

Airline Reservation System



Project Supervisor

Sir Laeeq Khan Niazi

Submitted By

Uswa Arif

2021-CS-77

Department of Computer Science
University of Engineering and Technology, Lahore
Pakistan

List of Figures

2.1	Class Diagram (part 1)	3
2.2	Class Diagram (part 2)	4
2.3	Use Case Diagram	5
2.4	Sequence Diagram of Login Process	6
2.5	Sequence Diagram of Admin Side	6
2.6	Sequence Diagram of Customer Side	7
2.7	Sequence Diagram of sign up process	7
2.8	Level 0 Data Flow Diagram	8
2.9	Level 1 Data Flow Diagram	9
2.10	Three-tier Architecture Diagram	10
3.1	Tasks created in Click Up	12
3.2	Use of Variables to the code	12
3.3	Adding comments to the code	13
3.4	Communication with the client	13
3.5	Code uploaded to the Github	14
3.6	Firebase database schema	14
4.1	Login Screen	15
4.2	Forget Password Screen	18
4.3	Add Baggage Screen	20
4.4	Hotel Recommendation Screen	22
4.5	View Flight Screen	25

List of Tables

3.1	Tasks and their sub-tasks created in Click up	11
-----	---	----

Contents

List of Figures	ii
List of Tables	iii
Table of Contents	iv
1 Introduction	1
1.1 Project Features	2
2 Diagrams	3
2.1 Class Diagram	3
2.2 Use Case Diagram	4
2.3 Sequence Diagram	5
2.4 DFD Level 0 Diagram	8
2.5 Architecture Diagram	9
3 Project Management Tools	11
3.1 PM tool	11
3.2 Coding Best Practice	12
3.3 Communication Tool	13
3.4 Version Control System	13
3.5 Front End Technology	14
3.6 Back End Technology	14
3.7 Database Schema	14
4 Application Working	15
4.1 Login Screen	15
4.2 Forget Password Screen	18
4.3 Add Baggage Screen	20
4.4 Hotel Recommendation Screen	22
4.5 View Flight Screen	25

Chapter 1

Introduction

The Airline Reservation System is a comprehensive and advanced platform designed to simplify and elevate the flight booking process specifically for travelers commuting between Pakistan and Manchester. This application aims to offer an intuitive, user-centric experience while providing an array of functionalities to facilitate seamless travel arrangements. At its core, the application enables users to register with authenticated credentials, ensuring a secure and reliable user base. Registration within the application is a streamlined process, ensuring security and reliability by allowing users to create accounts using verified credentials. This step sets the stage for a personalized and hassle-free experience throughout their interactions with the platform.

Once registered, users are greeted with a highly intuitive interface that provides extensive search functionalities, enabling seamless exploration and selection of available flights. Whether booking for individual travel or group trips of up to four tickets in a single transaction, the platform caters to diverse travel needs efficiently and effectively.

The system intelligently analyzes user data to suggest relevant discounts, enhancing the affordability of flight bookings. Users can also cancel their booked tickets with ease, subject to a specified timeframe and a corresponding cancellation fee, ensuring flexibility while adhering to policies. Admins have the flexibility to manage flight stops, set the number of stops per flight, and provide hotel recommendations during layovers, enhancing the overall travel experience.

Additionally, the platform maintains a comprehensive booking history accessible to users and admin for reference and reporting purposes. It also offers seat selection options, multiple payment methods, and the ability for users to request special assistance during the booking process, catering to diverse traveler needs.

Furthermore, the application implements a loyalty program to reward frequent flyers and incorporates a feedback system to collect user input post-flight, allowing continual improvements in service quality. Essential information regarding baggage allowances, restrictions, and fees is readily available, ensuring transparency.

The Airline Reservation System boasts an array of secure and versatile payment methods, empowering users with a seamless and trusted transactional experience. Users can opt for credit or debit card payments, facilitating hassle-free transactions through major card networks. This feature ensures flexibility and accessibility for a wide range of users. Additionally, the platform accommodates bank transfers, providing a direct and secure mode of payment, especially favored by individuals seeking a straightforward transfer process. Furthermore, the integration of digital wallets elevates the payment options, allowing users to leverage popular digital payment services for swift and secure transactions.

The Airline Reservation System adheres to stringent security standards, ensuring data privacy and compliance. Its reliability ensures round-the-clock availability with minimal downtime for maintenance. Scalability remains a priority, allowing seamless expansion to accommodate growing user numbers and flight options. The application prioritizes a user-friendly interface, aiming to cater to both tech-savvy and non-tech-savvy users, ensuring a

seamless and enjoyable travel booking experience for all.

The application's interface is designed for simplicity and ease of use, offering an intuitive flow from flight selection to payment completion. Users can navigate through the app effortlessly, ensuring a seamless booking experience that prioritizes convenience and efficiency throughout their journey planning.

1.1 Project Features

The following are the key features of this application:

- **User Registration:**
Allows new users to create accounts with accurate and validated credentials, ensuring the authenticity of user information, including email and password.
- **Ticket Booking:**
Empowers users to search for available flights based on their preferences and book tickets effortlessly.
- **Discount Suggestions:**
Analyzes user data and preferences to offer personalized discount suggestions during the flight booking process, enhancing the affordability of travel for users.
- **Ticket Cancellation:**
Enables users to cancel their booked tickets up to three days before the scheduled flight.
- **Flight Stops Information:**
Allows administrators to manage and set the number of stops for each flight, providing transparency to users about the flight itinerary.
- **Hotel Recommendations:**
Admin can add and maintain a list of hotel recommendations for passengers during flight stops, offering convenience and accommodation options.
- **Car Rental Recommendations:**
Similarly, the admin can create and maintain a list of recommended car rental services, offering passengers the option to easily find and rent cars for transportation needs during their stay or stopovers. This service complements the travel experience by providing added flexibility and convenience in commuting to and from airports or within cities.
- **Booking History:**
Provides users and administrators access to their complete booking history, allowing them to track and review past reservations for reference and reporting purposes.
- **Seat Selection:**
Permits users to select their preferred seats during the booking process, subject to availability, enhancing their comfort and choice on the flight.
- **Multiple Payment Options:**
Offers a diverse range of payment methods including credit/debit cards, bank transfers, and digital wallets, ensuring flexibility and convenience for users.
- **Feedback System:**
Collects user feedback after flights to gain insights and improve services, fostering a responsive and customer-centric approach.
- **Baggage Information:**
Provides comprehensive guidelines regarding baggage allowances, restrictions, and associated fees, ensuring travelers are well-informed about their luggage..

Chapter 2

Diagrams

2.1 Class Diagram

A class diagram in software engineering is a type of UML (Unified Modeling Language) diagram that illustrates the structure and relationships within a system by visualizing the classes, their attributes, methods, and the associations among them.

The main components of Class Diagram are:

- **Class:** A class contains attributes (variables) and methods (functions or operations) that define the behavior and characteristics of the objects created from that class.
- **Attributes:** Attributes are the data members or variables within a class.
- **Operations:** Methods or operations define the behavior of a class. They represent the actions that objects of the class can perform.

