FUEL UP

A

Project Report

Submitted in partial fulfilment of the

Requirements for the award of the Degree of

BACHELOR OF ENGINEERING

IN

INFORMATION TECHNOLOGY

By

S. AJAY (1602-17-737-005)

K.SAI KRISHNA REDDY (1602-17-737-033)

V.S.R. SRI SANJEEV (1602-17-737-042)

Under the guidance of

Ms. B. LEELAVATHY

ASSISTANT PROFESSOR

Department of Information Technology

Vasavi College of Engineering (Autonomous)

(Affiliated to Osmania University)

Ibrahimbagh, Hyderabad-31

2020

Vasavi College of Engineering (Autonomous)

(Affiliated to Osmania University)

Hyderabad-500 031

Department of Information Technology

DECLARATION BY THE CANDIDATE

I, **S.AJAY, K.SAI KRISHNA REDDY, V.S.R. SRI SANJEEV,** bearing hall ticket number, **1602-17-737-005, 1602-17-737-033,1602-17-737-042**, hereby declare that the project report entitled **"FUEL UP"** under the guidance of **Ms. B. LEELAVATHY,** ASSISTANT PROFESSOR, Department of Information Technology, Vasavi College of Engineering, Hyderabad, is submitted in partial fulfilment of the requirement of THEME BASED PROJECT of VI Semester of **Bachelor of Engineering** in **Information Technology**

This is a record of bonafide work carried out by me and the results embodied in this project.

S. Ajay

(1602-17-737-005)

K. Sai Krishna Reddy

(1602-17-737-033)

V.S.R. Sri Sanjeev

(1602-17-737-042)

Vasavi College of Engineering (Autonomous)

(Affiliated to Osmania University)

Hyderabad-500 031

Department of Information Technology

BONAFIDE CERTIFICATE

This is to certify that the project entitled **"FUEL UP"** being submitted by **S.AJAY, K.SAI KRISHNA REDDY, V.S.R. SRI SANJEEV,** bearing **1602-17-737-005, 1602-17-737-033, 1602-17-737-042**, in partial fulfilment of the requirements for the completion of MINI PROJECT of Bachelor of Engineering, VI Semester, in Information Technology is a record of bonafide work carried out by him/her under my guidance.

Ms. B. Leelavathy
Assistant Professor
Internal Guide

Dr. K. Ram Mohan Rao
HOD , IT

ACKNOWLEDGEMENT

The satisfaction that accompanies that the successful completion of the project would not have been possible without the kind support and help of many individuals. We would like to extend my sincere thanks to all of them.

We would like to take the opportunity to express our humble gratitude to **Mrs. B. Leelavathy Ma'am, Assistant Professor** under whom we executed this project.

We would also use this opportunity to thank our senior from IT department **Nagulavancha Lokesh (1602-16-737-081)**. We are grateful to his guidance, and constructive suggestions that helped us in the preparation of this project. His constant guidance and willingness to share his vast knowledge made us understand this project and its manifestations in great depths and helped us to complete the assigned tasks. We would like to thank all faculty members and staff of the Department of Information Technology for their generous help in various ways for the completion of this project.

Finally, yet importantly, We would like to express our heartfelt thanks to our **HOD Dr. K. Ram Mohan Rao Sir**, our theme based project coordinators **Mr. Sreenivasa Chakravarthy, Assistant Professor** and our classmates for their help and wishes for the successful completion of this project.

ABSTRACT

This project addresses the issue of break down of vehicles midway due to fuel insufficiency. Many people traveling in and around the cities and other places are facing problems with insufficient fuel when they miss the chance of refueling before the vehicle breaks down. Some times there is no enough fuel in the vehicle to reach the nearest fuel station. This leads them into difficult situations. The aim of this project is to find a solution for this problem.

The possible solution for the above problem is an online fuel delivery app, which will provide the user with the facility to order fuel online from anywhere at anytime from any fuel station. This will help the user to have fuel whenever and where ever he/she wants. This solution ensures that the user is provided with the best quality fuel possible and the delivery is made on time. This solution is one efficient way to save your time as you never have to stop for gas again and also saves you from many difficult situations where there is an insufficiency of fuel. The charge applicable for the fuel is also not much higher than the price you will have to spend at the station, rather all little amount is charged for the delivery.

TABLE OF CONTENTS

List of Figures

1.INTRODUCTION

1.1 Purpose

You must have already heard a million times about the convenience and efficiency brought in by the mobile apps. Like, you name and have it. And we really wonder whether there is a single service left behind for which there is no mobile app. I mean, from maid booking to car cleaning to salon booking to car-booking, today there is an app for every basic to every high-end service.

So, did you really think that there could be a mobile application even for gas or fuel on-demand Well, let me tell you there is. After all, that is the very purpose of mobile applications, to make your life easier in every way possible, so why not cover this zone as well. This project is about such app which deals with the requirement mentioned above

With this new service at disposal, now the users are heave a sigh of relief. Not anymore, they would have to run towards the gas station in order to get their vehicle tank filled and this certainly would make the daily commute for them immensely hassle-free and convenient. The fuel supply will reach them at their doorstep as and when required by simply using a mobile application. Its advantages are many

With a fuel delivery app, customers can get tanks of their vehicles filled and make their commute hassle-free. By Fuel UP fuel delivery app, they can place an order, and the supply of the fuel will reach their mentioned locations within the estimated time. The customers need to register their details in the app, share location and request fuel requirement

1.2 Scope

Online fuel booking will cut down the long line in petroleum retail outlets, increase digital transactions, and help maintain proper accounting with door delivery and portable petroleum stations.

1.3 How does it work

The functioning of an Fuel UP Fuel Delivery mobile app is the same as Uber. Like Uber comes at the customers doorstep upon booking, for pickup and drop. The fuel delivery services work in the same manner, where one can book the fuel for delivery at your location and the fuel is then supplied for the car or maybe generator, by simply using the mobile app. Here, the app will cover all of the information and do the reporting for stocking, procurement, delivery, and accounting of fuel. **Common challenges that the on-demand fuel delivery app comes across** It can be said that the only and one major challenge comes in the form of safety. For the concept of an on-demand fuel delivery mobile app, safety must be ensured at all cost and this concern needs to be addressed honestly, or it can even end up in a fatal accident.

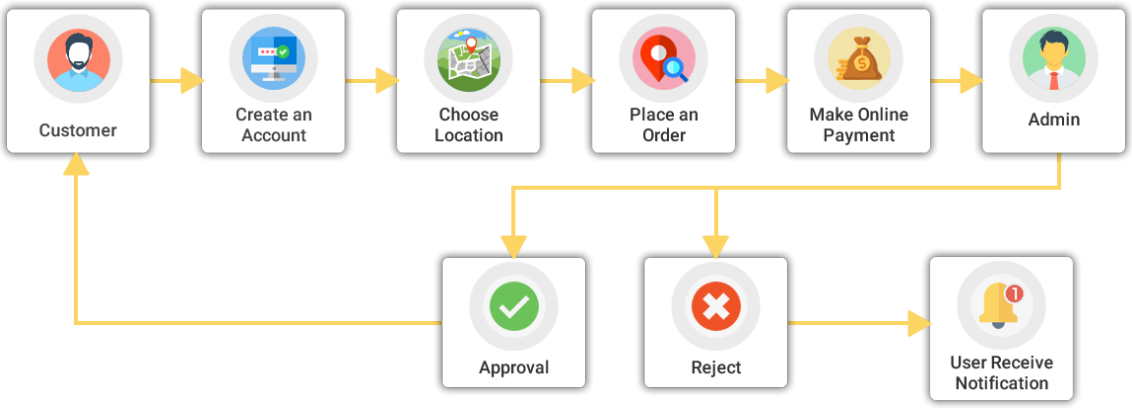

Fig 1.1 How does the app work?

