Dharmsinh Desai University, Nadiad

Faculty of Technology, Department of Computer Engineering

B.Tech. CE Semester – VI

Subject: (CE-621) System Design Practice

Project Title:

Next Generation Toll System

By:

Jaydeep Mahajan (Roll No:CE066   ID:17CEUOG098)

Raj Panchal (Roll No:CE074 ID:17CEUBG104)

Guided By:

Prof. Jigar M. Pandya

Dharmsinh Desai University, Nadiad

Faculty of Technology, Department of Computer Engineering

CERTIFICATE

This is to certify that System Design Practice project entitled “Next Generation Toll System” is the bonafied report of work carried out by

1) **Raj Panchal (17CEUBG104)**
2) **Jaydeep Mahajan (17CEUOG098)**

Of Department of Computer Engineering, Semester VI, academic year 2019-20, under our supervision and guidance.

Guide	Head of the Department
Prof. Jigar M. Pandya	**Dr. C.K. Bhensdadia**
Assistant Professor	Prof. & Head
Department of Computer Engineering	Department of Computer Engineering
Dharmsinh Desai University, Nadiad.	Dharmsinh Desai University, Nadiad.

Table of Contents

1.ABSTRACT

Today we all aware about FASTag is an electronic toll collection system in India, operated by the National Highway Authority of India (NHAI). It employs Radio Frequency Identification (RFID) technology for making toll payments directly from the prepaid or savings account linked to it or directly toll owner. It is affixed on the windshield of the vehicle and enables to drive through toll plazas without stopping for transactions. The tag can be purchased from official Tag issuers or participating Banks and if it is linked to a prepaid account, then recharging or top-up can be as per requirement. As per NHAI, FASTag has unlimited validity. 7.5% cashback offers were also provided to promote the use of FASTag. Dedicated Lanes at some Toll plazas have been built for FASTag.

We just tried to next generation version of this technology. It assumes that in user's car having one device installed. Before journey he could set the source and destination as his/her requirement then device shows the all the tollbooth which comes in journey and total amount that he has to pay for this trip. So, user can pay in advance via login into the system using credit card, debit card or takes another way to pay like whenever he crosses the toll booth automatically amount deducted from his account. Importantly toll booths are virtual now.

2. INTRODUCTION

2.1 Brief Introduction:

This project is going to build on subject "Next Generation Toll System". In which we emphases on consuming less amount of time. An individual can get rid from long queue of tollbooth. Here we are going to develop virtual tollbooth between source & destination. We will try to build system that compatible with cars. Now a days, though FASTAG comes in picture, but some of us are wait for paying toll by standing in long queue and lose precious time. Because of this many disadvantages like wasting fuel, time for journey get exceed. A system using android auto android automotive OS technologies that helps to pay toll-tax automatically without any kind of problem based on car location.

Currently we have developed app for this kind of system using flutter framework, which is able to display source and destination and virtual tollbooth between them. It can help you to pay the tax before journey started or during the journey whenever toll comes.

1) Advantages of System:
   - ➢ Convenience to Drivers/Consumers.
   - ➢ Saves Manpower cost.
   - ➢ Safer and more secure payment.
   - ➢ Time saver.

2.2 Tools/Technologies Used:

Technologies:

- ✓ Flutter – Framework
- ✓ Dart – Single codebase
- ✓ Firebase
- ✓ Google Map API for android SDK
- ✓ Google direction API (if possible)
  Google direction API is not free of charge available on google cloud. Though we use free credits on cloud , it didn't work.
- ✓ Payment Gateway API (Razorpay)

Tools:

- ✓ Microsoft Visual Code
- ✓ Android Studio
- ✓ Android Emulator
- ✓ Android Device & USB cable

3. SOFTWARE REQUIREMENTS SPECIFICATION

Types of Users :

- User/Traveller
- Admin

System function requirements:

R.1 User / Traveller

R.1.1 Register / Sign Up :

Input : Email and Password.

Output : Confirmation message.

R.1.2 Login / Sign In :

Input : Email and Password.

Output : Home Screen.

R.1.3 Select source and destination for trip :

Input : Source and destination.

Output : Put marker on source and destination.

R.1.4 View toll booths:

Input : User Selection .

Output : Display virtual tolbooths.

R.1.5 Edit trip details :

Input : Car type, Car no., Journey Date etc.

Output : Payment Page.

R.1.6 Calculate toll amount:

Input : toll between source and destination.

Output : Total Amount

R.1.7 Make Payment :

Input : Payment details.

Output : Payment success or Payment error message.

R.1.8 Display history :

Description : One can show all previous transaction.

Input : User selection.

Output : List of all previous transaction.

R.2 Admin

R.2.1 Add virtual toll booth:

Input : Geo location point of tollbooth, price details according to vehicle type.

Output : Added to the database and reflect on map.

R.2.2 Show user activity:

Input : User selection

Output : Display all user activity like log in, log out, sign up, payment details, transaction, history of user etc.

R.2.3 Edit virtual toll booth details:

Input : New details.

Output: Confirmation message.

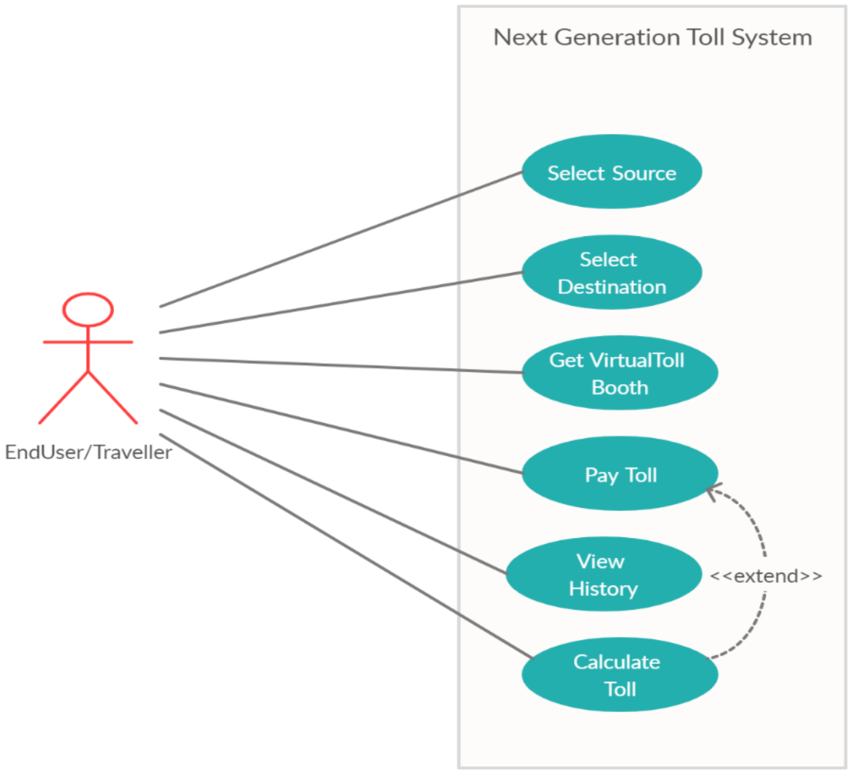
R.2.4 Manage user activity :

Input : Action for unusual activity like block user, unblock user.

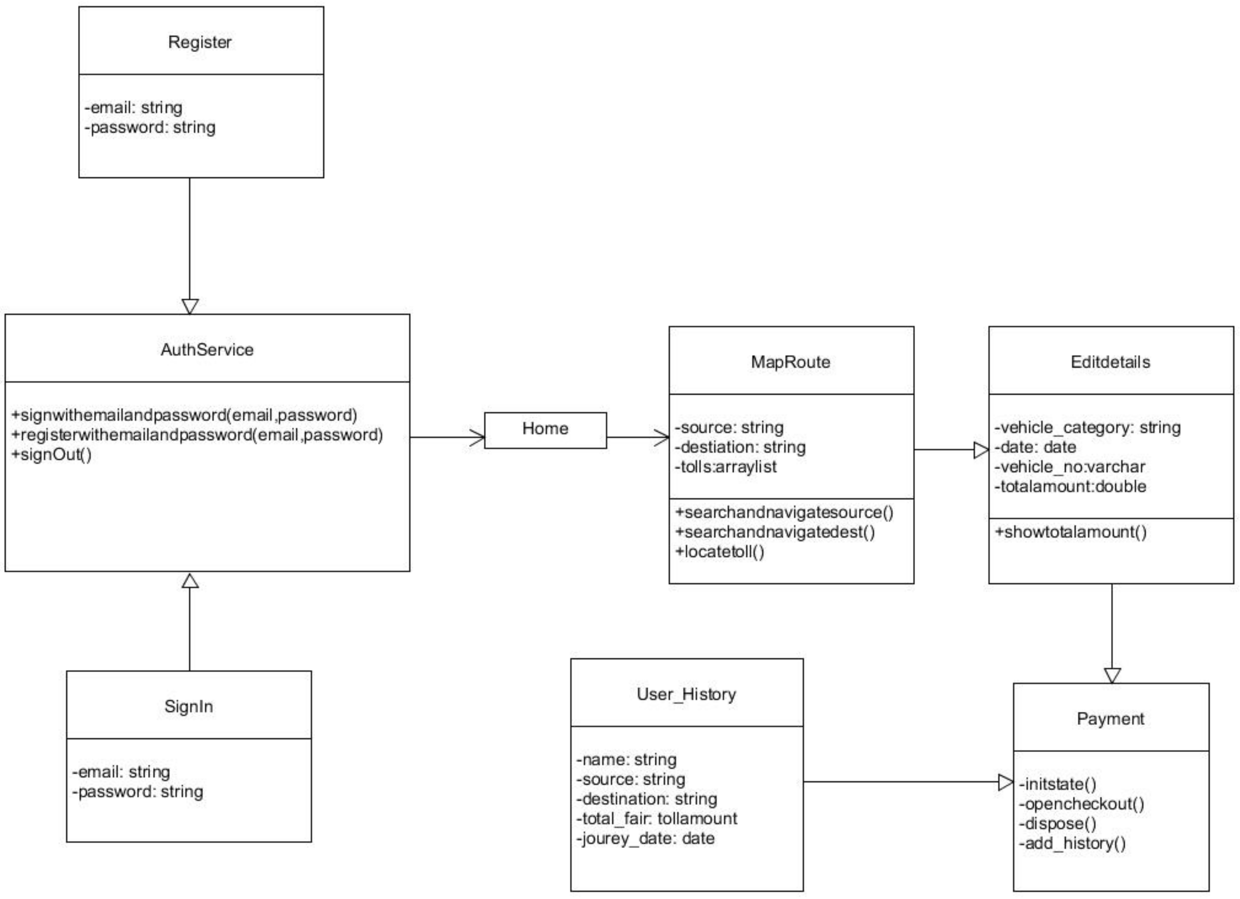
Output : Confirmation message.

