UNIVERSIDAD DEL VALLE DE GUATEMALA
CC3090 - Ingeniería de Software 1
Sección 30
Ing. Cristián Rafael Muralles Salguero

Sistema de Control Financiero para la CSPI

Bryan Alberto Martínez Orellana, 23542
Adriana Sophia Palacios Contreras, 23044
Pedro Rubén Avila Cofiño, 23089
Diego Javier López Reinoso 23747

GUATEMALA, 17 de marzo de 2025

Resumen

Este proyecto busca desarrollar un sistema digital para la Corporación de Servicios Profesionales e Inmobiliarios con el objetivo de optimizar la gestión y validación de cuotas, préstamos, moras e intereses respectivos. Actualmente, la cooperativa maneja su información de pagos y préstamos de manera manual a través de WhatsApp y Excel, lo que genera errores, retrasos y afecta su efectividad a la hora de llevar a cabo estos procesos operativos.

La necesidad de este sistema radica en la creciente expansión de la corporación, lo que hace insostenible el manejo actual de la información. Se requiere una solución que centralice los datos, automatice cálculos de cuotas y moras, y facilite la validación de pagos y la gestión de préstamos.

Los objetivos principales son: automatizar el control de pagos de cuotas ordinarias y extraordinarias (reduciendo errores y tiempos de validación), optimizar la gestión de préstamos, incorporando el seguimiento en tiempo real y generación automática de pagarés, y mejorar la comunicación entre socios y los miembros de la directiva al presentar depósitos. Este sistema garantizará transparencia, eficiencia y un crecimiento sostenible para la corporación.

Introducción

El presente proyecto está dirigido a la Corporación de Servicios Profesionales e Inmobiliarios (CSPI), una organización que gestiona cuotas, préstamos y otros procesos financieros para sus socios. La CSPI actualmente administra sus finanzas mediante herramientas manuales como WhatsApp y Excel, lo que genera errores, pérdida de información y una carga operativa significativa para la directiva. Sus principales procesos incluyen la validación de pagos, la gestión de cuotas ordinarias y extraordinarias, el manejo de préstamos y moras, y la generación de reportes financieros, áreas clave en las que se enfocará la solución a desarrollar.

El proyecto consiste en el desarrollo de un sistema digital de control financiero que permitirá a la CSPI gestionar de manera eficiente y automatizada los pagos de cuotas, la validación de depósitos, la administración de préstamos y el cálculo de moras. Actualmente, la falta de un sistema centralizado provoca desorganización, retrasos en la validación de pagos y errores en la asignación de moras, lo que genera insatisfacción entre los socios y sobrecarga a los miembros de la directiva. La solución propuesta buscará optimizar estos procesos, mejorar la comunicación entre la directiva y los socios, y brindar acceso a información financiera en tiempo real, garantizando una gestión más eficiente, transparente y escalable.

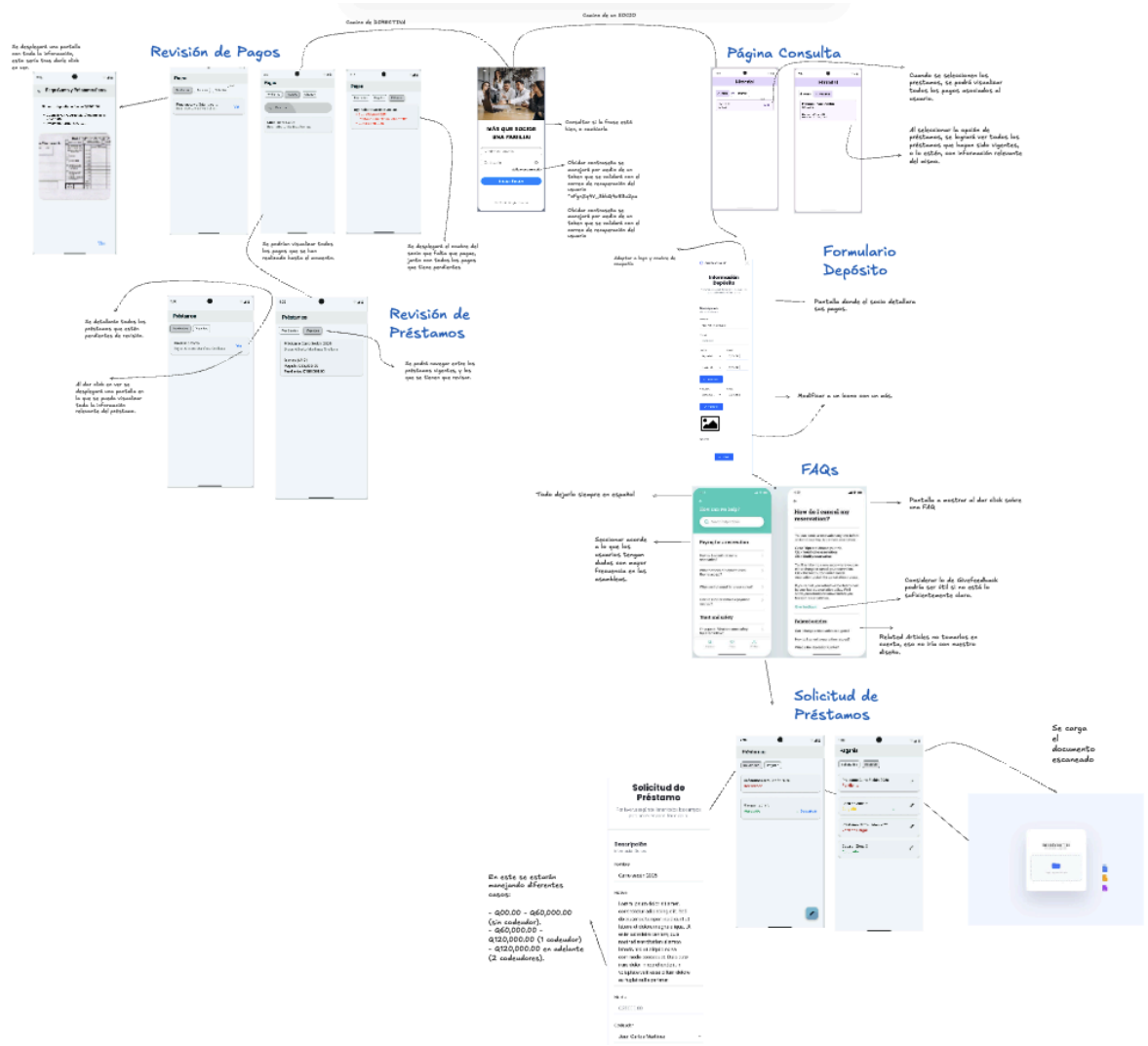
El objetivo general de este informe es documentar el análisis, diseño y planificación del desarrollo de un sistema digital de control financiero para la CSPI. Este sistema tiene como finalidad automatizar la gestión de pagos, préstamos y moras, reduciendo la carga operativa de la directiva y proporcionando una herramienta eficiente y transparente para los socios. A través de este informe, se presentará el contexto actual de la corporación, las problemáticas identificadas y la propuesta de solución, justificando la necesidad de modernizar los procesos financieros. Además, se detallarán los hallazgos obtenidos en la fase de investigación, incluyendo la recopilación de información a través de entrevistas y análisis de usuarios, lo que permitirá estructurar un plan de desarrollo sólido y alineado con las necesidades de la organización.

De manera específica, este informe profundizará en el análisis de los procesos actuales de la CSPI en la gestión de cuotas, préstamos y validación de pagos, para poder tomar decisiones técnicas y de diseño. Asimismo, se definirán los requerimientos funcionales y no funcionales del sistema digital propuesto, los cuales establecerán las bases para su diseño e implementación. Además, se documentarán los hallazgos obtenidos en la fase de prototipado, así como las decisiones tomadas en cuanto a la arquitectura y persistencia de datos, justificando la elección de las tecnologías a utilizar. Todo este análisis permitirá asegurar que el sistema desarrollado responda a las necesidades identificadas, optimizando los procesos administrativos y fortaleciendo la gestión financiera de la organización.

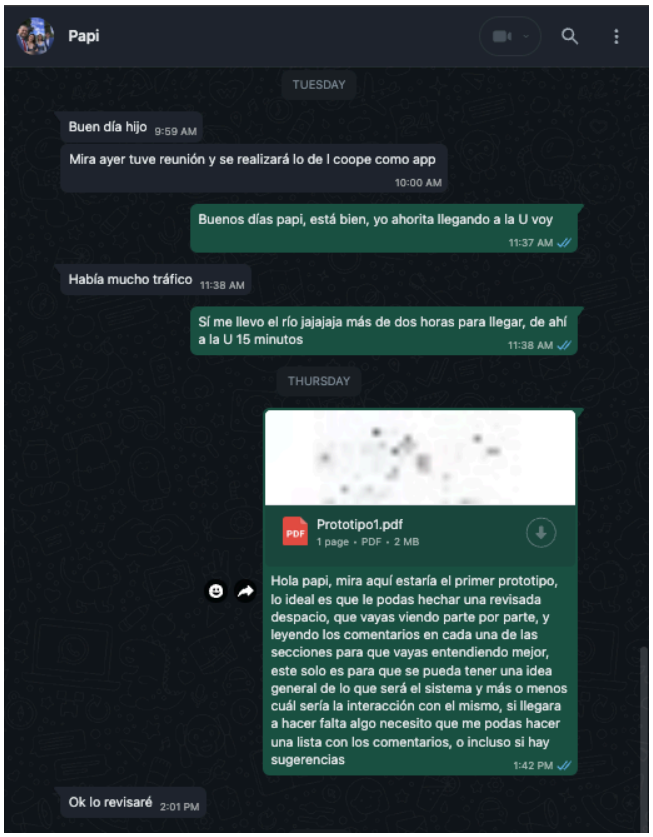
Prototipos

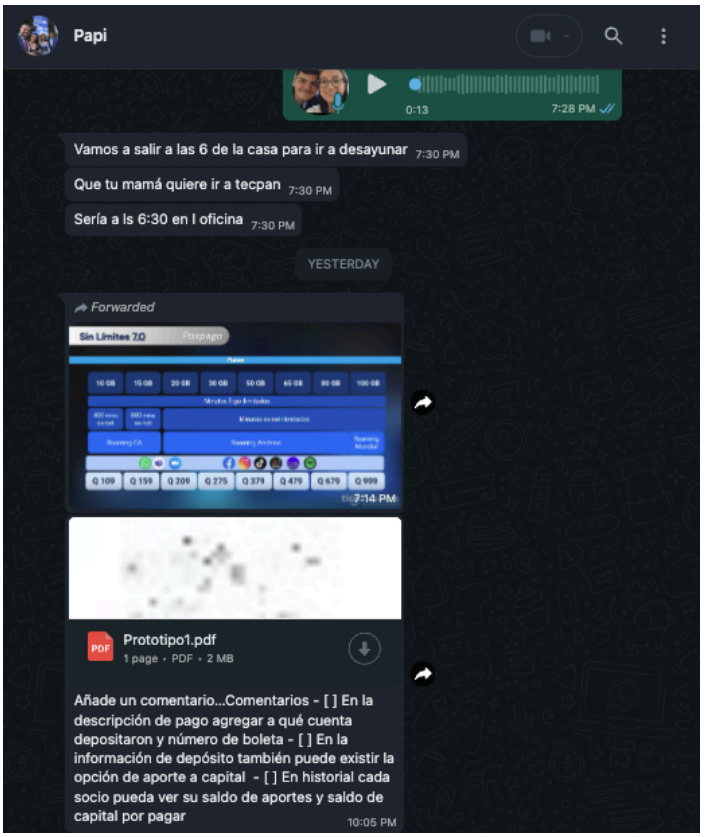
Primer Prototipo

[Link a Drive]



