Validation Document

for

Locus

Vete al Infierno
5/03/17

Prepared by: Zachariah Grummons
John Toland
Jessica Hao

Table of Contents

Project Overview

Project Definition

LOCUS is an Android smartphone application that uses the Google location services Application Programming Interface (API) and Google Firebase database allowing users to create groups and populate said groups with other LOCUS users. The application exists in two forms, the application on the user's Android smartphone and the Google Firebase database which houses all the information shared amongst LOCUS users.

Once a group has been created up to ten members can join the group. The following information will help clarify the differences between a user that joins a group and a user that creates a group.

1. **User creates a group**
    a. Group ID displayed as Quick Response Code (QR) and in plain text
    b. This user is the group leader
    c. The leader's location is marked with a green marker on the map
    d. All users that join his/her group will have their locations marked with a blue marker on the map
2. **User joins a group**
    a. Joins group by scanning QR Code on leader's screen, or manual input of group ID via touchscreen
    b. This user is a member of the group
    c. This user's location is blue on the leader's map
    d. This user's location is orange on his/her map
    e. All other members in the group will have their locations marked with a blue marker on the map

LOCUS offers users a manifesto (members list page) where they can find a list of all the members in the group. Along with the information of each user: username, user status (Leader or Member), and last seen time (the last time and date the user's location was updated).

The safe zone is a metric set by the group leader upon creation of the group. This metric is measured in meters. Limited in size, the user can choose a safe zone ranging from fifteen to fifty meters. The purpose of the safe zone is to notify the group leader when a member's location is greater than the metric set.

For example, a child is supposed to be in eyesight of his/her parent but has wondered off chasing butterflies. The parent can no longer see the child and failed to notice the distance growing between him/her and the child. LOCUS would notify the parent that his/her child is now further away than intended to be safe. The parent now knows it is time to find the child before it is too late.

Members will also be notified that they are moving too far from the leader and should return to his/her location. The notifications are sent via push notifications to the user's Android smartphone every thirty seconds whether the user is a member or leader.

The application will also allow members to leave their group at any time, allowing them to join a different group. Leaders can delete their group at any time, which will remove all the members from the group allowing all users to join and/or create a new group. All users of the LOCUS application are only allowed to exist in one group at a time (users cannot create or join more than one group at the same time).

Project Motivation

LOCUS is intended to help teachers and chaperones keep track of their students when on class trips to points of educational interest i.e. historical monuments, national parks, museums, etc. The primary motivation for the project is to limit the occurrences of accidents on these trips. The application is intended to keep the teacher notified and aware of what is going on around him/her while requiring little user interaction. The teacher can place LOCUS in his/her pocket and continue with the activity. If the application is running it will keep track of all the member's locations for the teacher.

Original design allowed for automatic removal of members from the group at end of the business day. This feature keeps the student's location private when he/she is no longer under the supervision of the school system.

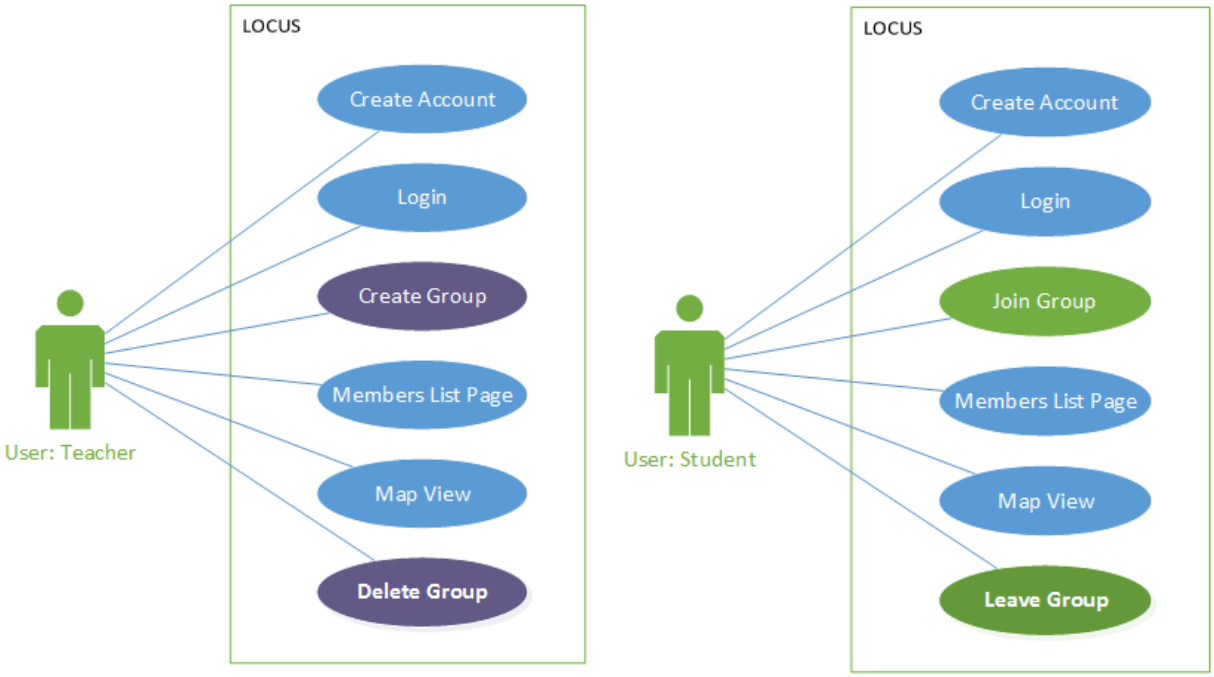
Design Details

The following information will be represented in Unified Modeling Language (UML). This information is intended to provide the reader with a visual representation of the LOCUS application's functionalities.

1. Use Case Diagrams

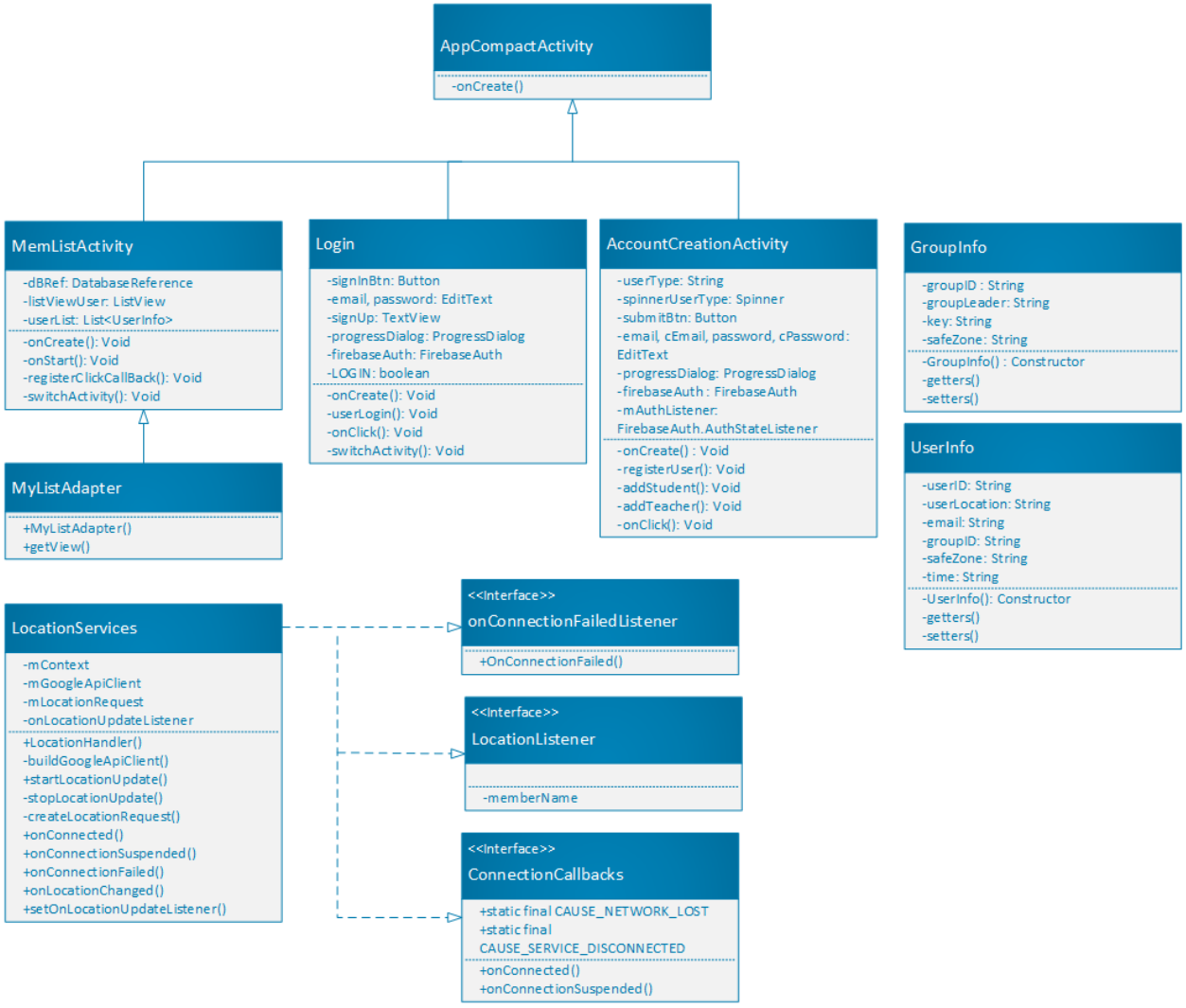
The following diagrams represent the different scenarios of user interaction.

1. Register account: All users
2. Login to account: All users
3. Create Group: Group Leader
4. Join Group: Group Member
5. Member list view: All users
6. Map view: All users
7. Leave Group: Group Member
8. Delete Group: Group Leader



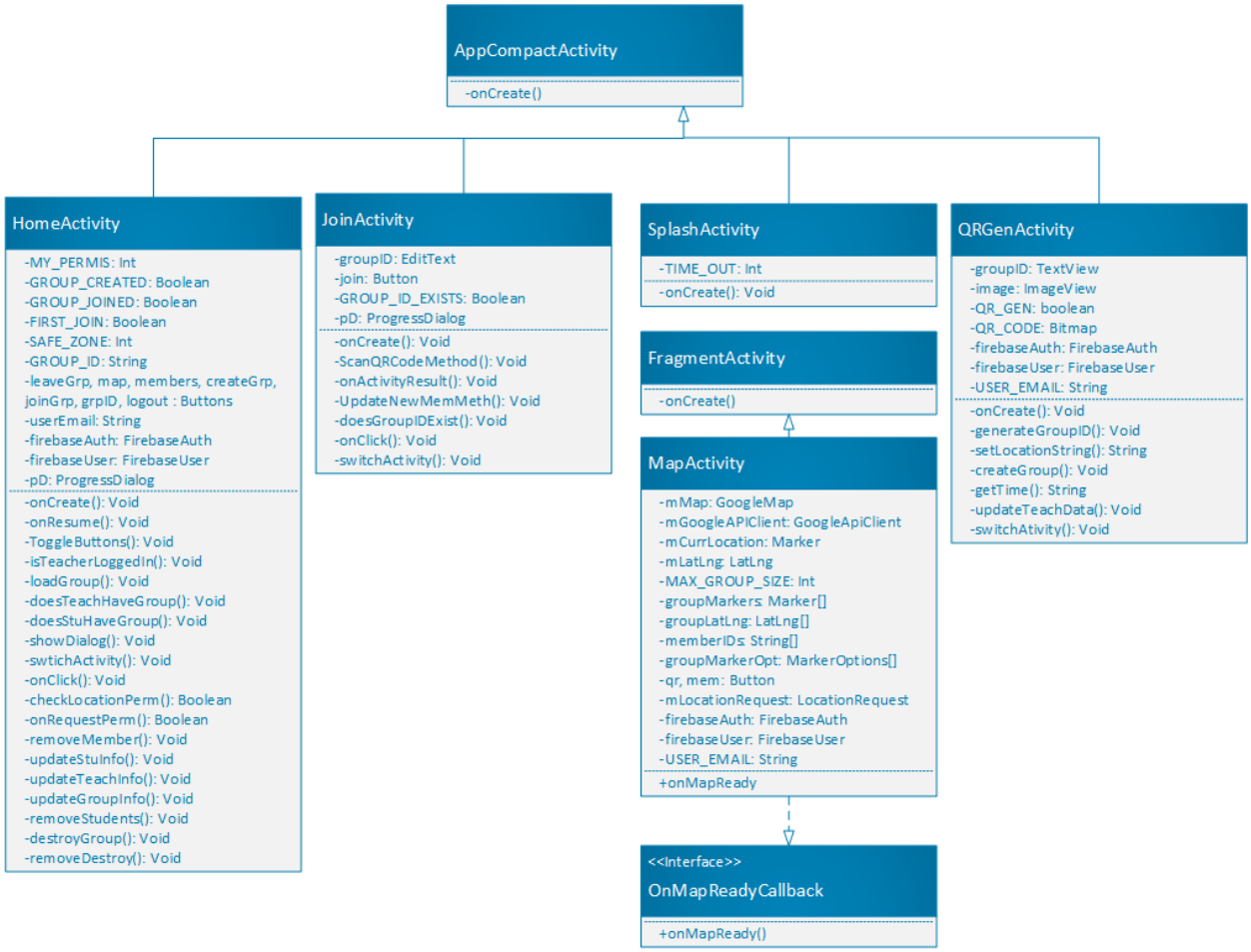
2. Class Diagrams

The following diagrams represent the activities and classes used to achieve LOCUS.



The following is a brief overview of the class diagram above:

1. All activities inherit from AppCompactActivity
2. MyListAdapter inherits from MemListActivity
3. Location Services is used to realize the respective interfaces
    a. onConnectedFailedListener: connection to Google API failed
    b. LocationListener: allows for current location updates
    c. ConnectionCallbacks: listener for callbacks from the API
4. LoginActivity allow user to login to the LOCUS application using a password and email address
5. AccountCreationActivity allows the user to create an account on the LOCUS database using a password and email address
6. GroupInfo and UserInfo are Java classes used to convert the JSON string into object

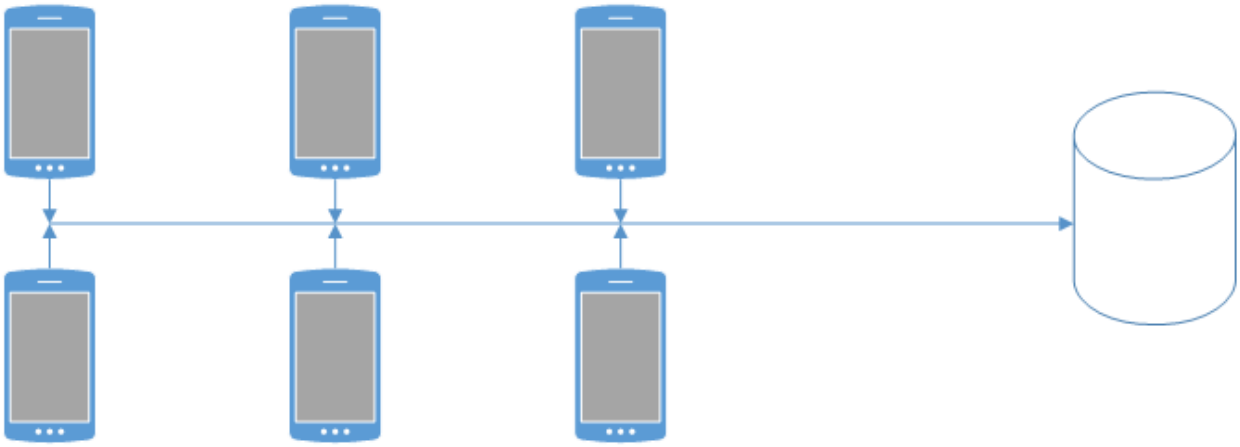


The following is a brief overview of the class diagram above:

1. All activities inherit from AppCompactActivity
2. HomeActivity is the user's home screen allowing for navigation within the application
3. JoinActivity is where the user can join a group
4. SplashActivity is the welcome screen when LOCUS is launched
5. MapActivity inherits from FragmentActivity
6. MapActivity realizes the OnMapReadyCallback interface
    a. MapActivity: user's map view
    b. OnMapReadyCallback: displays the Google map
7. QRGenActivity is where the user can display the GroupID as QR Code and in plain text

3. Design Architecture

LOCUS shall exist in two forms, a smartphone application and the database that the applications send and retrieve information from. It shall exist as an n-tier client-server architecture with a many-to-one smartphone-database relationship.

