**PONTIFICIA UNIVERSIDAD JAVERIANA**

**INGENIERÍA DE SISTEMAS**

**FUNDAMENTOS DE INGENIERÍA DE SOFTWARE**

**GESTIÓN DE BANCO: PMP**

Imagen que contiene Patrón de fondo

El contenido generado por IA puede ser incorrecto.

**INTEGRANTES:**

**Santiago Andrés Mesa N.**

**Luis Enrique Santos Marulanda**

**Damián Rey Salcedo**

**Juan Diego Barreto Castañeda**

**PROFESOR:**

**Jaime Andrés Pavlich Mariscal.**

# Historial de Cambios

|  |  |  |
| --- | --- | --- |
| Fecha del cambio | Descripción del cambio | Responsable |
| 25/03/2025 | Creación de diagrama de paquetes | Santiago mesa |
| 1/04/2025 | Corrección de diagrama de paquetes | Santiago Mesa |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

# Tabla de Contenidos

Propósito: Encontrar rápidamente una sección específica del documento. El resumen, y las listas de figuras y tablas no se incluyen en esta sección. La tabla de contenidos comienza en la Introducción, que es la página uno del documento.

# Introducción

Propósito: que el lector determine si vale la pena seguir leyendo el documento.

Contenido: Una breve explicación de qué trata el resto del documento y cuáles son sus principales secciones y de qué trata cada una.

# Arquitectura

## Vista Lógica del Sistema

Esta vista tiene como objetivo que el desarrollador comprenda la estructura lógica del sistema bancario, identificando los principales componentes de software y sus responsabilidades, sin enfocarse aún en cómo serán desplegados físicamente. Se utiliza el patrón arquitectónico Modelo-Vista-Controlador (MVC), que permite una separación clara de responsabilidades entre la lógica de negocio, la interfaz gráfica y la gestión de eventos.

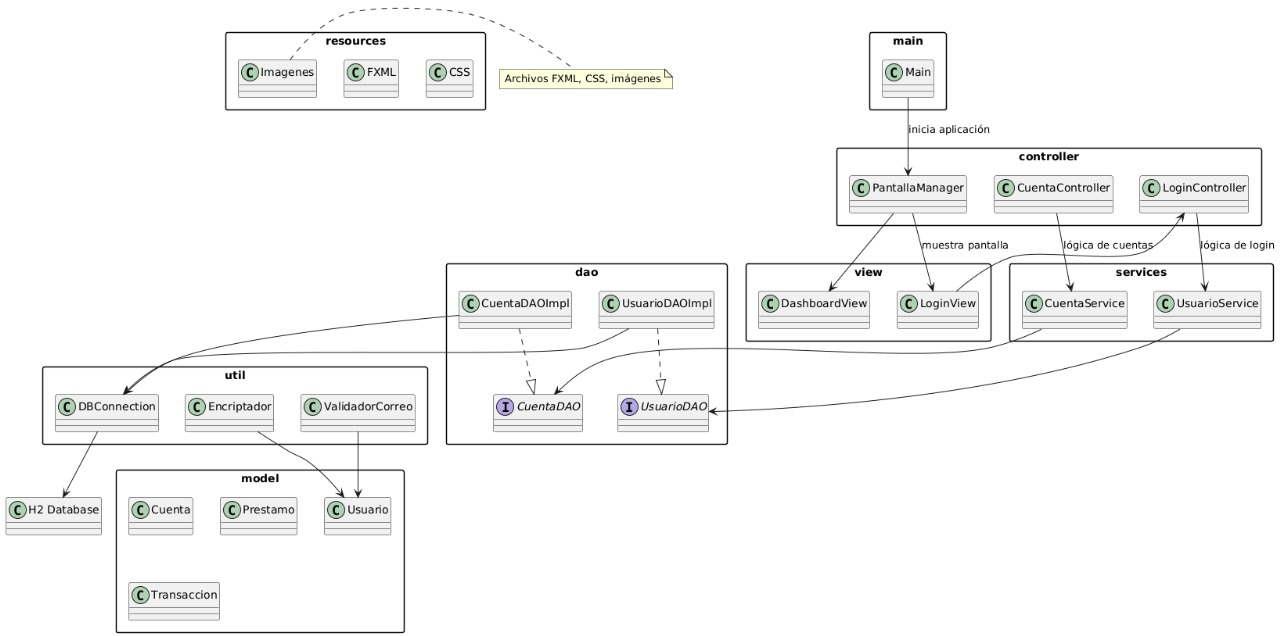
**Patrón arquitectónico: MVC**

El sistema se basa en el patrón **Modelo-Vista-Controlador**, ideal para proyectos desarrollados en **Java con JavaFX**, ya que facilita la escalabilidad, el mantenimiento del código y la colaboración en equipo.