4. DESIGN

Use case Diagram :

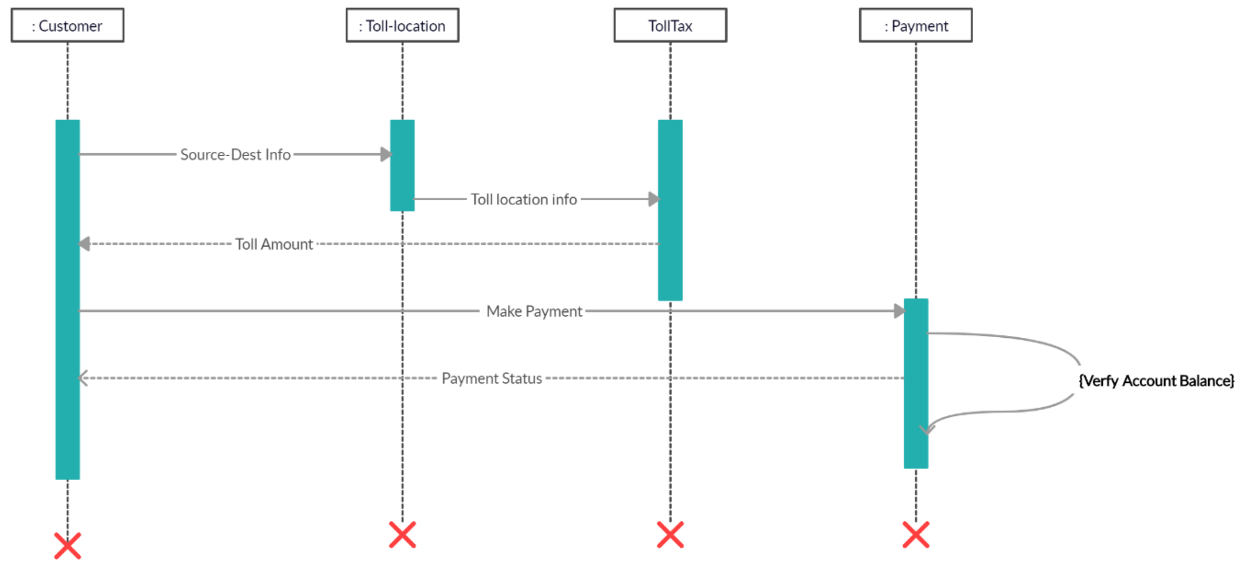


Class diagram :



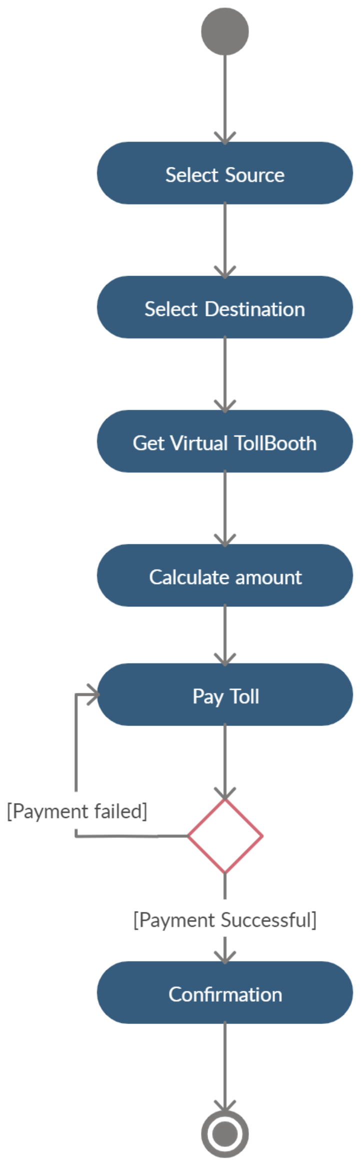
Sequence diagrams:

For Make-Payment :

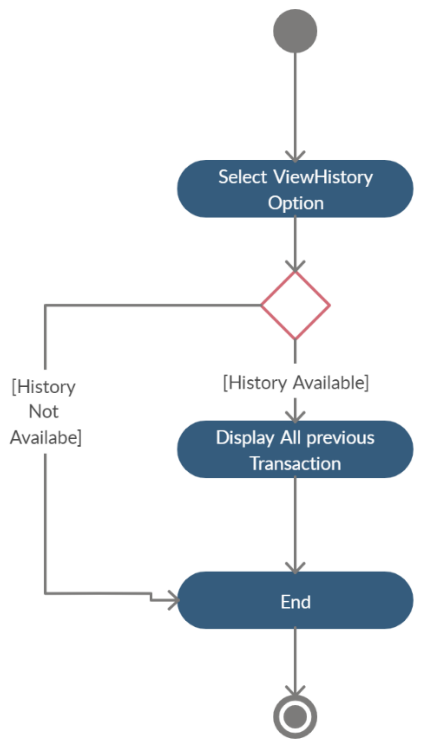


Activity diagrams:

For Tollpay:



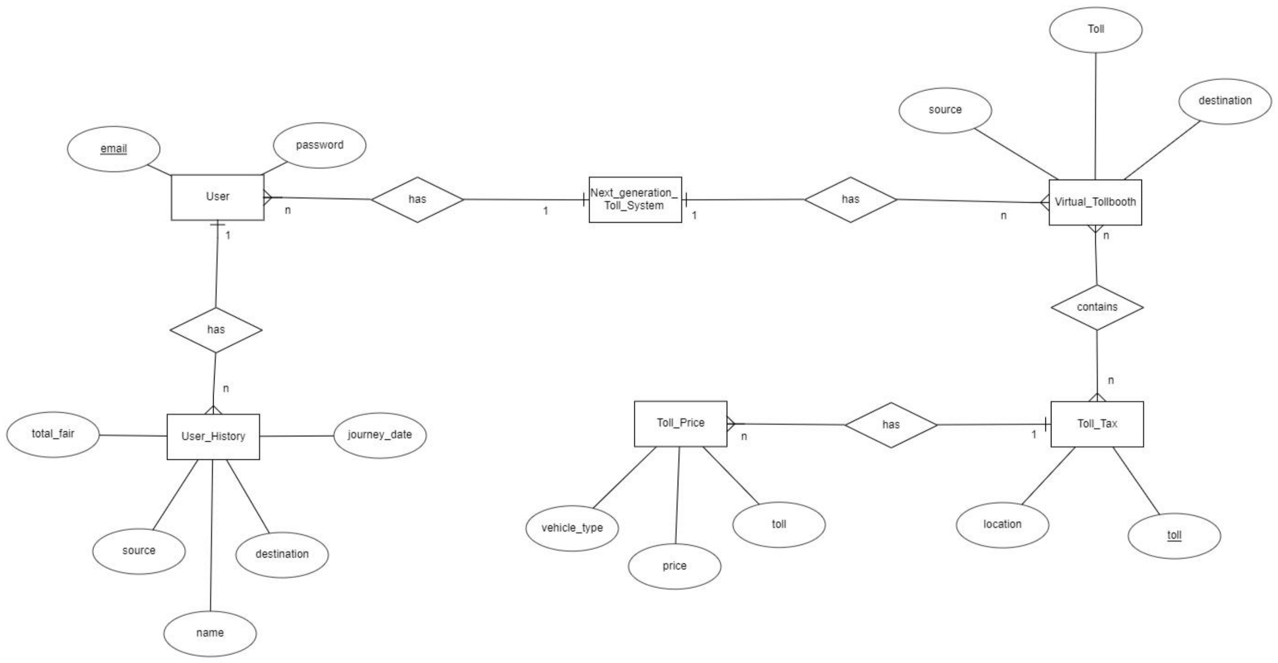
For view history:



State diagrams:

*State diagram is not applicable for this system because system doesn’t contain any kind of state.

