



# Second Increment Report

benamreview edited this page Oct 26, 2018 · 20 revisions

[Edit](#)[New Page](#)[Jump to bottom](#)

## PROJECT TITLE : FIXITUPP ANDROID APP

### TEAM : 7

### TEAM MEMBERS:

1. DUY HO

2. SIREESHA KESARA

3. RISHITHA BOBBA

4. KARTHEEK KATTA

### 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 explicitly the 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.

## PROJECT PLAN (Revised as of increment 2):

### Burndown Chart :



### Velocity Chart:



## 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.

### 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:

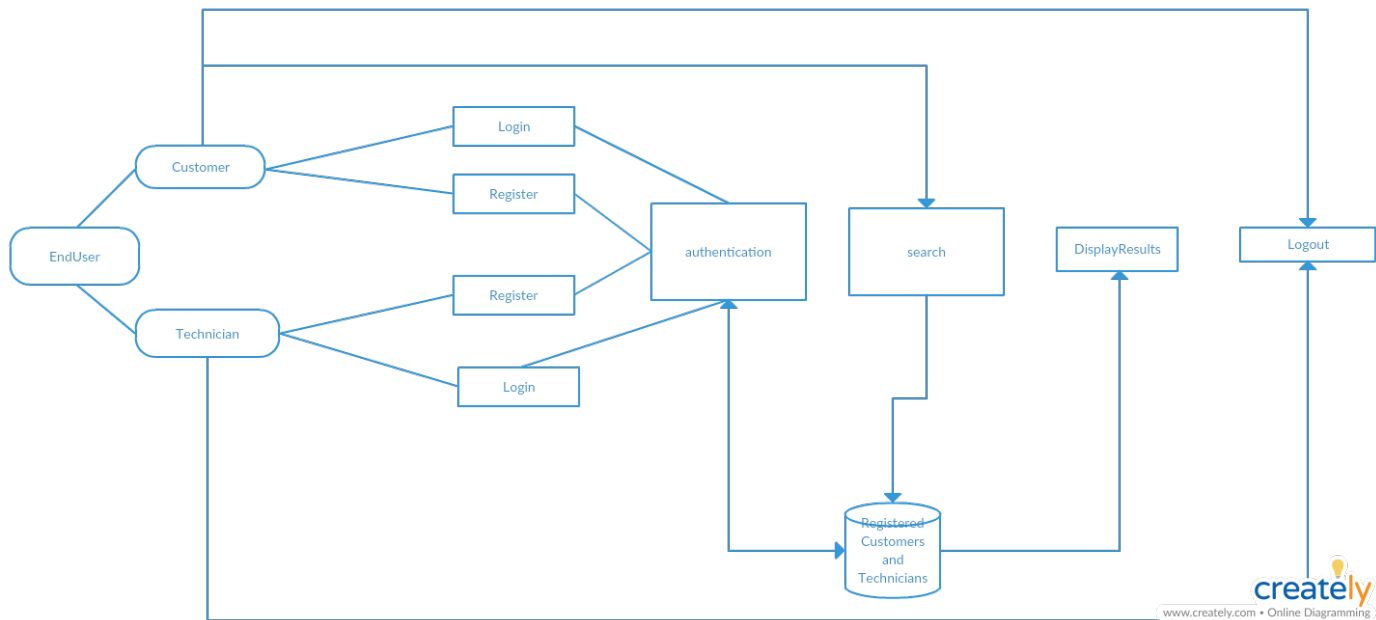
Customers will be able to search for what they are currently needing help with, enter their current location (or other locations that they want using zip code), and opt for the technician that they see fit. There is an option for payment if the problem (service) is clearly identified and the technician 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

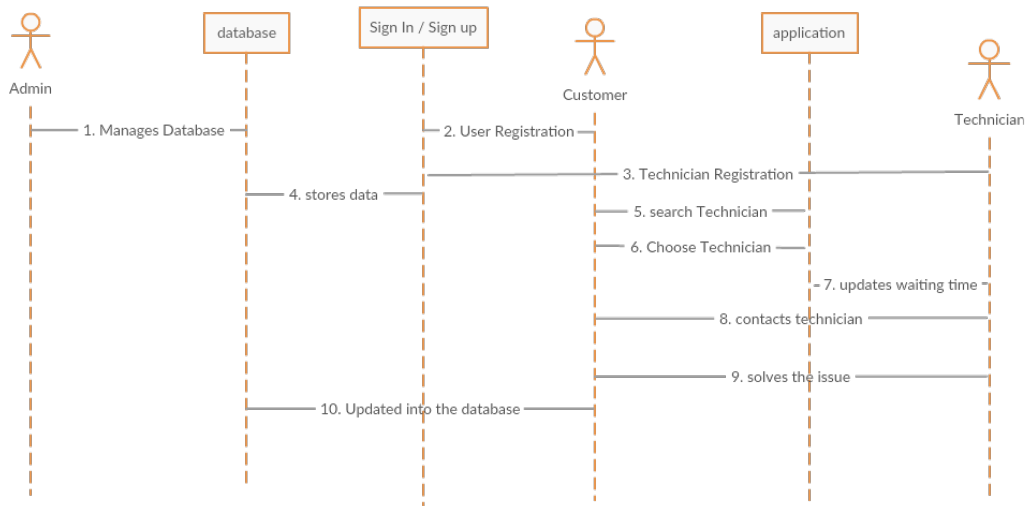
multiple services/ has multiple fields of expertise), what is the current queue is a particular service, and finally set his/her current waiting time for the customers to see. For the waiting time, we may ask the technician to enter a default waiting time per person so that the system can be show the 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:

