Booking App(Final Project ITI)

Overview

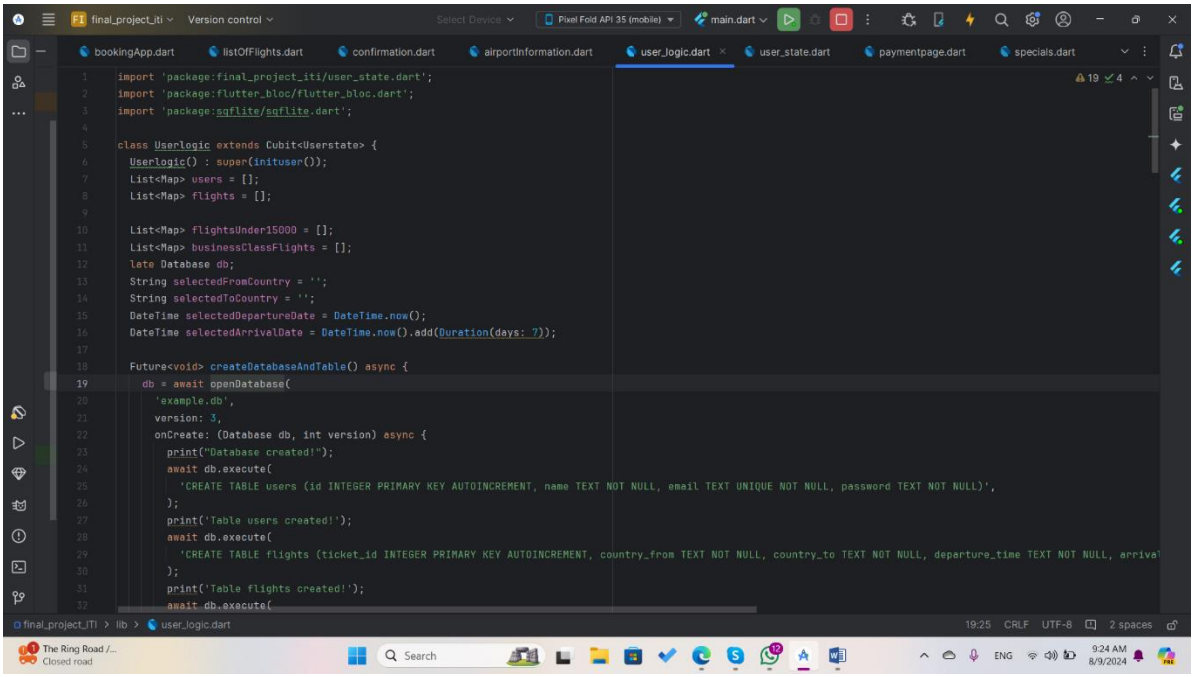
The Flight Booking Application is a comprehensive mobile solution designed to streamline the process of searching, booking, and managing flights. This application leverages Flutter, a popular cross-platform UI framework, and the Bloc state management pattern to ensure a seamless and responsive user experience.

Key Features:

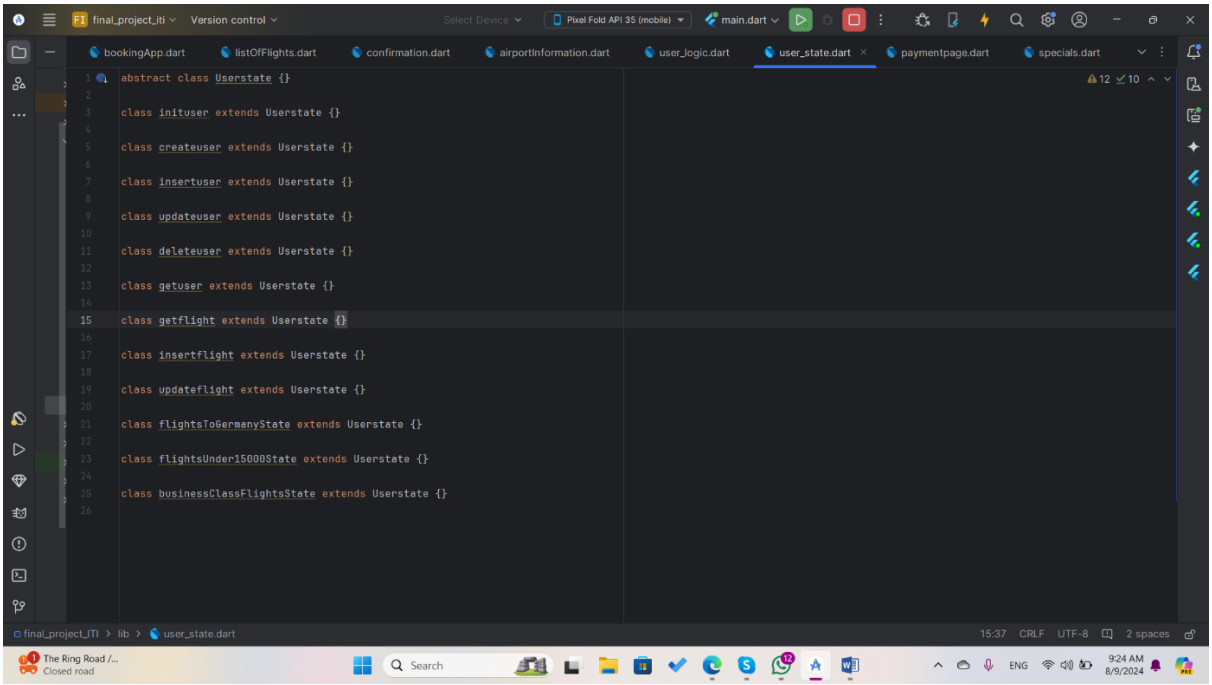
- **Flight Booking:** Users can book available flights based on their preferences, such country from, country to, class of service, and price. The intuitive user interface allows users to view details for each flight and proceed to book their tickets with ease.
- **Payment Integration:** The app integrates a secure payment gateway that enables users to complete their transactions and receive instant confirmation of their bookings. The payment process is streamlined to ensure user satisfaction and security.
- **Booking Confirmation:** After a successful transaction, users are provided with detailed confirmation of their booking, including travel details and price. This confirmation is accessible at any time for reference.
- **Airport Information:** To enhance the travel experience, the application includes an Airport Information section. This feature provides users with essential details about various facilities within the airport, such as terminals, lounges, parking, and more.

Purpose & Goals:

The primary goal of this project is to create a user-friendly and efficient flight booking experience on mobile devices. By focusing on the core functionalities of flight search, payment, and airport information, the application aims to cater to frequent travelers who value convenience and ease of use.



```dart
import 'package:final_project_iti/user_state.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:sqflite/sqflite.dart';

class Userlogic extends Cubit<Userstate> {
  Userlogic() : super(inituser());
  List<Map> users = [];
  List<Map> flights = [];

  List<Map> flightsUnder15000 = [];
  List<Map> businessClassFlights = [];
  late Database db;
  String selectedFromCountry = '';
  String selectedToCountry = '';
  DateTime selectedDepartureDate = DateTime.now();
  DateTime selectedArrivalDate = DateTime.now().add(Duration(days: 7));

  Future<void> createDatabaseAndTable() async {
    db = await openDatabase(
      'example.db',
      version: 3,
      onCreate: (Database db, int version) async {
        print("Database created!");
        await db.execute(
          'CREATE TABLE users (id INTEGER PRIMARY KEY AUTOINCREMENT, name TEXT NOT NULL, email TEXT UNIQUE NOT NULL, password TEXT NOT NULL)',
        );
        print('Table users created!');
        await db.execute(
          'CREATE TABLE flights (ticket_id INTEGER PRIMARY KEY AUTOINCREMENT, country_from TEXT NOT NULL, country_to TEXT NOT NULL, departure_time TEXT NOT NULL, arrival
        );
        print('Table flights created!');
        await db.execute(
```



```dart
abstract class Userstate {}

class inituser extends Userstate {}

class createuser extends Userstate {}

class insertuser extends Userstate {}

class updateuser extends Userstate {}

class deleteuser extends Userstate {}

class getuser extends Userstate {}

class getflight extends Userstate {}

class insertflight extends Userstate {}

class updateflight extends Userstate {}

class flightsToGermanyState extends Userstate {}

class flightsUnder15000State extends Userstate {}

class businessClassFlightsState extends Userstate {}
```