The class diagram of the Airline Reservation System project is as follows:

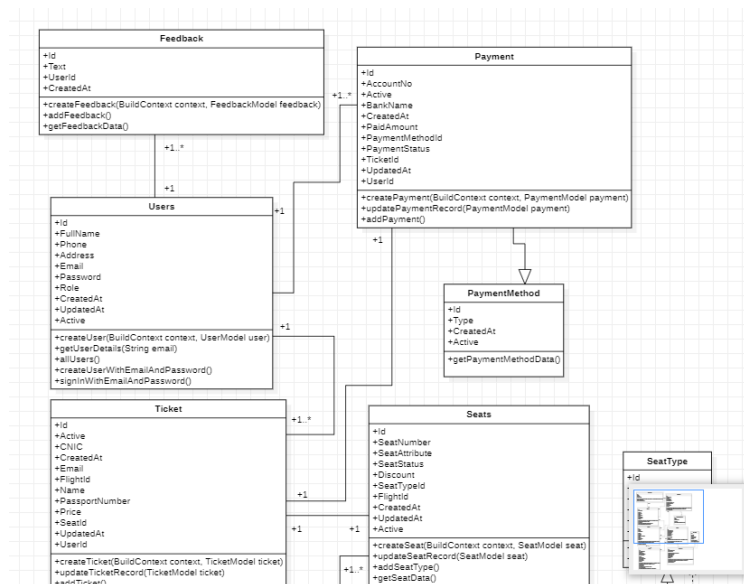


Figure 2.1: Class Diagram (part 1)

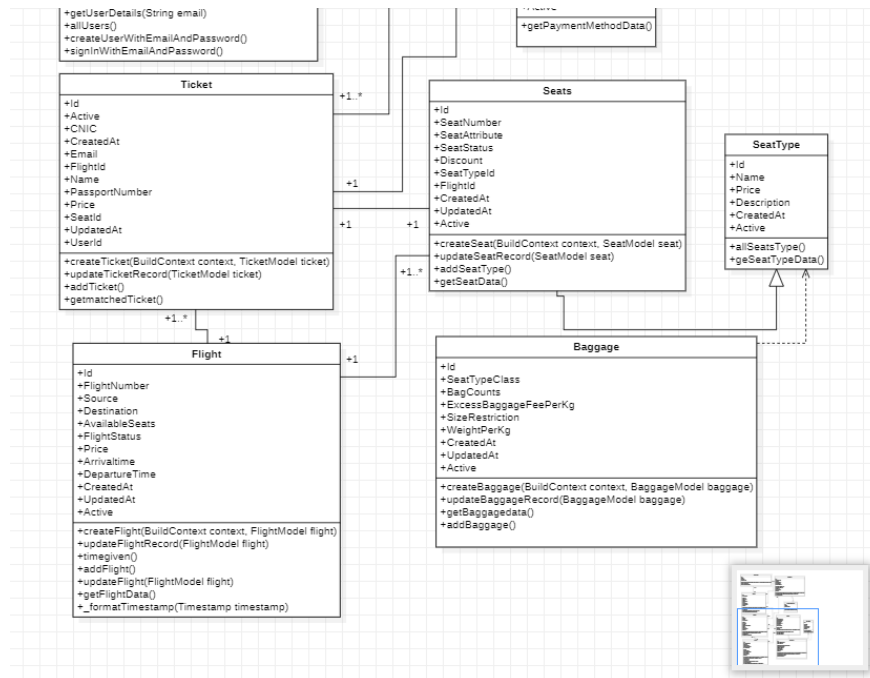


Figure 2.2: Class Diagram (part 2)

2.2 Use Case Diagram

A use case in software engineering is a type of UML (Unified Modeling Language) diagram that give the details of your system's users (also known as actors) and their interactions with the system. A use case diagram shows various use cases and different types of users the system has.

The main components of Use Case Diagrams are:

- **Actors:** An actor is an external entity that interacts with the system to perform some actions.
- **Use Case:** A use case represents a task that the system performs to achieve the goals of the actor.
- **Include Relationship:** Include relationships denote that one use case includes another use case. It means that if an action is performed then the use case that has include relationship must also be performed.
- **Extend Relationship:** Extend relationships denote that one use case can extend another use case by adding extra options. It means that if an action is performed then the use case that has extend relationship can also perform that action as an extra feature.
- **System Boundary:** The boundary box encloses all the use cases of the system.
- **Association:** Associations or lines between actors and use cases show that an actor interacts with the system to perform a particular use case.

The use case diagram of the Airline Reservation System project is as follows:

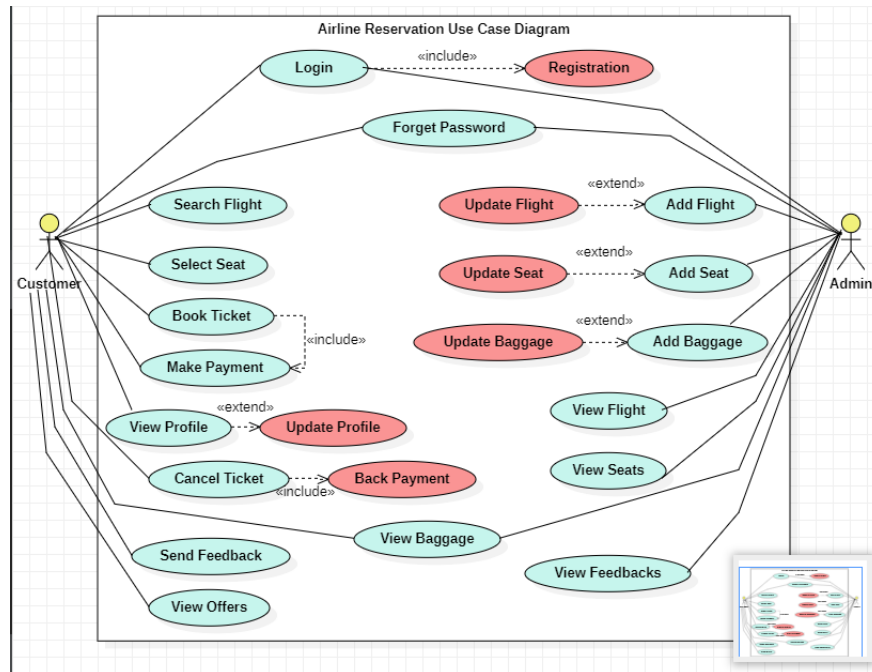


Figure 2.3: Use Case Diagram

2.3 Sequence Diagram

A sequence diagram in software engineering is a type of UML (Unified Modeling Language) diagram that gives the detail how operations are done. It defines that how the functions are performed. It gives the interaction between the actors and the actions they performs by showing the messages. The main components of Sequence Diagrams are:

- **Lifeline:** A lifeline is an object that gives the sequence of interactions. It is represented by vertical dashed lines and with the name of the functionality.
- **Activation Bar:** An activation bar is represented by a horizontal line extending from a lifeline, shows the period during which an object is active or processing a message.
- **Messages:** It is represented by a solid line with a solid arrowhead. This symbol is used when a user sends message and then wait for the response.
- **Return Messages:** It is represented by a dotted line with an arrowhead. Return messages represent the response or return communication from an object to the user to a received message.

