Imagen que contiene interior, mamífero, gato, viendo

Descripción generada automáticamente

**Tema**

EUREKABANK BDD NUBE REST DOTNET

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](#_Toc216221124)

[PARTE PRÁCTICA 7](#_Toc216221125)

[INSTALACIÓN DE HERRAMIENTAS 7](#_Toc216221126)

[ESTRUCTURA DE PROYECTO 7](#_Toc216221127)

[INSTALACIÓN Y CONFIGURACIÓN DE HERRAMIENTAS 7](#_Toc216221128)

[INSTALACIÓN Y CONFIGURACIÓN DE VISUAL STUDIO 8](#_Toc216221129)

[INSTALACIÓN Y CONFIGURACIÓN DE ANDROID STUDIO 9](#_Toc216221130)

[INSTALACIÓN Y CONFIGURACIÓN DE DOCKER DESKTOP 11](#_Toc216221131)

[INSTALACIÓN Y CONFIGURACIÓN DE DBEAVER 13](#_Toc216221132)

[CREACIÓN DE LA BASE DE DATOS 15](#_Toc216221133)

[DEFINICIÓN DEL CONTENEDOR 15](#_Toc216221134)

[GESTIÓN DEL SERVIDOR REMOTO 16](#_Toc216221135)

[DESPLIEGUE DEL SERVICIO 17](#_Toc216221136)

[VERIFICACIÓN DE LA CONEXIÓN 17](#_Toc216221137)

[APLICACIÓN SERVIDOR 18](#_Toc216221138)

[CREACIÓN DEL PROYECTO SERVIDOR EUREKABANK 18](#_Toc216221139)

[CODIFICACIÓN DEL PROYECTO SERVIDOR 19](#_Toc216221140)

[CONEXIÓN A LA BASE DE DATOS 19](#_Toc216221141)

[CREACIÓN DE PAQUETES 20](#_Toc216221142)

[CREACIÓN DE MODELOS 20](#_Toc216221143)

[CREACIÓN DE SERVICIO 22](#_Toc216221144)

[CREACIÓN DE CONTROLADOR 25](#_Toc216221145)

[PRUEBAS DE LOS SERVICIOS 28](#_Toc216221146)

[CODIFICACIÓN DE LOS CLIENTES 30](#_Toc216221147)

[CLIENTE CONSOLA 30](#_Toc216221148)

[CREACIÓN DEL PROYECTO 30](#_Toc216221149)

[ESTRUCTURA DEL PROYECTO 31](#_Toc216221150)

[CODIFICACIÓN DEL PROYECTO CLIENTE CONSOLA 32](#_Toc216221151)

[CREACIÓN DE MODELOS 32](#_Toc216221152)

[EJECUCIÓN DEL CLIENTE DE CONSOLA 32](#_Toc216221153)

[CLIENTE ESCRITORIO 34](#_Toc216221154)

[CREACIÓN DEL PROYECTO 34](#_Toc216221155)

[ESTRUCTURA DEL PROYECTO 34](#_Toc216221156)

[CODIFICACIÓN DEL PROYECTO CLIENTE ESCRITORIO 34](#_Toc216221157)

[CREACIÓN DE MODELOS 34](#_Toc216221158)

[EJECUCIÓN DEL CLIENTE DE ESCRITORIO 35](#_Toc216221159)

[CLIENTE WEB 37](#_Toc216221160)

[CREACIÓN DEL PROYECTO 37](#_Toc216221161)

[ESTRUCTURA DEL PROYECTO 37](#_Toc216221162)

[CODIFICACIÓN DEL PROYECTO CLIENTE WEB 37](#_Toc216221163)

[CREACIÓN DE CONTROLADORES 37](#_Toc216221164)

[CREACIÓN DE SERVICIOS 38](#_Toc216221165)

[CREACIÓN DE VISTAS 38](#_Toc216221166)

[EJECUCIÓN DEL CLIENTE DE WEB 38](#_Toc216221167)

[CLIENTE MÓVIL 39](#_Toc216221168)

[CREACIÓN DEL PROYECTO 39](#_Toc216221169)

[ESTRUCTURA DEL PROYECTO 39](#_Toc216221170)

[CODIFICACIÓN DEL PROYECTO CLIENTE MÓVIL 40](#_Toc216221171)

[CREACIÓN DE MODELOS 40](#_Toc216221172)

[CREACIÓN DE REPOSITORIOS 40](#_Toc216221173)

[CREACIÓN DE SERVICIOS 40](#_Toc216221174)

[CREACIÓN DE VISTAS 41](#_Toc216221175)

[EJECUCIÓN DEL CLIENTE DE MÓVIL 41](#_Toc216221176)

[CONCLUSIONES 42](#_Toc216221177)

[RECOMENDACIONES 42](#_Toc216221178)

**ÍNDICE DE IMÁGENES**

[Figura 1. Creación de la carpeta raíz 7](#_Toc216221179)

[Figura 2. Estructura inicial del proyecto 7](#_Toc216221180)

[Figura 3. Página oficial de descarga de Visual Studio 8](#_Toc216221181)

[Figura 4. Instalador Visual Studio 8](#_Toc216221182)

[Figura 5. Pantalla inicial de instalación Visual Studio 8](#_Toc216221183)

[Figura 6. Configuración instalación Visual Studio 9](#_Toc216221184)

[Figura 7. Pantalla de inicio de Visual Studio 9](#_Toc216221185)

[Figura 8. Página oficial de descarga de Android Studio 10](#_Toc216221186)

[Figura 9. Ejecutable Android Studio 10](#_Toc216221187)

[Figura 10. Pantalla inicial de instalación de Android Studio 10](#_Toc216221188)

[Figura 11. Aplicación Android Studio 11](#_Toc216221189)

[Figura 12. Administrador de tareas 11](#_Toc216221190)

[Figura 13. Características de Windows 11](#_Toc216221191)

[Figura 14. Página oficial de descarga de Docker Desktop 12](#_Toc216221192)

[Figura 15. Ejecutable Docker Desktop 12](#_Toc216221193)

[Figura 16. Pantalla inicial de instalación de Docker Desktop 12](#_Toc216221194)

[Figura 17. Configuración inicial de Docker Desktop 13](#_Toc216221195)

[Figura 18. Aplicación Docker Desktop 13](#_Toc216221196)

[Figura 19. Página oficial de descarga de DBeaver Community 13](#_Toc216221197)

[Figura 20. Ejecutable DBeaver 14](#_Toc216221198)

[Figura 21. Pantalla inicial de instalación de DBeaver 14](#_Toc216221199)

[Figura 22. Términos y condiciones DBeaver 14](#_Toc216221200)

[Figura 23. Instalación DBeaver 15](#_Toc216221201)

[Figura 24. Aplicación DBeaver 15](#_Toc216221202)

[Figura 25. Creación BD - Docker compose 16](#_Toc216221203)

[Figura 26. Creación BD - Terminal MobaXterm renombrando directorios en el servidor remoto 17](#_Toc216221204)

[Figura 27. Creación BD - Terminal corriendo el contenedor 17](#_Toc216221205)

[Figura 28. Conexión exitosa a BD eureka\_bank en DBeaver 18](#_Toc216221206)

[Figura 29. Creación del proyecto – Servidor EurekaBank 18](#_Toc216221207)

[Figura 30. Configuración del proyecto 1 - Servidor EurekaBank 19](#_Toc216221208)

[Figura 31. Configuración del proyecto 2 - Servidor EurekaBank 19](#_Toc216221209)

[Figura 33. Estructura del proyecto - Servidor EurekaBank 20](#_Toc216221210)

[Figura 34. Creación de entidades - Servidor EurekaBank 20](#_Toc216221211)

[Figura 35. Creación de servicio - Servidor EurekaBank 22](#_Toc216221212)

[Figura 36. Creación de controlador - Servidor EurekaBank 25](#_Toc216221213)

[Figura 37. Prueba 1: Login - Servidor EurekaBank 30](#_Toc216221214)

[Figura 38. Prueba 2: Registro de Depósito - Servidor EurekaBank 30](#_Toc216221215)

[Figura 39. Prueba 3: Registro de un Retiro - Servidor EurekaBank 30](#_Toc216221216)

[Figura 40. Prueba 4: Registro de una Transferencia - Servidor EurekaBank 30](#_Toc216221217)

[Figura 41. Prueba 5: Consulta de Historial de Transacciones- Servidor EurekaBank 30](#_Toc216221218)

[Figura 42. Prueba 6: Consulta de Balances - Servidor EurekaBank 30](#_Toc216221219)

[Figura 43. Creación del proyecto - Cliente consola 31](#_Toc216221220)

[Figura 44. Configuración del proyecto - Cliente consola 31](#_Toc216221221)

[Figura 45. Estructura del proyecto - Cliente consola 32](#_Toc216221222)

[Figura 46. Creación de modelos - Cliente consola 32](#_Toc216221223)

[Figura 47. Creación de controladores - Cliente consola 32](#_Toc216221224)

[Figura 48. Creación de vista - Cliente consola 32](#_Toc216221225)

[Figura 49. Login - Cliente consola 33](#_Toc216221226)

[Figura 50. Depósito - Cliente consola 33](#_Toc216221227)

[Figura 51. Retiro- Cliente consola 33](#_Toc216221228)

[Figura 52. Transferencia - Cliente consola 33](#_Toc216221229)

[Figura 53. Historial de Movimientos - Cliente consola 33](#_Toc216221230)

[Figura 54. Balances- Cliente consola 34](#_Toc216221231)

[Figura 55. Estructura del proyecto - Cliente escritorio 34](#_Toc216221232)

[Figura 56. Creación de modelos - Cliente escritorio 34](#_Toc216221233)

[Figura 57. Creación de vistas - Cliente escritorio 35](#_Toc216221234)

[Figura 58. Login - Cliente escritorio 35](#_Toc216221235)

[Figura 59. Depósito- Cliente escritorio 35](#_Toc216221236)

[Figura 60. Retiro- Cliente escritorio 36](#_Toc216221237)

[Figura 61. Transferencia - Cliente escritorio 36](#_Toc216221238)

[Figura 62. Historial de Movimientos- Cliente escritorio 36](#_Toc216221239)

[Figura 63. Balances- Cliente escritorio 37](#_Toc216221240)

[Figura 64. Estructura del proyecto - Cliente web 37](#_Toc216221241)