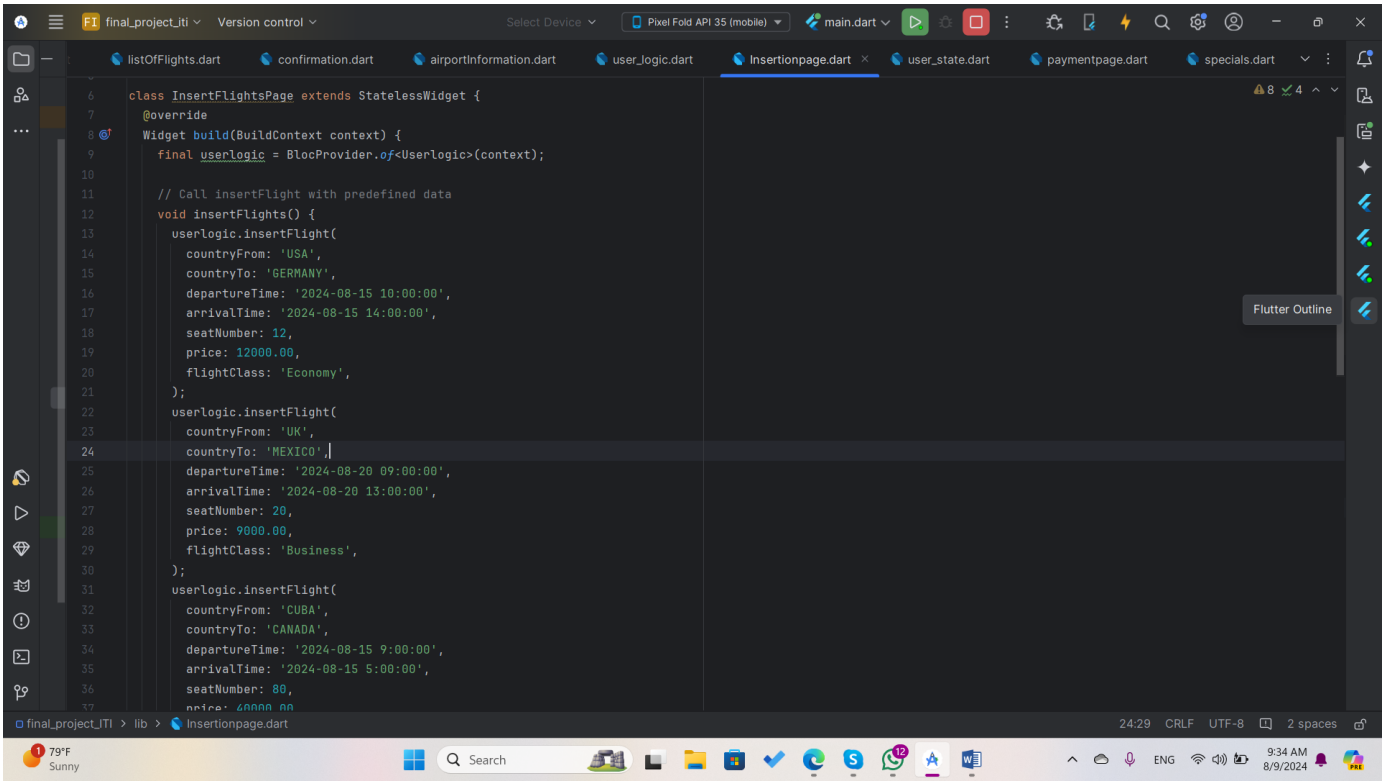
'User Logic' Class

The `Userlogic` class is responsible for managing the state of users and flights within the application. It handles database operations such as creating tables, inserting, updating, and fetching data. The class emits different states to reflect changes in the data, allowing the UI to update accordingly.

