Universidad Don Bosco

Escuela de Ingeniería en Computación

Desarrollo de Aplic. Web con Soft. Interpretado en el Servidor

Título del Proyecto:

Cooperativa “ACOEMPRENDEDORES”.

Catedrático:

Ing. Mario Alvarado

Equipo colaborativo:

Lider de proyecto: Derek Marcelo Monge Aguilar - MA230691

Programador back-end: Andrés Eduardo Navidad Flores - NF220677

Programador back-end: Abner Ismael Rivera Leiva - RL233297

DevOps y Frontend: Kenneth Gabriel Monge Aguilar - MA230693

DevOps y Frontend: Juan José Pérez Muñoz – PM230897

Carrera universitaria:

Técnico en Ingeniería en Computación

Fecha de entrega:

23/03/2025

Campus Soyapango

Tabla de contenido

1. Descripción del proyecto:

1.1 Objetivo General:

- ✓ Desarrollar un **Sistema Financiero** para la cooperativa **ACOEMPRENDEDORES**, que permita gestionar eficientemente el registro de empleados, clientes y productos financieros, así como el control de transacciones, cartera virtual y administración de usuarios, garantizando seguridad, accesibilidad y operatividad en la administración de dichos servicios.

1.2 Objetivos Específicos:

- ✓ Establecer un **control de usuarios basado en roles** que determine permisos y accesos según el rol o funcionalidad asignada.

- ✓ Implementar un **apartado de gestión de empleados** donde permita registrar y administrar su información personal, laboral y de contacto.

2. Gestión integral del proyecto

2.1 Gestión de la información:

El tipo de gestión que se tiene pensado utilizar para la realización del proyecto es la metodología **Scrum**, mediante springs semanales que permitirán el desarrollo eficiente, organizado y escalable. Se hará uso de **Github** para el controlador de versiones, como también **Trello** para organizar el seguimiento del proyecto y también **Docker y Kubernetes** para la contenedorización y despliegue de la aplicación.

2.1.1 Integración de Equipos y Herramientas:

Se hará uso específicamente tanto de estas tecnologías como metodología con la organización de entregas.

Github: Se hará uso de un repositorio en el cual se almacenará el código fuente, uso de ramas por cada integrante en el cual podremos ver reflejado y mantener un registro de cada uno de los desarrolladores con la ayuda del proyecto.

Trello: Se implementó un tablero donde semanalmente se actualiza el flujo de trabajos correspondientes en la semana, lo cual también nos ayuda a la organización como equipo.

Metodología: Se optó por seleccionar la Metodología Scrum ya que anteriormente se trabajó con la misma tecnología en el cual se definirá los spring semanales para el desarrollo de dicho proyecto.

2.1.2 Entorno de Contenedores:

Docker: Se crearán contenedores por cada apartado del sistema, tanto para el front -end back -end, etc.

Kubernetes: Se implementará en entornos de desarrollo homogéneo en el que se trata de facilitar el escalado y disponibilidad del sistema.

2.1.3 Entrega Coordinada:

Plan de Entregas: Se definen por medio de la metodología Scrum en el cual semanalmente se realizan spring con avances del proyecto.

Coordinador: Básicamente el Líder de nuestro equipo validará los avances de cada implementación, asegurando que cumplamos con los requisitos definidos del proyecto.

2.2 Gestión del Alcance

2.1.1 Definición de Requisitos:

- **Autenticación de usuarios:** Como mínimo debe existir una autenticación de roles para cada uno de los usuarios almacenados en la base de datos.

- **Gestión de Roles:** Definimos roles para cada usuario con el fin de asignar un rol específico(cliente, administrador, etc).

- **Cartera Virtual:** Es decir cada cliente podrá visualizar productos adquiridos, registro y consulta de dichas transacciones.

2.1.2 Límites del Proyecto:

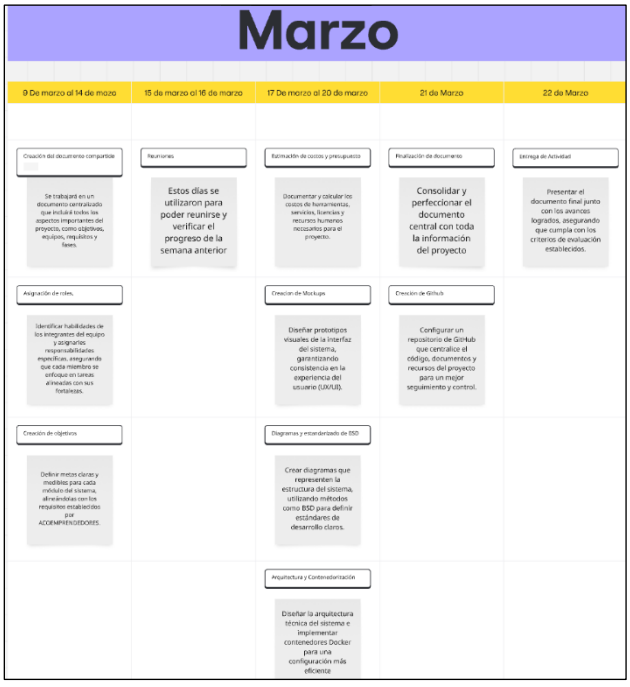
El sistema se ejecutará en contenedores Docker, y se desplegará con Kubernetes para la gestión de servicios en producción. No se agregan funcionalidades fuera del alcance definido sin un análisis de impacto en costos y tiempos de entrega.

2.3 Gestión de cronograma

La utilización de un calendario para el proyecto 'ACOEMPRENDEDORES' consideramos que esencial una planificación. Permitiendo una asignación eficiente de recursos y el cumplimiento de los plazos establecidos. Este calendario divide el proyecto en fases específicas, establece fechas de entrega y asegura que las metas del proyecto se cumplan dentro del tiempo estimado. Además, permite monitorear el progreso, anticiparse a posibles retrasos y mantener un flujo de trabajo organizado y eficiente.

- **Link de Miro:**

https://miro.com/app/board/uXjVINPqtzM=/?share_link_id=58287526365

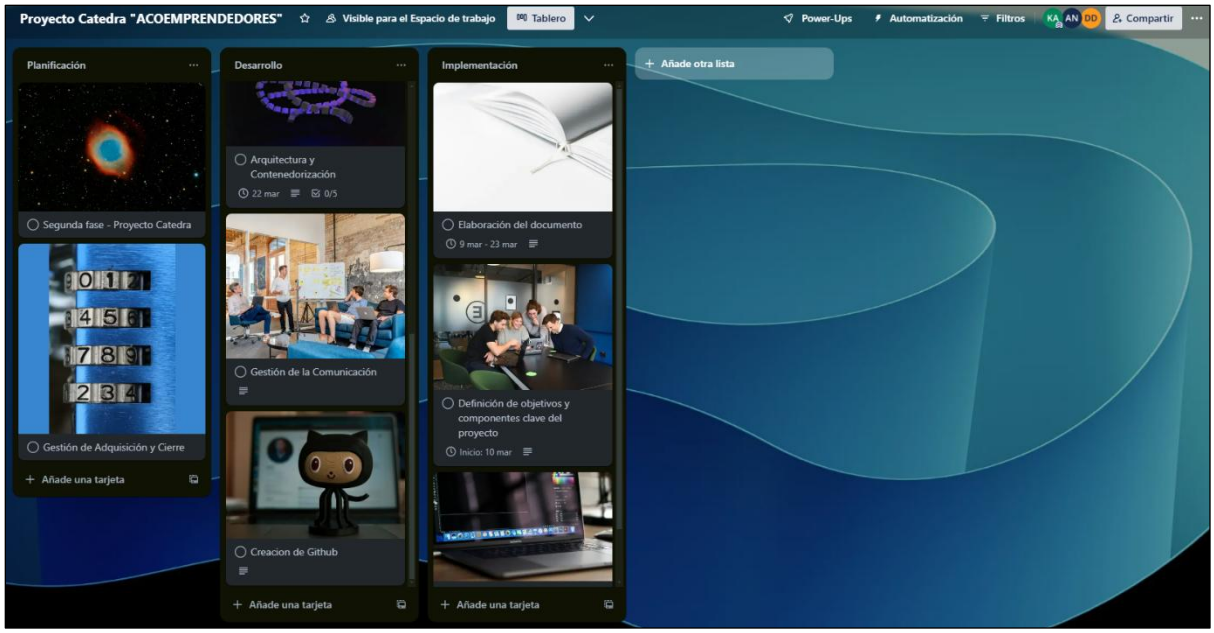


Igualmente, como equipo decidimos trabajar en un Trello para establecer fases en las cuales se iba trabajar de la siguiente manera.