* **Modelo:** Contiene la lógica de negocio y las entidades del dominio (por ejemplo: Usuario, Cuenta, Transaccion, etc.).
* **Vista:** Compuesta por interfaces gráficas en JavaFX (.fxml), muestra la información al usuario y captura las interacciones.
* **Controlador:** Coordina los eventos entre la vista y el modelo. Procesa la entrada del usuario y actualiza el modelo o la vista según corresponda.

|  |  |
| --- | --- |
| Paquete / Componente | Descripción |
| main | Punto de entrada de la aplicación (Main.java). Inicializa el entorno y carga la primera vista. |
| view | Contiene archivos: .fxml y clases JavaFX asociadas. Representa las pantallas del sistema. |
| controller | Clases que manejan eventos de UI y coordinan la lógica entre view y model. Ej: LoginController, CuentaController. |
| model | Entidades del dominio, lógica de negocio, validaciones. Ej: Usuario, Cuenta, Prestamo. |
| dao | Interfaces e implementaciones para acceso a datos en H2. Ej: UsuarioDAO, TransaccionDAOImpl. |
| util | Funciones auxiliares: validadores, formateadores, conexión a BD, cifrado, etc. |
| resources | Archivos estáticos (configuraciones, estilos CSS, imágenes, plantillas FXML). |

|  |  |
| --- | --- |
| Frameworks / Librerías | Descripción |
| JavaFX | Framework para la interfaz gráfica. |
| H2 Database | Motor de base de datos embebido. |
| JDBC | API para conectar Java con H2. |
| JUnit (opcional) | Para pruebas unitarias durante desarrollo |



El sistema está estructurado en capas siguiendo el patrón Modelo-Vista-Controlador (MVC) extendido con principios de diseño basado en dominio (DDD). Cada paquete agrupa componentes con responsabilidades específicas. El paquete view contiene las interfaces gráficas en JavaFX (archivos FXML y clases asociadas); controller gestiona eventos de la interfaz, coordina el flujo entre pantallas mediante la clase PantallaManager, e interactúa con la capa services para ejecutar la lógica de negocio. El paquete services encapsula los casos de uso del dominio, y se comunica con dao, que gestiona el acceso a datos mediante interfaces (UsuarioDAO, CuentaDAO) e implementaciones. Las entidades del sistema (Usuario, Cuenta, Prestamo, etc.) residen en el paquete model, mientras que util incluye clases auxiliares como DBConnection y ValidadorCorreo. La inyección de dependencias está representada explícitamente: los controladores no crean directamente instancias de DAOs ni servicios, sino que los reciben como dependencias, lo cual favorece la mantenibilidad, testeo e independencia de capas. Este diseño modular promueve la cohesión interna y el bajo acoplamiento entre componentes.