Evidencia de Interacción Primer Prototipo



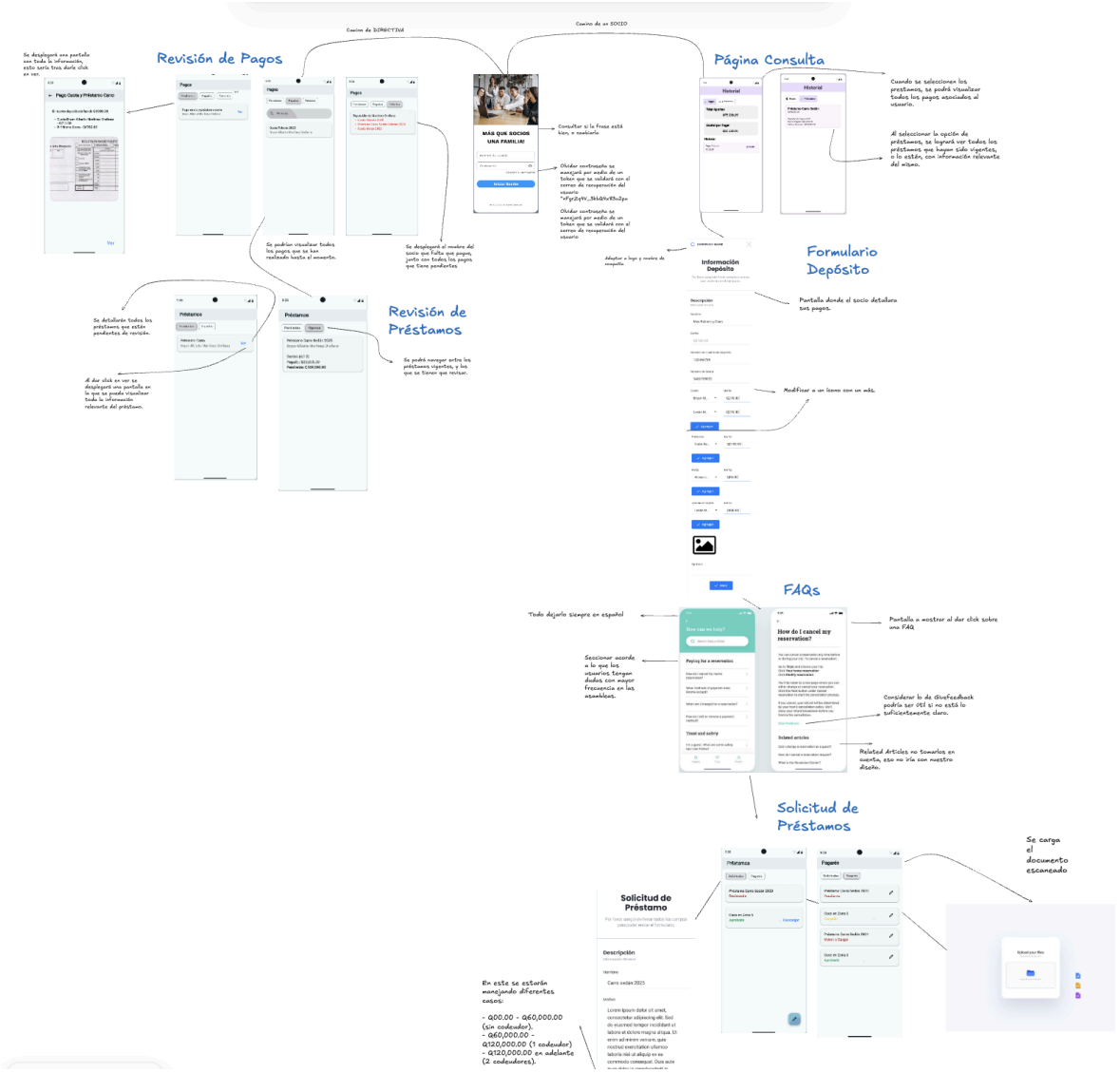


Sugerencias:

- En descripción de pago agregar a qué cuenta depositaron y el número de boleta.
- En la información de depósito también puede existir la opción de aporte a capital.
- En historial, cada socio puede ver su saldo de aportes y saldo de capital por pagar.

Segundo Prototipo

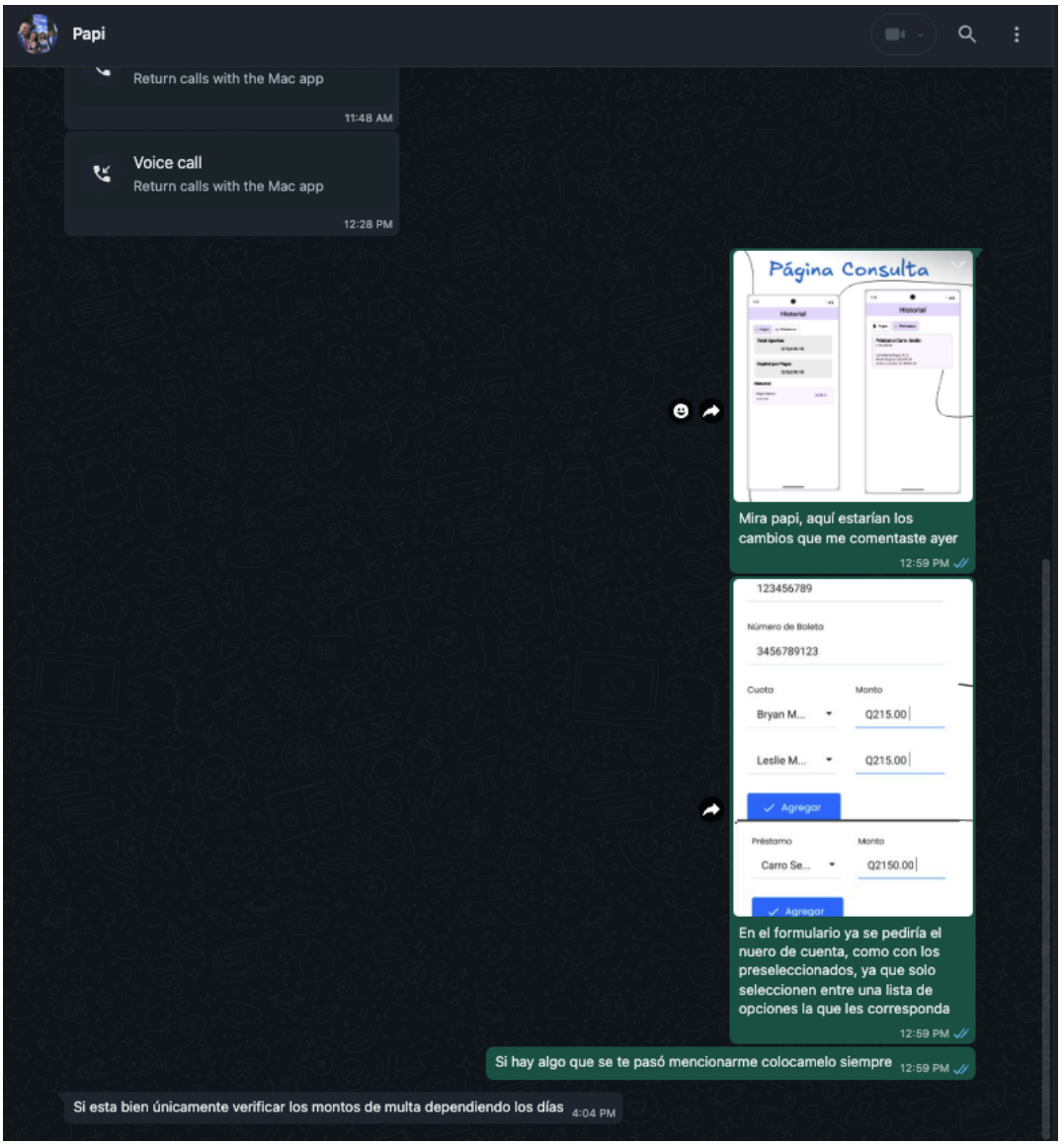
[Link a Drive]



Lista de cambios:

- En la sección de página de consulta, se modificó la vista de historial de pagos para que también se viera reflejado que se iba a poder visualizar el total de mis aportes, y el capital por pagar.
- En la sección de consultas, en el historial de pagos, se agregó que en cada carta también se mostrará el saldo por cancelar.
- En el formulario de depósito se modificó para que se incluyeran los campos de pago de multa o aporte a capital. Además, al inicio de este se estará solicitando el número de cuenta de depósito, y el número de boleta de la persona.

Evidencia de Interacción Segundo Prototipo



Sugerencias:

- Tomar en cuenta que internamente las multas se calcularán de forma automática (esto no tiene que ver con el prototipo, pero sí es algo que ocurre de forma interna).

Análisis de Requisitos Funcionales y No Funcionales Acorde a Prototipos

Es importante mencionar que la lista completa de recursos funcionales se encuentra en otra sección de este mismo documento, aquí solamente se mencionarán los que correspondan a las sugerencias presentadas.

