UNIVERSIDAD ESTATAL DE BOLÍVAR

FACULTAD DE CIENCIAS ADMINISTRATIVAS, GESTIÓN EMPRESARIAL E INFORMÁTICA

CARRERA DE SOFTWARE

TRABAJO DE INTEGRACIÓN CURRICULAR PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIEROS EN SOFTWARE

FORMA: PROYECTO TECNOLÓGICO

TEMA:

DESARROLLO DE UN SISTEMA INFORMÁTICO PARA LA GESTIÓN Y VENTA DE BOLETOS EN LA COOPERATIVA DE TRANSPORTES “SAN PEDRITO” APLICANDO DOMAIN DRIVEN DESIGN (DDD)

AUTORES:

MESIAS EDUARDO BONILLA GUASTAY
MELANIN VANESSA GANAN ILVAY

DIRECTOR:

ING. DANILO BARRENO

GUARANDA – ECUADOR

2024

TEMA DEL PROYECTO

DESARROLLO DE UN SISTEMA INFORMÁTICO PARA LA GESTIÓN Y VENTA DE BOLETOS EN LA COOPERATIVA DE TRANSPORTES “SAN PEDRITO” APLICANDO DOMAIN DRIVEN DESIGN (DDD)

DEDICATORIA

Este trabajo está dedicado a:

A mi madre, cuyo amor y apoyo incondicional han sido mi luz en los momentos más oscuros. A mi familia y seres queridos, cuya presencia y aliento han sido el sostén en cada paso de este camino, incluso aquellos que ya no están físicamente con nosotros, pero cuyo legado y recuerdo siguen vivos en nuestros corazones. Este proyecto es el fruto de su influencia y dedicación, y lo dedico con gratitud y cariño a todos ustedes, quienes han sido mi inspiración constante y mi fuerza inquebrantable, incluso más allá de esta vida terrenal. Sin su apoyo, nada de esto habría sido posible. Gracias por ser parte de mi historia, incluso desde el otro lado.

Eduardo Guastay

Este trabajo está dedicado con amor y gratitud a dos pilares fundamentales en mi vida: mi familia y mi novio.

A mis Padres Rogelio Ganan y Ana Ilvay, hermanos y abuelos, cuyo amor incondicional y apoyo constante han sido mi mayor fortaleza en este viaje académico. Gracias por estar siempre ahí, por alentarme, cuidarme, aconsejarme y guiarme para perseguir mis sueños y por ser mi refugio en los momentos difíciles. Su amor y apoyo han sido la luz que ha iluminado mi camino.

Y a ti, mi amado Jefferson Criollo, quiero agradecerte por ser mi compañero de vida y mi mayor confidente. Tu amor, tu paciencia y tu constante aliento han sido mi inspiración para superar los desafíos y alcanzar mis metas. Gracias por compartir este viaje conmigo y por ser mi mayor motivación para dar lo mejor de mí en cada paso del camino.

A ti ,mi precioso Hernan Ganan has sido mi mayor fuente de alegría, inspiración y orgullo. A través de cada desafío y cada logro, has sido mi compañero de aventuras, mi motivación para superarme y mi razón para seguir adelante.

Esta tesis es un tributo a su amor incondicional. Les dedico este logro con todo mi corazón, su apoyo inquebrantable y su presencia constante en mi vida. Sin ustedes, este logro no habría sido posible, los amo.

Melanin Ganan

AGRADECIMIENTO

Agradezco a Dios por la fortaleza y la sabiduría que me ha brindado para alcanzar este logro. También quiero expresar mi sincero agradecimiento a la Cooperativa de Transportes 'San Pedrito' por brindarme la oportunidad de trabajar en este proyecto.

Además, quiero extender mi más sincero agradecimiento al Ing. Danilo Barren, mi director de tesis, por su guía experta, paciencia y compromiso durante todo el proceso. Agradezco también a mis pares, Ing. Galuth García e Ing. Henry Albán, por su valioso aporte, críticas constructivas y apoyo en el desarrollo de este proyecto.

Asimismo, quiero agradecer al equipo de CODELY por su inspirador contenido relacionado con la programación y el desarrollo de software, el cual me ha brindado conocimientos adicionales y motivación en mi camino académico y profesional.

Mi gratitud se extiende a la Universidad Estatal de Bolívar en general, por proporcionarme las herramientas y el entorno propicio para mi formación académica. Este logro es el resultado del esfuerzo conjunto de todas estas personas y entidades, a quienes dedico mi más profundo agradecimiento.

Eduardo Guastay

En este momento de gratitud, elevo mi corazón lleno de agradecimiento a Dios, quien ha sido mi luz y mi roca en cada paso de este viaje académico.

A la distinguida Universidad Estatal de Bolívar, le agradezco profundamente por abrirme las puertas del conocimiento.

Quiero expresar mi más sincero reconocimiento al Ing. Danilo Barreno, mi director de tesis, cuya presencia fue una guía luminosa en cada etapa de este proyecto. Su liderazgo visionario, apoyo incondicional y dedicación incansable fueron fundamentales para mi crecimiento académico y profesional durante todo este proceso. A mis pares, Ing. Galuth García e Ing. Henry Albán, les doy gracias desde lo más profundo de mi ser. Su sabiduría, paciencia y apoyo incondicional han sido el timón que ha guiado este barco de conocimiento.

Finalmente, a mi familia y amigos, les debo todo. Su amor incondicional, aliento constante y comprensión infinita han sido mi mayor tesoro en los momentos de desafío y éxito. Sin ustedes a mi lado, este camino habría sido menos brillante y significativo.

Melanin Ganan

CERTIFICADO DE VALIDACIÓN

Ing. Danilo Barreno, Ing. Galuth García e Ing. Henry Alban, en su orden Director y Pares Académicos del Trabajo de Integración Curricular **"DESARROLLO DE UN SISTEMA INFORMÁTICO PARA LA GESTIÓN Y VENTA DE BOLETOS EN LA COOPERATIVA DE TRANSPORTES "SAN PEDRITO" APLICANDO DOMAIN DRIVEN DESIGN (DDD)"** desarrollado por los señores estudiantes **Bonilla Guastay Mesías Eduardo** con **C.I. 0250366515** y **Ganan Ilvay Melanin Vanessa** con **C.I. 0605584358**.

CERTIFICAN

Que, luego de revisado el Trabajo de Integración Curricular en su totalidad, cumple con las exigencias académicas de la carrera SOFTWARE, por lo tanto, autorizamos su presentación y defensa.

Guaranda, 23 de abril del 2024

Ing. Danilo Barreno
Director

Ing. Galuth Garcia
Par Académico

Ing. Henry Alban
Par Académico

DERECHOS DE AUTOR

Nosotros **BONILLA GUASTAY MESÍAS EDUARDO** y **GANAN ILVAY MELANIN VANESSA** portadores de la Cédula de Identidad No **0250366515** y **0605584358** en calidad de autores y titulares de los derechos morales y patrimoniales del Trabajo de Titulación: **DESARROLLO DE UN SISTEMA INFORMÁTICO PARA LA GESTIÓN Y VENTA DE BOLETOS EN LA COOPERATIVA DE TRANSPORTES "SAN PEDRITO" APLICANDO DOMAIN DRIVEN DESIGN (DDD)**, modalidad **TRABAJO TECNOLÓGICO**, de conformidad con el Art. 114 del CÓDIGO ORGÁNICO DE LA ECONOMÍA SOCIAL DE LOS CONOCIMIENTOS, CREATIVIDAD E INNOVACIÓN, concedemos a favor de la Universidad Estatal de Bolívar, una licencia gratuita, intransferible y no exclusiva para el uso no comercial de la obra, con fines estrictamente académicos.

Conservamos a nuestro favor todos los derechos de autor sobre la obra, establecidos en la normativa citada.

Así mismo, autorizamos a la Universidad Estatal de Bolívar, para que realice la digitalización y publicación de este trabajo de titulación en el Repositorio Digital, de conformidad a lo dispuesto en el Art. 144 de la Ley Orgánica de Educación Superior.

Los autores declaran que la obra objeto de la presente autorización es original en su forma de expresión y no infringe el derecho de autor de terceros, asumiendo la responsabilidad por cualquier reclamación que pudiera presentarse por esta causa y liberando a la Universidad de toda responsabilidad.

Firmado electrónicamente por:
MESIAS EDUARDO BONILLA GUASTAY

Eduardo Guastay
cc: 0250366515

Firmado electrónicamente por:
MELANIN VANESSA GANAN ILVAY

Melanin Ganan
cc: 0605584358

ÍNDICE DE CONTENIDOS

TEMA DEL PROYECTO ..... **I**

DEDICATORIA ..... **II**

AGRADECIMIENTO ..... **III**

CERTIFICADO DE VALIDACIÓN ..... **V**

DERECHOS DE AUTOR ..... **VI**

ÍNDICE DE CONTENIDOS ..... **VII**

INDICE DE TABLAS ..... **XI**

INDICE DE FIGURAS ..... **XII**

INTRODUCCIÓN **1**

RESUMEN **3**

ABSTRACT **4**

CAPÍTULO I **5**

FORMULACIÓN GENERAL DEL PROYECTO **5**

1.1 TEMA **5**

1.2 DESCRIPCIÓN DEL PROBLEMA **5**

1.3 JUSTIFICACIÓN **6**

1.4 OBJETIVOS **7**

1.4.1 GENERAL **7**

1.4.2 ESPECÍFICOS **7**

CAPÍTULO II **8**

MARCO TEÓRICO **8**

2.1 ANTECEDENTES **8**

2.2 CIENTÍFICO **10**

2.2.1 DOMAIN-DRIVEN DESIGN (DDD) **10**

2.2.2 ENTITIES (ENTIDADES) **10**

2.2.3 VALUE OBJECTS (OBJETOS DE VALOR) **11**

2.2.4 SERVICES (SERVICIOS) **11**

2.2.5 AGGREGATES (AGREGADOS) **11**

2.2.6 REPOSITORIES (REPOSITORIOS) 13

2.2.7 FACTORIAS 14

2.2.8 ARQUITECTURA EN CAPAS 14

2.2.9 CAPA DE DOMINIO 15

2.2.10 CAPA DE APLICACIÓN 15

2.2.11 CAPA DE INFRAESTRUCTURA 15

2.2.12 ARQUITECTURA HEXAGONAL 16

2.2.13 BOUNDED CONTEXT (CONTEXTO DELIMITADO) 17

2.2.14 COMMAND-QUERY RESPONSIBILITY SEGREGATION (CQRS) 17

2.2.15 EVENT-DRIVEN ARCHITECTURE (ARQUITECTURA ORIENTADA A EVENTOS) 18

2.2.16 EVENTOS DE DOMINIO 19

2.2.17 CONFIG SERVICE 20

2.2.18 MICROSERVICIOS 20

2.3 CONCEPTUAL 21

2.3.1 ARQUITECTURA DE SOFTWARE 21

2.3.2 HIBERNATE 21

2.3.3 PATRÓN CRITERIA 21

2.3.4 PRINCIPIO DE RESPONSABILIDAD ÚNICA 21

2.3.5 PRINCIPIO DE ABIERTO/CERRADO 21

2.3.6 PRINCIPIO DE SUSTITUCIÓN DE LISKOV 22

2.3.7 PRINCIPIO DE SEGREGACIÓN DE INTERFACES 22

2.3.8 PRINCIPIO DE INVERSIÓN DE DEPENDENCIAS 22

2.3.9 RABBITMQ 22

2.3.10 REACTJS 23

2.3.11 UUID 23

2.3.12 KUBERNETES 23

2.3.13 BOLETO DE VIAJE 23

2.3.14 COOPERATIVA DE TRANSPORTES 23

2.4 LEGAL **24**

2.4.1 LEY ORGÁNICA DE TRANSPORTE TERRESTRE TRANSITO Y SEGURIDAD VIAL **24**

2.4.2 LEY ORGÁNICA DE PROTECCIÓN DE DATOS **24**

2.4.3 DE LA INSTITUCIÓN. **24**

CAPÍTULO III **25**

METODOLOGÍA **25**

3.1 METODOLOGÍA DE DESARROLLO DE SOFTWARE (SCRUM) **25**

3.1.1 PROCESO DE SCRUM **25**

3.1.2 ROLES DE SCRUM: **26**

3.1.3 EVENTOS (CEREMONIAS): **26**

3.1.4 ARTEFACTOS: **27**

3.1.5 HISTORIA DE USUARIO **27**

3.1.6 SCRUM Y DOMAIN DRIVEN DESING **27**

3.2 TÉCNICAS E INSTRUMENTOS DE RECOPILACIÓN DE DATOS **28**

3.2.1 ENTREVISTA **28**

CAPÍTULO IV **29**

INGENIERÍA DEL PROYECTO **29**

4.1 ANÁLISIS Y PLANIFICACIÓN **29**

4.1.1 VISIÓN **29**

4.1.2 SCRUM TEAM **29**

4.1.3 STAKEHOLDERS **29**

4.1.4 HISTORIAS DE USUARIO **30**

4.1.5 BACKLOG **33**

4.1.6 PLAN DE PRUEBAS **34**

4.1.6.1 OBJETIVO DE LAS PRUEBAS **34**

4.1.6.2 ALCANCE DE LAS PRUEBAS **34**

4.1.6.3 CRITERIOS DE ACEPTACIÓN **34**

4.1.6.4 PROCEDIMIENTOS DE LAS PRUEBAS **34**

4.2 DISEÑO 35

4.2.1 ARQUITECTURA DE MICROSERVICIOS 35

4.2.2 DEFINICIÓN DE MICROSERVICIOS 36

4.2.3 COMUNICACIÓN ENTRE MICROSERVICIOS 36

4.2.4 ARQUITECTURA DEL SOFTWARE 50

CONCLUSIONES 72

RECOMENDACIONES 73

BIBLIOGRAFÍA 74

ANEXOS 76

INDICE DE TABLAS

Tabla 1: Resumen historias de usuario 30
Tabla 2: Backlog 33
Tabla 3: Product Backlog Sprint 1 52
Tabla 4 Product Backlog Sprint 2 59
Tabla 5 Product Backlog Sprint 3 63
Tabla 6 Product Backlog Sprint 4 67

INDICE DE FIGURAS

Figura 1: Diagrama de relación entre los building blocks 10
Figura 2: Ciclo de vida de una entidad..... 11
Figura 3: Agregado 12
Figura 4: Relación Entre Agregados. 12
Figura 5:Obtención de datos mediante el Patrón Criteria 13
Figura 6: Patrón factoría..... 14
Figura 7: Representación Arquitectura en Capas..... 15
Figura 8: Representación Arquitectura Hexagonal 16
Figura 9: Representación CQRS 18
Figura 10: Elementos Arquitectura basa en eventos. 19
Figura 11: Representación Microservicios..... 20
Figura 12: Proceso Scrum 25
Figura 13: Arquitectura de Microservicios 35
Figura 14: Comunicación entre microservicios 36
Figura 15: Estructura de Directorios Arquitectura Hexagonal 51
Figura 16 Cobertura de los TEST 51
Figura 17 Diagrama E-R Servicio de Usuarios..... 53
Figura 18: Pantalla Iniciar Sesión 54
Figura 19: Pantalla para restaurar la contraseña. 54
Figura 20: Despliegue del servicio..... 59
Figura 21 Diagrama E-R Servicio de Boletería 60
Figura 22 Interfaz para crear un Bus..... 61
Figura 23: Formulario para crear la carrocería 61
Figura 24 Interfaz para crear una Ruta..... 62
Figura 25 Diagrama E-R servicio de contabilidad..... 64
Figura 26 Diagrama E-R Servicio de Facturación Electrónica..... 65
Figura 27 Interfaz del plan de cuentas 66
Figura 28 Interfaz para crear un asiento contable 66
Figura 29 Interfaz del sistema..... 67
Figura 30 Servicios desplegados en GKE..... 70
Figura 31 Balanceador de Carga..... 71

Figura 32: Consumo de CPU 71

INTRODUCCIÓN

Domain-Driven Design (DDD), introducido por Eric Evans en su libro “Domain-Driven Design - Tackling Complexity in the Heart of Software”, redefine el diseño de software al incorporar una perspectiva no solo técnica, sino también organizativa. Este enfoque pone un énfasis especial en comprender a fondo el dominio de negocio, fomentando la colaboración entre expertos y desarrolladores mediante la creación de un lenguaje compartido.

En el ámbito del desarrollo de sistemas informáticos para la gestión y venta de boletos, existe una brecha en cuanto a la aplicación efectiva de metodologías que aborden de manera precisa las complejidades y relaciones del dominio específico de la Cooperativa de Transportes “San Pedrito”. Aquí es donde Domain Driven Design a diferencia de enfoques convencionales, ofrece herramientas y conceptos que permiten modelar el dominio (lógica de negocio) de manera más precisa, proporcionando así una solución a las necesidades específicas de la cooperativa.

Mediante la aplicación de DDD, no solo se desarrolló un sistema informático eficiente, sino también una herramienta estratégica que refleja con precisión los procesos de gestión y venta de boletos, promoviendo así una mayor eficiencia operativa y una experiencia mejorada para los usuarios finales. Al haber optado por una metodología que fomente la colaboración y la comprensión profunda del dominio, se buscó no solo cerrar la brecha identificada, sino también establecer un estándar más elevado para el desarrollo de sistemas informáticos en el contexto específico de la Cooperativa de Transportes “San Pedrito”.

El capítulo I describe la problemática en la gestión y venta de boletos de la Cooperativa de Transportes “San Pedrito”. Justifica la necesidad de desarrollar una aplicación informática, identificando beneficios y alcance.

El capítulo II explora los antecedentes relacionados con proyectos similares y fundamentos científicos y conceptuales.

El capítulo III detalla cómo aplicar la metodología de desarrollo de software (SCRUM) en el proyecto, además de las técnicas de recopilación de datos.

En el capítulo IV detalla la implementación especifica del proyecto utilizando la metodología SCRUM, Se destacan los elementos característicos de esta

metodología en la ingeniería del software, como la planificación de sprints, el product backlog y el desarrollo de los sprints.

RESUMEN

El sistema desarrollado para la Cooperativa de Transportes “San Pedrito” ha abordado con éxito varios desafíos, como la venta duplicada del número de asiento, la falta de control en la disponibilidad de asientos y la ineficiencia en la administración de la información. Estos problemas se resolvieron mediante la implementación de una aplicación informática que utiliza tecnologías avanzadas como Domain-Driven Design (DDD), Scrum, microservicios y GraphQL.

Con la adopción del enfoque de Domain-Driven Design (DDD), se logró una clara separación de la lógica de negocio del resto de la aplicación, lo que permitió resolver los problemas de duplicación de ventas de asientos y mejorar la gestión de la información. Además, la metodología Scrum posibilitó una entrega iterativa y adaptativa, lo que permitió abordar los desafíos de manera eficiente y responder a los cambios en los requisitos del proyecto.

La arquitectura basada en microservicios, utilizando Spring Boot como framework principal, brindó modularidad y flexibilidad al sistema. La implementación de GraphQL facilitó la comunicación entre los microservicios y permitió consultas más eficientes, mejorando así la experiencia del usuario final.

Además, se adoptó el patrón de diseño API Gateway para gestionar la comunicación entre los microservicios, simplificando la interacción cliente-servidor y centralizando la lógica de enrutamiento y seguridad. La arquitectura dirigida por eventos permitió una integración más flexible y escalable entre los microservicios, lo que redujo la dependencia entre ellos y facilitó la gestión de la concurrencia y la escalabilidad.

Es importante destacar que la evolución constante del sistema requiere capacitación continua del personal para garantizar su mantenimiento óptimo y aprovechar al máximo todas las funcionalidades ofrecidas por la aplicación.

Palabras clave: Domain Driven Design, Transporte, GraphQL, Arquitectura hexagonal, Event-driven-architecture.

ABSTRACT

The system developed for the Cooperativa de Transportes "San Pedrito" has successfully addressed several challenges, such as duplicate seat number sales, lack of control over seat availability, and inefficient information management. These problems were solved by implementing a software application using advanced technologies such as Domain-Driven Design (DDD), Scrum, microservices, and GraphQL.

By adopting the Domain-Driven Design (DDD) approach, a clear separation of the business logic from the rest of the application was achieved, solving the problems of duplicate seat sales and improving information management. In addition, the Scrum methodology enabled iterative and adaptive delivery, allowing challenges to be addressed efficiently and responding to changes in project requirements.

The microservices-based architecture, using Spring Boot as the main framework, provided modularity and flexibility to the system. The implementation of GraphQL facilitated communication between the microservices and enabled more efficient queries, thus improving the end-user experience.

In addition, the API Gateway design pattern was adopted to manage communication between the microservices, simplifying client-server interaction and centralizing routing and security logic. The event-driven architecture allowed for more flexible and scalable integration between microservices, reducing dependency between them and facilitating concurrency management and scalability.

It is important to note that the constant evolution of the system requires continuous staff training to ensure optimal maintenance and to take full advantage of all the functionalities offered by the application.

Keywords: Domain Driven Design, Transport, GraphQL, Hexagonal architecture, Event-driven-architecture.

CAPÍTULO I

FORMULACIÓN GENERAL DEL PROYECTO

1.1 Tema

DESARROLLO DE UN SISTEMA INFORMÁTICO PARA LA GESTIÓN Y VENTA DE BOLETOS EN LA COOPERATIVA DE TRANSPORTES “SAN PEDRITO” APLICANDO DOMAIN DRIVEN DESIGN (DDD)

1.2 Descripción del Problema

La Cooperativa de Transportes “San Pedrito” es una empresa que se dedica al transporte de pasajeros a diversas localidades de Ecuador, gestionaba y vendía boletos a través de un sistema de escritorio descentralizado ubicado en diferentes puntos de venta. Sin embargo, este enfoque descentralizado conllevaba a diversos problemas, como la venta duplicada de boletos a diferentes personas, falta de visibilidad en tiempo real de la disponibilidad de asientos y dificultades en la administración de la información de los pasajeros, entre los inconvenientes más comunes que tenían eran:

- **Venta Duplicada del número de Asiento**: Debido a la falta de comunicación entre los puntos de venta, se registraban casos en los que un mismo número de asiento se vendía a varias personas, lo que generaba conflictos y malestar entre los pasajeros.
- **Falta de Control en la Disponibilidad de Asientos:** Los puntos de venta no tenían una visión en tiempo real de la disponibilidad de asientos en los diferentes viajes, lo que causaban sobreventa o sobre venta de boletos.
- **Ineficiencia en la Administración de la Información:** El sistema no permitía una administración eficiente de la información de los pasajeros, lo que dificulta el seguimiento de ventas de boletos, cancelaciones y cambios en los itinerarios.

1.3 Justificación

La Cooperativa de Transportes “San Pedrito” reconoció la urgente necesidad de implementar un sistema informático para la gestión y venta de boletos. Esta iniciativa surgió como una respuesta fundamentada ante los desafíos derivados de la descentralización en la venta de boletos. La motivación principal radicó en la búsqueda de la modernización y optimización de los procesos existentes para superar problemas operativos. Esta inversión en tecnología no solo superó los obstáculos que existían, sino que también se estableció como un paso estratégico hacia la mejora continua y la eficiencia en la gestión de la cooperativa.

El sistema se caracteriza por contar con un módulo de contabilidad que se encarga de gestionar y organizar la información financiera de la empresa, sus funciones principales son registro de transacciones, libro mayor, libro diario e informes.

Además, controla en tiempo real de la disponibilidad de asientos, que garantiza la asignación de asientos únicos.

La centralización de la gestión no solo reduce los costos de mantenimiento, sino que también simplifica la administración de los puntos de venta, promoviendo así una mayor eficacia operativa.

La implementación del sistema fue indispensable, y los beneficios resultantes de su aplicación son significativos tanto para la cooperativa como para sus operaciones y clientes. La cooperativa estuvo firmemente comprometida con la mejora continua de sus procesos, considerando este proyecto como un paso esencial hacia la modernización y automatización de procesos.

La implementación de Domain Driven Design (DDD) en el proyecto de la Cooperativa de Transportes “San Pedrito” representó la adopción de un enfoque metodológico altamente estructurado. Este enfoque actuó como un plano detallado, proporcionando una guía precisa para la construcción de un sistema de venta de boletos que se alinean de manera óptima con los requisitos y desafíos técnicos específicos de la cooperativa. Este enfoque de diseño contribuyó a establecer un lenguaje común entre todos los miembros del equipo, evitando confusiones y mejorando la colaboración entre los que trabajaron en el proyecto. Además, permitió construir un sistema que se adapta fácilmente a los cambios en el negocio,

asegurando que la cooperativa pudiera crecer y ajustarse sin problemas a medida que evolucionan sus operaciones.

El proyecto se enmarcó en la línea de investigación Ingeniería De Software, Redes y Telecomunicaciones, sublínea Diseño e implementación de sistemas de información de la carrera de Software.

1.4 Objetivos

1.4.1 General

Desarrollar un sistema informático para la gestión y venta de boletos en la Cooperativa de Transportes “San Pedrito” aplicando Domain Driven Design (DDD).

1.4.2 Específicos

- Analizar los procesos de negocio de la Cooperativa de Transportes “San Pedrito” relacionados con la gestión y venta de boletos.
- Definir la arquitectura del sistema basada en DDD.
- Implementar sistema informático para la gestión y venta de boletos.

