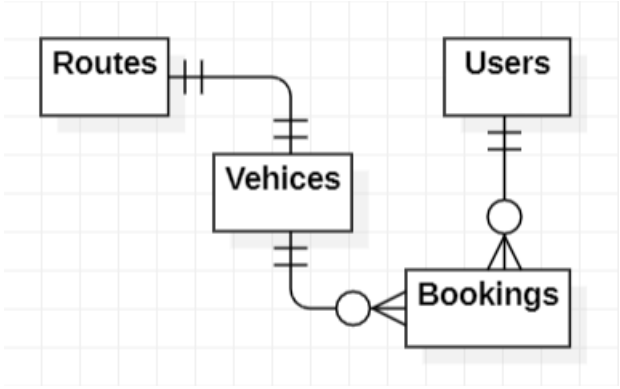
Code Overview

Introduction:

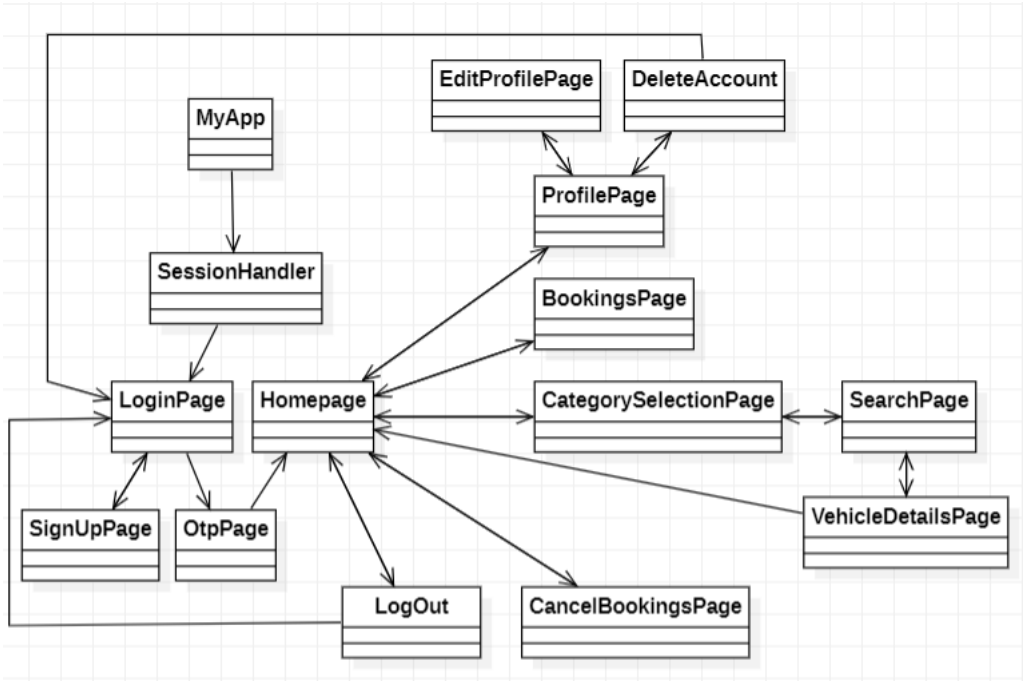
-A ticket booking app with basic functionalities for both buses and planes with a black and white theme.

Project Structure:

- Database Structure:



- App Structure:



Main.dart:

In this file two classes are used, first is the MyApp class that loads the App and second is the session handler class that has the method,

- Void checkSession()
  Which is used to see if user was logged in or not. From here we move on to the login screen or homepage, depending on situation.

There is also a commented section that is used to load the admin page to add routes or vehicle by pressing a button, it has no link to the app, just built it for testing covenince, the routes and vehicles have been hardcoded.

Login_page.dart:

In this file we have only one class login_page, this class has three methods,

- Bool validateEmail(String email)
  This check if email is entered or not

- Bool validatePassword(String password)
  This check if password is entered or not

- Future<void> login()
  This method check if the email and password exsist in database using a method created in db and sets the error message text accordingly. If email, password exsists, it takes us to the otp verification page.

Then in the build part of the code, we have two text fields, one button to login and another button to sign up.

SignUp.dart:

In this file we have a class by the name registerUser which has 5 text fields that ask user for data and I have also imported CSC widget for selecting the country, state and city. We have two methods in this class:

- Future<int> registerUser
  What this function does is it adds all the data the user has entered and adds it to the database and creates a P.K for this user so that we can access the data based on that P.K for this specific user. It also validates if any data has not been entered in the textfield by using the formkey.