States

1. **inituser**:
   This state is emitted when the `Userlogic` class is first initialized. It represents the initial state before any data is fetched or any operation is performed.
2. **getflight**:
   Similar to `getuser`, this state is emitted when flight data is retrieved from the database. It follows the `fetchFlights()` method and indicates that flight information is ready to be used by the application.
3. **insertflight**:
   Emitted after a new flight is successfully added to the `flights` table. This state allows the application to refresh the flight list or update the UI with the newly added flight.
4. **updateflight**:
   This state is emitted when an existing flight's details are updated in the database. It signals that the flight data has been modified, requiring the UI to reflect the changes.
5. **flightsUnder15000State**:
   Emitted after fetching flights with a price under 15,000. This state allows the application to display a list of affordable flights to the user.
6. **businessClassFlightsState**:
   This state is emitted when business class flights are retrieved from the database. The application can use this state to present the user with available premium flight options.

InsertFlightsPage:




```dart
class InsertFlightsPage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    final userlogic = BlocProvider.of<Userlogic>(context);

    // Call insertFlight with predefined data
    void insertFlights() {
      userlogic.insertFlight(
        countryFrom: 'USA',
        countryTo: 'GERMANY',
        departureTime: '2024-08-15 10:00:00',
        arrivalTime: '2024-08-15 14:00:00',
        seatNumber: 12,
        price: 12000.00,
        flightClass: 'Economy',
      );
      userlogic.insertFlight(
        countryFrom: 'UK',
        countryTo: 'MEXICO',
        departureTime: '2024-08-20 09:00:00',
        arrivalTime: '2024-08-20 13:00:00',
        seatNumber: 20,
        price: 9000.00,
        flightClass: 'Business',
      );
      userlogic.insertFlight(
        countryFrom: 'CUBA',
        countryTo: 'CANADA',
        departureTime: '2024-08-15 9:00:00',
        arrivalTime: '2024-08-15 5:00:00',
        seatNumber: 80,
        price: 40000.00
```

- **Purpose**: The primary purpose of this page is to allow the user to insert several predefined flight records into the database with a single button press.

- **Implementation**:

  - **BlocProvider**: The widget relies on the `Userlogic` class, which is managed by the Flutter Bloc library for state management. The `Userlogic` instance is accessed via `BlocProvider.of<Userlogic>(context)`.
  - **insertFlights Function**: This function is responsible for inserting five flight records with predefined data, such as departure and arrival times, seat numbers, prices, and flight classes.
    - For each flight, the `insertFlight` method from the `Userlogic` class is called with specific parameters like `countryFrom`, `countryTo`, `departureTime`, `arrivalTime`, `seatNumber`, `price`, and `flightClass`.

Homepage:

PeriodicUpdater:

- **Purpose**: The `PeriodicUpdater` class is responsible for periodically updating the list of flights by fetching new data every 7 seconds. It uses a `Timer` to trigger the update.

`Homepage`:

- **Purpose**: The `Homepage` widget serves as the main screen for searching flights. It allows users to select flight details such as the departure and arrival countries, dates, and class, and then search for available flights.
- **Features**:
  - **Periodic Updates**: When the page is built, `PeriodicUpdater.startUpdating(userlogic)` is called to start the periodic fetching of flight data.
  - **State Management**: It uses `BlocBuilder` to build the UI based on the current state of `Userlogic`. If the state is `getflight`, the flight data is processed and displayed; otherwise, a loading spinner is shown.
  - **Dropdown Menus**: The UI includes dropdowns for selecting the departure and arrival countries, as well as the flight class. These are populated dynamically based on the fetched flight data.
  - **Date Pickers**: Date pickers are provided to select the departure and arrival dates.
  - **Search Button**: A button labeled "Search Flights" navigates to the `FlightsListScreen` when pressed, where users can view the filtered list of flights.