## Vista Física del Sistema

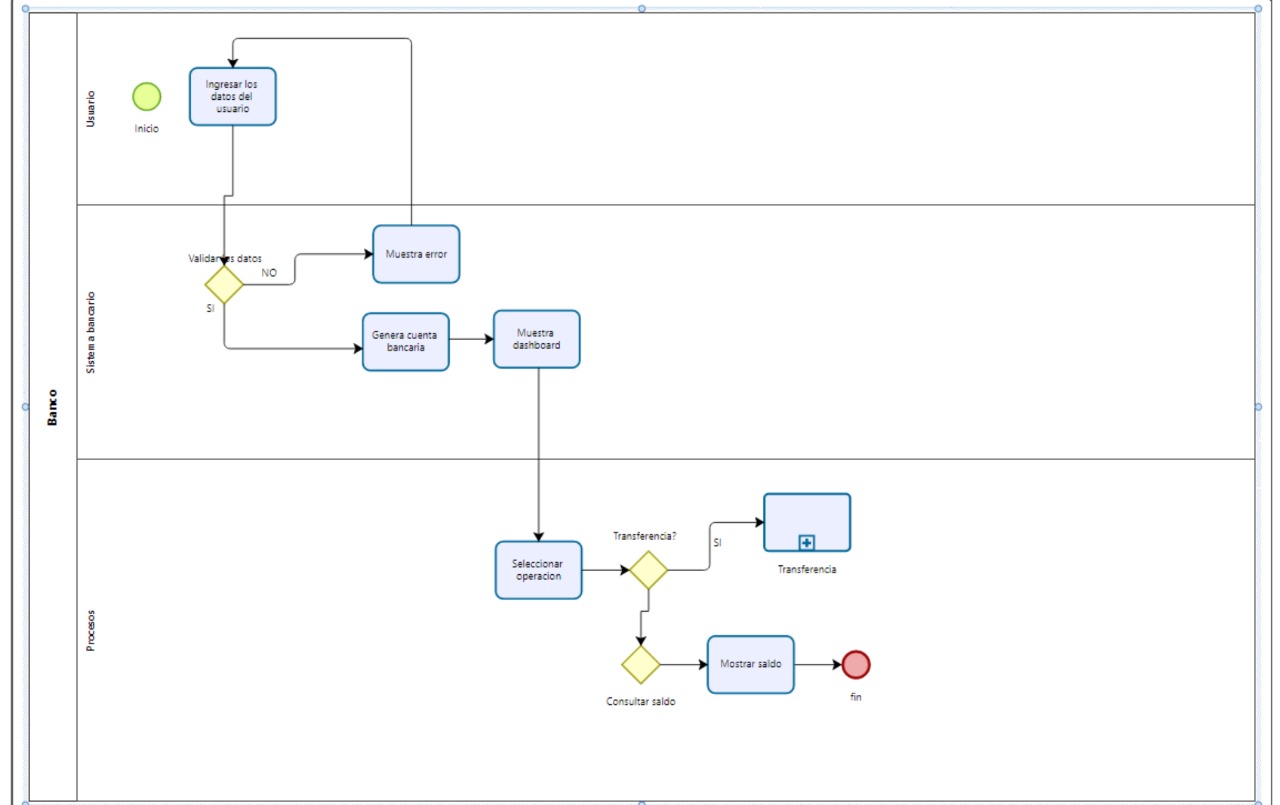
Propósito: que el desarrollador sepa cuáles son los componentes físicos (hardware) más importantes del sistema y dónde los componentes de software serán instalados.

Contenido: Un Diagrama de Despliegue (UML o C4), junto con texto o tablas que expliquen cada parte del diagrama que no haya sido descrita en la sección 2.1.

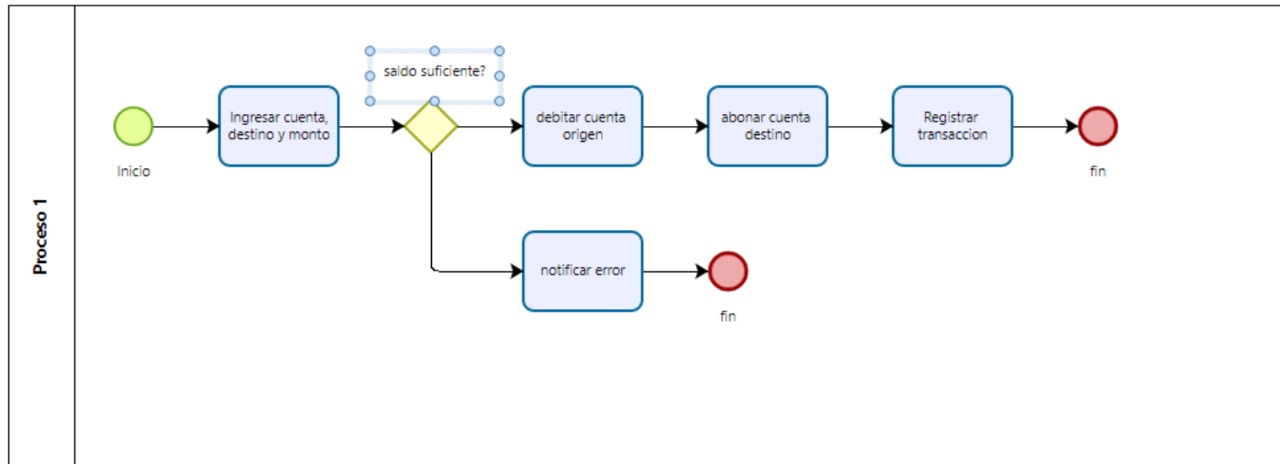
Este diagrama debe incluir los mismos componentes que aparecen en la sección 2.1, pero sin describir interfaces, sino enfatizando los lugares donde estarían instalados todos los componentes. Por ese motivo, algunos componentes como librerías, máquinas virtuales, etc., los cuales pueden ser instalados en múltiples computadores, podrían aparecer múltiples veces en este diagrama.

