar un documento electrónico de factura con el resumen de la compra.
RR007	Como API necesito consultar, editar, eliminar una factura.
RR008	Como API necesito implementar un carrito de compras para almacenar los productos seleccionados.
RR009	Como API necesito añadir, eliminar productos del carrito de compras.
RR010	Como API necesito responder a búsquedas por filtro a productos por el nombre, precio, categoría y promoción.
RR011	Como API necesito registrar información de la tienda: RUC, cédula del dueño del negocio, correo electrónico, teléfono, dirección.

RR012	Como API necesito enviar al usuario, vía correo electrónico la factura electrónica generada.

2. HISTORIAS DE USUARIO

En esta sección se presenta las tareas de historias de usuario del proyecto

TABLA VI: Historia de Usuario VI: Registrar Usuario

HISTORIA DE USUARIO	
Identificador: HU-01	**Usuario:** Administrador y usuario final
Nombre de historia: Registrar Usuario	
Prioridad en Negocio (Alto/Medio/Baja): Alta	**Riesgo en Desarrollo (Alto/Medio/Baja):** Medio
Iteración asignada: 1	
Responsable: Jhosef Rea	
Descripción: La API debe proporcionar la funcionalidad de registro, para que el usuario pueda registrarse; basta con los campos: • Nombre/nickname del usuario • Email del usuario • Contraseña del usuario	
Observación: No se deben pedir muchos datos para registrarse. Los usuarios podrán acceder al sistema a base de su rol.	

TABLA VII: Historia de Usuario VII: Autenticar Usuario

HISTORIA DE USUARIO	
Identificador: HU-02	**Usuario:** Administrador y usuario final
Nombre de historia: Autenticar Usuario	
Prioridad en Negocio (Alto/Medio/Baja): Alta	**Riesgo en Desarrollo (Alto/Medio/Baja):** Medio

Iteración asignada: 1
Responsable: Jhosef Rea
Descripción: La API debe proporcionar la funcionalidad de autenticación. El usuario puede acceder al sistema web a través: • Email del usuario • Contraseña del usuario
Observación: En caso de no poder acceder, existe la opción “olvidé mi contraseña”, se envía un correo de verificación.

TABLA VIII: Historia de Usuario VIII: Gestión de usuarios

HISTORIA DE USUARIO	
Identificador: HU-03	**Usuario:** Administrador
Nombre de historia: Gestión de usuarios	
Prioridad en Negocio (Alto/Medio/Baja): Alta	**Riesgo en Desarrollo (Alto/Medio/Baja):** Alta
Iteración asignada: 1	
Responsable: Jhosef Rea	
Descripción: El usuario administrador, a más de crear un usuario, es el encargado de gestionar usuarios. Como API necesito: • Crear usuarios • Consultar usuarios • Actualizar usuarios • Eliminar usuarios.	
Observación: El usuario administrador es el único encargado en ejecutar estas operaciones.	

TABLA IX: Historia de Usuario IX: Registrar Producto

HISTORIA DE USUARIO

Identificador: HU-04	**Usuario:** Administrador
Nombre de historia: Registrar Producto	
Prioridad en Negocio (Alto/Medio/Baja): Alta	**Riesgo en Desarrollo (Alto/Medio/Baja):** Medio
Iteración asignada: 1	
Responsable: Jhosef Rea	
Descripción: El usuario administrador es el encargado de registrar los productos por medio de una GUI. Los campos requeridos son: • Nombre del producto • Descripción del producto • Categoría del producto • Marca del producto • Costo del producto • Precio unitario del producto • Stock del producto • Promoción del producto, descuento.	
Observación: Un usuario registra los productos en el sistema a través de un formulario.	

TABLA X: Historia de Usuario X: Gestión de productos

HISTORIA DE USUARIO	
Identificador: HU-05	**Usuario:** Administrador
Nombre de historia: Gestión de productos	
Prioridad en Negocio (Alto/Medio/Baja): Alta	**Riesgo en Desarrollo (Alto/Medio/Baja):** Alta
Iteración asignada: 1	
Responsable: Jhosef Rea	

Descripción: A más de registrar productos, el usuario administrador es el encargado de gestionar los productos, realizando las siguientes actividades: • Consultar productos • Actualizar productos • Eliminar productos
Observación: El usuario administrador es el único encargado en realizar estas operaciones, sin embargo, el usuario cliente puede consultar productos.

TABLA XIII: Historia de Usuario XIII: Implementación del carrito de compras

HISTORIA DE USUARIO	
Identificador: HU-06	**Usuario:** Administrador
Nombre de historia: Implementación del carrito de compras	
Prioridad en Negocio (Alto/Medio/Baja): Media	**Riesgo en Desarrollo (Alto/Medio/Baja):** Medio
Iteración asignada: 3	
Responsable: Jhosef Rea	
Descripción: La API es la encargada de la funcionalidad del carrito de compras, teniendo en cuenta que, un carrito de compras almacena los productos seleccionados. La API debe proporcionar la información de cada producto añadido al carrito de compras: • Nombre del producto • Precio unitario del producto • Cantidad del producto.	
Observación: Un carrito de compras pertenece a un usuario.	

