Campus Parking

Software Requirements Specification

Version: 1.1.0

Aug 21, 2020

Team Members

Alekya Pochampally

Chetan Kudaravalli

Chinmayi Ambati

Manideep Chamala

Manisha Mengani

Xizi Chen

Submitted in partial fulfillment

Of the requirements of

CSIS 44-691 Graduate Directed Project 1

# Revision History

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| **Date** | **Description** | **Author** | **Comments** |
| June 10, 2020 | Introduction to the project | Chetan Kudaravalli,  Manideep Chamala | Added Introduction chapter-1 |
| June 10, 2020 | General description of the project | Alekya Pochampally,  Chinmayi Ambati | Added general description of project chapter-2 |
| June 10,  2020 | Requirements of the project | Manisha Mengani,  Xizi Chen | Added requirements of the project chapter-3 |
| August 21, 2020 | More functionalities | Chetan Kudaravalli,  Manideep Chamala,Alekya Pochampally,  Chinmayi Ambati,  Manisha Mengani,  Xizi Chen | Add GUI design at chapter-4  Add more content in chapter-7 |

# Document Approval

The following Software Requirements Specification has been accepted and approved by the following:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Chetan Kudaravalli | Mr. | June 10, 2020 |
|  | Manisha Mengani | Ms. | June 10, 2020 |
|  | Manideep Chamala | Mr. | June 10, 2020 |
|  | Alekya Pochampally | Ms. | June 10, 2020 |
|  | Xizi Chen | Mr. | June 10, 2020 |
|  | Chinmayi Ambati | Ms. | June 10, 2020 |

# Colors used by the team

# Chetan Kudaravalli - Dark Magenta 2

# Manisha Mengani - Dark Green 2

# Manideep Chamala - Magenta

# Alekya Pochampally - Red Berry

# Xizi Chen - Dark Cornflower Blue 1

# Chinmayi Ambati - PurpleTable of Contents

**Table of Contents Page Number**

1. Introduction
   1. Purpose
   2. Scope
   3. Definitions, Acronyms, and Abbreviations
   4. References
   5. Overview
2. General Description
   1. Product Perspective
   2. Product Functions
   3. User Characteristics
   4. General Constraints
   5. Assumptions and Dependencies
3. Specific Requirements
   1. External Interface Requirements
      1. User Interfaces
      2. Hardware Interfaces
      3. Software Interfaces
      4. Communications Interface
   2. Functional Requirements
   3. Use Cases
   4. Non-Functional Requirements

3.4.1. Performance

3.4.2. Reliability

3.4.3 Availability

3.4.4 Security

3.4.5 Portability

4. Design

4.1. ER diagram

4.2. GUI

1. Analysis Models

5.1. Data Flow Diagram

5.2. Sequence Diagram

6. Technical Manual

6.1. Feature Included in the Template

6.2. Installing the Software Manual Template

6.3. Front End Development

6.4. Back End Development

7. End-User Manual

7.1. Login Screen

7.2. Sign up

7.3. Forgot password

7.4 Parking screen

7.5 Hamburger menu

7.6 Profile

7.7 Map

7.8 Report

7.9 FAQ

7.10 Live chat

7.11 Ticket screen

7.12 Contact us screen

## CHAPTER 1

**Introduction**

**1.1 Purpose**

It is basic that Students and faculty have to get to class on time and quite often we see a lot of delay in searching for free parking space. We are introducing the Campus Parking app which gives people the best option in saving time in searching for the free parking space on the campus. The app shows users how many open spaces are available in selected campus parking lots. Not only this, but the app also gives details about the positions of Faculty, Student, Handicapped, and also the Visitor parking in the Campus area.

**1.2 Scope**

Initially, we have a couple of options that the particular university has to execute for the best use of the app. First thing is that the university may issue a rule that every user must install the app to use the campus parking and as there are no censors coming into the play the user itself has to select the parking space in the app once it is occupied by that user, so that once the user selects the particular parking space that they occupied, that particular parking space turns into Red color and the remaining empty spaces will be as it is in White color making everyone know the availability of free parking spaces. This can be achieved by just using Notices round the Campus Parking Zone and on Campus.

The second thing which is an alternative to the first step and also which could be a better idea is that the University may allot separate parking lots for this app so that only users who have this app installed with them can use this space without having conflicts with the users who don't have this app installed.

**1.3 Objectives**

This application will be providing information related to the availability of parking lots in the university.

Users will use this application for knowing the current status of the parking lots in the university.

Users can occupy, vacate the parking lots, and admin role users can also reserve a particular parking lot.

They can also contact admin for any queries by using the live chat feature in the application.

There are some other features like map screen which will help the user for better understanding of the parking lots locations, FAQ screen to seek their clarifications, report screen for reporting inconsistent data in the application, and report violation screen for reporting a violation of university parking terms.

**1.4 Definitions, Acronyms, and Abbreviations**

The following terms, acronyms, and abbreviations are used throughout this document.

**Campus Parking App:** We use this term many times and this is the actual app name that we are going to develop in the project which is Campus Parking. This helps users to know the availability of parking spaces in the Campus.

**Faculty, Student, and Visitor:** All these three terms refer to almost the same thing that these are the USERs who are going to use the parking space in the Campus and who has the app installed.

**Reserved:** This refers to the parking space that is already filled by other users.

**Vacant:** This refers to the parking space that is empty and any user can use the appropriate space by notifying the other users. This includes both the cases where the user forgets to update that he vacated the parking slot and also if the user reserved a slot and didn’t update it to the app.

**Token:** Token refers to the change that’s imposed on the user if he fails to update the usage of the parking lot.

**Parking Lots:** This refers to a cleared area that is intended for parking vehicles. Usually, the term refers to a dedicated area that has been provided with a durable or semi-durable surface.

**1.5**  **References**

• <https://fluttercrashcourse.com/>

• <https://www.tutorialspoint.com/dart_programming/index.htm>

• <https://flutter.dev/>

**1.6**  **Overview**

Campus Parking App provides users the details of the availability of the parking spaces in the Campus Parking Lot which also differentiates between Student, Faculty, and Visitors. This helps the users in saving big time in searching for the Parking space in the Campus.

**Chapter 2**

**General Description**

**2.1**  **Product Perspective**

The main perspective of the campus parking application is to be helpful for students, faculty, or visitors to book a parking slot without any struggle during peak times.

**2.2**  **Product Functions**

The functions of this application are divided among a team of 6 members and assigned their roles to develop, with daily meetings and discussions for daily progress in the app.

**2.3 User Characteristics**

There are different types of users in this application such as students, faculty, and visitors. Firstly, everyone has to register in the app to book a slot in the parking lot. After that, their characteristics depend on the type of parking they need such as handicapped, visitor, or faculty/worker parking. They are restricted to the availability of slots in a parking lot. This application serves as a first come first serve for users. Users can have their profile and report other vehicles with an image.

**2.4**  **General Constraints**

We as a team of 6 members have divided the application into various segments and decided to complete in 5 months and allocated a deadline for phase 1 which is the 1st week of July.

**2.5**  **Assumptions and Dependencies**

According to our project plan, we have to gather all our requirements which are necessary for the app development, then to have use cases and design the ER-Diagram to represent the relationships between entities. At this stage, we have these assumptions for our project

• Setting up a developer environment according to required technologies.

• Setting up the project for both frontend and backend in Github.

• Designing the prototype of the project

• Listing out all the functionalities to be provided in the project and work distribution

• Designing UI – Login Page, functionality – A login page, Designing UI – Home page

and Designing UI – Login Page.

• Routing the page navigations and building the database where we populating data into

database.

• Merging the frontend and backend and debugging till then.

**Chapter 3**

**Specific Requirements**

1. **External Interface Requirements**

**3.1.1 User Interfaces**

**3.1.1.1 Login page**

This pagewill show the title of the app, prompts, and text boxes that allow the user input username/password.

Button “Sign up” for creating a new account.

Button “Sign in” for login

Button “Reset” for resetting one’s password.

**3.1.1.2 New User Page**

This page will show text boxes, prompts where users can input their details and register in the application.

**3.1.1.3 Reset password**

This page will allow the user to reset his forgotten password with a new password through a valid username.

**3.1.1.4 Home page**

This page will have buttons such as maps, reports, logout, and parking lots, where there are various parking lots in the university.

**3.1.1.5 Map Page**

This page will have a top view image of all parking lots where users can zoom in and out, back, and navigation buttons.

**3.1.1.6 Parking Area Page**

This page will show available lots for parking which is interactive like select/unselect and two buttons such as home and reserve.

**3.1.1.7 Select Slot Page**

This page will allow users to select an available slot in the parking lot which has slot id as down drop text box and back, cancel, reserve as buttons.

**3.1.1.8 Profile Page**

This page will have text boxes such as vehicle number, tickets, and vehicle type and a button back to return home.

**3.1.1.9 Report page**

This page will have vehicle number and image as text boxes and back, cancel, report as buttons.

**3.1.1.10 FAQ page**

This page will show frequently asked questions, a search bar, email and phone number of the customer services.

**3.1.2 Hardware Interfaces**

• Processor: Minimum 1 GHz; Recommended 2GHz or more

• Wireless adapter (Wi-Fi) OR Mobile Data (3G/LTE)

• Hard Drive: Minimum 16 GB; Recommended 64 GB or more

• Memory (RAM): Minimum 1 GB; Recommended 4 GB or above

**3.1.3 Software Interfaces**

• Flutter

• SQLite

• Dart

• Visual Studio

**3.1.4 Communications Interface**

• IOS

• Android

1. **Functional Requirements**

A new user can sign up as a common user which could add one’s vehicle to the account. A common user could sign in, view, and manage his account including user information, vehicle, and balance.

Admin users not only have all functions of the common user but also can add funds for users.

**3.2.1 Login**

Users can login to the application.

**3.2.2 Sign up**

Users can register into the application.

**3.2.3 Password reset**

User can reset their application when they are unable to login with their current

credentials.

**3.2.4 Logout**

Users can exit the application by logging out.

**3.2.5 Parking lots**

Users can view the list of all parking lots in the university.

**3.2.6 Occupy slot**

Users can occupy a parking slot at the time of parking their vehicle in the slot.

**3.2.7 Vacate slot**

Users can vacate the parking slot at the time of removing their vehicle from the slot.

**3.2.8 Map review**

Users can see the university map to identify the location of parking lots in the campus.

**3.2.9 Vacancy/Occupancy correction report**

The user can report admin if there is any inconsistent data in the application.

**3.2.10 User profile**

Users can view their profile which contains the name, vehicle details, number of

tickets they have, etc.

**3.2.11 Parking Tickets**

If the user forgets to update the app (occupy/vacant) they will be fined.

**3.2.12 Report violations**

If a vehicle without a parking permit is parked, it can be reported.

**3.2.13 FAQ and contact us**

Users can view the frequently asked questions in the applications and as well as post

their queries. They can also contact us via email and there will also be a live chat

feature in the application.

**3.2.14 Admin module**

Admin will also be having a login. They will be having complete access to the

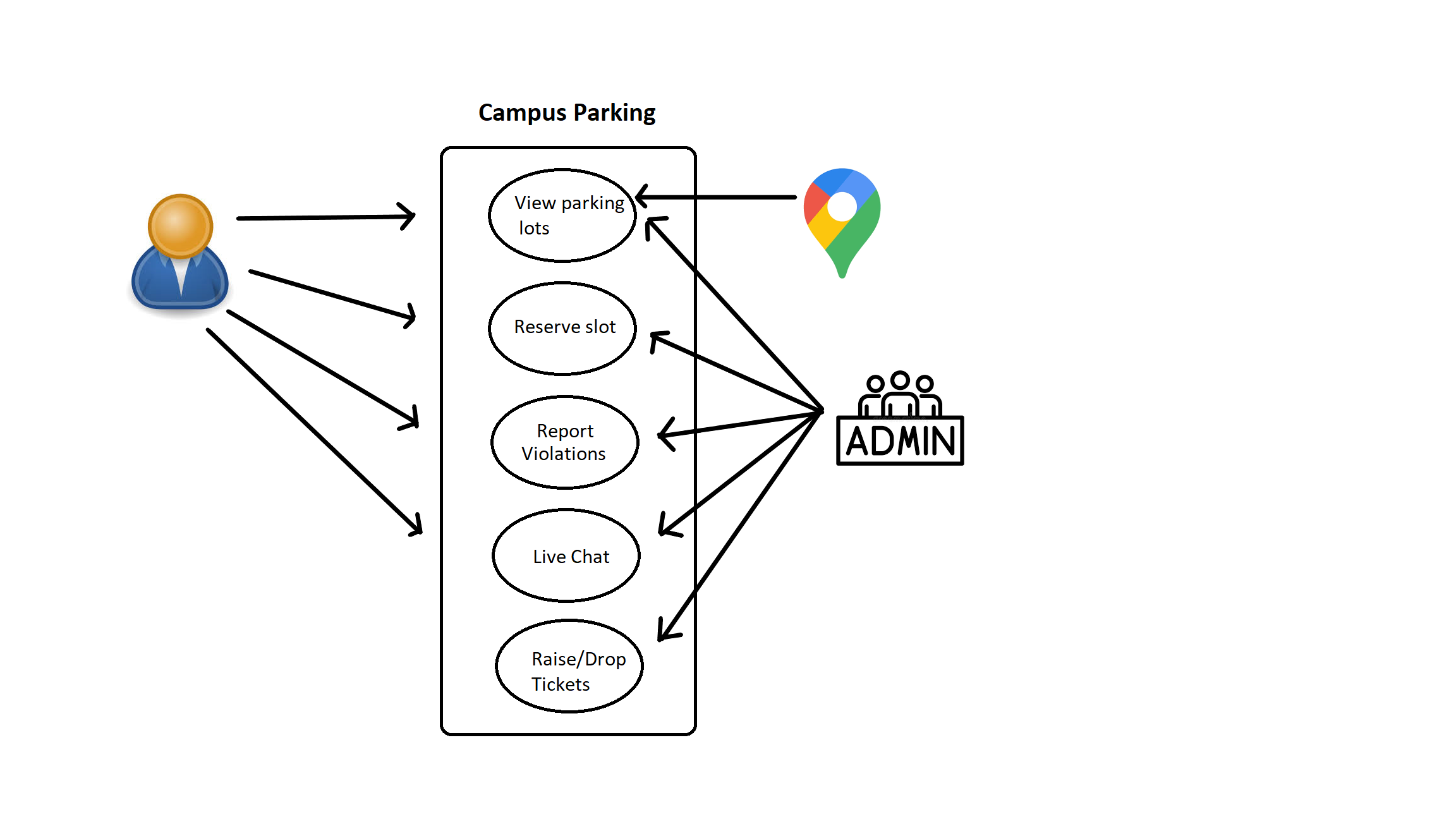
application i.e, they have access to both front end and also backend. In the backend

they can modify the changes to parking lots, number of slots in each parking lot,

number of available/ reserve slots, etc. They can even reserve parking lots for a

specific time for visitors.

1. **Use Cases**

****

**3.4 Non-Functional Requirements**

**3.4.1 Performance**

This cross-platform application is developed using Android Studio and Flutter which result in a high execution speed and also minimizes the response time.

**3.4.2 Reliability**

The application is tested before deployment and hence, it will perform in the way it was intended to.

**3.4.3 Availability**

The application is available to the users at all times except for any special cases such as backup. This will be announced beforehand to the users through push notifications.

**3.4.4 Security**

The data of the users is secured by the administrator and integrity is thoroughly maintained.