- ✓ **Planificación**:
    - Esta fase se centra en organizar las ideas, definir tareas y establecer objetivos claros.
- ✓ **Desarrollo**:
    - Aquí es donde el trabajo real empieza: la ejecución de las tareas planeadas.
    - Las tarjetas en esta fase deben representar tareas que estén en proceso (o en curso).
    - Ejemplo de tarjetas: "Programar la funcionalidad X", "Revisar la interfaz gráfica", "Corregir errores en el módulo Y".
    - *Cuando una tarea se complete, mueve la tarjeta a "Implementación".*
- ✓ **Implementación**:
    - En esta fase colocas las tareas ya finalizadas o que están listas para ser entregadas.
    - Sirve para mantener un registro de lo que ya está hecho y garantizar que todo cumpla con los criterios esperados.

- **Link de Trello:**

https://trello.com/invite/b/67d5c934d013984cde17604e/ATTI3eb8e7b185b76f3d85dab2fe0071ce5f19694AB6/proyecto-catedra-acoemprendedores



2.4 Gestión de Costos

Cuando se planifica el presupuesto para un proyecto tecnológico como el de la cooperativa “ACOEMPRENDEDORES”, es crucial tomar en cuenta diferentes escenarios que se ajusten a las necesidades y los recursos disponibles. En este caso, hemos diferenciado dos enfoques: **herramientas locales** para un proyecto de menor escala, y **alternativas en la nube** para un proyecto de gran escala. A continuación, presentamos las diferencias y ventajas de cada una de ellas:

Planificación.	
Plan Trello	**Costo Estimado (USD**
Trello Free	Gratis
Trello Standard	$5 por Usuario/mes
Trello Premium	$10 por Usuario/mes

Plan Miro	**Costo Estimado (USD**
Miro Free	Gratis
Miro Team Plan	$8 por Usuario/mes
Miro Business	$16 por Usuario/mes

Base de Datos.	
Plan	**Costo Estimado (USD**
MySQL	Gratis
Azure SQL (Mínimo)	$0,0001050/segundo de núcleo virtual ($0,378/hora de núcleo virtual)
Azure SQL (Estandar)	$266,684/mes

Kubernetes	
Plan	**Costo Estimado (USD**
Plataforma Aws, Azure Kubernetes	300$ - 900$ al mes
Aumento de Trafico o almacenamiento.	100$ - 600$ al mes.
Minikube	Gratis.

Docker	
Plan	**Costo Estimado (USD**
Docker (Comunnity Edtion)	Gratis
Docker (Enterprise Edtion)	1500$ al año

2.5 Definicion de roles

Lider de proyecto: Derek Marcelo Monge Aguilar

Programador Backend: Andrés Eduardo Navidad Flores y Abner Ismael Rivera Leiva

DevOps y Frontend: Kenneth Gabriel Monge Aguilar y Juan José Pérez Muñoz

2.6 Gestión de Comunicación

2.6.1 Informes Semanales de Progreso

Como equipo, utilizamos la aplicación Microsoft Teams para mantener una comunicación constante y coordinar nuestras reuniones de trabajo relacionadas con el proyecto. Las reuniones se realizan principalmente los miércoles y sábados, aunque también nos reunimos en otros días cuando todos los integrantes del equipo están disponibles. Estas sesiones tienen como objetivo principal discutir el avance del proyecto, resolver dudas, y asignar o reprogramar tareas según sea necesario.

Además, para mantener un seguimiento claro y organizado, implementamos un reporte semanal donde cada miembro del equipo reporta los avances que ha realizado en las tareas asignadas durante la semana. Este reporte es revisado y discutido en las reuniones para identificar posibles áreas de mejora y establecer los siguientes pasos a seguir. De esta forma, garantizamos que todos los miembros del equipo estén alineados y comprometidos con los plazos establecidos.

Las reuniones no solo se enfocan en la verificación de tareas, sino también en la identificación de problemas o desafíos que puedan surgir durante el desarrollo del proyecto, de modo que podamos ofrecer soluciones de manera colaborativa. Además, fomentamos un ambiente de retroalimentación constante, lo cual nos permite mejorar continuamente el proceso de trabajo y adaptarnos rápidamente a cualquier cambio que se presente.

El uso de Teams también nos permite almacenar toda la documentación relevante del proyecto, como planos, guías de diseño, especificaciones, y más, lo que facilita la consulta y el trabajo conjunto en cualquier momento.

Con este enfoque de colaboración y monitoreo constante, aseguramos que el proyecto avance de manera efectiva y eficiente, cumpliendo con los objetivos establecidos en tiempo y forma.

Link del reporte semanal: Reporte Semanal.docx

Link del equipo de Teams: Proyecto Catedra DSS | General | Microsoft Teams

2.6.2 Canales de Resolución de Dudas:

Como canal principal para resolver dudas y consultas, utilizaremos el **correo electrónico**. A través de este medio, los miembros del equipo pueden plantear cualquier inquietud relacionada con el proyecto, ya sea sobre tareas específicas, plazos, o cualquier otro aspecto que requiera aclaración. Aunque de momento no hemos tenido dudas significativas, el correo electrónico sigue siendo nuestra herramienta preferida para comunicarnos con usted ingeniero.

2.7. Gestión de Adquisición y Cierre

- ✓ **Documentación**

Necesitamos entregar dos cosas principales: un archivo PDF y el README detallado en GitHub. Todo esto debe estar bien organizado y accesible para la revisión. Les comento lo que debe incluir cada uno:

- ✓ **Archivo PDF:**

  - **Integrantes:** Hay que incluir la lista de todos los miembros del equipo con sus respectivas tareas y responsabilidades en el proyecto.
  - **Links a Notion/Trello:** Debemos poner los enlaces a las herramientas de gestión de tareas (Notion o Trello) que hemos estado usando. Esto tiene que incluir el acceso directo a las tareas y cualquier cosa que haya sido relevante para el progreso del proyecto.
  - **Licencia Creative Commons:** En el archivo también debemos incluir la licencia bajo la cual estamos liberando el proyecto, preferentemente Creative Commons si es un proyecto de código abierto.
  - **Confirmación de entregables en GitHub:** Asegurémonos de que todos los entregables están en el repositorio de GitHub. Esto incluye el código, la documentación y todo lo necesario.
  - **Verificación del despliegue en Kubernetes:** Hay que revisar que el clúster de Kubernetes esté funcionando bien y que la aplicación esté desplegada correctamente. Esto es clave para cerrar el proyecto.

- ✓ **README Detallado en GitHub:**

  El README debe estar super claro y tiene que contener:

  - **Descripción del Proyecto:** Explicar de forma concisa qué hace el proyecto.
  - **Requisitos:** Incluir todas las dependencias y herramientas necesarias para ejecutar el proyecto.
  - Instrucciones de Instalación y Ejecución: Explicar paso a paso cómo configurar el entorno local y desplegarlo en Kubernetes.
  - **Enlace al repositorio:** Asegurémonos de que el enlace al repositorio de GitHub esté accesible.
  - **Contribuciones**: Si alguien más quiere colaborar, que sepan cómo hacerlo.
  - **Licencia:** No olvidemos mencionar la licencia, en este caso, Creative Commons.
  - **Enlaces a Notion/Trello:** También tenemos que poner los enlaces a las herramientas que usamos para la gestión del proyecto.
  - **Link de Github**: https://github.com/DerekMarceloMongeAguilar/DSS-Catedra.git

- ✓ **Proceso de Cierre**

Al momento de cerrar el proyecto, tenemos que asegurarnos de que todo esté listo y validado:

- **Confirmación de los entregables en GitHub:** Revisen que todo esté en el repositorio. Eso incluye el código, documentos y cualquier archivo relacionado. Si hay algún pull request pendiente, hay que asegurarnos de que esté fusionado y que el código esté limpio.