TABLA XIV: Historia de Usuario XIV: Operatividad del carrito de compras

HISTORIA DE USUARIO	
Identificador: HU-07	**Usuario:** Administrador

Nombre de historia: Operatividad del carrito de compras	
Prioridad en Negocio (Alto/Medio/Baja): Media	**Riesgo en Desarrollo (Alto/Medio/Baja):** Medio
Iteración asignada: 3	
Responsable: Jhosef Rea	
Descripción: La API es la encargada de la funcionalidad del carrito de compras, teniendo en cuenta que, un usuario final puede añadir y quitar productos. Luego de efectuar una compra, el carrito queda vacío.	
Observación: El carrito de compras debe mantener la información al navegar en la página hasta que se finalice una compra.	

TABLA XVII: Historia de Usuario XVI: Registro de información del negocio

	HISTORIA DE USUARIO
Identificador: HU-08	**Usuario:** Administrador
Nombre de historia: Registro de información del negocio	
Prioridad en Negocio (Alto/Medio/Baja): Baja	**Riesgo en Desarrollo (Alto/Medio/Baja):** Baja
Iteración asignada: 4	
Responsable: Jhosef Rea	
Descripción: La API es la encargada de registrar la información de la tienda: • Nombre del negocio • RUC del dueño del negocio • Cédula del dueño del negocio • Correo electrónico del dueño del negocio • Teléfono del dueño del negocio • Dirección del negocio.	
Observación: La información del negocio es necesaria para facturar una compra.	

TABLA XI: Historia de Usuario XI: Generar factura electrónica

HISTORIA DE USUARIO	
Identificador: HU-09	**Usuario:** Administrador
Nombre de historia: Generar factura electrónica	
Prioridad en Negocio (Alto/Medio/Baja): Alta	**Riesgo en Desarrollo (Alto/Medio/Baja):** Alta
Iteración asignada: 2	
Responsable: Jhosef Rea	
Descripción: La API es la encargada de generar un documento electrónico de factura con el resumen de la compra. La información que contiene la factura electrónica es: • Información del negocio • Información del cliente • Detalle de la compra, que involucra: • Nombre del producto • Cantidad del producto • Precio unitario del producto • Subtotal de compra • IVA • Total de compra	
Observación: El documento es generado, luego de una compra exitosa.	

TABLA XII: Historia de Usuario XII: Gestión de factura electrónica

HISTORIA DE USUARIO	
Identificador: HU-10	**Usuario:** Administrador
Nombre de historia: Gestión de factura electrónica	
Prioridad en Negocio (Alto/Medio/Baja): Alta	**Riesgo en Desarrollo (Alto/Medio/Baja):** Alta
Iteración asignada: 2	

Responsable: Jhosef Rea
Descripción: La API con la intervención del usuario administrador tienen el control del sistema, quién a través de una GUI necesita: • Consultar una factura, • Anular una factura.
Observación: El usuario administrador es el único encargado en realizar estas operaciones y no puede actualizar ni eliminar una factura electrónica pues, el sistema se ajusta a la normativa del SRI.

TABLA XVI: Historia de Usuario XVI: Envío de factura vía e-mail.

HISTORIA DE USUARIO	
Identificador: HU-11	**Usuario:** Administrador
Nombre de historia: Envío de factura vía e-mail.	
Prioridad en Negocio (Alto/Medio/Baja): Medio	**Riesgo en Desarrollo (Alto/Medio/Baja):** Alta
Iteración asignada: 4	
Responsable: Jhosef Rea	
Descripción: La API es la encargada de enviar al usuario, vía correo electrónico la factura electrónica generada.	
Observación: El envío del documento electrónico se efectúa siempre y cuando el usuario haya generado la factura con el detalle de la compra.	

TABLA XV: Historia de Usuario XV: Herramienta de búsqueda

HISTORIA DE USUARIO	
Identificador: HU-12	**Usuario:** Administrador
Nombre de historia: Herramienta de búsqueda	
Prioridad en Negocio (Alto/Medio/Baja): Medio	**Riesgo en Desarrollo (Alto/Medio/Baja):** Medio

Iteración asignada: 4
Responsable: Jhosef Rea
Descripción: La API es la encargada de la funcionalidad de la herramienta de búsqueda, teniendo en cuenta que, un usuario puede buscar productos mediante una barra de búsqueda o puede filtrar a través de las siguientes características: • Nombre del producto • Precio unitario del producto • Categoría del producto • Promoción del producto. Como resultado, se obtiene un listado de productos relacionados con la búsqueda.
Observación: En la herramienta de búsqueda se define el método de búsqueda (búsqueda simple o filtro)

3. PRODUCT BACKLOG

En esta sección se presenta el *Product Backlog* del proyecto.