final_project_iti · Version control · Pixel Fold API 35 (mobile) · main.dart

ghts.dart | confirmation.dart | airportInformation.dart | user_logic.dart | Insertionpage.dart | Insertion.dart | homepage.dart | user_state.dart | paymentpag

```dart
import 'dart:async';

import 'package:final_project_iti/pages/listOfFlights.dart';
import 'package:final_project_iti/user_logic.dart';
import 'package:final_project_iti/user_state.dart';
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';



class PeriodicUpdater {
  static Timer? _timer;

  static void startUpdating(Userlogic userlogic) {
    _timer?.cancel();
    _timer = Timer.periodic(Duration(seconds: 7), (timer) {
      userlogic.fetchFlights().then((value) {
        userlogic.flights = value;
        userlogic.emit(getflight()); // Ensure this is the correct method
      });
    });  // Timer.periodic
  }

  static void stopUpdating() {
    _timer?.cancel();
  }
}

class Homepage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    final userlogic = BlocProvider.of<Userlogic>(context);
```

final_project_ITI > lib > homepage.dart · 217:45 · CRLF · UTF-8 · 2 spaces

Athletics Final result · Search · ENG · 9:49 AM 8/9/2024

final_project_iti · Version control · Pixel Fold API 35 (mobile) · main.dart

ghts.dart | confirmation.dart | airportInformation.dart | user_logic.dart | Insertionpage.dart | Insertion.dart | homepage.dart | user_state.dart | paymentpag

```dart
  }

  Widget _buildLocationDropdown(
      BuildContext context,
      String label,
      String? selectedValue,
      ValueChanged<String?> onChanged,
      List<String> options) {
    return Column(
      crossAxisAlignment: CrossAxisAlignment.center,
      children: [
        Text(
          label,
          style: TextStyle(color: Colors.white, fontSize: 16),
        ),  // Text
        DropdownButton<String>(
          value: options.contains(selectedValue)
              ? selectedValue
              : null, // Ensure the value is in the options list
          dropdownColor: Colors.blue[900],
          iconEnabledColor: Colors.white,
          style: TextStyle(
              color: Colors.white, fontSize: 24, fontWeight: FontWeight.bold),  // TextStyle
          onChanged: onChanged,
          items: options.map<DropdownMenuItem<String>>((String value) {
            return DropdownMenuItem<String>(
              value: value,
              child: Text(value),
            );  // DropdownMenuItem
          }).toList(),
        ),  // DropdownButton
      ],
```

