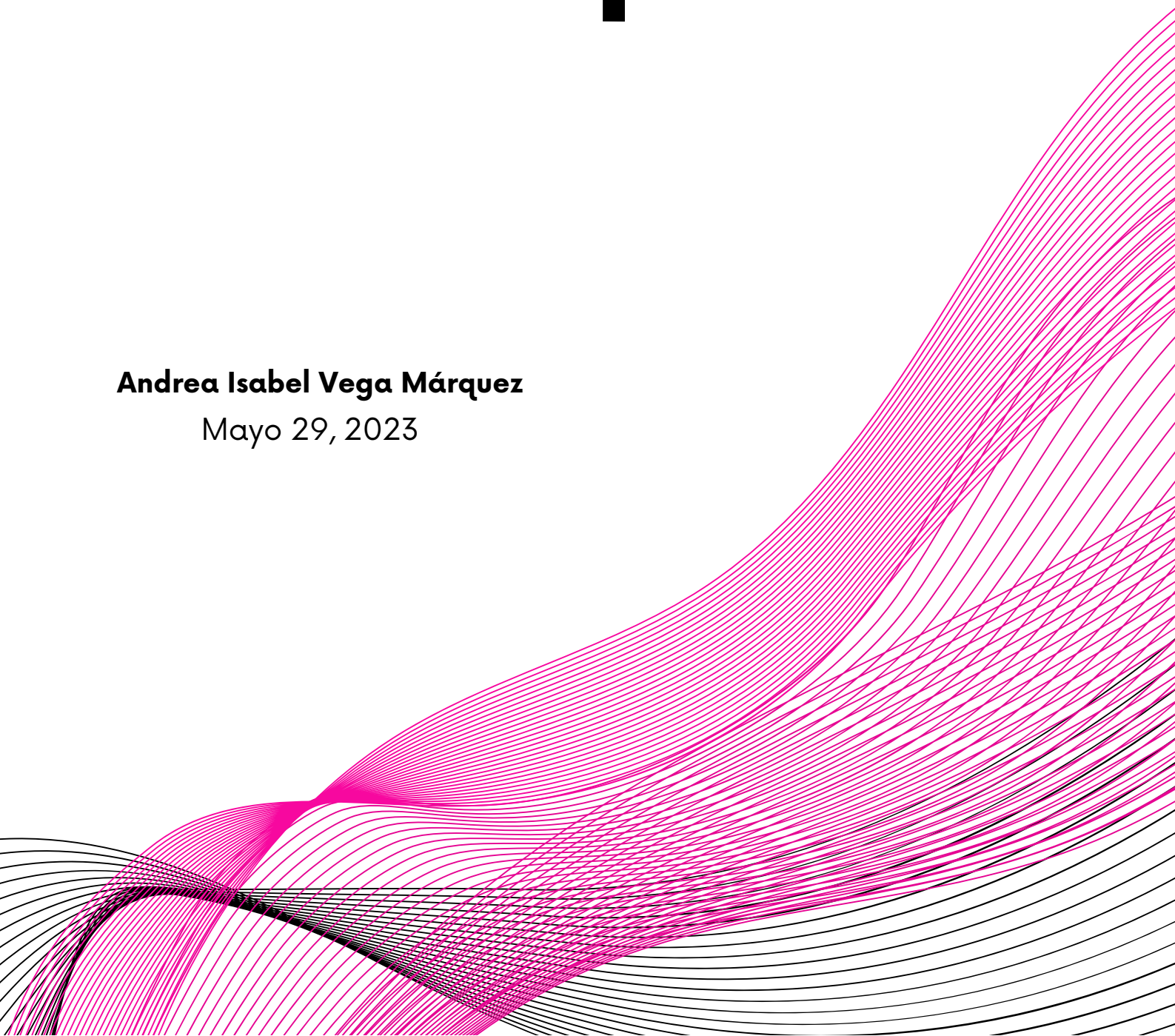
Manual del programador

CookUp

Andrea Isabel Vega Márquez

Mayo 29, 2023



ÍNDICE

INTRODUCCIÓN

¡¡Bienvenido al manual del programador de CookUp! Este manual te proporcionará información detallada sobre la estructura de la aplicación, las tecnologías utilizadas y las pautas de desarrollo para mantener y mejorar la aplicación CookUp. Sigue las instrucciones a continuación para comprender mejor el código y realizar tareas de desarrollo en CookUp.