TABLA XVIII: *Product Backlog*

ELABORACIÓN DEL *PRODUCT BACKLOG*				
ID –HU	**HISTORIA DE USUARIO**	**ITERACIÓN**	**ESTADO**	**PRIORIDAD**
HU-00	Configuraciones iniciales	0	Finalizado	Alta
HU-01	Registrar usuario	1	Finalizado	Alta
HU-02	Autenticar usuario	1	Finalizado	Alta
HU-03	Gestión de usuarios	2	Finalizado	Alta
HU-04	Registrar producto	2	Finalizado	Alta
HU-05	Gestión de productos	2	Finalizado	Alta
HU-06	Implementación del carrito de compras	3	Finalizado	Alta
HU-07	Operatividad del carrito de compras	3	Finalizado	Alta

HU-08	Registro de información del negocio	3	Finalizado	Media
HU-09	Generar factura electrónica	3	Finalizado	Media
HU-10	Gestión de factura electrónica	3	Finalizado	Media
HU-11	Envío de factura vía e-mail	4	Finalizado	Media
HU-12	Herramienta de búsqueda	4	Finalizado	Baja

4. SRINT BACKLOG

En esta sección se presenta el *Sprint Backlog* correspondiente al proyecto.

TABLA XIX: *Sprint Backlog*

ELABORACIÓN DEL SPRINT BACKLOG						
ID-SB	**NOMBRE**	**MÓDULO**	**ID-HU**	**HISTORIA DE USUARIO**	**TAREAS**	**TIEMPO ESTIMADO**
SB00	Configuración del ambiente de desarrollo		N/A	N/A	• Recopilación de requerimientos, HU. • Instalación de herramientas para el desarrollo del proyecto. • Creación de repositorio en *GitHub* • Modelamiento de la base de datos *NoSQL*.	50 H
SB01	Diseño e implementación del *endpoint* para los usuarios con perfil cliente y administrador		HU-01	Registrar usuario	• Diseño e Implementación del *endpoin*t del registro, inicio de sesión y cierre de sesión de usuarios. • Crear conexión con la base de datos de usuarios según su rol. • Consulta a la base de datos y autorización.	30 H

					• Validación de campos de registro de usuario. • Verificación de registro único.	
			HU-02	Autenticar usuario	• Validación de campos para la autenticación de usuarios. • Elaboración de ruta de inicio de sesión de usuarios. Diseño e Implementación del *endpoint* de perfil de usuario.	
SB02	Diseño e Implementación de *endpoints* del modelo de usuarios y productos.	Gestión de usuarios	HU-03	Gestión de usuarios	• Diseño e implementación de *endpoints* para registrar, leer, actualizar y eliminar usuarios. • Consulta en la Base de datos y autorización. • Diseño e Implementación del *endpoint* para registrar, leer, actualizar y eliminar clientes.	80 H

		Gestión de productos	HU-04	Registrar producto	• Diseño e Implementación del *endpoint* del registro de producto. • Crear conexión con la base de datos de productos. • Consulta a la base de datos y autorización. • Validación de campos de registro de producto. • Verificación del consumo de data desde el cliente REST.	
			HU-05	Gestión de productos	• Diseño e implementación de *endpoints* para registrar, leer, actualizar y eliminar productos. • Consulta en la Base de datos y autorización.	
SB03	Diseño e Implementación de *endpoints* del modelo		HU-06	Implementación del carrito de compras	• Diseño e Implementación de *endpoints* para la administración de productos agregados al carrito. • Consulta a la base de datos.	

	de carrito de compras y modelo de facturación electrónica.				• Validación de campos del carrito.	
			HU-07	Operatividad del carrito de compras	• Registrar, leer producto. • Consulta en la Base de datos.	
			HU-08	Registro de información del negocio	• Diseño e implementación de *endpoints* para registrar, leer, actualizar y eliminar la información de la tienda. • Consulta a la base datos y autorización.	
			HU-09	Generar factura electrónica	• Diseño e Implementación del *endpoint* de facturación electrónica. • Crear conexión con la base de datos de carritos. • Consulta a la base de datos y autorización. • Validación de campos.	

			HU-10	Gestión de factura electrónica	• Diseño e implementación de *endpoints* para registrar, leer, actualizar y eliminar facturas electrónicas. • Consulta en la Base de datos y autorización.	
SB04	Diseño e implementación de un *endpoint/s* de envío de correo electrónico y de Herramienta de búsqueda.	Gestión del carrito de compras	HU-11	Envío de factura vía e-mail	• Diseño e implementación de un *endpoint* para enviar la factura electrónica vía e-mail. • Consulta a la base de datos y autorización.	100 H
			HU-12	Herramienta de búsqueda	• Respuesta a búsqueda de un producto por nombre, precio, categoría y promoción. • Implementación de funciones para la búsqueda de productos.	
SB05	Pruebas en el *backend*.				• Pruebas unitarias. • Pruebas de rendimiento.	20 H

Diseño de la Base de datos *NoSQL*

En la **Fig. 39** se puede observar las seis colecciones que se han definido para la estructuración del API REST que se compone de *endpoints*.