[Figura 65. Creación de controladores - Cliente web 38](#_Toc216221242)

[Figura 66. Creación de servicios - Cliente web 38](#_Toc216221243)

[Figura 67. Creación de vistas - Cliente web 38](#_Toc216221244)

[Figura 68. Login - Cliente web 38](#_Toc216221245)

[Figura 69. Depósito- Cliente web 38](#_Toc216221246)

[Figura 70. Retiro- Cliente web 38](#_Toc216221247)

[Figura 71. Transferencia - Cliente web 39](#_Toc216221248)

[Figura 72. Historial de Movimientos- Cliente web 39](#_Toc216221249)

[Figura 73. Balances- Cliente web 39](#_Toc216221250)

[Figura 74. Configuración del proyecto - Cliente móvil 39](#_Toc216221251)

[Figura 75. Estructura del proyecto - Cliente móvil 40](#_Toc216221252)

[Figura 76. Creación de modelos - Cliente móvil 40](#_Toc216221253)

[Figura 77. Creación de controladores - Cliente móvil 40](#_Toc216221254)

[Figura 78. Creación de servicios - Cliente móvil 41](#_Toc216221255)

[Figura 79. Creación de vistas - Cliente móvil 41](#_Toc216221256)

[Figura 80. Login - Cliente móvil 41](#_Toc216221257)

[Figura 81. Depósito- Cliente móvil 41](#_Toc216221258)

[Figura 82. Retiro- Cliente móvil 42](#_Toc216221259)

[Figura 83. Transferencia - Cliente móvil 42](#_Toc216221260)

[Figura 84. Historial de Movimientos- Cliente móvil 42](#_Toc216221261)

[Figura 85. Balances- Cliente móvil 42](#_Toc216221262)

# ÍNDICE DE TABLAS

[Tabla 1. Cadena de conexión a BD - Servidor EurekaBank 19](#_Toc216221263)

[Tabla 2. Clase Cuenta - Servidor EurekaBank 20](#_Toc216221264)

[Tabla 3. Clase DepostioRequest- Servidor EurekaBank 21](#_Toc216221265)

[Tabla 3. Clase LoginRequest- Servidor EurekaBank 21](#_Toc216221266)

[Tabla 3. Clase Movimiento- Servidor EurekaBank 21](#_Toc216221267)

[Tabla 3. Clase RetgiroRequest- Servidor EurekaBank 22](#_Toc216221268)

[Tabla 3. Clase TransferenciaRequest- Servidor EurekaBank 22](#_Toc216221269)

[Tabla 4. Clase MovimientoServicio - Servidor EurekaBank 22](#_Toc216221270)

[Tabla 5. Clase MovimientoController - Servidor EurekaBank 25](#_Toc216221271)

[Tabla 6. Servicios - Servidor EurekaBank 28](#_Toc216221272)

# 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 VISUAL STUDIO

Para la instalación de la herramienta se accede a la página oficial de descarga de Visual studio <https://visualstudio.microsoft.com/es/downloads/> se selecciona la opción ‘Community’, ‘Descarga Gratuita’ e inmediatamente iniciará la descarga

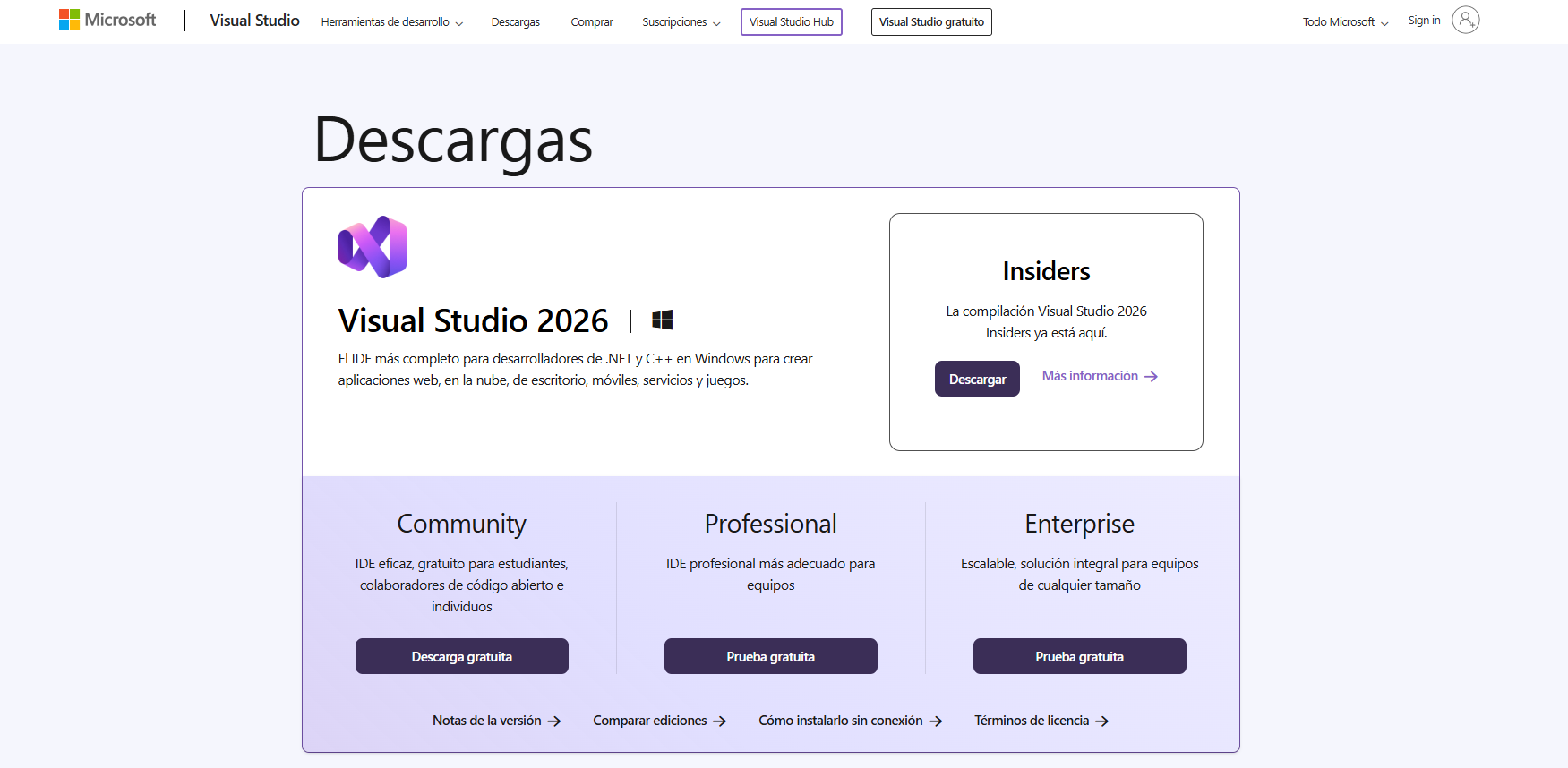


Figura 3. Página oficial de descarga de Visual Studio

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

Imagen que contiene Icono

El contenido generado por IA puede ser incorrecto.

Figura 4. Instalador Visual Studio

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.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

Figura 5. Pantalla inicial de instalación Visual Studio

Una vez finalizada la preparación se configura la Descarga y se selecciona la opción de ‘Instalar’

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

Figura 6. Configuración instalación Visual Studio

Posterior a la instalación se configura cuenta de usuario, opciones de personalización y se iniciará la aplicación