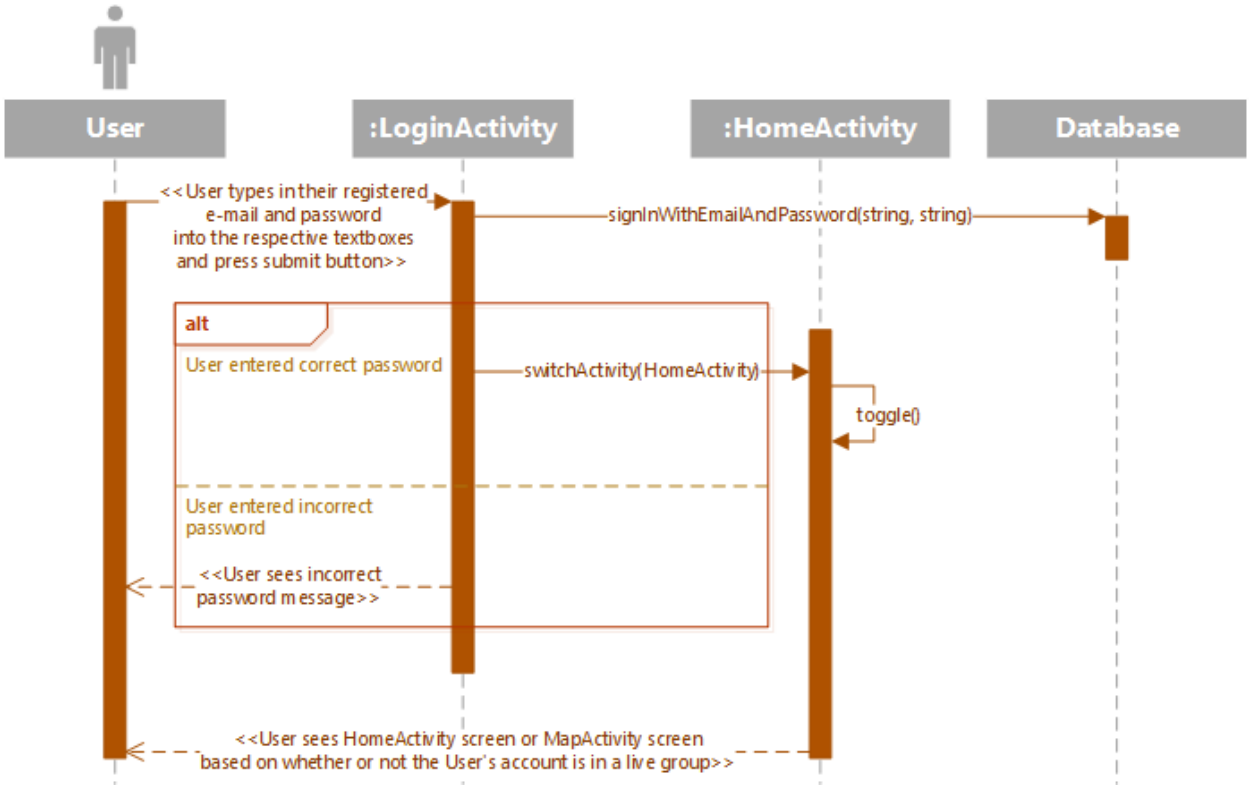


Sequence diagrams on next page…

4. Sequence Diagrams

The following diagrams represent the sequence of events that occur within LOCUS during user interaction.

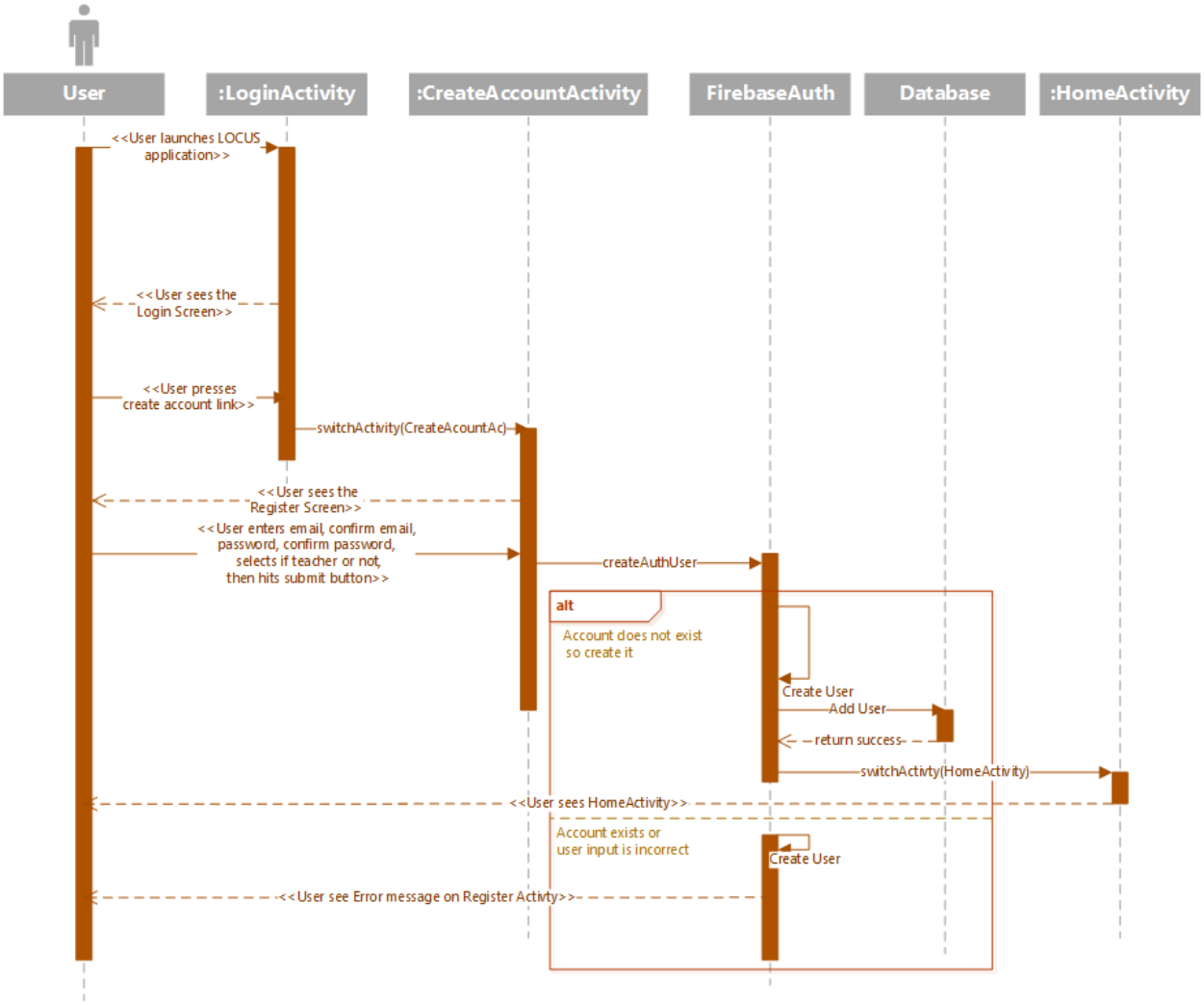
4.1. Login



The following is a brief description of the **Login** sequence diagram:

1. User enters credentials and presses submit button
    a. If credentials are correct → display home activity and toggle buttons
    b. Else if credentials are not correct → display error message
2. User is logged into database
3. Home screen is displayed to user (HomeActivity)

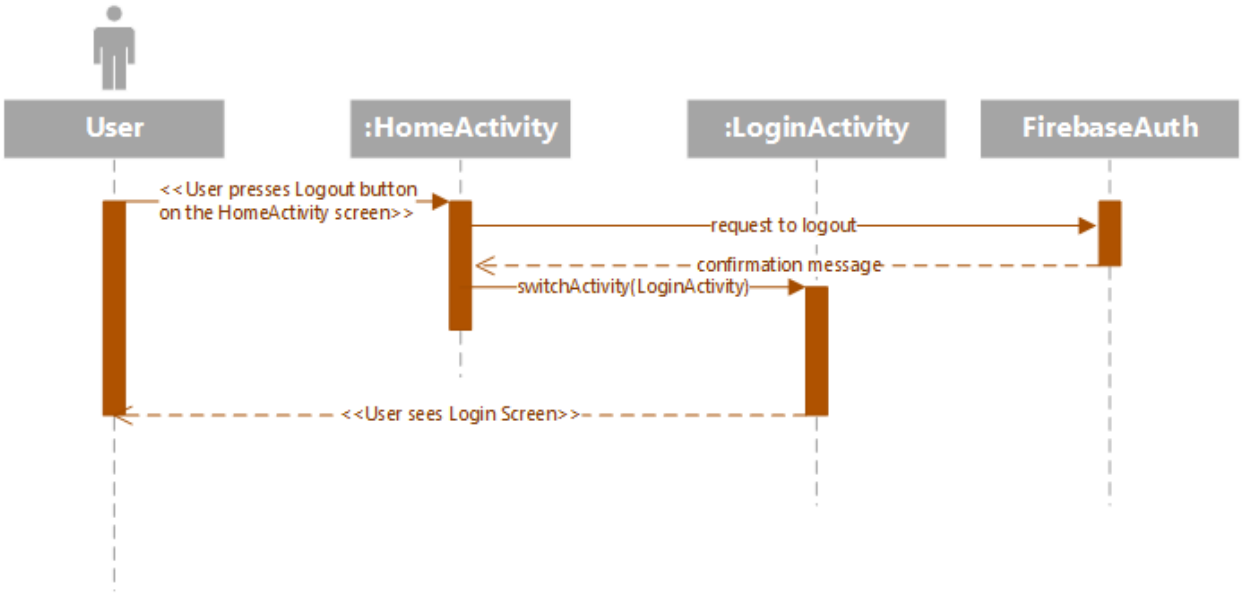
4.2. Register



The following is a brief description of the **Register** sequence diagram:

1. User launches LOCUS
2. Login screen is displayed → user presses the create account link
3. LOCUS switches the activity and displays the register screen to the user
4. User enters their intended credentials and presses submit button
5. LOCUS attempts to add the user to the Firebase Authorization list
    a. If the account does not exist → create the account and switch to HomeActivity
    b. Else if the account exists or input is inconsistent → display error message to user
6. User sees the home screen or user sees the error message on register screen

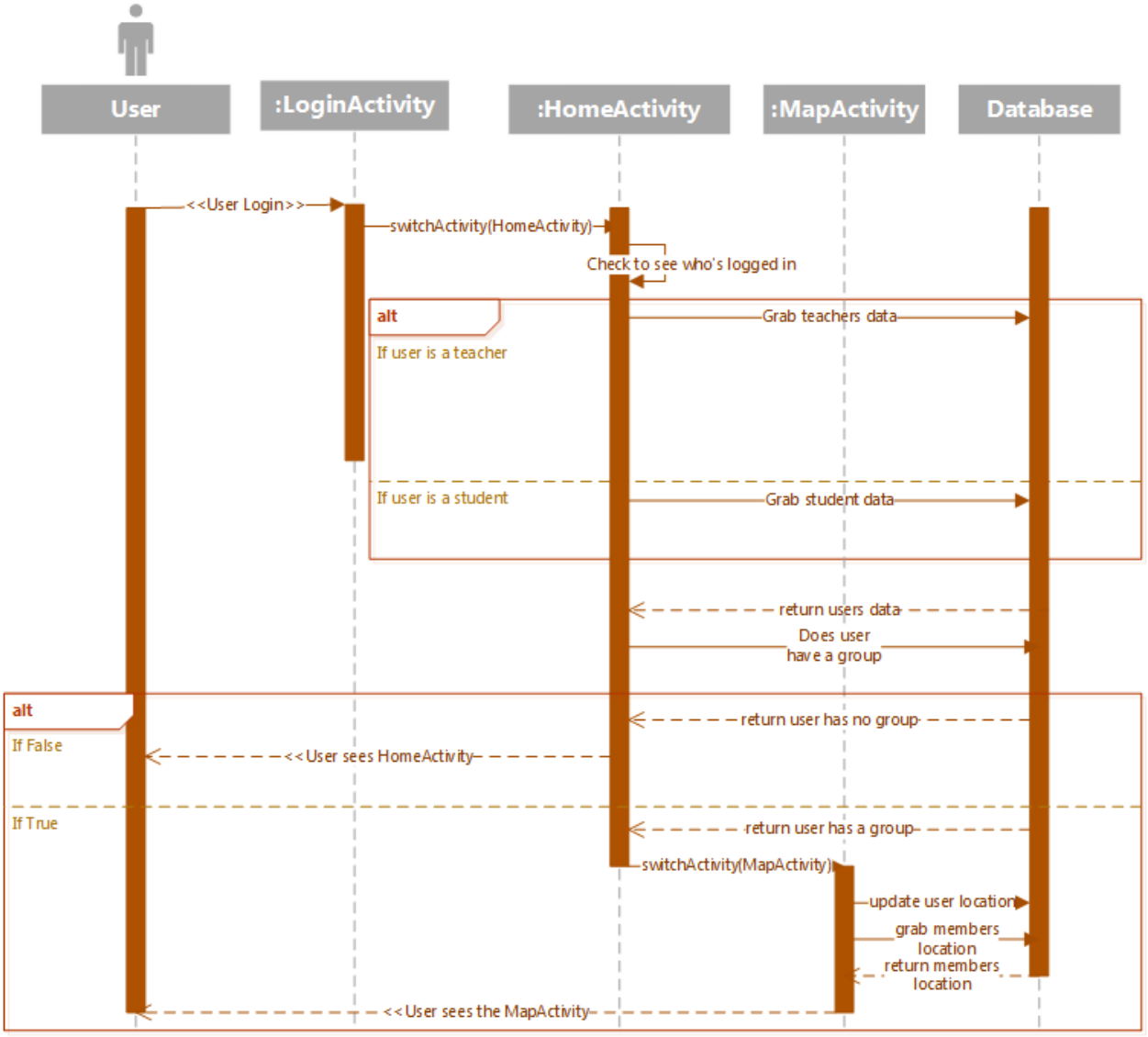
4.3. Logout



The following is a brief description of the **Logout** sequence diagram:

1. User presses the logout button on the HomeActivity screen (home screen)
2. Request sent to Firebase Authorization to log the user out of database
3. Firebase Authorization sends confirmation message back to LOCUS
4. Locus switches to LoginActivity (login screen)
5. User sees the login screen