CAPÍTULO II

MARCO TEÓRICO

2.1 Antecedentes

Según (SANDOVAL POZO, 2021) en su tesis de pregrado titulada “Aplicación Web para mejorar la Gestión Administrativa de la Cooperativa de Transportes Huaca - Julio Andrade", propusieron el desarrollo de un sistema informático con el objetivo de mejorar la gestión administrativa de la Cooperativa ubicada en la ciudad de Tulcán. El estudio identificó una problemática significativa: la ausencia de un archivo físico adecuado para la documentación, lo cual estaba ocasionando pérdidas, demoras y retrasos en los trámites administrativos. Para hacer frente a este desafío, se emplearon las tecnologías PHP y JavaScript en el desarrollo de la aplicación, respaldadas por una base de datos en MySQL. La aplicación se organiza en módulos diseñados específicamente para la administración y control del talento humano, facilitando las labores del personal administrativo y asegurando la agilidad y seguridad de los procesos. Además, la aplicación brinda a los usuarios la capacidad de mantenerse informados sobre la caducidad de sus documentos personales y reglamentarios. Se destaca la importancia de capacitar al personal para aprovechar eficientemente esta herramienta tecnológica, con el objetivo de optimizar la toma de decisiones gerenciales.

(Cando Jacome & Tenesaca Pérez, 2021) en su Monografía titulada Desarrollo de una aplicación web y móvil para la automatización en la gestión de venta y reserva de boletos en la Cooperativa de Transportes Ambateñita de la ciudad de Ambato crearon una aplicación para ayudar a la cooperativa a vender y reservar boletos de manera más eficiente, ya que no tenían una aplicación para esto. Recopilaron y analizaron los requisitos para el sistema web y móvil propuestos por la cooperativa. Utilizaron herramientas como MySQL para organizar la información en la base de datos, asegurando la integridad de los datos. La metodología Scrum fue utilizada para guiar el proceso ágil de codificación y diseño del sistema. Finalmente, entregaron tanto el sistema web como la aplicación móvil de manera segura,

cumpliendo con los requisitos establecidos en el certificado de implementación de la cooperativa.

Según (SALAN VILLENA, 2015) es su tesis de pregrado titulada Aplicación web para la gestión de rutas y pre-reservas de la cooperativa de transportes El Dorado, se abordó la carencia de un sistema de facturación de boletos y la realización de procesos manuales en la cooperativa. La ausencia de una guía de rutas y la falta de información sobre las unidades por parte de los socios revelaban la necesidad de automatizar los procesos de reservas.

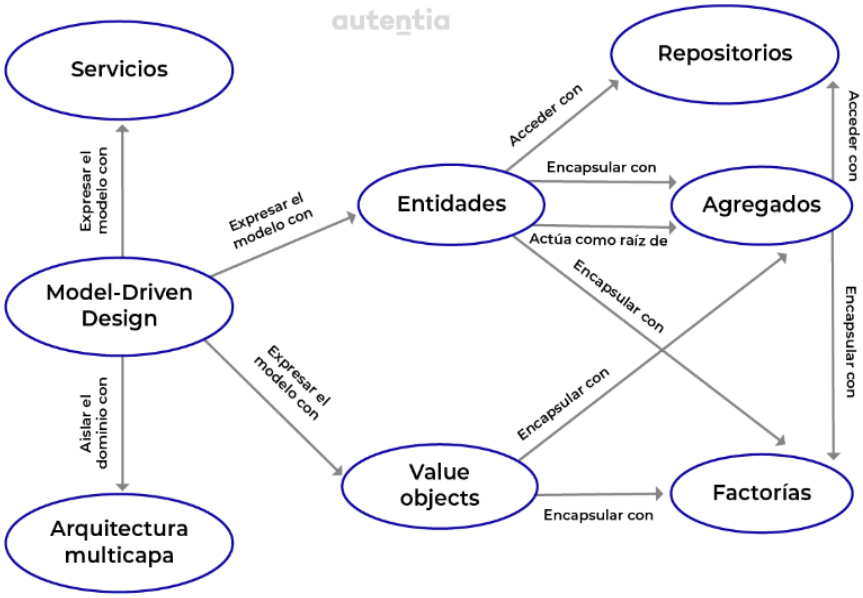
Para solucionar estos problemas, se desarrolló una aplicación utilizando MySQL y Visual Studio, aprovechando software libre para facilitar la gestión de rutas y lograr un control exhaustivo. Este desarrollo mejoró significativamente los procesos automatizados de la cooperativa, permitiendo una gestión más eficiente y proporcionando a los socios información detallada sobre las rutas y disponibilidad de unidades.

2.2 Científico

2.2.1 Domain-Driven Design (DDD)

Es un enfoque de diseño de software propuesto por Eric Evans en su libro "Domain-Driven Design: Tackling Complexity in the Heart of Software". DDD se centra en la colaboración estrecha entre expertos del dominio y desarrolladores para construir un modelo que refleje con precisión el negocio subyacente, en la figura 1 podemos observar la relación que existe entre los bloques que este lo componen.

Figura 1: Diagrama de relación entre los building blocks



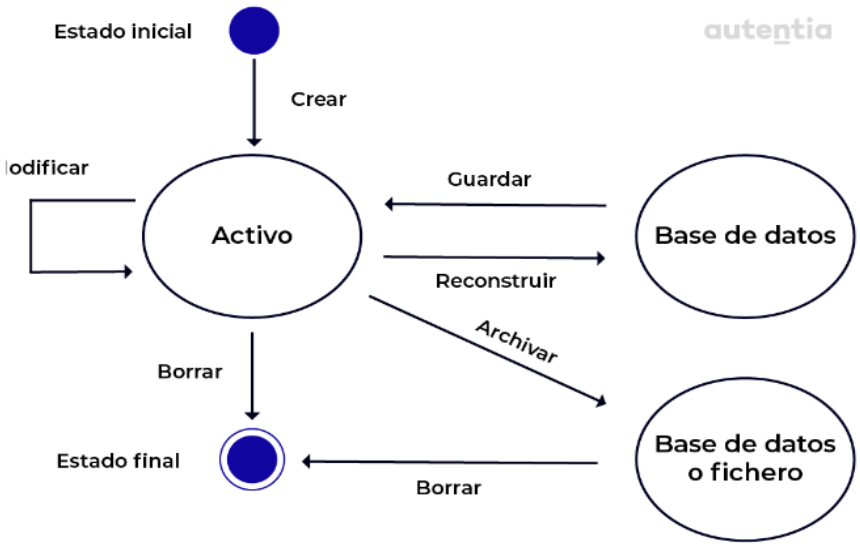
Fuente: Relación entre bloques tomado de (Autentia S.L., 2023)

2.2.2 Entities (Entidades)

Según (Evans, 2003) las "entidades" son como objetos que tienen una identidad única y que son distinguibles a lo largo del tiempo. Estas representan objetos clave dentro del dominio y son fundamentales para modelar la lógica de negocio.

Una entidad se refiere a un objeto en un dominio específico que puede estar compuesto por varios objetos de valor para enriquecer su significado. Una entidad en un dominio es un objeto autónomo con un ciclo de vida (Figura 2) definido. La finalidad de definir entidades en nuestro dominio es evitar el uso de tipos primitivos, proporcionar significado a nivel de código y facilitar la incorporación de la lógica de negocio en el dominio (Autentia S.L., 2023)

Figura 2: Ciclo de vida de una entidad.



Fuente: Ciclo de una entidad tomado de (Autentia S.L., 2023)

2.2.3 Value Objects (Objetos de Valor)

Según (Fowler, 2003) Son objetos que no tienen una identidad única y se definen únicamente por sus atributos. Los "objetos de valor" son inmutables y representan conceptos que son importantes, pero no necesitan una identidad propia.

2.2.4 Services (Servicios)

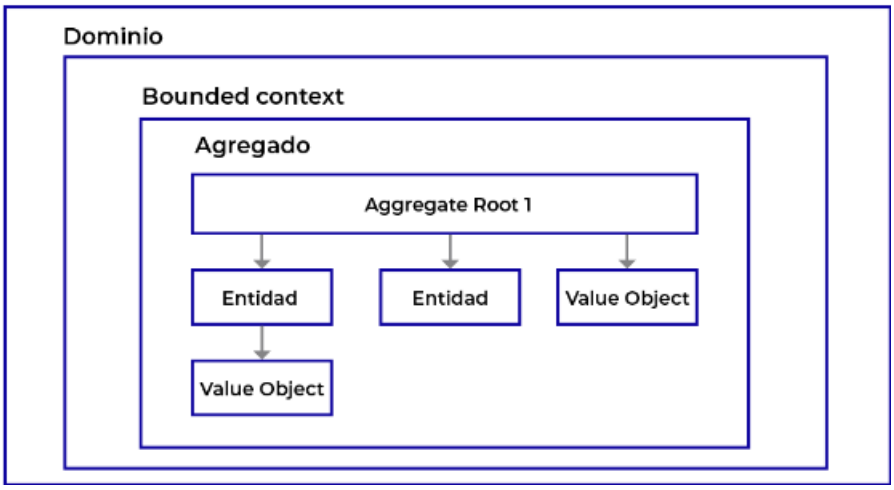
En DDD, los servicios son acciones que no pertenecen directamente a entidades u objetos de valor, desacoplando tareas en capas como infraestructura, aplicación y dominio. Se definen por la acción que realizan y no por un concepto de negocio específico. Pueden ser de dominio (orquestando funcionalidades), de aplicación (coordinando acciones entre servicios) y de infraestructura (proveyendo acceso a recursos externos). Sus operaciones no tienen estado y manejan conceptos de negocio (Autentia S.L., 2023).

2.2.5 Aggregates (Agregados)

Un agregado es una agrupación de entidades y objetos de valor que, en términos conceptuales, están vinculados como un conjunto, estos definen límites dentro de los cuales las operaciones se realizan de manera consistente y deben ser tratados como una unidad en el sistema. (Evans, 2003).

Cada Agregado cuenta con una entidad raíz (aggregate root) que sirve como punto de dependencia para las demás entidades y objetos de valor.

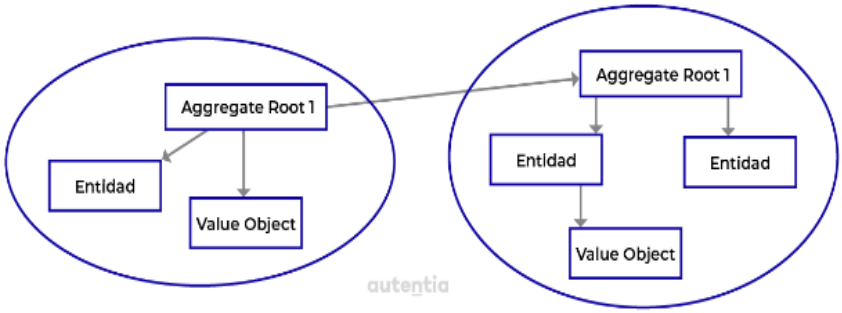
Figura 3: Agregado



Fuente: Agregados tomado de (Autentia S.L., 2023)

Cualquier modificación a un componente interno del agregado debe llevarse a cabo a través de métodos accesibles en la entidad raíz, asegurando la integridad y estado del conjunto en memoria. Las referencias desde fuera del agregado deben dirigirse exclusivamente a la raíz de este (Autentia S.L., 2023).

Figura 4: Relación Entre Agregados.



Fuente: Relación entre agregados tomado de (Autentia S.L., 2023)

Los agregados se relacionan entre sí exclusivamente a través de los identificadores de sus entidades raíces. Son la unidad fundamental para la transferencia de datos, ya que se cargan o guardan como entidades completas. Las transacciones no deben traspasar los límites de los agregados. Los clientes que consumen un agregado no necesitan conocer los detalles de implementación ni navegar entre referencias internas. Los atributos, propiedades, elementos y comportamientos internos del agregado no deben ser accesibles para el cliente (Autentia S.L., 2023).

2.2.6 Repositories (Repositorios)

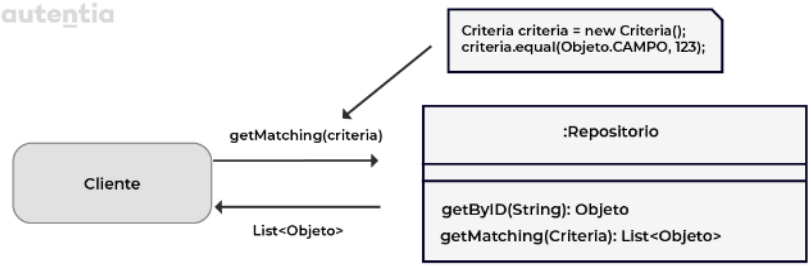
Según (Fowler, 2003) Los repositorios son responsables de la persistencia de los agregados. Proporcionan una interfaz para acceder y almacenar agregados en una forma que sea alineada con el modelo del dominio.

Existen tres maneras de obtener objetos durante su ciclo de vida: crear uno nuevo, obtenerlo mediante la asociación con otro objeto existente, o construirlo utilizando atributos de una base de datos. En el caso de la última opción, se emplea el patrón repositorio, que encapsula el acceso a la base de datos para obtener los valores necesarios. Este enfoque permite que el cliente del repositorio no se preocupe por cómo se recupera la información ni por la tecnología utilizada, manteniendo el enfoque en el dominio (Autentia S.L., 2023).

Un repositorio debe proporcionar al menos métodos para añadir, recuperar y eliminar objetos de un tipo, utilizando consultas a la base de datos para estas operaciones. También puede ofrecer métodos para que el cliente recupere datos según criterios específicos, como el identificador o un valor de atributo, devolviendo objetos individuales o colecciones que cumplan con el criterio. Además, puede incluir funciones que realicen cálculos sobre la base de datos, como sumas de atributos o el conteo de objetos que cumplen con un criterio (Autentia S.L., 2023).

Además, los repositorios pueden ofrecer métodos que aceptan especificaciones de criterios de consulta proporcionadas por el cliente. Esto permite que el cliente defina los criterios sin preocuparse por la tecnología utilizada para recuperar los datos, ya que es responsabilidad del repositorio transformar esos criterios en consultas a la base de datos (Autentia S.L., 2023).

Figura 5:Obtención de datos mediante el Patrón Criteria

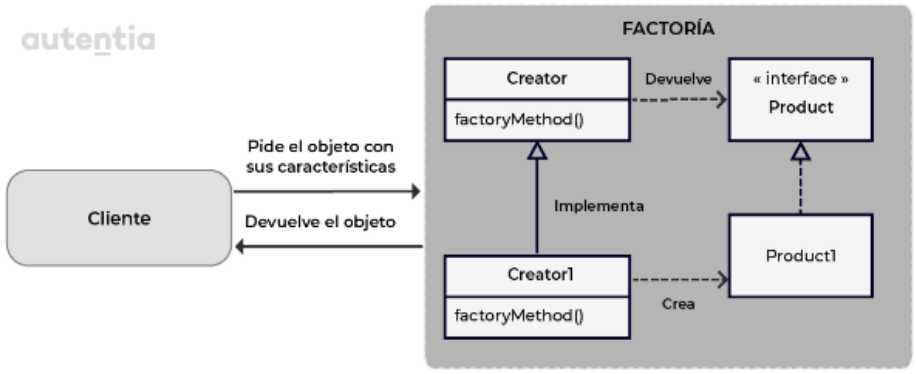


Fuente: Patrón criterio tomado de (Autentia S.L., 2023)

2.2.7 Factorias

Cuando la creación de objetos o agregados se vuelve complicada, el patrón factoría (factory) puede simplificar el proceso y ocultar la estructura interna de los objetos creados. Evita que el cliente, especialmente si forma parte de la capa de aplicación, asuma responsabilidades indebidas, lo cual violaría la encapsulación de los objetos de dominio y agregados. La creación de objetos complejos es responsabilidad de la capa de dominio, y para abordar esto, se introducen nuevos objetos en el diseño del dominio que no son entidades, value objects ni servicios, pero son esenciales para la capa de dominio (Autentia S.L., 2023).

Figura 6: Patrón factoría



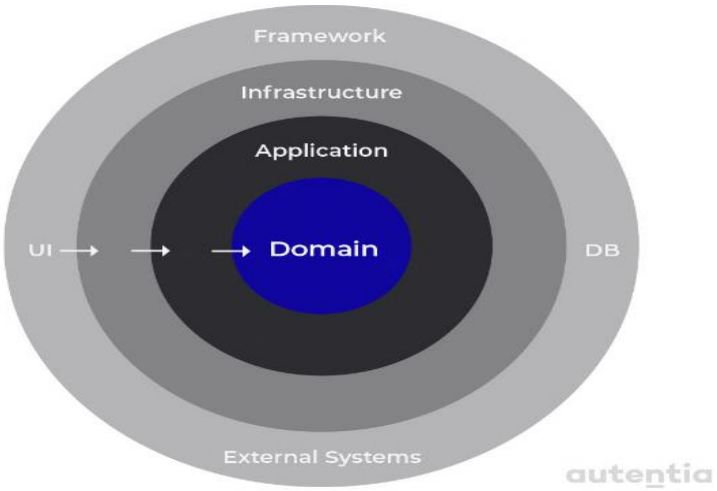
Fuente: Patrón Factoría tomado de (Autentia S.L., 2023)

Cada lenguaje orientado a objetos ofrece un método para crear objetos, como los constructores, pero surge la necesidad de mecanismos de construcción más abstractos que estén desacoplados de otros objetos. Un componente del programa encargado de la creación de objetos se denomina factoría.

2.2.8 Arquitectura en capas

Para diseñar una arquitectura por capas siguiendo los principios de DDD, es esencial comenzar con la regla de dependencia, asegurando que un elemento de una capa solo dependa de otros elementos dentro de la misma capa o de capas más internas. Esto facilita la incorporación y adaptación de nuevas piezas de código a medida que el proyecto crece, simplificando el mantenimiento al tener piezas de software claramente diferenciadas. En el contexto de DDD, se distinguen tres capas fundamentales: la capa de dominio, la capa de aplicación y la capa de infraestructura (Autentia S.L., 2023).

Figura 7: Representación Arquitectura en Capas



Fuente: Arquitectura en capas tomado de (Autentia S.L., 2023)

2.2.9 Capa de dominio

La capa de dominio constituye el núcleo o esencia de la aplicación, encargándose de traducir las principales necesidades del negocio en código. Para lograr esto, se emplean los conceptos de entidades, objetos de valor y agregados, como se han definido previamente. En esta capa, podemos diferenciar entre el modelo y los servicios de dominio. El modelo engloba aquellos objetos esenciales para el núcleo del dominio, representando de manera única aspectos del negocio, como, por ejemplo, las entidades. Cuando la lógica de negocio resulta compleja y no puede ser expresada en el modelo, surgen los servicios de dominio, que representan servicios con lógica específica para un dominio en particular (Autentia S.L., 2023).

2.2.10 Capa de aplicación

La capa de aplicación asume la responsabilidad de crear y recuperar objetos de dominio para atender las solicitudes del usuario. Aunque carece de lógica de negocio, desempeña funciones de coordinación entre ellos de acuerdo con los requisitos y casos de uso de la aplicación. En otras palabras, alberga los servicios que establecen la conexión entre la capa de dominio y el entorno externo (Autentia S.L., 2023).

2.2.11 Capa de Infraestructura

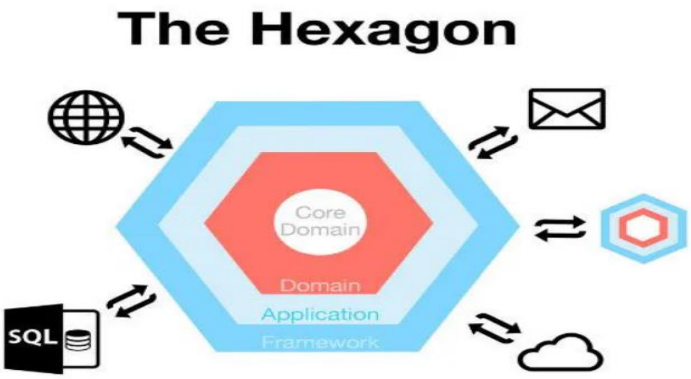
La capa de infraestructura suministra capacidades técnicas a las capas superiores o internas, como la posibilidad de persistir las entidades de dominio en bases de datos.

Es fundamental que esté totalmente desacoplada de la capa de dominio. Esto asegura que, por ejemplo, si actualmente utilizamos una base de datos relacional para la persistencia de datos y en el futuro deseamos cambiar a otro motor de persistencia, dicho cambio no afectará a las capas superiores (Autentia S.L., 2023).

2.2.12 Arquitectura Hexagonal

Según (Vieira, 2022) La arquitectura hexagonal, también conocida como la arquitectura de puertos y adaptadores, es un enfoque de diseño de software que busca separar las preocupaciones y minimizar las dependencias entre las diferentes capas de un sistema. Fue propuesta por Alistair Cockburn y se centra en la idea de organizar el código en hexágonos concéntricos para lograr un diseño modular y flexible.

Figura 8: Representación Arquitectura Hexagonal



Fuente: Arquitectura hexagonal. Tomado de (Fidao, 2023)

En esta arquitectura, el núcleo del sistema está representado por un hexágono central *(No está limitado a 6 capas por el número de lados)* que contiene la lógica de negocio o aplicación. Este hexágono no está directamente acoplado a ninguna tecnología externa, como bases de datos, interfaces de usuario o servicios web.

Alrededor del núcleo, se encuentran hexágonos externos que representan los diferentes puertos de entrada y salida del sistema. Estos puertos pueden incluir interfaces de usuario, servicios web, bases de datos, entre otros. Los adaptadores actúan como puentes entre el núcleo y los puertos, facilitando la comunicación.

La ventaja clave de la arquitectura hexagonal es su capacidad para aislar el núcleo de la aplicación de las implementaciones concretas de los componentes externos. Esto facilita la prueba unitaria, el mantenimiento y la evolución independiente de

las distintas partes del sistema. Además, permite cambiar fácilmente las tecnologías subyacentes sin afectar la lógica de negocio central.

2.2.13 Bounded context (Contexto Delimitado)

(Evans, 2003) Introduce el concepto de "contexto delimitado" para definir límites claros en los cuales un determinado término o concepto tiene un significado específico. Esto ayuda a evitar malentendidos y conflictos en la interpretación de conceptos en diferentes partes del sistema.

Este representa un área donde el dominio de un problema de negocio está completamente definido utilizando un lenguaje común, conocido como "ubiquitous language" (Autentia S.L., 2023).

En cualquier negocio, existen reglas y sistemas existentes que deben integrarse en la solución. Cada departamento de una empresa puede tener sus propios objetivos y métricas, así como términos específicos con significados particulares dentro de su contexto. El lenguaje común o "ubiquitous language" se utiliza para definir el dominio dentro de cada contexto o subdominio (Autentia S.L., 2023).

Por ejemplo, en un negocio donde existen dos departamentos: uno de ventas y otro de soporte, el termino cliente tiene un concepto diferente en cada uno de estos, para ventas el cliente realiza compras, pagos, pedidos y para soporte el cliente tiene problemas, tickets relacionados con los productos que tiene el negocio (Autentia S.L., 2023).

Es esencial emplear un lenguaje común en cada contexto para distinguir los términos específicos de cada departamento. Este lenguaje debe ser claro y compartido por todo el equipo, incluyendo desarrolladores, expertos de dominio y usuarios finales. La evolución de este lenguaje se logra con la participación continua del equipo, no se define en una única reunión. Cada "bounded context" normalmente representa un subdominio, separando diferentes áreas de negocio (Autentia S.L., 2023).

2.2.14 Command-Query Responsibility Segregation (CQRS)

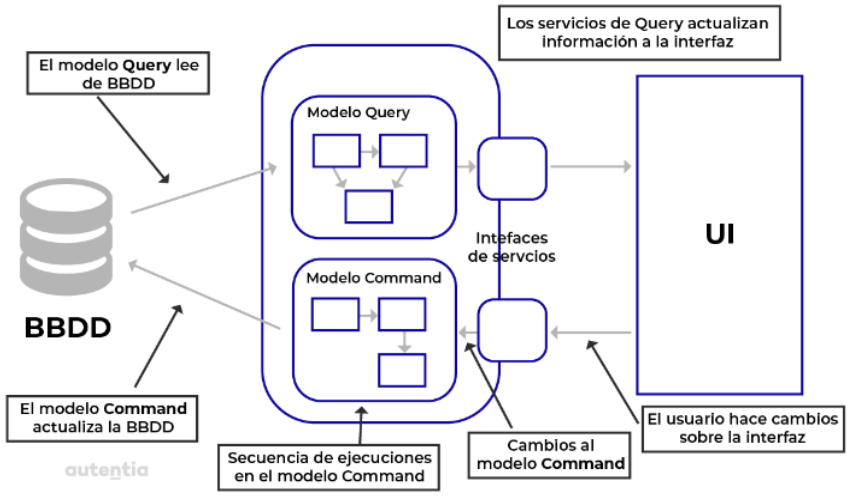
CQRS es un modelo arquitectónico que se fundamenta en la distinción entre las operaciones de lectura y las de escritura, denominadas como modelos Query y Command respectivamente. En términos conceptuales, esto implica la separación de ambos modelos de datos para que operen de manera independiente. El propósito

principal es mejorar la simplicidad en la escalabilidad, lectura y escritura, así como fortalecer la seguridad de ambas partes de forma individual. A pesar de estos beneficios, es importante destacar que introduce una complejidad adicional en nuestro sistema (Betts et al., 2012).

Los **Commands** Se alinea con el modelo de escritura encargado de almacenar la información proveniente de la capa de presentación. Debería ser la única vía para modificar el estado del sistema y podría abarcar reglas de validación aplicadas a los datos (Autentia S.L., 2023).

Los **Querys** corresponde con el modelo de lectura de datos, el cual proporciona un resultado sin modificar el estado del sistema. Su función principal es transformar los datos para presentarlos en la interfaz de usuario según lo requerido (Autentia S.L., 2023).

