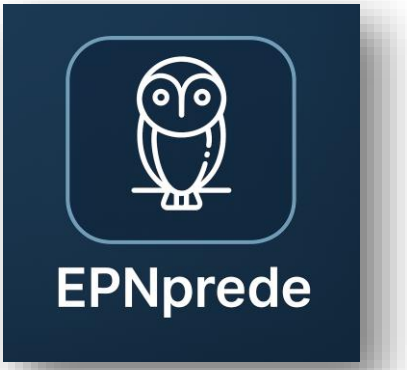
PROYECTO FINAL DE PROGRAMACIÓN ORIENTADA A OBJETOS

1. Título del Proyecto

EPNprende: Sistema de Gestión de Publicaciones y Productos para Emprendedores Universitarios



2. Objetivos

Objetivo General

Desarrollar una aplicación de escritorio en Java, utilizando principios de Programación Orientada a Objetos (POO), que permita a estudiantes emprendedores de la EPN gestionar la publicación de productos, contactos, reportes, ofertas y comentarios, con conexión a una base de datos en la nube (PostgreSQL).

Objetivos Específicos

- Aplicar herencia, encapsulamiento, polimorfismo y abstracción en el desarrollo de clases Java.
- Implementar formularios funcionales con JavaFX para publicar productos, ver detalles, reportar y buscar.
- Conectar el sistema con una base de datos PostgreSQL desplegada en la nube.
- Estructurar el código en paquetes modulares (modelo, controlador, vista, servicios, conexión).
- Garantizar una interfaz funcional y validaciones visuales en cada operación CRUD.
- Generar reportes automáticos y manejar favoritos, clics y ofertas en los productos publicados.

3. Justificación

EPNprende responde a la necesidad de una plataforma digital que promueva los emprendimientos estudiantiles dentro de la EPN, permitiendo a los usuarios publicar sus productos, gestionar sus perfiles y establecer canales de contacto. Este proyecto sirve como práctica integral de los conceptos fundamentales de la POO en un entorno realista, complementando también el desarrollo de la base de datos PostgreSQL con múltiples tablas, claves foráneas y procedimientos almacenados.

4. Alcance y Requerimientos – roles

Alcance del Proyecto

- Conexión con Firebase para la autenticación de los usuarios en la app.
- Los usuarios pueden registrase dentro de la aplicación e iniciar sesión con sus credenciales.
- Permite visualizar productos con imágenes, precio y nombre.
- Visualización de las diferentes ventanas como productos, categorías, ofertas, y favoritos dentro de la sección de comprar.
- Botón de salir que cierra por completo el aplicativo.

Requerimientos

Funcionales

- Registro y gestión de productos por el usuario.
- Visualización de las categorías de los productos disponibles.
- Visualización de productos.
- Navegación entre las diferentes secciones del aplicativo.
- Carga de la información de los usuarios en PostgreSQL.

No Funcionales

- Diseño sencillo y claro usando Java FX.
- Validaciones de entrada visuales y lógicas.
- Código modular y estructurado.
- Compatible con Firebase como motor principal de autenticación de usuarios.

Tecnológicos

- Java JDK 17+ (IDE IntelliJ IDEA)
- Java FX (
- CSS para diseño limpio y estilos modernos.

- PostgreSQL (en la nube).

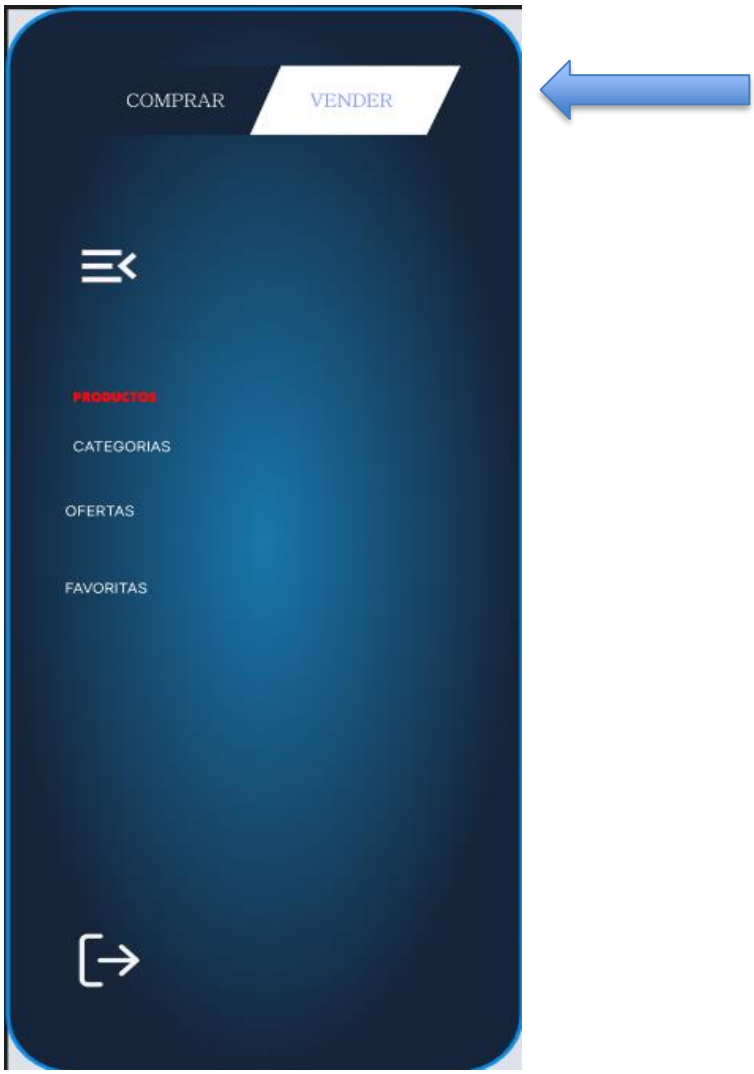
- Neon Wireless
- JDBC Driver PostgreSQL.
- GitHub (control de versiones).