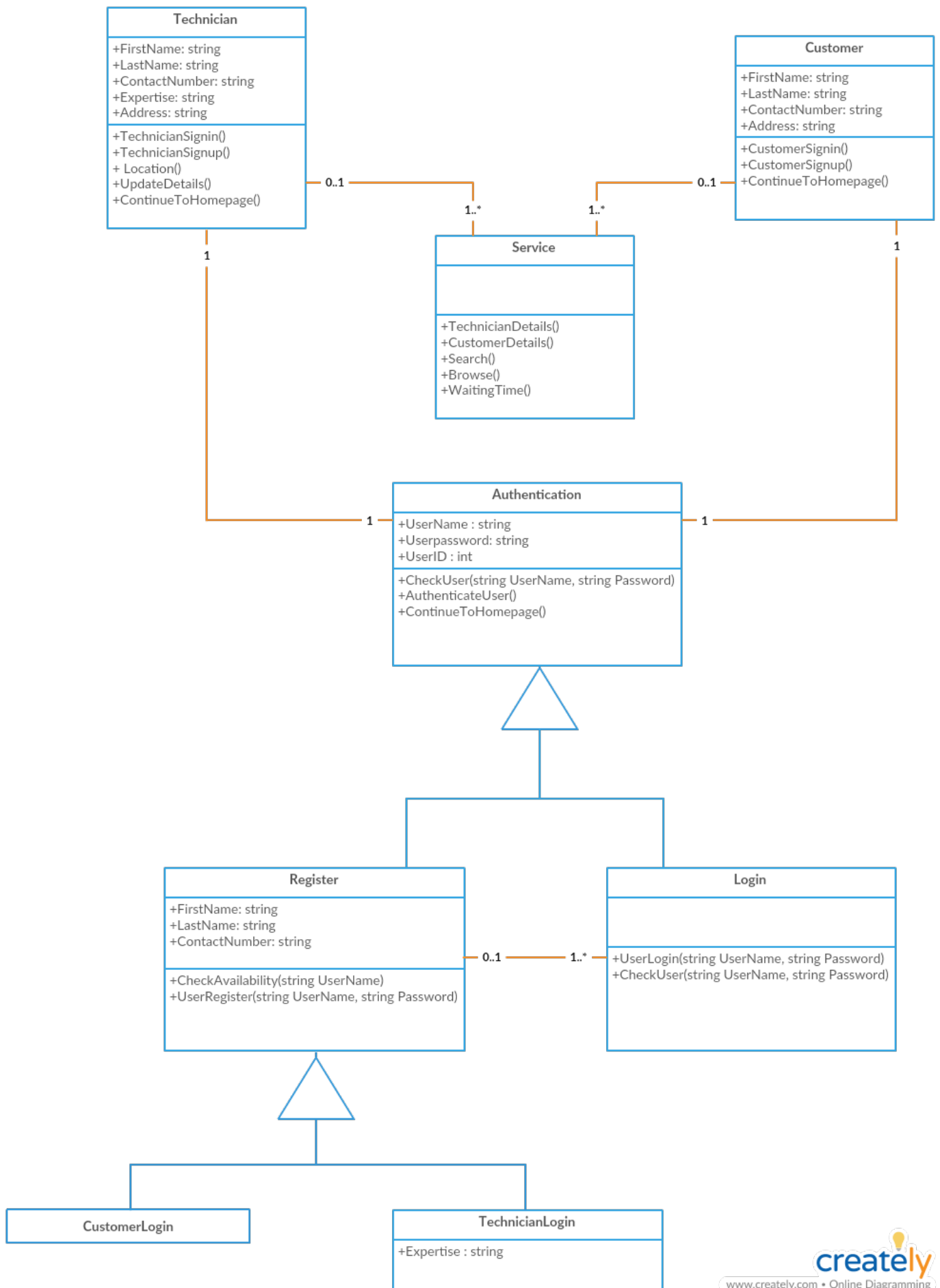
Architecture Diagram:



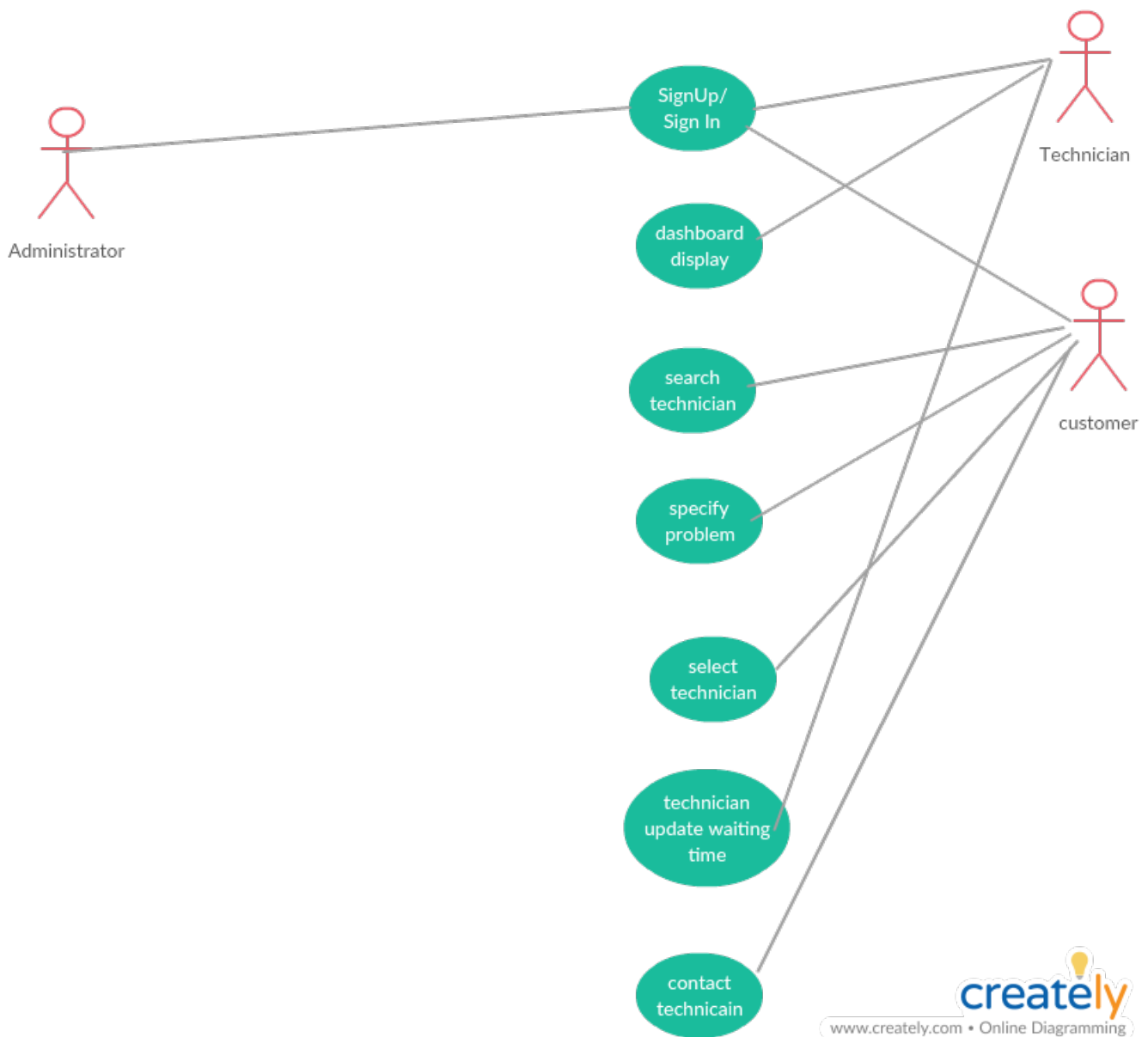
Sequence Diagram:



