Last updated: May 21, 2017

Android Application for Parking

Author: Kendrick Kwok

Platform and Software Used

Platform

1. Android SDK 7.1 API level 25
2. Google play-services 10.2.1
   Used to update Google apps and apps from Google Play. This component provides core functionality like authentication to your Google services
3. Google play-services-maps 10.2.1
   It allows us to add maps based on Google Maps data to the application. It automatically handles access to Google Maps servers, data downloading, map display, and response to map gestures.
4. Gson 2.8.0
   Used to serialize and deserialize Java objects to and from Json.

Software

1. Android Studio 2.3.1
2. Android Build Tools 25.0.2
3. Git 2.13.0
4. Java (sun-java6-jdk)

Project Description:

The main aim of the ParkMe application is to make an application which is easy to use and can give you available parking spots near your destination. Many people must be aware that finding parking spot in busy city like San francisco is quite difficult. ParkMe application tries to reduce that parking spot finding effort by presenting a simple and easy to use application.

When we started developing the ParkMe application, we came up with a new approach to find parking. We also added a seller part in this application who is willing to sell the parking spot that he currently have. So, ParkMe has two types of users, seller and a buyer. The basic flow of the actions in the application would be,

1. Seller posts his spot on the application as a sellable parking spot.
2. The application will store details of all such seller spots posted by the sellers in the database.(The database is hosted on amazon cloud server so it will be available for use to all the ParkMe application instances.)
3. The buyer starts using the ParkMe application, and puts a pin on map in the area where he wants to find the spot.
4. The application will return all such spots posted by seller which are within 1 mile radius from the buyers posted spot where he was willing to find the parking.
5. The buyer selects the spot most suitable for him from all the spots displayed within 1 mile radius.
6. Once the buyer selects the spot the application will give buyer the total distance of buyer’s current location to his selected parking spot, the time required to drive till that spot and also the shortest driving route till that spot.
7. Once, the buyer reaches to his selected parking spot, the coordinates of buyers and sellers location match and the payment options will pop up at the buyer's end.
8. Once the buyer pays the seller for the parking spot the notification will be received at seller's end that you have received the payment and seller will then be asked for confirm payment.
9. If the payment is successful at both the ends the transaction will be successful for both, and they will be notified that transaction is successful.

The development of the whole project was done using android studio and to keep consistency between the team members about developed code a github repository was used.

For ParkMe application we wanted to enable payment options to buyer only when he reaches the spot and once he can actually see and confirm that the parking spot is there at the marked spot to save the buyer from possible fraud. Because if the seller is fraud he will just post that he has a sellable parking spot on the application, but in real there might not be any selling spot, or the spot might be already occupied. To avoid this kind of situations the ParkMe application enables payment functionalities only when the buyer is at sellers location. This also follows the basic design principle in user oriented design "The user should be able to see only the functionalities that are required to him at the given point of time" otherwise in case all the functionalities of the application are shown to user at one point user might find the application complex and difficult to understand.

Project Description, by Activity

Map - Buyer

The buyer's map view will display the map for the buyer and all the information relevant to the purchase of a parking spot. The buyer will be able to set down a pin on the map resulting in a radius being cast around that pin looking for pins placed by the seller.

Map - Seller

The seller's map view will display the map for the seller and all the information relevant to the selling of a parking spot. The seller will be able to place a pin down in the location of the spot he/she would like to sell.

Login Page

The login page is the first screen that the user encounters, there the user will be able to login, provided they have previously registered, and access the rest of the functionality of the application. The user can also access the register page in the case that they do not already have login credentials.

Registration Page

The registration page is used to register new users to the database, here they will be able to choose enter information that will be used in the database as their

login credentials. Once successfully registered the user will the user will be taken back to the login page.