E-R Diagram:



Relationships :

User to User_History : one to many

Virtual_Tollbooth to Toll_Tax : many to many

Toll_Tax to Toll_Price : one to many

Data dictionary :

Data Dictionary for Restaurant Billing System

User

Field	Type	Required	Unique	PK/FK	Reference Table
email	varchar (30)	Yes	Yes	PK	-
password	varchar (50)	Yes	No	-	-

Virtual_Tollbooth

Field	Type	Required	Unique	PK/FK	Reference Table
source	varchar (20)	Yes	No	-	-
destination	varchar (20)	Yes	No	-	-
toll	array	Yes	Yes	PK	-

Toll_Tax

Field	Type	Required	Unique	PK/FK	Reference Table
toll	varchar (10)	Yes	Yes	PK	-
location	geopoint	Yes	No	-	-

Toll_Price

Field	Type	Required	Unique	PK/FK	Reference Table
toll	varchar (10)	Yes	Yes	FK	Toll_Tax
price	float	Yes	No	-	-
vehicle_type	varchar (10)	Yes	No	-	-

User_History					
Field	Type	Required	Unique	PK/FK	Reference Table
name	varchar (30)	Yes	No	-	-
source	varchar (50)	Yes	No	-	-
destination	varchar (50)	Yes	No	-	-
journey_date	date	Yes	No	-	-
total_fair	float	Yes	No	-	-

5. IMPLEMENTATION DETAIL

I). Modules

- Main User Module :

  This module contains the main homepage of the system which provides basic information about the website. It contains different functionalities like view history, pay toll etc. This module does require authentication.

- Login Module :

  This module takes user credentials and then verifies it with registered users, if user have entered incorrect credentials then alert with “Incorrect details” , otherwise user will be redirected to the home page.

- Registration Module :

  This module is used to create the account on system. All fields are containing some types of validations. Then user can login to our system using his/her credentials.

- Map route Module :

  This module allows driver to set source and destination , it helps to display tollbooth between two points and based on car type and finds total toll amount of journey. It has been implemented by use of Maps SDK for Android and geo-location API.

- Payment Module :

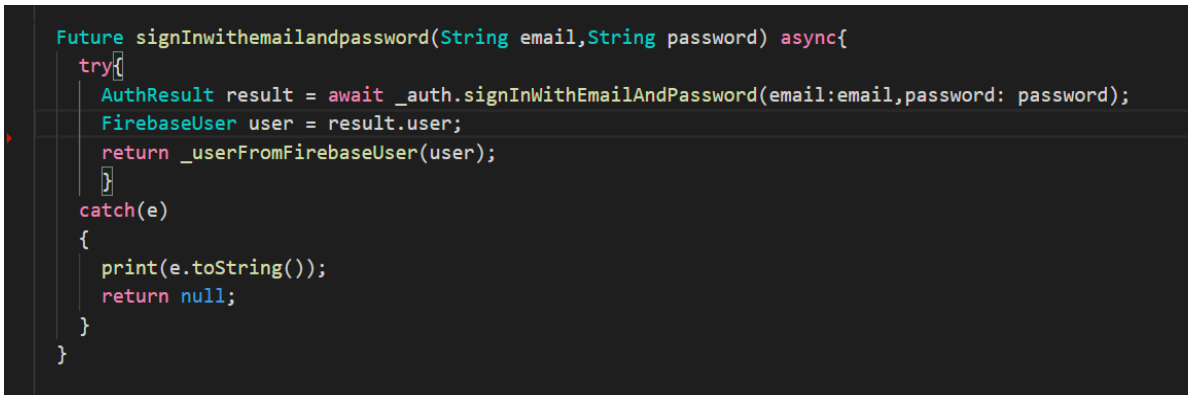
  This module is built for pay the toll by using Razorpay API. Razorpay is a payment gateway service. It accept and validate internet payments via credit card, debit card, Net banking modes, also other wallets like Paytm , PhonePe, fast charge.

- Previous transaction Module :

  This module is displaying all previous transaction for user of the system. It display all information regarding previous journey.

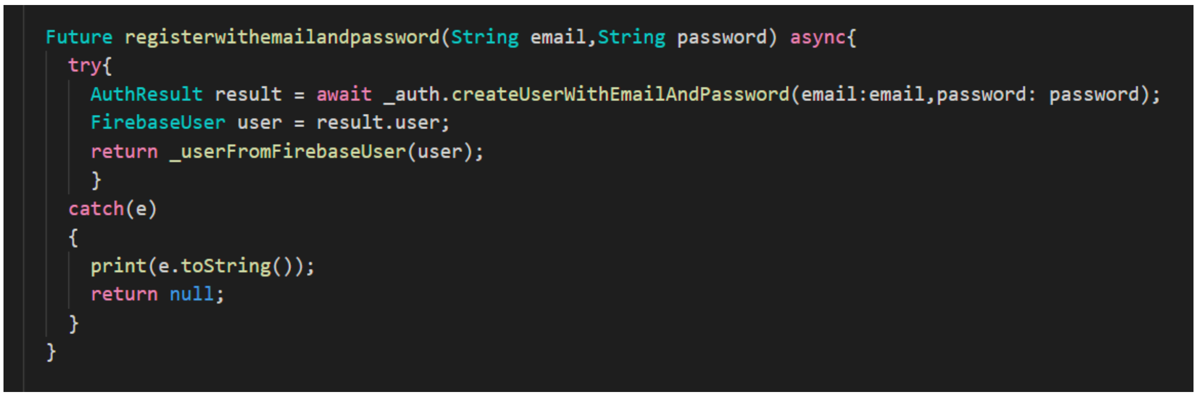
II). Function prototypes

- Login :
  Login is default functionality that provides by firebase using function which is shown below. We just need to enter email and password.



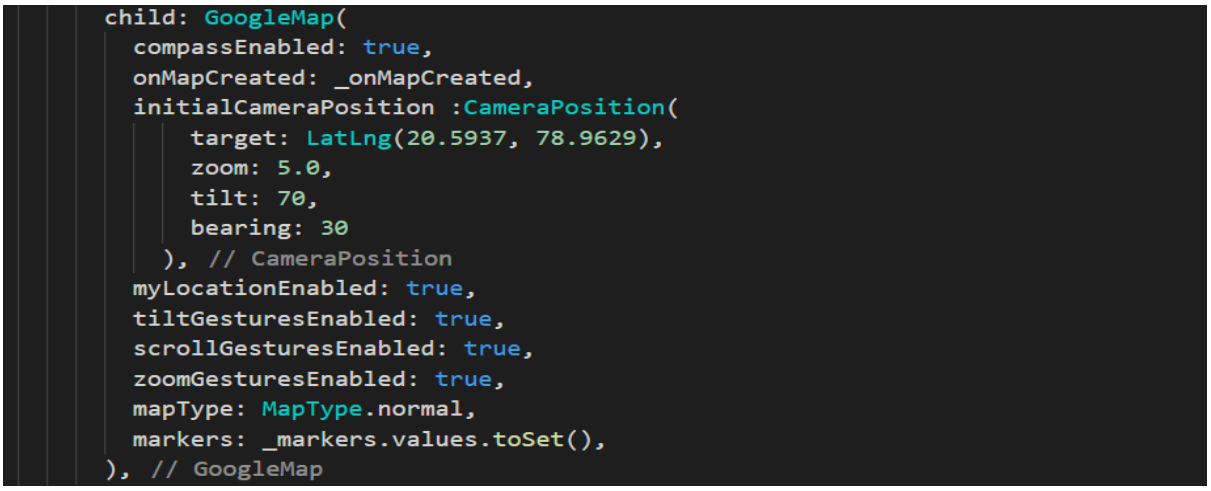
```
Future signInwithemailandpassword(String email,String password) async{
  try{
    AuthResult result = await _auth.signInWithEmailAndPassword(email:email,password: password);
    FirebaseUser user = result.user;
    return _userFromFirebaseUser(user);
    }
  catch(e)
  {
    print(e.toString());
    return null;
  }
}
```

- Register :
  Firebase also provides functionality of sign up by using following function.



```
Future registerwithemailandpassword(String email,String password) async{
  try{
    AuthResult result = await _auth.createUserWithEmailAndPassword(email:email,password: password);
    FirebaseUser user = result.user;
    return _userFromFirebaseUser(user);
    }
  catch(e)
  {
    print(e.toString());
    return null;
  }
}
```

