Spring 2018, CS5551:Advanced Software Engineering, Department of Computer Science & Electrical Engineering, University of Missouri-Kansas City

FIXITUP

Members:Duy Ho, Sireesha Keesara, Kartheek Katta, Rishitha Bobba

GitHub URL
ZenHub URL
Project Video URL

Table of Contents

PROJECT TITLE : FIXITUPP ANDROID APP 3
TEAM: 7 3
PROJECT DEPLOYMENT 3
PROJECT MANAGEMENT 11
Project management Report 11
Contribution (Design, Implementation, Testing) 11
Final Project evaluation 11
PROJECT DEMO – PRESENTATION SLIDES 13
PROJECT PROPOSAL 13
PROJECT GOAL: 13
MOTIVATION: 13
SIGNIFICANCE: 13
OBJECTIVES: 14
FIRST INCREMENT REPORT 14
Existing Services/ REST API 14
Detail Design of Features 15
Architecture Diagram: 16
Sequence Diagram: 16
Class Diagram: 18
Use case Diagram: 20
Testing: 20
Implementation: 20
Platform: 21
Tool: 21
Approach: 21
Screenshots: 21
Project Management 26
Implementation Status Report 26
Bibliography 27
SECOND INCREMENT REPORT 29
Existing Services/ REST API 29
Detail Design of Features 29
Implementation: 34
Platform: 34
Tool: 35
Deployment: 38
Project Management 45
THIRD INCREMENT REPORT 49
Existing Services/ REST API 49
Implementation: 49
Platform: 49

Tool: 49
Deployment: 56
Project Management 71
Bibliography 72

PROJECT TITLE : FIXITUPP ANDROID APP

TEAM: 7

TEAM MEMBERS:

1.DUY HO

2.SIREESHA KEESARA

3.RISHITHA BOBBA

4. KARTHEEK KATTA

PROJECT DEPLOYMENT

FIXITUP User Manual

Table of Contents:

1. Introduction
2. How to use the system with screen shots
    2.1. App Usage
        2.1.1. Registration Page
        2.1.2. Login Page
        2.1.3. Customer Home Page
        2.1.4. Technician Home Page
        2.1.5. Technician Requesting Page for Customers
        2.1.6. Maps Page
        2.1.7. Rating Page
3. Bugs and Deficiencies

1.Introduction:

An android application to help the user if customer to find the desired technician nearby using maps and requesting services from them. And to help the technician to respond to the requests from customer.

2. App Usage:

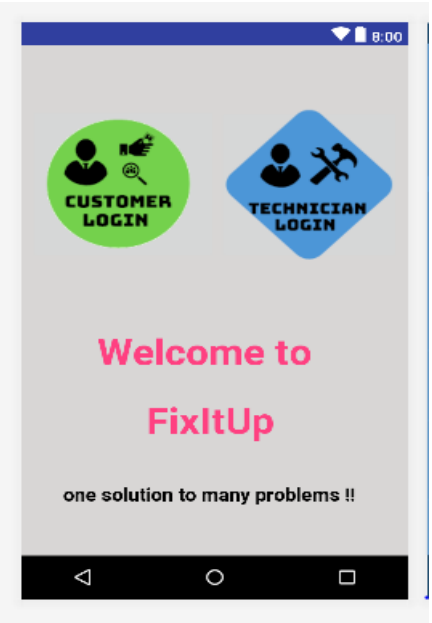
This app allows customer to do the following

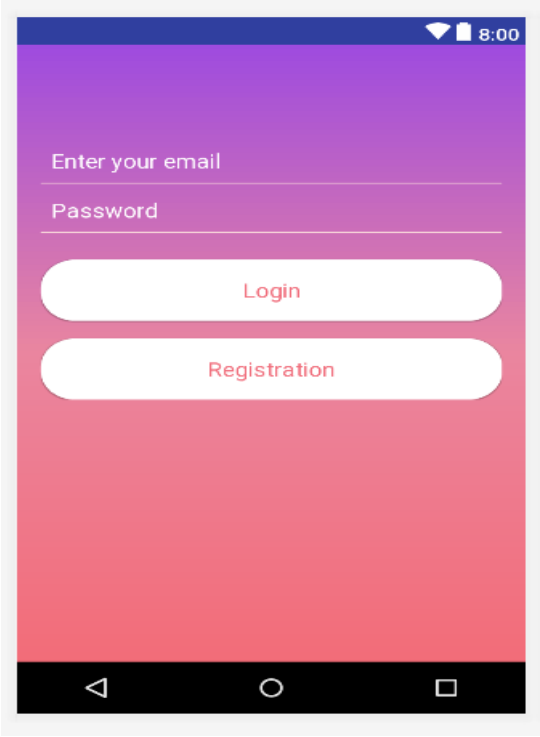
- Login via registered email credentials.
- Search for the technicians with their specialization.
- To choose from a list of technicians online of desired specialization.
- Check the distance of chosen technician using maps and if found nearby then requesting services from them.
- Whenever a request is sent to technician if the technician accepts the request then a session starts for the particular technician and customer.
- After the session ends customer can rate technician and vice versa.

Screen Shots:

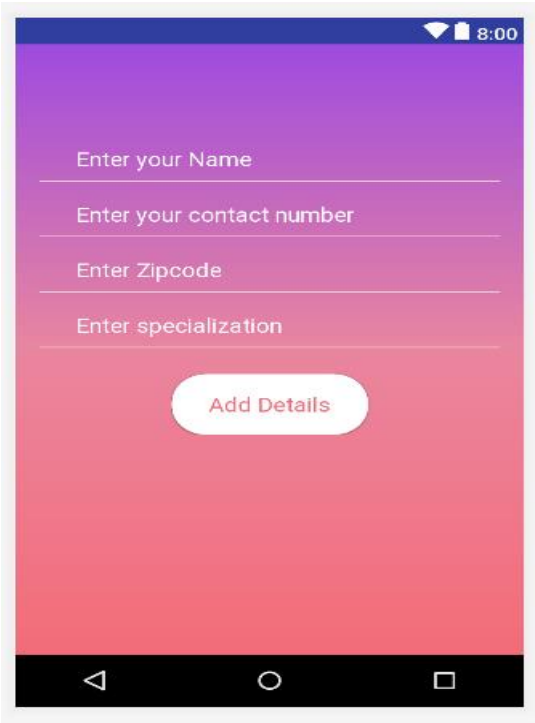
Application Home Page

Login Page for Customer and Technician

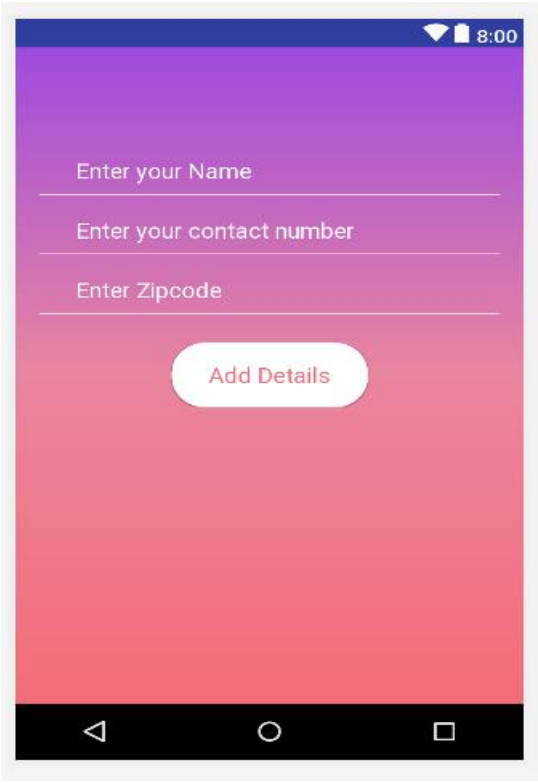




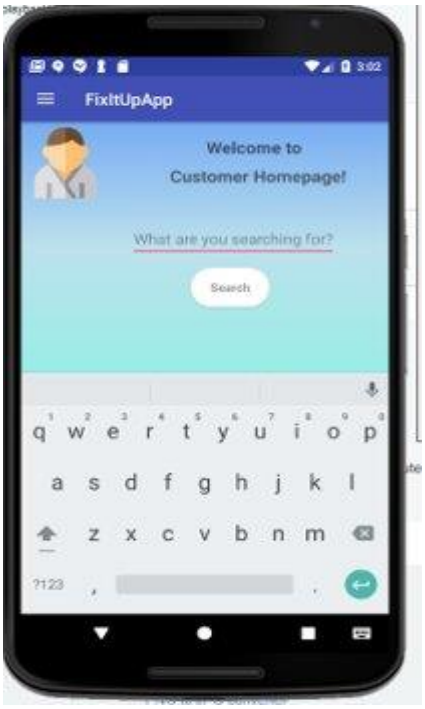
Registration Page for Technician

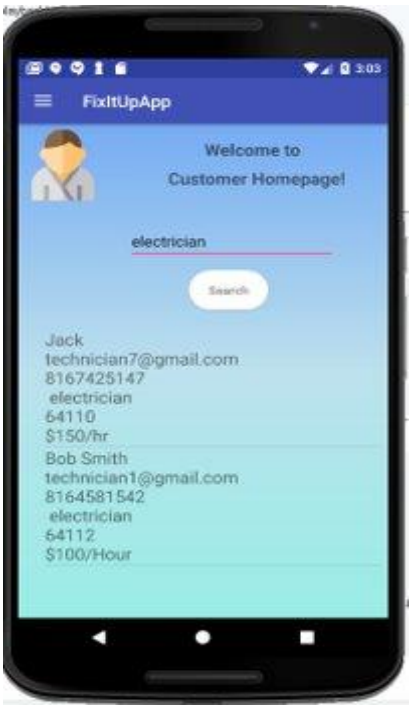


Registration Page for Customer

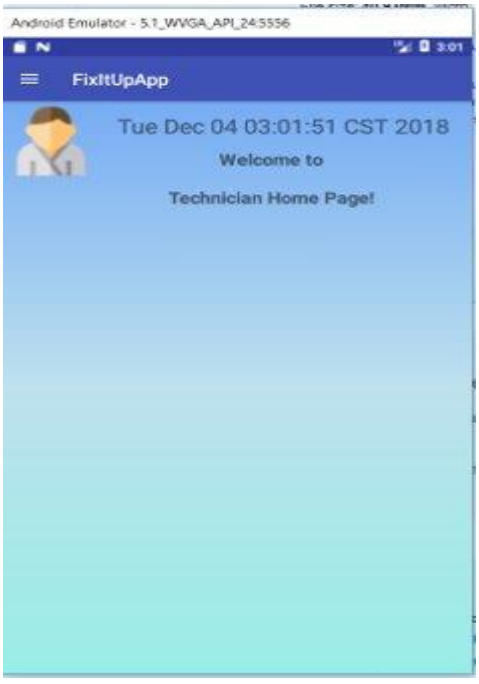


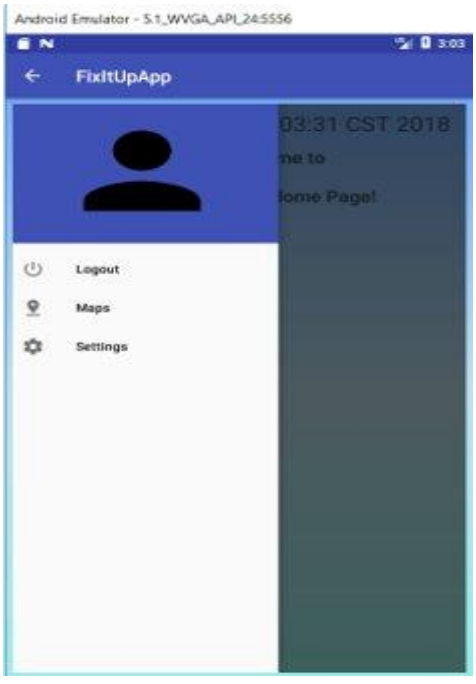
Customer Home Page



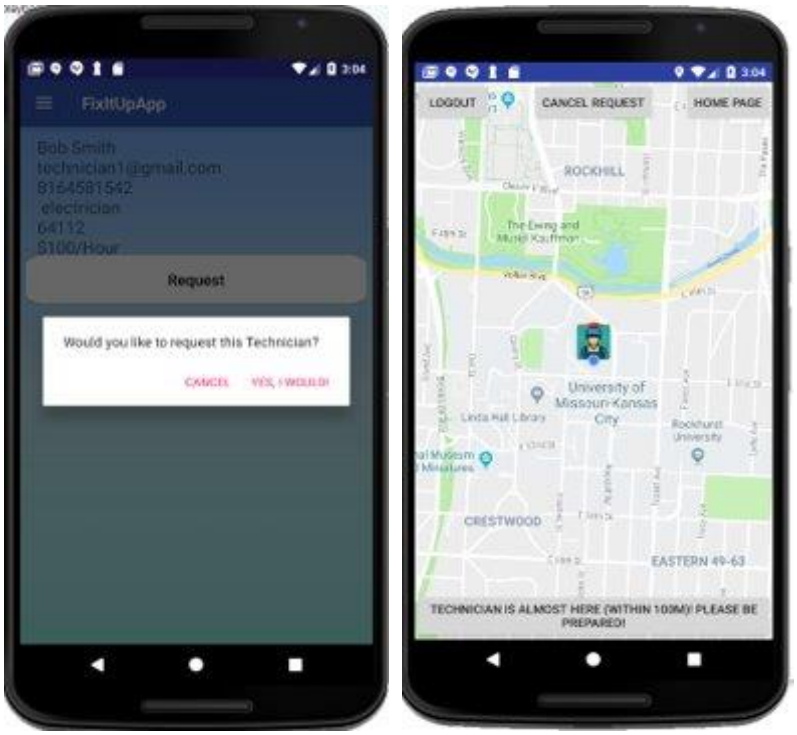


Technician Home Page



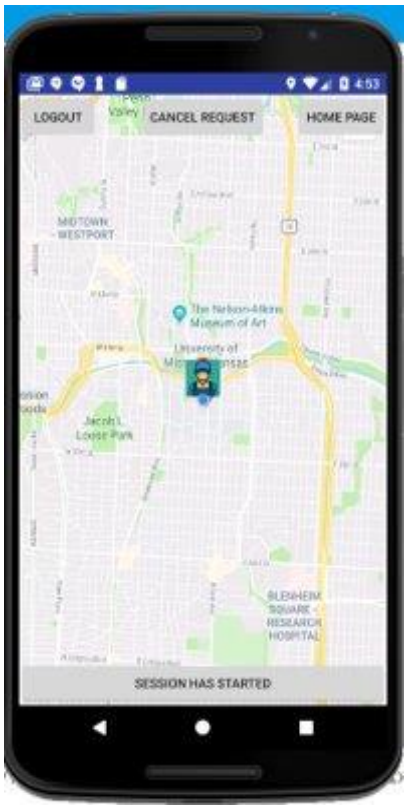


Customer Requesting Technician

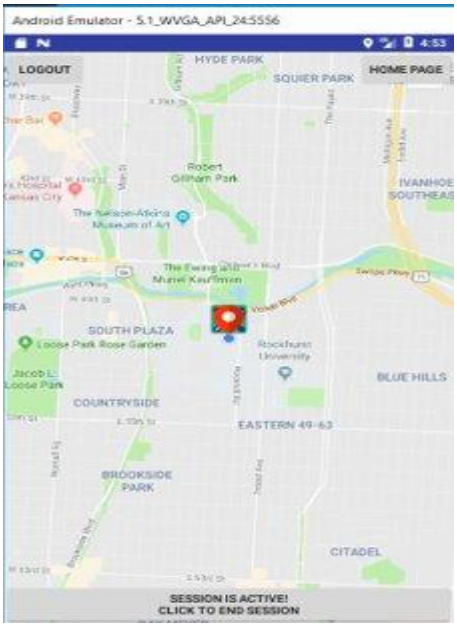


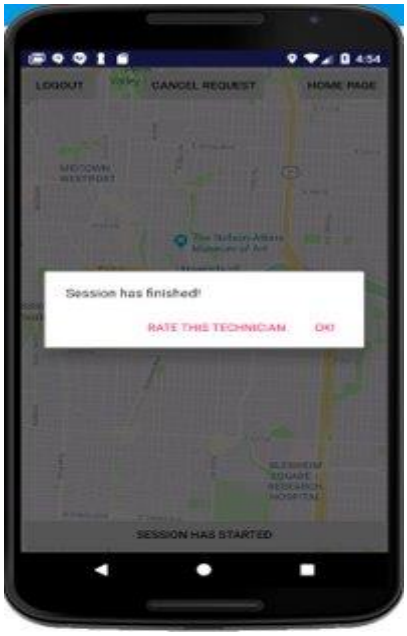
Upon Clicking Yes, I would customer would be able to see the technician location and a request is sent and if technician accepts it and reaches your then session starts.

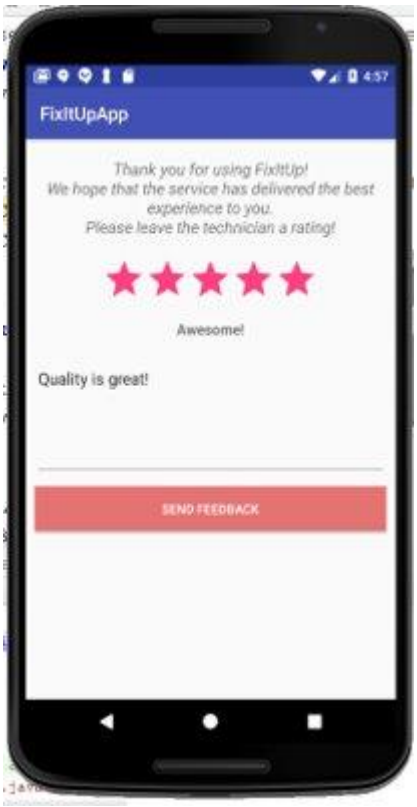
Technician Accepting Request



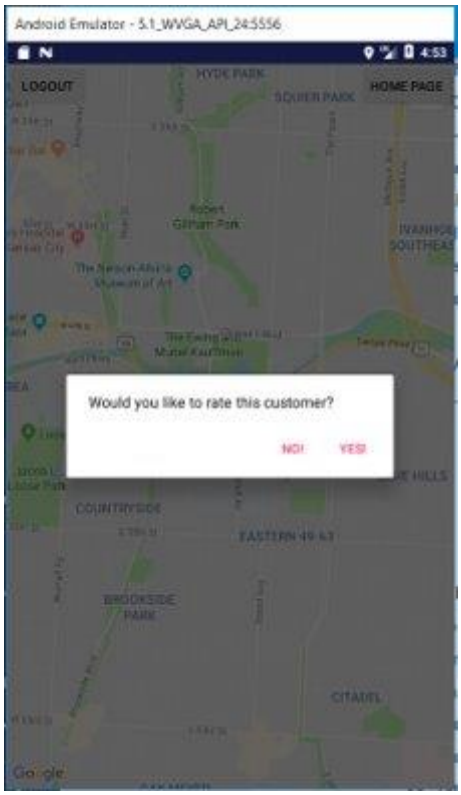
Customer ending the session

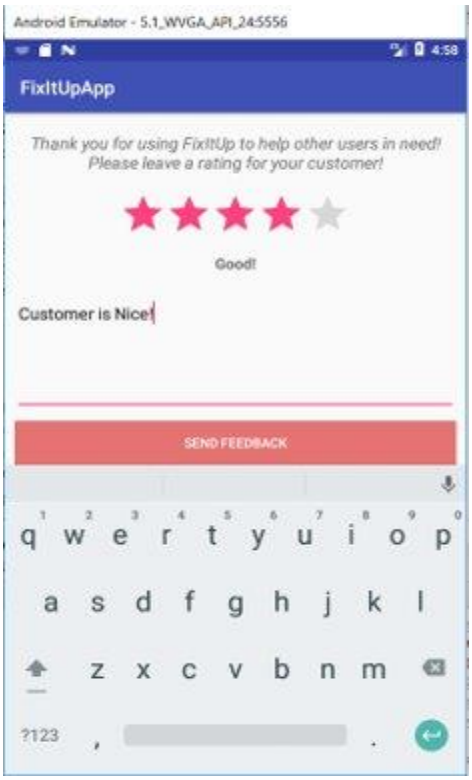






Technician Rating Customer





3. Bugs and Deficiencies

- If many customers request a single technician the list of customers is not queued at the technician home page.
- Payment gateway is not implemented

PROJECT MANAGEMENT
Project Management Report

Final Project Evaluation

1. Requirement Satisfaction
    - I would say that our project falls within at least 85% of the requirement specifications we first proposed. Surely, there are some more features and items we would like to add as a future scope, but most of them can be derived from the core components and structure we have built so far such as Firebase Database, Location Tracking, and Ordering.
2. Design Process
    - Design Process took just about as much time as coding, even more at the beginning of the project when we did not know which database to implement, and what a specific platform such as Android can bring forth for us. Afterwards, we mostly spent time designing the architecture, control flow, and binding of each role (technician/map/customer/database/map) within the app.
3. Agile Process's Helpfulness
    - This software development methodology did not work as well for us. To give our software development process an honest score, I would give a score of 3/10 because the team members did not provide a timely submission/feedback along with sufficient amount of work/effort into the project as we first expected. I, as a leader, have tried to speed up the project velocity by assigning features to each member but it seems to us that most of our team members are not quite concentrating on the real work and not truly dedicating enough time to developing the project, therefore not maximizing the outcome of an Agile process.
4. Schedule
    - Most of the milestones marked in the Google Sheets are submitted on time, without any late submission. However, all of the proposed/planned objectives were never reached throughout the increments. Instead, most of them are 70-80% completed since effort/time/unexpected complexity/debugging issues came into play with the progress.
5. Management Structure
    - The real management structure of our team is leader/members. There is the leader, who does the version control/resolution and major integration of the code whereas the members will contribute

their code to their Github which will be solved, merged, and integrated into the master branch by the leader afterwards. The members will be responsible for their own testing and functionality.