Figura 9: Representación CQRS



Fuente: Diagrama cqrs tomado de (Autentia S.L., 2023)

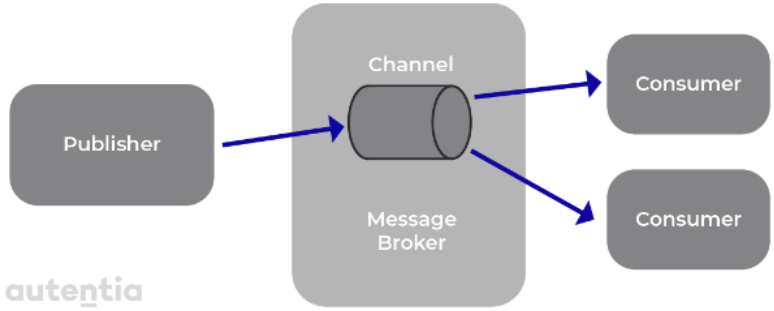
La ventaja principal proporcionada por esta distinción o segregación es la independencia entre los dos modelos, lo que permite escalar cada uno según nuestras necesidades. Incluso es posible ejecutarlos en máquinas diferentes. Al tratarse de dos "entidades" claramente diferenciadas, tenemos la capacidad de aplicar niveles de seguridad de manera separada y optimizar el tipo de tarea, ya sea consulta o persistencia en la base de datos (Autentia S.L., 2023).

2.2.15 Event-driven architecture (Arquitectura orientada a eventos)

Según (Green, 2022) La arquitectura orientada a eventos (EDA) es un patrón que fomenta el uso de eventos para la comunicación entre componentes independientes,

siendo común en aplicaciones basadas en microservicios. Un evento, en este contexto, representa un cambio asíncrono de estado o una actualización, como agregar un artículo al carrito en un sitio web de comercio electrónico. Estos eventos pueden contener información de estado, como los detalles de un artículo comprado, su precio y dirección de entrega, o ser identificadores, como una notificación de envío de un pedido.

Figura 10: Elementos Arquitectura basa en eventos.



Fuente: Event Driven architecture tomado de (Autentia S.L., 2023)

Las arquitecturas basadas en eventos constan de tres elementos clave:
los publicadores de eventos, los brokers de mensajes (routers) de eventos y los consumidores de eventos. Un publicador emite un evento al broker de mensajes, el cual filtra y envía los eventos a los consumidores registrados para ese tipo específico de evento. La relación entre el servicio que publica eventos y el servicio que los consume es desacoplada, permitiendo escalar, actualizar e implementar ambos de manera independiente. A través del canal (channel), los eventos se transmiten desde el publicador a los consumidores suscritos a ese canal.

2.2.16 Eventos de dominio

Un evento de dominio transporta información sobre un suceso dentro del dominio y tiene como objetivo comunicar este acontecimiento a otras partes de este. La reacción a dicho evento por otras partes del dominio puede variar. Cada evento de dominio captura información proveniente de su origen, y si se planea utilizar los registros como auditoría, es crucial que estos datos de origen sean inmutables. En otras palabras, una vez que se ha creado el objeto de evento, los datos de origen no pueden ser modificados. Sin embargo, existe otro tipo de datos relacionados con el evento que registra las acciones realizadas por el sistema en respuesta a dicho evento, conocidos como datos de procesamiento (Autentia S.L., 2023).

Los datos de un evento de dominio se dividen en dos categorías: datos de origen inmutables que describen la naturaleza del evento, y datos de procesamiento mutables que registran las acciones ejecutadas por el sistema en respuesta al evento. Aunque los datos de origen son inalterables, puede surgir la necesidad de realizar cambios, generalmente debido a que el evento original fue incorrecto. Para abordar esto, se emplea otro tipo de evento llamado evento retroactivo, utilizado para corregir las consecuencias de eventos erróneos anteriores (Autentia S.L., 2023).

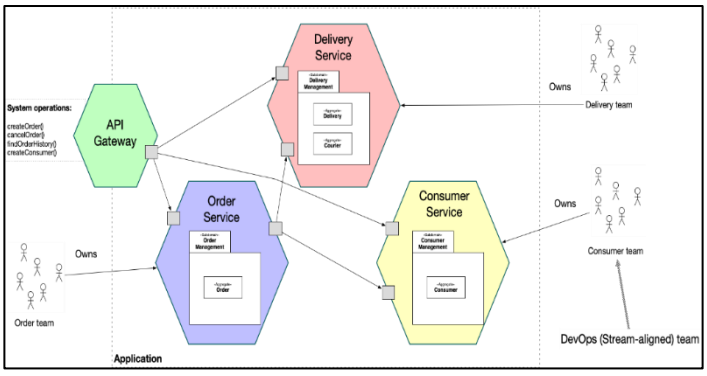
2.2.17 Config service

Con el Config Server, tienes un lugar centralizado para gestionar propiedades externas para aplicaciones en todos los entornos. Los conceptos en el cliente y el servidor se mapean de manera idéntica a las abstracciones Environment y PropertySource (Vieira, 2022).

2.2.18 Microservicios

La arquitectura de microservicios, también conocida como MSA, consiste en desarrollar una aplicación mediante un conjunto de pequeños servicios. Cada uno de estos servicios opera en su propio proceso y se comunica a través de mecanismos livianos, generalmente mediante una API de recursos HTTP. Cada microservicio asume la implementación completa de una funcionalidad de negocio y puede ser desplegado de manera autónoma, permitiendo la utilización de diferentes lenguajes de programación y tecnologías de almacenamiento de datos. Esta arquitectura descompone los modelos de dominio de negocio en contextos más reducidos, coherentes y bien definidos, implementados por los servicios individuales (Pacheco, 2018).

Figura 11: Representación Microservicios



Fuente: Diseño de microservicios tomado de (Richardson, 2023)

2.3 Conceptual

2.3.1 Arquitectura de software

Según (Pressman, 2010)La arquitectura de software abarca las estructuras del sistema, los componentes o módulos del software, las propiedades externamente visibles de esos componentes y las relaciones entre ellos**.**

2.3.2 Hibernate

Según (Tudose et al., 2023) Hibernate es un framework de mapeo objeto-relacional para Java que facilita la interacción entre aplicaciones Java y bases de datos relacionales. Permite a los desarrolladores trabajar con objetos Java en lugar de consultas SQL directas, gestionando el mapeo de objetos a tablas de base de datos, consultas mediante el lenguaje HQL, transacciones y caching. Con características como el mapeo de asociaciones y una gestión eficiente de la capa de persistencia, Hibernate simplifica el desarrollo de aplicaciones empresariales que requieren una conexión robusta entre la lógica de la aplicación y la persistencia de datos en una base de datos relacional.

2.3.3 Patrón Criteria

Según (Evans, 2003) es un patrón de diseño, mediante el cual, se permite filtrar una colección de objetos bajo diversos criterios, encadenándolos de una manera desacoplada por medio de operaciones lógicas. Este patrón se utiliza en escenarios específicos, donde la obtención de uno o más entidades depende de reglas de negocio.

2.3.4 Principio de Responsabilidad Única

Según (Iglberger, n.d.) El principio de responsabilidad única, también conocido como single responsibility, postula que un componente de software debe tener únicamente un motivo para ser modificado. La responsabilidad, en este contexto, se refiere a ese motivo de cambio específico.

2.3.5 Principio de Abierto/Cerrado

Según (Iglberger, n.d.) El principio Open/Closed (OCP) establece que los módulos de software deben ser diseñados de manera que sean abiertos para su extensión, permitiendo la incorporación de nuevos comportamientos conforme evolucionan los requisitos de la aplicación. Simultáneamente, deben ser cerrados para su modificación, lo cual implica que un módulo con una interfaz estable y bien

definida no debería requerir cambios en su código original al extender su funcionalidad. En otras palabras, se busca la capacidad de ampliar el comportamiento sin afectar el código existente, asegurando una estructura flexible y resistente a modificaciones innecesarias.

2.3.6 Principio de sustitución de Liskov

Según (Iglberger, n.d.) El principio de sustitución de Liskov establece que los objetos de un programa deben ser intercambiables por instancias de sus subtipos sin afectar el correcto funcionamiento del programa. En esencia, si utilizamos una clase en alguna parte de nuestro código y luego extendemos esa clase, deberíamos poder emplear cualquiera de las clases derivadas sin que el programa pierda su validez. Este principio nos ayuda a garantizar que la extensión de una clase no altere el comportamiento de la clase padre.

2.3.7 Principio de segregación de interfaces

Según (Iglberger, n.d.) El principio de segregación de interfaces sostiene que es preferible tener múltiples interfaces específicas para los clientes en lugar de una interfaz general para todos los propósitos. Cuando los clientes se ven obligados a utilizar partes de una interfaz que no necesitan en su totalidad, están expuestos a cambios innecesarios en esa interfaz. Esto conduce, en última instancia, a un acoplamiento innecesario entre los clientes y las interfaces, generando una mayor dependencia de lo requerido.

2.3.8 Principio de inversión de dependencias

Según (Iglberger, n.d.)El principio de segregación de interfaces propone que es más beneficioso contar con múltiples interfaces específicas para los clientes en lugar de una interfaz general que abarque todos los propósitos. Cuando los clientes se ven obligados a utilizar partes de una interfaz que no requieren completamente, quedan expuestos a cambios superfluos en esa interfaz. Este escenario, en última instancia, resulta en un acoplamiento innecesario entre los clientes y las interfaces, creando una dependencia mayor de lo necesario.

2.3.9 RabbitMQ

Según (Dobbelaere & Esmaili, 2017) RabbitMQ funciona como un broker de mensajería, actuando como intermediario en el proceso de intercambio de mensajes. Ofrece a las aplicaciones una plataforma compartida para el envío y recepción de

mensajes, garantizando un entorno seguro para que los mensajes permanezcan hasta su recepción.

2.3.10 ReactJS

Según (Frensia Tanaga Anaclaudia et al., 2023) React.js, o simplemente React, es una biblioteca de JavaScript de código abierto desarrollada por Facebook. Se utiliza para construir interfaces de usuario (UI) interactivas y eficientes para aplicaciones web. React se centra en la creación de componentes reutilizables que representan partes específicas de la interfaz de usuario.

2.3.11 UUID

Según (Leach et al., 2023) UUID, que significa "Identificador Único Universal" en inglés, es un estándar para identificadores únicos que se utiliza en el ámbito de la informática. Un UUID es una cadena de caracteres alfanuméricos que se asigna a un recurso o entidad de manera única en el tiempo y el espacio. La intención detrás de los UUID es proporcionar identificadores únicos incluso en entornos distribuidos y sin necesidad de coordinación centralizada.

2.3.12 Kubernetes

Kubernetes es una plataforma de código abierto que automatiza el despliegue, escalado y gestión de aplicaciones en contenedores, permitiendo a los usuarios agrupar contenedores en "pods" y gestionarlos eficientemente en un clúster de servidores, con características como autosanación, autoscalabilidad y gestión declarativa, lo que simplifica el proceso de implementación y administración de aplicaciones a gran escala en entornos de nube y locales. (Kubernetes, 2021)

2.3.13 Boleto de Viaje

Un boleto de viaje es un documento que confirma la reserva y pago de un servicio de transporte, como avión, tren o autobús. Contiene detalles esenciales como la fecha, hora y ruta del viaje, y es necesario para acceder al medio de transporte. Con el avance tecnológico, los boletos electrónicos son cada vez más comunes, almacenados en dispositivos móviles o enviados por correo electrónico.

2.3.14 Cooperativa de Transportes

Organización empresarial conformada por un grupo de individuos que se asocian para proporcionar servicios de transporte de manera conjunta.

2.4 Legal

2.4.1 Ley Orgánica de Transporte Terrestre Transito y Seguridad Vial

De acuerdo con el artículo 46 de la Ley Orgánica de Transporte Terrestre Transito y Seguridad Vial “El transporte terrestre automotor es un servicio público esencial y una actividad económica estratégica del Estado, que consiste en la movilización libre y segura de personas o de bienes de un lugar a otro, haciendo uso del sistema vial nacional, terminales terrestres y centros de transferencia de pasajeros y carga en el territorio ecuatoriano. Su organización es un elemento fundamental contra la informalidad, mejorar la competitividad y lograr el desarrollo productivo, económico y social del país, interconectado con la red vial internacional.”

2.4.2 Ley Orgánica de Protección de datos

De acuerdo con el artículo 66 numeral 19 de la Constitución de la República reconoce y garantiza a las personas: "19. El derecho a la protección de datos carácter personal, que incluye el acceso y la decisión sobre información y datos de este carácter, así como su correspondiente protección. La recolección, archivo, procesamiento, distribución o difusión de estos datos personales requerirán la autorización del titular o el mandato de ley";

2.4.3 De la Institución.

De acuerdo con el estatuto de la cooperativa de transportes “san pedrito” no existe impedimento para poder realizar la implementación de un sistema informático para la gestión y venta de boletos.

CAPÍTULO III

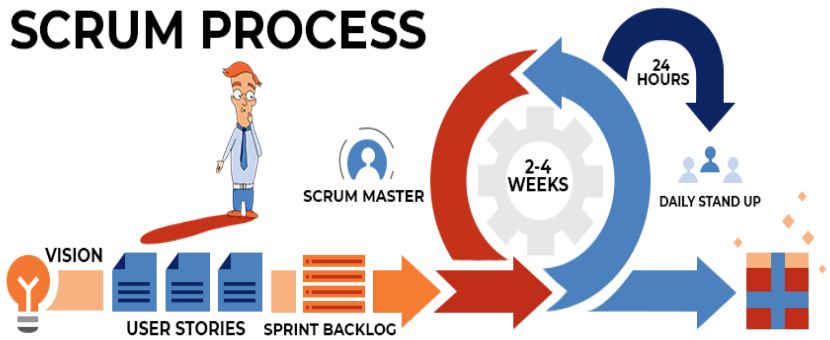
METODOLOGÍA

3.1 Metodología de Desarrollo de Software (Scrum)

Según (Schwaber & Sutherland, 2020) “Scrum es un marco de trabajo ágil para el desarrollo de software que se centra en la entrega iterativa e incremental de productos. Surgió en la década de 1990 y se ha convertido en uno de los enfoques más populares para gestionar proyectos complejos y adaptarse a cambios rápidos en los requisitos del cliente. Scrum se basa en principios de transparencia, inspección y adaptación, y se organiza en torno a roles definidos, eventos y artefactos”.

3.1.1 Proceso de Scrum

Figura 12: Proceso Scrum



Fuente: Scrum methodology: Understanding the process of agile software development tomato de (“Purpose of the Scrum Guide,” 2020)

De acuerdo con (“Purpose of the Scrum Guide,” 2020) el proceso de trabajo en Scrum se organiza en iteraciones llamadas "sprints" y contiene lo siguiente:

a) **Planificación del Sprint:**
   - El equipo y el propietario del producto se reúnen para seleccionar las tareas que se abordarán durante el próximo sprint.
   - Se define el objetivo del sprint.

b) Sprint:

- Durante el sprint, que suele durar de 2 a 4 semanas, el equipo trabaja en las tareas seleccionadas.
- Se llevan a cabo reuniones diarias de seguimiento, llamadas "Daily Scrum", para sincronizar el progreso.

c) Revisión del Sprint:

- Al final del sprint, se lleva a cabo una reunión de revisión en la que el equipo presenta el trabajo completado.
- Se recopilan comentarios y se pueden realizar ajustes para futuros sprints.

d) Retrospectiva del Sprint:

- Después de la revisión, el equipo tiene una retrospectiva para analizar lo que salió bien, lo que salió mal y cómo mejorar en el próximo sprint.

e) Reinicio del Ciclo:

- Se inicia un nuevo sprint y el ciclo se repite.

3.1.2 Roles de Scrum:

- **Product Owner:** "Es responsable de definir y priorizar el backlog del producto, representando los intereses del cliente y asegurándose de que el equipo desarrolle funcionalidades de alto valor" ("Purpose of the Scrum Guide," 2020).
- **Scrum Master:** "Facilita el proceso Scrum, elimina obstáculos para el equipo y asegura que se sigan las prácticas y reglas de Scrum"("Purpose of the Scrum Guide," 2020).
- **Equipo de Desarrollo:** "El grupo de profesionales que realiza el trabajo para entregar un incremento de producto potencialmente entregable al final de cada sprint" ("Purpose of the Scrum Guide," 2020).

3.1.3 Eventos (Ceremonias):

- **Sprint Planning:** "Reunión al comienzo de cada sprint donde se seleccionan las tareas a realizar y se planifica cómo se abordarán" ("Purpose of the Scrum Guide," 2020).

- **Daily Scrum:** “Reunión diaria corta para que el equipo sincronice actividades y planifique el trabajo para las próximas 24 horas”(“Purpose of the Scrum Guide,” 2020).
- **Sprint Review:** “Reunión al final de cada sprint para revisar el trabajo completado y adaptar el backlog del producto en consecuencia” (“Purpose of the Scrum Guide,” 2020).
- **Sprint Retrospective:** “Sesión al final de cada sprint para reflexionar sobre el proceso y buscar mejoras continuas” (“Purpose of the Scrum Guide,” 2020).

3.1.4 Artefactos:

- **Backlog del Producto:** “Una lista priorizada de todas las características, funciones, mejoras y correcciones que constituyen el trabajo que debe realizarse”(“Purpose of the Scrum Guide,” 2020)
- **Backlog del Sprint:** “Un conjunto de elementos seleccionados del backlog del producto que el equipo se compromete a completar durante un sprint” (“Purpose of the Scrum Guide,” 2020).
- **Incremento:** “El producto funcional y potencialmente entregable al final de cada sprint” (“Purpose of the Scrum Guide,” 2020).

3.1.5 Historia de Usuario

Las historias de usuario son descripciones breves de funcionalidades en el desarrollo de software, enfocadas en las necesidades del usuario. Se expresan como "Como [usuario], quiero [realizar una acción] para [lograr un objetivo]" (Singh, 2019).

3.1.6 Scrum y Domain Driven Desing

Integrar Scrum con Domain-Driven Design (DDD) ofrece beneficios sinérgicos para el desarrollo de software. Scrum proporciona una estructura ágil, iterativa e incremental, mientras que DDD enfoca la atención en comprender y modelar el dominio del problema. Al combinarlos, se logra una colaboración más estrecha entre los desarrolladores y los expertos en el dominio, lo que resulta en un entendimiento profundo y continuo del dominio. Esto facilita la adaptación rápida a cambios, la entrega de funcionalidades valiosas y la mejora constante del modelo

del dominio a lo largo del tiempo, asegurando un desarrollo de software más alineado con las necesidades del negocio.

3.2 Técnicas e Instrumentos de Recopilación de Datos

3.2.1 Entrevista

Según (Pressman, 2010) “Una entrevista es un proceso clave para recopilar información valiosa y comprender las necesidades de los stakeholders, usuarios finales y otros expertos relevantes”. Durante estas interacciones, el equipo de desarrollo se reunió con las partes interesadas, como los directivos, contadora y los empleados encargados de la venta de boletos de la cooperativa de transportes “San Pedrito”.

Para llevar a cabo las entrevistas, se utilizó un guion que incluirá preguntas específicas diseñadas para obtener una comprensión más profunda de los requisitos y expectativas del software (Ver Anexo 5).

CAPÍTULO IV

INGENIERÍA DEL PROYECTO

4.1 Análisis y Planificación

4.1.1 Visión

Se desarrollo un sistema informático integral para la gestión y venta de boletos en la Cooperativa de Transportes “San Pedrito” aplicando los principios y prácticas de Domain Driven Design (DDD).

4.1.2 Scrum Team

- **Scrum Master:** Eduardo Bonilla, responsable de coordinar y facilitar el proceso Scrum.
- **Product Owner:** Melanin Ganan, se encargó de definir y priorizar los elementos del producto.
- **Developers Team :** Integrado por Eduardo Bonilla y Melanin Ganan, quienes llevaron a cabo la implementación y desarrollo del producto.

4.1.3 Stakeholders

- **Personal de Venta:** Empleados encargados de la venta de los boletos, la atención al cliente en las boleterías y controlar la asignación de viajes en las frecuencias diarias.
- **Contador:** Encargado de la gestión contable, registro de ingresos y gastos, así como la generación de informes financieros para garantizar la transparencia y el cumplimiento de las obligaciones fiscales.
- **Administradores de la Cooperativa:** Personas responsables de la gestión general de la cooperativa y supervisión del sistema.
- **Desarrolladores del Sistema:** El equipo de desarrollo encargado de diseñar, construir y mantener el sistema informático.

4.1.4 Historias de Usuario

A continuación, se presenta un resumen de las historias de usuarios el documento completo se encuentra en el Anexo 1

Tabla 1: Resumen historias de usuario

ID	Nombre	Descripción	Prioridad
XS-9	Gestionar el inicio de sesión.	Como usuario, quiero implementado un módulo eficiente para la gestión del inicio de sesión, asegurando la autenticación segura y autorización adecuada de los usuarios.	Alta
XS-14	Gestionar recuperación contraseña.	Como usuario, quiero restablecer mi contraseña de manera segura en caso de olvido.	Media
XS-19	Gestionar Socios.	Como contador, quiero ser capaz de gestionar la información de los socios.	Alta
XS-41	Gestión de Carrocerías.	Como Administrador, quiero realizar un seguimiento detallado de la configuración de asientos en cada vehículo. Esto incluye el número de asientos, su disposición y ubicación en la carrocería.	Alta
XS-20	Gestionar Vehículos.	Como Administrador, quiero gestionar la	Alta

		información esencial para cada unidad de transporte.	
XS-42	Gestión de Conductores.	Como Administrador, quiero una herramienta de control que me permita registrar de manera efectiva la información clave asociada a cada conductor en los buses.	Alta
XS-34	Gestión de Frecuencias.	Como Administrador, quiero que me permita organizar de los horarios de los vehículos, la herramienta debe ser capaz de manejar información detallada sobre las frecuencias de salida y llegada.	Alta
XS-36	Gestión de Rutas.	Como Administrador, quiero una herramienta que me permita gestionar las rutas aprobadas por el ant.	Alta
XS-21	Gestión plan de cuentas contables.	Como contador, quiero crear, editar o eliminar las cuentas contables del sistema.	Media

XS-22	Gestión Asientos contables.	Como Contador, quiero un sistema que permita la gestión eficiente de asientos contables, Para mantener un seguimiento preciso de las transacciones financieras y generar informes financieros fiables.	Media
XS-23	Reporte de Mayores.	Como Contador, quiero un sistema que permita la gestión eficiente de asientos contables, para mantener un seguimiento preciso de las transacciones financieras y generar informes financieros fiables.	Media
XS-45	Implementar Facturación Electrónica.	Como Contador, quiero que el sistema emita la facturación electrónica de acuerdo con la normativa vigente del SRI.	Media
XS-32	Venta de boletos.	Como oficinista quiero realizar la venta de los boletos de una manera eficiente.	Alta

Realizado por: Eduardo G & Melanin G

4.1.5 Backlog

Tabla 2: Backlog

Código	Incidencia	Estado
XS-9	Gestionar el inicio de sesión.	Finalizada
XS-14	Gestionar recuperación contraseña.	Finalizada
XS-19	Gestionar Socios.	Finalizada
XS-41	Gestión de Carrocerías.	Finalizada
XS-20	Gestionar Vehículos.	Finalizada
XS-42	Gestión de Conductores.	Finalizada
XS-34	Gestión de Frecuencias.	Finalizada
XS-36	Gestión de Rutas.	Finalizada
XS-21	Gestión plan de cuentas contables.	Finalizada
XS-22	Gestión Asientos contables.	Finalizada
XS-23	Reporte de Mayores.	Finalizada
XS-45	Implementar Facturación Electrónica.	Finalizada
XS-32	Venta de boletos.	Finalizada

Realizado por: Eduardo G & Melanin G

4.1.6 Plan de pruebas

4.1.6.1 Objetivo de las pruebas

- Validar la correcta implementación de las reglas de negocio definidas en el dominio.
- Asegurar la integración adecuada entre los diferentes componentes del sistema.
- Verificar que las historias de usuario cumplan con los criterios de aceptación establecidos.

4.1.6.2 Alcance de las pruebas

- **Pruebas unitarias** se aplicó a los casos de uso.
- **Pruebas de integración** para validar la comunicación entre las capas del sistema.
- **Pruebas de integración** con elementos externos al sistema.
- **Pruebas de aceptación** del usuario para validar las historias implementadas.

4.1.6.3 Criterios de Aceptación

- Todas las pruebas unitarias deben pasar satisfactoriamente.
- Todas las historias de usuario deben pasar las pruebas de aceptación.
- Las pruebas de integración deben demostrar la comunicación correcta entre las capas.
- No se deben incluir librerías sin soporte y desactualizadas.

4.1.6.4 Procedimientos de las pruebas

- Las pruebas unitarias, integración y aceptación deberán pasar a nivel de desarrollo.
- Las pruebas unitarias, integración, aceptación análisis de código estático se analizar una vez que se envíen los cambios al repositorio, en el caso de no pasar no se aprobara el merge request.