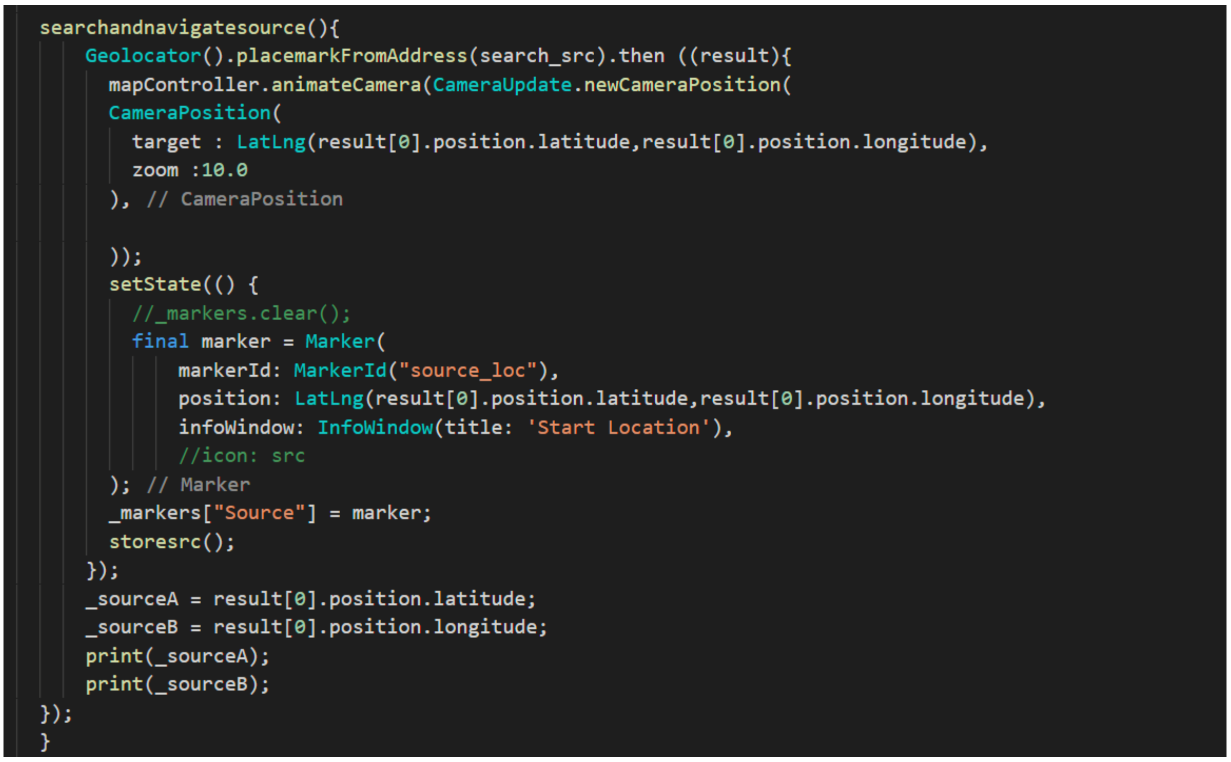
- Load google map :
  This code snippet describes the configuration about to how to load map in flutter. Also, it requires google API key which is defined in AndroidManifest.xml .



```
child: GoogleMap(
  compassEnabled: true,
  onMapCreated: _onMapCreated,
  initialCameraPosition :CameraPosition(
      target: LatLng(20.5937, 78.9629),
      zoom: 5.0,
      tilt: 70,
      bearing: 30
    ), // CameraPosition
  myLocationEnabled: true,
  tiltGesturesEnabled: true,
  scrollGesturesEnabled: true,
  zoomGesturesEnabled: true,
  mapType: MapType.normal,
  markers: _markers.values.toSet(),
), // GoogleMap
```

➤ Set source & destination :

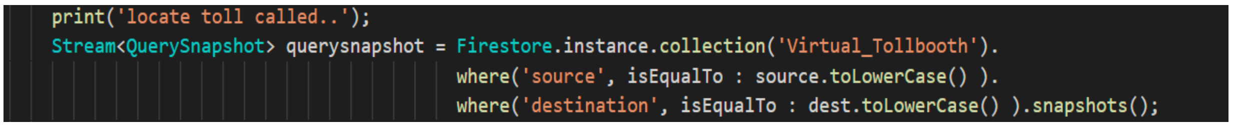
This function provides the set marker on source and destination in map. Here is the function for only source as well as similar for the destination.



```
searchandnavigatesource(){
    Geolocator().placemarkFromAddress(search_src).then ((result){
      mapController.animateCamera(CameraUpdate.newCameraPosition(
      CameraPosition(
        target : LatLng(result[0].position.latitude,result[0].position.longitude),
        zoom :10.0
      ), // CameraPosition

      ));
      setState(() {
        //_markers.clear();
        final marker = Marker(
            markerId: MarkerId("source_loc"),
            position: LatLng(result[0].position.latitude,result[0].position.longitude),
            infoWindow: InfoWindow(title: 'Start Location'),
            //icon: src
      ); // Marker
      _markers["Source"] = marker;
      storesrc();
    });
    _sourceA = result[0].position.latitude;
    _sourceB = result[0].position.longitude;
    print(_sourceA);
    print(_sourceB);
});
}
```

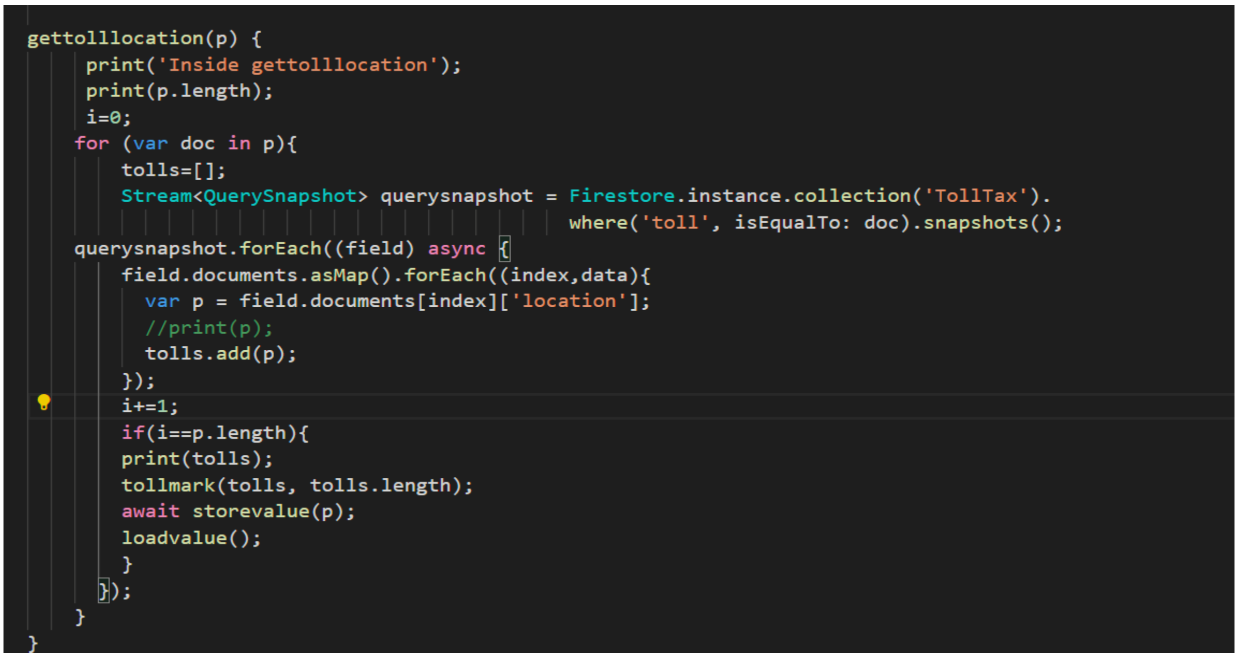
➤ Mark tollbooth :

Below query provides the list of all toll booth between source and destination.



```
print('locate toll called..');
Stream<QuerySnapshot> querysnapshot = Firestore.instance.collection('Virtual_Tollbooth').
                                      where('source', isEqualTo : source.toLowerCase() ).
                                      where('destination', isEqualTo : dest.toLowerCase() ).snapshots();
```

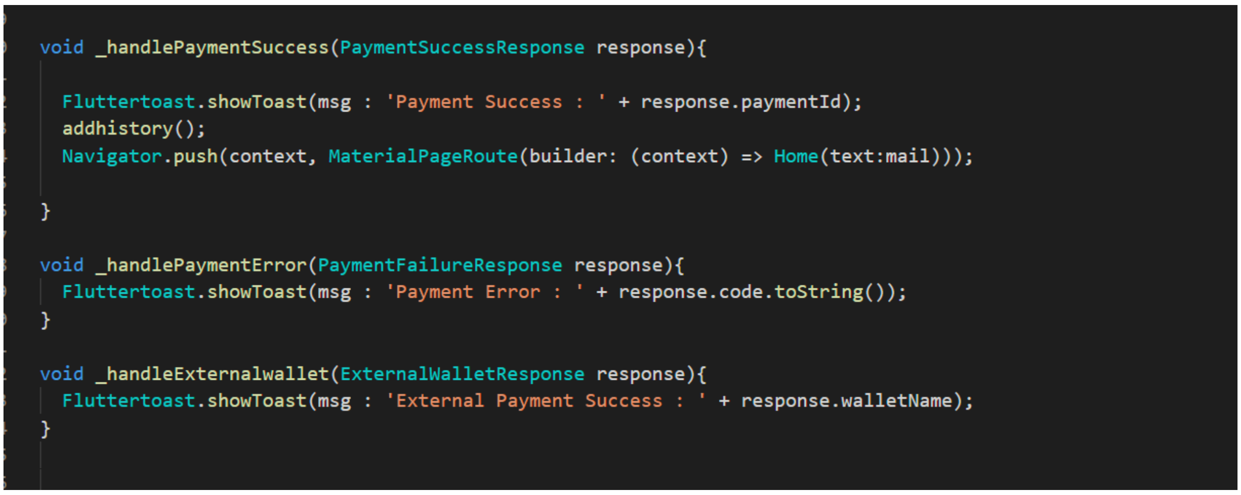
Getlocation() function helps to find the location of toll which comes between source and destination and using tollmark() function it marks the location of toll. This function takes list of toll booth as an argument and find location one by one.



```
gettolllocation(p) {
    print('Inside gettolllocation');
    print(p.length);
    i=0;
  for (var doc in p){
    tolls=[];
    Stream<QuerySnapshot> querysnapshot = Firestore.instance.collection('TollTax').
                                          where('toll', isEqualTo: doc).snapshots();
  querysnapshot.forEach((field) async {
    field.documents.asMap().forEach((index,data){
      var p = field.documents[index]['location'];
      //print(p);
      tolls.add(p);
    });
    i+=1;
    if(i==p.length){
    print(tolls);
    tollmark(tolls, tolls.length);
    await storevalue(p);
    loadvalue();
    }
  });
  }
}
```

