Imagen que contiene interior, mamífero, gato, viendo

Descripción generada automáticamente

**Tema**

EUREKABANK BDD NUBE SOAP JAVA

GRUPO 3

**Tutor**

Ing. Eduardo Mauricio Campaña Ortega

MIS.MDU.CCNA.CCIA.

PhD. (c) Ingeniería de Software

PhD. (c) Seguridad Información

**Fecha**

10/12/2025

**EXAMEN PRIMER PARCIAL DOTNET**

[ÍNDICE DE TABLAS 6](#_Toc216215933)

[PARTE PRÁCTICA 7](#_Toc216215934)

[INSTALACIÓN DE HERRAMIENTAS 7](#_Toc216215935)

[ESTRUCTURA DE PROYECTO 7](#_Toc216215936)

[INSTALACIÓN Y CONFIGURACIÓN DE HERRAMIENTAS 7](#_Toc216215937)

[INSTALACIÓN Y CONFIGURACIÓN DE APACHE NETBEANS 24 8](#_Toc216215938)

[INSTALACIÓN Y CONFIGURACIÓN DE ANDROID STUDIO 10](#_Toc216215939)

[INSTALACIÓN Y CONFIGURACIÓN DE DOCKER DESKTOP 11](#_Toc216215940)

[INSTALACIÓN Y CONFIGURACIÓN DE DBEAVER 14](#_Toc216215941)

[CREACIÓN DE LA BASE DE DATOS 16](#_Toc216215942)

[DEFINICIÓN DEL CONTENEDOR 16](#_Toc216215943)

[GESTIÓN DEL SERVIDOR REMOTO 16](#_Toc216215944)

[DESPLIEGUE DEL SERVICIO 17](#_Toc216215945)

[VERIFICACIÓN DE LA CONEXIÓN 18](#_Toc216215946)

[APLICACIÓN SERVIDOR 19](#_Toc216215947)

[CREACIÓN DEL PROYECTO SERVIDOR EUREKABANK 19](#_Toc216215948)

[CODIFICACIÓN DEL PROYECTO SERVIDOR 20](#_Toc216215949)

[CONEXIÓN A LA BASE DE DATOS 20](#_Toc216215950)

[CREACIÓN DE PAQUETES 21](#_Toc216215951)

[CREACIÓN DE MODELOS 21](#_Toc216215952)

[CREACIÓN DE SERVICIO 23](#_Toc216215953)

[CREACIÓN DE CONTROLADOR 28](#_Toc216215954)

[PRUEBAS DE LOS SERVICIOS 30](#_Toc216215955)

[CODIFICACIÓN DE LOS CLIENTES 32](#_Toc216215956)

[CLIENTE CONSOLA 32](#_Toc216215957)

[CREACIÓN DEL PROYECTO 32](#_Toc216215958)

[ESTRUCTURA DEL PROYECTO 33](#_Toc216215959)

[CODIFICACIÓN DEL PROYECTO CLIENTE CONSOLA 33](#_Toc216215960)

[CREACIÓN DE MODELOS 33](#_Toc216215961)

[EJECUCIÓN DEL CLIENTE DE CONSOLA 34](#_Toc216215962)

[CLIENTE ESCRITORIO 35](#_Toc216215963)

[CREACIÓN DEL PROYECTO 35](#_Toc216215964)

[ESTRUCTURA DEL PROYECTO 35](#_Toc216215965)

[CODIFICACIÓN DEL PROYECTO CLIENTE ESCRITORIO 36](#_Toc216215966)

[CREACIÓN DE MODELOS 36](#_Toc216215967)

[EJECUCIÓN DEL CLIENTE DE ESCRITORIO 37](#_Toc216215968)

[CLIENTE WEB 39](#_Toc216215969)

[CREACIÓN DEL PROYECTO 39](#_Toc216215970)

[ESTRUCTURA DEL PROYECTO 39](#_Toc216215971)

[CODIFICACIÓN DEL PROYECTO CLIENTE WEB 39](#_Toc216215972)

[CREACIÓN DE CONTROLADORES 39](#_Toc216215973)

[CREACIÓN DE SERVICIOS 39](#_Toc216215974)

[CREACIÓN DE VISTAS 39](#_Toc216215975)

[EJECUCIÓN DEL CLIENTE DE WEB 40](#_Toc216215976)

[CLIENTE MÓVIL 40](#_Toc216215977)

[CREACIÓN DEL PROYECTO 40](#_Toc216215978)

[ESTRUCTURA DEL PROYECTO 41](#_Toc216215979)

[CODIFICACIÓN DEL PROYECTO CLIENTE MÓVIL 41](#_Toc216215980)

[CREACIÓN DE MODELOS 41](#_Toc216215981)

[CREACIÓN DE REPOSITORIOS 42](#_Toc216215982)

[CREACIÓN DE SERVICIOS 42](#_Toc216215983)

[CREACIÓN DE VISTAS 42](#_Toc216215984)

[EJECUCIÓN DEL CLIENTE DE MÓVIL 43](#_Toc216215985)

[CONCLUSIONES 43](#_Toc216215986)

[RECOMENDACIONES 44](#_Toc216215987)

**ÍNDICE DE IMÁGENES**

[Figura 1. Creación de la carpeta raíz 7](#_Toc216216023)

[Figura 2. Estructura inicial del proyecto 7](#_Toc216216024)

[Figura 3. Versión de Java JDK 17 en el símbolo del sistema 8](#_Toc216216025)

[Figura 4. Página oficial de descarga de Apache NetBeans 8](#_Toc216216026)

[Figura 5. Instalador Apache NetBeans 24 9](#_Toc216216027)

[Figura 6. Pantalla inicial de instalación NetBeans 24 9](#_Toc216216028)

[Figura 7. Aplicación Apache NetBeans 24 9](#_Toc216216029)

[Figura 8. Página oficial de descarga de Android Studio 10](#_Toc216216030)

[Figura 9. Ejecutable Android Studio 10](#_Toc216216031)

[Figura 10. Pantalla inicial de instalación de Android Studio 10](#_Toc216216032)

[Figura 11. Aplicación Android Studio 11](#_Toc216216033)

[Figura 12. Administrador de tareas 11](#_Toc216216034)

[Figura 13. Características de Windows 12](#_Toc216216035)

[Figura 14. Página oficial de descarga de Docker Desktop 12](#_Toc216216036)

[Figura 15. Ejecutable Docker Desktop 12](#_Toc216216037)

[Figura 16. Pantalla inicial de instalación de Docker Desktop 13](#_Toc216216038)

[Figura 17. Configuración inicial de Docker Desktop 13](#_Toc216216039)

[Figura 18. Aplicación Docker Desktop 13](#_Toc216216040)

[Figura 19. Página oficial de descarga de DBeaver Community 14](#_Toc216216041)

[Figura 20. Ejecutable DBeaver 14](#_Toc216216042)

[Figura 21. Pantalla inicial de instalación de DBeaver 14](#_Toc216216043)

[Figura 22. Términos y condiciones DBeaver 15](#_Toc216216044)

[Figura 23. Instalación DBeaver 15](#_Toc216216045)

[Figura 24. Aplicación DBeaver 15](#_Toc216216046)

[Figura 25. Creación BD - Docker compose 16](#_Toc216216047)

[Figura 26. Creación BD - Terminal MobaXterm renombrando directorios en el servidor remoto 17](#_Toc216216048)

[Figura 27. Creación BD - Terminal corriendo el contenedor 18](#_Toc216216049)

[Figura 28. Conexión exitosa a BD eureka\_bank en DBeaver 18](#_Toc216216050)

[Figura 29. Creación del proyecto – Servidor EurekaBank 19](#_Toc216216051)

[Figura 30. Configuración del proyecto 1 - Servidor EurekaBank 19](#_Toc216216052)

[Figura 31. Configuración del proyecto 2 - Servidor EurekaBank 20](#_Toc216216053)

[Figura 32. Configuración de AccesoDB - Servidor EurekaBank 21](#_Toc216216054)

[Figura 33. Estructura del proyecto - Servidor EurekaBank 21](#_Toc216216055)

[Figura 34. Creación de entidades - Servidor EurekaBank 21](#_Toc216216056)

[Figura 35. Creación de servicio - Servidor EurekaBank 24](#_Toc216216057)

[Figura 36. Creación de controlador - Servidor EurekaBank 28](#_Toc216216058)

[Figura 37. Prueba 1: Login - Servidor EurekaBank 31](#_Toc216216059)

[Figura 38. Prueba 2: Registro de Depósito - Servidor EurekaBank 31](#_Toc216216060)

[Figura 39. Prueba 3: Registro de un Retiro - Servidor EurekaBank 31](#_Toc216216061)

[Figura 40. Prueba 4: Registro de una Transferencia - Servidor EurekaBank 31](#_Toc216216062)

[Figura 41. Prueba 5: Consulta de Historial de Transacciones- Servidor EurekaBank 32](#_Toc216216063)

[Figura 42. Prueba 6: Consulta de Balances - Servidor EurekaBank 32](#_Toc216216064)

[Figura 43. Creación del proyecto - Cliente consola 32](#_Toc216216065)

[Figura 44. Configuración del proyecto - Cliente consola 33](#_Toc216216066)

[Figura 45. Estructura del proyecto - Cliente consola 33](#_Toc216216067)

[Figura 46. Creación de modelos - Cliente consola 33](#_Toc216216068)

[Figura 47. Creación de controladores - Cliente consola 34](#_Toc216216069)

[Figura 48. Creación de vista - Cliente consola 34](#_Toc216216070)

[Figura 49. Login - Cliente consola 34](#_Toc216216071)

[Figura 50. Depósito - Cliente consola 34](#_Toc216216072)

[Figura 51. Retiro- Cliente consola 34](#_Toc216216073)

[Figura 52. Transferencia - Cliente consola 35](#_Toc216216074)

[Figura 53. Historial de Movimientos - Cliente consola 35](#_Toc216216075)

[Figura 54. Balances- Cliente consola 35](#_Toc216216076)

[Figura 55. Estructura del proyecto - Cliente escritorio 36](#_Toc216216077)

[Figura 56. Creación de modelos - Cliente escritorio 36](#_Toc216216078)

[Figura 57. Creación de vistas - Cliente escritorio 36](#_Toc216216079)

[Figura 58. Login - Cliente escritorio 37](#_Toc216216080)

[Figura 59. Depósito- Cliente escritorio 37](#_Toc216216081)

[Figura 60. Retiro- Cliente escritorio 38](#_Toc216216082)

[Figura 61. Transferencia - Cliente escritorio 38](#_Toc216216083)

[Figura 62. Historial de Movimientos- Cliente escritorio 38](#_Toc216216084)

[Figura 63. Balances- Cliente escritorio 38](#_Toc216216085)

[Figura 64. Estructura del proyecto - Cliente web 39](#_Toc216216086)

[Figura 65. Creación de controladores - Cliente web 39](#_Toc216216087)

[Figura 66. Creación de servicios - Cliente web 39](#_Toc216216088)

[Figura 67. Creación de vistas - Cliente web 40](#_Toc216216089)

[Figura 68. Login - Cliente web 40](#_Toc216216090)

[Figura 69. Depósito- Cliente web 40](#_Toc216216091)

[Figura 70. Retiro- Cliente web 40](#_Toc216216092)

[Figura 71. Transferencia - Cliente web 40](#_Toc216216093)

[Figura 72. Historial de Movimientos- Cliente web 40](#_Toc216216094)

[Figura 73. Balances- Cliente web 40](#_Toc216216095)

[Figura 74. Configuración del proyecto - Cliente móvil 41](#_Toc216216096)

[Figura 75. Estructura del proyecto - Cliente móvil 41](#_Toc216216097)

[Figura 76. Creación de modelos - Cliente móvil 42](#_Toc216216098)

[Figura 77. Creación de controladores - Cliente móvil 42](#_Toc216216099)

[Figura 78. Creación de servicios - Cliente móvil 42](#_Toc216216100)

[Figura 79. Creación de vistas - Cliente móvil 43](#_Toc216216101)

[Figura 80. Login - Cliente móvil 43](#_Toc216216102)

[Figura 81. Depósito- Cliente móvil 43](#_Toc216216103)

[Figura 82. Retiro- Cliente móvil 43](#_Toc216216104)

[Figura 83. Transferencia - Cliente móvil 43](#_Toc216216105)

[Figura 84. Historial de Movimientos- Cliente móvil 43](#_Toc216216106)

[Figura 85. Balances- Cliente móvil 43](#_Toc216216107)

# ÍNDICE DE TABLAS

[Tabla 1. Cadena de conexión a BD - Servidor EurekaBank 20](#_Toc216216108)

[Tabla 2. Clase Cuenta - Servidor EurekaBank 22](#_Toc216216109)

[Tabla 3. Clase Movimiento - Servidor EurekaBank 22](#_Toc216216110)

[Tabla 4. Clase EurekaService - Servidor EurekaBank 24](#_Toc216216111)

[Tabla 5. Clase WSEureka - Servidor EurekaBank 28](#_Toc216216112)

[Tabla 6. Servicios - Servidor EurekaBank 30](#_Toc216216113)

# PARTE PRÁCTICA

Para la realización de la parte práctica de este proyecto, es necesario realizar los siguientes pasos detallados a continuación y adaptarlo a las necesidades de cada individuo.