The sequence diagrams of the Airline Reservation System project is as follows:

- **Login Sequence Diagram**

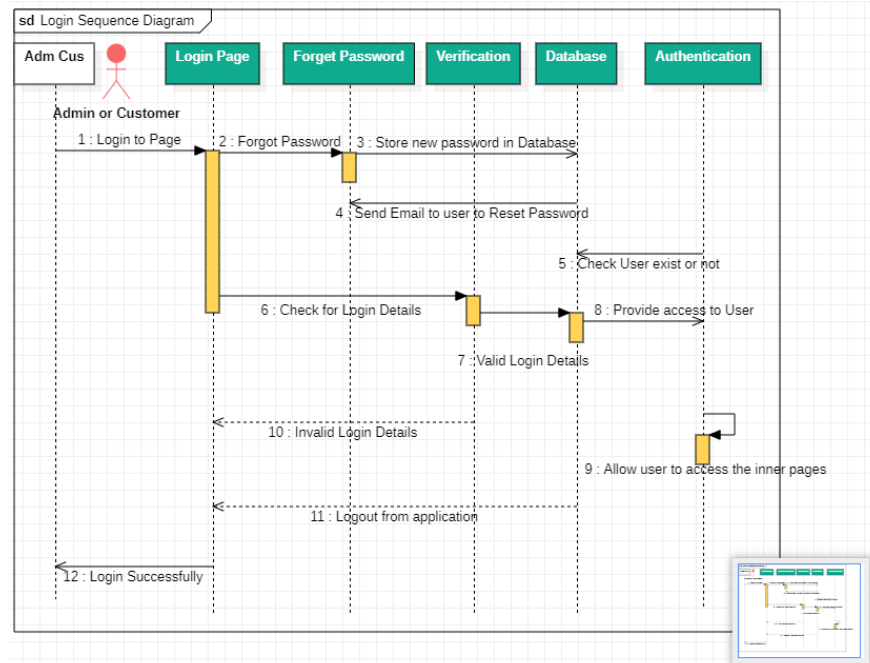


Figure 2.4: Sequence Diagram of Login Process

• Admin Side Sequence Diagram

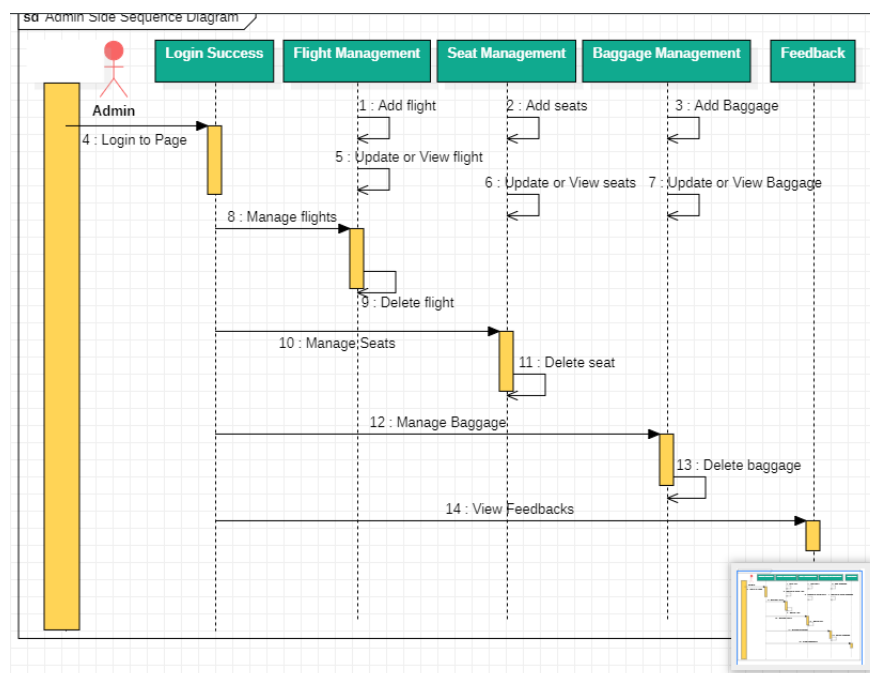


Figure 2.5: Sequence Diagram of Admin Side

- Customer Side Sequence Diagram

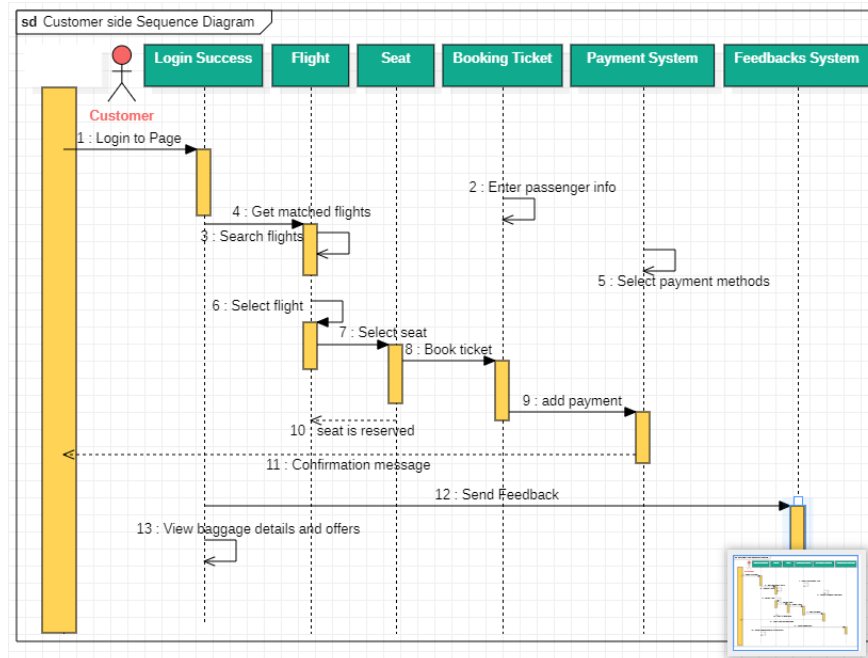


Figure 2.6: Sequence Diagram of Customer Side

- Sign Up Process Sequence Diagram

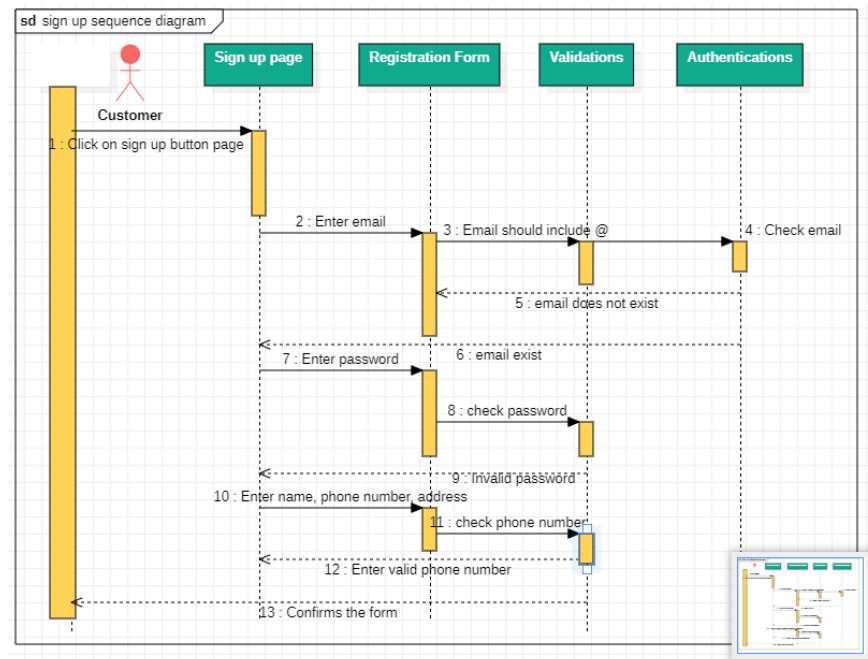


Figure 2.7: Sequence Diagram of sign up process

2.4 DFD Level 0 Diagram

A Data Flow Diagrams (DFD) in software engineering is a type of diagram that describes the flow of data within the system. DFDs are graphical representations of a system. DFDs are useful for understanding the processes, and interactions between different components in the system. They also define show data moves or flows in the system.

DFDs can be divided into different levels. Some of the levels are given below.

- **Level 0 DFD:** This is the highest-level DFD, which provides an overview of the entire system. It shows processes, data flows, and how the data is stored in the system. This level do not give details about how the internal work is processing.

The Level 0 data flow diagram of the Airline Reservation System project is as follows:

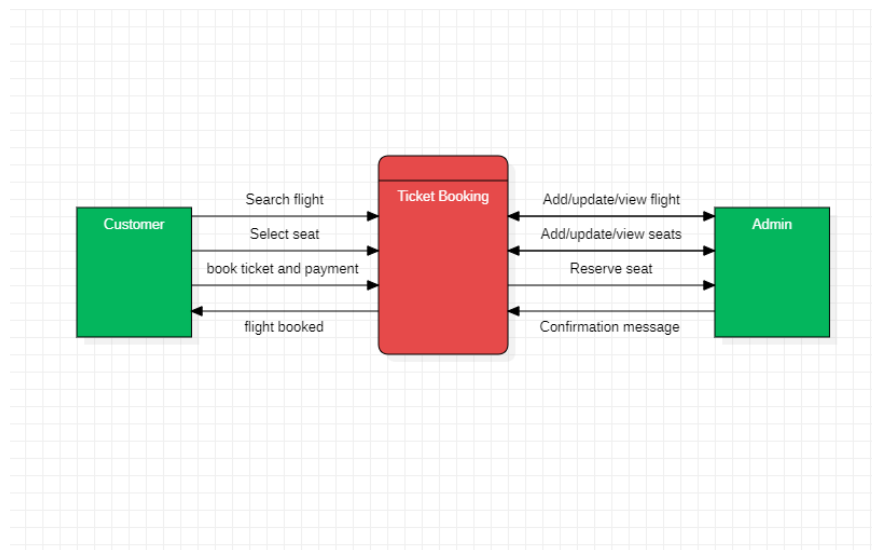


Figure 2.8: Level 0 Data Flow Diagram

- **Level 1 DFD:** This level provides a more detailed view of the system. It breakdown the major processes that are defined in the level 0 DFD into many small processes. Each sub-process is works as a separate process in the level 1 DFD. The data flows and data stores are also shown with each sub-processes. A level 1 data flow diagram highlights main functions of a system.

