Stripe Account

Table of contents:

1.OverView

1. Create a Stripe account and obtain API keys for testing:

Explanation:

- **Stripe Account**: To process payments, you'll need a Stripe account. Stripe provides a payment processing platform that allows you to accept payments via credit cards, debit cards, and other payment methods.
- **API Keys**: Stripe uses API keys to authenticate API requests. There are two types of keys: publishable keys (for the frontend) and secret keys (for the backend).

Steps:

1. **Sign Up for Stripe**: Go to the Stripe website and create an account if you don't already have one.
2. **Obtain API Keys**: After logging in, navigate to the Developers section in the Stripe Dashboard. Under API keys, you will find your publishable key and secret key. Use the test keys for development and testing purposes.

1.21Payment UI

1. Create a payment screen for users to enter payment details:

Explanation:

- **Payment Screen**: The interface where users input their payment information. This screen should be designed to capture all necessary details for processing a payment securely and efficiently.

Steps:

1. **Design Payment Form**: Create a form that collects the user's payment information.
2. **Security and Validation**: Ensure the form includes appropriate validation for each input field to prevent errors and enhance security.

2. Include fields for card number, expiration date, CVV, and cardholder name:

Explanation:

- **Card Number**: The 16-digit number on the front of the card.
- **Expiration Date**: The date the card expires, typically formatted as MM/YY.
- **CVV**: The Card Verification Value, a 3-digit number on the back of the card.
- **Cardholder Name**: The name of the person to whom the card is issued.

Steps:

1. **Input Fields**: Create input fields for each piece of information required.
2. **Validation**: Implement validation to ensure that the information entered is in the correct format (e.g., a valid card number, proper date format, etc.).

1.3 Backend Integration

1. Use Firebase Cloud Functions to securely handle payment processing:

Explanation:

- **Firebase Cloud Functions**: Serverless functions that let you run backend code in response to events triggered by Firebase features and HTTPS requests. They are ideal for handling sensitive operations like payment processing.
- **Secure Payment Handling**: The payment details should be sent to the backend securely, where the actual processing with Stripe takes place.

Steps:

1. **Set Up Firebase Cloud Functions**: Initialize Cloud Functions in your Firebase project.
2. **Install Stripe Node.js Library**: Use the Stripe Node.js library to interact with the Stripe API in your Cloud Functions.
3. **Create Payment Endpoint**: Write a function that handles the payment processing logic. This function will receive payment details from the frontend, create a payment intent with Stripe, and handle the response.

1.4 Success and Error Handling

1. Display success or error messages based on the payment outcome:

Explanation:

- **Success Handling**: Inform the user that their payment was successful. This may involve displaying a confirmation message and possibly sending a confirmation email.
- **Error Handling**: Inform the user if there was an error processing their payment, providing details about what went wrong and how they can resolve it.

Steps:

1. **Handle Response**: After the payment is processed by the backend, handle the response on the frontend to display appropriate messages.
2. **Display Messages**: Show a success message if the payment was successful or an error message if it failed.

2.Pages contained:

1.main.dart

2.paymentScreen.dart

3.theme.dart

4.firbase_options.dart

5.success.dart

6.failed.dart

3.Firebase Connection

Firebase is a comprehensive app development platform backed by Google, providing a suite of tools and services to help developers build high-quality applications quickly and efficiently. In this brief explanation, we'll cover the essential aspects of connecting to Firebase and highlight some of its key features.

Connecting to Firebase

To integrate Firebase with a Flutter application, follow these general steps:

1. Set Up Firebase Project:
    - Go to the Firebase Console.
    - Create a new project or select an existing one.
    - Add your app to the project (iOS, Android, or Web).
2. Add Firebase SDK:
    - For Flutter, include the necessary Firebase plugins in your pubspec.yaml file. Commonly used plugins include firebase_core, firebase_auth, and cloud_firestore.
    - We can add our firebase dependencies to our pubspec.yaml file by using flutter pub add name_of_dependency in out project directory.

3. Initialize Firebase:
    - Initialize Firebase in your app by calling Firebase.initializeApp() in the main entry point of your application. This is usually done in the main.dart file.

Key Features of Firebase

1. Firebase Authentication:
    - Simplifies user authentication by providing a variety of sign-in methods, including email/password, Google Sign-In, Facebook Login, and more.
    - Manages authentication state across different platforms.
2. Cloud Firestore:
    - A flexible, scalable database for mobile, web, and server development.
    - Supports real-time synchronization, enabling apps to respond to data changes instantly.
    - Provides robust querying capabilities and offline data persistence.
3. Firebase Storage:
    - Facilitates the storage of user-generated content like photos and videos.
    - Offers secure file uploads and downloads directly from mobile and web apps.