2.SOFTWARE REQUIREMENTS SPECIFICATION

2.1 General Description

2.1.1 Product Function Overview

The Firebase Database will allow access only to authorized users with specific roles. Depending upon the user's role, he/she will be able to access only specific modules of the system.

A summary of major functions that the user will perform

• A login facility for enabling only authorized access to the system.

• Customers will be able to buy any type of fuel from the available stations and procced to payment section and.

• Customers can also view their previous order details, personal information etc.

2.1.2 User characteristics

• There is no need of any specific qualifications, experience to use the application. Anyone with basic knowledge of mobile and how to use maps can make use of application to order fuel.

•He must be able to perform online transaction through credit card if he needs to make payment through online.

2.1.3 General Characteristics

The application or software could run in any android device of minimum android version marshmallow.

This application can also be run in desktop through a virtual device.

2.2 System Analysis and Requirements

2.2.1 Functional Requirements

1. **Registration**: If customer wants to order the food then he/she must be registered, unregistered user can't go for ordering.

2. **Login**: The customer login to the system by entering valid user id and password for ordering.

3. **Order now**: In the system all the locations of stations available to his location are displayed with their names as markers on map.

4. **Modify**: System can make changings in menu like adding or removing fuel types which are not available and their prices.

5**. Select fuel and quantity**: Type of fuel is selected and quantity to be supplied customer feel free to order.

6. **Changes to order**: Changes to order means the customer can make changings in order like he/she can delete or add quantity of fuel in order.

7. **Review the order before submitting**: Before submitting the complete order is reviewed to the customer. Customer name, phone number, location (address) and placed order, hen finally order is submitted.

8. **Payment:** For customer there are many types of secure billing will be prepaid as debit or credit card, postpaid as after delivering, check or bank draft.

9. **Provide delivery and payment details**: Here bill is generated, order no. and payment is given and confirmation of delivery is done.

10. **Logout**: After the payment or surf the product the customer will log out.

11. **View Orders:** User must be able to view all his past orders and details of orders

2.2.2 Non-functional Requirements

1. **Portability**: System running on one platform can easily be converted to run on another platform.

2. **Reliability:** The ability of the system to behave consistently in a user-acceptable manner when operating within the environment for which the system was intended.

3. **Availability**: The system should be available at all times, meaning the user can access it using a web browser, only restricted by the down time of the server on which the system runs.

4. **Maintainability**: A commercial database is used for maintaining the database and the application server takes care of the site.

5. **Security**: Secure access of confidential data (customer information).

6. **User friendly**: System should be easily used by the customer.

7. **Performance**: Performance should be fast.

8. **Efficient**: System should be efficient that it won't get hang if heavy traffic of order is placed.

9. **Safety**: Data in the database of system should not loss or damage.

10. **Privacy**: Personal data of the system should not disclose to anyone.

2.2.3 Software Requirements

PLATFORM- ANDROID STUDIO

- Front End - XML (Extended Mark up Language)
- Back End – Java

DATABASE- FIREBASE

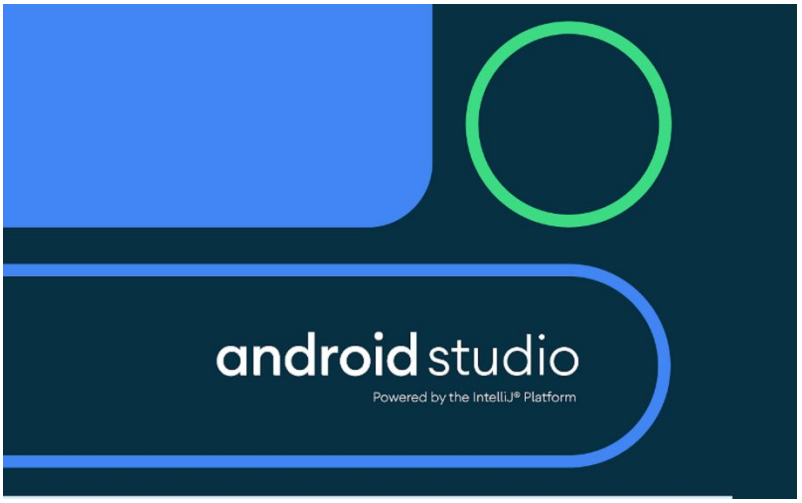

Fig 2.1 Android Studio

Fig 2.2 Firebase

2.3 Design Constraints

- Limited screen size displays few words
- One screen at a time
- Low Battery capacity
- Wireless LAN with high latency
- Low processor speed
- Low memory
- Software development constraints
- User constraint
- Platform constraint
- Task constraint
- Device constraint

2.4 Android and it's overview

Android is a Linux-based operating system developed for smart phones or tablet computers. It is a stack of software that includes operating system, middleware and libraries and APIs written in C.

2.5 Android Features

2.5.1 Application Framework

Android application framework is supported by number of open source libraries like OpenSSL, SQLite, and Libc. The application framework is also supported by the Android core libraries.

Android code is written primarily in Java programming language, and is compiled with the help of Android SDK tools. On compiling, it generates an Android package (commonly known as .apk) file which is used to install the application on android device.

2.5.2 App Components

To aid in android development, there are mainly four types of components. Each component serves a distinct purpose and allows system to interact with your application in different ways. Broadly speaking, there are four types of components.

- **Activity**- An activity represents a single screen with a user interface. For example, in this Attendance Tracker application, activities include adding a student, register or viewing student's attendance record.
  Life cycle of Activity is as shown below
- **Services**- A Service is an application component that can perform long-running operations in the background, and it doesn't provide a user interface. Another application component can start a service, and it continues to run in the background even if the user switches to another application.
- **Content provider** - A content provider manages access to a central repository of data. A provider is part of an Android application, which often provides its own UI for working with the data.
- **Broadcast receivers**- Broadcast receivers are components in your Android application that listen in on broadcast messages