Payment

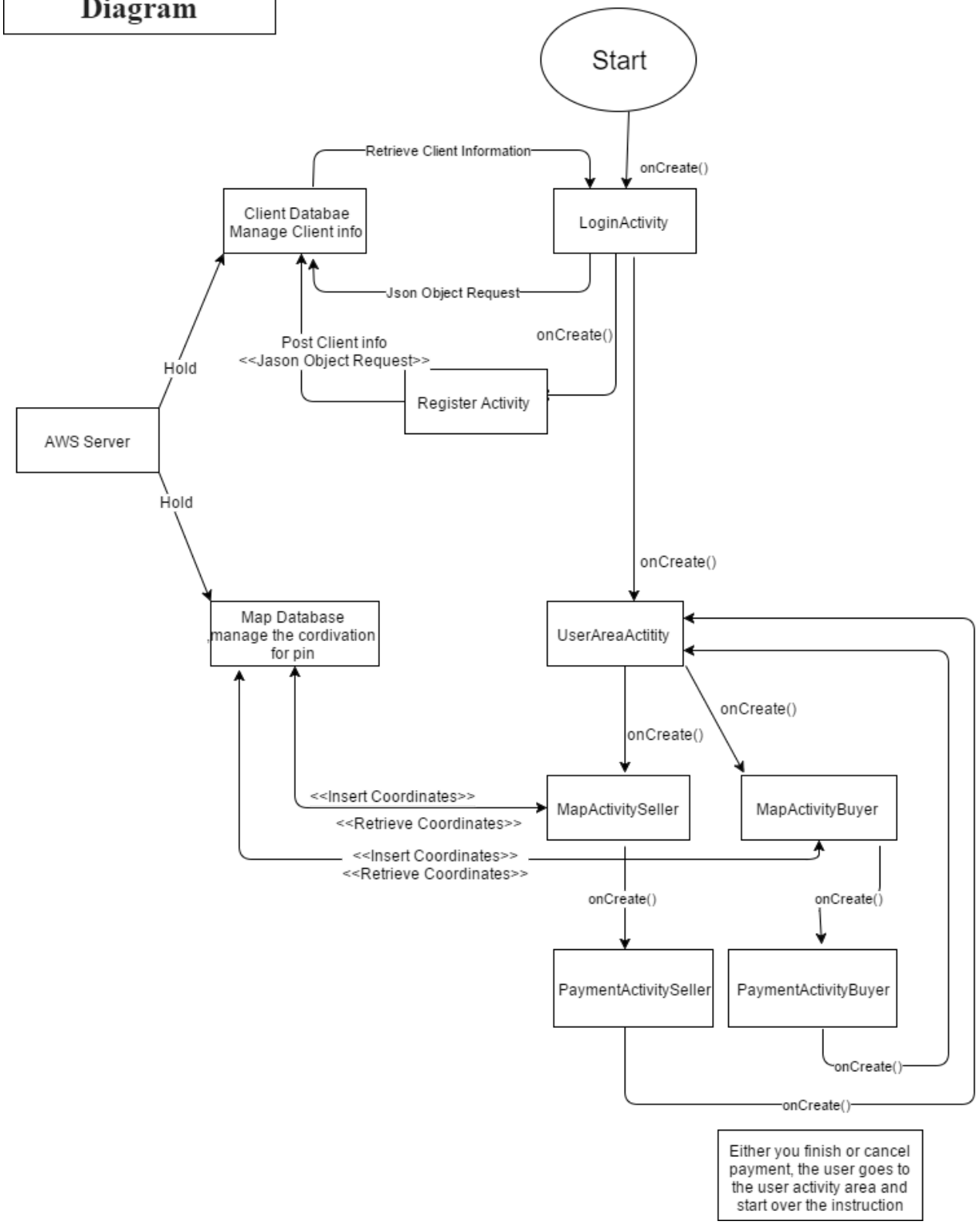
Once the buyer comes into close proximity with the seller, they will be able to initiate payment. A message will appear on both the buyer and sellers devices notifying them that they are ready to initiate the transaction.

User Area

On this page the user will be able to choose between being a buyer and seller. Depending on what the user wants to accomplish they will be able to choose to use the app as a buyer or seller. From here they will be taken to either the buyer map or seller map.

Activity Lifecycle Diagram

ActivityLifecycle Diagram



Extensibility/Flexibility of the Design :

Extensibility and Flexibility of the designs :

1. The Map Activities use external APIs, and the ParkMe application will be affected if the APIs are updated.

2. The payment activities are of modular nature, the modifications made to the way payment is handled will not affect any other functions of the application.

Constraints to design :

1. The current ParkMe application performs the activities in a sequential manner, meaning, users will have to complete specific steps in the application in order to utilize its intended features, depending on if the user is a buyer or seller.

2. Internet connection and GPS are required to use the application, without either one the applications features will not be able to run.

3. Users must be within close proximity to each other in order to initiate payment and confirm that the parking spot is sold, in the case of the buyer the device must be within the radius of the sellers device. This means that sellers can’t sell a spot that they are currently not in.

4. The ParkMe application was developed using a phone sized screen as our layout template so when using the application on a tablet device the layout may not appear as intended, responsive layout elements may be implemented in a later revision, but the focus and primary use cases are intended for mobile phone users.

User Guide

This Guide will show you how to use the ParkMe application step by step.