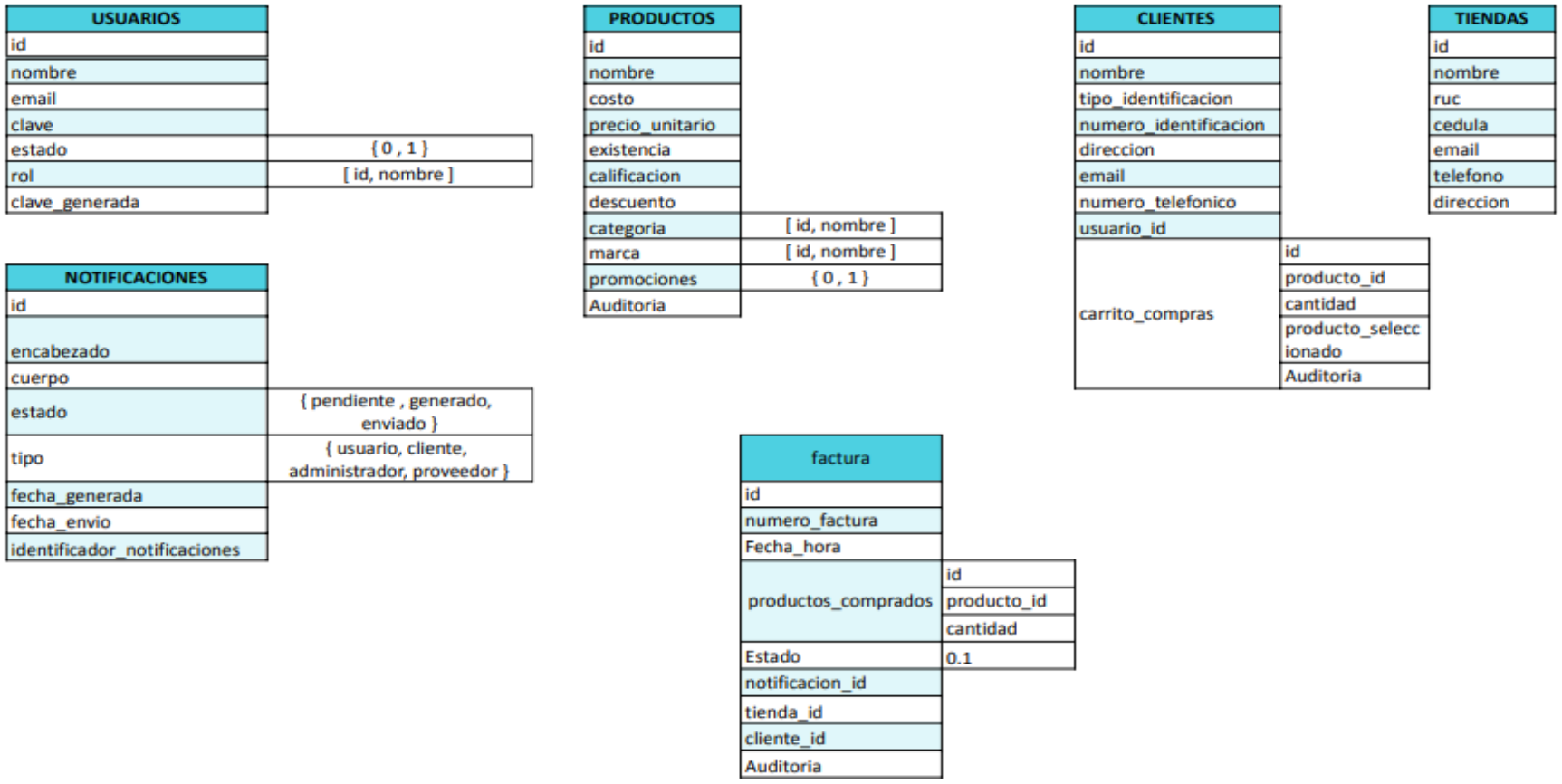


Fig. 39: Diseño de la Base de Datos no Relacional.

Arquitectura REST del backend

En la **Fig. 40** se visualiza la arquitectura REST final del proyecto con *NestJs*. Con base en los requerimientos se han establecido los recursos que se compone de un módulo, un controlador, un servicio y el *DTO*; más, se han definido utilidades, decoradores personalizados, *helpers*, roles, configuraciones y la estrategia de autenticación. *NestJs* proporciona una arquitectura escalable y nos obliga a seguir su estructura.

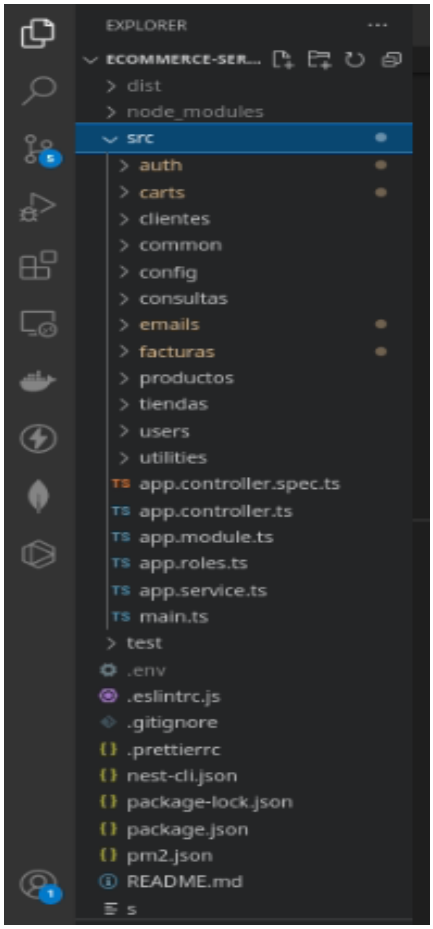


Fig. 40: Estructura final del *backend*.