- Communication tools: initially, there was a suggestion of using Slack for universal communication channel for work/project which is ideal for the industry as well. However, since the majority of the members are familiar with Whatsapp, the team communication mostly took place in Whatsapp.

6. Previous Project
    - Although some of the team members have had experience in developing Android applications before, this is our first time developing an Android application that incorporates API along with online and real-time features such as Firebase Database, Geofire, Google Maps, and matchmaking of multiple devices.
7. Problems/Conflicts
    - Since the communication and management process mentioned above were not as effective as expected, some team members found it really exhausting trying to keep every objective/goal/milestone in check since the rest of the team was not active in participating and contributing to the overall progress of the project. There was a repeated incident where a member was slacking off of his/her assigned tasks which slowed down/held back the project's integration and richness.
8. Suggestion
    - Team members would speak to each other about complaints and conflicts more directly without having to fear that it would disrupt team collaboration.
    - Tas could join and handle the case more effectively
    - More established guidelines and warnings set ahead for inactiveness within a team
9. Real world aspect
    - If this was carried out in a real world/industry, I would suggest having more scalable and secure solution instead of a public database and a project that works on a couple of devices at the same time. The app should be able to work seamlessly across thousands of concurrent devices and handling all request/responses simultaneously. It also should be able to be more UI/UX friendly as the UI layout is not as intuitive as it could be
10. Recommendation for future

- More help to students and resources.
- Instructions for working in team
- More time to present their project instead of 10 minutes each
- More time for each video presentation

PROJECT DEMO – PRESENTATION SLIDES

Demo Video: https://www.youtube.com/watch?v=3nHzO0CH5kM&t=2s

Presentation Slides URL:
https://docs.google.com/presentation/d/1ia2V0wyOKPKGXLneWOR7yQvelncJoM1O7mV9ooEra6Y/edit?usp=sharing

PROJECT PROPOSAL

PROJECT GOAL:

To create an android app which acts as a platform to connect users (who needs help with household, auto, or mechanical services) with the actual technicians who are able to perform and provide those services.

MOTIVATION:

Technology has transformed our lives tremendously. When we are in need to fix a problem immediately, browsing a lot of websites for technician's information and point of contact is a time-consuming process. Hence, it is advised that we have more centralized method of gathering information about these technicians/service providers in order to provide quicker and more convenient satisfaction to customers, which drives the need for developing an app which provides solution to many categories of problems.

SIGNIFICANCE:

FixItUpp provides a single platform where we can fix our problems online. • Our app is unique because it lists explicitlythe availability of the technicians in terms of their current workload. For instance, when a user needs a plumber immediately and if the plumber is currently busy, the app instantly acknowledges it and notifies the user so that the user can opt out for another plumber. • FixItUpp also

provides a useful live chat option within the app where the user can communicate with the technician about the problem and other related matter.

OBJECTIVES:

The main objective of "FixItUpp" is to develop an android application where the user enters his/her category of issues(either in electricity, vehicle, pipe system, household appliances) and the comprehensive list of technicians and their expertise will be displayed afterwards together with all the appropriate information. In addition, the user can also identify the location of each technician on Google Maps. We have an option called "availability" where the user can see the approximate waiting time for a particular technician. Moreover, the user will be able to rate and review each technician. As far as the problem's details are concerned, the user will be able to contact the technician directly to ask whether that technician is able to repair it.

SYSTEM FEATURES:

Searching and selecting: Users can search for what they need help with and who they wish to contact first • Payment:Users can choose to pay provider up front if there is already a price. • Reviews and ratings: after each service session, the user will be able to rate the technician based on their satisfaction level. • Maps: if user allows current location tracking, he/she will be able to view nearby technicians on Google Maps. Otherwise, the user can enter a zip code instead. • Text: basic communication to the technicians. This does not guarantee reply since the technician may be working elsewhere. • Waiting time: Shows the current waiting time of a user for a particular service. Should be updated in real time. • Contact details: a webpage or info page for the technician, if available.

FIRST INCREMENT REPORT

Existing Services/ REST API

Up to the end of this increment, we are not implementing any REST API (APIs that use get and parameters to retrieve information such as Weather Underground and Nutritionix). However, we are using:

Google Firebase: as cloud database/storage, and cloud authentication using User Email/ Password. This eliminates the constraints of validating against the database and Firebase has a built-in verification method for itself using FireBaseAuth. Google Maps for Android SDK: acts to show users where they are currently located and other technicians' current location as well. This will be useful as it gives the customers/technicians a better picture of waiting time/remaining distance from one another.

Up to the end of this increment, we have done validations for Login and Registration activities for customers/technicians. And done some UI modifications to the application. And whenever a customer logs in and it takes to the customer home page where it displays the list of technicians that are available database. GeoFire: Updates real-time location of customers/technicians to Firebase Database and queries the storage of locations based on distance range and requested location of customer.

Detail Design of Features

As a result, Increment 1 does not promise to show a lot of tangible features and output due to lack of team communication at initial stages and a need for more efficient workflow. Also, we spent a lot of time discussing design, version control, workflow among team members for future long-term development.

Our Mockup Design would look similar to Uber's platform in that we have sign for Technicians and Customers as well.

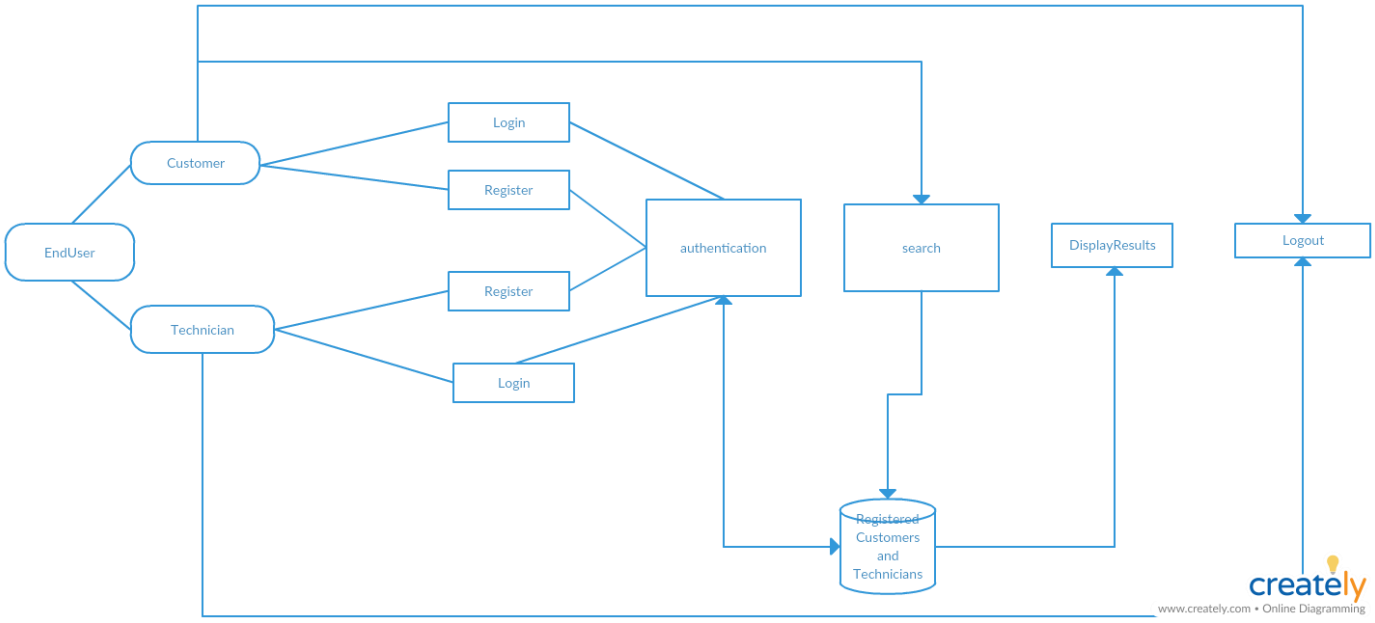
These two targeted users' paths will then diverge significantly in terms of UI design, content, and features:

Customers will be able to search for what they are currently needing help with, enter their current location (or other locations that they want using zip code), and opt for the technician that they see fit. There is an option for payment if the problem (service) is clearly identified and the technician has given a specific quote for that service; if not, they can always pay at the end of their session with the technician (however, this will not be our liability anymore). At the end of each session with the technicians, the customers will also be able to rate the service quality which would then be taken into account for the technician's overall rating as well.

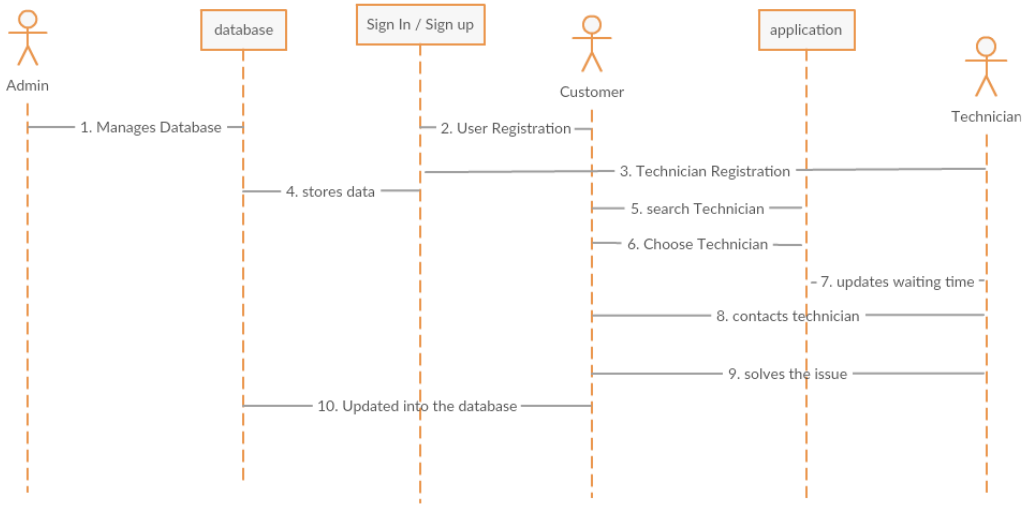
Technicians, on the other hand, would see what services have been requested (in case a technician is experienced in multiple services/ has multiple fields of expertise), what is the current queue is a particular service, and finally set his/her current waiting time for the customers to see. For the waiting time, we may ask the technician to enter a default waiting time per person so that the system can be show the customers its estimated waiting time in case the technician does not enter his own time estimation. Technicians may turn on and off his current location services in order for the customers to have a better idea of his current progress (ON is recommended for technicians).

Moreover, to have a better idea, here is the four different views of the FixItUp Application:

Architecture Diagram:

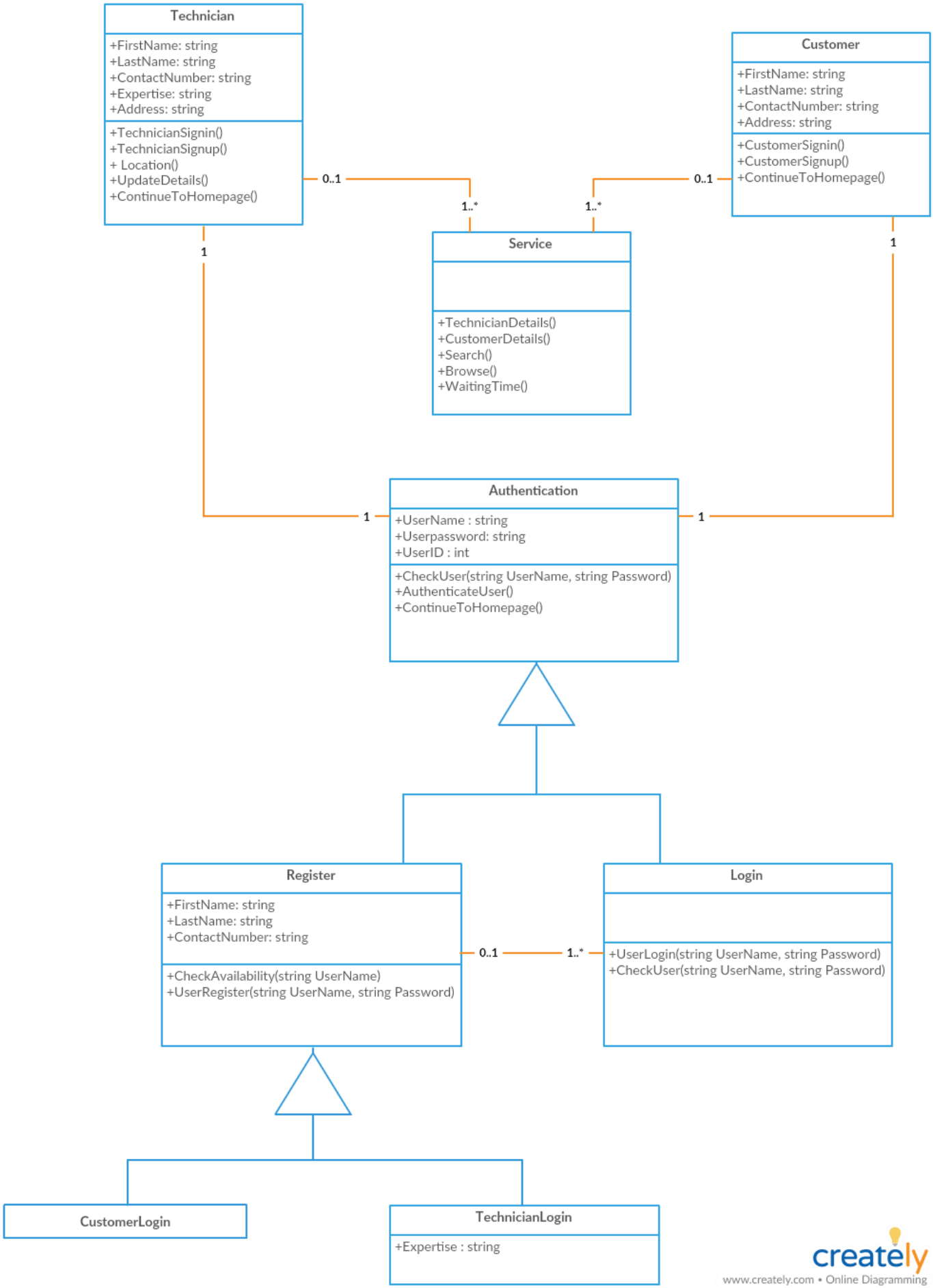


Sequence Diagram:



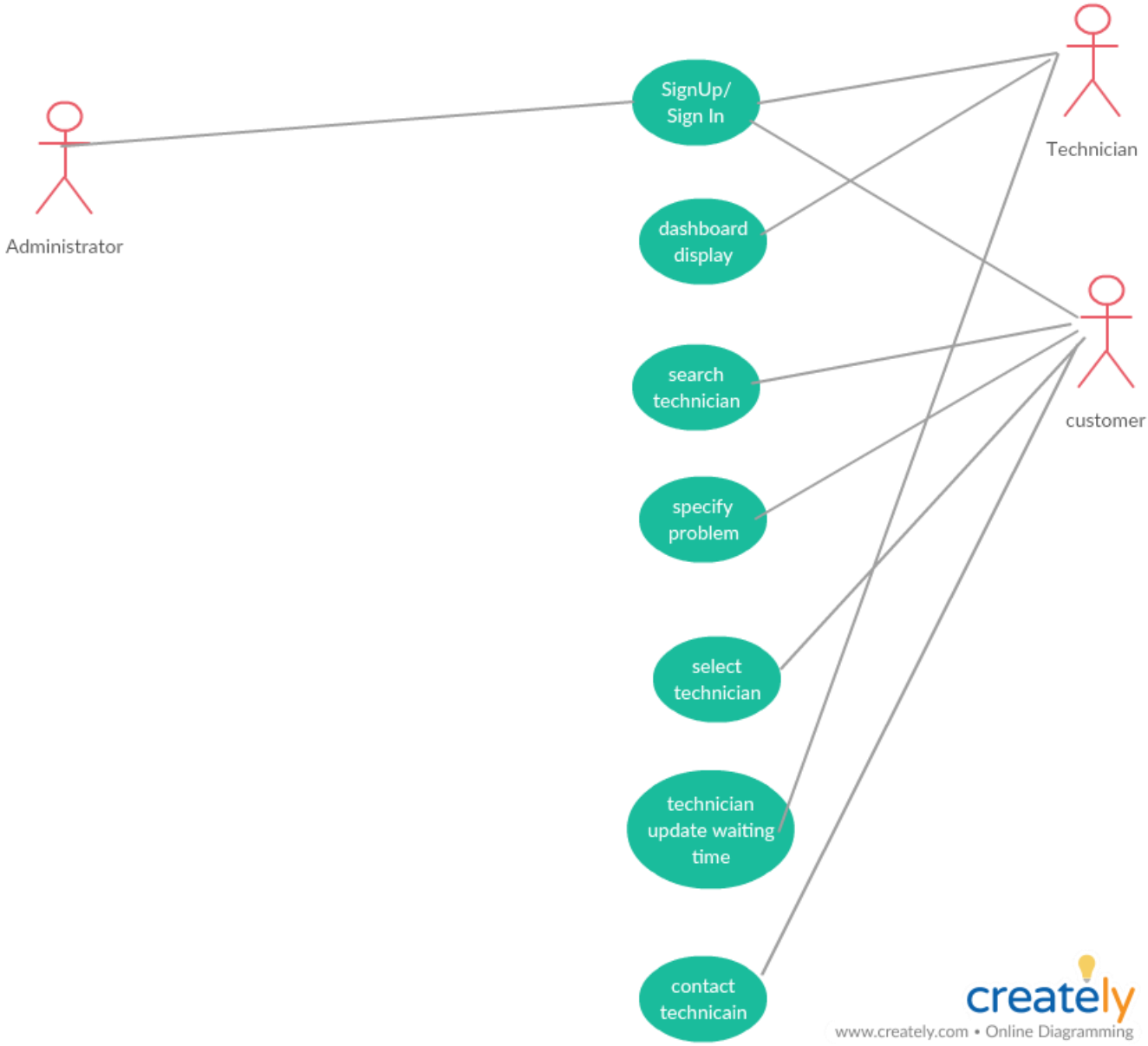
creately
www.creately.com • Online Diagramming

Class Diagram:



creately
www.creately.com • Online Diagramming

Use case Diagram:



Testing:

Unit Testing is not implemented for this increment due to time constraint and the unfamiliarity with the JUnit testing tool. Nonetheless, the team members did a lot of debugging to make sure that the core functions are executing properly without side effects.