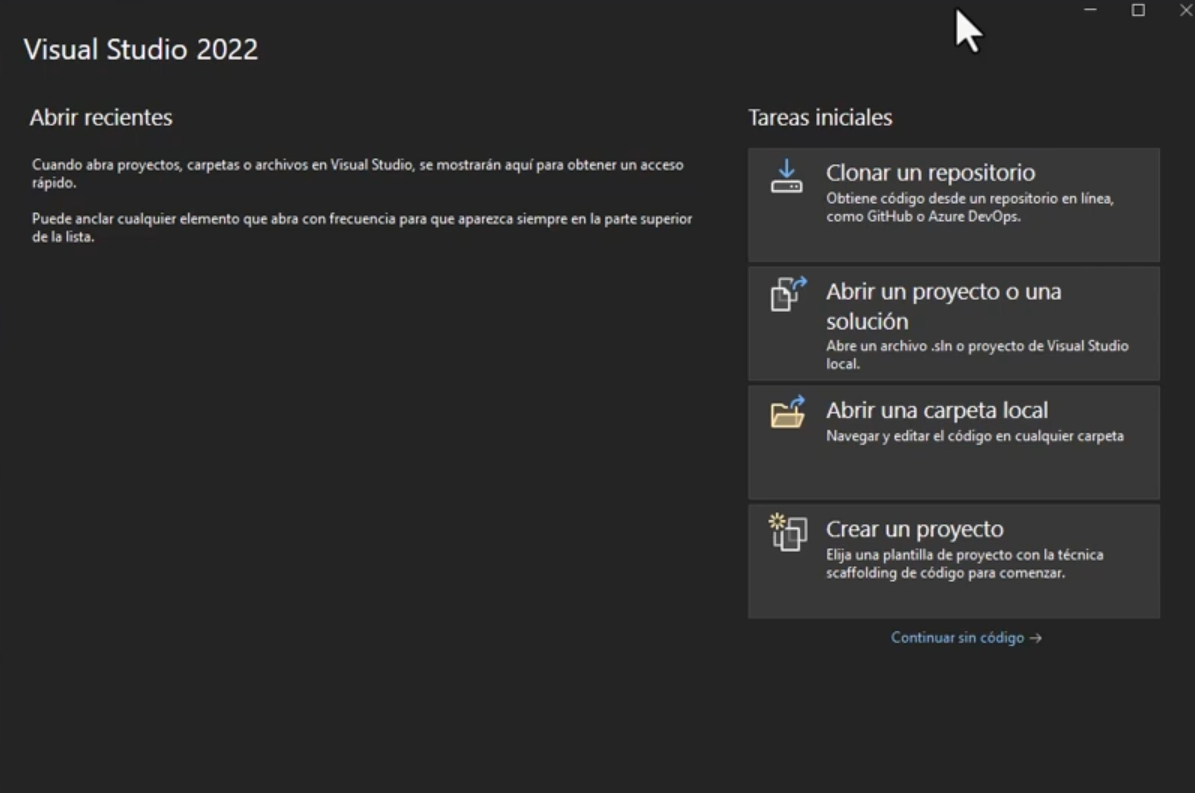


Figura 7. Pantalla de inicio de Visual Studio

### 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.

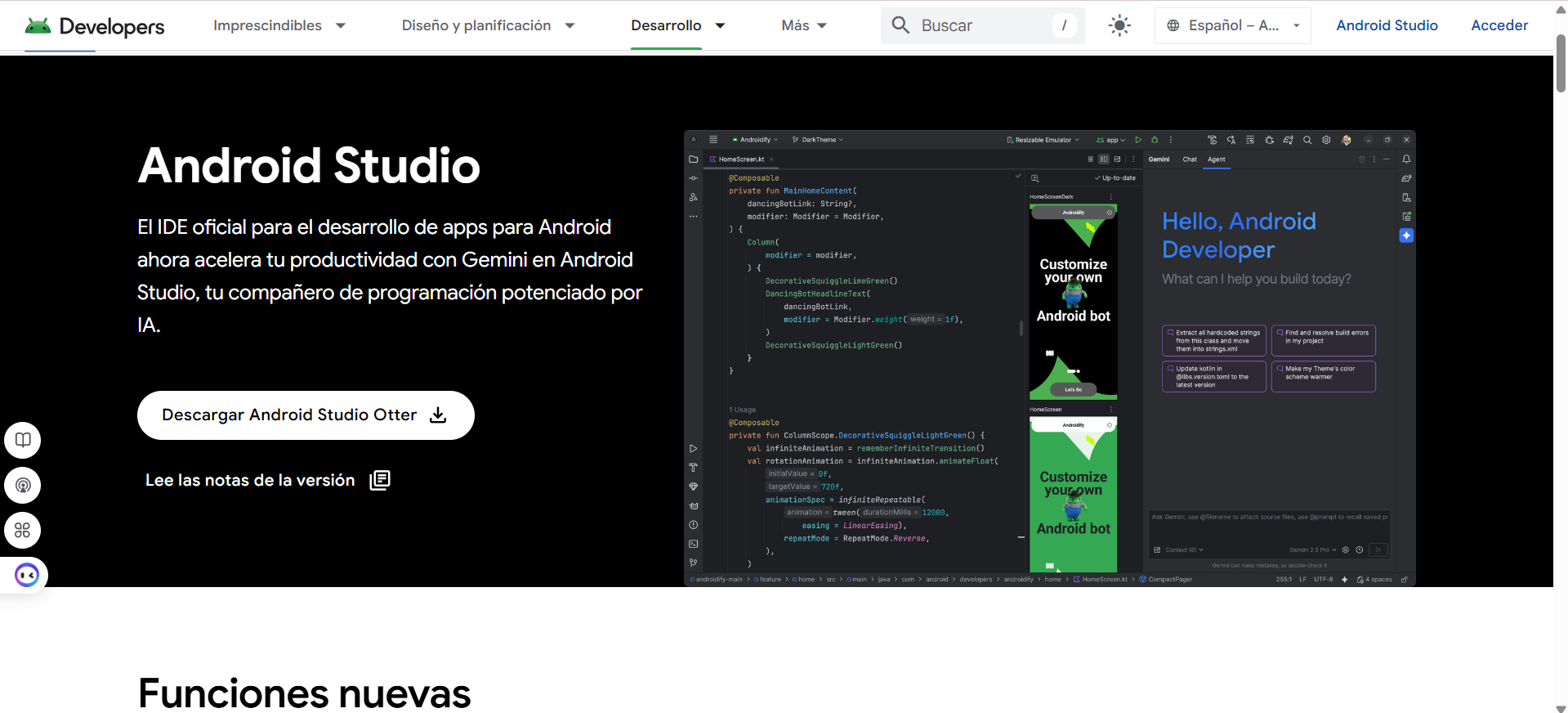


Figura 8. Página oficial de descarga de Android Studio

Al finalizar la descarga se obtendrá el instalador ejecutable.



Figura 9. Ejecutable Android Studio

Al ejecutar el instalador se mostrará la siguiente pantalla:

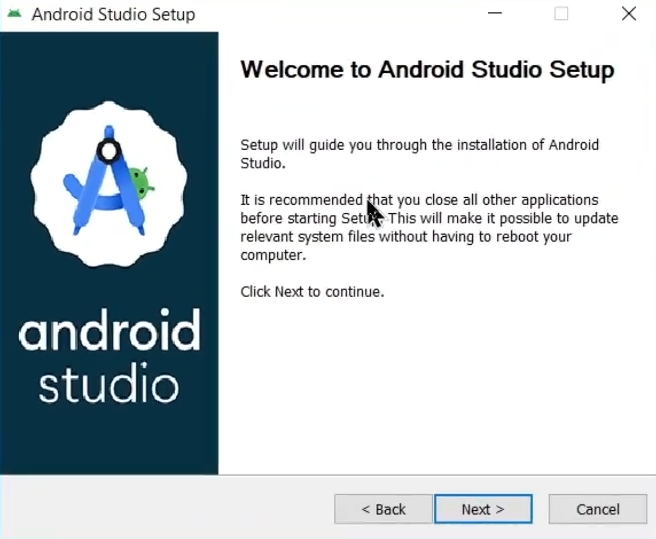


Figura 10. Pantalla inicial de instalación de Android Studio

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.

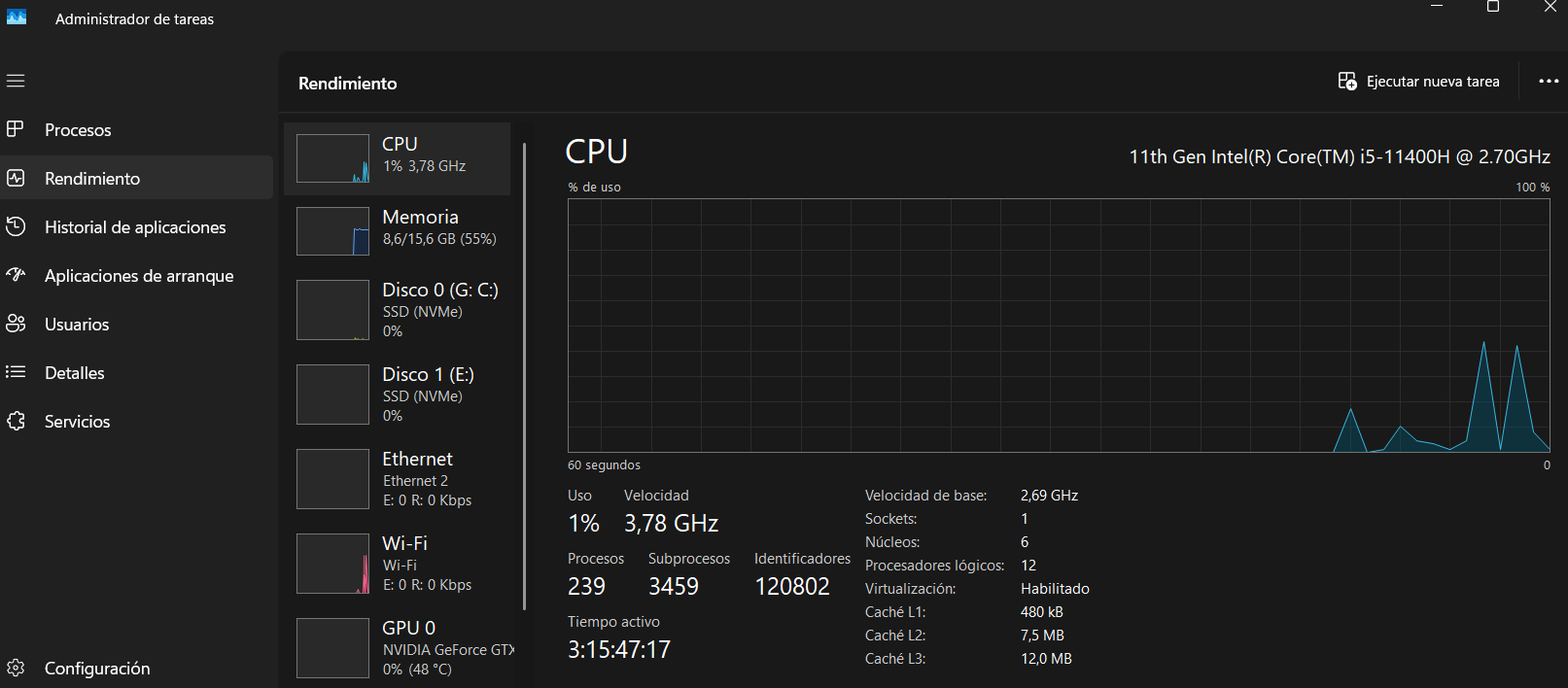


Figura 12. Administrador de tareas

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.

