Descripción del Diseño del Software

A continuación encontrará los elementos requeridos para su entrega del documento de Descripción del Diseño del Software. Todos los párrafos del color de este texto son las instrucciones para completar cada sección. Deben ser removidos del documento a entregar.

**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 |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

# 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

Propósito: que el desarrollador entienda la arquitectura del sistema, es decir, la definición de estructura y comportamiento del sistema con un alto nivel de abstracción.

Contenido: Ver sub-secciones

Referencias: [1], [2]

## 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 |

Diagrama

El contenido generado por IA puede ser incorrecto.

Cada paquete agrupa clases con responsabilidades específicas: view contiene las interfaces gráficas construidas en JavaFX; controller gestiona los eventos de la interfaz y coordina las acciones entre la vista y el modelo; model define las entidades del dominio y su lógica de negocio; dao abstrae el acceso a datos mediante interfaces (I) e implementaciones concretas (C); util incluye clases auxiliares como validadores o la conexión a la base de datos; main contiene la clase que inicia la aplicación, y resources almacena los archivos estáticos como FXML, CSS e imágenes. Las flechas indican relaciones de dependencia entre componentes, y las relaciones de herencia o implementación se representan mediante flechas con triángulo vacío. Este diseño modular favorece la escalabilidad y el mantenimiento del sistema, y permite una clara separación de responsabilidades.

Propósito: que el desarrollador sepa cuáles con los componentes más gruesos del sistema, sin preocuparse cómo serán instalados en el hardware.

Contenido: Diagrama de Componentes (UML o C4) o Diagrama de Paquetes (UML) que describa los principales componentes del software y sus interfaces, junto con texto o tablas que expliquen cada parte del diagrama. Si se utilizan algunos patrones arquitectónicos (por ejemplo MVC), éstos deberían ser evidentes en el diagrama.

El diagrama debe incluir como componentes todas las librerías o frameworks requeridos por el sistema para su funcionamiento.

Referencias: [4], sección The Development Architecture y The Logical Architecture

## 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

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

Propósito: Que el desarrollador comprenda los detalles de la estructura de todos los componentes de software del sistema. Proveer una guía para implementar las clases utilizadas en el software.

Contenido: Para cada componente de software identificado en la sección 2.2, un diagrama de clases acompañado de texto o tablas que lo expliquen. Todas las clases, métodos y asociaciones deben estar explicados.

Si en la sección 3.4 utiliza un diagrama de clases, no necesita repetir dichas clases en esta sección.

## 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, un 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

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.

# 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>