Requisito	**Tipo**
A la hora de poder consultar el historial de préstamos, mostrar el monto total del préstamo, la cantidad de pagos que lleva la persona, el monto pagado, y el saldo a cancelar.	Requisito funcional.
Al presentar un pago, también se pueden	Requisito funcional.

hacer aportes a capital si el socio así lo desea.	
Al presentar un pago, como requerimiento, también debemos solicitar el número de boleta, y la cuenta a la que depositada.	Requisito funcional.
En el historial de pagos, el socio debería poder ver su saldo de aportes.	Requisito funcional.
Cálculo de multas de forma automática, 5 quetzales por día de atraso en las cuotas, y 5 porciento como máximo al atrasarse con un pago de préstamo.	Requisito funcional.

Requisitos Funcionales:

- Log In:
    - Ambos
        - Manejo de autenticación de usuarios -> Log In
        - Restablecer contraseña en caso de que se olvide -> Log In

- Pagos
    - Socio
        - Al enviar la información de un pago, que esta se encuentre encriptada -> Enviar comprobante de pago
        - Registrar cada envío de pago en la base de datos -> Enviar comprobante de pago
        - Que cada socio pueda visualizar el estado de su comprobante de pago enviado actualmente -> Enviar comprobante de pago
        - A la hora de completar el formulario de pago, que el comprobante se pueda exportar del lado del cliente -> Enviar comprobante de pago
        - Poder realizar notificaciones de que deben realizar su pago, en caso de que no se haya registrado el que corresponde al mes actual, en la fecha límite, que sería el cinco de cada mes -> Enviar comprobante de pago
        - Al estar llenando el formulario de pago, que se valide que el monto que se está declarando sea distribuido entre cuotas y pago de préstamo -> Enviar comprobante de pago
        - Poder enviar más de un comprobante de pago por mes, ya que algunos usuarios pagan primero sus cuotas y luego los préstamos -> Enviar comprobante de pago
        - Al presentar un pago, se pueden hacer aportes a capital si el socio así lo deseara. -> Enviar comprobante de pago
        - Al estar presentando un pago, es necesario que el socio también coloque el número de cuenta al que depositó y su número de boleta. -> Enviar comprobante de pago.
        - En cuanto a los pagos, se debe habilitar el pago de una multa, la cual debe ser calculada automáticamente por cada día de atraso con una

cuota, y un máximo de 5% si es atraso de préstamo. -> Pago de multas.

- Directiva
  - Que en el momento en que se registró un pago por parte de un socio, que en el sistema de los miembros de la directiva se pueda visualizar dicho pago -> Validar pagos de socios
  - A la hora de revisar un pago presentado por un socio, que el miembro de la directiva pueda validar si está bien hecho, de ser, así aprobarlo y que se registre en la base de datos y el sistema, y en caso de que no se admita, poder enviar un mensaje dando retroalimentación de por qué no se aceptó -> Validar pagos de socios

- Préstamos
  - Socio
    - Enviar información de solicitud de préstamos, y que el monto de este se encuentre encriptado -> Solicitar préstamo
    - Que el formulario que se esté llenando sea dinámico acorde al monto que se está solicitando -> Solicitar préstamo
    - Que dentro del formulario se pueda listar la información de contacto del socio en caso de que la directiva requiera datos adicionales. -> Solicitar préstamo
    - Tener retroalimentación de sí, el envío se realizó exitosamente -> Solicitar préstamo
    - Registrar solicitud de préstamo en la base de datos, y que se reconozca en el sistema -> Solicitar préstamo
    - Poder visualizar si mi préstamo fue aprobado o rechazado, en caso de que se apruebe poder descargar el pagaré -> Envío de pagaré
    - Poder cargar, enviar, y validar que el formato del pagaré sea pdf -> Envío de pagaré
    - Recibir retroalimentación de un pagaré enviado, si fue exitoso, que me aparezca el préstamo a mi cuenta, y de necesitar cambios, recibir retroalimentación por parte de la directiva -> Envío de pagaré
    - Cuando se registre una solicitud de préstamo, mostrar una notificación a los miembros de la directa -> Envío de pagaré
  - Directiva
    - Habilidad de poder visualizar y exportar todas las solicitudes de préstamos pendientes de revisión -> Revisar préstamo
    - Visualizar la información de cada solicitud, y poder aprobarla o denegarla, en caso de rechazar, enviar retroalimentación al socio -> Revisar préstamo.
    - En caso de aprobación de un préstamo, poder generar de forma automática un pagaré (acorde al tipo de préstamo) que deberá ser visualizado por el socio -> Aprobación de préstamo
    - Notificar en caso de aprobación al socio -> Aprobación de préstamo
    - Ver el registro de todos los préstamos vigentes e históricamente aprobados -> Visualización de préstamos
    - Observar préstamos pendientes de verificación, y aprobarlos o rechazarlos, de ser lo último enviar comentarios al socio -> Aprobación de préstamos

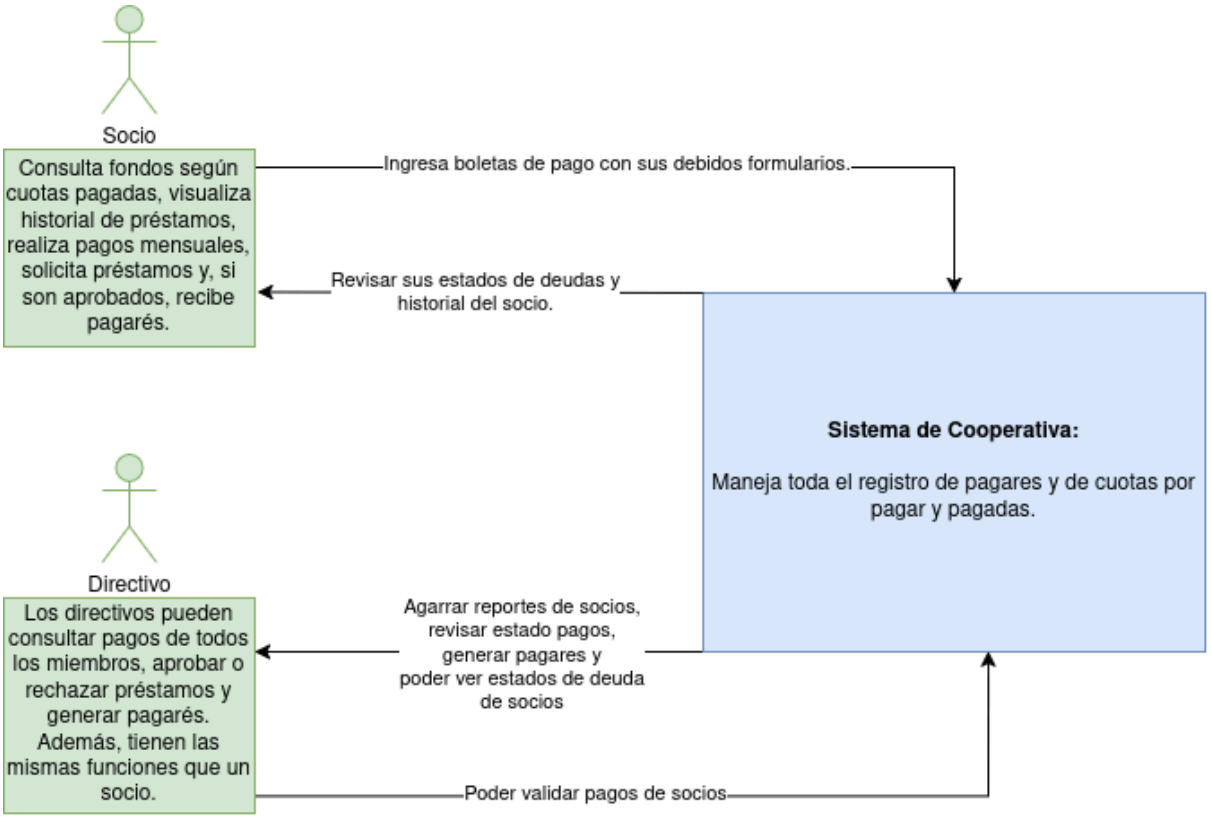
- Consultas
    - Socio
        - Visualizar el historial de pagos presentados, logrando mostrar el monto, título y fecha -> Ver estado de pagos
        - Ver historial de préstamos, divididos entre vigentes y pasados. En el caso de los que se tengan pendientes de cierre, mostrar información de cuántos pagos lleva el socio, contra los totales, así como el monto que ha pagado del préstamo y lo que debe. -> Ver estado de préstamos.
        - Tener una lista completa de preguntas frecuentes seccionadas en diferentes temas de relevancia para un socio que pueda ser consultada en cualquier momento. -> Ver FAQs
        - En el historial de pagos se tendría que visualizar el monto acumulado de los aportes que ha realizado, así como la deuda a capital que tenga. -> Ver estado de pagos.
    - Directiva
        - Observar el estado de pago de cada uno de los socios, permitiéndoles visualizar quiénes tienen moras, ya sea de cuotas o préstamos, y los que van al día con sus pagos -> Consulta de información de clientes.

Clases Preliminares/Mapa de Arquitectura

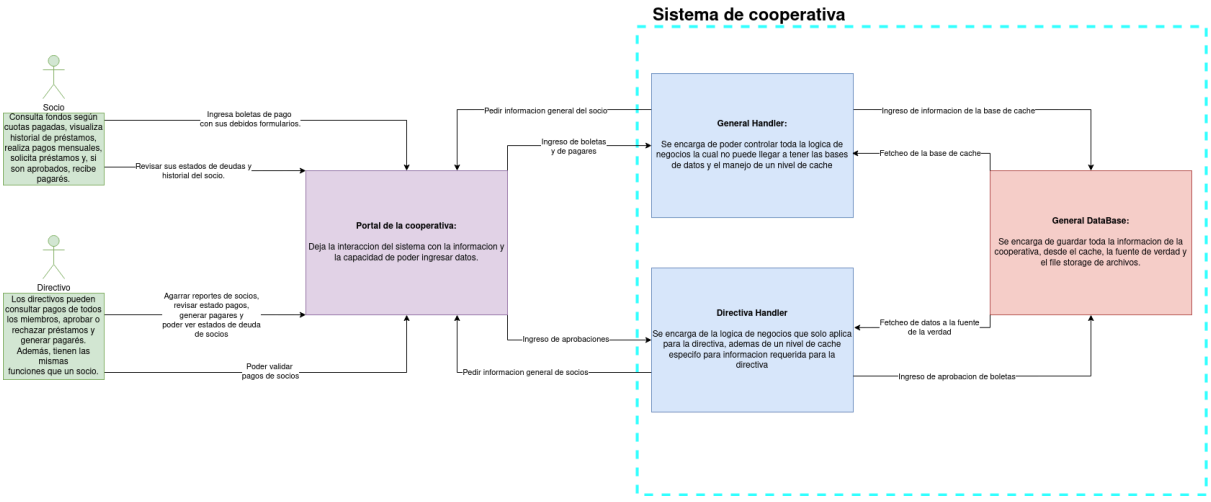
Diagrama Model C4 (Diagrama de referencia de nivel macro A nivel micro)/ Diagrama de paquetes.

