PONTIFICIA UNIVERSIDAD CATÓLICA DEL ECUADOR

FACULTAD DE INGENIERÍA

CARRERA DE: INGENIERIA EN SISTEMAS DE INFORMACIÓN

Trabajo de Titulación

Tema: Desarrollo de un Sistema Web para la Gestión de Comercialización de Productos Orientado a Modelos de Negocio de Compra, Venta y Consignación.

AUTOR:

David Mateo Balseca Velastegui

QUITO DM, 11 DE ABRIL DE 2024

ÍNDICE

ÍNDICE ILUSTRACIONES

1. INTRODUCCION

1.1 Tema

Desarrollo de un sistema web para la gestión de comercialización de productos orientado a modelos de negocio de compra, venta y consignación.

1.2 Justificación

El uso de la tecnología en todas las áreas de conocimiento es imprescindible, especialmente en la gestión de los procesos de los negocios. En la actualidad un número de PYMES manejan sus procesos manualmente, teniendo problemas como la duplicidad de información, la falta control de inventarios, falta de control de clientes, falta de eficiencia lo que crea una demora en realizar procesos. Estos negocios se encuentran en la necesidad de modernizar sus procesos de gestión. Esta obsoleta metodología presenta ineficiencias operativas, aumenta el riesgo de errores, y obstaculiza la capacidad de competir en un mercado dinámico.

La implementación de un sistema automatizado en negocios con modelos de compra, venta y consignación optimiza la eficiencia y la precisión en la gestión de ventas, compras, consignaciones e inventario y también fortalece la competitividad, además permite una mejor toma de decisiones y contribuye al crecimiento y sostenibilidad a largo plazo de la empresa en un entorno empresarial cada vez más tecnológico y competitivo. El sistema es una solución innovadora y sostenible que beneficia tanto a la empresa como a sus clientes.

1.3 Planteamiento del Problema

Las Pequeñas y Medianas Empresas (PYMES) que operan con sistemas manuales enfrentan varios desafíos. En primer lugar, la falta de información oportuna y de calidad obstaculiza la toma de decisiones estratégicas. En segundo lugar, la necesidad de un sistema contable flexible y sencillo es esencial para que la gerencia pueda tomar decisiones acertadas. En tercer lugar, los sistemas manuales no proporcionan las herramientas necesarias para encontrar posibles clientes. Por último, la falta de una cultura corporativa basada en la innovación puede limitar la eficiencia y la colaboración. En un contexto empresarial caracterizado por la creciente demanda de eficiencia y competitividad, negocios pequeños se enfrenta a un desafío crítico: la persistente dependencia de procesos manuales en la gestión de ventas, compras, consignaciones e inventario.

La pregunta fundamental se centra en cómo la falta de un sistema automatizado afecta la capacidad de una empresa para competir, crecer y tomar decisiones informadas. El desarrollo del sistema web se enfocó en abordar estas cuestiones críticas y en proveer soluciones que

puedan llevar a una transformación positiva en la gestión de la comercialización de productos de la empresa.

1.4 Objetivos

1.4.1 General

Desarrollo de un sistema web para la gestión de comercialización de productos orientado a modelos de negocio de compra, venta y consignación.

1.4.2 Específicos

1. Recopilar toda la información necesaria para definir los requisitos funcionales del sistema web.
2. Diseñar un sistema web que cumple con los requerimientos funcionales.
3. Desarrollar el sistema web empleando una metodología ágil.
4. Realizar pruebas y evaluar resultados.

1.5 Alcance

El presente proyecto culmina con la entrega de la aplicación desarrollada que cumpla con los requerimientos funcionales con enfoque a empresas de compra, venta y consignación de diversos productos. Los módulos por desarrollar son:

- Gestión de Usuarios:
  - Administración de usuarios.
  - Autenticación (login/logout).
  - Actualización y gestión de perfiles de usuario.
- Gestión de Productos:
  - Registro y actualización de productos.
  - Clasificación y organización por categorías.
  - Registro y control de costos de adquisición.
  - Gestión de precios de venta.
  - Control de stock en tiempo real.
- Gestión de Clientes:
  - Registro y administración de clientes.
  - Mantenimiento de información detallada de contacto.

        - Historial de compras y clasificación de clientes.
    - Gestión de Contratos:
        - Creación y administración de contratos de consignación.
        - Seguimiento de contratos.
        - Historial de términos y fechas de los contratos.
    - Gestión de Ventas:
        - Registro y administración de ventas.
        - Análisis de ventas.
    - Gestión de Compras:
        - Registro y seguimiento de compra.
        - Control de recepción de mercancías y actualización de inventario.

2. MARCO TEÓRICO

2.1 Introducción

El marco teórico ofrece la base conceptual y las herramientas teóricas para abordar el problema de investigación. En el desarrollo del sistema de gestión comercial web, el marco teórico facilita la comprensión de las metodologías, herramientas y conceptos clave que guían el desarrollo del sistema. Este capítulo se centra en las metodologías de desarrollo de sistemas, destacando especialmente la metodología Scrum, así como en las herramientas y tecnologías que se emplean en este proyecto.

2.2 Modelo de Negocio

El modelo de negocio de compra, venta y consignación, también conocido como comercio minorista, es uno de los más antiguos y comunes en el mundo. En este modelo, una empresa adquiere productos de un proveedor, generalmente a un precio mayorista, y luego los vende a los consumidores a un precio más alto para obtener ganancias. La diferencia entre el costo de adquisición y el precio de venta es el margen de beneficio. Este modelo puede aplicarse a una variedad de industrias y mercados, desde tiendas físicas hasta comercio electrónico.

La clave del éxito en este modelo de negocio es la capacidad de ofrecer productos de calidad a precios competitivos, es importante mantener bajos los costos operativos y proporcionar un excelente servicio al cliente.

2.3 Metodologías para el Desarrollo de Sistemas

Existen una gran variedad de metodologías para el desarrollo de sistemas. La elección de una metodología adecuada para el desarrollo de sistemas es importante para el éxito de cualquier proyecto de software. Todas las metodologías proporcionan un marco estructurado que guían el proceso de desarrollo, asegurando que se sigan buenas prácticas y se alcancen los objetivos del proyecto.

2.3.1 Modelo en Cascada

El modelo en cascada es una de las metodologías más tradicionales en el desarrollo de software. Se caracteriza por seguir una secuencia lineal de fases: análisis de requisitos, diseño, implementación, pruebas, despliegue y mantenimiento. Cada fase debe completarse antes de que la siguiente pueda comenzar. Este enfoque es mejor cuando los requisitos están bien entendidos y se pueden definir por adelantado. (Sommerville, 2011) se da a entender que este modelo termina siendo rígido y no se adapta bien a cambios en los requisitos del proyecto.

2.3.2 Modelo Incremental

El modelo incremental combina elementos del modelo en cascada con la iteración. En lugar de completar el sistema en una sola fase, el desarrollo se divide en incrementos pequeños, donde cada incremento añade funcionalidad al sistema (Pressman, 2014). Esto permite una entrega más rápida de partes funcionales del sistema y facilita la retroalimentación y ajustes durante el desarrollo.

2.3.3 Metodologías Agiles

Las metodologías ágiles, se enfocan en la flexibilidad y la capacidad de respuesta a cambios. Las metodologías ágiles se centran en la entrega temprana y continua de software funcional. (Schwaber & Sutherland, 2020). Estas metodologías promueven el desarrollo iterativo e incremental, con ciclos cortos de desarrollo, la colaboración cercana con el cliente y la retroalimentación continua son pilares de las metodologías ágiles.

2.3.4 Comparación de Metodologías

Tomar una decisión entre diferentes metodologías depende de varios factores, como la naturaleza del proyecto, el entorno de trabajo y las preferencias de desarrollo. El modelo en cascada es ideal para proyectos con requisitos claramente definidos y poco susceptibles de cambiar, mientras que el modelo incremental y las metodologías ágiles son más adecuadas para proyectos donde se anticipa la evolución de los requisitos y se valora la retroalimentación continua.

2.3.5 Ventajas de Scrum sobre otras Metodologías

Scrum mejora la flexibilidad y la adaptabilidad, mientras que fomenta la mejora continua a través de reuniones de retrospectiva, permitiendo al equipo reflexionar y mejorar su proceso en cada sprint. (Cohn, 2010). Algunas ventajas de Scrum sobre las metodologías tradicionales y el modelo incremental son:

- Flexibilidad y Adaptabilidad: Scrum permite adaptarse rápidamente a cambios en los requisitos del proyecto, lo que es crucial en entornos dinámicos.
- Entrega Continua: La estructura de sprints cortos asegura la entrega regular de partes funcionales del sistema, proporcionando valor continuo al cliente.
- Colaboración y Comunicación: Scrum fomenta una comunicación abierta y frecuente entre todos los miembros del equipo, mejorando la colaboración y transparencia.

- Mejora Continua: Las reuniones de retrospectiva al final de cada sprint permiten reflexionar sobre el desempeño y realizar mejoras continuas en el proceso de desarrollo.

2.4 Metodología Scrum

Scrum es una de las metodologías ágiles más populares y ampliamente utilizadas en el desarrollo de software. Creada por Ken Schwaber y Jeff Sutherland, esta metodología se enfoca en la entrega incremental de productos funcionales mediante iteraciones cortas y estructuradas conocidas como sprints.

2.4.1 Principios y Valores de Scrum

Scrum se fundamenta en tres principios: transparencia, inspección y adaptación. Estos principios garantizan que el equipo de desarrollo pueda adaptarse rápidamente a los cambios y mejorar continuamente su proceso de trabajo.

- Transparencia: En Scrum, es esencial que todos los aspectos importantes del proceso sean visibles para los responsables del resultado. Esto se logra mediante el uso de artefactos y reuniones que aseguran la disponibilidad de la información para todos los miembros del equipo. Según Schwaber y Sutherland (2020), la transparencia permite que todos los miembros del equipo comprendan el trabajo a realizar de manera uniforme.
- Inspección: Los usuarios de Scrum deben revisar frecuentemente los artefactos de Scrum y el progreso hacia un objetivo para identificar desviaciones no deseadas. Esta revisión constante garantiza que los problemas se identifiquen y resuelvan rápidamente.
- Adaptación: Se debe tener en cuenta que si un inspector detecta que un aspecto o varios aspectos de un proceso se desvían de los límites aceptables y que el producto resultante no será satisfactorio, se deben ajustar el proceso o los materiales a procesar. Esta adaptación ayuda a mantener la calidad y cumplir con los objetivos del proyecto.

2.4.2 Roles en Scrum

Scrum define tres roles principales: el Product Owner, el Scrum Master y el Development Team. Cada uno de estos roles tiene responsabilidades específicas que contribuyen al éxito del proyecto.

- Product Owner: El Product Owner se encarga de maximizar el valor del producto que resulta del trabajo del equipo de desarrollo. Administra el Product Backlog y

garantiza que el equipo se enfoque en las tareas de mayor valor para el cliente. Según Schwaber y Sutherland (2020), el Product Owner es el único responsable de la gestión del Product Backlog.

- Scrum Master: El Scrum Master se encarga de garantizar que Scrum sea comprendido e implementado adecuadamente. Facilita el proceso Scrum, ayuda a eliminar obstáculos y protege al equipo de interrupciones externas. Además, el Scrum Master organiza y dirige las reuniones de Scrum.
- Development Team: Es un grupo autoorganizado y multifuncional que es responsable de entregar incrementos de producto "Terminados" al final de cada sprint. El equipo es responsable de todas las actividades necesarias para convertir los elementos del Product Backlog en un incremento de producto.

2.4.3 Artefactos de Scrum

Scrum emplea tres artefactos principales para gestionar el trabajo y garantizar la transparencia: el Product Backlog, el Sprint Backlog y el Incremento.

- Product Backlog: Es una lista priorizada de todo lo necesario para el producto, gestionada por el Product Owner. Constituye la única fuente de requisitos para cualquier modificación del producto. El Product Backlog evoluciona conforme cambian las necesidades del negocio, los requisitos del mercado y las tecnologías.
- Sprint Backlog: Consiste en los elementos del Product Backlog seleccionados para el sprint, junto con un plan para entregar el incremento del producto y cumplir con el objetivo del sprint. Representa una previsión del trabajo que el equipo de desarrollo llevará a cabo para el próximo incremento del producto.
- Incremento: Comprende de todos los elementos del Product Backlog completados durante un sprint y en sprints anteriores. Debe estar en un estado utilizable y cumplir con la definición de "Terminado" acordada por el equipo.

2.4.4 Eventos de Scrum

Scrum organiza el trabajo mediante eventos específicos que establecen una cadencia y reducen la necesidad de reuniones no planificadas en Scrum. Estos eventos incluyen los sprints, las reuniones diarias (Daily Stand-Ups), las revisiones de sprint (Sprint Review) y las retrospectivas de sprint (Sprint Retrospective).

- Sprints: El Sprint es el núcleo de Scrum, un periodo de hasta un mes en el que se produce un Incremento de producto "Terminado". Los Sprints mantienen una duración constante durante todo el desarrollo. De acuerdo con Schwaber y Sutherland (2020), cada sprint se puede ver como un proyecto con un horizonte temporal de no más de un mes.
- Daily Stand-Ups: Son conocidas como reuniones diarias, Estas constan reuniones cortas que se realizan cada día del sprint. Durante estas reuniones, los miembros del equipo responden a tres preguntas: ¿Qué hice ayer? ¿Qué haré hoy? ¿Hay algún impedimento que me detenga?
- Sprint Review: Al final de cada sprint, se lleva a cabo una revisión para evaluar el Incremento y ajustar el Product Backlog si es necesario. En esta reunión, el equipo de desarrollo presenta el trabajo realizado y discute con el Product Owner y otros interesados sobre lo que se ha hecho y las posibles acciones futuras.
- Sprint Retrospective: El equipo realiza una retrospectiva para reflexionar sobre el sprint que acaba de terminar. La retrospectiva permite al equipo discutir lo que funcionó bien, lo que no funcionó y cómo pueden mejorar el proceso de desarrollo en el siguiente sprint.

2.5 Herramientas de Desarrollo

Las herramientas seleccionadas para el desarrollo del sistema de gestión comercial web son importantes para asegurar un desarrollo eficiente, escalable y de calidad. Este proyecto utilizará Node.js, Express.js, React.js y PostgreSQL, junto con algunas tecnologías adicionales como Sequelize y Bootstrap. A continuación, se detalla cada una de estas herramientas.

2.5.1 Node.js

Node.js es un entorno de ejecución de JavaScript para el lado del servidor que permite crear aplicaciones eficientes y escalables. Node.js ha ganado popularidad debido a su capacidad para manejar múltiples conexiones simultáneamente con alto rendimiento y bajo consumo de recursos. Su arquitectura permite a los desarrolladores utilizar JavaScript tanto en el frontend como en el backend, unificando el lenguaje de programación y facilitando el desarrollo full-stack. Además, "Node.js utiliza un modelo de E/S no bloqueante y una arquitectura basada en eventos, lo que lo hace especialmente adecuado para aplicaciones en tiempo real que requieren un alto rendimiento" (Node.js Foundation, 2023).

2.5.2 Express.js

Express ofrece un conjunto robusto de funcionalidades para aplicaciones web y móviles, incluyendo la gestión de rutas, middleware y plantillas. Express.js es conocido por su simplicidad y rendimiento, lo que lo hace adecuado para proyectos de cualquier tamaño. La flexibilidad de Express.js permite a los desarrolladores añadir fácilmente diversas funcionalidades a través de middleware, lo que simplifica la gestión de solicitudes HTTP y la respuesta del servidor. Su diseño modular facilita la integración con otras bibliotecas y herramientas de Node.js, mejorando la eficiencia y reduciendo el tiempo de desarrollo. Además, según el sitio oficial, "Express.js permite a los desarrolladores personalizar el comportamiento del servidor para manejar diferentes tipos de solicitudes y respuestas, lo que resulta en aplicaciones más dinámicas y responsivas" (Express.js, 2023).

2.5.3 React.js

React.js es una biblioteca de JavaScript para la construcción de interfaces de usuario. React permite la creación de componentes reutilizables y proporciona una forma eficiente de actualizar y renderizar componentes cuando los datos cambian. Su arquitectura basada en componentes y el uso del Virtual DOM optimizan el rendimiento de las aplicaciones web dinámicas. React.js facilita el desarrollo de interfaces de usuario complejas y de alta interacción, promoviendo la reutilización de componentes y la separación de preocupaciones. El enfoque declarativo de React simplifica la programación y mejora la previsibilidad del código, lo que resulta en aplicaciones más mantenibles y escalables. Además, React cuenta con un amplio respaldo de la comunidad y un ecosistema robusto de herramientas y bibliotecas que facilitan su integración y uso en proyectos de diversos tamaños (React.dev, 2023).

2.5.4 PostgreSQL

PostgreSQL es un motor para la gestión de bases de datos relacional orientado a objetos, reconocido por su estabilidad, solidez y capacidad de extensión. Admite transacciones ACID (Atomicidad, Consistencia, Aislamiento y Durabilidad), garantizando así la integridad y consistencia de los datos. PostgreSQL es altamente escalable y se utiliza en aplicaciones que requieren alta disponibilidad y manejo eficiente de grandes volúmenes de datos. PostgreSQL ofrece soporte avanzado para consultas SQL, índices, integridad referencial y procedimientos almacenados. Su capacidad para manejar datos geoespaciales, JSON y otros tipos de datos no estructurados lo hace una opción versátil para una amplia gama de aplicaciones. Además, "PostgreSQL implementa un mecanismo de recuperación conocido como Write-Ahead

Logging (WAL) que garantiza la durabilidad y recuperación de datos ante fallos del sistema" (Timescale, 2023).

2.5.5 Sequelize (ORM)

Sequelize es un ORM (Object-Relational Mapping) para Node.js que facilita la interacción con bases de datos SQL como PostgreSQL. Permite a los desarrolladores trabajar con datos utilizando objetos en lugar de consultas SQL, simplificando la manipulación de datos y mejorando la productividad. Sequelize soporta características avanzadas como asociaciones, transacciones y validaciones, y su sintaxis declarativa facilita la definición de modelos de datos y su relación, mejorando la claridad y mantenibilidad del código. Además, "Sequelize es ampliamente adoptado en la comunidad de desarrollo debido a su flexibilidad y la capacidad de soportar múltiples dialectos SQL, lo que lo convierte en una herramienta versátil para diversas aplicaciones" (Sequelize, 2023).

2.5.6 Bootstrap

Bootstrap es un framework de front-end de código abierto que proporciona estilos CSS y componentes de UI predefinidos, facilitando el desarrollo de interfaces web responsivas y atractivas. Fue desarrollado por Twitter y se ha convertido en una herramienta esencial para el diseño de sitios web y aplicaciones móviles. Bootstrap incluye un sistema de grid flexible, componentes de UI personalizables y utilidades CSS, lo que permite a los desarrolladores crear rápidamente diseños consistentes y adaptables a diferentes dispositivos y tamaños de pantalla. Además, "Bootstrap es uno de los framework más populares en el desarrollo web, respaldado por una amplia comunidad de desarrolladores que contribuyen continuamente con mejoras y nuevos componentes" (Bootstrap, 2023).

2.5.7 Draw.io

Draw.io, también conocido como diagrams.net, es una herramienta gratuita en línea para la creación de diagramas, que permite a los usuarios generar una amplia variedad de gráficos y diagramas directamente desde su navegador web. Esta herramienta es muy útil para el desarrollo de sistemas web, ya que permite a los desarrolladores visualizar la arquitectura del sistema, los flujos de trabajo, las relaciones de base de datos y más.

Draw.io es compatible con una amplia gama de tipos de diagramas, incluyendo diagramas de flujo, diagramas de red, diagramas UML, diagramas ER, y diagramas de arquitectura de software, entre otros. Los usuarios pueden arrastrar y soltar formas y líneas en el lienzo, y luego

personalizarlas con texto, colores, y más. Los diagramas pueden ser exportados en varios formatos

2.6 Conceptos Clave

Los conceptos clave formarán la base de las funcionalidades del sistema. Estos conceptos incluyen la gestión de usuarios, inventario, clientes, contratos y ventas.

2.6.1 Gestión de Usuarios

La gestión de usuarios es un componente esencial de cualquier sistema que requiera acceso controlado y personalizado. Incluye funcionalidades como la autenticación y la autorización.

- **Autenticación y Autorización:** La autenticación es el procedimiento de confirmar la identidad de un usuario. La autorización, por otro lado, es el proceso de determinar si un usuario autenticado tiene los permisos necesarios para acceder a determinados recursos o ejecutar acciones específicas. Según Sommerville (2011), la autenticación y la autorización son componentes críticos de la seguridad del sistema, ya que aseguran que solo los usuarios autorizados puedan acceder a los recursos sensibles.

2.6.2 Gestión de Productos

La gestión de productos es crucial para cualquier negocio que maneje productos físicos. Involucra el seguimiento y control de los productos en stock, asegurando que la cantidad de productos sean suficientes para satisfacer la demanda.

- **Clasificación de Productos:** Los productos pueden clasificarse por categorías para facilitar su gestión y búsqueda. Esta clasificación puede basarse en el tipo de producto, la marca, el proveedor, etc.

2.6.3 Gestión de Clientes

La gestión de clientes se centra en mantener registros detallados de todos los clientes, incluyendo información de contacto, historial de compras y preferencias.

- **Registro y Seguimiento de Clientes:** Registrar información detallada de los clientes permite un seguimiento más efectivo de sus interacciones y compras, lo que facilita una personalización del servicio y una mejor atención al cliente.
- **Historial de Compras:** Mantener un historial de compras ayuda a identificar patrones de comportamiento, lo que puede ser útil para promociones y

marketing personalizado. Según Connolly y Begg (2015), el análisis del historial de compras de los clientes puede proporcionar valiosos insights que mejoren la toma de decisiones y la estrategia de negocio.

2.6.4 Gestión de Contratos

La gestión de contratos es importante en negocios que operan con acuerdos formales con clientes o proveedores. Incluye la creación, administración y seguimiento de contratos.

- **Creación y Administración de Contratos:** El sistema debe permitir la creación de contratos detallados que incluyan términos, condiciones y fechas importantes. Estos contratos deben ser fáciles de acceder y actualizar según sea necesario.
- **Seguimiento de Contratos:** El seguimiento de las fechas de vencimiento y la renovación automática o manual de los contratos aseguran que no haya interrupciones en las relaciones comerciales. Boehm (1988) señala que la gestión efectiva de contratos ayuda a mantener la transparencia y la confianza entre las partes involucradas.
- **Historial de Términos y Fechas:** Mantener un historial detallado de los términos y fechas de los contratos permite una referencia fácil y asegura que se cumplan todas las condiciones acordadas.

2.6.5 Gestión de Ventas

La gestión de ventas implica registrar y administrar todas las transacciones de venta, así como generar facturas y recibos.

- **Registro y Administración de Transacciones:** Un sistema eficiente debe permitir registrar todas las ventas de manera detallada, incluyendo información sobre el cliente, los productos vendidos y los precios.
- **Generación de Facturas y Recibos:** La capacidad de generar facturas y recibos automáticamente reduce el tiempo administrativo y mejora la precisión. Según Pressman (2014), la automatización en la generación de documentos es beneficiosa ya que puede ahorrar tiempo, reducir errores humanos y mejorar la satisfacción del cliente.

2.7 Base de Datos

Las bases de datos son componentes esenciales en el desarrollo de aplicaciones modernas, permitiendo almacenar, gestionar y recuperar datos de manera eficiente y segura. Este subcapítulo aborda la definición de una base de datos, los tipos de bases de datos más comunes y las ventajas de utilizar bases de datos relacionales.

2.7.1 Definición de Base de Datos

Una base de datos es una colección estructurada de datos, almacenados y accesibles electrónicamente a través de un sistema informático. Estas bases de datos son gestionadas por sistemas de gestión de bases de datos (DBMS, por sus siglas en inglés), que proporcionan las herramientas y servicios necesarios para crear, gestionar y mantener los datos. Según Connolly y Begg (2015), una base de datos es una colección de datos relacionados de forma lógica, y un DBMS es un software que permite a los usuarios definir, crear, mantener y controlar el acceso a la base de datos.

2.7.2 Tipos de Bases de Datos

Existen varios tipos de bases de datos, cada una diseñada para satisfacer diferentes necesidades y aplicaciones. Los dos tipos de bases de datos más comunes son:

- Bases de Datos Relacionales: Organizan los datos en tablas que se vinculan entre sí mediante claves primarias y foráneas. Estas bases de datos son altamente estructuradas y emplean el lenguaje SQL (Structured Query Language) para la gestión y consulta de datos. Algunos ejemplos incluyen MySQL, PostgreSQL y Oracle Database.
- Bases de Datos NoSQL: Diseñadas para grandes cantidades de datos los cuales pueden ser estructurados o semiestructurados, proporcionando flexibilidad y escalabilidad horizontal. Incluyen bases de datos de documentos, clave-valor, de columnas y de grafos. Ejemplos: MongoDB, Cassandra, Redis, Neo4j.

2.7.3 Ventajas de Usar Bases de Datos Relacionales

Las bases de datos relacionales (RDBMS) ofrecen varias ventajas que las hacen ideales para variedad de soluciones empresariales y comerciales. Entre las principales ventajas se incluyen:

- **Integridad de Datos:** Las bases de datos relacionales garantizan la integridad y consistencia de los datos mediante restricciones y reglas de integridad referencial. Esto asegura que los datos sean precisos y estén relacionados correctamente.
- **Flexibilidad en Consultas:** SQL, el lenguaje estándar para las bases de datos relacionales, permite realizar consultas complejas y ad hoc, proporcionando una gran flexibilidad en la manipulación y recuperación de datos.
- **Escalabilidad Vertical:** Aunque las bases de datos relacionales pueden ser más difíciles de escalar horizontalmente que las bases de datos NoSQL, pueden manejar grandes volúmenes de datos y transacciones a través de la escalabilidad vertical, mediante la adición de más recursos a un solo servidor.
- **Transacciones ACID:** Las bases de datos relacionales soportan transacciones ACID (Atomicidad, Consistencia, Aislamiento y Durabilidad), lo que asegura que todas las operaciones dentro de una transacción se completen correctamente y que los datos se mantengan en un estado consistente.
- **Seguridad:** Los RDBMS proporcionan robustas características de seguridad, incluyendo control de acceso basado en roles, autenticación, y cifrado de datos, lo que ayuda a proteger la información sensible y cumplir con las regulaciones de privacidad de datos.