4.2 Diseño

4.2.1 Arquitectura de Microservicios

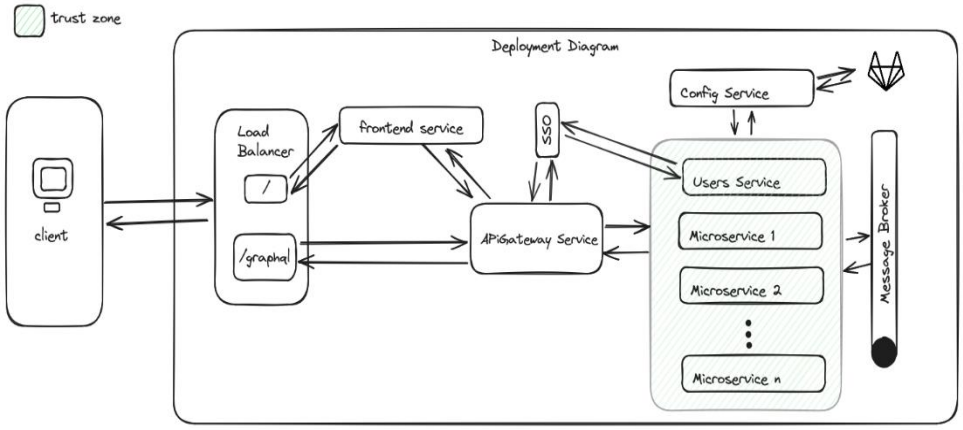
La arquitectura de microservicios orientada a eventos es un enfoque arquitectónico que se centra en la comunicación asíncrona y la interacción basada en eventos entre los distintos componentes de un sistema. En contraste con los modelos tradicionales basados en solicitudes y respuestas, donde los componentes se comunican de manera síncrona, la arquitectura orientada a eventos permite una mayor flexibilidad, escalabilidad y desacoplamiento entre los servicios.

En la Figura 13 se observa la arquitectura implementada en el proyecto. Esta arquitectura cuenta con un balanceador de carga que dirige el tráfico tanto al servicio de frontend a través del endpoint "/" como al servicio de backend mediante el endpoint "/graphql". El servicio backend se comunica con el API Gateway, implementado con NestJS, cuya función es centralizar la comunicación con los microservicios, además de gestionar la autenticación y autorización.

Los microservicios están ubicados en una zona segura, donde se asume que las solicitudes entrantes están debidamente autenticadas y autorizadas. Además, estos microservicios se comunican con el servicio de configuración centralizada (config-service) desarrollado con Spring Config Service, que obtiene la configuración alojada en GitLab.

Es importante mencionar que cada microservicio genera eventos de dominio, los cuales se envían y reciben a través de una cola de mensajería implementada en RabbitMQ.

Figura 13: Arquitectura de Microservicios



Realizado por: Eduardo G & Melanin G

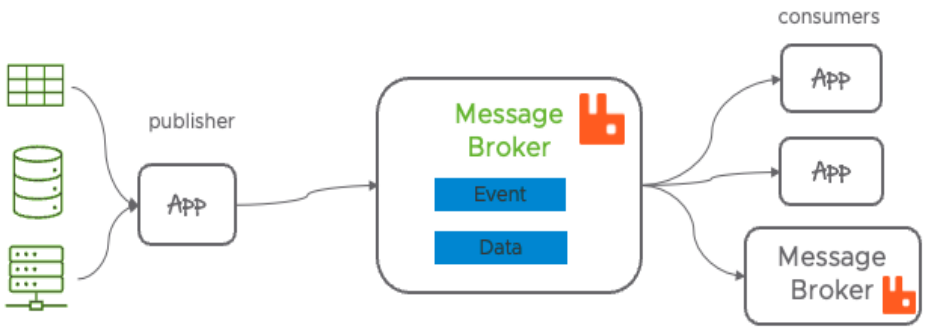
4.2.2 Definición de Microservicios

- **Servicio de Usuarios.-** Este servicio se encarga de gestionar la información relacionada con los usuarios del sistema. Incluye funcionalidades como registro de usuarios, autenticación, gestión de roles y permisos.
- **Servicio de Contabilidad.-** Este servicio se dedica a la gestión contable de la cooperativa.
- **Servicio de gestión de buses y boletos.-** Este servicio está orientado a la gestión operativa de los autobuses de la cooperativa. Incluye funciones como programación de rutas, asignación de conductores y buses a rutas específicas, la venta de boletos de transporte.
- **Servicio de facturación electrónica.-** Este servicio se encarga de la emisión de facturas electrónicas por los servicios de transporte prestados. Incluye funciones como la generación y envío de facturas electrónicas a los clientes, además cumple con la normativa establecida por el Servicio de Rentas Internas de Ecuador.

4.2.3 Comunicación entre microservicios

La comunicación entre los distintos servicios se facilita mediante RabbitMQ, donde los servicios actúan como productores de eventos al publicar mensajes relevantes en colas específicas, y otros servicios, actuando como consumidores, se suscriben a estas colas para recibir y procesar los mensajes cuando estén disponibles, permitiendo una comunicación asíncrona y confiable que mejora la flexibilidad, escalabilidad y robustez del sistema.

Figura 14: Comunicación entre microservicios



Fuente: Comunicación de microservicios tomado de (Green, 2022)

A continuación, se muestra el código fuente para implementar y configurar los publicadores y consumidores en springboot con rabbitmq.

RabbitMqDomainEventsConsumer.java

```
package
com.devsoftec.xyphire.auth.shared.infrastructure.bus.event.ra
bbitmq;

import com.devsoftec.xyphire.auth.shared.domain.Service;
import com.devsoftec.xyphire.auth.shared.domain.Utils;
import
com.devsoftec.xyphire.auth.shared.domain.bus.event.DomainEven
t;
import
com.devsoftec.xyphire.auth.shared.infrastructure.bus.event.Do
mainEventJsonDeserializer;
import
com.devsoftec.xyphire.auth.shared.infrastructure.bus.event.Do
mainEventSubscribersInformation;
import org.springframework.amqp.core.Message;
import org.springframework.amqp.core.MessageBuilder;
import
org.springframework.amqp.core.MessagePropertiesBuilder;
import
org.springframework.amqp.rabbit.annotation.RabbitListener;
import
org.springframework.amqp.rabbit.listener.AbstractMessageListe
nerContainer;
import
org.springframework.amqp.rabbit.listener.RabbitListenerEndpoi
ntRegistry;
import org.springframework.context.ApplicationContext;

import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;
import java.util.HashMap;
import java.util.Map;

@Service
public final class RabbitMqDomainEventsConsumer {
    private final
String                      CONSUMER_NAME          =
"domain_events_consumer";
    private final
int                         MAX_RETRIES            = 2;
    private final DomainEventJsonDeserializer deserializer;
```

```
    private final ApplicationContext          context;
    private final RabbitMqPublisher           publisher;
    private final HashMap<String,
Object>     domainEventSubscribers = new HashMap<>();
    RabbitListenerEndpointRegistry registry;
    private DomainEventSubscribersInformation information;

    public RabbitMqDomainEventsConsumer(
            RabbitListenerEndpointRegistry registry,
            DomainEventSubscribersInformation information,
            DomainEventJsonDeserializer deserializer,
            ApplicationContext context,
            RabbitMqPublisher publisher
    ) {
        this.registry     = registry;
        this.information  = information;
        this.deserializer = deserializer;
        this.context      = context;
        this.publisher    = publisher;
    }

    public void consume() {
        AbstractMessageListenerContainer container =
(AbstractMessageListenerContainer)
registry.getListenerContainer(
                CONSUMER_NAME
        );

        container.addQueueNames(information.rabbitMqFormatted
Names());

        container.start();
    }

    @RabbitListener(id = CONSUMER_NAME, autoStartup =
"false")
    public void consumer(Message message) throws Exception {
        String         serializedMessage = new
String(message.getBody());
        DomainEvent domainEvent       =
deserializer.deserialize(serializedMessage);

        String queue =
message.getMessageProperties().getConsumerQueue();

        Object subscriber =
domainEventSubscribers.containsKey(queue)
```

```
                ? domainEventSubscribers.get(queue)
                : subscriberFor(queue);

        Method subscriberOnMethod =
subscriber.getClass().getMethod("on",
domainEvent.getClass());

        try {
            subscriberOnMethod.invoke(subscriber,
domainEvent);
        } catch (IllegalAccessException |
IllegalArgumentException | InvocationTargetException error) {
            throw new Exception(String.format(
                    "The subscriber <%s> should implement a
method `on` listening the domain event <%s>",
                    queue,
                    domainEvent.eventName()
            ));
        } catch (Exception error) {
            handleConsumptionError(message, queue);
        }
    }

    private void handleConsumptionError(Message message,
String queue) {
        if (hasBeenRedeliveredTooMuch(message)) {
            sendToDeadLetter(message, queue);
        } else {
            sendToRetry(message, queue);
        }
    }

    private void sendToRetry(Message message, String queue) {
        sendMessageTo(RabbitMqExchangeNameFormatter.retry("do
main_events"), message, queue);
    }

    private void sendToDeadLetter(Message message, String
queue) {
        sendMessageTo(RabbitMqExchangeNameFormatter.deadLette
r("domain_events"), message, queue);
    }

    private void sendMessageTo(String exchange, Message
message, String queue) {
        Map<String, Object> headers =
message.getMessageProperties().getHeaders();
```

```
        headers.put("redelivery_count", (int)
headers.getOrDefault("redelivery_count", 0) + 1);

        MessageBuilder.fromMessage(message).andProperties(
                MessagePropertiesBuilder.newInstance()
                        .setContentEncoding("utf-8")
                        .setContentType("application/json")
                        .copyHeaders(headers)
                        .build());

        publisher.publish(message, exchange, queue);
    }

    private boolean hasBeenRedeliveredTooMuch(Message
message) {
        return (int)
message.getMessageProperties().getHeaders().getOrDefault("red
elivery_count", 0) >= MAX_RETRIES;
    }

    public void
withSubscribersInformation(DomainEventSubscribersInformation
information) {
        this.information = information;
    }

    private Object subscriberFor(String queue) throws
Exception {
        String[] queueParts     = queue.split("\\.");
        String   subscriberName =
Utils.toCamelFirstLower(queueParts[queueParts.length - 1]);

        try {
            Object subscriber =
context.getBean(subscriberName);
            domainEventSubscribers.put(queue, subscriber);

            return subscriber;
        } catch (Exception e) {
            throw new Exception(String.format("There are not
registered subscribers for <%s> queue", queue));
        }
    }
}
```

Este código implementa un consumidor de eventos de dominio basado en RabbitMQ en Java utilizando Spring AMQP. El consumidor está diseñado para manejar mensajes de eventos de dominio que son publicados en colas específicas en RabbitMQ. Utiliza un deserializador para convertir los mensajes JSON en objetos de eventos de dominio y luego los enruta a los suscriptores correspondientes basados en el nombre de la cola. Además, el consumidor maneja la lógica de reintento y envío a la cola de dead-letter en caso de errores durante el procesamiento del evento. También proporciona métodos para iniciar el consumo de eventos y configurar la información de los suscriptores.

RabbitMqEventBusConfiguration.java

```
package
com.devsoftec.xyphire.auth.shared.infrastructure.bus.event.ra
bbitmq;

import
com.devsoftec.xyphire.auth.shared.infrastructure.bus.event.Do
mainEventSubscribersInformation;
import
com.devsoftec.xyphire.auth.shared.infrastructure.bus.event.Do
mainEventsInformation;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.amqp.core.*;
import
org.springframework.amqp.rabbit.connection.CachingConnectionF
actory;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import java.util.ArrayList;
import java.util.Collection;
import java.util.HashMap;
import java.util.List;
import java.util.stream.Collectors;


@Configuration
public class RabbitMqEventBusConfiguration {
    private final DomainEventSubscribersInformation
domainEventSubscribersInformation;
```

```
    private final DomainEventsInformation
domainEventsInformation;
    private final Logger logger =
LoggerFactory.getLogger(RabbitMqEventBusConfiguration.class);
    //RabbitMq Variables
    @Value("${rabbitmq.host}")
    private String rabbitmqHost;
    @Value("${rabbitmq.port}")
    private String rabbitmqPort;
    @Value("${rabbitmq.login}")
    private String rabbitmqLogin;
    @Value("${rabbitmq.password}")
    private String rabbitmqPassword;
    @Value("${rabbitmq.vhost}")
    private String rabbitmqVhost;
    @Value("${rabbitmq.exchange}")
    private String exchangeName;


    public RabbitMqEventBusConfiguration(
        DomainEventSubscribersInformation
domainEventSubscribersInformation,
        DomainEventsInformation domainEventsInformation
    ) {
        this.domainEventSubscribersInformation =
domainEventSubscribersInformation;
        this.domainEventsInformation =
domainEventsInformation;
    }

    @Bean
    public CachingConnectionFactory connection() {
        logger.info("RabbitMQ connection created");
        CachingConnectionFactory factory = new
CachingConnectionFactory();

        factory.setHost(rabbitmqHost);
        factory.setPort(Integer.parseInt(rabbitmqPort));
        factory.setUsername(rabbitmqLogin);
        factory.setPassword(rabbitmqPassword);
        factory.setVirtualHost(rabbitmqVhost);

        return factory;
    }

    @Bean
    public Declarables declaration() {
```

```
        String retryExchangeName =
RabbitMqExchangeNameFormatter.retry(exchangeName);
        String deadLetterExchangeName =
RabbitMqExchangeNameFormatter.deadLetter(exchangeName);

        TopicExchange domainEventsExchange = new
TopicExchange(exchangeName, true, false);
        TopicExchange retryDomainEventsExchange = new
TopicExchange(retryExchangeName, true, false);
        TopicExchange deadLetterDomainEventsExchange = new
TopicExchange(deadLetterExchangeName, true, false);
        List<Declarable> declarables = new ArrayList<>();
        declarables.add(domainEventsExchange);
        declarables.add(retryDomainEventsExchange);
        declarables.add(deadLetterDomainEventsExchange);

        Collection<Declarable> queuesAndBindings =
declareQueuesAndBindings(
            domainEventsExchange,
            retryDomainEventsExchange,
            deadLetterDomainEventsExchange
        ).stream().flatMap(Collection::stream).collect(Collec
tors.toList());

        declarables.addAll(queuesAndBindings);

        return new Declarables(declarables);
    }

    private Collection<Collection<Declarable>>
declareQueuesAndBindings(
        TopicExchange domainEventsExchange,
        TopicExchange retryDomainEventsExchange,
        TopicExchange deadLetterDomainEventsExchange
    ) {
        return
domainEventSubscribersInformation.all().stream().map(informat
ion -> {
            String queueName =
RabbitMqQueueNameFormatter.format(information);
            String retryQueueName =
RabbitMqQueueNameFormatter.formatRetry(information);
            String deadLetterQueueName =
RabbitMqQueueNameFormatter.formatDeadLetter(information);

            Queue queue =
QueueBuilder.durable(queueName).build();
```

```
            Queue retryQueue =
QueueBuilder.durable(retryQueueName).withArguments(
                retryQueueArguments(domainEventsExchange,
queueName)
            ).build();
            Queue deadLetterQueue =
QueueBuilder.durable(deadLetterQueueName).build();

            Binding fromExchangeSameQueueBinding =
BindingBuilder
                .bind(queue)
                .to(domainEventsExchange)
                .with(queueName);

            Binding fromRetryExchangeSameQueueBinding =
BindingBuilder
                .bind(retryQueue)
                .to(retryDomainEventsExchange)
                .with(queueName);

            Binding fromDeadLetterExchangeSameQueueBinding =
BindingBuilder
                .bind(deadLetterQueue)
                .to(deadLetterDomainEventsExchange)
                .with(queueName);

            List<Binding> fromExchangeDomainEventsBindings =
information.subscribedEvents().stream().map(
                domainEventClass -> {
                    String eventName =
domainEventsInformation.forClass(domainEventClass);
                    return BindingBuilder
                        .bind(queue)
                        .to(domainEventsExchange)
                        .with(eventName);
                }).collect(Collectors.toList());

            List<Declarable> queuesAndBindings = new
ArrayList<>();
            queuesAndBindings.add(queue);
            queuesAndBindings.add(fromExchangeSameQueueBindin
g);
            queuesAndBindings.addAll(fromExchangeDomainEvents
Bindings);

            queuesAndBindings.add(retryQueue);
```

```
            queuesAndBindings.add(fromRetryExchangeSameQueueB
inding);

            queuesAndBindings.add(deadLetterQueue);
            queuesAndBindings.add(fromDeadLetterExchangeSameQ
ueueBinding);

            return queuesAndBindings;
        }).collect(Collectors.toList());
    }

    private HashMap<String, Object>
retryQueueArguments(TopicExchange exchange, String
routingKey) {
        return new HashMap<String, Object>() {{
            put("x-dead-letter-exchange",
exchange.getName());
            put("x-dead-letter-routing-key", routingKey);
            put("x-message-ttl", 1000);
        }};
    }
}
```

Este código Java configura la conexión a RabbitMQ y declara los elementos necesarios, como intercambios y colas, para establecer un bus de eventos en una aplicación. Utiliza Spring AMQP para gestionar la comunicación asíncrona entre los componentes, permitiendo el enrutamiento y procesamiento eficiente de mensajes de eventos de dominio en colas específicas, incluyendo reintentos y dead-letter queues para manejar errores y fallos en el procesamiento de eventos.

RabbitMqPublisher.java

```
package
com.devsoftec.xyphire.auth.shared.infrastructure.bus.event.ra
bbitmq;


import com.devsoftec.xyphire.auth.shared.domain.Service;
import
com.devsoftec.xyphire.auth.shared.domain.bus.event.DomainEven
t;
import
com.devsoftec.xyphire.auth.shared.infrastructure.bus.event.Do
mainEventJsonSerializer;
import org.springframework.amqp.AmqpException;
import org.springframework.amqp.rabbit.core.RabbitTemplate;
import org.springframework.amqp.core.Message;
import
org.springframework.amqp.core.MessagePropertiesBuilder;

@Service
public final class RabbitMqPublisher {
    private final RabbitTemplate rabbitTemplate;

    public RabbitMqPublisher(RabbitTemplate rabbitTemplate) {
        this.rabbitTemplate = rabbitTemplate;
    }

    public void publish(DomainEvent domainEvent, String
exchangeName) throws AmqpException {
        String serializedDomainEvent =
DomainEventJsonSerializer.serialize(domainEvent);

        Message message = new Message(
                serializedDomainEvent.getBytes(),
                MessagePropertiesBuilder.newInstance()
                        .setContentEncoding("utf-8")
                        .setContentType("application/json")
                        .build()
        );

        rabbitTemplate.send(exchangeName,
domainEvent.eventName(), message);
    }

    public void publish(Message domainEvent, String
exchangeName, String routingKey) throws AmqpException {
```

```
        rabbitTemplate.send(exchangeName, routingKey,
domainEvent);
    }
}
```

Este código Java implementa un publicador para el bus de eventos en RabbitMQ utilizando Spring AMQP. El publicador se encarga de enviar mensajes de eventos de dominio a un intercambio específico en RabbitMQ. Utiliza un RabbitTemplate proporcionado por Spring para enviar los mensajes, y serializa los eventos de dominio en formato JSON antes de enviarlos. Además, proporciona métodos para publicar mensajes utilizando un intercambio y clave de enrutamiento específicos, lo que permite una mayor flexibilidad en la configuración de la comunicación entre los componentes del sistema.

RabbitMqEventBus.java

```java
package
com.devsoftec.xyphire.auth.shared.infrastructure.bus.event.ra
bbitmq;

import com.devsoftec.xyphire.auth.shared.domain.Service;
import
com.devsoftec.xyphire.auth.shared.domain.bus.event.DomainEven
t;
import
com.devsoftec.xyphire.auth.shared.domain.bus.event.EventBus;
import
com.devsoftec.xyphire.auth.shared.infrastructure.bus.event.ma
riadb.MariaDBEventBus;
import org.springframework.amqp.AmqpException;
import org.springframework.context.annotation.Primary;

import java.util.Collections;
import java.util.List;

@Service
@Primary
public class RabbitMqEventBus implements EventBus {
    private final RabbitMqPublisher publisher;
    private final MariaDBEventBus     failoverPublisher;
    private final String            exchangeName;
```

```
    public RabbitMqEventBus(RabbitMqPublisher publisher,
MariaDBEventBus failoverPublisher) {
        this.publisher         = publisher;
        this.failoverPublisher = failoverPublisher;
        this.exchangeName      = "domain_events";
    }

    @Override
    public void publish(List<DomainEvent> events) {
        events.forEach(this::publish);
    }

    private void publish(DomainEvent domainEvent) {
        try {
            this.publisher.publish(domainEvent,
exchangeName);
        } catch (AmqpException error) {
            failoverPublisher.publish(Collections.singletonLi
st(domainEvent));
        }
    }
}
```

Este código Java implementa un bus de eventos utilizando RabbitMQ como mecanismo principal de comunicación. La clase `RabbitMqEventBus` implementa la interfaz `EventBus` y utiliza un `RabbitMqPublisher` para publicar eventos en un intercambio específico en RabbitMQ. En caso de que ocurra un error al publicar un evento, como una excepción de AmqpException, el bus de eventos realiza un failover utilizando un `MariaDBEventBus`, que es otro mecanismo de publicación de eventos basado en MariaDB. Esto garantiza una mayor robustez y resiliencia en la comunicación de eventos dentro del sistema. La anotación `@Primary` indica que esta implementación del bus de eventos es la principal en caso de que haya varias implementaciones disponibles.

RabbitMqExchangeNameFormatter.java,
RabbitMqQueueNameFormatter.java

```java
package
com.devsoftec.xyphire.auth.shared.infrastructure.bus.event.ra
bbitmq;

public final class RabbitMqExchangeNameFormatter {
    public static String retry(String exchangeName) {
        return String.format("retry-%s", exchangeName);
    }

    public static String deadLetter(String exchangeName) {
        return String.format("dead_letter-%s", exchangeName);
    }
}
```

```java
package
com.devsoftec.xyphire.auth.shared.infrastructure.bus.event.ra
bbitmq;

import
com.devsoftec.xyphire.auth.shared.infrastructure.bus.event.Do
mainEventSubscriberInformation;

public final class RabbitMqQueueNameFormatter {
    public static String
format(DomainEventSubscriberInformation information) {
        return information.formatRabbitMqQueueName();
    }

    public static String
formatRetry(DomainEventSubscriberInformation information) {
        return String.format("retry.%s",
format(information));
    }

    public static String
formatDeadLetter(DomainEventSubscriberInformation
information) {
        return String.format("dead_letter.%s",
format(information));
    }
}
```

Los primeros dos fragmentos de código Java presentan utilidades para formatear nombres de intercambio y cola en RabbitMQ. `RabbitMqExchangeNameFormatter` proporciona métodos estáticos para agregar prefijos a los nombres de intercambio, como "retry-" y "dead_letter-", mientras que `RabbitMqQueueNameFormatter` ofrece funciones para generar nombres de cola y sus variantes de reintentos y dead-letter, basándose en la información del suscriptor de eventos de dominio. Estas clases son útiles para establecer una convención de nomenclatura consistente en la configuración del bus de eventos de RabbitMQ.

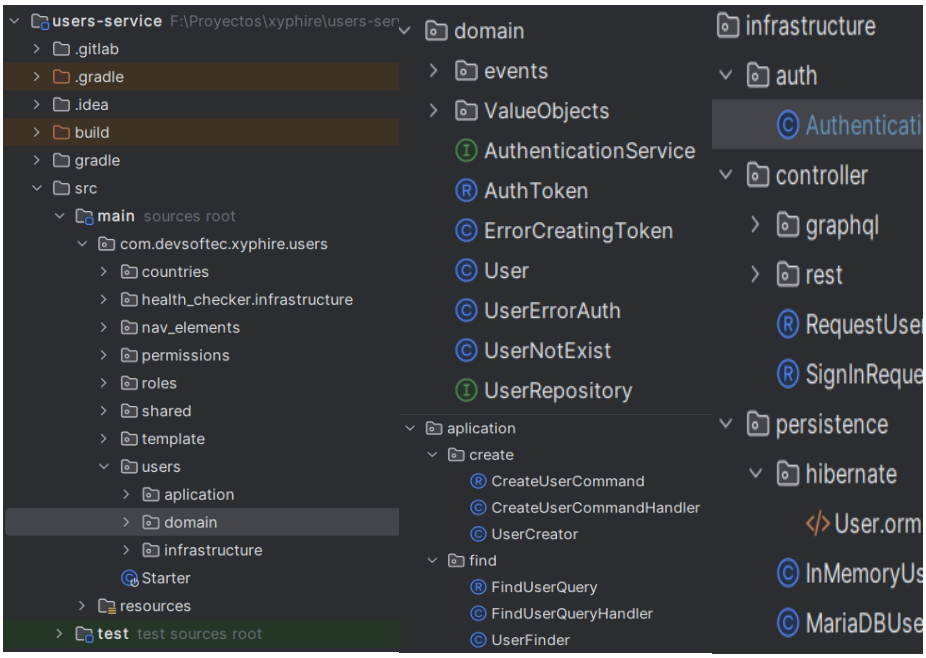
4.2.4 Arquitectura del Software

La arquitectura hexagonal, es un estilo de diseño de software que busca crear sistemas más mantenibles, flexibles y testables. Este enfoque podemos observar en la figura 15 la estructura de directorios que mantienen los microservicios, tenemos el package raíz com.devsoftec.xyphire.contexto, este contiene nuestros módulos (users, countries, etc), a su vez estos directorios contienen la ***capa de dominio*** que incluye objetos que están dentro de nuestro contexto (Usuario, roles, Countries, etc), la lógica o dominio de negocio y servicios de dominio, en la ***capa de aplicación*** se encuentran los casos de uso del microservicio (search, find, créate, sign_in, etc) y la ***capa de infraestructura*** depende de decisiones externas aquí se realizó las implementaciones específicas de la infraestructura externa del microservicio, incluyen componentes como por ejemplo base de datos, frameworks, entre otros elementos que no forman parte de la lógica de dominio.