- **Verificación del Despliegue en Kubernetes:** Tenemos que asegurarnos de que el clúster de Kubernetes esté funcionando sin problemas. Revisemos que la aplicación sea accesible, sin errores en el acceso y que los contenedores estén configurados correctamente.

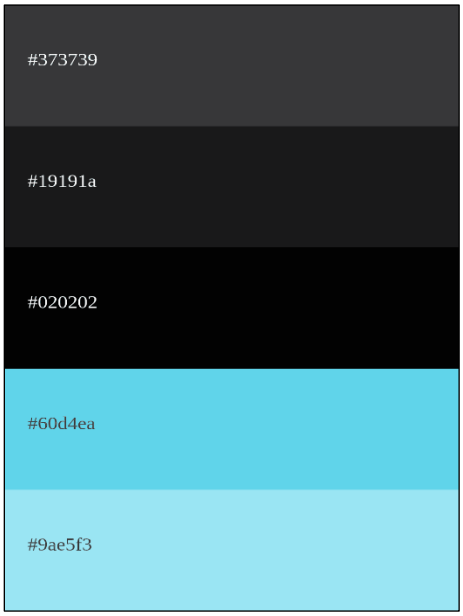
3. Descripción Técnica

3.1 Diseño de la Aplicación

Como equipo decidimos utilizar la herramienta de Figma para diseñar nuestros mockups, ya que es muy fácil de usar y permite que todos los integrantes del equipo puedan participar, la topografía que utilizamos es **Istok Web**, la paleta de colores que utilizamos fue colores negros y también una paleta de colores celestes y azules.

3.1.1 Mock Ups UX/UI:

.



Link donde esta los mockups: https://www.figma.com/design/QTD1a4DyHPgG9GSRxmKqYD/Untitled?node-id=0-1&t=Aaj0X7RgQcH1Z55m-1

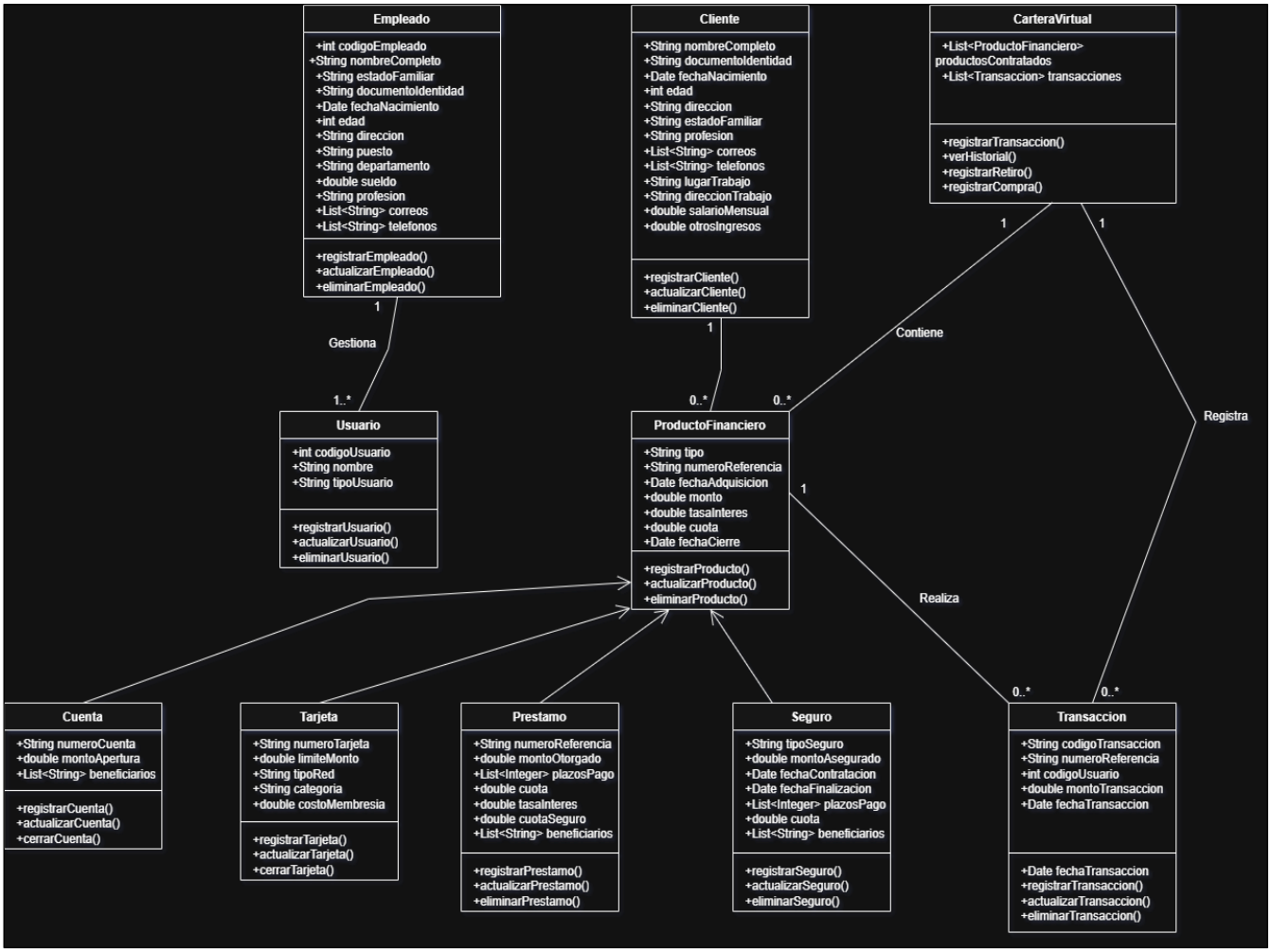
En nuestros Mockups se pueden visualizar todas las pantallas las que nosotros vamos a realizar en el sistema, por ejemplo, como el registro de empleado y del cliente, etc.

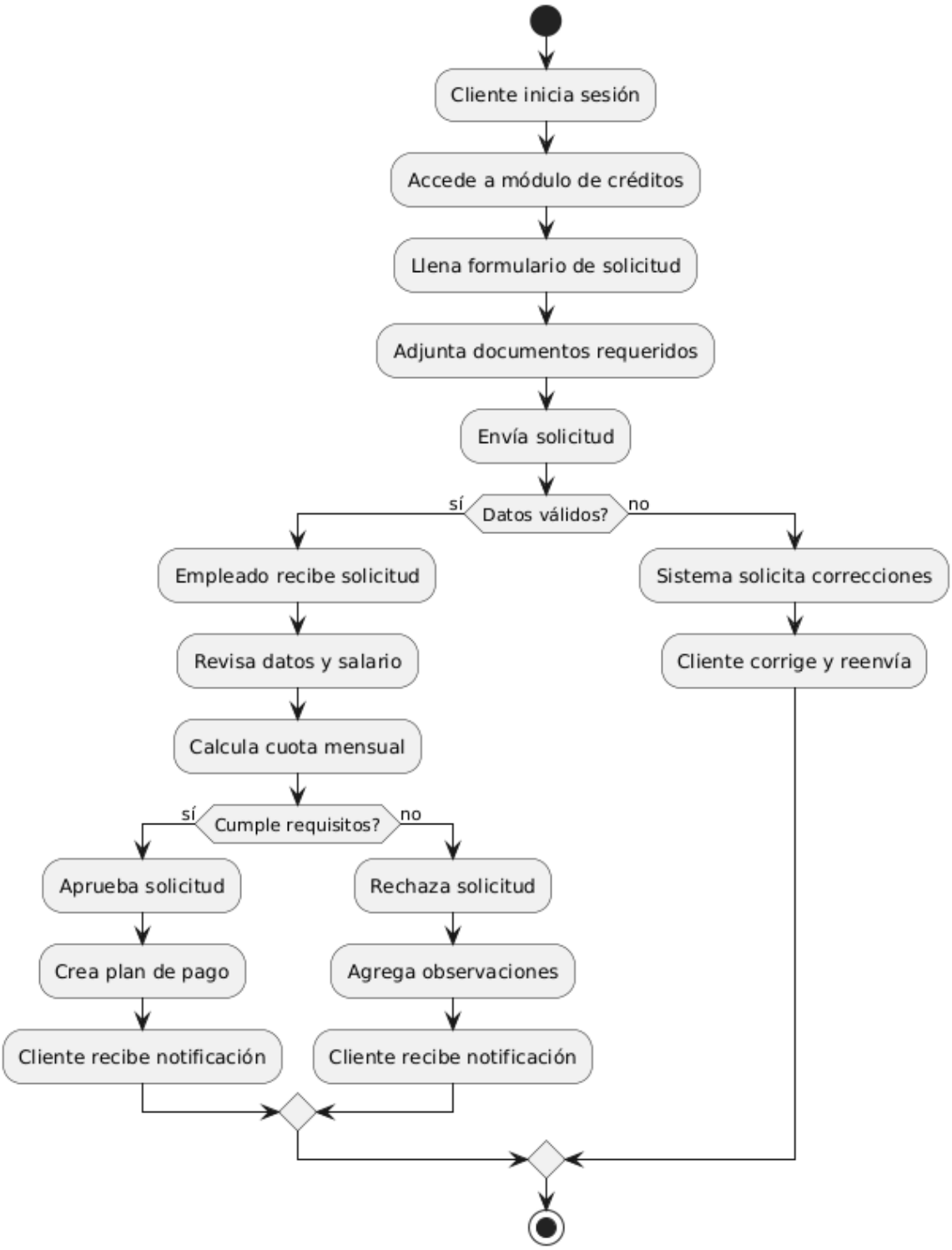
Tenemos pensado que en una sola página aparezca en la lista de los datos registrados y que a partir de un modelo se pueden agregar todos los datos del registro a crear, además de poder realizar todas las funciones de un CRUD como en el siguiente ejemplo:

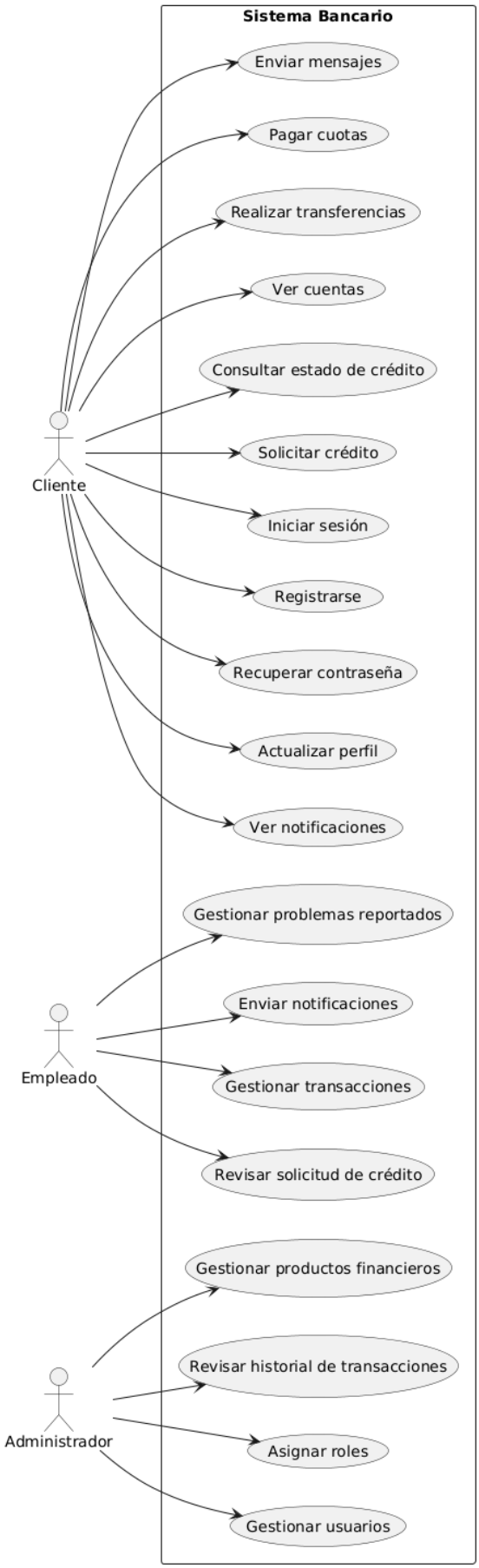
Este sería un pequeño vistazo de nuestro modelo donde se podría agregar toda la información de registro.

En esta fotografía se puede verificar la lista donde se encontrarán todos los registros guardados y se podrá agregar desde ese formulario otro registro, además de eliminar cualquier registro y editar cualquier registro seleccionado.

3.1.2 Diagramas UML



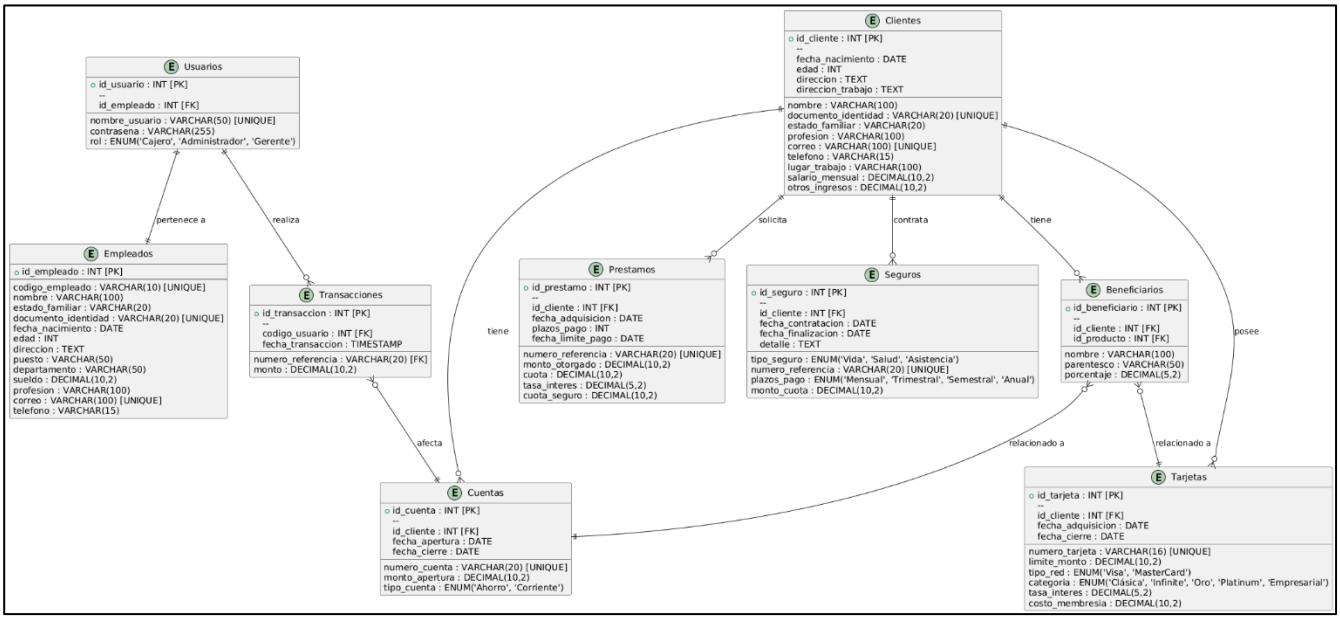

Cliente inicia sesión
Accede a módulo de créditos
Llena formulario de solicitud
Adjunta documentos requeridos
Envía solicitud
Datos válidos?
sí
no
Empleado recibe solicitud
Revisa datos y salario
Calcula cuota mensual
Cumple requisitos?
sí
no
Aprueba solicitud
Crea plan de pago
Cliente recibe notificación
Rechaza solicitud
Agrega observaciones
Cliente recibe notificación
Sistema solicita correcciones
Cliente corrige y reenvía


Sistema Bancario
Enviar mensajes
Pagar cuotas
Realizar transferencias
Ver cuentas
Consultar estado de crédito
Solicitar crédito
Iniciar sesión
Registrarse
Recuperar contraseña
Actualizar perfil
Ver notificaciones
Cliente
Gestionar problemas reportados
Enviar notificaciones
Gestionar transacciones
Revisar solicitud de crédito
Empleado
Gestionar productos financieros
Revisar historial de transacciones
Asignar roles
Gestionar usuarios
Administrador

4. Gestión de la Base de Datos

4.1 Estructura de la Base de Datos:

Diagrama entidad-relación (ER).