De acuerdo con Connolly y Begg (2015) se entiende que las bases de datos relacionales no solo facilitan la organización y recuperación de grandes cantidades de datos, sino que también aseguran la integridad y consistencia de los datos a través de transacciones ACID y restricciones de integridad.

En este proyecto, utilizaremos PostgreSQL como sistema de gestión de bases de datos relacional debido a su robustez, soporte de transacciones ACID, y su capacidad para manejar grandes volúmenes de datos y operaciones complejas.

2.8 Definiciones Importantes

En el desarrollo de un sistema de gestión comercial web, es crucial entender diversos términos y conceptos fundamentales que se emplearán durante todo el proyecto. Estos términos incluyen sistema de información, aplicación web, arquitectura cliente-servidor y API RESTful.

2.8.1 Sistema de Información

Los sistemas de información, esenciales para el funcionamiento eficiente de las empresas modernas, pueden ser manuales o apoyarse en tecnologías de la información. Un sistema de información se compone de diversos componentes interconectados que recolectan, procesan, almacenan y distribuyen datos, con el objetivo de apoyar la toma de decisiones y el control dentro de una organización (Laudon & Laudon, 2012).

2.8.2 Aplicación Web

Una aplicación web es un software que se ejecuta en un servidor web y es accesible a través de un navegador web mediante una red como Internet o una intranet. Las aplicaciones web pueden variar desde simples páginas informativas hasta complejas plataformas interactivas que permiten la interacción dinámica con el usuario.

El desarrollo de la World Wide Web ha tenido un impacto profundo en nuestras vidas, transformando la manera en que se organizan las aplicaciones empresariales, ahora altamente distribuidas y accesibles globalmente a través de navegadores web (Sommerville, 2011).

2.8.3 Cliente-Servidor

La arquitectura cliente-servidor es un modelo de red distribuida donde las tareas y las cargas de trabajo se dividen entre los servidores, que son los proveedores de recursos o servicios, y los clientes, que son los que solicitan estos servicios. Los clientes realizan solicitudes a los servidores, que luego procesan esas solicitudes y devuelven las respuestas adecuadas. Este paradigma de computación distribuida separa las funciones de cliente y servidor, donde los clientes son dispositivos o aplicaciones que solicitan servicios y los servidores son quienes los proporcionan (Tanenbaum & Van Steen, 2007).

2.8.4 API RESTful

Una API RESTful (Interfaz de Programación de Aplicaciones Representational State Transfer) es un tipo de interfaz web que se basa en los principios de la arquitectura REST. REST es un estilo de arquitectura que utiliza métodos HTTP y URL para acceder y manipular recursos representados en forma de datos. Las API RESTful son utilizadas debido a la simplicidad, escalabilidad y compatibilidad con una amplia variedad de plataformas y lenguajes de programación. Estas APIs se adhieren a los principios de REST, facilitando la comunicación entre sistemas mediante solicitudes y respuestas HTTP, y son apreciadas por su simplicidad y capacidad de escalar (Fielding, 2000).

2.9 Seguridad en Aplicaciones Web

La seguridad en aplicaciones web es una prioridad esencial en el desarrollo de software. Los sistemas web están expuestas a una variedad de amenazas y ataques que pueden comprometer la integridad, confidencialidad y disponibilidad de los datos y servicios. Este subcapítulo aborda los conceptos clave de autenticación y autorización, los ataques comunes y las buenas prácticas de seguridad.

2.9.1 Autenticación y Autorización

La autenticación y la autorización son elementos importantes para la seguridad de los sistemas web. La autenticación implica verificar la identidad del usuario, mientras que la autorización define los permisos y accesos que se le otorgan a ese usuario una vez autenticado dentro del sistema.

- Autenticación: Este proceso asegura que el usuario que intenta acceder al sistema es quien dice ser. Los métodos de autenticación pueden incluir contraseñas, tokens, certificados digitales y autenticación multifactor (MFA). Según Sommerville (2011) podemos entender que, la autenticación es el primer paso en la seguridad de un sistema, proporcionando una base sobre la cual se pueden aplicar políticas de autorización y control de acceso.
- Autorización: Una vez que un usuario está autenticado, la autorización determina qué acciones puede realizar dentro del sistema. Esto se gestiona mediante políticas de control de acceso que definen permisos basados en roles o atributos. La autorización asegura que los usuarios solo puedan acceder a los recursos necesarios para sus funciones.

2.9.2 Protección contra Ataques Comunes

Las aplicaciones web son susceptibles a varios tipos de ataques. El ataque más común a aplicaciones web es la inyección de SQL.

- Inyección de SQL: Este ataque ocurre cuando un atacante inserta o "inyecta" una consulta SQL maliciosa a través de la entrada de un usuario. Esto puede resultar en la manipulación o acceso no autorizado a la base de datos. Pressman (2014) menciona que la validación de entradas y el uso de consultas parametrizadas son métodos efectivos para prevenir las inyecciones de SQL.

- Cross-Site Scripting (XSS): XSS es un ataque en el que un atacante inyecta scripts maliciosos en contenido que se envía a otro usuario. Estos scripts pueden robar información sensible, como cookies de sesión, o realizar acciones en nombre del usuario. La sanitización de entradas y la correcta codificación de salida pueden mitigar este riesgo.

2.9.3 Buenas Prácticas de Seguridad

Implementar buenas prácticas de seguridad es fundamental para proteger las aplicaciones web de posibles amenazas y vulnerabilidades. Algunas de estas prácticas incluyen:

- Validación de Entradas: Siempre validar y limpiar las entradas del usuario para evitar la inyección de código malicioso. Esto incluye la validación del lado del cliente y del servidor.
- Uso de HTTPS: Asegurar la comunicación entre el cliente y el servidor mediante el uso de HTTPS, lo que cifra los datos en tránsito y protege contra ataques de intermediarios.
- Gestión Segura de Sesiones: Implementar medidas para gestionar las sesiones de manera segura, incluyendo el uso de cookies seguras, tiempos de expiración de sesión adecuados y protección contra el secuestro de sesiones.
- Autenticación Multifactor (MFA): Añadir una capa adicional de seguridad mediante la implementación de MFA, que requiere que los usuarios proporcionen más de un método de autenticación.
- Actualización Regular de Software: Mantener el software actualizado con los últimos parches y actualizaciones de seguridad para protegerse contra vulnerabilidades conocidas.
- Pressman (2014) menciona que las mejores prácticas de seguridad, como la validación de entradas y el uso de autenticación multifactor, son esenciales para proteger las aplicaciones web contra una amplia gama de amenazas y vulnerabilidades.

2.10 Principios SOLID

Los principios SOLID son un conjunto de cinco principios de diseño orientado a objetos que buscan mejorar la robustez y flexibilidad del software. Estos principios fueron introducidos por Robert C. Martin y son fundamentales para escribir código limpio y mantenible.

- S (Principio de Responsabilidad Única): Una clase debe tener una única responsabilidad y, por lo tanto, una única razón para cambiar. Este principio ayuda a mantener el código modular y fácil de entender.
- O (Principio Abierto/Cerrado): Las entidades de software, como clases, módulos y funciones, deben permitir la extensión de su comportamiento sin necesidad de modificar su código fuente. Esto implica que pueden ser ampliadas pero no alteradas en su forma original.
- L (Principio de Sustitución de Liskov): Los objetos de una clase derivada deben ser sustituibles por objetos de la clase base sin alterar el correcto funcionamiento del sistema. Este principio asegura que las clases derivadas puedan reemplazar a las clases base sin problemas.
- I (Principio de Segregación de Interfaces): Los clientes no deben verse forzados a depender de interfaces que no utilizan. Este principio aboga por el uso de interfaces específicas y pequeñas en lugar de interfaces grandes y generales.
- D (Principio de Inversión de Dependencias): Los módulos de nivel superior no deberían depender de los módulos de nivel inferior; en su lugar, ambos deberían depender de abstracciones. Asimismo, las abstracciones no deberían depender de los detalles, sino que los detalles deberían depender de las abstracciones.

Los principios SOLID ofrecen una guía para diseñar software orientado a objetos que sea robusto, flexible y fácil de mantener, ayudando a los desarrolladores a crear sistemas sostenibles y escalables (Martin, 2003).

3 ANÁLISIS Y DISEÑO DEL SISTEMA

3.1 Introducción

El capítulo está enfocado en el análisis y diseño del sistema de gestión comercial web para negocios con modelos de ‘compra, venta y consignación’. Este capítulo aborda el proceso desde la recolección de los requerimientos hasta el diseño detallado del sistema. Se describen las metodologías utilizadas, los análisis de los requerimientos, y los diseños del sistema, la base de datos, y las interfaces tanto del frontend como del backend.

El objetivo principal de este capítulo es proporcionar una visión clara y detallada de cómo se llevó a cabo el análisis y diseño del sistema, asegurando que cumple con los requerimientos del negocio ejemplo y proporciona una solución eficaz para la gestión de sus operaciones. Se cubren aspectos clave como el levantamiento de requerimientos a través de entrevistas, el

análisis y la definición de estos requerimientos, y el diseño detallado del sistema, incluyendo la arquitectura, la base de datos y las interfaces de usuario y servidor.

3.2 Levantamiento de Requerimientos

Para comprender las necesidades y expectativas de un sistema de información orientado a empresas con un modelo de negocio de ‘compra, venta y consignación’, se realizó una entrevista con el gerente general Roberto Balseca de “El Gran Ahorro”, el cual cuenta con 20 años de experiencia en este tipo de modelo de negocio. Esta entrevista fue fundamental para identificar los requerimientos funcionales y no funcionales del sistema. A continuación, se presenta un resumen de la entrevista, incluyendo las preguntas realizadas y los puntos importantes de las respuestas obtenidas.

Preguntas de la Entrevista:

1. ¿Cuál es el nombre del negocio y quiénes son los principales responsables de la gestión?

   Respuesta: El negocio se llama "El Gran Ahorro". Los principales responsables de la gestión es el gerente general, y el gerente financiero de ventas.

2. Describa brevemente las operaciones principales del negocio (compra, venta, consignación).

   Respuesta: La empresa se dedica a la compraventa y consignación de muebles de hogar, oficina, equipo industrial de cocina, restaurantes, supermercados, electrodomésticos y computadores.

3. ¿Cuántos empleados trabajan actualmente en el negocio y qué roles desempeñan?

   Respuesta: Actualmente, hay 2 administrativos, 4 de apoyo y 1 de transporte.

4. ¿Cómo se clasifican los productos (por categorías, estado, precio, etc.)?

   Respuesta: Los productos se clasifican por categorías como muebles de hogar, muebles de oficina, equipos industriales, electrodomésticos, etc. Además, se clasifican por subcategorías como tipo, marca y proveedor.

5. ¿Desea que el sistema maneje el inventario de productos? De ser así, ¿qué detalles específicos necesita registrar de cada producto (marca, modelo, estado, precio de compra, precio de venta, etc.)?

Respuesta: Sí, el sistema debe manejar el inventario. Se deben registrar detalles como título, categoría, subcategoría, cantidad, descripción, valor unitario, valor total y observaciones.

6. ¿El sistema debe permitir registrar las transacciones de compra y venta? ¿Qué información específica se requiere para cada transacción?

   Respuesta: Sí, se debe registrar el nombre del proveedor o cliente, número de cédula o RUC, número telefónico, dirección, tipo de transacción (compra o consignación), y detalles del producto (precio unitario y total).

7. ¿Cómo se gestionan actualmente las consignaciones? ¿Qué procesos y datos específicos debería manejar el sistema para las consignaciones?

   Respuesta: Las consignaciones se gestionan a través de comunicación inicial por WhatsApp o correo electrónico, visita física para verificación del artículo, y firma de contrato de consignación válido por 60 días renovables hasta 180 días. El sistema debe gestionar la información del proveedor, el estado del artículo, y las fechas de consignación.

8. ¿Qué métodos de pago desea integrar en el sistema (efectivo, tarjetas, transferencias bancarias, etc.)?

   Respuesta: El sistema debe soportar pagos en efectivo, transferencias bancarias, cheques y canjes.

9. ¿Necesita el sistema generar informes? Si es así, ¿qué tipo de informes (ventas diarias, inventario, consignaciones, etc.)?

   Respuesta: Sí, el sistema debe generar informes de ventas diarias y mensuales, informes de contratos, reportes de inventario y de liquidaciones de contratos.

10. ¿Cómo desea gestionar los usuarios del sistema (roles, permisos, etc.)?

    Respuesta: Los usuarios deben ser gestionados con roles y permisos. Los administradores deben tener acceso completo, mientras que los ayudantes pueden consultar stocks y precios.

Puntos Importantes de las Respuestas:

- Gestión de Inventario: Clasificación por categorías y subcategorías, registro de detalles específicos como estado, cantidad y precios.
- Gestión de Transacciones: Registro detallado de compras y consignaciones, incluyendo datos de proveedores y clientes.
- Métodos de Pago: Soporte para múltiples métodos de pago, incluyendo efectivo, transferencias, cheques y canjes.
- Generación de Informes: Necesidad de informes detallados para ventas, inventario, contratos y liquidaciones.
- Gestión de Usuarios: Diferenciación de roles y permisos, con acceso específico para administradores y ayudantes.

Esta información recolectada en la entrevista proporciona una base sólida para el análisis y diseño del sistema, asegurando que se cumplan las necesidades y expectativas del negocio.

3.3 Análisis de Requerimientos

El análisis de requerimientos permite identificar y documentar las necesidades y expectativas del usuario final. En este subcapítulo, se detallarán los requerimientos funcionales y no funcionales del sistema de gestión comercial web para empresas con un modelo de negocio de compra, venta y consignación. Estos requerimientos se basan en la información recolectada durante el levantamiento de requerimientos y la entrevista con el gerente del negocio.

3.3.1 Requerimientos Funcionales

Los requerimientos funcionales describen las funcionalidades específicas que el sistema va a proporcionar:

1. Gestión de Usuarios
    - RF1: El sistema debe permitir la modificación de usuarios.
    - RF2: Incluye la autenticación de usuarios (login/logout).
    - RF3: Gestionar la configuración y parámetros del sistema.
2. Gestión de Productos
    - RF4: El sistema debe permitir la clasificación de productos por categorías.
    - RF5: El sistema debe registrar y controlar los costos de adquisición de cada producto.
    - RF6: El sistema debe gestionar los precios de venta de los productos.
    - RF7: El sistema debe proporcionar un control de stock en tiempo real.

3. Gestión de Clientes
    - RF8: El sistema debe permitir el registro de clientes, mantenimiento de información detallada y clasificación de clientes.
    - RF9: El sistema debe mantener un historial de compras detallado.
4. Gestión de Contratos
    - RF10: El sistema debe permitir la creación y administración de contratos de consignación, incluyendo su seguimiento.
    - RF11: El sistema debe mantener un historial detallado de los términos y fechas de los contratos.
5. Gestión de Ventas
    - RF12: El sistema debe permitir la administración de transacciones de venta.
    - RF13: El sistema debe gestionar reservas.
6. Gestión de Compras
    - RF14: El sistema debe permitir la administración de transacciones compras.
    - RF15: El sistema debe mantener el inventario actualizado constantemente.

3.3.2 Requerimientos No Funcionales

Requerimientos no funcionales describen características y restricciones para el sistema que no están relacionadas directamente con las funcionalidades específicas, pero que son cruciales para la operatividad y calidad del sistema.

1. Escalabilidad
    - RNF1: El sistema debe ser escalable para manejar el crecimiento en el número de usuarios y transacciones sin necesidad de rediseñar la arquitectura.
2. Seguridad
    - RNF2: El sistema debe utilizar protocolos de seguridad estándar como HTTPS.
    - RNF3: Los datos sensibles, como contraseñas, deben ser almacenados de forma encriptada.
3. Usabilidad
    - RNF4: La interfaz del usuario debe ser intuitiva y fácil de usar para minimizar la curva de aprendizaje.
4. Mantenibilidad
    - RNF5: El sistema debe estar diseñado de manera modular para facilitar su mantenimiento y actualización.

- RNF6: El código fuente debe seguir las mejores prácticas de codificación y estar bien documentado.

5. Compatibilidad
   - RNF7: El sistema debe ser compatible con los navegadores web más utilizados, incluyendo Chrome, Firefox, Safari y Edge.
   - RNF8: El sistema debe ser accesible desde dispositivos móviles y de escritorio.

Estos requerimientos proporcionan una guía detallada para el diseño y desarrollo del sistema, asegurando que se cumplan las expectativas del negocio y se proporcionen las funcionalidades necesarias para la gestión eficiente de sus operaciones.

3.4 Diseño del Sistema

El diseño del sistema es una fase crucial en el desarrollo del sistema de gestión comercial web, ya que proporciona una guía detallada para la implementación del sistema. Este subcapítulo aborda la arquitectura del sistema, el diseño de la base de datos, el diseño del frontend y el backend, y el caso de uso UML.

3.4.1 Arquitectura del Sistema

La arquitectura del sistema establece la estructura y los componentes principales del mismo, así como sus interacciones. Para este proyecto, se ha adoptado una arquitectura de tres capas, que divide las responsabilidades en tres niveles distintos: presentación, lógica de negocio y datos.

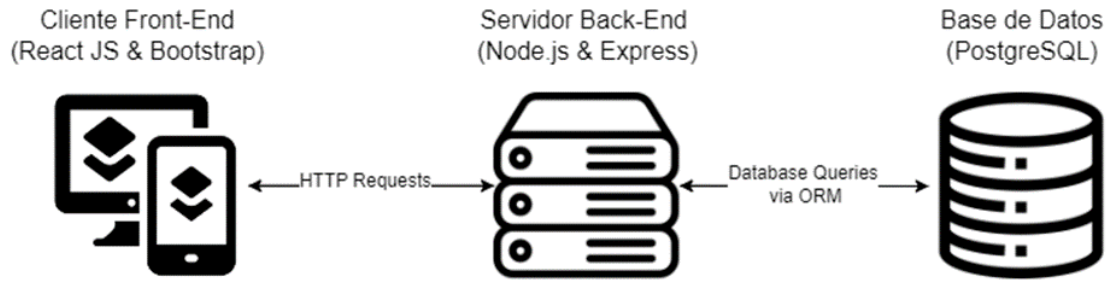

Figure 1 Arquitectura Tres Capas – Elaboración Propia

3.4.1.1 Descripción de la Arquitectura

La arquitectura de tres capas ofrece una clara separación de responsabilidades, lo que facilita la escalabilidad, el mantenimiento y el desarrollo simultáneo del sistema. Las tres capas son:

1. Capa de Presentación: Esta capa abarca la interfaz de usuario del sistema, creada utilizando React.js y Bootstrap. Es responsable de la interacción con el usuario y la visualización de datos.
2. Capa de Lógica de Negocio: Esta capa gestiona la lógica de la aplicación y las reglas de negocio. Se implementa con Node.js y Express.js, manejando las solicitudes del cliente, procesando los datos y aplicando las reglas de negocio.
3. Capa de Datos: Esta capa incluye la base de datos, en este caso, PostgreSQL, y el acceso a los datos. Se utiliza Sequelize como ORM para interactuar con la base de datos de manera eficiente.

3.4.1.2 Justificación de la Arquitectura Elegida

Se optó por la arquitectura de tres capas debido a sus ventajas en cuanto a la separación de responsabilidades, escalabilidad y mantenibilidad. Esta arquitectura permite desarrollar y mantener cada capa de manera independiente, lo que facilita la actualización y el escalado del sistema sin impactar las demás capas. Además, el uso de tecnologías modernas como React.js, Node.js, Express.js y PostgreSQL asegura un rendimiento eficiente y una experiencia de usuario fluida.

3.4.2 Diseño de la Base de Datos

El diseño de la base de datos es fundamental para asegurar que los datos se almacenen de manera eficiente y se puedan acceder de forma óptima. El modelo entidad-relación (ER) se utiliza para diseñar la estructura de la base de datos, definiendo las entidades, sus atributos y las relaciones entre ellas.

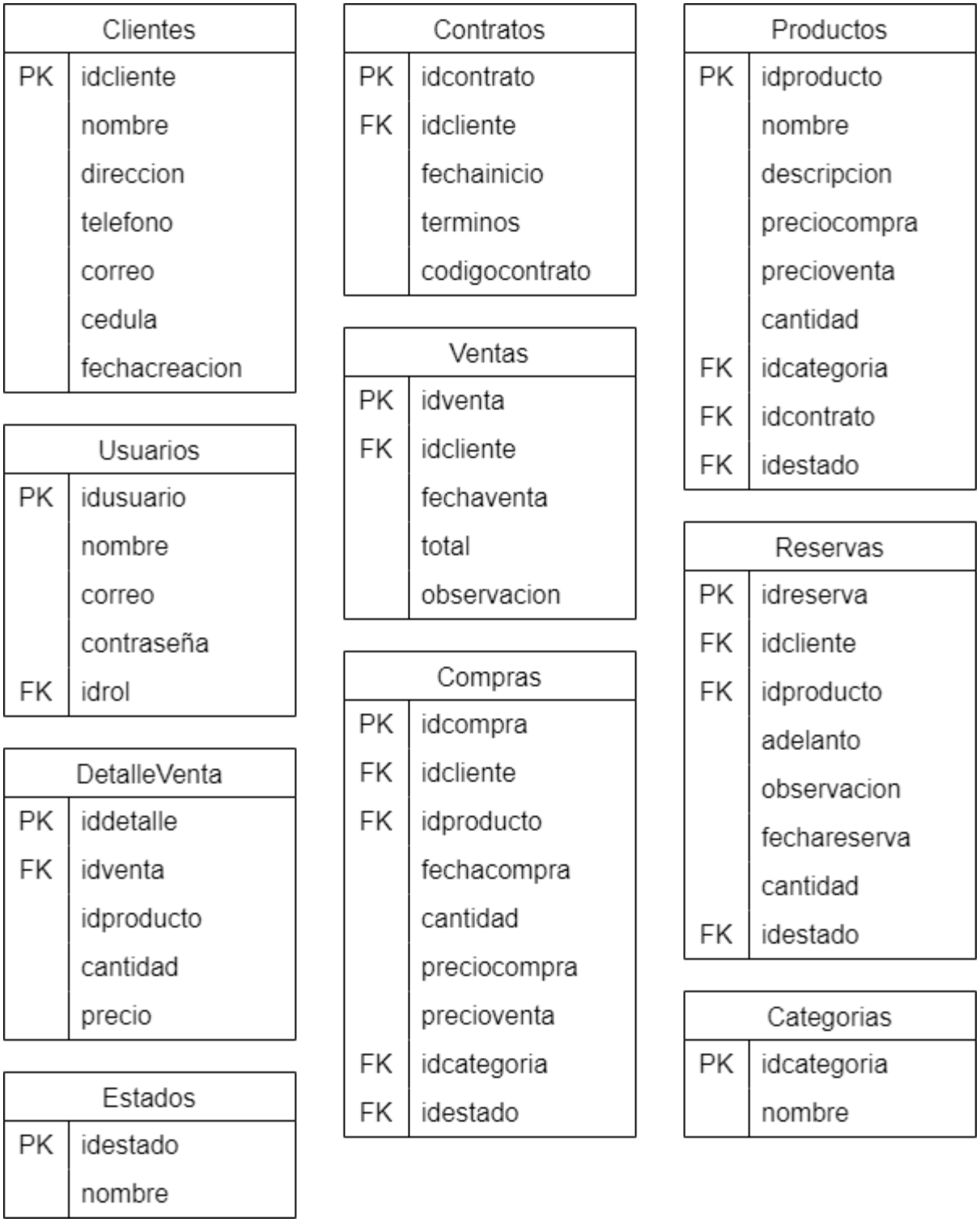

Figure 2 Esquema No Relacionado – Elaboración Propia

3.4.2.1 Modelo Entidad-Relación (ER)

El modelo ER para el sistema de gestión comercial incluye las siguientes entidades principales: Usuarios, Productos, Categorías, Clientes, Contratos, Compras, Ventas, Detalle

Ventas, Estados y Reservas. Cada entidad tiene sus atributos específicos y relaciones definidas con otras entidades.