ESTRUCTURA DE LA APLICACIÓN

La aplicación CookUp sigue una estructura de proyecto organizada y modularizada. A continuación se detallan las carpetas y archivos principales:

- /backend: Contiene los archivos relacionados con la lógica del backend de CookUp, incluyendo la configuración de Firebase y los modelos de datos.
- /flutter_flow: Contiene archivos relacionados con FlutterFlow, como el tema de la aplicación, utilidades personalizadas y archivos de internacionalización.
- /screens: Contiene las diferentes pantallas de la aplicación, como la pantalla de inicio, la pantalla de recetas y la pantalla de perfil.
- /widgets: Contiene widgets reutilizables utilizados en varias pantallas de la aplicación.

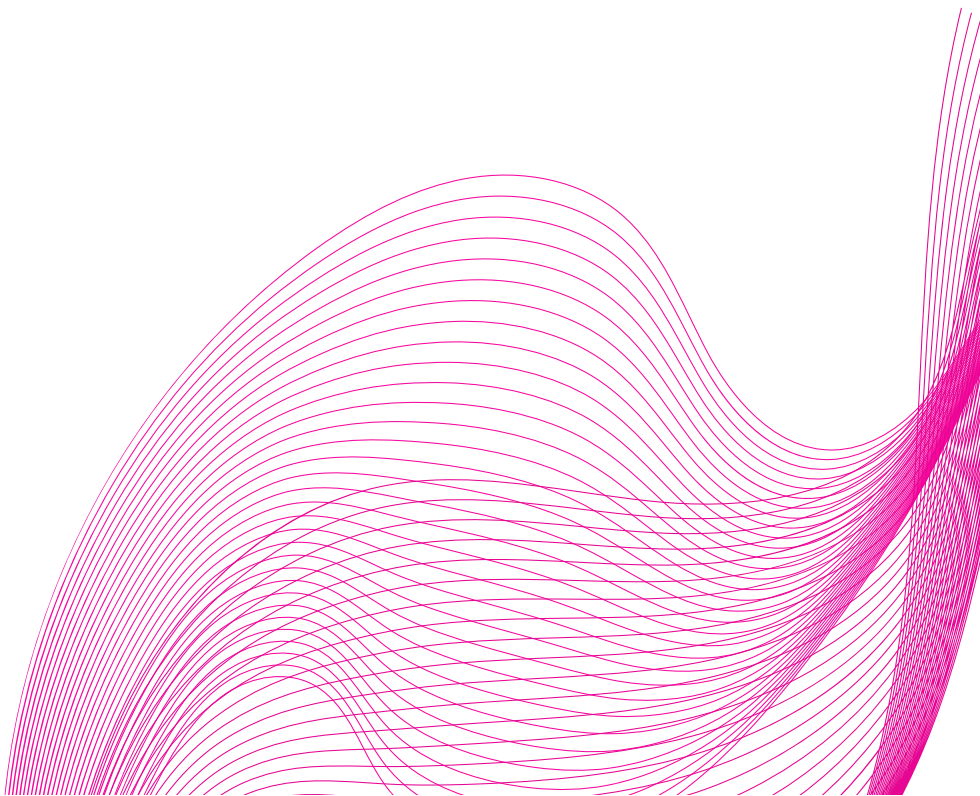
TECNOLOGÍAS UTILIZADAS

- Flutter: CookUp está desarrollado utilizando el framework de Flutter, que permite crear aplicaciones nativas para iOS y Android desde un solo código base.
- Firebase: CookUp utiliza Firebase como plataforma de backend, lo que permite almacenar y sincronizar datos, autenticar usuarios y realizar otras funciones relacionadas con la nube.
- Provider: Se utiliza el paquete de Flutter llamado Provider para gestionar el estado de la aplicación y permitir la comunicación entre diferentes componentes.