Implementation:

The form of Application we are building is native application.

Platform:
Android

Tool:
Android Studio version 3.1.x

Approach:
For this increment, we have created 3 main pages that encompasses 2 main features: authentication and registration.

For both purposes, we are using the provided service from Google Firebase to do the identity validation and storage. We include the firebase implementation in the gradle in order for Firebase source code and modules to work in Android Studio. Afterwards, we created two buttons, each for customers and technicians. If the user is a customer, then he/she would click on that and it will direct to another customer-related activity such as customerlogin activity. The same process applies to the technicians.

At the end of this increment, we have accomplished designing the paths for customers and technicians by not mixing them together and use identity validation. Instead, we let them diverge to their own activities. The same strategy is used in the Firebase database by creating 2 separate columns such as "Technicians" and "Customers" under "Users" column to explicitly distinguish between customers and technicians.

Screenshots:
This is the Firebase authentication in the TechnicianLoginActivity. The authentication is executed directly through the FirebaseAuth and FirebaseAuthListener. When it returns user's valid info (or invalid), we will direct the user to the right activity

```
public class TechnicianLoginActivity extends AppCompatActivity {
    private EditText mEmail, mPassword;
    private Button mLogin, mRegistration;

    private FirebaseAuth mAuth;
    private FirebaseAuth.AuthStateListener firebaseAuthListener;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_technician_login);

        mAuth = FirebaseAuth.getInstance();
        firebaseAuthListener = new FirebaseAuth.AuthStateListener() {
            @Override
            public void onAuthStateChanged(@NonNull FirebaseAuth firebaseAuth) {
                FirebaseUser user = FirebaseAuth.getInstance().getCurrentUser(); //get the information of the current user
                if (user != null){
                    //if not null, move to another activity, to be created later.
                    //Remember the current context!
                    Intent intent = new Intent( packageContext: TechnicianLoginActivity.this, TechnicianRegisterActivity.class);
                    startActivity(intent);
                    finish();
                    return;
                }
```

This will be triggered when user clicks the registration button. Afterwards, the user will be able to enter the credentials which will then be validated against the Firebase authentication directory. If not exists, it will add the user id to the database.

```
//Initialize variables from xml UI layout.
mEmail = (EditText) findViewById(R.id.email);
mPassword = (EditText) findViewById(R.id.password);
mLogin = (Button) findViewById(R.id.login);
mRegistration = (Button) findViewById(R.id.registration);

mRegistration.setOnClickListener(new View.OnClickListener(){

    @Override
    public void onClick(View v) {
        final String email = mEmail.getText().toString();
        final String password = mPassword.getText().toString();
        mAuth.createUserWithEmailAndPassword(email, password).addOnCompleteListener( activity: TechnicianLoginActivity.this, ne
            @Override
            public void onComplete(@NonNull Task<AuthResult> task) {
                //If user has already signed up, therefore the createUserWithEmailandPassword would fail.
                if (!task.isSuccessful()){
                    Toast.makeText( context: TechnicianLoginActivity.this, text: "Registration Failed!", Toast.LENGTH_SHORT).sh
                }
                //If the user email cannot be found in the database, reference the database and add variables to it.
                else {
                    String user_id = mAuth.getCurrentUser().getUid(); //id assigned to Technician at moment of sign-up
                    //this database reference is pointing to the technicians
                    DatabaseReference current_user_db = FirebaseDatabase.getInstance().getReference().child("Users").child
                    current_user_db.setValue(true);
```

The same logic happens in Login mechanism. However, for Login, we don't need to do anything if login is successful because FirebaseAuthListener will take care of valid info/state change.

```
mLogin.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        final String email = mEmail.getText().toString();
        final String password = mPassword.getText().toString();
        mAuth.signInWithEmailAndPassword(email, password).addOnCompleteListener( activity: TechnicianLoginActivity.this, new OnC
            @Override
            public void onComplete(@NonNull Task<AuthResult> task) {
                if (!task.isSuccessful()){
                    Toast.makeText( context: TechnicianLoginActivity.this,  text: "Authentication Failed!", Toast.LENGTH_SHORT).s
                }
            }
        });
    }
});
```

This is the landing page of the app where the user will be able to identify himself/herself as a customer or a technician.

Upon clicking the buttons, they will be directed to the right path/activity/login pages.

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    mTechnician = (Button) findViewById(R.id.technician);
    mCustomer = (Button) findViewById(R.id.customer);

    //Set behaviors for Customer and Technician buttons to redirect to its proper corresponding activity.
    mCustomer.setOnClickListener((v) -> {
            Intent intent = new Intent( packageContext: MainActivity.this, CustomerLoginActivity.class);
            startActivity(intent);
            finish();
            return;
    });
    mTechnician.setOnClickListener((v) -> {
            Intent intent = new Intent( packageContext: MainActivity.this, TechnicianLoginActivity.class);
            startActivity(intent);
            finish();
            return;
    });

}
```

To simplify the process of pushing and adding information to Firebase Database for storage, we packaged our users' info in a class which contains various attributes/fields so that the whole object can be delivered.

```java
public class TechnicianDetails {

    String name;
    String contact;
    String Zipcode;
    String type;
public TechnicianDetails(){

}
    public TechnicianDetails(String name, String contact, String zipcode, String type) {
        this.name = name;
        this.contact = contact;
        Zipcode = zipcode;
        this.type = type;
    }

    public String getName() { return name; }

    public String getContact() { return contact; }

    public String getZipcode() { return Zipcode; }

    public String getType() { return type; }
}
```

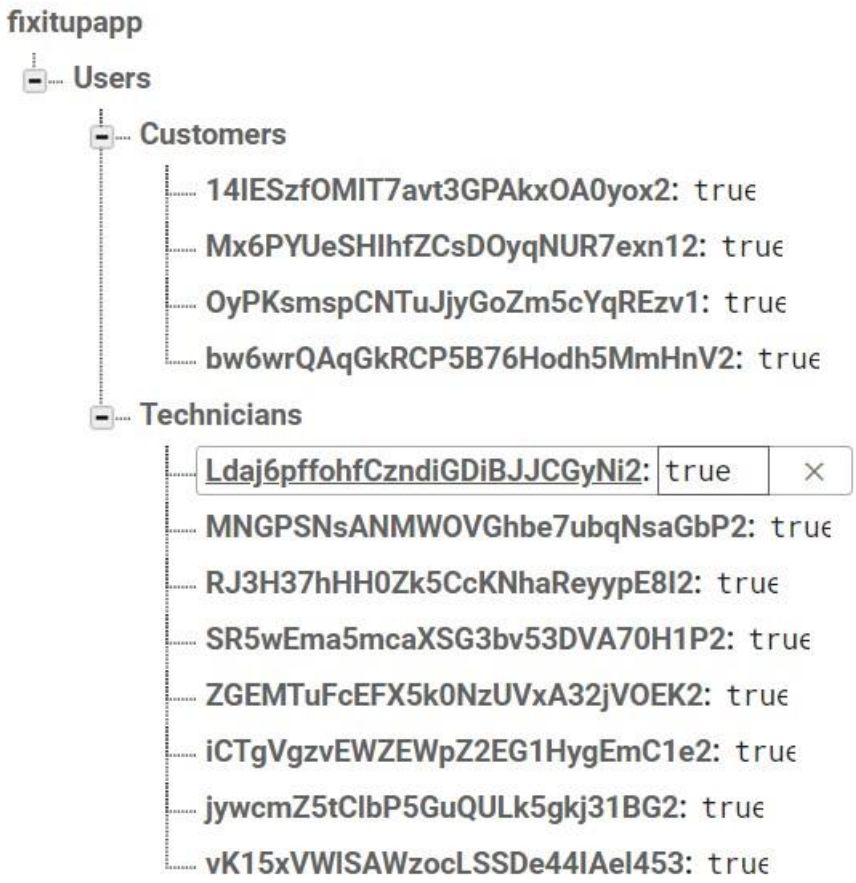
To further continue the logic of the previous screenshot, this activity will be used to process the input from user, package it into a TechnicianDetails object, and pass it to the database. As of now, the ID is not consistent because the ID that the user initially logins is stored in a different key than the ID that is packaged here. It should be the same. This will be fixed in Increment 2

```java
public class TechnicianRegisterActivity extends AppCompatActivity {
EditText name,contact, zip, specialization;
Button btn;
DatabaseReference dbr;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_technician_register);
        btn = (Button) findViewById(R.id.detailButton);
        dbr= FirebaseDatabase.getInstance().getReference().child("Users").child("Technicians");
        name=(EditText)findViewById(R.id.editTextName);
        contact =(EditText)findViewById(R.id.editTextConatact);
        zip=(EditText)findViewById(R.id.editTextZip);
        specialization=(EditText)findViewById(R.id.editTextType);
        btn.setOnClickListener((v) → { addTechnician(); });
    }
    public void addTechnician(){