- Void singleImage()
  In this method I use the image picker library to select a single image from gallery or camera to add in the profile of user.

The concept of email, password, etc formats have been taken from a previous project. The add user button doesn't work until a user has met all requirements prompted to him/her.

OtpPage.dart:

In this file we have one class OtpPage, it has three methods,

- Void sendOtp()

  The code for this method was picked from firebase's website and I have integrated it in my code with some alterations. It sends a request to the cloud and sends the otp code on the phone number the user enters while registering.

- Void verifyOtp()

  This method verifys if the entered code is correct or not. There is a try and catch block inside which throws an error if entered code does not match.

- Void navigateToHomePage()

  This method takes user to the homepage and also adds shared preference for session maintenance so that user stays logged in.

HomePage.dart:

This file also has one class that has one method:

- Future<void> Getdata()

  This method gets users data that is required by using the getters made in database.

  Inside the build widget we have firstly a carouselSlider for which I imported an external package from pub.dev and it shows a few images on the home screen. Below the slider we have 5 customs buttons, they, each navigate to specific pages.

LogoutPage.dart:

This file has one class only and no methods, it return a dialog box that is shown on top of the home screen, if the users confirms logging out, the Shared preference is removed and user gets rerouted to the login screen.

ProfilePage.dart:

This file again has only one stateful widget class, ProfilePage, it has contaners and textfields displayed in a list, the container at the top shows all the data of the user by getting it from the database. The two tiles below are the edit profile and delete account which when pressed reroute to specific pages. Ths class has three methods:

- Future<void> getdata()

Which gets all the users data from his/her pid.

- Future<void> editpage()
  This just takes the user to the editprofile page.

- Future<void> deleteAccountPrompt()
  This method return a dialog box that asks user to enter his email and password to confirm that he/she wants to delete their account. It also added the bookings and seats, etc back in the database.

EditProfilePage.dart:

This file contains the class EditProfilePage that has three methods,

- Future<void> getdata()
  Which gets all the users data from his/her pid.

- Future<void> saveProfileData()
  Which saves all the changes made to the user's account.

- singleImage()
  This is the same method as used on the signup page.

Overall this page contains 90% of the same code that is used while signing up with a few adjustments. Moreover the user cannot change his name, email and cnic as they are fixed.

BookinsPage.dart:

This file contains one class BookingsPage, and one method:

- void fetchBookings()
  This method return the list of bookings associated with the users pid.

In the build widget we have a listbuilder that builds all the bookings in a card. In the body we call another widget buildBookingCard, that takes list of bookings we fetch from table and then generates all the booking cards. VehicleId and PersonId alongwithBookingId gets us all the required information by using the appropriate getters.

FutureBuilder is used as we have to wait for the data to come from the database and as soon as the data is available, it builds the card. The card also shows two images, a logo which is stored in the database against the vehicle and a barcode, both these images are kept in the assets folder of this project in android studio.

CancelBookingsPage.dart:

This file contains one class CancelBookingsPage, and three methods:

- void fetchBookings()
  This method return the list of bookings associated with the users pid.

- cancelBooking(int bookingId)
  This method cancels the booking and adds all the seats and seat numbers to the database by calling a method from the database.

- Void showconfirmationDialog(int bookingid)
  This method prompts user with a dialogbox to confirm if he really wants to cancel his/her booking or not, if user confirms then it calls the cancelBooking method.

The rest build widget is same as the ones in BookingsPage class also with the alteration that no barcode is shown and a button is added for canceling instead.

CategorySelectionPage.dart:

This file has only one class Category_selection_page, it has two methods,

- Void onPlanePressed()

  This method calls the search page for planes.

- Void onPlanePressed()
  This method calls the search page for busses.

It shows two squares in the middle of the screen that lets user navigate to the next page based on his selection. An animated container widget was used to implement the press effect.

SearchPage

This page is divided into two sun files, they are exactly the same pages just one for plane and one for bus.

- PlaneBookingPage.dart
- BusBookingPage.dart

These pages are responsible for the selection criteria of showing available vehicles. Both pages have 5 methods:

- List<String> getAvailableCompanies()
  This method return al list of companies that are available for planes or busses.

- List<String> getAvailableLocations()
  This method is used to filter out the location selected in the from dialog box so that it doesn't appear in the To dialog box.