El ***directorio Test*** mantiene la misma estructura que el directorio src, y se realizó la implementación de los tests necesarios conforme se puede observar la cobertura de los tests en la figura 16.

Nota: Se respeto estrictamente la regla de dependencia que nos dice que *"cada una de nuestras capas solo conocerá las clases de la capa inmediatamente siguiente"* en este orden desde la capa externa a la interna: Infraestructura → Aplicación → Dominio.

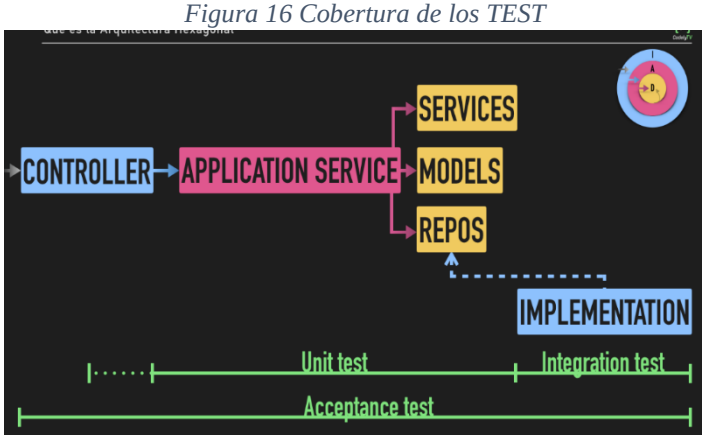
Figura 15: Estructura de Directorios Arquitectura Hexagonal



Realizado por: Eduardo G & Melanin G

En la figura 15 observamos que las pruebas unitarias se aplicaron a la capa de aplicación en conjunto con el dominio. Las pruebas de Integración cubren la capa de infraestructura y las pruebas de aceptación cubren todas las capas del sistema.

Figura 16 Cobertura de los TEST



Fuente: Cobertura de test tomado de: (CodelyTv, 2020)

4.1.Desarrollo de los Sprint

4.1.1. Sprint 1

4.1.1.1. Objetivos del Sprint

Desarrollar funciones de inicio de sesión, validando credenciales, y permitir la recuperación segura de contraseñas.

4.1.1.2. Product Backlog

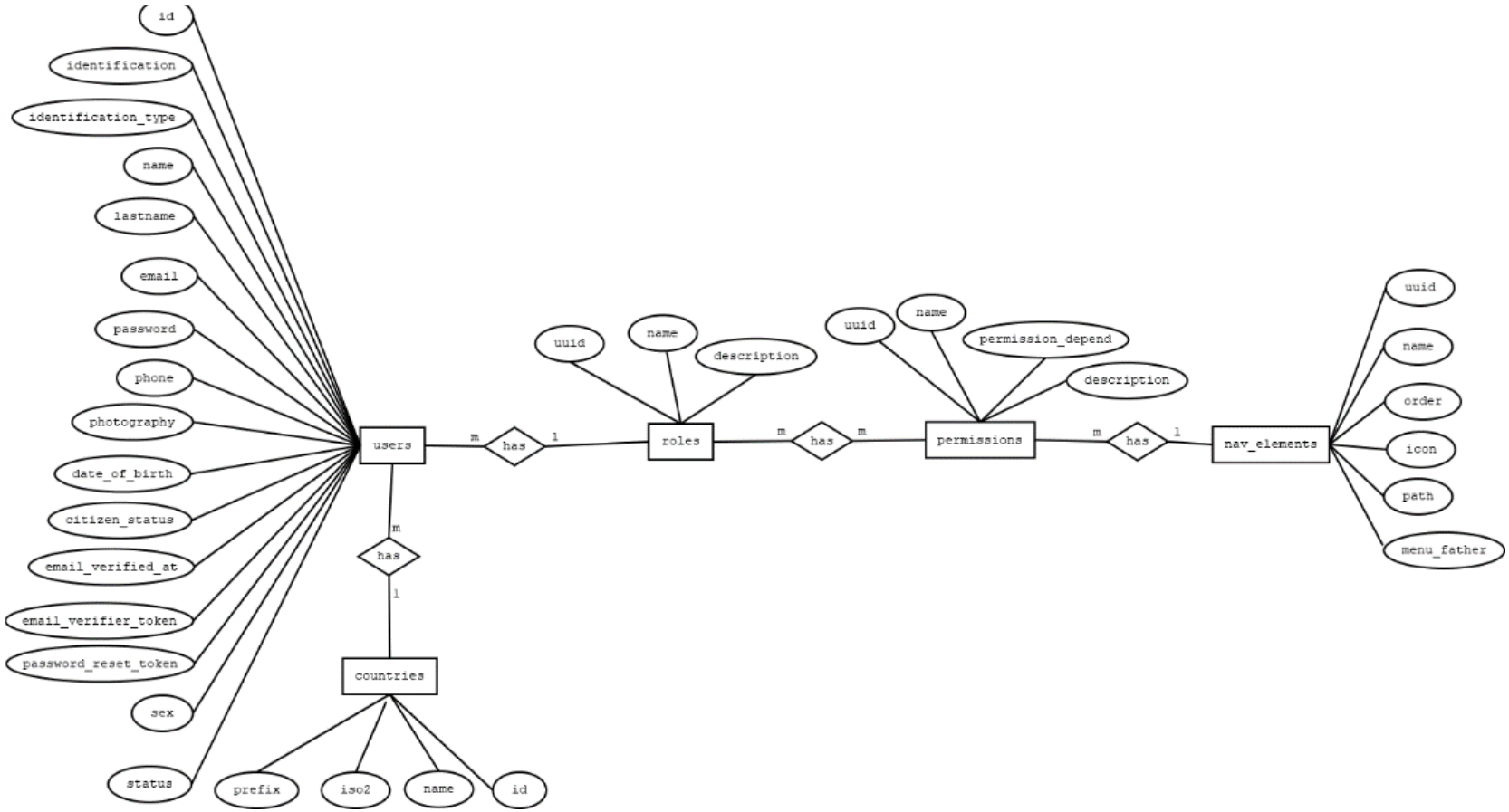
Tabla 3: Product Backlog Sprint 1

Código	Historia de Usuario	Epic (Módulo)	Estado
XS-9	Gestionar el inicio de sesión.	Gestión de Usuarios	Finalizada
XS-14	Gestionar recuperación contraseña.	Gestión de Usuarios	Finalizada
XS-9-1	Implementar interfaz gráfica.	Gestión de Usuarios	Finalizada

Realizado por: Eduardo G & Melanin G

4.1.1.3. Diagrama Entidad Relación

Figura 17 Diagrama E-R Servicio de Usuarios

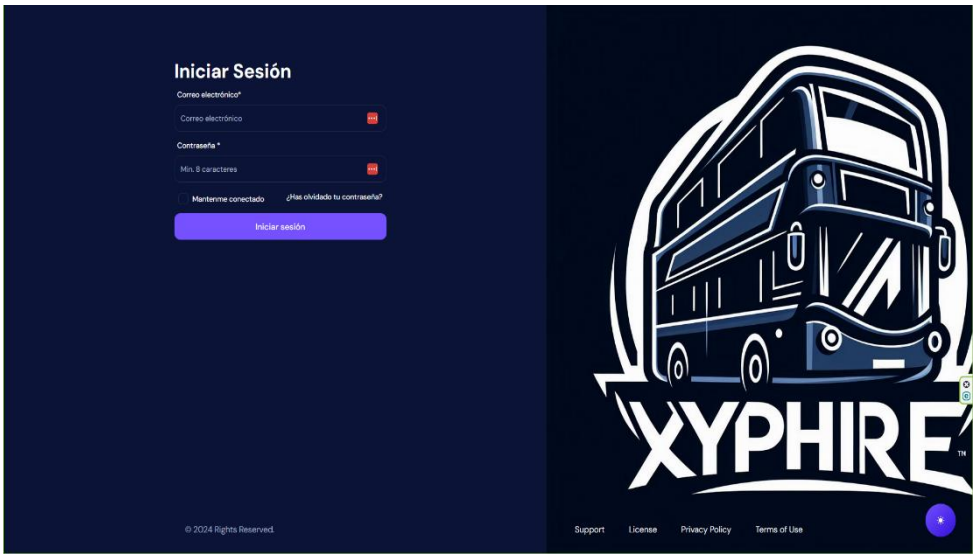


Realizado por: Eduardo G & Melanin G

4.1.1.4. Interfaz de Usuario

En la figura 19 se muestra la interfaz para iniciar sesión.

Figura 18: Pantalla Iniciar Sesión



Realizado por: Eduardo G & Melanin G

Figura 19: Pantalla para restaurar la contraseña.

Realizado por: Eduardo G & Melanin G

4.1.1.5.Codificación

A continuación, se describe las clases necesarias para implementar la autenticación y autorización basada en jwt.

AuthenticationServiceImpl.java

```
package com.devsoftec.xyphire.auth.users.infrastructure.auth;

import com.devsoftec.xyphire.auth.shared.domain.Service;
import
com.devsoftec.xyphire.auth.users.aplication.AuthenticatedUser
;
import
com.devsoftec.xyphire.auth.users.aplication.TokenResponse;
import
com.devsoftec.xyphire.auth.users.aplication.UserResponse;
import com.devsoftec.xyphire.auth.users.domain.*;
import
com.devsoftec.xyphire.auth.users.domain.ValueObjects.UserEmai
l;
import
com.devsoftec.xyphire.auth.users.domain.ValueObjects.UserPass
word;
import
org.springframework.security.authentication.AuthenticationMan
ager;
import
org.springframework.security.authentication.UsernamePasswordA
uthenticationToken;

@Service
public final class AuthenticationServiceImpl implements
AuthenticationService {

    private final AuthenticationManager
authenticationManager;
    private final UserRepository userRepository;
    private final JwtUtils jwtUtils;

    public AuthenticationServiceImpl(AuthenticationManager
authenticationManager, UserRepository userRepository,
JwtUtils jwtTokenProvider, JwtUtils jwtUtils) {
        this.authenticationManager = authenticationManager;
        this.userRepository = userRepository;
        this.jwtUtils = jwtUtils;
    }
```

```
    @Override
    public AuthenticatedUser authenticate(UserEmail email,
UserPassword password) {
        authenticationManager.authenticate(new
UsernamePasswordAuthenticationToken(email.value(),
password.value()));
        User user =
userRepository.findByEmail(email).orElseThrow(()->new
UserNotExist(email.value()));
        AuthToken token = jwtUtils.generateToken(user);
        return new
AuthenticatedUser(UserResponse.fromAggregate(user),
TokenResponse.fromAggregate(token));
    }
}
```

El código Java proporcionado define un servicio de autenticación que implementa la interfaz `AuthenticationService`. Utiliza Spring Security para autenticar usuarios mediante correo electrónico y contraseña. Luego, genera un token JWT para el usuario autenticado.

JwtUtils.java

```
package com.devsoftec.xyphire.auth.users.infrastructure.auth;


import com.devsoftec.xyphire.auth.users.domain.AuthToken;
import com.devsoftec.xyphire.auth.users.domain.User;
import io.jsonwebtoken.Claims;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.security.SignatureAlgorithm;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import org.springframework.beans.factory.annotation.Value;
import
org.springframework.security.core.userdetails.UserDetails;
import org.springframework.stereotype.Component;

import java.security.interfaces.RSAPrivateKey;
import java.security.interfaces.RSAPublicKey;
import java.time.Instant;
import java.util.Date;
import java.util.HashMap;
```

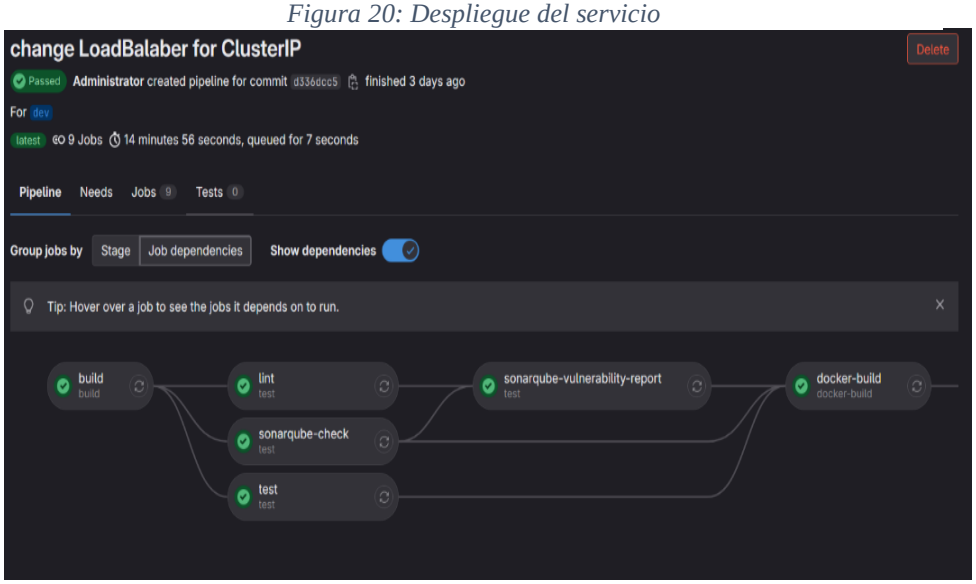
```
import java.util.Map;
import java.util.function.Function;

@Component
public final class JwtUtils {
    private final Logger logger =
LogManager.getLogger(JwtUtils.class);
    @Value("${jwt.public.key}")
    RSAPublicKey public_key;

    @Value("${jwt.private.key}")
    RSAPrivateKey private_key;

    Integer expiration_time = 3600;


    public AuthToken generateToken(User user) {
        Map<String, Object> claims = new HashMap<>();
        claims.put("UserName", user.name().value());
        claims.put("LastName", user.lastname().value());
        claims.put("email", user.email().value());
        claims.put("status", user.status().value());
        Instant issue_at = Instant.now();
        Instant expiration =
issue_at.plusSeconds(expiration_time);
        return createToken(claims, user, this.private_key,
issue_at, expiration);
    }

    private AuthToken createToken(Map<String, Object> claims,
User user, RSAPrivateKey jwtPrivateKey, Instant issue_at,
Instant expiration) {
        SignatureAlgorithm alg = Jwts.SIG.RS256;
        String token = Jwts.builder()
            .subject(user.id().value())
            .claims(claims)
            .issuedAt(Date.from(issue_at))
            .expiration(Date.from(expiration))
            .signWith(jwtPrivateKey, alg)
            .compact();
        logger.info("Token create to user: " +
user.email().value());
        return new AuthToken(token, expiration, issue_at);
    }

    public String extractUsername(String token) {
```

```
        return extractClaims(token, claims -> (String)
claims.get("email"));
    }

    private <T> T extractClaims(String token,
Function<Claims, T> claimsTFunction) {
        return
claimsTFunction.apply(Jwts.parser().verifyWith(public_key).bu
ild().parseSignedClaims(token).getPayload());
    }

    public boolean isTokenValid(String token, UserDetails
userDetails) {
        final String username = extractUsername(token);
        if (username.equals(userDetails.getUsername()) &&
!isTokenExpired(token))
            return true;
        logger.warn("Token is invalid");
        return false;
    }

    public boolean isTokenExpired(String token) {
        if (!extractClaims(token,
Claims::getExpiration).before(new Date()))
            return false;
        logger.info(extractUsername(token) + " user's token
is expired");
        return true;
    }
}
```

Este código Java define una clase llamada `JwtUtils`, que es un componente de Spring encargado de generar y validar tokens JWT (JSON Web Tokens) para la autenticación de usuarios. La clase utiliza una clave privada y una clave pública RSA para firmar y verificar los tokens JWT. La función `generateToken` crea un token JWT con información del usuario y una fecha de expiración. Las funciones `extractUsername`, `isTokenValid` y `isTokenExpired` se utilizan para extraer información del token, verificar su validez y comprobar si ha caducado. Además, el componente utiliza el framework Log4j para registrar eventos y mensajes de registro.

4.1.1.6. Reporte de puesta en producción

Figura 20: Despliegue del servicio



Realizado por: Eduardo G & Melanin G

4.1.2. Sprint 2

4.1.2.1.Objetivo del Sprint

Proporcionar funcionalidades relacionadas con la gestión de conductores, frecuencias, vehículos y venta de boletos.

4.1.2.2. Historias de usuario seleccionadas

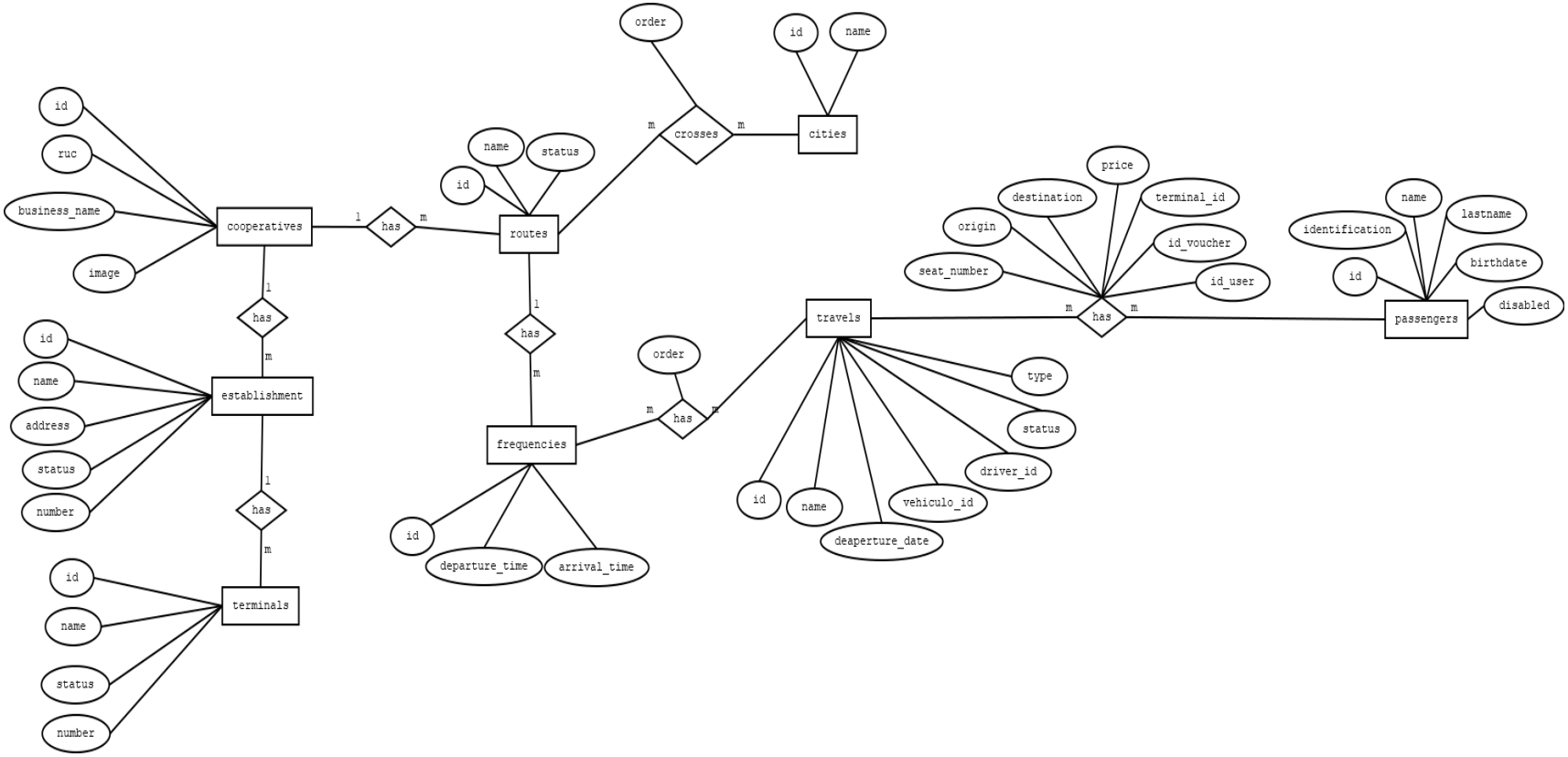
Tabla 4 Product Backlog Sprint 2

Código	Historia de Usuario	Epic	Estado
XS-19	Gestionar Socios.	BackOffice Cooperativa	Finalizada
XS-41	Gestión de Carrocerías.	BackOffice Cooperativa	Finalizada
XS-20	Gestionar Vehículos.	BackOffice Cooperativa	Finalizada
XS-42	Gestión de Conductores.	BackOffice Cooperativa	Finalizada
XS-34	Gestión de Frecuencias.	BackOffice Cooperativa	Finalizada
XS-36	Gestión de Rutas.	BackOffice Cooperativa	Finalizada
XS-32	Venta de boletos.	Gestión Contable	Finalizada
XS-45	Despacho de viajes	Gestión Contable	Finalizada

Realizado por: Eduardo G & Melanin G

4.1.2.3. Diagrama Entidad Relación

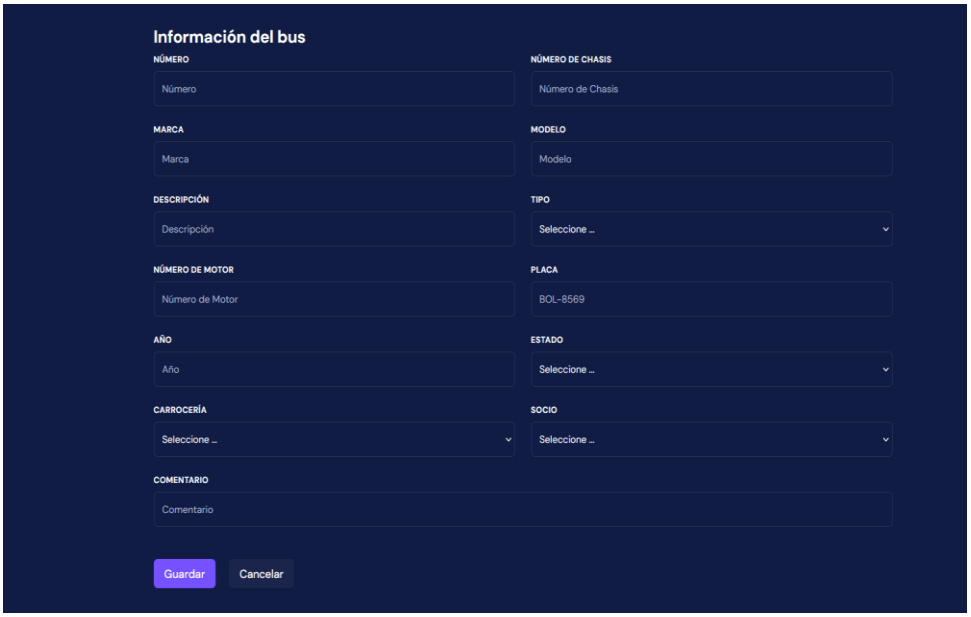
Figura 21 Diagrama E-R Servicio de Boletería



Realizado por: Eduardo G & Melanin G

4.1.2.4.Interfaz

Figura 22 Interfaz para crear un Bus

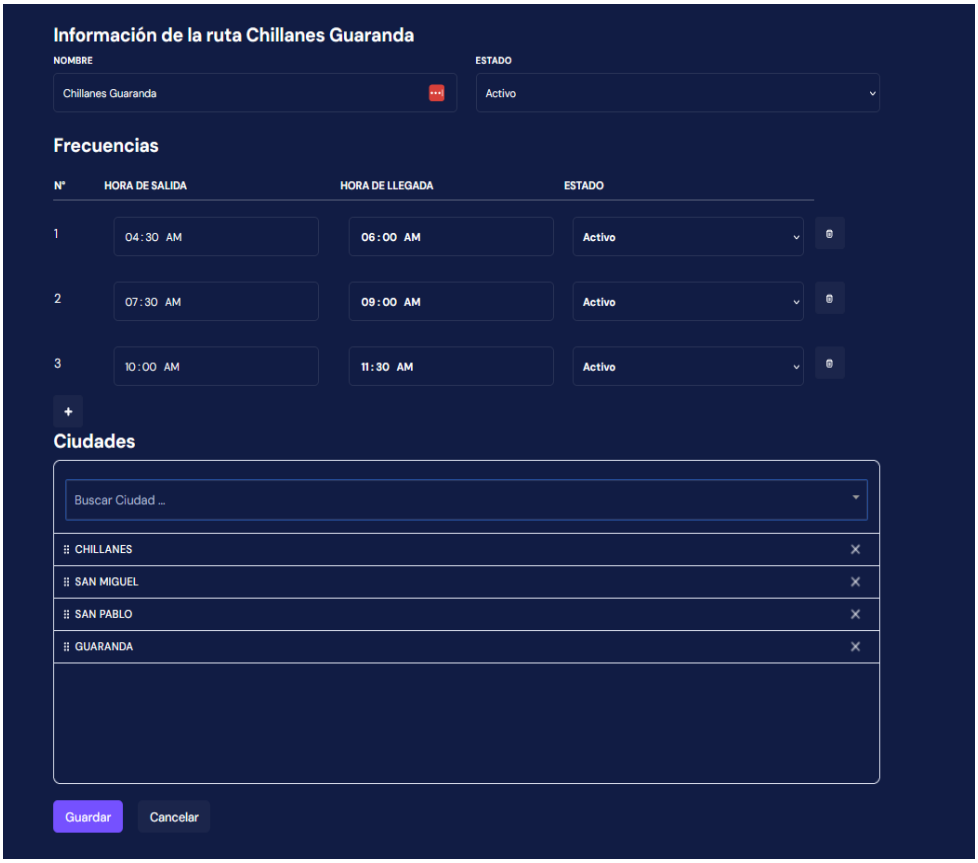


Realizado por: Eduardo G & Melanin G

Figura 23: Formulario para crear la carrocería

Realizado por: Eduardo G & Melanin G

Figura 24 Interfaz para crear una Ruta



Realizado por: Eduardo G & Melanin G

4.1.3. Sprint 3

4.1.3.1.Objetivo del Sprint

Desarrollar un sistema contable completo que incluya la gestión de cuentas, asientos contables, reportes de mayores y la implementación de facturación electrónica.