# INSTALACIÓN DE HERRAMIENTAS

## ESTRUCTURA DE PROYECTO

Para la elaboración del proyecto, en primer lugar, es necesario crear una carpeta en la cual se alojarán los archivos necesarios para su funcionamiento.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Figura 1. Creación de la carpeta raíz

Dentro de la carpeta principal del proyecto, se debe ubicar la siguiente subestructura.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Figura 2. Estructura inicial del proyecto

Todo esto con la finalidad de llevar buenas prácticas y mantener organizado el proyecto.

## INSTALACIÓN Y CONFIGURACIÓN DE HERRAMIENTAS

Para dar inicio al desarrollo de la práctica, primero se deben instalar y configurar las herramientas esenciales. Para este proyectó se utilizó Windows 11.

### INSTALACIÓN Y CONFIGURACIÓN DE APACHE NETBEANS 24

Previo a este proceso se supone la instalación y configuración de Java, en este proyecto se utilizó JDK 17 descargado desde la página oficial de Oracle <https://www.oracle.com/java/technologies/javase/jdk17-archive-downloads.html> y agregada a las variables de entorno. Se verifica la instalación ejecutando el siguiente comando en el símbolo del sistema o Powershell:

|  |
| --- |
| Java --version |

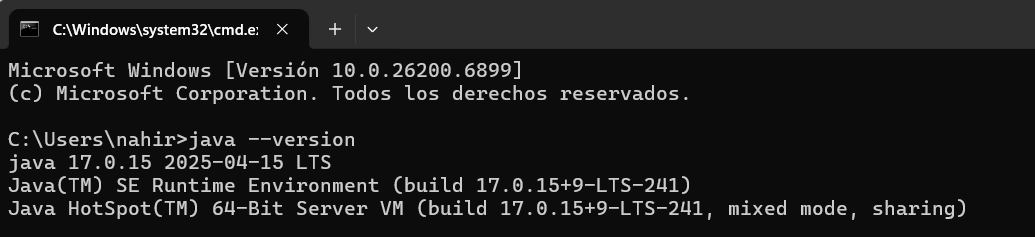


Figura 3. Versión de Java JDK 17 en el símbolo del sistema

Para la instalación de la herramienta se accede a la página oficial de descarga de Apache <https://netbeans.apache.org/front/main/download/> se selecciona la opción ‘Older Releases’, ‘Apache NetBeans 24’ y la opción para el sistema operativo.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Figura 4. Página oficial de descarga de Apache NetBeans

Una vez seleccionada la opción para el sistema operativo, se inicia la descarga y se obtendrá el siguiente ejecutable.

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

Figura 5. Instalador Apache NetBeans 24

Al ejecutar el instalador se muestra la siguiente pantalla, en el proceso se instalación se aceptan los términos y condiciones, se selecciona la ubicación de la descarga y se finaliza la instalación.

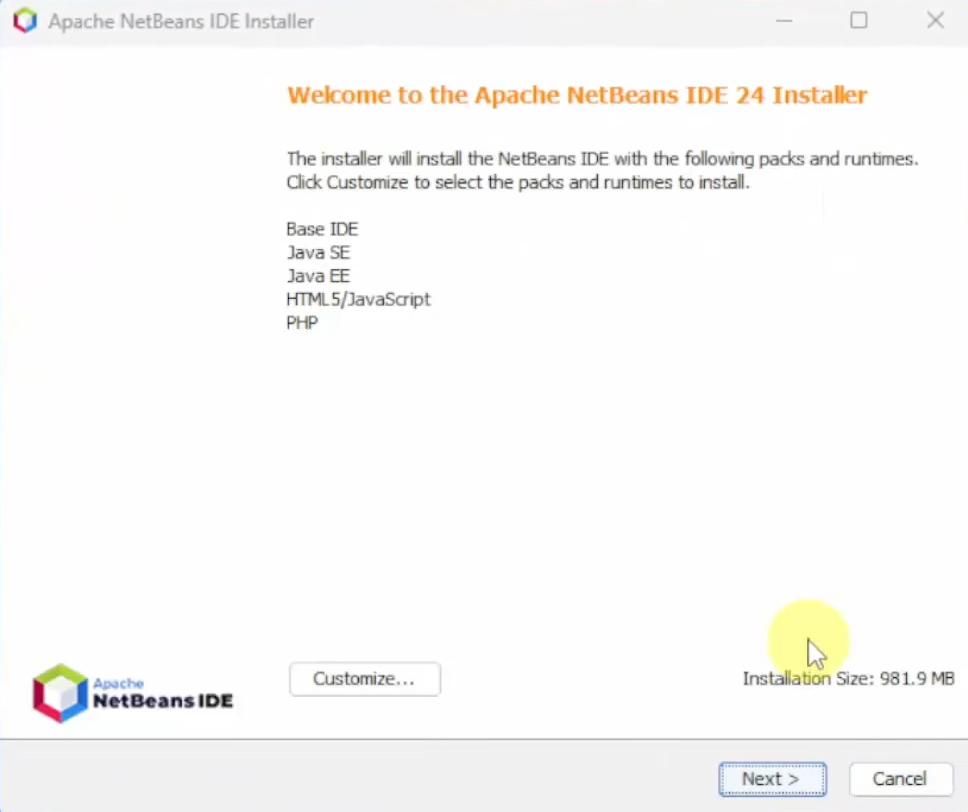


Figura 6. Pantalla inicial de instalación NetBeans 24

Una vez finalizada la descarga se ejecuta la aplicación y se iniciará Apache NetBeans 24.

Imagen que contiene Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

Figura 7. Aplicación Apache NetBeans 24

### INSTALACIÓN Y CONFIGURACIÓN DE ANDROID STUDIO

Para la instalación de la herramienta se accede a la página oficial de descarga de Android Studio <https://developer.android.com/studio> se selecciona la opción de descarga, se aceptan términos y condiciones e iniciará la descarga.

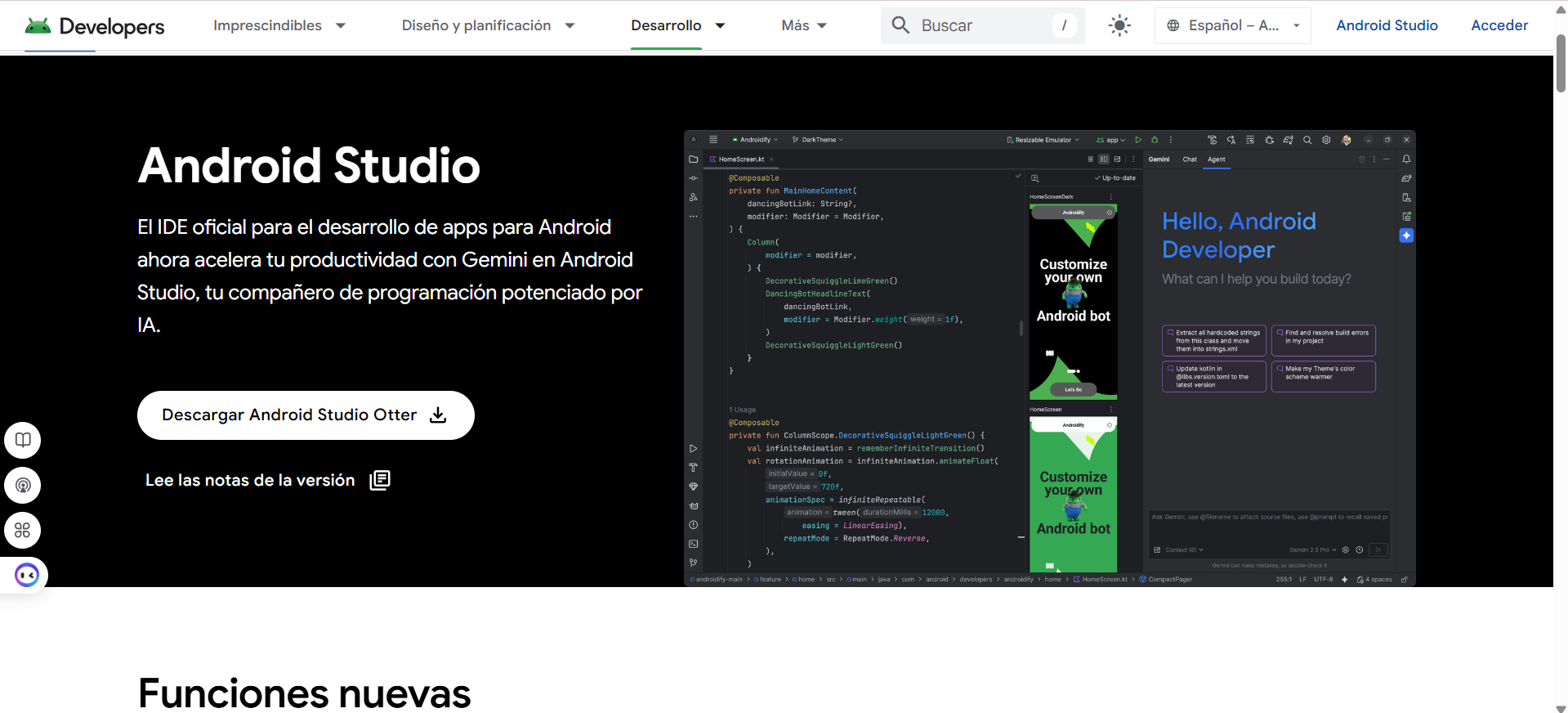


Figura 8. Página oficial de descarga de Android Studio

Al finalizar la descarga se obtendrá el instalador ejecutable.



Figura 9. Ejecutable Android Studio

Al ejecutar el instalador se mostrará la siguiente pantalla:

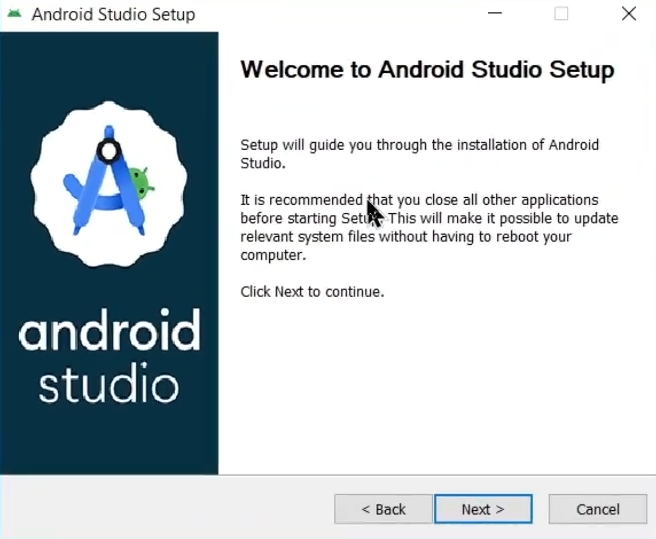


Figura 10. Pantalla inicial de instalación de Android Studio

Durante la instalación se selecciona los componentes a instalar, ubicación y carpeta de inicio. Al finalizar la instalación se iniciará la configuración en donde se selecciona el Setup Standard se aceptan términos y condiciones y se inicia la descarga de componentes.

