## Backend:

Node.js (Express) y MySQL.

## Frontend:

Android.

Base de datos:

Un diagrama ER mostrando las relaciones entre las tablas.

A diagram of a computer

Description automatically generated

## Código:

Código bien estructurado y comentado, siguiendo buenas prácticas de desarrollo.

**Link a repositorio en Github:**

https://github.com/MiguelVieirar16/Prueba\_Telefonica/tree/main

## Documentación:

Un documento explicando las decisiones de uso del backend, y que decisiones implementaría para que la aplicación fuese escalable, Un documento explicando las decisiones de uso del backend, y que decisiones implementaría para que la aplicación fuese escalable

## Decisiones de Diseño y Arquitectura

1. **Arquitectura de Capas**
   * **Descripción**: La aplicación sigue una arquitectura de capas, separando la lógica de presentación, la lógica de negocio y la lógica de acceso a datos.
   * **Justificación**: Esta separación facilita el mantenimiento y la escalabilidad de la aplicación, permitiendo que cada capa evolucione independientemente.
2. **Uso de Retrofit para la Comunicación con el Backend**
   * **Descripción**: Retrofit se utiliza como cliente HTTP para realizar las llamadas a la API del backend.
   * **Justificación**: Retrofit simplifica la implementación de las llamadas a la API, proporcionando una forma eficiente y fácil de manejar las solicitudes y respuestas HTTP. Además, su integración con Gson facilita la serialización y deserialización de objetos JSON.
3. **Modelo-Vista-Controlador (MVC)**
   * **Descripción**: La aplicación sigue el patrón de diseño MVC, donde las actividades actúan como controladores, las vistas son los layouts XML y los modelos son las clases de datos.
   * **Justificación**: Este patrón organiza el código de manera que la lógica de negocio y la interfaz de usuario estén claramente separadas, mejorando la mantenibilidad y la escalabilidad.
4. **Uso de Adaptadores para Listas y Spinners**
   * **Descripción**: Se utilizan adaptadores personalizados para manejar la presentación de listas y spinners.
   * **Justificación**: Los adaptadores permiten una mayor flexibilidad y control sobre cómo se muestran los datos en la interfaz de usuario, mejorando la experiencia del usuario.

## Decisiones de Uso del Backend

1. **Validación de Usuario y Verificación de Código**
   * **Descripción**: El backend maneja la validación del usuario y la verificación del código de autenticación.
   * **Justificación**: Centralizar la lógica de validación y verificación en el backend mejora la seguridad y permite una gestión más eficiente de los datos de los usuarios.
2. **Uso de MySQL para el Almacenamiento de Datos**
   * **Descripción**: MySQL se utiliza como base de datos para almacenar la información de los usuarios, los detalles de los teléfonos y los códigos de verificación.
   * **Justificación**: MySQL es una base de datos relacional robusta y escalable que proporciona integridad de datos y soporte para transacciones, lo cual es crucial para la gestión de datos sensibles.
3. **Envío de Correos Electrónicos para la Verificación**
   * **Descripción**: El backend se encarga de enviar correos electrónicos con los códigos de verificación a los usuarios.
   * **Justificación**: Delegar el envío de correos electrónicos al backend asegura que los códigos de verificación se generen y envíen de manera segura y eficiente.

## Decisiones para la Escalabilidad de la Aplicación

1. **Implementación de Microservicios**
   * **Descripción**: Dividir el backend en microservicios independientes que manejen diferentes aspectos de la aplicación, como la autenticación, la gestión de usuarios y el envío de correos electrónicos.
   * **Justificación**: Los microservicios permiten escalar cada componente de manera independiente, mejorando la flexibilidad y la capacidad de respuesta del sistema.
2. **Uso de Caché**
   * **Descripción**: Implementar un sistema de caché para almacenar temporalmente los datos que se consultan con frecuencia.
   * **Justificación**: El uso de caché reduce la carga en la base de datos y mejora el rendimiento de la aplicación, proporcionando tiempos de respuesta más rápidos.
3. **Balanceo de Carga**
   * **Descripción**: Utilizar balanceadores de carga para distribuir el tráfico entre múltiples instancias del servidor backend.
   * **Justificación**: El balanceo de carga asegura que ninguna instancia del servidor se sobrecargue, mejorando la disponibilidad y la capacidad de manejo de tráfico de la aplicación.
4. **Escalado Horizontal**
   * **Descripción**: Añadir más instancias de servidores backend y bases de datos a medida que aumenta la demanda.
   * **Justificación**: El escalado horizontal permite manejar un mayor volumen de usuarios y solicitudes sin comprometer el rendimiento de la aplicación.
5. **Monitoreo y Logging**
   * **Descripción**: Implementar herramientas de monitoreo y logging para rastrear el rendimiento y detectar problemas en tiempo real.
   * **Justificación**: El monitoreo y logging proactivos permiten identificar y resolver problemas rápidamente, asegurando la estabilidad y el rendimiento continuo de la aplicación.