4.4. Launch



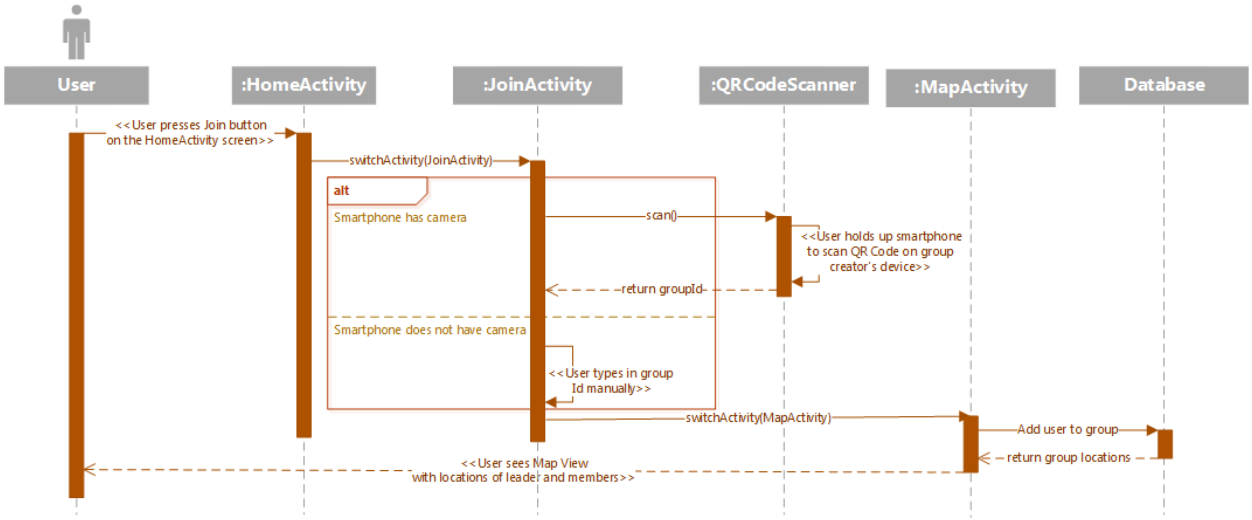
The following is a brief description of the **Launch** sequence diagram (extension of Login sequence diagram):

1. User logs into LOCUS → LOCUS switches to HomeActivity
2. LOCUS checks to see who is logged in
    a. If teacher → grab teacher's data from database
    b. Else if student → grab student's data from database
3. Database returns the user's data to LOCUS
4. LOCUS checks to see if user has a group
    a. If false → return user has no group and display HomeActivity
    b. If true → LOCUS switches to MapActivity
        i. LOCUS updates user's current location to database

ii. LOCUS requests all the members' locations from the database
iii. Database returns locations

5. User sees the MapActivity (mapview) or the HomeActivity (home screen)

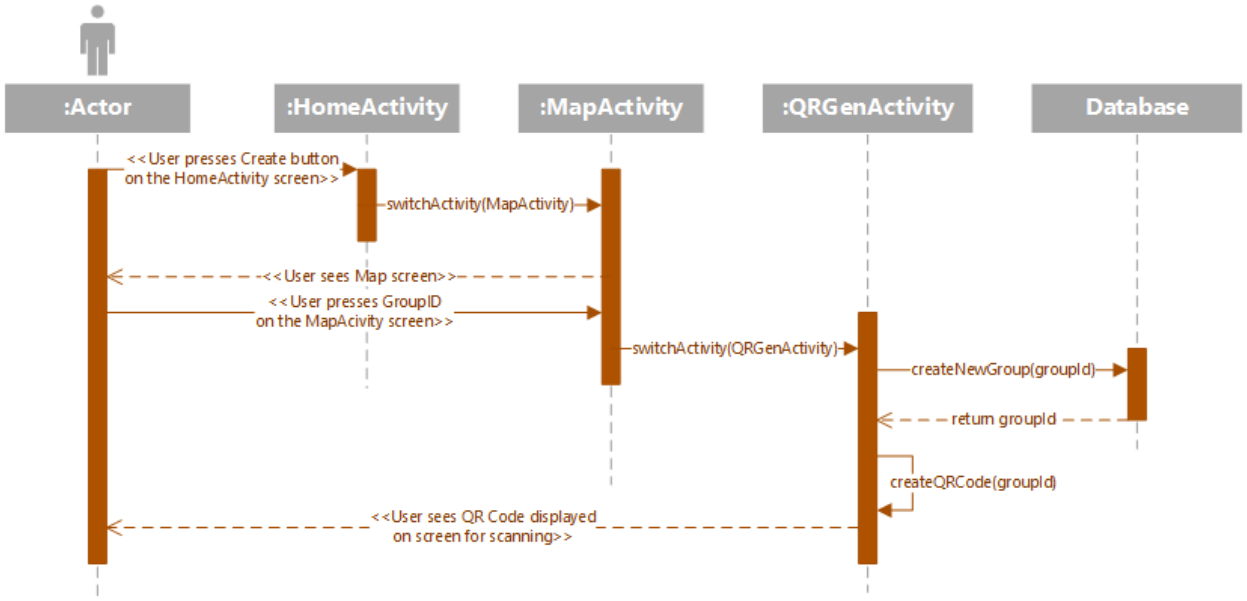
4.5. Join



The following is a brief description of the **Join** sequence diagram:

1. User presses Join button on home screen
2. LOCUS switches to the JoinActivity (join screen)
3. User joins group
    a. Option 1: scan QR Code
    b. Option 2: enter group ID manually with touch screen
4. LOCUS switches to MapActivity (mapview) → MapActivity adds user and location to database
5. Database returns the group's locations to LOCUS
6. MapActivity with leader's location and members' location are displayed

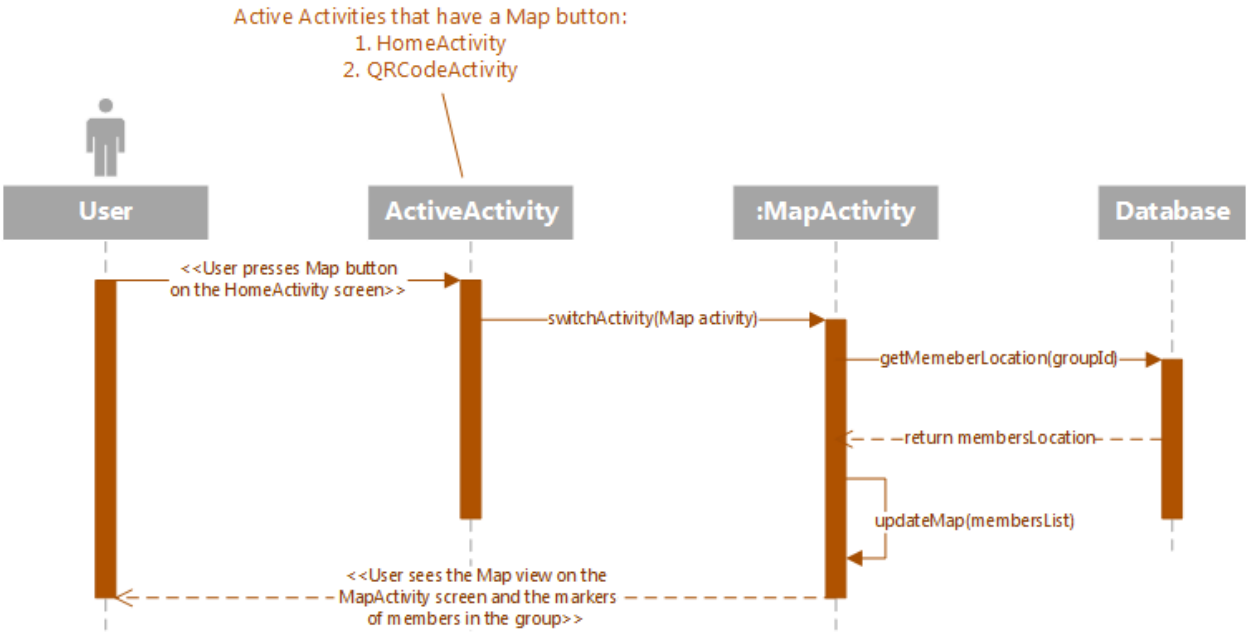
4.6. Create



The following is a brief description of the **Create** sequence diagram:

1. User presses the create button on the home screen
2. LOCUS switches to the MapActivity → User sees MapActivity (mapview)
3. User presses Group ID button on mapview
4. LOCUS switches to QRGenActivity (QR Code view) → group is created on database
5. Database returns the group ID → LOCUS generates the QR Code
6. User sees the QR Code displayed along with group ID in plain text

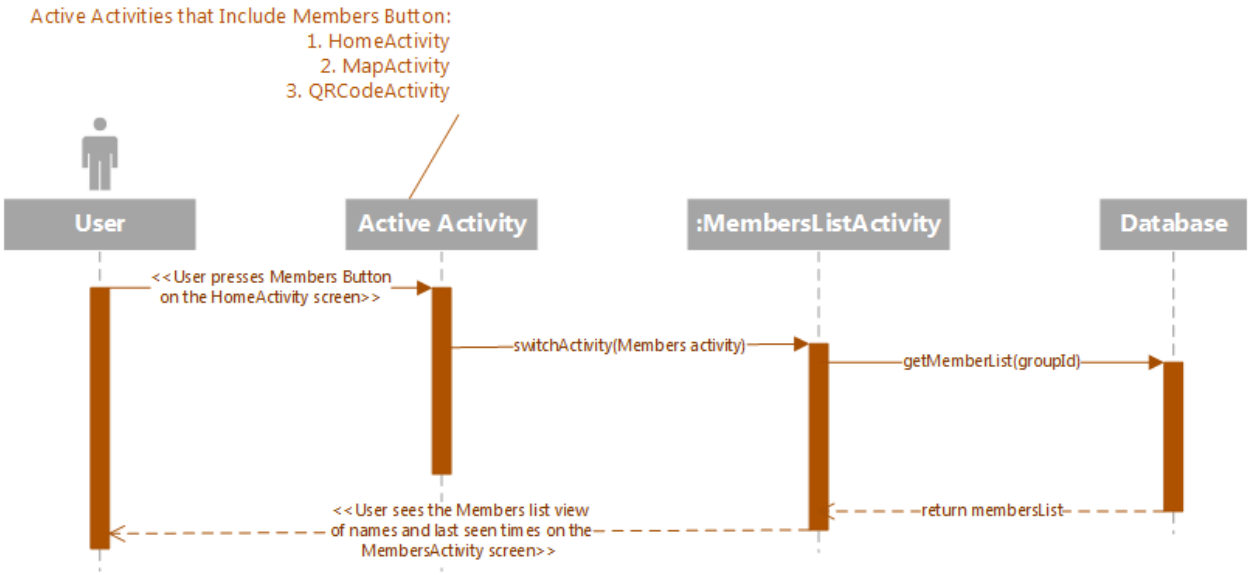
4.7. Map



The following is a brief description of the **Map** sequence diagram:

1. User presses Map button on active activity
2. LOCUS switches to MapActivity (mapview)
3. LOCUS gets the member's location and updates current location
4. Database returns members locations
5. LOCUS updates the mapview with new locations
6. User sees the mapview with updated locations

4.8. Members



The following is a brief description of the **Members** sequence diagram:

1. User presses the members button on the active activity
2. LOCUS switches to the MembersListActivity
3. LOCUS gets the list of members from the database
4. Database returns list of members
5. User sees the members list view with update information (last seen time and username)

Implementation Details

Implementation Overview

The process of implementation occurred in the following order:

1. User Interface (UI)
    a. ToggleButton()
    b. ShowDialog()
    c. ShowConfirmation()
    d. SwitchActivity()
    e. ShowButton()
    f. ShowProgress()
2. Map Activity
    a. checkLocationPermission()
    b. onRequestPermissionsResult()
    c. buildGoogleAPIClient()
    d. onMapReady()
    e. onLocationChanged()
3. Firebase authentication: User Info Class
    a. Login
        i. userLogin()
    b. Register
        i. registerUser()
        ii. addStudent()
        iii. addTeacher()
4. Generate QR Code
    a. generateGroupID()
5. Scan QR Code
    a. scanQRCodeMethod()
6. Firebase database: Group Info Class
    a. Creating groups
        i. showSafeZoneDialog()
        ii. createGroup()
    b. Joining groups
        i. doesGroupIDExist()
        ii. joinGroup()
    c. Deleting or leaving a group
        i. removeMember()
        ii. destroyGroup()
    d. Updating the user's location
        i. updateMembersLocationToDB()
        ii. removeMarkers()
        iii. grabGroupMembersLocationFromDB()
        iv. updateMarkers()

        v. checkDistanceFromMembers()
    e. Notification of user outside the safe zone
        i. PushNotification()
7. Displaying list of members and relevant information
    a. onStart()
        i. onDataChange()

The following information will provide a clear description of each step during implementation:

User Interface

The Locus user interface was the first phase of the implementation of the overall project, without a user interface it is difficulty to apply the more complex features i.e. database query, and displaying the current location of the user on the map, etc.

Implementation Techniques:

1. Drawing sketches of the user interface to decide on button location
2. Designing the flow of activities (how the user navigates through the app)
    a. Question in consideration
        i. What order will the user view the screens?
        ii. What buttons will navigate to which screens?
        iii. When will certain buttons be active and inactive?

Figures on next page...