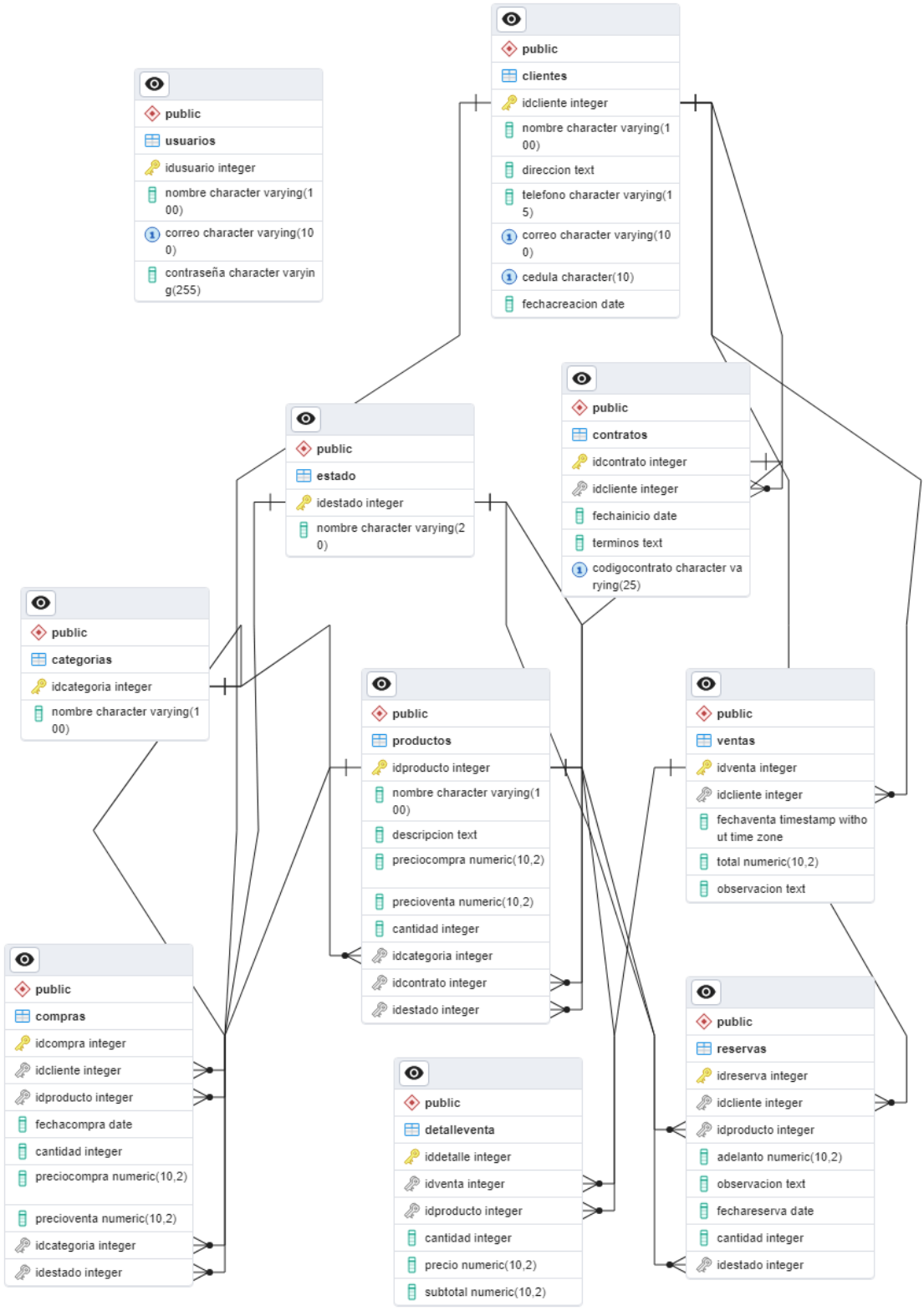

Figure 3 Modelo Entidad Relación – Obtenido de pgAdmin4

3.4.3 Diseño del Frontend

El diseño del frontend se centra en la creación de una interfaz de usuario intuitiva y responsiva que permita a los usuarios interactuar fácilmente con el sistema. Se utiliza React.js para desarrollar componentes reutilizables y Bootstrap para el diseño responsivo.

3.4.3.1 Estructura de Componentes React

La estructura de componentes en React se diseña de manera jerárquica, con componentes padres que contienen y gestionan los estados de los componentes hijos. Los componentes principales incluyen:

- App Component: El componente raíz que contiene la estructura principal de la aplicación.
- Sidebar Component: Componente para la barra lateral de navegación.
- Usuario Component: Componente para la gestión de usuarios.
- Producto Component: Componente para la gestión de productos y categorías.
- Cliente Component: Componente para la gestión de clientes.
- Contrato Component: Componente para la gestión de contratos.
- Venta Component: Componente para la gestión de ventas y detalle venta.
- Compra Component: Componente para la gestión de compras.
- Reserva Component: Componente para la gestión de reservas.

3.4.4 Diseño del Backend

El diseño del backend se centra en la implementación de la lógica de negocio y el manejo de las solicitudes del cliente. Se utiliza Node.js con Express.js para gestionar las rutas y controladores, y Sequelize para interactuar con la base de datos.

3.4.4.1 Estructura de Rutas y Controladores en Express

La estructura de rutas y controladores en Express se organiza de manera modular para facilitar la mantenibilidad y escalabilidad. Las principales rutas incluyen:

- /api/usuarios: Ruta para la gestión de usuarios.
- /api/productos: Ruta para la gestión de productos.
- /api/categorías: Ruta para la gestión de categorías.
- /api/clientes: Ruta para la gestión de clientes.
- /api/contratos: Ruta para la gestión de contratos.

- /api/ventas: Ruta para la gestión de ventas.
- /api/detalleventa: Ruta para la gestión de detalles ventas.
- /api/compras: Ruta para la gestión de compras.
- /api/reservas: Ruta para la gestión de reservas.

Cada ruta tiene sus controladores correspondientes que manejan las solicitudes HTTP y aplican la lógica de negocio.

3.4.5 Caso de Uso UML

El diagrama de caso de uso UML proporciona una representación visual de las interacciones entre los usuarios y el sistema, destacando los principales casos de uso. El diagrama de casos de uso muestra cómo los actores interactúan con el sistema para llevar a cabo estas tareas. Los actores principales son el administrador y los ayudantes.

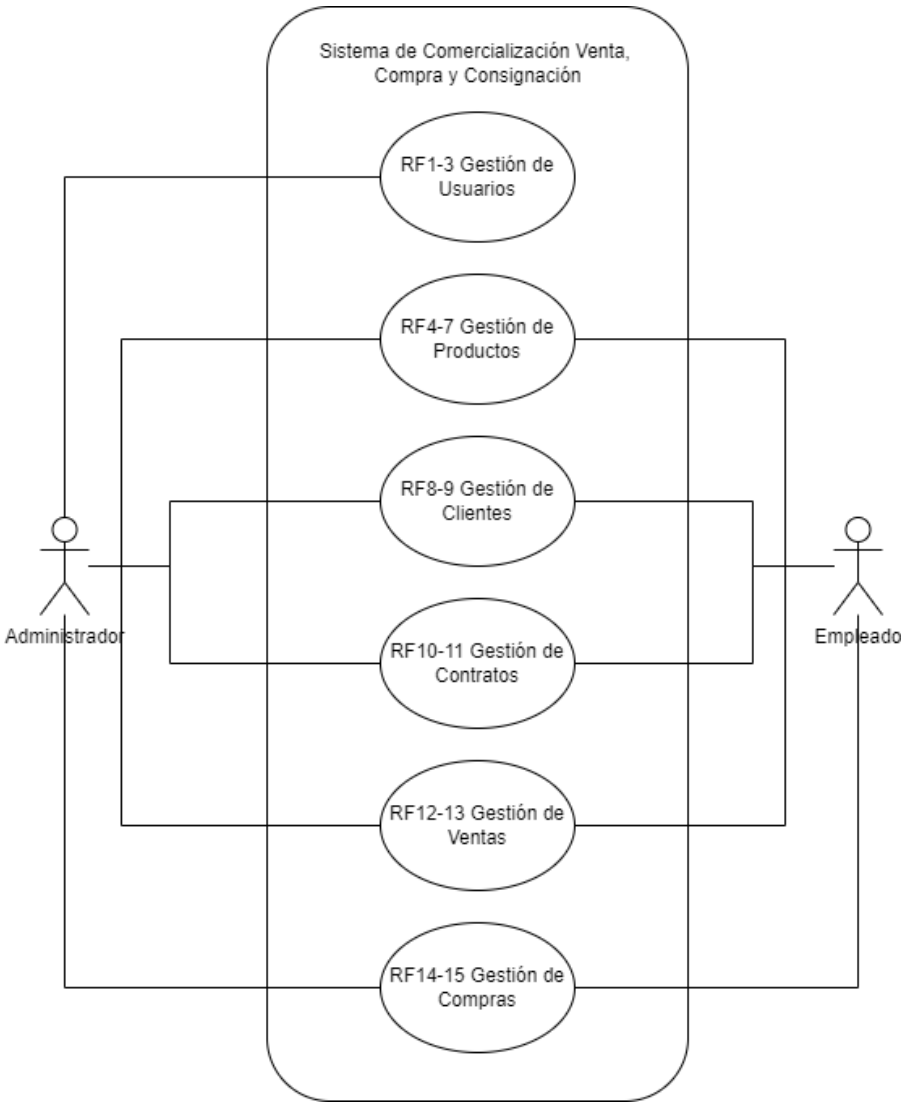

Figure 4 Caso de Uso UML Sistema Comercialización – Elaboración Propia

4 DESARROLLO DEL SISTEMA

El desarrollo del sistema se va a llevar a cabo aplicando la metodología Scrum, debido al tiempo y el personal del proyecto se va a modificar la metodología para poder aplicarla en el contexto. Se extrae los siguientes lineamientos aplicables:

- Creación de Backlog
- Se establece un plazo de 3 semanas para cada sprint
- Planificación del Sprint

Al trabajar con Scrum, los objetivos a cumplir se dividen en iteraciones llamadas sprints. Los sprints (iteraciones) van a tener un plazo de 3 semanas cada uno, y un total de 3 sprints. Se inicia con una planificación en cada sprint la cual define los objetivos y el alcance del sprint, y se eligen las tareas que se realizarán en la iteración. Al finalizar cada sprint, se llevan a cabo actividades para asegurar la calidad y la mejora continua del proyecto. Estas actividades son:

1. Revisión del Sprint (Sprint Review):
    - Objetivo: Inspeccionar el incremento de producto entregado y adaptar el Product Backlog si es necesario.
    - Participantes: Todo el equipo, incluyendo el Product Owner, el Scrum Master, y cualquier stakeholder interesado.
    - Actividades:
        - Demostración del trabajo completado durante el sprint.
        - Recopilación de feedback de los stakeholders.
        - Discusión sobre las posibles mejoras y ajustes necesarios.
2. Retrospectiva del Sprint:
    - Objetivo: Reflexionar sobre el sprint que acaba de terminar y planificar mejoras para los próximos sprints.
    - Participantes: Todo el equipo Scrum.
    - Actividades:
        - Evaluación de lo que funcionó bien y lo que no funcionó.
        - Identificación de áreas de mejora.
        - Desarrollo de un plan de acción para implementar las mejoras identificadas.

3. Actualización del Product Backlog:

    - Objetivo: Mantener el Product Backlog actualizado con nuevas historias de usuario y prioridades ajustadas.
    - Participantes: Product Owner y equipo de desarrollo.
    - Actividades:
        - Refinamiento de las historias de usuario basadas en el feedback recibido.
        - Priorización de las historias de usuario para el próximo sprint.

4. Planificación del Próximo Sprint:

    - Objetivo: Definir los objetivos y el alcance del siguiente sprint.
    - Participantes: Todo el equipo Scrum.
    - Actividades:
        - Selección de las historias de usuario del Product Backlog que se abordarán en el próximo sprint.
        - Estimación de tiempo y recursos necesarios para completar las tareas.

En este proyecto, el experto en negocios con experiencia en modelos de compra, venta y consignación es considerado un stakeholder. Yo asumiré los roles de Product Owner y del equipo de desarrollo. El Product Owner se encarga de definir y gestionar el Product Backlog, una lista priorizada de los requisitos del producto. También será responsable de estimar cómo se llevará a cabo el trabajo y de entregar un incremento del producto al final de cada sprint.

4.1 Backlog

Se crea el Backlog inicial:

Historial	Estado
H1 Crear la estructura de usuario y login del sistema de seguridad	PENDIENTE
H2 Crear la pantalla para login	PENDIENTE
H3 Crear pantalla para modificar contraseña usuario	PENDIENTE
H4 Crear la estructura para gestión de clientes	PENDIENTE
H5 Crear las pantallas para la gestión de clientes	PENDIENTE
H6 Crear la estructura para gestión de contratos	PENDIENTE
H7 Crear las pantallas para la gestión de contratos	PENDIENTE
H8 Crear la estructura para gestión de productos	PENDIENTE

H9 Crear las pantallas para la gestión de productos	PENDIENTE
H10 Crear la estructura para la gestión de compras	PENDIENTE
H11 Crear las pantallas para la gestión de compras	PENDIENTE
H12 Crear la estructura para la gestión de reservas	PENDIENTE
H13 Crear las pantallas para la gestión de reservas	PENDIENTE
H14 Crear la estructura para la gestión de ventas	PENDIENTE
H15 Crear las pantallas para la gestión de ventas	PENDIENTE

4.2 Sprint 1

Objetivos:

- Crear la estructura básica del sistema de seguridad y la funcionalidad inicial para la gestión de usuarios.
- Implementar la estructura y las funcionalidades básicas para la gestión de clientes.
- Crear pantalla para que los usuarios puedan modificar su contraseña y pantallas para la gestión de clientes.

Actividades Realizadas:

- Creación de modelos para usuarios y clientes:
    - Diseño y desarrollo de los modelos de datos para usuarios y clientes.
    - Definición de las relaciones entre entidades en la base de datos utilizando Sequelize.
- Creación de controladores para usuarios y clientes:
    - Desarrollo de los controladores para manejar las operaciones CRUD (Crear, Leer, Actualizar, Eliminar) para usuarios y clientes.
- Desarrollo de la lógica de autenticación (login/logout) utilizando Node.js y Express.js:
    - Implementación de la autenticación segura para el login y logout de usuarios.
- Creación de control de rutas para usuarios y clientes:
    - Definición y configuración de las rutas necesarias para manejar las solicitudes HTTP relacionadas con usuarios y clientes.
- Diseño y desarrollo de pantallas para la gestión de usuarios y clientes:
    - Diseño de las interfaces de usuario para la creación, visualización y modificación de usuarios y clientes utilizando React.js y Bootstrap.
- Integración de las pantallas con las API endpoints correspondientes.

Revisión del Sprint:

- Demostración: Presentación de la funcionalidad de login, modificación de contraseñas a usuarios y funcionalidad de gestión de clientes.
- Feedback: Recopilación de sugerencias y comentarios para mejorar la interfaz de usuario y la experiencia de autenticación.

Retrospectiva del Sprint:

- Evaluación: Se destaca la modalidad del código para reutilizar en los siguientes Sprints. Funciono bien fue la integración fluida entre el frontend y backend. Las áreas de mejora incluyen la necesidad de optimizar la validación de entradas de usuario.

Historial	**Estado**
H1 Crear la estructura de usuario y login del sistema de seguridad	ENTREGADO
H2 Crear la pantalla para login	ENTREGADO
H3 Crear pantalla para modificar contraseña usuario	ENTREGADO
H4 Crear la estructura para gestión de clientes	ENTREGADO
H5 Crear las pantallas para la gestión de clientes	ENTREGADO