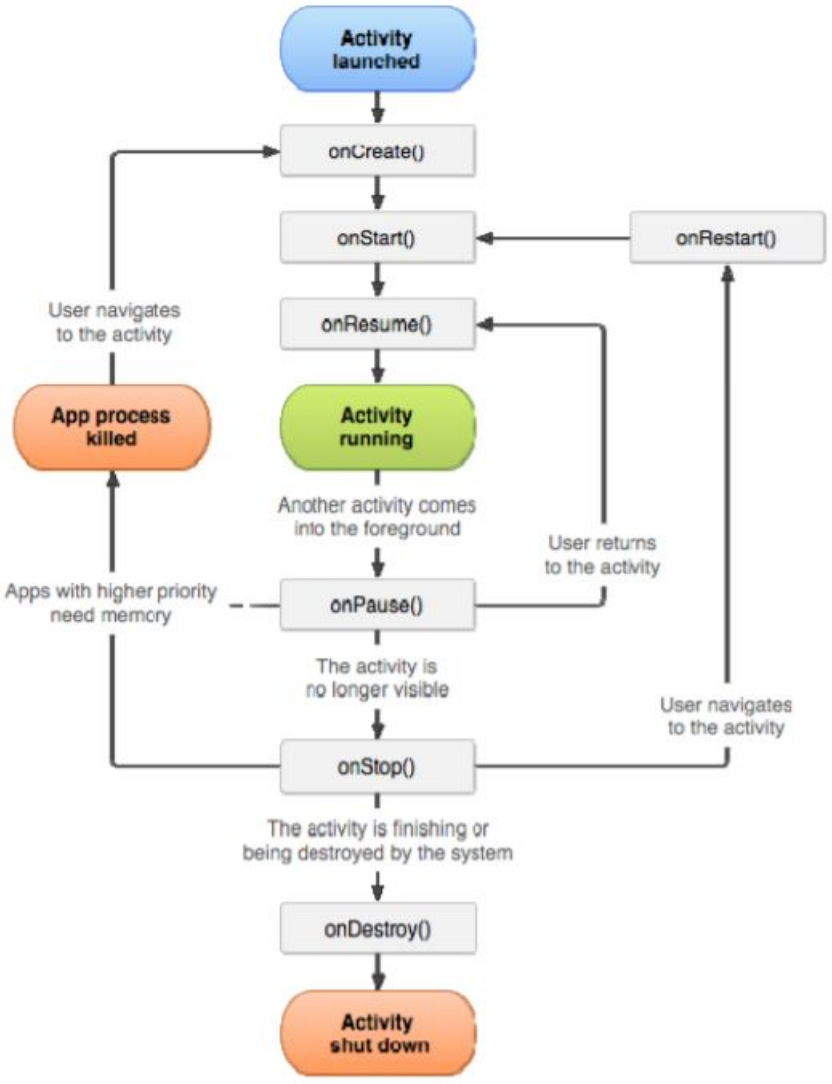

Fig 2.3 Activity Life Cycle

2.5.3 Android Application files

Files can be broadly divided into three categories

- Java file
- Layout file
- Manifest file

Java Files

These files are the files where all the processing of the events happens, and allows user to interact with the system. Through these files, the layouts can be added

dynamically, the user entered values in the text boxes or other input can be obtained and stored.

Layout Files

A layout defines the visual structure for a user interface, such as the UI for any activity. These files are responsible for defining the user Input. The Android framework gives you the flexibility to use either or both of these methods for declaring and managing your application's UI.

Manifest Files

It is the main part of the android application. This file contains all the information about the application – what android operating system components are present in the application, what permissions are required by the application etc.

2.6 Firebase

Firebase is a toolset to “build, improve, and grow your app”, and the tools it gives you cover a large portion of the services that developers would normally have to build themselves, but don’t really want to build, because they’d rather be focusing on the app experience itself. This includes things like analytics, authentication, databases, configuration, file storage, push messaging, and the list goes on. The services are hosted in the cloud, and scale with little to no effort on the part of the developer.

3.SYSTEM DESIGN

3.1 USE CASES

Use case diagrams are the diagrammatic representation depicting user's interactions with the system. This diagram shows different types of users and various ways in which these users interact with the system.

The following are the actors for Fuel UP App:

1.Customer- Customer who uses the application can be able to perform all the use cases such as login, register, ordering etc.

2.Delivery Person- A delivery person gets the order from user and has to deliver the required fuel.

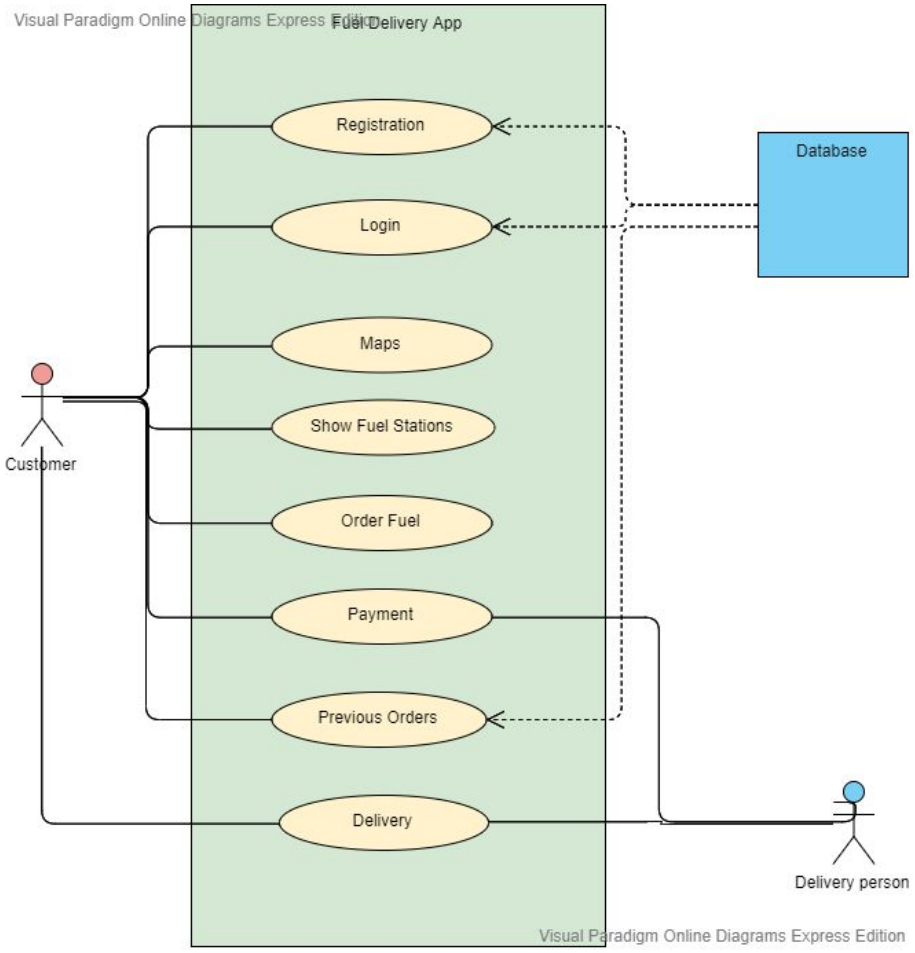

Fig 3.1 Use Case Diagram

3.2 UML Static Diagram

Class Diagram- a class diagram in the is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.

3.3 Run Time Diagram

Sequence Diagram- Sequence Diagrams are interaction diagrams that detail how operations are carried out. They capture the interaction between objects in the context of a collaboration. Sequence Diagrams are time focus and they show the order of the interaction visually by using the vertical axis of the diagram to represent time what messages are sent and when.