CONFIGURACIÓN DEL ENTORNO DE DESARROLLO

Antes de comenzar a desarrollar en CookUp, asegúrate de tener el entorno de desarrollo adecuado configurado:

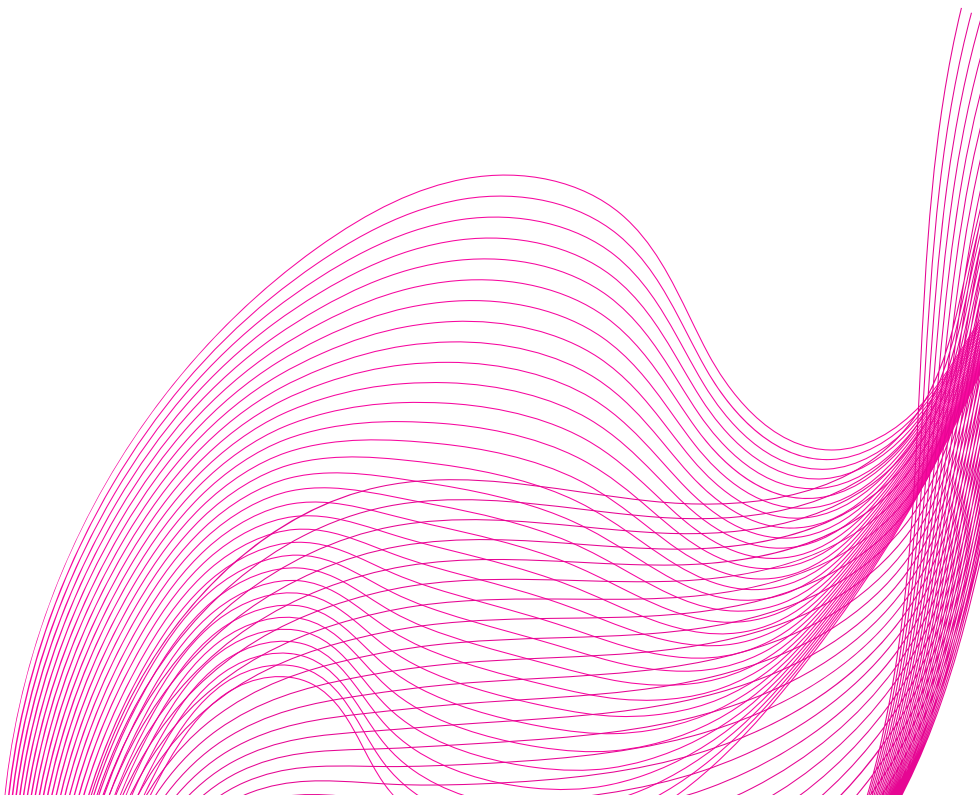
- Instala Flutter: Sigue las instrucciones en la documentación oficial de Flutter para instalar Flutter en tu sistema operativo.
- Configura el emulador o dispositivo físico: Configura un emulador de Android o iOS o conecta un dispositivo físico para probar la aplicación.
- Configura Firebase: Crea un proyecto en Firebase y configura las credenciales necesarias para que la aplicación pueda conectarse a Firebase.



ESTRUCTURA DEL CÓDIGO

El código de CookUp sigue las mejores prácticas de Flutter y está bien estructurado para facilitar el mantenimiento y la escalabilidad. Asegúrate de familiarizarte con los siguientes aspectos:

- Separación de responsabilidades: El código sigue el principio de separación de responsabilidades, dividiendo la lógica de la aplicación en diferentes clases y archivos para mejorar la legibilidad y el mantenimiento.
- Patrón de diseño MVC: CookUp sigue el patrón de diseño Modelo-Vista-Controlador (MVC), donde los modelos representan los datos, las vistas muestran la interfaz de usuario y los controladores manejan las interacciones y la lógica empresarial.
- Uso de proveedores: Se utiliza el paquete provider de Flutter para administrar el estado de la aplicación de manera eficiente y permitir la comunicación entre diferentes componentes.