**Class Diagram:**



**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, new TaskCompleteListener() {
            @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).show();
                }
                //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(user_id);
                    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 OnCompleteListener() {  
            @Override  
            public void onComplete(@NonNull Task<AuthResult> task) {  
                if (!task.isSuccessful()) {  
                    Toast.makeText(context: TechnicianLoginActivity.this, text: "Authentication Failed!", Toast.LENGTH_SHORT).show();  
                }  
            }  
        });  
    }  
});  
});
```

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.

```

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

```

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) && !TextUtils.isEmpty(type)) {
            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.

## fixitupapp

### Users

#### Customers

..... 14IESzf0MIT7avt3GPAkxOA0yox2: true  
..... Mx6PYUeSHlhfZCsD0yqNUR7exn12: true  
..... OyPKsmsspCNTuJjyGoZm5cYqREzv1: true  
..... bw6wrQAqGkRCP5B76Hodh5MmHnV2: true

#### Technicians

|                               |      |   |
|-------------------------------|------|---|
| Ldaj6pffohfCzndiGDiBJJCGyNi2: | true | × |
|-------------------------------|------|---|

..... MNGPSNsANMWOVGhbe7ubqNsaGbP2: true  
..... RJ3H37hHH0Zk5CcKNhaReyypE8I2: true  
..... SR5wEma5mcaXSG3bv53DVA70H1P2: true  
..... ZGEMTuFcEFX5k0NzUVxA32jVOEK2: true  
..... iCTgVgzvEWZEWpZ2EG1HygEmC1e2: true  