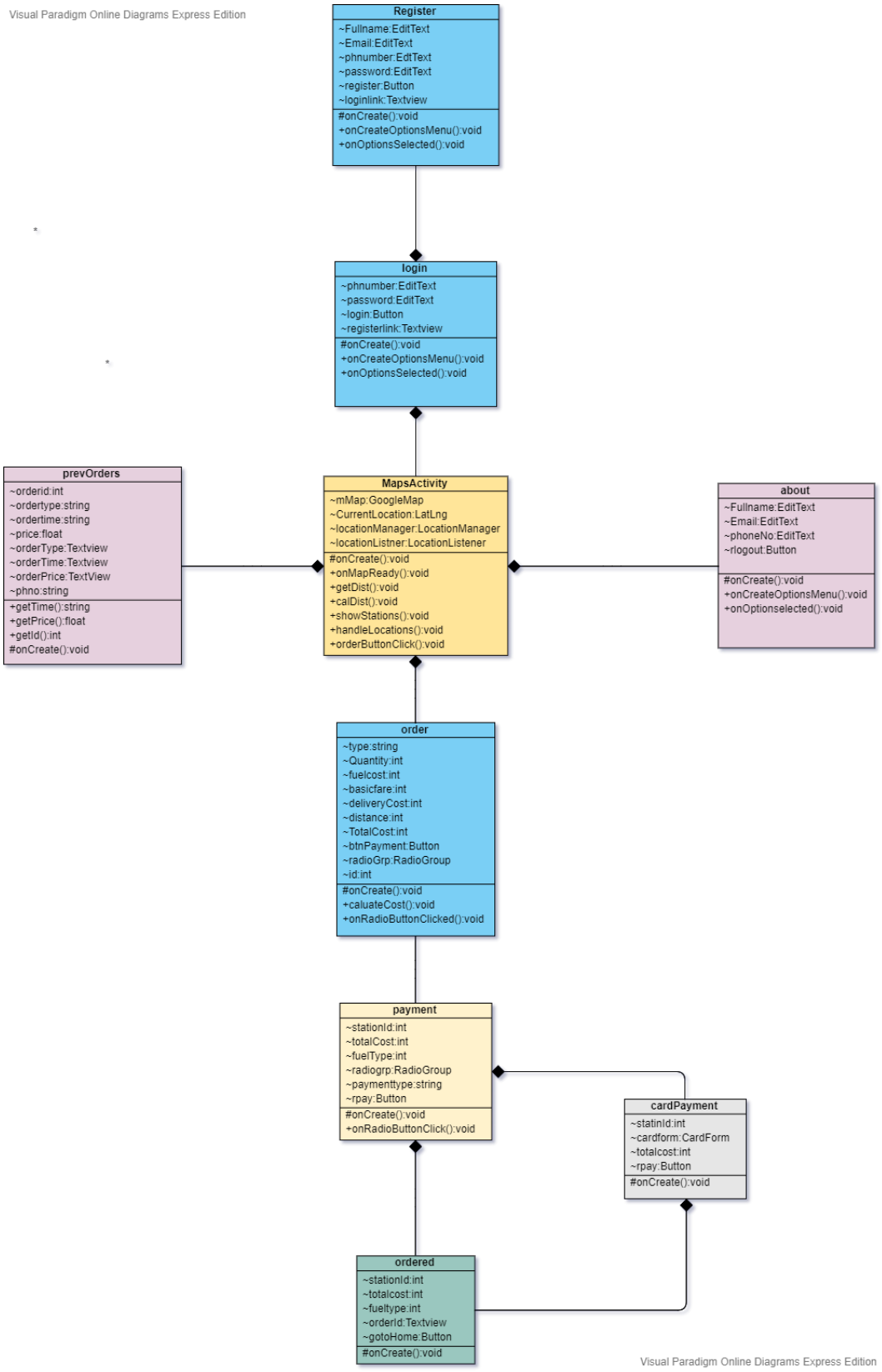

Fig 3.2 Class Diagram

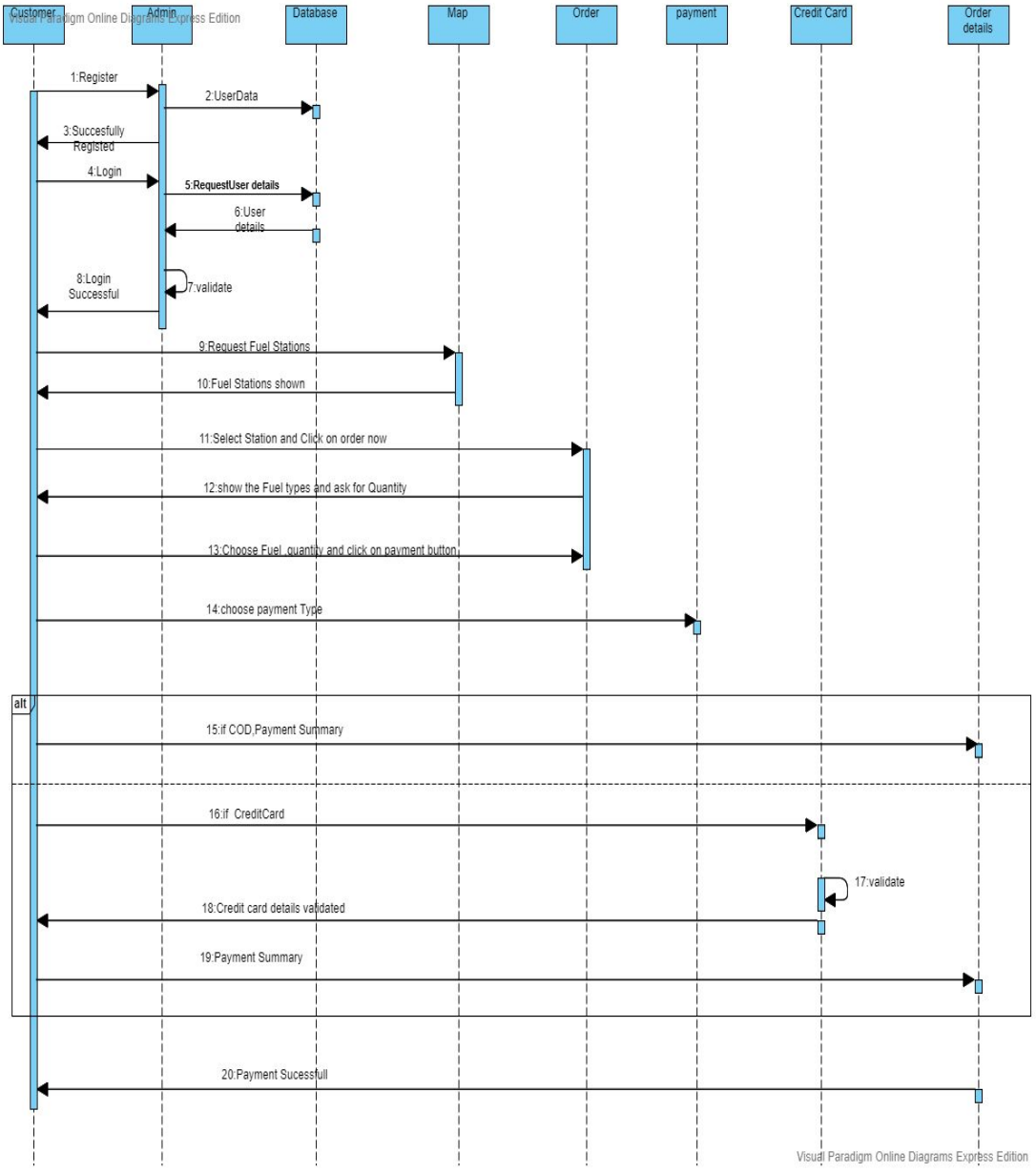

Fig 3.3 Sequence Diagram

3.4 Database Design

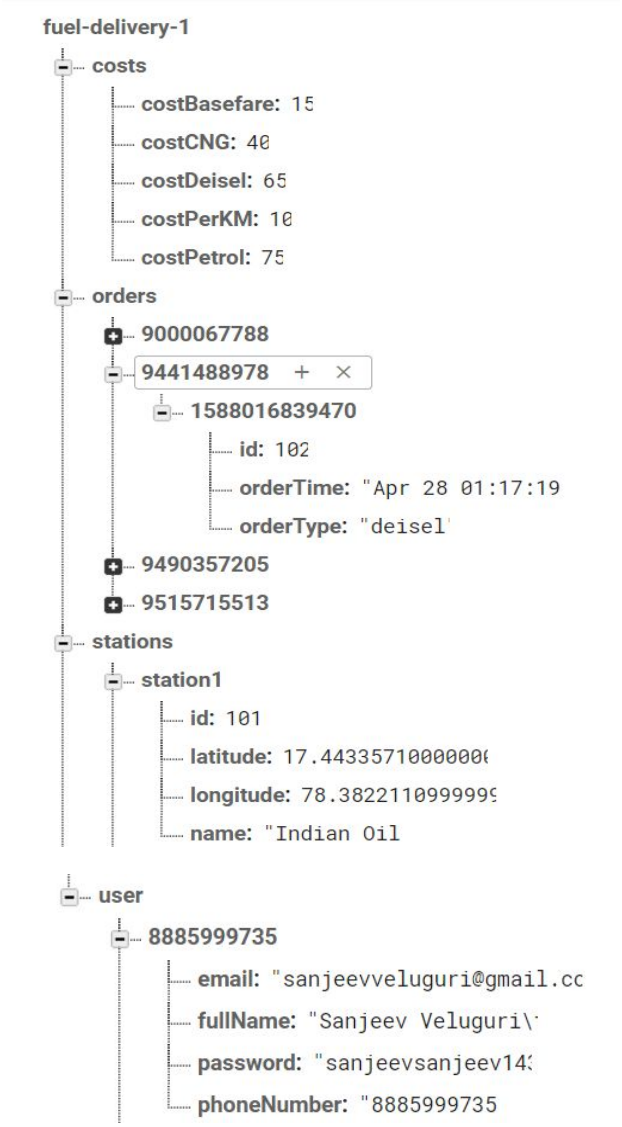

Fig 3.4 Database design

4. IMPLEMENTATION

4.1 Modules

Maps Activity- This is the main activity of the application. Any further process must begin from here. This module or activity displays the current location of the user when logged in through your account and displays available fuel stations to your location from where you could order fuel by clicking on location marker. This also contains a menu bar where a list of options are mentioned like login, order now, about etc.