Referencias: [4], secciones The Physical Architecture

## Vista de Procesos del Sistema



El diagrama BPMN muestra el flujo del proceso de registro y operaciones básicas de un sistema bancario, dividido en tres roles: Usuario, Sistema bancario y Procesos. El proceso inicia cuando el usuario ingresa sus datos; el sistema los valida y, si son correctos, genera una cuenta bancaria y muestra el dashboard. Si los datos no son válidos, se muestra un mensaje de error y se retorna al ingreso de datos. Desde el dashboard, el usuario puede seleccionar una operación: realizar una transferencia o consultar el saldo. Dependiendo de la elección, el sistema ejecuta la transferencia o muestra el saldo disponible, finalizando luego el proceso.



Cabe resaltar que transferencia es un subproceso y realiza lo siguiente **transferencia bancaria** entre cuentas. El flujo inicia cuando el usuario ingresa los datos de la transferencia: cuenta de origen, cuenta de destino y el monto. A continuación, el sistema evalúa si el saldo en la cuenta de origen es suficiente. Si lo es, se debita la cuenta de origen, se abona la cuenta de destino y se registra la transacción, finalizando el proceso exitosamente. En caso de que no haya saldo suficiente, se genera una notificación de error y el proceso concluye sin realizar la transferencia.

Propósito: que el desarrollador sepa cuáles son los principales procesos en los cuales el usuario interactúa con el sistema.

Contenido: Representar a través de diagramas de actividad (o similares) las principales acciones que los usuarios realizan en el sistema. **Esta es una vista de alto nivel, por lo que no requiere mostrar demasiado detalle.**

Estos diagramas deben ser consistentes con el Diagrama de Navegación de la interfaz gráfica. Los diagramas deben ir acompañados de texto o tablas que expliquen cada parte del diagrama.

Los diagramas a utilizar pueden ser de los siguientes tipos:

* Diagramas de Actividad
* Diagrama BPMN

Referencias: [4]

# Diseño Detallado

Propósito: Mostrar los detalles más importantes del diseño de bajo nivel del sistema.