**3.4.5 Portability**

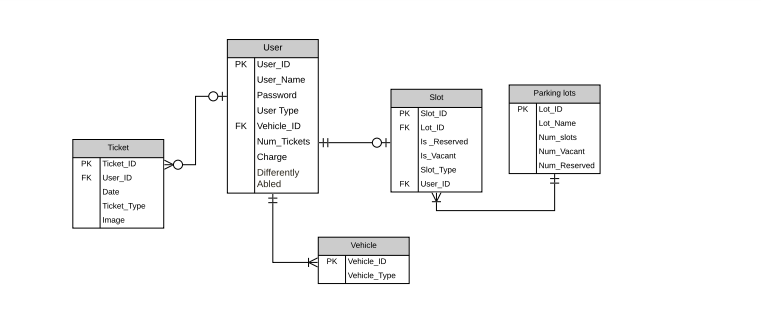
This application can be operated on any OS and hence is highly portable.

**Chapter 4**

**Design**

**4.1 ER Diagram**

Below is the ER Diagram of our application where we have 5 entities: token, user, slot, vehicle and parking lots.



**4.1.1 About ER Diagram**

There are 5 entities namely Tickets, User, Slot, Parking lots, Vehicle.

**4.1.1.1 Ticket**

This entity is used for finding the user if they forget to update the application. The

users will be charged based on the number of tickets they have. This entity will be

having few attributes like

Date : which will be telling the user on which day they got fined.

Image : image of the screenshot or user vehicle

Token type : reserve/ vacant

· UserID : This is the foreign key of the table and user to identify an individual

Relations : Each ticket is for 1 user only.

**4.1.1.2 User**

This entity is used to represent an individual using this app. The following are the attributes

· UserID : Unique number for each user

· User\_Name : Name of the user

· Password : Password of the user

· User Type: faculty or student

· Vehicle\_ID: this is a foreign key used to track individuals vehicle

· Num\_Ticket: Number of tokens per user

· Charge: will be calculated based on the number of tickets users have.

· Differently abled: This is used for directing to reserved parking slots for specific users

Relations : Each user can have

· multiple tickets

· multiple vehicles

· 0 or 1 slot

**4.1.1.3 Vehicle Entity**

This entity represents the vehicle used by the user. The following are its attributes

· Vehicle\_ID: This is a primary key used to uniquely identify the vehicle

· Vehicle\_Type: 2 or 4 wheeler

Relations: 1 Vehicle will be having only 1 user

**4.1.1.4 Parking lots**

This entity represents all the parking lots in the university. The following are its attributes

· Lot\_ID: This is the primary key and uniquely identify particular parking lot

· Lot\_Name: name of the parking lot

· Num\_slots: this gives the capacity of the parking lot

· Num\_Vacant: Number of available slots for parking

· Num\_Reserved: Number of reserved slots

Relations: Each parking lot can have 1 or more slots

**4.1.1.5 Slot**

This entity represents each slot in any parking lot. This entity have the following attributes

· Slot\_ID: This is the primary key and uniquely identifies each slot

· Lot\_ID: This is the foreign key and represents the Lot\_ID in parking lots entity

· Is\_Reserved: states whether the entity is reserved or not

· Is\_Vacant: states whether the entity is vacant or not.

· Slot\_Type: slot meant for differently abled persons or not.

· User\_ID: this is a foreign key which identifies users uniquely.

Relations:

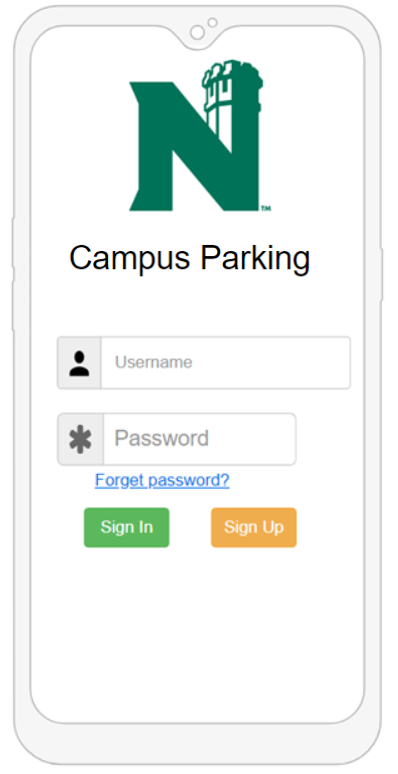
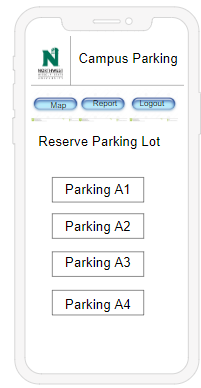
· Each slot have 1 user

· Belongs to only 1 parking lot

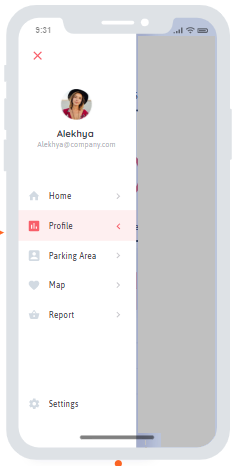
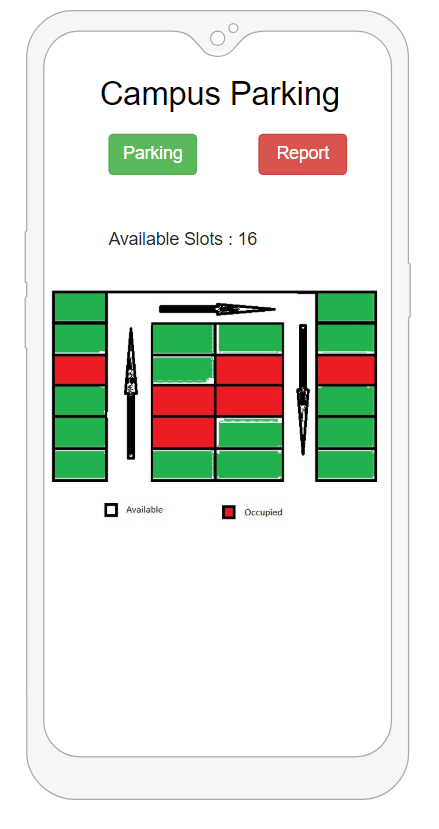
**4.2 GUI**

These are mockup’s of our application’s Graphical user interface.

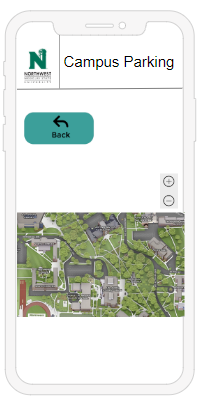
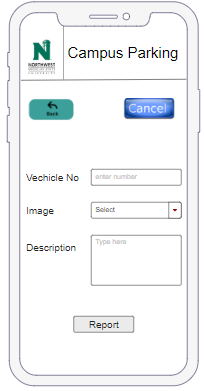
**Login Page Home Page**

** **

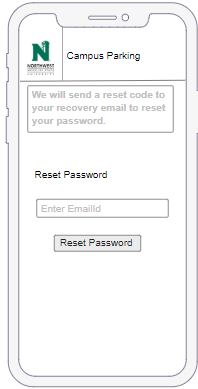
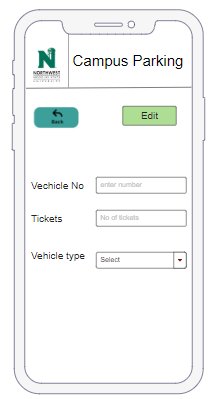
**Index Page Parking Page**

** **

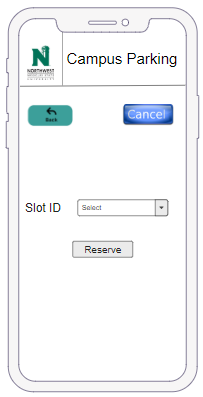
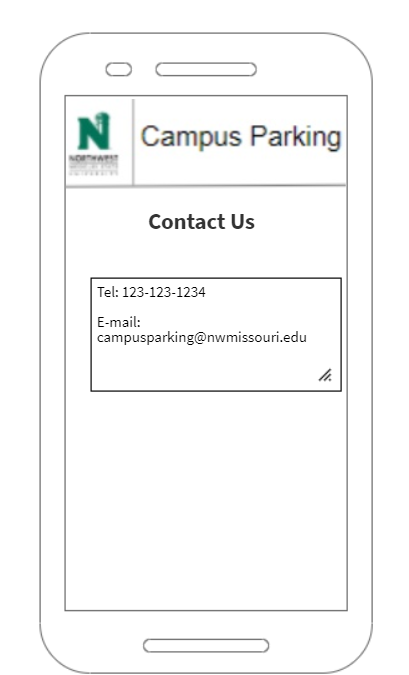
**Map Page Report Page**

** **

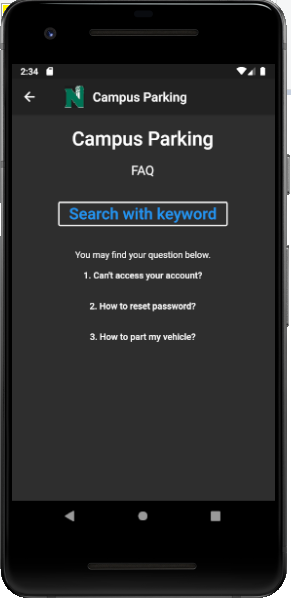
**Forgot Password Page Profile Page**

** **

**Connection Page Contact us Page**

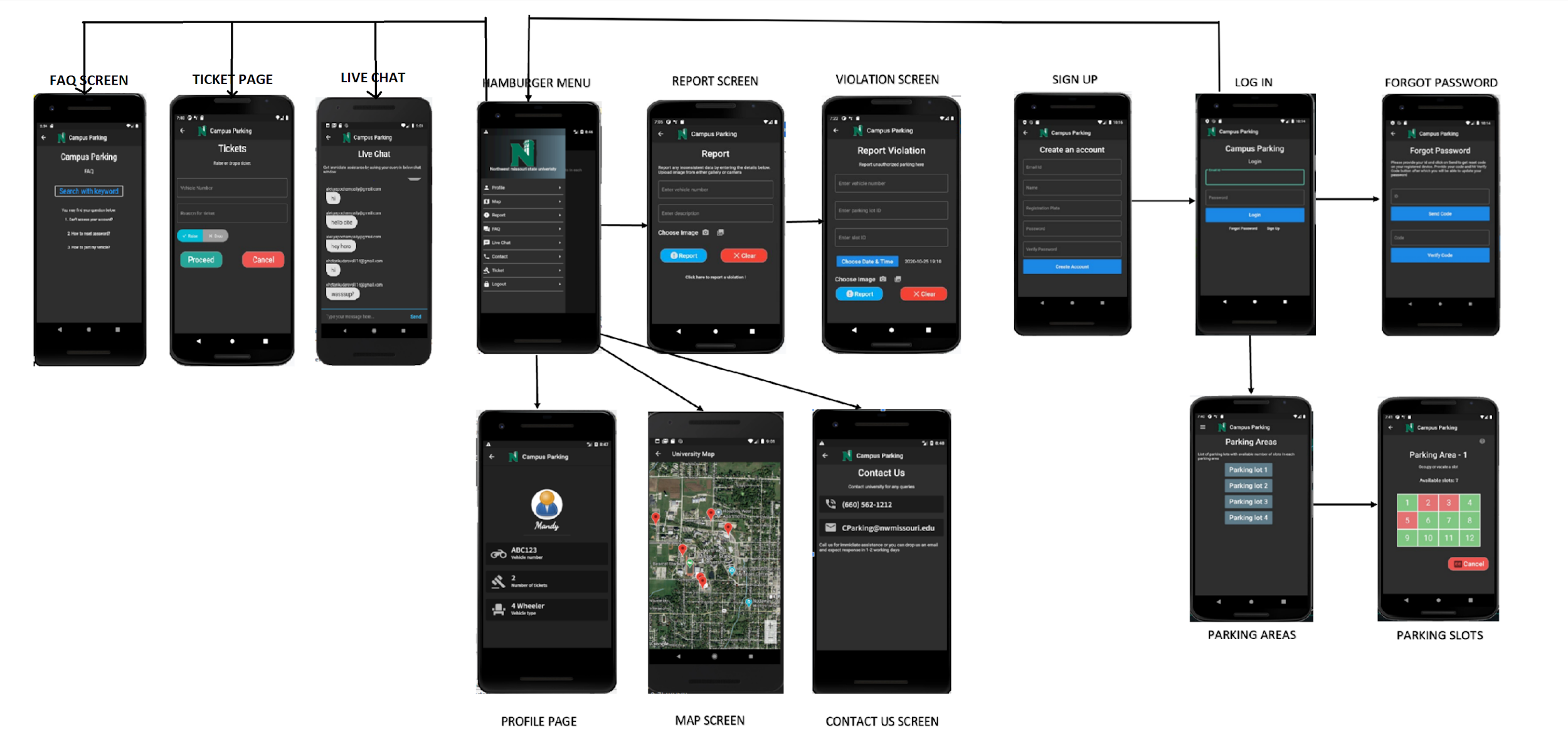
** **

**FAQ**

****

## CHAPTER 5

**Flow Chart**

****

## 

## CHAPTER 6

**Technical Manual**

**6.1. Features of the application**

Listing of features application currently includes :

1. Sign in
2. Sign up
3. Reset password

This application is intended to attain parking slots at Northwest Missouri University and operates on smartphones(Android 5.0 or newer/ IOS 7 or newer).

**6.2. Installing the Software Manual**

* For Android phones, run the .apk file.
* For the iPhone, install the iPhone SDK.

**6.3. Front End development**

**6.3.1 Login page**

For clear understanding a detailed comment is mentioned inside the code for the login page.

class LoginPage extends StatefulWidget {

static const String route = '/login';

@override

\_LoginPageState createState() => \_LoginPageState();

}

* Every class created must extend any of the state framework. A stateful widget can create a state() method multiple times over its lifetime. A widget has a mutable state, route for the page is created in the login page class.

class \_LoginPageState extends State<LoginPage> {

final \_formKey = GlobalKey<FormState>();

UserServices \_userServices = UserServices();

String \_email;

String \_password;

@override

void initState() {

super.initState();

}

* A global key uniquely identifies the form widget which makes the validation easy.\_email and \_password are the form elements of the login form.

initState() is used to insert widgets inside the stateful tree.

@override

Widget build(BuildContext context) {

//building a form widget using \_formkey provided earlier

return Scaffold(

// scaffold is a framework to provide a basic layout to the application.

appBar: ApplicationBar(),

//appBar is used to provide fixed height widget to the screen

body: SafeArea(

//Safe area considers its padding and adjustments of the screen which works for both android and iPhone

child: Form(

key: \_formKey,

//\_formkey widget is used to start its functionality

child: Padding(

padding: EdgeInsets.all(10),

child: ListView(

// child is the layout widget in dart

children: [

Container(

// container combines positioning and sizing widgets

alignment: Alignment.center,

padding: EdgeInsets.all(10),

child: Text(

'Campus Parking',

style: TextStyle(fontWeight: FontWeight.w600, fontSize: 30),

),

),

Container(

alignment: Alignment.center,

padding: EdgeInsets.all(10),

child: Text(

'Login',

style: TextStyle(fontSize: 20),

),

),

)

}

Below are the form elements which are inserted into individual containers for their alignment and positionings

// Username

Container(

padding: EdgeInsets.all(10),

child: TextFormField(

keyboardType: TextInputType.emailAddress,

decoration: InputDecoration(

border: OutlineInputBorder(),

// border online for the form

labelText: 'Email Id',

// email label of type text

),

validator: (value) {

if (value.isEmpty) {

return 'Provide an email id';

// validations check if value provided is empty

}

\_formKey.currentState.save();

return null;

},

onSaved: (value) {

\_email = value;

},

),

),

Check the password form element which with its alignment

// Password

Container(

padding: EdgeInsets.all(10),

child: TextFormField(

obscureText: true,

// is used to hide the input given

decoration: InputDecoration(

border: OutlineInputBorder(),

labelText: 'Password',

// label password for text type

),

validator: (value) {

if (value.isEmpty) {

return 'Provide your password';

}

\_formKey.currentState.save();

return null;

},

onSaved: (value) {

// optional value to be saved with the final value after form is saved

\_password = value;

},

),

),

The code below illustrates the actions that need to be performed after the login button is pressed.