Register- This screen enables us a new user to register to the app by entering information Name, phone number, email and password

Login- If the user is already registered to app this screen enables him to login directly by entering his/her phone number and password. His credentials are validated with that of stored in the database. He could only if his credentials are validated true.

About- This screen shows the details of the user currently logged in.

Order now- When this option is selected from the menu bar available petrol stations are showed on the map.

Order - After selecting a station this screen shows you the type of fuel to be selected as radio button and quantity is to be selected. Based on your selection and distance a bill is generated and shown.

Payment- This screen shows you the available modes of payment for the bill.

Card Payment- This screen makes the user to enter his card details such as card number, expiry date and cvv.

Previous Orders- This screen displays you all the previous orders made by you and details of the order.

Ordered- After confirmation of payment an ID for delivery is generated and displayed.

4.2 Activities and layout files

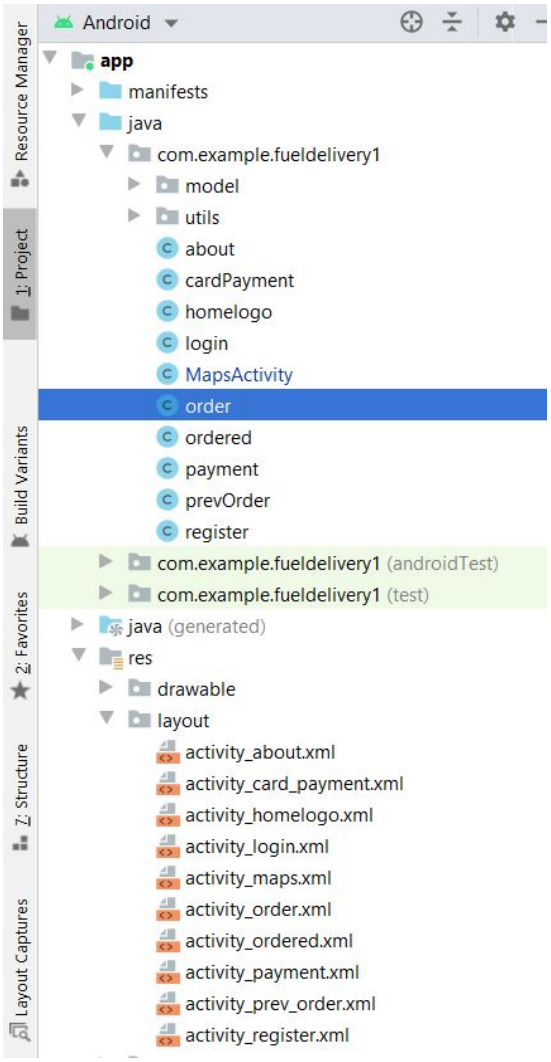

Fig 4.1 Activity and Layout files

4.3 Code

MapsActivity.java

```
public class MapsActivity extends AppCompatActivity implements
OnMapReadyCallback {

    public boolean homePageDisplay = true;

    private GoogleMap mMap;
```

```
    LocationManager locationManager;
    LocationListener locationListener;
    LatLng currentLocation;
    public int dp(float value) {
        if (value == 0) {
            return 0;
        }
        return (int) Math.ceil(getResources().getDisplayMetrics().density * value);
    }
    private Bitmap createUserBitmap() {
        Bitmap result = null;
        try {
            result = Bitmap.createBitmap(dp(62), dp(76),
Bitmap.Config.ARGB_8888);
            result.eraseColor(Color.TRANSPARENT);
            Canvas canvas = new Canvas(result);
            Drawable drawable = getResources().getDrawable(R.drawable.livepin);
            drawable.setBounds(0, 0, dp(62), dp(76));
            drawable.draw(canvas);
            Paint roundPaint = new Paint(Paint.ANTI_ALIAS_FLAG);
            RectF bitmapRect = new RectF();
            canvas.save();
            Bitmap bitmap = BitmapFactory.decodeResource(getResources(),
R.drawable.marker_image_new);
            if (bitmap != null) {
                BitmapShader shader = new BitmapShader(bitmap,
Shader.TileMode.CLAMP, Shader.TileMode.CLAMP);
                Matrix matrix = new Matrix();
                float scale = dp(52) / (float) bitmap.getWidth();
```

```
                matrix.postTranslate(dp(5), dp(5));
                matrix.postScale(scale, scale);
                roundPaint.setShader(shader);
                shader.setLocalMatrix(matrix);
                bitmapRect.set(dp(5), dp(5), dp(52 + 5), dp(52 + 5));
                canvas.drawRoundRect(bitmapRect, dp(26), dp(26), roundPaint);
            }
            canvas.restore();
            try {
                canvas.setBitmap(null);
            } catch (Exception e) {}
        } catch (Throwable t) {
            t.printStackTrace();
        }
        return result;
    }
    public void onRequestPermissionsResult(int requestCode, @NonNull String[]
permissions, @NonNull int[] grantResults) {
        super.onRequestPermissionsResult(requestCode, permissions, grantResults);
        if (grantResults.length > 0 && grantResults[0] ==
PackageManager.PERMISSION_GRANTED) {
            if (ContextCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_FINE_LOCATION) ==
PackageManager.PERMISSION_GRANTED)

locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 1,
10, locationListener);
        }
    }
    protected void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_maps);


        final SupportMapFragment mapFragment = (SupportMapFragment)
getSupportFragmentManager().findFragmentById(R.id.map);
        assert mapFragment != null;
        mapFragment.getMapAsync(this);
    }
    public void onMapReady(GoogleMap googleMap) {
        mMap = googleMap;
        mMap.getUiSettings().setZoomControlsEnabled(true);
        locationManager = (LocationManager)
this.getSystemService(Context.LOCATION_SERVICE);
        locationListener = new LocationListener() {
            public void onLocationChanged(Location location) {}
            public void onStatusChanged(String provider, int status, Bundle extras) {}
            public void onProviderEnabled(String provider) {}
            public void onProviderDisabled(String provider) {}
        };
        if (ContextCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_FINE_LOCATION) !=
PackageManager.PERMISSION_GRANTED) {
            ActivityCompat.requestPermissions(this, new
String[]{Manifest.permission.ACCESS_FINE_LOCATION}, 1);
        }
        else {
locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 1,
10, locationListener);
            Location lastKnownLocation = (Location)
locationManager.getLastKnownLocation(LocationManager.GPS_PROVIDER);
```

```
            LatLng lastKnownCoordinates = new
LatLng(lastKnownLocation.getLatitude(), lastKnownLocation.getLongitude());
            handleNewLocation(lastKnownLocation,mMap);
            currentLocation = lastKnownCoordinates;
        }
    }
    public int getDistance(LatLng StartP, LatLng EndP) {
        int Radius = 6371;// radius of earth in Km
        double lat1 = StartP.latitude;
        double lat2 = EndP.latitude;
        double lon1 = StartP.longitude;
        double lon2 = EndP.longitude;
        double dLat = Math.toRadians(lat2 - lat1);
        double dLon = Math.toRadians(lon2 - lon1);
        double a = Math.sin(dLat / 2) * Math.sin(dLat / 2)
                + Math.cos(Math.toRadians(lat1))
                * Math.cos(Math.toRadians(lat2)) * Math.sin(dLon / 2)
                * Math.sin(dLon / 2);
        double c = 2 * Math.asin(Math.sqrt(a));
        Double valueResult = Radius * c;
        double km = valueResult / 1;
        DecimalFormat newFormat = new DecimalFormat("####");
        int kmInDec = Integer.valueOf(newFormat.format(km));
        return kmInDec;
    }
    public boolean CalculationByDistance(LatLng StartP, LatLng EndP) {
        int Radius = 6371;// radius of earth in Km
        double lat1 = StartP.latitude;
```

```
        double lat2 = EndP.latitude;
        double lon1 = StartP.longitude;
        double lon2 = EndP.longitude;
        double dLat = Math.toRadians(lat2 - lat1);
        double dLon = Math.toRadians(lon2 - lon1);
        double a = Math.sin(dLat / 2) * Math.sin(dLat / 2)
                + Math.cos(Math.toRadians(lat1))
                * Math.cos(Math.toRadians(lat2)) * Math.sin(dLon / 2)
                * Math.sin(dLon / 2);
        double c = 2 * Math.asin(Math.sqrt(a));
        double valueResult = Radius * c;
        double km = valueResult / 1;
        DecimalFormat newFormat = new DecimalFormat("####");