The Level 0 data flow diagram of the Airline Reservation System project is as follows:

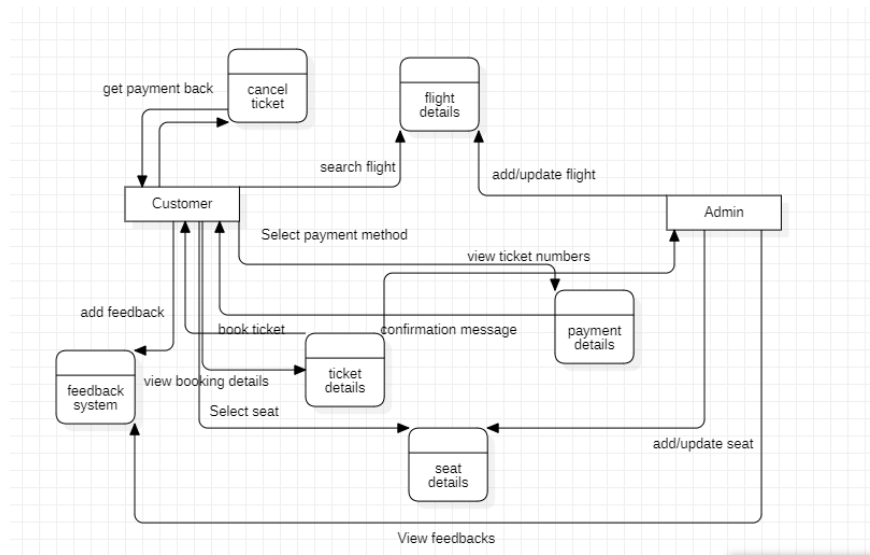


Figure 2.9: Level 1 Data Flow Diagram

2.5 Architecture Diagram

An Architecture Diagram is a graphical representation used in software engineering and system design to describe the structure, components, relationships, and interactions within a system.

The architecture of the Airline Reservation System is based on the three-tier architecture.

This three-tier architecture mainly consists of three layers namely:

- Presentation Tier
- Business Logic Tier
- Data Access Tier

Presentation Tier:

The Presentation Tier is the user interface and communication layer of the application, where the user interacts with the application. This tier displays information related to services such as browsing the Airline website, purchasing tickets etc. The presentation tier of the Airline Reservation System is mainly formed by the **flutter and dart**. Each screen is well-designed and user-friendly.

Business Logic Tier:

The Business Logic Tier is the middle tier of the three-tier architecture. The Business Logic Tier collects the information in the presentation tier and then it is processed. The presentation tier can also add, delete or modify data in the data tier. The business logic tier of airline reservation system is typically developed using Dart, and communicates with the data tier using Firebase. The Business Logic tier is mainly responsible for information exchange between the user interface and the database of the project. The language used in this system for business logic is **Dart**.