MAIN.DART

```dart
import 'package:provider/provider.dart';
import 'package:flutter/gestures.dart';
import 'package:flutter/material.dart';

import 'package:flutter_localizations/flutter_localizations.dart';
import 'package:firebase_core/firebase_core.dart';
import 'auth/firebase_auth/firebase_user_provider.dart';
import 'auth/firebase_auth/auth_util.dart';

import 'backend/firebase/firebase_config.dart';
import 'flutter_flow/flutter_flow_theme.dart';
import 'flutter_flow/flutter_flow_util.dart';
import 'flutter_flow/internationalization.dart';
import 'flutter_flow/nav/nav.dart';
import 'index.dart';

void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await initFirebase();

  await FlutterFlowTheme.initialize();

  final appState = FFAppState(); // Initialize FFAppState
  await appState.initializePersistedState();

  runApp(ChangeNotifierProvider(
    create: (context) => appState,
    child: MyApp(),
  ));
}
```

```dart
class MyApp extends StatefulWidget {
  // This widget is the root of your application.
  @override
  State<MyApp> createState() => _MyAppState();

  static _MyAppState of(BuildContext context) =>
      context.findAncestorStateOfType<_MyAppState>()!;
}

class _MyAppState extends State<MyApp> {
  Locale? _locale;
  ThemeMode _themeMode = FlutterFlowTheme.themeMode;

  late Stream<BaseAuthUser> userStream;

  late AppStateNotifier _appStateNotifier;
  late GoRouter _router;

  final authUserSub = authenticatedUserStream.listen((_) {});
```

MAIN.DART

```dart
  @override
  void initState() {
    super.initState();
    _appStateNotifier = AppStateNotifier();
    _router = createRouter(_appStateNotifier);
    userStream = cookUpFirebaseUserStream()
      ..listen((user) => _appStateNotifier.update(user));
    jwtTokenStream.listen((_) {});
    Future.delayed(
      Duration(seconds: 1),
      () => _appStateNotifier.stopShowingSplashImage(),
    );
  }

  @override
  void dispose() {
    authUserSub.cancel();

    super.dispose();
  }

  void setLocale(String language) {
    setState(() => _locale = createLocale(language));
  }

  void setThemeMode(ThemeMode mode) => setState(() {
        _themeMode = mode;
        FlutterFlowTheme.saveThemeMode(mode);
      });

  @override
  Widget build(BuildContext context) {
    return MaterialApp.router(
      title: 'cookUp',
      localizationsDelegates: [
        FFLocalizationsDelegate(),
        GlobalMaterialLocalizations.delegate,
        GlobalWidgetsLocalizations.delegate,
        GlobalCupertinoLocalizations.delegate,
      ],
      locale: _locale,
      supportedLocales: const [Locale('en', '')],
      theme: ThemeData(brightness: Brightness.light),
      darkTheme: ThemeData(brightness: Brightness.dark),
      themeMode: _themeMode,
      routeInformationParser: _router.routeInformationParser,
      routerDelegate: _router.routerDelegate,
    );
  }
}
```

El archivo **main.dart** es el punto de entrada principal de la aplicación CookUp. Aquí se inicializan y configuran diversos componentes esenciales, como el enrutador de navegación, la configuración del tema y la localización, así como la inicialización de Firebase y la gestión del estado de la aplicación.

El método **main** se encarga de inicializar los elementos necesarios para que la aplicación funcione correctamente. Entre estas tareas, se encuentra la inicialización de Firebase mediante el método **initFirebase()**. Además, se establece el tema de la aplicación utilizando **FlutterFlowTheme.initialize()**.

El widget **MyApp** es el widget raíz de la aplicación. Este widget es un **StatefulWidget** que gestiona el estado de la aplicación y contiene el enrutador de la aplicación y la configuración de idioma y tema.

Dentro de **MyApp**, se declara un estado **_MyAppState**, el cual se encarga de gestionar la configuración de idioma, el tema de la aplicación y la gestión del estado global de la aplicación mediante **AppStateNotifier** y **GoRouter**.