➢ Payment handle methods :

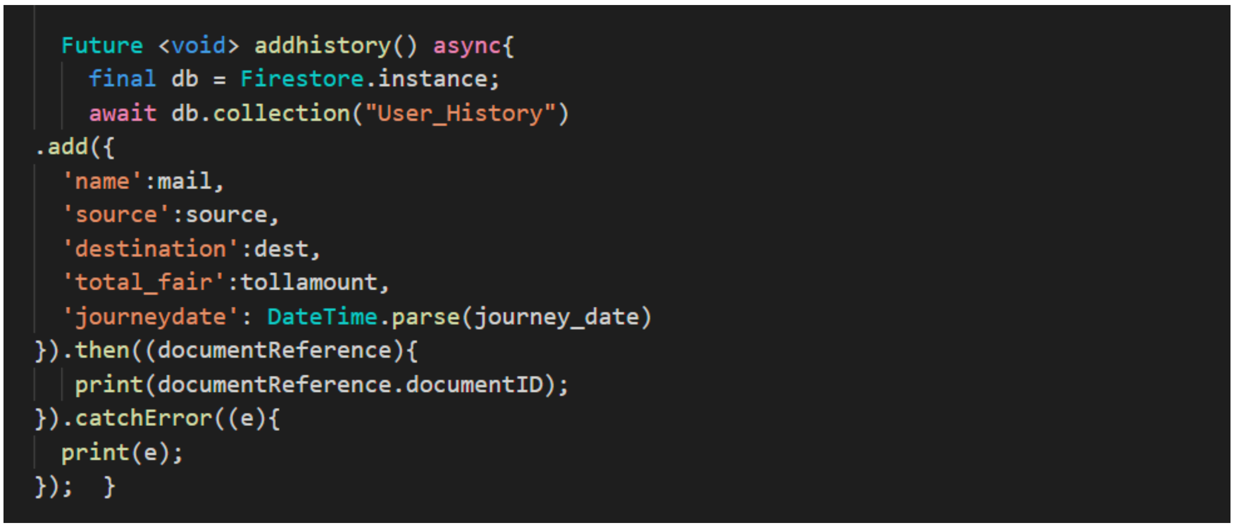
Here is the three method for handle payment.
handlePaymentSuccess() is for payment success that shows the message "Payment Success" . handlePaymentError() is for payment failure. Due to some reasons if payment fails then shows message "payment Error". handleExternalWallet() is for payment from external wallet like paytm, freecharge etc. If the external payment is succeeded then shows the message "External Payment Success".



```
void _handlePaymentSuccess(PaymentSuccessResponse response){

  Fluttertoast.showToast(msg : 'Payment Success : ' + response.paymentId);
  addhistory();
  Navigator.push(context, MaterialPageRoute(builder: (context) => Home(text:mail)));

}

void _handlePaymentError(PaymentFailureResponse response){
  Fluttertoast.showToast(msg : 'Payment Error : ' + response.code.toString());
}

void _handleExternalwallet(ExternalWalletResponse response){
  Fluttertoast.showToast(msg : 'External Payment Success : ' + response.walletName);
}
```

- Add history in database:

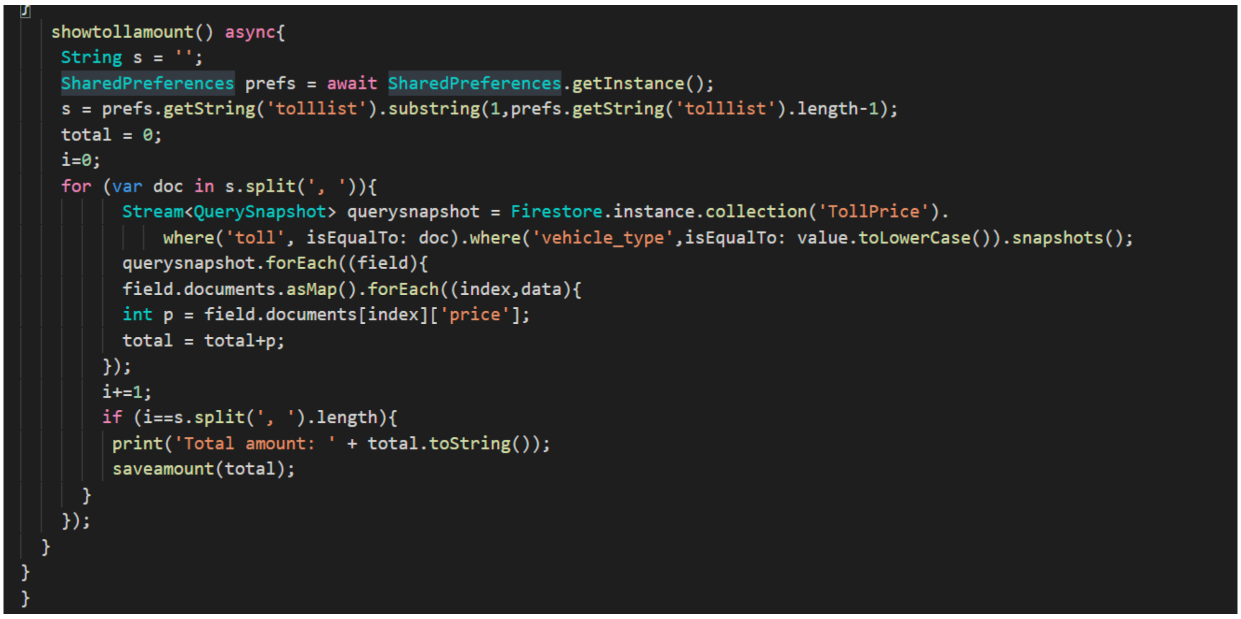
  Addhistory() function is used for add history details of user in database. It contains name, source, destination, total fair, journey date.



```
  Future <void> addhistory() async{
    final db = Firestore.instance;
    await db.collection("User_History")
.add({
  'name':mail,
  'source':source,
  'destination':dest,
  'total_fair':tollamount,
  'journeydate': DateTime.parse(journey_date)
}).then((documentReference){
   print(documentReference.documentID);
}).catchError((e){
  print(e);
});  }
```

- Show toll amount :

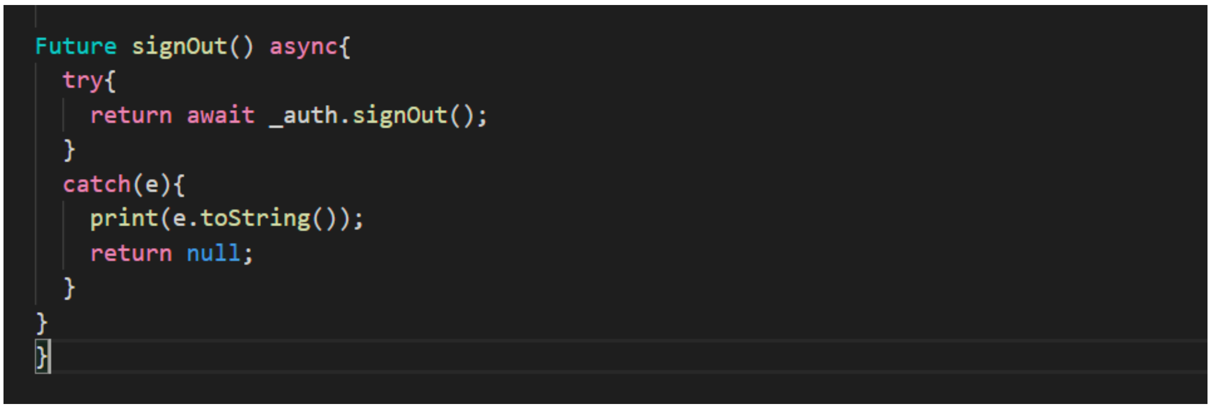
  This function calculate toll amount according to vehicle type and display. It uses the list of toll booth for particular source and destination .



```
  showtollamount() async{
   String s = '';
   SharedPreferences prefs = await SharedPreferences.getInstance();
   s = prefs.getString('tolllist').substring(1,prefs.getString('tolllist').length-1);
   total = 0;
   i=0;
   for (var doc in s.split(', ')){
         Stream<QuerySnapshot> querysnapshot = Firestore.instance.collection('TollPrice').
             where('toll', isEqualTo: doc).where('vehicle_type',isEqualTo: value.toLowerCase()).snapshots();
         querysnapshot.forEach((field){
         field.documents.asMap().forEach((index,data){
         int p = field.documents[index]['price'];
         total = total+p;
       });
       i+=1;
       if (i==s.split(', ').length){
        print('Total amount: ' + total.toString());
        saveamount(total);
     }
   });
 }
}
}
```

➢ Sign out :

This functionality provides the for logout which is also built in function provided by firebase.



```
Future signOut() async{
  try{
    return await _auth.signOut();
  }
  catch(e){
    print(e.toString());
    return null;
  }
}
}
```

6. TESTING

In this system we have used Black Box testing.