Captura de pantalla de un celular

El contenido generado por IA puede ser incorrecto.

Figura 11. Aplicación Android Studio

### INSTALACIÓN Y CONFIGURACIÓN DE DOCKER DESKTOP

Previo a realizar la instalación de la herramienta se debe verificar que la virtualización esté habilitada en el administrador de tareas, si no es así, se debe activar desde la BIOS.

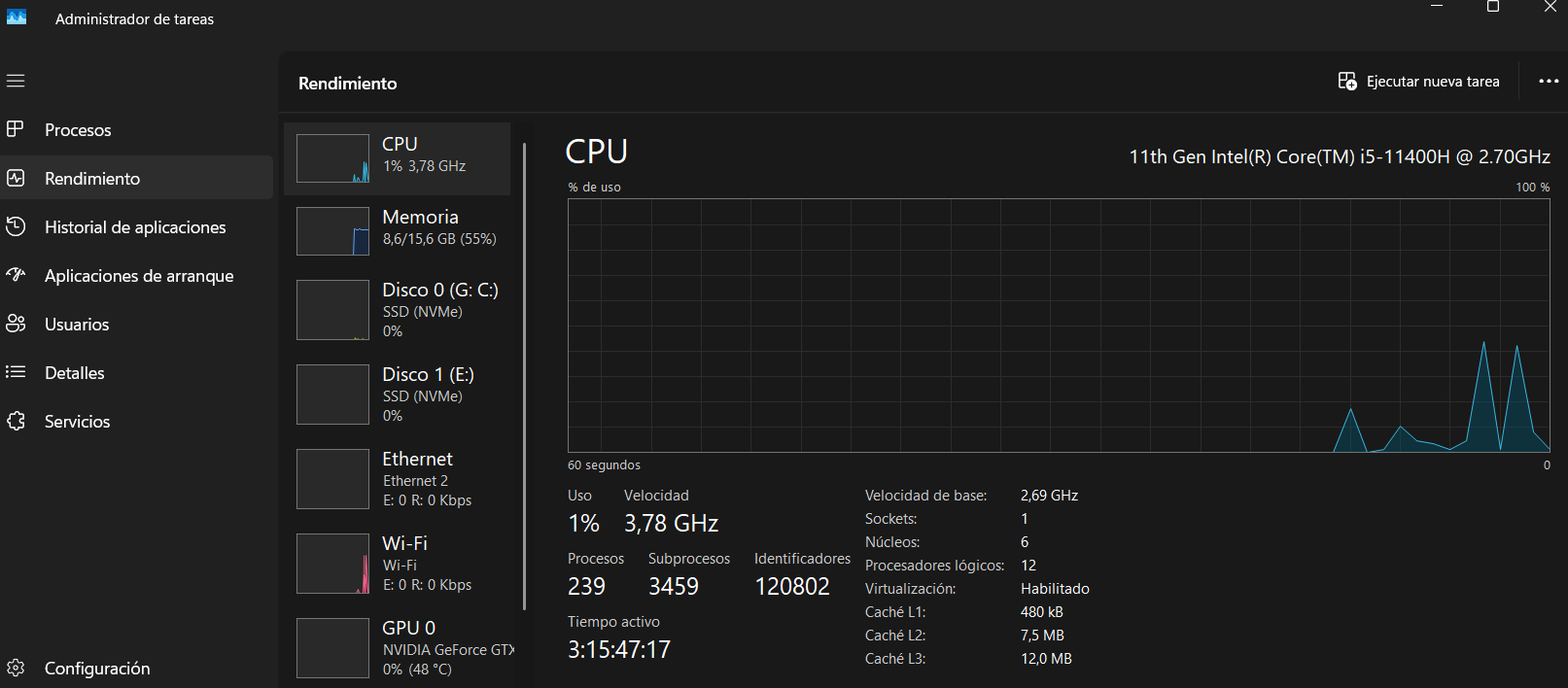


Figura 12. Administrador de tareas

Se debe habilitar la característica WSL 2 para lo cual se debe buscar la opción de ‘Activar o desactivar las características de Windows’ en el buscador de Windows. Y se habilita la opción ‘

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Figura 13. Características de Windows

Para descargar el instalador ejecutable se debe acceder al siguiente enlace <https://www.docker.com/products/docker-desktop/> del sitio oficial de Docker Hub.

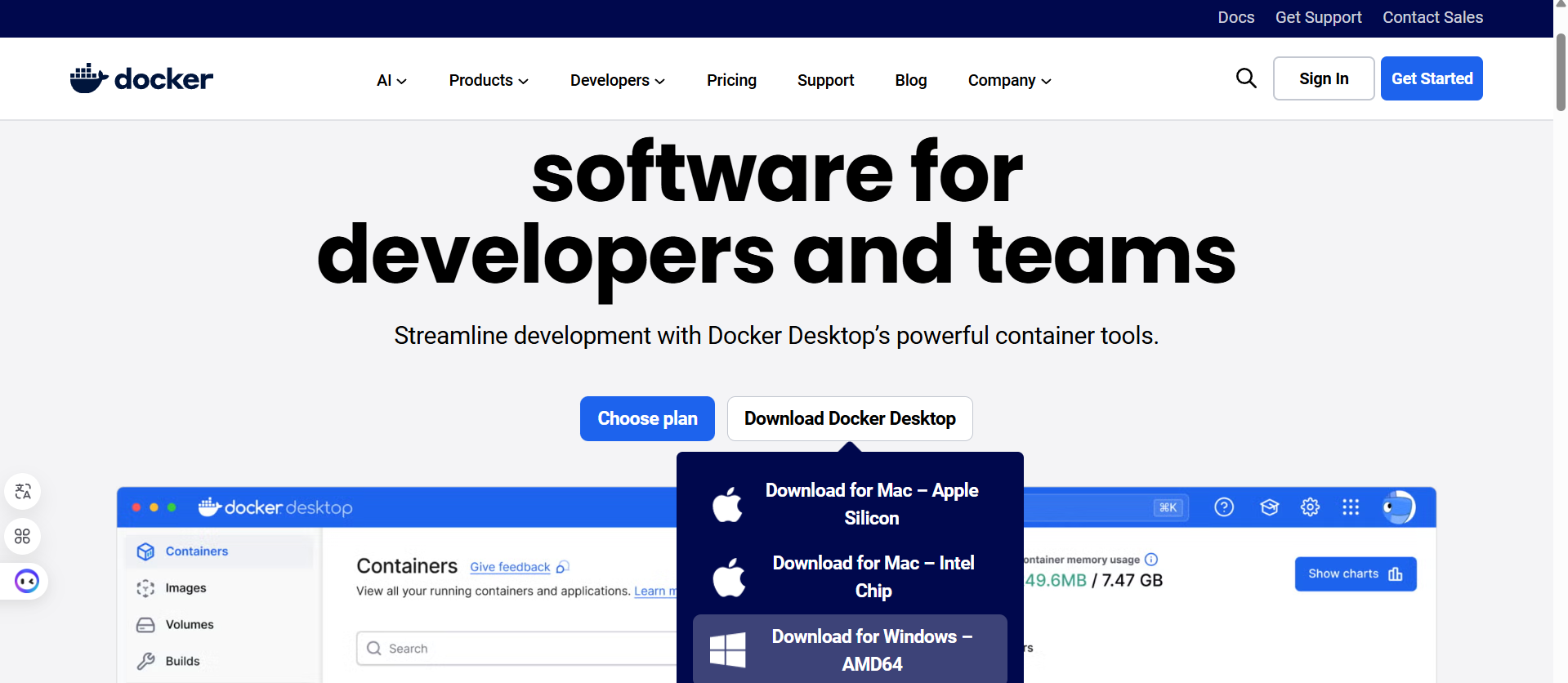


Figura 14. Página oficial de descarga de Docker Desktop

Una vez seleccionada la opción para el sistema operativo, se inicia la descarga y se obtendrá el siguiente ejecutable.

Icono

El contenido generado por IA puede ser incorrecto.

Figura 15. Ejecutable Docker Desktop

Al ejecutar el archivo se mostrará esta pantalla e iniciará la descarga, en donde se selecciona si se desea agregar el atajo al escritorio.

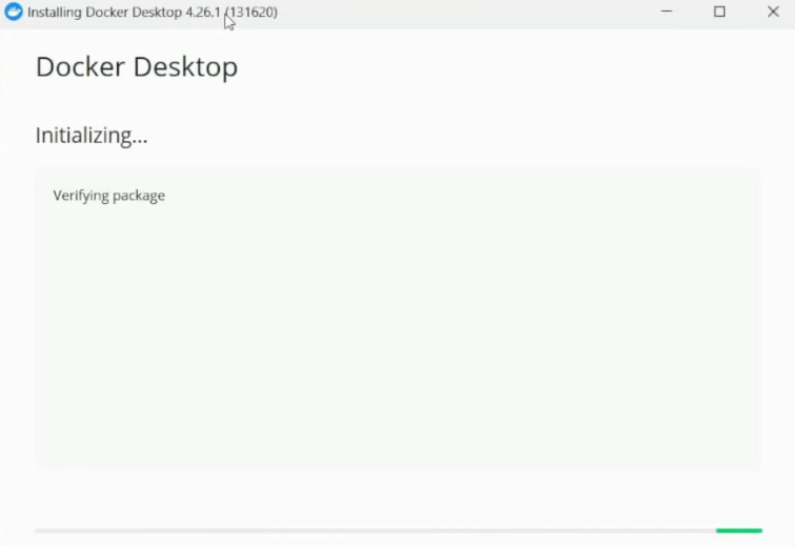


Figura 16. Pantalla inicial de instalación de Docker Desktop

Una vez finalizada la descarga se ejecuta la aplicación y se aceptan términos y condiciones y se realiza la configuración inicial que se recomienda sea ‘Use recommended settings’.



Figura 17. Configuración inicial de Docker Desktop

Una vez seleccionada la configuración se iniciará Docker Desktop.