        String techname = name.getText().toString().trim();
        String mobile = contact.getText().toString().trim();
        String zipcode = zip.getText().toString().trim();
        String type = specialization.getText().toString().trim();
        if (!TextUtils.isEmpty(techname)&& !TextUtils.isEmpty(mobile)&& !TextUtils.isEmpty(zipcode)&& !TextUti
            String id= dbr.push().getKey();
```

This is how the User IDs are stored in the FireBase real-time database. Whenever there is a registration, it will automatically update with the user id and their relative detailed information package.



Deployment:

Our project is up-to-date with our GitHub as far as the most stable release is concerned. Other ongoing development and features need to be further tested, analyzed, and discussed. They are not yet published and are intended to be released in the upcoming increments.

GitHub Repository URL

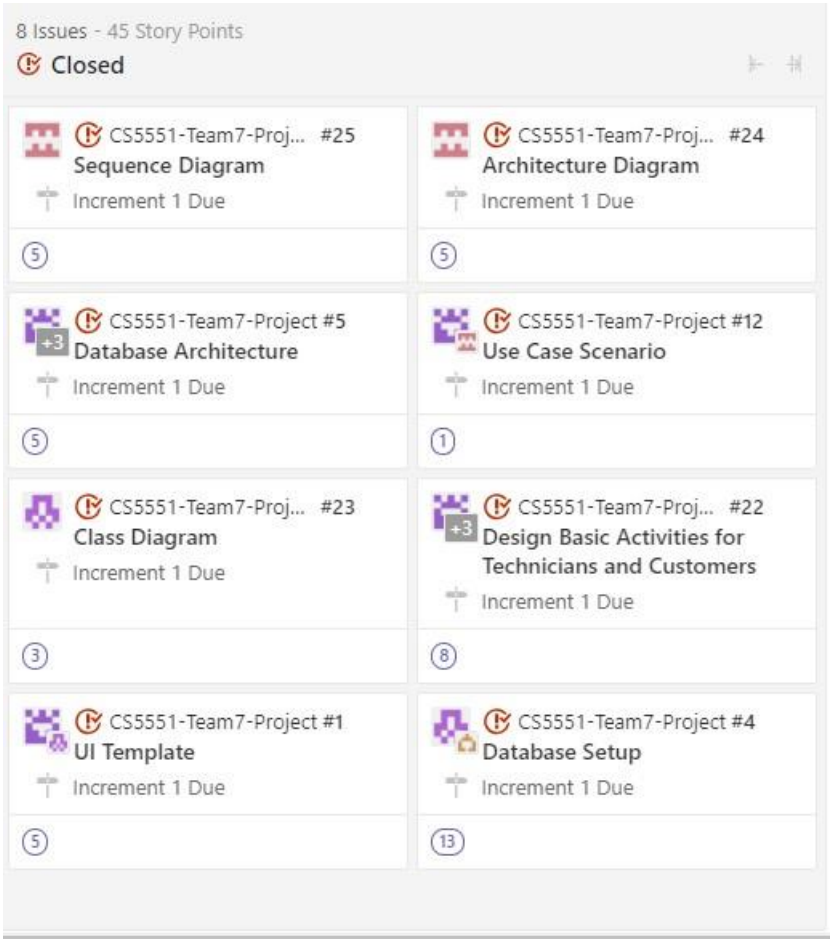
Wiki Link

Project Management

Implementation Status Report

Work Completed

Description: There are 8 main issues that we have completed for this increment:



Responsiblity:

- Database Architecture, Design, and Setup: Duy and Sireesha: 5
- hours Code Design and Implementation: Duy, Sireesha, Karthik,
  - Rishitha - 15 hours
- Diagrams:

    Class and Use case Diagrams: Karthik - 5 hours

    Sequence and Architecture Diagrams: Rishita - 5

hours Wiki, Documentation, and Report: Duy - 4 hours

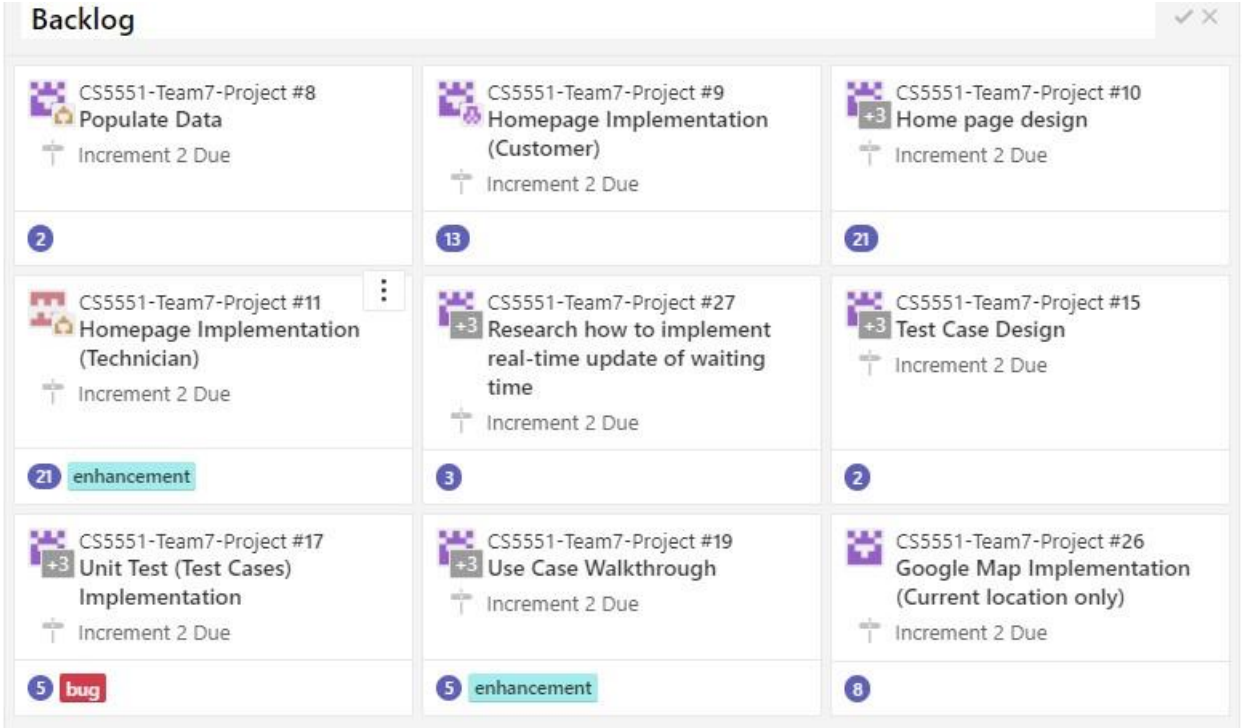
Work to be Completed :

For increment 2: We will expand our scope to:

- Populate and retrieve data in Firebase database
- Provide homepages for both customer and technicians:
  - For Customer: search bar, technician information, rating, and request a service
  - For Technician: home page, update information, update status, and current request queue.

Provide a map of current location for both customers and technicians: Customers will be able to see their current location, Technicians will be able to see their location.

**NOTE: For increment 2: Customers and Technicians may not be able to see each other's location yet. **



Bibliography

Firebase Tutorial

Android Tutorial

Google Maps Tutorial

Android Resources

SECOND INCREMENT REPORT

Existing Services/ REST API

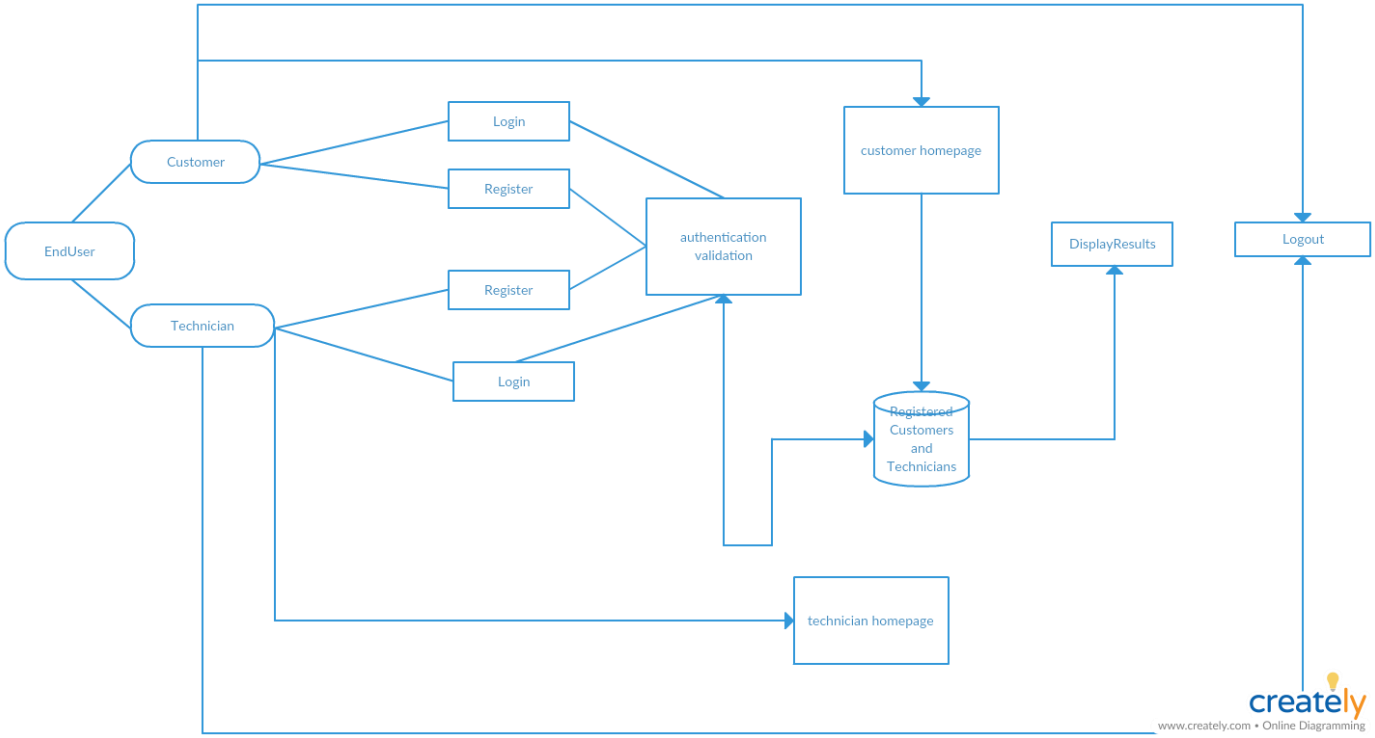
Up to the end of this increment, we are not implementing any REST API (APIs that use get and parameters to retrieve information such as Weather Underground and Nutritionix). However, we are using:

- Google Firebase: as cloud database/storage, and cloud authentication using User Email/ Password. This eliminates the constraints of validating against the database and Firebase has a built-in verification method for itself using FireBaseAuth
- Google Maps for Android SDK: acts to show users where they are currently located and other technicians' current location as well. This will be useful as it gives the customers/technicians a better picture of waiting time/remaining distance from one another.
- GeoFire: Updates real-time location of customers/technicians to Firebase Database and queries the storage of locations based on distance range and requested location of customer.

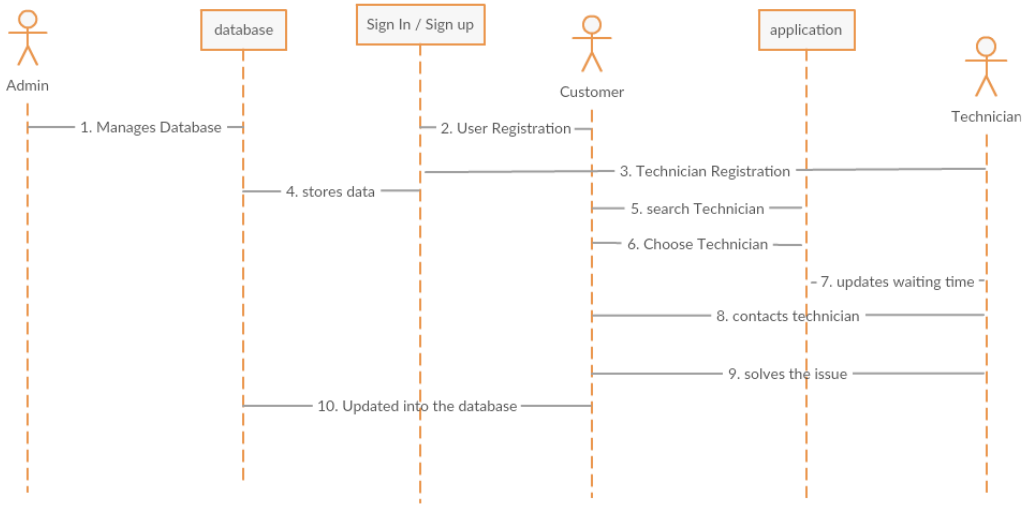
Detail Design of Features

Here is the four different views of the FixItUp Application:

Architecture Diagram:

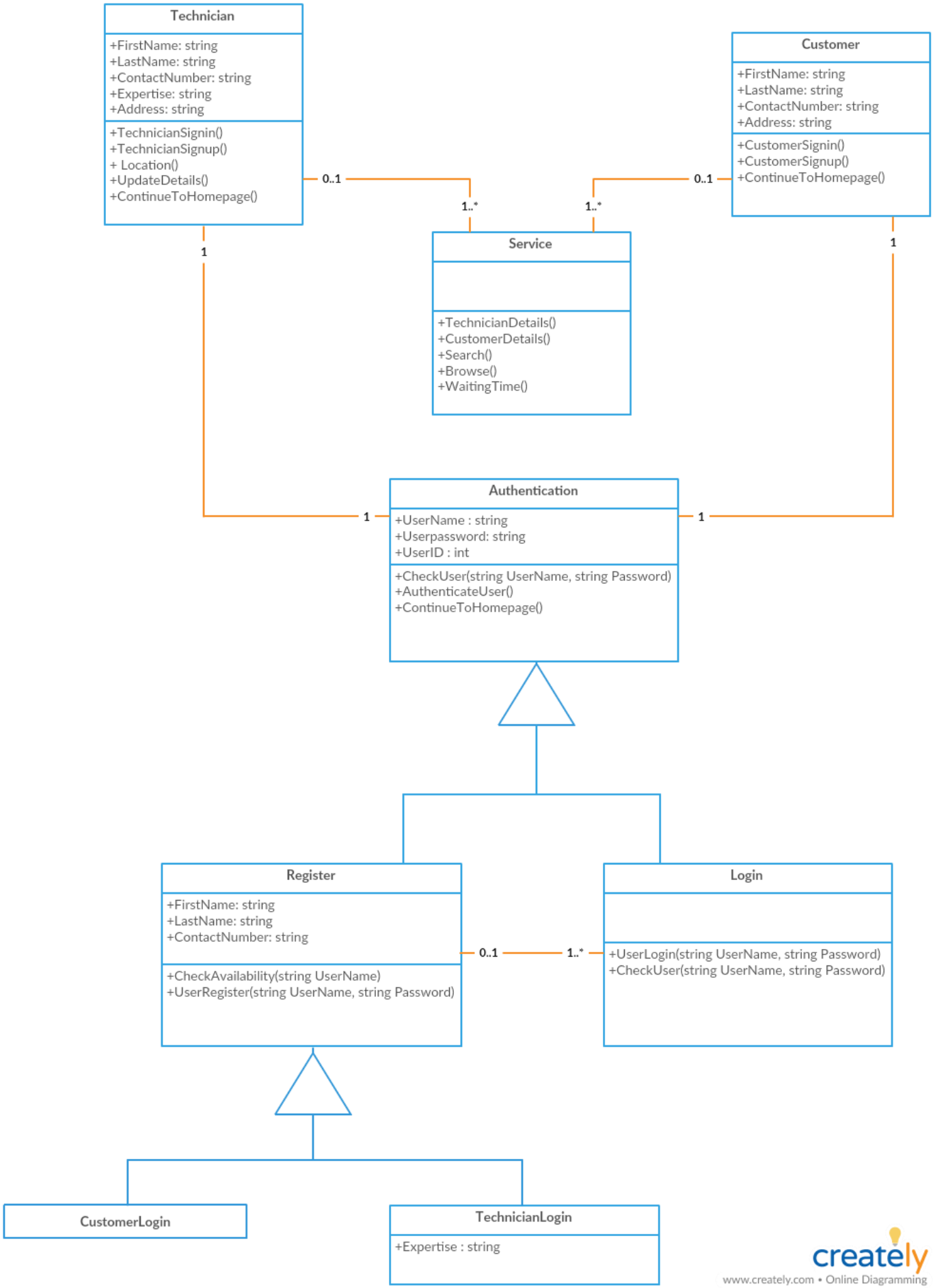


Sequence Diagram:



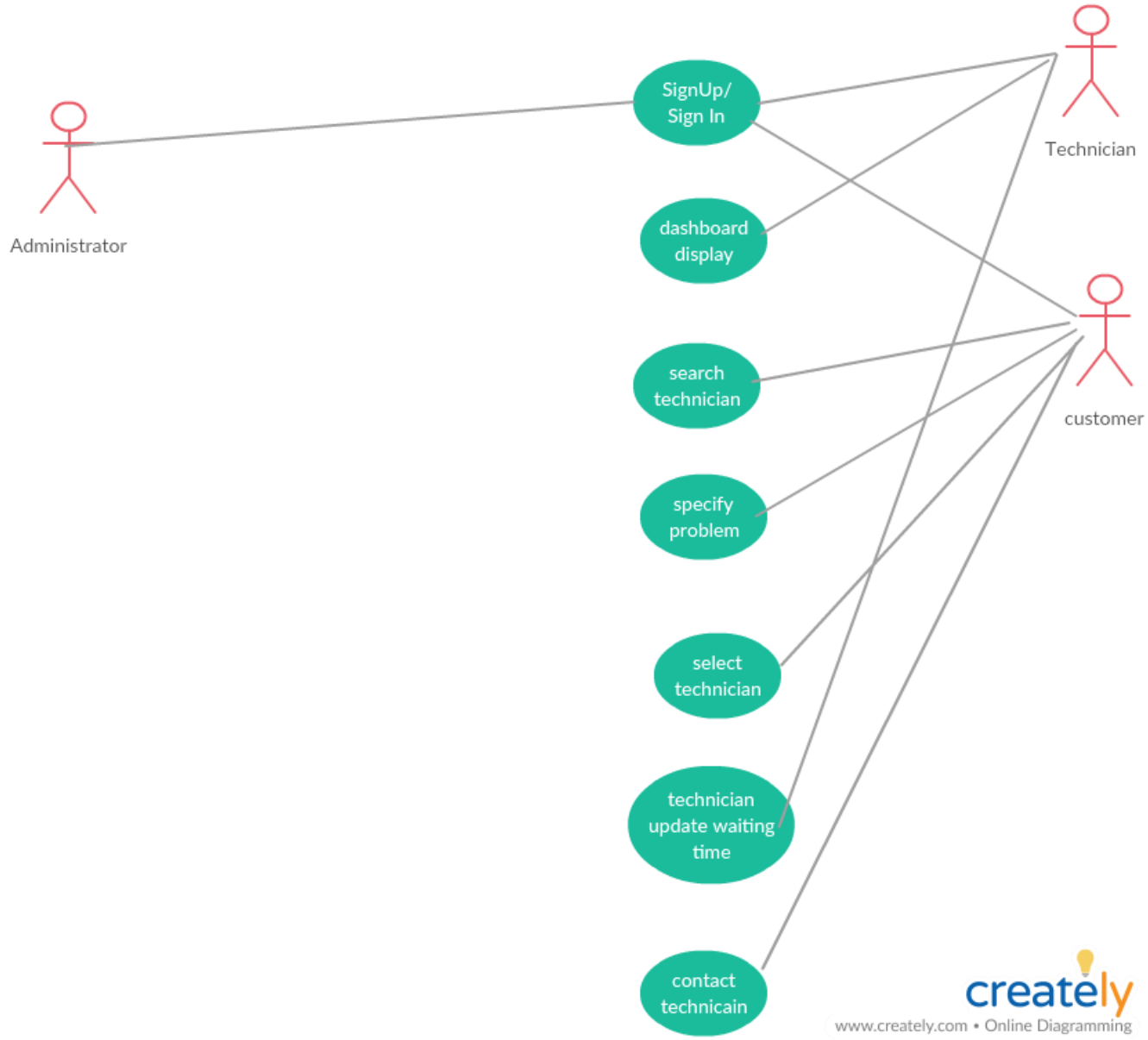
creately
www.creately.com • Online Diagramming

Class Diagram:



creately
www.creately.com • Online Diagramming

Use case Diagram:



Testing:

Unit Testing is not implemented for this increment due to time constraint and the unfamiliarity with the JUnit testing tool. Nonetheless, the team members did a lot of debugging to make sure that the core functions are executing properly without side effects.

Implementation:
The form of Application we are building is native application.

Platform:
Android

Tool:
Android Studio version 3.1.x For UI layout improvement:

We have created a custom logo for both technician and customer for the homepage We have added a dimen resource file and style.xml file in the values and added the images and background.xml files for choosing the background.

getLocationRequest() is responsible for determining the frequency of updating the real-time location. For FixitUp, we set it to be around 4000 milliseconds, or 4 seconds to be the average. The fastest it can update is 2 seconds.

```
private LocationRequest getLocationRequest() {
    LocationRequest locationRequest = new LocationRequest();
    locationRequest.setInterval(4000);
    locationRequest.setFastestInterval(2000);
    locationRequest.setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY)
    return locationRequest;
}
```

LocationCallback will be returned as soon as the getLocationRequest() finishes. This will update the UI with a marker that indicates the current location and also updates the last known location of the technician to the database

```
mLocationCallback = new LocationCallback() {
    @Override
    public void onLocationResult(LocationResult locationResult) {
        new FetchAddressTask( applicationContext: CustomerMapActivity.this,  listener: CustomerMapActivity.tl
                .execute(locationResult.getLastLocation());
        mLastKnownLocation=locationResult.getLastLocation();
        mMap.animateCamera(CameraUpdateFactory.newLatLng(new LatLng(mLastKnownLocation.getLatitude
                mLastKnownLocation.getLongitude() )));
        if (currentLocationMarker != null) {
            currentLocationMarker.remove();
            currentLocationMarker = mMap.addMarker(new MarkerOptions().position(new LatLng(mLastKn
                    .title("Current Location"));
            currentLocationMarker.setSnippet("Latitude: " + mLastKnownLocation.getLatitude() + ",
        }
```

This request button is placed within the Customer Map activity with a click listener that will pinpoint the current location as the repair location and also update that location to the Firebase database

```
});
mRequest = (Button) findViewById(R.id.request);
mRequest.setOnClickListener(new View.OnClickListener(){

    @Override
    public void onClick(View v) {
        String userID = FirebaseAuth.getInstance().getCurrentUser().getUid();
        DatabaseReference ref = FirebaseDatabase.getInstance().getReference( s: "customerRequest");
        GeoFire geoFire = new GeoFire(ref);
        getDeviceLocation();
        geoFire.setLocation(userID, new GeoLocation(mLastKnownLocation.getLatitude(), mLastKnownL
            @Override
            public void onComplete(String key, DatabaseError error) {
                //Do some stuff if you want to
            }
        });
        repairLocation = new LatLng(mLastKnownLocation.getLatitude(), mLastKnownLocation.getLongi
        mMap.addMarker(new MarkerOptions().position(repairLocation)
                .title("Repair Location"));
        mRequest.setText("Sending request to Technicians...");
    }
});
```

When Google Map is ready, it has four big functions to call, each of which is a wrapper function of its own auxiliary functions. getLocationPermission() will ask for user's permission updateLocationUI() and getDeviceLocation() will update the UI and location based on the permission. If permission is denied, default location of Sydney, Australia is used startTrackingLocation() will be used to listen to the user's location changes and update the Firebase database accordingly.

```
public void onMapReady (GoogleMap googleMap){
    mMap = googleMap;

    // Prompt the user for permission. If permission is denied, no location is provided
    getLocationPermission();

    //Based on the provided location permision
    //,turn on (or off) the My Location layer and the related control on the map.
    updateLocationUI();
    //This will be called to move the camera to its initial position
    getDeviceLocation();

    //Based on the provided location permission,
    // Get the current location of the device and set the position of the map.
    startTrackingLocation();

}
```

getLocationPermission() will request user's permission to access fine location and check whether permission is granted

```
private void getLocationPermission() {
    /*
     * Request location permission, so that we can get the location of the
     * device. The result of the permission request is handled by a callback,
     * onRequestPermissionsResult.
     */
    /*There are many variables for location such as ACCESS_FINE_LOCATION, ACCESS_COARSE_LOCATION, ...
    if (ContextCompat.checkSelfPermission(this.getApplicationContext(),
            android.Manifest.permission.ACCESS_FINE_LOCATION)
            == PackageManager.PERMISSION_GRANTED) {
        mLocationPermissionGranted = true;
    } else {
        ActivityCompat.requestPermissions( activity: this,
                new String[]{android.Manifest.permission.ACCESS_FINE_LOCATION},
                PERMISSIONS_REQUEST_ACCESS_FINE_LOCATION);
    }
}
```

startTrackingLocation() takes care of listening to location changes, updating the latest locations on the map, and updating that location to the Firebase database

```
private void startTrackingLocation() {
    if (ActivityCompat.checkSelfPermission( context: this,
            android.Manifest.permission.ACCESS_FINE_LOCATION)
            != PackageManager.PERMISSION_GRANTED) {
        ActivityCompat.requestPermissions( activity: this, new String[]
                        {android.Manifest.permission.ACCESS_FINE_LOCATION},
                REQUEST_LOCATION_PERMISSION);
    } else {
        mFusedLocationProviderClient.requestLocationUpdates
                (getLocationRequest(),
                        mLocationCallback,
                         looper: null /* Looper */);
    }
}
```

getDeviceLocation() will be called to update the latest location to update the initial location based on permission. startTrackingLocation() is not meticulous in tracking permission and default location.

```
private void getDeviceLocation() {
    /*
     * Get the best and most recent location of the device, which may be null in rare
     * cases when a location is not available.
     */
    try {
        if (mLocationPermissionGranted) {
            Task<Location> locationResult = mFusedLocationProviderClient.getLastLocation();
            locationResult.addOnCompleteListener( activity: this, (task) → {
                    if (task.isSuccessful()) {
                        mLastKnownLocation = task.getResult();
                        /*I modified this portion of the method because there is a tricky case
                         * such that even the FusedLocationProviderClient's getLastLocation was succes
                         * the mLastKnownLocation is set to null for some reason. As a result,
                         * the getLatitude() and getLongitude() will crash the app due to errors from
                         * access a null value*/
                        if (mLastKnownLocation == null) {
                            Log.d(TAG, msg: "Current location is null. Using defaults.");
                            Log.e(TAG, msg: "Exception: %s", task.getException());
                            mMap.moveCamera(CameraUpdateFactory
                                    .newLatLngZoom(mDefaultLocation, DEFAULT_ZOOM));
```

stopTrackingLocation() is called when the application is paused or closed.