User Interface Figures

Fig 1.1 Splash Screen

Fig 1.2 User Login Screen

Fig 1.3 User Registration Screen

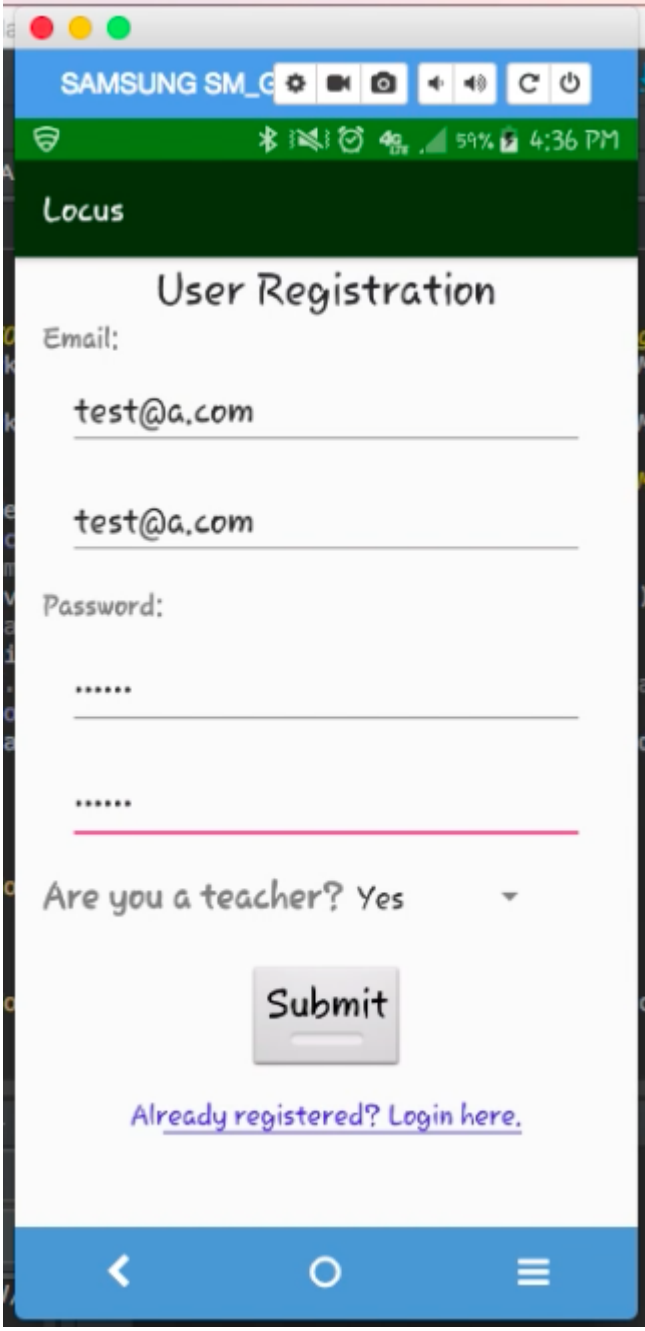


Fig 1.4 Registration Dialog

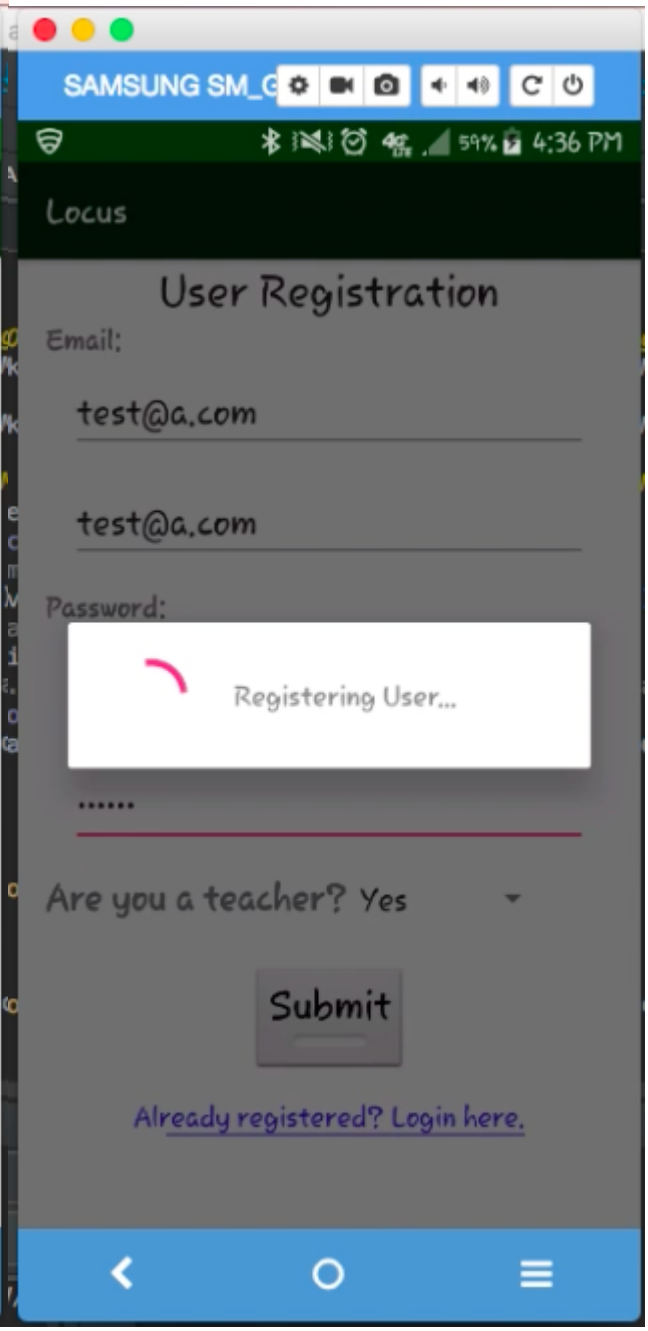


Fig 1.5 User Login Screen with Login Dialog **Fig 1.6 Group Found Dialog**

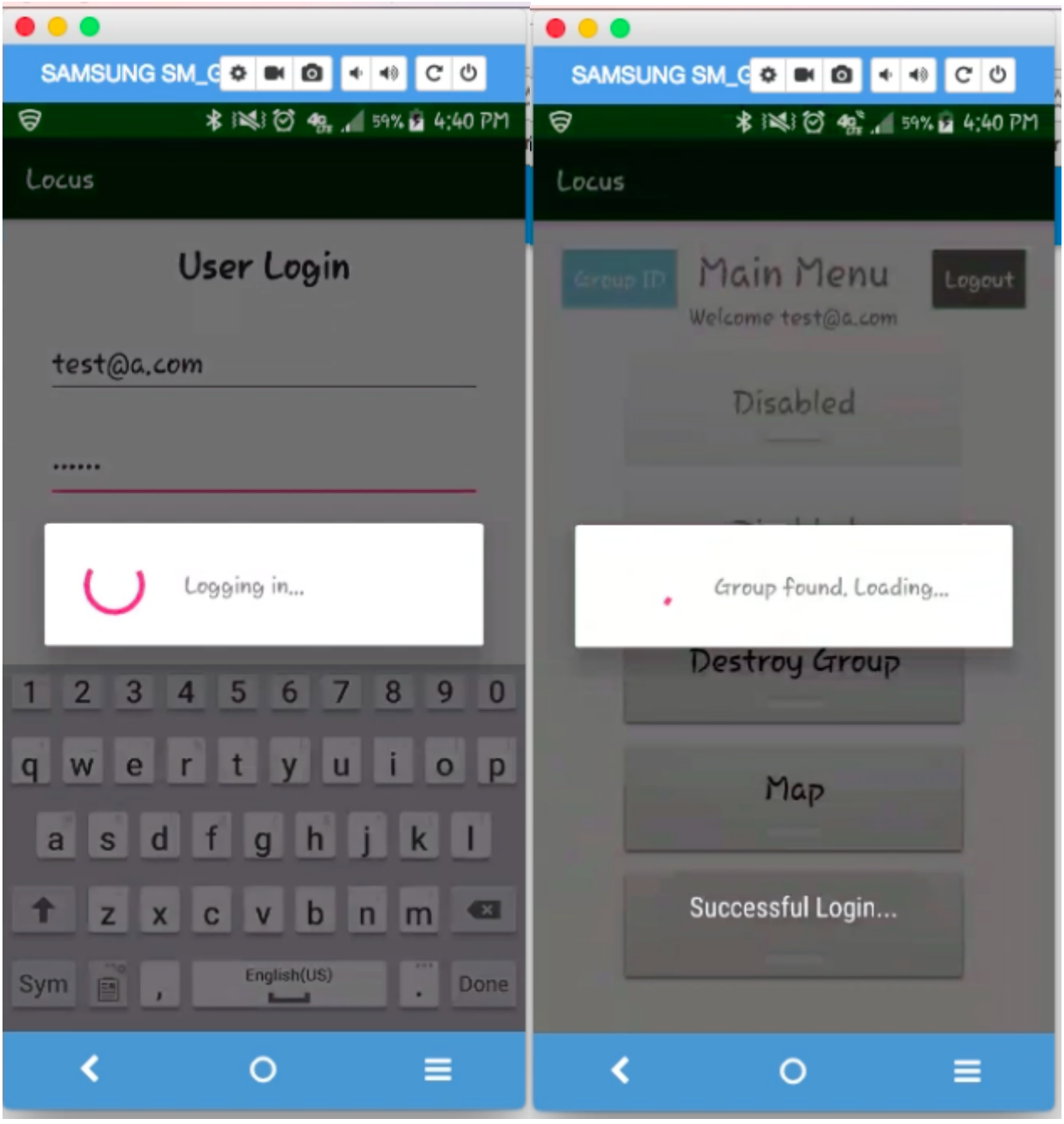


Fig 1.7
Teacher Home Screen (before group created)

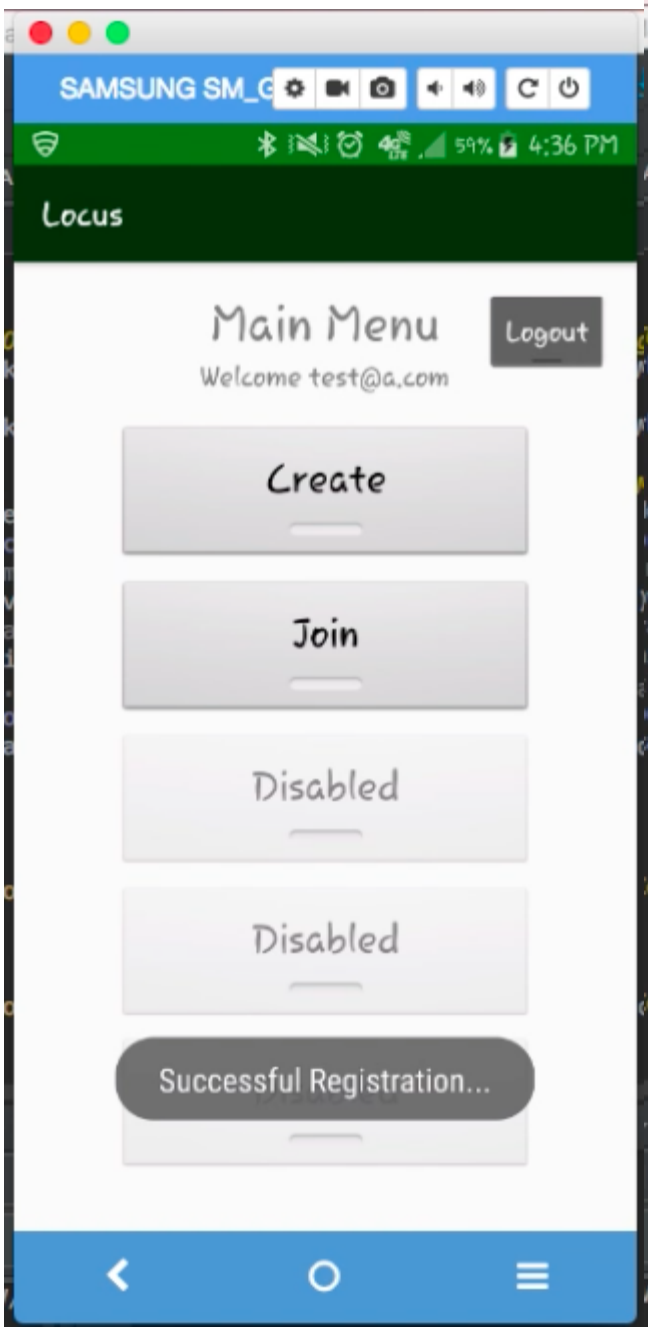


Fig 1.8
Student Home Screen (before group joined)

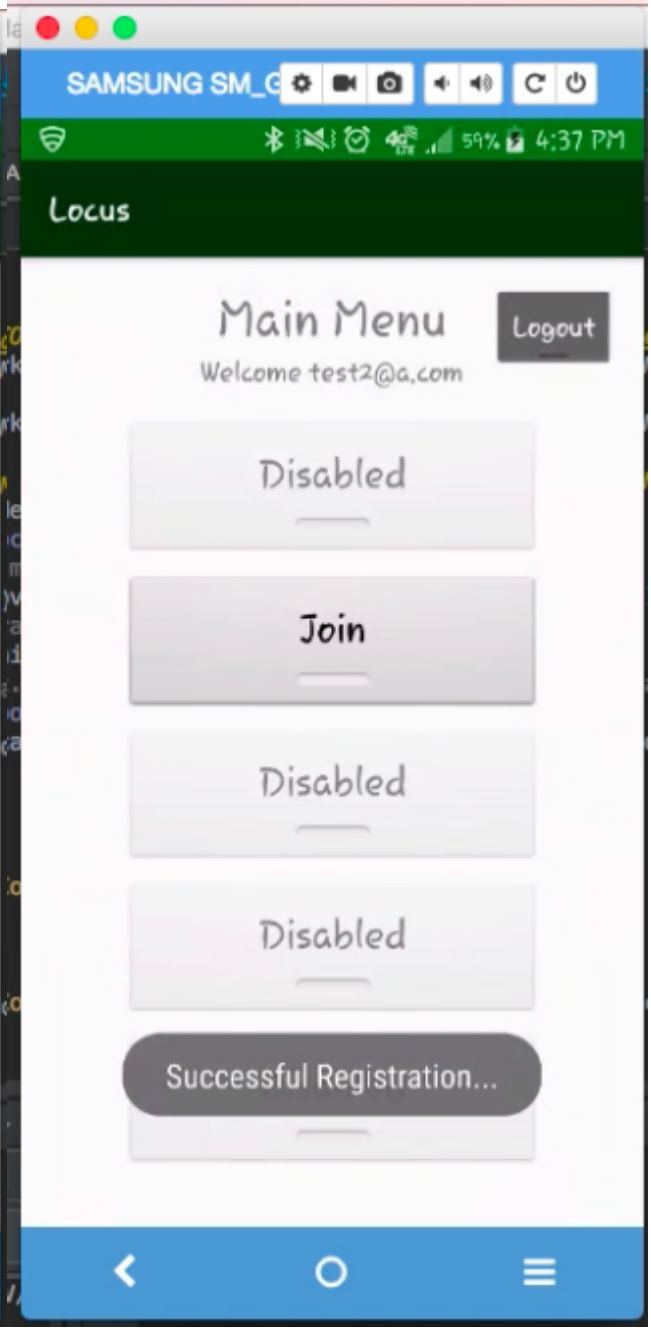


Fig 1.9 Teacher Map Page (before creation)

Fig 1.10 Teacher QR Generation

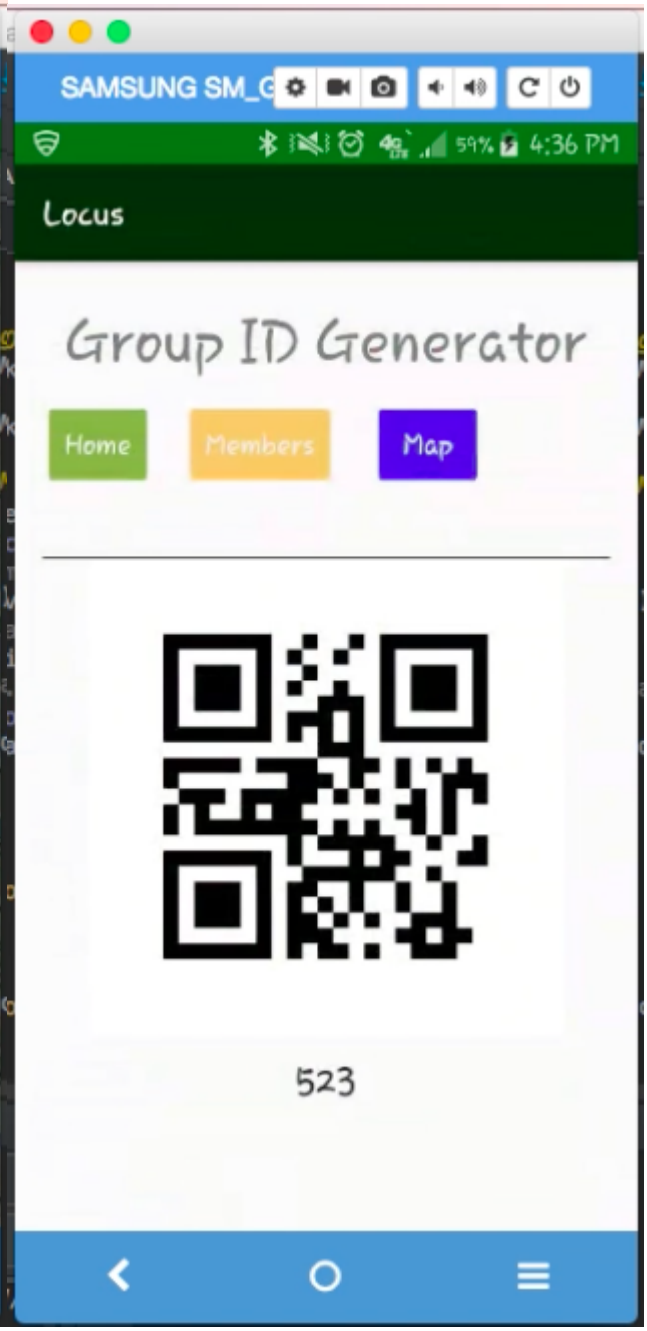


Fig 1.11
Teacher Map (after group created)