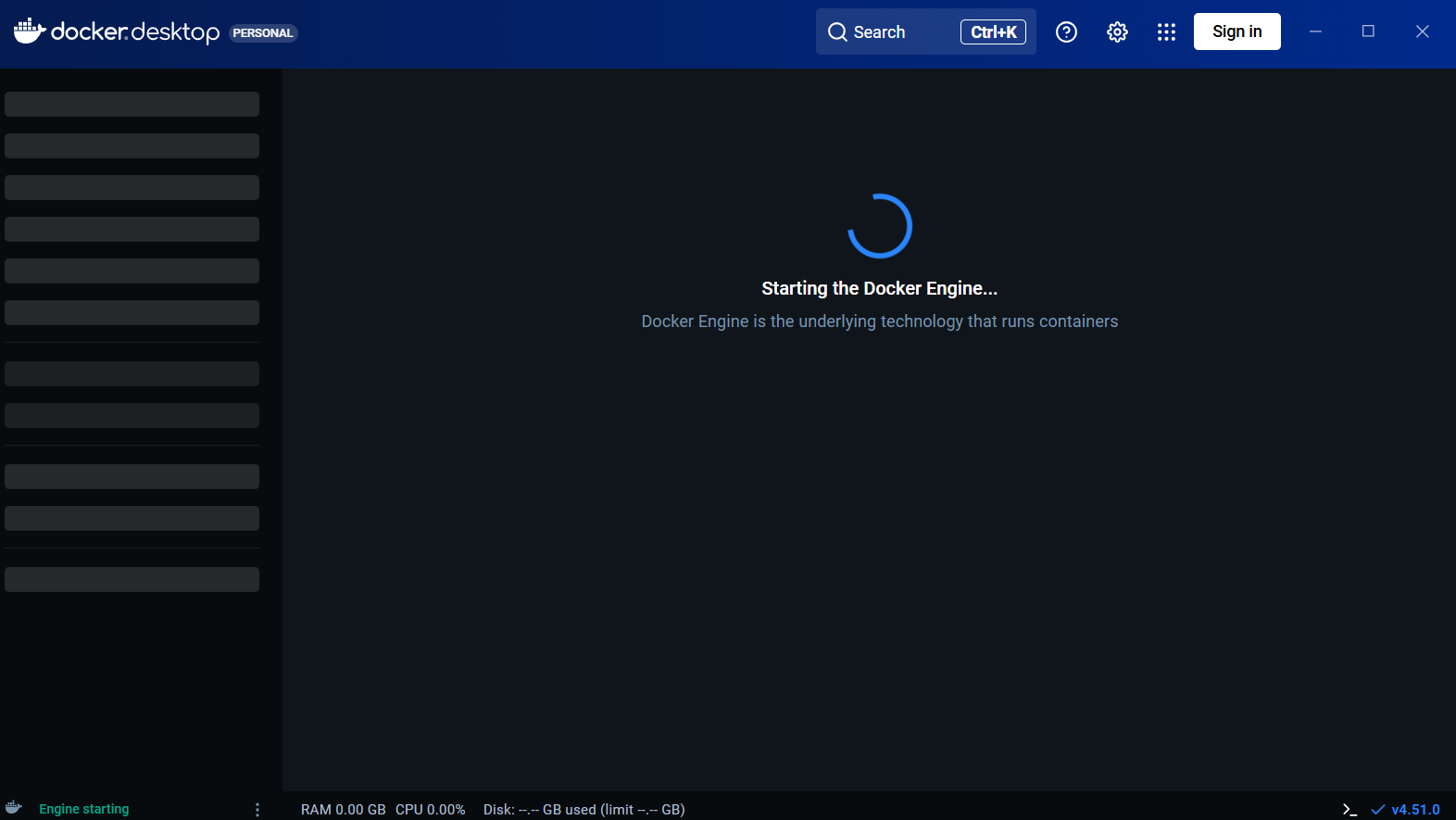


Figura 18. Aplicación Docker Desktop

### INSTALACIÓN Y CONFIGURACIÓN DE DBEAVER

Para instalar la herramienta se accede a la página oficial de descarga <https://dbeaver.io/download/> de DBeaver Community.

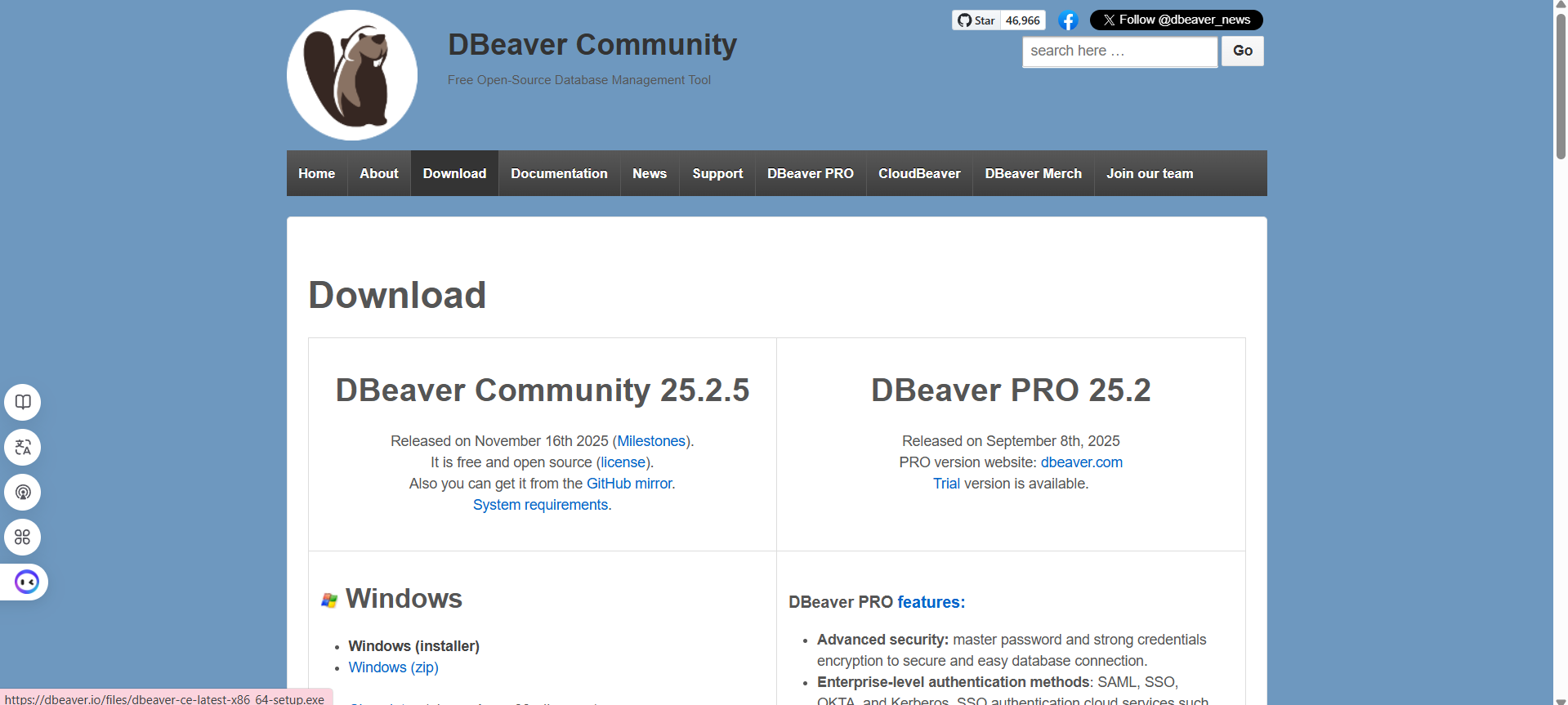


Figura 19. Página oficial de descarga de DBeaver Community

Se debe seleccionar la opción para el sistema operativo, esto iniciará la descarga y se obtendrá el siguiente instalador ejecutable.

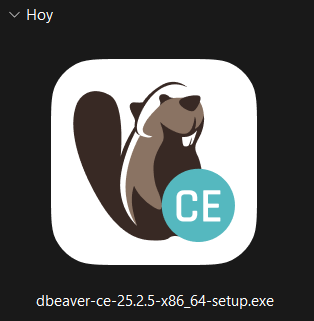


Figura 20. Ejecutable DBeaver

Al ejecutar el instalador se mostrará la siguiente pantalla.



Figura 21. Pantalla inicial de instalación de DBeaver

Al hacer click en ‘siguiente’ se muestran los términos y condiciones los cuales deben aceptarse.

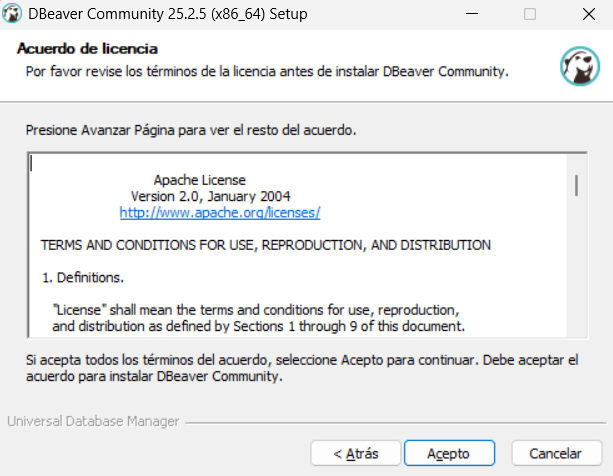


Figura 22. Términos y condiciones DBeaver

Se procede a seleccionar el usuario, los componentes a instalar, la ubicación de la instalación, la carpeta del menú inicio sin hacer cambios. Finalmente, se selecciona ‘instalar’

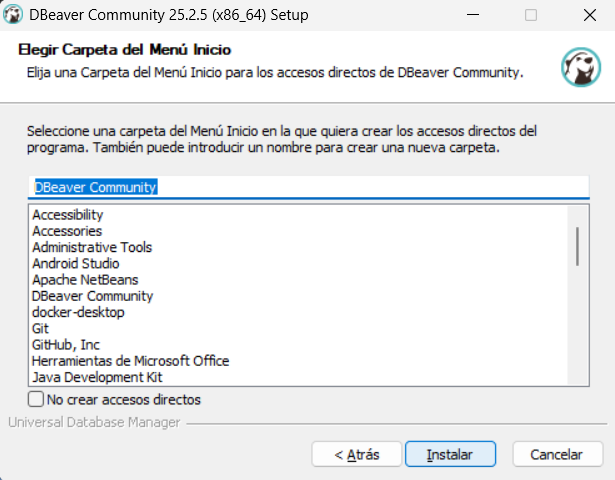


Figura 23. Instalación DBeaver

Cuando finaliza la instalación se ejecutará la aplicación.

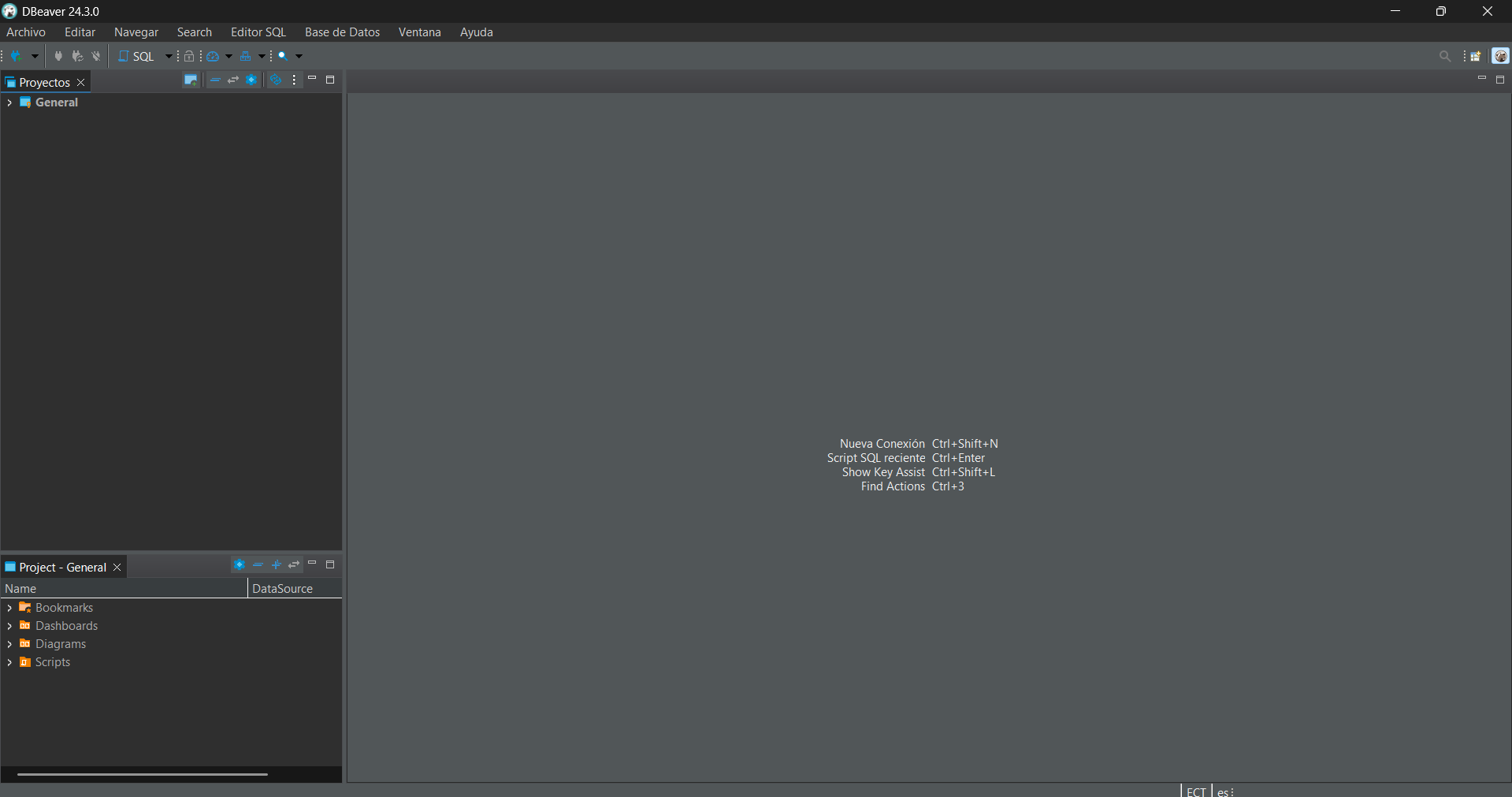


Figura 24. Aplicación DBeaver

# CREACIÓN DE LA BASE DE DATOS

Para garantizar la disponibilidad y accesibilidad del sistema EurekaBank, se procedió a desplegar la capa de persistencia (Base de Datos) en un servidor en la nube utilizando la infraestructura de DigitalOcean. El despliegue se realizó mediante Docker, lo que facilita la portabilidad y la gestión del entorno.