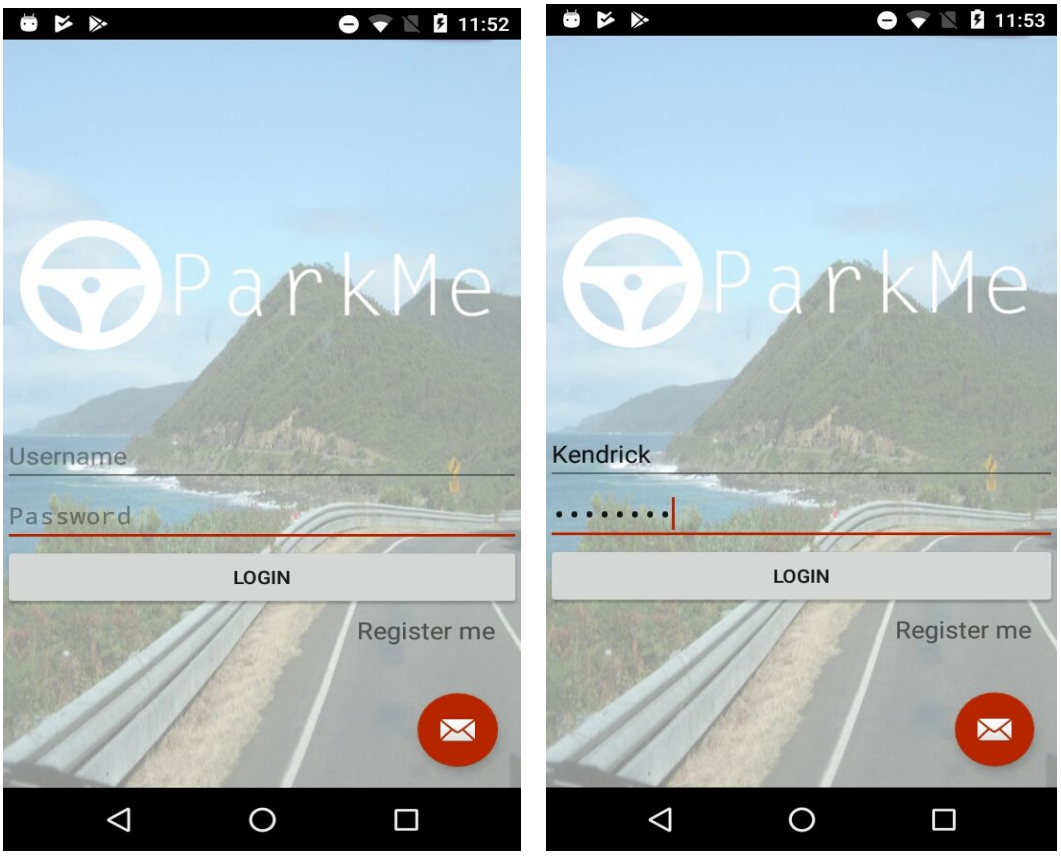
User Guide Table of Contents

1: Login/Register

Step 1A: Login

When you start using the application, you will be redirected to login page. If you have already registered to be a user of the application you can enter your username and password on this screen and start using the application.