4.1.3.2. Historias de usuario seleccionadas

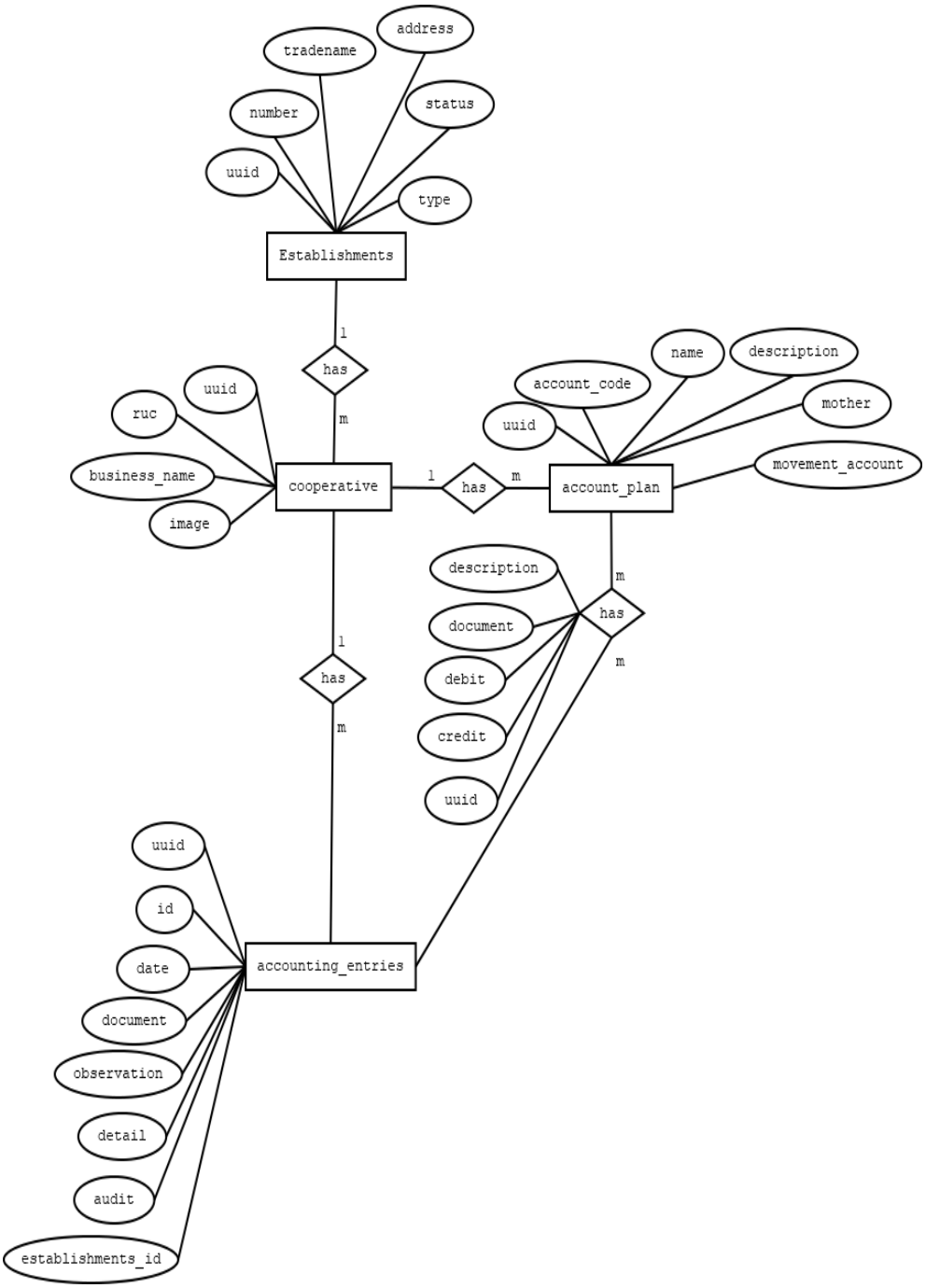
Tabla 5 Product Backlog Sprint 3

Código	Historia de Usuario	Epic	Estado
XS-21	Gestión plan de cuentas contables.	Gestión Contable	Finalizada
XS-22	Gestión Asientos contables.	Gestión Contable	Finalizada
XS-23	Reporte de Mayores.	Gestión Contable	Finalizada
XS-45	Implementar Facturación Electrónica.	Gestión Contable	Finalizada

Realizado por: Eduardo G & Melanin G

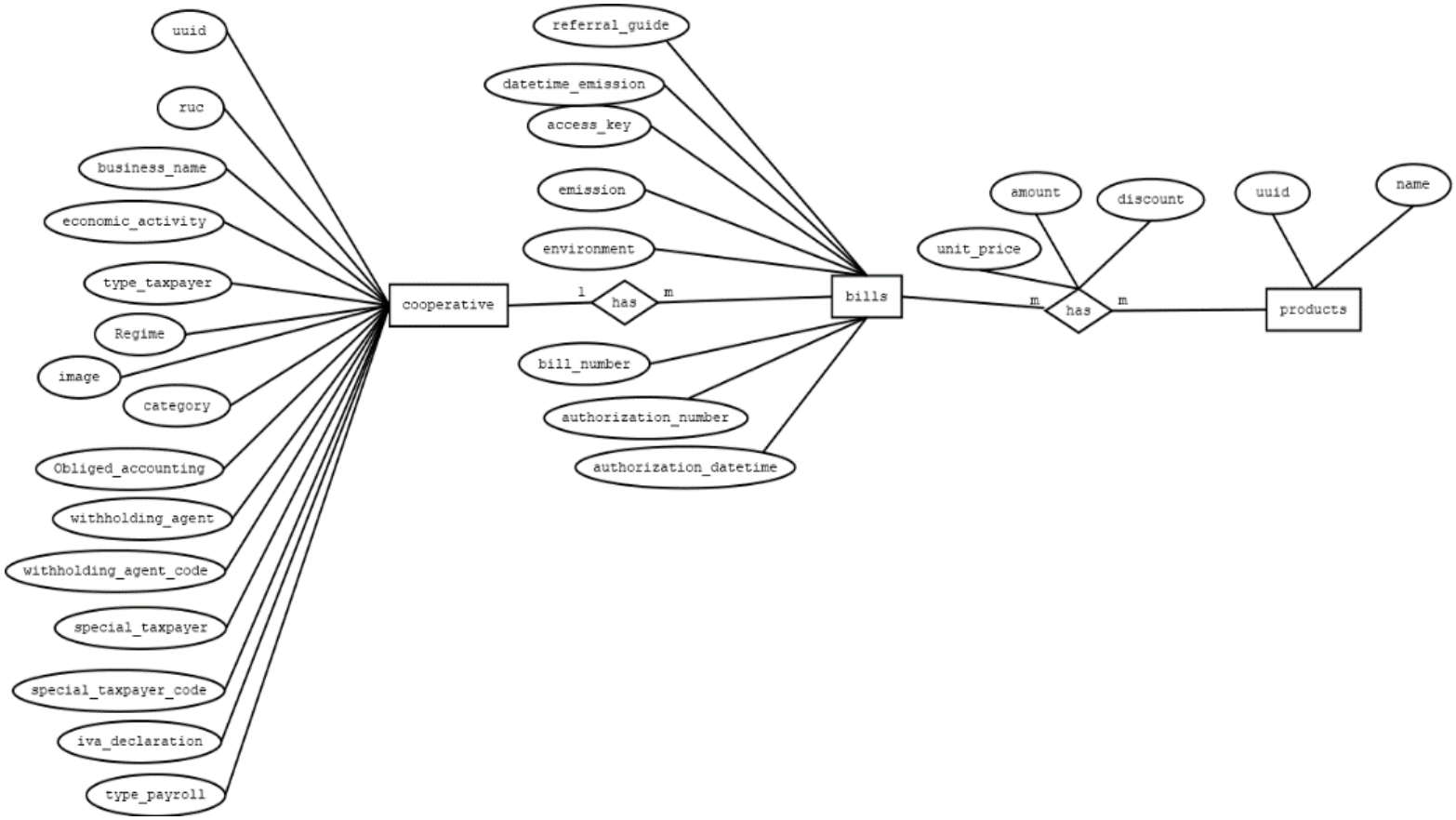
4.1.3.3. Diagrama Entidad Relación.

Figura 25 Diagrama E-R servicio de contabilidad



Realizado por: Eduardo G & Melanin G

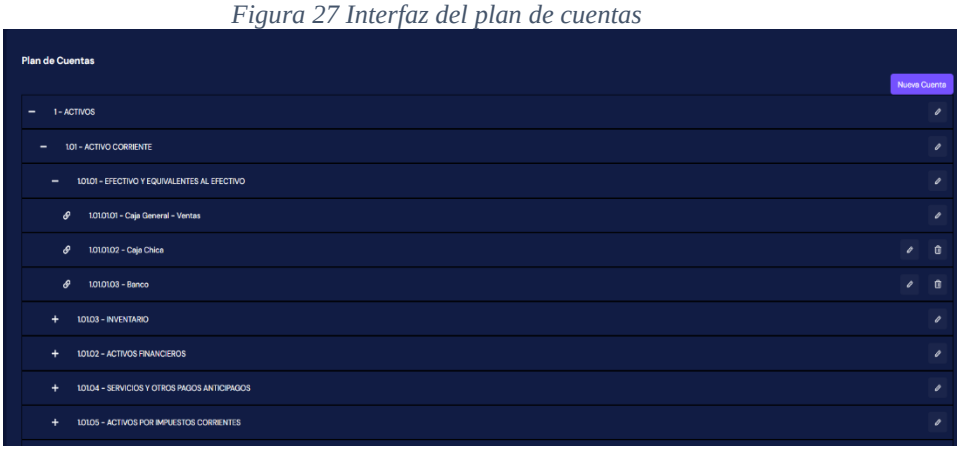
Figura 26 Diagrama E-R Servicio de Facturación Electrónica



Realizado por: Eduardo G & Melanin G

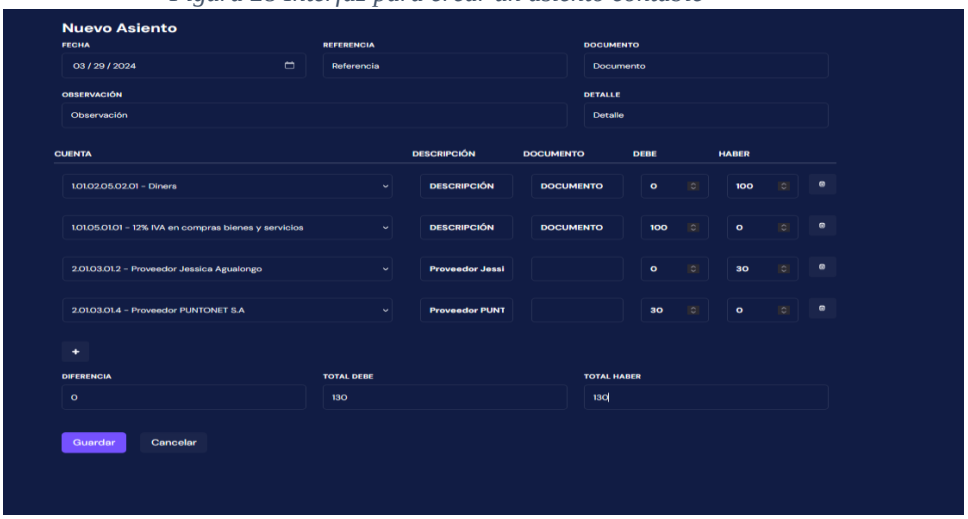
4.1.3.4. Interfaz

Figura 27 Interfaz del plan de cuentas



Realizado por: Eduardo G & Melanin G

Figura 28 Interfaz para crear un asiento contable



Realizado por: Eduardo G & Melanin G

4.1.4. Sprint 4

4.1.4.1.Objetivo del Sprint

Implementar un API Gateway y un frontend para una aplicación o sistema.

4.1.4.2. Historias de usuario seleccionadas

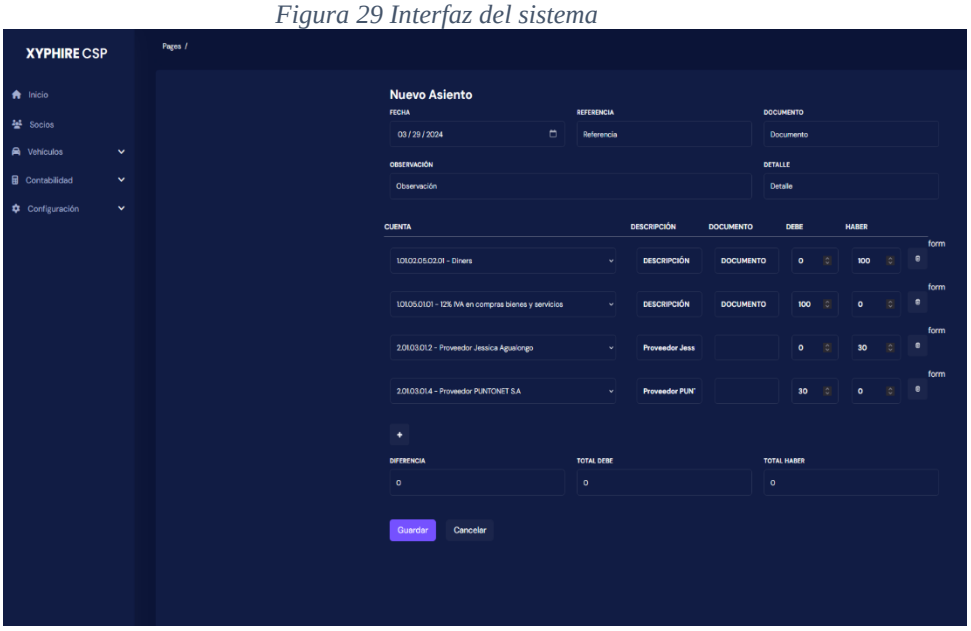
Tabla 6 Product Backlog Sprint 4

Código	Historia de Usuario	Epic	Estado
XS-45	Implementar el Api Gateway	Infraestructura	Finalizada
XS-45	Implementar el frontend	Sistema	Finalizada
XS-60	Desplegar el sistema	Infraestructura	Finalizada

Realizado por: Eduardo G & Melanin G

4.1.4.3. Interfaz

Figura 29 Interfaz del sistema



Realizado por: Eduardo G & Melanin G

4.1.4.4.Codificación

ApiGateway.ts

```
import { Module } from '@nestjs/common';
import { AppController } from './app.controller';
import { AppService } from './app.service';
import { GraphQLModule } from '@nestjs/graphql';
import { ApolloGatewayDriver, ApolloGatewayDriverConfig }
from '@nestjs/apollo';
import { IntrospectAndCompose } from '@apollo/gateway';

@Module({
  imports: [
    GraphQLModule.forRoot<ApolloGatewayDriverConfig>({
      driver: ApolloGatewayDriver,
      server: {
        path: 'api/graphql',
      },
      gateway: {
        supergraphSdl: new IntrospectAndCompose({
          subgraphs: [
            { name: 'authService', url: 'http://xyphire-auth-
service/graphql' },
            { name: 'ContabilityService', url:
'http://xyphire-contability-service/graphql' },
            { name: 'TicketsService', url: 'http://xyphire-
tickets-service/graphql' },
            { name: 'facturationService', url:
'http://xyphire-facturation-service/graphql' },
          ],
        }),
      },
    }),
  ],
  controllers: [AppController],
  providers: [AppService],
})
export class AppModule {}
```

Este código define un módulo en una aplicación NestJS que utiliza GraphQL y el patrón de Gateway de Apollo Federation. Se configura el módulo para componer múltiples servicios GraphQL en un único gráfico superpuesto, especificando las URLs de los subgráficos a través de `IntrospectAndCompose`, lo que permite la introspección y composición del supergráfico.

4.1.4.5.Despliegue

Desplegamos el sistema de gestión y venta de boletos en Kubernetes utilizando contenedores Docker para cada microservicio. Definimos recursos en Kubernetes con archivos YAML y los desplegamos en el clúster usando `kubectl apply`. Configuramos servicios de balanceo de carga para distribuir el tráfico y utilizamos herramientas de monitorización como Prometheus y Grafana para supervisar el sistema. Kubernetes se encargó de la orquestación, escalabilidad y autorecuperación, garantizando un funcionamiento eficiente y confiable del sistema en todo momento.

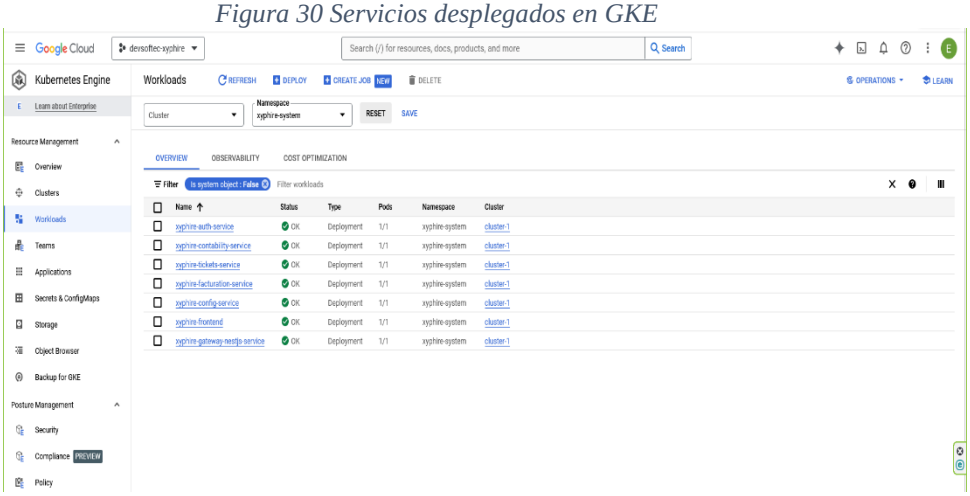
Estructura del archivo YAML

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: xyphire-auth-service
  namespace: xyphire-system
spec:
  replicas: 1
  selector:
    matchLabels:
      app: xyphire-auth-service
  template:
    metadata:
      labels:
        app: xyphire-auth-service
    spec:
      containers:
        - name: auth-service
          image: git.devsoftec.com:5050/xyphire/auth-
service:${DOCKER_IMAGE_TAG}
          ports:
            - containerPort: 8081
          env:
            - name: SPRING_PROFILES_ACTIVE
              value: ${DOCKER_IMAGE_PROFILE}
          livenessProbe:
            httpGet:
              path: /health
              port: 8081
            initialDelaySeconds: 60
            periodSeconds: 60
          readinessProbe:
```

```
            httpGet:
              path: /health
              port: 8081
            initialDelaySeconds: 60
            periodSeconds: 60
      imagePullSecrets:
        - name: gitlab-login-devsoftec