```
private void stopTrackingLocation() {
    mFusedLocationProviderClient.removeLocationUpdates(mLocationCallback);
}
@Override
```

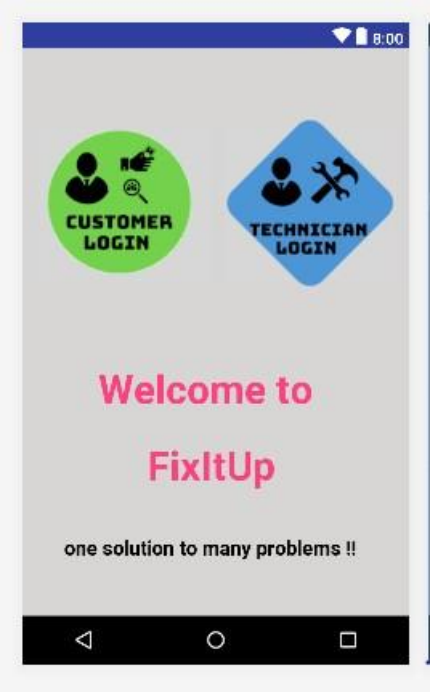
Deployment:

Our project is up-to-date with our GitHub as far as the most stable release is concerned. Other ongoing development and features need to be further tested, analyzed, and discussed. They are not yet published and are intended to be released in the upcoming increments.

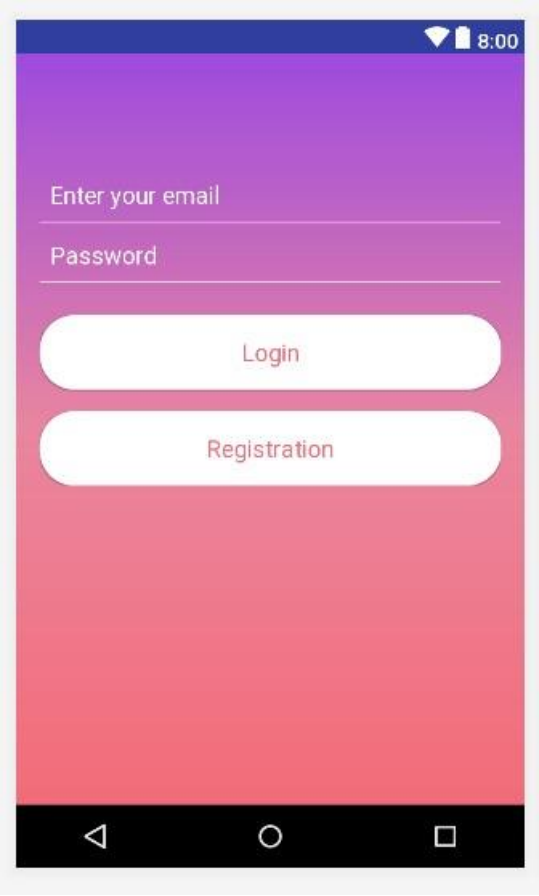
GitHub Repository URL

Wiki Link

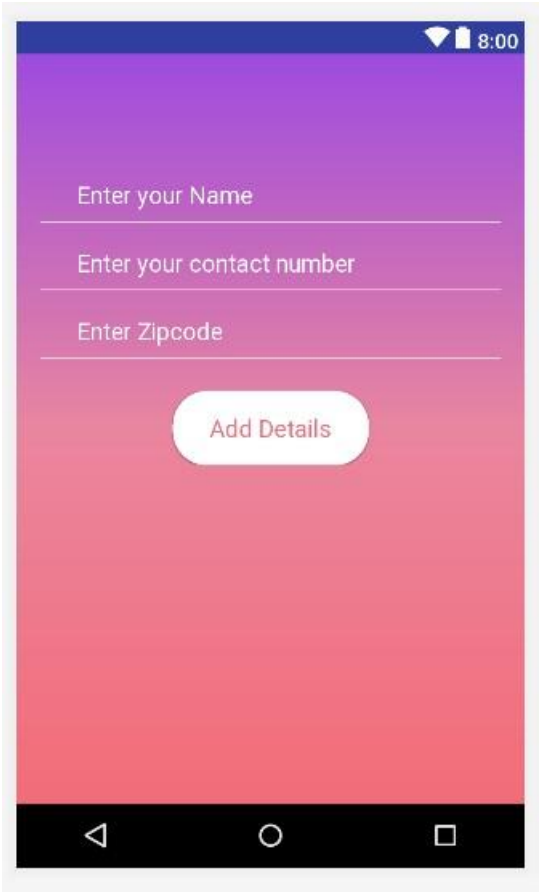
Homepage



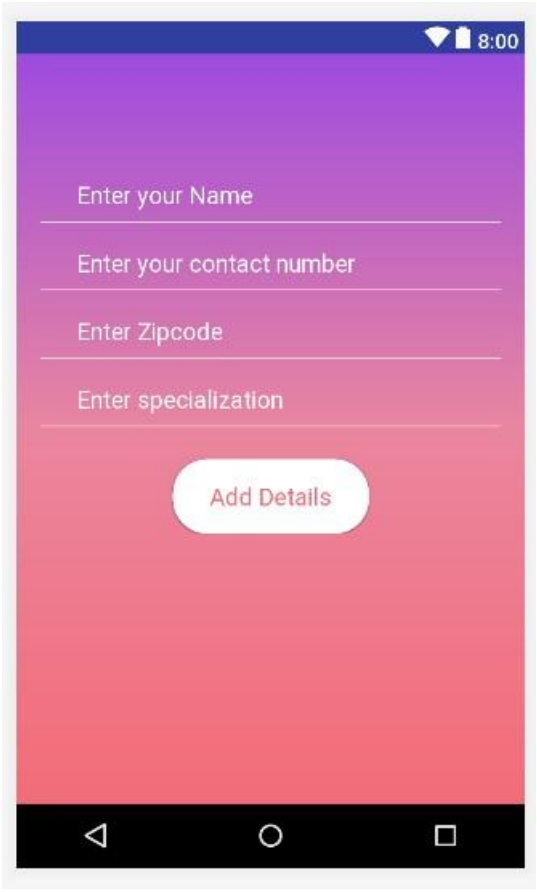
Login page for customer and technician



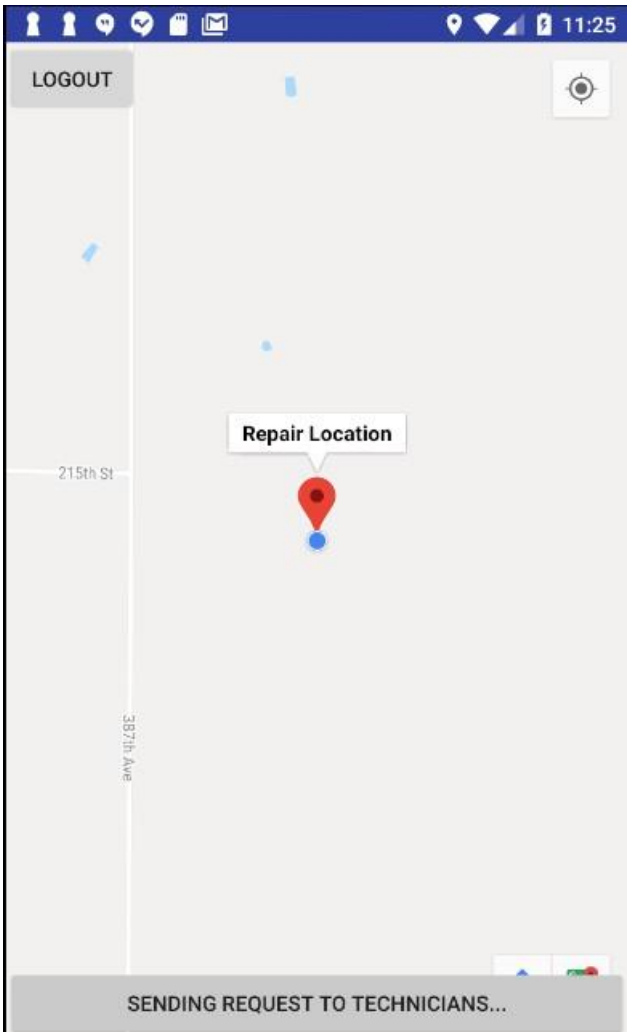
Customer Registration



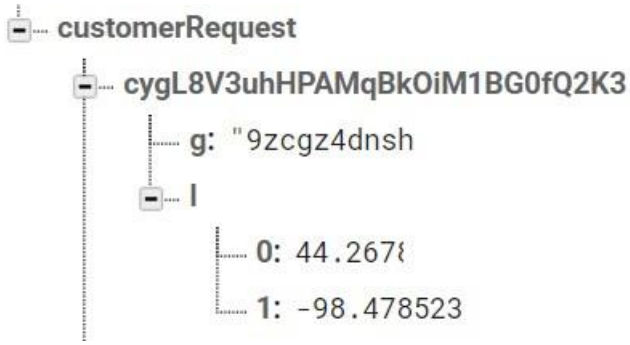
Technician Registration



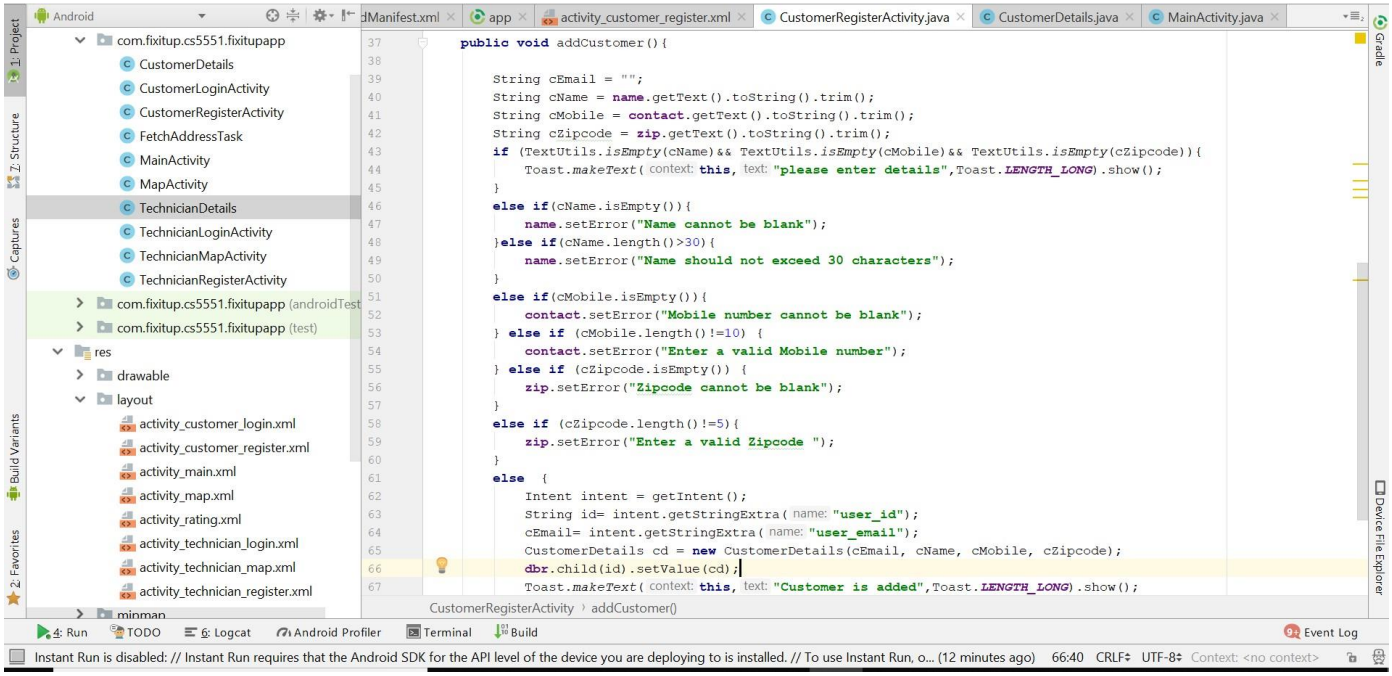
Demo of Map Feature 2 (Request button is pressed)



Firebase Database's stored latitude and longitude. Technician's stored location will be deleted as soon as he leaves the application so that the location is always fresh and updated.



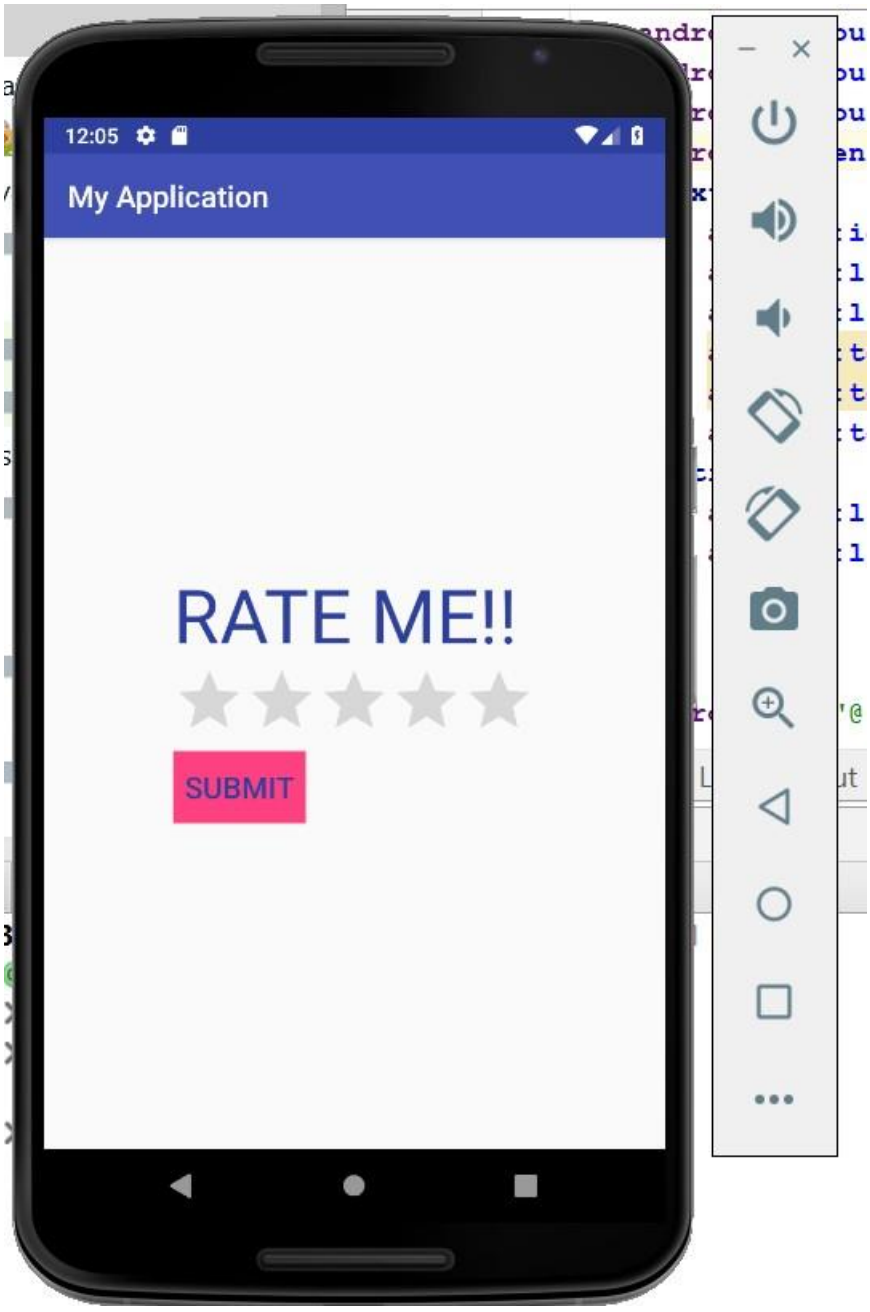
Customer Validation

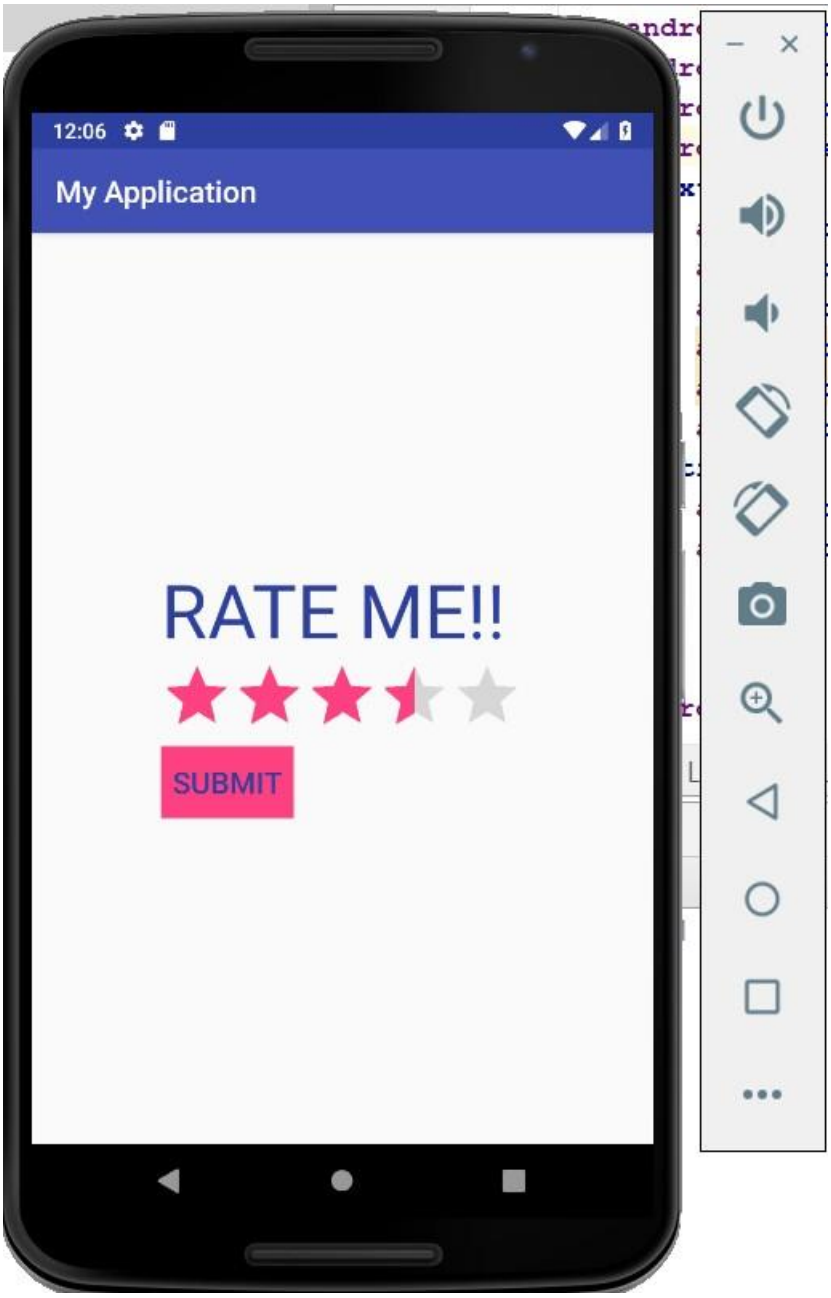


Rating For a Technician

Customer is given chance to rate the technician based on his performance. Once the rating is done, it is stored into the database and the average of all the ratings will be display for a particular Technician.

Demo of Rating


12:05
My Application
RATE ME!!
SUBMIT



Project Management

Implementation Status Report

Work Completed

Description:

- Map Feature (realtime display, realtime update to database)
-
- Logout function
-
- UI layout improvement

Registration data validation

Database retrieval in Home page

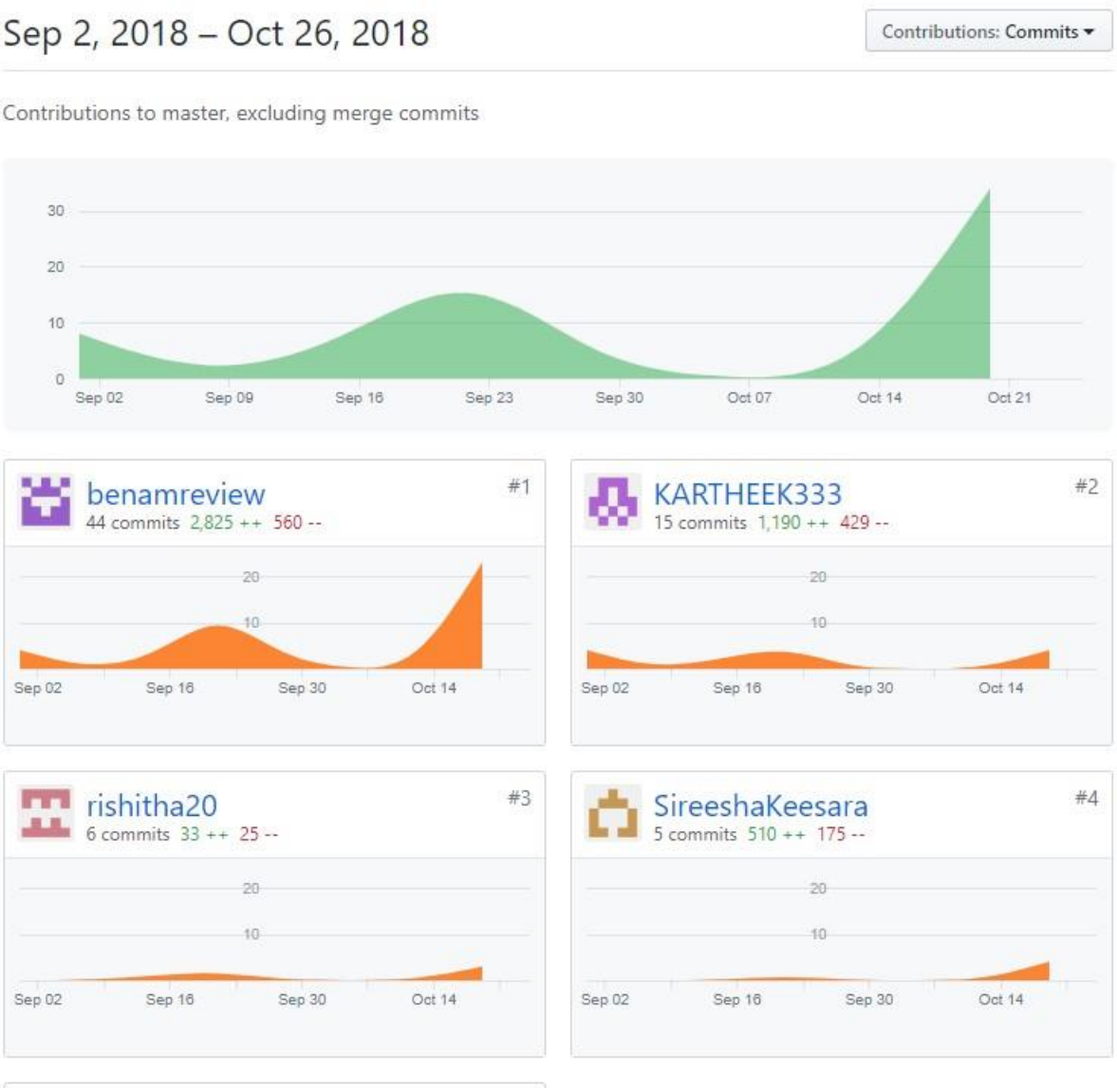
Responsiblity:

- Duy: Map Feature, Log out function, Version control
-
- Karthik: UI layout for all activities, Logo Design

Sireesha: Database Retrieval, connection between Customer and Technician activity.

- Rishitha: Customer data validation

Time Taken: 45 hours Contributions:



Work to be Completed :

Description:

- Home page improvement for Customers and Technicians
- Rating and Feedback
- Request the closest technician
- Waiting time, Queue
- Pricing and charges

Distance and Time Estimation

Scheduling

Server side setup

UI Improvement and interaction

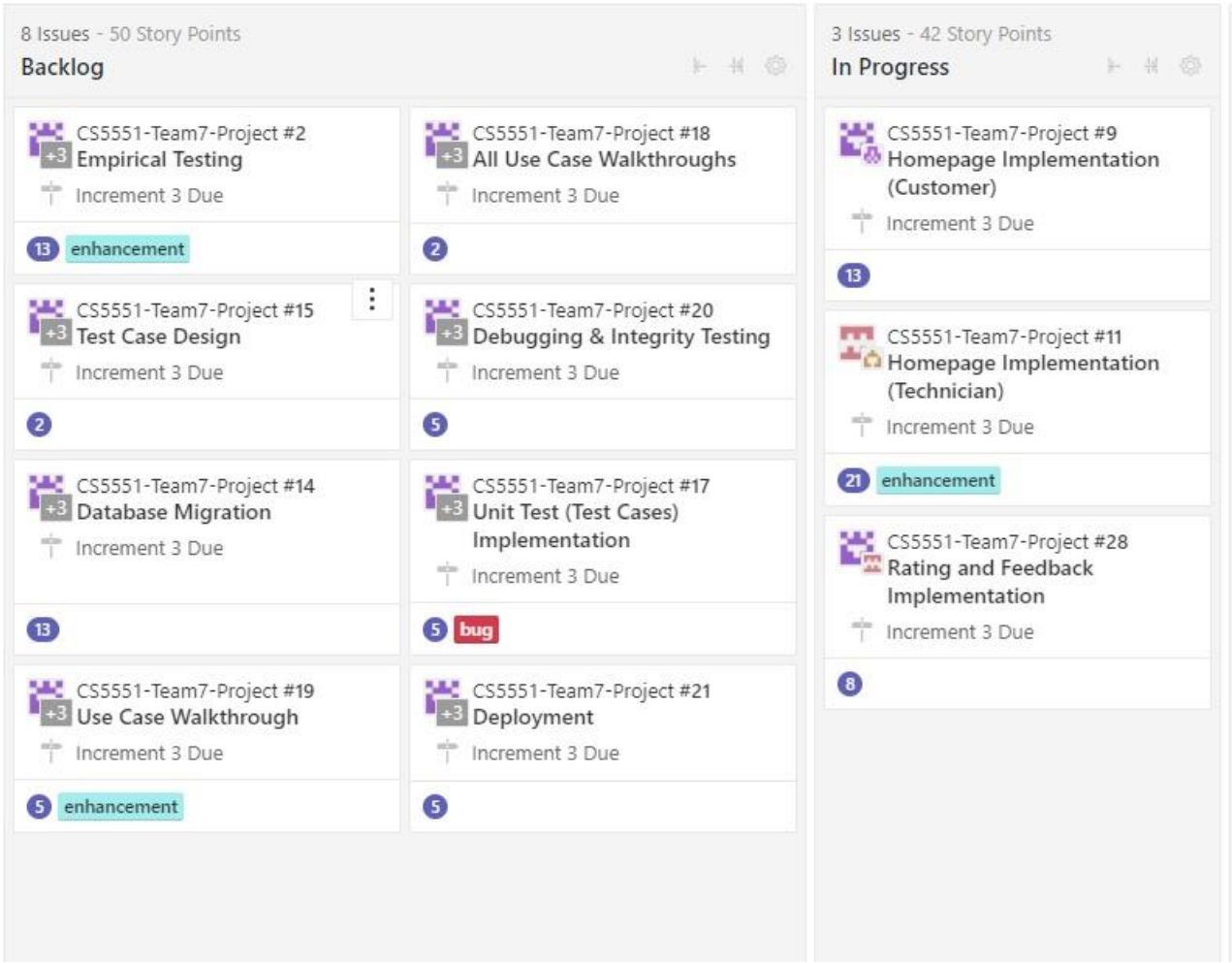
Unit Testing

Responsibility:

- Duy: Home page improvement for Customers and Technicians, Request the closest technician, Server
- setup Karthik: UI improvement and interaction

Sireesha: Waiting time, Queue, Pricing and charges

- Rishitha: Rating and feedback, Home page improvement for Customers and Technicians, Testing



THIRD INCREMENT REPORT

Existing Services/ REST API

Up to the end of this increment, we are not implementing any REST API (APIs that use get and parameters to retrieve information such as Weather Underground and Nutritionix). However, we are using:

- Google Firebase: as cloud database/storage, and cloud authentication using User Email/ Password. This eliminates the constraints of validating against the database and Firebase has a built-in verification method for itself using FireBaseAuth
- Google Maps for Android SDK: acts to show users where they are currently located and other technicians' current location as well. This will be useful as it gives the customers/technicians a better picture of waiting time/remaining distance from one another.
- GeoFire: Updates real-time location of customers/technicians to Firebase Database and queries the storage of locations based on distance range and requested location of customer. Up to the end of this increment we have made changes to the customer home page which now displays the technicians with the searched specialization and enables the customer to order the technician. Using GeoFire requesting a technician nearby. And also implemented cancellation of orders.

Implementation:

The form of Application we are building is native application.

Platform:
Android

Tool:
Android Studio version 3.1.x For UI layout improvement:

CustomerDetails page wit hname, contact, zipcode, and email. This object will be used to push to the Firebase database