- Future<void> searchAvailableVehicles()
  This method invokes a query that uses 5 parameters for searching vehicles, the from and to locations, the from and to date, and lastly the 5th optional parameter Compnay, it then return a list of all available vehicles that have these specified parameters.

- Future<String> getFromDate(int routeId)
  This method invokes a database query that takes routeid and finds the date against that routeid and returns it.

- void _resetToDateDropdown()
  This method resets the 2nd date dialog whenever we reset out from date.

Besides these method, there are two widgets that are used, a listbuilder that builds a list of vehicles available in the form of listtile by using getters and shows information on those tiles. And the 2nd is the date widget which I has two parts, one the styling of the widget and 2nd is the logic which I learned from stack overflow.

DetailsPage.dart:

This file also has one class DetailsPage, that has 6 methods:

- void removeAllFromList(List<int> sourcsList, List<int>elemestToRemove)
  This is a helper function that takes two list one that has all available seats and the 2nd list the seats the user selects and it updates the available list.

- void updateSeatSelection(int seatNumber)
  This method is called inside the grid on tap feature, this adds the selected seat to the list and if user re presses the same seat, it removes the seat from the list.

- Future<void> getdata()
  This function is used to get data from the database that is to be used in this file.

- void updateseatControllerText()
  This method is used to show the remaining seats available of the vehicle, this is shown in the 2nd text field of this page.

- void calculateTotalPrice(String value)
  This method simply takes number of seats and multiplys it wwith the price andreturns it to display in the textfield at the end of page.

- void addBookings()
  This is the main method of this page, in this method the available seats list is updated and then all the data entered by the user is taken and added to the database. It updates the seat count of vehicle and also creates a new booking for the user. After the booking Is added it takes the user back to homepage.

DatabaseHelper.dart:

This file contains my SQFlite database for this project. It contains 4 tables:

- Users
- Vehicles
- Bookings
- Routes

And contains 38 methods that are used throughout the project:

- `static Future<void> createTables(Database database)`:
  Creates the required tables in the given Database object.

- `static Future<int> addUser(String email, String firstname, String lastname, int cnic, int phoneNumber, String country, String password, String? imagePath)`:
  Adds a user to the database and returns the inserted user's PID (Primary ID).

- `static Future<void> deleteAccount(int pid)`:
  Deletes a user's account and their associated bookings from the database.

- `static Future<void> deleteAllUsers()`:
  Deletes all users and their associated bookings from the database.

- `static Future<Map<String, dynamic>?> checkLogin(String email, String password)`:
  Checks if a user with the given email and password exists in the database and returns their details as a map.

- `static Future<String> getEmail(int pid)`:
  Retrieves the email of a user with the specified PID.

- `static Future<int> getPhoneNumber(int pid)`:
  Retrieves the phone number of a user with the specified PID.

- `static Future<String> getFirstname(int pid)`:
  Retrieves the first name of a user with the specified PID.

- `static Future<String> getLastname(int pid)`:
  Retrieves the last name of a user with the specified PID.

- `static Future<int> getCNIC(int pid)`:
  Retrieves the CNIC (National Identity Card) number of a user with the specified PID.

- `static Future<String> getCountry(int pid)`:
  Retrieves the country of a user with the specified PID.

- `static Future<String?> getImage(int userId)`:
  Retrieves the image path of a user with the specified PID.

- `static Future<void> updateProfileData({required int pid, String? firstName, String? lastName, int? phoneNumber, String? address})`:
  Updates the profile data (first name, last name, phone number, address) of a user with the specified PID.

- `static Future<void> setImage(int userId, String imagePath)`:
  Sets the image path for a user with the specified PID.

- `static Future<int> addRoute(String toLocation, String fromLocation, String toDate, String fromDate)`:
  Adds a route to the database and returns the inserted route's RID (Route ID).

- `static Future<String> getToLocation(int routeId)`:
  Retrieves the "to" location of a route with the specified RID.

- `static Future<String> getFromLocation(int routeId)`:
  Retrieves the "from" location of a route with the specified RID.

- `static Future<String> getToDate(int routeId)`:
  Retrieves the "to" date of a route with the specified RID.

- `static Future<String> getFromDate(int routeId)`:
  Retrieves the "from" date of a route with the specified RID.

- `static Future<List<String>> getFromLocations()`:
  Retrieves a list of all "from" locations available in the database.

- `static Future<int> addBooking(int personId, int vehicleId, int noOfSeats, List<int> seatNumbers)`:
  Adds a booking to the database and returns the inserted booking's BookingID.