Model C4:

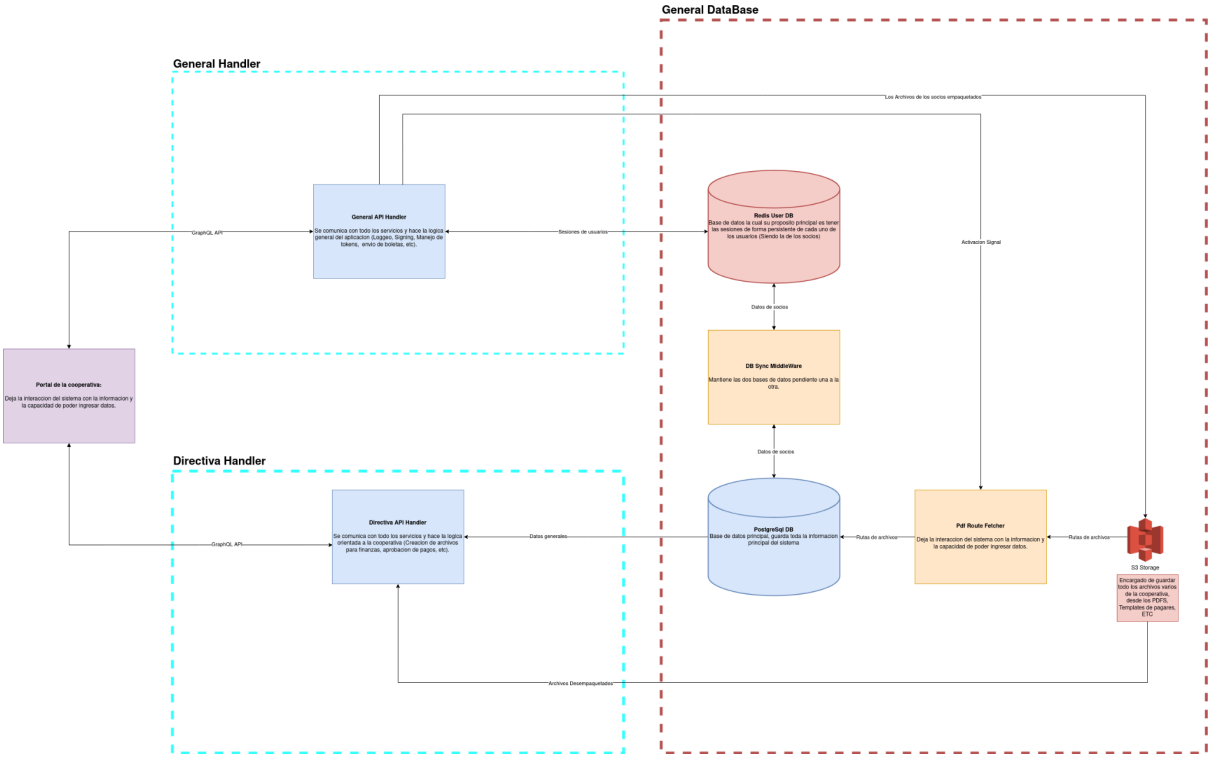
Primera Abstraccion:



Segunda Abstraccion:



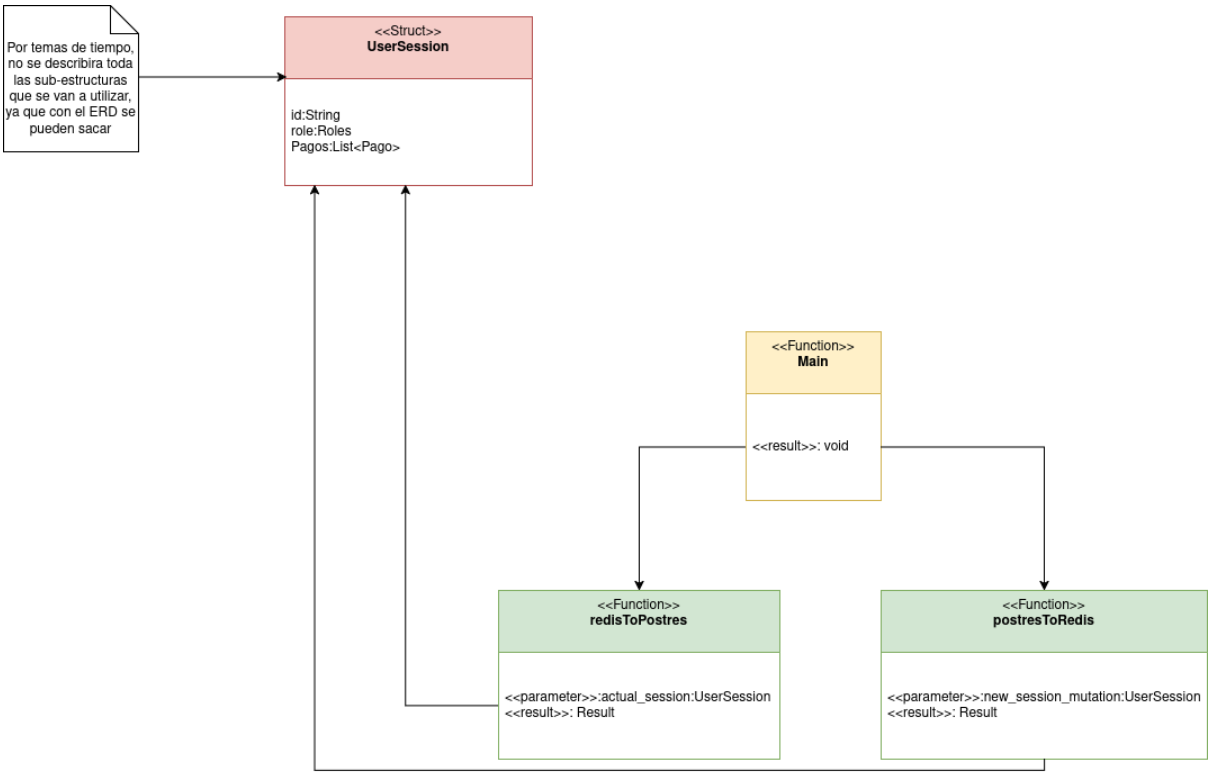
Tercera Abstracción:



Diagrams de UML:

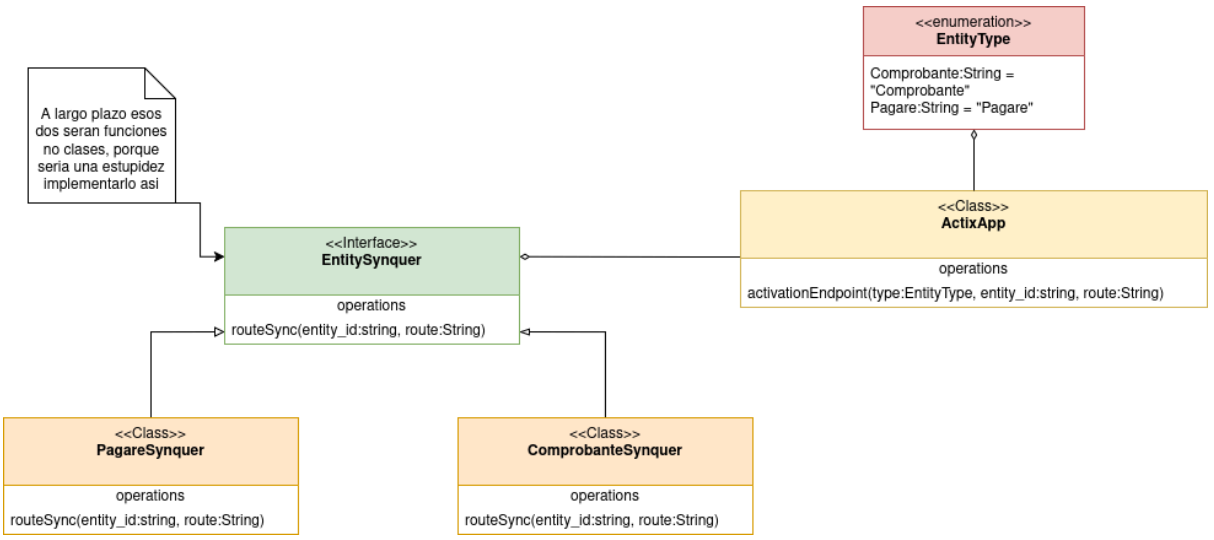
[Diagramas en drawio]

- Db Sync MiddleWare:

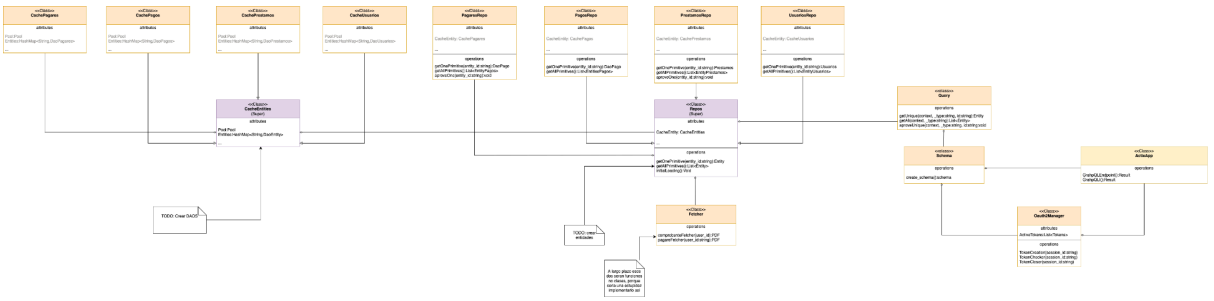


- Justificacion de clases/funciones:
  - redisToPostgres/postgresToRedis: para mantener en sintonía las dos bases de datos dependiendo de las sesiones de usuario existentes.
  - UserSession: estructura para tener cada sesión empaquetada por sí sola.