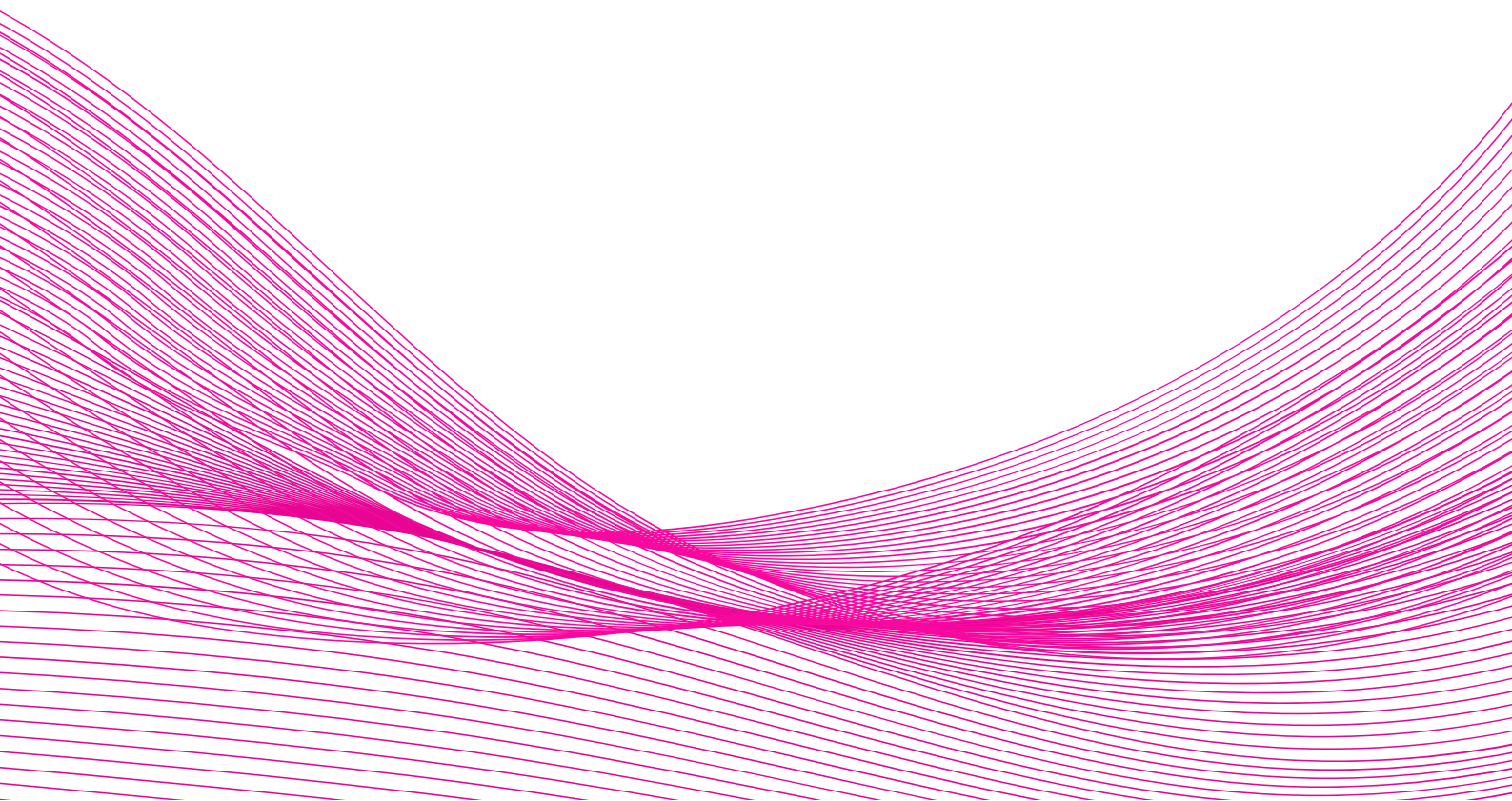
En el método **initState()** del estado de **MyApp**, se realiza la configuración inicial de los diferentes elementos necesarios. Aquí se inicializa **AppStateNotifier** y se crea el enrutador de la aplicación mediante **createRouter()**. También se configura el flujo de datos para la autenticación de Firebase y se establece un temporizador para ocultar la imagen de presentación después de 1 segundo.

El método **dispose()** se utiliza para realizar tareas de limpieza y cancelar cualquier suscripción o flujo de datos activo, como en el caso de authUserSub.

El método **setLocale()** se utiliza para establecer el idioma de la aplicación mediante la actualización de _locale.

El método **setThemeMode()** se utiliza para cambiar el modo de tema de la aplicación y guardar la configuración seleccionada.

El método **build()** se encarga de construir y retornar el MaterialApp.router, que es el widget raíz de la aplicación y utiliza un enrutador personalizado (_router) para gestionar la navegación entre pantallas. También se configuran las localizaciones admitidas y se establece el tema de la aplicación.
.