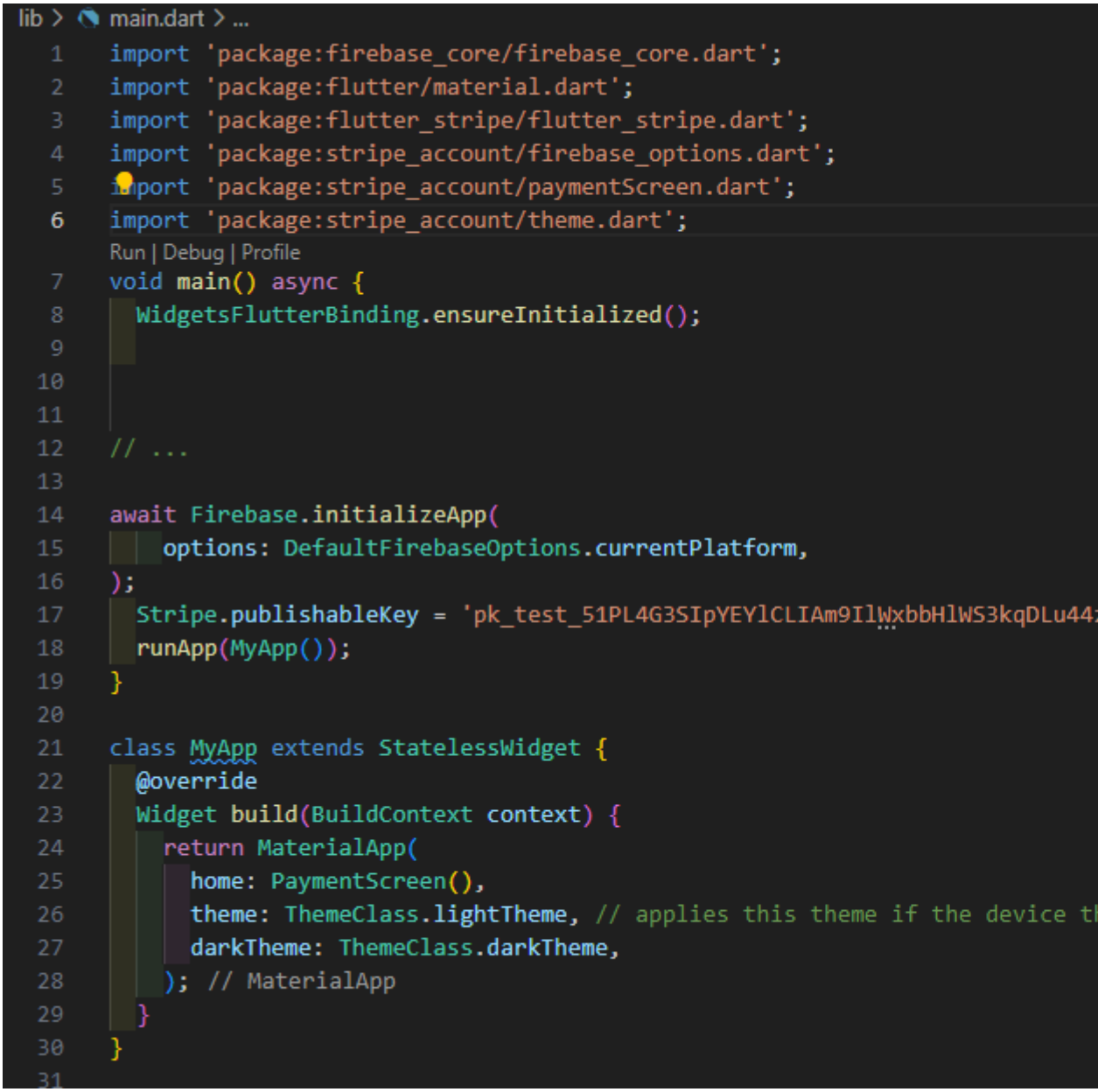
4. A Detailed Explanation on Each Page:

1.main.dart(main.dart):

The **main.dart** file is the entry point of the Flutter application. It initializes Firebase, sets up the Stripe payment gateway, and launches the application with specific themes.