- Pdf Route Fetcher:



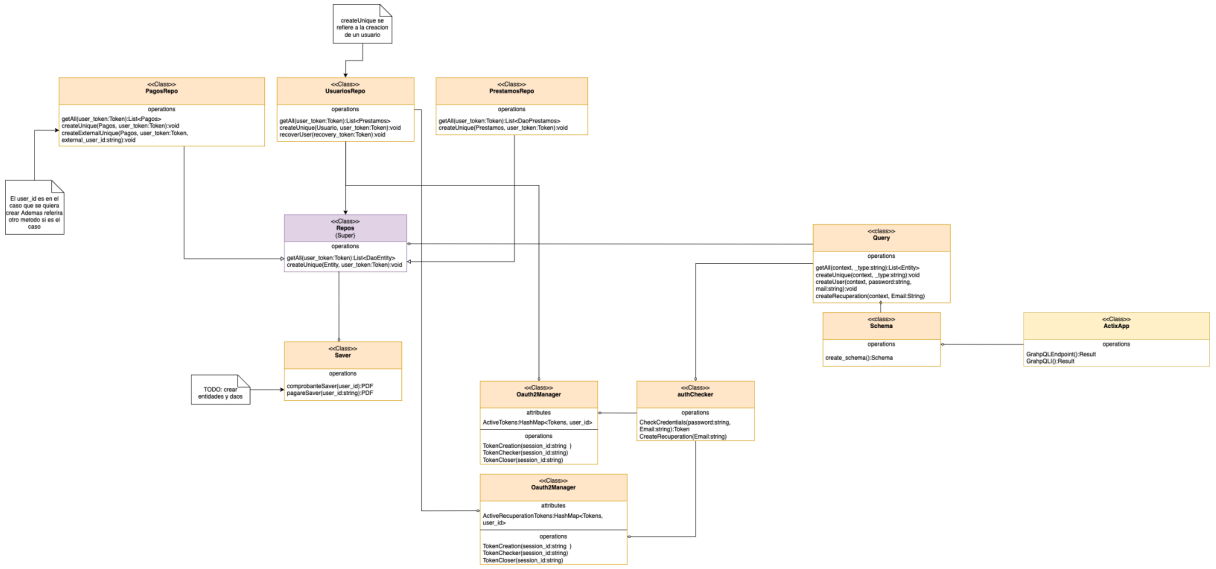
- Justificación de clases/funciones:
  - ActixApp: es la única forma en la que se puede dar el comando de activación para poder tener los ID de los archivos en la base de datos de PostgreSQL.
  - PagareSynquer: paar poder manejar los archivos de Pagaré específicamente.
  - ComprobanteSynquer: para poder manager los comprobantes precisamente.
- Directiva API Handler:



- Justificación de clases:
  - CachePagos: Almacena en caché los datos de pagos para optimizar el acceso y reducir la carga en la base de datos.
  - CachePrestamos: Gestiona la caché de los préstamos, permitiendo un acceso más rápido a los datos de los préstamos registrados.
  - CacheUsuarios: Mantiene en memoria los datos de los usuarios para mejorar la eficiencia en consultas frecuentes.
  - CacheEstados: Almacena en caché los estados de los préstamos y pagos, asegurando un acceso rápido a su información.
  - CacheEntities (Superclase): Proporciona una estructura base para la gestión de caché en diferentes entidades, permitiendo la reutilización de lógica común.

- PagosRepo: Maneja la persistencia de los pagos, ofreciendo métodos para obtener, aprobar y almacenar registros de pagos en caché.
- PrestamosRepo: Administra la persistencia de préstamos con métodos para obtener y aprobar solicitudes de préstamos.
- UsuariosRepo: Controla el acceso y almacenamiento de usuarios en caché y en la base de datos.
- Repo (Superclase): Define la estructura base para los repositorios de datos, facilitando la implementación de métodos genéricos para la manipulación de entidades.
- Fetcher: Se encarga de la generación de documentos PDF relacionados con comprobantes y pagarés.
- Query: Define las consultas disponibles en la API, permitiendo recuperar y gestionar datos de diversas entidades.
- Schema: Estructura el esquema GraphQL utilizado en la API para manejar las consultas y mutaciones.
- ActixApp: Representa la aplicación principal que gestiona los endpoints GraphQL y responde a las peticiones de los clientes.
- OAuth2Manager: Administra la autenticación mediante OAuth2, gestionando tokens de sesión y validando accesos de usuarios.

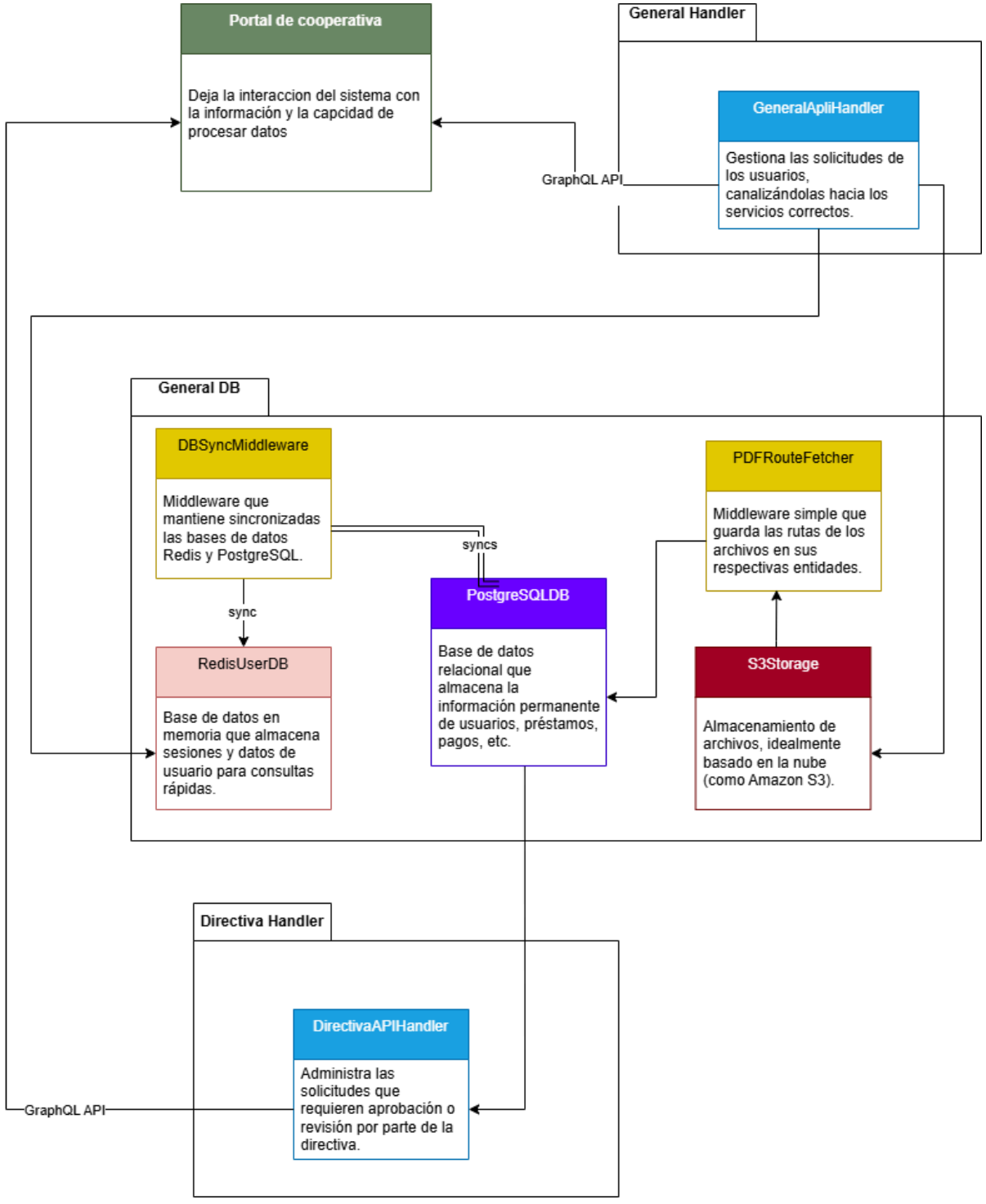
- General API Handler:



- Justificacion de clases:
  - PagosRepo:Maneja operaciones relacionadas con los pagos dentro del sistema, además que tiene métodos de pago para obtener pagos de un usuario, generar nuevos pagos y registrar pagos de fuentes externas.
  - UsuariosRepo: Administra datos de usuarios en la base de datos con métodos para obtener usuarios, registrar nuevos y recuperación de cuenta.
  - PrestamosRepo: Gestiona préstamos de usuarios mediante métodos para recuperar préstamos y registrar nuevos.
  - Repo (superclase):Actúa como clase base para los repositorios de datos, definiendo métodos genéricos de consulta y creación.
  - Saver: Se encarga del almacenamiento de documentos PDF, proporcionando métodos para guardar comprobantes y pagarés.

- OAuth2Manager: Controla la autenticación mediante OAuth2, gestionando tokens activos y ofreciendo validaciones de sesión.
- authChecker: Verifica credenciales de usuario con métodos para autenticación y recuperación de cuenta.
- Query: Define consultas generales sobre la base de datos, permitiendo la creación y recuperación de usuarios.
- Schema: Representa el esquema del sistema para GraphQL y contiene la estructura de datos.
- ActixApp:Maneja los endpoints de GraphQL y sirve como el punto de entrada para consultas y mutaciones.

Mapa de paquetes:



Justificación de componentes:

General API Handler:

- Se comunica con todo los servicios y hace la logica general del aplicacion (Loggeo, Signing, Manejo de tokens, envío de boletas, etc).
    - Motivos:

        - Tener la lógica y todos los métodos de la directiva separada, para poder escalar dependiendo el tráfico de socios o de directiva.

File Saver Handler:

- Deja la interacción del sistema con la información y la capacidad de poder ingresar datos, además de poder empaquetar los archivos y creación de directorios.
    - Motivos: Poder escalar específicamente del lado solo de los usuarios y tener desacoplado de la parte de la directiva.

Directiva API Handler:

- Se comunica con todo los servicios y hace la lógica orientada a la cooperativa (Creación de archivos para finanzas, aprobación de pagos, etc).
    - Motivos: Tener la lógica y todos los métodos de la directiva separada, para poder escalar dependiendo el tráfico de socios o de directiva.