4.2 Diccionario de datos:

Campo	Tipo de Datos	Descripción	Tamaño	Restricciones
id_empleado	INT	Identificador único del empleado.	-	PRIMARY KEY
codigo_empleado	VARCHAR(10)	Código único para identificar al empleado.	10	UNIQUE
nombre	VARCHAR(100)	Nombre completo del empleado.	100	-
estado_familiar	VARCHAR(20)	Estado civil del empleado.	20	-
documento_identidad	VARCHAR(20)	Documento de identidad del empleado.	20	UNIQUE
fecha_nacimiento	DATE	Fecha de nacimiento del empleado.	-	-
edad	INT	Edad del empleado.	-	-
direccion	TEXT	Dirección del empleado.	-	-
puesto	VARCHAR(50)	Puesto del empleado en la empresa.	50	-
departamento	VARCHAR(50)	Departamento al que pertenece el empleado.	50	-
sueldo	DECIMAL(10,2)	Sueldo del empleado.	10,2	-
profesion	VARCHAR(100)	Profesión del empleado.	100	-
correo	VARCHAR(100)	Correo electrónico del empleado.	100	UNIQUE
telefono	VARCHAR(15)	Teléfono del empleado.	15	-

1. **Tabla: Empleados**

Relaciones:

Usuarios tiene una clave foránea *id_empleado* que hace referencia a *id_empleado* en esta tabla.

2. Tabla: Clientes

Campo	Tipo de Datos	Descripción	Tamaño	Restricciones
id_cliente	INT	Identificador único del cliente.	-	PRIMARY KEY
nombre	VARCHAR(100)	Nombre completo del cliente.	100	-
documento_identidad	VARCHAR(20)	Documento de identidad del cliente.	20	UNIQUE
fecha_nacimiento	DATE	Fecha de nacimiento del cliente.	-	-
edad	INT	Edad del cliente.	-	-
direccion	TEXT	Dirección del cliente.	-	-
estado_familiar	VARCHAR(20)	Estado civil del cliente.	20	-
profesion	VARCHAR(100)	Profesión del cliente.	100	-
correo	VARCHAR(100)	Correo electrónico del cliente.	100	UNIQUE
telefono	VARCHAR(15)	Teléfono del cliente.	15	-
lugar_trabajo	VARCHAR(100)	Lugar de trabajo del cliente.	100	-
direccion_trabajo	TEXT	Dirección del lugar de trabajo del cliente.	-	-
salario_mensual	DECIMAL(10,2)	Salario mensual del cliente.	10,2	-
otros_ingresos	DECIMAL(10,2)	Otros ingresos del cliente.	10,2	-

Relaciones:

Cuentas tiene una clave foránea *id_cliente* que hace referencia a *id_cliente* en esta tabla.

Tarjetas tiene una clave foránea *id_cliente* que hace referencia a *id_cliente* en esta tabla.

Prestamos tiene una clave foránea *id_cliente* que hace referencia a *id_cliente* en esta tabla.

Seguros tiene una clave foránea *id_cliente* que hace referencia a *id_cliente* en esta tabla.

Beneficiarios tiene una clave foránea *id_cliente* que hace referencia a *id_cliente* en esta tabla.

3. Tabla: Cuentas

Campo	Tipo de Datos	Descripción	Tamaño	Restricciones	Relaciones
id_cuenta	INT	Identificador único de la cuenta.	-	PRIMARY KEY	-
id_cliente	INT	Referencia al cliente dueño de la cuenta.	-	FOREIGN KEY (Clientes)	Clientes
numero_cuenta	VARCHAR(20)	Número único de la cuenta.	20	UNIQUE	-
fecha_apertura	DATE	Fecha de apertura de la cuenta.	-	-	-
monto_apertura	DECIMAL(10,2)	Monto inicial de la cuenta.	10,2	-	-
fecha_cierre	DATE	Fecha de cierre de la cuenta.	-	-	-
tipo_cuenta	ENUM('Ahorro', 'Corriente')	Tipo de cuenta (ahorro o corriente).	-	-	-

Relaciones:

Clientes tiene una clave foránea *id_cliente* que hace referencia a *id_cliente* en esta tabla.

Transacciones tiene una clave foránea *numero_referencia* que hace referencia a *numero_cuenta* en esta tabla.

Campo	Tipo de Datos	Descripción	Tamaño	Restricciones	Relaciones
id_tarjeta	INT	Identificador único de la tarjeta.	-	PRIMARY KEY	-
id_cliente	INT	Referencia al cliente dueño de la tarjeta.	-	FOREIGN KEY (Clientes)	**Clientes**
numero_tarjeta	VARCHAR(16)	Número único de la tarjeta.	16	UNIQUE	-
fecha_adquisicion	DATE	Fecha en que se adquirió la tarjeta.	-	-	-
limite_monto	DECIMAL(10,2)	Límite de crédito de la tarjeta.	10,2	-	-
fecha_cierre	DATE	Fecha de cierre de la tarjeta.	-	-	-
tipo_red	ENUM('Visa', 'MasterCard')	Tipo de red de la tarjeta. (Visa, MasterCard)	-	-	-
categoria	ENUM('Clásica', 'Infinite', 'Oro', 'Platinum', 'Empresarial')	Categoría de la tarjeta.	-	-	-
tasa_interes	DECIMAL(5,2)	Tasa de interés asociada a la tarjeta.	5,2	-	-
costo_membresia	DECIMAL(10,2)	Costo anual de la membresía de la tarjeta.	10,2	-	-

4. Tabla: Tarjetas

Relaciones:

Clientes tiene una clave foránea *id_cliente* que hace referencia a *id_cliente* en esta tabla.

Transacciones tiene una clave foránea *numero_referencia* que hace referencia a *numero_tarjeta* en esta tabla.

5. Tabla: Prestamos

Campo	Tipo de Datos	Descripción	Tamaño	Restricciones	Relaciones
id_prestamo	INT	Identificador único del préstamo.	-	PRIMARY KEY	-
id_cliente	INT	Referencia al cliente que solicita el préstamo.	-	FOREIGN KEY (Clientes)	**Clientes**
numero_referencia	VARCHAR(20)	Número de referencia del préstamo.	20	UNIQUE	-
fecha_adquisicion	DATE	Fecha en que se adquirió el préstamo.	-	-	-
monto_otorgado	DECIMAL(10,2)	Monto total otorgado en el préstamo.	10,2	-	-
plazos_pago	INT	Número de plazos para el pago del préstamo.	-	-	-
cuota	DECIMAL(10,2)	Cuota mensual a pagar en el préstamo.	10,2	-	-
fecha_limite_pago	DATE	Fecha límite para el pago del préstamo.	-	-	-
tasa_interes	DECIMAL(5,2)	Tasa de interés aplicada al préstamo.	5,2	-	-
cuota_seguro	DECIMAL(10,2)	Cuota mensual del seguro asociado al préstamo.	10,2	-	-

Relaciones:

Clientes tiene una clave foránea *id_cliente* que hace referencia a *id_cliente* en esta tabla.

Transacciones tiene una clave foránea *numero_referencia* que hace referencia a *numero_referencia* en esta tabla.

6. Tabla: Seguros

Campo	Tipo de Datos	Descripción	Tamaño	Restricciones	Relaciones
id_seguro	INT	Identificador único del seguro.	-	PRIMARY KEY	-
id_cliente	INT	Referencia al cliente que contrata el seguro.	-	FOREIGN KEY (Clientes)	**Clientes**
tipo_seguro	ENUM('Vida', 'Salud', 'Asistencia')	Tipo de seguro (vida, salud, asistencia).	-	-	-
numero_referencia	VARCHAR(20)	Número de referencia del seguro.	20	UNIQUE	-
fecha_contratacion	DATE	Fecha de contratación del seguro.	-	-	-
fecha_finalizacion	DATE	Fecha de finalización del seguro.	-	-	-
plazos_pago	ENUM('Mensual', 'Trimestral', 'Semestral', 'Anual')	Periodicidad de pago del seguro.	-	-	-
monto_cuota	DECIMAL(10,2)	Monto de la cuota del seguro.	10,2	-	-
detalle	TEXT	Descripción adicional sobre el seguro.	-	-	-

Relaciones:

Clientes tiene una clave foránea *id_cliente* que hace referencia a *id_cliente* en esta tabla.