Dependencies

- **firebase_core**: To initialize Firebase.
- **flutter/material.dart**: Core Flutter package for building UI.
- **flutter_stripe**: To integrate Stripe payment processing.
- **firebase_options.dart**: Provides Firebase configuration options.
- **paymentScreen.dart**: Contains the **PaymentScreen** widget, the home screen of the app.
- **theme.dart**: Contains the theme settings for the application.



```dart
lib > main.dart > ...
import 'package:firebase_core/firebase_core.dart';
import 'package:flutter/material.dart';
import 'package:flutter_stripe/flutter_stripe.dart';
import 'package:stripe_account/firebase_options.dart';
import 'package:stripe_account/paymentScreen.dart';
import 'package:stripe_account/theme.dart';
Run | Debug | Profile
void main() async {
  WidgetsFlutterBinding.ensureInitialized();



// ...

await Firebase.initializeApp(
    options: DefaultFirebaseOptions.currentPlatform,
);
  Stripe.publishableKey = 'pk_test_51PL4G3SIpYEYlCLIAm9IlWxbbHlWS3kqDLu44
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: PaymentScreen(),
      theme: ThemeClass.lightTheme, // applies this theme if the device t
      darkTheme: ThemeClass.darkTheme,
    ); // MaterialApp
  }
}
```

- **Main Initialization**: The main function initializes Flutter bindings and Firebase, then sets the Stripe publishable key.
- **Stripe Configuration**: Sets the Stripe publishable key, necessary for handling payments.
- **App Start**: Runs the MyApp widget, starting the app.
- **MyApp Widget**: A stateless widget that sets up MaterialApp with a PaymentScreen as the home, and applies light and dark themes based on the device's theme settings.
- This code sets up the necessary configurations for Firebase and Stripe, then launches a basic Flutter application with theming support and a designated home screen for payment processing.

2.Payment Screen (paymentScreen.dart)

- **dart**: Provides functionality to encode and decode JSON.