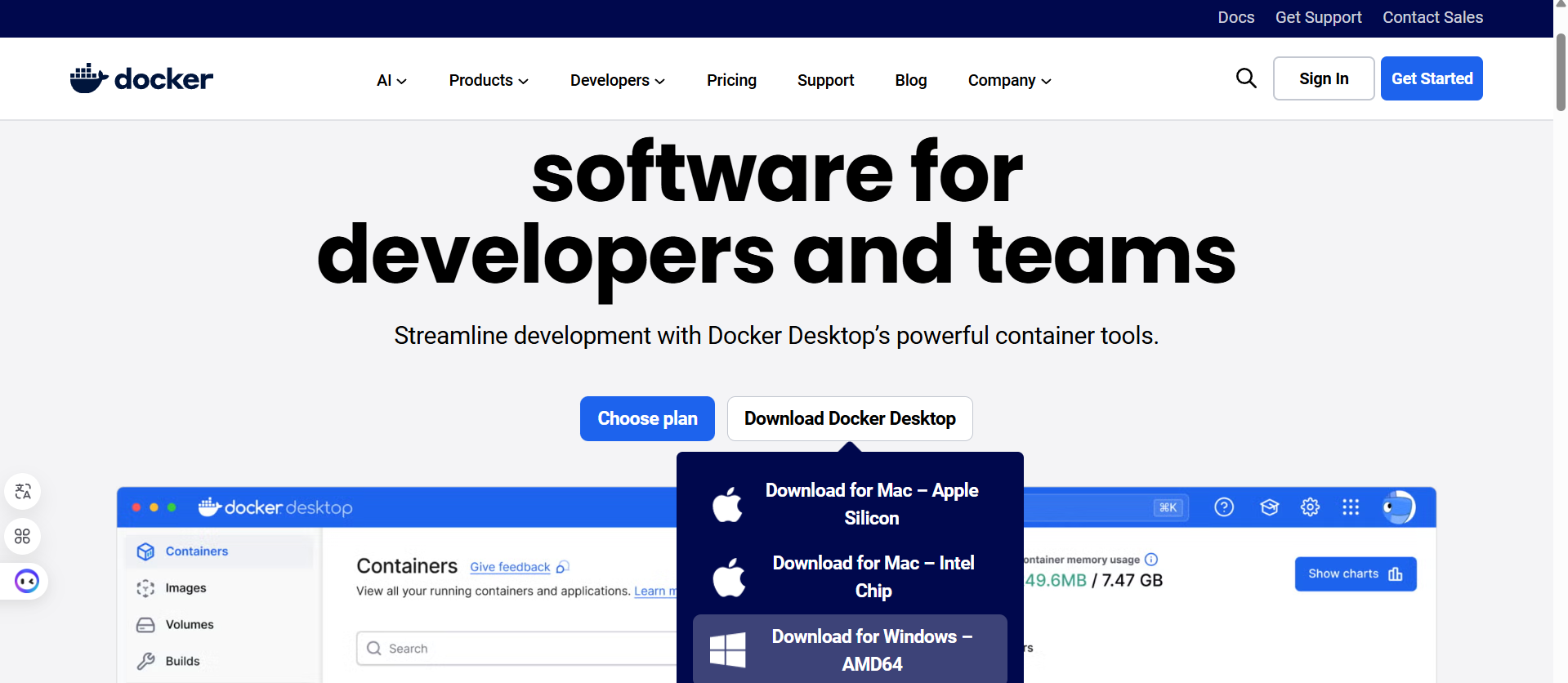


Figura 14. Página oficial de descarga de Docker Desktop

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.

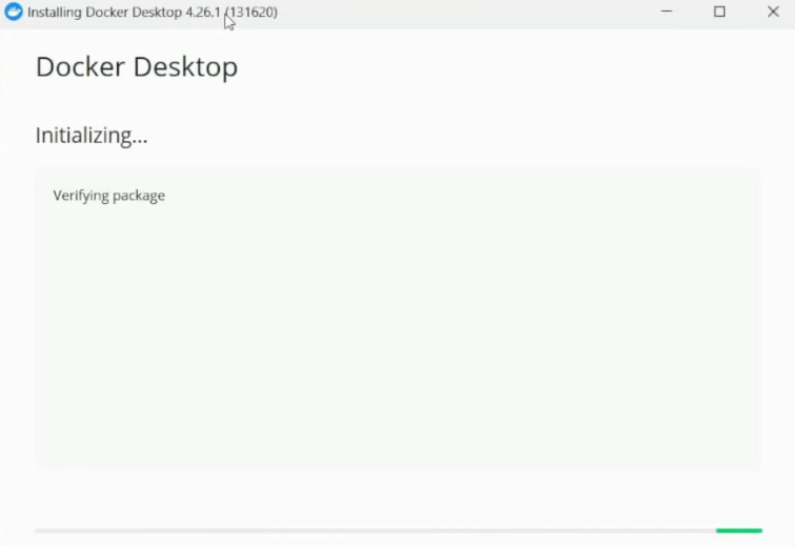


Figura 16. Pantalla inicial de instalación de Docker Desktop

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.

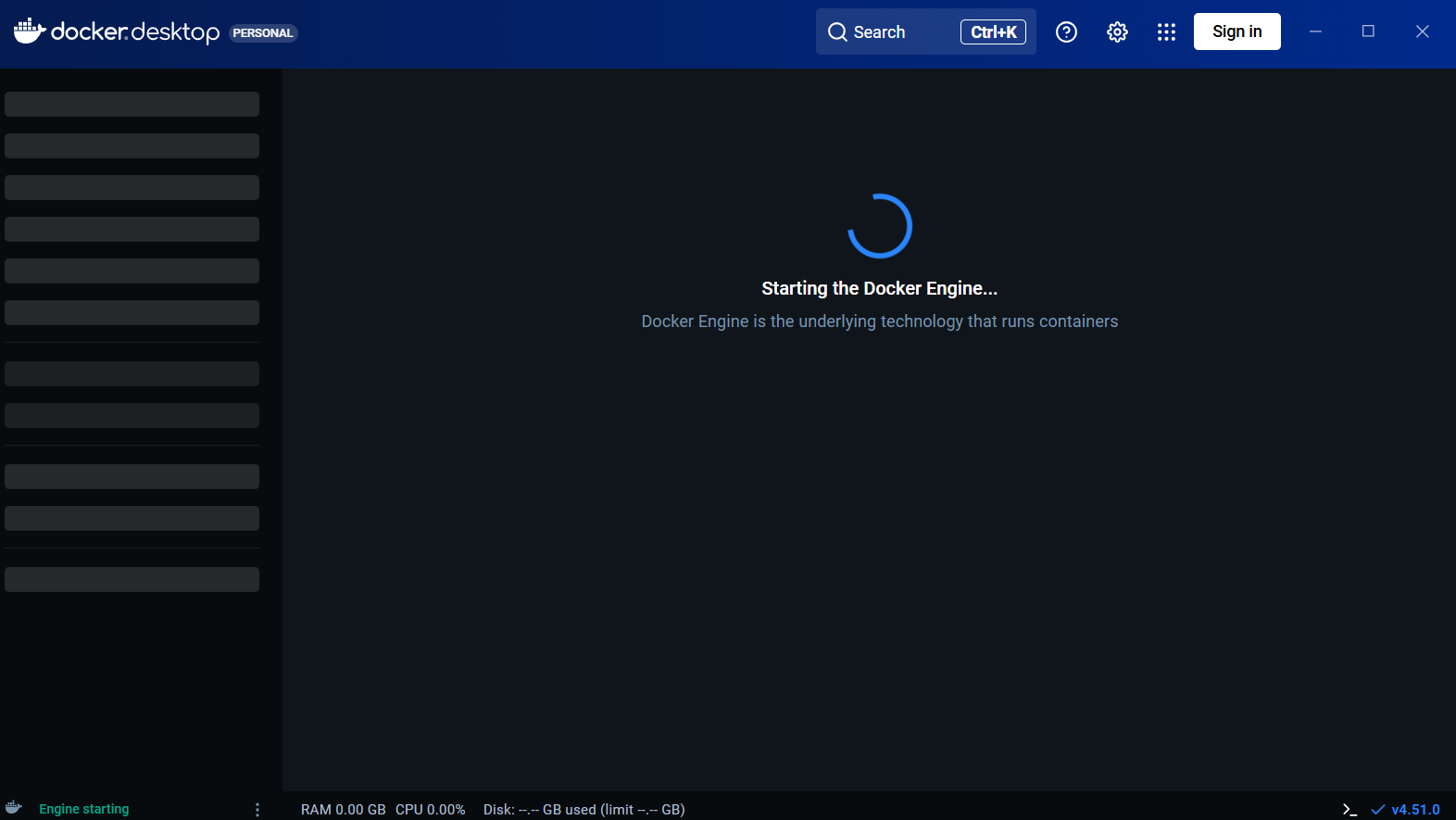


Figura 18. Aplicación Docker Desktop

### INSTALACIÓN Y CONFIGURACIÓN DE DBEAVER

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

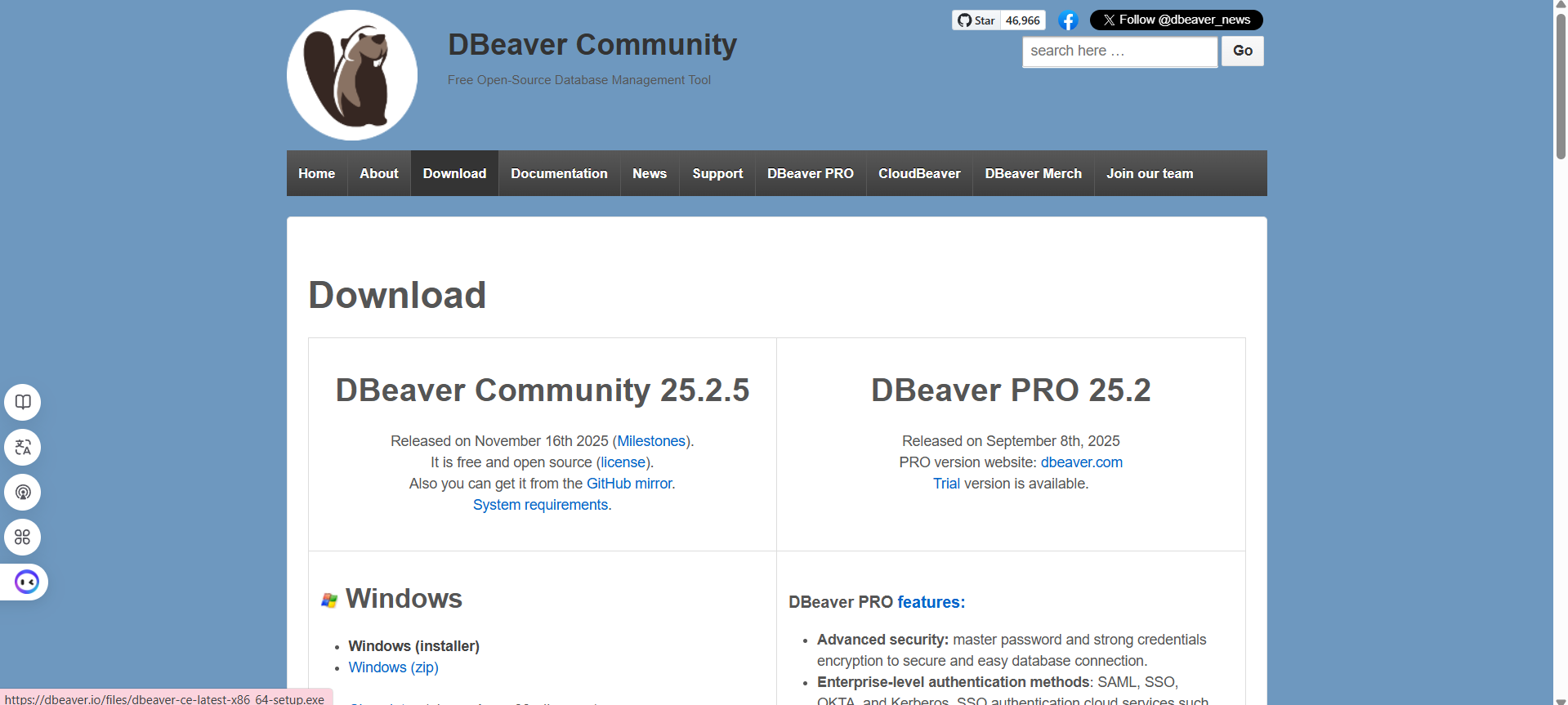


Figura 19. Página oficial de descarga de DBeaver Community

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

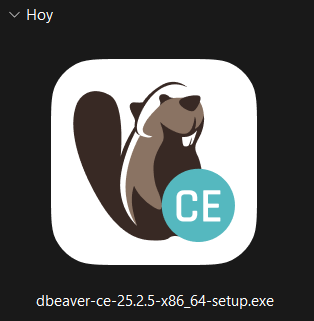


Figura 20. Ejecutable DBeaver

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.