4.2.1 H1 Crear la Estructura de usuario y login del sistema de seguridad

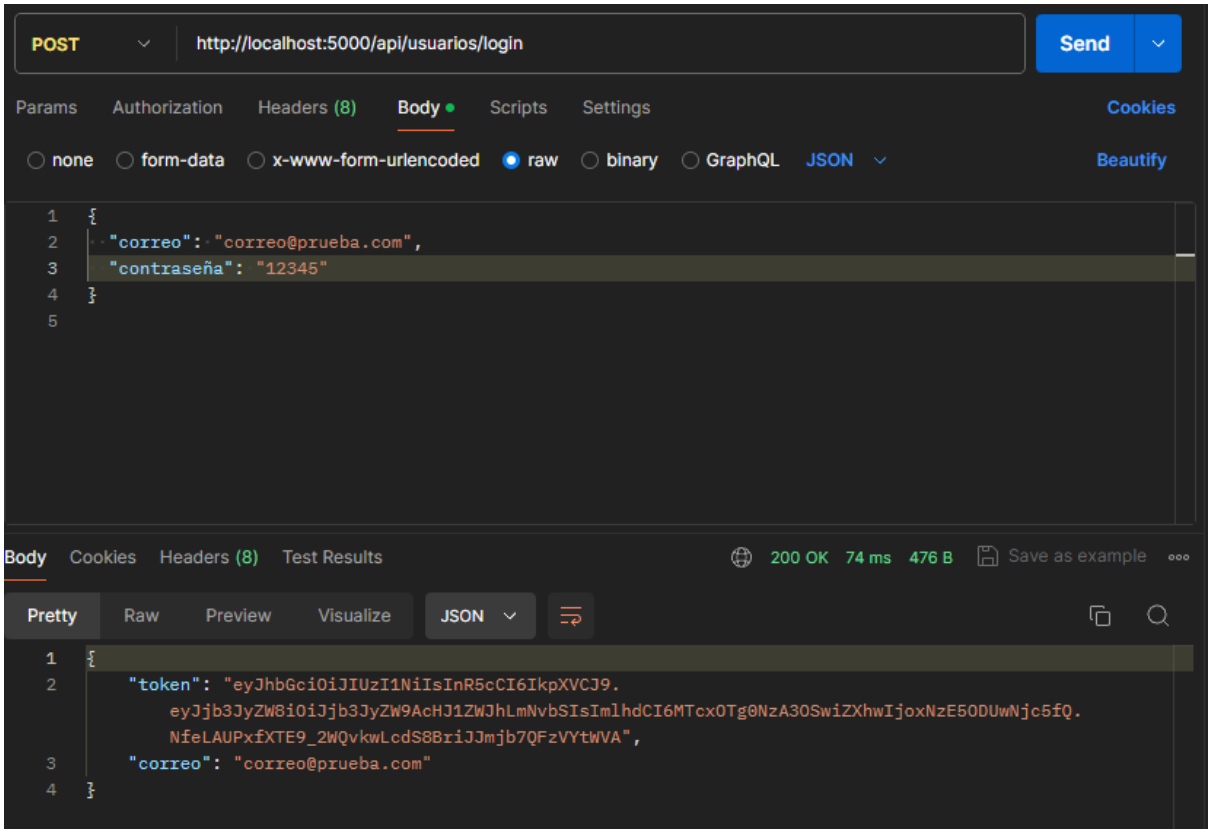

Figure 5 Estructura Login del Sistema de Seguridad – Postman

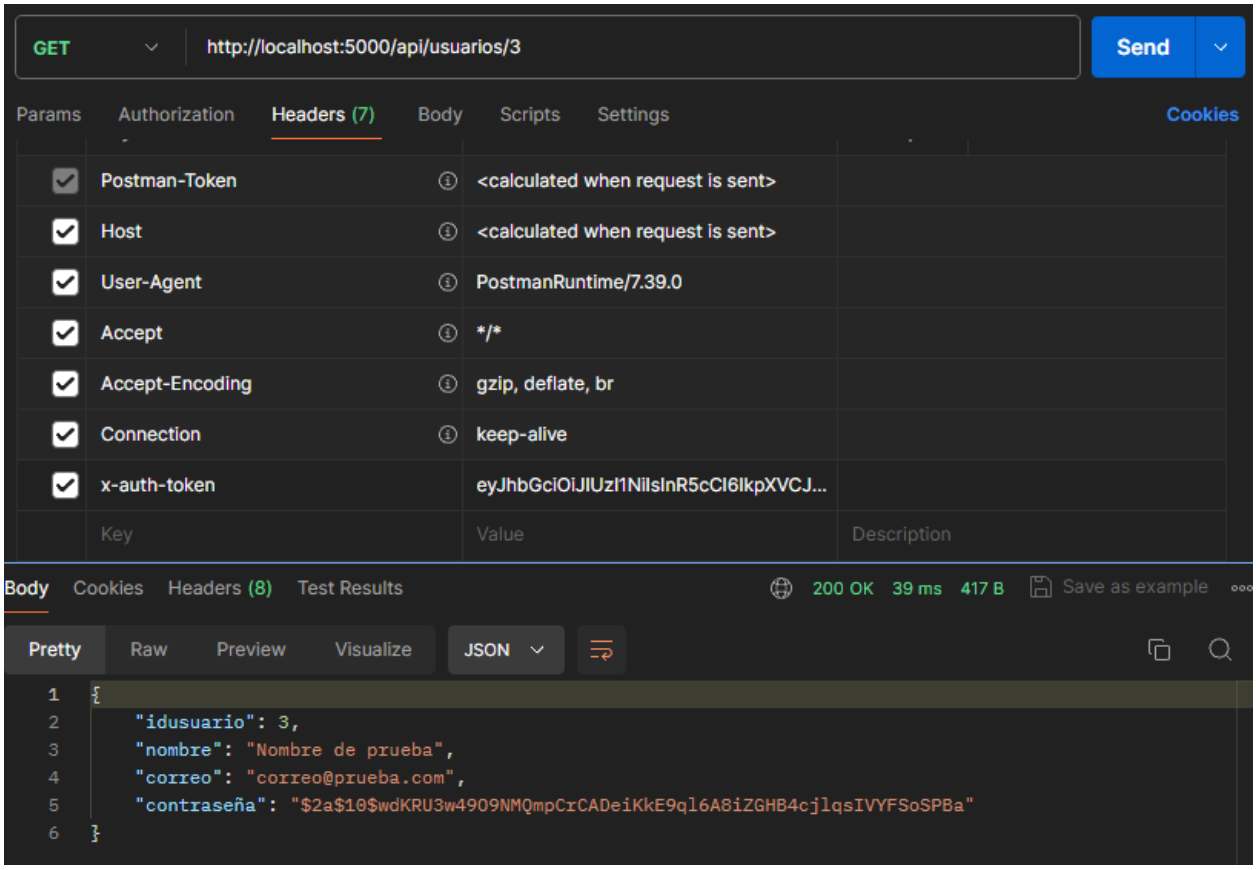

Figure 6 Estructura de Usuario del Sistema - Postman

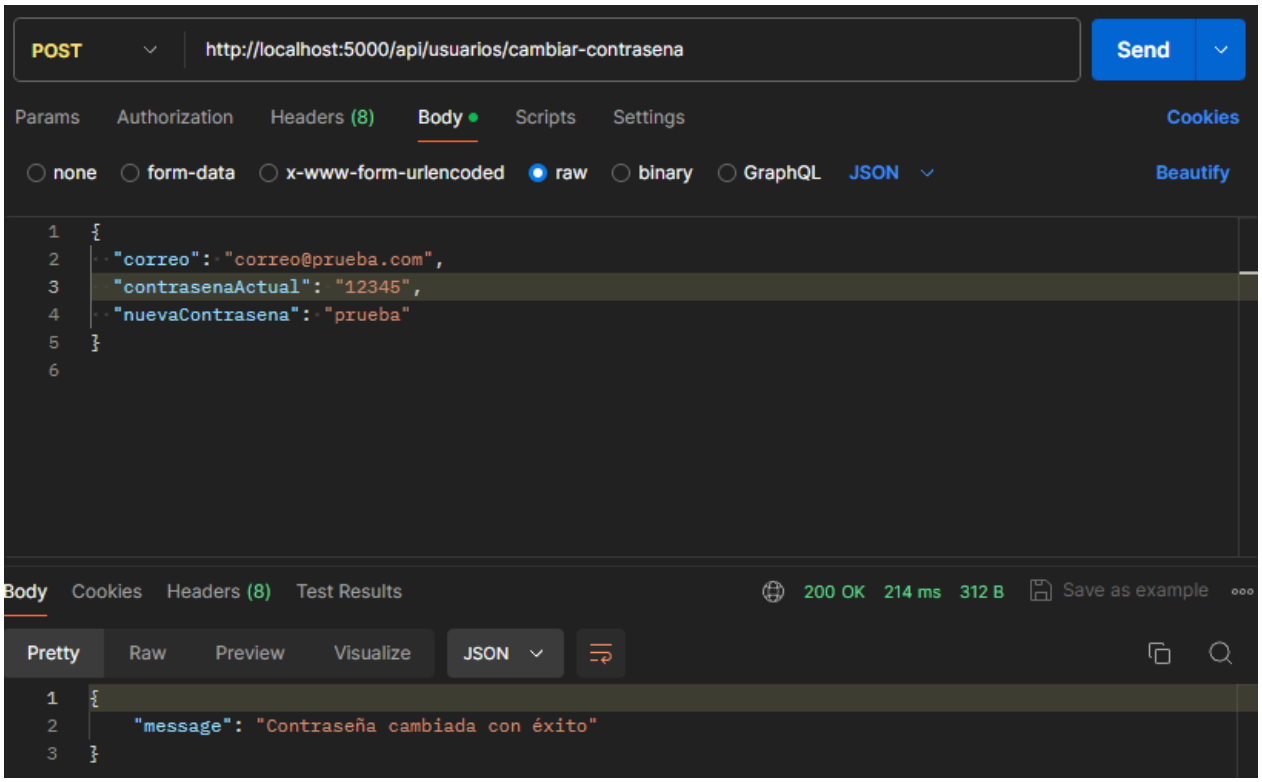

Figure 7 Estructura Cambiar Contraseña Usuario – Postman

4.2.2 H2 Crear la Pantalla para Login

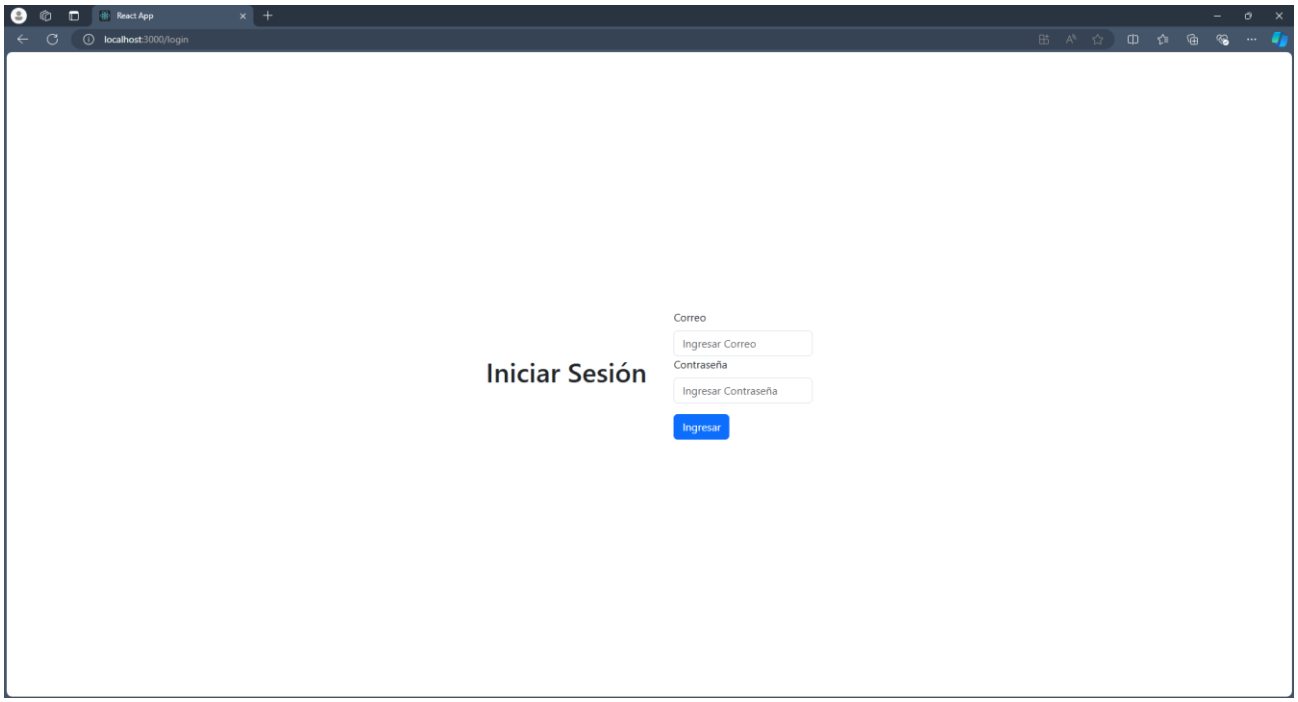

Figure 8 Pantalla Login

4.2.3 H3 Crear pantalla para modificar contraseña usuario

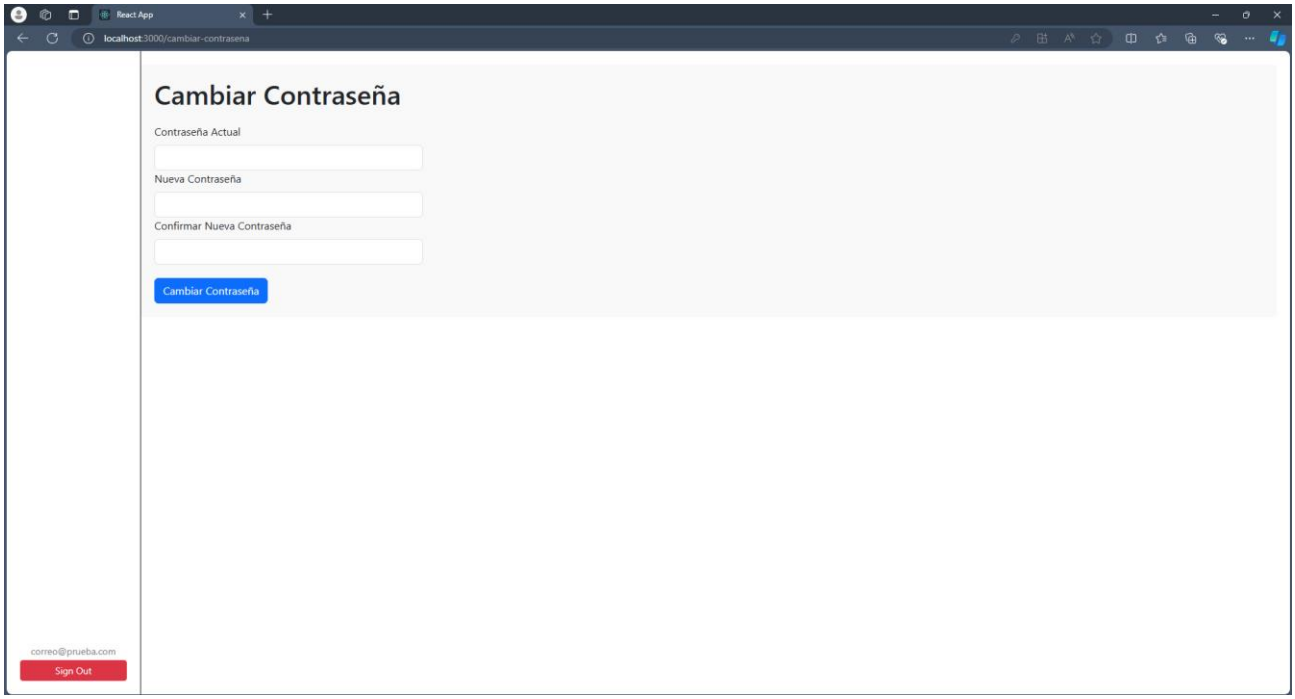

Figure 9 Pantalla Modificar contraseña Usuario

4.2.4 H4 Crear la estructura para gestión de clientes

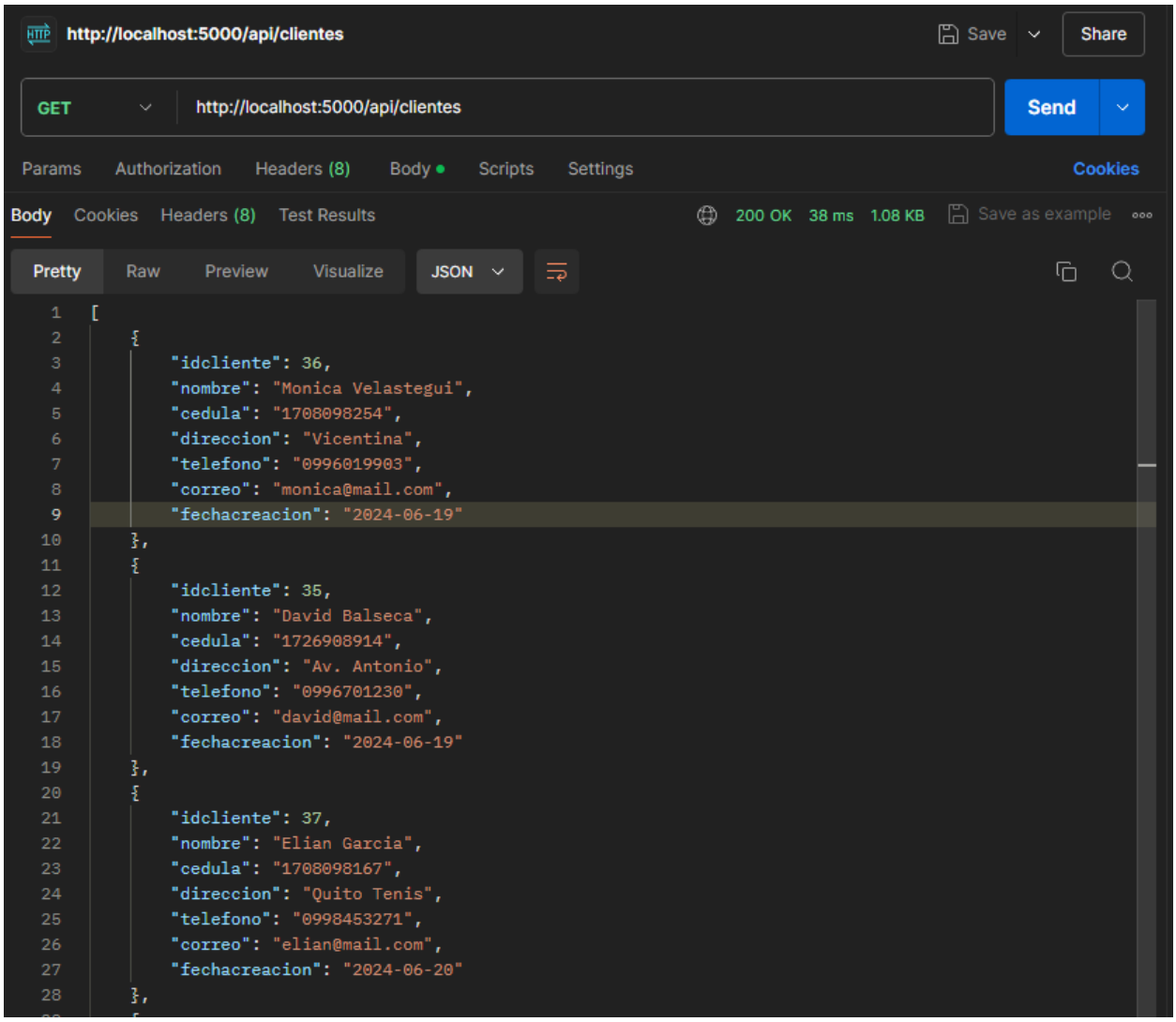

Figure 10 Estructura Clientes Método Get - Postman

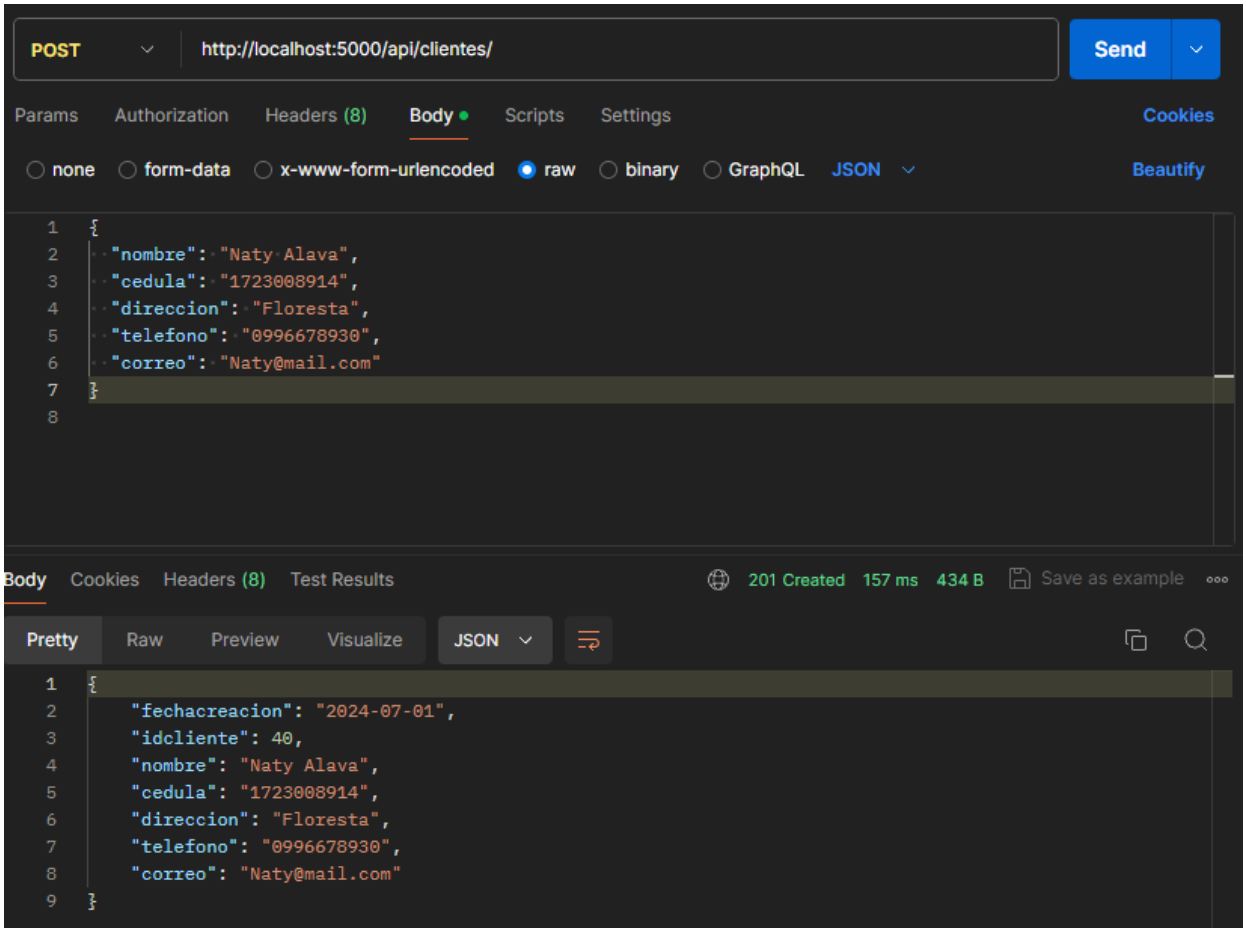

Figure 11 Estructura Clientes Método Post - Postman

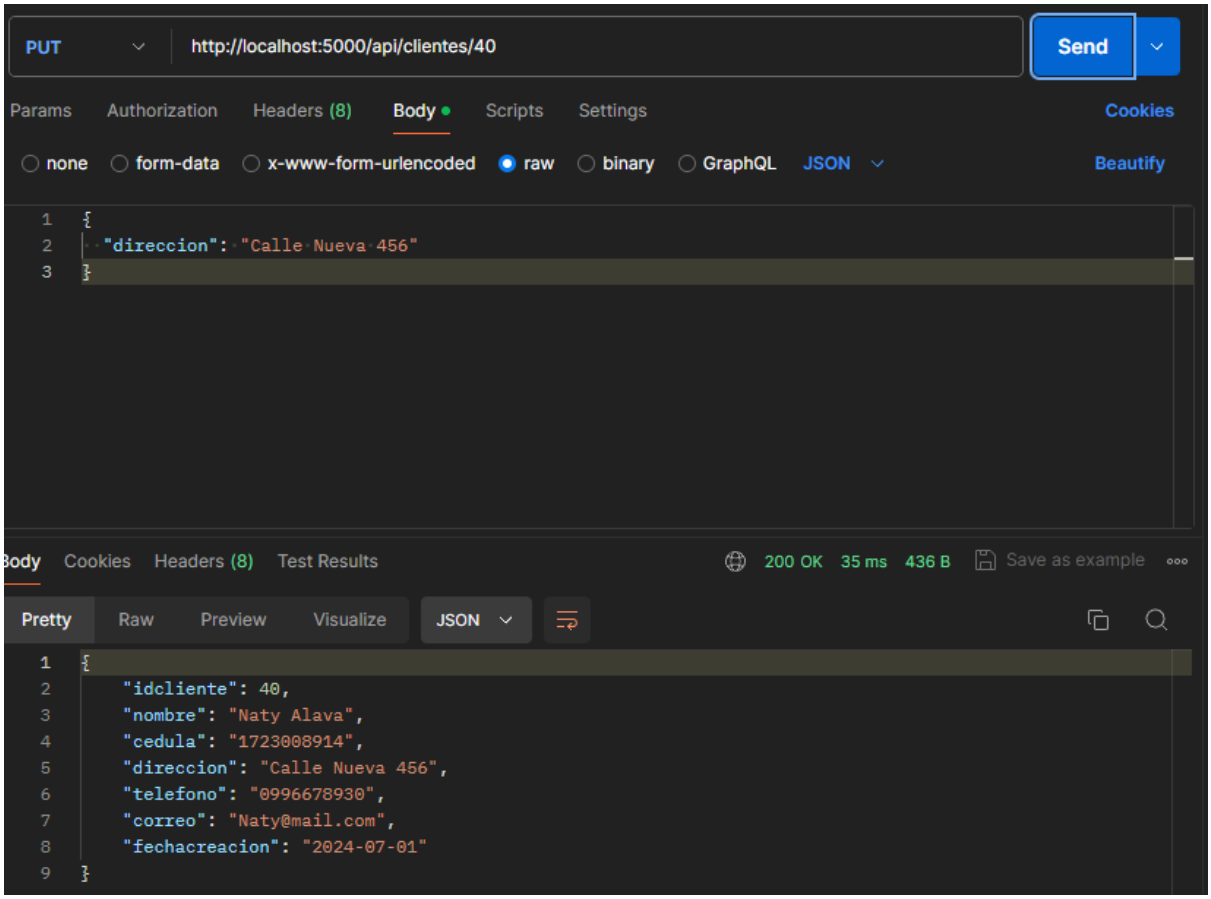

Figure 12 Estructura Cliente Método Put - Postman

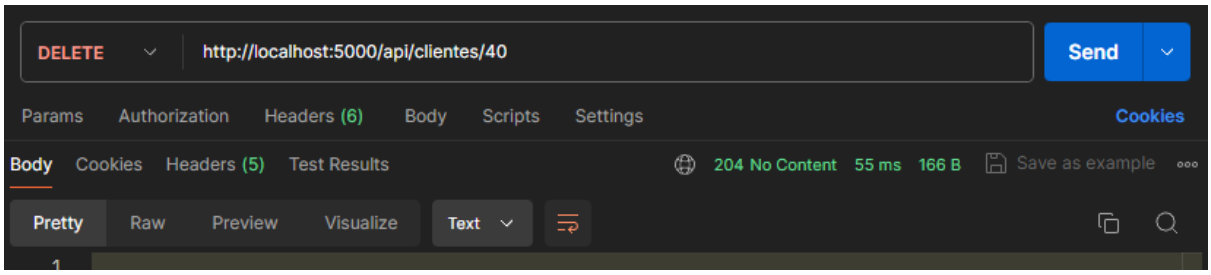

Figure 13 Estructura Cliente Método Delete - Postman

4.2.5 H5 Crear las pantallas para la gestión de clientes

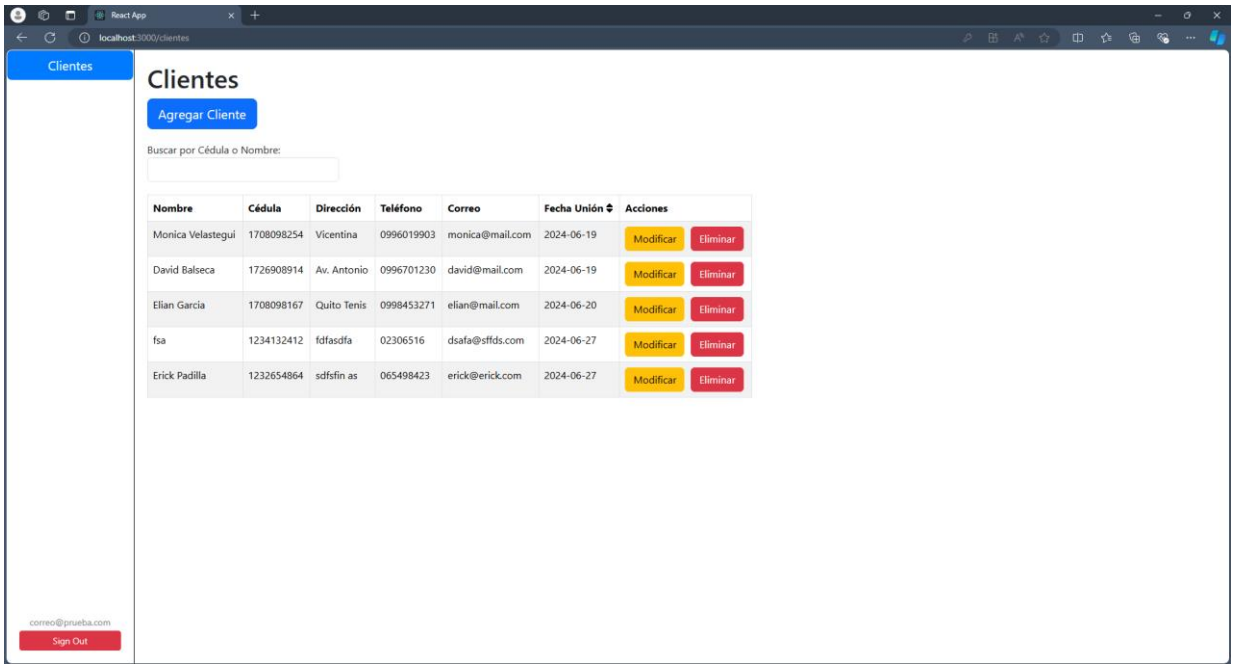

Figure 14 Pantalla Gestion Clientes

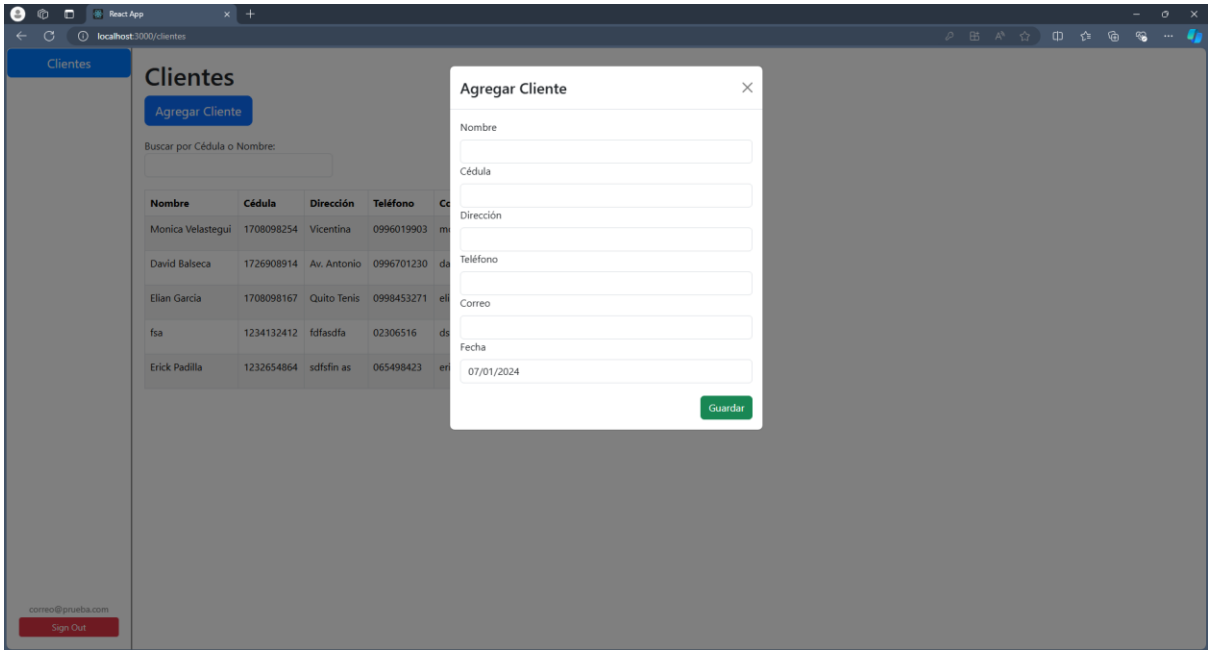

Figure 15 Pantalla Agregar Cliente

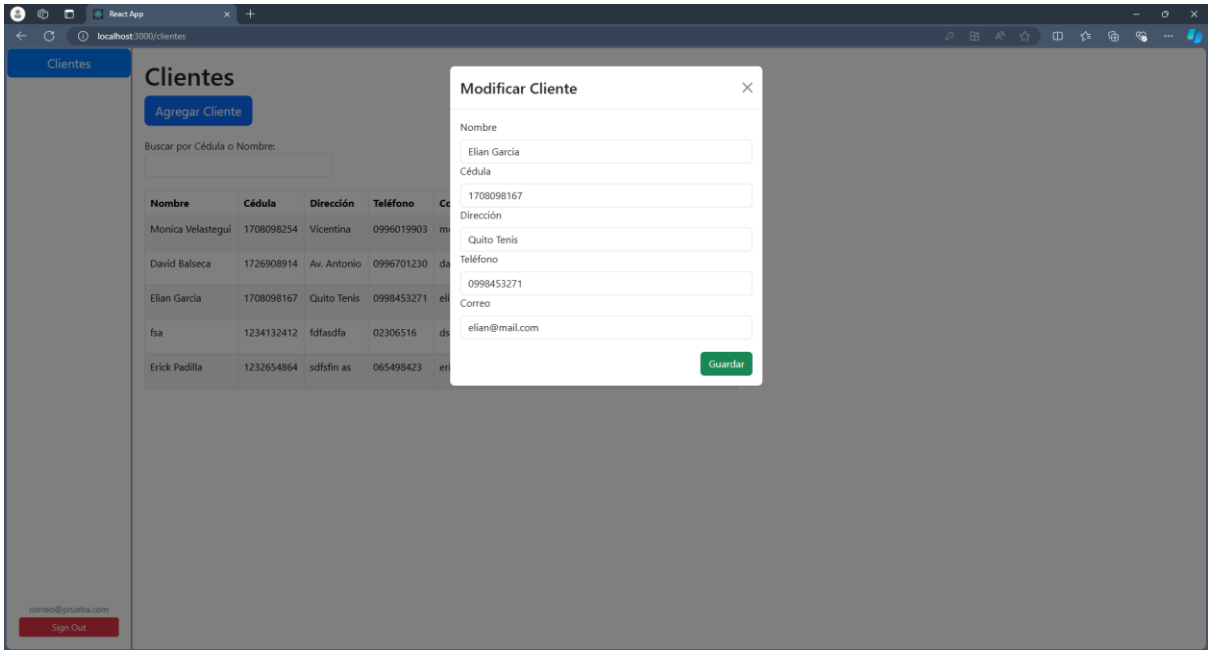

Figure 16 Pantalla Modificar Cliente

4.3 Sprint 2

Objetivos:

- Crear la estructura y las funcionalidades básicas para la gestión de contratos, productos y compras.
- Crear pantallas para la gestión de contratos y la gestión de productos.
- Desarrollar la lógica de negocio para la gestión de contratos de consignación y productos.

Actividades Realizadas:

- Creación de modelos para reservas, ventas y compras:
  - Diseño y desarrollo de los modelos de datos para reservas, ventas y compras.
  - Definición de las relaciones entre entidades en la base de datos utilizando Sequelize.
- Creación de controladores para reservas, ventas y compras:
  - Desarrollo de los controladores para manejar las operaciones CRUD (Crear, Leer, Actualizar, Eliminar) para reservas, ventas y compras.
- Desarrollo de la lógica de negocio para reservas:
  - Implementación de las reglas de negocio para la creación y gestión de reservas.
  - Gestión de estados y fechas de reservas.
- Desarrollo de la lógica de negocio para ventas:

    - Implementación de las reglas de negocio para la gestión de ventas.
    - Integración con el sistema de gestión de productos para reflejar las ventas en tiempo real.
- Desarrollo de la lógica de negocio para compras:
    - Implementación de las reglas de negocio para la gestión de compras.
    - Integración con el sistema de gestión de productos para reflejar las compras en tiempo real.
- Creación de control de rutas para reservas y ventas:
    - Definición y configuración de las rutas necesarias para manejar las solicitudes HTTP relacionadas con reservas y ventas.
- Diseño y desarrollo de pantallas para la gestión de reservas y ventas:
    - Diseño de las interfaces de usuario para la creación, visualización y modificación de reservas y ventas utilizando React.js y Bootstrap.
- Integración de las pantallas con las API endpoints correspondientes.