6. Diseño de Interfaz (Mockup)

Formulario Principal de inicio de sesión

Formulario Principal de registro de usuario

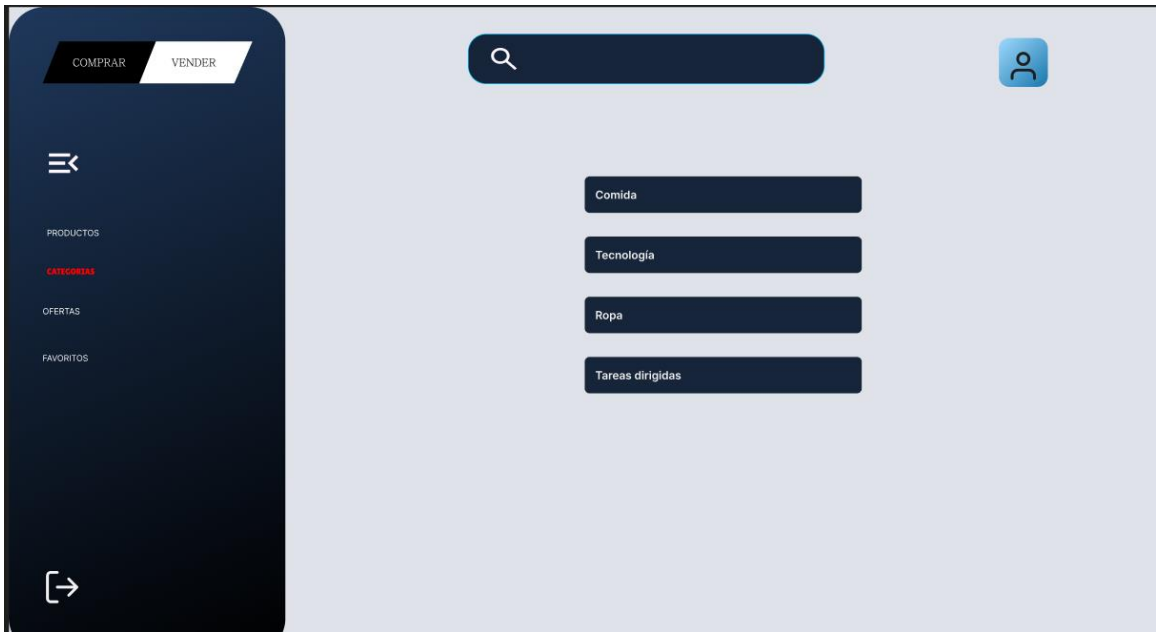
Panel dinámico para cambiar de sección



Sección de compra:

Opción productos de la sección de compra

Opción categorías de la sección de compra

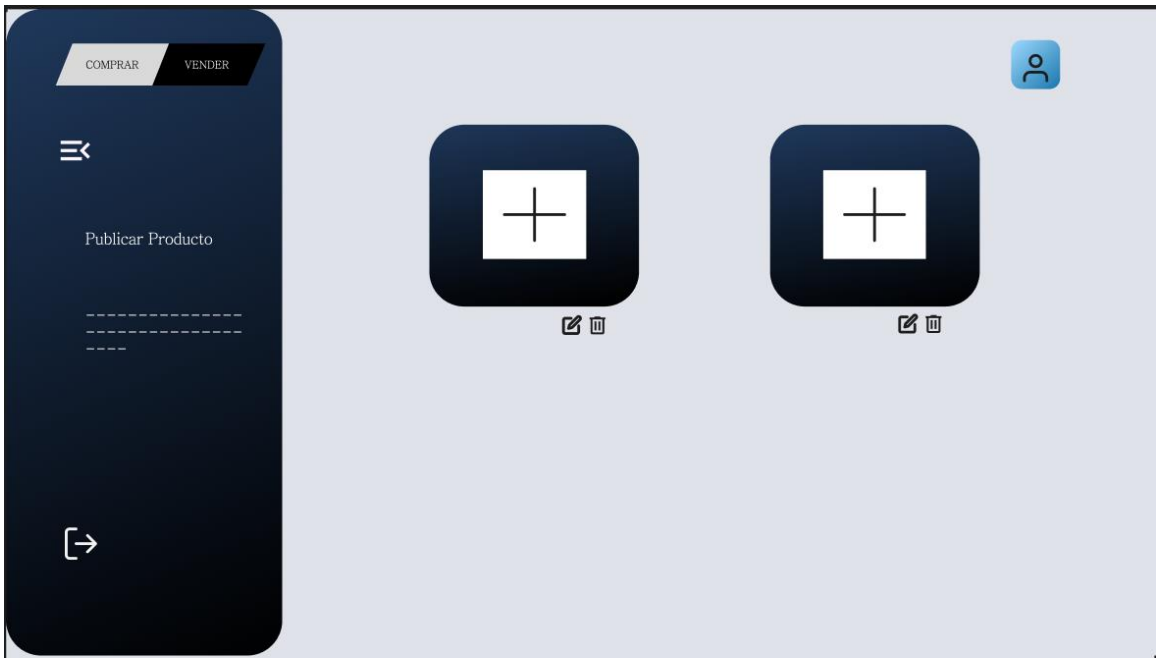


Opción ofertas de la sección de compra

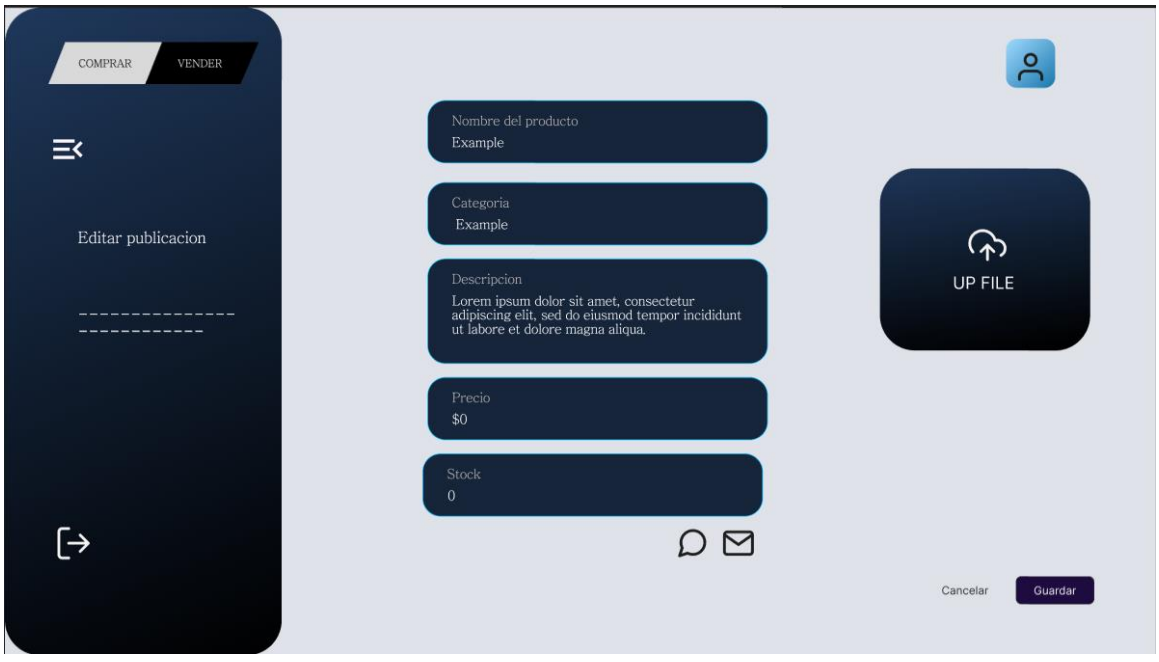
Opción favoritos de la sección de compra