```java
package com.fixitup.cs5551.fixitupapp;

public class CustomerDetails {

    String name;
    String contact;
    String zipcode;
    String email;
    public CustomerDetails(String Email,String Name, String Contact, String Zipcode) {
        name = Name;
        email = Email;
        contact = Contact;
        zipcode = Zipcode;

    }
    public String getEmail(){ return email;}

    public String getName() {
        return name;
    }

    public String getContact() { return contact; }

    public String getZipcode() { return zipcode; }

}
```

Search function is implemented with A listview and an adapter that converts the object retrieved from the Firebase to a

TechnicianDetails object to be displayed as a list on the Customer Home page

```
search.setOnClickListener((v) → {
        Query q = dbr.orderByChild("type").equalTo(type.getText().toString().trim());
        q.addValueEventListener(new ValueEventListener() {
            @Override
            public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
                //To-be implemented: have a spinner and s
                list.clear();
                for(DataSnapshot ds: dataSnapshot.getChildren()){
                    //String s = dataSnapshot.child("type").getValue().toString();

                    td = ds.getValue(TechnicianDetails.class);
                    //  if(s.equalsIgnoreCase(st)) {
                    // boolean b= st.equalsIgnoreCase(td.getType().toString());
                    //if(b==true){
                    list.add(td.getName().toString() + "\n" + td.getEmail().toString() + "\n" + td.getContact().toString() + "\n " + td.
                    //}
                }
                lv.setAdapter(ad);
                lv.setOnItemClickListener((parent, view, position, id) → {
                        Object o = parent.getItemAtPosition(position);
                        Intent i = new Intent( packageContext: CustomerHome.this, CustomerProfile.class);
                        startActivity(i);
                });
            }

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

            }
        });

});
```

3 new features that have been added, including Setting buttons (for changing display and personal information), Logout to end the current user session, and Profile Image for displaying the user's image from photos or gallery

```
    });
    mSettings = (Button) findViewById(R.id.settingsBtn);
    mSettings.setOnClickListener((v) → {
            Intent intent = new Intent( packageContext: CustomerHome.this, CustomerSettingsActivity.class);
            startActivity(intent);
            finish();
            return;
    });
    mLogout = (Button) findViewById(R.id.logout);
    mLogout.setOnClickListener((v) → {
            FirebaseAuth.getInstance().signOut();
            Intent intent = new Intent( packageContext: CustomerHome.this, MainActivity.class);
            startActivity(intent);
    });
    //Click a picture
    mProfileImage.setOnClickListener((v) → {
            Intent intent = new Intent(Intent.ACTION_PICK);
            intent.setType("image/*"); //restricts selection
            startActivityForResult(intent,  requestCode: 1);
    });
    getUserInfo();
}
```

Implementation of code that sets the profile image. Image is stored as bitmap and compressed to JPEG and uploaded to

Firebase Storage which will then later store the image and provide a URL in the Firebase Database

```
protected void onActivityResult(int requestCode, int resultCode, @Nullable Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    //See if it returns the same code from the startactivityforresult above
    if (requestCode == 1 && resultCode == Activity.RESULT_OK){
        final Uri imageURI = data.getData();
        resultURI = imageURI;
        mProfileImage.setImageURI(resultURI);
        applyChanges();
    }

}

//Apply changes for profile images on Firebase Storage
//Note: need to enable storage first for permission issues.
private void applyChanges(){

    if (resultURI != null){
        final StorageReference filePath = FirebaseStorage.getInstance().getReference().child("profile_images").child(userID);
        Bitmap bitmap = null;
        //Add uri to bitmap
        try {
            bitmap = MediaStore.Images.Media.getBitmap(getApplication().getContentResolver(), resultURI);
        } catch (IOException e) {
            e.printStackTrace();
        }
        ByteArrayOutputStream baos = new ByteArrayOutputStream(); //compress images
        bitmap.compress(Bitmap.CompressFormat.JPEG, quality: 20, baos);
        byte[] data = baos.toByteArray();
        UploadTask uploadTask = filePath.putBytes(data);
        //Upload task will save the image to Firebase Database
        uploadTask.addOnSuccessListener((OnSuccessListener) (taskSnapshot) → {
                filePath.getDownloadUrl().addOnSuccessListener((OnSuccessListener) (uri) → {
                        Map newImage = new HashMap();
```

The Old Customer Login activity has been split into CustomerLoginActivity,

CustomerLoginScreenActivity, and CustomerSignUp activity for better separation of concerns.

The implementation of the rating bar and its corresponding translation of number of stars to text

```
mRatingBar.setOnRatingBarChangeListener(new RatingBar.OnRatingBarChangeListener() {
    @Override
    public void onRatingChanged(RatingBar ratingBar, float v, boolean b) {
        mRatingScale.setText(String.valueOf(v));
        int mNum = (int) ratingBar.getRating();
        mRatingNum =  Integer.toString(mNum);
        switch (mNum) {
            case 1:
                mRatingScale.setText("Bad");
                break;
            case 2:
                mRatingScale.setText("Need improvement");
                break;
            case 3:
                mRatingScale.setText("Fair");
                break;
            case 4:
                mRatingScale.setText("Good!");
                break;
            case 5:
                mRatingScale.setText("Awesome!");
                break;
            default:
                mRatingScale.setText("None");
        }
    }
});
```

Implementation of feedback: feedback is not stored to Firebase yet, but the rating will be stored under the profile of the respective Technician/Customer

```
mSendFeedback.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        if (mRatingNum == null){
            Toast.makeText( context: CustomerRatingActivity.this,  text: "Please give a rating!", Toast.LENGTH_LONG).show();
        }
        if (mFeedback.getText().toString().isEmpty()) {
            Toast.makeText( context: CustomerRatingActivity.this,  text: "Please fill in feedback text box", Toast.LENGTH_LONG).show();
        }
        else {
            Intent intent = getIntent();
            String technicianID = intent.getStringExtra( name: "tID");
            //Make changes to database
            DatabaseReference technicianRef = FirebaseDatabase.getInstance().getReference().child("Users").child("Technicians").child(te
            if (technicianRef!=null){
                technicianRef.setValue(true);
            }


            mFeedback.setText("");
            mRatingBar.setRating(0);
            Toast.makeText( context: CustomerRatingActivity.this,  text: "Thank you for sharing your feedback", Toast.LENGTH_SHORT).show();
            try {
                Thread.sleep( millis: 1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            intent = new Intent( packageContext: CustomerRatingActivity.this, CustomerHome.class);
            startActivity(intent);
        }
    }
});
```

Dialog box that warns the user that information will be changed permanently.

```
final CustomerDetails cd = new CustomerDetails( userEmail, tName, tMobile, tZipcode);
AlertDialog.Builder builder = new AlertDialog.Builder( context: CustomerSettingsActivity.this);
builder.setMessage("WARNING!!! If you proceed, all your previous informatio...")
        .setPositiveButton( text: "Yes, I Understand!", (dialog, id) → {

                dbr.child(userID).setValue(cd);
                Toast.makeText( context: CustomerSettingsActivity.this, text: "Information Updated Successfully",Toast.LENGTH_LONG).show()
        })
        .setNegativeButton( text: "Cancel", (dialog, id) → {
                // User cancelled the dialog
        });
builder.show();
```

Order object that contains the TechnicianID, customerID, and status

```
package com.fixitup.cs5551.fixitupapp;

public class Order {
    String technicianID;
    String customerID;
    String status;
    public Order(){
        technicianID = "";
        customerID = "";
        status = "";
    }
    public Order(String tID, String cID, String stat) {
       technicianID = tID;
       customerID = cID;
       status = stat;

    }
    public String getTechnicianID() { return technicianID; }

    public String getCustomerID() { return customerID; }
    public String getStatus() { return status; }

}
```

Distance notification whether a technician or customer is nearby

```
    //Calculate distance
    float distance = loc1.distanceTo(loc2);
    if (distance <100){
        mRequest.setText("Technician is almost here (within 100m)! Please be prepared!");
    }
    else if (distance < 200){
        mRequest.setText("Technician is nearby (within 200m)!Please be prepared!");
    }
    else{
        mRequest.setText("Technician Found at Location (" + String.valueOf(distance) + " meters) from you");
    }
    ;
    //Add marker to map to show technician's current location
    mTechnicianMarker = mMap.addMarker(new MarkerOptions().position(technicianLatLng).title("Currently Selected Technician"));
    mTechnicianMarker.setIcon(BitmapDescriptorFactory.fromResource(R.mipmap.technician_icon));
}
```

This function is placed in the customermapactivity where the activity is constantly looking for changes in the customer table to see if the technician has started the session or not. If yes, the orderID will be available afterwards

```
private void getOrderID() {

    final DatabaseReference orderIDref = FirebaseDatabase.getInstance().getReference().child("Users").child("Customers").child(userID).child
    orderIDref.addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
            if (dataSnapshot.getValue() != null) {
                orderID = dataSnapshot.getValue().toString();
            }
            getOrderUpdate();
        }

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

        }
    });
}
```

Notification on the TechnicianMapActivity that is invisible in the beginning and only shows when technician arrives at the repair location. Used to display a button for technician to start the session

```
mNotification.setOnClickListener((v) → {
        //To Start a session
        if (!sessionStarted){
            sessionStarted=true;
            mNotification.setVisibility(View.VISIBLE);
            mNotification.setText("Session is Active!\nClick to End Session");
            //Add order details
            Order order = new Order (userID, customerID, stat: "ongoing");
            DatabaseReference orderRef = FirebaseDatabase.getInstance().getReference( s: "Orders").push();
            orderRef.setValue(order);
            orderID = orderRef.getKey().toString();
            DatabaseReference customerRef= FirebaseDatabase.getInstance().getReference().child("Users").child("Customers").child(custome
            customerRef.child("orderID").setValue(orderID);

        }

        //To finish a session
        else {
            AlertDialog.Builder builder = new AlertDialog.Builder( context: TechnicianMapActivity.this);
            builder.setMessage("WARNING!!! If session is ended, order is complete and f...")
                    .setPositiveButton( text: "Yes, I do!", (dialog, id) → {
                            sessionStarted=false;
                            mNotification.setVisibility(View.GONE);
                            mNotification.setText("");
                            //Add Order Details to Firebase
                            DatabaseReference orderRef = FirebaseDatabase.getInstance().getReference( s: "Orders").child(orderID);
                            orderRef.child("status").setValue("completed");


                            //Delete customerRequest
                            DatabaseReference requestRef = FirebaseDatabase.getInstance().getReference().child("customerRequest").child(
                            requestRef.removeValue();

                            AlertDialog.Builder builder = new AlertDialog.Builder( context: TechnicianMapActivity.this);
                            builder.setMessage("Would you like to rate this customer?")
```

Send feedback to the Customers table of the Firebase database