..... jywcmZ5tClbP5GuQULk5gkj31BG2: true  
..... vK15xVWISAWzocLSSDe44IAel453: true

## 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:

|   |  |
|---|--|
| 8 Issues - 45 Story Points  |  |
| Closed  |  |
|  CS5551-Team7-Proj... #25<br><b>Sequence Diagram</b><br> Increment 1 Due<br>5     |  CS5551-Team7-Proj... #24<br><b>Architecture Diagram</b><br> Increment 1 Due<br>5                                  |
|  CS5551-Team7-Project #5<br><b>Database Architecture</b><br> Increment 1 Due<br>5 |  CS5551-Team7-Project #12<br><b>Use Case Scenario</b><br> Increment 1 Due<br>1                                     |
|  CS5551-Team7-Proj... #23<br><b>Class Diagram</b><br> Increment 1 Due<br>3        |  CS5551-Team7-Proj... #22<br><b>Design Basic Activities for Technicians and Customers</b><br> Increment 1 Due<br>8 |
|  CS5551-Team7-Project #1<br><b>UI Template</b><br> Increment 1 Due<br>5           |  CS5551-Team7-Project #4<br><b>Database Setup</b><br> Increment 1 Due<br>13  |

#### Responsibility:




























- 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. **\*\***

| Backlog  |   |  |
|--|---|--|
|  CS5551-Team7-Project #8<br> Populate Data<br> Increment 2 Due                          |  CS5551-Team7-Project #9<br> Homepage Implementation (Customer)<br> Increment 2 Due                          |  CS5551-Team7-Project #10<br> Home page design<br> Increment 2 Due                                  |
| 2  | 13  | 21   |
|  CS5551-Team7-Project #11<br> Homepage Implementation (Technician)<br> Increment 2 Due  |  CS5551-Team7-Project #27<br> Research how to implement real-time update of waiting time<br> Increment 2 Due |  CS5551-Team7-Project #15<br> Test Case Design<br> Increment 2 Due                                  |