Revisión del Sprint:

- Demostración: Presentación de las funcionalidades de gestión de contratos, productos y compras.
- Feedback: Recopilación de sugerencias para mejorar la visualización de datos en las tablas y la experiencia de usuario en la gestión de contratos.

Retrospectiva del Sprint:

- Evaluación: Se destaca la capacidad de adaptar el código ya creado a las nuevas funcionalidades que se van a desarrollan. Las áreas de mejora incluyen la optimización de consultas en las pantallas y la mejora en la eficiencia del sistema.

Historial	Estado
H6 Crear la estructura para gestión de contratos	ENTREGADO
H7 Crear las pantallas para la gestión de contratos	ENTREGADO
H8 Crear la estructura para gestión de productos	ENTREGADO
H9 Crear las pantallas para la gestión de productos	ENTREGADO
H10 Crear la estructura para la gestión de compras	ENTREGADO

4.3.1 H6 Crear la estructura para gestión de contratos

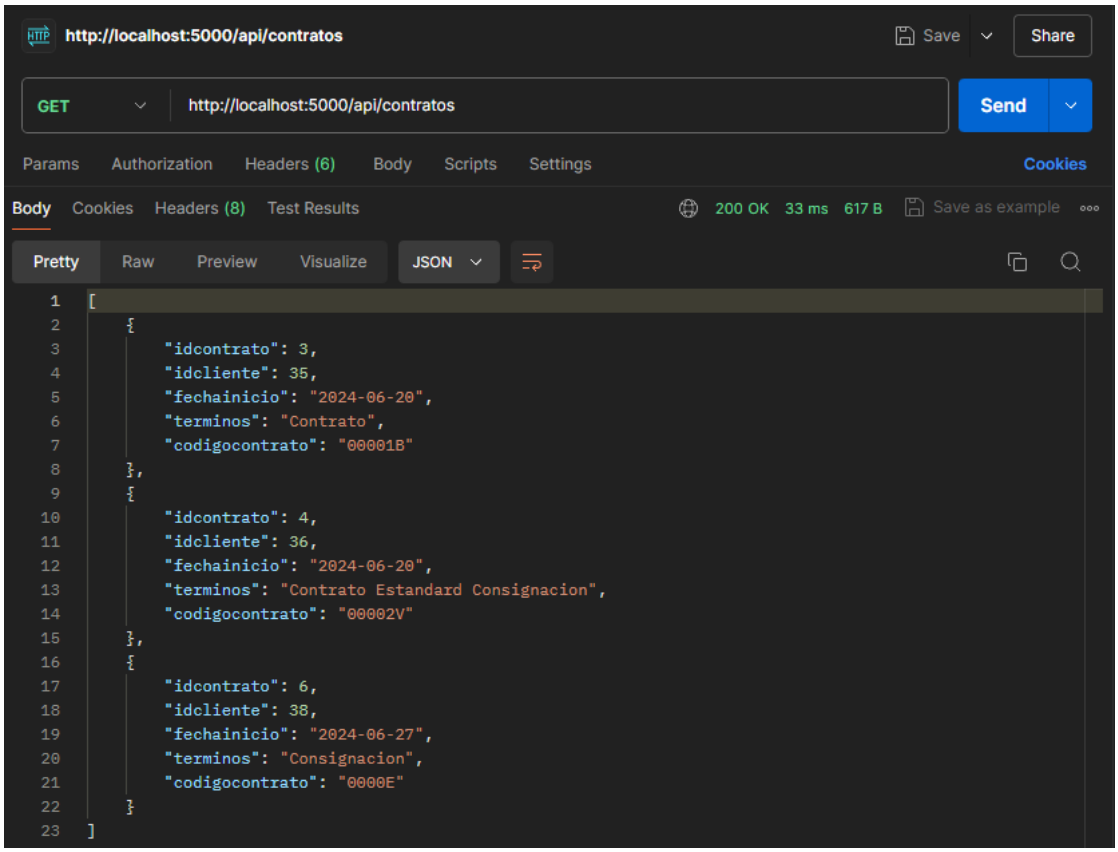

Figure 17 Estructura Contrato Método Get - Postman

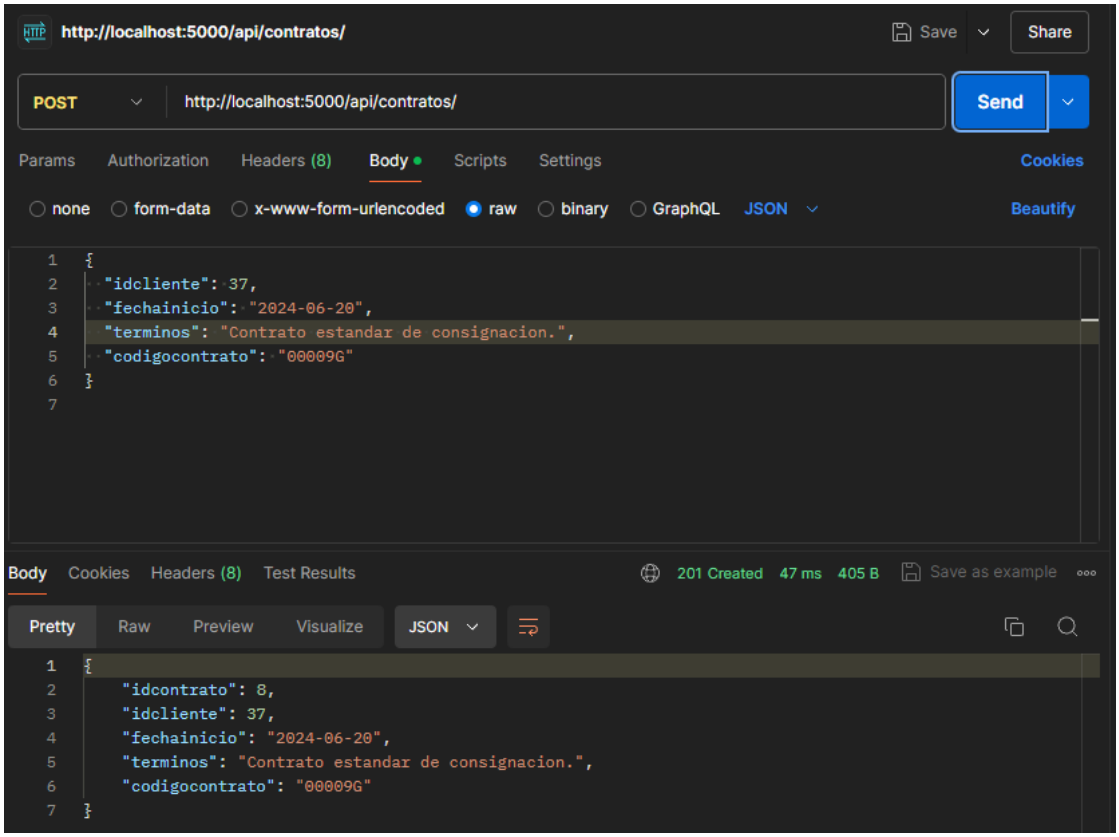

Figure 18 Estructura Contratos Método Post - Postman

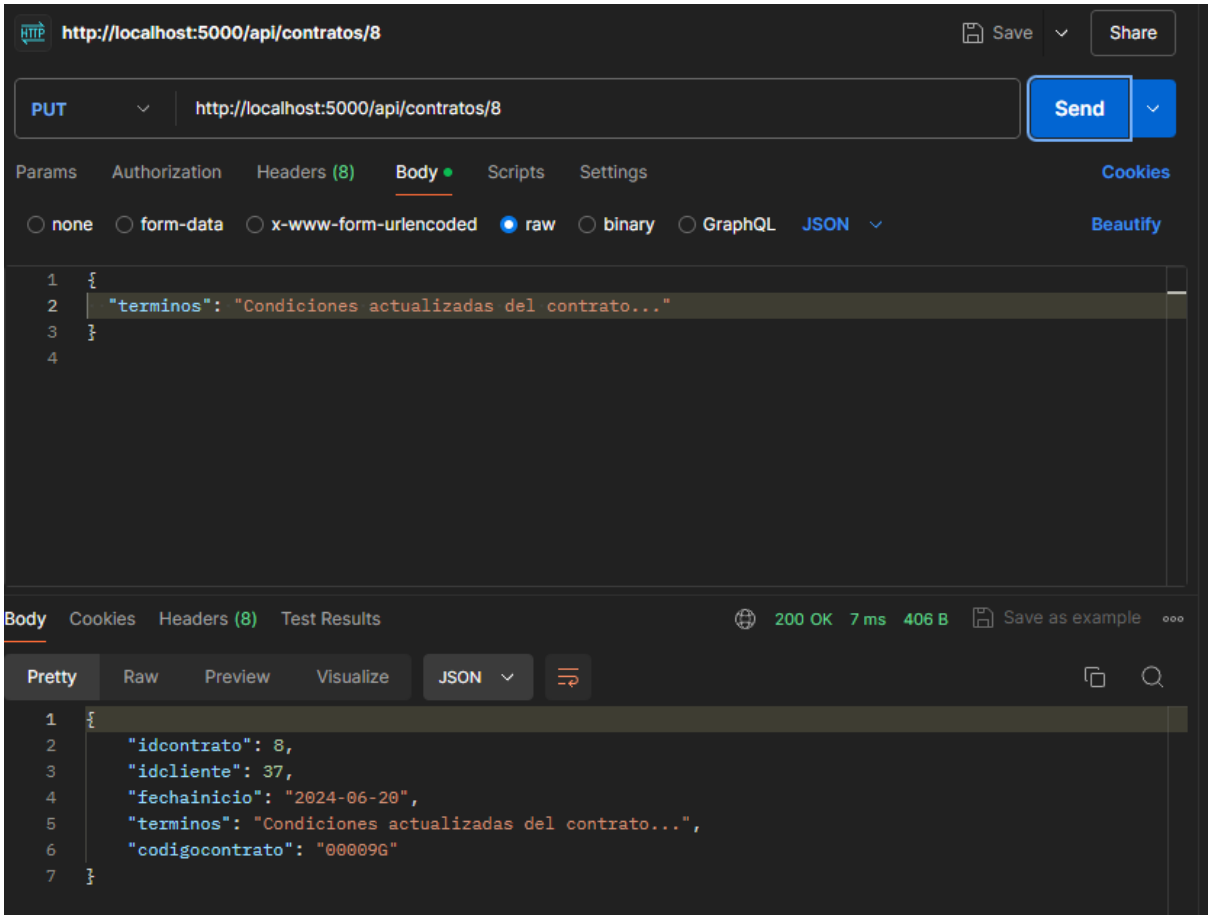

Figure 19 Estructura Contratos Método Put - Postman

4.3.2 H7 Crear las pantallas para la gestión de contratos

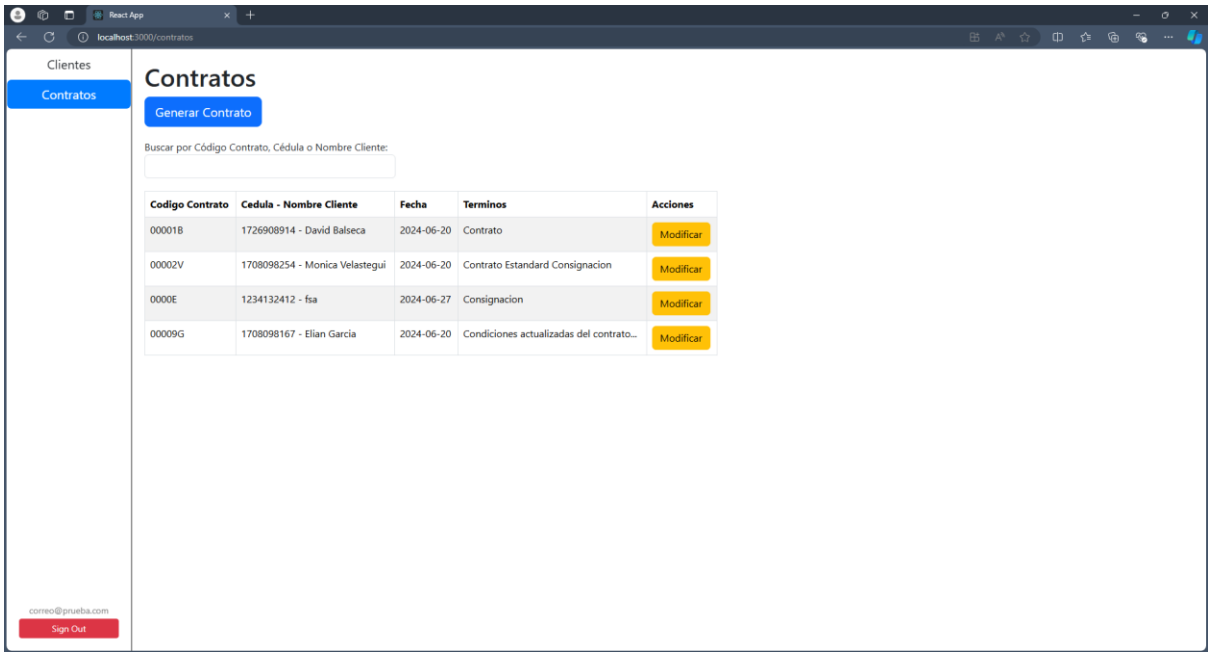

Figure 20 Pantalla Gestión Contratos

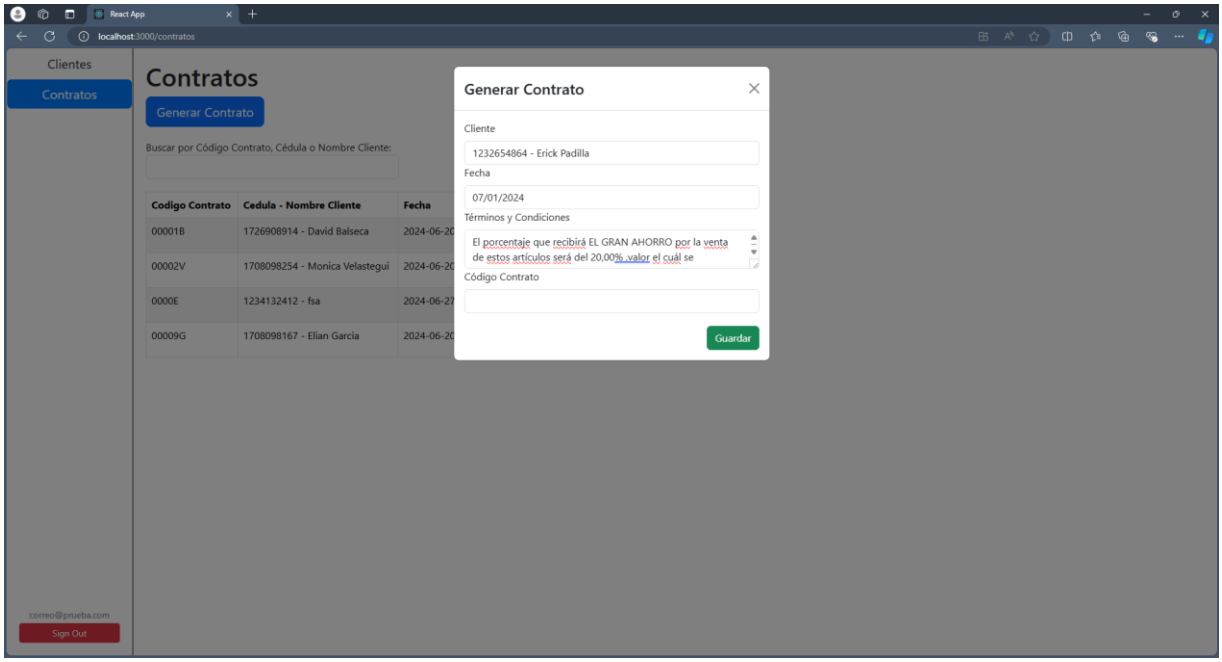

Figure 21 Pantalla Generar Contrato

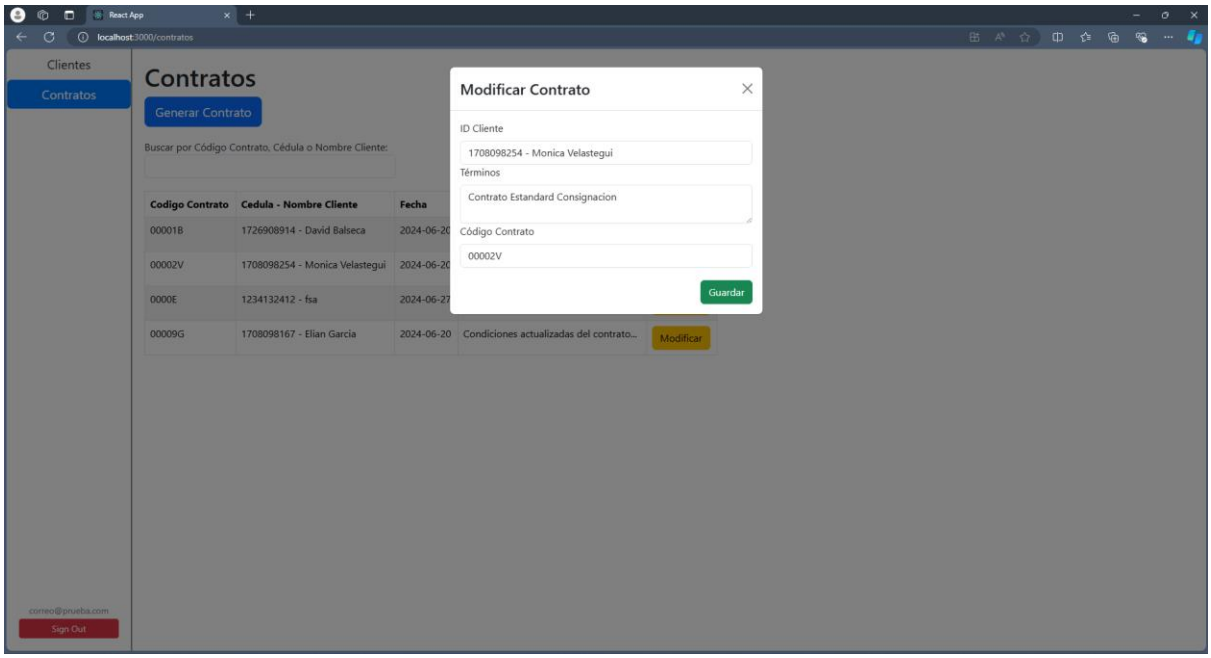

Figure 22 Pantalla Modificar Contrato

4.3.3 H8 Crear la estructura para gestión de productos

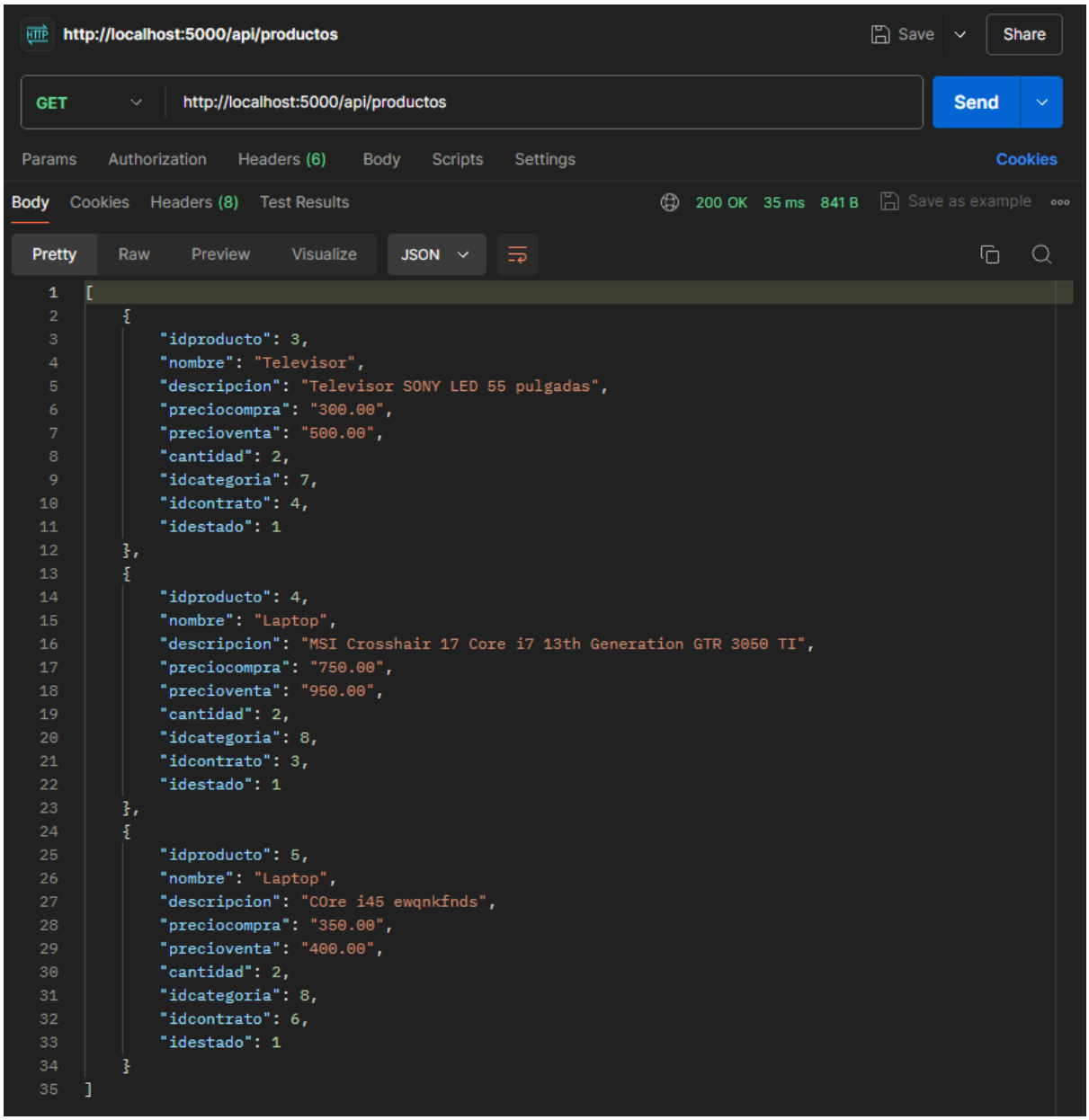

Figure 23 Estructura Productos Metodo Get - Postman

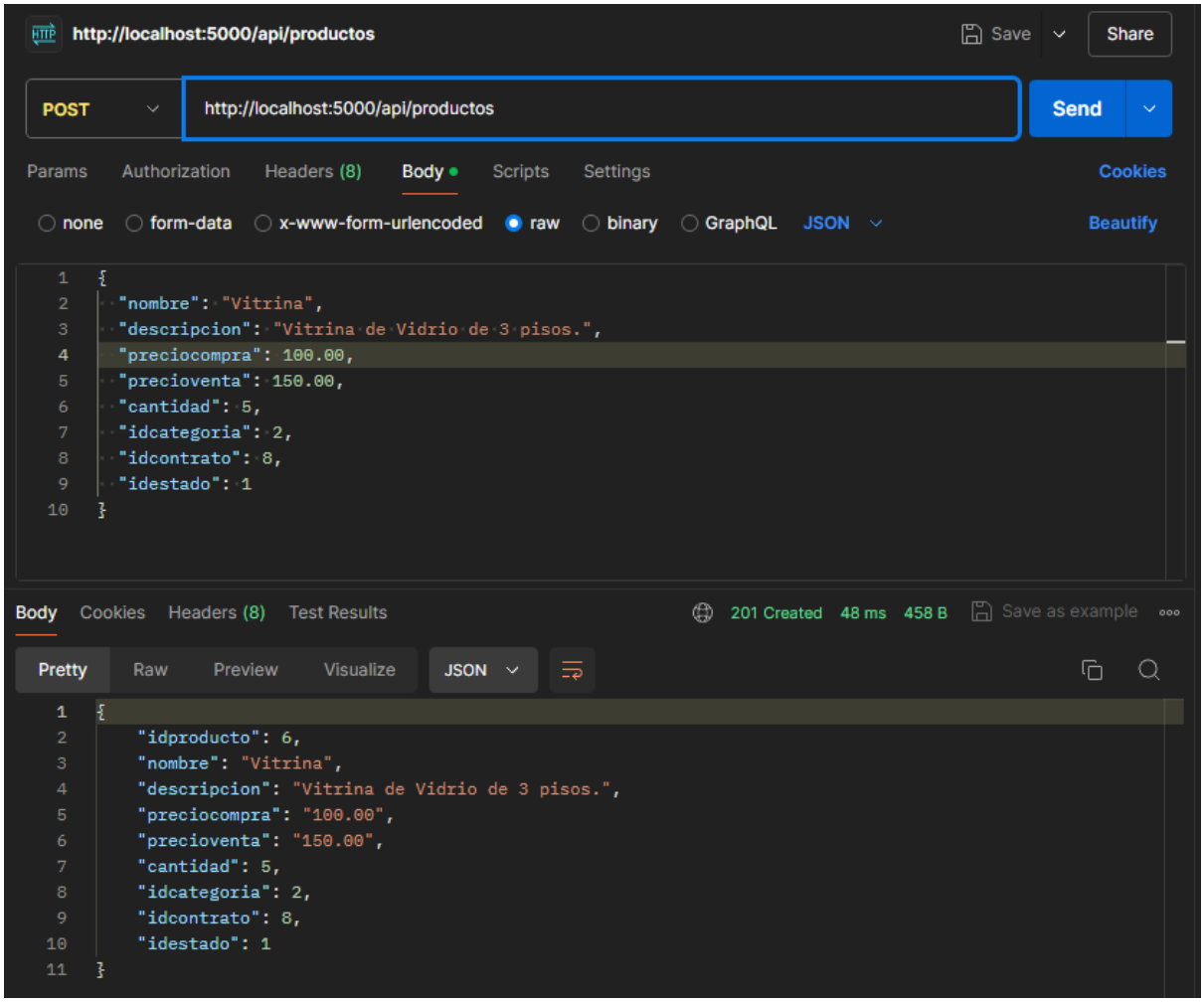

Figure 24 Estructura Productos Método Post - Postman

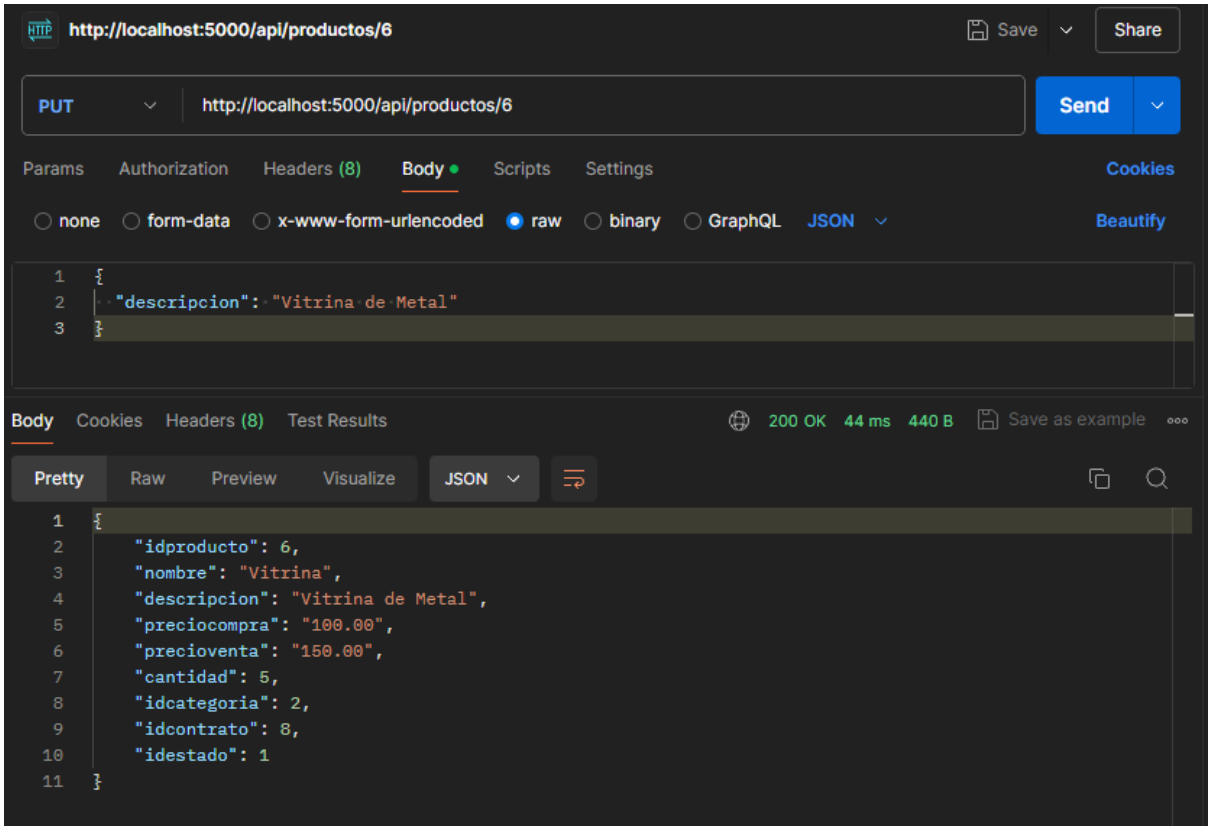

Figure 25 Estructura Productos Método Put - Postman

4.3.4 H9 Crear las pantallas para la gestión de productos

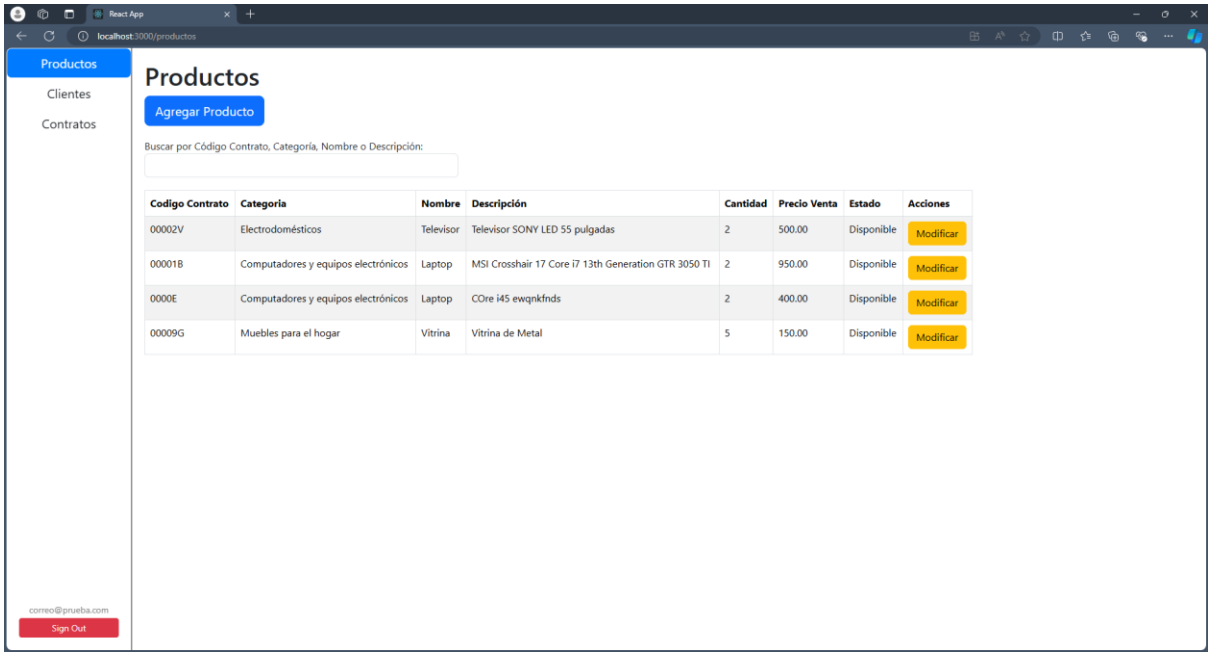

Figure 26 Pantalla Gestión Productos

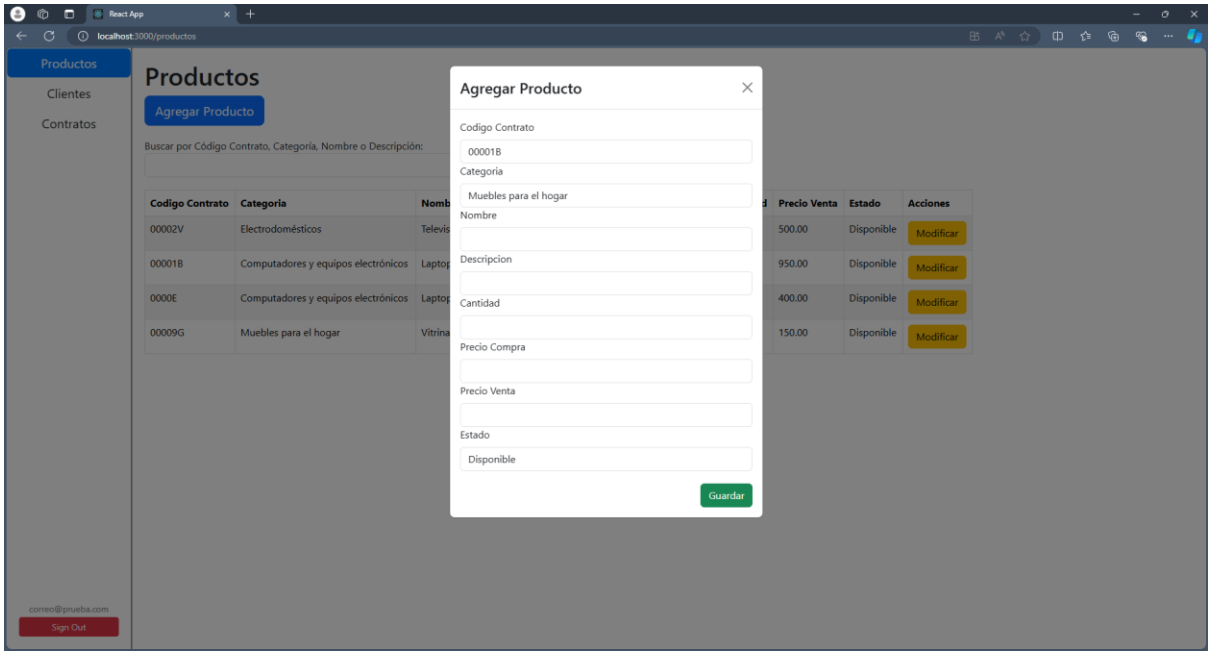

Figure 27 Pantalla Agregar Producto

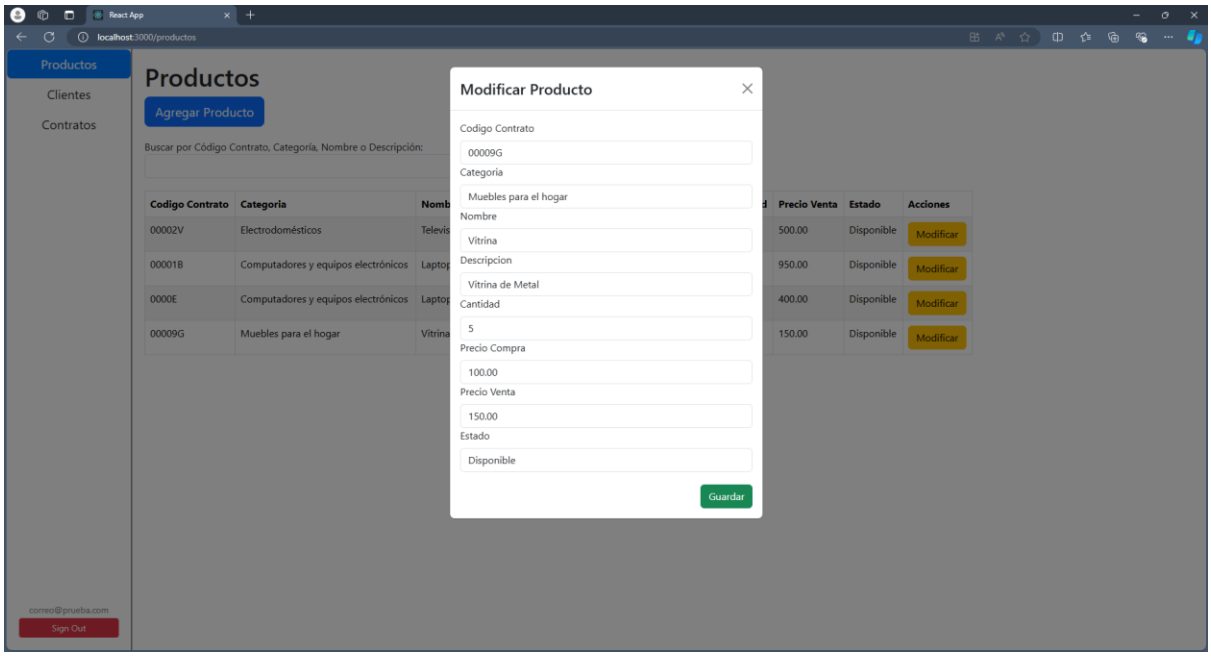

Figure 28 Pantalla Modificar Producto

4.3.5 H10 Crear la estructura para la gestión de compras

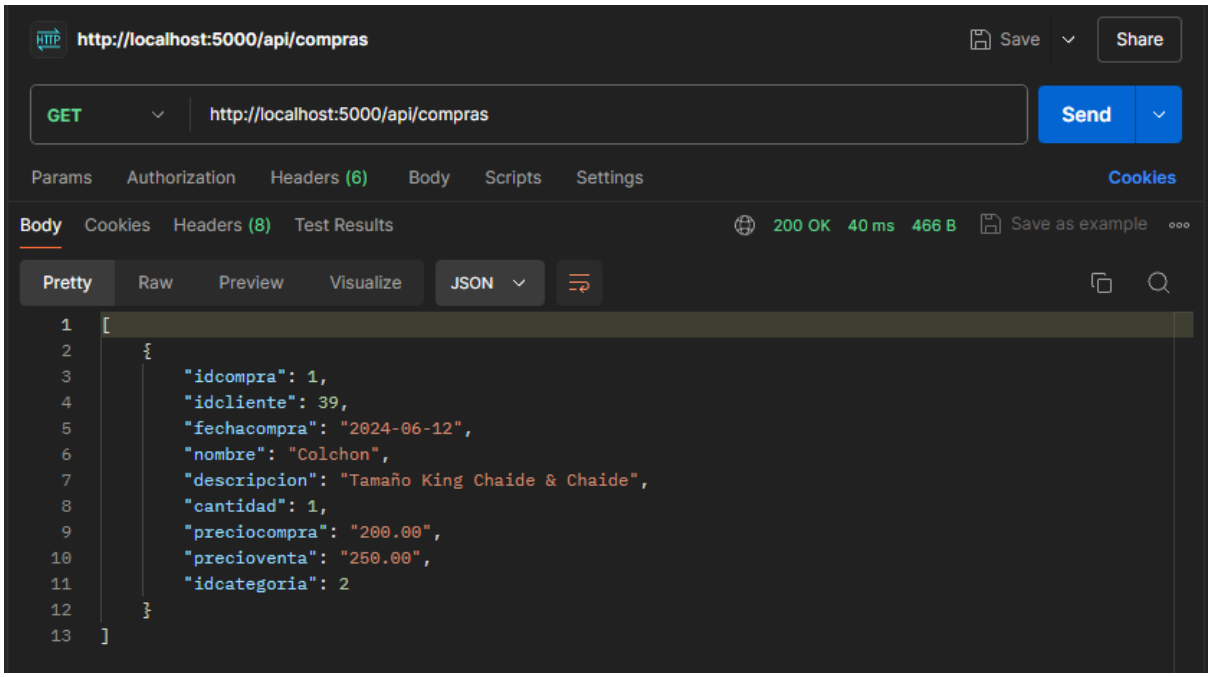

Figure 29 Estructura Compra Método Get - Postman