// Login button

Container(

height: 55,

padding: EdgeInsets.fromLTRB(10, 0, 10, 0),

child: RaisedButton(

onPressed: () async {

if (\_formKey.currentState.validate()) {

\_formKey.currentState.save();

var authenticateUserRequest = AuthenticateUserRequest(emailId: \_email, password: \_password);

//after successful entering the username and password,authentication call is sent to backend

final result =

await \_userServices.authenticate(authenticateUserRequest);

// waits until it receives back the data

if (result != null) {

var sharedPreference = await SharedPreferences.getInstance();

sharedPreference.setString('cp\_token', result.token);

Navigator.of(context).pushReplacementNamed(UserHomePage.route);

}

}

},

child: Text(

'Login',

style: TextStyle(fontSize: 18),

),

),

),

button functionality for forgot password and signup is made available below the login button.

This is completely immersed in to a single container and aligned into single child element.

// Signup information

Container(

child: Row(

mainAxisAlignment: MainAxisAlignment.center,

// alignment of the buttons to appear

children: [

// single child element for both the buttons

FlatButton(

onPressed: () => Navigator.of(context)

.pushNamed(ForgotPasswordPage.route),

// navigating to forgot page if user clicks on forgot password button

child: Text('Forgot Password')),

FlatButton(

onPressed: () =>

Navigator.of(context).pushNamed(SignUpPage.route),

// navigating to signup page which will be discussed below

child: Text('Sign Up'))

],

Type ID and password where indicated, then tap the Login button if ID and password correct page will navigate into the application

**6.3.2 Sign up page**

* When a user clicks on sign up this snippet helps to route to sign up page and the class extends statefulwidget as usual for implementing other widgets.

class SignUpPage extends StatefulWidget {

static const String route = '/sign-up';

@override

\_SignUpPageState createState() => \_SignUpPageState();

}

* This is a simple declaration for the containers we are going to use for user input where the class extends State<signupPage>.

class \_SignUpPageState extends State<SignUpPage> {

final \_formKey = GlobalKey<FormState>();

CreateUserBloc \_createUserBloc;

UserServices \_userServices = UserServices();

String \_emailId;

String \_name;

String \_registrationPlate;

String \_password;

String \_repeatPassword;

This is a small part of the widget where we have a container which is the title of the page “Create an account” and then there are many other containers for text boxes and buttons.

@override

void initState() {

*super*.initState();

}

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: ApplicationBar(),

body: Form(

key: \_formKey,

child: SafeArea(

child: Padding(

padding: EdgeInsets.all(10),

child: ListView(

children: [

Container(

alignment: Alignment.center,

padding: EdgeInsets.all(10),

child: Text(

'Create an account',

style: TextStyle(fontWeight: FontWeight.w600, fontSize: 30),

),

),

**6.3.3. Forgot password page**

* This code is a route to the forgot password page when the user clicks on the forgot password and extend statefulWidget.

class ForgotPasswordPage extends StatefulWidget {

static const String route = '/forgot-password';

@override

\_ForgotPasswordPageState createState() => \_ForgotPasswordPageState();

}

* The class forgot password has only two variables, email, and otp as strings. As the page contains two text fields and two buttons.

class \_ForgotPasswordPageState extends State<ForgotPasswordPage> {

final \_formKey = GlobalKey<FormState>();

UserServices \_userServices = UserServices();

String \_email;

String \_otp;

* The two text fields are email which is taken from the user for conformation and code which is like OTP to confirm it is the same user. The two buttons are “send code” and “verify code”.
* Here you can see the widget for forgot password and container where it contains forgot password as title and in actual code, you can see building up other containers for other fields as shown in the UI page.

@override

void initState() {

*super*.initState();

}

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: ApplicationBar(),

body: SafeArea(

child: Form(

child: Padding(

padding: EdgeInsets.all(10),

child: ListView(

children: [

Container(

alignment: Alignment.center,

padding: EdgeInsets.all(10),

child: Text(

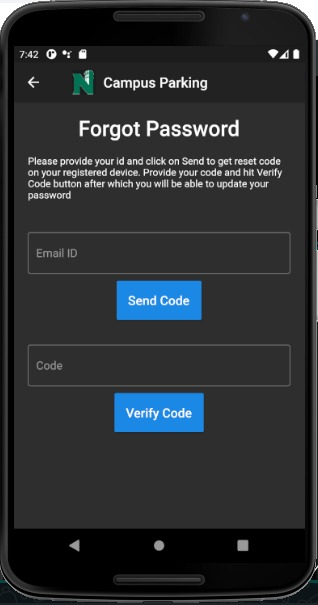
'Forgot Password',

style: TextStyle(fontWeight: FontWeight.w600, fontSize: 30),

),

),

Below is the screenshot of the page



**6.3.4 Change password page**

Routing code for changing password page routes to this page when the user presses on “verify code” and if the code is matched with the sent code.

**class ChangePasswordPage extends StatefulWidget {**

**static const String route = '/change-password';**

**@override**

**\_ChangePasswordPageState createState() => \_ChangePasswordPageState();**

**}**

This class contains widgets and containers, where it contains two text boxes and one button.

Two text boxes are “New password” and “re-enter new password” if both are same and password matches with required constraints the password will be updated.

**class \_ChangePasswordPageState extends State<ChangePasswordPage> {**

**@override**

**Widget build(BuildContext context) {**

**return Scaffold(**

**appBar: ApplicationBar(),**

**body: SafeArea(**

**child: Padding(**

**padding: EdgeInsets.all(10),**

**child: ListView(**

**children: [**

**Container(**

**alignment: Alignment.center,**

**padding: EdgeInsets.all(10),**

**child: Text(**

**'Reset Password',**

**style: TextStyle(fontWeight: FontWeight.w600, fontSize: 30),**

**),**

**),**

**Container(**

**alignment: Alignment.center,**

**padding: EdgeInsets.all(10),**

**child: Text('Please provide a new password and verify it'),**

**),**

**SizedBox(height: 25),**

**Container(**

**padding: EdgeInsets.all(10),**

**child: TextField(**

**obscureText: true,**

**decoration: InputDecoration(**

**border: OutlineInputBorder(),**

**labelText: 'New Password',**

**),**

**),**

**),**

**Container(**

**padding: EdgeInsets.all(10),**

**child: TextField(**

**obscureText: true,**

**decoration: InputDecoration(**

**border: OutlineInputBorder(),**

**labelText: 'Verify Password',**

**),**

**),**

**),**

**Container(**

**height: 55,**

**padding: EdgeInsets.fromLTRB(10, 0, 10, 0),**

**child: RaisedButton(**

**onPressed: () => {},**

**child: Text(**

**'Change Password',**

**style: TextStyle(fontSize: 18),**

**),**

**),**

**),**

**],**

**),**

**),**

**),**

**);**

**}**

**}**

**6.3.5 User Home page**

Routes to this page, when the user opens his application and this page internally routes to different parking lots when clicked on them.

**final \_firestore = Firestore.instance;**

**FirebaseUser loggedInUser;**

**class UserHomePage extends StatefulWidget {**

**static const String route = '/user-home';**

**static String Lot\_name;**

**static int available = 00;**

**@override**

**\_UserHomePageState createState() => \_UserHomePageState();**

**}**

The user home page consists of the title, parking lots list and also consists a hamburger menu button.

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: ApplicationBar(),

body: SafeArea(

child: ListView(

// mainAxisAlignment: MainAxisAlignment.center,

children: <Widget>[

Container(

alignment: Alignment.*center*,

padding: EdgeInsets.all(10),

child: Text(

'Parking Areas',

style: TextStyle(fontWeight: FontWeight.*w600*, fontSize: 30),

),

),

Container(

alignment: Alignment.*center*,

padding: EdgeInsets.all(10),

child: Text(

'List of parking lots with available number of slots in each parking area',

),

),

Container(

child: Column(

mainAxisAlignment: MainAxisAlignment.center,

children: [

Row(

mainAxisAlignment: MainAxisAlignment.center,

children: [

ButtonTheme(

minWidth: 175.0,

height: 50.0,

child: RaisedButton(

splashColor: Colors.*grey*,

color: Colors.*blueGrey*,

onPressed: () async {

UserHomePage.*Lot\_name* = 'PA1';

Navigator.*of*(context).pushReplacementNamed(Parking.*route*);

},

child: Text('Parking lot 1',

style: TextStyle(fontSize: 25.0)),

),

),

Padding(

padding: const EdgeInsets.all(10.0),

// child: Text(

// count.toString(),

// style: TextStyle(

// fontSize: 25.0,

// fontWeight: FontWeight.bold,

// ),

// ),

),

],

),

SizedBox(height: 10),

Row(

mainAxisAlignment: MainAxisAlignment.center,

children: [

ButtonTheme(

minWidth: 175.0,

height: 50.0,

child: RaisedButton(

splashColor: Colors.*grey*,

color: Colors.*blueGrey*,

onPressed: () async {

UserHomePage.*Lot\_name* = 'PA2';

Navigator.*of*(context).pushReplacementNamed(Parking.*route*);

},

child: Text('Parking lot 2',

style: TextStyle(fontSize: 25.0)),

),

),

Padding(

padding: const EdgeInsets.all(10.0),

),

],

),

SizedBox(height: 10),

Row(

mainAxisAlignment: MainAxisAlignment.center,

children: [

ButtonTheme(

minWidth: 175.0,

height: 50.0,

child: RaisedButton(

splashColor: Colors.*grey*,

color: Colors.*blueGrey*,

onPressed: () async {

UserHomePage.*Lot\_name* = 'PA3';

Navigator.*of*(context).pushReplacementNamed(Parking.*route*);

},

child: Text('Parking lot 3',

style: TextStyle(fontSize: 25.0)),

),

),

Padding(

padding: const EdgeInsets.all(10.0),

),

],

),

SizedBox(height: 10),

Row(

mainAxisAlignment: MainAxisAlignment.center,

children: [

ButtonTheme(

minWidth: 175.0,

height: 50.0,

child: RaisedButton(

splashColor: Colors.*grey*,

color: Colors.*blueGrey*,

onPressed: () async {

UserHomePage.*Lot\_name* = 'PA4';

Navigator.*of*(context).pushReplacementNamed(Parking.*route*);

},

child: Text('Parking lot 4',

style: TextStyle(fontSize: 25.0)),

),

),

Padding(

padding: const EdgeInsets.all(10.0),

),

],

),

],

),

)

]),

),

drawer: Drawer(

child: ListView(

padding: EdgeInsets.*zero*,

children: <Widget>[

DrawerHeader(

decoration: BoxDecoration(

gradient: LinearGradient(colors: <Color>[

Colors.*black54*,

Colors.*blueGrey*[500]

])),

child: Container(

child: Column(

children: <Widget>[

Material(

borderRadius: BorderRadius.all(Radius.circular(50.0)),

child: Image.asset(

'assets/images/nwm\_logo.png',

width: 100,

height: 100,

),

),

Padding(

padding: EdgeInsets.all(0.01),

child: Text(

'Northwest missouri state univeristy',

style:

TextStyle(color: Colors.*white*, fontSize: 15.0),

),

)

],

),

)),

CustomeListTile(Icons.*person*, 'Profile',

() => {Navigator.*of*(context).pushNamed(profilePage.*route*)}),

CustomeListTile(Icons.*map*, 'Map',

() => {Navigator.*of*(context).pushNamed(mapPage.*route*)}), //mapPage

CustomeListTile(Icons.*report*, 'Report',

() => {Navigator.*of*(context).pushNamed(reportPage.*route*)}),

CustomeListTile(Icons.*question\_answer*, 'FAQ',

() => {Navigator.*of*(context).pushNamed(FaQPage.*route*)}),

CustomeListTile(Icons.*chat*, 'Live Chat',

() => {Navigator.*of*(context).pushNamed(ChatScreen.*route*)}),

CustomeListTile(Icons.*phone*, 'Contact',

() => {Navigator.*of*(context).pushNamed(ContactUs.*route*)}),

CustomeListTileNew(Icons.*gavel*, 'Ticket',

() => {Navigator.*of*(context).pushNamed(ticket.*route*)}),

CustomeListTile(Icons.*lock*, 'Logout', () async {

final user = await \_auth.signOut();

Navigator.*of*(context).pushNamed(LoginPage.*route*);

Toast.*show*("Logged out Successfully", context, duration: Toast.*LENGTH\_LONG*, gravity: Toast.*BOTTOM*);

}),

],

),

));

}

}

class CustomeListTile extends StatelessWidget {

IconData icon;

String text;

Function onTap;

CustomeListTile(this.icon, this.text, this.onTap);

@override

Widget build(BuildContext context) {

return Padding(

padding: const EdgeInsets.fromLTRB(8.0, 0, 8.0, 0),

child: Container(

decoration: BoxDecoration(

border: Border(bottom: BorderSide(color: Colors.*grey*.shade400))),

child: InkWell(

splashColor: Colors.*blueGrey*,

onTap: onTap,

child: Container(

height: 50,

child: Row(

mainAxisAlignment: MainAxisAlignment.spaceBetween,

children: <Widget>[

Row(

children: <Widget>[

Icon(icon),

Padding(

padding: const EdgeInsets.all(8.0),

child: Text(text, style: TextStyle(fontSize: 16.0)),

),

],

),

Icon(Icons.*arrow\_right*)

],

),

),

),

),

);

}

}

class CustomeListTileNew extends StatelessWidget {

IconData icon;

String text,route;

Function onTap;

CustomeListTileNew(this.icon, this.text, this.onTap);

@override

Widget build(BuildContext context) {

return Padding(

padding: const EdgeInsets.fromLTRB(8.0, 0, 8.0, 0),

child: Container(

decoration: BoxDecoration(

border: Border(bottom: BorderSide(color: Colors.*grey*.shade400))),

child: InkWell(

splashColor: Colors.*blueGrey*,

onTap:() => {

if(\_UserHomePageState.*autho*)

Navigator.*of*(context).pushNamed(ticket.*route*)

else

{

Navigator.*pushNamed*(context,UserHomePage.*route*),

Toast.*show*("Logged out Successfully", context, duration: Toast.*LENGTH\_LONG*, gravity: Toast.*BOTTOM*),

}

},

child: Container(

height: 50,

child: Row(

mainAxisAlignment: MainAxisAlignment.spaceBetween,

children: <Widget>[

Row(

children: <Widget>[

Icon(icon),

Padding(

padding: const EdgeInsets.all(8.0),

child: Text(text, style: TextStyle(fontSize: 16.0)),

),

],

),

Icon(Icons.*arrow\_right*)

],

),

),

),

),

);

}

}

**6.3.6 Ticket Page**

Admin user can navigate to this by clicking on the ticket tab in the hamburger menu. In this page admin can raise or drop tickets to the user.

Below is the code for routing to this page and the firebase instances declaration.

final \_firestore = Firestore.*instance*;

FirebaseUser loggedInUser;

class ticket extends StatefulWidget {

static const String *route* = '/ticket';

@override

\_ticketState createState() => \_ticketState();

}

class \_ticketState extends State<ticket> {

final \_auth = FirebaseAuth.*instance*;

bool autho=false;

@override

void initState() {

getCurrentUser();

}

This page can be accessed only by the admins.Below is the code for user check.