Arquitectura del *frontend* para obtener una factura electrónica

Cuando un usuario efectúa una factura electrónica, esta es generada en el *backend* y a través de una ruta *HTTP GET* el usuario puede visualizar la información; de hecho, esta respuesta es visualizada en **Fig. 25**. En efecto, no es apreciable y menos la implementación del código QR ni el detalle de la compra. En este sentido, para poder revisar la estructura del documento PDF obtenido, se codificó un pequeño *frontend* con *Ionic* y *Angular* (**Fig. 41**); cabe resaltar que, estos *frameworks* no forman parte de las herramientas de desarrollo;

pues, únicamente se adoptaron para poder visualizar la estructura del documento PDF generado para poder armarlo.

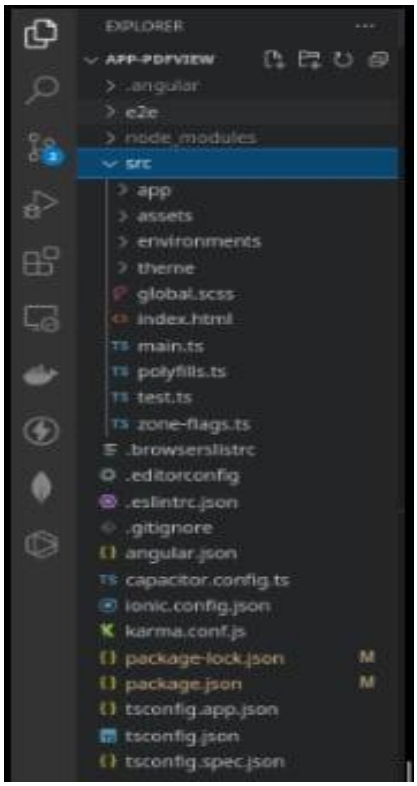


Fig. 41: Estructura de *Frontend* para consultar una factura electrónica.

Pruebas

Una vez finalizada la etapa de codificación se ha desarrollado la ejecución de pruebas unitarias y pruebas de compatibilidad para comprobar la calidad del código del *backend* con sus respectivos módulos.

Pruebas unitarias

A continuación, en la **Fig. 42** se observa el código y la respuesta que envía el *backend* en cuanto a la autenticación de un usuario. En la **Fig. 43** hace referencia a la prueba unitaria realizada a la creación de productos. En cambio, en la **Fig. 44** y **Fig. 45¡Error! No se encuentra el origen de la referencia.** corresponde al recurso de consultas y facturas respectivamente.

Fig. 42: Código de prueba al controlador del recurso de autenticación.

Fig. 43: Código de prueba a la creación de un producto.

```
describe('ConsultasController', () => {
  let controller: ConsultasController;

  beforeEach(async () => {
    const module: TestingModule = await Test.createTestingModule({
      imports: [
        ConsultasModule,
        ProductosModule,
        AccessControlModule.forRoles(roles),
      ],
      controllers: [ConsultasController],
      providers: [
        ConsultasService,
        {
          provide: getModelToken(Productos.name),
          useValue: { Symbol: jest.fn() },
        },
      ],
    })
      .overrideProvider(getModelToken(Productos.name))
      .useValue(jest.fn())
      .compile();
    controller = module.get<ConsultasController>(ConsultasController);
  });

  it('should be defined', () => {
    expect(controller).toBeDefined();
  });
});
```

Fig. 44: Código de prueba al controlador del recurso Consultas.

```
describe('ConsultasController', () => {
  let controller: ConsultasController;

  beforeEach(async () => {
    const module: TestingModule = await Test.createTestingModule({
      imports: [
        ConsultasModule,
        ProductosModule,
        AccessControlModule.forRoles(roles),
      ],
      controllers: [ConsultasController],
      providers: [
        ConsultasService,
        {
          provide: getModelToken(Productos.name),
          useValue: { Symbol: jest.fn() },
        },
      ],
    })
      .overrideProvider(getModelToken(Productos.name))
      .useValue(jest.fn())
      .compile();
    controller = module.get<ConsultasController>(ConsultasController);
  });

  it('should be defined', () => {
    expect(controller).toBeDefined();
  });
});
```

Fig. 45: Código de prueba al controlador del recurso facturas.

Finalmente, se ejecuta todas las pruebas y se obtuvo el resultado de ejecución por cada prueba, como se ilustra en la **Fig. 46** con respuesta exitosa de cada una. Por consiguiente, en **Fig. 47** se observa el reporte del código referente al recurso de usuarios.