Sección de venta:

Opción de publicar producto



Opción editar producto



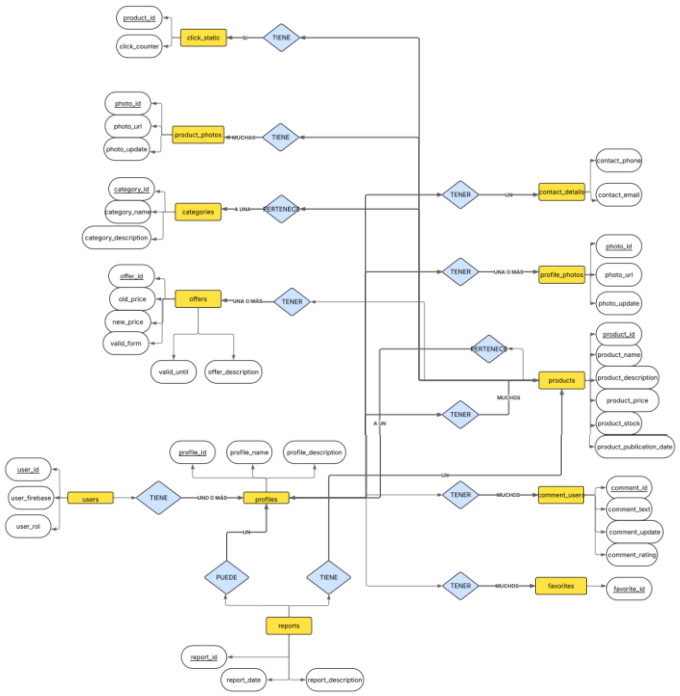
7. Modelado de Base de Datos

Tablas Principales:

- users (conectado con Firebase, rol: admin o user)
- profiles (datos públicos del usuario)

- products (categoría, perfil, nombre, disponibilidad, precio, foto)
- categories (catálogo de tipos de productos)
- comments, favorites, reports, offers, photos, notifications (en desarrollo)
- Sistema normalizado a Tercera Forma Normal (3FN)
- Integridad referencial completa y restricciones CHECK, NOT NULL, etc.

Diagrama ER



8. Organización del Código

Carpetas:

- **src** → Carpeta principal que contiene todos los recursos del programa.

- **main** → Recursos para el arranque del programa.

  java:

  - application → Clase App y Laucher para arranque del sistema.

  - controllers → Controladores de cada una de las vistas (FXML) que supervisan los eventos que pasan.

- **interfaces** → Contiene la interfaz con los métodos onShow(): Se ejecuta cuando la componente se muestra o se hace visible.

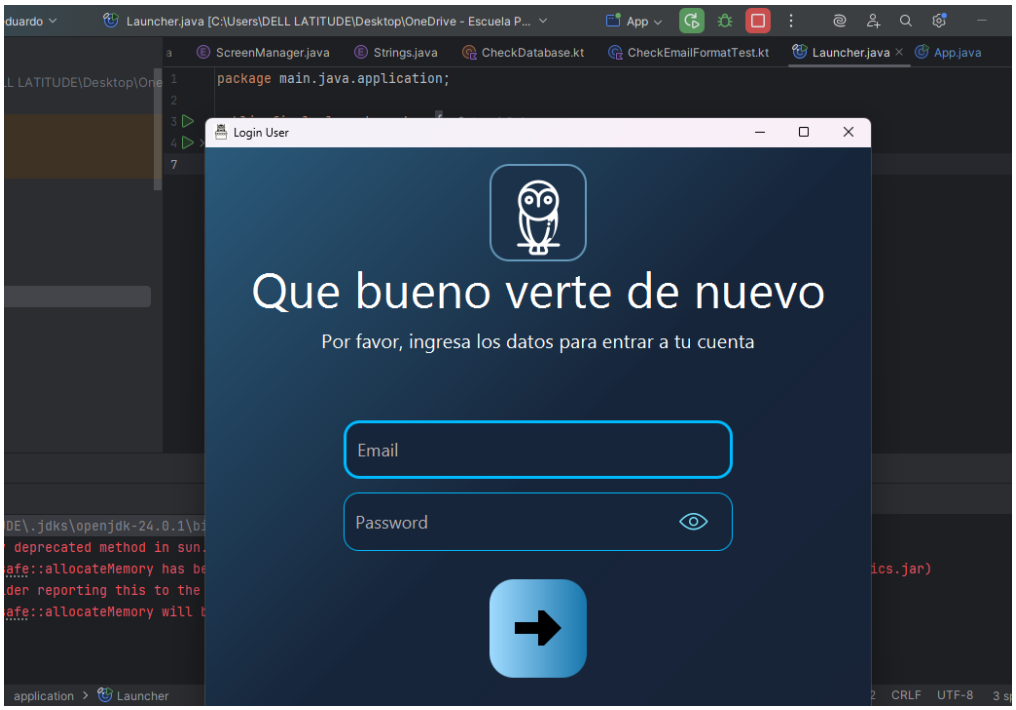
onHide(): Se ejecuta cuando la vista se oculta o deja de ser visible.