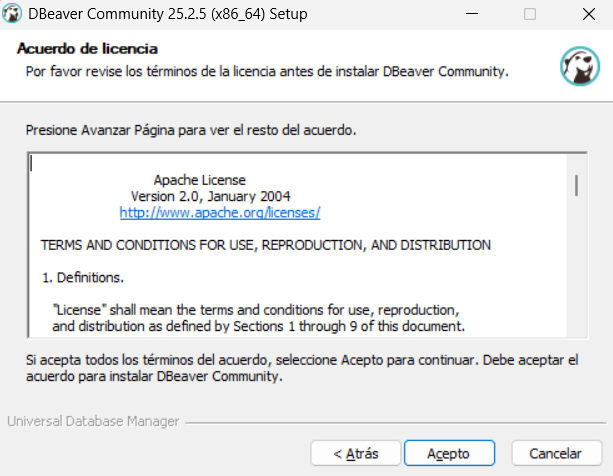


Figura 22. Términos y condiciones DBeaver

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’

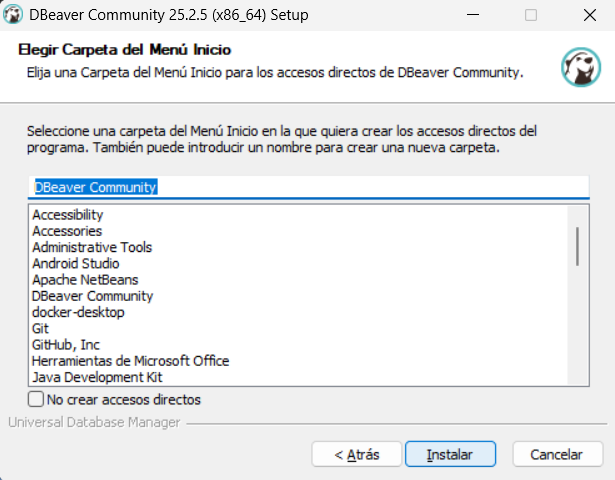


Figura 23. Instalación DBeaver

Cuando finaliza la instalación se ejecutará la aplicación.

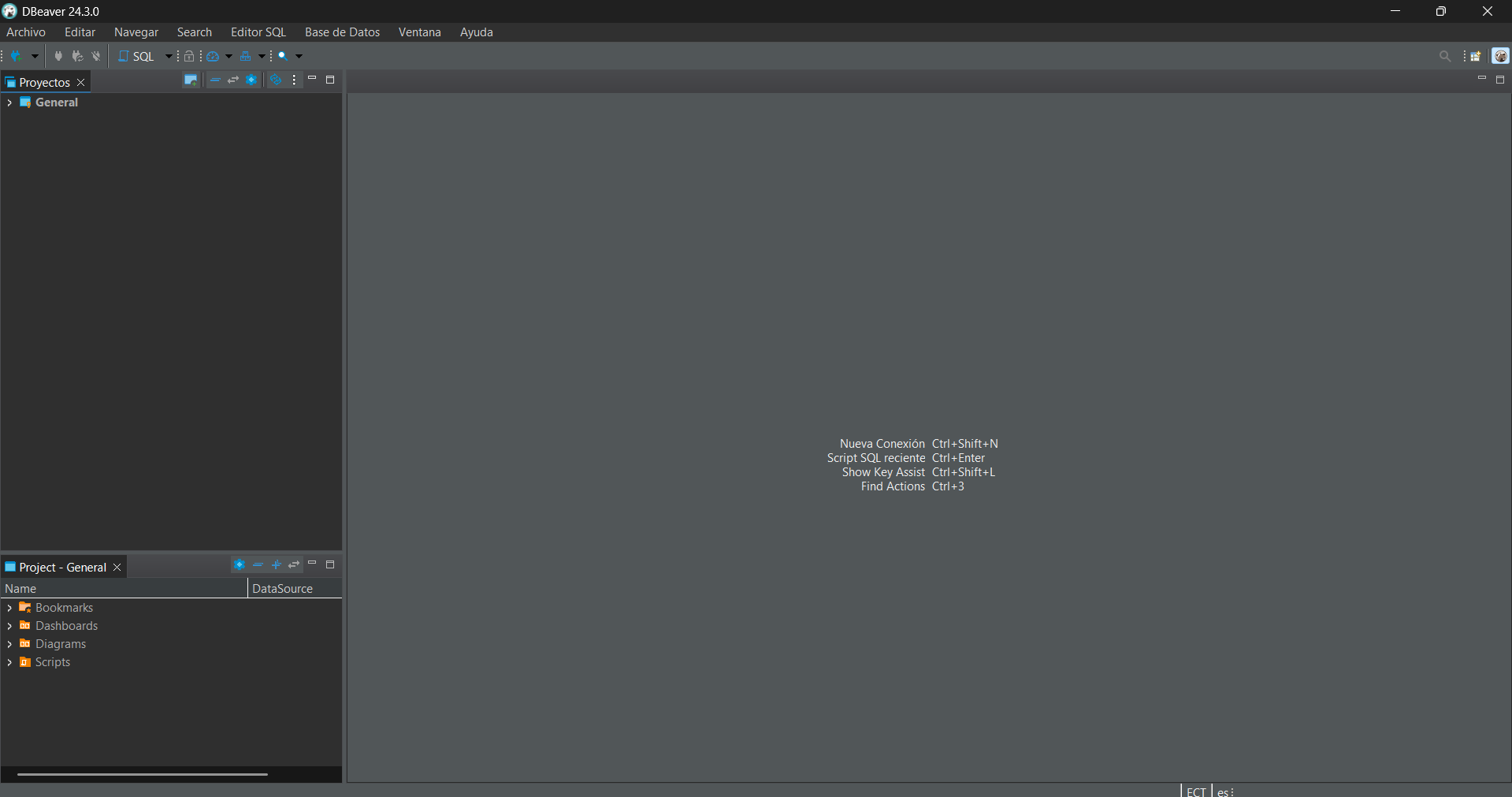


Figura 24. Aplicación DBeaver

# 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.

## 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 **SQL Server** de manera automatizada.

* **Imagen:** Se utilizó mcr.microsoft.com/mssql/server:2022-latest para disponer de la versión más reciente y estable de SQL Server en entorno Linux.
* **Configuración de Entorno:** Se definieron las variables críticas como SA\_PASSWORD (contraseña del administrador del sistema), ACCEPT\_EULA: "Y" (aceptación de licencia) y la edición MSSQL\_PID: "Express".
* **Mapeo de Puertos:** Se expuso el puerto **1434** del servidor anfitrión mapeado al puerto estándar 1433 del contenedor, permitiendo así conexiones remotas personalizadas.
* **Persistencia y Inicialización:** Se configuró un volumen (./init:/docker-entrypoint-initdb.d) y un *entrypoint* personalizado para que, al iniciarse el contenedor, se ejecuten los scripts de configuración iniciales, además de un volumen sqlserver\_data para la persistencia de los datos.

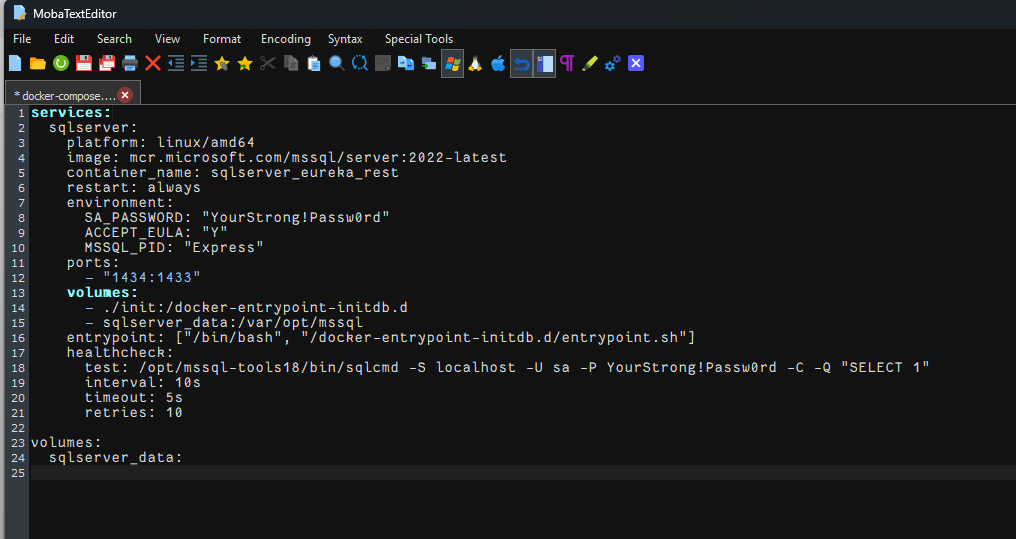


Figura 25. Creación BD - Docker compose

## 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 organizar el proyecto. Como se observa, se estructuró el directorio bajo el nombre BD\_T.I.2.4\_REST\_DOTNET(SqlServer) para diferenciar la arquitectura de despliegue..