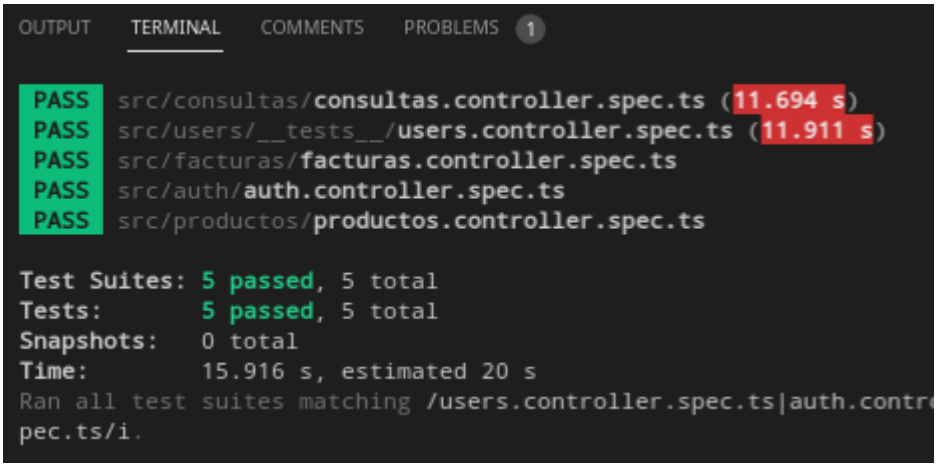


Fig. 46: Resultado exitoso de cada prueba.

Fig. 47: Reporte del recurso Usuarios

Prueba de Compatibilidad

Las pruebas de compatibilidad se han efectuado utilizando *Swagger* y *Postman* como clientes *HTTP*. A continuación, en la **Fig. 48** se ilustra el acceso al *endpoint* de registro; en la **Fig. 49** se invoca al método *GET* en el *endpoint* de productos en *Postman*; asimismo, en la **Fig. 50** corresponde a un método *GET* que recibe los *params* y *query* del *endpoint* para realizar una consulta por clasificación. Por último, en **Fig. 51** se observa el intento de acceso de un usuario a un *endpoint* protegido; en este caso pertenece a un usuario sin autenticación que intenta visualizar las facturas electrónicas del sistema; pero obtiene la respuesta esperada del servidor.

.