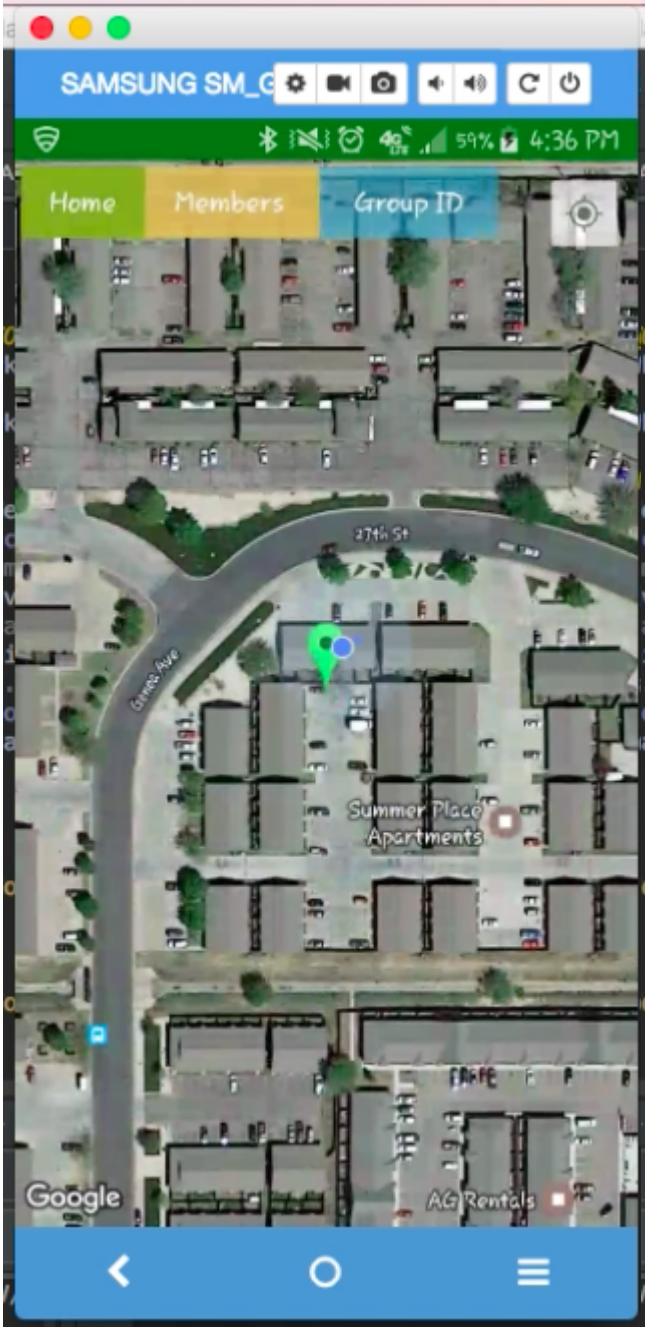


Fig 1.12
Teacher Member Page (with no members)

Fig 1.13
Teacher Home Page (after group created)

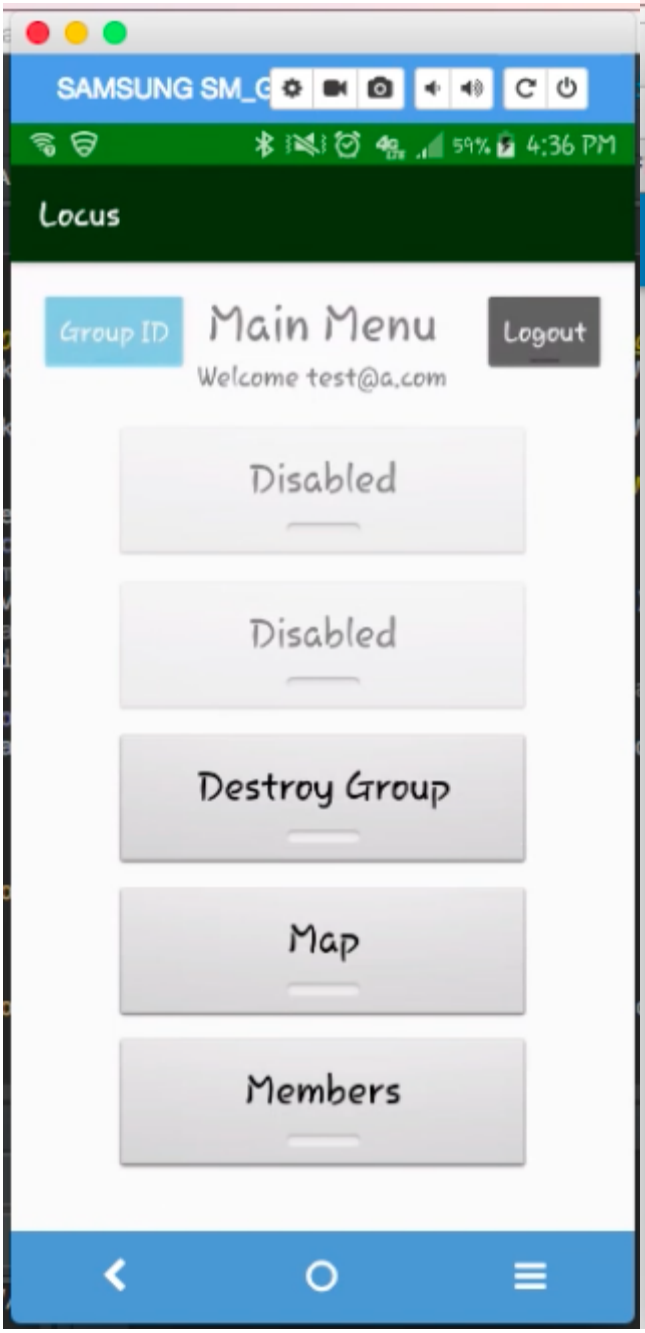


Fig 1.14
Student Home Page (after group joined)

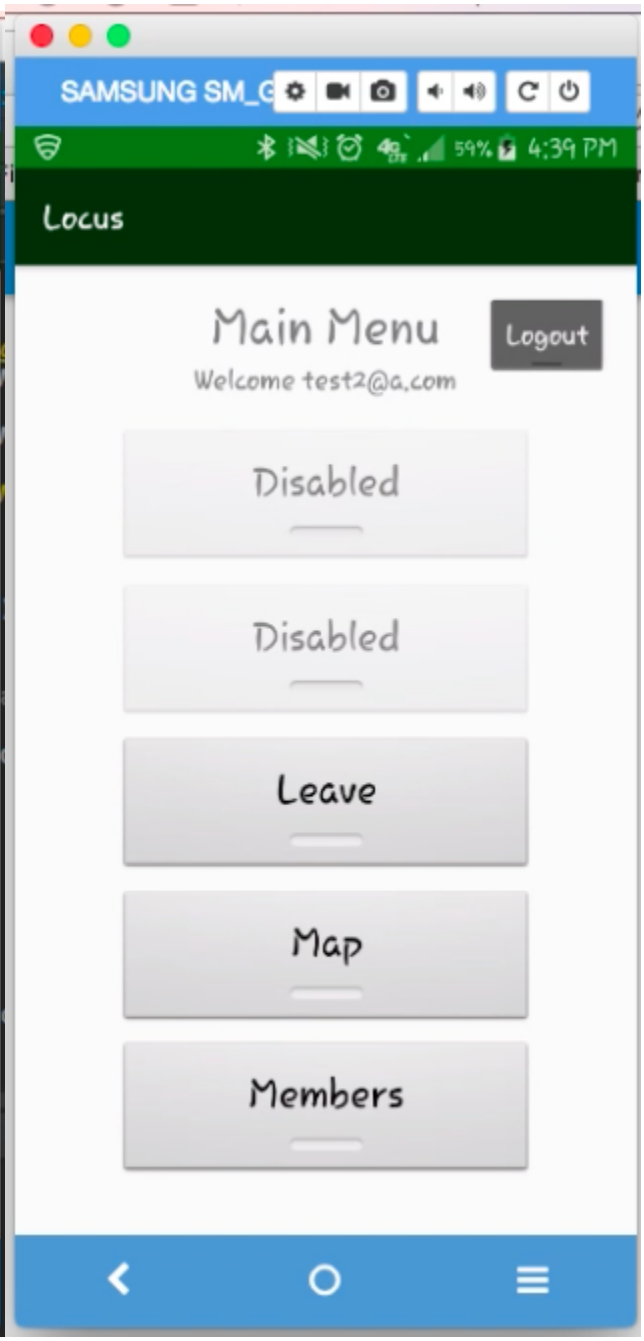
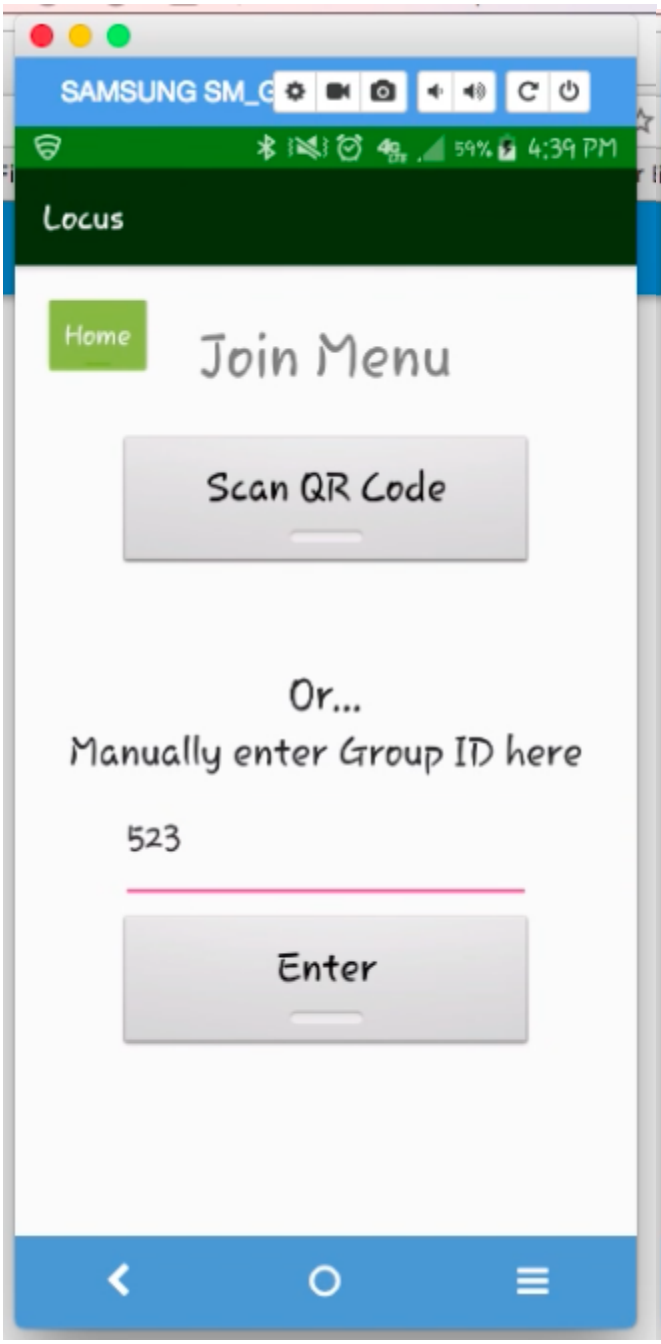


Fig 1.15 Student Group Id Input Page



F 1.16 Searching For Group Dialog

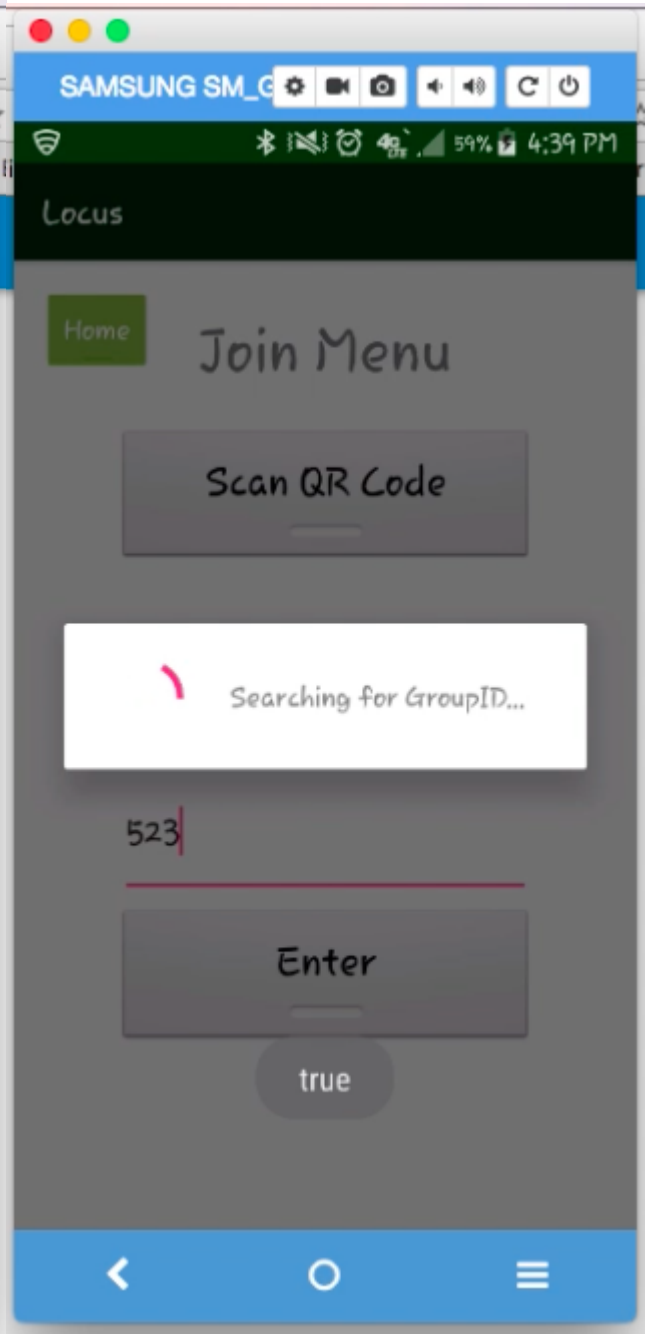


Fig 1.17
Student Map Page (after group joined)

Fig 1.18
Student Member Page (after group joined)