| 21 enhancement   | 3   | 2  |
|  CS5551-Team7-Project #17<br> Unit Test (Test Cases) Implementation<br> Increment 2 Due |  CS5551-Team7-Project #19<br> Use Case Walkthrough<br> Increment 2 Due                                       |  CS5551-Team7-Project #26<br> Google Map Implementation (Current location only)<br> Increment 2 Due |
| 5 bug  | 5 enhancement   | 8  |

## Bibliography

[Firebase Tutorial](#)

[Android Tutorial](#)

[Google Maps Tutorial](#)

[Android Resources](#)

## Second Increment Report

### Existing Services/ REST API

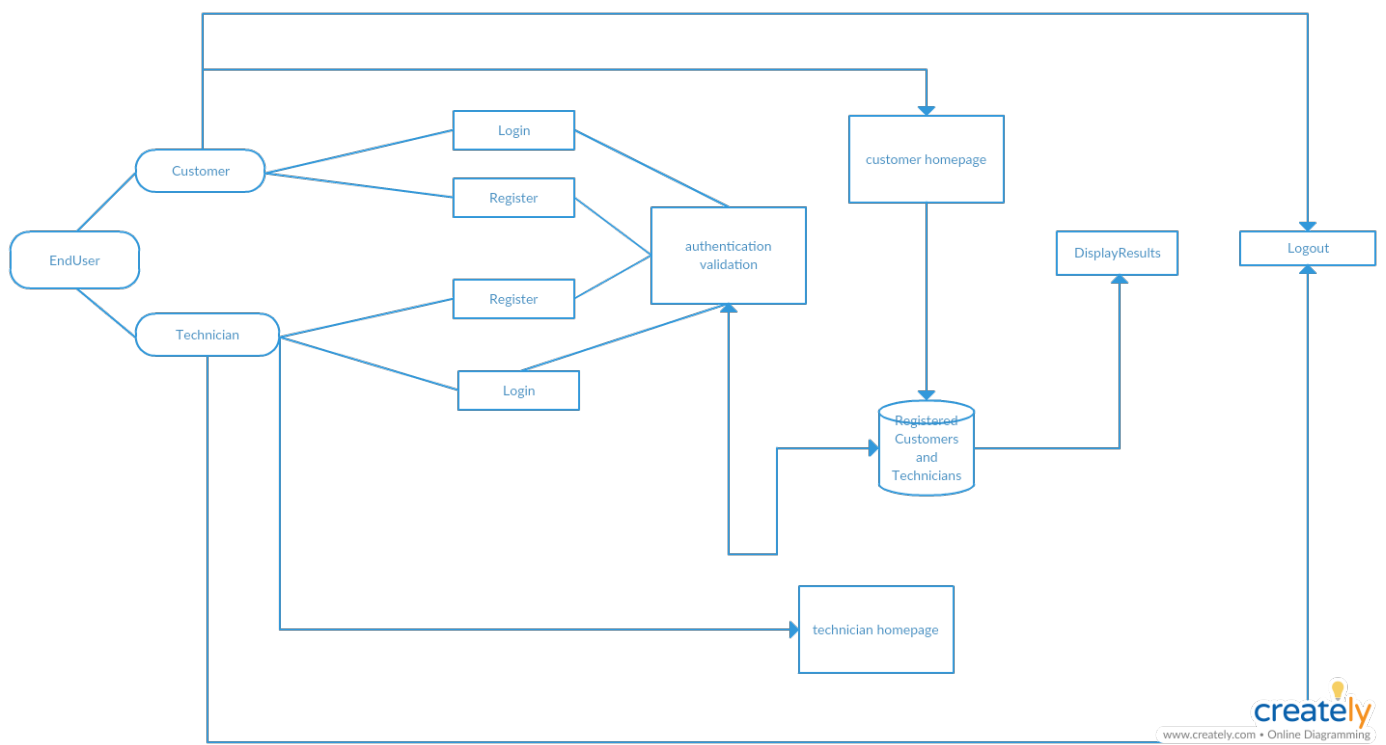
Up to the end of this increment, we are not implementing any REST API (APIs that use get and parameters to retrieve information such as Weather Underground and Nutritionix). However, we are using:

- 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.

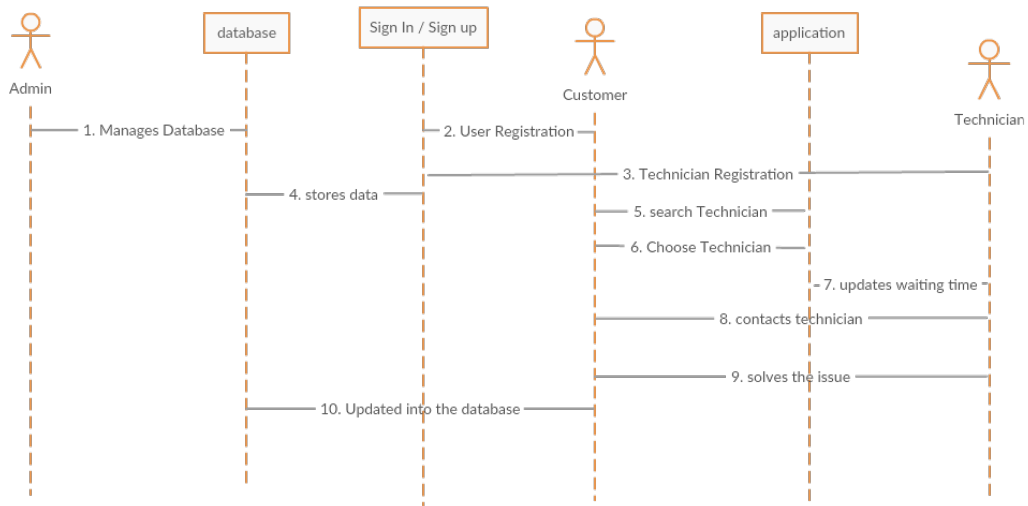
### Detail Design of Features

Here is the four different views of the FixItUp Application:

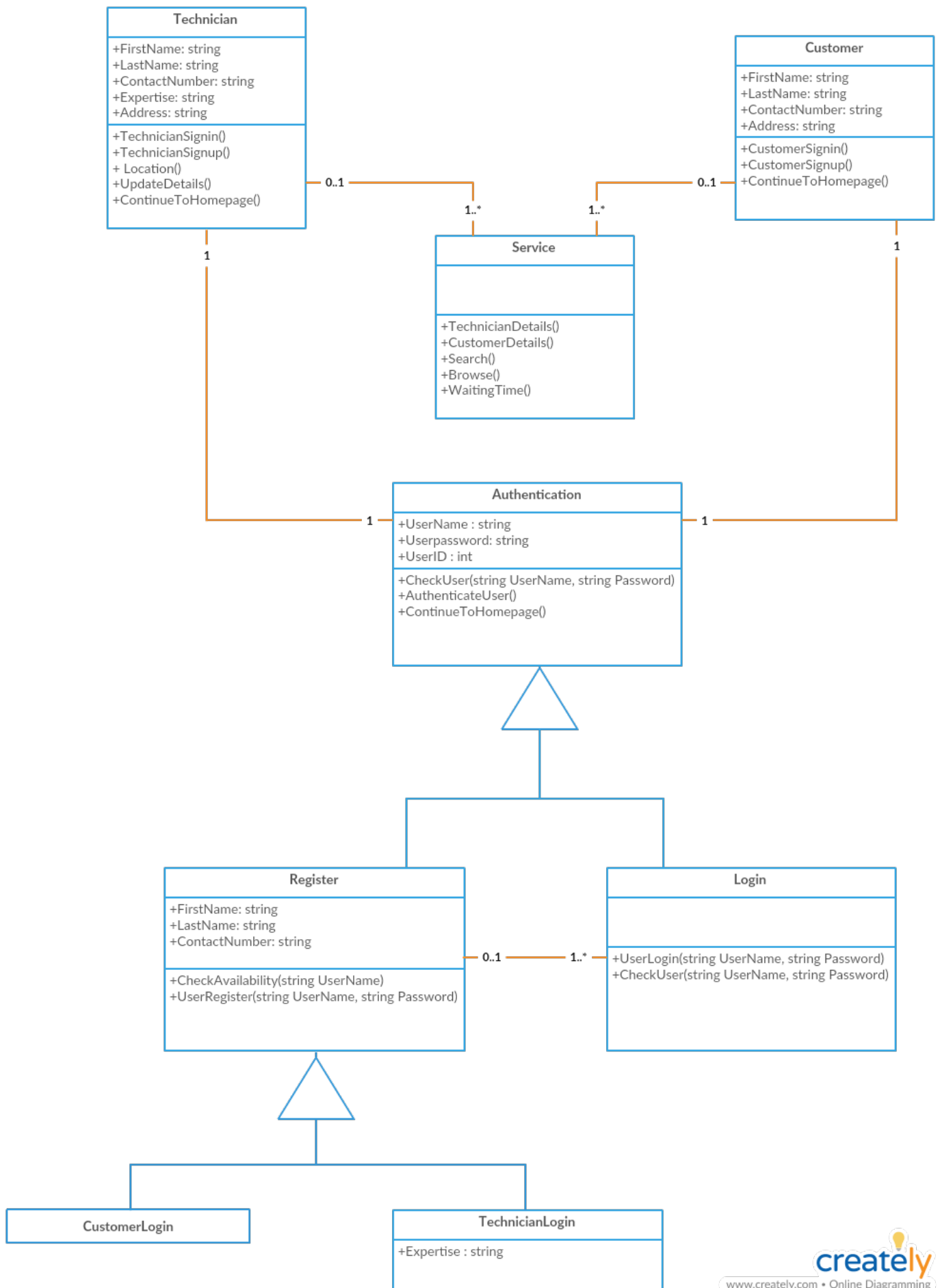
**Architecture Diagram:**



**Sequence Diagram:**

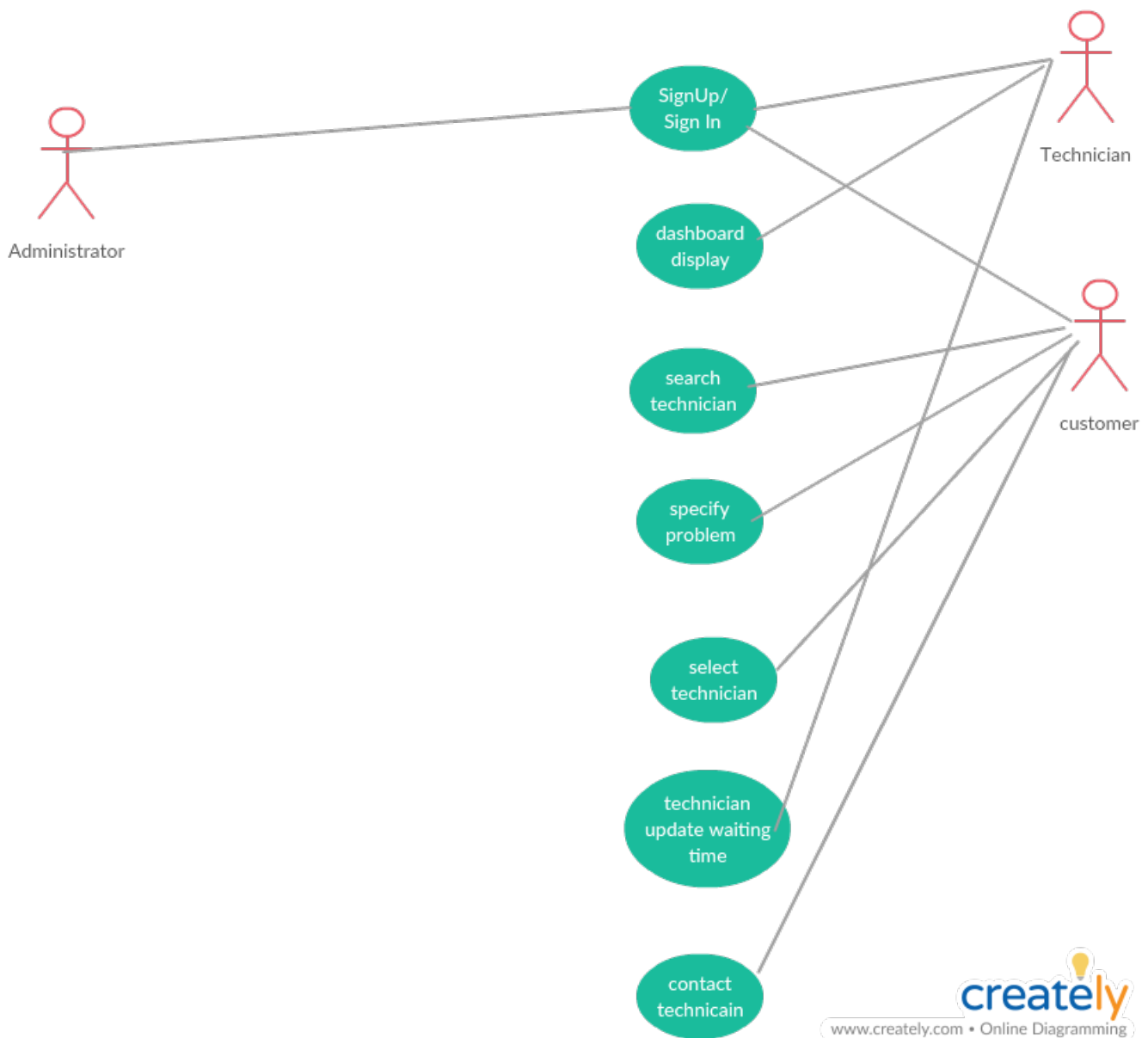


**Class Diagram:**



**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.this)
            .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(mLastKnownLocation.getLatitude(),
                mLastKnownLocation.getLongitude())).title("Current Location"));
            currentLocationMarker.setSnippet("Latitude: " + mLastKnownLocation.getLatitude() + ",
                Longitude: " + mLastKnownLocation.getLongitude());
        }
    }
}
```