---
apiVersion: v1
kind: Service
metadata:
  name: xyphire-auth-service
  namespace: xyphire-system
spec:
  selector:
    app: xyphire-auth-service
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8081
  type: ClusterIP
```

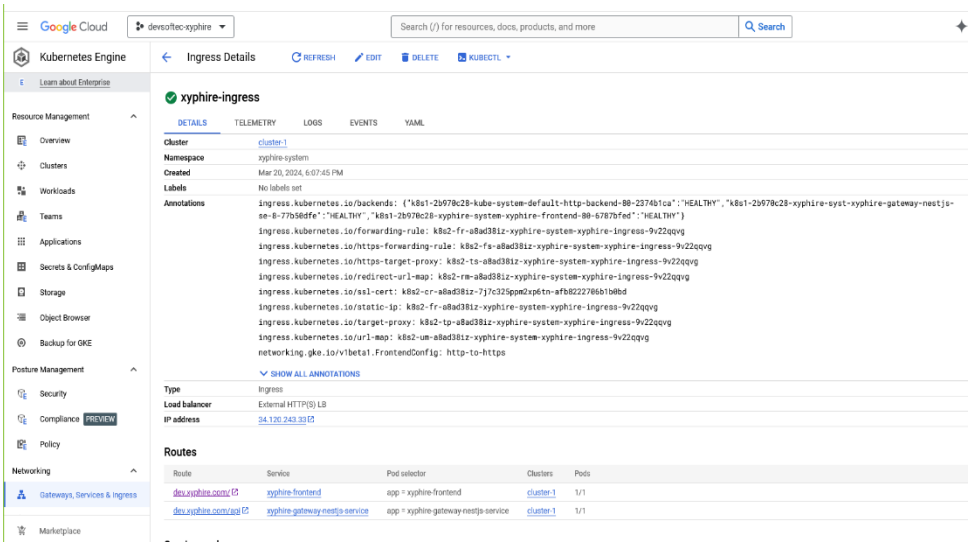
A continuación, en la figura 30 observamos los servicios desplegados en Google kubernertes service (GKE) y en la figura 31 el ingress o balanceador de carga implementado.

Figura 30 Servicios desplegados en GKE



Realizado por: Eduardo G & Melanin G

Figura 31 Balanceador de Carga



Realizado por: Eduardo G & Melanin G

4.2. Evaluación del Sistema

Basándonos en las métricas de Kubernetes y el monitoreo continuo del sistema de gestión y venta de boletos en la Cooperativa de Transportes "San Pedrito", nos complace informar que todo se encuentra en orden. Las métricas de uso de recursos muestran un aprovechamiento eficiente de la infraestructura, con un escalado automático que responde de manera precisa a las demandas de carga de trabajo.

Figura 32: Consumo de CPU

Realizado por: Eduardo G & Melanin G

CONCLUSIONES

- Tras el análisis de los procesos de negocio de la Cooperativa de Transportes “San Pedrito”, se ha identificado una serie de áreas clave relacionadas con la gestión y venta de boletos, este análisis ha proporcionado una comprensión profunda de los flujos de trabajo, los puntos críticos y las oportunidades de mejora dentro de la organización.
- La definición de la arquitectura del sistema basada en Domain-Driven Design (DDD) ha permitido establecer un marco sólido para el desarrollo de un sistema informático adaptado a las necesidades específicas de la cooperativa, esta metodología ha facilitado la identificación y el modelado de los dominios principales, así como la definición de las interacciones entre ellos, lo que garantiza una estructura coherente y escalable.
- Se desarrollo con éxito el sistema informático para la gestión y venta de boletos cumpliendo con los requisitos y procesos establecidos por la Cooperativa de Transportes “San Pedrito”.
- La implementación de la metodología Domain-Driven Design (DDD) ha permitido desarrollar un sistema informático integral que resuelve eficazmente desafíos previos en la Cooperativa de Transportes “San Pedrito”, tales como la venta duplicada de asientos y la gestión ineficiente de información.

RECOMENDACIONES

- Para el análisis de procesos de negocio, se sugiere establecer un ciclo de mejora continua que incluya la evaluación periódica de la eficiencia y efectividad de los procesos, además se recomienda considerar la contratación de personal especializado para brindar mantenimiento del sistema y soporte al personal, esto garantizará un funcionamiento óptimo del sistema informático y una respuesta eficaz ante posibles problemas técnicos, lo que contribuirá significativamente a la eficacia operativa y la satisfacción del cliente.
- En cuanto a la definición de la arquitectura del sistema basada en DDD, se recomienda promover la colaboración entre equipos de desarrollo y expertos en dominio, documentar exhaustivamente la arquitectura del sistema y establecer un proceso de revisión periódica para garantizar su coherencia y alineación con los objetivos del negocio.
- Para el uso del sistema proporcionar formación y soporte adecuados al personal, y establecer mecanismos de retroalimentación para recoger las opiniones de los usuarios y realizar mejoras continuas en el sistema.
- Para mejorar el funcionamiento y la satisfacción del cliente, es esencial que la empresa esté en constante evolución, esto implica capacitar al personal, asegurar que el sistema esté bien mantenido y brindar apoyo continuo.

BIBLIOGRAFÍA

Autentia S.L. (2023). *Software Design: La guía completa*. https://www.autentia.com/libros/

Betts, D., Domínguez, J., Melnik, G., Simonazzi, F., & Subramanian, M. (2012). *Exploring CQRS and Event Sourcing A journey into high scalability, availability, and maintainability with Windows Azure*.

Cando Jacome, R. I., & Tenesaca Pérez, J. E. (2021). *Desarrollo de una aplicación web y móvil para la automatización en la gestión de venta y reserva de boletos en la Cooperativa de Transportes Ambateñita de la ciudad de Ambato*.

CodelyTv. (2020). *Curso de Testing: Introducción y buenas prácticas*. Codely.

Dobbelaere, P., & Esmaili, K. S. (2017). *Kafka versus RabbitMQ*.

Evans, E. (2003). *Domain-Driven Design: Tackling Complexity in the Heart of Software*.

Fidao, C. (2023). *Hexagonal Architecture*. Fideloper.Com.

Fowler, M. (2003). *Patterns of enterprise application architecture*. Addison-Wesley.

Frensia Tanaga Anaclaudia, Dian Pramana, & I Made Arya Budhi Saputra. (2023). Reactjs and Expressjs Implementation In PMK ITB STIKOM Bali Activity Management. *Aptisi Transactions on Technopreneurship (ATT)*, *5*(3), 1–11. https://doi.org/10.34306/att.v5i3.313

Green, G. (2022). RabbitMQ vs Kafka: How to choose an event-streaming broker. In *Vmware.com*.

Iglberger, K. (n.d.). *The SOLID Principles*.

Kubernetes. (2021). *Production-Grade Container Orchestration*. Kubernetes.Io. https://kubernetes.io/

Leach, P. J., Salz, R., & Mealling, M. H. (2023, November 20). *RFC 4122: A Universally Unique IDentifier (UUID) URN namespace*. Https://Datatracker.Ietf.Org/Doc/Html/Rfc4122.

Pacheco, V. Feitosa. (2018). *Microservice Patterns and Best Practices : Explore patterns like CQRS and event sourcing to create scalable, maintainable, and testable microservices*. Packt Publishing.

Pressman, R. S. (2010). *Ingeniería del software : un enfoque práctico*. McGraw-Hill.

Purpose of the Scrum Guide. (2020). In *Scrumguides.org*.

Richardson, C. (2023). *Microservices Pattern: Microservice Architecture pattern*. Microservices.Io; Chris Richardson.

SALAN VILLENA, J. C. (2015). *APLICACIÓN WEB PARA LA GESTIÓN DE RUTAS Y PRE-RESERVAS DE LA COOPERATIVA DE TRANSPORTES EL DORADO*.

SANDOVAL POZO, D. M. (2021). *APLICACIÓN WEB PARA MEJORAR LA GESTIÓN ADMINISTRATIVA DE LA COOPERATIVA DE TRANSPORTES HUACA – JULIO ANDRADE*. https://dspace.uniandes.edu.ec/bitstream/123456789/13486/1/UT-SIS-EXC-001-2021.pdf

Schwaber, K., & Sutherland, J. (2020). *The Scrum Guide The Definitive Guide to Scrum: The Rules of the Game*.

Singh, A. (2019). *Agile & Scrum*.

Tudose, C., Bauer, C., & King, G. (2023). *Java persistence with spring data and hibernate*. Manning Publications.

Vieira, D. (2022). *Designing Hexagonal Architecture with Java: Build maintainable and long-lasting applications with Java and Quarkus*.

ANEXOS

ANEXO 1

Historias de Usuario

HISTORIAS DE USUARIO
XYPHIRE

XS-UT-DP-001
Versión 1.0

Nombre del Proyecto:	XYPHIRE
Realizado por:	Eduardo Guastay & Melanin Ganan
Fecha:	03-01-2024

CONTROL DEL DOCUMENTO

XS: XYPHIRE SOFTWARE
UT: Unidad de Tecnología
DP: Documento Plantilla
001: Secuencial

Nombre del Documento: Historias de Usuario
No. Documento: XS-UT-DP-001

REGISTROS DE CAMBIOS EN EL DOCUMENTO			
Versión	**Motivo**	**Realizado por**	**Fecha**
1.0	Creación del documento	Eduardo Bonilla, Melanin Ganan	03/01/2024

CONTENIDO

1 MODELO DE LA SOLUCIÓN PROPUESTA 4

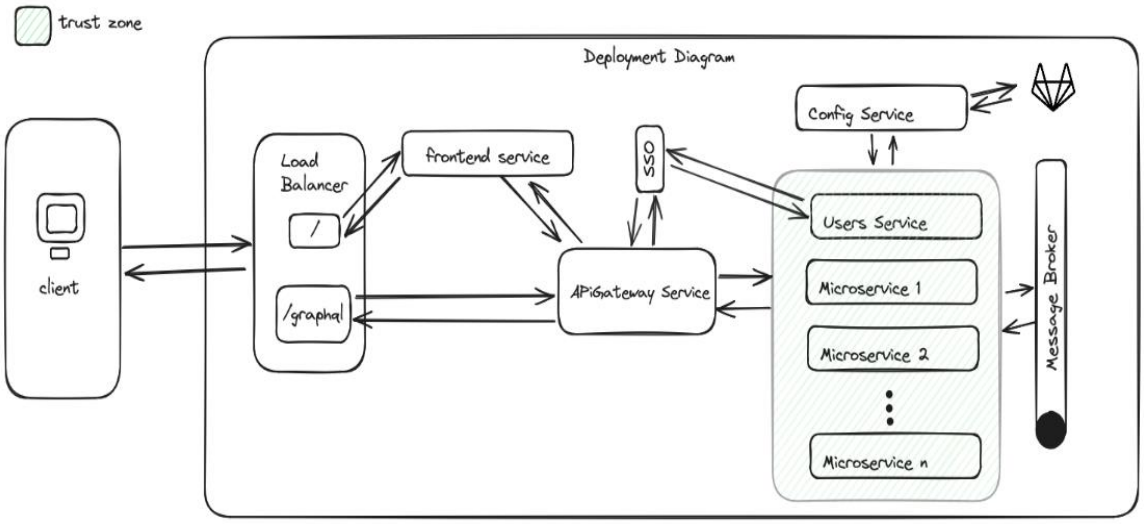
2 REQUERIMIENTOS FUNCIONALES 5

3 GLOSARIO DE TÉRMINOS: 11

1 MODELO DE LA SOLUCIÓN PROPUESTA

Para el desarrollo del sistema informático de gestión y venta de boletos en la Cooperativa de Transportes "San Pedrito", proponemos aplicar Domain Driven Design (DDD) para crear un modelo de dominio sólido y flexible. Este modelo guiará la arquitectura de microservicios, permitiendo una fácil escalabilidad y mantenimiento. Priorizaremos una interfaz de usuario intuitiva y segura para clientes y empleados, incluyendo funciones de búsqueda, reserva y seguimiento de boletos. Además, integraremos herramientas modernas para análisis de datos, optimizando así las operaciones de la cooperativa y mejorando la experiencia del usuario.

1.1 Diagrama de la solución



2 REQUERIMIENTOS FUNCIONALES

2.1 Historias de Usuario

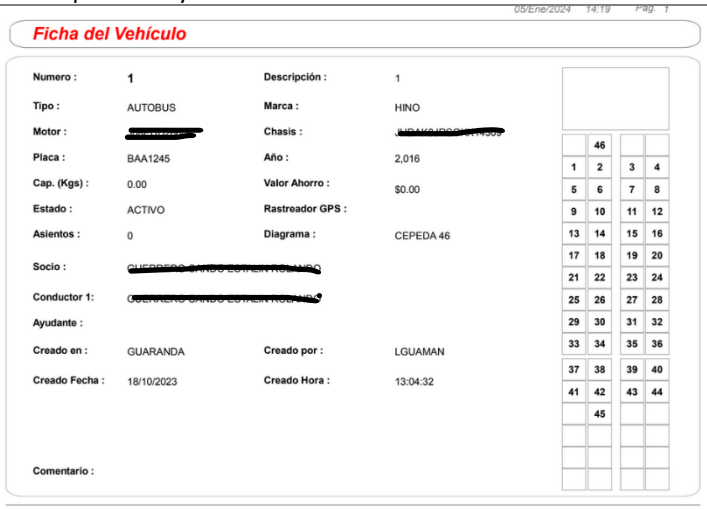
Nombre:	**[XS-9] Gestionar el inicio de sesión**
Descripción:	**Como** usuario, **Quiero** implementado un módulo eficiente para la gestión del inicio de sesión, asegurando la autenticación segura y autorización adecuada de los usuarios. La herramienta debe proporcionar una interfaz robusta para la gestión de credenciales, control de acceso y registro de actividades de inicio de sesión Para garantizar la seguridad y privacidad de la plataforma.
Criterios de Aceptación:	1. **Autenticación Segura:** • Como desarrollador, debo implementar un sistema de autenticación segura que utilice algoritmos de hash y cifrado para almacenar y verificar las contraseñas de los usuarios, protegiendo así la información sensible. 2. **Gestión de Credenciales:** • Como desarrollador, necesito una interfaz para gestionar las credenciales de los usuarios, permitiendo la creación, actualización y eliminación de cuentas, así como la recuperación segura de contraseñas olvidadas. 3. **Control de Acceso Basado en Roles:** • Como desarrollador, debo implementar un sistema de control de acceso basado en roles, permitiendo asignar roles específicos a cada usuario y restringir el acceso a recursos y funcionalidades según sus privilegios. 4. **Implementación de Token JWT (JSON Web Token):** • Como desarrollador, debo implementar un sistema de token JWT para gestionar sesiones de usuario de manera segura y eficiente, evitando la necesidad de almacenar información de sesión en el servidor.
Actores:	No aplica

Nombre:	**[XS-14] Gestionar recuperación contraseña**
Descripción:	**Como** usuario del sistema, **Quiero** implementado un mecanismo para la gestión de recuperación de contraseñas, permitiendo a los usuarios restablecer sus contraseñas de manera segura en caso de olvido. **Para** recuperar el acceso a la cuenta.
Criterios de Aceptación:	1. Envio de Acceso de recuperación via email.
Actores:	usuarios

Nombre:	**[XS-19] Gestionar Socios**
Descripción:	**Como** contador, **Quiero** ser capaz de gestionar la información de los socios. **Para** garantizar un seguimiento preciso de sus participaciones y derechos en la empresa.
Criterios de Aceptación:	1. **Registro de Socios:** • Como contador, quiero poder registrar la información básica de un nuevo socio, con los siguientes datos:  05/Ene/2024 14:22 Pag. 1 **Ficha del Socio** RUC : [redacted] Nombre : [redacted] Nacionalidad : ECUATORIANO Dirección : SELVA ALEGRE Y SALINAS Ciudad : GUARANDA Tel. Casa : Tel. Movil : [redacted] Email : a.aguilari@yahoo.es Profesión : (NINGUNA) Ingreso : 06/11/1994 Estado : ACTIVO Comentario : Vehículo : 4 Cta. Contable : Creado en : CONTABILIDAD Creado por : SUPERVISOR Creado Fecha : 04/10/2023 Creado Hora : 12:55:43 2. **Acceso Restringido:** • Como usuario, quiero asegurarme de que solo personal autorizado tenga acceso a la información confidencial de los socios para garantizar la privacidad y la seguridad de los datos.
Actores:	Contador

Nombre:	**[XS-41] Gestión de Carrocerías**
Descripción:	**Como** Administrador, **Quiero** realizar un seguimiento detallado de la configuración de asientos en cada vehículo. Esto incluye el número de asientos, su disposición y ubicación en la carrocería. **Para** planificar y asignar vehículos según las necesidades específicas de capacidad de pasajeros y requisitos operativos.
Criterios de Aceptación:	1. **Registro de Configuración de Asientos:** • Como usuario, debo poder registrar la configuración de asientos para cada carrocería, especificando el número total de asientos y su disposición, ya sea en filas o configuración personalizada. 2. **Ubicación de Asientos:**

	• Como usuario, necesito la capacidad de indicar la ubicación específica de los asientos en la carrocería, identificando claramente las filas y columnas para una referencia fácil. 3. **Capacidad de Pasajeros:** • Como usuario, necesito que el sistema calcule automáticamente la capacidad total de pasajeros basándose en la configuración de asientos registrada, proporcionando información útil para la planificación y asignación de vehículos. 4. **Visualización Gráfica:** • Como usuario, deseo una representación visual de la configuración de asientos, permitiéndome identificar fácilmente la disposición y ubicación de los asientos en la carrocería.
Actores:	Administrador

Nombre:	**[XS-20] Gestionar Vehículos**
Descripción:	**Como** Administrador, **Quiero** gestionar de manera eficiente la información esencial para cada unidad. La herramienta debe capturar datos clave. **Para** facilitar la supervisión y el mantenimiento adecuado de la flota.
Criterios de Aceptación:	 Ficha del Vehículo Numero : 1 — Descripción : 1 Tipo : AUTOBUS — Marca : HINO Motor : [redacted] — Chasis : [redacted] Placa : BAA1245 — Año : 2,016 Cap. (Kgs) : 0.00 — Valor Ahorro : $0.00 Estado : ACTIVO — Rastreador GPS : Asientos : 0 — Diagrama : CEPEDA 46 Socio : [redacted] Conductor 1: [redacted] Ayudante : Creado en : GUARANDA — Creado por : LGUAMAN Creado Fecha : 18/10/2023 — Creado Hora : 13:04:32 Comentario : 1. **Datos Básicos del Vehículo:** • Como usuario, necesito registrar información básica, para identificar claramente cada vehículo, con los siguientes datos.
Actores:	Administrador

Nombre:	**[XS-42] Gestión de Conductores.**
Descripción:	**Como** Administrador, **Quiero** una herramienta de control que me permita registrar y monitorear de manera efectiva la información clave asociada a cada conductor **Para** un mejor control interno.
Criterios de Aceptación:	1. **Registro Integral de Conductores:** • Como usuario, necesito la capacidad de registrar información integral de cada conductor.
Actores:	No aplica

Nombre:	**[XS-34] Gestión de Frecuencias.**
Descripción:	**Como** Administrador **Quiero** que me permita organizar de manera eficiente los horarios de los vehículos. La herramienta debe ser capaz de manejar información detallada sobre las frecuencias de salida y llegada, **Para** optimizar la programación de rutas para garantizar un servicio puntual y eficiente.
Criterios de Aceptación:	1. **Registro de Rutas:** • Como usuario, necesito poder registrar cada ruta, especificando los puntos de origen y destino, así como cualquier parada intermedia relevante. 2. **Frecuencias de Llegada:** • Como usuario, necesito registrar las frecuencias de llegada para cada ruta, asegurando que la información sea clara y esté sincronizada con los horarios de salida.
Actores:	Administrador

Nombre:	**[XS-36] Gestión de Rutas**
Descripción:	**Como** Administrador, **Quiero** organizar de manera eficiente los itinerarios y precios de los servicios entre diferentes origenes y destinos. La herramienta debe brindarme la capacidad y gestionar de manera centralizada toda la información relevante. **Para** optimizar la planificación y operación de la flota.
Criterios de Aceptación:	1. **Registro de Rutas:** • Como usuario, deseo poder registrar cada ruta, especificando claramente los puntos de origen y destino, así como cualquier parada intermedia que sea relevante. 2. **Configuración de Tarifas:**

	• Como usuario, necesito la capacidad de establecer tarifas específicas para cada combinación origen-destino, considerando diferentes categorías de pasajeros en base a las directrices del ant.
Actores:	No aplica

Nombre:	**[XS-21] Gestión plan de cuentas contables.**
Descripción:	**Como** contador, **Quiero** simplificar mis tareas contables diarias. Necesito realizar fácilmente acciones como crear, listar, editar y eliminar cuentas, además de generar informes rápidos, **Para** tener una visión clara de mi situación financiera. La interfaz debe ser intuitiva y segura.
Criterios de Aceptación:	1. **Creación de cuentas contables** • Como usuario contador quiero poder ingresar las cuentas con los parámetros necesarios que incluye:  2. **Actualizar y eliminar cuentas contables.** • Deseo poder editar la información de una cuenta contable existente, así como eliminarlo si es necesario. 3. **Consulta del Plan de Cuentas** • Como contador, deseo poder buscar por nombre de cuenta y encontrarla fácilmente lo requerido
Actores:	contador