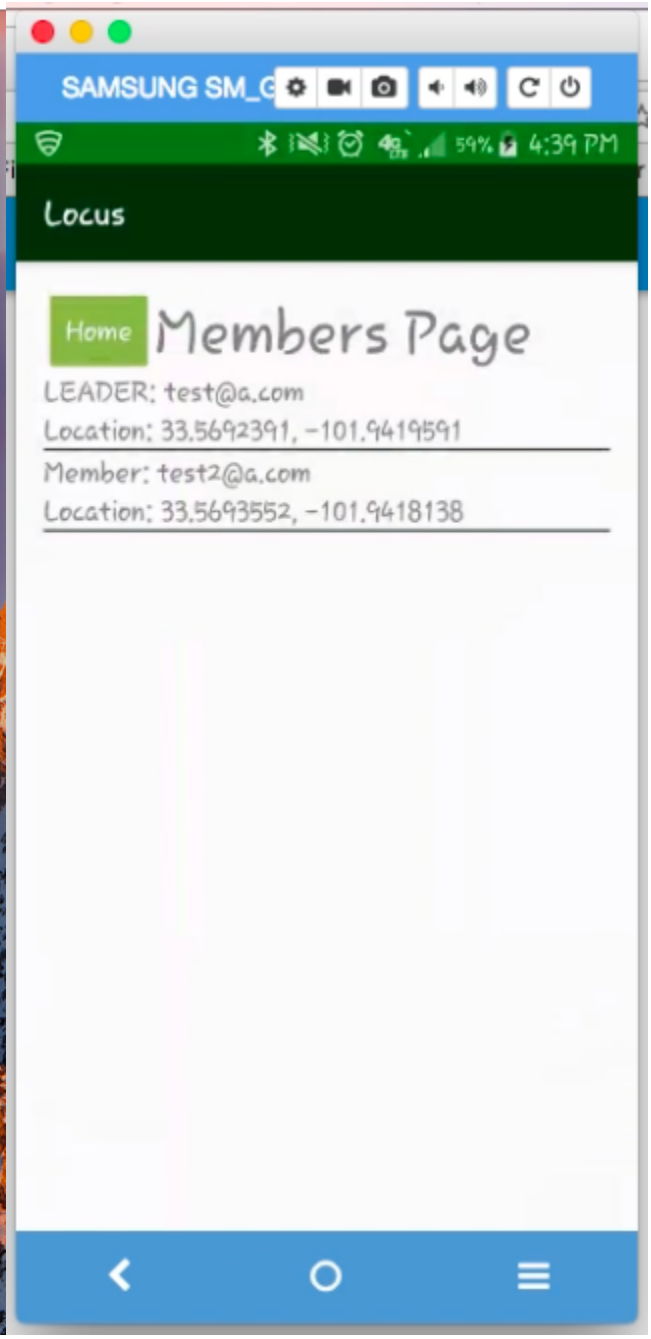


Fig 1.19
Deleting or Leaving a Group Dialog

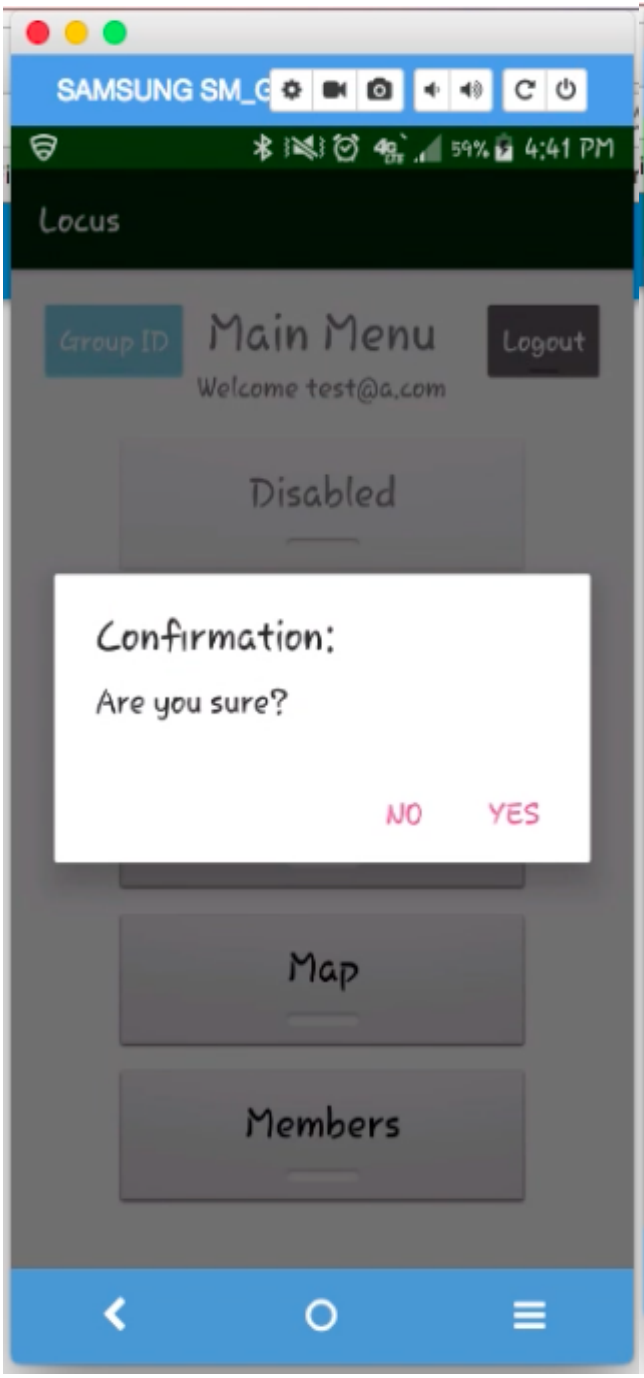


Fig 1.20
Teacher Home Page (after group deleted)

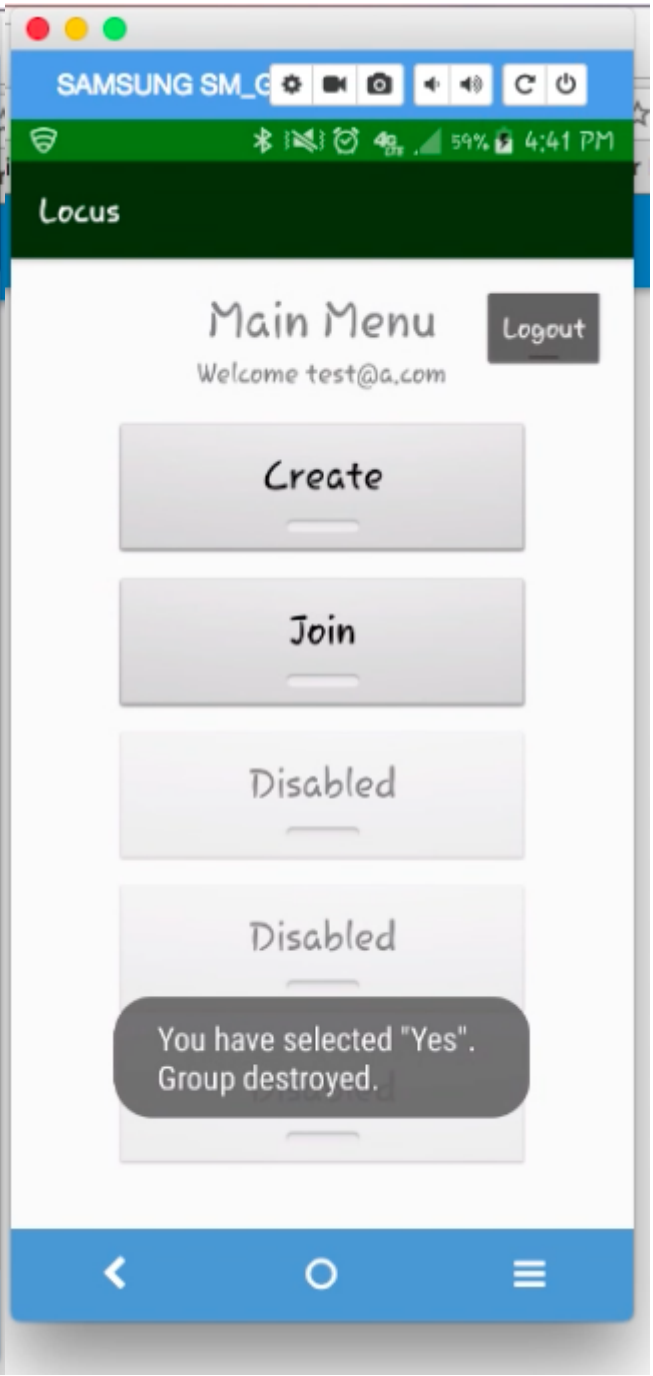


Fig 1.21
Teachers Map with Members

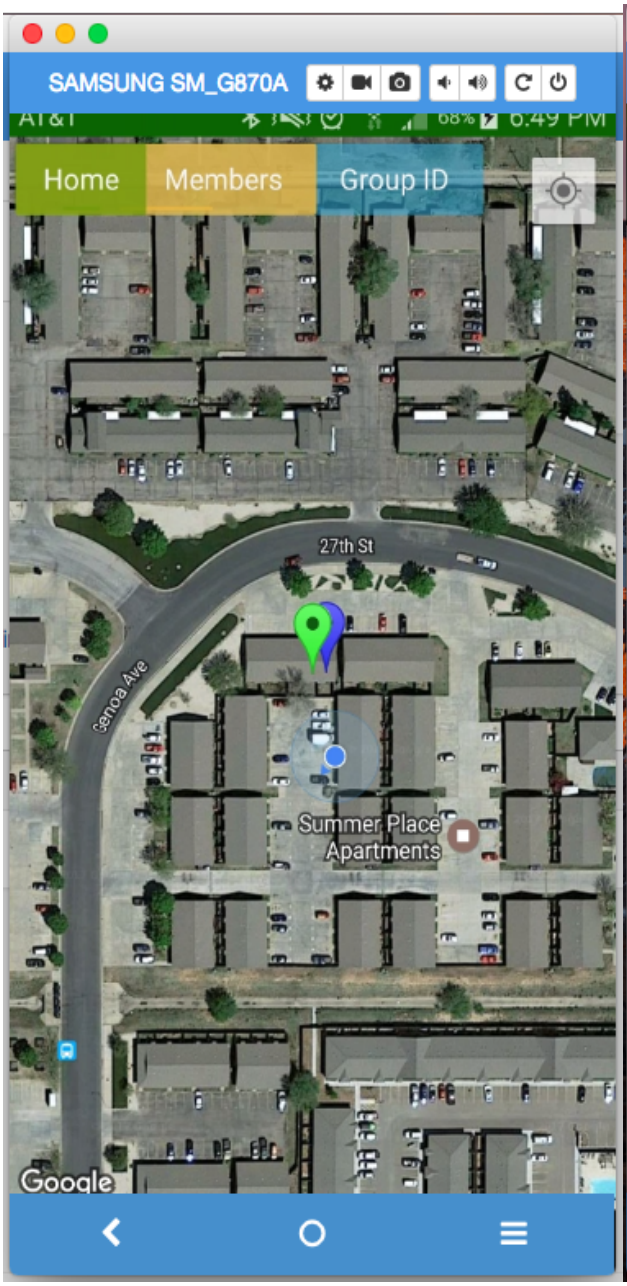
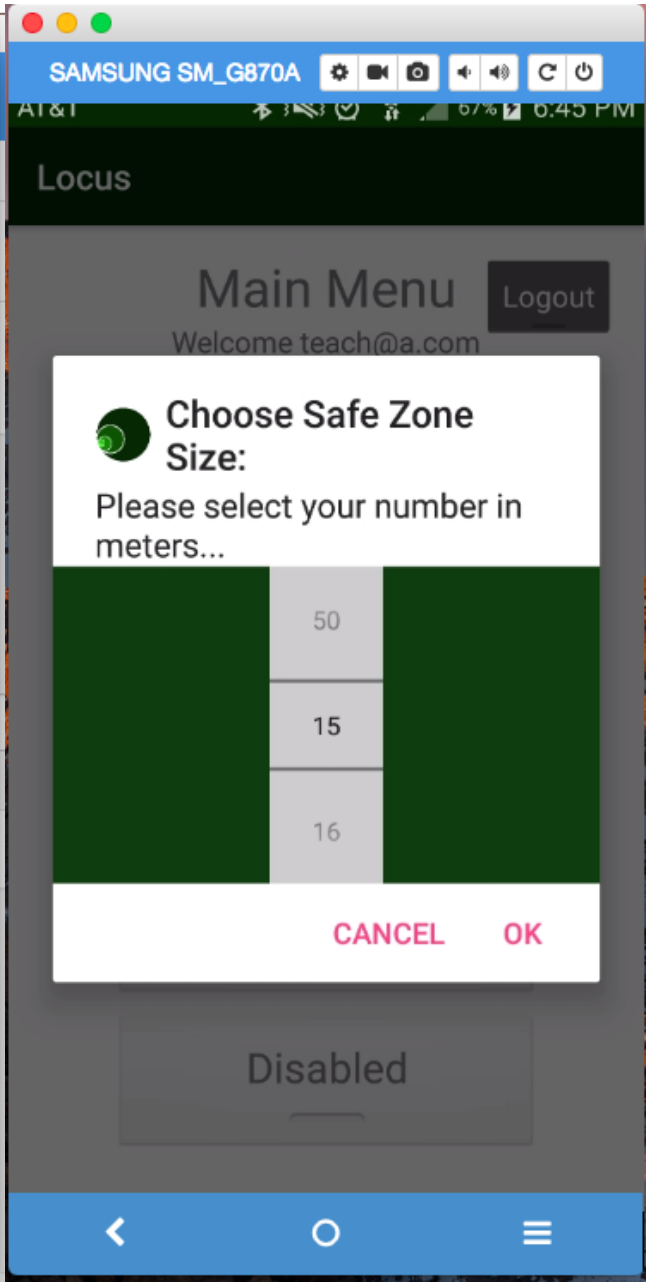


Fig 1.22
Teacher Safe Zone before Group Created



Map Activity

Before beginning implementation on the complex features in the Locus project, extensive research and practice took place. Research and practice began with module applications (practice/testing applications) that consist of individual complex features that are then integrated into the Locus application.

Implementation Techniques:

1. Develop mock application before integration with Locus
    a. **Practice Application:** Practice Map Application
        i. **Features Tested:**
            1. Location Services
            2. On Location Changed
            3. Location Permissions
2. Integrate mock application features into Locus application
3. Add features to update group member locations
4. Add algorithm for placing the markers on the map and determining distance from leader
    a. If distance is greater than the safe zone chosen by a group leader then send a notification to the members and group leader of the distance increase.

Location Permission Figures

Fig 2.1 Check Permissions Method: Making sure user has given permission

```
public boolean checkLocationPermission(){
    if (ContextCompat.checkSelfPermission(this,
            android.Manifest.permission.ACCESS_FINE_LOCATION)
            != PackageManager.PERMISSION_GRANTED) {
        // Asking user if explanation is needed
        if (ActivityCompat.shouldShowRequestPermissionRationale(this,
                android.Manifest.permission.ACCESS_FINE_LOCATION)) {
            // Show an explanation to the user *asynchronously* -- don't block
            // this thread waiting for the user's response! After the user
            // sees the explanation, try again to request the permission.
            //Prompt the user once explanation has been shown
            ActivityCompat.requestPermissions(this,
                    new String[]{android.Manifest.permission.ACCESS_FINE_LOCATION},
                    MY_PERMISSIONS_REQUEST_LOCATION);
        } else {
            // No explanation needed, we can request the permission.
            ActivityCompat.requestPermissions(this,
                    new String[]{android.Manifest.permission.ACCESS_FINE_LOCATION},
                    MY_PERMISSIONS_REQUEST_LOCATION);
        }
        return false;
    } else {
        return true;
    }
}
```

Fig 2.2 Permission Result Method: Handling result of user input

```
@Override
public void onRequestPermissionsResult(int requestCode,
                                       String permissions[], int[] grantResults) {
    switch (requestCode) {
        case MY_PERMISSIONS_REQUEST_LOCATION: {
            // If request is cancelled, the result arrays are empty.
            if (grantResults.length > 0
                    && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
                // Permission was granted.
                if (ContextCompat.checkSelfPermission(this,
                        android.Manifest.permission.ACCESS_FINE_LOCATION)
                        == PackageManager.PERMISSION_GRANTED) {
                    ///this is called because the object was not created before permissions where granted
                }
            } else {
                // Permission denied, Disable the functionality that depends on this permission.
                Toast.makeText(this, "permission denied", Toast.LENGTH_LONG).show();
            }
        }
    }
}
```

Map Activity Figures

Fig 3.1 Build API Method: Connects Map to Google API

```
//region Google Maps/Location Services Methods Region ######
//Building the apiClient used for updating location services
protected synchronized void buildGoogleApiClient() {
    mGoogleApiClient = new GoogleApiClient.Builder(this)  //
            .addConnectionCallbacks(this)                 //
            .addOnConnectionFailedListener(this)          //
            .addApi(LocationServices.API)                 //
            .build();
    mGoogleApiClient.connect();                           //
}
```