**void getCurrentUser() async {**

**try {**

**final user = await \_auth.currentUser();**

**if (user != null) loggedInUser = user;**

**print(loggedInUser.email);**

**} catch (e) {**

**print(e);**

**}**

**}**

Below is the code for the ticket raising and dropping functionality

**void Ticket(String regNumber, String reason, bool action) async {**

**String UID = '', docId;**

**int tickets = 0;**

**// raise ticket, action = true**

**print('regNumber $regNumber reason $reason action $action');**

**final users =await \_firestore.collection('User').getDocuments();**

**for (var user in users.documents) {**

**print(user.data['Registration\_ID']);**

**if (user.data['Registration\_ID'] == regNumber) {**

**UID = user.data['User\_ID'];**

**docId = user.documentID;**

**// print(UID);**

**if (action)**

**{ tickets = user.data['Num\_Tickets'] + 1; //adding new ticket**

**Toast.*show*("Successfully Raised a ticket", context, duration: Toast.*LENGTH\_SHORT*, gravity: Toast.*BOTTOM*);**

**}**

**else**

**{ tickets = user.data['Num\_Tickets'] - 1; //removing ticket**

**Toast.*show*("Successfully Droped a ticket", context, duration: Toast.*LENGTH\_SHORT*, gravity: Toast.*BOTTOM*);**

**}**

**// print('tickets $tickets');**

**\_firestore**

**.collection('User')**

**.document(docId)**

**.updateData({'Num\_Tickets': tickets});**

**// print('updated user table');**

**break;**

**//user.data.update('Num\_Tickets', (tickets) => null);**

**} else**

**print('invalid vehicle number');**

**Toast.*show*("Invalid Vehicle number", context, duration: Toast.*LENGTH\_SHORT*, gravity: Toast.*BOTTOM*);**

**}**

**var date = new DateTime.now();**

**if (action) {**

**// print('started session 2 $date');**

**\_firestore.collection('Ticket').add({**

**'User\_ID': UID,**

**'Description': reason,**

**'Ticket\_ID': 0,**

**'Date': date.toString(),**

**'Active': true**

**});**

**} //updated ticket table**

**}**

**@override**

**Widget buildButton(**

**String buttonText, double buttonHeight, Color buttonColor) {**

**return Container(**

**// height: MediaQuery.of(context).size.height \* 0.1 \* buttonHeight,**

**color: buttonColor,**

**child: FlatButton(**

**shape: RoundedRectangleBorder(**

**borderRadius: BorderRadius.circular(0.0),**

**side: BorderSide(**

**color: Colors.*white*, width: 2, style: BorderStyle.solid)),**

**padding: EdgeInsets.all(16.0),**

**//onPressed: () => buttonPressed(buttonText),**

**child: Text(**

**buttonText,**

**style: TextStyle(**

**fontSize: 30.0,**

**fontWeight: FontWeight.*normal*,**

**color: Colors.*white*),**

**)),**

**);**

**}**

**@override**

**Widget build(BuildContext context) {**

**String vehicleNum, reason;**

**bool action;**

**return Scaffold(**

**appBar: ApplicationBar(),**

**body: SafeArea(**

**child: ListView(**

**// mainAxisAlignment: MainAxisAlignment.center,**

**children: <Widget>[**

**Container(**

**alignment: Alignment.*center*,**

**padding: EdgeInsets.all(10),**

**child: Text(**

**'Tickets',**

**style: TextStyle(fontWeight: FontWeight.*w600*, fontSize: 30),**

**),**

**),**

**Container(**

**alignment: Alignment.*center*,**

**padding: EdgeInsets.all(10),**

**child: Text(**

**'Raise or drop a ticket',**

**),**

**),**

**SizedBox(**

**height: 20,**

**),**

**Container(**

**padding: EdgeInsets.all(10),**

**child: TextFormField(**

**decoration: InputDecoration(**

**border: OutlineInputBorder(),**

**labelText: 'Vehicle Number',**

**),**

**onChanged: (value) {**

**vehicleNum = value;**

**},**

**),**

**),**

**Container(**

**padding: EdgeInsets.all(10),**

**child: TextFormField(**

**decoration: InputDecoration(**

**border: OutlineInputBorder(),**

**labelText: 'Reason for ticket',**

**),**

**onChanged: (value) {**

**reason = value;**

**},**

**),**

**),**

**Column(**

**crossAxisAlignment: CrossAxisAlignment.start,**

**children: [**

**Padding(**

**padding: const EdgeInsets.all(10.0),**

**child: ToggleSwitch(**

**minWidth: 90.0,**

**cornerRadius: 20.0,**

**activeBgColor: Colors.*cyan*,**

**activeFgColor: Colors.*white*,**

**inactiveBgColor: Colors.*grey*,**

**inactiveFgColor: Colors.*white*,**

**labels: ['Raise', 'Drop'],**

**icons: [Icons.*check*, Icons.*close*],**

**onToggle: (index) {**

**print('switched to: $index');**

**if (index == 0)**

**action = true;**

**else**

**action = false;**

**},**

**),**

**),**

**],**

**),**

**Padding(**

**padding: const EdgeInsets.all(20.0),**

**child: Row(**

**mainAxisAlignment: MainAxisAlignment.spaceBetween,**

**children: [**

**ButtonTheme(**

**minWidth: 150.0,**

**height: 50.0,**

**child: RaisedButton(**

**onPressed: () {**

**// Navigator.pop(context);**

**Ticket(vehicleNum, reason, action);**

**},**

**color: Colors.*teal*[400],**

**//blue[700],**

**shape: RoundedRectangleBorder(**

**borderRadius: BorderRadius.circular(18.0),**

**side: BorderSide(color: Colors.*grey*)),**

**splashColor: Colors.*green*,**

**child: Row(**

**children: [**

**Text('Proceed',**

**style: TextStyle(fontSize: 25.0)),**

**],**

**),**

**),**

**),**

**ButtonTheme(**

**minWidth: 150.0,**

**height: 50.0,**

**child: RaisedButton(**

**onPressed: () {**

**// Navigator.pop(context);**

**Navigator.*of*(context)**

**.pushReplacementNamed(UserHomePage.*route*);**

**},**

**color: Colors.*red*[400],**

**//blue[700],**

**shape: RoundedRectangleBorder(**

**borderRadius: BorderRadius.circular(18.0),**

**side: BorderSide(color: Colors.*grey*)),**

**splashColor: Colors.*green*,**

**child: Row(**

**children: [**

**Text('Cancel', style: TextStyle(fontSize: 25.0)),**

**],**

**),**

**),**

**),**

**],**

**),**

**),**

**],**

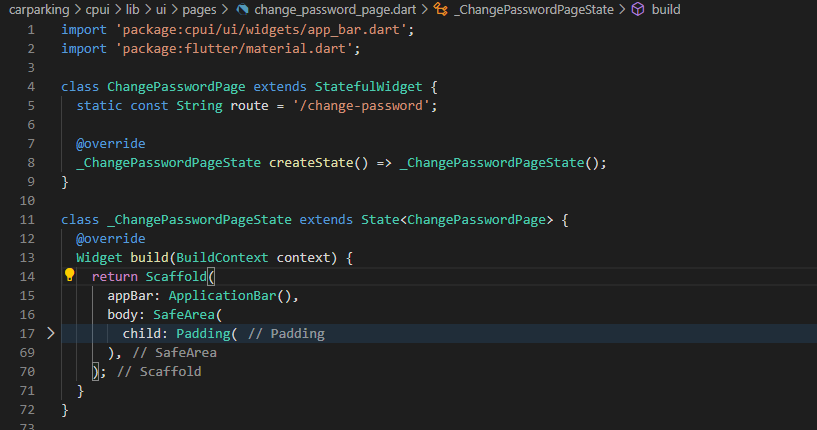
**),**

**));**

**}**

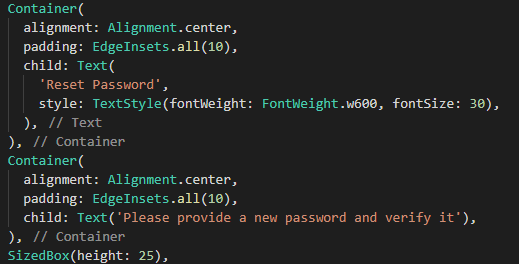
**6.3.7 Reset Password page**

This page implemented in change\_password\_page.dart as follow:



The folded line 17 which consists of 5 containers and layout as listview.

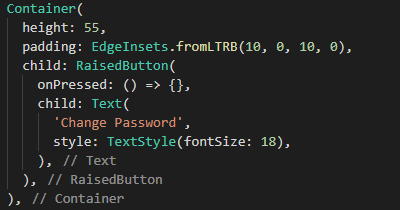
The first 2 containers are texts.



The following 2 are text fields.

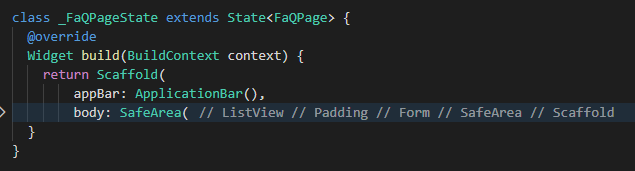


The last container is a button.

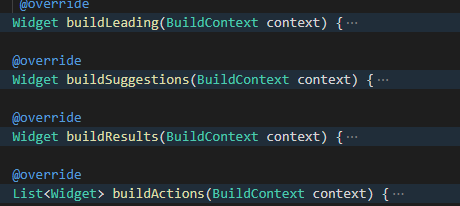


**6.3.8 FAQ page**

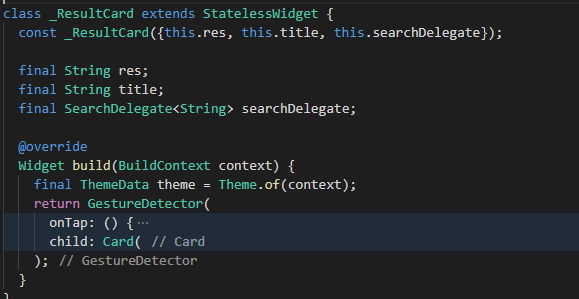
This page implemented in FaQPage.dart as follow:



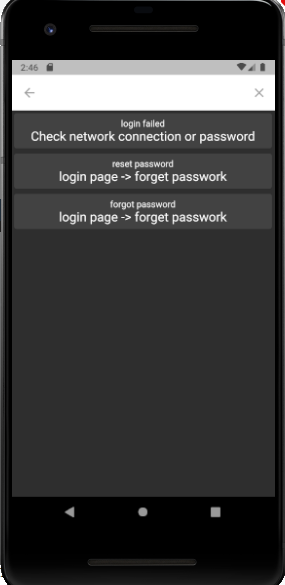
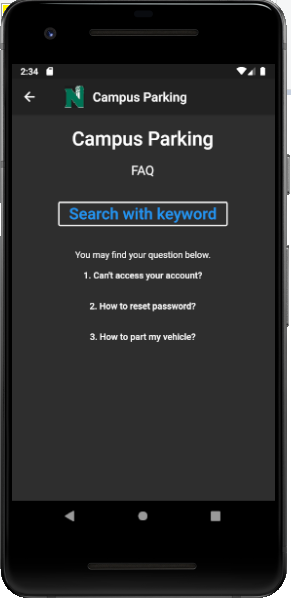
Following 4 widgets implemented the search function:



Search results are displayed in a list if card container:

****

Below is the capture of the default page and search page:

****

**6.3.9 Report Screen**

The code for the report screen is as follows.

The following includes all the import statements necessary and a firestore instance with a default method for the class.

import 'dart:io';

import 'package:flutter/material.dart';

import 'package:flutter/widgets.dart';

import 'package:flutter/cupertino.dart';

import 'package:firebase\_storage/firebase\_storage.dart';

import 'package:image\_cropper/image\_cropper.dart';

import 'package:image\_picker/image\_picker.dart';

import 'package:toast/toast.dart';

import 'package:campusparking/ui/widgets/app\_bar.dart';

import 'package:cloud\_firestore/cloud\_firestore.dart';

import 'package:campusparking/ui/pages/user\_home\_page.dart';

import 'package:campusparking/ui/pages/violationPage.dart';

final \_firestore = Firestore.instance;

/// ignore: camel\_case\_types

class reportPage extends StatefulWidget {

static const String route = '/report';

@override

\_reportPageState createState() => \_reportPageState();

}

The following code consists of a form key and all the variables necessary including methods for the image to be uploaded. The pick, crop and clear functions are called later when the image has to be uploaded by the user.

**// ignore: camel\_case\_types**

**class \_reportPageState extends State<reportPage> {**

**@override**

**void initState() {**

**super.initState();**

**}**

**final \_formKey = GlobalKey<FormState>();**

**bool loading = false;**

**// ignore: non\_constant\_identifier\_names**

**String vehicle\_no ='';**

**String image;**

**String description = '';**

**/// Active image file**

**File \_imageFile;**

**/// Select an image via gallery or camera**

**Future<void> \_pickImage(ImageSource source) async {**

**File selected = await ImagePicker.pickImage(source: source);**

**setState(() {**

**\_imageFile = selected;**

**});**

**}**

**/// Cropper plugin**

**Future<void> \_cropImage() async {**

**File cropped = await ImageCropper.cropImage(**

**sourcePath: \_imageFile.path,**

**toolbarColor: Colors.purple,**

**toolbarWidgetColor: Colors.white,**

**toolbarTitle: 'Crop It');**

**setState(() {**

**\_imageFile = cropped ?? \_imageFile;**

**});**

**}**

**/// Remove image**

**void \_clear() {**

**setState(() => \_imageFile = null);**

**}**

The code for the container of the report screen is below. This holds all the text fields, labels, and buttons including the validators.