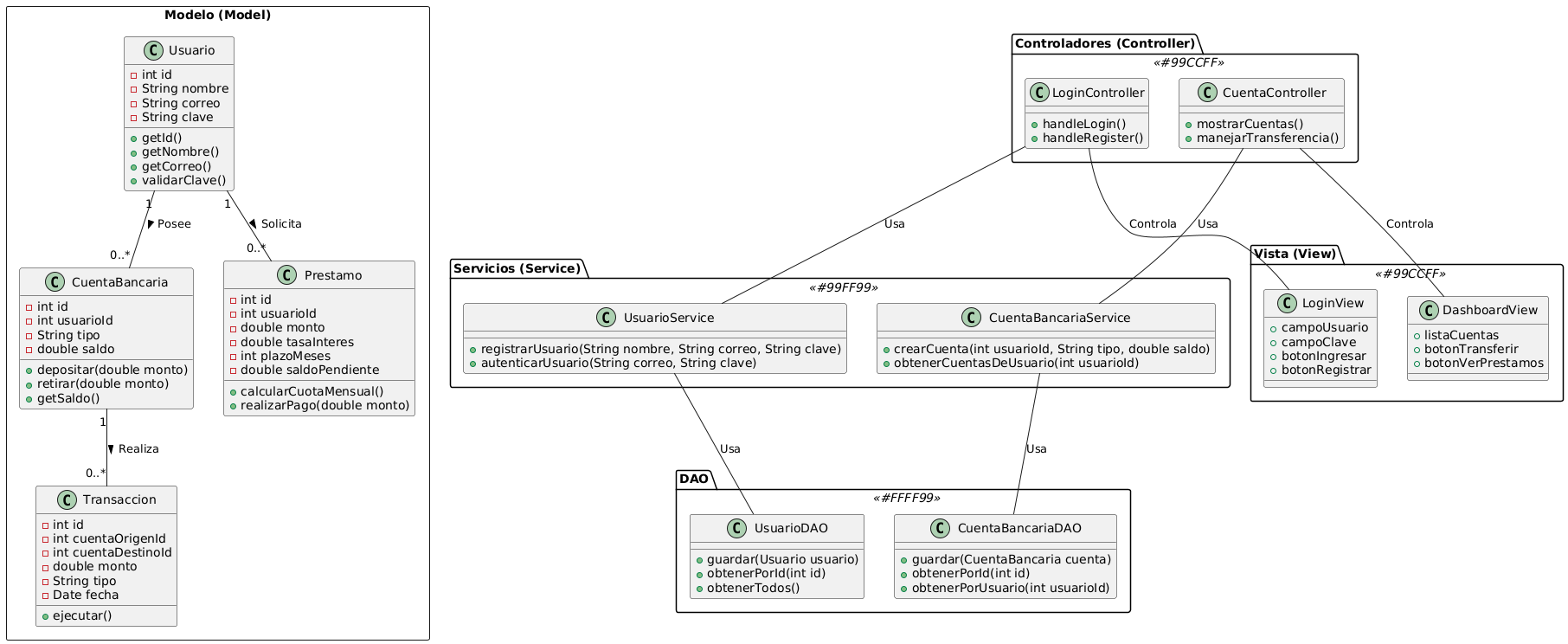
Contenido: Ver subsecciones

## Interfaz de Usuario

Propósito: Que el desarrollador entienda todos los detalles de la interfaz de usuario. Contenido: Diagramas que expliquen las principales pantallas de la interfaz gráfica y la navegación entre ellas. Por ejemplo, diagramas de navegación, diagramas de estado, Diagramas de Flujo de Interfaz de Usuario [6]. Todos los diagramas deben ir acompañados de texto o tablas que expliquen sus componentes

Nota: El concepto de interfaz de usuario es más general que el de interfaz gráfica, por lo cual en una situación real podría ser necesario describir aspectos adicionales al de la interfaz gráfica. Para efectos de esta entrega sólo se le pide especificar esto último.

## Estructura del Sistema



**Imagen expandida en anexos**

El sistema sigue una arquitectura basada en el patrón **Modelo-Vista-Controlador (MVC)**, donde cada componente cumple un rol específico para garantizar una separación clara de responsabilidades y facilitar el mantenimiento.

* **La capa de Modelo (Model)** representa los datos del sistema y su lógica de negocio. Contiene entidades como Usuario, CuentaBancaria, Prestamo y Transaccion, cada una con atributos y métodos que permiten manipular la información. La persistencia de estos datos se maneja mediante **clases DAO**, como UsuarioDAO y CuentaBancariaDAO, que interactúan directamente con la base de datos en H2.
* **La capa de Servicios (Service)** actúa como intermediaria entre el modelo y los controladores. Clases como UsuarioService y CuentaBancariaService encapsulan la lógica de negocio y validaciones, asegurando que las reglas del sistema se cumplan antes de realizar operaciones en la base de datos.
* **La capa de Controladores (Controller)** gestiona la interacción del usuario con la aplicación. LoginController y CuentaController reciben entradas desde la interfaz gráfica y delegan las acciones a los servicios correspondientes.
* **La capa de Vista (View)** contiene los elementos gráficos creados con JavaFX, como LoginView.fxml y DashboardView.fxml. Estas interfaces presentan la información de manera intuitiva y se comunican con los controladores para actualizar el estado del sistema en tiempo real.

Esta organización modular facilita la escalabilidad del sistema, permitiendo la integración de nuevas funcionalidades sin afectar su estructura principal.

## Comportamiento del Sistema

Propósito: Que el desarrollador entienda los detalles de cómo cambia el sistema en el tiempo. Proveer una guía para implementar los métodos más complejos de las clases del software.