A continuación, se detalla el procedimiento realizado:

## DEFINICIÓN DEL CONTENEDOR

Se definió la infraestructura de la base de datos utilizando un archivo docker-compose.yml. Esta configuración establece los parámetros necesarios para levantar el servicio de MySQL de manera automatizada.

* **Imagen:** Se utilizó mysql:8.0 para garantizar compatibilidad con las sentencias SQL modernas.
* **Configuración de Entorno:** Se definieron las credenciales (MYSQL\_ROOT\_PASSWORD) y el nombre de la base de datos inicial (eureka\_bank).
* **Mapeo de Puertos:** Se expuso el puerto 3306 del contenedor al puerto 3306 del servidor anfitrión para permitir conexiones remotas.
* **Persistencia y Inicialización:** Se configuró un volumen (./init:/docker-entrypoint-initdb.d) para que, al iniciarse el contenedor por primera vez, se ejecute automáticamente el script 1\_crear\_bd.sql, creando así todas las tablas (Cliente, Cuenta, Movimiento, etc.) sin intervención manual.

A screenshot of a computer

AI-generated content may be incorrect.

Figura 25. Creación BD - Docker compose

## GESTIÓN DEL SERVIDOR REMOTO

Para la gestión del servidor virtual (Droplet) en DigitalOcean, se utilizó el cliente SSH MobaXterm. Esto permitió transferir los archivos de configuración y los scripts SQL necesarios al servidor Linux (Ubuntu). Se organizó el proyecto en directorios específicos para mantener el orden de la arquitectura SOAP/Java.

A computer screen shot of a computer

AI-generated content may be incorrect.

Figura 26. Creación BD - Terminal MobaXterm renombrando directorios en el servidor remoto

## DESPLIEGUE DEL SERVICIO

Una vez configurados los archivos en el servidor de DigitalOcean, se procedió a levantar el servicio mediante el comando:

docker-compose up

Como se evidencia en la ejecución, Docker realizó automáticamente:

1. La descarga (Pull) de la imagen de MySQL desde Docker Hub.
2. La creación de la red virtual (bd\_tl12\_soap\_java\_default) para aislar la comunicación.
3. La creación e inicio del contenedor mysql\_eureka.

A screen shot of a computer

AI-generated content may be incorrect.

Figura 27. Creación BD - Terminal corriendo el contenedor

## VERIFICACIÓN DE LA CONEXIÓN

Para validar que el despliegue fue exitoso y que la base de datos es accesible desde la aplicación (o herramientas de gestión), se realizó una prueba de conexión utilizando un cliente de base de datos.

Se procede a colocar los datos de la conexión:

* **Host:** 159.223.110.176 (IP Pública del Droplet de DigitalOcean).
* **Puerto:** 3306.
* **Base de Datos:** eureka\_bank.
* **Usuario:** root.

Luego se hace click en el botón ‘Probar conexión’.

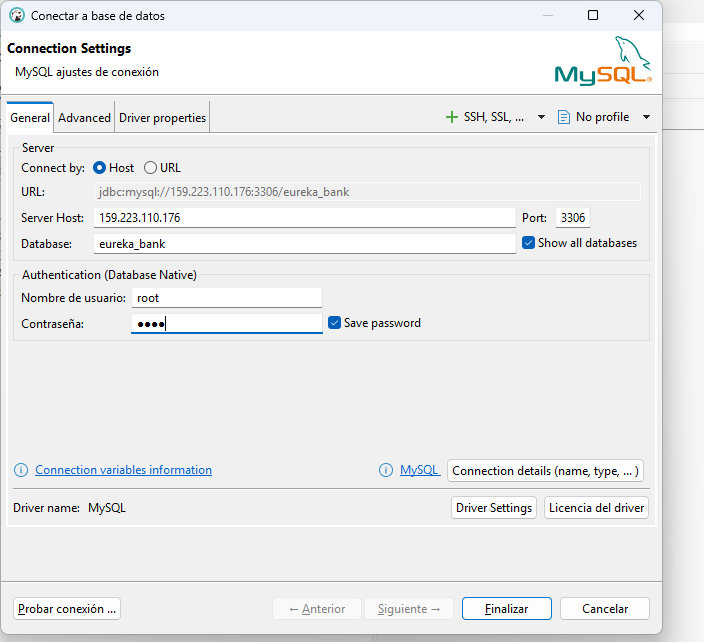


Figura 28. Conexión exitosa a BD eureka\_bank en DBeaver

# APLICACIÓN SERVIDOR

## CREACIÓN DEL PROYECTO SERVIDOR EUREKABANK

Se crea el proyecto en NetBeans seleccionando ‘New Project’, ‘Java with Maven’ y ‘Web Application’. Se coloca el nombre del proyecto, ubicación, grupo. Se selecciona el servidor Payara y versión de Jakarta. Finalmente se hace click en ‘Finish’

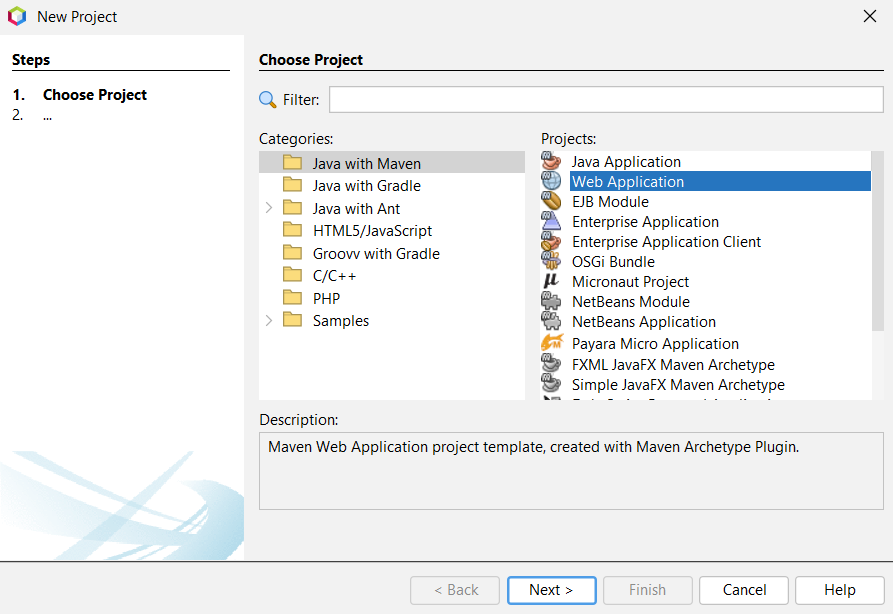


Figura 29. Creación del proyecto – Servidor EurekaBank

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

Figura 30. Configuración del proyecto 1 - Servidor EurekaBank

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

Figura 31. Configuración del proyecto 2 - Servidor EurekaBank

## CODIFICACIÓN DEL PROYECTO SERVIDOR

### CONEXIÓN A LA BASE DE DATOS

En el proyecto creado se crea el archivo AccesoDB en el paquete ec.edu.eureka.db

Tabla 1. Cadena de conexión a BD - Servidor EurekaBank

|  |
| --- |
| package ec.edu.eurekabank.db;  import java.sql.Connection;  import java.sql.DriverManager;  import java.sql.SQLException;  public class AccesoDB {    // CAMBIO IMPORTANTE AQUÍ:  // 1. Cambiamos 'localhost' por tu IP: 159.223.110.176  // 2. Agregamos '&allowPublicKeyRetrieval=true' para evitar errores de conexión con Docker/MySQL 8  private static final String URL = "jdbc:mysql://159.223.110.176:3306/eureka\_bank?useSSL=false&allowPublicKeyRetrieval=true";    // Estas credenciales coinciden con tu docker-compose (root/root)  private static final String USER = "root";  private static final String PASS = "root";  public AccesoDB() {  }    public static Connection getConnection() throws SQLException {  Connection cn = null;  try {  // Driver moderno  String driver = "com.mysql.cj.jdbc.Driver";    // Cargar el driver  Class.forName(driver); // El .newInstance() ya no es necesario en versiones nuevas de Java, pero no estorba.    // Obtener la conexión  cn = DriverManager.getConnection(URL, USER, PASS);    } catch (SQLException e) {  System.out.println("Error de conexión SQL: " + e.getMessage()); // Agregado para ver errores en consola  throw e;  } catch (ClassNotFoundException e){  throw new SQLException("ERROR: No se encuentra el driver de MySQL (mysql-connector-j).");  } catch (Exception e){  e.printStackTrace();  throw new SQLException("ERROR: No se tiene acceso el servidor");  }  return cn;  }  } |

Texto

El contenido generado por IA puede ser incorrecto.

Figura 32. Configuración de AccesoDB - Servidor EurekaBank

### CREACIÓN DE PAQUETES

Se crea la estructura del proyecto incluyendo paquetes: *config, controller, db, model, service*

Interfaz de usuario gráfica, Texto, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.

Figura 33. Estructura del proyecto - Servidor EurekaBank

### CREACIÓN DE MODELOS

Se crean las clases de las modelos *Cuenta y Movimiento* dentro del paquete *model.*

Texto

El contenido generado por IA puede ser incorrecto.

Figura 34. Creación de entidades - Servidor EurekaBank

En cada clase se crean los atributos definidos en cada tabla de la base de datos.

Tabla 2. Clase Cuenta - Servidor EurekaBank

|  |
| --- |
| package ec.edu.eurekabank.model;  import jakarta.xml.bind.annotation.XmlRootElement;  @XmlRootElement(name = "cuenta")  public class Cuenta {  private String numeroCuenta;  private String nombreCliente;  private double saldo;  private String moneda;  private String estado;  public Cuenta() {  }  public Cuenta(String numeroCuenta, String nombreCliente, double saldo, String moneda, String estado) {  this.numeroCuenta = numeroCuenta;  this.nombreCliente = nombreCliente;  this.saldo = saldo;  this.moneda = moneda;  this.estado = estado;  }  public String getNumeroCuenta() {  return numeroCuenta;  }  public void setNumeroCuenta(String numeroCuenta) {  this.numeroCuenta = numeroCuenta;  }  public String getNombreCliente() {  return nombreCliente;  }  public void setNombreCliente(String nombreCliente) {  this.nombreCliente = nombreCliente;  }  public double getSaldo() {  return saldo;  }  public void setSaldo(double saldo) {  this.saldo = saldo;  }  public String getMoneda() {  return moneda;  }  public void setMoneda(String moneda) {  this.moneda = moneda;  }  public String getEstado() {  return estado;  }  public void setEstado(String estado) {  this.estado = estado;  }  } |

Tabla 3. Clase Movimiento - Servidor EurekaBank

|  |
| --- |
| package ec.edu.eurekabank.model;  import java.util.Date;  import jakarta.xml.bind.annotation.XmlRootElement;  @XmlRootElement(name = "movimiento")  public class Movimiento {  private String cuenta;  private int nromov;  private Date fecha;  private String tipo;  private String accion;  private double importe;  public Movimiento() {  }  public Movimiento(String cuenta, int nromov, Date fecha, String tipo, String accion, double importe) {  this.cuenta = cuenta;  this.nromov = nromov;  this.fecha = fecha;  this.tipo = tipo;  this.accion = accion;  this.importe = importe;  }  public String getCuenta() {  return cuenta;  }  public void setCuenta(String cuenta) {  this.cuenta = cuenta;  }  public int getNromov() {  return nromov;  }  public void setNromov(int nromov) {  this.nromov = nromov;  }  public Date getFecha() {  return fecha;  }  public void setFecha(Date fecha) {  this.fecha = fecha;  }  public String getTipo() {  return tipo;  }  public void setTipo(String tipo) {  this.tipo = tipo;  }  public String getAccion() {  return accion;  }  public void setAccion(String accion) {  this.accion = accion;  }  public double getImporte() {  return importe;  }  public void setImporte(double importe) {  this.importe = importe;  }    } |

### CREACIÓN DE SERVICIO

Para conectar todo lo anterior se crea el servicio *EurekaService* en el paquete *service*.