**@override**

**Widget build(BuildContext context) {**

**return Scaffold(**

**appBar: ApplicationBar(),**

**body: SafeArea(**

**child: Form(**

**key: \_formKey,**

**child: Padding(**

**padding: EdgeInsets.all(10),**

**child: ListView(**

**children: <Widget>[**

**Container(**

**alignment: Alignment.center,**

**padding: EdgeInsets.all(10),**

**child: Text(**

**'Report',**

**style: TextStyle(**

**fontWeight: FontWeight.w600, fontSize: 30),**

**),**

**),**

**Container(**

**alignment: Alignment.center,**

**padding: EdgeInsets.all(10),**

**child: Text(**

**'Report any inconsistent data by entering the details below. Upload image from either gallery or camera',**

**),**

**),**

**Container(**

**padding: EdgeInsets.all(10),**

**child: TextFormField(**

**decoration: InputDecoration(**

**border: OutlineInputBorder(),**

**labelText: 'Enter vehicle number',**

**),**

**validator: (value) {**

**if (value.isEmpty) {**

**return 'Vehicle number cannot be empty';**

**}**

**\_formKey.currentState.save();**

**return null;**

**},**

**onChanged: (value) =>**

**setState(() => vehicle\_no = value),**

**),**

**),**

**Container(**

**padding: EdgeInsets.all(10),**

**child: TextFormField(**

**decoration: InputDecoration(**

**border: OutlineInputBorder(),**

**labelText: 'Enter description',**

**),**

**validator: (value) {**

**if (value.isEmpty) {**

**return 'Description cannot be empty';**

**}**

**\_formKey.currentState.save();**

**return null;**

**},**

**onChanged: (value) =>**

**setState(() => description = value),**

**),**

**),**

**Container(**

**margin: EdgeInsets.fromLTRB(10.0, 0.0, 0, 0),**

**child: Row(**

**children: <Widget>[**

**Text(**

**'Choose Image',**

**style: TextStyle(**

**fontWeight: FontWeight.w600,**

**fontFamily: 'Raleway',**

**fontSize: 20),**

**),**

**IconButton(**

**icon: Icon(Icons.photo\_camera),**

**onPressed: () => \_pickImage(ImageSource.camera),**

**color: Colors.grey[400],**

**),**

**IconButton(**

**icon: Icon(Icons.photo\_library),**

**onPressed: () => \_pickImage(ImageSource.gallery),**

**color: Colors.grey[400],**

**),**

**],**

**),**

**),**

**if (\_imageFile != null) ...[**

**Image.file(\_imageFile),**

**Row(**

**children: <Widget>[**

**FlatButton(**

**child: Icon(Icons.crop),**

**onPressed: \_cropImage,**

**),**

**FlatButton(**

**child: Icon(Icons.refresh),**

**onPressed: \_clear,**

**),**

**],**

**),**

**Uploader(file: \_imageFile)**

**],**

**Container(**

**height: 75,**

**margin: EdgeInsets.fromLTRB(17.0, 15.0, 0, 0),**

**child: Row(**

**children: <Widget>[**

**ButtonTheme(**

**minWidth: 150.0,**

**height: 45.0,**

**child: RaisedButton(**

**onPressed: () async {**

**if (\_formKey.currentState.validate()) {**

**\_firestore.collection('Report').add({**

**'Vehicle ID': vehicle\_no,**

**'Description': description,**

**});**

**Toast.show("Reported Successful", context, duration: Toast.LENGTH\_LONG, gravity: Toast.BOTTOM);**

**Navigator.of(context)**

**.pushReplacementNamed(reportPage.route);**

**}**

**},**

**color: Colors.lightBlue,**

**shape: RoundedRectangleBorder(**

**borderRadius: BorderRadius.circular(18.0),**

**side: BorderSide(color: Colors.grey)),**

**splashColor: Colors.blue,**

**child: Row(**

**children: [**

**Icon(**

**Icons.report,**

**size: 30.0,**

**color: Colors.grey[300],**

**),**

**Text('Report',**

**style: TextStyle(fontSize: 20.0,**

**),**

**),**

**],**

**),**

**),**

**),**

**SizedBox(**

**width: 45.0,**

**),**

**ButtonTheme(**

**minWidth: 150.0,**

**height: 45.0,**

**child: RaisedButton(**

**onPressed: () {**

**Navigator.of(context).pushReplacementNamed(UserHomePage.route);**

**},**

**color: Colors.red,**

**shape: RoundedRectangleBorder(**

**borderRadius: BorderRadius.circular(18.0),**

**side: BorderSide(color: Colors.grey)),**

**splashColor: Colors.red,**

**child: Row(**

**children: [**

**Icon(**

**Icons.clear,**

**size: 30.0,**

**color: Colors.grey[300],**

**),**

**Text('Clear', style: TextStyle(fontSize: 20.0)),**

**],**

**),**

**),**

**),**

**],**

**),**

**),**

**Container(**

**alignment: Alignment.center,**

**padding: EdgeInsets.all(10),**

**child: Row(**

**mainAxisAlignment: MainAxisAlignment.center,**

**children: [**

**FlatButton(**

**onPressed: () async => Navigator.of(context)**

**.pushReplacementNamed(ViolationPage.route),**

**child: Text(**

**'Click here to report a violation !',**

**)),**

**],**

**),**

**),**

**]**

**)**

**))));**

**}**

**}**

The code for uploading an image to the fire store is below.

**class Uploader extends StatefulWidget {**

**final File file;**

**Uploader({Key key, this.file}) : super(key: key);**

**createState() => \_UploaderState();**

**}**

**class \_UploaderState extends State<Uploader> {**

**final FirebaseStorage \_storage =**

**FirebaseStorage(storageBucket: 'gs://campus-parking-e2a4b.appspot.com/');**

**StorageUploadTask \_uploadTask;**

**void \_startUpload() {**

**String filePath = 'imagesReportPage/${DateTime.now()}.png';**

**setState(() {**

**\_uploadTask = \_storage.ref().child(filePath).putFile(widget.file);**

**});**

**}**

**@override**

**Widget build(BuildContext context) {**

**if (\_uploadTask != null) {**

**return StreamBuilder<StorageTaskEvent>(**

**stream: \_uploadTask.events,**

**builder: (context, snapshot) {**

**var event = snapshot?.data?.snapshot;**

**double progressPercent = event != null**

**? event.bytesTransferred / event.totalByteCount**

**: 0;**

**return Column(**

**children: [**

**if (\_uploadTask.isComplete) Text('Upload Successful!'),**

**if (\_uploadTask.isPaused)**

**FlatButton(**

**child: Icon(Icons.play\_arrow),**

**onPressed: \_uploadTask.resume,**

**),**

**if (\_uploadTask.isInProgress)**

**FlatButton(**

**child: Icon(Icons.pause),**

**onPressed: \_uploadTask.pause,**

**),**

**LinearProgressIndicator(value: progressPercent),**

**Text('${(progressPercent \* 100).toStringAsFixed(2)} % '),**

**],**

**);**

**});**

**} else {**

**return FlatButton.icon(**

**label: Text('Upload Image'),**

**color: Colors.teal[500],**

**icon: Icon(Icons.cloud\_upload),**

**onPressed: \_startUpload,**

**);**

**}**

**}**

**}**

**6.3.10 Report Violation Page**

The code for the report violation page is as follows. This includes the import statements and variables necessary.

**import 'dart:io';**

**import 'package:flutter/cupertino.dart';**

**import 'package:flutter/material.dart';**

**import 'package:flutter/widgets.dart';**

**import 'package:intl/intl.dart';**

**import 'package:firebase\_storage/firebase\_storage.dart';**

**import 'package:image\_cropper/image\_cropper.dart';**

**import 'package:image\_picker/image\_picker.dart';**

**import 'package:cloud\_firestore/cloud\_firestore.dart';**

**import 'package:toast/toast.dart';**

**import 'package:campusparking/ui/widgets/app\_bar.dart';**

**import 'package:campusparking/ui/pages/user\_home\_page.dart';**

**final \_firestore = Firestore.instance;**

**class ViolationPage extends StatefulWidget {**

**static const String route = '/violation\_page';**

**@override**

**ViolationPageState createState() => ViolationPageState();**

**}**

**class ViolationPageState extends State<ViolationPage> {**

**@override**

**void initState() {**

**super.initState();**

**}**

**DateTime selectedDate = DateTime.now();**

**final DateFormat dateFormat = DateFormat('yyyy-MM-dd HH:mm');**

**final \_formKey = GlobalKey<FormState>();**

**bool loading = false;**

**// ignore: non\_constant\_identifier\_names**

**String vehicle\_no = '';**

**// ignore: non\_constant\_identifier\_names**

**String parking\_lot\_id = '';**

**// ignore: non\_constant\_identifier\_names**

**String slot\_id = '';**

**String time = '';**

**String date = '';**

**String formatted;**

**String timer;**

The methods for uploading an image such as pick, crop and clear can be referred from the report page. The following code contains the container for this page. All the text fields, buttons and labels are contained here.

**@override**

**Widget build(BuildContext context) {**

**return Scaffold(**

**appBar: ApplicationBar(),**

**body: Form(**

**key: \_formKey,**

**child: SafeArea(**

**child: Padding(**

**padding: EdgeInsets.all(10),**

**child: ListView(**

**children: <Widget>[**

**Container(**

**alignment: Alignment.center,**

**padding: EdgeInsets.all(10),**

**child: Text(**

**'Report Violation',**

**style: TextStyle(**

**fontWeight: FontWeight.w600,**

**fontSize: 30),**

**),**

**),**

**Container(**

**alignment: Alignment.center,**

**padding: EdgeInsets.all(10),**

**child: Text(**

**'Report unauthorized parking here',**

**),**

**),**

**Container(**

**padding: EdgeInsets.all(15),**

**child: TextFormField(**

**decoration: InputDecoration(**

**border: OutlineInputBorder(),**

**labelText: 'Enter vehicle number',**

**),**

**validator: (value) {**

**if (value.isEmpty) {**

**return 'Vehicle number cannot be empty';**

**}**

**\_formKey.currentState.save();**

**return null;**

**},**

**onChanged: (value) =>**

**setState(() => vehicle\_no = value),**

**),**

**),**

**Container(**

**padding: EdgeInsets.all(15),**

**child: TextFormField(**

**decoration: InputDecoration(**

**border: OutlineInputBorder(),**

**labelText: 'Enter parking lot ID',**

**),**

**validator: (value) {**

**if (value.isEmpty) {**

**return 'Parking lot ID cannot be empty';**

**}**

**\_formKey.currentState.save();**

**return null;**

**},**

**onChanged: (value) =>**

**setState(() => parking\_lot\_id = value),**

**),**

**),**

**Container(**

**padding: EdgeInsets.all(15),**

**child: TextFormField(**

**decoration: InputDecoration(**

**border: OutlineInputBorder(),**

**labelText: 'Enter slot ID',**

**),**

**validator: (value) {**

**if (value.isEmpty) {**

**return 'Slot ID cannot be empty';**

**}**

**\_formKey.currentState.save();**

**return null;**

**},**

**onChanged: (value) =>**

**setState(() => slot\_id = value),**

**),**

**),**

**Container(**

**alignment: Alignment.center,**

**padding: EdgeInsets.all(10),**

**margin:**

**EdgeInsets.fromLTRB(0.0, 0.0, 16.0, 0.0),**

**child: Row(**

**mainAxisAlignment:**

**MainAxisAlignment.spaceAround,**

**children: <Widget>[**

**RaisedButton(**

**onPressed: () async {**

**final selectedDate =**

**await \_selectDateTime(context);**

**if (selectedDate == null) return;**

**// Formatting the date**

**var formatter =**

**new DateFormat('yyyy-MMM-dd');**

**formatted =**

**formatter.format(selectedDate);**

**final selectedTime =**

**await \_selectTime(context);**

**if (selectedTime == null) return;**

**// Formatting the time**

**final dt = DateTime(**

**1969,**

**1,**

**1,**

**selectedTime.hour,**

**selectedTime.minute);**

**var timerfor = new DateFormat('HH:mm');**

**timer = timerfor.format(dt);**

**print(timer);**

**print(formatted);**

**setState(() {**

**this.selectedDate = DateTime(**

**selectedDate.year,**

**selectedDate.month,**

**selectedDate.day,**

**selectedTime.hour,**

**selectedTime.minute,**

**);**

**});**

**},**

**child: Text(**

**'Choose Date & Time',**

**style: TextStyle(fontSize: 18),**

**),**

**),**

**Text(**

**dateFormat.format(selectedDate),**

**style: TextStyle(fontSize: 15),**

**),**

**],**

**),**

**),**

**Container(**

**margin:**

**EdgeInsets.fromLTRB(15.0, 0.0, 0.0, 0.0),**

**child: Row(**

**children: <Widget>[**

**Text(**

**'Choose Image',**

**style: TextStyle(**

**fontWeight: FontWeight.w600,**

**fontSize: 20),**

**),**

**IconButton(**

**icon: Icon(Icons.photo\_camera),**

**onPressed: () =>**

**\_pickImage(ImageSource.camera),**

**color: Colors.grey[400],**

**),**

**IconButton(**

**icon: Icon(Icons.photo\_library),**

**onPressed: () =>**

**\_pickImage(ImageSource.gallery),**

**color: Colors.grey[400],**

**),**

**],**

**),**

**),**

**if (\_imageFile != null) ...[**

**Image.file(\_imageFile),**

**Row(**

**children: <Widget>[**

**FlatButton(**

**child: Icon(Icons.crop),**

**onPressed: \_cropImage,**

**),**

**FlatButton(**

**child: Icon(Icons.refresh),**

**onPressed: \_clear,**

**),**

**],**

**),**

**Uploader(file: \_imageFile)**

**],**

**Container(**

**child: Row(**

**mainAxisAlignment:**

**MainAxisAlignment.spaceEvenly,**

**children: <Widget>[**

**ButtonTheme(**

**minWidth: 150.0,**

**height: 45.0,**

**child: RaisedButton(**

**onPressed: () async {**

**if (\_formKey.currentState**

**.validate()) {**

**setState(() => loading = true);**

**print(date);**

**print(time);**

**\_firestore.collection('Violations').add({**

**'Vehicle ID': vehicle\_no,**

**'ParkingLot ID': parking\_lot\_id,**

**'Slot ID' : slot\_id,**

**'Time' : timer,**

**'Date' : formatted,**

**});**

**Toast.show("Reported Violation Successful", context, duration: Toast.LENGTH\_LONG, gravity: Toast.BOTTOM);**

**Navigator.of(context)**

**.pushReplacementNamed(**

**ViolationPage.route);**

**}**

**},**

**color: Colors.lightBlue,**

**shape: RoundedRectangleBorder(**

**borderRadius:**

**BorderRadius.circular(18.0),**

**side:**

**BorderSide(color: Colors.grey)),**

**splashColor: Colors.blue,**

**child: Row(**

**children: [**

**Icon(**

**Icons.report,**

**size: 30.0,**

**color: Colors.grey[300],**

**),**

**Text(**

**'Report',**

**style: TextStyle(**

**fontSize: 20.0,**

**),**

**),**

**],**

**),**

**),**

**),**

**SizedBox(**

**width: 20.0,**

**),**

**ButtonTheme(**

**minWidth: 150.0,**

**height: 45.0,**

**child: RaisedButton(**

**onPressed: () {**

**Navigator.of(context)**

**.pushReplacementNamed(**

**UserHomePage.route);**

**},**

**color: Colors.red,**

**shape: RoundedRectangleBorder(**

**borderRadius:**

**BorderRadius.circular(18.0),**

**side:**

**BorderSide(color: Colors.grey)),**

**splashColor: Colors.red,**

**child: Row(**

**children: [**

**Icon(**

**Icons.clear,**

**size: 30.0,**

**color: Colors.grey[300],**

**),**

**Text('Clear',**

**style:**

**TextStyle(fontSize: 20.0)),**

**],**

**),**

**),**

**),**

**],**

**),**

**),**

**],**

**),**

**),**

**),**

**),**

**);**

**}**

**}**

**6.3.11 Parking page**

This page will be having a list of parking slots. All the blue parking slots are available slots and the red are occupied slots.