The main focus of black box testing is on the validation of your functional requirements.

Here are the generic steps followed to carry out any type of Black Box Testing.

- ✓ Initially, the requirements and specifications of the system are examined.
- ✓ Tester chooses valid inputs (positive test scenario) to check whether SUT processes them correctly. Also, some invalid inputs (negative test scenario) are chosen to verify that the SUT is able to detect them.
- ✓ Tester determines expected outputs for all those inputs.
- ✓ Software tester constructs test cases with the selected inputs.
- ✓ The test cases are executed.
- ✓ Software tester compares the actual outputs with the expected outputs.
- ✓ Defects if any are fixed and re-tested.

Different test cases :

Test Case ID	TC_01
Requirement ID	R.1.2
Test Case Objective	Test Login
Step	1.Enter Email & Password 2.Login
Input Data	Email : xyz@gmail.com password: abc123
Expected Output	Login Success
Actual Output	Login Success
Status	Pass

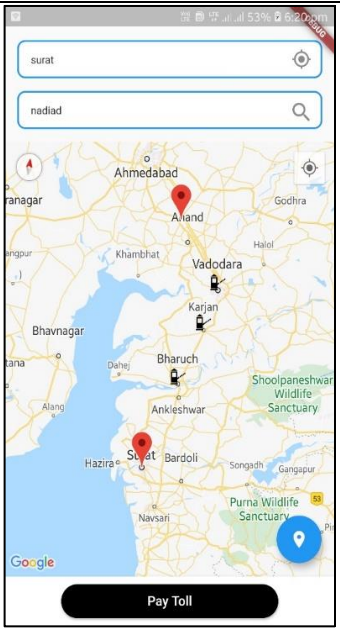
Test Case ID	TC_02
Requirement ID	R.1.2
Test Case Objective	Test Login
Step	1.Enter Email & Password 2. Login
Input Data	Emai:Null password:Null
Expected Output	Validation message
Actual Output	Validation message
Status	Pass

Test Case ID	TC_03
Requirement ID	R.1.2
Test Case Objective	Test Login
Step	1.Enter Email & Password 2.Login
Input Data	Email : xyz@gmail.com password: 123abc
Expected Output	Invalid Credentials
Actual Output	Invalid Credentials
Status	Pass

Test Case ID	TC_04
Requirement ID	R.1.1
Test Case Objective	Test Sign Up
Step	1.Enter Email & Password 2. Sign Up
Input Data	Email :abc@gmail.com password: xyz123
Expected Output	Sign Up Success
Actual Output	Success
Status	Pass

Test Case ID	TC_05
Requirement ID	R.1.1
Test Case Objective	Test Sign Up
Step	1.Enter Email & Password 2. Sign Up
Input Data	Email :abc@gmail.com password: xy12
Expected Output	Enter Password 6+ chars long
Actual Output	Enter Password 6+ chars long
Status	Pass

Test Case ID	TC_06
Requirement ID	R.1.1
Test Case Objective	Test Sign Up
Step	1.Enter Email & Password 2. Sign Up
Input Data	Email : abc password: xyy123
Expected Output	Enter valid email
Actual Output	Enter valid email
Status	Pass

Test Case ID	TC_07
Requirement ID	R.1.3 & R.1.4
Test Case Objective	Test Display Toll Booth
Step	1.Enter source 2.Enter Destination 3.Get Tollbooth
Input Data	Source : Surat Destination : Nadiad
Expected Output	Location of tollbooth at near Bharuch , Karjan , Vadodara
Actual Output	 surat nadiad Pay Toll
Status	Pass

Test Case ID	TC_08
Requirement ID	R.1.5 & R.1.6
Test Case Objective	Test Journey details
Step	1.Select Vehicle type 2. Journey date 3. Vehicle No.
Input Data	Vehicle type : Car/Jeep Journey date: 2020/04/23 Vehicle No: GJ055863
Expected Output	Payment amount :305
Actual Output	Payment amount :305
Status	Pass

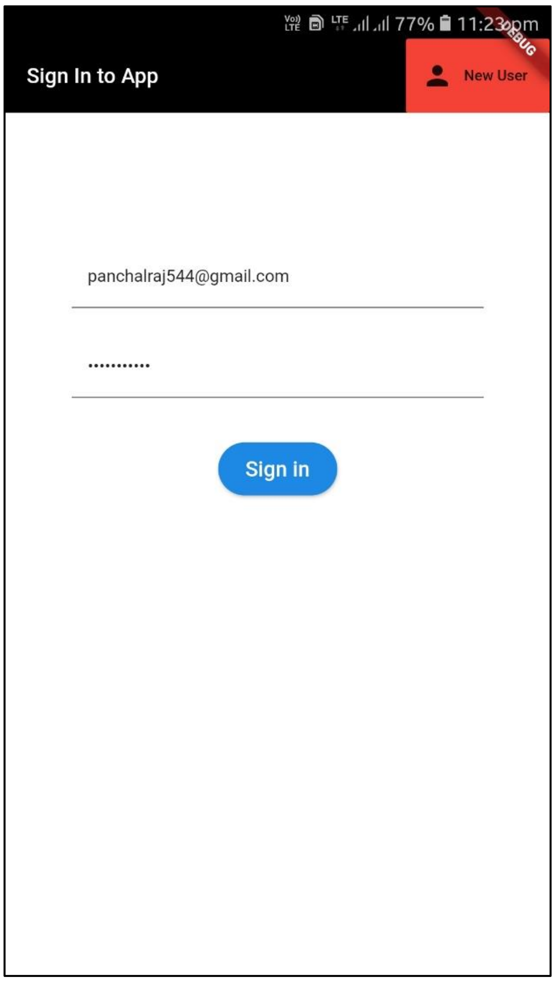
Test Case ID	TC_09
Requirement ID	R.1.5
Test Case Objective	Test Journey details
Step	1.Select Vehicle type 2. Journey date 3. Vehicle No.
Input Data	Vehicle type : Car/Jeep Journey date: 2020/04/23 Vehicle No: NULL
Expected Output	Enter Vehicle No.
Actual Output	Enter Vehicle No.
Status	Pass

Test Case ID	TC_10
Requirement ID	R.1.7
Test Case Objective	Test Payment Process
Step	1.Choose payment method 2. Enter payment details 3. Pay
Input Data	Correct details
Expected Output	Payment success
Actual Output	Payment success
Status	Pass

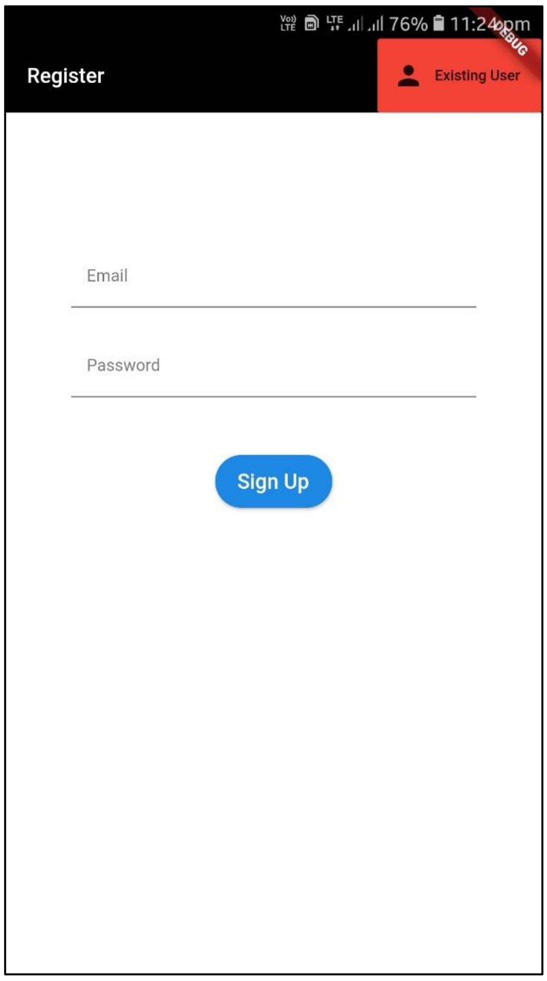
Test Case ID	TC_11
Requirement ID	R.1.7
Test Case Objective	Test Payment Process
Step	1.Choose payment method 2. Enter payment details 3. Pay
Input Data	Wrong details
Expected Output	Payment fail
Actual Output	Payment fail
Status	Pass