Figura 35. Creación de servicio - Servidor EurekaBank

Tabla 4. Clase EurekaService - Servidor EurekaBank

|  |
| --- |
| package ec.edu.eurekabank.service;  import ec.edu.eurekabank.db.AccesoDB;  import ec.edu.eurekabank.model.Movimiento;  import ec.edu.eurekabank.model.Cuenta;  import java.sql.Connection;  import java.sql.PreparedStatement;  import java.sql.ResultSet;  import java.sql.SQLException;  import java.util.ArrayList;  import java.util.List;  public class EurekaService {  public List<Movimiento> leerMovimientos(String cuenta) {  Connection cn = null;  List<Movimiento> lista = new ArrayList<>();  String sql = "SELECT \n"  + " m.chr\_cuencodigo cuenta, \n"  + " m.int\_movinumero nromov, \n"  + " m.dtt\_movifecha fecha, \n"  + " t.vch\_tipodescripcion tipo, \n"  + " t.vch\_tipoaccion accion, \n"  + " m.dec\_moviimporte importe \n"  + "FROM tipomovimiento t INNER JOIN movimiento m \n"  + "ON t.chr\_tipocodigo = m.chr\_tipocodigo \n"  + "WHERE m.chr\_cuencodigo = ? \n"  + "ORDER BY m.int\_movinumero DESC";  try {  cn = AccesoDB.getConnection();  PreparedStatement pstm = cn.prepareStatement(sql);  pstm.setString(1, cuenta);  ResultSet rs = pstm.executeQuery();  while (rs.next()) {  Movimiento rec = new Movimiento();  rec.setCuenta(rs.getString("cuenta"));  rec.setNromov(rs.getInt("nromov"));  rec.setFecha(rs.getDate("fecha"));  rec.setTipo(rs.getString("tipo"));  rec.setAccion(rs.getString("accion"));  rec.setImporte(rs.getDouble("importe"));  lista.add(rec);  }  rs.close();  } catch (SQLException e) {  throw new RuntimeException(e.getMessage());  } finally {  try {  if (cn != null) cn.close();  } catch (Exception e) {  }  }  return lista;  }  public void registrarDeposito(String cuenta, double importe, String codEmp) {  ejecutarMovimiento(cuenta, importe, codEmp, "003", true);  }  public void registrarRetiro(String cuenta, double importe, String codEmp) {  ejecutarMovimientoDescuento(cuenta, importe, codEmp, "004", false);  }  public void registrarTransferencia(String cuentaOrigen, String cuentaDestino, double importe, String codEmp) {  Connection cn = null;  try {  cn = AccesoDB.getConnection();  cn.setAutoCommit(false);  // Retirar de la cuenta de origen  ejecutarMovimientoInternoDescuento(cuentaOrigen, importe, codEmp, "009", cn, false);  // Depositar en la cuenta de destino  ejecutarMovimientoInterno(cuentaDestino, importe, codEmp, "008", cn, true);  // Confirmar la transacción  cn.commit();  } catch (SQLException e) {  try {  if (cn != null) cn.rollback();  } catch (Exception ex) {  }  throw new RuntimeException(e.getMessage());  } finally {  try {  if (cn != null) cn.close();  } catch (Exception e) {  }  }  }  private void ejecutarMovimiento(String cuenta, double importe, String codEmp, String tipoMov, boolean permiteNegativo) {  Connection cn = null;  try {  cn = AccesoDB.getConnection();  cn.setAutoCommit(false);  ejecutarMovimientoInterno(cuenta, importe, codEmp, tipoMov, cn, permiteNegativo);  cn.commit();  } catch (SQLException e) {  try {  if (cn != null) cn.rollback();  } catch (Exception ex) {  }  throw new RuntimeException(e.getMessage());  } finally {  try {  if (cn != null) cn.close();  } catch (Exception e) {  }  }  }    private void ejecutarMovimientoDescuento(String cuenta, double importe, String codEmp, String tipoMov, boolean permiteNegativo) {  Connection cn = null;  try {  cn = AccesoDB.getConnection();  cn.setAutoCommit(false);  ejecutarMovimientoInternoDescuento(cuenta, importe, codEmp, tipoMov, cn, permiteNegativo);  cn.commit();  } catch (SQLException e) {  try {  if (cn != null) cn.rollback();  } catch (Exception ex) {  }  throw new RuntimeException(e.getMessage());  } finally {  try {  if (cn != null) cn.close();  } catch (Exception e) {  }  }  }      private void ejecutarMovimientoInterno(String cuenta, double importe, String codEmp, String tipoMov, Connection cn, boolean permiteNegativo) throws SQLException {  String sql = "SELECT dec\_cuensaldo, int\_cuencontmov "  + "FROM cuenta "  + "WHERE chr\_cuencodigo = ? AND vch\_cuenestado = 'ACTIVO' FOR UPDATE";  PreparedStatement pstm = cn.prepareStatement(sql);  pstm.setString(1, cuenta);  ResultSet rs = pstm.executeQuery();  if (!rs.next()) {  throw new SQLException("ERROR: Cuenta no existe o no está activa.");  }  double saldo = rs.getDouble("dec\_cuensaldo");  int cont = rs.getInt("int\_cuencontmov");  rs.close();  pstm.close();  saldo += importe;  if (!permiteNegativo && saldo < 0) {  throw new SQLException("ERROR: Saldo insuficiente.");  }  cont++;  sql = "UPDATE cuenta SET dec\_cuensaldo = ?, int\_cuencontmov = ? WHERE chr\_cuencodigo = ?";  pstm = cn.prepareStatement(sql);  pstm.setDouble(1, saldo);  pstm.setInt(2, cont);  pstm.setString(3, cuenta);  pstm.executeUpdate();  pstm.close();  sql = "INSERT INTO movimiento(chr\_cuencodigo, int\_movinumero, dtt\_movifecha, chr\_emplcodigo, chr\_tipocodigo, dec\_moviimporte) "  + "VALUES (?, ?, SYSDATE(), ?, ?, ?)";  pstm = cn.prepareStatement(sql);  pstm.setString(1, cuenta);  pstm.setInt(2, cont);  pstm.setString(3, codEmp);  pstm.setString(4, tipoMov);  pstm.setDouble(5, importe);  pstm.executeUpdate();  pstm.close();  }      private void ejecutarMovimientoInternoDescuento(String cuenta, double importe, String codEmp, String tipoMov, Connection cn, boolean permiteNegativo) throws SQLException {  String sql = "SELECT dec\_cuensaldo, int\_cuencontmov "  + "FROM cuenta "  + "WHERE chr\_cuencodigo = ? AND vch\_cuenestado = 'ACTIVO' FOR UPDATE";  PreparedStatement pstm = cn.prepareStatement(sql);  pstm.setString(1, cuenta);  ResultSet rs = pstm.executeQuery();  if (!rs.next()) {  throw new SQLException("ERROR: Cuenta no existe o no está activa.");  }  double saldo = rs.getDouble("dec\_cuensaldo");  int cont = rs.getInt("int\_cuencontmov");  rs.close();  pstm.close();  saldo -= importe;  if (!permiteNegativo && saldo < 0) {  throw new SQLException("ERROR: Saldo insuficiente.");  }  cont++;  sql = "UPDATE cuenta SET dec\_cuensaldo = ?, int\_cuencontmov = ? WHERE chr\_cuencodigo = ?";  pstm = cn.prepareStatement(sql);  pstm.setDouble(1, saldo);  pstm.setInt(2, cont);  pstm.setString(3, cuenta);  pstm.executeUpdate();  pstm.close();  sql = "INSERT INTO movimiento(chr\_cuencodigo, int\_movinumero, dtt\_movifecha, chr\_emplcodigo, chr\_tipocodigo, dec\_moviimporte) "  + "VALUES (?, ?, SYSDATE(), ?, ?, ?)";  pstm = cn.prepareStatement(sql);  pstm.setString(1, cuenta);  pstm.setInt(2, cont);  pstm.setString(3, codEmp);  pstm.setString(4, tipoMov);  pstm.setDouble(5, importe);  pstm.executeUpdate();  pstm.close();  }    public boolean login(String username, String password) {  Connection cn = null;  boolean acceso = false;  String sql = "SELECT COUNT(1) AS total " +  "FROM usuario " +  "WHERE vch\_emplusuario = ? " +  "AND vch\_emplclave = SHA(?) " +  "AND vch\_emplestado = 'ACTIVO'";  try {  cn = AccesoDB.getConnection();  PreparedStatement pstm = cn.prepareStatement(sql);  pstm.setString(1, username);  pstm.setString(2, password);  ResultSet rs = pstm.executeQuery();  if (rs.next()) {  acceso = rs.getInt("total") == 1;  }  rs.close();  pstm.close();  } catch (SQLException e) {  throw new RuntimeException("Error al validar login: " + e.getMessage());  } finally {  try {  if (cn != null) cn.close();  } catch (Exception e) {  }  }  return acceso;  }  public List<Cuenta> leerBalances() {  Connection cn = null;  List<Cuenta> lista = new ArrayList<>();  String sql = "SELECT " +  "c.chr\_cuencodigo AS numeroCuenta, " +  "CONCAT(cl.vch\_clienombre, ' ', cl.vch\_cliepaterno, ' ', cl.vch\_cliematerno) AS nombreCliente, " +  "c.dec\_cuensaldo AS saldo, " +  "m.vch\_monedescripcion AS moneda, " +  "c.vch\_cuenestado AS estado " +  "FROM cuenta c " +  "INNER JOIN cliente cl ON c.chr\_cliecodigo = cl.chr\_cliecodigo " +  "INNER JOIN modena m ON c.chr\_monecodigo = m.chr\_monecodigo " +  "WHERE c.vch\_cuenestado = 'ACTIVO' " +  "ORDER BY c.chr\_cuencodigo";  try {  cn = AccesoDB.getConnection();  PreparedStatement pstm = cn.prepareStatement(sql);  ResultSet rs = pstm.executeQuery();  while (rs.next()) {  Cuenta cuenta = new Cuenta();  cuenta.setNumeroCuenta(rs.getString("numeroCuenta"));  cuenta.setNombreCliente(rs.getString("nombreCliente"));  cuenta.setSaldo(rs.getDouble("saldo"));  cuenta.setMoneda(rs.getString("moneda"));  cuenta.setEstado(rs.getString("estado"));  lista.add(cuenta);  }  rs.close();  pstm.close();  } catch (SQLException e) {  throw new RuntimeException("Error al leer balances: " + e.getMessage());  } finally {  try {  if (cn != null) cn.close();  } catch (Exception e) {  }  }  return lista;  }  } |

### CREACIÓN DE CONTROLADOR

Finalmente se crea el controlador *WSEureja* en el paquete *controller.*

*Texto

El contenido generado por IA puede ser incorrecto.*

Figura 36. Creación de controlador - Servidor EurekaBank

El controlador recibe la petición HTTP, llama al servicio y devuelve la respuesta. Esta es la capa más superficial.