final_project_ITI > lib > homepage.dart · 217:45 · CRLF · UTF-8 · 2 spaces

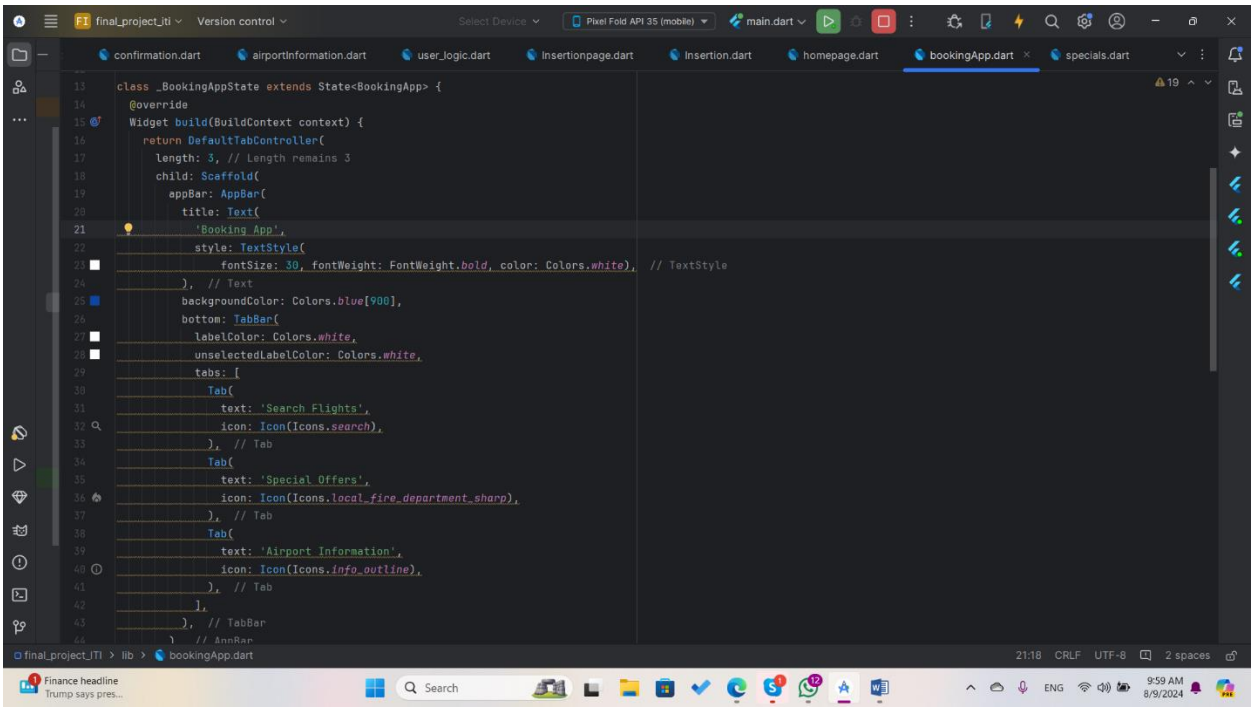
Athletics Final result · Search · ENG · 9:49 AM 8/9/2024

```dart
Widget _buildDatePicker(BuildContext context, String label,
    DateTime selectedDate, Function(DateTime?) onDateSelected) {
  return Row(
    mainAxisAlignment: MainAxisAlignment.spaceBetween,
    children: [
      Text(
        label,
        style: TextStyle(fontSize: 16, fontWeight: FontWeight.bold),
      ),  // Text
      TextButton(
        onPressed: () async {
          DateTime? pickedDate = await showDatePicker(
            context: context,
            initialDate: selectedDate,
            firstDate: DateTime.now(),
            lastDate: DateTime(2101),
          );
          onDateSelected(pickedDate);
        },
        child: Text(
          "${selectedDate.day}/${selectedDate.month}/${selectedDate.year}",
          style: TextStyle(fontSize: 16),
        ),  // Text
      ),  // TextButton
    ],
  );  // Row
}

Widget _buildDropdownMenu(BuildContext context, List<String> options) {
  return Row(
    mainAxisAlignment: MainAxisAlignment.spaceBetween,
    children: [
```

```dart
        style: TextStyle(fontSize: 16),
      ),  // Text
    ),  // TextButton
  ],
);  // Row
}

Widget _buildDropdownMenu(BuildContext context, List<String> options) {
  return Row(
    mainAxisAlignment: MainAxisAlignment.spaceBetween,
    children: [
      Text(
        'CLASS',
        style: TextStyle(fontSize: 16, fontWeight: FontWeight.bold),
      ),  // Text
      DropdownButton<String>(
        value: options.isNotEmpty
            ? options.first
            : null, // Ensure a default value is set if options are not empty
        onChanged: (String? newValue) {
          // Handle class change
        },
        items: options.map<DropdownMenuItem<String>>((String value) {
          return DropdownMenuItem<String>(
            value: value,
            child: Text(value),
          );  // DropdownMenuItem
        }).toList(),
      ),  // DropdownButton
    ],
  );  // Row
}
```

BookingApp Class:

The `BookingApp` class is a `StatefulWidget` that provides the main interface for a flight booking application. It includes a tab-based navigation system with three primary sections: `Search Flights`, `Special Offers`, and `Airport Information`. It also features a navigation drawer with options like "Payment," "About us," and "Log Out."



Specials Widget:

The `Specials` class is a `StatelessWidget` that displays special flight offers. It integrates with the BLoC pattern to fetch and display flight data, using `BlocBuilder` and `FutureBuilder` to handle asynchronous data fetching and state management.

`buildQuerySection`:

- **Purpose**: Creates sections for different types of flight queries (e.g., flights under 15000, business class flights).
- **Parameters**:
  - `queryName`: The title for the section (e.g., "Flights Under 15000").
  - `flightsFuture`: A `Future` that resolves to a list of flight

```dart
class Specials extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    final userlogic = BlocProvider.of<Userlogic>(context);

    return Scaffold(
      appBar: AppBar(
        title: Text('Upcoming Flights'),
        backgroundColor: Colors.blue[900],
      ),  // AppBar
      body: BlocBuilder<Userlogic, Userstate>(
        builder: (context, state) {
          if (state is getflight) {
            return SingleChildScrollView(
              child: Column(
                children: [
                  _buildQuerySection(
                    'Flights Under 15000',
                    userlogic.fetchFlightsUnder15000(),
                  ),
                  _buildQuerySection(
                    'Business Class Flights',
                    userlogic.fetchBusinessClassFlights(),
                  ),
                ],
              ),  // Column
            );  // SingleChildScrollView
          }
          return Center(child: CircularProgressIndicator());
        },
      ),  // BlocBuilder
```