- **utils:** Contiene código reutilizable guardadas en clases enum(constantes), como mensajes de alertas, string para diseño y rutas, además una clase especifica que administra las vistas.

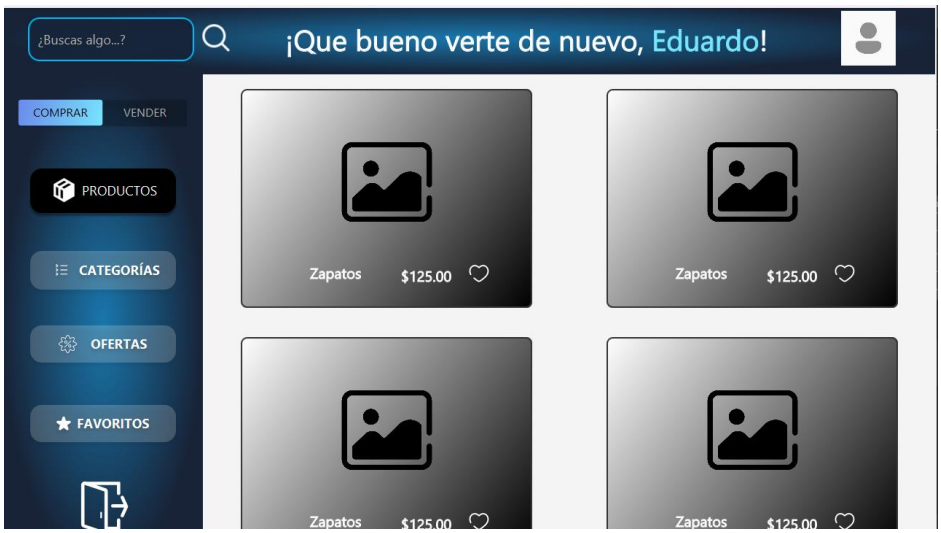
- Kotlin → Maneja todo lo relacionado con la lógica del negocio, conexiones a las bases de datos Firebase y PostgreSQL
- conexion → Conexión a BD.
- modelo → Clases Participante, Categoria.
- controlador → Lógica CRUD.
- vista → Formularios Swing.

9. Ejecución Completa del Proyecto

- Inicio del sistema desde la clase Main.



- Menú principal de navegación.



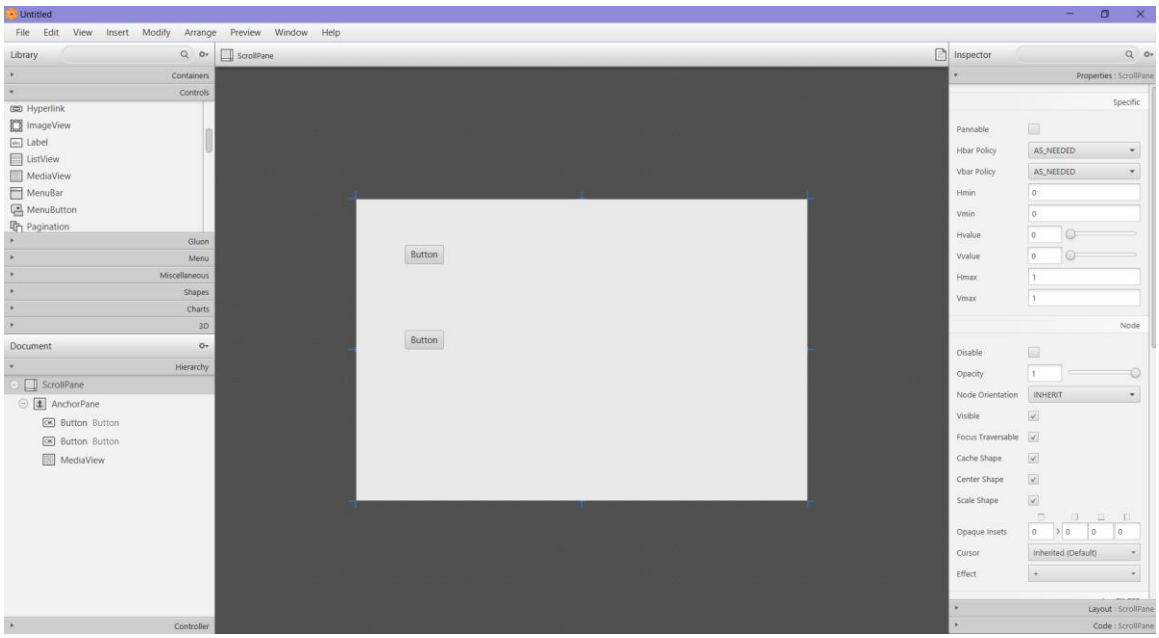
- CRUD completo desde formularios.
- Conexión exitosa con MySQL en la nube.
- Ejecución verificada en máquina local con internet.

10. Explicación del Desarrollo

Para el desarrollo se implementó la arquitectura MVC (modelo, vista, controlador).

En ese caso se cómo la herramienta utilizada para el diseño de interfaces graficas fue Java FX, todas las ventanas y sus respectivos componentes fueron diseñados con la herramienta

SceneBuilder que permite el arrastre de botones, labels, nanvar entre muchos más facilitando el proceso.