Directiva Cache:

- Encargado de tener los datos, los cuales normalmente se tienen en cuenta para usarse más a menudo, los cuales son fetcheados desde PostreSql.
    - Motivos: Para poder tener la información más usada y requerida en memoria volátil y fetchear constantemente, evitando tener el bottle-neck de estar fetcheando constantemente a la DB kPostgreSQL.

File Fetcher Handler:

- Deja la interacción del sistema con la información y la capacidad de poder ingresar datos.
    - Motivos: Poder escalar específicamente del lado solo de los usuarios y tener desacoplado de la parte de la directiva.

Redis User DB:

- Base de datos la cual su propósito principal es tener las sesiones de forma persistente de cada uno de los usuarios (Siendo la de los socios).
    - Motivos: No sobrecargar la PostgreSQL DB.

Db Sync Middleware:

- Mantiene las dos bases de datos pendientes una a la otra.
    - Motivos: Tener las dos bases de datos al tanto del estado de una y la otra.

PostgreSQL:

- Base de datos principal, guarda toda la información principal del sistema.
    - Motivos: Por la falta de deuda técnica que habría en el equipo.

Pdf Route Fetcher:

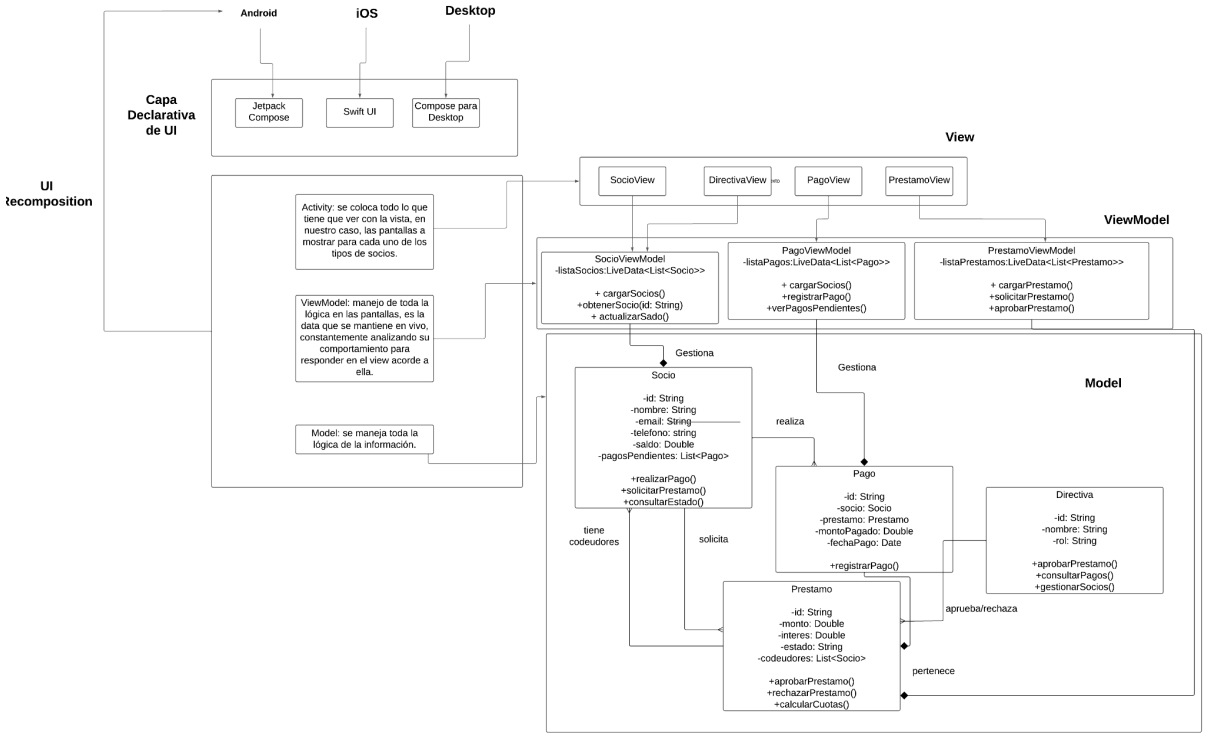
- Deja la interacción del sistema con la información y la capacidad de poder ingresar datos.
    - Motivos: Middleware necesario para poder tener al tanto del File Storage (S3) PostgreSQL.

S3 Storage:

- Encargado de guardar todos los archivos varios de la cooperativa, desde los PDFS, Templates de pagarés, ETC.

- Motivos: Una forma que no requiere hosteo.

Diagrama de MVVM (Arquitectura a utilizar en el Frontend)



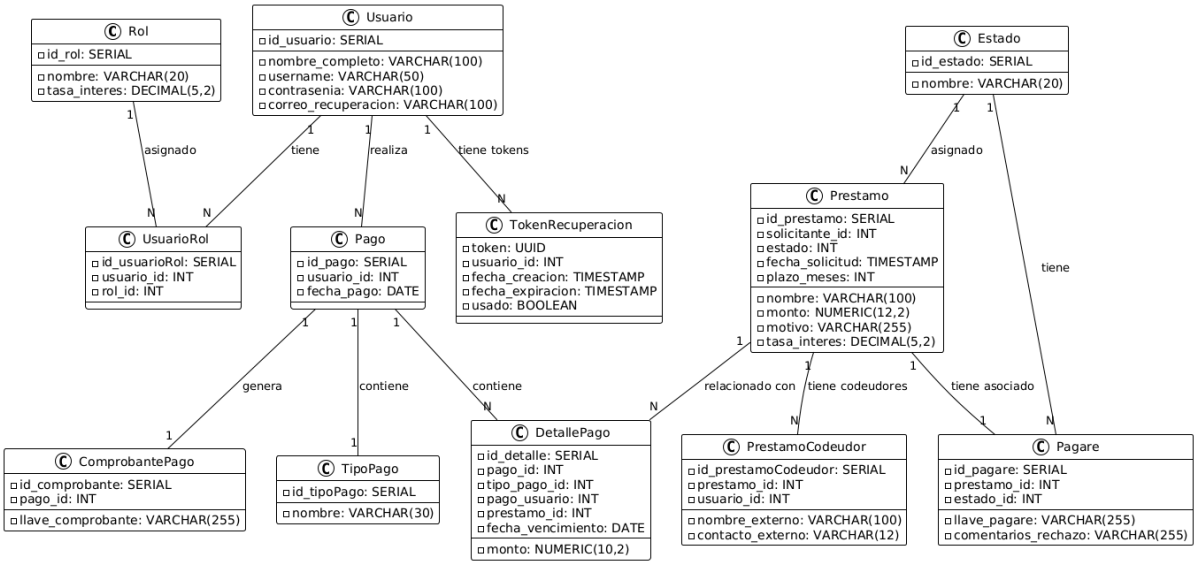
- Modelo
    - Clase Socio: Representa a un miembro de la cooperativa. Contiene información personal y financiera, así como la lista de pagos pendientes. Permite realizar pagos, solicitar préstamos y consultar su estado financiero.
    - Clase Prestamo: Define los préstamos dentro del sistema. Contiene información sobre el monto, intereses, estado y codeudores. Permite la aprobación, rechazo y cálculo de cuotas de un préstamo.
    - Clase Pago: Representa un pago realizado por un socio hacia un préstamo. Contiene información sobre el monto pagado, la fecha de pago y el préstamo asociado.
    - Clase Directiva: Representa a los administradores del sistema. Tienen la capacidad de aprobar o rechazar préstamos, consultar pagos y gestionar socios.
- ViewModel
    - Clase SocioViewModel: Gestiona los datos de los socios para la vista. Se encarga de cargar la lista de socios, obtener detalles de un socio específico y actualizar su saldo.
    - Clase PrestamoViewModel: Administra la información de los préstamos dentro de la interfaz de usuario. Permite cargar, solicitar y aprobar préstamos.
    - Clase PagoViewModel: Maneja los datos relacionados con los pagos. Facilita la carga de pagos realizados, el registro de nuevos pagos y la visualización de pagos pendientes.
- View
    - Clase SocioView: Interfaz de usuario donde un socio puede consultar su estado financiero, realizar pagos y solicitar préstamos.
    - Clase PrestamoView: Pantalla en la que se visualizan los detalles de los

préstamos disponibles y su estado.

- Clase PagoView: Vista dedicada a mostrar y registrar los pagos realizados por los socios.
- Clase DirectivaView: Interfaz destinada a los administradores para la gestión de préstamos, pagos y socios.

Persistencia de Datos

Diagrama de Clases Persistentes:

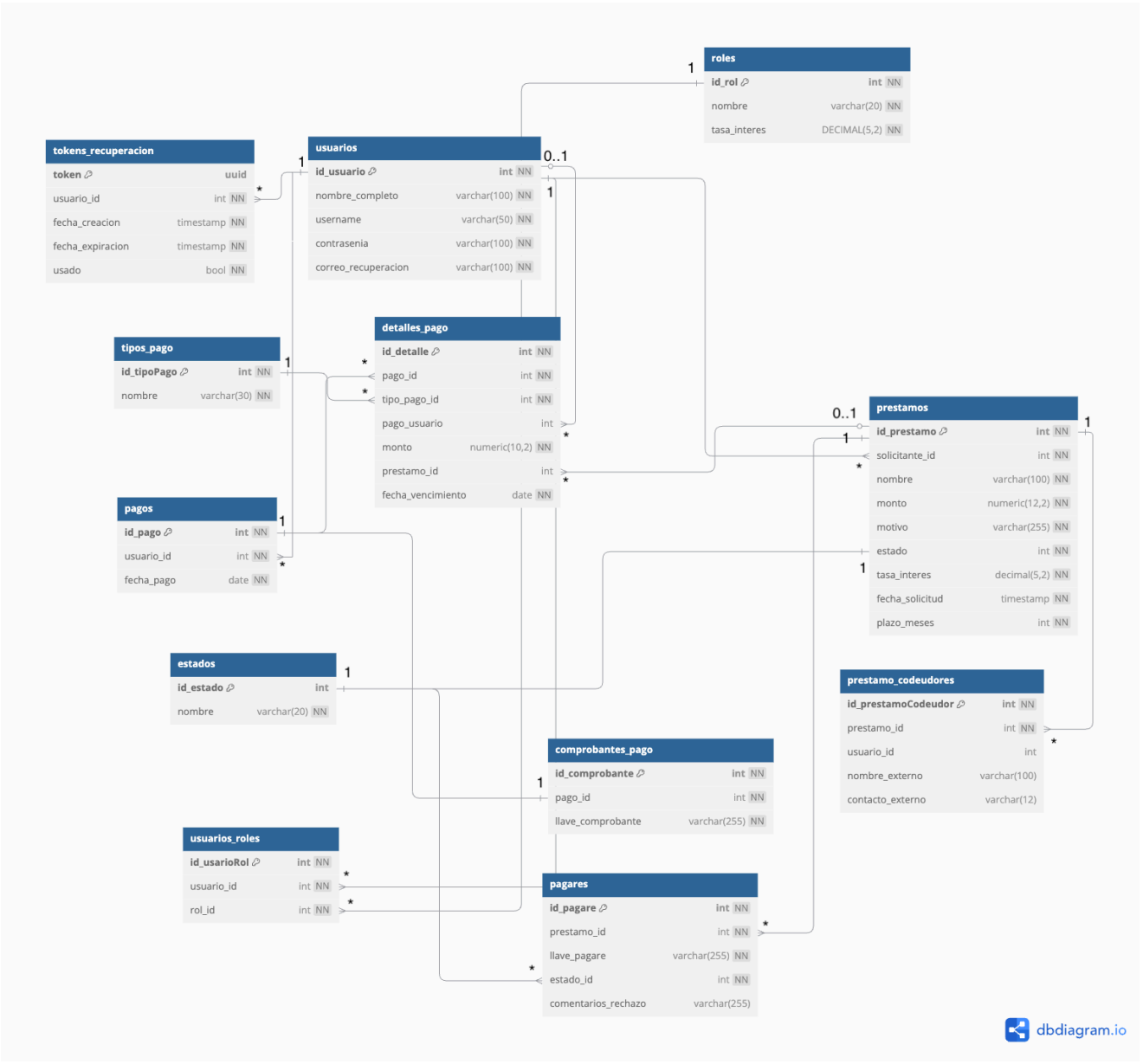


Seleccionar de Tecnología de Almacenamiento:

Selección	Descripción	Ventajas	Desventajas	¿Por qué se seleccionó?
Redis	Redis es una base de datos que principalmente tiene sus datos en memoria volátil, para poder ser accedidos de la forma más rápida posible, además de tener la forma de poder persistir los datos.	• Rendimiento muy alto en operaciones de lectura/escritura. • Fácil integración con múltiples lenguajes de programación. • Estructuras de datos versátiles que permiten un desarrollo rápido de funcionalidades como colas, contadores, etc.	• Para grandes volúmenes de datos, puede ser costoso en términos de RAM. • Su escalabilidad horizontal requiere configuraciones adicionales (clustering, sharding). • La versión más reciente (7.4) no es totalmente open source.	Para no crear cuellos de botella con la base de datos principal, se tomó en cuenta esta idea de diseño.

PostgreSql	PostgreSql es una base de datos SQL la cual tiene un ecosistema bastante grande de diferentes “plugins” y que da simplicidad en la forma que se crean los queries, sin escatimar en “Performance”.	• Altamente confiable y estable, con gran comunidad de soporte. • Excelente para consultas complejas y transacciones seguras. • Soporta una amplia gama de tipos de datos, facilitando la flexibilidad en el modelado.	• La escalabilidad horizontal nativa es limitada sin extensiones adicionales (p.ej., Citus). • Requiere más recursos y configuraciones para grandes volúmenes de datos o cargas muy altas. • La curva de aprendizaje puede ser mayor para equipos sin experiencia en SQL avanzado.	Al ser una herramienta bastante adoptada, se tomó esta opción.
S3 AWS	Es un servicio que ofrece Amazon para poder guardar diferentes tipos de archivos.	• Es bastante accesible y también no requiere hosteo. • Escalabilidad prácticamente ilimitada.	• Es un servicio de Amazon, por lo que tiende a ser más costoso. • Latencia de acceso mayor que en bases de datos en memoria o disco local.	Por su simpleza de uso y la habilidad de poder ser integrado bastante fácilmente

Diagrama de Entidad Relación


roles
id_rol int NN
nombre varchar(20) NN
tasa_interes DECIMAL(5,2) NN
tokens_recuperacion
token uuid
usuario_id int NN
fecha_creacion timestamp NN
fecha_expiracion timestamp NN
usado bool NN
usuarios
id_usuario int NN
nombre_completo varchar(100) NN
username varchar(50) NN
contrasenia varchar(100) NN
correo_recuperacion varchar(100) NN
tipos_pago
id_tipoPago int NN
nombre varchar(30) NN
detalles_pago
id_detalle int NN
pago_id int NN
tipo_pago_id int NN
pago_usuario int
monto numeric(10,2) NN
prestamo_id int
fecha_vencimiento date NN
prestamos
id_prestamo int NN
solicitante_id int NN
nombre varchar(100) NN
monto numeric(12,2) NN
motivo varchar(255) NN
estado int NN
tasa_interes decimal(5,2) NN
fecha_solicitud timestamp NN
plazo_meses int NN
pagos
id_pago int NN
usuario_id int NN
fecha_pago date NN
estados
id_estado int
nombre varchar(20) NN
prestamo_codeudores
id_prestamoCodeudor int NN
prestamo_id int NN
usuario_id int
nombre_externo varchar(100)
contacto_externo varchar(12)
comprobantes_pago
id_comprobante int NN
pago_id int NN
llave_comprobante varchar(255) NN
usuarios_roles
id_usarioRol int NN
usuario_id int NN
rol_id int NN
pagares
id_pagare int NN
prestamo_id int NN
llave_pagare varchar(255) NN
estado_id int NN
comentarios_rechazo varchar(255)
dbdiagram.io

Selección de la Tecnología:

Estimaciones de las Dimensiones:

Requisito No Funcional	**Estimación**
Contar con una interfaz amigable para los usuarios.	Que se pueda realizar cualquier acción dentro de la aplicación con no más de 4 clicks.
Incorporación de tickets para autenticación de usuarios.	Manejar 2 pasos extra de seguridad para tener acceso a la información.
Que se pueda utilizar en iOS, Android, y Windows.	Tener un 80% de cobertura en todas estas plataformas.
El sistema debe ser capaz de gestionar simultáneamente al menos 30 usuarios activos.	Que al menos 30 usuarios puedan utilizar la aplicación sin comprometer el funcionamiento del sistema.
Las transacciones de validaciones de pagos y aprobación de préstamos deberían ocurrir en un lapso prudencial.	Que la consulta se pueda concretar en menos de 3 segundos.
La base de datos debe ser capaz de soportar el crecimiento de la corporación.	Crecimiento a 100 usuarios.
El sistema debe tener un método seguro para recuperar contraseñas.	Permitir que los usuarios restablezcan acceso mediante un enlace enviado a su correo electrónico, que tenga vigencia por 10 minutos.

Selección de Tecnología: (lenguaje, frameworks)

- Frontend

Selección	Descripción	Ventajas	Desventajas	¿Por qué se seleccionó?
Framework - Compose Multiplatform	Es un framework de la familia de Kotlin que permite desarrollar aplicaciones multiplataforma (Android, iOS, Desktop, e incluso Web) con un solo código base y un enfoque declarativo para la construcción de interfaces.	● Reutilización de gran parte del código para múltiples plataformas. ● Enfoque declarativo que facilita la actualización de la UI. ● Soporte oficial de JetBrains y Google.	● Es relativamente nuevo, por lo que la comunidad y la documentación pueden ser más reducidas en comparación con otras opciones, como flutter. ● Requiere familiaridad con Jetpack Compose (desarrollo nativo en Android).	Facilita con el cumplimiento del requerimiento de llegar a iOS, Android y Windows desde un único código base. Esto puede ayudarnos a reducir tiempos y costos de desarrollo.
Lenguaje - Kotlin	Lenguaje de programaciónn moderno, estáticamente tipado, que se ejecuta en la JVM y soporta compilación nativa y JavaScript mediante Kotlin Multiplatform. Tiene sintaxis concisa y un fuerte enfoque en la seguridad de tipos.	● Sintaxis clara y concisa. ● Total interoperabilidad con Java. ● Soporte de Google. ● Integración directa con Compose Multiplatform. ● Amplia comunidad y buena documentación.	● Si no se conoce el lenguaje puede existir una curva de aprendizaje. ● Tiempo de compilación más lento que Java.	Es el lenguaje principal soportado por Compose Multiplatform, lo que nos permitiría tener un único código base funcional en las diferentes plataformas que nuestros product owners nos están solicitando.

Arquitectura - MVVM	Patrón de arquitectura Model-View-ViewModel, que separa la lógica de negocio (ViewModel) de la interfaz de usuario (view) y el manejo de datos (model), facilitando el mantenimiento, las pruebas y la escalabilidad de la aplicación.	● Favorece la separación de responsabilidades, lo que ayuda con el mantenimiento. ● Encaja bien con el enfoque declarativo de Compose, que es básicamente que la UI reaccione a los cambios de estado.	● En proyectos que son pequeños se puede ver como sobredimensionado. ● Requiere una buena definición de cada una de las capas mencionadas, de lo contrario se pueden presentar errores.	Se integra de forma natural con la programación declarativa de Compose, esto ayuda a cumplir también con los requerimientos no funcionales asociados con el rendimiento de la aplicación, al buscar que el sistema pueda realizar transacciones rápidas y que sea escalable.