1. Click the application icon on your android mobile phone.
2. Enter the username and password
3. Click on login button.

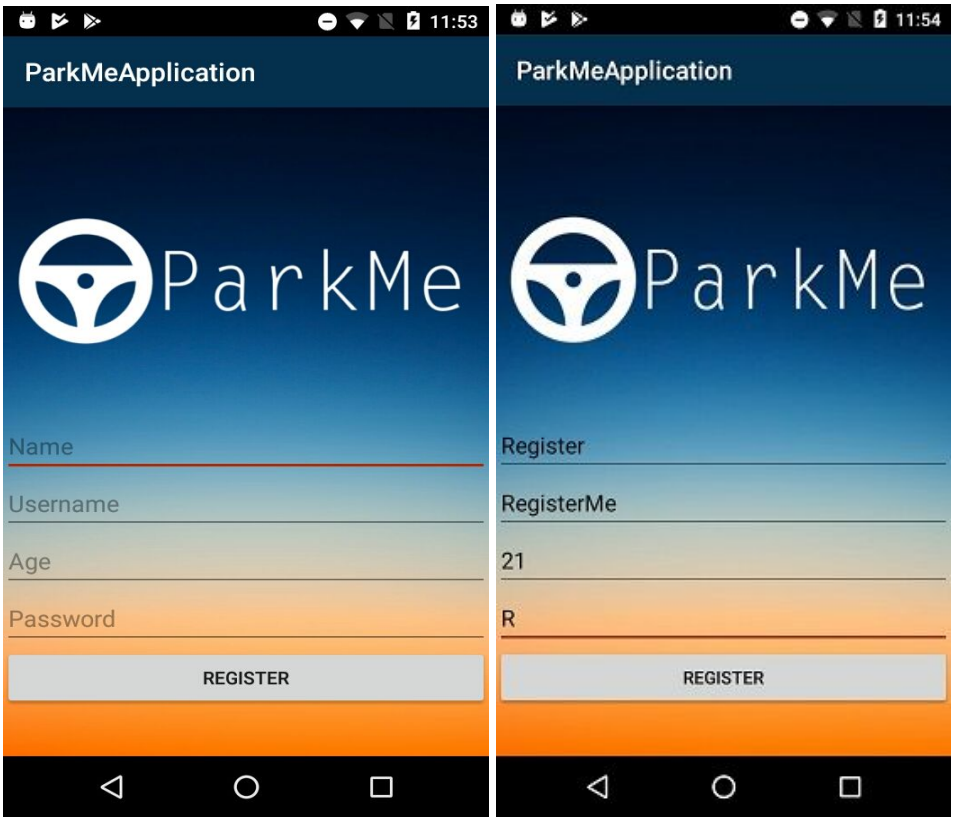


If the username and password are correct, you will be able to use application further. In case there in an error occurs during the login process, a message with “Login Error.” Will appear.

Step 1b: Registration

If the user does not have an existing account, then they have the option to click the "Register Me" button and that will take the user to the Registration page where they can create an account. All the user needs to do is enter their name, username, age, and password to create an account. Once all these information is entered, click register and the account will be created.User will be redirected back on login page to login the application.

1. On login page click on register me link.
2. On the registration page enter the details like name, username, age, password.
3. Click on register button on the page.



Step 2: Choose your role as buyer/seller in the application:

Once the User has successfully logged in or has successfully created an account, they will be redirected to a page where user will select which role they want to play: Buyer/Seller in the current transaction.

There are two options you can do in this activity:

1a. Click on buyer if you want to buy a parking spot using ParkMe application

1b.Click on seller if you want to sell a parking spot using ParkMe application.

Step 3 : What to do as a Seller/Buyer

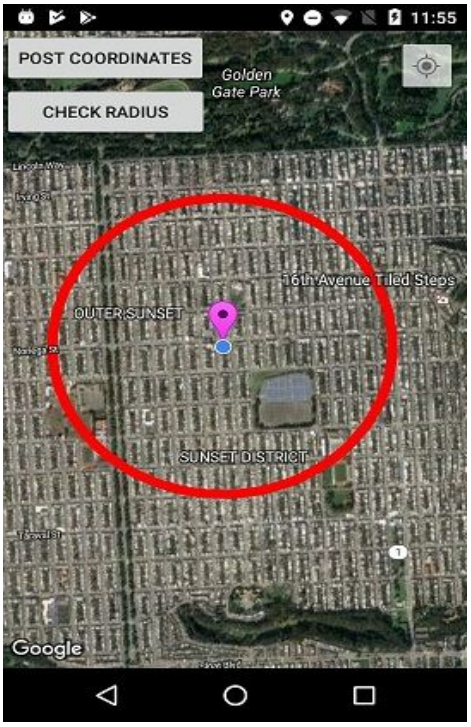
Step 3A: Seller

Sell your spot using ParkMe Application:

The user is a seller if the user clicks on the “Seller” button on the above screen. The user will then be taken to a page where he can see his current location on map. The seller can then post the sellable spot on the application.

1. Click on seller on UserActivity Page in step 2
2. Seller should be at the spot that he want to sell in the application. Only the current location of the seller's phone will be taken as sellable parking spot.

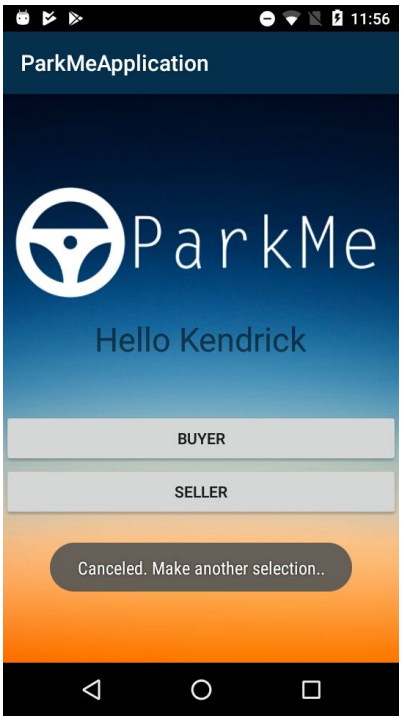
3. (Optional)Verify if the ParkMe is taking correct location that he want to sell.
4. (Optional) Check in what radius his posted spot will be visible to buyers by clicking on check radius button.