7. Tabla: Beneficiarios

Campo	Tipo de Datos	Descripción	Tamaño	Restricciones	Relaciones
id_beneficiario	INT	Identificador único del beneficiario.	-	PRIMARY KEY	-
id_cliente	INT	Referencia al cliente al que pertenece el beneficiario.	-	FOREIGN KEY (Clientes)	**Clientes**
id_producto	INT	Referencia al producto relacionado (puede ser préstamo, seguro, cuenta, etc.).	-	FOREIGN KEY (productos)	**Productos** (depende del contexto de la relación)
nombre	VARCHAR(100)	Nombre del beneficiario.	100	-	-
parentesco	VARCHAR(50)	Parentesco con el cliente.	50	-	-
porcentaje	DECIMAL(5,2)	Porcentaje asignado del producto.	5,2	-	-

Relaciones:

Clientes tiene una clave foránea *id_cliente* que hace referencia a *id_cliente* en esta tabla.

La relación con productos depende de la entidad a la que se asocie (como **Seguros, Prestamos, Cuentas,** etc.).

8. Tabla: Transacciones

Campo	Tipo de Datos	Descripción	Tamaño	Restricciones	Relaciones
id_transaccion	INT	Identificador único de la transacción.	-	PRIMARY KEY	-
numero_referencia	VARCHAR(20)	Número de referencia asociado a la transacción.	20	FOREIGN KEY (Cuentas, Tarjetas, Prestamos)	**Cuentas**, **Tarjetas**, **Prestamos**
codigo_usuario	INT	Referencia al usuario que realiza la transacción.	-	FOREIGN KEY (Usuarios)	**Usuarios**
monto	DECIMAL(10,2)	Monto de la transacción.	10,2	-	-
fecha_transaccion	TIMESTAMP	Fecha y hora en que se realiza la transacción.	-	-	-

Relaciones:

Usuarios tiene una clave foránea *codigo_usuario* que hace referencia a *id_usuario* en esta tabla.

Cuentas, Tarjetas, y Prestamos pueden estar relacionados con esta tabla a través del campo *numero_referencia*.

9. Tabla: Usuarios

Campo	Tipo de Datos	Descripción	Tamaño	Restricciones	Relaciones
id_usuario	INT	Identificador único del usuario.	-	PRIMARY KEY	-
id_empleado	INT	Referencia al empleado que administra el usuario.	-	FOREIGN KEY (Empleados)	**Empleados**
nombre_usuario	VARCHAR(50)	Nombre de usuario del empleado.	50	UNIQUE	-
contrasena	VARCHAR(255)	Contraseña del usuario.	255	-	-
rol	ENUM('Cajero', 'Administrador', 'Gerente')	Rol del usuario dentro del sistema.	-	-	-

Relaciones:

Empleados tiene una clave foránea *id_empleado* que hace referencia a *id_empleado* en esta tabla.

5. Desarrollo del Proyecto:

5.1 Requerimientos de los usuarios utilizando la técnica de “How might We”

Posibles problemas o desafíos actuales:

- **Registro de empleados:**

El proceso de registro de empleados es complejo y requiere mucho tiempo.

La información de los empleados no está organizada de manera eficiente.

- **Registro de clientes:**

El proceso de registro de clientes es lento y propenso a errores.

Los clientes tienen dificultades para actualizar su información personal.

- **Registro de productos financieros:**

Los clientes tienen dificultades para gestionar sus productos financieros.

La información sobre productos financieros no está fácilmente accesible.

- **Registro de transacciones de productos:**

Los clientes no pueden verificar fácilmente sus transacciones.

El sistema de transacciones no es intuitivo.

- **Cartera virtual del cliente:**

Los clientes tienen dificultades para visualizar y gestionar sus productos financieros.

La interfaz de la cartera virtual no es amigable.

- **Control de usuarios:**

El control de usuarios no está optimizado según los roles y permisos.

Los empleados tienen dificultades para acceder a la información necesaria.

Ahora nos hacemos algunas preguntas con la técnica solicitada:

- **Registro de empleados:**

¿HMW mejorar la eficiencia del registro de empleados?

¿HMW organizar mejor la información de los empleados?

- **Registro de clientes:**

¿HMW agilizar el proceso de registro de clientes?

¿HMW facilitar la actualización de la información personal de los clientes?

- **Registro de productos financieros:**

HMW facilitar la gestión de productos financieros para los clientes?

HMW hacer que la información sobre productos financieros sea más accesible?

- **Registro de transacciones de productos:**

¿HMW permitir a los clientes verificar sus transacciones de manera sencilla y rápida?

¿HMW hacer que el sistema de transacciones sea más intuitivo?

- **Cartera virtual del cliente:**

¿HMW mejorar la visualización y gestión de productos financieros en la cartera virtual?

¿HMW hacer que la interfaz de la cartera virtual sea más amigable?

- **Control de usuarios:**

¿HMW optimizar el control de usuarios según sus roles y permisos?

¿HMW facilitar el acceso a la información necesaria para los empleados?

Ahora proveemos posibles herramientas y soluciones a cada desafío:

- **Registro de empleados:**

Implementar formularios dinámicos que se adapten según el tipo de empleado.

Utilizar validación en tiempo real para asegurar que los datos ingresados sean correctos.

Integrar la opción de importar datos desde archivos CSV para registros masivos.

- **Registro de clientes:**

Implementar un sistema de registro rápido con autocompletado de datos.

Permitir a los clientes actualizar su información personal a través de una interfaz sencilla.

Utilizar validación en tiempo real para reducir errores en el registro.

- **Registro de productos financieros:**

Crear un panel de control intuitivo donde los clientes puedan ver y gestionar todos sus productos financieros.

Ofrecer tutoriales y guías en línea para ayudar a los clientes a entender cómo usar el sistema.

Implementar notificaciones automáticas para recordar a los clientes sobre pagos y vencimientos.

- **Registro de transacciones de productos:**

Desarrollar una función de búsqueda avanzada para que los clientes puedan encontrar transacciones específicas.

Mostrar un historial de transacciones claro y detallado en el panel de control del cliente.

Permitir a los clientes descargar reportes de sus transacciones en formato PDF o Excel.

- **Cartera virtual del cliente:**

Mejorar la visualización de productos financieros con gráficos y estadísticas.

Implementar una interfaz amigable y fácil de usar para la cartera virtual.

Permitir a los clientes realizar transacciones directamente desde la cartera virtual.

- **Control de usuarios:**

Definir roles y permisos claros para cada tipo de usuario (administrador, cajero, cliente, etc.).

Implementar un sistema de gestión de usuarios que permita asignar y modificar roles fácilmente.

Realizar capacitaciones periódicas para asegurar que todos los usuarios entiendan sus roles y responsabilidades.

5.2 Implementación del Backend (PHP):

Configuración del entorno en Windows 10

- **Instalar PHP:**

Descargar el instalador de PHP desde php.net.

Ejecutar el instalador y seguir las instrucciones para instalar PHP en nuestro sistema.

- **Instalar Composer:**

Descargar el instalador de Composer desde getcomposer.org.

Ejecutar el instalador y seguit las instrucciones para instalar Composer en nuestro sistema.

- **Configurar Apache:**

Descargar e instalar XAMPP desde apachefriends.org.

Abrir el panel de control de XAMPP y asegurarse de que Apache y MySQL estén ejecutándose.

- **Instalar dependencias:**

Crear un archivo composer.json en la raíz de nuestro proyecto y definir las dependencias necesarias.

```
{
  "require": {
    "php": "^8.1",
    "slim/slim": "^4.8",
    "slim/psr7": "^1.4",
    "monolog/monolog": "^2.3"
  }
}
```

Abrir una terminal en la carpeta de nuestro proyecto y ejecutar *composer install* para instalar las dependencias.

5.3 Desarrollo de la lógica de la aplicación

- **Estructura del proyecto:**

Crear las siguientes carpetas: src, public, config, logs.

En public, crear un archivo index.php que será el punto de entrada de la aplicación.