Fig 3.2 On Map Ready Method: Displays Map after API connect

```
Description of onMapReady Region
@Override
public void onMapReady(GoogleMap googleMap) {
    mMap = googleMap;
    //setting map type to HYBRID
    mMap.setMapType(GoogleMap.MAP_TYPE_HYBRID);
    //Initialize Google Play Services
    if (android.os.Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
        if (ContextCompat.checkSelfPermission(this,
                android.Manifest.permission.ACCESS_FINE_LOCATION)
                == PackageManager.PERMISSION_GRANTED) {
            //used to enable location layer which will allow a user to interact with current user location.
            mMap.setMyLocationEnabled(true);
            buildGoogleApiClient();
        }
    } else {
        buildGoogleApiClient();
        mMap.setMyLocationEnabled(true);
    }
}
```

Fig 3.3 Displaying User's Location: Displays the Markers on the Map

```
//getting the coordinates of current location and updating the camera
@Override
public void onLocationChanged(Location location) {
        if (mCurrLocationMarker != null) {
            mCurrLocationMarker.remove();
        }
        mLatLng = new LatLng(location.getLatitude(), location.getLongitude());
        // updates the members location to database
        updateMemberLocationToDB(mLatLng);
        MarkerOptions userMarkerOptions = new MarkerOptions();
        userMarkerOptions.position(mLatLng);
        userMarkerOptions.title("ME!!");
        if(GROUP_CREATED){
            checkDistanceFromMembers(location);
            userMarkerOptions.icon(BitmapDescriptorFactory.defaultMarker(BitmapDescriptorFactory.HUE_GREEN));
        }else if(GROUP_JOINED){
            if(FIRST_JOIN){
                FIRST_JOIN = false;
                joinGroup(GROUP_ID, mLatLng);
            }else{
                checkDistanceFromMembers(location);
            }
            userMarkerOptions.icon(BitmapDescriptorFactory.defaultMarker(BitmapDescriptorFactory.HUE_ORANGE));
        }else{
            userMarkerOptions.icon(BitmapDescriptorFactory.defaultMarker(BitmapDescriptorFactory.HUE_RED));
        }
        mCurrLocationMarker = mMap.addMarker(userMarkerOptions);
        // move map camera
        mMap.moveCamera(CameraUpdateFactory.newLatLng(mLatLng));
        // camera zoom into map
        mMap.animateCamera(CameraUpdateFactory.zoomTo(18));
}
```

Database Interaction

After the user interface was implemented and integration with the map activity from the practice application, the next phase was to do more research on how to connect the application to the database. Another practice module application was developed to understand the design and logic behind the communication, and setup with Google Firebase database.

The user account is created as a class object and posted to the database as a JSON tree. This allows us to convert the information received from the database back into a class object for easy access to the details of each user with getter and setter methods.

Implementation Techniques:

1. Develop mock application before integration with Locus
    a. **Practice Application:** Practice Database Interaction Application
        i. **Features Tested:**
            1. Account creation
            2. Login
            3. Updating information to database
            4. Retrieving information from database
2. Integrate mock application features into Locus application

1. Create user class to house user info
2. Create methods for adding teachers and students upon registration

Fig 4.1 User Info Class: Used to convert JSON string into object

```java
package com.vetealinfierno.locus;
//***** 3/24/17 jGAT
public class UserInfo {
    String userID;
    String userLocation;
    String email;
    String status;
    String groupID;

    public UserInfo() {
    }

    public UserInfo(String userID, String userLocation, String email, String status, String groupID) {
        this.userID = userID;
        this.userLocation = userLocation;
        this.email = email;
        this.status = status;
        this.groupID = groupID;
    }
```

Fig 4.2 Login Method: Login for an existing user to Firebase Authentication

```java
//region UserLogin Method Region ##########################################################################
private void userLogin(){
    String email = EditText_email.getText().toString().trim();
    String password = EditText_password.getText().toString().trim();
    if(TextUtils.isEmpty(email) || TextUtils.isEmpty(password)){
        //fields are empty
        print("Please Enter Email or Password");
        return;
    }
    progressDialog.setMessage("Logging in...");
    progressDialog.show();
    firebaseAuth.signInWithEmailAndPassword(email, password).addOnCompleteListener(this, (task) -> {
            progressDialog.dismiss();
            if(task.isSuccessful()){
                print("Successful Login...");
                //start the profile activity
                switchToHomeActivity();
            }
    }).addOnFailureListener(this, (e) -> { print(e.getMessage()); });
}
//endregion
```

Fig 4.3 Account Creation Method: Register a new user to Firebase Authentication

```
//region Register New User Methods Region ###################################
private void registerUser(){
    String email = EditText_email.getText().toString().trim();
    String cEmail = EditText_cEmail.getText().toString().trim();
    String password = EditText_password.getText().toString().trim();
    String cPassword = EditText_cPassword.getText().toString().trim();
    userType = spinnerUSERTYPE.getSelectedItem().toString();
    if(TextUtils.isEmpty(email) || TextUtils.isEmpty(cEmail) ||
            TextUtils.isEmpty(password) || TextUtils.isEmpty(cPassword)){
        //fields are empty
        print("Please Enter Email or Password");
    }else if(userType.contains("Choose")){
        print("Please answer the final question");
    }else if(email.equals(cEmail) && password.equals(cPassword)){
        progressDialog.setMessage("Registering User...");
        progressDialog.show();
        firebaseAuth.createUserWithEmailAndPassword(email, password)
                .addOnCompleteListener(this, (task) → {
                        progressDialog.dismiss();
                        if(task.isSuccessful()){
                            print("Successful Registration...");
                            if(userType.equals("Yes")){
                                addTeacher();
                            }else{
                                addStudent();
                            }
                        }
                }).addOnFailureListener(this, (e) → {
                print(e.getMessage());
        });
    }else if(!email.equals(cEmail)){
        print("Emails do NOT match...");
    }else if(!password.equals(cPassword)){
        print("Passwords do NOT match...");
    }
}
```

Fig 4.4 Add Student and Add Teacher Methods: Adding a student or teacher to the Firebase database

```
public void addStudent(){
    GROUP_JOINED = false;
    GROUP_ID = "";
    FIRST_JOIN = true;
    FirebaseUser user = firebaseAuth.getCurrentUser();
    String email = user.getEmail();
    DatabaseReference dBRef = FirebaseDatabase.getInstance().getReference("Students");
    String id = dBRef.push().getKey();
    UserInfo userInfo = new UserInfo(id, "Null", email, "No", "Null");
    dBRef.child(id).setValue(userInfo).addOnCompleteListener(this, new OnCompleteListener<Void>() {
        @Override
        public void onComplete(@NonNull Task<Void> task) {

        }
    });
}

public void addTeacher(){
    GROUP_CREATED = false;
    GROUP_ID = "";
    QR_GEN = false;
    FirebaseUser user = firebaseAuth.getCurrentUser();
    String email = user.getEmail();
    DatabaseReference dBRef = FirebaseDatabase.getInstance().getReference("Teachers");
    String id = dBRef.push().getKey();
    UserInfo userInfo = new UserInfo(id, "Null", email, "No", "Null");
    dBRef.child(id).setValue(userInfo).addOnCompleteListener(this, new OnCompleteListener<Void>() {
        @Override
        public void onComplete(@NonNull Task<Void> task) {

        }
    });

}
```

The features implemented so far allowed us to continue implementation without making additional module applications. Most testing was done as unit tests in the module applications and then integration testing once the module is implemented in the Locus project.

Additional testing of the following features was done directly in the LOCUS application.

Generating and Scanning QR Code

The techniques for generating and scanning the QR code are as follows:

Implementation Techniques:

1. Method for generating QR code
    a. Call java library
        i. Convert text to bitmap
    b. Display the bitmap as an image in the activity to the user
2. Import ZXing library for scanning QR code
3. Method for scanning QR code
    a. Calls ZXing library to import the scanner
    b. Request camera permissions
    c. Pull up the camera
    d. Open scanner
4. Test the feature in different scenarios

Generating and Scanning QR Code Figures

5.1 Generating QR Code Method: Generates the QR Code and displays it as an image

```
//region Generate QR Code Region ##############################################################################
public void generateGroupID(){
    image.setVisibility(View.VISIBLE);
    if (!GROUP_ID.equals("")) {
        MultiFormatWriter multiFormatWriter = new MultiFormatWriter();
        try {
            BitMatrix bitMatrix = multiFormatWriter.encode(GROUP_ID, BarcodeFormat.QR_CODE, 400, 400);
            BarcodeEncoder barcodeEncoder = new BarcodeEncoder();
            QR_CODE = barcodeEncoder.createBitmap(bitMatrix);
            image.setImageBitmap(QR_CODE);
        } catch (WriterException e) {
            e.printStackTrace();
        }
        //disables join and create btns, enables members, leave, map
        if(!GROUP_CREATED){
            createGroup(GROUP_ID);
            GROUP_CREATED = true;
        }
        QR_GEN = true;
        ToggleQRGenBtn(false);
        String temp = "Group ID: "+GROUP_ID;
        groupID.setText(temp);
    }else{
        print("Error: Enter Text First");
    }
}
//endregion
```

5.2 Scanning QR Code Method: Used by student to scan QR Code, opens camera

```
//region Scanning QRCode Methods Region ######################################
///the scanning of the QR code
public void ScanQRCodeMethod(View view){
    final Activity activity = this;
    IntentIntegrator integrator = new IntentIntegrator(activity);
    integrator.setDesiredBarcodeFormats(IntentIntegrator.QR_CODE_TYPES);
    integrator.setPrompt("Scan QR Code");
    integrator.setCameraId(0);
    integrator.setBeepEnabled(true);
    integrator.setBarcodeImageEnabled(false);
    integrator.initiateScan();
}
```

Creating Group on Database

Since we can create the group ID, we can now implement the features for creating the group on the Firebase database. This process was practiced in the database practice application so this implementation phase is just a matter of applying the techniques learned for pushing to the database into the Locus application.

The group is created as a class object just like the user account, allowing us for easy access to the details of each group with getters and setters.

Implementation Techniques:

1. Create group class to house the group information
2. Method for Creating Group
    a. Prompt user to choose a safe zone
    b. Grab snapshot of data
    c. Convert snapshot into group class object
    d. Check if group exists
        i. If not return → error message to user
        ii. Else → create group
            1. Push group class object to the database
3. Test feature in different scenarios

Figures on next page...

Creating Group on Database Figures

Fig 6.1 Group Info Class: Used for converting JSON string into object

```
public class GroupInfo {

    String groupID;
    String groupLeader;
    String key;

    public GroupInfo() {
    }

    public GroupInfo(String key, String groupID, String groupLeader) {
        this.key = key;
        this.groupID = groupID;
        this.groupLeader = groupLeader;
    }
```

Fig 6.2 Choosing Safe Zone: Allows user to set a safe zone Min: 15 Max: 50 [in meters]

```
public void showSafeZoneDialog() {
    Context mContext = HomeActivity.this;
    RelativeLayout linearLayout = new RelativeLayout(mContext);
    final NumberPicker aNumberPicker = new NumberPicker(mContext);
    aNumberPicker.setMaxValue(50);
    aNumberPicker.setMinValue(15);
    aNumberPicker.setClickable(false);
    aNumberPicker.setBackgroundColor(Color.LTGRAY);

    RelativeLayout.LayoutParams params = new RelativeLayout.LayoutParams(50, 50);
    RelativeLayout.LayoutParams numPicerParams = new RelativeLayout.LayoutParams(RelativeLa
    numPicerParams.addRule(RelativeLayout.CENTER_HORIZONTAL);

    linearLayout.setLayoutParams(params);
    linearLayout.setBackgroundColor(getResources().getColor(R.color.colorSlightDrkGreen));
    linearLayout.addView(aNumberPicker, numPicerParams);

    AlertDialog.Builder alertDialogBuilder = new AlertDialog.Builder(mContext);
    alertDialogBuilder.setTitle("Choose Safe Zone Size:");
    alertDialogBuilder.setMessage("Please select your number in meters...");
    alertDialogBuilder.setView(linearLayout);
    alertDialogBuilder.setIcon(R.mipmap.ic_launcher);
    alertDialogBuilder.setCancelable(false).setPositiveButton("Ok", (dialog, id) → {
            //Log.e("","New Quantity Value : "+ aNumberPicker.getValue());
            SAFE_ZONE = aNumberPicker.getValue();
            //print("number picker = "+aNumberPicker.getValue());
            if (SAFE_ZONE < 15) {
                print("Error: Cannot be less than 15");
            } else if (SAFE_ZONE > 50) {
                print("Error: Cannot be greater than 50");
            } else {
                print("safe zone = " + SAFE_ZONE);
                startMapActivity();
            }
    }).setNegativeButton("Cancel", (dialog, id) → {
            dialog.cancel();
            print("You selected cancel, no group created.");
    });
    AlertDialog alertDialog = alertDialogBuilder.create();
    alertDialog.show();
```

Fig 6.3 Creating Group Method: Used by teacher, creates group on Firebase database

```
//region CreagteGroup/Update Teach Info Methods Region ################################################
public void createGroup(final String groupID){
    DatabaseReference dBRef;
    String leaderEmail = USER_EMAIL;
    String leadersLocation = setLocationString(mLatLng);
    dBRef = FirebaseDatabase.getInstance().getReference(groupID);
    String id = dBRef.push().getKey();
    UserInfo userInfo = new UserInfo(id, leadersLocation, leaderEmail, "LEADER", groupID);
    dBRef.child(id).setValue(userInfo).addOnCompleteListener(this, (task) → {
            print("Group: "+groupID+" Created");
    });
    dBRef = FirebaseDatabase.getInstance().getReference("Groups");
    String id2 = dBRef.push().getKey();
    GroupInfo groupInfo = new GroupInfo(id2, groupID,leaderEmail);
    dBRef.child(id2).setValue(groupInfo).addOnCompleteListener(this, new OnCompleteListener<Void>() {
        @Override
        public void onComplete(@NonNull Task<Void> task) {

        }
    });
    updateTeachersData(groupID, leadersLocation);
}
```

Joining Group on Database

After the implementation of creating the group on the database, the next step is to allow other users to join this group. The user interface already allows the user to either scan the QR code or input the group ID manually into a text field.

Implementation Techniques:

1. Method for Joining Group
    a. Called after QR code is scanned or Group ID is input manually
    b. Grab snapshot of all the groups
        i. Check if group exists
            1. If not → return
            2. Else → add member
                a. push user class object to database under group ID
2. Test feature in different scenarios

Figures on next page...

Join Group on Database Figures

Fig 7.1 Does Group Exist Method: Checks if the user's manual input exists on Firebase database (no need to check if QR code is scanned)

```
public void doesGroupIDExist(final String groupID){
    DatabaseReference dBRef = FirebaseDatabase.getInstance().getReference("Groups");
    dBRef.addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(DataSnapshot dataSnapshot) {
            DatabaseReference dBRef = FirebaseDatabase.getInstance().getReference("Groups");
            for(DataSnapshot groupSnapShot: dataSnapshot.getChildren()){
                GroupInfo groupInfo = groupSnapShot.getValue(GroupInfo.class);
                if(groupID.equals(groupInfo.getGroupID())){
                    dBRef.removeEventListener(this);
                    setGroupExists(true);
                }
            }
        }
        @Override
        public void onCancelled(DatabaseError databaseError) {

        }
    });
}
//endregion
```

Fig 7.2 Join Group Method: Used by student, adds member to group on Firebase database

```
//region Joining Group Methods Region ################################################################################
public void joinGroup(final String groupID, LatLng latLng){
    DatabaseReference dBRef = FirebaseDatabase.getInstance().getReference(groupID);
    FirebaseAuth firebaseAuth = FirebaseAuth.getInstance();
    FirebaseUser user = firebaseAuth.getCurrentUser();
    String email = user.getEmail();
    String id = dBRef.push().getKey();
    String membersLocation = getLocationString(latLng);
    UserInfo userInfo = new UserInfo(id, membersLocation, email, "Member", groupID);
    dBRef.child(id).setValue(userInfo).addOnCompleteListener(this, (task) → {
            Toast.makeText(MapsActivity.this, "You Have Joined Group: "+groupID, Toast.LENGTH_SHORT).show();
    });
    updateUserStatus(groupID, membersLocation);

}
```

Leaving and Deleting Group on Database

Now that we can create and join a group the next phase of implementation is adding the feature of deleting the group and leaving the group.