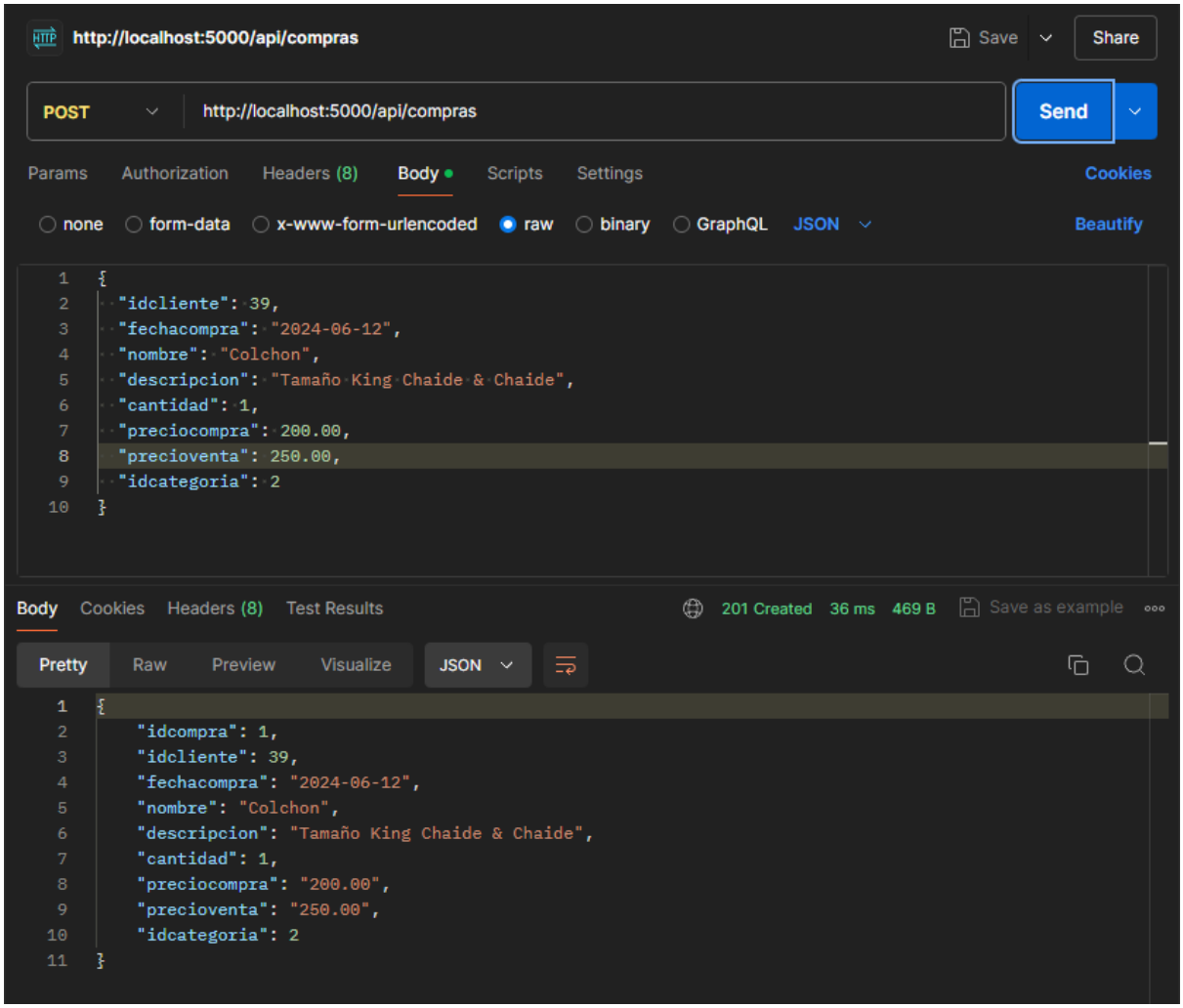

Figure 30 Estructura Compra Método Post - Postman

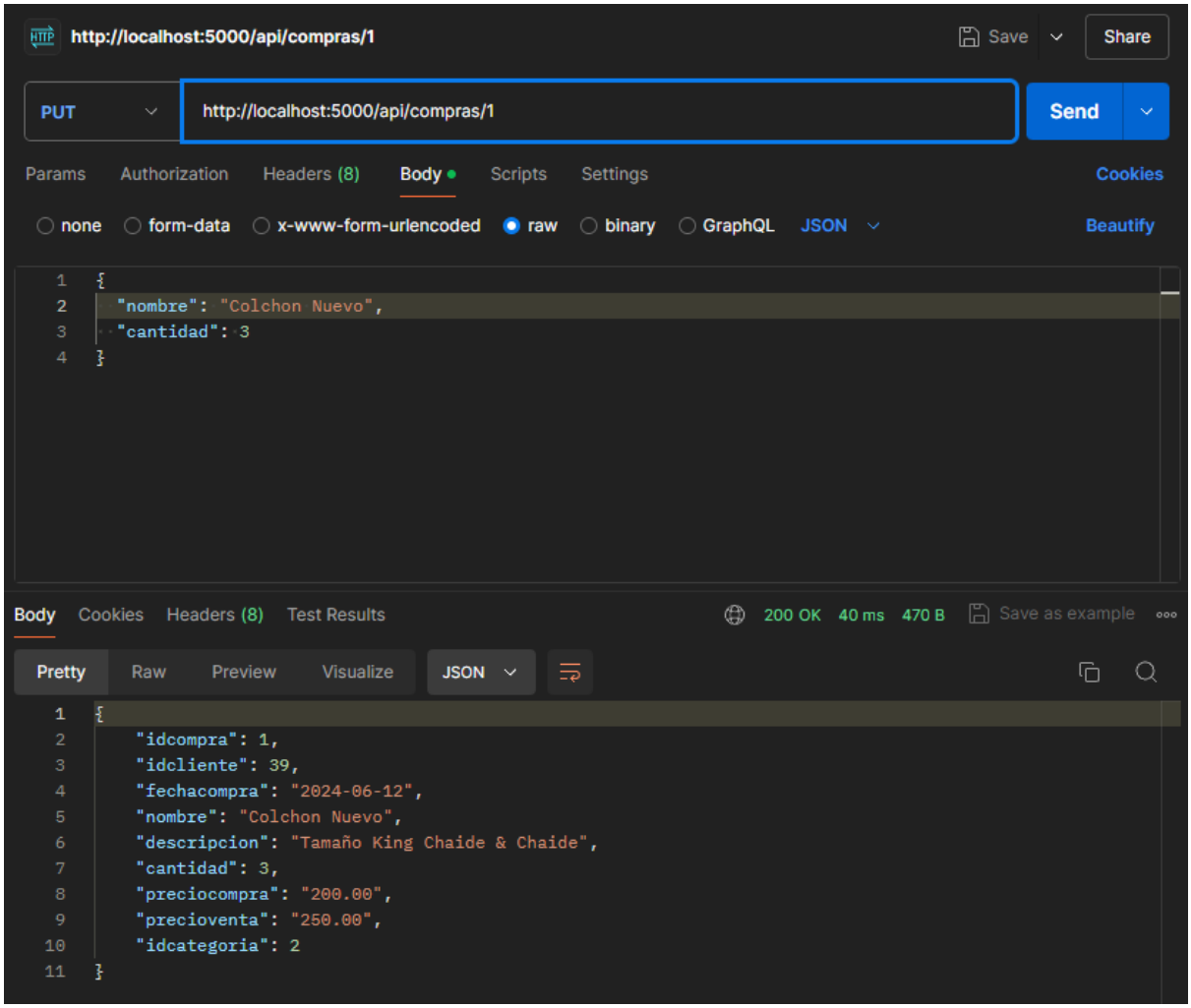

Figure 31 Estructura Compra Método Put - Postman

4.4 Sprint 3

Objetivos:

- Crear la estructura y las funcionalidades básicas para la gestión de reservas y ventas.
- Crear pantallas para la gestión de reservas y la gestión de ventas.
- Desarrollar la lógica de negocio para la gestión de reservas y ventas.

Actividades Realizadas:

- Creación de modelos para reservas y ventas:
    - Diseño y desarrollo de los modelos de datos para reservas y ventas.
    - Definición de las relaciones entre entidades en la base de datos utilizando Sequelize.
- Creación de controladores para reservas y ventas:

    - Desarrollo de los controladores para manejar las operaciones CRUD (Crear, Leer, Actualizar, Eliminar) para reservas y ventas.
- Desarrollo de la lógica de negocio para reservas:
    - Implementación de las reglas de negocio para la creación y gestión de reservas.
    - Gestión de estados y fechas de reservas.
- Desarrollo de la lógica de negocio para ventas:
    - Implementación de las reglas de negocio para la gestión de ventas.
    - Integración con el sistema de gestión de inventarios para reflejar las ventas en tiempo real.
- Creación de control de rutas para reservas y ventas:
    - Definición y configuración de las rutas necesarias para manejar las solicitudes HTTP relacionadas con reservas y ventas.
- Diseño y desarrollo de pantallas para la gestión de compras, reservas y ventas:
    - Diseño de las interfaces de usuario para la creación, visualización y modificación de compras, reservas y ventas utilizando React.js y Bootstrap.
    - Integración de las pantallas con las API endpoints correspondientes.

Revisión del Sprint:

- Demostración: Presentación de las funcionalidades de gestión de reservas y ventas. Se mostraron las pantallas de creación, visualización y modificación de reservas y ventas.
- Feedback: Recomendaciones para mejorar la interfaz de usuario y optimizar el flujo de trabajo en la gestión de reservas y ventas.

Retrospectiva del Sprint:

- Evaluación: Se destacó la integración de múltiples módulos y la gestión eficiente de productos. Las áreas de mejora incluyen la optimización del rendimiento del sistema y la usabilidad de la interfaz.