Data Access Tier:

The data tier is also called database tier, data access tier or back-end, is used when the information processed by the application is stored and managed. In airline reservation system, the data access tier is **Cloud Firebase**. The information related to the Airline Reservation System is stored and retrieved from here.

The three-tier architecture diagrams of the Airline Reservation System project is as follows:

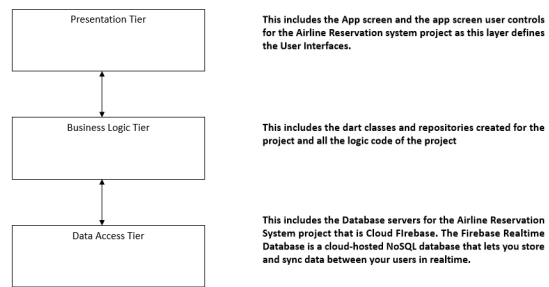


Figure 2.10: Three-tier Architecture Diagram

Chapter 3

Project Management Tools

3.1 PM tool

For the project management, I have used the tool **Click Up**. ClickUp facilitated various aspects of project management, collaboration, and task tracking for the development team.

The tasks, I have created in the Clickup are:

Table 3.1: Tasks and their sub-tasks created in Click up

Tasks	Sub Tasks
User Registration	1- Sign Up 2- Sign In 3- Forget Password 4- Unit Testing
Flight Management	1- Add Flight 2- Delete Flight 3- Update Flight 4- Unit Testing
Seat Management	1- Seat Type 2- Add Seats 3- Update Seats 4- Search Seat
Hotel Recommendation	1- Hotel Deals 2- Car Rentals
Baggage Information	1- Baggage Weight and Allowance 2- Update Baggage details 3- Rules and Regulations for Baggage 4- View Baggage
Customer Side Management	1- Flight Data 2- Seat Management
Ticket Management	1- Ticket Booking 2- Ticket Cancellation 3- Discount suggestion 4- History
Feedback System	
Payment Management	1- Add payment 2- Tax calculated

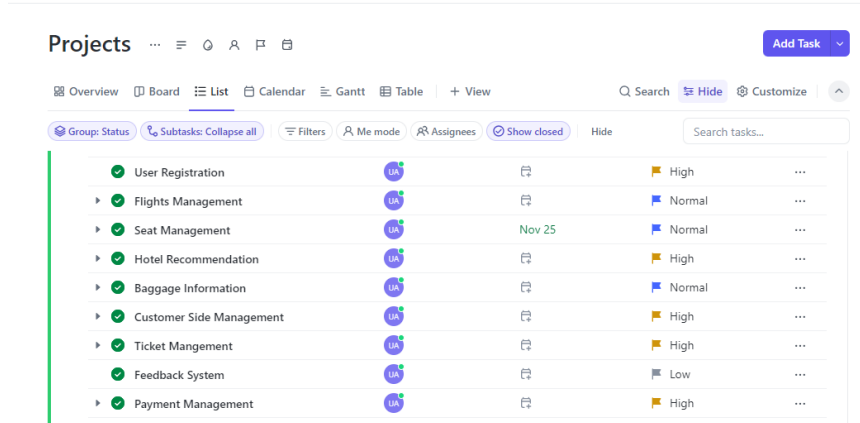


Figure 3.1: Tasks created in Click Up

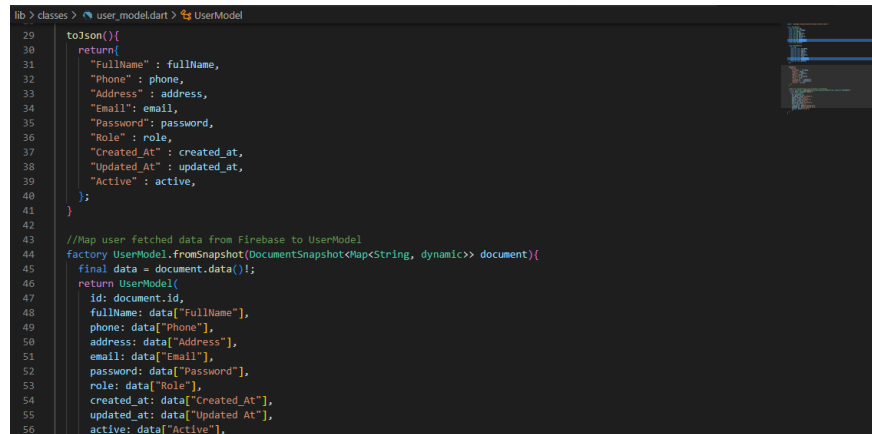
3.2 Coding Best Practice

The Coding Best Practice, I have used in the project Airline Reservation system are:

- Consistent Naming Conventions
- Code Structure like using three-tier architecture separately
- Adding comments
- Use of Variables
- Readable code

```
lib > classes > user_model.dart > UserModel
1  import 'package:cloud_firestore/cloud_firestore.dart';
2
3  class UserModel{
4    final String? id;
5    final String fullName;
6    final int phone;
7    final String address;
8    final String email;
9    final String password;
10   final String role;
11   final String created_at;
12   final String updated_at;
13   final bool active;
14
15   const UserModel({
16     this.id,
17     required this.fullName,
18     required this.phone,
19     required this.address,
20     required this.email,
21     required this.password,
22     required this.role,
23     required this.created_at,
24     required this.updated_at,
25     required this.active,
26   });
27
28
```

Figure 3.2: Use of Variables to the code



```

lib > classes > user_model.dart > UserModel
29
30 toJson(){
31   return{
32     "FullName" : fullName,
33     "Phone" : phone,
34     "Address" : address,
35     "Email": email,
36     "Password": password,
37     "Role" : role,
38     "Created_At" : created_at,
39     "Updated_At" : updated_at,
40     "Active" : active,
41   };
42 }
43
44 //Map user fetched data from Firebase to UserModel
45 factory UserModel.fromSnapshot(DocumentSnapshot<Map<String, dynamic>> document){
46   final data = document.data();
47   return UserModel(
48     id: document.id,
49     fullName: data["FullName"],
50     phone: data["Phone"],
51     address: data["Address"],
52     email: data["Email"],
53     password: data["Password"],
54     role: data["Role"],
55     created_at: data["Created_At"],
56     updated_at: data["Updated_At"],
57     active: data["Active"],
58   );
59 }

```

Figure 3.3: Adding comments to the code

3.3 Communication Tool

The communication tools, I have used for the conversation with the client is **Slack**. Slack served as a central platform for real-time communication and collaboration between the development team and the client. The communication months are September, October and November.