This is the routing for the page

final \_firestore = Firestore.*instance*;

FirebaseUser loggedInUser;

class Parking extends StatefulWidget {

static const String *route* = '/parking';

@override

\_ParkingState createState() => \_ParkingState();

}

Here is the code for parking slots color changing and occupy

class \_ParkingState extends State<Parking> {

final \_auth = FirebaseAuth.*instance*;

bool Is\_Occupied, Slot\_Type, changeColor = false, changeColor1 = false;

String User\_ID, document\_id, Lot\_Name;

int Lot\_ID, Slot\_ID;

@override

void initState() {

super.initState();

getCurrentUser();

UserHomePage.*available* = 0;

Lot\_Name = UserHomePage.*Lot\_name*.substring(2);

availableSlots();

}

//code for available number of slots

void availableSlots() async {

print('this is available slot function');

print(UserHomePage.*Lot\_name*);

await for (var snapshot in \_firestore.collection('slot').snapshots()) {

for (var slot in snapshot.documentChanges) {

//print('got inside for');

if (slot.document.documentID.contains(UserHomePage.*Lot\_name*)) {

//print('got inside 1st if');

if (slot.document.data['Is\_Occupied']) {

setState(() {

UserHomePage.*available*--;

print(slot.document.documentID);

});

} else {

setState(() {

UserHomePage.*available*++;

// print(slot.document.documentID);

});

}

}

//print('Available ${UserHomePage.available}');

}

}

}

//code for occupying a slot

void occupySlot() {

var occupied =

\_firestore.collection('slot').document(document\_id).updateData({

'Is\_Occupied': true,

'User\_ID': User\_ID,

});

setState(() {

changeColor1 = true;

});

print('requested slot occupied successfully');

Toast.*show*("You have Ocuupied the slot", context,

duration: Toast.*LENGTH\_LONG*, gravity: Toast.*BOTTOM*);

}

//code for vacating a slot

void vacateSlot() {

var occupied =

\_firestore.collection('slot').document(document\_id).updateData({

'Is\_Occupied': false,

'User\_ID': User\_ID,

});

setState(() {

changeColor1 = false;

});

Toast.*show*("You have Vacated the slot", context,

duration: Toast.*LENGTH\_LONG*, gravity: Toast.*BOTTOM*);

print('requested slot vacated successfully');

}

//code for reserving a slot: for admin only

void reserveSlot(String buttonText) {

document\_id = UserHomePage.*Lot\_name* + '-' + buttonText;

var occupied =

\_firestore.collection('slot').document(document\_id).updateData({

'Is\_Reserved': true,

'User\_ID': User\_ID,

});

print('requested slot reserved successfully');

Toast.*show*("You have Reserved the slot", context,

duration: Toast.*LENGTH\_LONG*, gravity: Toast.*BOTTOM*);

}

//get current user

void getCurrentUser() async {

try {

final user = await \_auth.currentUser();

if (user != null) {

loggedInUser = user;

print(loggedInUser.email);

User\_ID = loggedInUser.email;

}

} catch (e) {

print(e);

}

}

// chaging button color

void changeButtonColor() async {}

//checking whether slot is vacant or occupied

Future buttonPressed(String buttonText) async {

document\_id = UserHomePage.*Lot\_name* + '-' + buttonText;

print('button pressed $document\_id');

await for (var snapshot

in \_firestore.collection('slot').document(document\_id).snapshots()) {

var data = snapshot.data;

// if (data['User\_ID'] == loggedInUser.email)

// {

if (!data['Is\_Occupied']) {

occupySlot();

break;

} else {

if (data['User\_ID'] == loggedInUser.email) {

vacateSlot();

break;

} else {

print("This user is not allowed to perform the action " +

loggedInUser.email);

Toast.*show*("Someone else have already occupied this slot", context,

duration: Toast.*LENGTH\_LONG*, gravity: Toast.*BOTTOM*);

}

}

}

}

Widget buildButton(String buttonText) {

return Container(

child: GestureDetector(

child: RaisedButton(

shape: RoundedRectangleBorder(

borderRadius: BorderRadius.circular(0.0),

side: BorderSide(

color: Colors.*white*, width: 2, style: BorderStyle.solid)),

padding: EdgeInsets.all(16.0),

color: \_hasBeenPressed ? Colors.*red*[300] : Colors.*green*[300],

//Colors.lightBlueAccent,

onLongPress: () {

reserveSlot(buttonText);

},

onPressed: () => {

// setState(() {

buttonPressed(buttonText),

setState(() {

\_hasBeenPressed = !\_hasBeenPressed;

})

// }),

},

child: Text(

buttonText,

style: TextStyle(

fontSize: 30.0,

fontWeight: FontWeight.*normal*,

color: Colors.*white*),

)),

),

);

}

bool \_hasBeenPressed = false;

bool \_hasBeenPressed1 = false;

bool \_hasBeenPressed2 = false;

bool \_hasBeenPressed3 = false;

bool \_hasBeenPressed4 = false;

bool \_hasBeenPressed5 = false;

bool \_hasBeenPressed6 = false;

bool \_hasBeenPressed7 = false;

bool \_hasBeenPressed8 = false;

bool \_hasBeenPressed9 = false;

bool \_hasBeenPressed10 = false;

bool \_hasBeenPressed11 = false;

bool \_hasBeenPressed12 = false;

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(

title: Row(

children: [

Padding(

padding: EdgeInsets.symmetric(vertical: 10, horizontal: 10),

child: Container(

height: 35.0,

width: 35.0,

child: Image.asset('assets/images/nwm\_logo.png')

),

),

Text("Campus Parking"),

],

),

leading: new IconButton(

icon: new Icon(Icons.*arrow\_back*),

onPressed: () {

Navigator.*of*(context)

.pushReplacementNamed(UserHomePage.*route*);

},

),

),

body: SafeArea(

child: ListView(

// mainAxisAlignment: MainAxisAlignment.center,

children: <Widget>[

Align(

alignment: Alignment.*topRight*,

child: Tooltip(

message: 'Long Press for Reserve \n\n Press for vacate and occupy',

child: FlatButton(

child: Icon(Icons.*help*),

),

),

),

Container(

alignment: Alignment.*center*,

padding: EdgeInsets.all(10),

child: Text(

'Parking Area - $Lot\_Name',

style: TextStyle(fontWeight: FontWeight.*w600*, fontSize: 30),

),

),

Container(

alignment: Alignment.*center*,

padding: EdgeInsets.all(10),

child: Text(

'Occupy or vacate a slot',

),

),

SizedBox(

height: 20,

),

Center(

child: Row(

mainAxisAlignment: MainAxisAlignment.center,

children: [

Text(

//availableSlots(),

' Available slots: ',

style: TextStyle(

color: Colors.*grey*[100],

fontSize: 20,

fontFamily: 'SourceSansPro',

fontWeight: FontWeight.*bold*,

),

),

Text(

UserHomePage.*available*.toString(),

style: TextStyle(

color: Colors.*grey*[100],

fontSize: 20,

fontFamily: 'SourceSansPro',

fontWeight: FontWeight.*bold*,

),

),

],

),

),

SizedBox(

height: 40,

),

Row(

mainAxisAlignment: MainAxisAlignment.center,

children: <Widget>[

Container(

width: MediaQuery.*of*(context).size.width \* .75,

child: Table(

children: [

TableRow(children: [

Container(

child: GestureDetector(

child: RaisedButton(

shape: RoundedRectangleBorder(

borderRadius: BorderRadius.circular(0.0),

side: BorderSide(

color: Colors.*white*, width: 2, style: BorderStyle.solid)),

padding: EdgeInsets.all(16.0),

color: \_hasBeenPressed1 ? Colors.*red*[300] : Colors.*green*[300],

//Colors.lightBlueAccent,

onLongPress: () {

reserveSlot("1");

},

onPressed: () => {

// setState(() {

buttonPressed("1"),

setState(() {

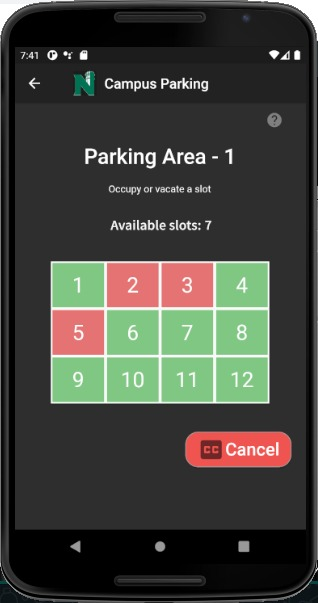
\_hasBeenPressed1 = !\_hasBeenPressed1;

})

// }),

},

Below is the screenshot of the page



**6.3.12 Hamburger Menu**

This is a drawer present on the navigation menu and used to route to different pages such as profile, ticket, contact us and other pages. Below is the simple code for this hamburger menu.

drawer: Drawer(

child: ListView(

padding: EdgeInsets.zero,

children: <Widget>[

DrawerHeader(

decoration: BoxDecoration(

gradient: LinearGradient(colors: <Color>[

Colors.black54,

Colors.blueGrey[500]

])),

child: Container(

child: Column(

children: <Widget>[

Material(

borderRadius: BorderRadius.all(Radius.circular(50.0)),

child: Image.asset(

'assets/images/nwm\_logo.png',

width: 100,

height: 100,

),

),

Padding(

padding: EdgeInsets.all(0.01),

child: Text(

'Northwest missouri state univeristy',

style:

TextStyle(color: Colors.white, fontSize: 15.0),

),

)

],

),

)),

CustomeListTile(Icons.person, 'Profile',

() => {Navigator.of(context).pushNamed(profilePage.route)}),

CustomeListTile(Icons.map, 'Map',

() => {Navigator.of(context).pushNamed(mapPage.route)}), //mapPage

CustomeListTile(Icons.report, 'Report',

() => {Navigator.of(context).pushNamed(reportPage.route)}),

CustomeListTile(Icons.question\_answer, 'FAQ',

() => {Navigator.of(context).pushNamed(FaQPage.route)}),

CustomeListTile(Icons.chat, 'Live Chat',

() => {Navigator.of(context).pushNamed(ChatScreen.route)}),

CustomeListTile(Icons.phone, 'Contact',

() => {Navigator.of(context).pushNamed(ContactUs.route)}),

CustomeListTileNew(Icons.gavel, 'Ticket',

() => {Navigator.of(context).pushNamed(ticket.route)}),

CustomeListTile(Icons.lock, 'Logout', () async {

final user = await \_auth.signOut();

Navigator.of(context).pushNamed(LoginPage.route);

Toast.show("Logged out Successfully", context, duration: Toast.LENGTH\_LONG, gravity: Toast.BOTTOM);

}),

// () => {Navigator.of(context).pushNamed(LoginPage.route)}),

// onPressed: () async {

// try {

// print(\_email);

// print(\_password);

// final user = await \_auth.signInWithEmailAndPassword(

// email: \_email, password: \_password);

// if (user != null)

// Navigator.pushNamed(context,UserHomePage.route);

// } catch (e) {

// print(e);

// }

// }

],

),

));

}

}

**6.3.13 Profile Page**

This is a profile page where the user gets to know about his tickets raised or dropped and other details about the user. Below is the code for the profile page.

Code for routing

class profilePage extends StatefulWidget {

static const String route = '/profile-page';

@override

\_profilePageState createState() => \_profilePageState();

}

Remaining Code for profile page

class \_profilePageState extends State<profilePage> {

final \_auth = FirebaseAuth.instance;

String registration\_num = '', userID = '', user\_name = '';

int num\_tickets, vehicle\_type;

@override

void initState() {

super.initState();

getUserDetails();

getCurrentUser();

// print(UserHomePage.loggedInUser);

}

void getCurrentUser() async {

try {

final user = await \_auth.currentUser();

if (user != null) loggedInUser = user;

print(loggedInUser.email);

} catch (e) {

print(e);

}

}

void getUserDetails() async {

final users = await \_firestore.collection('User').getDocuments();

for (var user in users.documents) {

if (user.data['User\_ID'] == loggedInUser.email) {

print('found required user');

setState(() {

registration\_num = user.data['Registration\_ID'];

userID = user.data['User\_ID'];

num\_tickets = user.data['Num\_Tickets'];

user\_name = user.data['User\_Name'];

});

}

}

final vehicles = await \_firestore.collection('Vehicle').getDocuments();

for (var vehicle in vehicles.documents) {

if (vehicle.data['Registration\_Number'] == registration\_num) {

setState(() {

vehicle\_type = vehicle.data['Vehicle\_Type'];

});

break;

}

}

}

@override

Widget build(BuildContext context) {

return Scaffold(

backgroundColor: Colors.grey[900],

appBar: ApplicationBar(),

body: SafeArea(

child: Column(

mainAxisAlignment: MainAxisAlignment.center,

children: <Widget>[

CircleAvatar(

radius: 50.0,

// backgroundColor: Colors.blueAccent,

backgroundImage: AssetImage('assets/images/user.jpg'),

),

Text(user\_name,

style: TextStyle(

color: Colors.white,

fontFamily: 'DancingScript',

fontSize: 30.0,

fontWeight: FontWeight.bold,

)),

SizedBox(

height: 30.0,

width: 150.0,

child: Divider(color: Colors.grey[600])),

Card(

color: Colors.black26,

margin: EdgeInsets.symmetric(vertical: 10.0, horizontal: 10.0),

child: ListTile(

leading: Icon(

Icons.motorcycle,

size: 50.0,

color: Colors.grey[400],

),

//

title: Text(

registration\_num,

style: TextStyle(

color: Colors.grey[100],

fontSize: 25,

fontFamily: 'SourceSansPro',

fontWeight: FontWeight.bold,

),

),

subtitle: Text(

'Vehicle number',

style: TextStyle(

color: Colors.grey[100],

fontSize: 15,

fontFamily: 'SourceSansPro',

fontWeight: FontWeight.bold,

),

),

),

),

Card(

color: Colors.black26,

//padding: EdgeInsets.all(10.0),

margin: EdgeInsets.symmetric(vertical: 10.0, horizontal: 10.0),

child: ListTile(

leading: Icon(

Icons.gavel,

size: 50.0,

color: Colors.grey[400],

),

title: Text(

num\_tickets.toString(),

style: TextStyle(

color: Colors.grey[100],

fontSize: 25,

fontFamily: 'SourceSansPro',

fontWeight: FontWeight.bold,

),

),

subtitle: Text(

'Number of tickets',

style: TextStyle(

color: Colors.grey[100],

fontSize: 15,

fontFamily: 'SourceSansPro',

fontWeight: FontWeight.bold,

),

),

),

),

Card(

color: Colors.black26,

//padding: EdgeInsets.all(10.0),

margin: EdgeInsets.symmetric(vertical: 10.0, horizontal: 10.0),

child: ListTile(

leading: Icon(

Icons.event\_seat,

size: 50.0,

color: Colors.grey[400],

),

//vehicle\_type

title: Text(

vehicle\_type.toString() + ' Wheeler',

style: TextStyle(

color: Colors.grey[100],

fontSize: 25,

fontFamily: 'SourceSansPro',

fontWeight: FontWeight.bold,

),

),

subtitle: Text(

'Vehicle type',

style: TextStyle(

color: Colors.grey[100],

fontSize: 15,

fontFamily: 'SourceSansPro',

fontWeight: FontWeight.bold,

),

),

),

),

])),

);

}

}

**6.3.14 Contact us**

This is the only static UI page in this application which gives information about admin phone number and email. Below is the code for contact us page.

class \_ContactUsState extends State<ContactUs> {

@override

void initState() {

super.initState();

}

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: ApplicationBar(),

body: SafeArea(

child: ListView(

// mainAxisAlignment: MainAxisAlignment.center,

children: <Widget>[

Container(

alignment: Alignment.center,

padding: EdgeInsets.all(10),

child: Text(

'Contact Us',

style: TextStyle(fontWeight: FontWeight.w600, fontSize: 30),

),

),

Container(

alignment: Alignment.center,

padding: EdgeInsets.all(10),

child: Text(

'Contact university for any queries',

),

),

Card(

color: Colors.black26,

margin:EdgeInsets.symmetric(vertical: 10.0,horizontal: 10.0),

child: ListTile(

leading: Icon(

Icons.phone\_in\_talk,

size: 40.0,

color: Colors.grey[400],

),

title: Text('(660) 562-1212',

style: TextStyle(

color: Colors.grey[100],

fontSize: 25,

fontFamily: 'SourceSansPro',

fontWeight: FontWeight.bold,

),

),

),

),

Card(

color: Colors.black26,

margin:EdgeInsets.symmetric(vertical: 10.0,horizontal: 10.0),

child: ListTile(

leading: Icon(

Icons.email,

size: 40.0,

color: Colors.grey[400],

),

title: Text('CParking@nwmissouri.edu',

style: TextStyle(

color: Colors.grey[100],

fontSize: 25,

fontFamily: 'SourceSansPro',

fontWeight: FontWeight.bold,

),

),

),

),

Container(

alignment: Alignment.center,

padding: EdgeInsets.all(10),

child: Text(

'Call us for immidiate assistance or you can drop us an email and expect response in 1-2 working days ',

),

),

])));

}

}

**6.3.15 Map Page**

Map page helps users to get the parking lot locations which are around campus.This page helps users to get to know the number of parking locations available in each parking lot.