## Optimización y Escalabilidad Adicionales

1. **Optimización de Consultas SQL**
   * **Descripción**: Revisar y optimizar las consultas SQL para asegurar que sean eficientes y rápidas.
   * **Justificación**: Consultas SQL optimizadas reducen el tiempo de respuesta y la carga en la base de datos, mejorando el rendimiento general de la aplicación.
2. **Indexación de Base de Datos**
   * **Descripción**: Implementar índices en las tablas de la base de datos para acelerar las consultas.
   * **Justificación**: La indexación mejora significativamente el tiempo de búsqueda y recuperación de datos, especialmente en tablas grandes.
3. **Uso de CDN (Content Delivery Network)**
   * **Descripción**: Utilizar una CDN para distribuir el contenido estático de la aplicación, como imágenes y archivos JavaScript.
   * **Justificación**: Una CDN reduce la latencia y mejora la velocidad de carga al servir el contenido desde servidores geográficamente más cercanos a los usuarios.
4. **Compresión de Datos**
   * **Descripción**: Implementar compresión de datos para las respuestas de la API y los recursos estáticos.
   * **Justificación**: La compresión reduce el tamaño de los datos transferidos, mejorando los tiempos de carga y reduciendo el uso de ancho de banda.
5. **Optimización de la Interfaz de Usuario**
   * **Descripción**: Mejorar la eficiencia de la interfaz de usuario mediante la reducción de la complejidad de las vistas y el uso de técnicas de carga diferida (lazy loading).
   * **Justificación**: Una interfaz de usuario optimizada mejora la experiencia del usuario y reduce la carga en el dispositivo del usuario.
6. **Implementación de Pruebas Automatizadas**
   * **Descripción**: Desarrollar pruebas automatizadas para asegurar la calidad del código y la funcionalidad de la aplicación.
   * **Justificación**: Las pruebas automatizadas permiten detectar y corregir errores rápidamente, asegurando que la aplicación funcione correctamente a medida que se escala.

# Base de Datos

## Estructura de la Base de Datos

La base de datos hecha en MySQL usuarios\_saldos\_telefonica contiene tres tablas principales:

1. **usuarios:**
   * id: Identificador único del usuario.
   * nombre: Nombre del usuario.
   * mail: Correo electrónico del usuario.
   * numero\_movil: Número de teléfono del usuario.
   * documento\_identidad: Documento de identidad del usuario.

A screenshot of a computer

Description automatically generated

1. **cupos:**
   * id: Identificador único del cupo.
   * numero\_movil: Número de teléfono asociado al cupo.
   * saldo: Saldo del cupo.
   * datos: Datos restantes del cupo.
   * plataforma: Plataforma del cupo.
   * documento\_identidad: Documento de identidad del usuario asociado.
   * plan: Plan del cupo.