        int kmInDec = Integer.valueOf(newFormat.format(km));
        if(kmInDec<6){return true;}
        else{return false;}
    }
    private void handleNewLocation(Location location,GoogleMap mMap) {
        Log.d("USER REQUEST :", location.toString());
        double currentLatitude = location.getLatitude();
        double currentLongitude = location.getLongitude();
        LatLng latLng = new LatLng(currentLatitude, currentLongitude);
        mMap.moveCamera(CameraUpdateFactory.newLatLng(latLng));
        mMap.animateCamera(CameraUpdateFactory.newLatLngZoom(latLng,
15));
        mMap.setMyLocationEnabled(true);
    }
    public void orderButtonClicked(final GoogleMap mMap)
```

```
    {
        mMap.clear();
                    if (ContextCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_FINE_LOCATION) !=
PackageManager.PERMISSION_GRANTED) {
            ActivityCompat.requestPermissions(this, new
String[]{Manifest.permission.ACCESS_FINE_LOCATION}, 1);
        }
        else {

locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 1,
10, locationListener);
            Location lastKnownLocation = (Location)
locationManager.getLastKnownLocation(LocationManager.GPS_PROVIDER);
            LatLng lastKnownCoordinates = new
LatLng(lastKnownLocation.getLatitude(), lastKnownLocation.getLongitude());
            currentLocation = lastKnownCoordinates;
            handleNewLocation(lastKnownLocation,mMap);
        }
        if(userSharedPreference.getUserName(MapsActivity.this).length() != 0) {
            FirebaseDatabase firebaseDatabase = FirebaseDatabase.getInstance();
            DatabaseReference dbref = firebaseDatabase.getReference();
            dbref.child("stations").addValueEventListener(new ValueEventListener()
{
                @Override
                public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
                    Bitmap bitmap = createUserBitmap();
                    for (DataSnapshot f : dataSnapshot.getChildren()) {
                        stations st = f.getValue(stations.class);
                        double latitude = st.getLatitude();
                        double longitude = st.getLongitude();
```

```
                final LatLng storedLocation = new LatLng(latitude, longitude);

                if(CalculationByDistance(currentLocation,storedLocation) == true)
{

                    mMap.addMarker(new
MarkerOptions().position(storedLocation).title(st.getName()).icon(BitmapDescrip
torFactory.fromBitmap(bitmap))).setTag(st.getId());

                }

                mMap.setOnInfoWindowClickListener(new
GoogleMap.OnInfoWindowClickListener() {

                    public void onInfoWindowClick(Marker marker) {

                        if(marker.getPosition().latitude != currentLocation.latitude &
marker.getPosition().longitude!=currentLocation.longitude) {

                            Intent intent1 = new Intent(MapsActivity.this, order.class);

                            Integer id = (Integer) marker.getTag();

                            intent1.putExtra("stationID", id);

                            Integer distance = getDistance(currentLocation,
marker.getPosition());

                            intent1.putExtra("distance", distance);

                            startActivity(intent1);

                        }

                        else

                        {

                            Toast.makeText(MapsActivity.this, "That's Your Location",
Toast.LENGTH_LONG).show();

                        }

                    }

                });

            }

            Toast.makeText(MapsActivity.this, "Please select a station from
those displayed on the MAP!", Toast.LENGTH_LONG).show();

        }
```

```
            public void onCancelled(@NonNull DatabaseError databaseError) {

            }

        });

    }

    else

    {

        Toast.makeText(MapsActivity.this, "Please login before you Order!",
Toast.LENGTH_LONG).show();

    }

  }

  public void showAllStationsButtonClicked(final GoogleMap mMap)

  {

    mMap.clear();

    FirebaseDatabase firebaseDatabase = FirebaseDatabase.getInstance();

    DatabaseReference dbref = firebaseDatabase.getReference();

    dbref.child("stations").addValueEventListener(new ValueEventListener() {

      public void onDataChange(@NonNull DataSnapshot dataSnapshot) {

        for (DataSnapshot f : dataSnapshot.getChildren()) {

          stations st = f.getValue(stations.class);

          LatLng storedLocation = new LatLng(st.getLatitude(),
st.getLongitude());

          Bitmap bitmap = createUserBitmap();

          mMap.addMarker(new
MarkerOptions().position(storedLocation).title(st.getName()).icon(BitmapDescrip
torFactory.fromBitmap(bitmap))).setTag(st.getId());

        }

        Toast.makeText(MapsActivity.this, "All the Stations are showed on the
MAP!", Toast.LENGTH_LONG).show();

      }

      public void onCancelled(@NonNull DatabaseError databaseError) {
```

```
        }
    });
    mMap.setOnInfoWindowClickListener(null);
}
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.main_axtivity_menu, menu);
    return true;
}
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.login_item:
            Intent login=new Intent(MapsActivity.this,login.class);
            startActivity(login);
            return true;
        case R.id.home_item:
            Intent home = new Intent(MapsActivity.this,MapsActivity.class);
            startActivity(home);
            return true;
        case R.id.order_item:
            orderButtonClicked(mMap);
            return true;
        case R.id.prev_orders:
            Intent prev = new Intent(MapsActivity.this,prevOrder.class);
            startActivity(prev);
        case R.id.showAllStations:
            showAllStationsButtonClicked(mMap);
        default:
```

```
            return super.onOptionsItemSelected(item);
        }
    }
}
```

4.3.1 Station Locations

The code for getting information about the fuel stations only nearer or available to user present location is as follows

```
    public boolean CalculationByDistance(LatLng StartP, LatLng EndP) {
        int Radius = 6371;// radius of earth in Km
        double lat1 = StartP.latitude;
        double lat2 = EndP.latitude;
        double lon1 = StartP.longitude;
        double lon2 = EndP.longitude;
        double dLat = Math.toRadians(lat2 - lat1);
        double dLon = Math.toRadians(lon2 - lon1);
        double a = Math.sin(dLat / 2) * Math.sin(dLat / 2)
                + Math.cos(Math.toRadians(lat1))
                * Math.cos(Math.toRadians(lat2)) * Math.sin(dLon / 2)
                * Math.sin(dLon / 2);
        double c = 2 * Math.asin(Math.sqrt(a));
        double valueResult = Radius * c;
        double km = valueResult / 1;
        DecimalFormat newFormat = new DecimalFormat("####");
        int kmInDec = Integer.valueOf(newFormat.format(km));
        if(kmInDec<6){return true;}
        else{return false;}
```

```
    }
```