- **Configuración de Slim Framework:**

En index.php, configurar Slim para manejar las rutas y controladores.

```
<?php
require __DIR__ . '/../vendor/autoload.php';

use Slim\Factory\AppFactory;

$app = AppFactory::create();

$app->get('/employees', function ($request, $response, $args) {
    // Lógica para obtener empleados
    return $response;
});

$app->run();
```

- **Conexión a la base de datos:**

Crear un archivo config/db.php para manejar la conexión a la base de datos.

```
<?php
$host = 'localhost';
$db = 'acoemprendedores';
$user = 'root';
$pass = 'password';

$dsn = "mysql:host=$host;dbname=$db;charset=utf8mb4";
try {
    $pdo = new PDO($dsn, $user, $pass);
} catch (PDOException $e) {
    echo 'Connection failed: ' . $e->getMessage();
}
```

5.4 Desarrollo del Frontend (PHP)

Configuración del entorno en Windows 10

- **Instalar Node.js y npm:**

Descargar el instalador de Node.js desde nodejs.org.

Ejecutar el instalador y seguir las instrucciones para instalar Node.js y npm en nuestro sistema.

- **Configurar el proyecto:**

Crear un archivo package.json en la raíz del proyecto y definir las dependencias necesarias.

```
{
  "name": "frontend",
  "version": "1.0.0",
  "scripts": {
    "start": "webpack serve --mode development",
    "build": "webpack --mode production"
  },
  "dependencies": {
    "bootstrap": "^5.1.3",
    "jquery": "^3.6.0"
  },
  "devDependencies": {
    "webpack": "^5.64.4",
    "webpack-cli": "^4.9.1",
    "webpack-dev-server": "^4.7.4"
  }
}
```

Abrir una terminal en la carpeta de nuestro proyecto y ejecutar *npm install* para instalar las dependencias.

Desarrollo de la interfaz de usuario

- **Estructura del proyecto:**

Crear las siguientes carpetas: src, dist, assets.

En src, crear archivos index.html, styles.css, y app.js.

- **Configurar Webpack:**

Crear un archivo webpack.config.js para manejar la configuración de Webpack.

```
const path = require('path');

module.exports = {
  entry: './src/app.js',
  output: {
    filename: 'bundle.js',
    path: path.resolve(__dirname, 'dist')
  },
  module: {
    rules: [
      {
        test: /\.css$/,
        use: ['style-loader', 'css-loader']
      }
    ]
  },
  devServer: {
    contentBase: path.join(__dirname, 'dist'),
    compress: true,
    port: 9000
  }
};
```

- **Desarrollar la interfaz:**

En index.html, crear la estructura básica de la interfaz.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>ACOEMPRENDEDORES</title>
  <link rel="stylesheet" href="styles.css">
</head>
<body>
  <div class="container">
    <h1>Registro de Empleados</h1>
    <form id="employeeForm">
      <!-- Campos del formulario -->
    </form>
  </div>
  <script src="bundle.js"></script>
</body>
</html>
```

5.5 Integración de Docker y Kubernetes

Implementación y pruebas de contenedores

- **Instalar Docker Desktop:**

Descargar Docker Desktop desde docker.com.

Ejecutar el instalador y seguir las instrucciones para instalar Docker Desktop en nuestro sistema.

- **Crear Dockerfile:**

En la raíz del proyecto, crear un archivo Dockerfile para definir la imagen del contenedor.

```
FROM php:8.1-apache
COPY . /var/www/html
RUN docker-php-ext-install pdo pdo_mysql
```

- **Crear archivo docker-compose.yml:**

Definir los servicios necesarios para ejecutar el sistema.

```
version: '3.8'
services:
  php:
    build: .
    container_name: backend_php
    volumes:
      - .:/var/www/html
    ports:
      - "8080:80"
    depends_on:
      - db
  db:
    image: mysql:5.7
    container_name: mysql_db
    environment:
      MYSQL_ROOT_PASSWORD: root
      MYSQL_DATABASE: acoemprendedores
      MYSQL_USER: user
      MYSQL_PASSWORD: password
    ports:
      - "3306:3306"
```

- **Ejecutar Docker Compose:**

Abrir una terminal en la carpeta de nuestro proyecto y ejecutar:

```
docker-compose up -d
```

Despliegue en Kubernetes

- **Instalar Kubernetes:**

Descargar e instalar Minikube desde minikube.sigs.k8s.io.

Ejecutar Minikube para iniciar un clúster local:

```
minikube start
```

- **Crear archivo YAML para el despliegue:**

Definir los componentes del sistema en Kubernetes en un archivo, por ejemplo, backend-deployment.yaml.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: backend-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: backend
  template:
    metadata:
      labels:
        app: backend
    spec:
      containers:
        - name: backend
          image: mi-registro/backend:v1
          ports:
            - containerPort: 80
```

- **Crear servicio para exponer el backend:**

Definir el servicio en un archivo YAML, por ejemplo, backend-service.yaml.

```yaml
apiVersion: v1
kind: Service
metadata:
  name: backend-service
spec:
  type: NodePort
  selector:
    app: backend
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
      nodePort: 30080
```

- **Aplicar los archivos YAML en Kubernetes:**

Abrir una terminal y navegar hasta la carpeta donde guardamos los archivos backend-deployment.yaml y backend-service.yaml.

Ejecutar los siguientes comandos para aplicar los archivos y crear el despliegue y el servicio:

```
kubectl apply -f backend-deployment.yaml
kubectl apply -f backend-service.yaml
```

- **Verificar el despliegue y el servicio:**

Ejecutar los siguientes comandos para verificar que el despliegue y el servicio se hayan creado correctamente:

```
kubectl get deployments
kubectl get services
```

- **Finalmente, deberíamos ver una salida similar a esta:**

```
NAME                READY   UP-TO-DATE   AVAILABLE   AGE
backend-deployment  2/2     2            2           1m

NAME              TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
backend-service   NodePort    10.96.0.1       <none>        80:30080/TCP   1m
```

6. Pruebas y Control de Calidad

Pruebas unitarias (PHP): formalizaremos las herramientas de prueba de PHP, definiendo que es lo que se probara escribiendo pruebas claras y fáciles de seguir, utilizando mocks y stubs para simular comportamientos.

Para comenzar utilizaremos PHPUnit, instalándolo en el proyecto PHP mediante Composer. Luego, creamos las clases de prueba y utilizamos las aserciones de PHPUnit para verificar los resultados correctos de tus funciones y métodos. Para escribir una prueba unitaria en PHPUnit, crea una clase que extienda PHPUnit_Framework_TestCase y definiremos los diferentes escenarios de prueba que se desean probar.

Pruebas de integración (Docker): Se analizará y comprenderá el flujo del código, seleccionando escenarios relevantes para La cooperativa "ACOEMPRENDEDORES" configurando un entorno de producción y desarrollando casos de prueba exhaustivos para cubrir todos los escenarios. Añadiremos test de integración y los test unitarios en conjunto, La forma más sencilla de ver esto es con un ejemplo:

El punto principal de un test unitario es verificar que una pequeña parte de nuestro proyecto, como puede ser una función para calcular la mayoría de edad, funciona correctamente.

```
bool EsMayorDeEdad(int edad) => edad >= 18;
```

Por otra parte, cuando ejecutamos un test de integración queremos saber que ningún usuario menor de edad puede acceder a la web.

Pruebas de carga (Kubernetes): Se realizará una simulación de carga con usuarios concurrentes interactuando con servicios en ejecución en pods de Kubernetes, siguiendo métricas de Recursos Monitorea el uso de CPU, memoria y E/S de red durante las pruebas para detectar cuellos de botella de rendimiento e identificar puntos críticos, identificando alertas de Rendimiento, análisis de escalabilidad y interpretación de resultados.

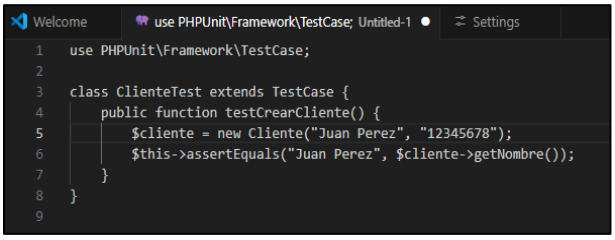
6.1 Pruebas Unitarias y de Integración

Se utilizará PHPUnit como habíamos detallado para las pruebas unitarias y esto para validar las funciones del backend, aplicando a los módulos clave del sistema, como:

Gestión de Clientes: Verificar que los clientes se registran correctamente

Gestión de Transacciones: Validar que las operaciones de abono y retiro afectan los saldos correctamente.