Nombre:	**[XS-22] Gestión Asientos contables.**
Descripción:	**Como** Contador, **Quiero** un sistema que permita la gestión eficiente de asientos contables, **Para** mantener un seguimiento preciso de las transacciones financieras y generar informes financieros fiables.
Criterios de Aceptación:	1. **Ingreso de Asiento Contable:** o Como contador, quiero poder ingresar un asiento contable con información detallada, incluyendo fecha, cuentas afectadas, descripción y montos. 2. **Validación de Equilibrio:** o Como contador, quiero que el sistema valide automáticamente que el asiento contable esté equilibrado, es decir, que la suma de los débitos sea igual a la suma de los créditos. 3. **Edición y Eliminación de Asientos:**

	○ Como contador, quiero poder editar la información de un asiento contable existente, así como eliminarlo si es necesario. 4. **Consulta de Historial de Asientos:** ○ Como contador, quiero poder consultar un historial completo de asientos contables, filtrando por fechas y cuentas involucradas. 5. **Control de Acceso:** ○ Como contador, quiero que el sistema tenga un control de acceso seguro para garantizar que solo usuarios autorizados puedan realizar modificaciones en los asientos contables.
Actores:	Contador

Nombre:	**[XS-23] Reporte de Mayores.**
Descripción:	**Como** contador, **Quiero** acceder a un reporte de mayores para obtener un resumen detallado de las transacciones contables en una cuenta específica. **Para** analizar y auditar la información financiera de manera eficiente.
Criterios de Aceptación:	1. **El reporte debe mostrar un resumen claro y detallado** de las transacciones, incluyendo fecha, descripción, débito, crédito y saldo acumulado. 2. **El sistema debe garantizar la precisión matemática de los cálculos**, asegurando que el saldo acumulado sea correcto. 3. **La generación del reporte debe ser eficiente,** incluso para cuentas con un gran volumen de transacciones.
Actores:	contador

Nombre:	**[XS-45] Implementar Facturación Electrónica.**
Descripción:	No Aplica
Criterios de Aceptación:	El sistema una vez que genera el boleto tiene que enviar la factura electrónica acorde a los lineamientos establecidos por el Servicio de Rentas Internas (SRI)
Actores:	No aplica

Nombre:	**[XS-32] Venta de boletos.**
Descripción:	**Como** oficinista, **Quiero** poder vender boletos de viaje utilizando los datos del cliente obtenidos del registro civil. **Para** facilitar el proceso de compra y mejorar la experiencia del cliente al proporcionar un servicio más personalizado.
Criterios de Aceptación:	1. Como empleado, deseo obtener los datos desde del registro civil para buscar y obtener los datos del cliente, como nombre y documento de identidad. 2. Debo poder ingresar los datos del cliente obtenidos del registro civil en el sistema de venta de boletos de transporte público. 3. El sistema de venta de boletos debe ser capaz de almacenar y asociar los datos del cliente con la compra del boleto de transporte público.

	4. Una vez que los datos del cliente estén asociados con la compra, el sistema debe generar automáticamente el boleto de transporte con la información proporcionada. 5. El cliente debe recibir un comprobante de compra que incluya los detalles del viaje y los datos personales obtenidos del registro civil.
Actores:	Oficinista

3 GLOSARIO DE TÉRMINOS:

Domain Driven Design (DDD): Método para el desarrollo de software que se enfoca en el diseño del modelo de dominio basado en el negocio real de la aplicación.

Autenticación Segura: Proceso de verificar la identidad de un usuario de forma segura y confiable.

Control de Acceso Basado en Roles: Sistema que restringe el acceso a recursos y funcionalidades en función de los roles asignados a los usuarios.

Token JWT (JSON Web Token): Estándar abierto que define una forma compacta y autónoma de transmitir información de forma segura entre dos partes como un objeto JSON.

HTTPS: Protocolo de transferencia de hipertexto seguro que garantiza una comunicación segura a través de Internet.

Pruebas Unitarias: Evaluación de cada componente individual de un software de forma aislada.

Mecanismos de Recuperación Segura: Procesos para recuperar cuentas de forma segura, como restablecimiento de contraseña mediante correo electrónico o autenticación multifactor.

Flota: Conjunto de vehículos que pertenecen a una empresa de transporte.

Configuración de Asientos: Distribución y disposición de los asientos en un vehículo.

Frecuencias de Salida y Llegada: Horarios y frecuencias en los que los vehículos salen y llegan a sus destinos.

Tarifas: Costos asociados a viajes en transporte público.

Plan de Cuentas Contables: Estructura que organiza y clasifica las cuentas financieras de una empresa.

Reporte de Mayores: Resumen detallado de transacciones contables en una cuenta específica.

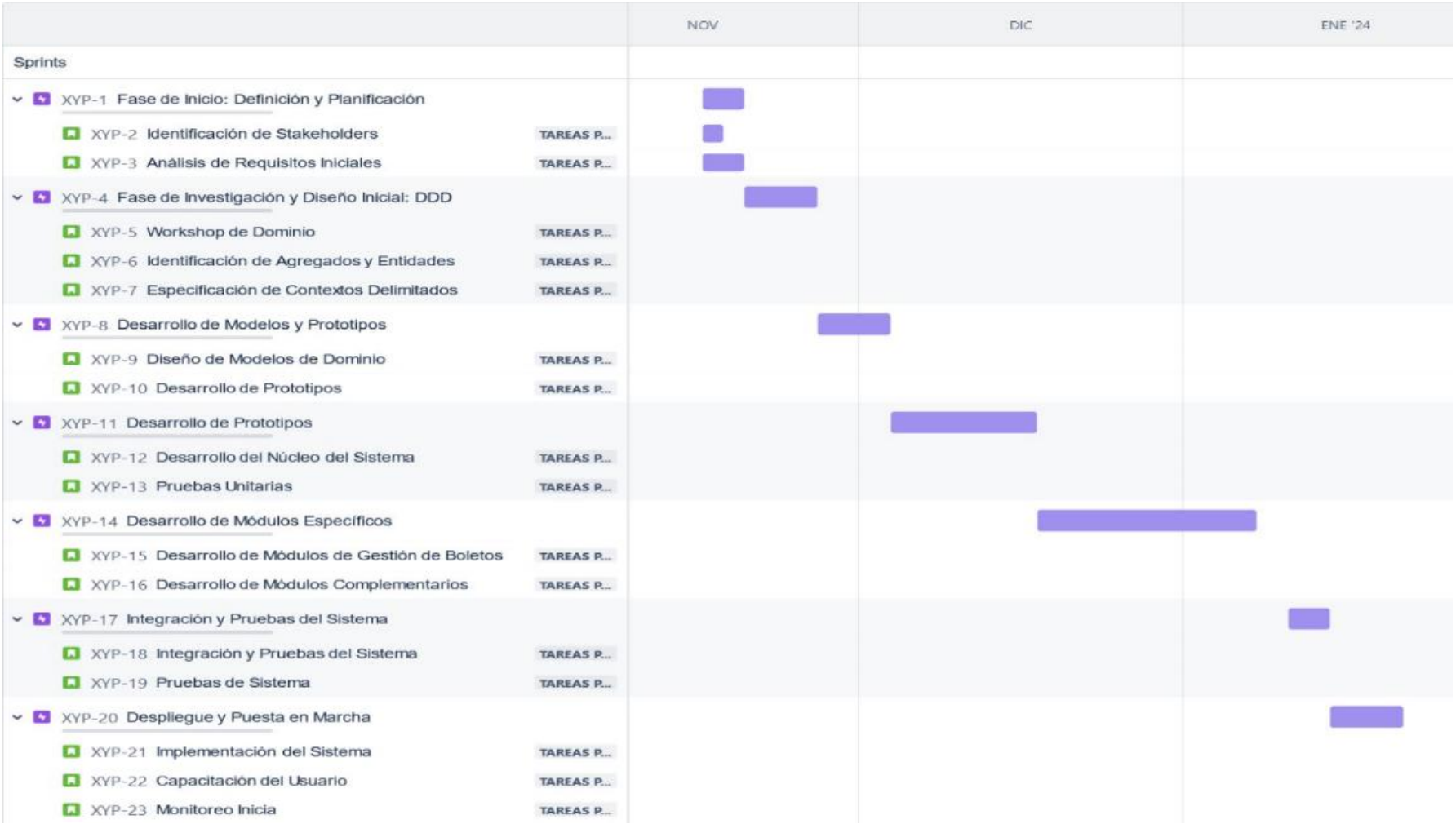
Facturación Electrónica: Emisión de facturas de forma electrónica siguiendo los lineamientos legales establecidos.

Venta de Boletos: Proceso de venta de boletos de viaje a clientes.

Oficinista: Empleado encargado de vender boletos en la oficina de la empresa de transporte.

ANEXO 2

Cronograma (Gantt)


NOV
DIC
ENE '24
Sprints
XYP-1 Fase de Inicio: Definición y Planificación
XYP-2 Identificación de Stakeholders
XYP-3 Análisis de Requisitos Iniciales
XYP-4 Fase de Investigación y Diseño Inicial: DDD
XYP-5 Workshop de Dominio
XYP-6 Identificación de Agregados y Entidades
XYP-7 Especificación de Contextos Delimitados
XYP-8 Desarrollo de Modelos y Prototipos
XYP-9 Diseño de Modelos de Dominio
XYP-10 Desarrollo de Prototipos
XYP-11 Desarrollo de Prototipos
XYP-12 Desarrollo del Núcleo del Sistema
XYP-13 Pruebas Unitarias
XYP-14 Desarrollo de Módulos Específicos
XYP-15 Desarrollo de Módulos de Gestión de Boletos
XYP-16 Desarrollo de Módulos Complementarios
XYP-17 Integración y Pruebas del Sistema
XYP-18 Integración y Pruebas del Sistema
XYP-19 Pruebas de Sistema
XYP-20 Despliegue y Puesta en Marcha
XYP-21 Implementación del Sistema
XYP-22 Capacitación del Usuario
XYP-23 Monitoreo Inicia
TAREAS P...

ANEXO 3

Presupuesto Ejecutado

N	Recurso	Cantidad	Precio Unitario	Total
1	Computador I7 16GB RAM / mes	4	$41,66	$166,64
2	Cluster Kubernetes / mes	4	$36,00	$144,00
3	vps 2 nucleos, 4 RAM, 40GB x 4mes	2	$60,00	$120,00
4	Materiales Varios	1	$400,00	$400,00
5	Capacitación / mes	4	$29,00	$116,00
Total				$946,64

ANEXO 4

Carta de aceptación de la organización donde se realizó el trabajo de integración curricular.

Cooperativa de Transporte Interprovincial
"SAN PEDRITO"
Legalizada Mediante Acuerdo Ministerial N° [illegible]
RUC: 0290029821001
PROV BOLÍVAR - GUARANDA - ECUADOR

Guanujo, a 16 de noviembre del 2023
OF-10369-CTSP

Señores
Melanin Vanessa Ganan Illvay
Mesias Eduardo Bonilla Guastay
ESTUDIANTES DE LA UNIVERSIDAD ESTATAL DE BOLIVAR
Presente. -

De mi consideración:

Reciban un cordial y atento saludo.

Yo, Ing. **LARA REAL ÁNGEL EDUARDO,** en calidad de **Representante Legal De la Cooperativa De Transportes "San Pedrito"**, me permito **AUTORIZAR** que en nuestra Empresa procedan con la ejecución de su proyecto de Integración curricular denominado: **"DESARROLLO DE UN SISTEMA INFORMATICO PARA LA GESTIÒN Y VENTA DE BOLETOS EN LA COOPERATIVA DE TRANSPORTES "SAN PEDRITO" APLICANDO DOMAIN DRIVEN DESIGN(DDD)".**

Agradeciéndoles por la atención a la presente, me suscribo.

Atentamente,

Cooperativa de Transportes "SAN PEDRITO"
GERENCIA
Ing. Angel Lara
Guanujo - Prov. Bolívar

Ing. Angel Lara Real
GERENTE-CTSP

ANEXO 5

Instrumentos de recopilación de datos “guion entrevista”.

Guion de la Entrevista

Objetivo: Desarrollar un sistema informático para la gestión y venta de boletos en la Cooperativa de Transportes “San Pedrito” aplicando Domain Driven Design (DDD).

- ¿Cuáles son los principales problemas que enfrenta al vender boletos y asignar asientos?
- ¿Cómo la falta de sincronización entre la venta de boletos y la asignación de asientos afecta su eficiencia diaria al no poder evitar ventas duplicadas de asientos con otros puntos de venta?
- ¿Qué funcionalidades adicionales o mejoras en el proceso de venta de boletos y asignación de asientos consideraría beneficiosas?
- ¿Cómo describiría la eficiencia actual en la generación de facturas y seguimiento de transacciones?
- ¿Qué información financiera específica considera crucial para mejorar la toma de decisiones?
- ¿Sería de su agrado y beneficio que se desarrolle un nuevos plataforma web, con funcionalidades más concretas?

ANEXO 6

Manual de usuario

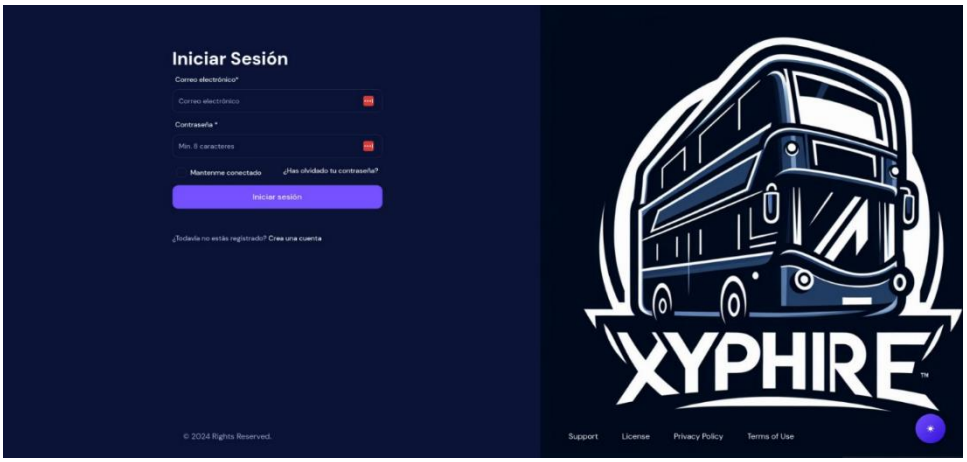
Manual de Usuario

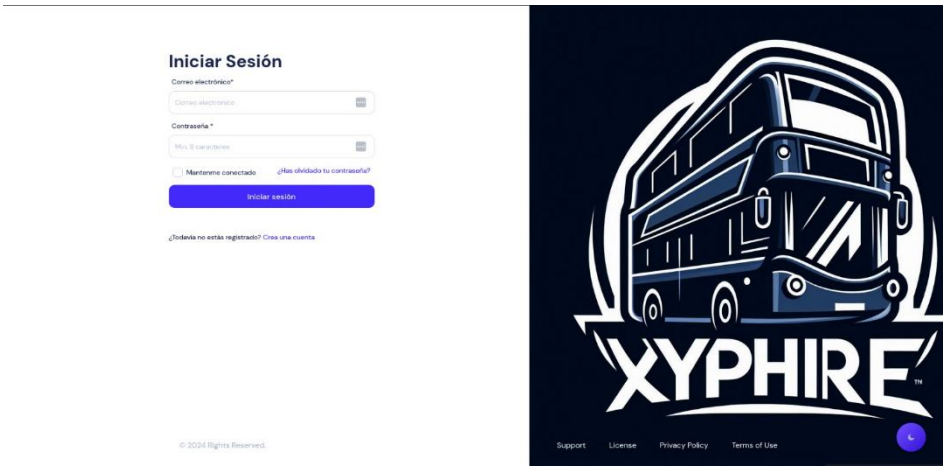
Sistema de Venta de Boletos

XYPHIRE CSP 1.0

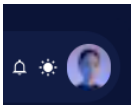
Sistema de Venta de Boletos (XYPHIRE**)**

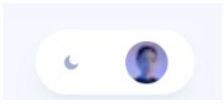
1. Ingreso al sistema de venta de boletos. Para acceder a la plataforma debe tipear la https://www.xyphire.com/ y se mostrará la página principal del sistema XYPHIRE
2. Para iniciar sesión se debe ingresar el usuario y la contraseña exclusiva creado para xyphire, al hacer clic en el botón "Acceder", se ingresará a la plataforma y mostrará la siguiente pantalla en modo oscuro por defecto, para cambiar al modo claro puede darle clic en el icono del sol ubicado en la parte inferior derecha de la pantalla.



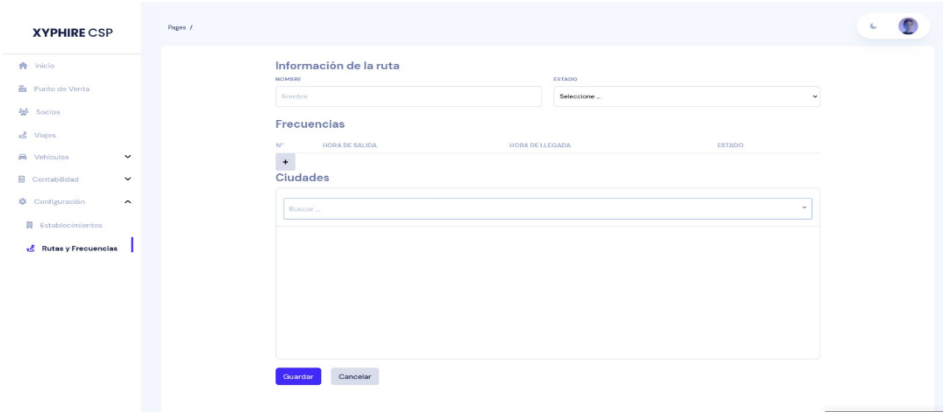


3. Una vez que inicia sesión puede activar o desactivar el modo oscuro en la barra superior sección en el icono de la luna o el sol.





Pantalla en modo claro

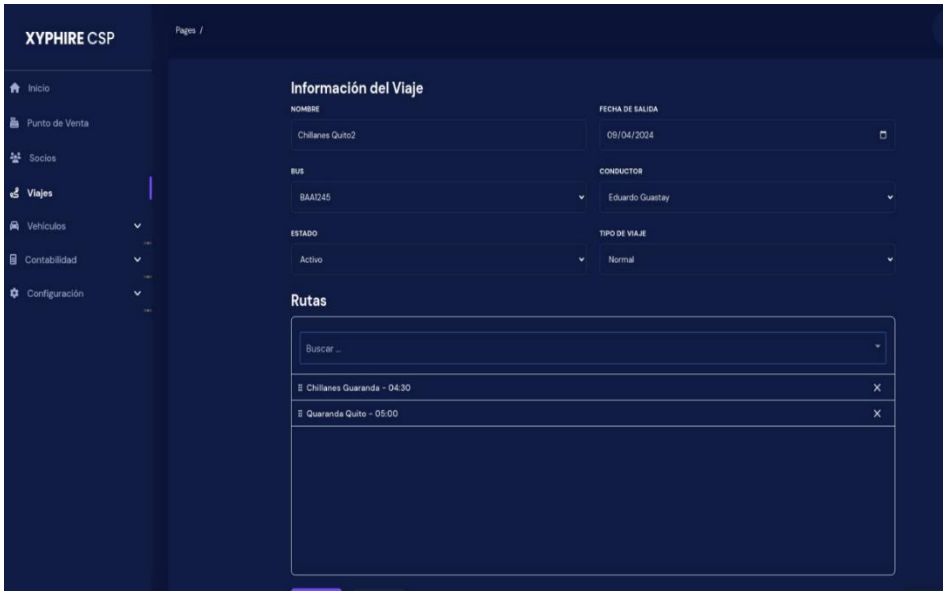


Pantalla en modo Oscuro

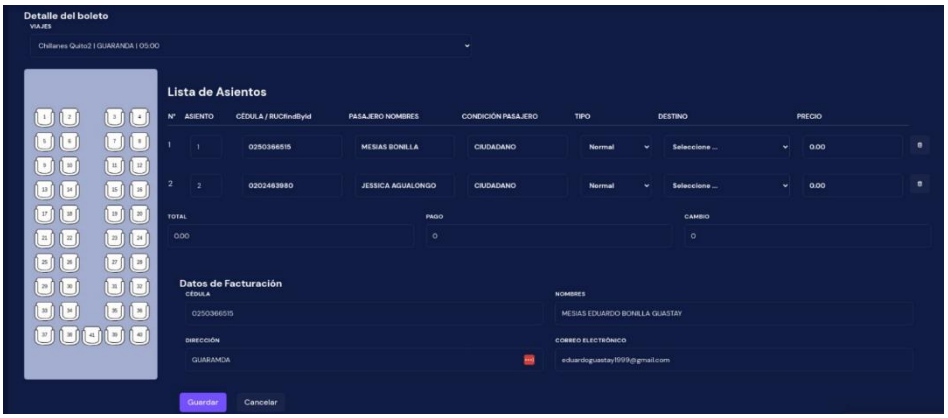
4. Una vez que haya ingresado al sistema, se mostrará la siguiente ventana donde se podrá observar el menú de navegación con los servicios de la cooperativa disponibles para casa usuario.

5. En la primera sección de "**Socios"** podrán encontrar los socios de la cooperativa e insertar o eliminar si lo requiere.

6. En la segunda sección de **“Viajes”** hace referencia para la venta de boletos donde se puede elegir rutas y frecuencias y el asiento que el cliente obtendrá.



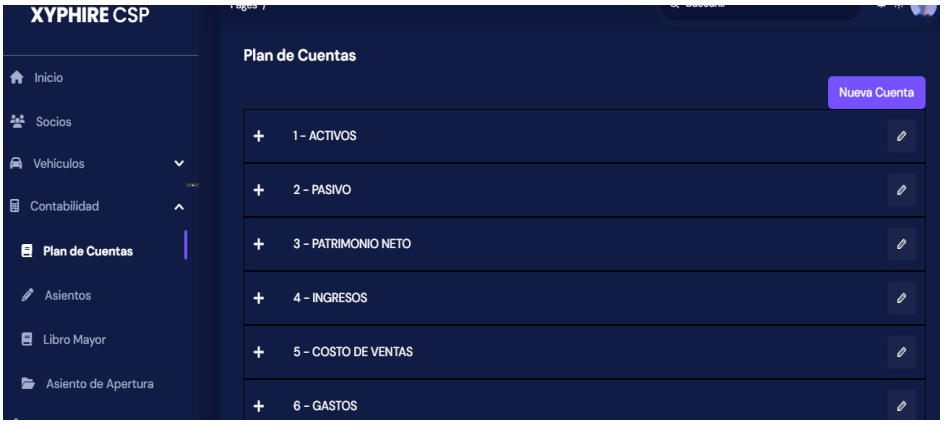
7. El módulo POS permite generar el ticket de viaje para los pasajeros, al momento de realizar la selección del destino se observa la siguiente pantalla en la que el se puede seleccionar los asientos y estos se agregan a la tabla en la que se puede ingresar la identificación de los pasajeros y este obtendrá los datos almacenados previamente.



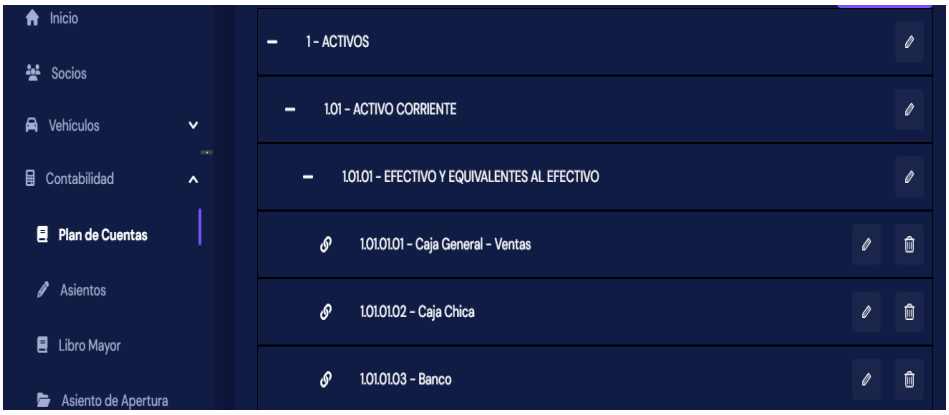
8. En la tercera sección de **“Vehículos”** podrán encontrar características del Bus donde se contempla dos botones que contiene:
    - En la sección denominado **“Carrocerías”** añada las siguientes características requeridas para el sistema

- En la sección denominada **“Buses** “añada las siguientes características requeridas para ser identificado a que socio pertenece.

9. En la tercera sección **“Contabilidad”** podrán encontrar fundamentos esenciales del proceso contable que requiere la cooperativa.
    - En la sección de **“Plan de Cuentas”** de manera organizada se desglosa las cuentas financieras que requiere el proceso de contabilidad que la cooperativa utiliza para registrar sus transacciones financieras.

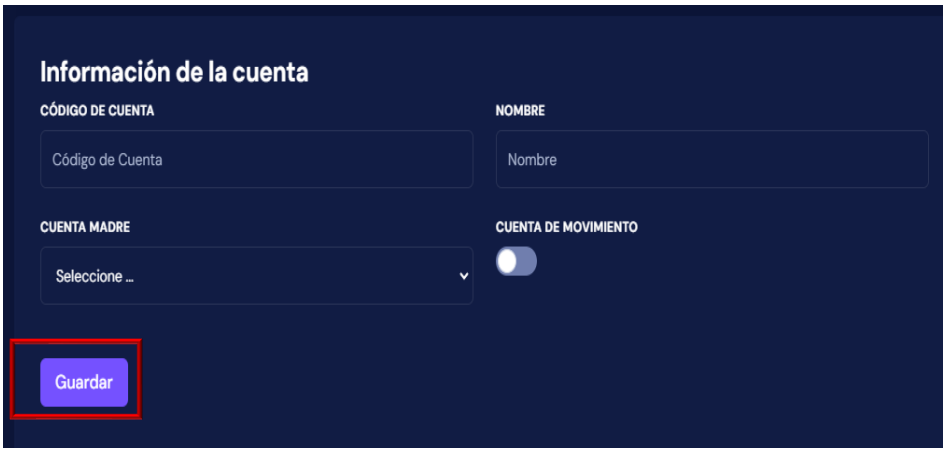


- Al hacer clic izquierdo en el botón (+) se desglosa en diferentes opciones según las necesidades del usuario.



- Crear una nueva cuenta haga clic en el siguiente botón y se desglosara la siguiente pantalla donde insertara los datos.

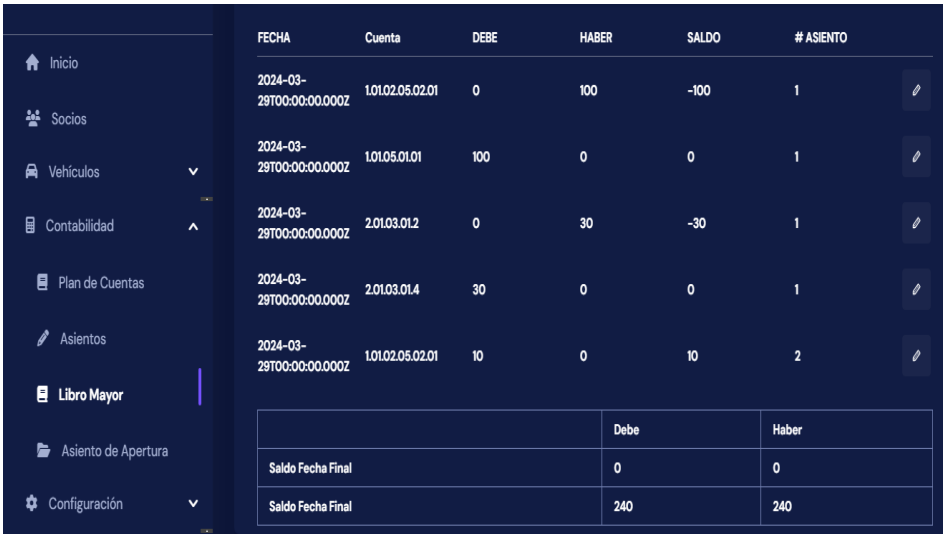
- Después de insertar los datos para la creación de una cuenta haga clic en el botón Guardar



- Una cuenta creada o un grupo puede ser editada o eliminada con los siguientes botones.

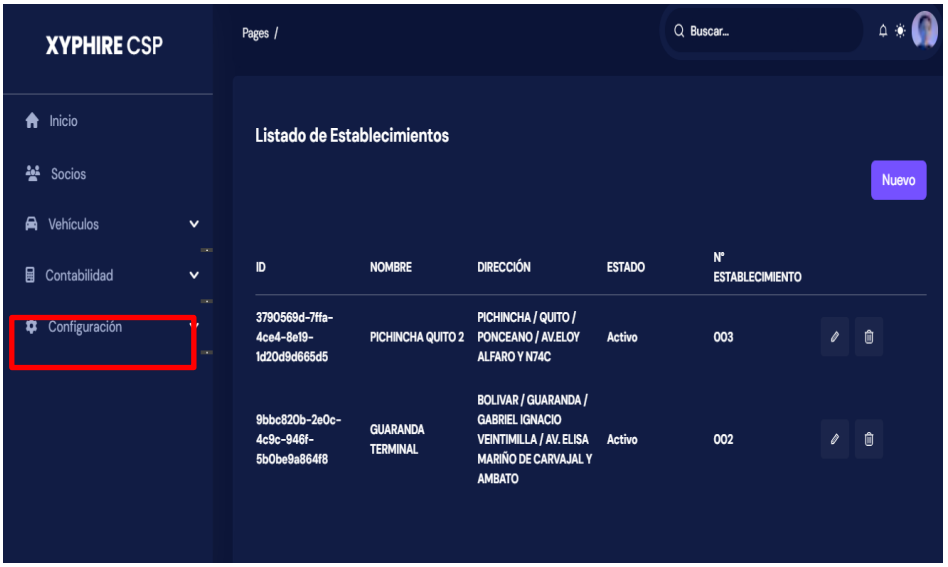
- En la sección denominada **"Asientos"** refleja las entradas que se realizan en el libro mayor de una empresa para registrar transacciones financieras específicas. Estas partes reflejan la forma en que los activos, pasivos, capital y cuentas de ingresos y gastos se ven afectados por la transacción. También cuenta con la opción de crear, editar y eliminar

- En la sección denominada **"Libro Mayor"** Contiene información como el número y la descripción de las cuentas, la fecha de las transacciones, los montos de débito y crédito, y los saldos acumulados.



- En la sección denominada **"Asiento de Apertura"** donde se contempla el primer registro contable al iniciar un nuevo período. Transfiere los saldos de cuentas del período anterior al nuevo, asegurando que los registros financieros comiencen actualizados y precisos.

10. En la cuarta sección **"Configuración"** al hacer clic se desglosa dos secciones donde se muestra dos botones



- En la sección denominada **“Establecimientos”** puedes insertar la ubicación de las sucursales de la cooperativa.

- En la sección denominada **“Rutas y frecuencias”** las direcciones a donde el bus va a transportarse y en que horarios.

ANEXO 7

Certificado Anti-plagió

UEB
UNIVERSIDAD ESTATAL DE BOLIVAR

FACULTAD DE CIENCIAS ADMINISTRATIVAS, GESTIÓN EMPRESARIAL E INFORMÁTICA

ING. DANILO GEOVANNY BARRENO NARANJO EN CALIDAD DE DIRECTOR DEL TRABAJO DE INTEGRACIÓN CURRICULAR,

CERTIFICA

Que el trabajo de integración curricular denominado **"DESARROLLO DE UN SISTEMA INFORMÁTICO PARA LA GESTIÓN Y VENTA DE BOLETOS EN LA COOPERATIVA DE TRANSPORTES "SAN PEDRITO" APLICANDO DOMAIN DRIVEN DESIGN (DDD)"**, presentado por los señores estudiantes **BONILLA GUASTAY MESÍAS EDUARDO** con **C.I. 0250366515** y **GANAN ILVAY MELANIN VANESSA** con **C.I. 0605584358**, estudiantes de la **carrera de Software** pasó el análisis de coincidencia no accidental en la herramienta TURNITIN, reflejando un **porcentaje de similitud del 4%**, como se puede evidenciar en el documento adjunto.

Guaranda, 15 de abril del 2024

Atentamente,

Ing. Danilo Barreno
Director

Dirección: Av. Ernesto Che Guevara y Gabriel Secaira
Guaranda-Ecuador
Teléfono: (593) 3220 6059
www.ueb.edu.ec

NOMBRE DEL TRABAJO

Proyecto2042024-2.pdf

RECUENTO DE PALABRAS

14374 Words

RECUENTO DE CARACTERES

98122 Characters

RECUENTO DE PÁGINAS

90 Pages

TAMAÑO DEL ARCHIVO

2.8MB

FECHA DE ENTREGA

Apr 23, 2024 10:35 AM GMT-5

FECHA DEL INFORME

Apr 23, 2024 10:37 AM GMT-5

4% de similitud general

El total combinado de todas las coincidencias, incluidas las fuentes superpuestas, para cada base de datos.

- 3% Base de datos de Internet
- 1% Base de datos de publicaciones
- Base de datos de Crossref
- Base de datos de contenido publicado de Crossref
- 3% Base de datos de trabajos entregados

Excluir del Reporte de Similitud

- Material bibliográfico
- Material citado
- Material citado
- Coincidencia baja (menos de 15 palabras)
- Fuentes excluidas manualmente
- Bloques de texto excluidos manualmente

ANEXO 8

Acta entrega recepción del sistema informático.

Cooperativa de Transporte Interprovincial "SAN PEDRITO"

Legalizada Mediante Acuerdo Ministerial N° [illegible]
RUC: 0290029821001
PROV. BOLÍVAR - GUARANDA - ECUADOR

Guanujo, 23 de abril del 2024

Señores
Melanin Vanessa Ganan Ilvay
Mesias Eduardo Bonilla Guastay
EGRESADOS UNIVERSIDAD ESTATAL DE BOLIVAR
Presente -

De mi consideración:

Reciban un cordial y atento saludo

Yo, **Ing. LARA REAL ANGEL EDUARDO**, en calidad de **Representante Legal de la Cooperativa de Transportes "San Pedrito"**, **CERTIFICO** que el proyecto de Integración curricular denominado "**"DESARROLLO DE UN SISTEMA INFORMÁTICO PARA LA GESTIÓN Y VENTA DE BOLETOS EN LA COOPERATIVA DE TRANSPORTES "SAN PEDRITO" APLICANDO DOMAIN DRIVEN DESIGN (DDD)**" cumple con los requisitos establecidos por esta organización.

Agradeciéndoles por la atención a la presente, me suscribo

Atentamente,

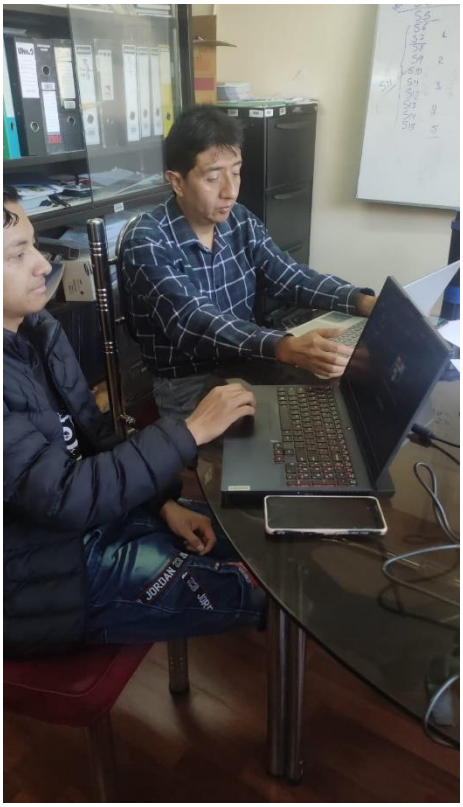
Cooperativa de Transportes "SAN PEDRITO"
GERENCIA
Ing. Angel Lara
Guanujo - Prov. Bolívar

Ing. Angel Lara Real
GERENTE-CTSP

ANEXO 9

Fotografías.

Reunión con el director y pares académicos.



Reunión con representantes de la Coop. San Pedrito.