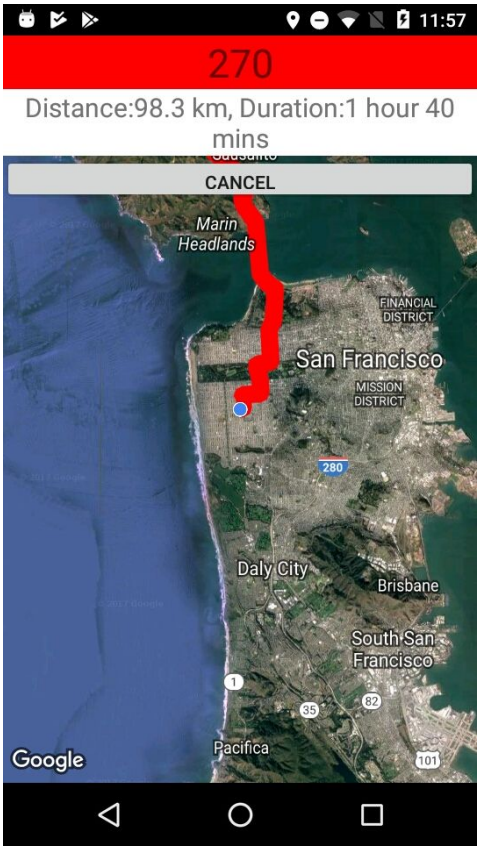
5. Click on Post Coordinates button to post the spot on ParkMe application.

6. (Informational)The spot is posted on the application only for certain amount of fixed time, after that the spot will expire from the application. Once the seller posts his spot the timer (300 seconds) will start. This shows how much time the spot will be posted in the database before it gets deleted.

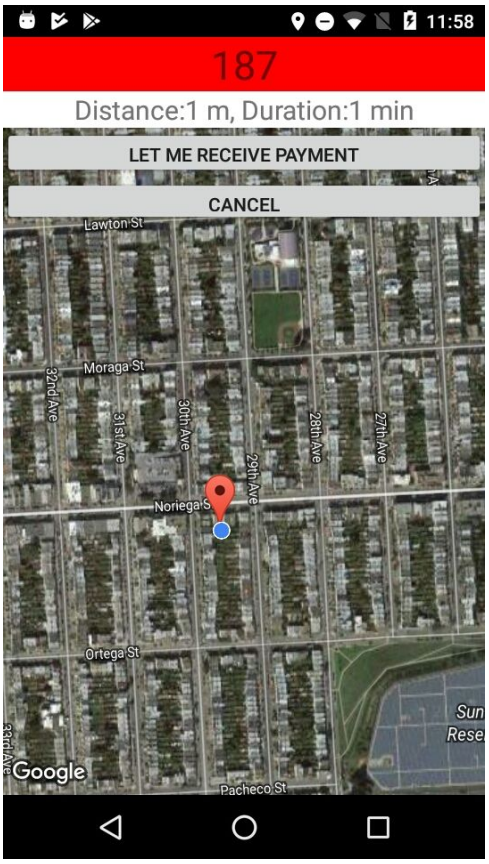
7. If the seller want to remove the posted spot from ParkMe application before the timer expires, the seller can simply click on cancel button on the above screen. The user will be redirected to the User Area Activity with the notification that the user has canceled the spot.



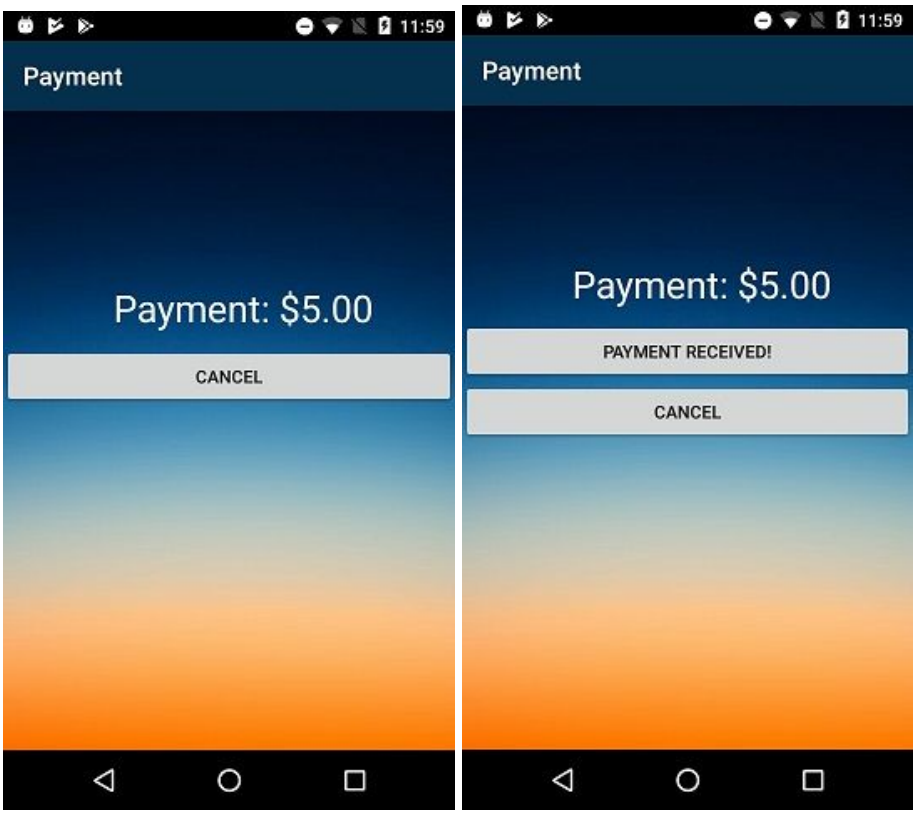
8. If the seller has been selected by a Buyer, the seller would see see the distance and time it takes for the buyer to reach the seller.