A screenshot of a computer

Description automatically generated

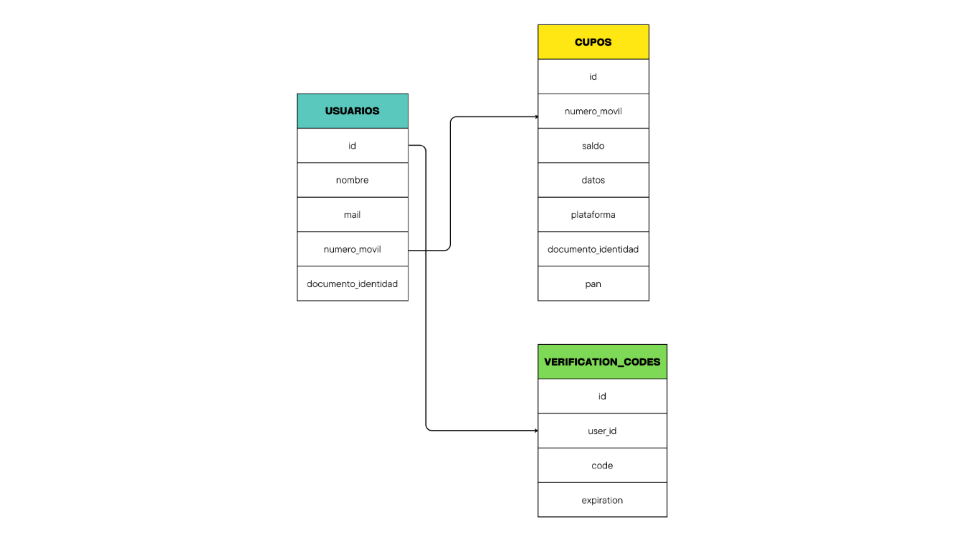
1. **verification\_codes:**
   * id: Identificador único del código de verificación.
   * user\_id: Identificador del usuario asociado.
   * code: Código de verificación.
   * expiration: Fecha y hora de expiración del código.

A screenshot of a computer

Description automatically generated

## Relaciones

* La tabla cupos está relacionada con la tabla usuarios a través del campo numero\_movil.
* La tabla verification\_codes está relacionada con la tabla usuarios a través del campo user\_id.



# Documentación del Backend

El backend está desarrollado en Node.js y utiliza una base de datos MySQL para almacenar la información de los usuarios, sus cupos y los códigos de verificación.

## Estructura de Archivos

**1. config/db.js**

Este archivo configura la conexión a la base de datos MySQL utilizando Sequelize.

* **Funcionalidad**:
  + Importa Sequelize.
  + Configura la conexión a la base de datos usuarios\_saldos\_telefonica con las credenciales proporcionadas.
  + Autentica la conexión y maneja posibles errores.
  + Exporta la instancia de Sequelize para ser utilizada en otros archivos.

**2. config/mailer.js**

Este archivo configura el servicio de envío de correos electrónicos utilizando Nodemailer.

* **Funcionalidad**:
  + Importa Nodemailer.
  + Configura el transporte SMTP con las credenciales del correo electrónico.
  + Define la función sendCode para enviar un correo electrónico con el código de verificación.
  + Exporta la función sendCode para ser utilizada en otros archivos.

**3. controllers/userController.js**

Este archivo contiene las funciones que manejan las solicitudes relacionadas con los usuarios.

* **Funcionalidad**:
  + initializeConnection: esta función verifica si la conexión a la base de datos ya existe. Si no existe, crea una nueva conexión a la base de datos usuarios\_saldos\_telefonica usando las credenciales proporcionadas.
  + saveCodeToMySQL: primero, asegura que la conexión a la base de datos esté inicializada llamando a initializeConnection. Luego, calcula una fecha de expiración para el código de verificación (5 minutos a partir del momento actual), elimina cualquier código de verificación anterior asociado con el userId proporcionado. Finalmente, inserta el nuevo código de verificación en la tabla verification\_codes junto con el userId y la fecha de expiración
  + getCodeFromMySQL: asegura que la conexión a la base de datos esté inicializada llamando a initializeConnection. Selecciona el código de verificación más reciente que no haya expirado para el userId proporcionado. devuelve el código si existe, o null si no hay un código válido.
  + validateUser: Valida la identidad del usuario y envía un código de verificación por correo electrónico.
  + verifyCode: Verifica el código de verificación ingresado por el usuario.
  + getUserCupos: Obtiene los cupos asociados al usuario.
  + getUserDetails: Obtiene los detalles del usuario, incluyendo sus teléfonos y saldo.

**4. models/userModel.js**

Este archivo define el modelo de datos para los usuarios utilizando Sequelize.

* **Funcionalidad**:
  + Define la estructura de la tabla usuarios con los campos id, nombre, mail, numero\_movil y documento\_identidad.

**5. models/cupoModel.js**

Este archivo define el modelo de datos para los cupos utilizando Sequelize.

* **Funcionalidad**:
  + Define la estructura de la tabla cupos con los campos id, numero\_movil, saldo, datos, plataforma, documento\_identidad y plan.
  + Establece la relación con la tabla usuarios a través del campo numero\_movil.