- **flutter/material.dart**: Provides material design components and utilities for building the UI.
- **flutter_stripe/flutter_stripe.dart**: A package that provides Stripe integration for Flutter applications.
- **http/http.dart**: A package for making HTTP requests.

```
import 'dart:convert';

import 'package:flutter/material.dart';
import 'package:flutter_stripe/flutter_stripe.dart';
import 'package:http/http.dart' as http;

import 'failed.dart';
import 'success.dart';

class PaymentScreen extends StatefulWidget {
  @override
  _PaymentScreenState createState() => _PaymentScreenState();
}

class _PaymentScreenState extends State<PaymentScreen> {
  final _cardNumberController = TextEditingController();
  final _expiryDateController = TextEditingController();
  final _cvvController = TextEditingController();
  final _cardHolderNameController = TextEditingController();
  final _moneyController=TextEditingController();
  Map<String, dynamic>? paymentIntent;
  final GlobalKey<FormState> _formKey = GlobalKey<FormState>();
  @override
  Future<void> payment() async{
    try{
    Map<String,dynamic> body={
      'amount':_moneyController.text,//10 rupees == 1000 cents
      'currency':"INR",
    };
    var response =await http.post(
      Uri.parse('https://api.stripe.com/v1/payment_intents'),
      headers:{
        'Content-Type': 'application/json',
        'Authorization':'Bearer
sk_test_51PL4G3SIpYEYlCLIULDKZXJkg0ojofG3DDus2VyRBubKK9YwAoo7UTpe
AJv6JB95QhZdflG271qRylITk6RUQ6tO00lU4luKzM',
      }
    );
    paymentIntent=json.decode(response.body);
  }catch(e){
    throw Exception(e);
  }
```

```
•
•    //initializing payment sheet
•    await Stripe.instance.initPaymentSheet(paymentSheetParameters:
  SetupPaymentSheetParameters(
•          paymentIntentClientSecret: paymentIntent!['Client
  Secret'],
•          style:ThemeMode.light,
•          merchantDisplayName: _cardHolderNameController.text,
•
•    )).then((value)=>{});
•
•    // Display Payment Sheet
•
•    try{
•        await Stripe.instance.presentPaymentSheet().then((value)=>{
•          // Success State
•          _showSuccessMessage(),
•        });
•    }catch(error){
•        _showErrorMessage(error.toString());
•    }
•    }
•    void _showSuccessMessage() {
•      ScaffoldMessenger.of(context).showSnackBar(
•        SnackBar(content: Text('Payment successful!')),
•      );
•      Navigator.push(context,MaterialPageRoute(builder:(_)=>Success
  ()));
•
•    }
•
•    void _showErrorMessage(String message) {
•      ScaffoldMessenger.of(context).showSnackBar(
•        SnackBar(content: Text('Payment failed: $message')),
•      );
•      Navigator.push(context,MaterialPageRoute(builder:(_)=>Failed(
  )));
•    }
•
•    @override
•    Widget build(BuildContext context) {
•      final screenWidth = MediaQuery.of(context).size.width;
•      final screenHeight = MediaQuery.of(context).size.height;
•      final Color containerColor =
  Theme.of(context).colorScheme.surface;
•      return Scaffold(
•        appBar: AppBar(title: Text('Payment')),
•
```

```
        body: SingleChildScrollView(
          child: Center(
            child: Form(
              key: _formKey,
              child: Container(
                decoration: const BoxDecoration(
                  gradient: LinearGradient(
                    colors: [
                      Color.fromARGB(255, 216, 233, 247),
                      Color.fromARGB(255, 243, 189, 185),
                    ],
                    begin: Alignment.topLeft,
                    end: Alignment.bottomRight,
                  ),
                  borderRadius:
    BorderRadius.all(Radius.circular(7.0)),
                ),
                child: Column(
                  children: [
                    Padding(
                      padding: EdgeInsets.fromLTRB(
                        screenWidth * 0.1,
                        screenWidth * 0.3,
                        screenWidth * 0.1,
                        0,
                      ),
                      child: TextFormField(
                        validator:(value){
                          if (value==null || value.isEmpty){
                          return "Required";
                          }
                          return null;
                        },
                        controller: _cardHolderNameController,
                        keyboardType: TextInputType.name,
                        decoration: InputDecoration(
                          labelText: 'Card Holder Name',
                          border: OutlineInputBorder(
                            borderRadius:
    BorderRadius.circular(9.0),
                          ),
                        ),
                      ),
                    ),
                    Padding(
                      padding: EdgeInsets.fromLTRB(
                        screenWidth * 0.1,
                        screenWidth * 0.05,
```

```
                        screenWidth * 0.1,
                        0,
                      ),
                      child: TextFormField(
                        validator:(value){
                          if (value==null || value.isEmpty){
                          return "Required";
                          }
                          return null;
                        },
                        controller: _cardNumberController,
                        keyboardType: TextInputType.number,
                        decoration: InputDecoration(
                          labelText: 'Card Number',
                          border: OutlineInputBorder(
                            borderRadius:
    BorderRadius.circular(9.0),
                          ),
                        ),
                      ),
                    ),
                    Padding(
                      padding: EdgeInsets.fromLTRB(
                        screenWidth * 0.1,
                        screenWidth * 0.05,
                        screenWidth * 0.1,
                        0,
                      ),
                      child: TextFormField(
                          validator:(value){
                          if (value==null || value.isEmpty){
                          return "Required";
                          }
                          return null;
                        },
                        controller: _expiryDateController,
                        keyboardType: TextInputType.number,
                        decoration: InputDecoration(
                          labelText: 'Expiry Date (MM/YY)',
                          border: OutlineInputBorder(
                            borderRadius:
    BorderRadius.circular(9.0),
                          ),
                        ),
                      ),
                    ),
                    Padding(
                      padding: EdgeInsets.fromLTRB(
```

```
                        screenWidth * 0.1,
                        screenWidth * 0.05,
                        screenWidth * 0.1,
                        0,
                      ),
                      child: TextFormField(
                        validator:(value){
                          if (value==null || value.isEmpty){
                          return "Required";
                          }
                          return null;
                        },
                        controller: _cvvController,
                        keyboardType: TextInputType.number,
                        decoration: InputDecoration(
                          labelText: 'CVV',
                          border: OutlineInputBorder(
                            borderRadius:
    BorderRadius.circular(9.0),
                          ),
                        ),
                      ),
                    ),
                      Padding(
                      padding: EdgeInsets.fromLTRB(
                        screenWidth * 0.1,
                        screenWidth * 0.05,
                        screenWidth * 0.1,
                        0,
                      ),
                      child: TextFormField(
                        validator:(value){
                          if (value==null || value.isEmpty){
                          return "Required";
                          }
                          return null;
                        },
                        controller: _moneyController,
                        keyboardType: TextInputType.number,
                        decoration: InputDecoration(
                          labelText: 'money in cents eg:1000
    cents=10 rupees',
                          border: OutlineInputBorder(
                            borderRadius:
    BorderRadius.circular(9.0),
                          ),
                        ),
                      ),
```

```
                    ),
                    Padding(
                      padding: EdgeInsets.fromLTRB(
                        screenWidth * 0.1,
                        screenWidth * 0.05,
                        screenWidth * 0.1,
                        screenWidth * 0.5,
                      ),
                      child: ElevatedButton(
                        onPressed: ()=>payment,
                        child: Text(
                          "Pay Now",
                          style: TextStyle(color: containerColor),
                        ),
                        style: ElevatedButton.styleFrom(
                          backgroundColor: Colors.blue,
                        ),
                      ),
                    ),
                  ],
                ),
              ),
            ),
          ),
        ),
      );
    }
  }
```