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("customerRequest");
        GeoFire geoFire = new GeoFire(ref);
        geoFire.setLocation(userID, new GeoLocation(mLastKnownLocation.getLatitude(), mLastKnownLocation.getLongitude()));
        @Override
        public void onComplete(String key, DatabaseError error) {
            //Do some stuff if you want to
        }
    });
    repairLocation = new LatLng(mLastKnownLocation.getLatitude(), mLastKnownLocation.getLongitude());
    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 permission
    //,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

```

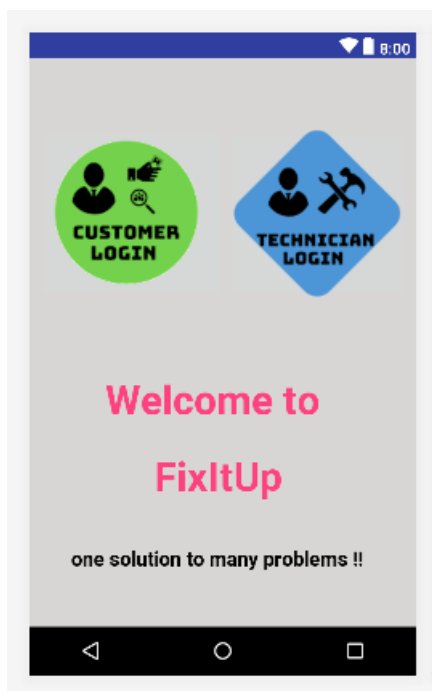
## Deployment:

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.

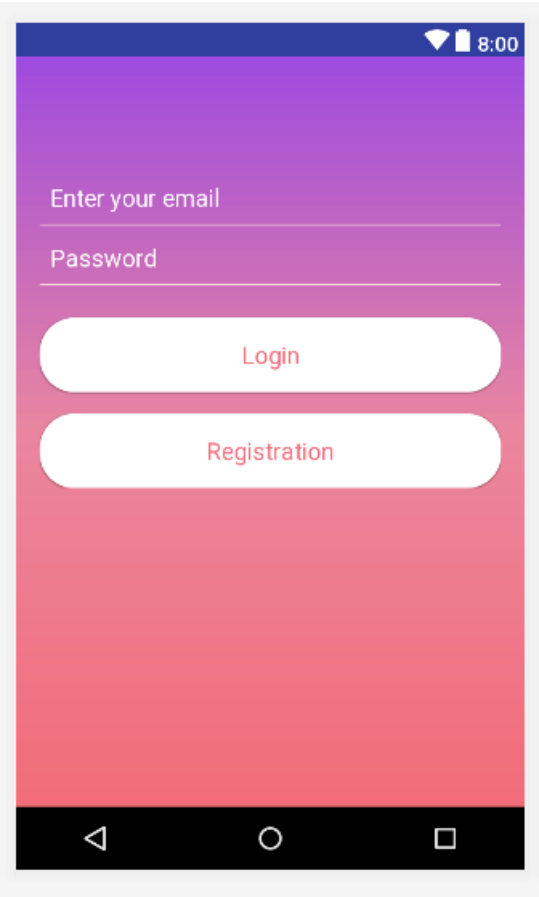
[GitHub Repository URL](#)

[Wiki Link](#)

[Homepage](#)

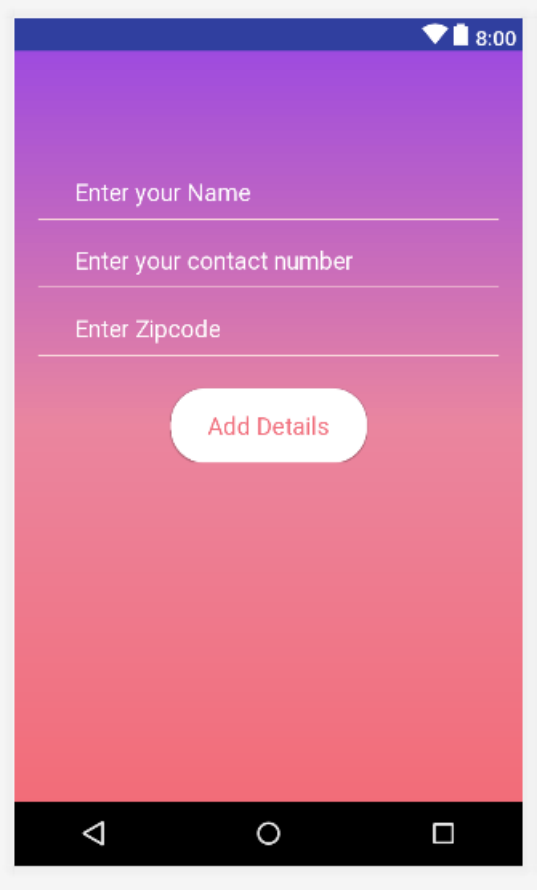


Login page for customer and technician



The image shows a mobile app login screen with a purple-to-pink gradient background. At the top, there is a status bar with a blue header, signal strength, battery, and time (8:00). Below the header, there are two input fields: "Enter your email" and "Password". Below these fields are two buttons: "Login" and "Registration". At the bottom, there is a black navigation bar with three icons: a back arrow, a circle, and a square.

Customer Registration

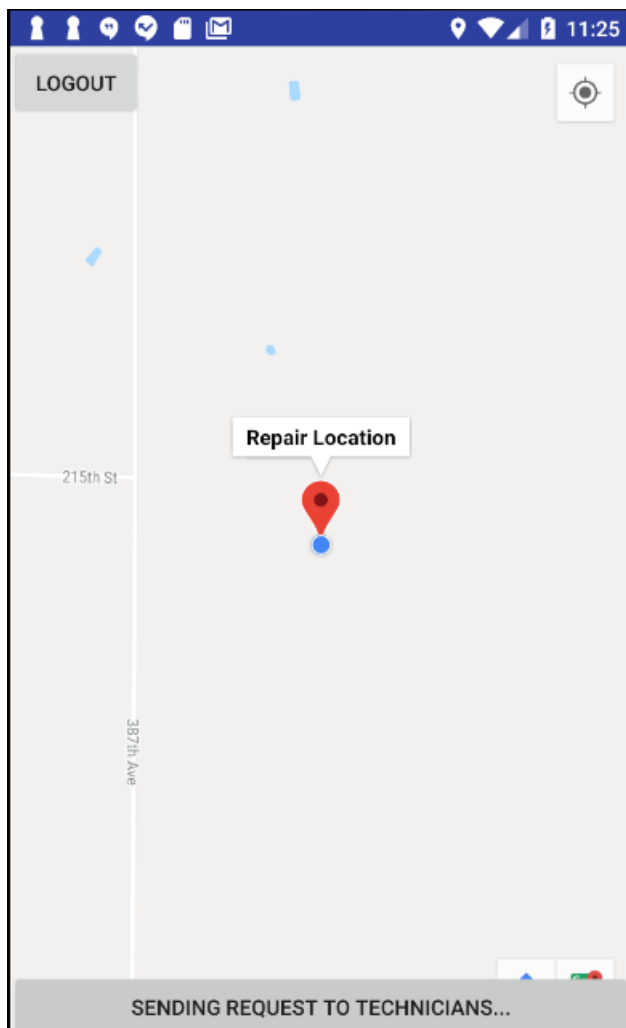