```
mSendFeedback.setOnClickListener((view) → {
        if (mRatingNum == null){
            Toast.makeText( context: TechnicianRatingActivity.this,  text: "Please give a rating!", Toast.LENGTH_LONG).show();
        }
        if (mFeedback.getText().toString().isEmpty()) {
            Toast.makeText( context: TechnicianRatingActivity.this,  text: "Please fill in feedback text box", Toast.LENGTH_LONG).show();
        }
        else {
            Intent intent = getIntent();
            String customerID = intent.getStringExtra( name: "cID");
            //Make changes to database
            DatabaseReference customerRef = FirebaseDatabase.getInstance().getReference().child("Users").child("Customers").child(custome
            if (customerRef!=null){
                customerRef.setValue(true);
            }


            mFeedback.setText("");
            mRatingBar.setRating(0);
            Toast.makeText( context: TechnicianRatingActivity.this,  text: "Thank you for sharing your feedback!", Toast.LENGTH_SHORT).show();
            try {
                Thread.sleep( millis: 1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            intent = new Intent( packageContext: TechnicianRatingActivity.this, TechnicianHome.class);
            startActivity(intent);
        }
});
```

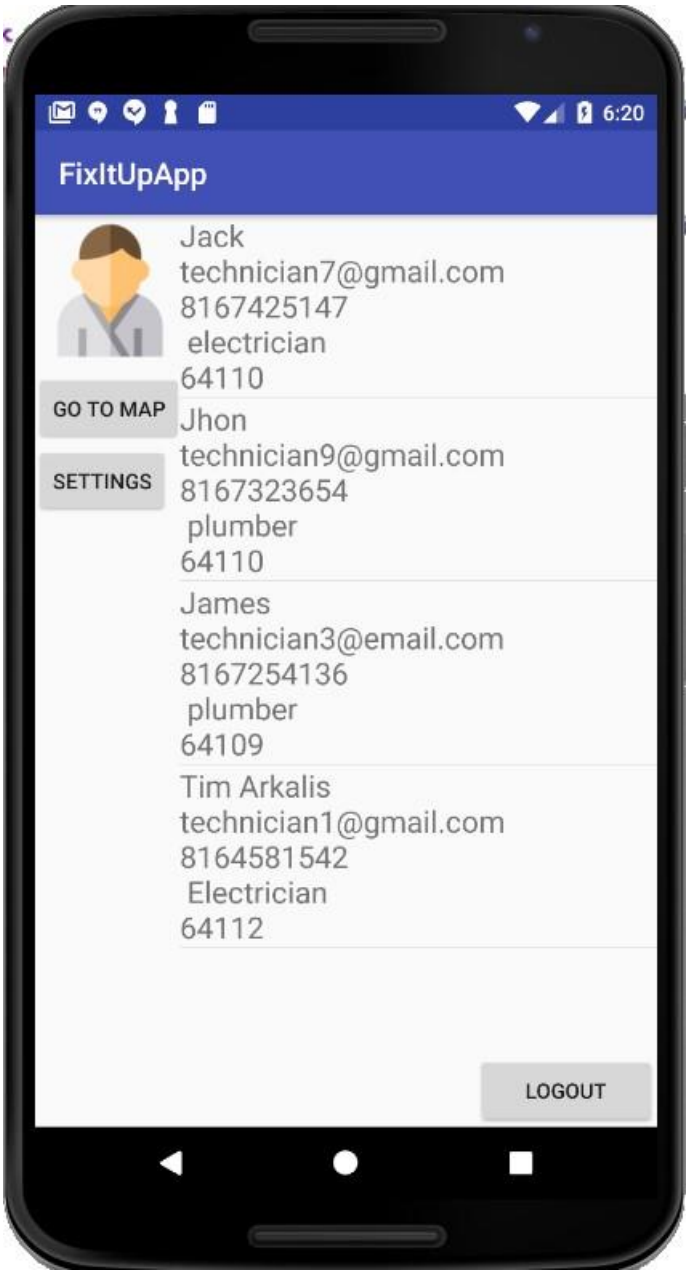
Deployment:

Our project is up-to-date with our GitHub as far as the most stable release is concerned. Other ongoing development and features need to be further tested, analyzed, and discussed. They are not yet published and are intended to be released in the upcoming increments.

GitHub Repository URL

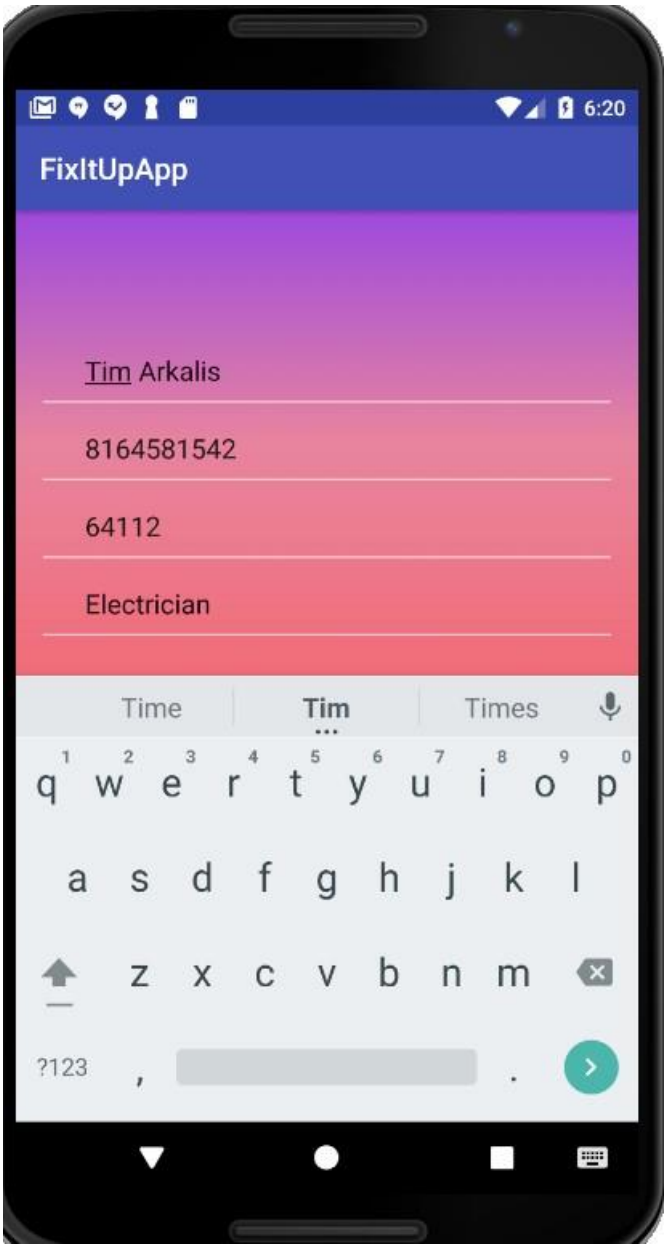
Wiki Link

Customer and Technician Home Page’s layout

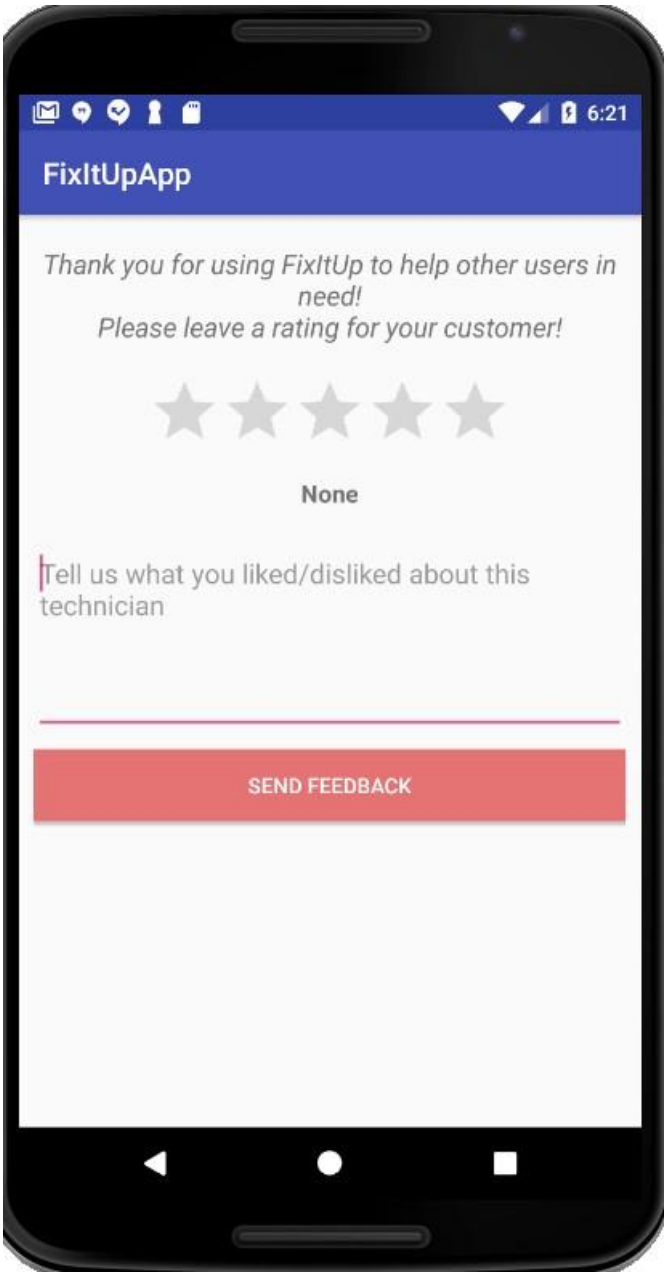


Customer Map Activity (default)

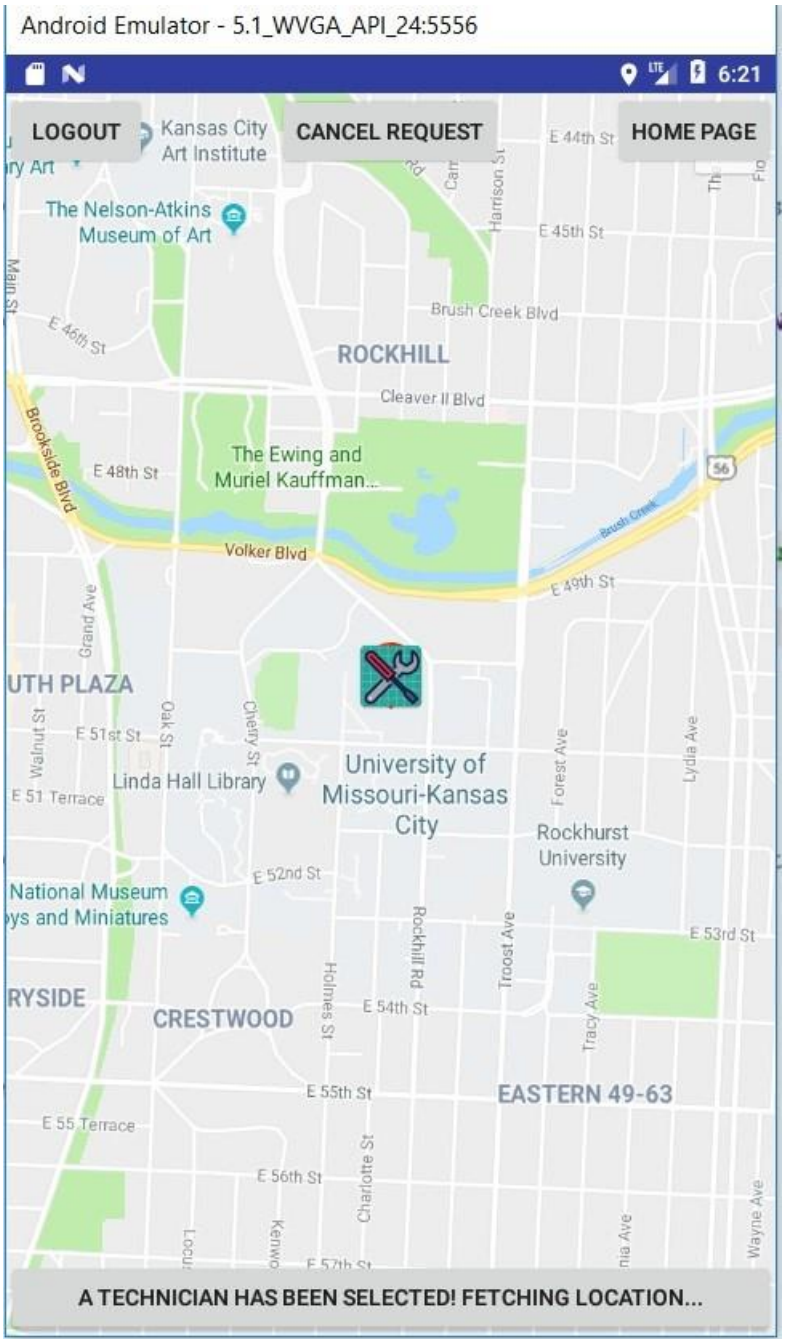
Technician Settings Activity



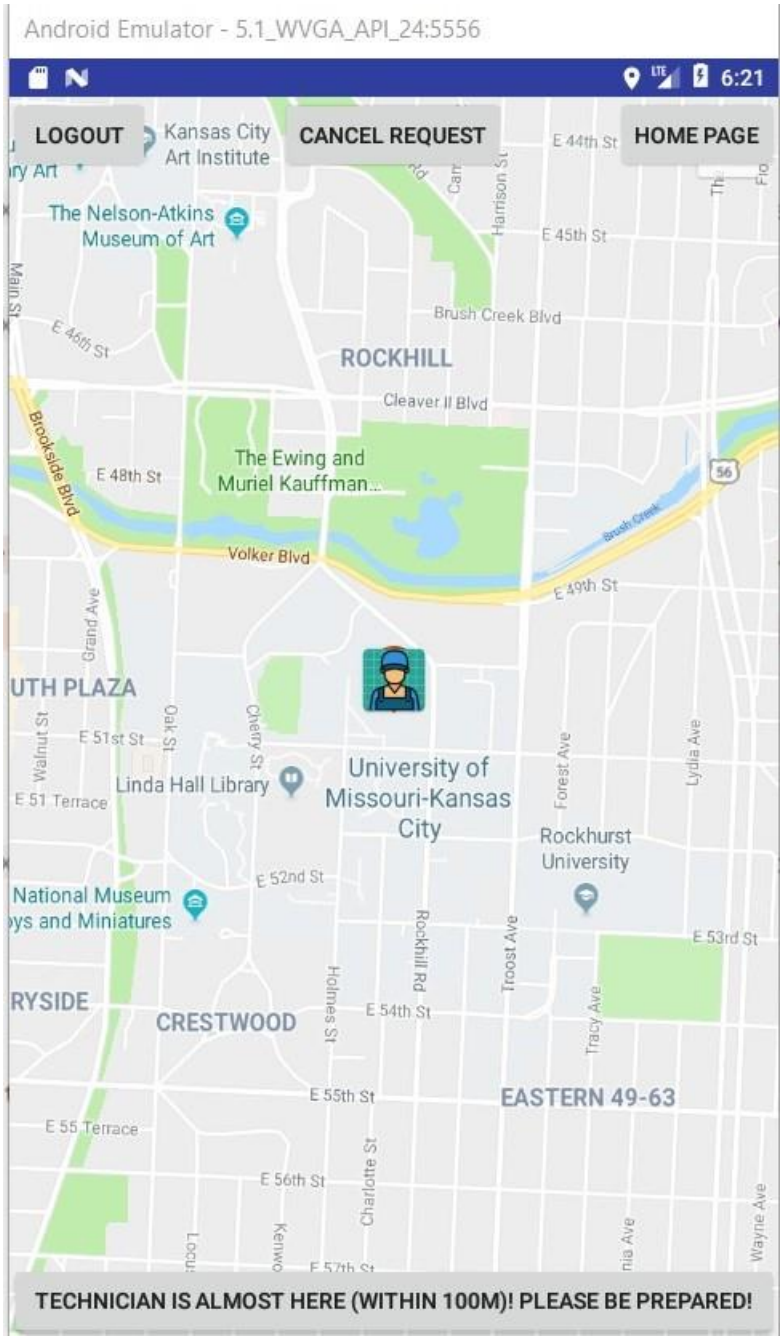
Technician and Customer Rating Activity



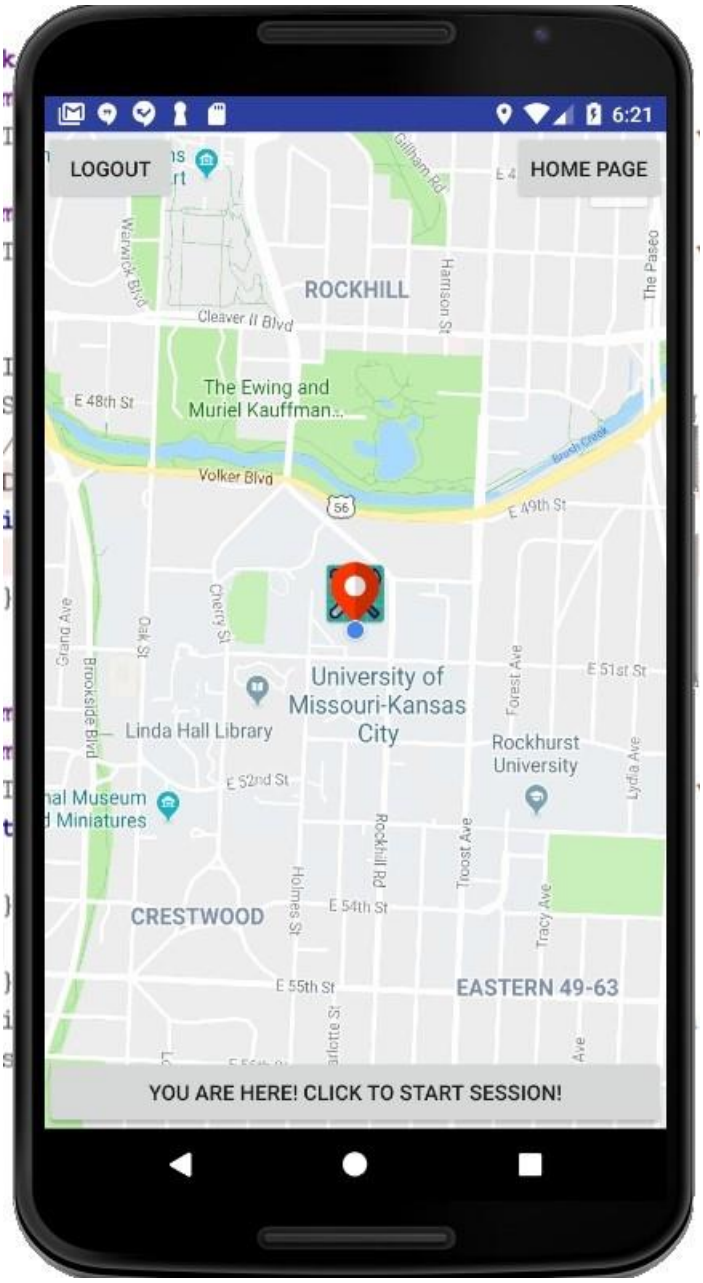
Customer: Fetching Location of Technician (displayed after clicking Request a Technician)



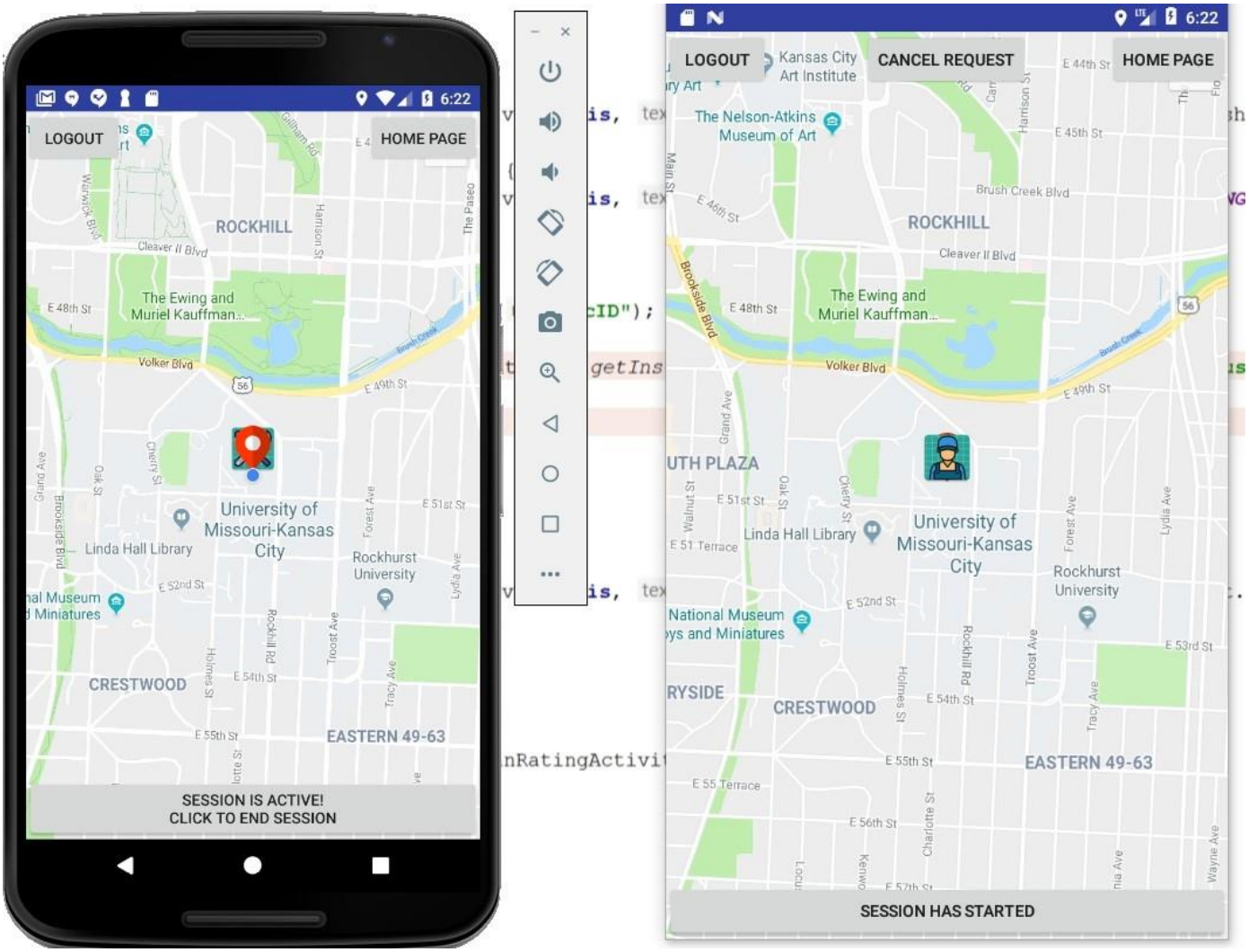
Customer: Tracking Location of Technician (after a technician has been matched)



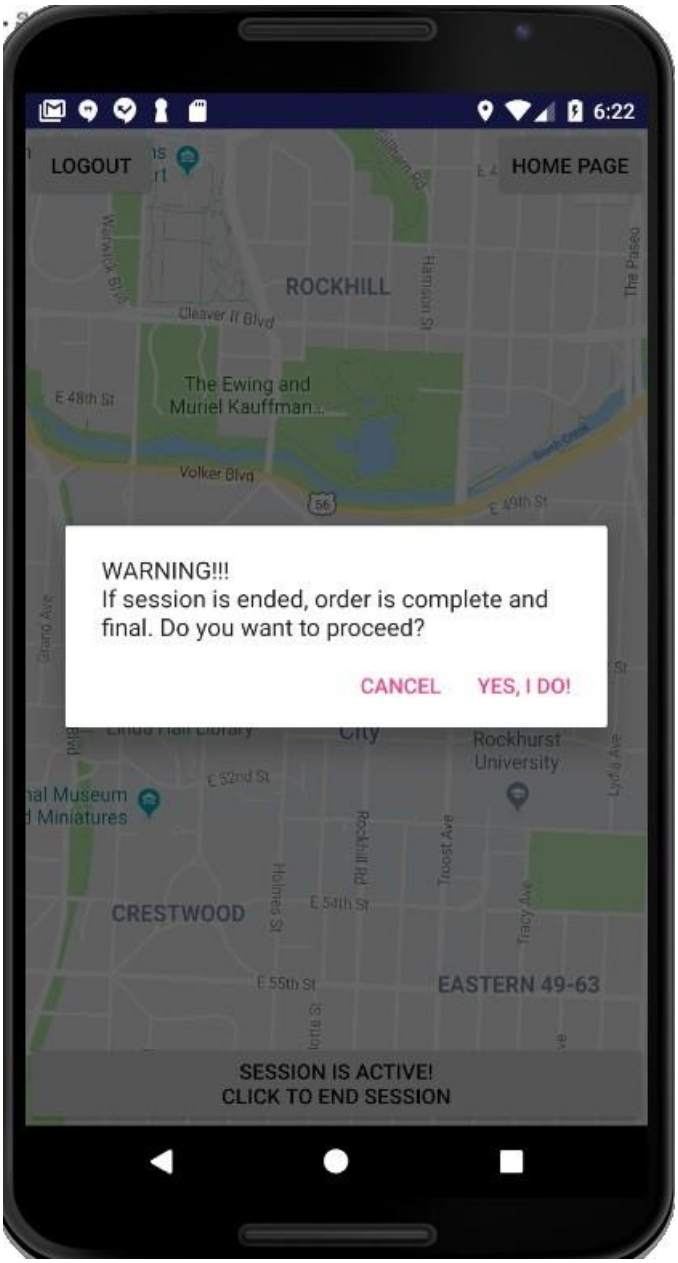
Technician: tracking location of Customer (when arrived)



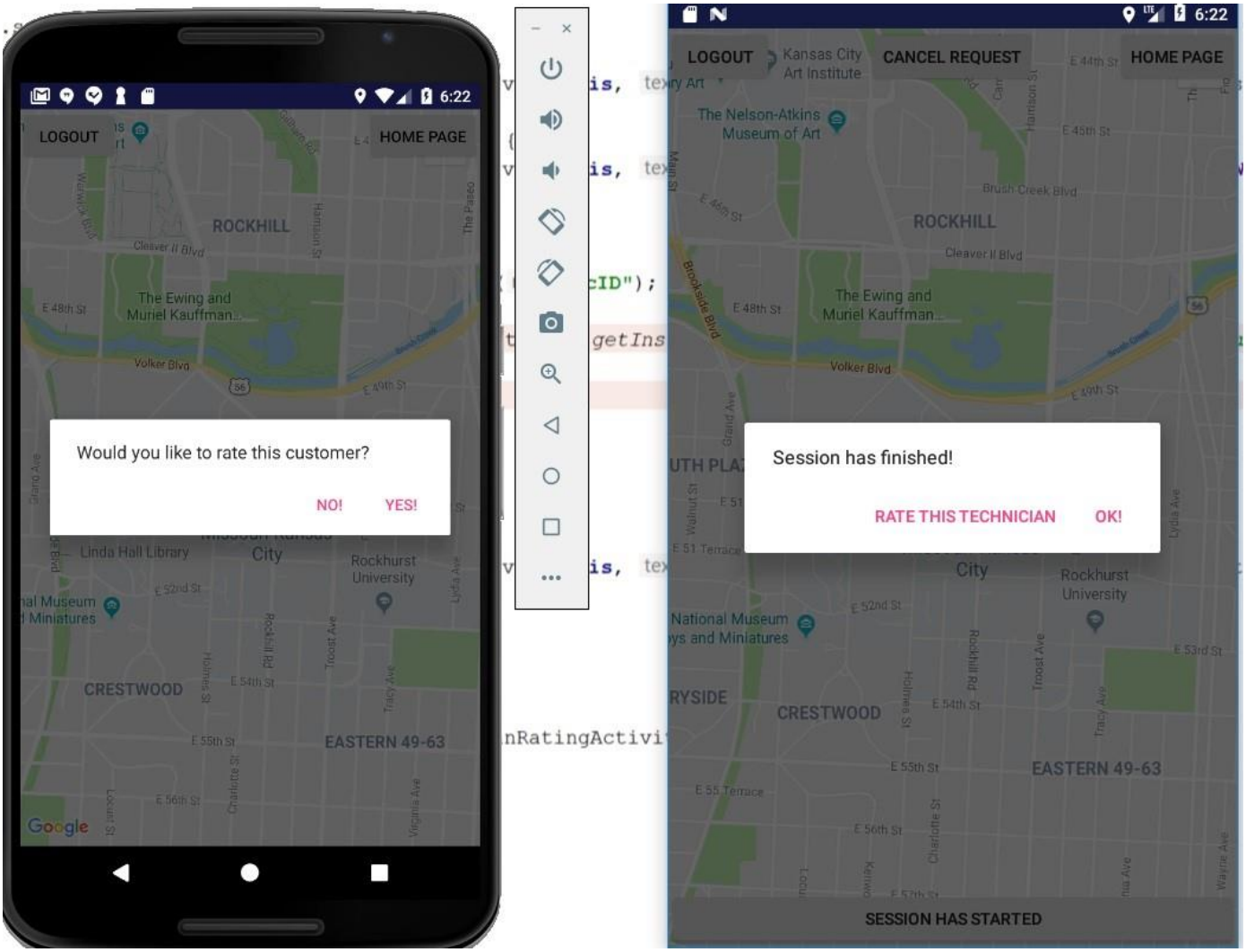
Technician and Customer: active session



Technician: session ending’s warning



Technician and Customer: Session ending notification



//added by sireesha