Un ejemplo de una prueba unitaria en PHP (simulación en PHPUnit):



Pruebas de Integración (Docker): Las pruebas de integración verificarán que los distintos módulos del sistema funcionen correctamente dentro de contenedores Docker. Se enfocarán en:

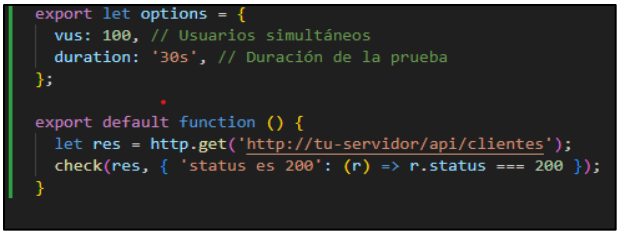
Conexión entre el backend PHP y la base de datos MySQL.

Comunicación entre el frontend y el backend.

Para probar la integración, se usarán los comandos:

- docker-compose up -d
- curl -X GET http://localhost/api/clientes

Para evaluar el rendimiento y la escalabilidad del sistema en Kubernetes, se realizaron pruebas de carga utilizando herramientas como **k6** o **Apache JMeter**. Un ejemplo de cómo quedaría sería el siguiente:



7. Resultados y Conclusiones

- ✓ **Resultados Esperados:**

Dado que esta fase del proyecto se centra en la planificación e implementación inicial, los resultados esperados de las pruebas y control de calidad son:

- **Validación del código:** mediante pruebas unitarias para asegurar la correcta ejecución de las funciones del backend.

- **Integración exitosa:** entre los módulos del sistema utilizando contenedores Docker, garantizando comunicación entre backend, base de datos y frontend.

- **Evaluación del rendimiento del sistema**: mediante pruebas de carga en Kubernetes, verificando tiempos de respuesta y estabilidad ante múltiples usuarios concurrentes.

Estos resultados permitirán una implementación más eficiente y reducirán errores en la segunda fase del proyecto.

Durante la fase de pruebas y control de calidad, se pueden enfrentar los siguientes desafíos:

- Compatibilidad del backend PHP con Docker y Kubernetes.
- Gestión eficiente de recursos en el clúster de Kubernetes.
- Optimización del tiempo de respuesta de la API para mejorar la experiencia del usuario.

Para mitigar estos problemas, se propone:

- Definir estándares de desarrollo y buenas prácticas en PHP.
- Realizar monitoreo del rendimiento con herramientas como **k6** y **Prometheus** en Kubernetes.
- Implementar caching y optimización en las consultas a la base de datos.

- ✓ **Conclusiones**

La primera fase del proyecto ha permitido definir una estrategia clara para el desarrollo e implementación del sistema de información de “ACOEMPRENDEDORES”. Se han establecido las bases para la contenerización con Docker y Kubernetes, así como la estructura de la base de datos y las pruebas de calidad.

- ✓ **Reflexión sobre el desarrollo**

Durante esta fase, se han identificado los principales retos del proyecto, tales como la integración de los contenedores, la gestión de datos en la base de datos y la implementación de pruebas de rendimiento. La planificación detallada de las pruebas garantizará que en la siguiente fase se puedan detectar errores de manera temprana y optimizar el desempeño del sistema.

- ✓ **Aprendizaje y mejoras identificadas**

El equipo ha aprendido sobre las mejores prácticas en la gestión de entornos de desarrollo basados en contenedores, la configuración de herramientas de automatización de pruebas y la organización eficiente de un sistema de información escalable. Se ha identificado la necesidad de optimizar consultas SQL y mejorar la configuración de Kubernetes para asegurar una mejor distribución de carga.

- ✓ **Siguientes pasos**

Para la segunda fase, se implementará la funcionalidad completa del sistema, se ejecutarán las pruebas documentadas y se afinará el desempeño en un entorno real. Se espera que la estructura planificada en esta fase sirva como base sólida para una implementación exitosa.

8. Anexos y Documentación de Apoyo

- ✓ **Documentación técnica adicional:**

Estas consultas serán utilizadas en la implementación del sistema para la gestión de clientes, empleados y transacciones.

- ✓ **Creación de la base de datos y tablas principales**

```
CREATE DATABASE ACOEMPRENDEDORES;

USE ACOEMPRENDEDORES;


CREATE TABLE Clientes (
        id INT AUTO_INCREMENT PRIMARY KEY,
        nombre VARCHAR(100) NOT NULL,
        documento_identidad VARCHAR(20) UNIQUE NOT NULL,
        fecha_nacimiento DATE NOT NULL,
        direccion TEXT,
        telefono VARCHAR(15),
        correo VARCHAR(100)
);


CREATE TABLE Empleados (
        id INT AUTO_INCREMENT PRIMARY KEY,
        codigo_empleado VARCHAR(10) UNIQUE NOT NULL,     nombre VARCHAR(100) NOT NULL,
```

```
    puesto VARCHAR(50),
    departamento VARCHAR(50),
    sueldo DECIMAL(10,2),
    correo VARCHAR(100),
    telefono VARCHAR(15)
);

CREATE TABLE Transacciones (
    id INT AUTO_INCREMENT PRIMARY KEY,
    cliente_id INT,
    tipo_transaccion ENUM('Abono', 'Retiro') NOT NULL,
    monto DECIMAL(10,2) NOT NULL,
    fecha TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (cliente_id) REFERENCES Clientes(id)
);
```

- ✓ **Ejemplo de inserciones de prueba**

```
INSERT INTO Clientes (nombre, documento_identidad, fecha_nacimiento, direccion, telefono, correo)
VALUES ('Juan Pérez', '12345678', '1990-05-15', 'Calle Falsa 123, San Salvador', '7890-1234', 'juanperez@mail.com');

INSERT INTO Empleados (codigo_empleado, nombre, puesto, departamento, sueldo, correo, telefono)
```

```
VALUES ('EMP001', 'Ana Martínez', 'Cajero', 'Atención al Cliente', 650.00, 'ana.martinez@empresa.com', '7891-2345');
```

- ✓ **Configuración Avanzada de Docker Compose**

Este archivo docker-compose.yml define los servicios necesarios para ejecutar el sistema en contenedores de esta forma:

```
version: '3.8'

services:

  php:

        image: php:8.1-apache

        container_name: backend_php

        volumes:

        - .:/var/www/html

        ports:

        - "8080:80"

        depends_on:

        - db


  db:

        image: mysql:5.7

        container_name: mysql_db

        restart: always

        environment:

        MYSQL_ROOT_PASSWORD: root
```

```
        MYSQL_DATABASE: acoemprendedores
        MYSQL_USER: user
        MYSQL_PASSWORD: password
        ports:
        - "3306:3306"

  phpmyadmin:
        image: phpmyadmin/phpmyadmin
        container_name: pma
        environment:
        PMA_HOST: db
        MYSQL_ROOT_PASSWORD: root
        ports:
        - "8081:80"
```

- ✓ **Configuración Avanzada de Kubernetes**

Para el despliegue en Kubernetes, se incluirá un archivo YAML que define los componentes del sistema.

```
Deployment para el Backend PHP
apiVersion: apps/v1
kind: Deployment
metadata:
  name: backend-deployment
spec:
```

```
 replicas: 2
 selector:
        matchLabels:
        app: backend
 template:
        metadata:
        labels:
        app: backend
        spec:
        containers:
        - name: backend
        image: mi-registro/backend:v1
        ports:
        - containerPort: 80
```

Service para Exponer el Backend

```
apiVersion: v1
kind: Service
metadata:
 name: backend-service
spec:
 type: NodePort
 selector:
        app: backend
```

```
ports:
    - protocol: TCP
    port: 80
    targetPort: 80
    nodePort: 30080
```

- ✓ **Herramientas y Recursos Utilizados**

Para facilitar la implementación y pruebas del sistema, se utilizamos el uso de las siguientes herramientas:

Herramienta	**Descripción**	**Enlace**
PHPUnit	Framework para pruebas unitarias en PHP	https://phpunit.de
Docker	Contenedorización de la aplicación	https://docs.docker.com
Kubernetes	Orquestación de contenedores	https://kubernetes.io
k6	Herramienta para pruebas de carga	https://k6.io
JMeter	Pruebas de rendimiento y carga	https://jmeter.apache.org