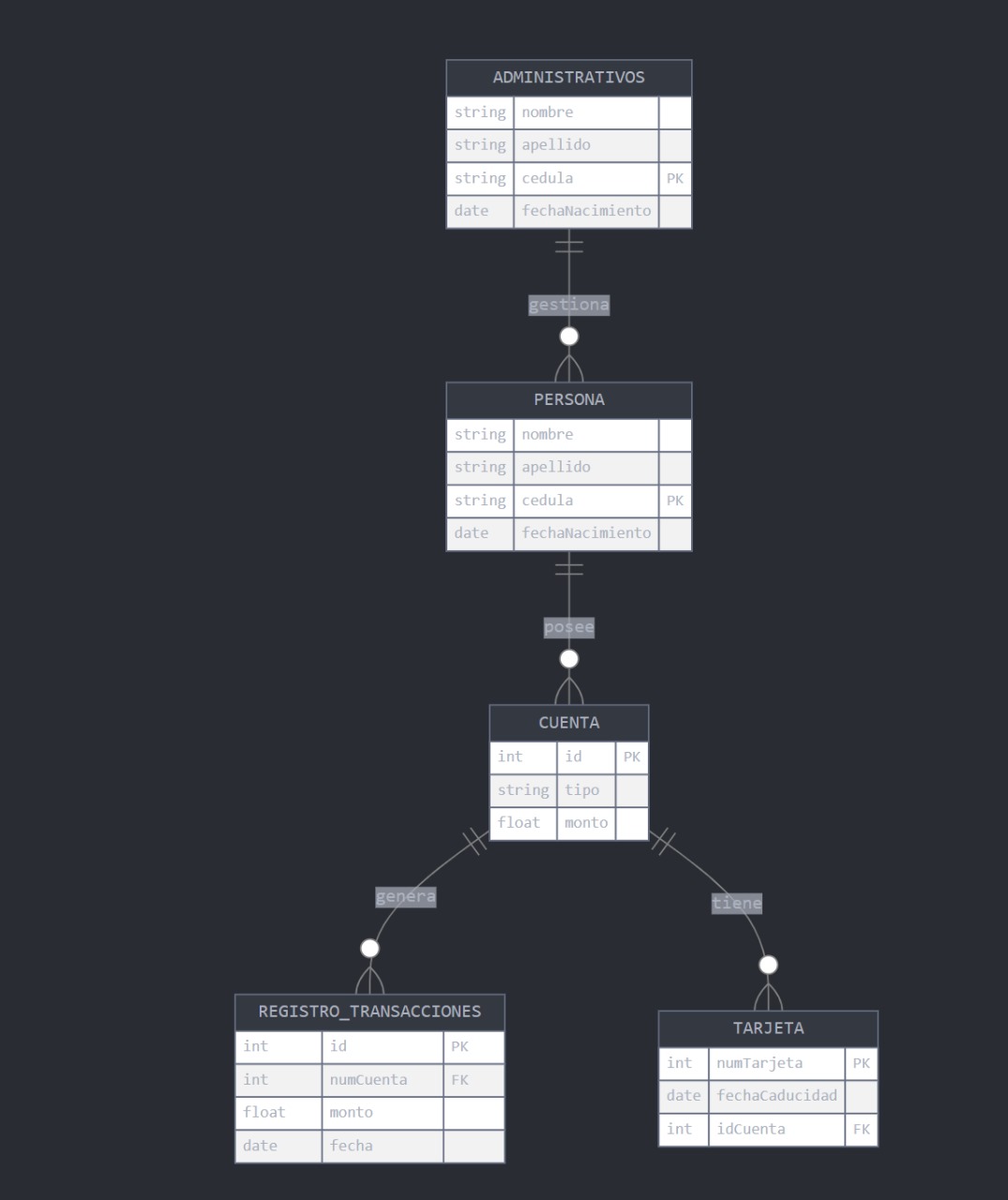
Contenido: Para cada acción compleja descrita en la sección 2.3, unh diagrama de secuencia, de comunicación, de actividad, de estados, o pseudocódigo que describa cómo se realiza dicha acción. Debe ser consistente con las clases descritas en 3.2.

Si una acción no es suficientemente compleja y se entiende suficientemente bien mirando solo la documentación de la sección 2.3, no es necesario detallarla en esta sección.

Si una acción se puede entender mejor desde el punto de vista de su interfaz gráfica, descríbala únicamente en la sección 3.1

No describa en esta sección lo que sucede dentro de las librerías que Ud. utilice y que no sean de su creación.

## Persistencia



Este diagrama entidad-relación representa el sistema bancario descrito en el documento, mostrando las siguientes entidades principales y sus relaciones:

1. **PERSONA**: Almacena la información básica de los clientes, como nombre, apellido, cédula y fecha de nacimiento.
2. **CUENTA**: Contiene información sobre las cuentas bancarias de los clientes, incluyendo un identificador único, tipo de cuenta y monto disponible.
3. **REGISTRO\_TRANSACCIONES**: Registra todas las transacciones realizadas, con referencia a la cuenta correspondiente y el monto de la transacción.
4. **TARJETA**: Almacena información sobre las tarjetas asociadas a las cuentas, incluyendo el número de tarjeta y fecha de caducidad.
5. **ADMINISTRATIVOS**: Gestiona la información del personal administrativo del banco.