Implementation Techniques:

1. Method for removing members
    a. Called when a member presses leave button and confirms yes
    b. Grab snapshot of Group data
    c. Locate member in the group
    d. Remove the member from the group
    e. Update user's information to reflect change
2. Method for removing groups

a. Called when a leader presses destroy button and confirms yes
b. Grab snapshot of Group data
c. Locate the leaders group
d. Remove group from database
e. Grab snapshot of Students data
f. Find students in the group destroyed
    i. Update user information to reflect the change
3. Test feature in different scenarios

Leaving and Deleting Group from Database Figures

8.1 Leaving Group Method: Used by student when leaving a group on Firebase database

```
//region Destroy/Remove Member Methods Region ###################################################
public void removeMember(){
    print(GROUP_ID);
    final DatabaseReference dBRef = FirebaseDatabase.getInstance().getReference(GROUP_ID);
    dBRef.addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(DataSnapshot dataSnapshot) {
            for(DataSnapshot groupSnapShot: dataSnapshot.getChildren()){
                UserInfo userInfo = groupSnapShot.getValue(UserInfo.class);
                if(userEmail.equals(userInfo.getEmail())){
                    String id = userInfo.getUserID();
                    dBRef.child(id).removeValue();
                }
                dBRef.removeEventListener(this);
            }
        }
        @Override
        public void onCancelled(DatabaseError databaseError) {
        }
    });
}
```

8.2 Destroy Group Method: Used by teacher when deleting a group on Firebase database

```
public void destroyGroup(){
    //destroys the group
    DatabaseReference dBRef = FirebaseDatabase.getInstance().getReference();
    dBRef.child(GROUP_ID).removeValue();
}
```

Updating User Location

Now that we have implemented the features of adding, joining, deleting, and leaving groups. The feature of updating the user's location is implemented.

Implementation Techniques:

1. Method for checking if user has a group
    a. Called when user launches application, opens map screen
        i. Call method for updating member location (also called on create/join)
            1. If user belongs to a group or has a group → Check location permissions
                a. If true → Update location
                    i. Push location to database in form of user class object
                b. Else → return
            2. Else → return
2. Method for Removing the markers (clearing the markers from the past locations
3. Grab group members' location from database
    a. Gets all the locations of the members and stores them in an array
4. Update the markers (places the markers in their new locations)
5. Check distance from leader with all the locations
6. If distance is greater than the safe zone send a push notification to the leader and member whose distance is greater than safe zone
7. Test feature in different scenarios

Figures on next page...

Updating User Location Figures

Fig 9.1 Update Members Location: Updates the location of current user to Firebase database

```
//region Updating Member Location Methods Region ##########################################################################
public void updateMemberLocationToDB(final LatLng latLng){
    DatabaseReference dBRef = FirebaseDatabase.getInstance().getReference(GROUP_ID);
    dBRef.addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(DataSnapshot dataSnapshot) {
            int i = 0;
            for(DataSnapshot locationSnapShot: dataSnapshot.getChildren()){
                UserInfo user = locationSnapShot.getValue(UserInfo.class);
                memberIDs[i] = null;
                groupLatLng[i] = null;
                groupMarkers[i] = null;
                groupMarkerOptions[i] = null;
                i++;
                if(USER_EMAIL.equals(user.getEmail())){
                    //print("USER email = " + USER_EMAIL);
                    String id = user.getUserID();
                    String location = getLocationString(latLng);
                    user.setUserLocation(location);
                    DatabaseReference dBRef = FirebaseDatabase.getInstance().getReference(GROUP_ID);
                    dBRef.child(id).setValue(user).addOnCompleteListener(MapsActivity.this, new OnCompleteListener·
                    dBRef.removeEventListener(this);
                }
            }
        }
        @Override
        public void onCancelled(DatabaseError databaseError) {...}
    });
    if(GROUP_CREATED || GROUP_JOINED){
        removeMarkers();
        grabGroupMembersLocationFromDB();
        updateMarkers();
    }
}
//endregion
```

Fig 9.2 Remove Markers: Clears the old location markers on map

```
public void removeMarkers(){
    if(groupMarkers != null) {
        for (Marker marker : groupMarkers) {
            if (marker != null) {
                marker.remove();
            }
        }
    }
}
```

Fig 9.3 Grab Group Members Location from Database: Collects the location of all members in the group from Firebase database and stores them in an array

```
public void grabGroupMembersLocationFromDB() {
    final DatabaseReference dBRef = FirebaseDatabase.getInstance().getReference(GROUP_ID);
    dBRef.addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(DataSnapshot dataSnapshot) {
            int i = 0;
            for(DataSnapshot locationSnapShot: dataSnapshot.getChildren()){
                UserInfo user = locationSnapShot.getValue(UserInfo.class);
                //print("User status = "+ user.getStatus());
                if(user.getStatus().contentEquals("Member") && GROUP_CREATED || !USER_EMAIL.equals(user.getEmail())
                        && GROUP_JOINED){
                    memberIDs[i] = user.getEmail();
                    //print(" i = " + i);
                    //print("member id = " + memberIDs[i]);
                    addMemberLocationToArray(user.getUserLocation(), i);
                    i++;
                }
            }
            dBRef.removeEventListener(this);
        }
        @Override
        public void onCancelled(DatabaseError databaseError) {...}
    });
}
```

Fig 9.4 Update Markers: Places the markers of all members in their new locations on the map

```
public void updateMarkers(){
    new Handler().postDelayed(() -> {
            for(int i = 0; i<groupLatLng.length; i++){
                if(groupLatLng[i] != null) {
                    //print("inside placing the markers, i = " + i);
                    groupMarkerOptions[i] = new MarkerOptions();
                    groupMarkerOptions[i].position(groupLatLng[i]);
                    // this will display the user email on the map
                    groupMarkerOptions[i].title(memberIDs[i]);
                    if (GROUP_JOINED) {
                        if (i == 0) {
                            //groupMarkerOptions[i].title("Leader");
                            groupMarkerOptions[i].icon(BitmapDescriptorFactory.defaultMarker(BitmapDescriptorFactory.HUE_GREEN));
                        } else {
                            //groupMarkerOptions[i].title("Member");
                            groupMarkerOptions[i].icon(BitmapDescriptorFactory.defaultMarker(BitmapDescriptorFactory.HUE_BLUE));
                        }
                        groupMarkers[i] = mMap.addMarker(groupMarkerOptions[i]);
                    }else{
                        //groupMarkerOptions[i].title("Member");
                        groupMarkerOptions[i].icon(BitmapDescriptorFactory.defaultMarker(BitmapDescriptorFactory.HUE_BLUE));
                        groupMarkers[i] = mMap.addMarker(groupMarkerOptions[i]);
                    }
                }
            }
    },4000);
}
```

Fig 9.5 Check Distance from Members: Uses distanceTo() to check the distance between all members and the leader

```
public void checkDistanceFromMembers(Location location) {
    Location memberLocation = new Location("");
    // user is a group leader check distance from each member
    if (GROUP_CREATED) {
        for (int i = 0; i < groupLatLng.length; i++) {
            if (groupLatLng[i] != null) {
                memberLocation.setLatitude(groupLatLng[i].latitude);
                memberLocation.setLongitude(groupLatLng[i].longitude);
                // print("groupLatLng["+i+"] = "+ groupLatLng[i]);
                // print("distance between = " + location.distanceTo(memberLocation));
                // print("safe Zone = " + SAFE_ZONE);
                if (location.distanceTo(memberLocation) > SAFE_ZONE) {
                   // print("member is far from leader");
                    groupMarkerOptions[i].icon(BitmapDescriptorFactory.defaultMarker(BitmapDescriptorFactory.HUE_AZURE));
                    PushNotification(i, location.distanceTo(memberLocation));
                }
            }
        }
    }else{
        // user is a member check distance from leader
        for(int i =0; i<1; i++){
            if(groupLatLng[i] != null){
                memberLocation.setLatitude(groupLatLng[i].latitude);
                memberLocation.setLongitude(groupLatLng[i].longitude);
                // print("distance between = " + location.distanceTo(memberLocation));
                // print("safe zone = " + SAFE_ZONE);
                if (location.distanceTo(memberLocation) > SAFE_ZONE) {
                    // print("you are far from leader");
                    PushNotification(i , location.distanceTo(memberLocation));
                }
            }
        }
    }
}
```

Fig 9.6 Push Notification: Method is called if distanceTo() returns a value greater than the safe zone [meters]

```
//region PushNotifications ##########################################################################################
public void PushNotification(int i, float distanceBetween){
    String title;
    if(GROUP_CREATED){
        title = "Caliente: Member falling behind!";
    }else{
        title = "Caliente: Return to leader!";
    }
    long[] pattern = {500,500,500,500,500,500,500,500,500};
    NotificationManager pushNotify = (NotificationManager)getSystemService(NOTIFICATION_SERVICE);
    Intent intent = new Intent(this, MapsActivity.class);
    PendingIntent pIntent = PendingIntent.getActivity(this, (int) System.currentTimeMillis(), intent, 0);
    Notification notification = new Notification.Builder(this)
            .setSmallIcon(R.mipmap.ic_launcher)
            .setContentTitle(title)
            .setContentText(memberIDs[i] +" is "+ (Math.floor(distanceBetween * 100) / 100)+ "meters from you.")
            .setAutoCancel(true)
            .setSound(Settings.System.DEFAULT_NOTIFICATION_URI)
            .setLights(Color.BLUE, 500, 500)
            .setVibrate(pattern)
            .setContentIntent(pIntent)
            .build();
    pushNotify.notify(0, notification);
}
//endregion
```

Updating Members List Page

Updating the members list page is a feature that allows the members or leader to view the overall information of each user in real time.

Implementation Techniques:

1. Create list view to display an array list of user information
2. Add code to the onStart() method in members list activity
    a. Grabs snapshot of data anytime the data in the database has been changed
    b. Updates the array list to reflect the changes made to the user
3. Test feature in different scenarios

Updating Members List Page Figures

Fig 10.1 Update Members List On Start Method: Updates the list of members view and displays the change to the user every time there is a change to the group on Firebase database

```
@Override
protected void onStart() {
    super.onStart();
    dBRef.addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(DataSnapshot dataSnapshot) {
            userList.clear();
            for(DataSnapshot userSnapShot: dataSnapshot.getChildren()){
                UserInfo user = userSnapShot.getValue(UserInfo.class);
                userList.add(user);
            }
            UserList adapter = new UserList(MembersListActivity.this, userList);
            listViewUser.setAdapter(adapter);
        }
        @Override
        public void onCancelled(DatabaseError databaseError) {

        }
    });
}
//endregion
```

In App Messaging and Notification of Group Member Inactivity

These features were proposed during implementation and considered to be prospective features in LOCUS. The team was unable to implement these features due to the time constraint of the project.

Functionalities Completed

1. **High fidelity user interface**
    a. Buttons: Disabled/Enabled
    b. Dialog Boxes for user confirmation
    c. Dialog Boxes to show progress in application to user
    d. Smooth transitions between activities
    e. Wide variety of navigation options
2. **Map activity**
    a. **Basic functionality:** features provided by Google API
    b. Additional features were added to achieve LOCUS functionality
3. **Request location permissions**
    a. Request permissions and handle the result from the user
4. **Displaying user's current location**
    a. Displaying the marker of the user (color coded)
5. **QR code generation**
    a. Generates the Group ID in the form a QR Code
6. **Request camera permissions**
    a. Request permissions and handle the result from the user
7. **QR code scanning**
    a. Opens the camera for scanning
8. **Firebase Authentication**
    - **Account creation:** Creates an account, gives read and write permissions to database
    - **Account login:** Allows for existing user to use their read and write permissions
9. **Firebase Database interaction**
    - **Creating groups:** Creates a group on the database
        - Choosing custom safe zone: metric in meters and the distance from the leader deemed safe
    - **Joining groups:** Adds a member to the group on the database
    - **Updating current location:** Updates the current location of the user to the database
    - **Retrieving locations of all members in database:** grabs the location of all the members from the database and updates the map with respective markers of these locations
    - **Leaving groups:** removes student from the group on the database
    - **Destroying groups:** removes the entire group from the database
10. **Updating members list page**
    a. Updates the view of the members list after any data change in the group

11. **Updating map with member locations**
    a. Places the locations of all members on the map with color coded markers
12. **Determining distance between members and leader**
    a. Calculates the distance between the members and the leader in meters
13. **Push notification**
    a. If the distance between any member and leader is greater than safe zone metric
14. **Automatic background updates**
    a. Update of user location to database every 30 seconds, achieve by Map activity (fragment activity that runs in the background)

Functionalities not Completed

The following is a list of features that were not completed due to the time constraint of the project:

1. Group messaging: in app messaging for users to communicate with members and group leader
2. Heartbeat message: keeps the teacher informed of members' activity (notifies leader if user has closed the app or left the group)
3. Automatic remove of members from group: deletes the group at the end of the business day (privacy purposes, keep students' locations private after school hours)

Technical Issues

We attempted to use SQL Server for a database engine; however, due to issues with implementing the Java Database Connectivity (JDBC) API and the transient nature of our database records, we decided a real-time database, instead of a relational database, was in order. So, we switched to Google's Firebase service to more readily satisfy our project's needs.

Also, when trying to debug the application and make sure that each user's application would reflect the changes made to the database i.e. after a member has joined a leader's group. We found it easier to debug if we add a Login/Create Account (Firebase Authentication) feature. In this way, we can log back into a leader's account after testing a members account for joining said leader's group. Making sure that each account is updating after a change occurred from a different account within the same group.

Member Accomplishments

Jessica Hao

1. Developer
2. Worked in pair programming with John and Zack to further understand the LOCUS application
3. Design the user interface for Login and Register activities
4. Implemented user input functionality of Login and Register activities
5. Implemented button functionality for the user interface
6. Helped prepare all documents and presentation slides
7. Helped prepare all UML diagrams
8. Designed the original paper prototype and storyboard

John G. Toland

1. Lead Developer: Helped the development team with any issues, knows all aspects of the systems, managed the flow of the project (making sure implementation stayed on schedule), and tested all features of the project (unit tests and integration testing).
2. Implementation of user interface functionality
3. Implemented the view for members list activity
4. Added functionality of scanning QR Code
5. Added functionality of generating QR Code
6. Added the MapActivity and Google Location Services API
7. Added Firebase Authorization and Firebase database to LOCUS application
8. Handled all data received from the database
9. Implemented algorithm for updating members' location on the map and calculating the distance from the leader
10. Implemented the functionality of sending push notifications
11. Added all dialog boxes and user input functionality
12. Designed LOCUS icon and name
13. Helped prepare all UML diagrams
14. Helped prepare all documents and presentation slides

Zachariah Grummons

1. Developer
2. Designed initial project design and architecture
3. Designed structure of Firebase database
4. Implemented create group functionality to Firebase database
5. Implemented join group functionality to Firebase database
6. Added the feature of updating the members list page from Firebase database
7. Added the feature of deleting groups
8. Helped prepare all UML diagrams
9. Added the feature of removing members from groups
10. Helped prepare all documents and presentation slides