import 'package:campusparking/ui/pages/Parking.dart';

import 'package:campusparking/ui/pages/user\_home\_page.dart';

import 'package:campusparking/ui/pages/user\_home\_page.dart';

import 'package:cloud\_firestore/cloud\_firestore.dart';

import 'package:firebase\_auth/firebase\_auth.dart';

import 'package:flutter/cupertino.dart';

import 'package:flutter/material.dart';

import 'package:google\_maps\_flutter/google\_maps\_flutter.dart';

import 'ParkingLotList.dart';

import 'user\_home\_page.dart';

final \_firestore = Firestore.instance;

FirebaseUser loggedInUser;

// ignore: camel\_case\_types

class mapPage extends StatefulWidget {

static const String route = '/map';

@override

\_mapPageState createState() => \_mapPageState();

}

// ignore: camel\_case\_types

class \_mapPageState extends State<mapPage> {

GoogleMapController mapController;

List<Marker> allMarkers = [];

PageController \_pageController;

int prevPage;

@override

void initState() {

super.initState();

UserHomePage.Lot\_name = "PA1";

getAvailableSlots();

UserHomePage.Lot\_name = "PA2";

getAvailableSlots();

UserHomePage.Lot\_name = "PA3";

getAvailableSlots();

UserHomePage.Lot\_name = "PA4";

getAvailableSlots();

\_callLast();

// print('Available Maps ${UserHomePage.available}');

\_pageController = PageController(initialPage: 1, viewportFraction: 0.8)

..addListener(\_onScroll);

}

getAvailableSlots() {

UserHomePage.available = 0;

\_availableSlots();

}

\_callLast() {

lots.forEach((element) {

print('inside another for loop');

allMarkers.add(Marker(

markerId: MarkerId(element.lotName),

draggable: false,

infoWindow: InfoWindow(

title: element.lotName,

snippet: 'Available Maps ${element.availableParkingLots}',

),

position: element.locationCoords));

});

}

void \_onScroll() {

if (\_pageController.page.toInt() != prevPage) {

prevPage = \_pageController.page.toInt();

moveCamera();

}

}

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(

title: Text('University Map'),

),

body: Stack(children: <Widget>[

Container(

height: MediaQuery.of(context).size.height - 80.0,

width: MediaQuery.of(context).size.width,

child: GoogleMap(

mapType: MapType.hybrid,

initialCameraPosition: CameraPosition(

target: LatLng(40.3520, -94.8825), zoom: 15.0),

markers: Set.from(allMarkers),

onMapCreated: onMapCreated),

),

// ignore: missing\_required\_param

Positioned(

bottom: 20.0,

child: Container(

height: 200.0,

width: MediaQuery.of(context).size.width,

// child: PageView.builder(

// controller: \_pageController,

// itemCount: lots.length,

// itemBuilder: (BuildContext context, int index) {

// return \_LotsList(index);

// },

)),

// ),

]));

}

void onMapCreated(controller) {

setState(() {

mapController = controller;

});

}

moveCamera() {

mapController.animateCamera(CameraUpdate.newCameraPosition(CameraPosition(

target: lots[\_pageController.page.toInt()].locationCoords,

zoom: 14.0,

bearing: 30.0,

tilt: 45.0)));

}

//code for available number of slots

\_availableSlots() {

print('this is available slot function');

print(UserHomePage.Lot\_name);

for (var slot in UserHomePage.snapshotGlobal.documentChanges) {

//print('got inside for');

if (slot.document.documentID.contains(UserHomePage.Lot\_name)) {

print("check parking page");

print(slot.document.documentID);

print(UserHomePage.Lot\_name);

//print('got inside 1st if');

if (slot.document.data['Is\_Occupied']) {

setState(() {

UserHomePage.available--;

print("check parking count occupied: ${UserHomePage.available}");

print(slot.document.documentID);

});

} else {

setState(() {

UserHomePage.available++;

print(

"check parking count not occupied: ${UserHomePage.available}");

// print(slot.document.documentID);

});

}

}

//print('Available ${UserHomePage.available}');

}

if (UserHomePage.Lot\_name.contains("PA1")) {

print("In PA1 ${UserHomePage.available}");

UserHomePage.parkingLot1 = UserHomePage.available;

} else if (UserHomePage.Lot\_name.contains("PA2")) {

print("In PA2 ${UserHomePage.available}");

UserHomePage.parkingLot2 = UserHomePage.available;

} else if (UserHomePage.Lot\_name.contains("PA3")) {

print("In PA3 ${UserHomePage.available}");

UserHomePage.parkingLot3 = UserHomePage.available;

} else if (UserHomePage.Lot\_name.contains("PA4")) {

print("In PA4 ${UserHomePage.available}");

UserHomePage.parkingLot4 = UserHomePage.available;

}

}

}

**6.3.16 Live Chat Page**

This page helps users to interact in live with application administrators.This page helps user as a discussion forum to view previous conversation which helps user for easy understanding.

import 'package:campusparking/ui/widgets/app\_bar.dart';

import 'package:campusparking/ui/widgets/constants.dart';

import 'package:flutter/material.dart';

import 'package:firebase\_auth/firebase\_auth.dart';

import 'package:cloud\_firestore/cloud\_firestore.dart';

final \_firestore = Firestore.instance;

FirebaseUser loggedInUser;

class ChatScreen extends StatefulWidget {

static const String route = 'chat\_screen';

@override

\_ChatScreenState createState() => \_ChatScreenState();

}

class \_ChatScreenState extends State<ChatScreen> {

final messageTextController = TextEditingController();

final \_auth = FirebaseAuth.instance;

String messageText;

@override

void initState() {

super.initState();

getCurrentUser();

}

void getCurrentUser() async {

try {

final user = await \_auth.currentUser();

if (user != null) {

loggedInUser = user;

print(loggedInUser.email);

}

} catch (e) {

print(e);

}

}

// void getMessages()async {

// final messages = await \_firestore.collection('messages').getDocuments();

// for(var message in messages.documents)

// {

// print(message.data);

// }

// }

void messagesStream() async {

await for (var snapshot in \_firestore.collection('messages').snapshots()) {

for (var message in snapshot.documents) {

print(message.data);

}

}

}

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: ApplicationBar(),

body: SafeArea(

child: Column(

// mainAxisAlignment: MainAxisAlignment.spaceBetween,

// crossAxisAlignment: CrossAxisAlignment.stretch,

children: <Widget>[

Container(

alignment: Alignment.center,

padding: EdgeInsets.all(10),

child: Text(

'Live Chat',

style: TextStyle(fontWeight: FontWeight.w600, fontSize: 30),

),

),

Container(

alignment: Alignment.center,

padding: EdgeInsets.all(10),

child: Text(

'Get immidiate assistance by asking your query in below chat window',

// style: TextStyle(fontSize: 20),

),

),

MessageStream(),

Container(

decoration: kMessageContainerDecoration,

child: Row(

crossAxisAlignment: CrossAxisAlignment.center,

children: <Widget>[

Expanded(

child: TextField(

controller: messageTextController,

onChanged: (value) {

messageText = value;

},

decoration: kMessageTextFieldDecoration,

),

),

FlatButton(

onPressed: () {

messageTextController.clear();

\_firestore.collection('messages').add({

'text': messageText,

'sender': loggedInUser.email,

"time": DateTime.now()

});

},

child: Text(

'Send',

style: kSendButtonTextStyle,

),

),

],

),

),

],

),

),

);

}

}

class MessageStream extends StatelessWidget {

@override

Widget build(BuildContext context) {

return StreamBuilder<QuerySnapshot>(

stream: \_firestore

.collection('messages')

.orderBy('time', descending: false)

.snapshots(),

// ignore: missing\_return

builder: (context, snapshot) {

if (!snapshot.hasData) {

return Center(

child: CircularProgressIndicator(

backgroundColor: Colors.lightBlueAccent,

),

);

}

final messages = snapshot.data.documents.reversed;

List<MessageBubble> messageBubbles = [];

for (var message in messages) {

final messageText = message.data['text'];

final messageSender = message.data['sender'];

final messageTime = message.data["time"] as Timestamp;

final currentUser = loggedInUser.email;

// if (currentUser == messageSender) {

// //msg from logged in user

// }

final messageWidget = MessageBubble(

sender: messageSender,

text: messageText,

time: messageTime,

isMe: currentUser == messageSender,

);

messageBubbles.add(messageWidget);

messageBubbles.sort((a, b) => b.time.compareTo(a.time));

}

return Expanded(

child: ListView(

reverse: true,

padding: EdgeInsets.symmetric(horizontal: 10.0, vertical: 20.0),

children: messageBubbles,

),

);

});

}

}

class MessageBubble extends StatelessWidget {

final String sender, text;

final Timestamp time;

final bool isMe;

MessageBubble({this.sender, this.text, this.isMe, this.time});

@override

Widget build(BuildContext context) {

return Padding(

padding: EdgeInsets.all(10.0),

child: Column(

crossAxisAlignment:

isMe ? CrossAxisAlignment.end : CrossAxisAlignment.start,

children: <Widget>[

Text(' $sender ${time.toDate()}',

style: TextStyle(

fontSize: 15.0,

color: Colors.grey[100],

)),

Material(

borderRadius: isMe

? BorderRadius.only(

topLeft: Radius.circular(30.0),

bottomLeft: Radius.circular(30.0),

bottomRight: Radius.circular(30.0))

: BorderRadius.only(

topRight: Radius.circular(30.0),

bottomLeft: Radius.circular(30.0),

bottomRight: Radius.circular(30.0)),

elevation: 5.0,

color: isMe ? Colors.grey[500] : Colors.grey[300],

child: Padding(

padding: EdgeInsets.symmetric(vertical: 10.0, horizontal: 20.0),

child: Text(text,

style: TextStyle(

color: isMe ? Colors.white : Colors.black,

fontSize: 20.0,

)),

),

),

],

));

}

}

**6.4. Back End development**

**6.4.1 configurations**

6.4.1.1 Application configuration

This code contains comments which clearly explain the functionality, where it contains several variables that are used by the application and it was named as application configuration.

*/\*\**

*\* Application level common thread pool for all @{@link org.springframework.scheduling.annotation.Async} task*

*\* executions.*

*\**

*\* @param poolSize Pool size*

*\* @param maxPoolSize Max pool size*

*\* @param queueCapacity Max queue capacity*

*\* @param threadNamePrefix Prefix for thread names associated with this pool*

*\* @return Executor for executing tasks in allocated thread pool*

*\*/*

@Bean(name = "asyncExecutionThreadPool")

public Executor asyncExecutionThreadPool(

@Value("${cp.thread-pool.pool-size:3}") int poolSize,

@Value("${cp.thread-pool.max-pool-size:5}") int maxPoolSize,

@Value("${cp.thread-pool.queue-capacity:100}") int queueCapacity,

@Value("${cp.thread-pool.thread-name-prefix}") String threadNamePrefix) {

ThreadPoolTaskExecutor threadPoolTaskExecutor = new ThreadPoolTaskExecutor();

threadPoolTaskExecutor.setCorePoolSize(poolSize);

threadPoolTaskExecutor.setMaxPoolSize(maxPoolSize);

threadPoolTaskExecutor.setQueueCapacity(queueCapacity);

threadPoolTaskExecutor.setThreadNamePrefix(threadNamePrefix);

threadPoolTaskExecutor.initialize();

return threadPoolTaskExecutor;

}

}

6.4.1.2 Application security configuration

This code is about parameters for security configurations for user’s authentication and resets password configurations.

@Configuration

@EnableWebSecurity

@RequiredArgsConstructor

public class ApplicationSecurityConfigurations extends WebSecurityConfigurerAdapter {

private final IJwtTokenProvider tokenProvider;

@Override

protected void configure(HttpSecurity http) throws Exception {

http.csrf().disable();

http.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);

http.authorizeRequests()

.antMatchers("/api/users/create").permitAll()

.antMatchers("/api/users/authenticate").permitAll()

.antMatchers("/api/users/reset-password/\*\*").permitAll()

.anyRequest().authenticated();

http.apply(new JwtTokenFilterConfigurer(tokenProvider));

}

@Bean

@Override

public AuthenticationManager authenticationManagerBean() throws Exception {

return *super*.authenticationManagerBean();

}

@Bean

public PasswordEncoder passwordEncoder() {

return new BCryptPasswordEncoder();

}

}

6.4.1.3 Application table naming strategy

This is for database identification through the user's name.

public class ApplicationTableNamingStrategy extends SpringPhysicalNamingStrategy {

@Override

public Identifier toPhysicalTableName(Identifier name, JdbcEnvironment jdbcEnvironment) {

String tableName = *super*.toPhysicalTableName(name, jdbcEnvironment).getText();

tableName = tableName.toUpperCase();

return Identifier.toIdentifier(String.format("CP\_%s", tableName));

}

}

6.4.1.4 Application web configuration

This is a web configuration that has an MVC structure.

@Configuration

public class ApplicationWebConfigurations implements WebMvcConfigurer {

@Override

public void addResourceHandlers(ResourceHandlerRegistry registry) {

if (!registry.hasMappingForPattern("/ui/\*\*")) {

registry.addResourceHandler("/ui/\*\*")

.addResourceLocations("classpath:static/ui/");

}

}

@Override

public void addCorsMappings(CorsRegistry corsRegistry) {

corsRegistry.addMapping("/\*\*")

.allowedOrigins("\*");

}

}

**6.4.2 Users**

6.4.2.1 profile

This snippet is about user profile which has email id and registration page

@Getter

@Setter

@Builder

@ToString

@NoArgsConstructor

@AllArgsConstructor

@Entity

public class Profile {

@Id

private String emailId;

private String registrationPlate;

@Column(name="fullname")

private String name;

}

6.4.2.2 Role

public enum Role {

USER

}

6.4.2.3 User

This code snippet gives information about the user, whether his account is active or not.

public class User extends FunctionEntity<User> {

@Id

private String emailId;

@Column(nullable = false)

private String password;

private boolean isAccountActivated;

@OneToOne(

fetch = FetchType.LAZY,

cascade = CascadeType.ALL,

orphanRemoval = true

)

@JoinColumn(name = "password\_reset\_token\_id")

private Token passwordResetToken;

@ElementCollection(targetClass = Role.class)

@Enumerated(EnumType.STRING)

@CollectionTable(name = "user\_role")

@Column(name = "role", nullable = false)

private Set<Role> roles;

@MapsId

@OneToOne(

fetch = FetchType.LAZY,

cascade = CascadeType.ALL,

orphanRemoval = true

)

@JoinColumn(name = "emailId")

private Profile profile;

}

**6.4.3 Requests**

6.4.3.1 Authentication request

This code is used to request authentication from users.