Tabla 5. Clase WSEureka - Servidor EurekaBank

|  |
| --- |
| /\*  \* To change this license header, choose License Headers in Project Properties.  \* To change this template file, choose Tools | Templates  \* and open the template in the editor.  \*/  package ec.edu.gr03.controller;  import ec.edu.eurekabank.model.Movimiento;  import ec.edu.eurekabank.model.Cuenta;  import ec.edu.eurekabank.service.EurekaService;  import java.util.ArrayList;  import java.util.List;  import jakarta.jws.WebService;  import jakarta.jws.WebMethod;  import jakarta.jws.WebParam;  import jakarta.jws.WebResult;  /\*\*  \*  \* @author DELL  \*/  @WebService(serviceName = "WSEureka")  public class WSEureka {  /\*\*  \* Web service operation  \* @param cuenta  \* @return Retorna la lista de movimientos de la cuenta  \*/  @WebMethod(operationName = "traerMovimientos")  @WebResult(name = "movimiento")  public List<Movimiento> traerMovimientos(@WebParam(name = "cuenta") String cuenta) {  List<Movimiento> lista;  try {  EurekaService service = new EurekaService();  lista = service.leerMovimientos(cuenta);  } catch (Exception e) {  lista = new ArrayList<>();  }  return lista;  }  /\*\*  \* Web service operation  \* @param cuenta  \* @param importe  \* @return Estado 1 o -1  \*/  @WebMethod(operationName = "regDeposito")  @WebResult(name = "estado")  public int regDeposito(@WebParam(name = "cuenta") String cuenta, @WebParam(name = "importe") double importe) {  int estado;  String codEmp = "0001";  try {  EurekaService service = new EurekaService();  service.registrarDeposito(cuenta, importe, codEmp);  estado = 1;  } catch (Exception e) {  estado = -1;  }  return estado;  }  /\*\*  \* Web service operation  \* @param cuenta  \* @param importe  \* @return Estado 1 o -1  \*/  @WebMethod(operationName = "regRetiro")  @WebResult(name = "estado")  public int regRetiro(@WebParam(name = "cuenta") String cuenta, @WebParam(name = "importe") double importe) {  int estado;  String codEmp = "0001";  try {  EurekaService service = new EurekaService();  service.registrarRetiro(cuenta, importe, codEmp);  estado = 1;  } catch (Exception e) {  estado = -1;  }  return estado;  }  /\*\*  \* Web service operation  \* @param cuentaOrigen  \* @param cuentaDestino  \* @param importe  \* @return Estado 1 o -1  \*/  @WebMethod(operationName = "regTransferencia")  @WebResult(name = "estado")  public int regTransferencia(  @WebParam(name = "cuentaOrigen") String cuentaOrigen,  @WebParam(name = "cuentaDestino") String cuentaDestino,  @WebParam(name = "importe") double importe) {  int estado;  String codEmp = "0001";  try {  EurekaService service = new EurekaService();  service.registrarTransferencia(cuentaOrigen, cuentaDestino, importe, codEmp);  estado = 1;  } catch (Exception e) {  estado = -1;  }  return estado;  }      @WebMethod(operationName = "login")  public boolean login(@WebParam(name = "username") String username, @WebParam(name = "password") String password) {  EurekaService service = new EurekaService();  return service.login(username, password);  }    /\*\*  \* Web service operation  \* @return Retorna la lista de todas las cuentas activas con sus balances  \*/  @WebMethod(operationName = "traerBalances")  @WebResult(name = "cuenta")  public List<Cuenta> traerBalances() {  List<Cuenta> lista;  try {  EurekaService service = new EurekaService();  lista = service.leerBalances();  } catch (Exception e) {  lista = new ArrayList<>();  }  return lista;  }  } |

## PRUEBAS DE LOS SERVICIOS

Para realizar la prueba de los servicios se procede a ejecutar el proyecto servidor, y hacer uso de SOAPUI.

A continuación, se describen lo servicios Implementados

Tabla 6. Servicios - Servidor EurekaBank

| ***Operación SOAP*** | ***Descripción Funcional*** | ***Parámetros de Entrada*** | ***Retorno*** | ***Reglas de Negocio Aplicadas*** |
| --- | --- | --- | --- | --- |
| **login** | Autentica a un empleado del banco verificando sus credenciales y estado en el sistema. | username (String)  password (String) | **Boolean**    (true si es válido, false si no) | • Valida usuario y contraseña (SHA) contra la tabla USUARIO.  • Verifica que el estado del usuario sea 'ACTIVO'. |
| **regDeposito** | Registra un depósito bancario, incrementando el saldo de la cuenta especificada. | cuenta (String)  importe (double) | **Int**    (1: Éxito, -1: Error) | • Valida que la cuenta exista y esté activa.  • Actualiza saldo (Suma).  • Registra movimiento con código **'003'**. |
| **regRetiro** | Registra un retiro de dinero, disminuyendo el saldo tras validar disponibilidad de fondos. | cuenta (String)  importe (double) | **Int**    (1: Éxito, -1: Error) | • Valida saldo suficiente (No permite saldo negativo).  • Actualiza saldo (Resta).  • Registra movimiento con código **'004'**. |
| **regTransferencia** | Realiza una transferencia de fondos entre dos cuentas internas de la entidad. | cuentaOrigen (String)  cuentaDestino (String)  importe (double) | **Int**    (1: Éxito, -1: Error) | • **Transacción ACID:** Se ejecuta todo o nada (commit/rollback).  • Realiza retiro en origen (cód. '009') y depósito en destino (cód. '008') en una sola operación atómica. |
| **traerMovimientos** | Consulta el historial de transacciones financieras de una cuenta específica. | cuenta (String) | **List<Movimiento>**  (Lista de objetos XML) | • Devuelve la lista ordenada por número de movimiento descendente.  • Incluye fecha, tipo, acción e importe. |
| **traerBalances** | Consulta general para obtener el estado financiero actual de todas las cuentas del banco. | *Ninguno* | **List<Cuenta>**  (Lista de objetos XML) | • Realiza un JOIN entre las tablas CUENTA, CLIENTE y MONEDA.  • Filtra solo cuentas con estado 'ACTIVO'. |

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Figura 37. Prueba 1: Login - Servidor EurekaBank

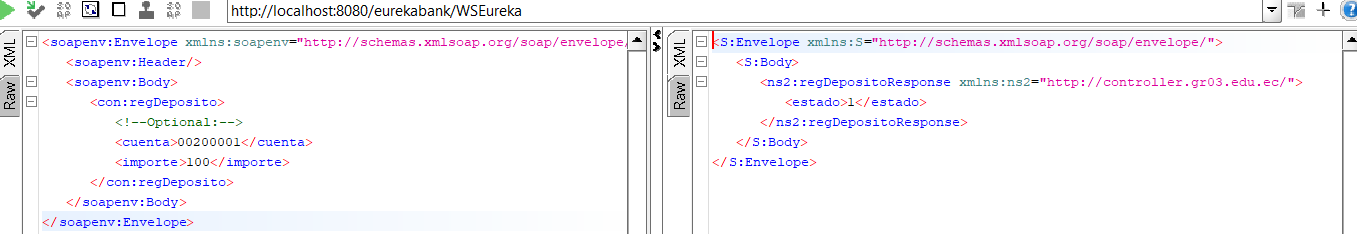


Figura 38. Prueba 2: Registro de Depósito - Servidor EurekaBank

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

Figura 39. Prueba 3: Registro de un Retiro - Servidor EurekaBank

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.

Figura 40. Prueba 4: Registro de una Transferencia - Servidor EurekaBank

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

Figura 41. Prueba 5: Consulta de Historial de Transacciones- Servidor EurekaBank

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Figura 42. Prueba 6: Consulta de Balances - Servidor EurekaBank

# CODIFICACIÓN DE LOS CLIENTES

## CLIENTE CONSOLA

### CREACIÓN DEL PROYECTO

El proyecto se crea usando NetBeans seleccionando ‘Nuevo Proyecto’, ‘Java with Maven’ y ‘Java Application’.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Figura 43. Creación del proyecto - Cliente consola

En este paso se define el nombre del proyecto y la ubicación.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

Figura 44. Configuración del proyecto - Cliente consola

### ESTRUCTURA DEL PROYECTO

Una vez creado el proyecto se crea la estructura de paquetes basada en el patrón MVC

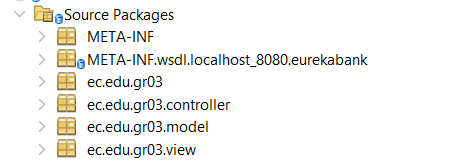


Figura 45. Estructura del proyecto - Cliente consola

## CODIFICACIÓN DEL PROYECTO CLIENTE CONSOLA

### CREACIÓN DE MODELOS

Se crean la clase *EurekaBankClient* dentro del paquete *models.*



Figura 46. Creación de modelos - Cliente consola

**CREACIÓN DE CONTROLADORES**

Se crean la clase *MainController* en el paquete *controller.*



Figura 47. Creación de controladores - Cliente consola

**CREACIÓN DE VISTAS**

Se crea la clase *LoginView y MenuView* en el paquete *view.*

Texto

El contenido generado por IA puede ser incorrecto.

Figura 48. Creación de vista - Cliente consola

### EJECUCIÓN DEL CLIENTE DE CONSOLA

Texto

El contenido generado por IA puede ser incorrecto.

Figura 49. Login - Cliente consola

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

Figura 50. Depósito - Cliente consola

Patrón de fondo

El contenido generado por IA puede ser incorrecto.

Figura 51. Retiro- Cliente consola

Imagen que contiene Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

Figura 52. Transferencia - Cliente consola

Tabla

El contenido generado por IA puede ser incorrecto.

Figura 53. Historial de Movimientos - Cliente consola

Tabla

El contenido generado por IA puede ser incorrecto.

Figura 54. Balances- Cliente consola

## CLIENTE ESCRITORIO

### CREACIÓN DEL PROYECTO

El proyecto se crea de igual manera que el cliente de consola *Java Application*

### ESTRUCTURA DEL PROYECTO

Una vez creado el proyecto se crea la estructura de paquetes basada en el patrón MVC

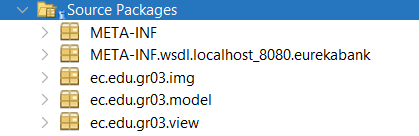


Figura 55. Estructura del proyecto - Cliente escritorio

## CODIFICACIÓN DEL PROYECTO CLIENTE ESCRITORIO

### CREACIÓN DE MODELOS

Se crean las clases *EurekaBankClient* dentro del paquete *model.*

Una captura de pantalla de un celular con texto e imagen

El contenido generado por IA puede ser incorrecto.

Figura 56. Creación de modelos - Cliente escritorio

**CREACIÓN DE VISTAS**

Se crean las clases *BalancesFrm, DepositoFrm, LoginFrm, MovimientoFrm, MovimientoTablaFrm, RetirnoFrm y TransferenciasFrm* en el paquete *view.*

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.

Figura 57. Creación de vistas - Cliente escritorio

### EJECUCIÓN DEL CLIENTE DE ESCRITORIO

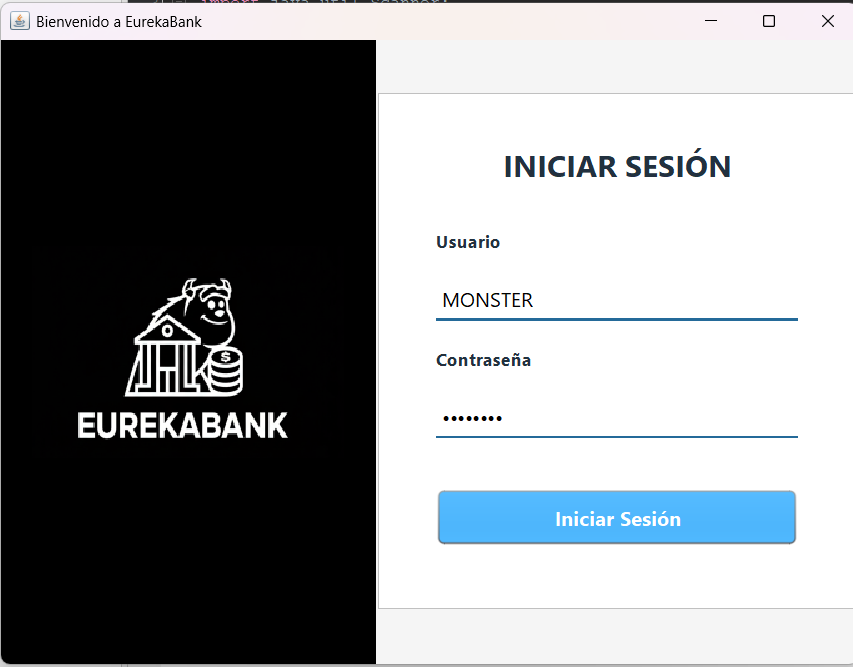


Figura 58. Login - Cliente escritorio

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

Figura 59. Depósito- Cliente escritorio

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

Figura 60. Retiro- Cliente escritorio

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

Figura 61. Transferencia - Cliente escritorio

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

Figura 62. Historial de Movimientos- Cliente escritorio

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

Figura 63. Balances- Cliente escritorio

## CLIENTE WEB

### CREACIÓN DEL PROYECTO

El proyecto se crea de igual manera que el Servicio EurekaBank con ‘Web Application’.