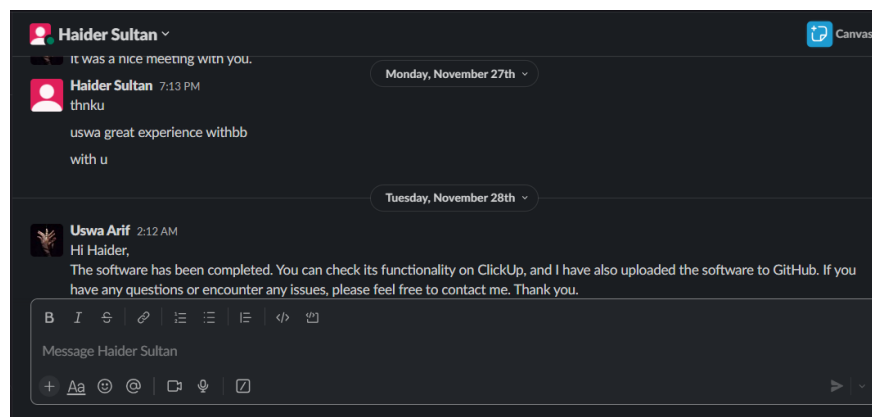


Figure 3.4: Communication with the client

3.4 Version Control System

For Version Control System, I used GitHub to keep track of my project's code. I made 22 commits to the code in September, October, and November. Each time I made a change, I wrote a note to explain what I did, so it was clear to others. I have created two branches and uploaded all the code to the branch. Also, the commits message are correct as required.

I put all my code on GitHub so everyone working on the project could see it. GitHub also kept a record of all the changes I made, so I could go back if I needed to see how things were before.

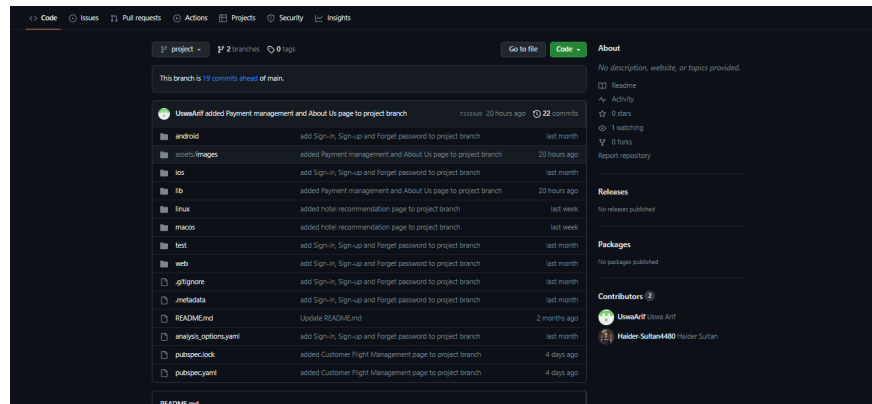


Figure 3.5: Code uploaded to the Github

3.5 Front End Technology

The front end technology stack, I have used in the project is:

- flutter (software development kit)
- dart (programming language)

3.6 Back End Technology

The back-end technology is done by Model, View and Controllers. For the BL, I have created Repositories. The back end technology stack, I have used in the project is:

- dart (programming language)

3.7 Database Schema

For database storage, I have used Cloud Firestore. The schema is created as:

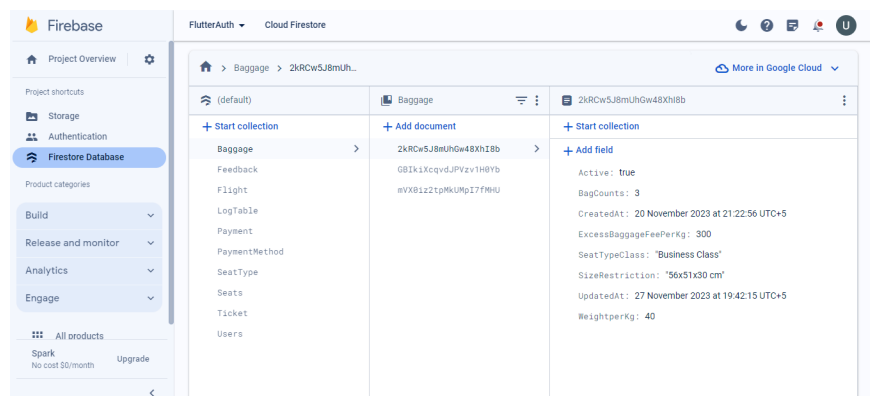


Figure 3.6: Firebase database schema

Chapter 4

Application Working

The Application Working screenshots with some code snippets are given below:

4.1 Login Screen

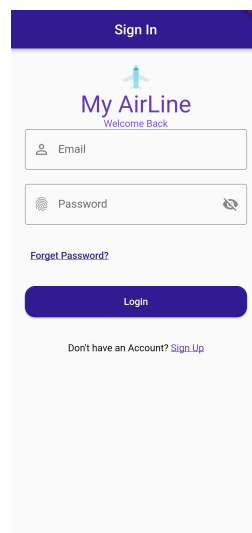


Figure 4.1: Login Screen

Code:

```
1  @override
2  Widget build(BuildContext context) {
3    return Scaffold(
4      body: ListView(
5        children: [
6          Container(
7            padding: const EdgeInsets.all(8.0),
8            child: Row(
9              children: [
10               IconButton(
```

```

11         icon: const Icon(Icons.close),
12         onPressed: () {
13             Navigator.pop(context);
14         },
15     ),
16 ],
17 ),
18 ),
19 const SizedBox(height: 16),
20 Align(
21     alignment: Alignment.centerLeft, // Align "View Flight" text to the
22     left
23     child: Row(
24         mainAxisAlignment: MainAxisAlignment.spaceBetween,
25         children: [
26             const Padding(
27                 padding: EdgeInsets.only(left: 16.0), // Margin from the left
28                 side
29                 child: Text(
30                     "View Flight",
31                     style: TextStyle(
32                         fontSize: 26,
33                         fontWeight: FontWeight.bold,
34                         color: Colors.deepPurple,
35                     ),
36                 ),
37             Padding(
38                 padding: const EdgeInsets.only(right: 16.0), // Margin from the
39                 right side
40                 child: Column(
41                     children: [
42                         const Text(
43                             "Add Flight",
44                             style: TextStyle(
45                                 fontSize: 16,
46                                 color: Colors.deepPurple,
47                             ),
48                         ),
49                         Material(
50                             color: Colors.deepPurple,
51                             shape: RoundedRectangleBorder(
52                                 borderRadius: BorderRadius.circular(5.0),
53                             ),
54                             child: InkWell(
55                                 onTap: () {
56                                     Navigator.push(
57                                         context,
58                                         MaterialPageRoute(builder: (context) => const
59                                             AddFlight()),
60                                     );
61                                 },
62                                 child: Container(
63                                     width: 40,
64                                     height: 40,
65                                     child: const Center(