@Getter

@Setter

@Builder

@NoArgsConstructor

@AllArgsConstructor

public class AuthenticationRequest implements IVerifiableRequestBody {

private String emailId;

private String password;

@Override

public Verified verify() {

return null;

}

}

6.4.3.2 create user request

This request for a sign-up page with constraints for each text box.

public class CreateUserRequest implements IVerifiableRequestBody {

private String emailId;

private String password;

private String verifiedPassword;

private String registrationPlate;

private String name;

@Override

public Verified verify() {

if (emailId == null || emailId.trim().isEmpty()) {

return Verified.failedValidation("Email id cannot be empty");

}

if (password == null || password.trim().isEmpty()) {

return Verified.failedValidation("Password cannot be empty");

}

if (verifiedPassword == null || verifiedPassword.trim().isEmpty()) {

return Verified.failedValidation("Verification password cannot be empty");

}

if (password.length() <= 5) {

return Verified.failedValidation("Password must have 6 or more characters");

}

if (!password.equals(verifiedPassword)) {

return Verified.failedValidation("Password and verification passwords do not match");

}

if (registrationPlate == null || registrationPlate.trim().isEmpty()) {

return Verified.failedValidation("Registration plate cannot be empty");

}

if (name == null || name.trim().isEmpty()) {

return Verified.failedValidation("Name cannot be empty");

}

return Verified.passedValidation();

}

}

6.4.3.3 Reset password otp generation request

@Getter

@Setter

@Builder

@NoArgsConstructor

@AllArgsConstructor

public class ResetPasswordOtpGenerationRequest {

private String emailId;

}

6.4.3.3 Update password request

This request is when the user is trying to update his password and his constraints.

public class UpdatePasswordRequest implements IVerifiableRequestBody {

private String emailId;

private String newPassword;

private String verifyNewPassword;

@Override

public Verified verify() {

if (newPassword == null || newPassword.trim().isEmpty()) {

return Verified.failedValidation("Password cannot be empty");

}

if (verifyNewPassword == null || verifyNewPassword.trim().isEmpty()) {

return Verified.failedValidation("Verification password cannot be empty");

}

if (newPassword.length() <= 5) {

return Verified.failedValidation("Password must have 6 or more characters");

}

if (!newPassword.equals(verifyNewPassword)) {

return Verified.failedValidation("Password and verification passwords do not match");

}

return Verified.passedValidation();

}

}

6.4.3.3 verify password reset otp request

public class VerifyPasswordResetOtp {

private String emailId;

private String otp;

}

**6.4.4 Responses**

This is a code snippet for responses for every request on every page.

@RestController

@RequestMapping("/api/users")

@RequiredArgsConstructor

public class UserEndpoints {

public static final String PASSWORD\_RESET = "Password Reset";

public static final String PASSWORD\_RESET\_MESSAGE = "Use following code: %s to reset your password";

private final IUserService userService;

private final PasswordEncoder passwordEncoder;

private final AuthenticationManager authenticationManager;

private final IJwtTokenProvider tokenProvider;

private final ITokenService tokenService;

private final IEmailService emailService;

@VerifyRequestBody

@PostMapping("/create")

public CreateUserResponse createUser(@RequestBody final CreateUserRequest userCreationRequest) {

return User.builder()

.emailId(userCreationRequest.getEmailId())

.password(passwordEncoder.encode(userCreationRequest.getPassword()))

.emailId(userCreationRequest.getEmailId())

.roles(List.of(Role.USER).toJavaSet())

.isAccountActivated(Boolean.TRUE)

.profile(

Profile.builder()

.emailId(userCreationRequest.getEmailId())

.name(userCreationRequest.getName())

.registrationPlate(userCreationRequest.getRegistrationPlate())

.build()

)

.build()

.andThenWith(userService::createNewUser)

.andThenTransform(

user -> CreateUserResponse.builder()

.emailId(user.getEmailId())

.name(user.getProfile().getName())

.registrationPlate(user.getProfile().getRegistrationPlate())

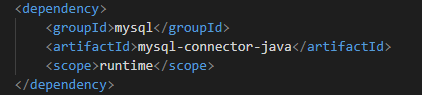
.build()

);

}

**6.4.5 pom.xml**

Add MySQL dependency to <dependencies>

****

**6.4.6 application.yml**

Here .yml is the same as .properties. We use jpa mapping to the database table and set up necessary parameters.

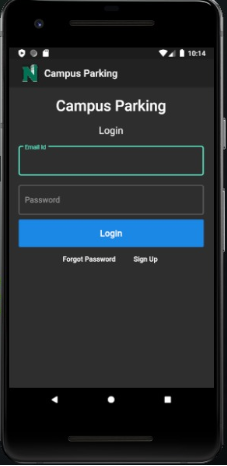


**Chapter 7**

**End-user Manual**

**7.1 Login**

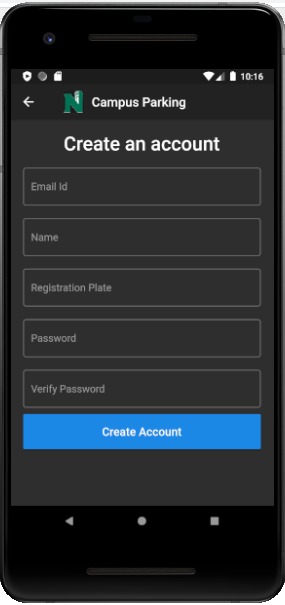
Below is the landing screen once the user opens our application.



* If you're a new user please click on the signup button to stay signed in into the application. Actions to complete after clicking on the signup button are provided in 7.2 please refer to it.
* Once you're finished with the signup portion in the application you're qualified to go and sign in to the application.
* In the login page, you are prompted to enter the email address and password. Kindly use the email address and password that were provided on the sign-up page for the successful login.
* In any case, if the password doesn't match with the current and the earlier password given during signup please click on the forgot password button. It gets redirected to the reset password page. Please refer to 7.3 for detailed steps.

**7.2 Sign up**

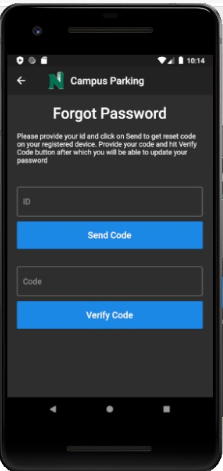
Here is the sign-up screen of our application.



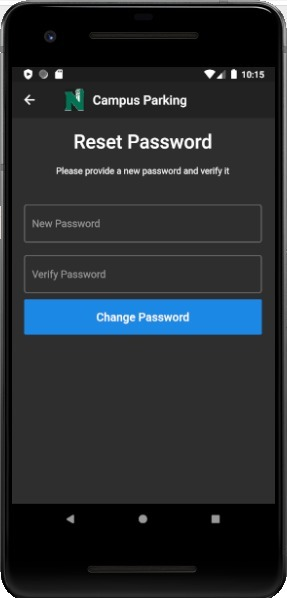
* In the signup page of the application, a new user must enter the email address which must be unique to the application. Please note if you have already signed up into the application then you're in the wrong window. Kindly refer to 7.1 and login to the application.
* If you forget your password, Kindly refer to 7.3 for more details.
* Signup page prompts for details like email address, name, registration plate, password, and password text fields which are mandatory to complete signup form in the application.

**7.3 Forgot password**

Here is the forgot password screen in the application.

****

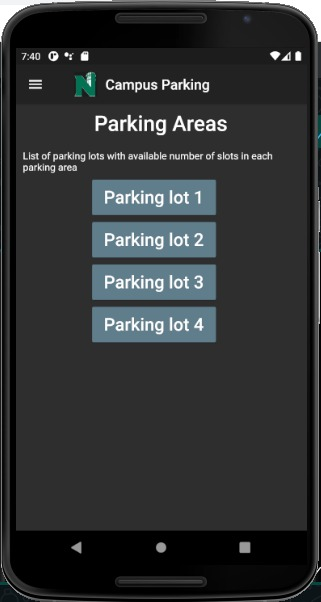
* User should type in one’s ID then tap button Send Code, an email with verification code will be sent.
* Type the code to Code text field, then tap the Verify Code button. If the code verification passed, the page will be navigated to the following page to reset the password.

****

* In this page, the user is required to input the new password in each text field. then tap the Change Password button. If the string in each text field is the same, the new password will be applied.

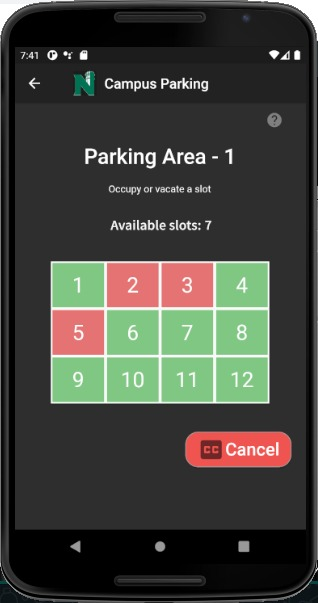
**7.4 Home Screen**

In this page user can tab on the hamburger menu to see the different functionalities available in our application. In the home screen user can tap on the desired parking lot to view the parking lot page and the number of available slots in the respective parking lot.



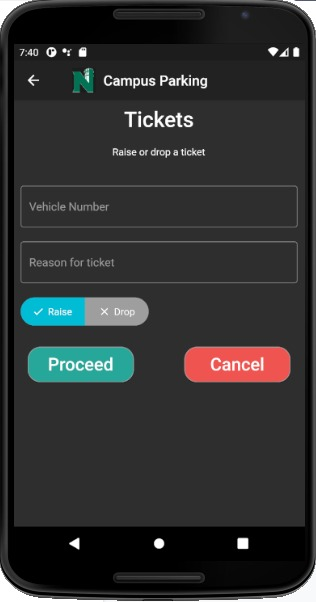
**7.5 Parking screen**

* Users can choose a parking slot which is available or vacate a slot which was already occupied by them. User have to click on the available slot( green color) to occupy that slot and have to tap on the occupied( red color) to vacate it.User an click on the cancel button to redirect them to the home page if he or she doesn't want to perform any action in this page.

****

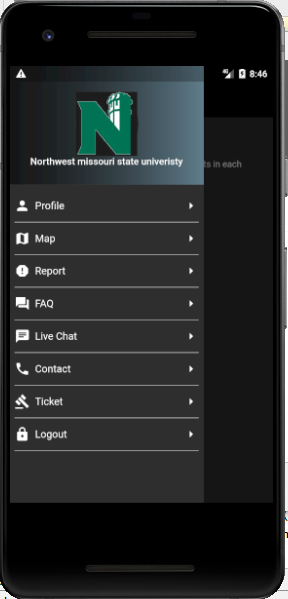
**7.6 Ticket Screen**

User cant enter this page. This is only for admins. In this page admin can raise or drop a ticket by filling the details asked for and selecting raise or drop option and then click on proceed to perform the desired action.



**7.7 Hamburger menu**

* This is the Hamburger Menu Bar which has all the list of UI pages, whenever user clicks on particular page, user gets directed to that page through the internal routings.

****

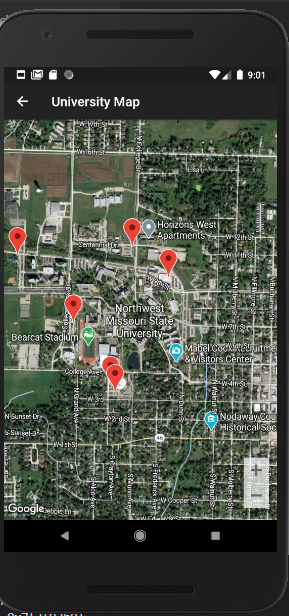
**7.8 Profile screen**

* Users can check the profile of oneself, including username, icon account balance, etc.



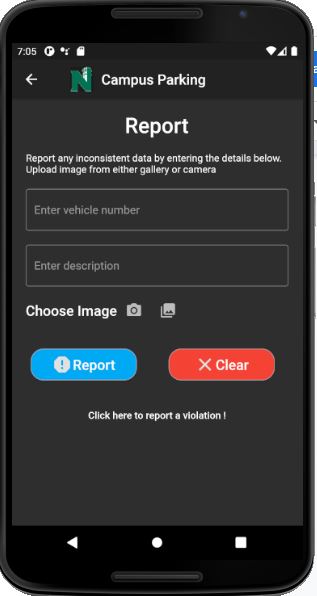
**7.9 Map screen**

* This page makes easy for the user to have a look at the campus map and to check the parking locations.
* Helps users to view the location of parking lots which are available in the campus.
* This screen helps users to get to know the number of slots available in the parking lots.
* As soon as user open the map screen from the hamburger menu geolocation of the campus open on the screen.
* This page makes the user job easy.User can check for the parking lots in home page and reserve one and reach the parking location by checking in the map screen.



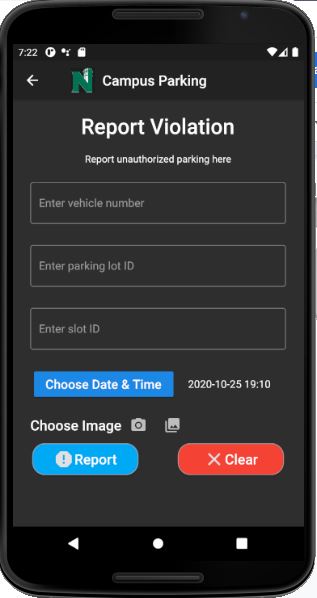
**7.10 Report screen**

* Users can report inconsistent data by submitting necessary information to the administrator.
* This page takes in the vehicle number and description.
* Users can choose to upload images either through gallery or camera. They can also crop and rotate the image as necessary and upload it.
* It also has a link to redirect to the “Report Violation” page.
* Once, the inconsistent data is reported a toast message is displayed notifying the users that they have reported successfully.



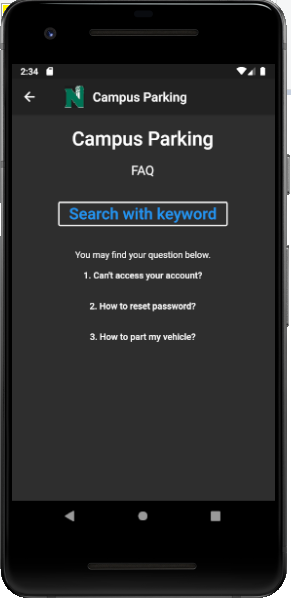
**7.10.1 Report Violation Screen**

* Users can report any violations (vehicles parked without parking permit) to the administrator on this page.
* This page takes in the vehicle number, parking lot ID, slot ID as inputs from the user.
* Users can choose to upload images either through gallery or camera. They can also crop and rotate the image as necessary and upload it.
* Users can also select the date and time on clicking the “Choose Date & Time” button which pops up a dialog box to choose the date from a calendar and time from a clock
* Once, the violations are reported a toast message is displayed notifying the users that they have reported successfully.

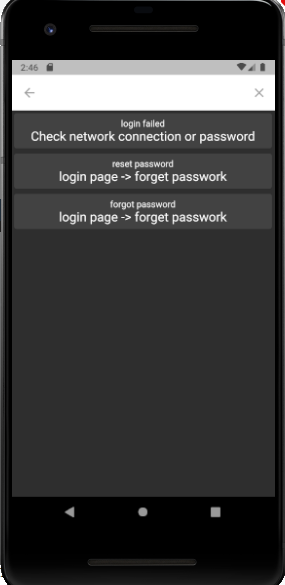


**7.11 FAQ screen**

* Users can search one’s question by keywords or check the frequently asked.
* In this page users can tap the questions listed in the page to get answers.

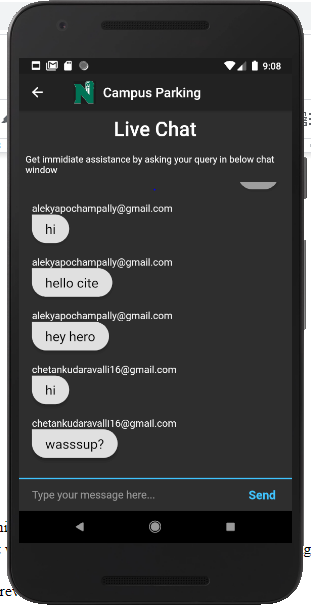
****

* Users can tap the search bar to search, by clicking the suggested keywords answers will be displayed.

****

**7.12 Live chat screen**

* Users can send messages to the administrator on this page.
* Live chat screen helps users to interact with the administrator for any queries regarding campus car parking applications.
* This screen also helps to review the previous queries asked by other users.
* Message sent form the user in case of any enquiry or for any discussion live chat screen makes the users job easy,
* All the messages are time stamped and user can directly check them and keep a track of the latest once.



**7.13Contact-us screen**

* This page presents available information to contact administrators.