ESTRUCTURA DEL PROYECTO

Una vez creado el proyecto se crea la estructura de paquetes basada en el patrón MVC

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Figura 64. Estructura del proyecto - Cliente web

## CODIFICACIÓN DEL PROYECTO CLIENTE WEB

CREACIÓN DE CONTROLADORES

Se crean las clases *DepositoServlet, LoginServlet, LogoutServlet, RetiroServlet, TransferenciaServlet* en el paquete *controller.*

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.

Figura 65. Creación de controladores - Cliente web

CREACIÓN DE SERVICIOS

Se crean los archivos *EurekaBankClient* en la carpeta *service.*

Una captura de pantalla de un celular con texto e imagen

El contenido generado por IA puede ser incorrecto.

Figura 66. Creación de servicios - Cliente web

CREACIÓN DE VISTAS

Se crear los archivos *balances, deposito, error, login, movimientos, retiro y transferencia* en la carpeta *Web Pages.*

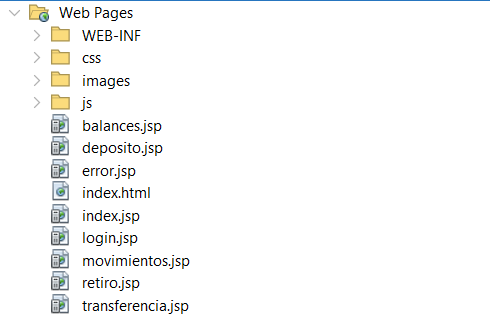
**

Figura 67. Creación de vistas - Cliente web

## EJECUCIÓN DEL CLIENTE DE WEB

Figura 68. Login - Cliente web

Figura 69. Depósito- Cliente web

Figura 70. Retiro- Cliente web

Figura 71. Transferencia - Cliente web

Figura 72. Historial de Movimientos- Cliente web

Figura 73. Balances- Cliente web

## CLIENTE MÓVIL

CREACIÓN DEL PROYECTO

El proyecto se debe crear en Android Studio. Se selecciona la opción ‘New Project’, en la pantalla siguiente ‘Empty Activity’ y finalmente se configura el proyecto con nombre, ubicación y la opción ‘Kotlin’ en *Build Configuration Laguaje.* Finalizamos haciendo click en el botón ‘Finish’.

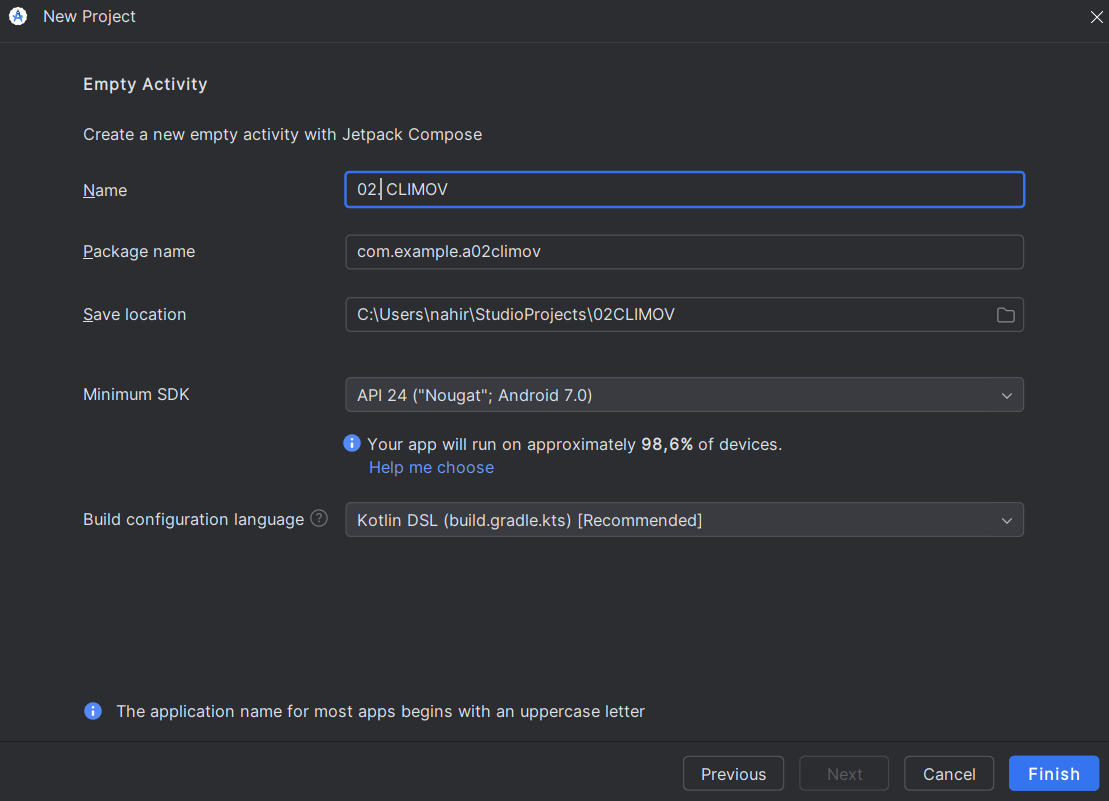


Figura 74. Configuración del proyecto - Cliente móvil

ESTRUCTURA DEL PROYECTO

Una vez creado el proyecto se crea la estructura de paquetes basada en el patrón MVC

Texto

El contenido generado por IA puede ser incorrecto.

Figura 75. Estructura del proyecto - Cliente móvil

## CODIFICACIÓN DEL PROYECTO CLIENTE MÓVIL

### CREACIÓN DE MODELOS

Se crean las clases *Cuenta, Movimiento y OperacionResult* dentro del paquete *models.*

*Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.*

Figura 76. Creación de modelos - Cliente móvil

CREACIÓN DE REPOSITORIOS

Se crean los archivos *EurekaBanRepository* en el paquete *repository.*

Texto

El contenido generado por IA puede ser incorrecto.

Figura 77. Creación de controladores - Cliente móvil

CREACIÓN DE SERVICIOS

Se crean el archivo *AuthService, BalancesService, DepositoService, MovimientoService, RetiroService y TransferenciaService* en el paquete *services.*

Texto

El contenido generado por IA puede ser incorrecto.

Figura 78. Creación de servicios - Cliente móvil

CREACIÓN DE VISTAS

Se crean los archivos en las carpetas *activity, adapter, fragments y viewmodel*

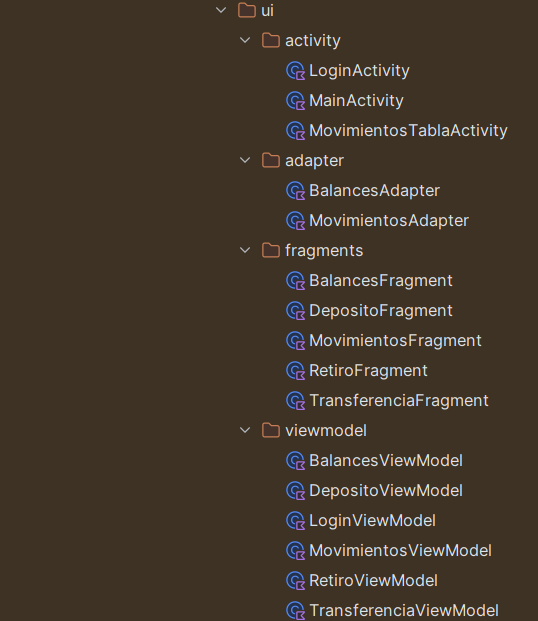


Figura 79. Creación de vistas - Cliente móvil

## EJECUCIÓN DEL CLIENTE DE MÓVIL

Figura 80. Login - Cliente móvil

Figura 81. Depósito- Cliente móvil

Figura 82. Retiro- Cliente móvil

Figura 83. Transferencia - Cliente móvil

Figura 84. Historial de Movimientos- Cliente móvil

Figura 85. Balances- Cliente móvil

# CONCLUSIONES

* El desarrollo del proyecto permitió integrar múltiples tecnologías y herramientas necesarias para la construcción de un sistema completo basado en arquitectura REST, aplicando buenas prácticas de ingeniería de software. La instalación y configuración de entornos como NetBeans, Android Studio, Docker y DBeaver facilitaron un flujo de trabajo ordenado y reproducible, garantizando que todos los componentes del sistema se ejecuten de forma consistente.
* La implementación del servicio EurekaBank evidenció la importancia de separar correctamente la lógica en capas, utilizando entidades, repositorios, DTOs, servicios y controladores. Esto dio como resultado soluciones mantenibles, escalables y con un nivel adecuado de desacoplamiento. Asimismo, el uso de Digital Ocean para el despliegue de la base de datos en la nube asegura la portabilidad y evita inconsistencias entre entornos.
* La construcción de los clientes (consola, escritorio, web y móvil) permitió evidenciar la correcta comunicación con los servicios SOAP, demostrando la interoperabilidad entre tecnologías Java y Android. Además, se verificó el funcionamiento completo del flujo de negocio, desde la realización de retiros, transferencias, depósitos hasta la visualización de movimientos y balances.
* En conjunto, el proyecto consolida los aprendizajes del ciclo completo de desarrollo de software: configuración de herramientas, diseño de arquitectura, modelado de datos, exposición de servicios SOAP, despliegue de bases de datos en la nube, consumo desde distintos clientes y validación funcional mediante pruebas controladas.

# RECOMENDACIONES

1. Mantener la estructura modular utilizada en el proyecto para futuros desarrollos, ya que permite escalar funcionalidades sin comprometer el diseño inicial.
2. Implementar manejo de excepciones más detallado tanto en el servidor como en los clientes, de modo que los mensajes de error sean más claros y faciliten el diagnóstico de fallos.
3. Añadir validaciones más estrictas en los servicios, especialmente en operaciones críticas como otorgamiento de créditos, registro de movimientos y creación de facturas.
4. Integrar un sistema de logs persistentes en los servidores, lo cual permitiría rastrear solicitudes y analizar comportamientos anómalos en escenarios reales de producción.
5. Para optimizar la portabilidad y el despliegue, se sugiere utilizar Docker para contenerizar la instancia de MySQL dentro del *Droplet*. Esto desacopla la base de datos del sistema operativo del servidor, facilitando la migración entre entornos, simplificando la gestión de dependencias y permitiendo una recuperación ante desastres más ágil mediante el reinicio de contenedores.
6. Optimizar consultas y operaciones sobre la base de datos mediante índices, vistas o procedimientos almacenados cuando el sistema escale a un mayor volumen de datos.
7. Implementar pruebas unitarias y pruebas de integración para asegurar la estabilidad del sistema ante cambios futuros.
8. Automatizar la creación del entorno mediante scripts o archivos de configuración como Docker Compose, lo cual facilitaría aún más la preparación del entorno para nuevos desarrolladores.
9. Mejorar la interfaz de usuario en el cliente web y móvil incorporando retroalimentación visual más precisa, controles de validación e indicadores de carga.
10. Documentar la API REST utilizando una herramienta como Swagger/OpenAPI, lo que permitirá generar documentación interactiva y facilitar su consumo desde otros sistemas.