4.3.2 Card Payment

```
public class cardPayment extends AppCompatActivity {
    public Integer stationID,totalcost;
    String FuelType;
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_card_payment);
        stationID = getIntent().getIntExtra("stationID",0);
        totalcost = getIntent().getIntExtra("Totalcost",0);
        FuelType = getIntent().getStringExtra("FuelType");
        final CardForm cardForm = findViewById(R.id.card_form);
        Button rbtnCompletePayment = findViewById(R.id.btnCompletePayment);
        cardForm.cardRequired(true)
                .expirationRequired(true)
                .cvvRequired(true)
                .setup(cardPayment.this);

cardForm.getCvvEditText().setInputType(InputType.TYPE_CLASS_NUMBER |
InputType.TYPE_NUMBER_VARIATION_PASSWORD);
        getWindow().clearFlags(WindowManager.LayoutParams.FLAG_SECURE);
        rbtnCompletePayment.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                if (cardForm.isValid()) {
                    AlertDialog.Builder alertBuilder = new
AlertDialog.Builder(cardPayment.this);
                    alertBuilder.setTitle("Confirm before purchase");
                    alertBuilder.setMessage("Card number: " +
cardForm.getCardNumber() + "\n" +
```

```
                    "Card expiry date: " +
cardForm.getExpirationDateEditText().getText().toString()
                    );
            alertBuilder.setPositiveButton("Confirm", new
DialogInterface.OnClickListener() {
                @Override
                public void onClick(DialogInterface dialogInterface, int i) {
                    dialogInterface.dismiss();
                    Intent del = new Intent(cardPayment.this,ordered.class);
                    del.putExtra("Totalcost",totalcost);
                    del.putExtra("stationID",stationID);
                    del.putExtra("FuelType",FuelType);
                    startActivity(del);
                }
            });
            alertBuilder.setNegativeButton("Cancel", new
DialogInterface.OnClickListener() {
                @Override
                public void onClick(DialogInterface dialogInterface, int i) {
                    dialogInterface.dismiss();
                }
            });
            AlertDialog alertDialog = alertBuilder.create();
            alertDialog.show();

        }else {
            Toast.makeText(cardPayment.this, "Please fill all the details
correctly!", Toast.LENGTH_LONG).show();
        }
    }
});
```

```
    }
}
```