- Backend

Selección	**Descripción**	**Ventajas**	**Desventajas**	**¿Por qué se seleccionó?**
Actix/Rust	Attix es un framework web el cual está hecho para utilizar todo el poder que Rust da como lenguaje y poder implementarlo en el WebServer.	Bundles pequeños, buen “Perfomance”.	Alta curva de aprendizaje y complejidad.	Por la naturaleza de la aplicación y su filosofía, tenemos muy en cuenta la velocidad.
GraphQL	GraphQL es un lenguaje de Queries, especializado para APIs.	Ayuda a que no haya under/over fetching al momento de hablar con el back-end.	No tan buen performance, por tener que siempre fetchear todos los datos para después poder parsearlos.	Aunque pueda tener bottlenecks de performance, la naturaleza del sistema hace que no haya la necesidad de crear tantas rutas para

				acceder a la misma información o overfetchear. Por esa misma razon usamos lenguajes y frameworks tan rapidos, para poder sobrellevar mejor este lado.
Rust (Middleware)	Rust es un lenguaje que deja manipulacion de bajo nivel, con ventajas de seguridad de memoria y otros aspectos que no tienen otros lenguajes del mismo estilo.	Bundles pequenos, buen “Perfomance”.	Alta curva de aprendizaje y complejidad.	Por la naturaleza de la aplicacion y su filosifia, tenemos muy en cuenta la velocidad.
Redis	Redis es una base de datos que principalmente tiene sus datos en memoria volátil, para poder ser accedidos de la forma más rápida posible, además de tener la forma de poder persistir los datos.	● Rendimiento muy alto en operaciones de lectura/escritura. ● Fácil integración con múltiples lenguajes de programación. ● Estructuras de datos versátiles que permiten un desarrollo rápido de funcionalidades como colas, contadores, etc.	● Para grandes volúmenes de datos, puede ser costoso en términos de RAM. ● Su escalabilidad horizontal requiere configuraciones adicionales (clustering, sharding). ● La versión más reciente (7.4) no es totalmente open source.	Para no crear cuellos de botella con la base de datos principal, se tomó en cuenta esta idea de diseño.
PostgreSql	PostgreSql es una base de datos SQL la	● Altamente confiable y estable, con	● La escalabilidad horizontal	Al ser una herramienta bastante

	cual tiene un ecosistema bastante grande de diferentes “plugins” y que da simplicidad en la forma que se crean los queries, sin escatimar en “Performance”.	gran comunidad de soporte. ● Excelente para consultas complejas y transacciones seguras. ● Soporta una amplia gama de tipos de datos, facilitando la flexibilidad en el modelado.	nativa es limitada sin extensiones adicionales (p.ej., Citus). ● Requiere más recursos y configuraciones para grandes volúmenes de datos o cargas muy altas. ● La curva de aprendizaje puede ser mayor para equipos sin experiencia en SQL avanzado.	adoptada, se tomó esta opción.
S3 AWS	Es un servicio que ofrece Amazon para poder guardar diferentes tipos de archivos.	● Es bastante accesible y también no requiere hosteo. ● Escalabilidad prácticamente ilimitada.	● Es un servicio de Amazon, por lo que tiende a ser más costoso. ● Latencia de acceso mayor que en bases de datos en memoria o disco local.	Por su simpleza de uso y la habilidad de poder ser integrado bastante fácilmente
Podman	Es un Runtime de containers, el cual tiene bastantes integraciones con diferente software de “Orchestation” y es de código libre.	Ayuda a integraciones más rápidas y salidas a producción por medio de contenedores y tiene buena documentación.	Al ser Podman no igual de famoso que otras herramientas de contenedores, no tiene tanta documentación.	Para poder integrarlo con herramientas de escalado con orquestadores.
Kubernetes	Es un orquestador de contenedores, el cual deja a	Ayuda al escalado horizontal y vertical.	Tiene complejidad para poder ser implementado	Para ayudar el escalado en todo los sentidos.

	los mismos contenedores poder escalar horizontalmente.		de forma eficiente.	
OpenTelemetry	Es una herramienta de recolección de telemetría, para poder ver los diferentes logs de todo un sistema.	Fácil de implementar con kubernetes, además de poder ofrecer todos los logs en dashboards intuitivos.	Pone otra capa de complejidad a un sistema ya bastante complejo.	Para poder llevar un control del sistema sin tener que interactuar directamente con él.

Informe de Gestión

Lista de Tareas Propuestas:

- Resumen
    - Adriana Palacios
- Introducción
    - Bryan Martínez
- Prototipos
    - Bryan Martínez
    - Adriana Palacios
- Requisitos Funcionales
    - Todos
- Clases Preliminares
    - Pedro Ávila
    - Diego López
- Mapa y Diagrama de Clases
    - Pedro Avila
    - Diego López
- Persistencia de Datos
    - Todos
- Selección de la Tecnología
    - Pedro Avila
    - Bryan Martinez
- Informe de gestión de Tiempo
    - Bryan Martínez
    - Adriana Palacios

Formulario LOGT

Nombre: Diego Javier López Reinoso
Carne: 23747

Fecha	Inicio	Fin	Tiempo Interrupción	Delta Tiempo	Fase	Comentarios
14/03/25	20:00	23:00	10min	1h50min	Desarrollo de UML de paquetes	Se realizó en draw.io
15/03/25	8::00	11:00	30min	1h30min	Justificación de componentes UML de middleware	Se justificó el porque de los componentes en los uml de los middleware
16/03/25	16:00	19:00	3h	0	Diagrama de clases persistentes	Se creó el diagrama de clases persistentes con PlantUML

Nombre: Bryan Alberto Martínez Orellana
Carne: 23542

Fecha	**Inicio**	**Fin**	**Tiempo Interrupción**	**Delta Tiempo**	**Fase**	**Comentarios**
11/03/25	13:00	14:00	0	60 min	Elaboración Primer Prototipo	Se elaboraron un prototipo inicial
13/03/25	10:00	11:00	0	60 min	Elaboración Segundo Prototipo	Se elaboró un segundo prototipo tomando en cuenta los comentarios del product owner.
15/02/25	12:00	17:00	30 min	150 min	Introducción	Se ocupó el tiempo por partes, se

						hizo la actividad en espacios libres durante el día.
10/03/25	11:00	12:30	20 min	70 min	Requisitos Funcionales	Se identificaron los requisitos funcionales.
10/03/25	13:00	14:00	0	60 min	Persistencia de datos	Se realizó el diagrama de entidad relación
11/03/25	11:00	12:00	0	60 min	Selección de la Tecnología	Se eligió la tecnología con la que se trabajará.
13/03/25	22:00	23:00	0	60 min	Informe de gestión de Tiempo	Se redactó el informe de gestión de tiempo

Nombre: Adriana Sophia Palacios Contreras
Carne: 23044

Fecha	Inicio	Fin	Tiempo Interrupción	Delta Tiempo	Fase	Comentarios
12/03/25	10:00	11:00	10 min	50 min	Resumen	Se redactó el resumen del corte.
11/03/25	13:00	14:00	0	60 min	Elaboración Primer Prototipo	Se elaboró un prototipo inicial
13/03/25	10:00	11:00	0	60 min	Elaboración Segundo Prototipo	Se elaboró un segundo prototipo basándose en la retroalimentación del product owner
10/03/25	11:00	12:30	20 min	70 min	Requisitos	Se definieron los

					Funcionales	requisitos funcionales.
10/03/25	13:00	14:00	0	60 min	Persistencia de datos	Se elaboró el diagrama de entidad relación
11/03/25	11:00	12:00	0	60 min	Selección de la Tecnología	Se seleccionó la tecnología para el proyecto.
13/03/25	22:00	23:00	0	60 min	Informe de gestión de Tiempo	Se redactó el informe de gestión de tiempo

Nombre: Pedro Rubén Avila Cofiño
Carne: 23089

Fecha	Inicio	Fin	Tiempo Interrupción	Delta Tiempo	Fase	Comentarios
10/03/25	12:00 pm	8:00 pm	2 h	8 h	Mapa de paquetes y clases preliminares	Creacion de mapa de arquitectura
12/03/25	5:00 pm	10:00 pm	30 min	5 h	Mapa de paquetes y clases preliminares	Creacion de diagramas de UML
13/03/25	2:00	2:55	0	55 min	Mapa de paquetes y clases preliminares	Creacion de diagramas C4 para referencia de todo el sistema
14/03/25	1:00	5:00	3 horas	4 horas	Seleccion de tecnologias	Seleccion de toda las tecnologias de backend
15/03/25	5:00	7:00	1 hora	2 horas	Persistencia de datos	Ayuda en ORM y estructura de base de datos

16/03/25	5:00	7:00	1 hora	2 horas	Seleccion de tecnologia de persistencia de datos	Seleccion de tecnologias de persistencia de datos

Informe de Gestión de Tiempo:

Este informe detalla la gestión de tiempo del equipo durante el desarrollo del sistema de control financiero para la CSPI, correspondiente a las actividades realizadas en esta fase del proyecto. Se describen las tareas ejecutadas, el tiempo invertido, el desempeño del equipo, así como los aspectos positivos y las áreas de mejora identificadas para optimizar el avance en las próximas etapas.

El equipo inició las actividades de esta fase el de marzo de 2025 y concluyó el de marzo de 2025. Durante este período, se avanzó en la creación de prototipos, el análisis y definición de los requisitos funcionales y no funcionales, la selección de tecnologías, y el diseño preliminar de la arquitectura y persistencia de datos del sistema.

Las actividades realizadas fueron las siguientes:

1. Elaboración de prototipos: Se desarrollaron dos prototipos iniciales y uno más elaborado, permitiendo visualizar el diseño preliminar del sistema.
2. Interacción y retroalimentación de prototipos: Se documentaron las interacciones realizadas con usuarios de la cooperativa para validar los prototipos.
3. Identificación de requisitos funcionales y no funcionales: Se establecieron las funciones clave del sistema, así como los criterios que deben cumplirse para su correcto desempeño.
4. Selección de tecnologías: Se analizó y justificó la selección de tecnologías para el desarrollo del sistema, priorizando soluciones multiplataforma.
5. Diseño preliminar de la arquitectura del sistema: Se desarrollaron diagramas preliminares para definir la estructura del sistema y su persistencia de datos.
6. Documentación de reglas de negocio: Se identificaron las principales reglas operativas que guiarán el comportamiento del sistema.

Los aspectos positivos del desempeño del equipo fueron:

1. Distribución equitativa de tareas: Las responsabilidades se asignaron considerando las fortalezas de cada integrante, lo que agilizó el avance.
2. **I**nteracción con el cliente: Se realizaron consultas directas con miembros de la CSPI, lo que permitió validar ideas y obtener retroalimentación.
3. Se mantuvo contacto permanente a través de WhatsApp y reuniones periódicas en Discord, permitiendo aclarar dudas y revisar avances en tiempo real.

Por otro lado, estas son las áreas de mejora que como equipo hemos identificado:

1. Mejor gestión de reuniones: Algunas reuniones se extendieron más de lo planeado debido a retrasos en su inicio, por lo que se busca establecer horarios más estrictos.
2. Control de interrupciones: En ciertas actividades, las interrupciones afectaron la concentración y avance.

La gestión del tiempo en esta fase del proyecto fue eficiente, permitiendo completar las actividades necesarias para el desarrollo del sistema de control financiero. La comunicación fluida y el uso de herramientas colaborativas facilitaron el cumplimiento de las tareas, aunque se identificaron áreas de mejora relacionadas con la puntualidad y gestión de

reuniones. Estas observaciones se tomarán en cuenta para optimizar la eficiencia en las siguientes fases del proyecto. En general, el desempeño del equipo ha sido satisfactorio, sentando bases sólidas para la implementación del sistema.