Captura de pantalla de un videojuego

El contenido generado por IA puede ser incorrecto.

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 -d

Como se evidencia en la ejecución, Docker realizó automáticamente:

1. La creación de la red virtual (bd\_tl24\_rest\_dotnetsqlserver\_default) para el aislamiento del servicio.
2. La creación del volumen de datos.
3. El inicio exitoso del contenedor identificado como sqlserver\_eureka\_rest.

Texto

El contenido generado por IA puede ser incorrecto.

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 aplicaciones externas o herramientas de administración, se realizó una prueba de conexión remota.

Se configuraron los siguientes parámetros de conexión:

* **Host:** 159.223.110.176 (IP Pública del Droplet).
* **Puerto:** 1434 (Puerto expuesto en el docker-compose).
* **Base de Datos:** eurekabank.
* **Autenticación:** SQL Server Authentication.
* **Usuario:** sa (System Administrator).

La prueba resultó exitosa, confirmando la operatividad del servicio en la nube.

A screenshot of a computer

AI-generated content may be incorrect.

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

# APLICACIÓN SERVIDOR

## CREACIÓN DEL PROYECTO SERVIDOR EUREKABANK

Se crea el proyecto en VisualStudio seleccionando ‘Nuevo Proyecto’, ‘ASP.NET Core Web API’ y ‘Web Application’. Se coloca el nombre del proyecto, ubicación, versión de .NET. Se selecciona Finalmente se hace click en ‘Crear’

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

Figura 29. Creación del proyecto – Servidor EurekaBank

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

Figura 30. Configuración del proyecto 1 - Servidor EurekaBank

Captura de pantalla de computadora

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 ConexionDB en el paquete ec.edu.pinza.db configurando previamente la *connectionString.*

Tabla 1. Cadena de conexión a BD - Servidor EurekaBank

|  |
| --- |
| using System;  using System.Configuration;  using System.Data.SqlClient;  namespace ec.edu.pinza.db  {  public static class ConexionBD  {  private static readonly string connectionString =  ConfigurationManager.ConnectionStrings["EurekaDB"]?.ConnectionString;  public static SqlConnection ObtenerConexion()  {  if (string.IsNullOrWhiteSpace(connectionString))  throw new InvalidOperationException("La cadena de conexión 'EurekaDB' no está configurada.");  var cn = new SqlConnection(connectionString);  try  {  cn.Open();  return cn;  }  catch (SqlException ex)  {  throw new Exception("Error al conectar a la base de datos: " + ex.Message);  }  }  }  } |

### CREACIÓN DE PAQUETES

Se crea la estructura del proyecto incluyendo paquetes: *controlador, db, modelo y servicio*

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 DepositoRequest, LoinRequest, Movimiento, RetiroRequest y TransferenciaRequest* dentro del paquete *modelo.*

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

|  |
| --- |
| using System;  namespace ec.edu.pinza.modelo  {  public class Cuenta  {  public string NumeroCuenta { get; set; }  public string NombreCliente { get; set; }  public double Saldo { get; set; }  public string Moneda { get; set; }  public string Estado { get; set; }  public Cuenta() { }  public Cuenta(string numeroCuenta, string nombreCliente, double saldo, string moneda, string estado)  {  NumeroCuenta = numeroCuenta;  NombreCliente = nombreCliente;  Saldo = saldo;  Moneda = moneda;  Estado = estado;  }  }  } |

Tabla 3. Clase DepostioRequest- Servidor EurekaBank

|  |
| --- |
| using Newtonsoft.Json;  namespace ec.edu.pinza.modelo  {  public class DepositoRequest  {  [JsonProperty("cuenta")]  public string Cuenta { get; set; }    [JsonProperty("importe")]  public double Importe { get; set; }  }  } |

Tabla 3. Clase LoginRequest- Servidor EurekaBank

|  |
| --- |
| using Newtonsoft.Json;  namespace ec.edu.pinza.modelo  {  public class LoginRequest  {  [JsonProperty("username")]  public string Username { get; set; }    [JsonProperty("password")]  public string Password { get; set; }  }  } |

Tabla 3. Clase Movimiento- Servidor EurekaBank

|  |
| --- |
| using System;  namespace ec.edu.pinza.modelo  {  public class Movimiento  {  public string Cuenta { get; set; }  public int NroMov { get; set; }  public DateTime Fecha { get; set; }  public string Tipo { get; set; }  public string Accion { get; set; }  public double Importe { get; set; }  public Movimiento() { }  public Movimiento(string cuenta, int nromov, DateTime fecha, string tipo, string accion, double importe)  {  Cuenta = cuenta;  NroMov = nromov;  Fecha = fecha;  Tipo = tipo;  Accion = accion;  Importe = importe;  }  }  } |

Tabla 3. Clase RetgiroRequest- Servidor EurekaBank

|  |
| --- |
| using Newtonsoft.Json;  namespace ec.edu.pinza.modelo  {  public class RetiroRequest  {  [JsonProperty("cuenta")]  public string Cuenta { get; set; }    [JsonProperty("importe")]  public double Importe { get; set; }  }  } |

Tabla 3. Clase TransferenciaRequest- Servidor EurekaBank

|  |
| --- |
| using Newtonsoft.Json;  namespace ec.edu.pinza.modelo  {  public class TransferenciaRequest  {  [JsonProperty("cuentaOrigen")]  public string CuentaOrigen { get; set; }    [JsonProperty("cuentaDestino")]  public string CuentaDestino { get; set; }    [JsonProperty("importe")]  public double Importe { get; set; }  }  } |

### CREACIÓN DE SERVICIO

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



Figura 35. Creación de servicio - Servidor EurekaBank

Tabla 4. Clase MovimientoServicio - Servidor EurekaBank

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Configuration;  using System.Data;  using System.Data.SqlClient;  using ec.edu.pinza.modelo;  namespace ec.edu.pinza.servicio  {  public class MovimientoServicio  {  public static List<Movimiento> ListarPorCuenta(string cuenta)  {  var lista = new List<Movimiento>();  string sql = @"  SELECT  m.chr\_cuencodigo cuenta,  m.int\_movinumero nromov,  m.dtt\_movifecha fecha,  t.vch\_tipodescripcion tipo,  t.vch\_tipoaccion accion,  m.dec\_moviimporte importe  FROM tipomovimiento t  INNER JOIN movimiento m ON t.chr\_tipocodigo = m.chr\_tipocodigo  WHERE m.chr\_cuencodigo = @cuenta";  using (SqlConnection cn = new SqlConnection(ConfigurationManager.ConnectionStrings["EurekaDB"].ConnectionString))  {  cn.Open();  SqlCommand cmd = new SqlCommand(sql, cn);  cmd.Parameters.AddWithValue("@cuenta", cuenta);  SqlDataReader dr = cmd.ExecuteReader();  while (dr.Read())  {  lista.Add(new Movimiento  {  Cuenta = dr["cuenta"].ToString(),  NroMov = Convert.ToInt32(dr["nromov"]),  Fecha = Convert.ToDateTime(dr["fecha"]),  Tipo = dr["tipo"].ToString(),  Accion = dr["accion"].ToString(),  Importe = Convert.ToDouble(dr["importe"])  });  }  }  return lista;  }  public static void RegistrarDeposito(string cuenta, double importe, string codEmp)  {  EjecutarMovimiento(cuenta, importe, codEmp, "003", true);  }  public static void RegistrarRetiro(string cuenta, double importe, string codEmp)  {  EjecutarMovimiento(cuenta, -importe, codEmp, "004", false);  }  public static void RegistrarTransferencia(string cuentaOrigen, string cuentaDestino, double importe, string codEmp)  {  using (SqlConnection cn = new SqlConnection(ConfigurationManager.ConnectionStrings["EurekaDB"].ConnectionString))  {  cn.Open();  SqlTransaction tx = cn.BeginTransaction();  try  {  EjecutarMovimientoInterno(cuentaOrigen, -importe, codEmp, "009", cn, tx, false);  EjecutarMovimientoInterno(cuentaDestino, importe, codEmp, "008", cn, tx, true);  tx.Commit();  }  catch  {  tx.Rollback();  throw;  }  }  }  private static void EjecutarMovimiento(string cuenta, double importe, string codEmp, string tipoMov, bool permiteNegativo)  {  using (SqlConnection cn = new SqlConnection(ConfigurationManager.ConnectionStrings["EurekaDB"].ConnectionString))  {  cn.Open();  SqlTransaction tx = cn.BeginTransaction();  try  {  EjecutarMovimientoInterno(cuenta, importe, codEmp, tipoMov, cn, tx, permiteNegativo);  tx.Commit();  }  catch  {  tx.Rollback();  throw;  }  }  }  private static void EjecutarMovimientoInterno(string cuenta, double importe, string codEmp, string tipoMov,  SqlConnection cn, SqlTransaction tx, bool permiteNegativo)  {  string sql = "SELECT dec\_cuensaldo, int\_cuencontmov FROM cuenta WHERE chr\_cuencodigo = @cuenta AND vch\_cuenestado = 'ACTIVO'";  SqlCommand cmd = new SqlCommand(sql, cn, tx);  cmd.Parameters.AddWithValue("@cuenta", cuenta);  SqlDataReader dr = cmd.ExecuteReader();  if (!dr.Read())  {  dr.Close();  throw new Exception("ERROR: Cuenta no existe o no está activa.");  }  double saldo = Convert.ToDouble(dr["dec\_cuensaldo"]);  int cont = Convert.ToInt32(dr["int\_cuencontmov"]);  dr.Close();  saldo += importe;  if (!permiteNegativo && saldo < 0)  {  throw new Exception("ERROR: Saldo insuficiente.");  }  cont++;  cmd = new SqlCommand("UPDATE cuenta SET dec\_cuensaldo = @saldo, int\_cuencontmov = @cont WHERE chr\_cuencodigo = @cuenta", cn, tx);  cmd.Parameters.AddWithValue("@saldo", saldo);  cmd.Parameters.AddWithValue("@cont", cont);  cmd.Parameters.AddWithValue("@cuenta", cuenta);  cmd.ExecuteNonQuery();  cmd = new SqlCommand(@"  INSERT INTO movimiento(chr\_cuencodigo, int\_movinumero, dtt\_movifecha, chr\_emplcodigo, chr\_tipocodigo, dec\_moviimporte)  VALUES (@cuenta, @nro, GETDATE(), @codEmp, @tipoMov, @importe)", cn, tx);  cmd.Parameters.AddWithValue("@cuenta", cuenta);  cmd.Parameters.AddWithValue("@nro", cont);  cmd.Parameters.AddWithValue("@codEmp", codEmp);  cmd.Parameters.AddWithValue("@tipoMov", tipoMov);  cmd.Parameters.AddWithValue("@importe", Math.Abs(importe));  cmd.ExecuteNonQuery();  }  public static bool Login(string username, string password)  {  string sql = @"  SELECT COUNT(1)  FROM usuario  WHERE vch\_emplusuario = @user  AND vch\_emplclave = CONVERT(VARCHAR(40), HASHBYTES('SHA1', @pass), 2)  AND vch\_emplestado = 'ACTIVO'";  using (SqlConnection cn = new SqlConnection(ConfigurationManager.ConnectionStrings["EurekaDB"].ConnectionString))  {  cn.Open();  using (SqlCommand cmd = new SqlCommand(sql, cn))  {  cmd.Parameters.Add("@user", SqlDbType.VarChar, 50).Value = username;  cmd.Parameters.Add("@pass", SqlDbType.VarChar, 100).Value = password;  object result = cmd.ExecuteScalar();  return Convert.ToInt32(result) == 1;  }  }  }  public static List<Cuenta> ListarBalances()  {  var lista = new List<Cuenta>();  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";  using (SqlConnection cn = new SqlConnection(ConfigurationManager.ConnectionStrings["EurekaDB"].ConnectionString))  {  cn.Open();  SqlCommand cmd = new SqlCommand(sql, cn);  SqlDataReader dr = cmd.ExecuteReader();  while (dr.Read())  {  lista.Add(new Cuenta  {  NumeroCuenta = dr["numeroCuenta"].ToString(),  NombreCliente = dr["nombreCliente"].ToString(),  Saldo = Convert.ToDouble(dr["saldo"]),  Moneda = dr["moneda"].ToString(),  Estado = dr["estado"].ToString()  });  }  }  return lista;  }  }  } |