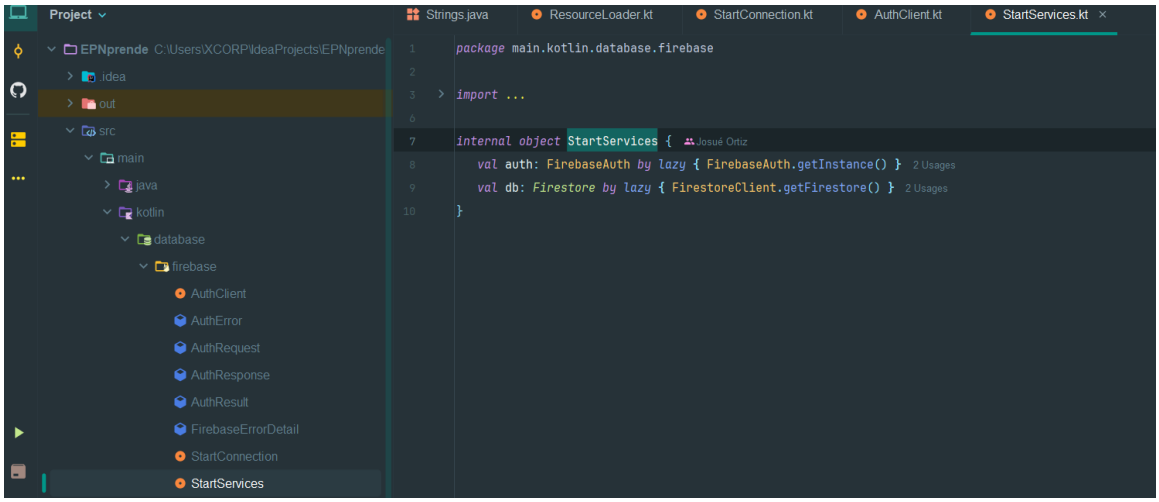


Cada clase cuenta con su propia responsabilidad:
- Modelo: Representa entidades con atributos privados y públicos según el caso, métodos get/set, clases relacionadas a cada funcionalidad, se hizo la implementación de verificaciones y control de las acciones dentro del sistema.

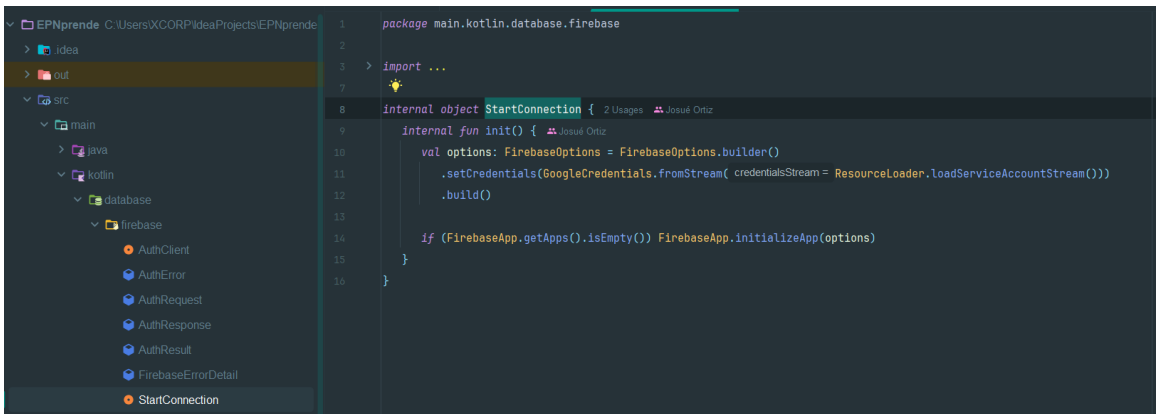
- Controlador: Accede
- Vista: Interfaces gráficas y eventos asociados cada uno con su propio controlador.
- Conexión:

Firebase

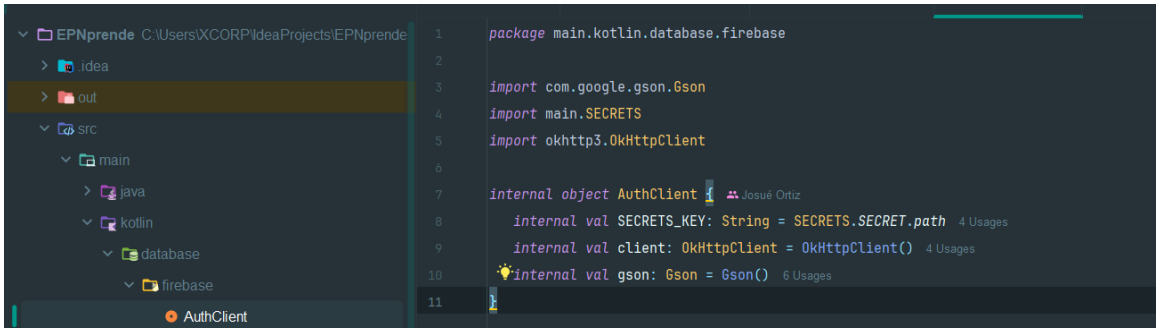
StartServices perimte inicializar de forma los servicios de Firebase: **Auth** (autenticación) y **Firestore** (base de datos), centralizando su acceso en un único punto.



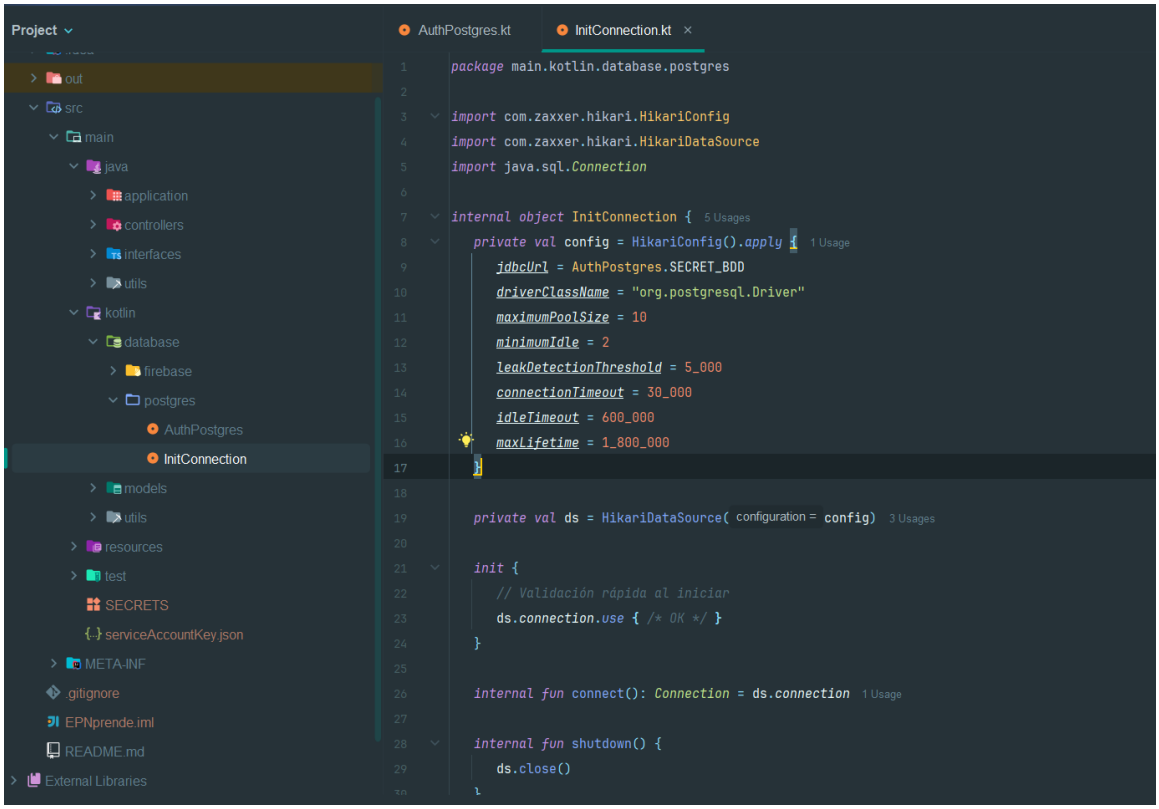
Clase StarConnection.java para enlace nube con parámetros externos.