9. Once the buyer has reached the seller, the seller will see a button “Let me Receive Payment” and “Cancel” button. Cancel button will cancel the transaction and “Let me Receive Payment” will allow the user to go to the payment screen to pay for the spot.

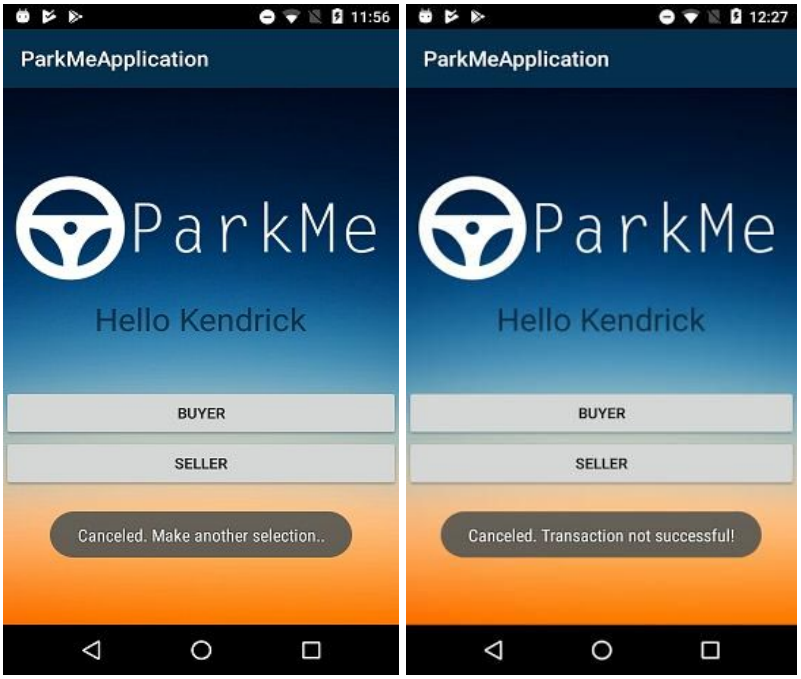


10. After clicking “Let me Receive Payment” button, the buyer will go into the payment screen. The seller will then wait for the buyer to pay for his spot and the seller will wait for the payment. If the payment as been received, a button called “ Payment Received!” will pop up on the screen.



11a.Once the seller confirms that he has received payment, both the buyer and seller will be notified that transaction has been successful.

11b. If the user declines the payment, the seller is redirected back to the UserAreaActivity page with a notification saying “Cancel. Make another selection.” Or “Canceled. Transaction not successful!