```

```

64         child: Text(
65             "+",
66             style: TextStyle(
67                 color: Colors.white,
68                 fontSize: 18,
69                 fontWeight: FontWeight.bold,
70             ),
71         ),
72     ),
73 ),
74 ),
75 ),
76 ],
77 ),
78 ),
79 ],
80 ),
81 ),
82
83 const SizedBox(height: 12),
84 // Curved top shape SizedBox
85 ClipRRect(
86     borderRadius: const BorderRadius.only(
87         topLeft: Radius.circular(20.0),
88         topRight: Radius.circular(20.0),
89     ),
90
91     child: Container(
92         decoration: BoxDecoration(
93             color: Colors.deepPurpleAccent, // Background color
94             border: Border.all(color: Colors.deepPurple, width: 2.0), // Add
95                 a border
96         ),
97         const SizedBox(height: 10),
98         const Row(
99             mainAxisAlignment: MainAxisAlignment.spaceBetween,
100             children: [
101                 Text("Departure Time",
102                     style: TextStyle(
103                         fontSize: 10,
104                     ),
105                 ),
106                 Text("Arrival Time",
107                     style: TextStyle(
108                         fontSize: 10,
109                     ),
110                 ),
111             ],
112         ),
113         const SizedBox(height: 2),
114         Row(
115             mainAxisAlignment: MainAxisAlignment.spaceBetween,
116             children: [
117                 Text(_formatTimestamp(flightData['DepartureTime']),
118                     style: const TextStyle(
119                         fontSize: 10,

```

```

120         ),
121       ),
122       Text(_formatTimestamp(flightData['ArrivalTime']),
123         style: const TextStyle(
124           fontSize: 10,
125         ),
126       ),
127     ],
128   ),
129 ),
130 ),
131 ),
132 ],
133 ),
134 );
135 }

```

Listing 4.1: Dart code snippet

4.2 Forget Password Screen

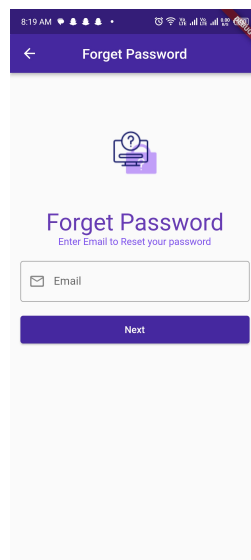


Figure 4.2: Forget Password Screen

Code:

```

1 @override
2 Widget build(BuildContext context) {
3   final size = MediaQuery.of(context).size;    //Responsive
4   return Scaffold(
5     appBar: AppBar(
6       centerTitle: true,
7       title: const Text("Forget Password"),
8     ),
9     body: SingleChildScrollView(

```

```

10   child: Container(
11     padding: const EdgeInsets.all(15.0),
12     child: Column(
13       crossAxisAlignment: CrossAxisAlignment.center,
14       children: [
15         const SizedBox(height: 20,),
16         Image(image: AssetImage('assets/images/forgetPassword.png'), height:
           size.height*0.2,),
17         Text("Forget Password", style: Theme.of(context).textTheme.headline4?.
           copyWith(color: Colors.deepPurple),),
18         Text("Enter Email to Reset your password", style: Theme.of(context).
           textTheme.bodyText2?.copyWith(color: Colors.deepPurpleAccent),),
19
20         const SizedBox(height: 20.0,),
21         Form(child: Column(
22           children: [
23             TextFormField(
24               controller: emailController,
25               decoration: const InputDecoration(
26                 label: Text("Email"),
27                 hintText: "Enter your Email",
28                 prefixIcon: Icon(Icons.mail_outline_rounded),
29                 border: OutlineInputBorder(),
30             ),
31           ),
32           const SizedBox(height: 20.0,),
33           SizedBox(width: double.infinity,
34             height: 40,
35             child: ElevatedButton(onPressed: (){
36               auth.sendPasswordResetEmail(email: emailController.text.toString
37                 ()).then((value){
38                 print("We have sent you an email to recover password, please
39                   check email.");
40               }).onError((error, stackTrace){
41                 print("Error sending password reset email.");
42
43                 ////////////////log table ///////////////////
44                 final log = LogTable(
45                   page_name: "sign_up_page",
46                   error: error.toString(),
47                 );
48                 final logRepository = LogTableRepository();
49                 // ignore: use_build_context_synchronously
50                 logRepository.createLog(context, log);
51               });
52             }, child: const Text("Next"),),
53           ),
54         ],
55       ),
56     ),
57   ),
58 ),
59 );
60 };
61 }

```

Listing 4.2: Dart code snippet

4.3 Add Baggage Screen

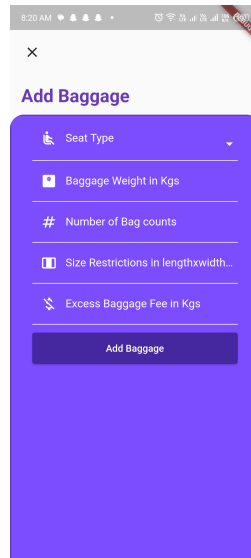


Figure 4.3: Add Baggage Screen

Code:

```

1  @override
2  void initState() {
3    super.initState();
4    // Call the function to get flight data from Firebase when the widget is
      initialized
5    getBaggagedata();
6  }
7
8  Future<void> getBaggagedata() async {
9    try {
10     QuerySnapshot querySnapshot = await FirebaseFirestore.instance
11       .collection("Baggage")
12       .get();
13
14     if (querySnapshot.docs.isNotEmpty) {
15       // Retrieve and store flight data in the list
16       BaggageDataList = querySnapshot.docs.map((baggageSnapshot) {
17         final data = baggageSnapshot.data() as Map<String, dynamic>;
18         final documentId = baggageSnapshot.id; // Get the document ID
19         data['documentId'] = documentId; // Add the document ID to the data
20         return data;
21       }).toList();
22       setState(() {});
23     } else {
24       print("No baggage data found in Firestore.");

```



```

25     }
26   } catch (e) {
27     print("An unexpected error occurred: $e");
28     final log = LogTable(
29       page_name: "add_baggage_page",
30       error: e.toString(),
31     );
32     final logRepository = LogTableRepository();
33     // ignore: use_build_context_synchronously
34     logRepository.createLog(context, log);
35   }
36 }
37
38 addBaggage() {
39   try {
40     setState(() {
41       isLoading = true;
42     });
43
44     // Add Baggage object using your class
45     final baggage = BaggageModel(
46       seat_type_class: selectedSeatType ?? "Economy",
47       weight_per_kg: int.parse(_weight.text),
48       bag_counts: int.parse(_bagcounts.text),
49       size_restriction: _sizerestriction.text,
50       excess_baggage_fee_per_kg: int.parse(_excessbaggagefee.text),
51       created_at: Timestamp.now(),
52       updated_at: Timestamp.now(),
53       active: true,
54     );
55
56
57
58     // Call the createFlight method from FlightRepository to store flight data
59     final baggageRepository = BaggageRepository();
60     baggageRepository.createBaggage(context, baggage);
61     print("Baggage added Successfully.");
62
63     setState(() {
64       isLoading = false;
65     });
66   }
67   catch (e) {
68     setState(() {
69       isLoading = false;
70       print("Not done");
71     });
72     print("here is error");
73     print(e);
74   }
75 }