4.4 GitHub link

https://github.com/sanju029/fueldelivery

5.RESULTS

5.1 UI (Screenshots)

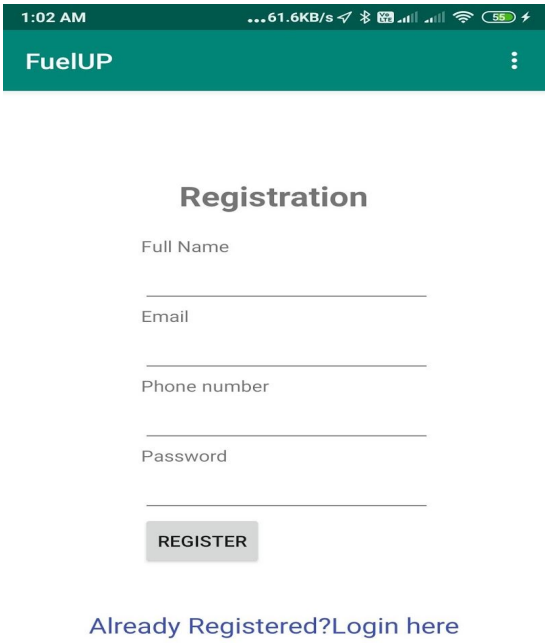

Fig 5.1 Registration Screen

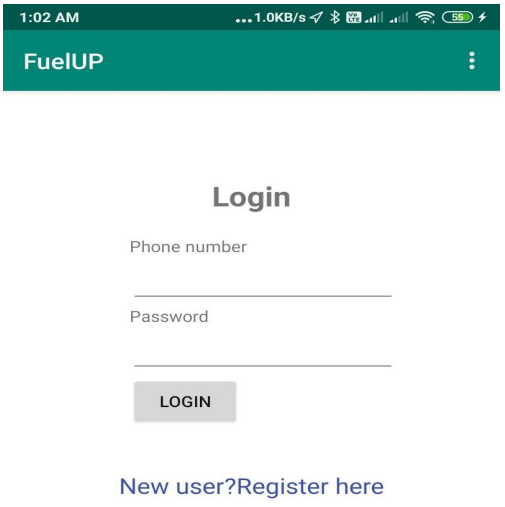

Fig 5.2 Login Screen

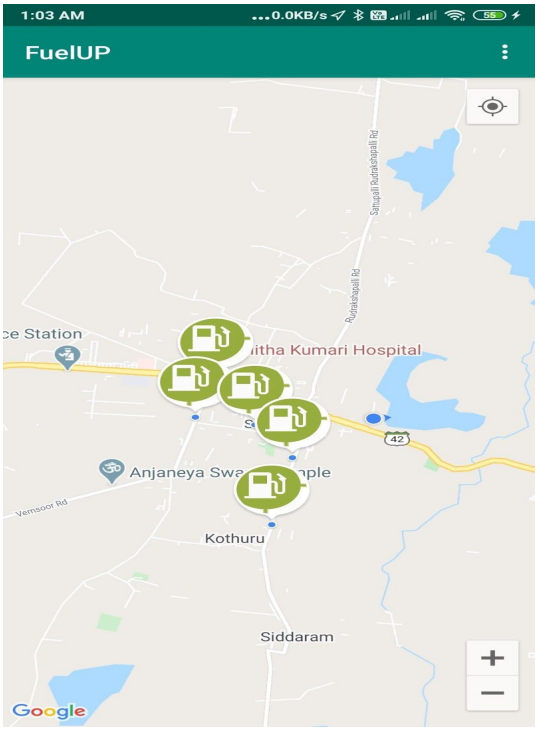

Fig 5.3 Fuel Stations Screen

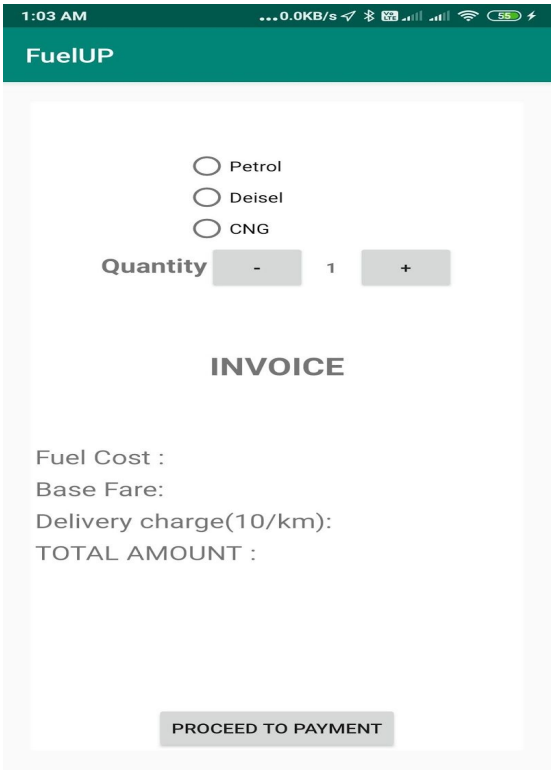

Fig 5.4 Fuel and Invoice Screen

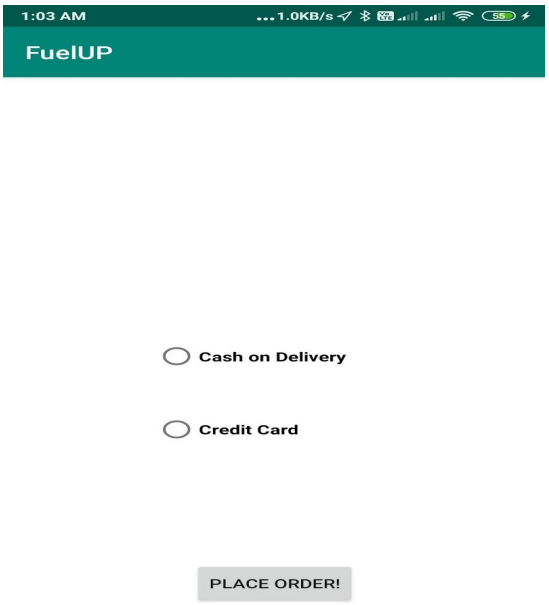

Fig 5.5 Payment Screen

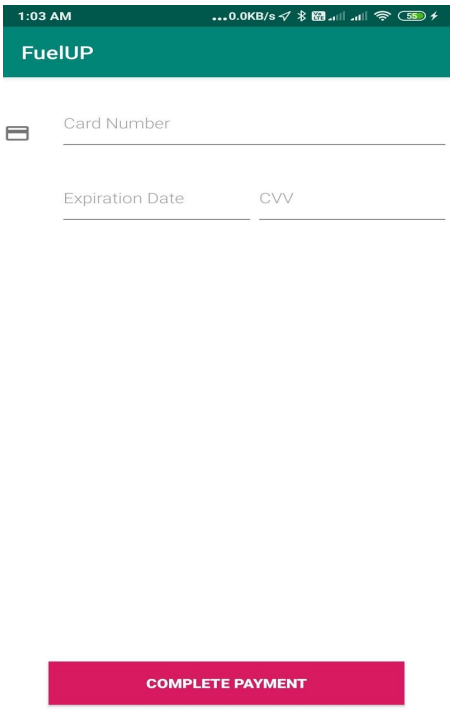

Fig 5.6 Card Payment Screen

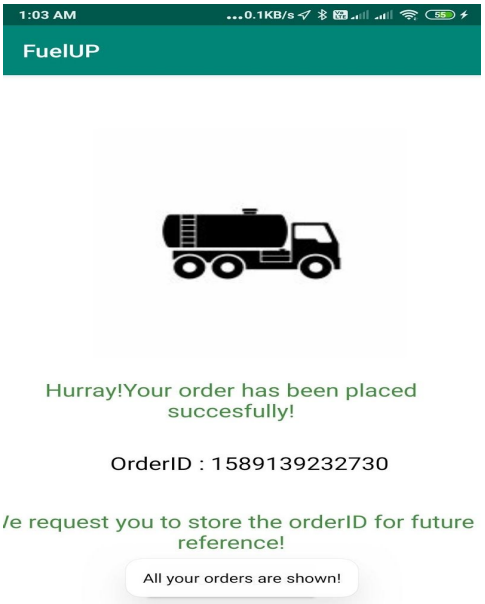

Fig 5.7 Order Placed Screen

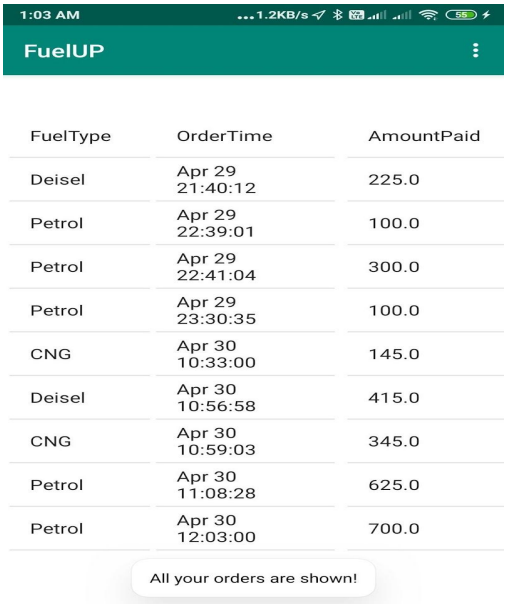

Fig 5.8 View Orders Screen

5.2 Benefits over existing System

1. Save Time

With the on-demand application, you can save time and can get the fuel in a few minutes after ordering it. With this mobile app development, no more waiting in long queues at the fuel station.

2. Save Environment

There is no fuel or little fuel spills due to loose carrying and storing the fuel at the site for future needs.

3. Emergencies Handling

Ran out of fuel in the middle of the road trip? Don’t worry, within a few taps on the mobile app the fuel will be delivered to you in a few minutes.

4. Favourable

Sooner you will have no need to wait in the long queue and refill our vehicle’s tank. You can get immediate fuel with the app. Also, you can book in advance for future requirements.

5. Quality & Cleanliness

Refilling the tank using the fuel delivery app is a cleaner procedure as it is double filtered and does not sit down in the petrol tank for weeks. It is regularly cleaned.

6.TESTING

6.1 Unit Testing

Testing of an individual software component or module is termed as Unit Testing. It is typically done by the programmer and not by testers, as it requires detailed knowledge of the internal program design and code. It may also require developing test driver modules or test harnesses.

Following tests are made so that the application runs

Registration:

- **Phone number** – is necessary, should match format and should not have duplicates in database.
- **Password** – is necessary, must not be visible to user and should not have duplicates in database.
- **Full name**- is necessary and should not have duplicates in database.
- **Email-** is necessary and must match the format.

Login:

- **Phone number and Password**- should match with that of in database else could not log in.

Order now:

A station must be selected from the available stations otherwise error is shown and also user must be logged in to place order.

View Order:

User must be logged in to view orders.

Invoice:

Bill must be generated with selected type of fuel and fare must be calculated with the distance between locations and quantity of the fuel selected.

Card Payment:

- **Card Number-** must be a 16 digit and valid number.
- **CVV**- must not be visible and be a 3 digit and valid number.
- **Expiry date**- must be after the present date.

Payment Successful-

A valid delivery ID must be generated and displayed to the user.

6.2 Functional Testing

This application is made used by a random user who has no idea of the implementation of code (black-box testing) and resulted as successful as he could use the application with ease and is user-friendly.

7. CONCLUSION

On demand fuel delivery services can be very useful for the customers just like so many other services like food delivery apps and taxi booking apps. Access to the internet and smartphones have been instrumental in making many on-demand services popular among people. It won't be a surprise to see more apps alike being launched and become popular in time to come. This application could be developed more including many more futures and could be used in real-time which benefits more.

7.1 Future Scope

Fuel delivery app holds a promising growth scope in the coming years. If certain points are taken care of while developing an app, it is likely to gain huge popularity and success in the coming years. With increasing responsibilities and shortage of time, the on-demand fuel delivery solution is foreseen to save you the hassle of running to the fuel station to refuel your vehicle.

By adding many more other features like advance booking, live tracking, increasing the number of fuel stations that could supply, categorizing type of fuel stations, developing a station end application to accept and reject orders, also a tracking application for the delivery person and using better User Interface could possibly made so that this app meet the present world requirements and be of great use as it has many advantages.

8.REFERENCES

[1] https://www.udemy.com/course/complete-android-n-developer-course/

[2] https://developer.android.com/docs

[3] https://stackoverflow.com/

9.LOGO

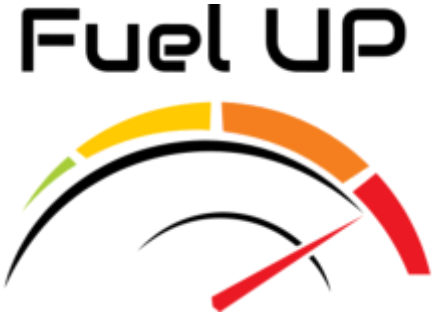

Fig 9.1 Logo