7. SCREEN-SHOTS

Sign In screen:



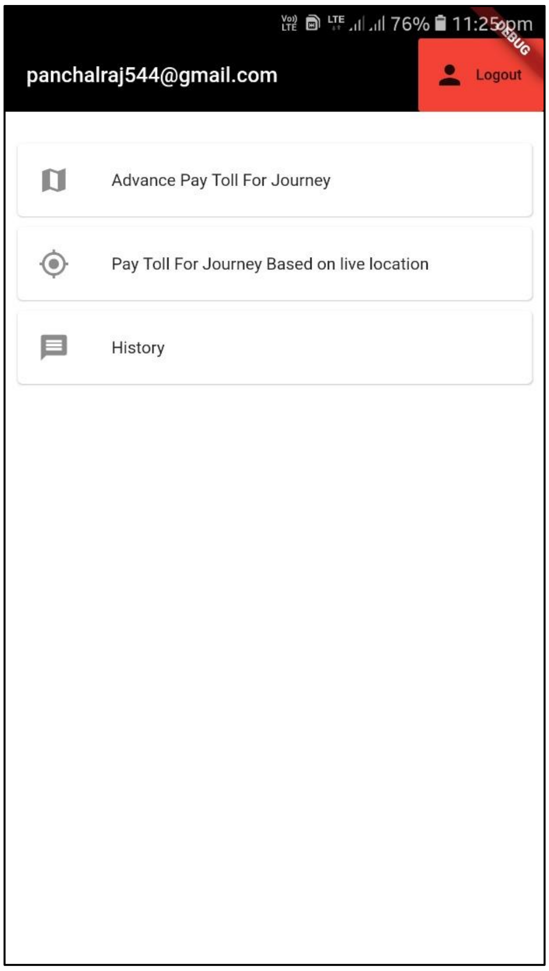
Sign Up screen:



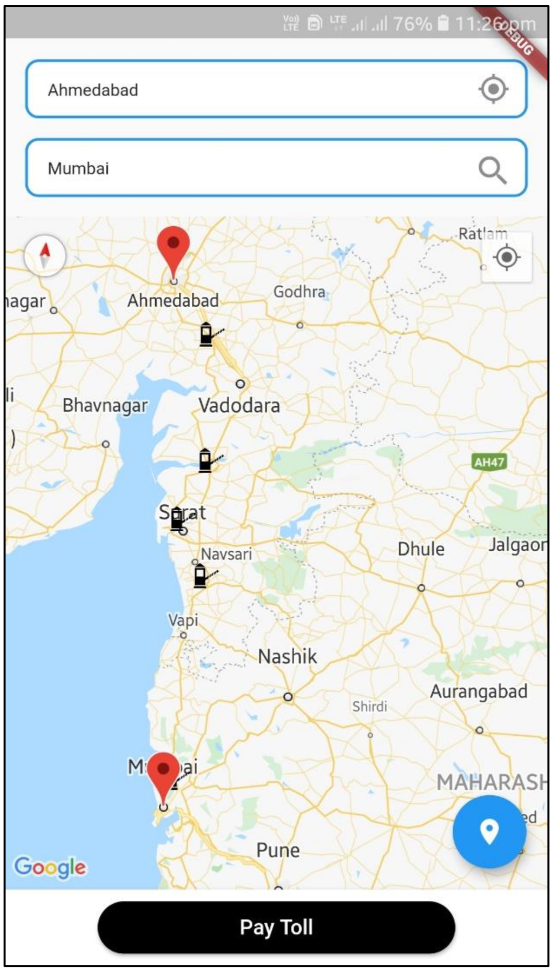
<u>Sign In screen</u> : It includes the field email and password so that user can login to app by entering email and password.

<u>Sign Up screen</u> : It also contains email and password field for new register for the app. By clicking on sign up button user can register yourself to the app.

Home screen:


panchalraj544@gmail.com
Logout
Advance Pay Toll For Journey
Pay Toll For Journey Based on live location
History

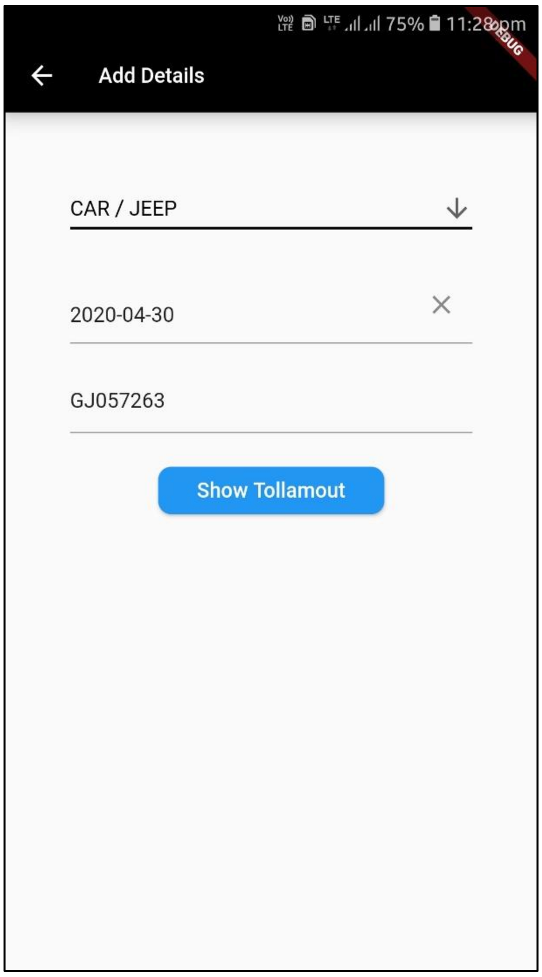
Map screen:



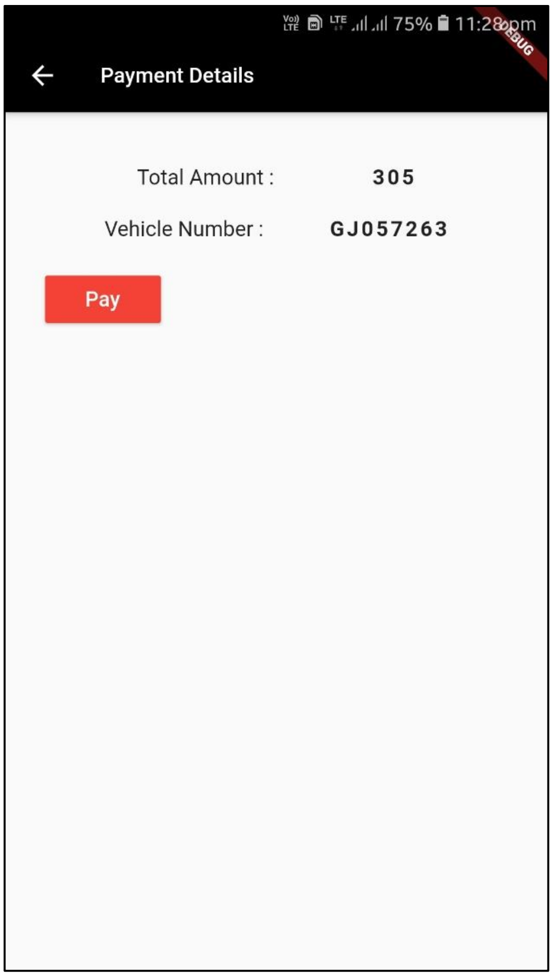
<u>Home screen:</u> When user enters correct and valid details for log in and sign up then clicks on the sign in and sign up respectively, then app shows home screen with different option.

<u>Map screen:</u> When user clicks on ‘Advance Pay Toll for Journey’ then app shows this screen. It has two field source and destination . If user enters source and destination in fields then clicks on floating button(blue button) all tollbooths would display on screen corresponding to source and destination.

Add details screen:



Payment details screen:



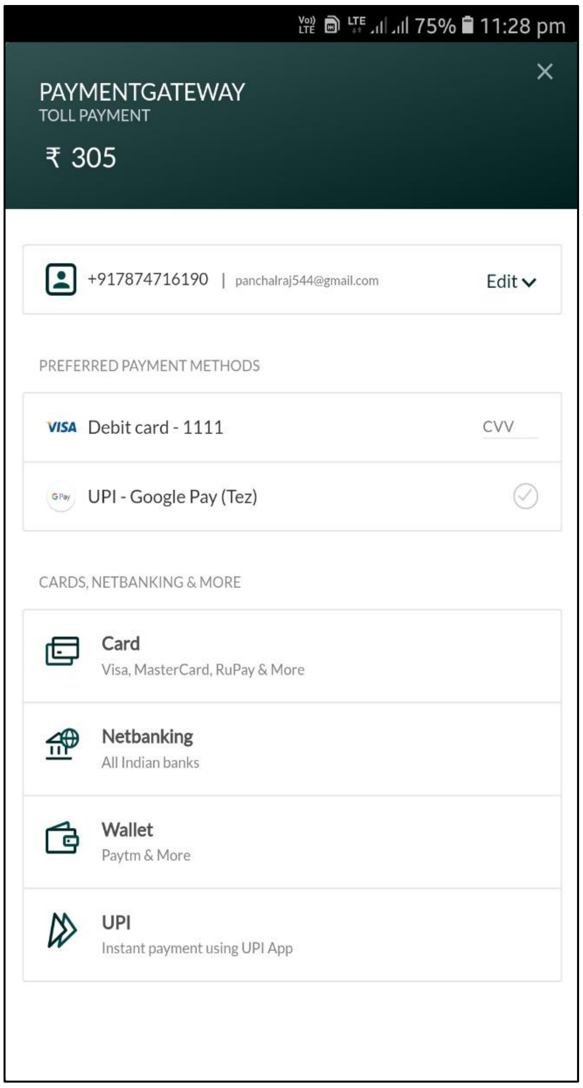
Total Amount : 305

Vehicle Number : GJ057263

Add details screen : When user click on paytoll from Map screen app shows this screen and ask for journey details like vehicle type, journey date, vehicle no etc. User needs to enter this detail and can know the toll amount by clicking on “Show Toll amount” button.