Las relaciones entre estas entidades son:

* Una PERSONA puede poseer varias CUENTAS
* Una CUENTA puede generar múltiples REGISTRO\_TRANSACCIONES
* Una CUENTA puede tener varias TARJETAS asociadas
* Los ADMINISTRATIVOS pueden gestionar información de múltiples PERSONAS

Propósito: Que el desarrollador entienda cómo se van a almacenar los datos del sistema en forma persistente. Proveer una guía para crear las bases de datos, archivos u otros medios de almacenamiento persistentes utilizados por el software.

Contenido: Diagramas que describan la forma en que la información será almacenada, ya sea en bases de datos, archivos, etc. Los diagramas que se pueden usar son: Diagramas Crowfoot (Diagramas E-R clásicos pueden ser ineficientes en términos de espacio), Diagramas Lógicos o Físicos de Datos o Diagramas de Clases UML con estereotipos [5]. Los diagramas deben ser explicados en párrafos y/o tablas.

# Reporte gerencial

Tareas que se realizaron y como se dividieron:

Santiago – 2.1 Vista lógica del sistema

Define la estructura lógica del sistema con base en el patrón MVC. Identifica los principales paquetes: model, controller, view, dao, util, main. Incluye un diagrama de paquetes o componentes (UML o C4) que muestre cómo se comunican entre sí. Agrega una explicación de qué contiene cada paquete y qué librerías se utilizarán. Esta sección es base para el resto, así que debe completarse primero.

Damián – 2.2 Vista física del sistema

Explica cómo se desplegará el sistema localmente en el computador del usuario. Incluye un diagrama de despliegue (UML o C4) que muestre nodos como: aplicación Java, JavaFX, la base de datos H2, y librerías. Asegúrate de incluir los mismos componentes definidos en 2.1. Debes esperar a que Santiago finalice su diagrama para tomar de ahí la estructura lógica.

Juan Diego – 2.3 Vista de procesos del sistema

Modela al menos dos procesos importantes del sistema (como registro de usuario y transferencias) mediante diagramas BPMN o de actividad. Acompáñalos con una explicación textual o en tabla. Ten en cuenta que estos procesos deberán implementarse luego dentro del controlador del MVC, así que tus diagramas deben ser coherentes con las acciones del usuario y la lógica del sistema.

Luis – 3.1 Interfaz de usuario

Diseña los diagramas que representen las pantallas principales del sistema, indicando cómo se navega entre ellas. Usa diagramas de navegación o de flujo de interfaz. Las pantallas deben representar la capa view del patrón MVC. Agrega texto que explique qué se ve en cada pantalla y qué eventos o acciones dispara hacia el controlador. Consulta los procesos de 2.3 para mantener coherencia.