Historial	Estado
H11 Crear las pantallas para la gestión de compras	ENTREGADO
H12 Crear la estructura para la gestión de reservas	ENTREGADO
H13 Crear las pantallas para la gestión de reservas	ENTREGADO
H14 Crear la estructura para la gestión de ventas	ENTREGADO
H15 Crear las pantallas para la gestión de ventas	ENTREGADO

4.4.1 H11 Crear las pantallas para la gestión de compras

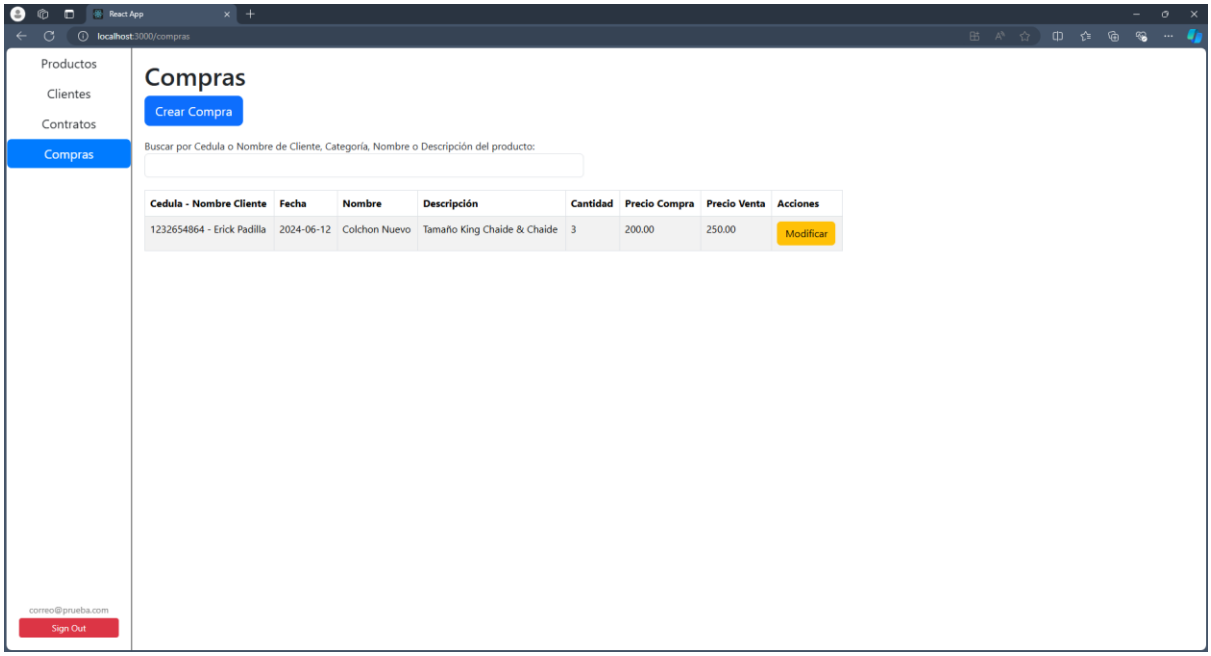

Figure 32 Pantalla Gestión Compras

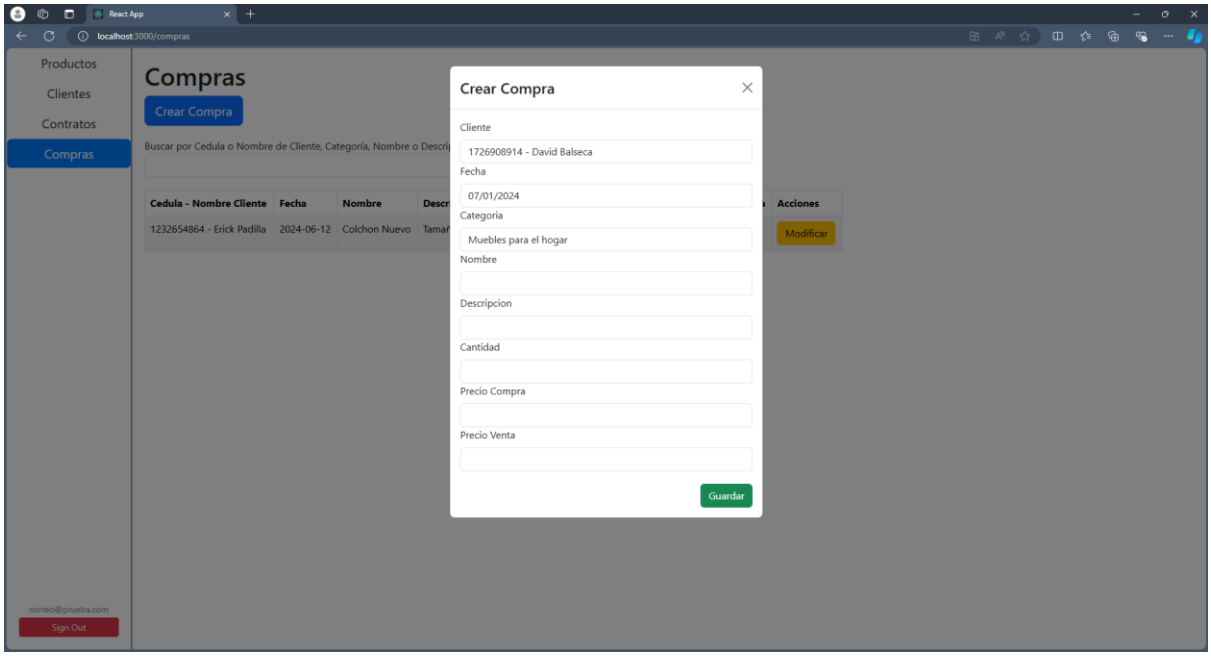

Figure 33 Pantalla Crear Compra

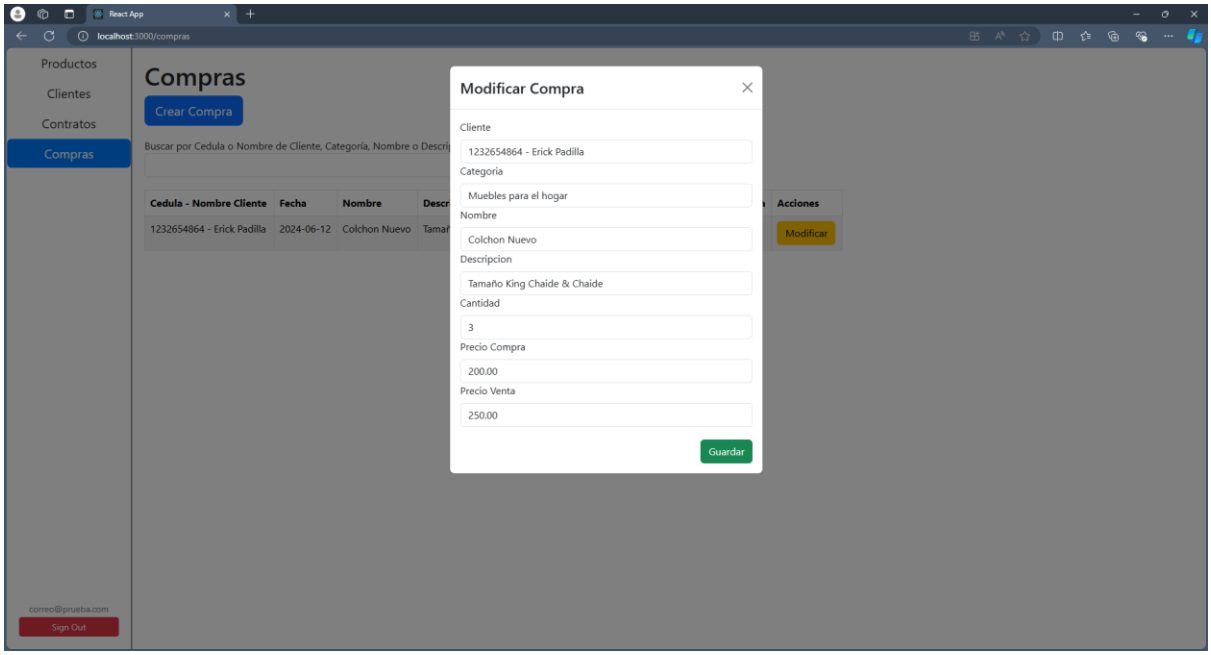

Figure 34 Pantalla Modificar Compra

4.4.2 H12 Crear la estructura para la gestión de reservas

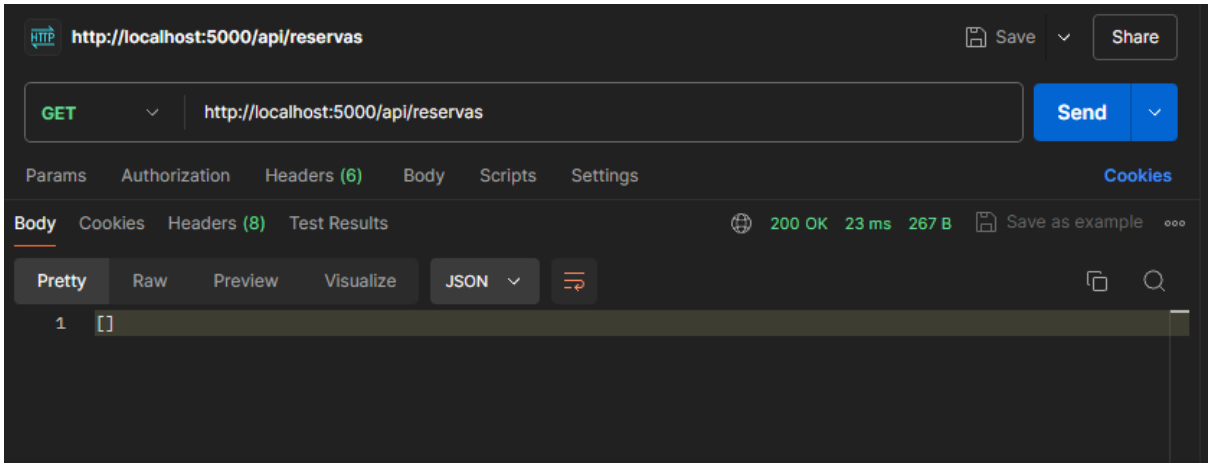

Figure 35 Estructura Reserva Método Get - Postman

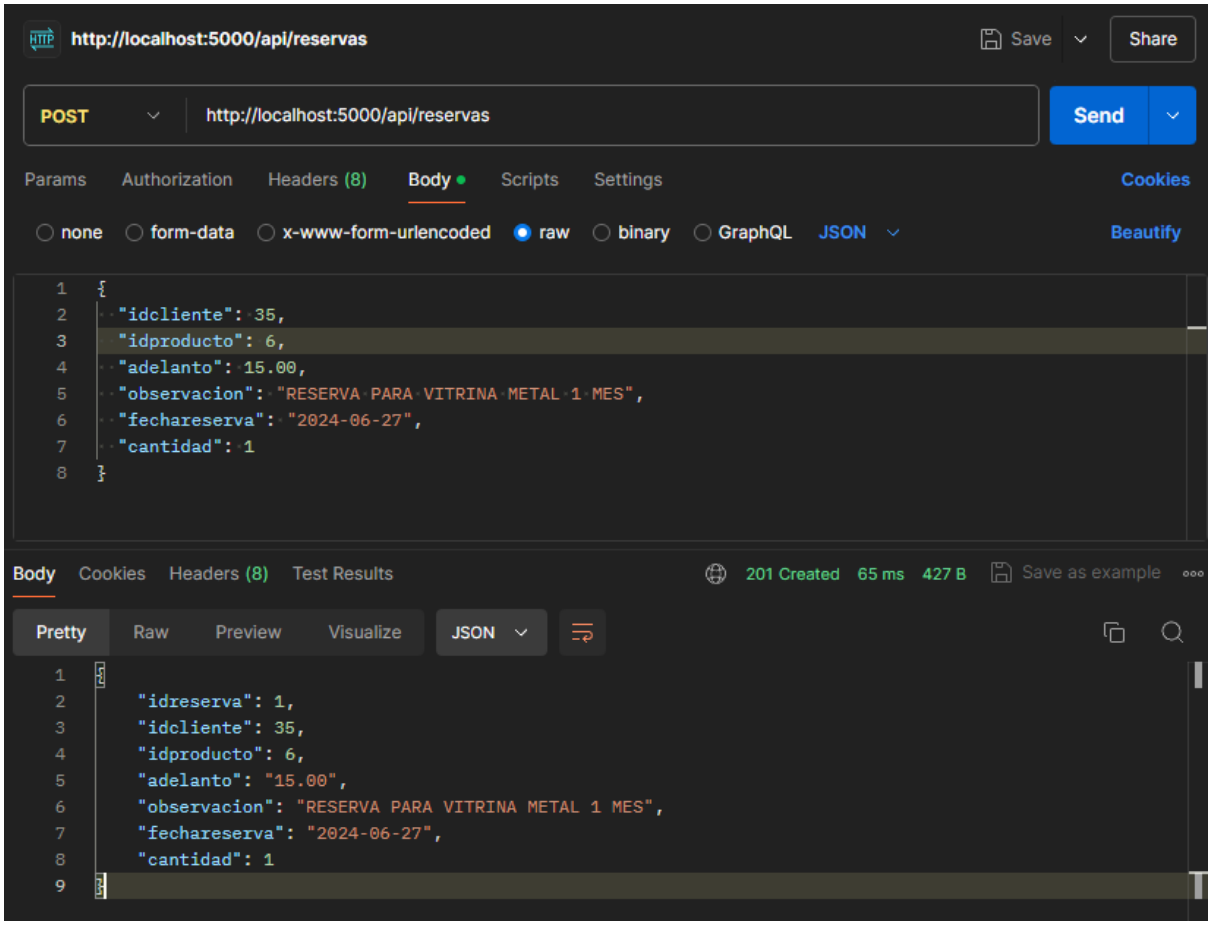

Figure 36 Estructura Reserva Método Post - Postman

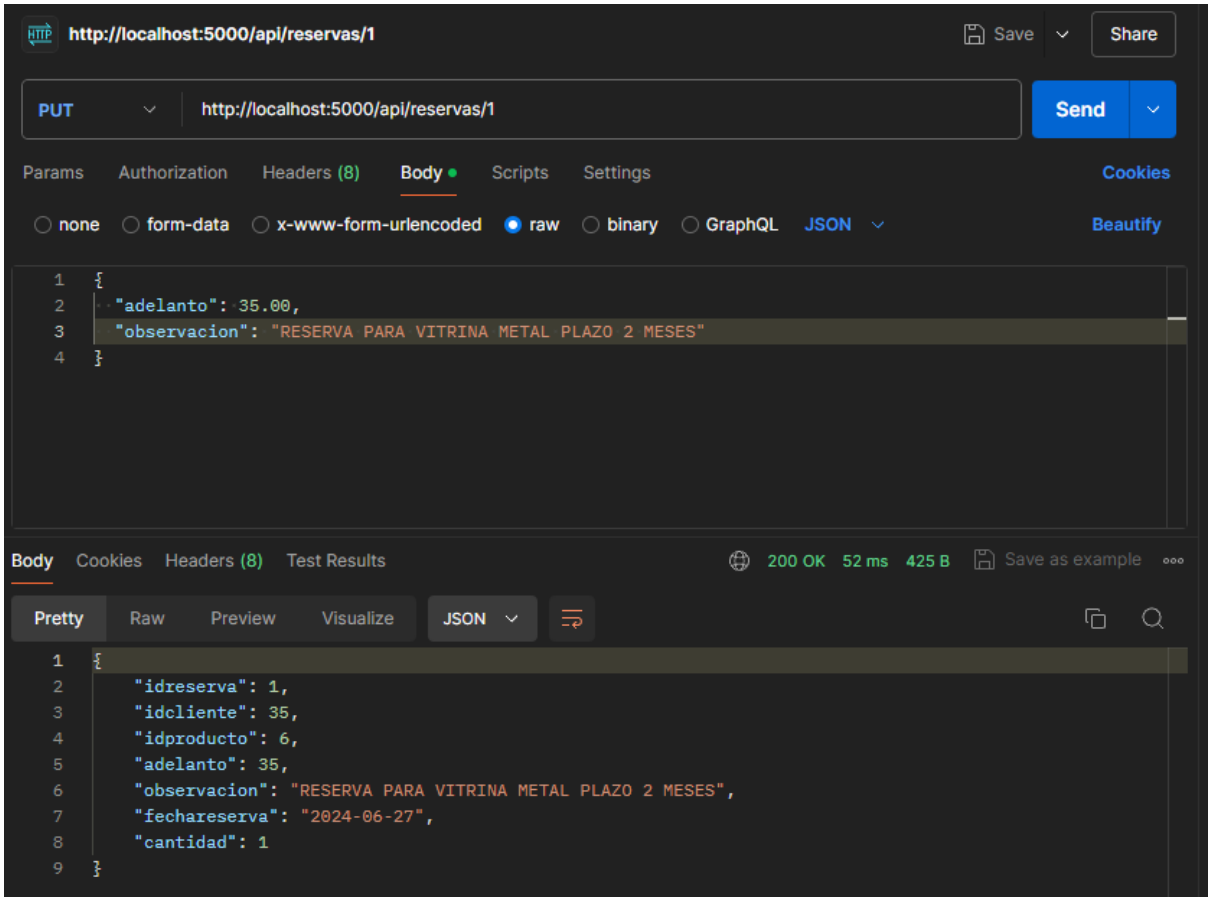

Figure 37 Estructura Reserva Método Put - Postman

4.4.3 H13 Crear las pantallas para la gestión de reservas

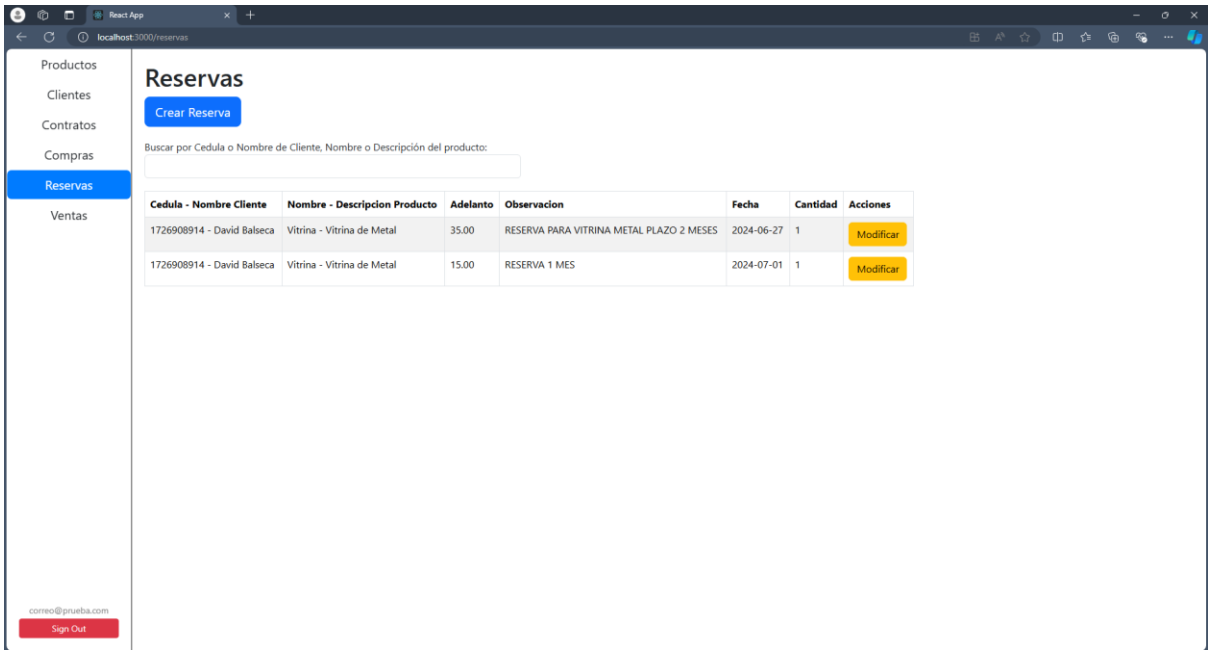

Figure 38 Pantalla Gestión Reservas

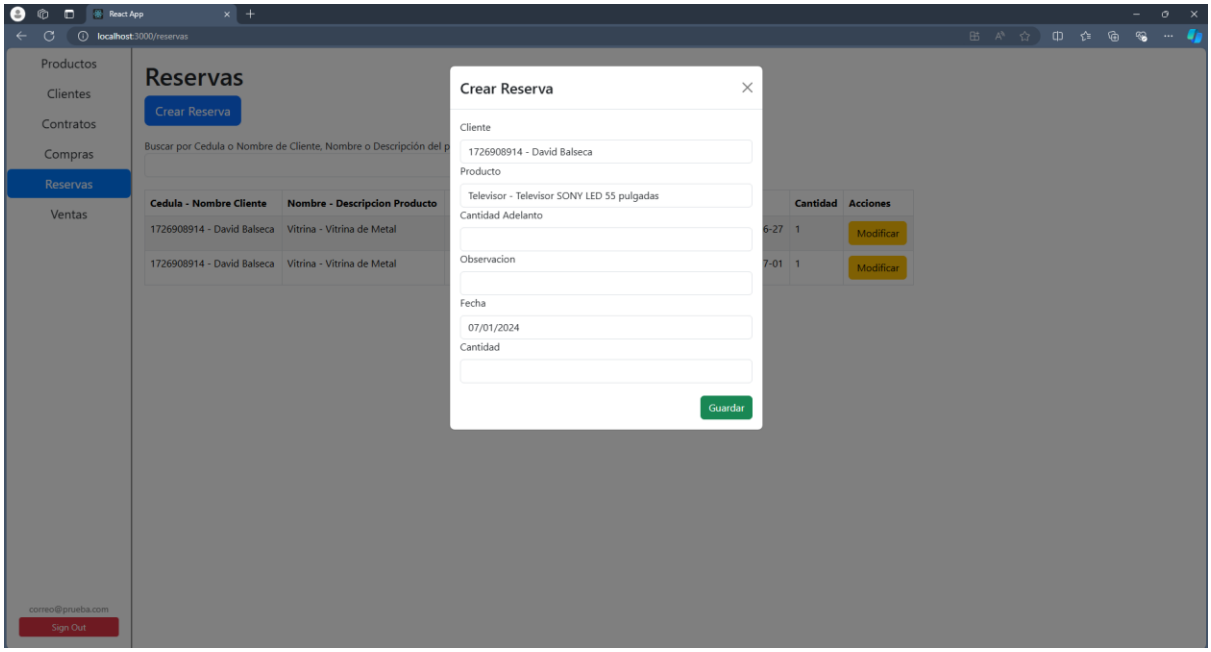

Figure 39 Pantalla Crear Reserva

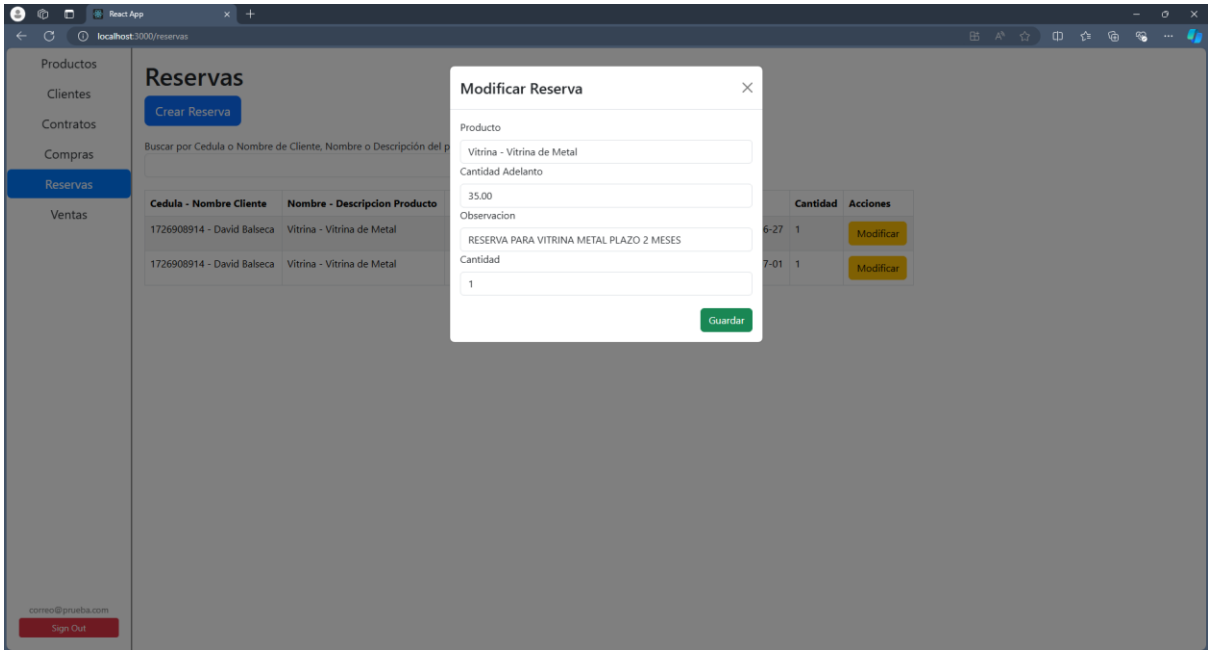

Figure 40 Pantalla Modificar Reserva

4.4.4 H14 Crear la estructura para la gestión de ventas

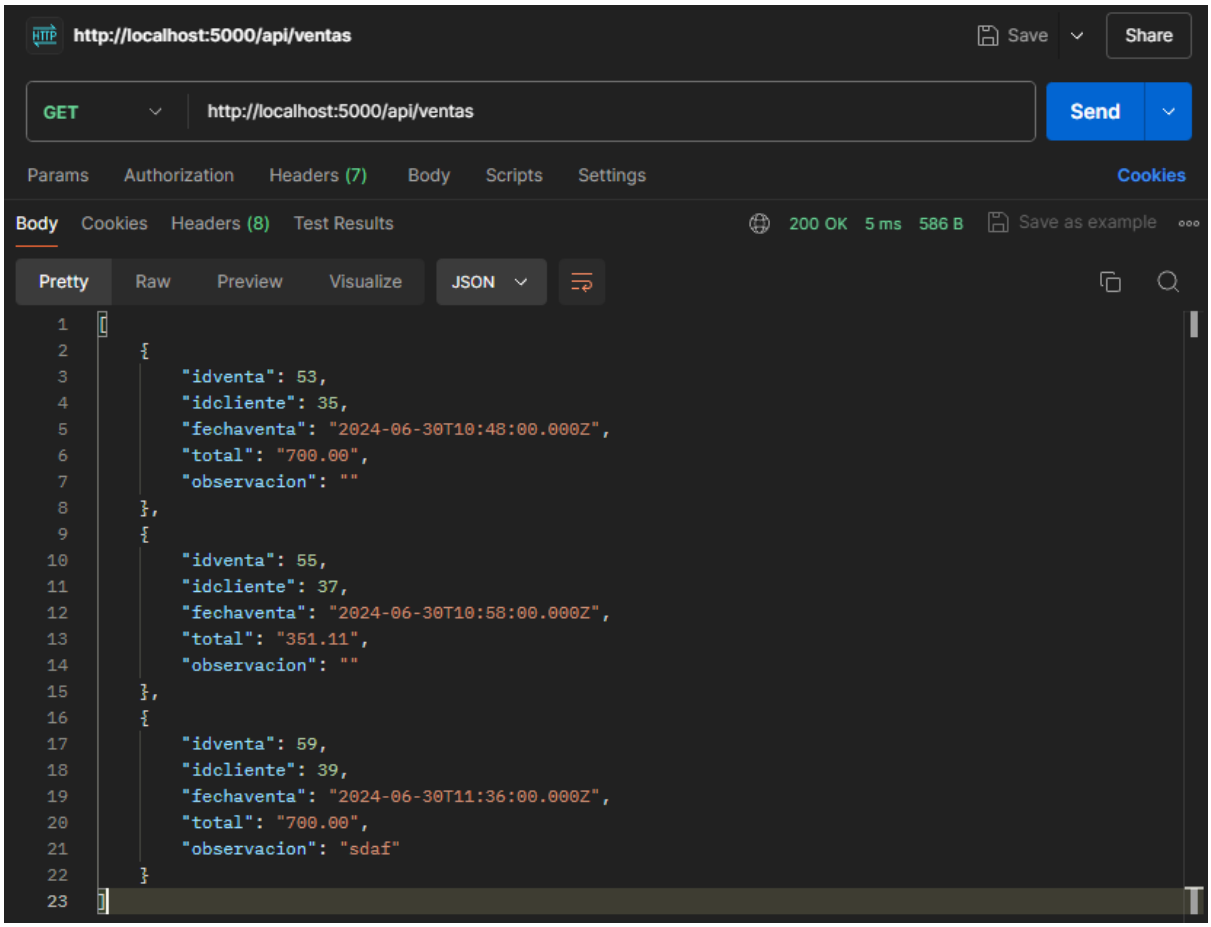

Figure 41 Estructura Venta Método Get - Postman

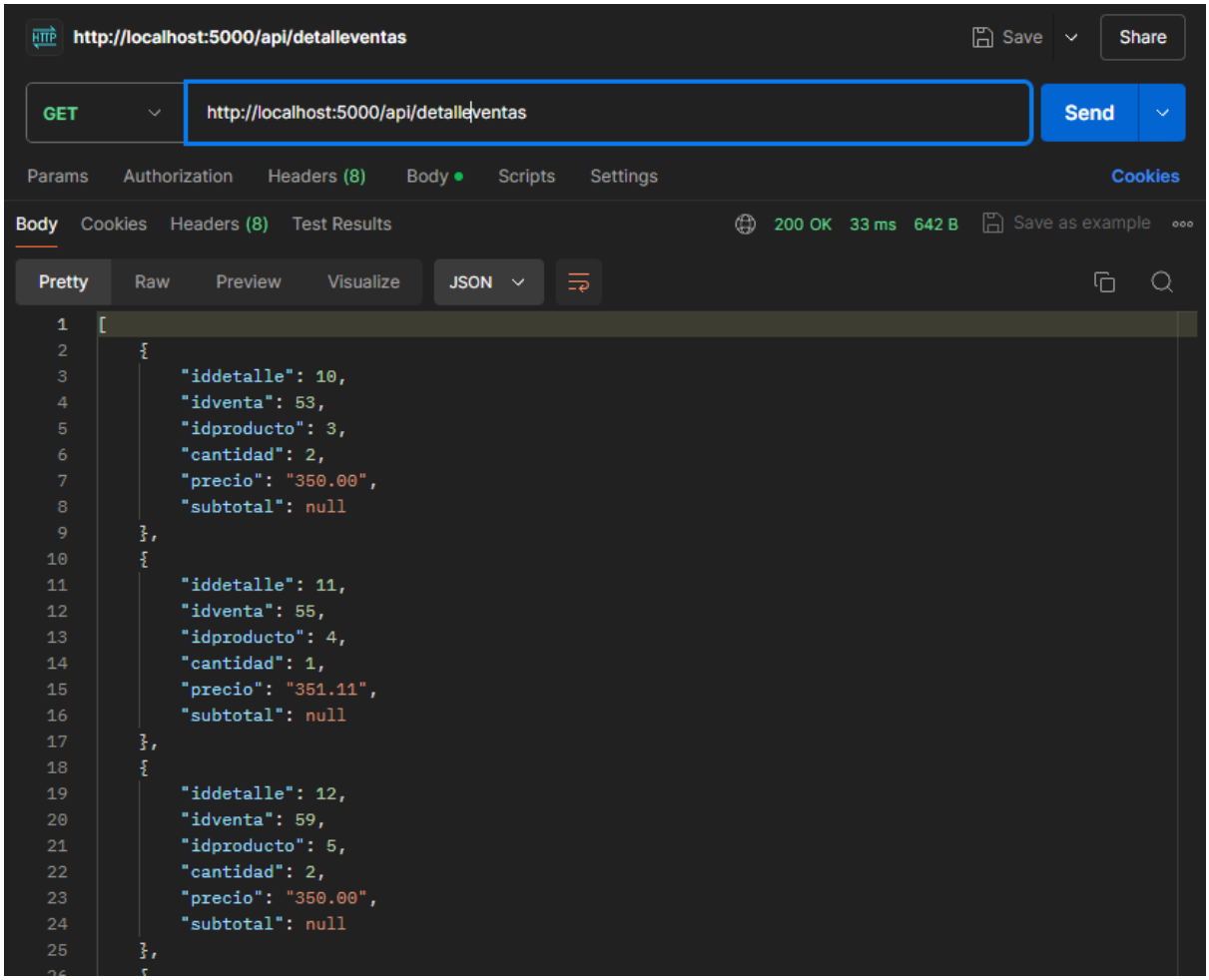

Figure 42 Estructura Detalle Venta Método Get - Postman

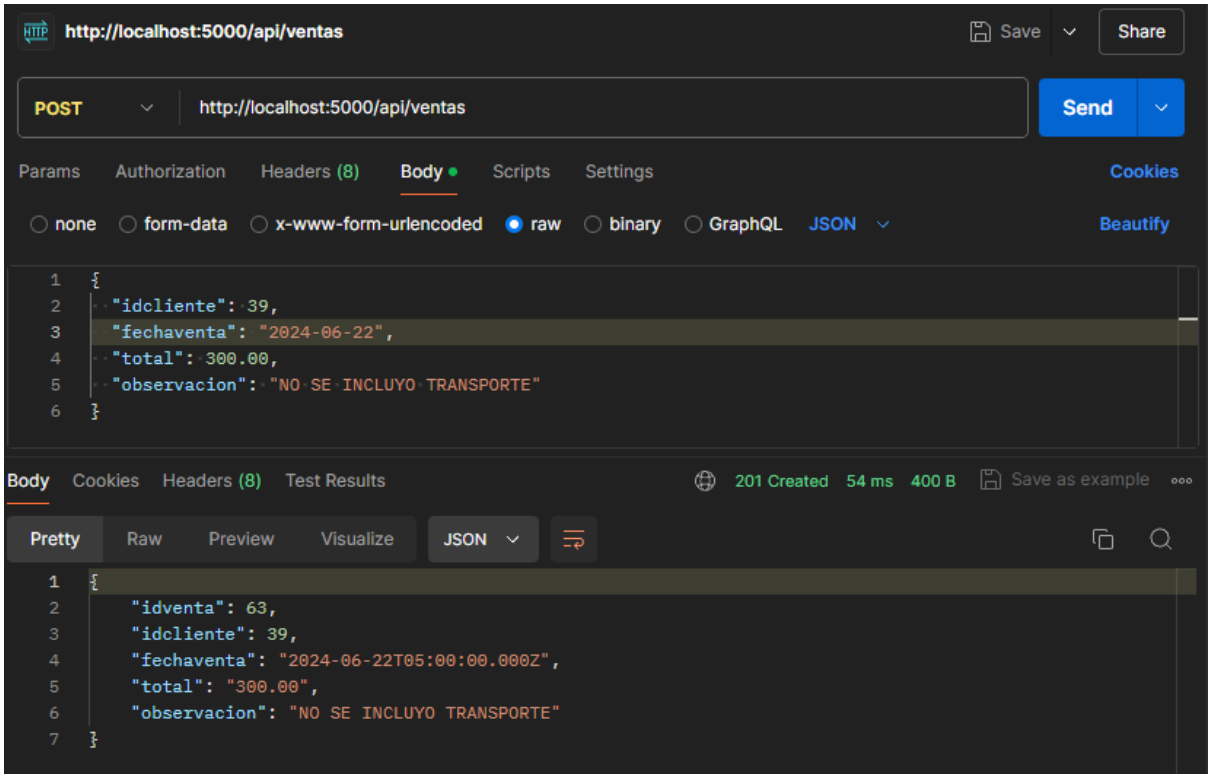

Figure 43 Estructura Venta Método Post - Postman

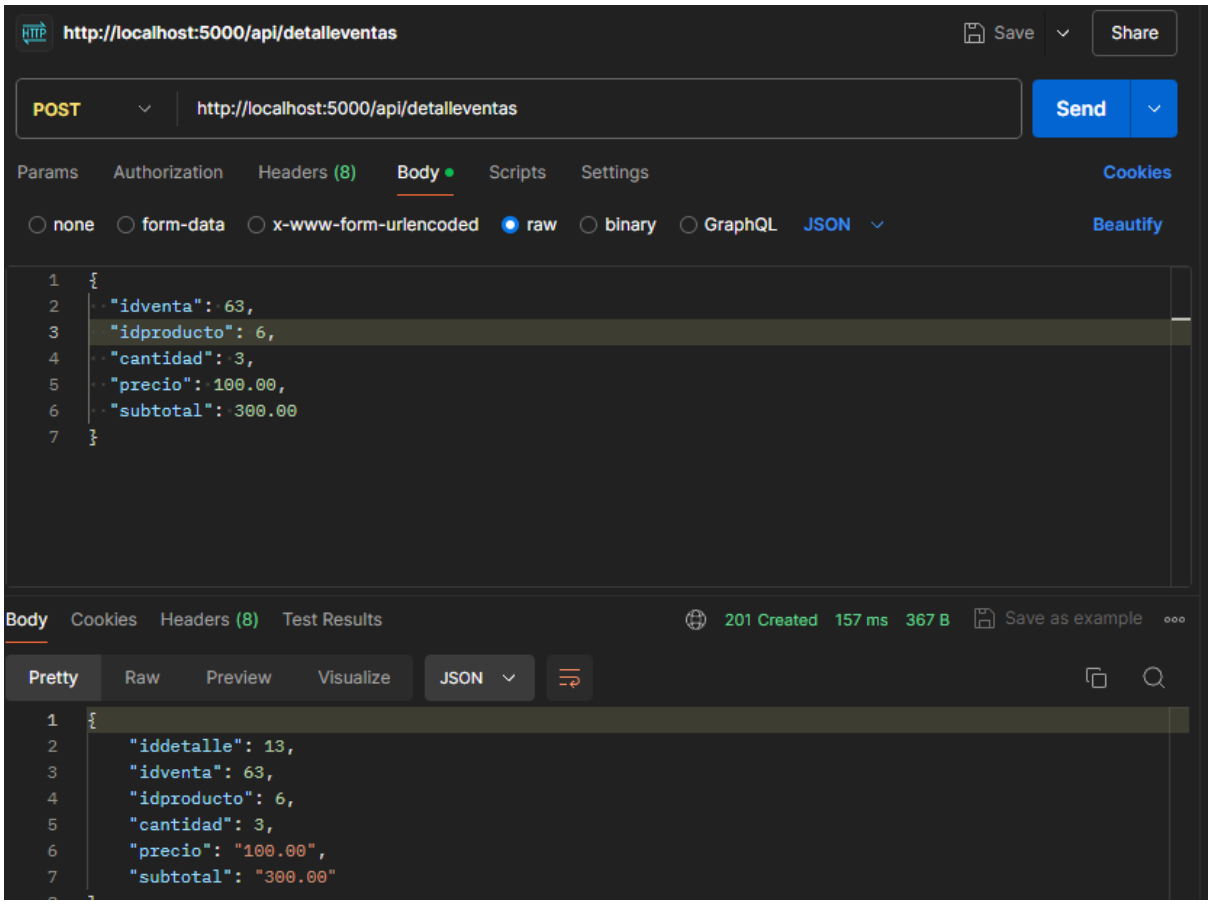

Figure 44 Estructura Detalle Venta Método Post - Postman

4.4.5 H15 Crear las pantallas para la gestión de ventas

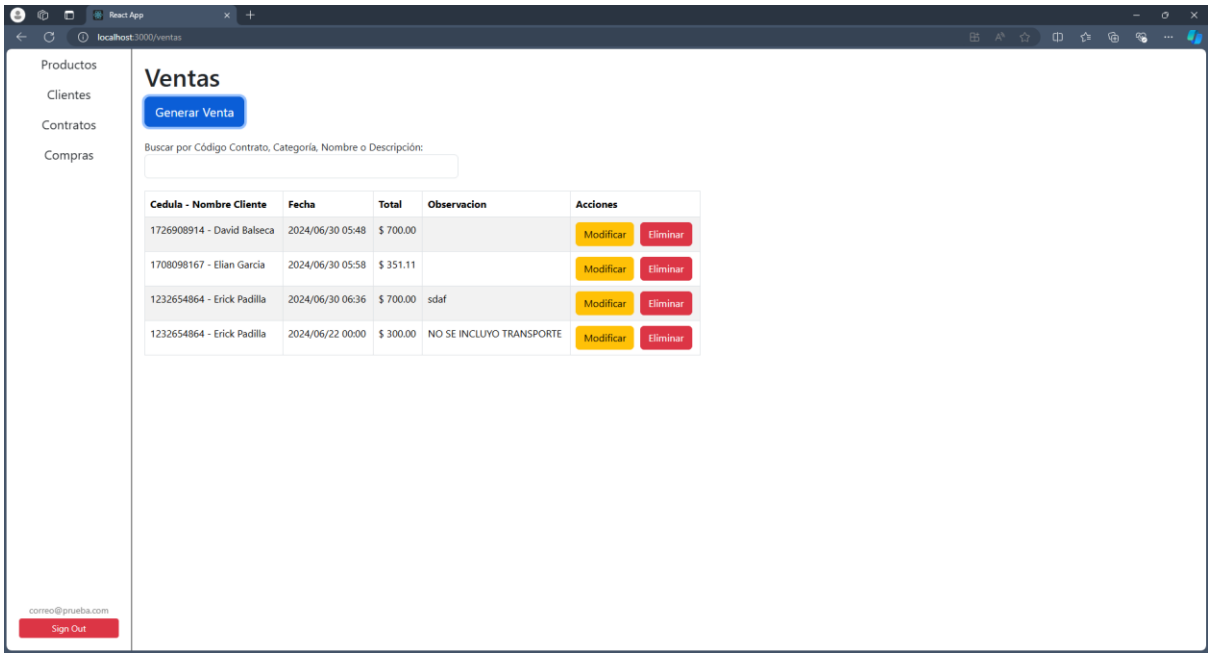

Figure 45 Pantalla Gestión Ventas

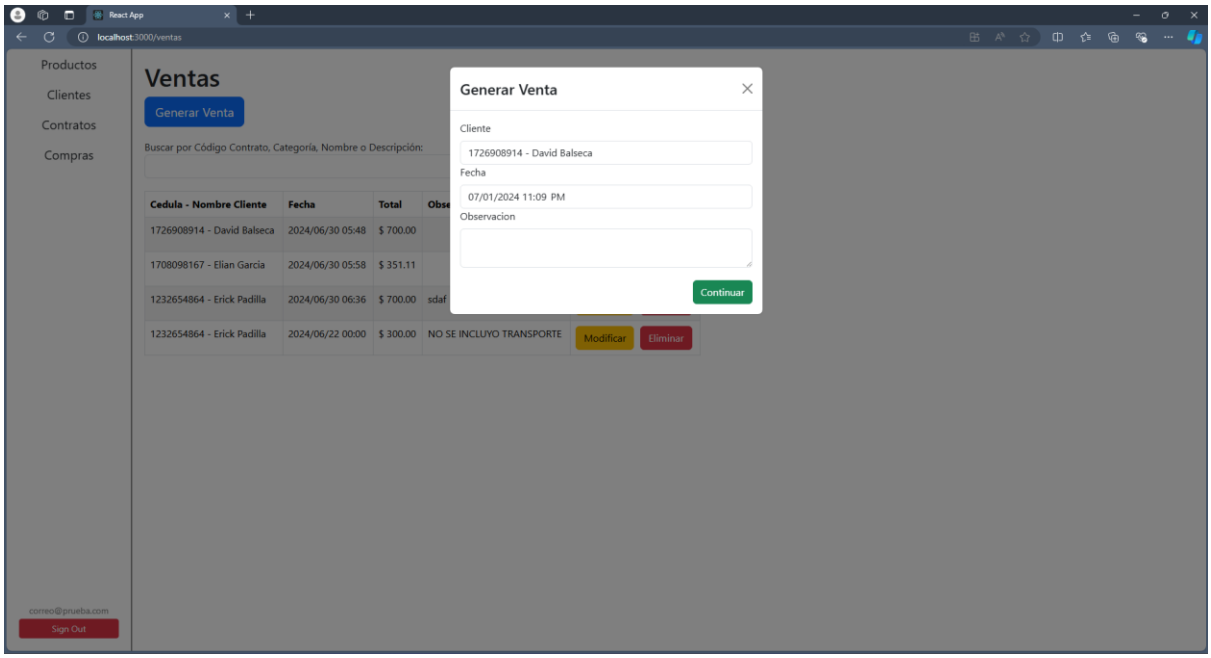

Figure 46 Pantalla Generar Venta

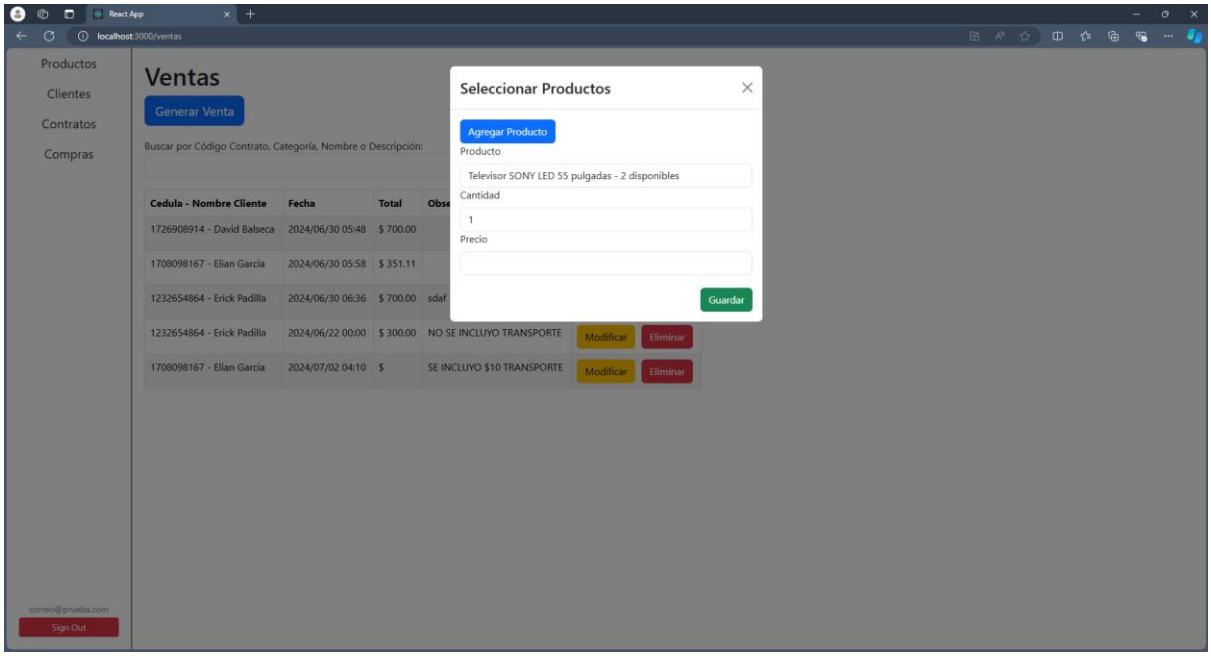

Figure 47 Pantalla Modificar Detalle Venta

5 CONCLUSIONES Y RECOMENDACIONES

Conclusiones:

- El desarrollo del sistema web de gestión de comercialización demostró ser una solución efectiva para automatizar y mejorar los procesos operativos de las PYMES. La transición de sistemas manuales a un sistema automatizado va a reducir significativamente los errores humanos y mejora la eficiencia en la gestión de inventarios, ventas, compras y consignaciones. Esta mejora en los procesos optimiza la operatividad diaria tanto como facilita una mejor toma de decisiones estratégicas.
- La utilización de la metodología ágil Scrum aseguro flexibilidad y adaptabilidad del sistema. El enfoque iterativo e incremental permitió la incorporación de cambios y ajustes basados en retroalimentación continua, asegurando que el sistema cumpla con las necesidades dinámicas del negocio. Resultó en un sistema robusto y capaz de evolucionar con las demandas del mercado y los requerimientos del negocio.
- La adopción de tecnologías modernas como Node.js, Express.js, React.js y PostgreSQL ha proporcionado una plataforma sólida y escalable. El sistema mejora la gestión interna y también añade valor a los clientes al proporcionarles una experiencia de usuario mejorada y servicios más eficientes. Esto fortalece la competitividad de las PYMES en un mercado cada vez más digitalizado y competitivo.
- Cada sprint se enfocó en diferentes módulos del sistema, desde la gestión de usuarios y clientes, hasta contratos, productos, compras, reservas y ventas. La integración entre estos módulos se logró de manera efectiva gracias al modularidad del código y la reutilización de componentes. Esta integración eficiente permitió una gestión fluida de las diferentes operaciones de negocio, optimizando la experiencia del usuario final.
- Aunque se encontraron desafíos durante el desarrollo, como la optimización de consultas con filtros y la validación de entradas de usuario, el desarrollador pudo abordarlos de manera efectiva. Las retrospectivas de sprint fueron esenciales para identificar áreas de mejora, como resultado fue un proceso de desarrollo más robusto y eficiente.

Recomendaciones:

- Para maximizar los beneficios del sistema desarrollado, es crucial invertir en la capacitación continua del personal. Hay que asegurar que todos los usuarios comprendan cómo utilizar el sistema y sus funcionalidades lo cual permitirá una adopción más rápida y efectiva. La capacitación en nuevas tecnologías puede preparar al equipo para futuras actualizaciones y mejoras del sistema.