ADMINISTRADOR DE CATEGORIAS .DART

```dart
import '/backend/backend.dart';
import '/flutter_flow/flutter_flow_theme.dart';
import '/flutter_flow/flutter_flow_util.dart';
import 'package:flutter/material.dart';
import 'package:google_fonts/google_fonts.dart';
import 'package:provider/provider.dart';
import 'admin_categoria_model.dart';
export 'admin_categoria_model.dart';

class AdminCategoriaWidget extends StatefulWidget {
  const AdminCategoriaWidget({
    Key? key,
    int? valoracion,
  })  : this.valoracion = valoracion ?? 4,
        super(key: key);

  final int valoracion;

  @override
  _AdminCategoriaWidgetState createState() => _AdminCategoriaWidgetState();
}

class _AdminCategoriaWidgetState extends State<AdminCategoriaWidget> {
  late AdminCategoriaModel _model;

  final scaffoldKey = GlobalKey<ScaffoldState>();
  final _unfocusNode = FocusNode();

  @override
  void initState() {
    super.initState();
    _model = createModel(context, () => AdminCategoriaModel());
  }

  @override
  void dispose() {
    _model.dispose();

    _unfocusNode.dispose();
    super.dispose();
  }

  @override
  Widget build(BuildContext context) {
    context.watch<FFAppState>();

    return GestureDetector(
      onTap: () => FocusScope.of(context).requestFocus(_unfocusNode),
      child: Scaffold(
        key: scaffoldKey,
        backgroundColor: FlutterFlowTheme.of(context).primaryBackground,
        appBar: AppBar(
          backgroundColor: Color(0xFFF7089F),
          automaticallyImplyLeading: true,
          actions: [],
          centerTitle: true,
          elevation: 4.0,
        ),
        body: SafeArea(
          top: true,
          child: SingleChildScrollView(
            child: Column(
              mainAxisSize: MainAxisSize.max,
              children: [
                Align(
                  alignment: AlignmentDirectional(0.0, 1.16),
                  child: Container(
                    width: MediaQuery.of(context).size.width * 1.0,
                    height: MediaQuery.of(context).size.height * 1.0,
                    decoration: BoxDecoration(
                      color: FlutterFlowTheme.of(context).secondaryBackground,
                    ),
                    child: StreamBuilder<List<CategoriasRecord>>(
                      stream: queryCategoriasRecord(
                        queryBuilder: (categoriasRecord) =>
                            categoriasRecord.orderBy('idCategoria'),
                      ),
                      builder: (context, snapshot) {
                        // Customize what your widget looks like when it's loading.
                        if (!snapshot.hasData) {
                          return Center(
                            child: SizedBox(
                              width: 50.0,
                              height: 50.0,
                              child: CircularProgressIndicator(
                                color: FlutterFlowTheme.of(context).primary,
                              ),
                            ),
                          );
                        }
```

```dart
List<CategoriasRecord> columnCategoriasRecordList =
    snapshot.data!;
return Column(
  mainAxisSize: MainAxisSize.max,
  children: List.generate(
      columnCategoriasRecordList.length, (columnIndex) {
    final columnCategoriasRecord =
        columnCategoriasRecordList[columnIndex];
    return Padding(
      padding: EdgeInsetsDirectional.fromSTEB(
          16.0, 12.0, 16.0, 0.0),
      child: Container(
        width: double.infinity,
        height: 60.0,
        decoration: BoxDecoration(
          color: Colors.white,
          boxShadow: [
            BoxShadow(
              blurRadius: 5.0,
              color: Color(0x3416202A),
              offset: Offset(0.0, 2.0),
            )
          ],
          borderRadius: BorderRadius.circular(12.0),
          shape: BoxShape.rectangle,
        ),
        child: Padding(
          padding: EdgeInsetsDirectional.fromSTEB(
              8.0, 8.0, 8.0, 8.0),
          child: Row(
            mainAxisSize: MainAxisSize.max,
            children: [
              Expanded(
                child: Padding(
                  padding:
                      EdgeInsetsDirectional.fromSTEB(
                          12.0, 0.0, 0.0, 0.0),
                  child: Text(
                    columnCategoriasRecord.categoria,
                    style: FlutterFlowTheme.of(context)
                        .bodyLarge
                        .override(
                          fontFamily:
                              'Plus Jakarta Sans',
                          color: Color(0xFF14181B),
                          fontSize: 16.0,
                          fontWeight: FontWeight.normal,
                        ),
                  ),
                ),
              ),
              Align(
                alignment:
                    AlignmentDirectional(0.9, 0.0),
                child: InkWell(
                  splashColor: Colors.transparent,
                  focusColor: Colors.transparent,
                  hoverColor: Colors.transparent,
                  highlightColor: Colors.transparent,
                  onTap: () async {
                    context.pushNamed(
                      'editarCategoria',
                      queryParams: {
                        'nomCategoria': serializeParam(
                          columnCategoriasRecord
                              .categoria,
                          ParamType.String,
                        ),
                      }.withoutNulls,
                    );
                  },
                  child: Icon(
                    Icons.edit,
                    color: Color(0xFF7089F),
                    size: 18.0,
                  ),
                ),
              ),
              Align(
                alignment:
                    AlignmentDirectional(0.9, 0.0),
                child: Padding(
                  padding:
                      EdgeInsetsDirectional.fromSTEB(
                          15.0, 0.0, 5.0, 0.0),
                  child: InkWell(
                    splashColor: Colors.transparent,
                    focusColor: Colors.transparent,
                    hoverColor: Colors.transparent,
                    highlightColor: Colors.transparent,
                    onTap: () async {
                      await columnCategoriasRecord
                          .reference
                          .delete();
                    },
                    child: Icon(
                      Icons.delete,
                      color: Color(0xFF7089F),
                      size: 20.0,
                    ),
                  ),
                ),
              ),
            ],
          ),
        ),
      ),
    );
```