### CREACIÓN DE CONTROLADOR

Finalmente se crea el controlador *MovimientoController* en el paquete *controlador.*



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 MovimientoController - Servidor EurekaBank

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Web.Http;  using ec.edu.pinza.modelo;  using ec.edu.pinza.servicio;  namespace ec.edu.pinza.controlador  {  [RoutePrefix("api/movimiento")]  public class MovimientoController : ApiController  {  // GET: api/movimiento/listar?cuenta=00100001  [HttpGet]  [Route("listar")]  public IHttpActionResult ObtenerPorCuenta(string cuenta)  {  try  {  if (string.IsNullOrWhiteSpace(cuenta))  {  return BadRequest("El número de cuenta es requerido.");  }  var movimientos = MovimientoServicio.ListarPorCuenta(cuenta);  return Ok(movimientos);  }  catch (Exception ex)  {  return InternalServerError(new Exception("Error al obtener movimientos: " + ex.Message));  }  }  // POST: api/movimiento/deposito  [HttpPost]  [Route("deposito")]  public IHttpActionResult RegistrarDeposito([FromBody] DepositoRequest request)  {  try  {  if (request == null || string.IsNullOrWhiteSpace(request.Cuenta) || request.Importe <= 0)  {  return BadRequest("Datos de depósito inválidos.");  }  MovimientoServicio.RegistrarDeposito(request.Cuenta, request.Importe, "0001");  return Ok(new { mensaje = "Depósito registrado exitosamente.", resultado = "1" });  }  catch (Exception ex)  {  return InternalServerError(new Exception("Error al registrar depósito: " + ex.Message));  }  }  // POST: api/movimiento/retiro  [HttpPost]  [Route("retiro")]  public IHttpActionResult RegistrarRetiro([FromBody] RetiroRequest request)  {  try  {  if (request == null || string.IsNullOrWhiteSpace(request.Cuenta) || request.Importe <= 0)  {  return BadRequest("Datos de retiro inválidos.");  }  MovimientoServicio.RegistrarRetiro(request.Cuenta, request.Importe, "0001");  return Ok(new { mensaje = "Retiro registrado exitosamente.", resultado = "1" });  }  catch (Exception ex)  {  return InternalServerError(new Exception("Error al registrar retiro: " + ex.Message));  }  }  // POST: api/movimiento/transferencia  [HttpPost]  [Route("transferencia")]  public IHttpActionResult RegistrarTransferencia([FromBody] TransferenciaRequest request)  {  try  {  if (request == null || string.IsNullOrWhiteSpace(request.CuentaOrigen) ||  string.IsNullOrWhiteSpace(request.CuentaDestino) || request.Importe <= 0)  {  return BadRequest("Datos de transferencia inválidos.");  }  if (request.CuentaOrigen == request.CuentaDestino)  {  return BadRequest("La cuenta origen y destino no pueden ser iguales.");  }  MovimientoServicio.RegistrarTransferencia(request.CuentaOrigen, request.CuentaDestino, request.Importe, "0001");  return Ok(new { mensaje = "Transferencia registrada exitosamente.", resultado = "1" });  }  catch (Exception ex)  {  return InternalServerError(new Exception("Error al registrar transferencia: " + ex.Message));  }  }  // POST: api/movimiento/login  [HttpPost]  [Route("login")]  public IHttpActionResult Login([FromBody] LoginRequest request)  {  try  {  if (request == null || string.IsNullOrWhiteSpace(request.Username) || string.IsNullOrWhiteSpace(request.Password))  {  return BadRequest("Usuario y contraseña son requeridos.");  }  bool resultado = MovimientoServicio.Login(request.Username, request.Password);    if (resultado)  {  return Ok(new { mensaje = "Login exitoso.", autenticado = true });  }  else  {  return Ok(new { mensaje = "Credenciales inválidas.", autenticado = false });  }  }  catch (Exception ex)  {  return InternalServerError(new Exception("Error al autenticar: " + ex.Message));  }  }  // GET: api/movimiento/balances  [HttpGet]  [Route("balances")]  public IHttpActionResult TraerBalances()  {  try  {  var balances = MovimientoServicio.ListarBalances();  return Ok(balances);  }  catch (Exception ex)  {  return InternalServerError(new Exception("Error al obtener balances: " + ex.Message));  }  }  }  } |

## PRUEBAS DE LOS SERVICIOS

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

A continuación, se describen lo servicios Implementados

Tabla 6. Servicios - Servidor EurekaBank

| **Método HTTP** | **Endpoint (URI)** | **Descripción** | **Parámetros de Entrada (Input)** | **Respuesta (Output)** |
| --- | --- | --- | --- | --- |
| **GET** | /api/movimiento/listar | Obtiene el historial de movimientos de una cuenta específica. | **Query String:**    cuenta (string): Número de la cuenta a consultar. | **200 OK:** Lista de objetos Movimiento.    **400 BadRequest:** Si no se envía la cuenta.    **500 Error:** Excepción del servidor. |
| **POST** | /api/movimiento/deposito | Registra un depósito bancario en una cuenta. | **JSON Body (DepositoRequest):**    { "Cuenta": "string", "Importe": double } | **200 OK:** Objeto JSON con mensaje de éxito y resultado: "1".    **400 BadRequest:** Datos inválidos.    **500 Error:** Excepción del servidor. |
| **POST** | /api/movimiento/retiro | Registra un retiro de dinero de una cuenta. | **JSON Body (RetiroRequest):**    { "Cuenta": "string", "Importe": double } | **200 OK:** Objeto JSON con mensaje de éxito y resultado: "1".    **400 BadRequest:** Datos inválidos.    **500 Error:** Excepción del servidor. |
| **POST** | /api/movimiento/transferencia | Realiza una transferencia de dinero entre dos cuentas. | **JSON Body (TransferenciaRequest):**    { "CuentaOrigen": "string", "CuentaDestino": "string", "Importe": double } | **200 OK:** Objeto JSON con mensaje de éxito y resultado: "1".    **400 BadRequest:** Datos inválidos o cuentas iguales.    **500 Error:** Excepción del servidor. |
| **POST** | /api/movimiento/login | Autentica a un usuario (empleado) en el sistema. | **JSON Body (LoginRequest):**    { "Username": "string", "Password": "string" } | **200 OK:** JSON con autenticado: true/false y mensaje.    **400 BadRequest:** Faltan credenciales.    **500 Error:** Excepción del servidor. |
| **GET** | /api/movimiento/balances | Lista todas las cuentas activas con sus saldos actuales. | **Ninguno** | **200 OK:** Lista de objetos con balances.    **500 Error:** Excepción del servidor. |