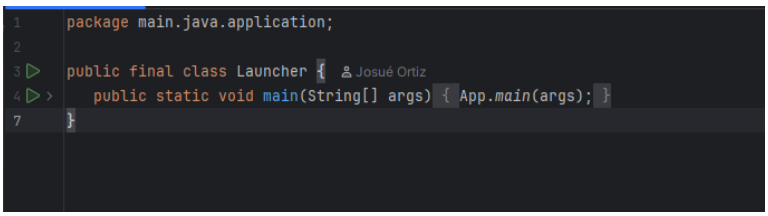
AuthClient permite la carga de las claves de Firebase, genera la petición al html y guarda la información en formato JSON.



PostgreSQL



- Main: Inicializa el sistema, función de arranque mediante la clase LAUNCHER y App es la clase por defecto para iniciar cualquier programa en Java.



```java
// Inicia la aplicación de JavaFX.
public static void main(String[] args) { launch(args); }

@Override
public void start(Stage stage) throws IOException {
    // Se inicia Firebase
    StartConnection.INSTANCE.init$EPNprende();

    // Creación de la interfaz base desde el FXML.
    FXMLLoader loader = new FXMLLoader(ResourceLoader.INSTANCE.getResource$EPNprende(Paths.LAUNCHER.getPath()));

    // Se carga la interfaz base previamente hecha en un layout raiz.
    StackPane root = loader.load();

    // Se crea la escena con el layout raiz que contiene la interfaz cargada.
    Scene scene = new Scene(root);

    // La escena es asignada a la ventana para mostrarse ya en pantalla.
    stage.setScene(scene);

    // Mediante el controlador se invoca a la pantalla de login para que esta se muestre en la escena.
    RootController.showLogin(stage);

    // Se establece el icono de la ventana
    Image logo = new Image(ResourceLoader.INSTANCE.getResource$EPNprende(Paths.ICON_LOGO.getPath()).openStream());
    stage.getIcons().add(logo);
```

Cada controlador está asociado a una Interfaces gráficas y eventos asociados cada uno con su propio controlador

```java
package main.java.controllers;                                                    9

import ...

public class LoginScreenController implements ViewLifecycle {  1 usage  Josué Ortiz

    // Aquí almacenamos el último estado de la verificación:
    private final BooleanProperty emailVerified = new SimpleBooleanProperty( b: false);  4 usages

    private final BooleanProperty passwordVisible = new SimpleBooleanProperty( b: false);  7 usages

    private PauseTransition pause;  3 usages

    @FXML
    private Button btn_hide_password;

    @FXML
    private Button btn_view_password;

    @FXML
    private Label lbl_message;

    @FXML
    private StackPane loginPane;

    @FXML
    private PasswordField pass_field;
```

Herramienta Recortes

Captura de pantalla copiada en el portapapeles
Guardado automáticamente en la carpeta de capturas de pantalla.

```java
public class RootController {  23 usages  Josué Ortiz

    @FXML
    private StackPane root;

    // Métodos delegados
    public static void showLogin(Stage stage) {  5 usages  Josué Ortiz
        ScreenManager.show(Paths.LOGIN_SCREEN.getPath());
        stage.setTitle(Strings.LOGIN_TITLE.getText());
        stage.setWidth(700);
        stage.setHeight(700);
        stage.centerOnScreen();
    }

    static void showSign(Stage stage) {  1 usage  Josué Ortiz
        ScreenManager.show(Paths.SIGN_SCREEN.getPath());
        stage.setTitle(Strings.SIGN_TITLE.getText());
        stage.setWidth(700);
        stage.setHeight(700);
        stage.centerOnScreen();
    }

    public static void showDashboard(Stage stage) {  6 usages  Josué Ortiz
        ScreenManager.show(Paths.DASHBOARD_SCREEN.getPath());
        stage.setTitle(Strings.DASHBOARD_TITLE.getText());
        stage.setWidth(1300);
        stage.setHeight(800);
        stage.centerOnScreen();
```

```java
// 1. Métodos Estáticos
private static byte checkPassword(String password, String password_confirmation) {  1 usage  Josué O
    boolean isFormatPasswordCorrect = CheckPassword.INSTANCE.checkPasswordFormat$EPNprende(passwor
    boolean isLengthPasswordCorrect = CheckPassword.INSTANCE.checkPasswordLength$EPNprende(passwor
    boolean isSimilarPassword = CheckPassword.INSTANCE.checkPasswordSimilarity$EPNprende(password,

    if (!isSimilarPassword) return 1;
    else if (!isFormatPasswordCorrect) return 2;
    else if (!isLengthPasswordCorrect) return 3;
    else return 0;
}


// 2. Ciclo de vida
@FXML  Josué Ortiz
public void initialize() {
    txt_field_email.textProperty().addListener(( ObservableValue<extends String> _, String _, String _)
        lbl_message_login.setText(Strings.EMPTY_TEXT.getText());
        lbl_message_password.setText(Strings.EMPTY_TEXT.getText());
        checkEmail();
    });

    txt_field_name.textProperty().addListener(( ObservableValue<extends String> _, String _, String _) -
        lbl_message_login.setText(Strings.EMPTY_TEXT.getText());
        lbl_message_password.setText(Strings.EMPTY_TEXT.getText());
    });
        pass_field.textProperty().addListener(( ObservableValue<extends String> _, String _, String _) -> {
            lbl_message_login.setText(Strings.EMPTY_TEXT.getText());
            lbl_message_password.setText(Strings.EMPTY_TEXT.getText());
        });

        confirm_field.textProperty().addListener(( ObservableValue<extends String> _, String _, String _) ->
            lbl_message_login.setText(Strings.EMPTY_TEXT.getText());
            lbl_message_password.setText(Strings.EMPTY_TEXT.getText());
        });

        // Vinculación bidireccional entre campos
        pass_field.textProperty().bindBidirectional(txt_view_password.textProperty());

        // Bind de visibilidad
        txt_view_password.visibleProperty().bind(passwordVisible);
        pass_field.visibleProperty().bind(passwordVisible.not());
        btn_hide_password.visibleProperty().bind(passwordVisible);
        btn_view_password.visibleProperty().bind(passwordVisible.not());
    }
```