- `static Future<List<Map<String, dynamic>>> getBookingsByPersonId(int personId)`:
  Retrieves a list of bookings made by a user with the specified PID.

- `static Future<void> cancelBooking(int bookingId)`:
  Cancels a booking with the specified BookingID and updates the corresponding vehicle's seat numbers.

- `static Future<List<int>> getSeatNumbersByBookingId(int bookingId)`:
  Retrieves a list of seat numbers for a booking with the specified BookingID.

- `static Future<int> addVehicle(String vehicleName, String vehicleType, int seats, double price, int routeId, String companyName, String logo, List<int> seatNumbers, String fromTime)`:
  Adds a vehicle to the database and returns the inserted vehicle's VehicleID.

- `static Future<String> getTime(int routeId)`:
  Retrieves the time from a vehicle with the specified VehicleID.

- `static Future<String> getVehicleName(int vehicleId)`:
  Retrieves the name of a vehicle with the specified VehicleID.

- `static Future<String> getVehicleType(int vehicleId)`:
  Retrieves the type of a vehicle with the specified VehicleID.

- `static Future<int> getTotalSeats(int vehicleId)`:
  Retrieves the total number of seats available in a vehicle with the specified VehicleID.

- `static Future<double> getPrice(int vehicleId)`:
  Retrieves the price per seat of a vehicle with the specified VehicleID.

- `static Future<String> getCompanyName(int vehicleId)`:
  Retrieves the company name associated with a vehicle with the specified VehicleID.

- `static Future<String> getVehicleLogo(int vehicleId)`:
  Retrieves the logo path associated with a vehicle with the specified VehicleID.

- `static Future<List<int>> getSeatNumbers(int vehicleId)`:
  Retrieves a list of seat numbers available in a vehicle with the specified VehicleID.

- `static Future<void> updateSeatNumbers(int vehicleId, List<int> seatNumbers)`:
  Updates the list of seat numbers for a vehicle with the specified VehicleID.

- `static Future<void> updateVehicleSeatNumbers(int vehicleId, List<int> seatNumbers)`:
  Updates the combined list of seat numbers (existing and new) for a vehicle with the specified VehicleID.

- `static Future<void> updateSeats(int vehicleId, int newSeats)`:
  Updates the total number of seats for a vehicle with the specified VehicleID.

- `static Future<int> getRouteIDFromVehicleID(int vehicleId)`:
  Retrieves the RouteID associated with a vehicle with the specified VehicleID.

- `static Future<void> deleteAllData()`:
  Deletes all data from the database (including bookings, vehicles, and routes).

Widgets.dart:

This files contains list of locations, plane & bus companies as well as 10 reusable widgets. Some of them were imported from the last projected and were altered based on the requirements for this project.

- `HeaderText`:
  A widget that displays a centered text with customizable label text. It is used to show headings with a bold font and a font size of 30.

- `BodyLabel`:
  Another text widget for displaying labels, but it doesn't center the text, and it uses the default font style.

- `CustomTextField`:
  A customizable text field widget that takes a `TextEditingController` to handle the text input and a `labelText` to display as a hint inside the text field. The text field is styled with rounded corners and filled with a light gray color when focused.

- `CustomTextFieldDecoration`:
  A helper class that provides the `getInputDecoration` method to create a consistent `InputDecoration` for the `CustomTextField`. It sets up the appearance of the text field, such as the label text color, fill color, and border styles.

- `CustomElevatedButton`:
  A customized elevated button widget that takes a `text` to display on the button and an optional `onPressed` callback for handling button taps. The button is styled with a black background and white text.

- `CustomButton`:
  A custom button widget that includes an icon along with text. It takes `buttonText` to display the text, `iconData` for the icon to show, and `onPressed` for the callback function when the button is pressed. The button has rounded corners and a shadow effect.

- `CustomBackButton`:
  A custom back button widget that creates a simple text button with the label "Back" and an `onPressed` callback that pops the current route from the navigation stack when pressed.

- `CustomListTile`:
  A custom list tile widget that represents a single item in a list. It displays `text` as the main content, `icon` on the trailing side, and `onPressed` to handle the tap event.

- `CustomInfoContainer`:
  A container that displays information in a row with an icon and text. It takes `text` to show the information and an `icon` to display at the beginning of the row.

- `LoadingWidget`:
  A simple widget to show a centered text "Loading..." with a black color. This is likely used to display a loading indicator or message when data is being fetched or processed.