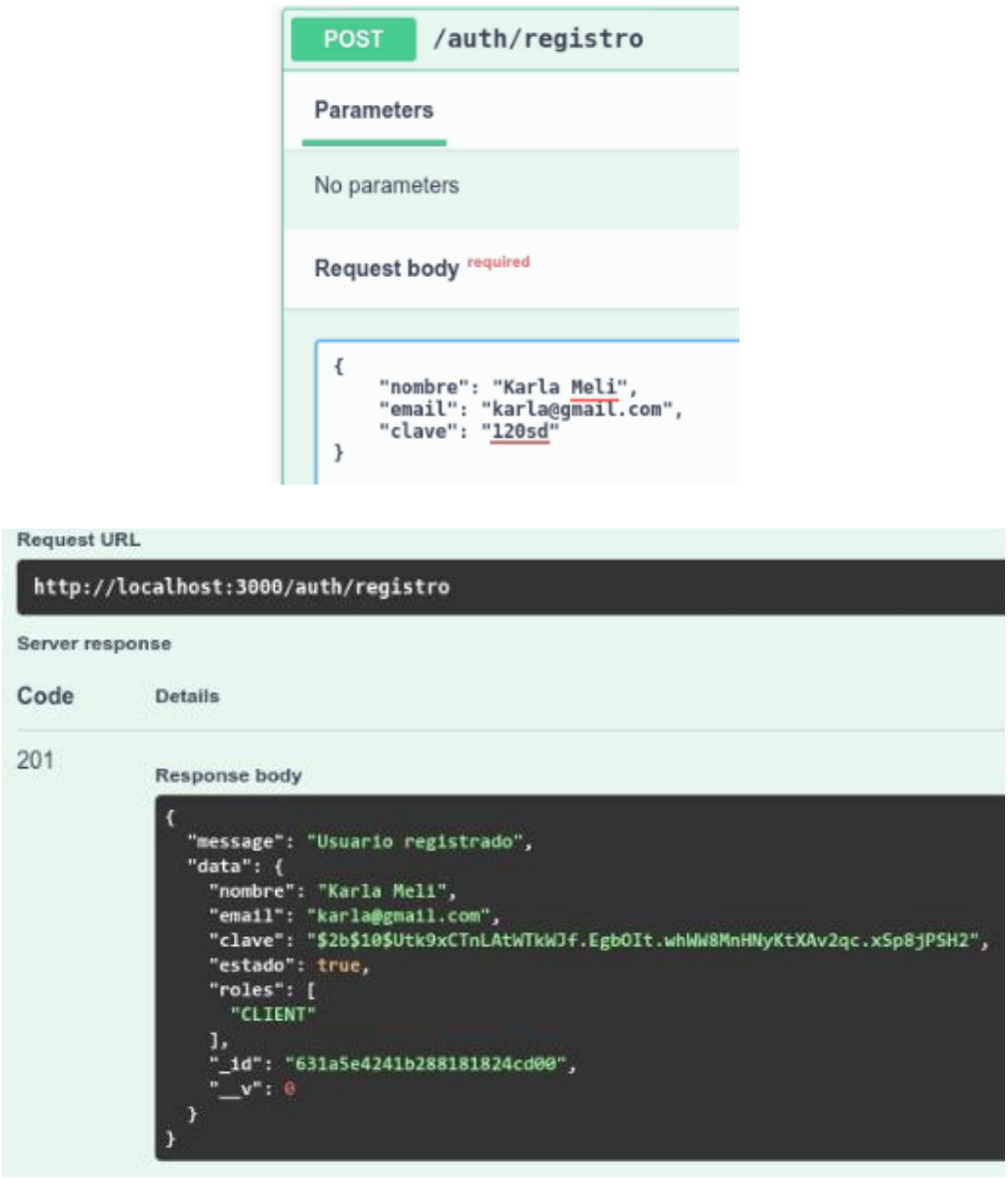


Fig. 48: *Endpoint* para probar el registro de un usuario.

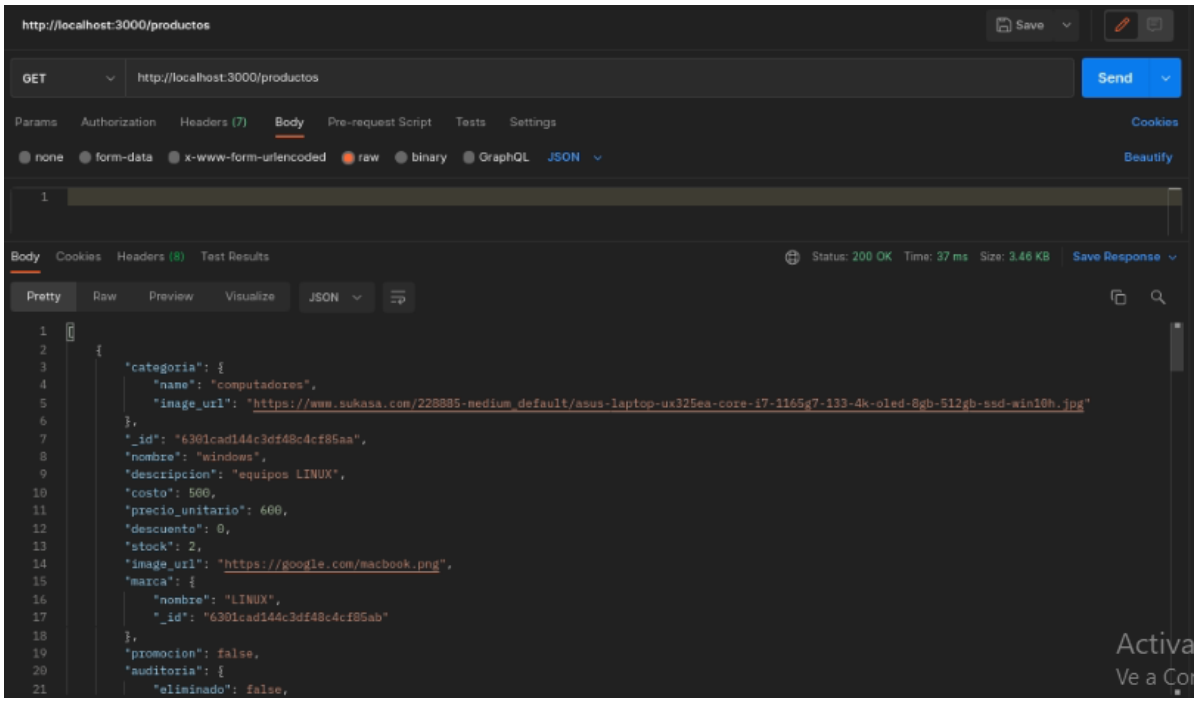
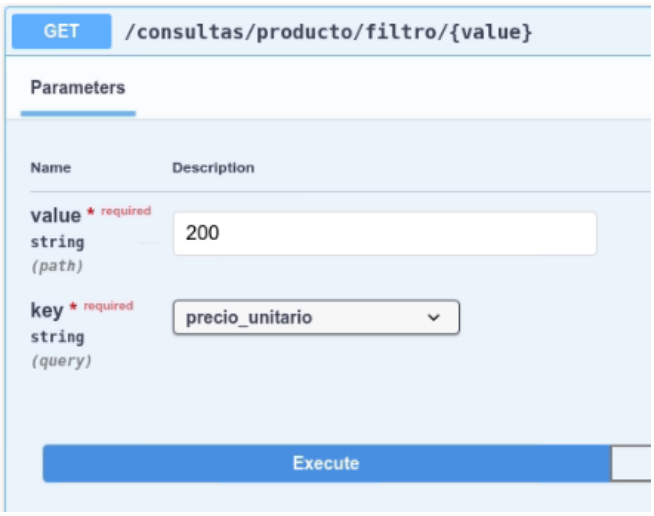


Fig. 49: *Endpoint* para probar la visualización de productos.