La vistas se determinan por visibles o también ocultas

```java
package main.java.interfaces;

public interface ViewLifecycle {  8 usages  2 im
    default void onShow() {  2 usages  2 overrides
    }

    default void onHide() {  1 usage  Josué Ortiz
    }
```

Los utils son archivos que se pueden reutilizar y que la aplicación se escalable

```java
package main.java.utils;

/**
 * Rutas a los archivos FXML de la aplicación.
 */
public enum Paths {  13 usages  Josué Ortiz +1

    // ── Pantalla principal ──────────────────────────────
    LAUNCHER("/main/resources/views/RootScreen.fxml"),  1 usage

    // ── Autenticación ──────────────────────────────
    LOGIN_SCREEN("/main/resources/views/LoginScreen.fxml"),  1 usage
    SIGN_SCREEN("/main/resources/views/SignScreen.fxml"),  1 usage

    // ── Interfaz de usuario ──────────────────────────────
    DASHBOARD_SCREEN("/main/resources/views/DashboardScreen.fxml"),  1 usage
    CATEGORIES_SCREEN("/main/resources/views/CategoriesScreen.fxml"),  1 usage
    FAVORITES_SCREEN("/main/resources/views/FavoritesScreen.fxml"),  1 usage
    OFFERS_SCREEN("/main/resources/views/OffersScreen.fxml"),  1 usage

    // ── Firebase ──────────────────────────────
    KEY_FILE("main/serviceAccountKey.json"),  2 usages

    // ── Icons ──────────────────────────────
    ICON_LOGO("/main/resources/images/logo.jpg");  1 usage

    private final String path;  2 usages
```

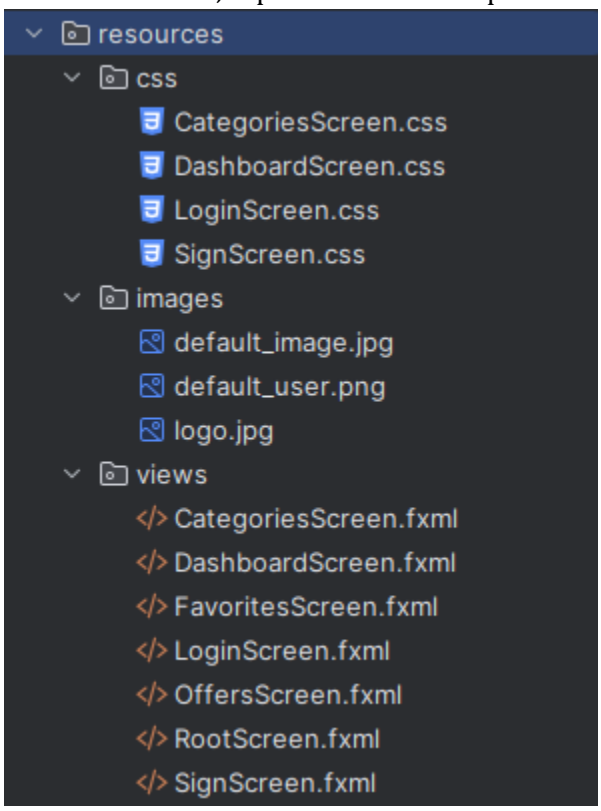
Los modelos de Login y Sing son ocupados para darle la lógica de las ventanas como validaciones, formatos y las igualdades de las contraseñas al momento de crear.

```kotlin
package main.kotlin.models.login

import main.kotlin.database.firebase.StartServices.auth

internal object CheckDatabase {  2 Usages  Josué Ortiz
    internal fun checkExistingUser(email: String): Boolean {  Josué Ortiz
        try {
            auth.getUserByEmail(email)
            return true
        } catch (_: Exception) {
            return false
        }
    }
}
```

Resources son carpetas con archivos de las imágenes, estilos css para cada vista y las vistas en formato .fxml , separadas en sus respectivas carpetas



La carpeta tests esta hecha para realizar pruebas unitarias a funciones específicas de la aplicación .

```
package main.test.models.sing                                    ⚠8 ⚠5 ✓24 ^

import ...

class CheckEmailFormatTest {  Eduardo Ganchala

    @Test  Eduardo Ganchala
    fun emailválido_con_dominio_epn() {
        val email = "juan123@epn.edu.ec"
        val result = CheckEmailFormat.checkEmailFormat(email)
        assertTrue( condition = result,  message = "Debe aceptar emails válidos con dominio @epn.edu.ec")
    }

    @Test  Eduardo Ganchala
    fun emailinválido_con_dominio_diferente() {
        val email = "juan123@gmail.com"
        val result = CheckEmailFormat.checkEmailFormat(email)
        assertFalse( condition = result,  message = "Debe rechazar emails que no sean de @epn.edu.ec")
    }

    @Test  Eduardo Ganchala
    fun emailinválido_sin_empezarporletra() {
        val email = "1juan@epn.edu.ec"
        val result = CheckEmailFormat.checkEmailFormat(email)
        assertFalse( condition = result,  message = "Debe rechazar emails que no empiecen con letra")
    }
```