The image shows a mobile app customer registration screen with a purple-to-pink gradient background. At the top, there is a status bar with a blue header, signal strength, battery, and time (8:00). Below the header, there are three input fields: "Enter your Name", "Enter your contact number", and "Enter Zipcode". Below these fields is a button labeled "Add Details". At the bottom, there is a black navigation bar with three icons: a back arrow, a circle, and a square.

Technician Registration

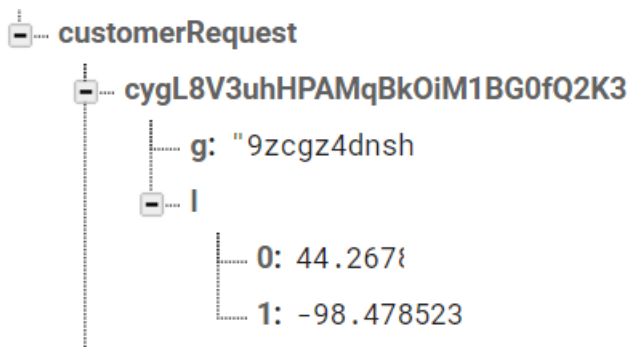
A screenshot of a mobile application interface. At the top, there is a status bar with a blue background showing a Wi-Fi icon, a battery icon, and the time 8:00. The main area has a pink-to-purple gradient background. It contains four white input fields with placeholder text: "Enter your Name", "Enter your contact number", "Enter Zipcode", and "Enter specialization". Below these fields is a white rounded rectangular button with the text "Add Details" in red. At the bottom, there is a black navigation bar with three white icons: a back arrow, a circle, and a square.

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.



## Project Management

---

### Implementation Status Report

#### Work Completed

##### *Description:*

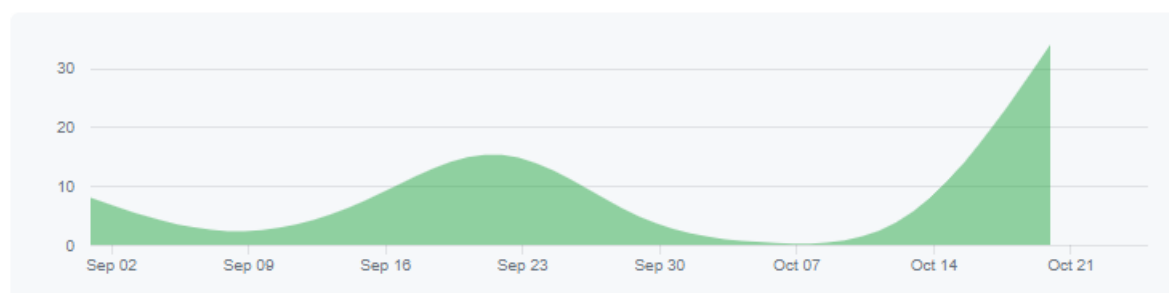
- Map Feature (realtime display, realtime update to database)
- Logout function
- UI layout improvement
- Registration data validation
- Database retrieval in Home page

##### *Responsibility:*

- 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:

Contributions to master, excluding merge commits