Santiago – 3.2 Estructura del sistema

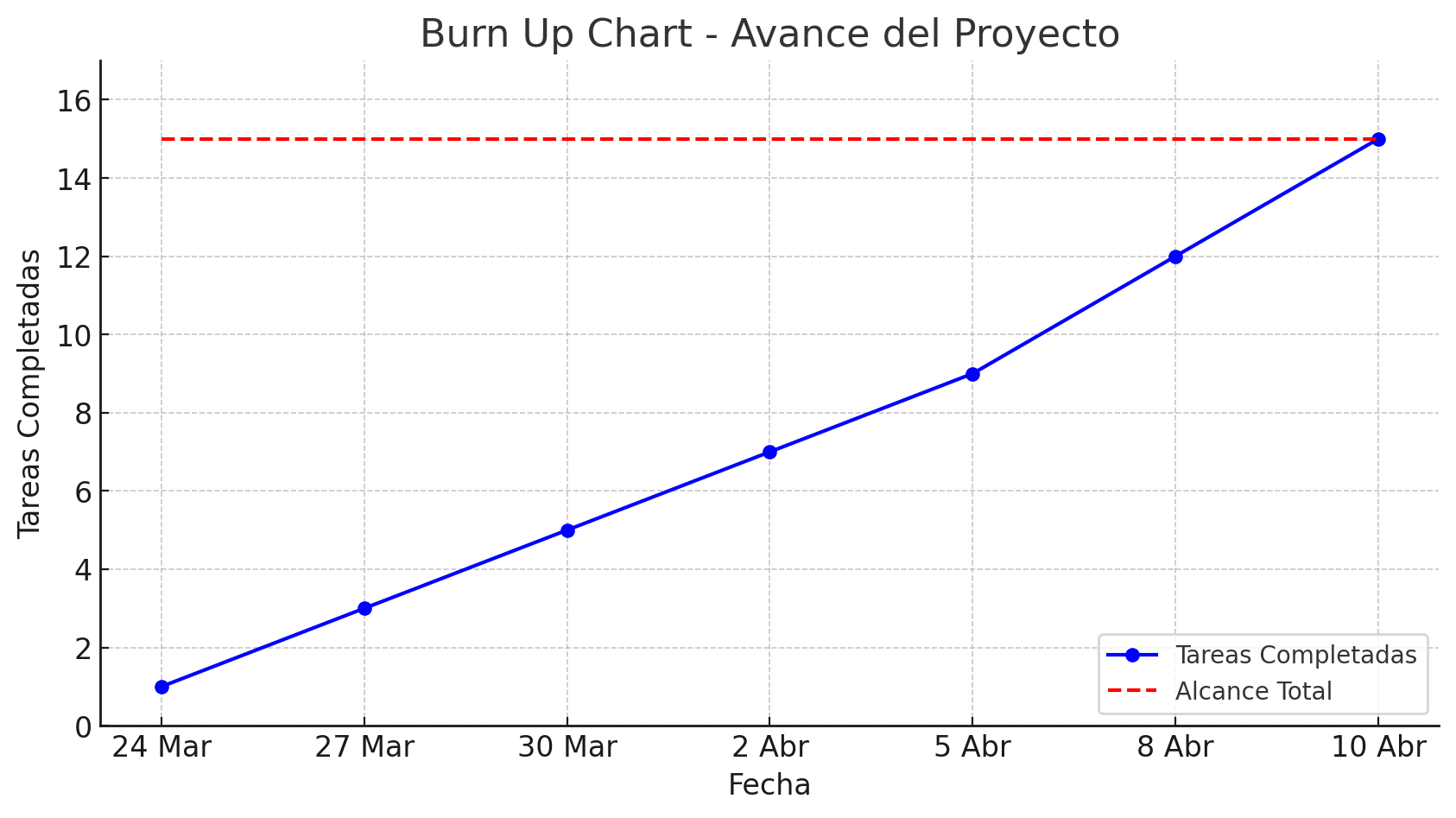
Construye los diagramas de clases que describan los componentes de software definidos en el sistema, particularmente en la capa model del MVC. Incluye clases como Usuario, Cuenta, Transaccion, Prestamo, etc. Describe atributos, métodos y relaciones. Asegúrate de que estas clases respondan a los procesos de 2.3 y tengan sentido con la estructura general.

luis – 3.3 Comportamiento del sistema

Representa con diagramas de secuencia o pseudocódigo al menos dos acciones complejas del sistema (como simular crédito o programar pagos). Muestra cómo se comunican los objetos, siguiendo el flujo entre view, controller y model del MVC. Puedes empezar cuando Santiago tenga el diseño básico de clases listo (3.2).

Juan Diego – 3.4 Persistencia

Diseña cómo se almacenará la información de manera persistente (base de datos H2). Usa diagramas ER, Crowfoot o UML con estereotipos para representar tablas, campos, relaciones, tipos de datos. Las entidades deben reflejar las clases del model y dar soporte a los procesos definidos en 2.3. Incluye una explicación escrita o en tabla de cada entidad.



**División del Código**

* **Santiago**
  + Implementación de modelo y lógica de negocio (Clases Usuario, CuentaBancaria, Prestamo, Transaccion).
  + Conexión con la base de datos H2 y DAO.
  + Servicios y controladores.
* **Luis**
  + Implementación de la interfaz gráfica en JavaFX.
  + Controladores para manejar interacciones con la UI.
  + Integración entre la UI y la lógica de negocio.

# Referencias

Indique aquí todas las referencias bibliográficas utilizadas en el documento. Utilice formato IEEE o APA para definirlas. Para administrar automáticamente las referencias, se recomienda el uso de la herramienta Zotero (www.zotero.org).

----

Referencias usadas en el formato

[1] ISO/IEC/IEEE 42010

[2] IEEE Std 1016-2009

[3] RUP. Software Architecture Document <http://www.ts.mah.se/RUP/RationalUnifiedProcess/webtmpl/templates/a_and_d/rup_sad.htm>

[4] Philippe Kruchten, Architectural Blueprints—The “4+1” View Model of Software Architecture. <http://www.cs.ubc.ca/~gregor/teaching/papers/4+1view-architecture.pdf>

[5] Scott Ambler, UML Data Modeling Profile. <http://www.agiledata.org/essays/umlDataModelingProfile.html>

[6] Scott Ambler, User Interface Flow Diagrms. <http://www.agilemodeling.com/artifacts/uiFlowDiagram.htm>

# Anexos