- **TextEditingController**: Controllers to manage the input text for card details and amount.
- **GlobalKey<FormState>**: A key to uniquely identify the form and validate its state.
- **Map<String, dynamic>? paymentIntent**: A map to store the payment intent data from Stripe.
- **_showSuccessMessage():** Displays a Snackbar with a success message and Navigates to Success Screen.
- **_showErrorMessage(String message):** Displays a Snackbar with an error message and Navigates to Failed Screen.
- **http.post:** Sends a POST request to Stripe's API to create a payment intent.
- **presentPaymentSheet**: Displays the Stripe payment sheet to the user.
- **paymentIntent**: Stores the response from Stripe.
- **initPaymentSheet**: Initializes the Stripe payment sheet with parameters, including the payment intent client secret.

3.Success Screen (success. dart)

1. **AppBar:**
    - The AppBar contains an IconButton that, when pressed, navigates the user back to the PaymentScreen. The navigation is handled using Navigator.pushReplacement, which replaces the current screen with the PaymentScreen without allowing the user to go back to the success screen.
2. **Body:**
    - The body of the scaffold contains a Container widget.
    - The Container has a BoxDecoration with an DecorationImage. This image is fetched from the assets and displayed as the background of the container, covering the entire screen (BoxFit.cover).
3. **Background Image:** The success message is visually represented with an image, enhancing the user interface with a graphical indication of success.

```
import 'package:flutter/material.dart';
import 'package:stripe_account/paymentScreen.dart';
class Success extends StatefulWidget {
  const Success({super.key});

  @override
  State<Success> createState() => _SuccessState();
}

class _SuccessState extends State<Success> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(actions: [
        IconButton(onPressed: ()=>Navigator.pushReplacement(context, MaterialPageRoute(builder: (_)=>PaymentScreen())), icon: Icon(Icons.arrow_back))
      ],),
      body:Container(
        decoration: BoxDecoration(
          image:DecorationImage(image: AssetImage("assets/PaymentSuccess.png"),fit:BoxFit.cover)
        ),
      )
    );
  }
}
```

4.Failed Screen(failed.dart)

The Failed class represents a screen in a Flutter application that displays a failure message, typically after an unsuccessful payment. It is a stateless widget, which means it does not maintain any state over its lifecycle.

- The **AppBar** contains an **IconButton** that, when pressed, navigates the user back to the **PaymentScreen**. This is done using **Navigator.pushReplacement**, which replaces the current screen with the **PaymentScreen**, preventing the user from going back to the failure screen.
- The body of the scaffold contains a Container widget.
- The Container has a BoxDecoration with an DecorationImage. This image is fetched from the assets and displayed as the background of the container, covering the entire screen (BoxFit.cover).
- **Background Image:** The Failure message is visually represented with an image, enhancing the user interface with a graphical indication of Faliure while transaction.

```
import 'package:flutter/material.dart';
import 'package:stripe_account/paymentScreen.dart';
class Failed extends StatelessWidget {
  const Failed({super.key});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(actions: [
        IconButton(onPressed: ()=>Navigator.pushReplacement(context,
MaterialPageRoute(builder: (_)=>PaymentScreen())), icon:
Icon(Icons.arrow_back))
      ],),
      body:Container(
        decoration: BoxDecoration(
          image:DecorationImage(image:
AssetImage("assets/failed.png"),fit:BoxFit.cover)
        ),
      )
    );
  }
}
```

5. Default Firebase Options (firebase_optons.dart)

Overview:

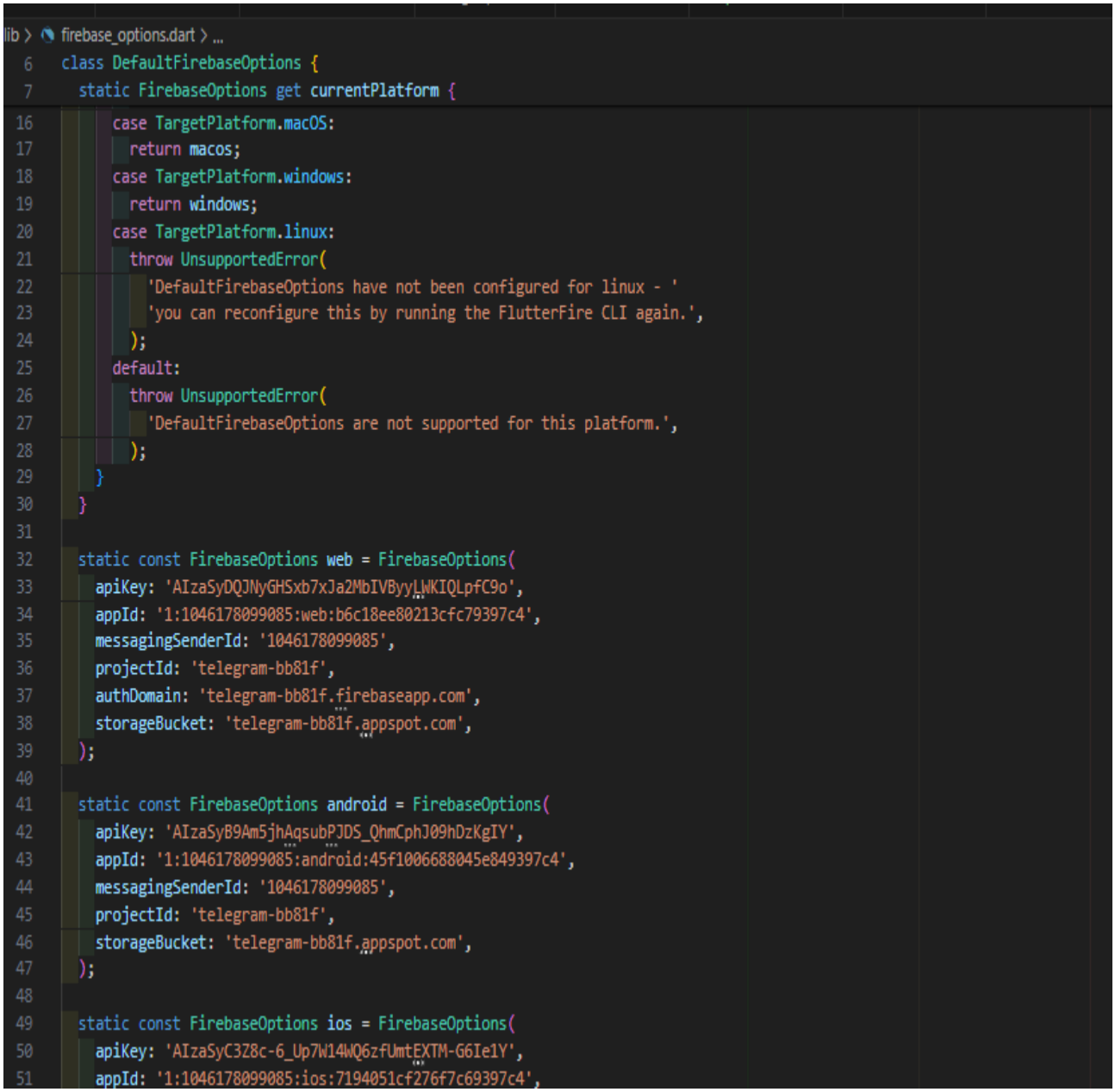
The **DefaultFirebaseOptions** class defines default Firebase options for different platforms. These options are used for initializing Firebase within the Flutter app.

Platforms Supported:

1. Web
2. Android
3. iOS

4. macOS
5. Windows

Here is the Firebase_Options.dart file which defines the firebase options for different platforms.