## Work to be Completed :

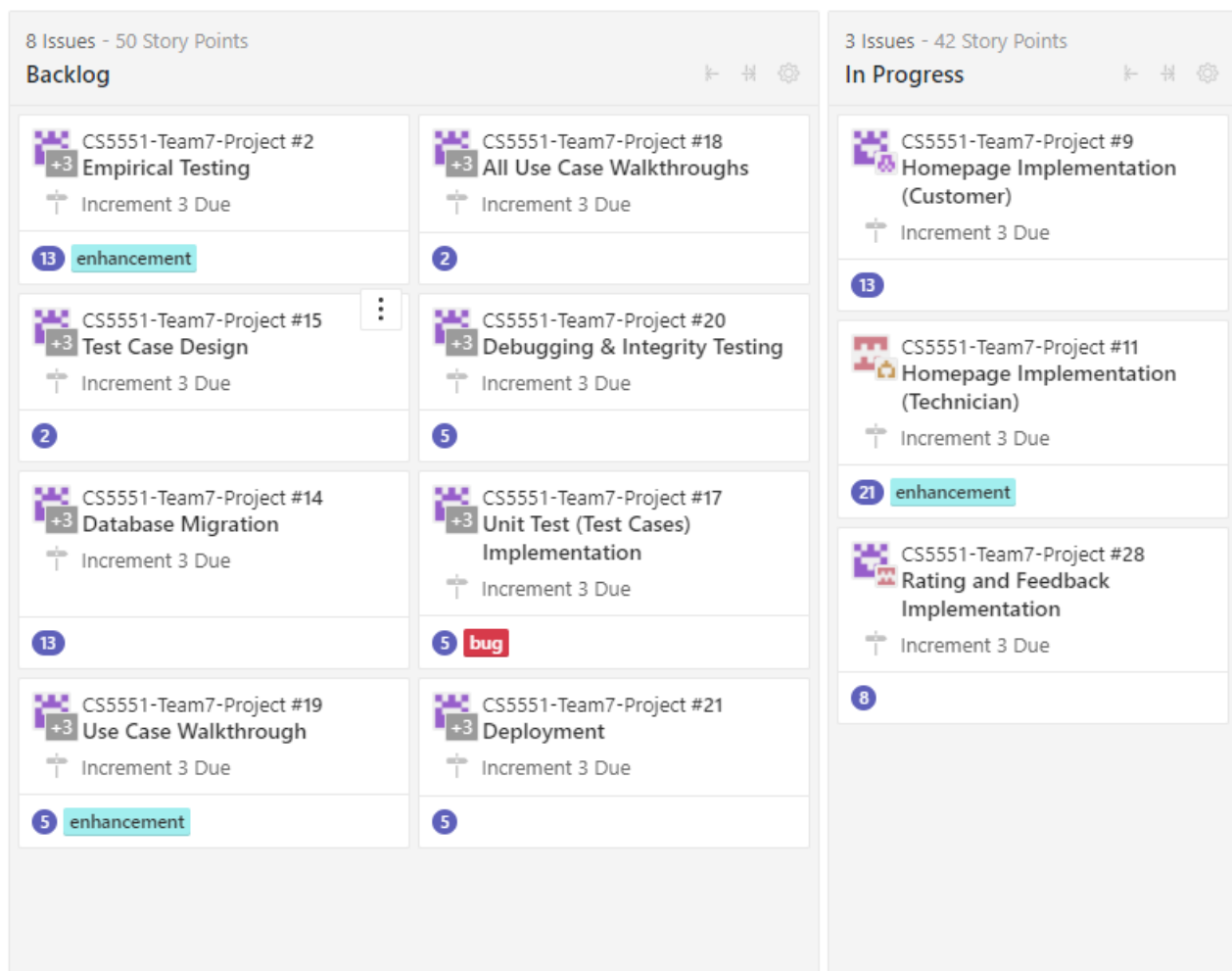
Description:

- 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





### Issues and Concerns:

Since the increment 2 was packed with other deadlines, milestones, and exams, we were not able to implement all the issues in the backlog successfully in terms of vertical implementation, we did manage to set up the backbone horizontally so that more features can be expanded upon that skeleton. One big improvement of this increment is that we were able to use version control more efficiently so that all members in the team could contribute more evenly, leading to more features being added along the way. Increment 3 will be obviously heavy and dense of features and functionality left to accomplish. However, as deadlines and other assignments lessen, we believe we will be able to implement the rest of the project's core features successfully.

## Bibliography

[Firebase Tutorial](#)

[Android Tutorial](#)

[Google Maps Tutorial](#)

[Android Resources](#)

[Youtube Tutorial](#)

[Logo design](#)

[GeoFire](#)

▼ Pages 4

Find a Page...

[Home](#)

[First Increment Report](#)

[Project Plan](#)

[Second Increment Report](#)

+ Add a custom sidebar

Clone this wiki locally

<https://github.com/benamreview/CS5551-Team7-Project.wiki.git>



© 2018 GitHub, Inc.

[Terms](#)

[Privacy](#)

[Security](#)

[Status](#)

[Help](#)

[Contact GitHub](#)

[Pricing](#)

[API](#)

[Training](#)

[Blog](#)

[About](#)