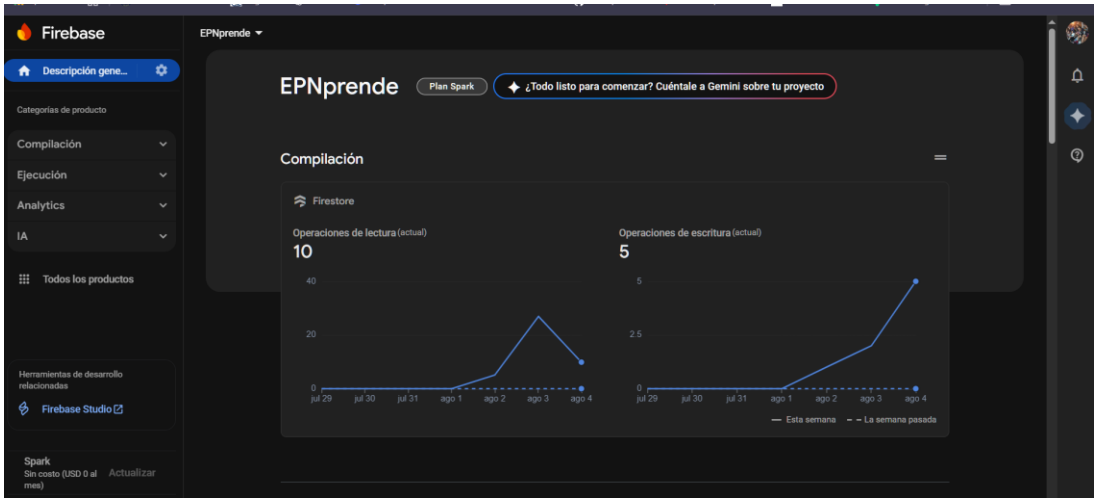
11. Repositorio GitHub y .exe(ejecutable)

https://github.com/NW08/EPNprende.git

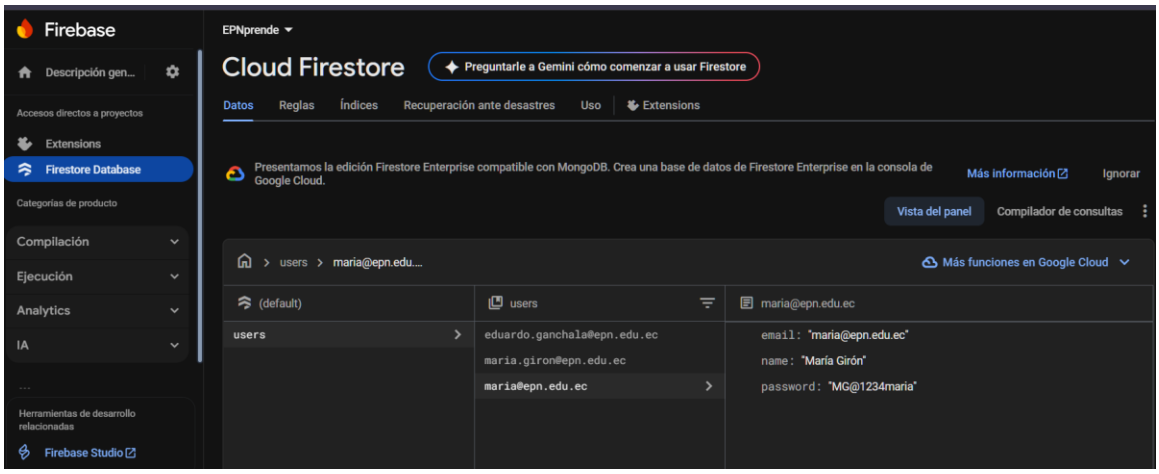
12. Funcionamiento con Base de Datos en la Nube

Se implementó una base de datos en PostgreSQL desplegada con la ayuda de la herramienta Neon Wireless para la carga de datos en línea, esta permite la subida el línea de los datos de la aplicación de escritorio desarrollada y para la carga de los datos de los usuarios autenticados se usó la herramienta de Firebase.

Panel de conexión a Firebase:

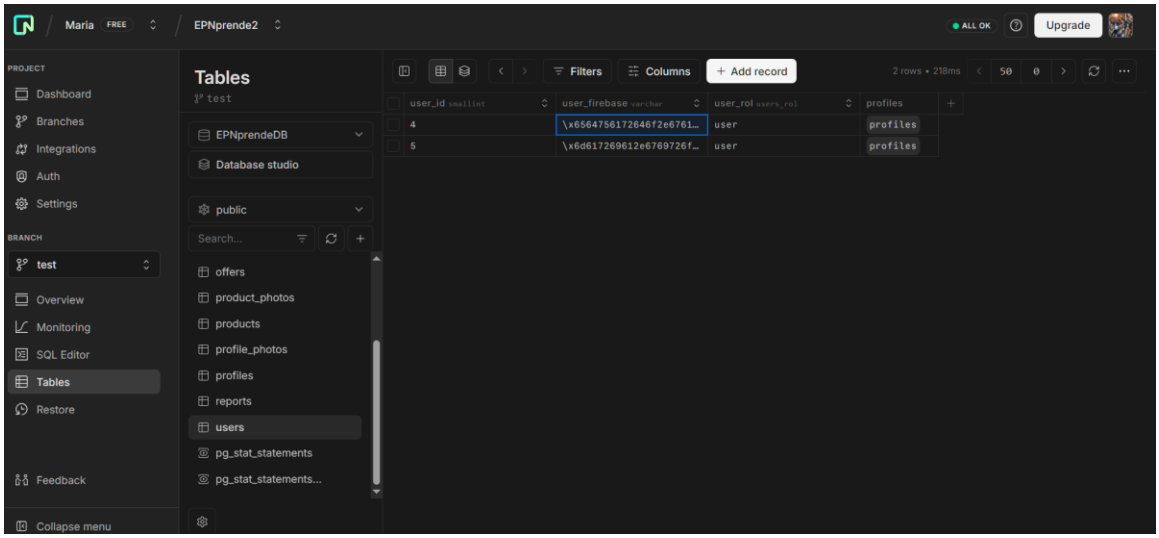


Colección de usuarios almacenados:



Panel de administración de la base de PostgreSQL en Neon Wireless:

En este se muestra que actualmente existen dos usuarios registrados y se han guardado sus respectivos id de Firebase.



13. Anexos

Enlace al repositorio: https://github.com/NW08/EPNprende.git