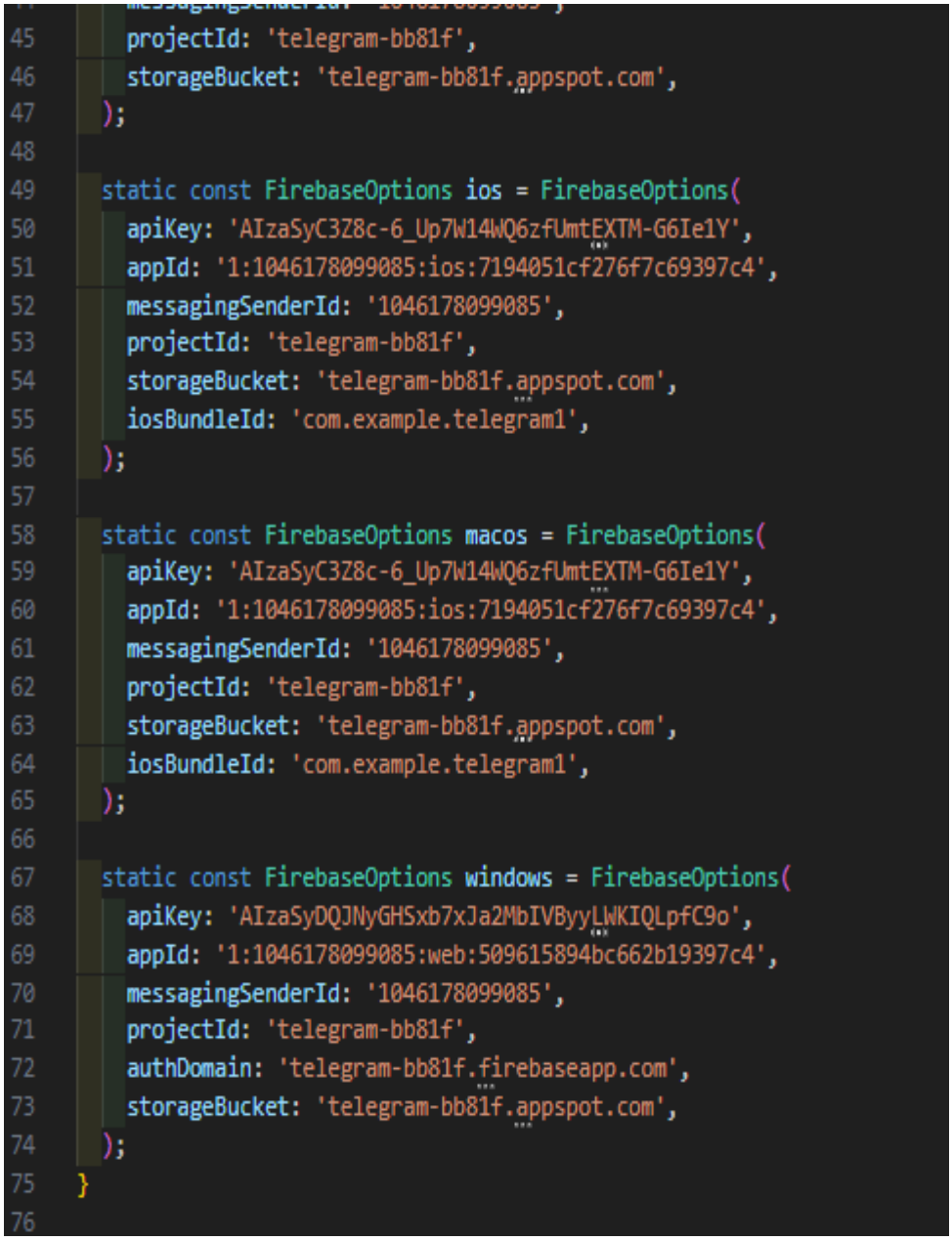
lib > firebase_options.dart > ...

```dart
class DefaultFirebaseOptions {
  static FirebaseOptions get currentPlatform {
      case TargetPlatform.macOS:
        return macos;
      case TargetPlatform.windows:
        return windows;
      case TargetPlatform.linux:
        throw UnsupportedError(
          'DefaultFirebaseOptions have not been configured for linux - '
          'you can reconfigure this by running the FlutterFire CLI again.',
        );
      default:
        throw UnsupportedError(
          'DefaultFirebaseOptions are not supported for this platform.',
        );
    }
  }

  static const FirebaseOptions web = FirebaseOptions(
    apiKey: 'AIzaSyDQJNyGHSxb7xJa2MbIVByyLWKIQLpfC9o',
    appId: '1:1046178099085:web:b6c18ee80213cfc79397c4',
    messagingSenderId: '1046178099085',
    projectId: 'telegram-bb81f',
    authDomain: 'telegram-bb81f.firebaseapp.com',
    storageBucket: 'telegram-bb81f.appspot.com',
  );

  static const FirebaseOptions android = FirebaseOptions(
    apiKey: 'AIzaSyB9Am5jhAqsubPJDS_QhmCphJ09hDzKgIY',
    appId: '1:1046178099085:android:45f1006688045e849397c4',
    messagingSenderId: '1046178099085',
    projectId: 'telegram-bb81f',
    storageBucket: 'telegram-bb81f.appspot.com',
  );

  static const FirebaseOptions ios = FirebaseOptions(
    apiKey: 'AIzaSyC3Z8c-6_Up7W14WQ6zfUmtEXTM-G6Ie1Y',
    appId: '1:1046178099085:ios:7194051cf276f7c69397c4',
```



```
    projectId: 'telegram-bb81f',
    storageBucket: 'telegram-bb81f.appspot.com',
  );

  static const FirebaseOptions ios = FirebaseOptions(
    apiKey: 'AIzaSyC3Z8c-6_Up7W14WQ6zfUmtEXTM-G6Ie1Y',
    appId: '1:1046178099085:ios:7194051cf276f7c69397c4',
    messagingSenderId: '1046178099085',
    projectId: 'telegram-bb81f',
    storageBucket: 'telegram-bb81f.appspot.com',
    iosBundleId: 'com.example.telegram1',
  );

  static const FirebaseOptions macos = FirebaseOptions(
    apiKey: 'AIzaSyC3Z8c-6_Up7W14WQ6zfUmtEXTM-G6Ie1Y',
    appId: '1:1046178099085:ios:7194051cf276f7c69397c4',
    messagingSenderId: '1046178099085',
    projectId: 'telegram-bb81f',
    storageBucket: 'telegram-bb81f.appspot.com',
    iosBundleId: 'com.example.telegram1',
  );

  static const FirebaseOptions windows = FirebaseOptions(
    apiKey: 'AIzaSyDQJNyGHSxb7xJa2MbIVByyLWKIQLpfC9o',
    appId: '1:1046178099085:web:509615894bc662b19397c4',
    messagingSenderId: '1046178099085',
    projectId: 'telegram-bb81f',
    authDomain: 'telegram-bb81f.firebaseapp.com',
    storageBucket: 'telegram-bb81f.appspot.com',
  );
}
```