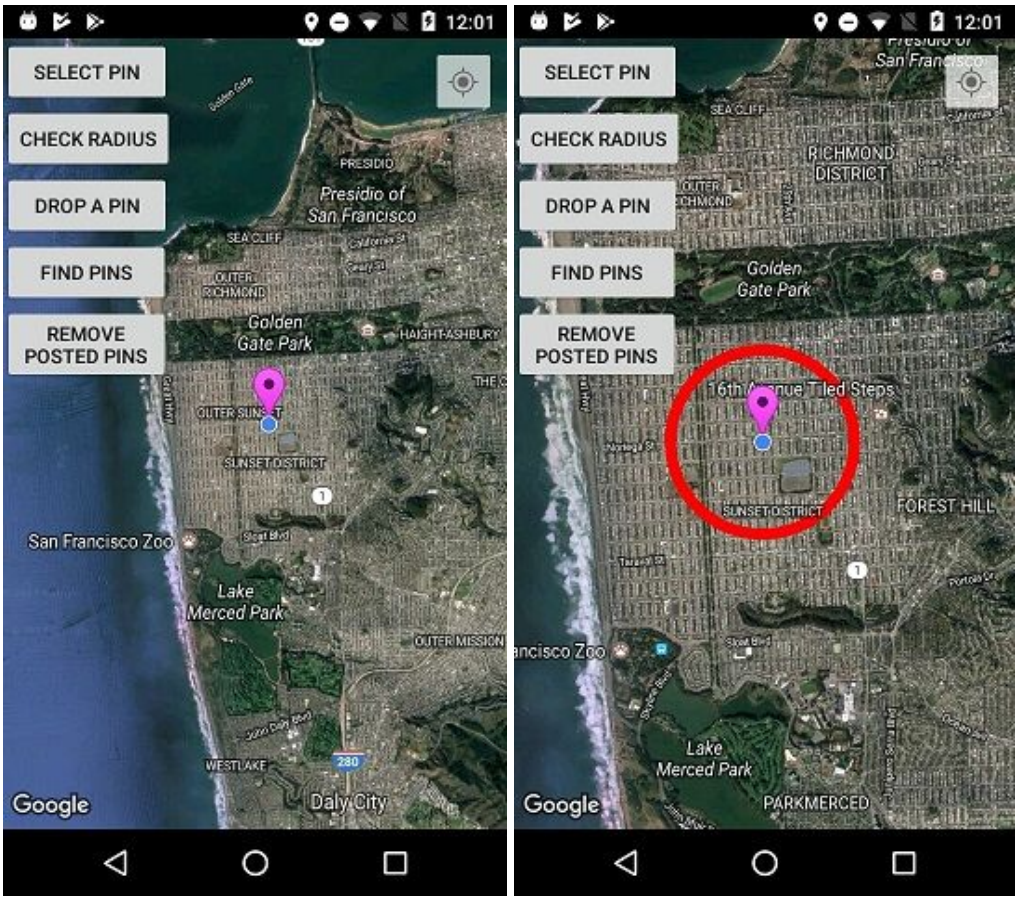
Step 3B: Buyer

Buy a parking spot using ParkMe application:

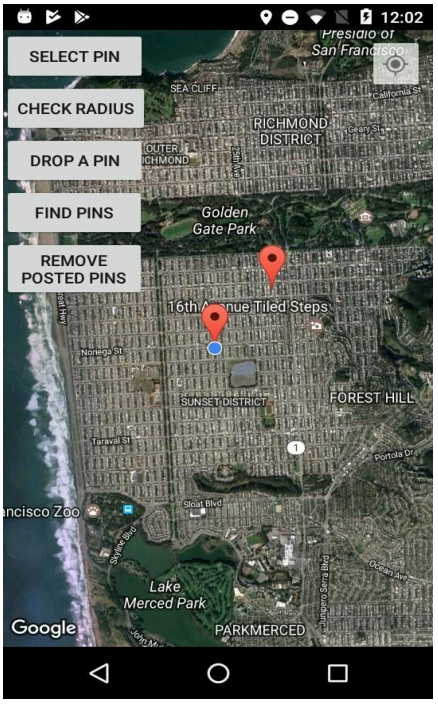
If the user would like to be the role of buyer, the buyer will click on the Buyer button at the role selection (UserAreaActivity) page after login, and the user will become the buyer and go into MapActivityBuyer.

1. The pink pin shown in below map is the current location of the buyer.

2. By clicking on button Check radius, the buyer can check the area from which the seller pins will be visible to buyer.

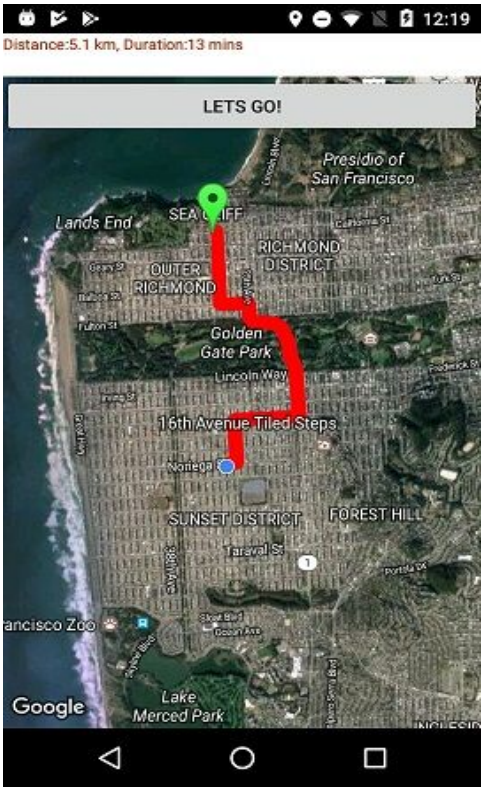


3. User can drop a pin on map, using “Drop A Pin” button, which will help them drop a pin in the area where they want to buy a parking spot.