```

Listing 4.3: Dart code snippet

4.4 Hotel Recommendation Screen

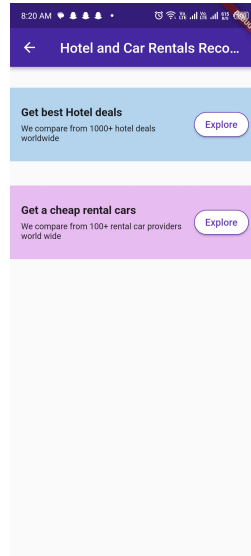


Figure 4.4: Hotel Recommendation Screen

Code:

```

1  @override
2  Widget build(BuildContext context) {
3    return Scaffold(
4      appBar: AppBar(
5        centerTitle: true,
6        title: const Text("Hotel and Car Rentals Recommendations"),
7      ),
8      body: Padding(
9        padding: const EdgeInsets.only(top: 20.0),
10       child: Column(
11         children: <Widget>[
12           Container(
13             margin: const EdgeInsets.symmetric(vertical: 10.0),
14             padding: const EdgeInsets.all(16.0),
15             decoration: const BoxDecoration(
16               //border: Border.all(),
17               //borderRadius: BorderRadius.circular(10.0),
18               color: Color.fromARGB(255, 180, 212, 238),
19             ),
20             child: Row(
21               mainAxisAlignment: MainAxisAlignment.spaceBetween,
22               children: <Widget>[
23                 const Expanded(
24                   child: Column(
25                     crossAxisAlignment: CrossAxisAlignment.start,
26                     children: [
27                       SizedBox(height: 10.0),
28                       Text(
29                         'Get best Hotel deals',
30                       style: TextStyle(

```

```

31         fontSize: 16.0,
32         fontWeight: FontWeight.bold,
33     ),
34 ),
35 SizedBox(height: 7.0),
36 Text(
37     'We compare from 1000+ hotel deals worldwide',
38     textAlign: TextAlign.left,
39     style: TextStyle(
40         fontSize: 12.0,
41     ),
42 ),
43 SizedBox(height: 10.0),
44 ],
45 ),
46 ),
47 ElevatedButton(
48     onPressed: () =>
49         _openWebsite(),
50     style: ElevatedButton.styleFrom(
51         backgroundColor: Colors.white,
52         side: const BorderSide(width: 1, color: Colors.deepPurple),
53         shape: RoundedRectangleBorder(
54             borderRadius: BorderRadius.circular(20.0),
55         ),
56     ),
57     child: const Text('Explore', style: TextStyle(color: Colors.
58         deepPurple),),
59 ),
60 ],
61 ),
62 const SizedBox(height: 15.0),
63 Container(
64     margin: const EdgeInsets.symmetric(vertical: 10.0),
65     padding: const EdgeInsets.all(16.0),
66     decoration: const BoxDecoration(
67         //border: Border.all(),
68         //borderRadius: BorderRadius.circular(10.0),
69         color: Color.fromARGB(255, 231, 188, 240)
70     ),
71     child: Row(
72         mainAxisAlignment: MainAxisAlignment.spaceBetween,
73         children: <Widget>[
74             const Expanded(
75                 child: Column(
76                     crossAxisAlignment: CrossAxisAlignment.start,
77                     children: [
78                         SizedBox(height: 10.0),
79                         Text(
80                             'Get a cheap rental cars',
81                             style: TextStyle(
82                                 fontSize: 16.0,
83                                 fontWeight: FontWeight.bold,
84                             ),
85                         ),
86                         SizedBox(height: 7.0),

```

```
87         Text(  
88           'We compare from 100+ rental car providers',  
89           textAlign: TextAlign.left,  
90           style: TextStyle(  
91             fontSize: 12.0,  
92           ),  
93         ),  
94         Text(  
95           'world wide',  
96           textAlign: TextAlign.left,  
97           style: TextStyle(  
98             fontSize: 12.0,  
99           ),  
100        ),  
101        SizedBox(height: 10.0),  
102      ],  
103    ),  
104  ),  
105  ElevatedButton(  
106    onPressed: () =>  
107      _openCarWebsite(),  
108    style: ElevatedButton.styleFrom(  
109      backgroundColor: Colors.white,  
110      side: const BorderSide(width: 1, color: Colors.deepPurple),  
111      shape: RoundedRectangleBorder(  
112        borderRadius: BorderRadius.circular(20.0),  
113      ),  
114    ),  
115    child: const Text('Explore', style: TextStyle(color: Colors.  
116      deepPurple)),  
116  ),  
117  ],  
118  ),  
119  ),  
120  ],  
121  ),  
122  ),  
123  ),  
124  );  
125 }
```

Listing 4.4: Dart code snippet

4.5 View Flight Screen

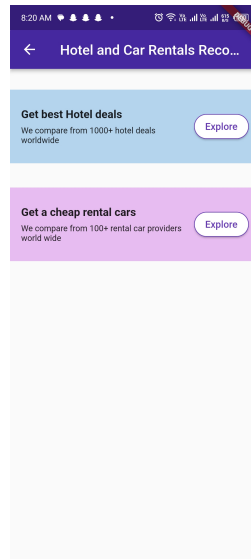


Figure 4.5: View Flight Screen

Code:

```

1  @override
2  void initState() {
3    super.initState();
4    // Call the function to get flight data from Firebase when the widget is
      initialized
5    getFlightData();
6  }
7
8  String _formatTimestamp(Timestamp timestamp) {
9    final DateTime dateTime = timestamp.toDate();
10   //final formattedDateTime = DateFormat('yyyy-MM-dd HH:mm').format(dateTime);
11   final DateFormat format = DateFormat('MMMM d, HH:mm');
12   final String formattedDateTime = format.format(dateTime);
13   return formattedDateTime;
14 }
15
16 Future<void> getFlightData() async {
17   try {
18     QuerySnapshot querySnapshot = await FirebaseFirestore.instance
19       .collection("Flight")
20       .get();
21
22     if (querySnapshot.docs.isNotEmpty) {
23       // Retrieve and store flight data in the list
24       /*flightDataList = querySnapshot.docs
25         .map((flightSnapshot) => flightSnapshot.data() as Map<String, dynamic>
26           >)
27         .toList();*/
28       flightDataList = querySnapshot.docs.map((flightSnapshot) {
29         final data = flightSnapshot.data() as Map<String, dynamic>;

```

```
29     final documentId = flightSnapshot.id; // Get the document ID
30     data['documentId'] = documentId; // Add the document ID to the data
31     return data;
32   }).toList();
33   //print(flightDataList);
34   // Force a rebuild to display the flight data
35   setState(() {});
36   } else {
37     print("No flight data found in Firestore.");
38   }
39   } catch (e) {
40     print("An unexpected error occurred: $e");
41     //log table
42     final log = LogTable(
43       page_name: "view_flight_page",
44       error: e.toString(),
45     );
46     final logRepository = LogTableRepository();
47     // ignore: use_build_context_synchronously
48     logRepository.createLog(context, log);
49   }
50 }
```

Listing 4.5: Dart code snippet