Usage:

```dart
import 'package:flutter/material.dart';
import 'package:sampl1/chats.dart';
import 'package:sampl1/firebase_options.dart';
import 'package:sampl1/theme.dart';

import 'loginScreen.dart';


Run | Debug | Profile
void main() async{
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp(
    options: DefaultFirebaseOptions.currentPlatform
  );

}
```

- The **currentPlatform** getter returns the Firebase options based on the current platform or target platform.

6. Theme Class (theme_class.dart)

Overview:

The **ThemeClass** defines a class for managing app themes and color schemes. It includes color schemes for primary, secondary, and accent colors, supporting both light and dark themes.

Features:

1. Define color schemes for light and dark themes
2. Toggle between light and dark themes
3. Customize primary, secondary, and accent colors.

```dart
import 'package:flutter/material.dart';
class ThemeClass{
  Color lightPrimaryColor = Colors.blue;//light theme
  Color secondaryColor = Colors.green;
  Color acccentColor = Colors.amberAccent;
  Color darkPrimaryColor = Colors.black;
  static ThemeData lightTheme=ThemeData(
    primaryColor: ThemeData.light().scaffoldBackgroundColor,
    colorScheme: const ColorScheme.light(
          primary: Colors.blue,
          secondary: Colors.green,
          background: Colors.white,
          surface:Colors.grey)
```

```
  );

  static ThemeData darkTheme =ThemeData(
    primaryColor: ThemeData.dark().scaffoldBackgroundColor,
    colorScheme: const ColorScheme.dark(
      primary: Colors.blue,
      background: Colors.black,
      secondary:Colors.green,
      surface: Colors.grey,

    )
  );
}

ThemeClass _themeClass =ThemeClass();
```

Conclusion:

The code effectively integrates Stripe into a Flutter application, providing a solid foundation for building a payment feature. It combines necessary input management, payment processing, and user feedback mechanisms, making it a comprehensive example of handling payments in a mobile app. By focusing on both functionality and user experience, the application ensures a secure and efficient payment process for users.