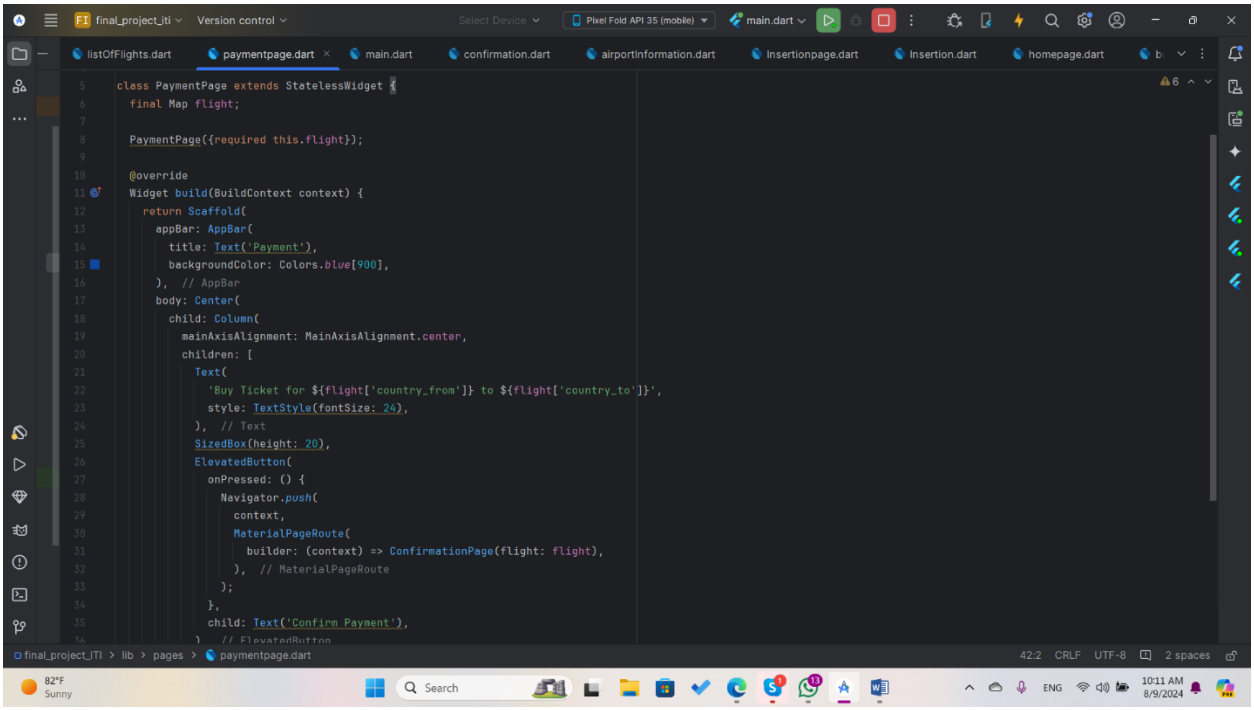
FlightsListScreen Widget:

The `FlightsListScreen` class is a `StatelessWidget` that displays a list of available flights and allows users to navigate to a payment page for purchasing tickets. It uses the BLoC pattern to manage state and handle flight data.

```dart
class FlightsListScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    final userlogic = BlocProvider.of<Userlogic>(context);

    return Scaffold(
      appBar: AppBar(
        title: Text('Available Flights'),
        backgroundColor: Colors.blue[900],
      ),  // AppBar
      body: BlocBuilder<Userlogic, Userstate>(
        builder: (context, state) {
          if (state is getflight) {
            final flights = userlogic.flights;

            return ListView.builder(
              itemCount: flights.length,
              itemBuilder: (context, index) {
                final flight = flights[index];
                return ListTile(
                  title: Text(
                      '${flight['country_from']} to ${flight['country_to']}'),  // Text
                  subtitle: Text(
                      'Class: ${flight['class']} - Price: ${flight['price']}'),  // Text
                  trailing: ElevatedButton(
                    onPressed: () {
                      Navigator.push(
                        context,
                        MaterialPageRoute(
                          builder: (context) => PaymentPage(flight: flight),
                        ),  // MaterialPageRoute
```