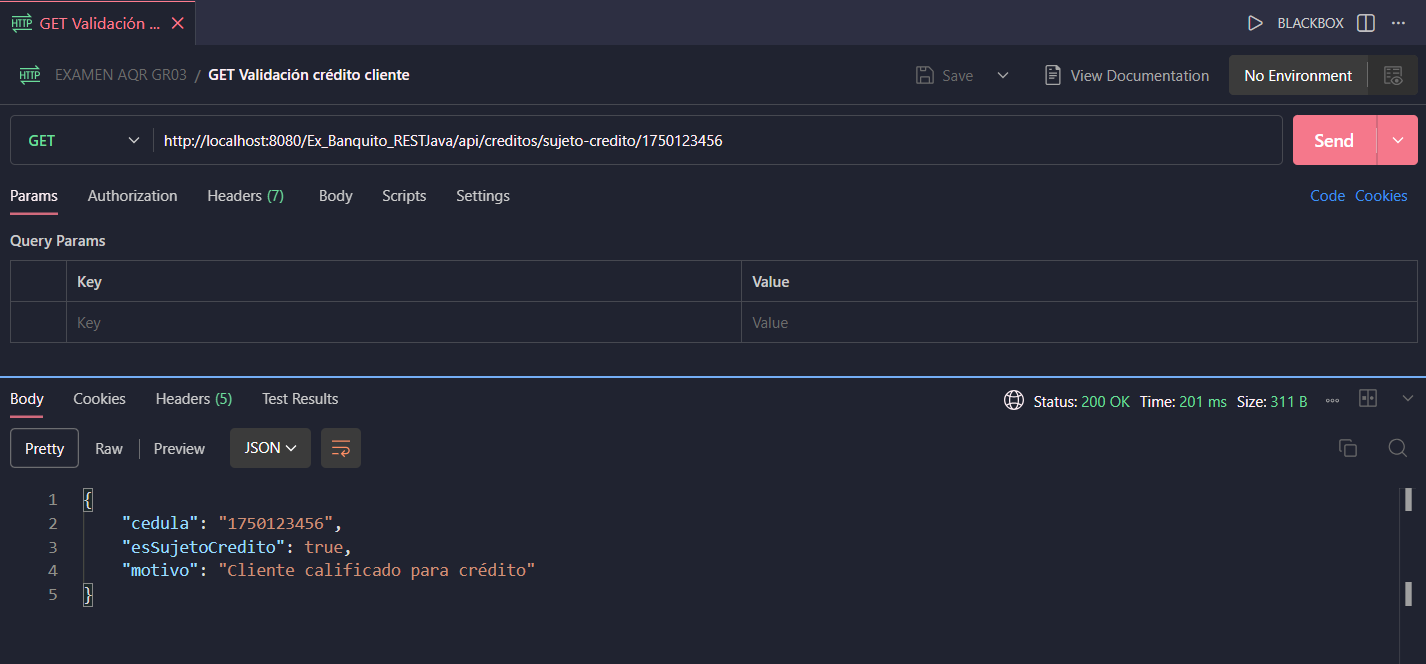


Figura 37. Prueba 1: Login - Servidor EurekaBank

Captura de pantalla de un celular

El contenido generado por IA puede ser incorrecto.

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

Captura de pantalla de un celular

El contenido generado por IA puede ser incorrecto.

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

Captura de pantalla de un celular

El contenido generado por IA puede ser incorrecto.

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

Interfaz de usuario gráfica, Texto, Aplicación

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, Correo electrónico

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

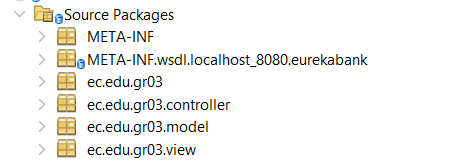


Figura 45. Estructura del proyecto - Cliente consola

## 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

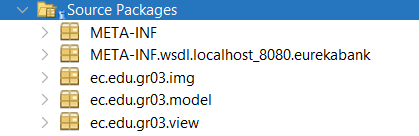


Figura 55. Estructura del proyecto - Cliente escritorio

## 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

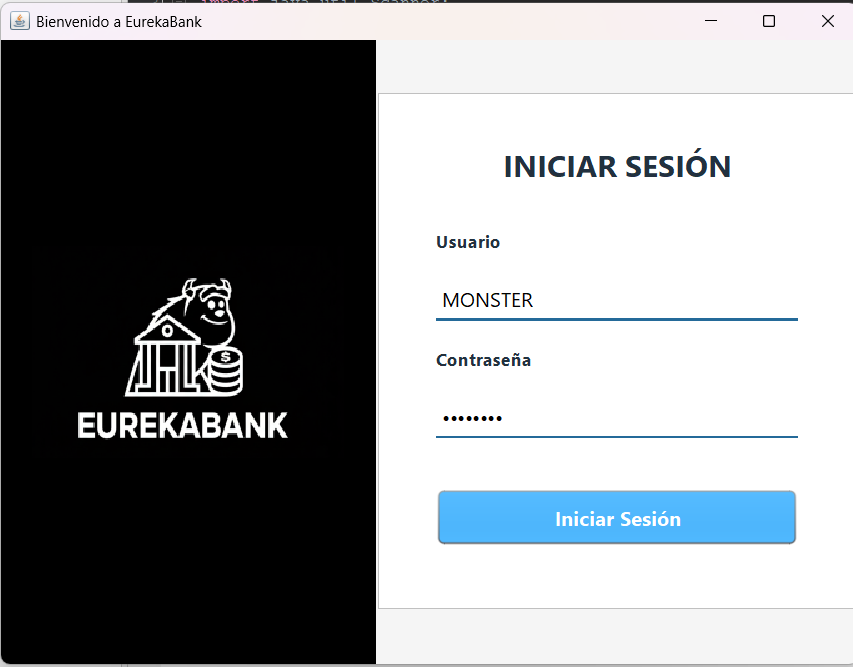


Figura 58. Login - Cliente escritorio

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.*

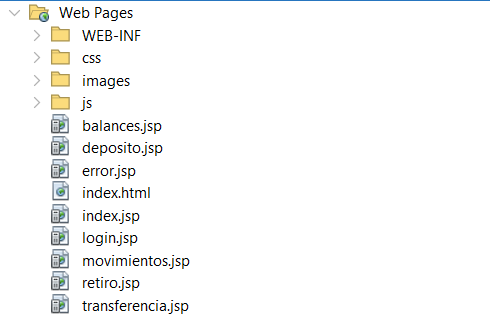
**

Figura 67. Creación de vistas - Cliente web

## EJECUCIÓN DEL CLIENTE DE WEB

## Interfaz de usuario gráfica, Aplicación El contenido generado por IA puede ser incorrecto.

Figura 68. Login - Cliente web

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

Figura 69. Depósito- Cliente web

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

Figura 70. Retiro- Cliente web

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

Figura 71. Transferencia - Cliente web

Interfaz de usuario gráfica, Tabla

El contenido generado por IA puede ser incorrecto.

Figura 72. Historial de Movimientos- Cliente web

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

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’.

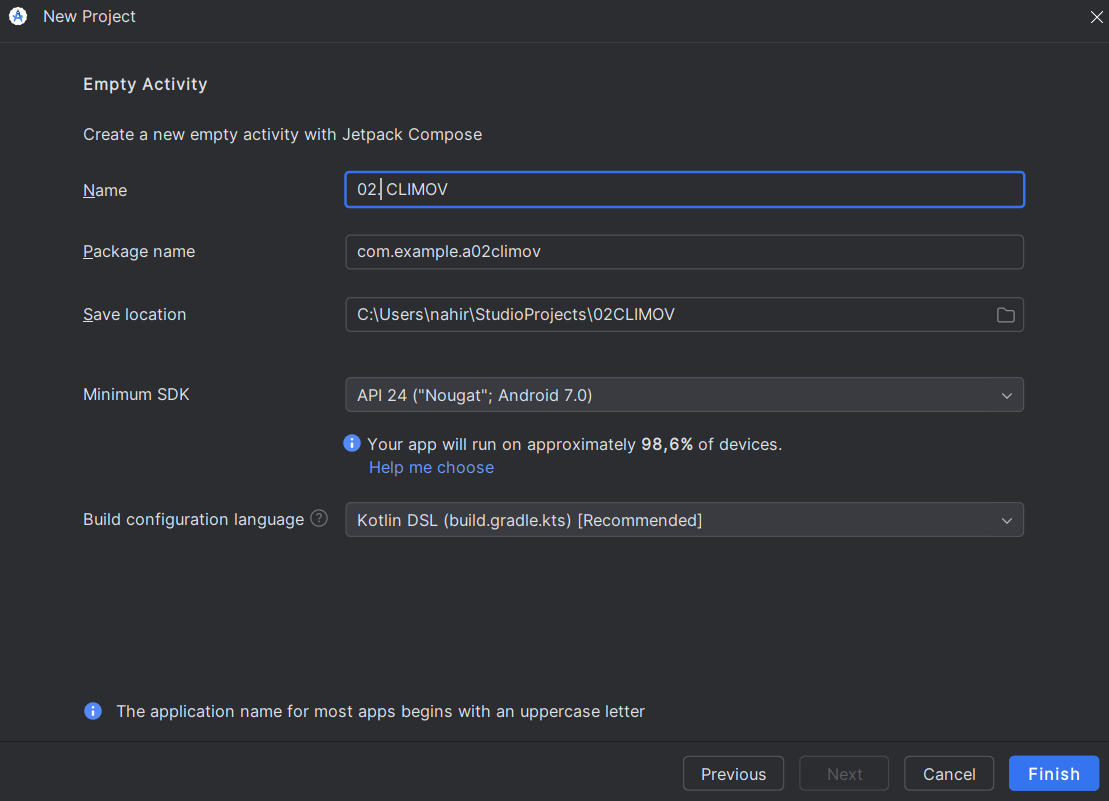


Figura 74. Configuración del proyecto - Cliente móvil

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*

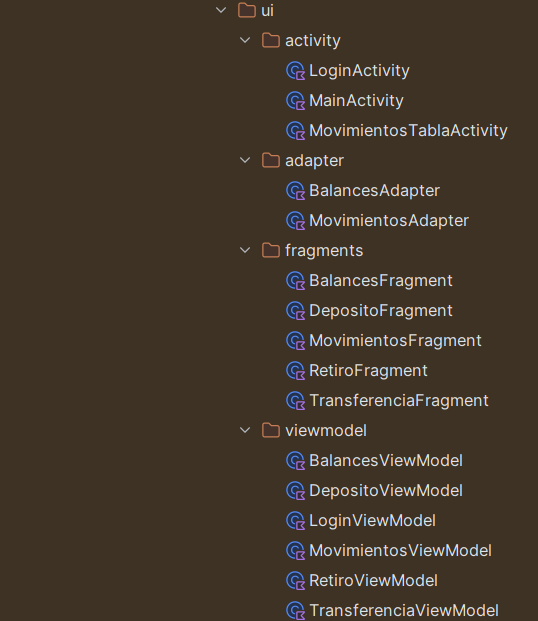


Figura 79. Creación de vistas - Cliente móvil

## EJECUCIÓN DEL CLIENTE DE MÓVIL

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

Figura 80. Login - Cliente móvil

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

El contenido generado por IA puede ser incorrecto.

Figura 81. Depósito- Cliente móvil

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

El contenido generado por IA puede ser incorrecto.

Figura 82. Retiro- Cliente móvil

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Figura 83. Transferencia - Cliente móvil

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

Figura 84. Historial de Movimientos- Cliente móvil

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

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 Visual Studio, 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 REST, 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 REST, 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 las transacciones.
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. 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.
6. Para optimizar la portabilidad y el despliegue, se sugiere utilizar **Docker** para contenerizar la instancia de SQL Server 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.
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.