```
mCustomerDatabase= FirebaseDatabase.getInstance().getReference().child("Users").child("Customers").child(userID);

search=(Button)findViewById(R.id.btn);
type=(EditText)findViewById(R.id.search);
//
dbr = FirebaseDatabase.getInstance().getReference().child("Users").child("Technicians");
lv = (ListView)findViewById(R.id.listView);
td = new TechnicianDetails();
list = new ArrayList<>();
ad = new ArrayAdapter<String>( context: this, R.layout.list_layout, R.id.technician, list);
search.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Query q = dbr.orderByChild("type").equalTo(type.getText().toString().trim());
        q.addValueEventListener(new ValueEventListener() {
            @Override
            public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
                //To-be implemented: have a spinner and s
                list.clear();
                for(DataSnapshot ds: dataSnapshot.getChildren()){


                    td = ds.getValue(TechnicianDetails.class);

                    list.add(td.getName().toString() + "\n" + td.getEmail().toString() + "\n" + td.getContact().toStri

                }
                lv.setAdapter(ad);
                lv.setOnItemClickListener(new AdapterView.OnItemClickListener() {
                    @Override
```

6:14 PM ... 55%

FixItUpApp

What are you searching for?

Search

GO TO MAP

SETTINGS

LOGOUT

6:14 PM ... 55%

FixItUpApp

plumber

Search

GO TO MAP

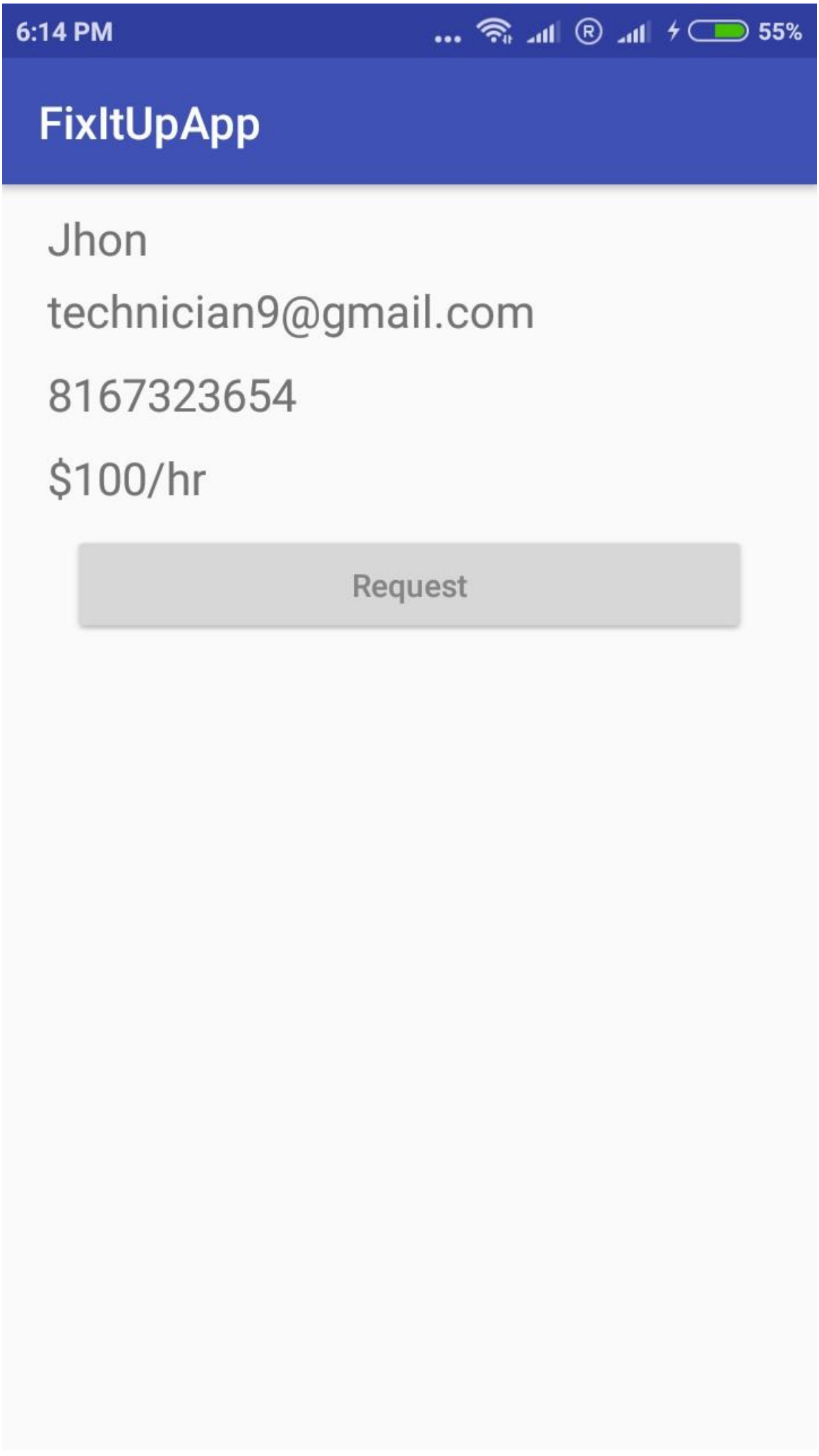
SETTINGS

Jhon
technician9@gmail.com
8167323654
plumber
64110
$100/hr

James
technician3@email.com
8167254136
plumber
64109
$120/hr

LOGOUT

```
TextView Name,Email,Contact,Fee;
Button btn;
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_customer_profile);
   Name=(TextView)findViewById(R.id.name);
   Email=(TextView)findViewById(R.id.email);
    Contact=(TextView)findViewById(R.id.contact);
    Fee=(TextView)findViewById(R.id.fee);
   btn=(Button)findViewById(R.id.book);
   Intent intent=getIntent();
   String s1 = intent.getStringExtra( name: "name");
   String s2 = intent.getStringExtra( name: "email");
    String s3 = intent.getStringExtra( name: "contact");
    String s4 = intent.getStringExtra( name: "fee");
    String s5 = intent.getStringExtra( name: "id");
   Name.setText(s1);
   Email.setText(s2);
   Contact.setText(s3);
   Fee.setText(s4);
```

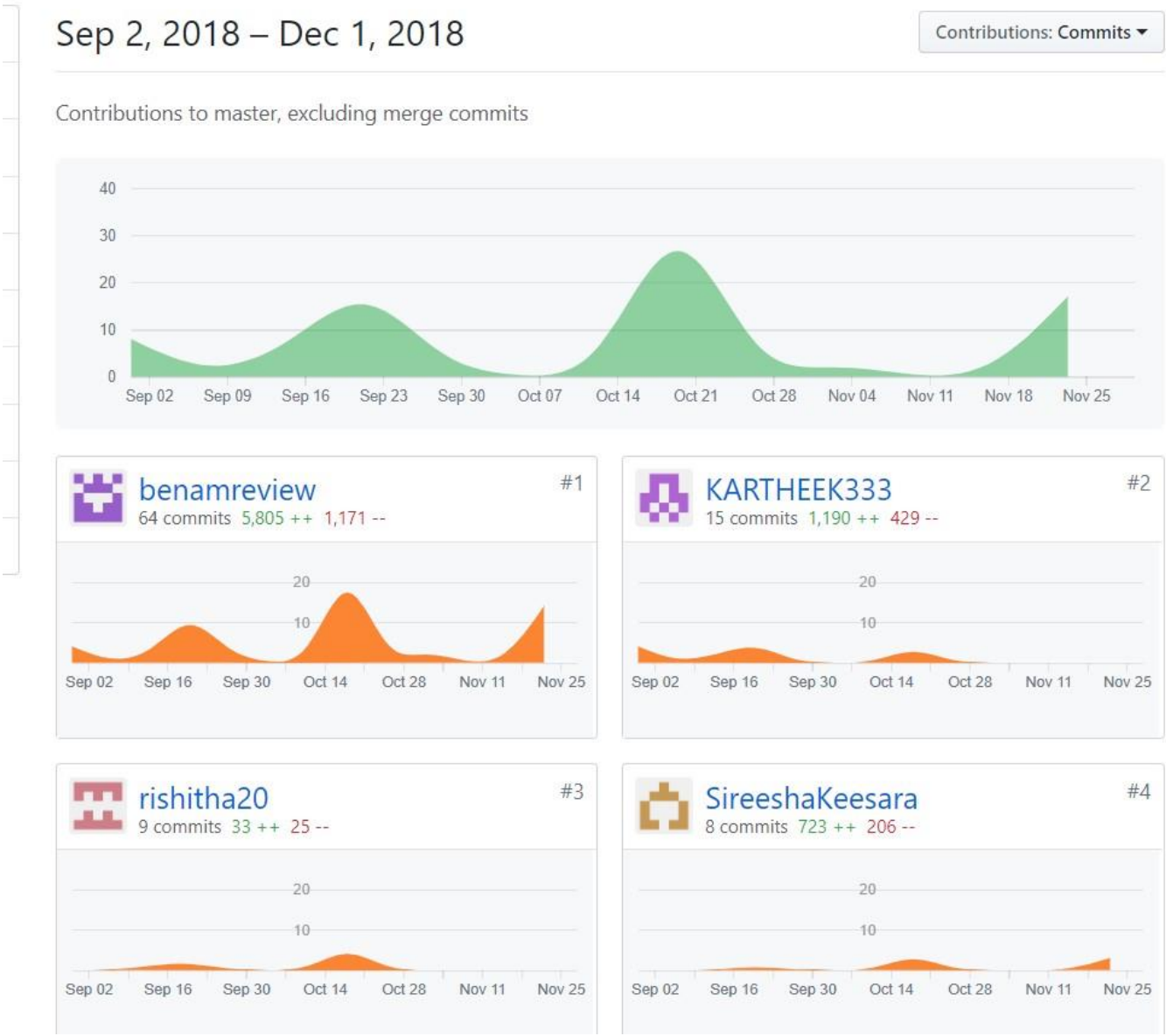

6:14 PM
55%
FixItUpApp
Jhon
technician9@gmail.com
8167323654
$100/hr
Request

Project Management

Implementation Status Report

Responsiblity:

- Duy: Location binding between Customers/Technicians, Completed order and session binding (cancel, started, completed), user's profile image, Customer and Technician ratings, Restructured app logic, layout, and flow.
- Karthik:
- Sireesha: Search function for customer, Customer Home Page, Customer Orders Technician.
- Rishitha: Estimated number of hours: 50



Bibliography

Firebase Tutorial

Android Tutorial

Google Maps Tutorial

Android Resources

"The work has been completed under the guidance of Dr. Yugi Lee, Rajaram Anantharaman, and TAs (Ruthvic Punyamurtula, Bhargavi Nadendla, Sravanthi Gogadi) in CS5551 Advanced Software Engineering, University of Missouri -Kansas City), Fall 2018.