**7. routes/userRoutes.js**

Este archivo define las rutas de la API relacionadas con los usuarios.

* **Funcionalidad**:
  + Define las rutas para validar usuarios, verificar códigos, obtener detalles de usuarios y obtener detalles de teléfonos.
  + Asocia cada ruta con su respectivo controlador.

**8. app.js**

Este archivo es el punto de entrada principal de la aplicación.

* **Funcionalidad**:
  + Importa y configura Express.
  + Configura CORS para permitir solicitudes desde cualquier origen.
  + Configura el middleware para parsear JSON.
  + Define las rutas de la API.
  + Conecta a la base de datos y maneja posibles errores de conexión.
  + Inicia el servidor en el puerto especificado.

# Frontend

El frontend está compuesto en su totalidad por un código hecho en Java

## Archivos del Proyecto

1. **AccesoDirectoAdapter.java**
   * **Función**: Define un adaptador para un RecyclerView que muestra una lista de accesos directos en la interfaz de usuario.
2. **PhoneDetailsAdapter.java**
   * **Función**: Define un adaptador para un Spinner que muestra una lista de detalles de teléfonos.
3. **AccesoDirecto.java**
   * **Función**: Define una clase modelo para los accesos directos, incluyendo atributos como título, subtítulo, ícono y URL.
4. **PhoneDetails.java**
   * **Función**: Define una clase modelo para los detalles de los teléfonos, incluyendo atributos como número de móvil, plataforma
5. **UserDetailsResponse.java**
   * **Función**: Define una clase modelo para la respuesta de los detalles del usuario, incluyendo el nombre del usuario y una lista de detalles de teléfonos.
6. **UserValidationRequest.java**
   * **Función**: Define una clase modelo para la solicitud de validación del usuario, incluyendo el documento de identidad y el número de móvil.
7. **ValidationResponse.java**
   * **Función**: Define una clase modelo para la respuesta de validación, indicando si la validación fue exitosa para poder corroborar que el número de teléfono y el número de identificación concuerda.
8. **VerifyRequest.java**
   * **Función**: Define una clase modelo para la solicitud de verificación de código, incluyendo el documento de identidad y el código de verificación, indicando si el código enviado concuerda con el número de identificación en la DB
9. **VerifyResponse.java**
   * **Función**: Define una clase modelo para la respuesta de verificación, indicando si la verificación fue exitosa y proporcionando un mensaje.
10. **ValidarCampos.java**
    * **Función**: Define una clase para validar los campos de entrada, como el número de teléfono, el documento de identidad y el código de verificación.
    * **Métodos Principales**:
      + ValidarTelefono(String telefono): Valida que el número de teléfono tenga la longitud adecuada.
      + ValidarID\_CI(String cedula): Valida que la cédula tenga la longitud adecuada.
      + Validar\_Code(String code): Valida que el código tenga la longitud adecuada.
11. **ApiClient.java**
    * **Función**: Define una clase para configurar y obtener una instancia de Retrofit, que se utiliza para realizar llamadas a la API.
    * **Métodos Principales**:
      + getApiClient(): Devuelve una instancia de Retrofit configurada con la URL base y el convertidor de Gson.
12. **ApiService.java**
    * **Función**: Define una interfaz para las llamadas a la API, incluyendo métodos para obtener detalles del usuario, validar al usuario y verificar el código.
    * **Métodos Principales**:
      + getUserDetails(String documento\_identidad): Método GET para obtener los detalles del usuario.
      + validateUser(UserValidationRequest request): Método POST para validar al usuario.
      + verifyCode(VerifyRequest request): Método POST para verificar el código.
13. **MainActivity.java**
    * **Función**: Define la actividad principal de la aplicación. Incluye métodos para validar los campos de entrada y realizar la validación del usuario mediante una llamada a la API.
14. **VerificationActivity.java**
    * **Función**: Define una actividad para la verificación del código de usuario. Incluye métodos para validar el código de entrada y realizar la verificación mediante una llamada a la API.
15. **UserDetailsActivity.java**
    * **Función**: Define una actividad para mostrar los detalles del usuario, incluyendo un saludo, detalles del teléfono y accesos directos. Incluye métodos para obtener los detalles del usuario y actualizar la interfaz de usuario.