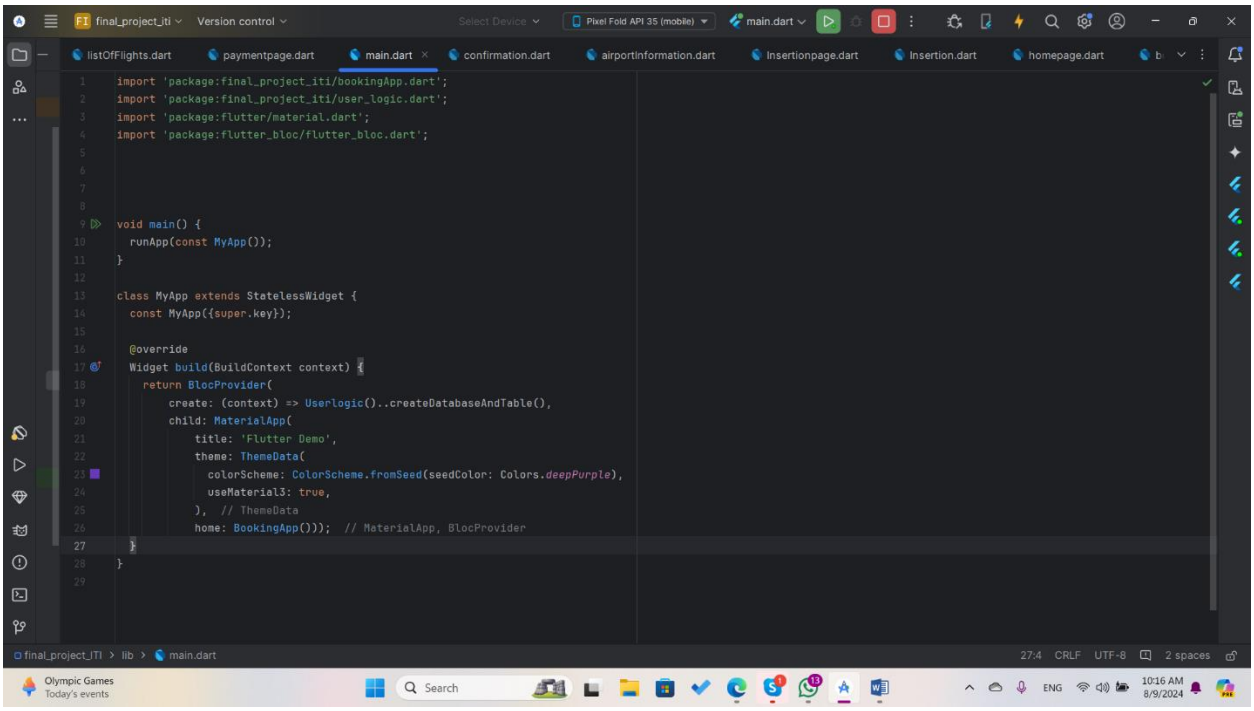
PaymentPage Widget:

The `PaymentPage` class is a `StatelessWidget` that provides a user interface for confirming payment for a flight. It includes a title, flight details, and a button to proceed to the confirmation page.



Main:

1. **main** Function:
    - Launches the app by calling `runApp` and passing the `MyApp` widget.
2. **`MyApp` Class**:
    - Sets up the app's main settings.
    - Uses `BlocProvider` to provide `Userlogic` throughout the app.
    - Defines the app's theme.
    - Sets `BookingApp` as the starting screen of the app.



EXTRAS:

- **Queries.**
- **Dropdown Menu for ‘Country_from’.**
- **Dropdown Menu for ‘Country_to’.**
- **Dropdown Menu of Calender.**

Document Prepared by:

- **Jayan Ahmed Samer**
- **Omar Sameh Abdelaziz Elsheikh**
- **Zeina Amr Mohsen**

Submitted to:

Eng. Muhammed Shushan