El widget AdminCategoriaWidget es un StatefulWidget que se encarga de mostrar y administrar la lista de categorías en el panel de administración. Este widget permite editar y eliminar categorías existentes.
Dentro del widget, se declaran variables y objetos necesarios, como scaffoldKey para gestionar el scaffold, _unfocusNode para manejar el enfoque de los elementos de entrada y _model para manejar la lógica y el estado de las categorías.
En el método initState(), se inicializa _model mediante la función createModel(), que crea una instancia de AdminCategoriaModel, el cual es responsable de la lógica y el estado relacionados con las categorías.
En el método dispose(), se realiza la limpieza necesaria, como la liberación de recursos y la eliminación de suscripciones y flujos de datos.
El método build() construye y retorna la estructura de la interfaz de usuario del widget. La interfaz de usuario consiste en un GestureDetector que permite ocultar el teclado al tocar fuera de los elementos de entrada, seguido de un Scaffold que contiene una AppBar y el cuerpo principal de la pantalla.
Dentro del cuerpo principal, se utiliza un StreamBuilder para obtener la lista de categorías desde la base de datos. Mientras se carga la lista, se muestra un CircularProgressIndicator. Una vez que los datos están disponibles, se muestran las categorías en forma de lista mediante un Column y ListView.builder.
Cada categoría se representa mediante un Container con un diseño personalizado y contiene el nombre de la categoría. También se incluyen iconos de edición y eliminación para cada categoría, que permiten realizar acciones correspondientes.
En general, este widget AdminCategoriaWidget es responsable de mostrar la lista de categorías en el panel de administración y proporcionar funcionalidades para editar y eliminar categorías.

SOPORTE Y CONTACTO

Si necesitas ayuda adicional, reportar problemas o tienes alguna consulta sobre la aplicación Cook Up, puedes ponerte en contacto con el equipo de soporte técnico. A continuación, se proporciona la información de contacto y los recursos disponibles:

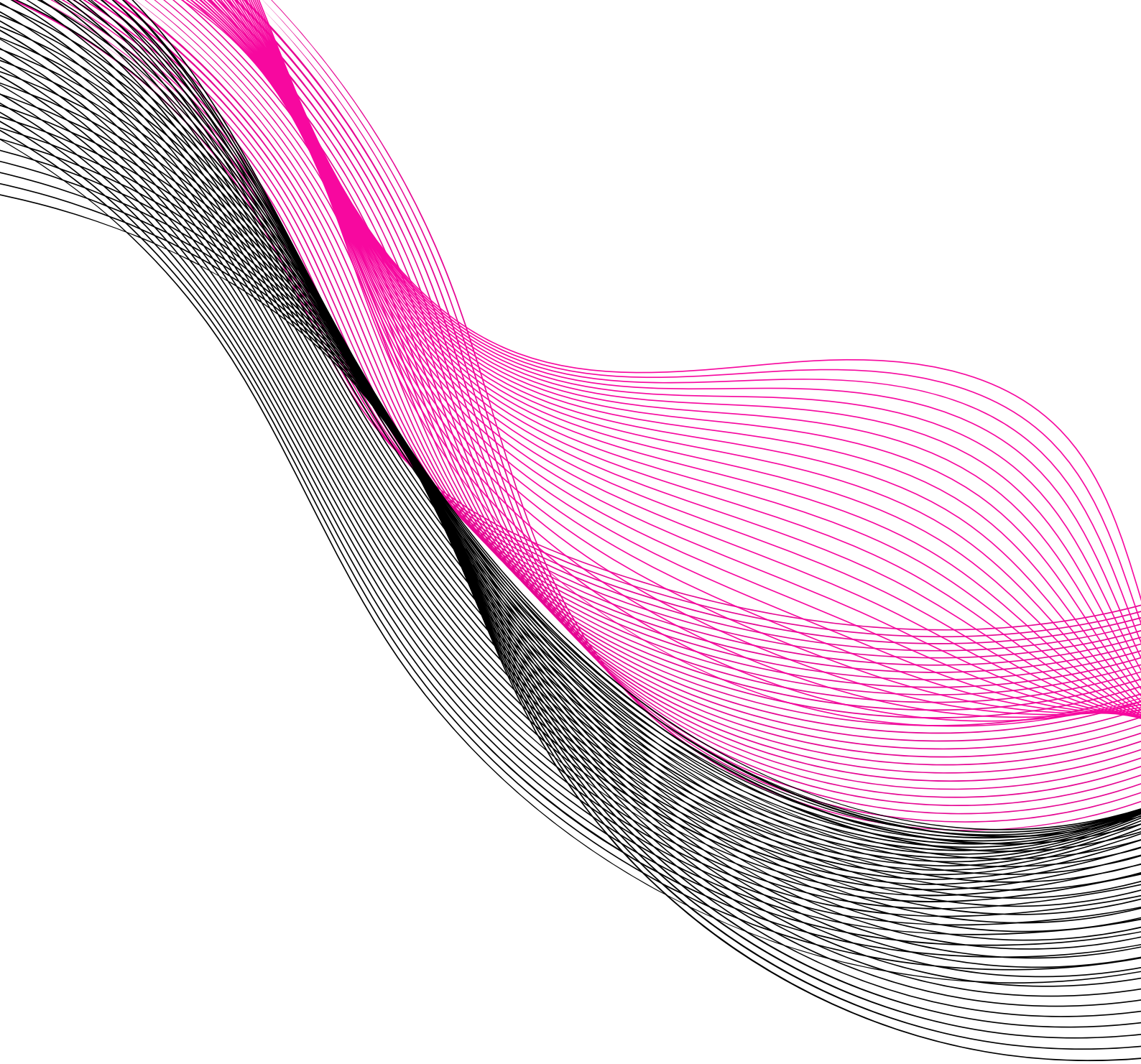
Desarrollador: Andrea Isabel Vega Márquez
Correo electrónico: a329634@hotmail.com

Plataformas utilizadas para el desarrollo:

- FlutterFlow: Una plataforma visual para el desarrollo de aplicaciones móviles.
- Firebase: Un conjunto de herramientas en la nube de Google para desarrollar aplicaciones web y móviles.
- Android Studio: Un entorno de desarrollo integrado (IDE) utilizado para crear aplicaciones Android.

Si tienes alguna pregunta específica relacionada con el desarrollo de la aplicación Cook Up utilizando estas plataformas, puedes enviar un correo electrónico a a329634@hotmail.com para recibir asistencia directa del desarrollador.

Además, te recomendamos explorar los recursos de soporte en línea proporcionados por las plataformas utilizadas, como los foros de comunidad, la documentación oficial y los tutoriales disponibles. Estos recursos pueden brindarte información adicional, soluciones a problemas comunes y guías paso a paso para aprovechar al máximo la aplicación Cook Up.



Gracias.