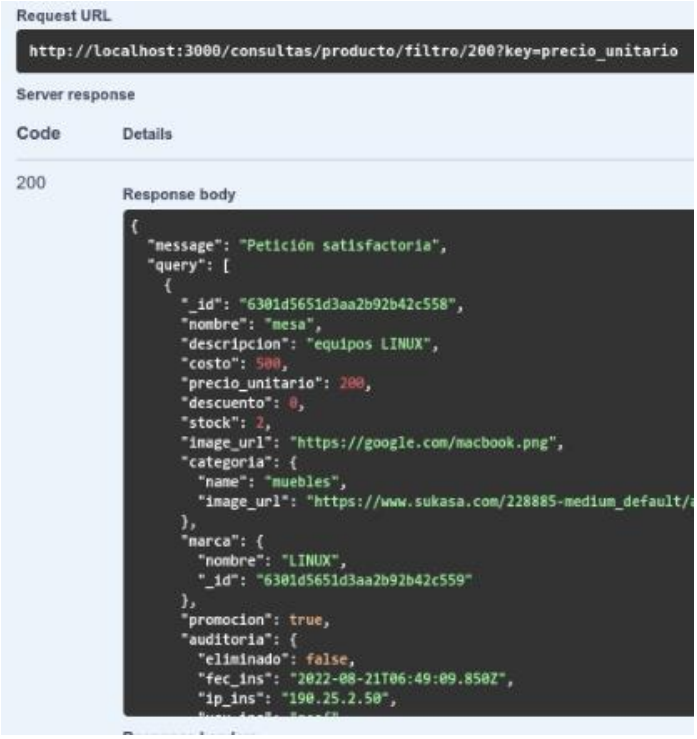


Fig. 50: Endpoint para probar la herramienta de búsqueda por clasificaciones.

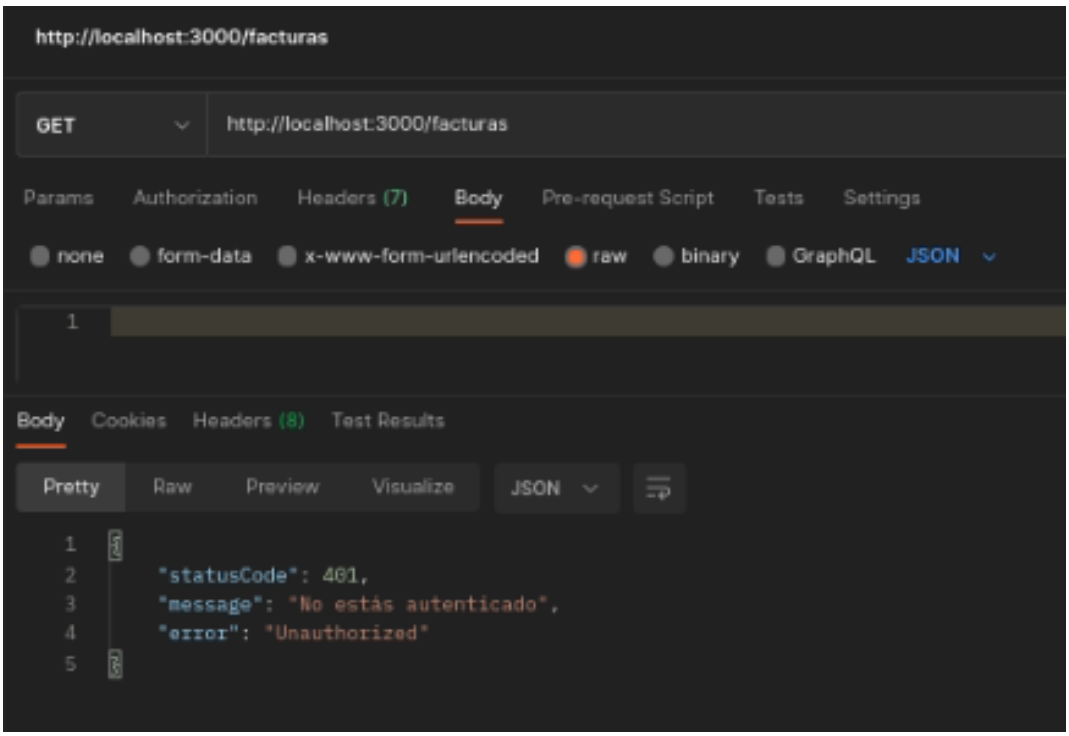


Fig. 51: *Endpoint* para obtener las facturas electrónicas según el rol de usuario.

ANEXO III. Manual de usuario

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

https://drive.google.com/drive/folders/1f2zgIjjD2j57kB5DmsvrVf_5PZeHCQbk?usp=sharing

En el vídeo que se presenta en el enlace, se menciona de forma clara, ordenada y detallada cada una de las funcionalidades del *backend*.

ANEXO IV. Manual de instalación

A continuación, se procede a exponer el código fuente desarrollado para el *backend*, que se encuentra en el enlace al repositorio de *GitHub*; asimismo, las instrucciones de instalación detalladas en el *README* con los pasos para realizar la instalación de forma local. A este se puede acceder en el siguiente URL: https://github.com/TitulacionEcommerce/server-ecommerce.git