Payment details screen : It displays the toll amount corresponding to previous details which has been entered by the user.

Razorpay screen:


PAYMENTGATEWAY
TOLL PAYMENT
₹ 305

+917874716190 | panchalraj544@gmail.com Edit

PREFERRED PAYMENT METHODS

VISA Debit card - 1111 CVV

UPI - Google Pay (Tez)

CARDS, NETBANKING & MORE

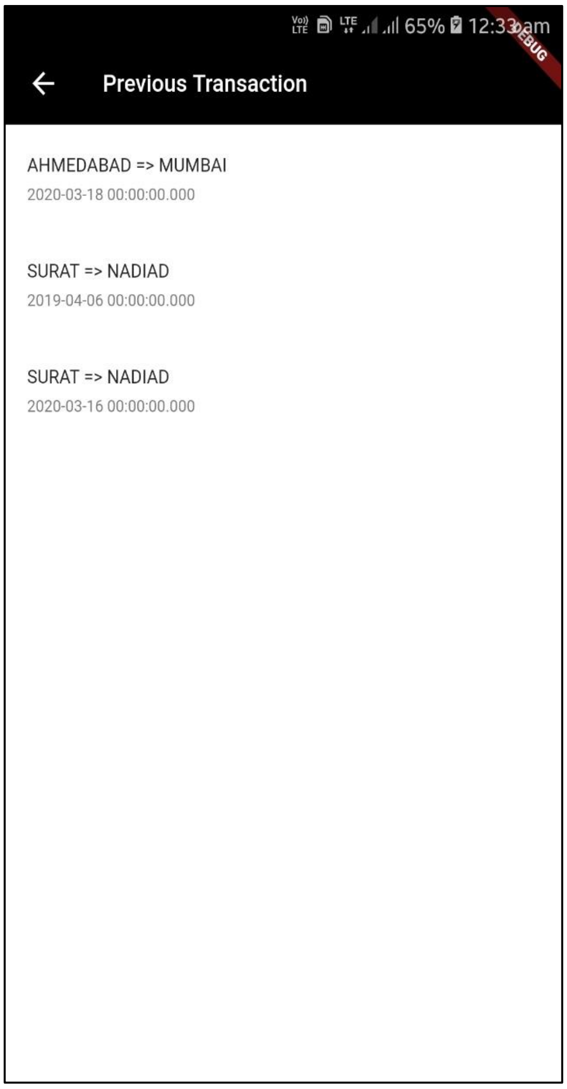
Card
Visa, MasterCard, RuPay & More

Netbanking
All Indian banks

Wallet
Paytm & More

UPI
Instant payment using UPI App

History screen:


Previous Transaction

AHMEDABAD => MUMBAI
2020-03-18 00:00:00.000

SURAT => NADIAD
2019-04-06 00:00:00.000

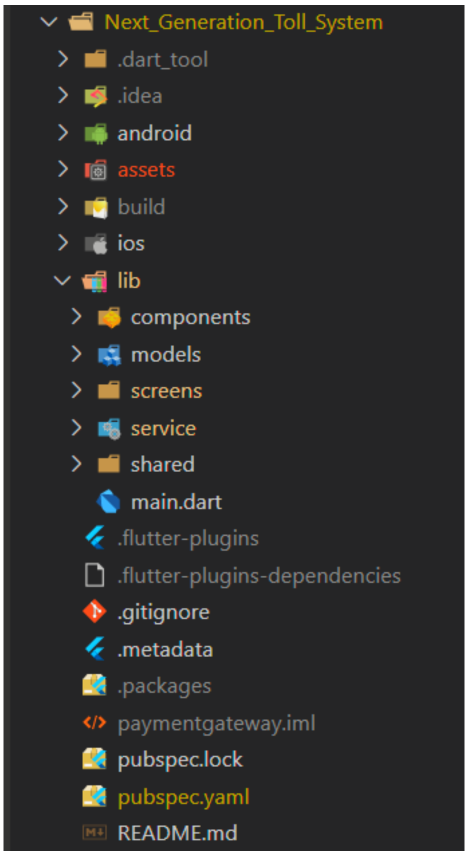
SURAT => NADIAD
2020-03-16 00:00:00.000

<u>Razor pay screen</u> : This screen provides for payment of toll. User can go for different way to pay like debit card, UPI, net banking and wallet.

<u>History screen</u> : User can show list of his previous transaction through this screen .

8. DEPLOYMENT

Folder structure of the project :



```
Next_Generation_Toll_System
  .dart_tool
  .idea
  android
  assets
  build
  ios
  lib
    components
    models
    screens
    service
    shared
    main.dart
  .flutter-plugins
  .flutter-plugins-dependencies
  .gitignore
  .metadata
  .packages
  paymentgateway.iml
  pubspec.lock
  pubspec.yaml
  README.md
```

Here lib folder contains all the necessary code file for the app. Assets folder contains all the assets like static data (JSON) file , configuration files, icons, and images.

In lib folder, screens contains all the dart file for UI , service contains all dart file for service like log in, sign in, log out, sign out etc., shared contains dart file for global purpose and main.dart file is like initial point of our app.

Application deployment:

During a typical development cycle, you test an app using flutter run at the command line, or by using the **Run** and **Debug** options in your IDE. By default, Flutter builds a debug version of your app.

When you’re ready to prepare a release version of your app, for example to publish to the Google Play Store.

Before publishing, you might want to put some finishing touches on your app.

1. Signing the app
    - Create key store
    - Reference the keystore from the app
    - Configure signing in gradle
2. Shrinking the code with R8
3. Reviewing the app manifest
4. Reviewing the build configuration
5. Building the app for release
    - Build an app bundle
    - Test the app bundle
    - upload the bundle to Google Play to test it. use the internal test track, or the alpha or beta channels to test the bundle before releasing it in production
    - Build an APK
    - Install an APK on a device
6. Publishing to the Google Play Store
7. Updating the app’s version number

For more information, you can refer this page.

9. CONCLUSION

Functionality Successfully Implemented :

- Sign in / Login
- Sign Up / Register
- Set source and destination
- Get all virtual tollbooths between source and destination
- Calculate toll amount according to requirement
- Payment method
- View history
- Add details of journey

Above all functionality has been successfully implemented by us.

10. LIMITATION AND FUTURE EXTENSION

Limitation of System :

This system only provides facility of pay toll in advanced and view history. This system doesn't display routes between source and destination because for this task direction API necessary but it is not available free so we can't able to use it.

Also, we have not implemented functionality that provides facility to pay toll based on live location because of some limitation. System display only virtual toll booth not original located toll booth. How to get original located toll booth that is question for us.

We want to fit this system in car but we have currently only implemented as mobile app. It is not possible to implement system in all cars. So, if system is not including in car then user need to follow original process.

Functionality which was not implemented:

- Pay toll based on live location:
  This means that whenever toll comes in journey amount automatically deducted from individual user's or traveller's account.

- Add virtual tollbooth:
  Though we add virtual location from directly through the database we haven't implement functionality that add virtual toll booth directly through the app.

- Monitoring user activity :
  We did not implement functionality that track the record of user's all activities like log In, log out , transaction record , search record etc.

Possible future extension to project:

In future we want to remove limitation from project that we have described above and also,

- Try to build better optimized system.
- Setup the system that can be used in as many cars as possible.

11. BIBLIOGRAPHY

Reference Link :

- ✓ https://console.firebase.google.com
- ✓ https://dashboard.razorpay.com/#/app
- ✓ https://cloud.google.com/docs/authentication/api-keys
- ✓ https://www.youtube.com/
- ✓ https://medium.com/@rajesh.muthyala/flutter-with-google-maps-and-google-place-85ccee3f0371
- ✓ https://pub.dev/
- ✓ https://github.com/
- ✓ https://console.cloud.google.com/home/dashboard?project=tollmap-266314
- ✓ https://flutter.dev/docs/deployment/android#install-an-apk-on-a-device

Work Distribution

Work done by Raj :

- ✓ Database Design
- ✓ Backend implementation
- ✓ Payment API implementation
- ✓ Gathering information for develop app
- ✓ Content writing

Work done by Jaydeep :

- ✓ Frontend implementation
- ✓ Entry of needed data into database
- ✓ Creation of map using map SDK for android API