- Es recomendable establecer un plan de monitoreo y evaluación regular del sistema para identificar áreas de mejora y asegurar que el sistema siga cumpliendo con los objetivos del negocio. La retroalimentación de los usuarios puede proporcionar insights valiosos para optimizar los procesos y funcionalidades.
- A medida que el negocio crezca, es importante considerar la expansión y actualización tecnológica del sistema. Incorporar nuevas funcionalidades, integrar herramientas adicionales y actualizar tecnologías pueden mantener al sistema alineado con las mejores prácticas del sector y las necesidades emergentes del negocio. La inversión en innovación tecnológica continua es clave para mantener la competitividad y asegurar el éxito a largo plazo.
- Implementar un marco de pruebas automatizadas para garantizar la calidad y funcionalidad continua del sistema. La automatización de pruebas, incluyendo pruebas unitarias, y de integración, puede ayudar a identificar errores y regresiones de manera temprana, mejorando la eficiencia del desarrollo y la confiabilidad del sistema.
- Desarrollar y mantener una documentación detallada y actualizada del sistema, incluyendo manuales de usuario, guías de administración y documentación técnica. Esto facilitará la transferencia de conocimiento, la resolución de problemas y la incorporación de nuevos miembros al equipo.
- Explorar y desarrollar integraciones con otros sistemas y herramientas que la empresa ya utilicé o que puedan añadir valor. Esto puede incluir integraciones con sistemas de contabilidad, CRM, herramientas de análisis de datos, entre otros.

BIBLIOGRAFIA

- Boehm, B. (1988). A Spiral Model of Software Development and Enhancement. ACM SIGSOFT Software Engineering Notes, 11(4), 14-24. https://www.cse.msu.edu/~cse435/Homework/HW3/boehm.pdf
- Bootstrap. (2023). Introduction to Bootstrap. https://getbootstrap.com/
- Cohn, M. (2010). Succeeding with Agile: Software Development Using Scrum. Addison-Wesley Professional. https://github.com/manish-old/ebooks-2/blob/master/succeeding-with-agile-software-development-using-scrum.9780321579362.53099.pdf
- Connolly, T., & Begg, C. (2015). *Database Systems: A Practical Approach to Design, Implementation, and Management* (6th ed.). Pearson Education. https://www.cherrycreekeducation.com/bbk/b/Pearson_Database_Systems_A_Practical_Approach_to_Design_Implementation_and_Management_6th_Global_Edition_1292061189.pdf
- Express.js. (2023). *Introduction to Express*. https://expressjs.com/
- Fielding, R. T. (2000). Architectural Styles and the Design of Network-based Software Architectures (Doctoral dissertation, University of California, Irvine). https://ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf
- Laudon, K. C., & Laudon, J. P. (2012). Management Information Systems: Managing the Digital Firm (12th ed.). Pearson Education. https://repository.dinus.ac.id/docs/ajar/Kenneth_C._Laudon,Jane_P_._Laudon_--_Management_Information_System_12th_Edition_.pdf
- Martin, R. C. (2003). *Agile Software Development: Principles, Patterns, and Practices*. Prentice Hall. https://dl.ebooksworld.ir/motoman/Pearson.Agile.Software.Development.Principles.Patterns.and.Practices.www.EBooksWorld.ir.pdf

- Node.js Foundation. (2023). Introduction to Node.js. https://nodejs.org/en
- Pressman, R. S. (2014). *Software Engineering: A Practitioner's Approach* (8th ed.). McGraw-Hill Education. https://invent.ilmkidunya.com/images/Section/12.pdf
- React.dev. (2023). *Introduction to React*. https://react.dev/
- Schwaber, K., & Sutherland, J. (2020). The Scrum Guide. Scrum.org. https://scrumguides.org/scrum-guide.html
- Sequelize. (2023). *Introduction to Sequelize*. https://sequelize.org/
- Sommerville, I. (2011). *Software Engineering* (9th ed.). Pearson Education. https://engineering.futureuniversity.com/BOOKS%20FOR%20IT/Software-Engineering-9th-Edition-by-Ian-Sommerville.pdf
- Tanenbaum, A. S., & Van Steen, M. (2007). Distributed Systems: Principles and Paradigms (2nd ed.). Pearson Education. https://www.redalyc.org/pdf/6380/638067311005.pdf
- Timescale. (2023). *Understanding ACID Compliance in PostgreSQL*. https://www.timescale.com/