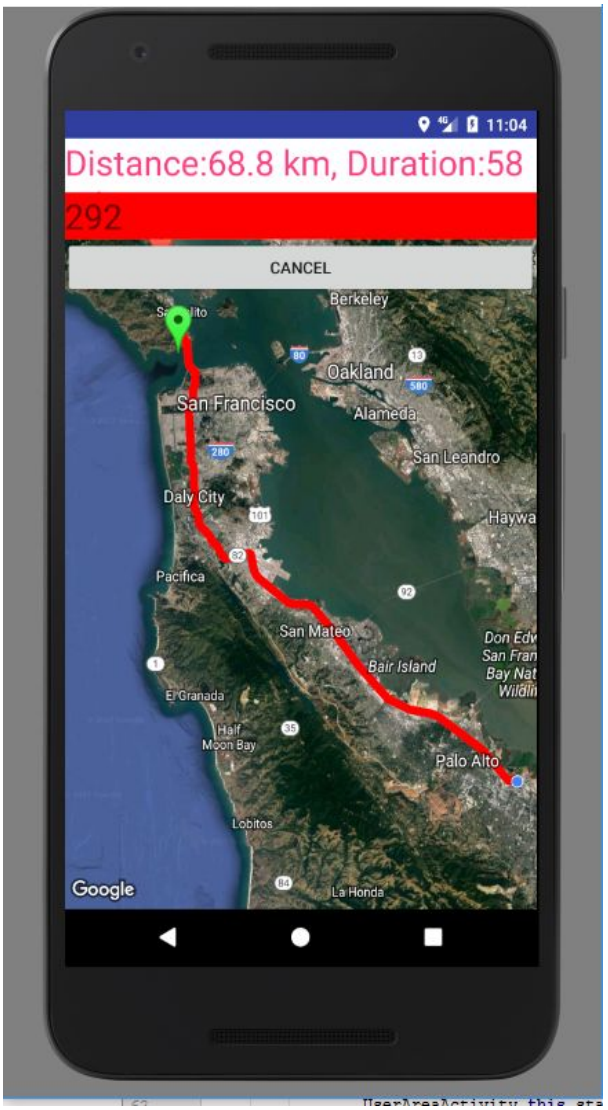


4. The buyer will be able to find the spots near his dropped pin, by clicking on “Find pins”. If there are sellable spots available, green pins will show up.

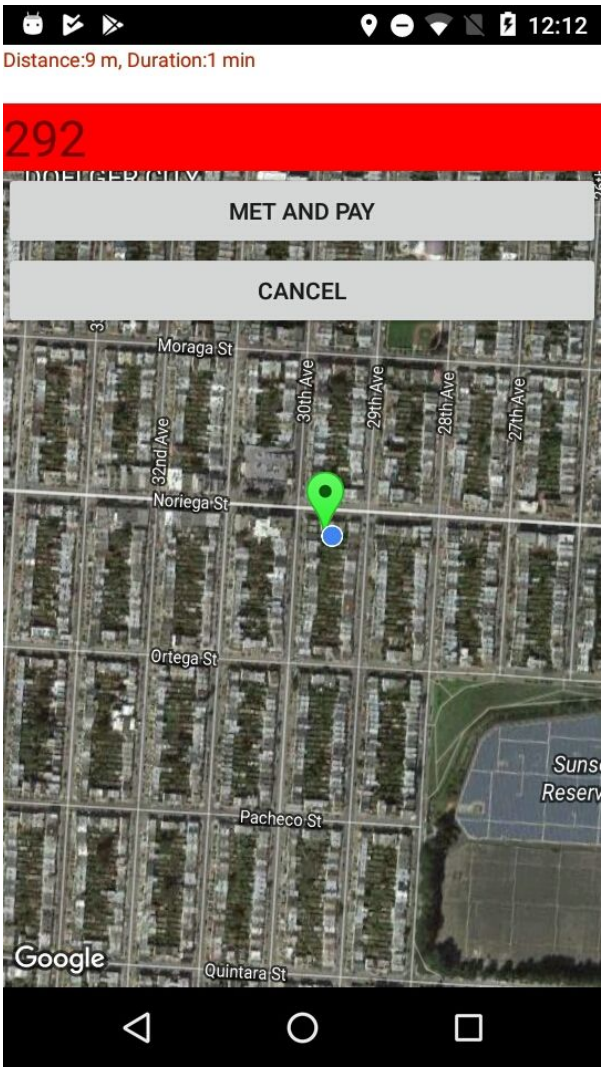
5.Once the user finds a parking spot they like, they can hit the Select Pin button and the application will show the distance and time difference between the user’s current location and the parking spot.



6. The user will then hit Let's Go and the timer in the red block will start counting down, telling the user they have that amount of time to reach their destination before the spot is gone. If the timer hits zero or if the user hits the cancel button, they will be redirected back to the UserAreaActivity page.

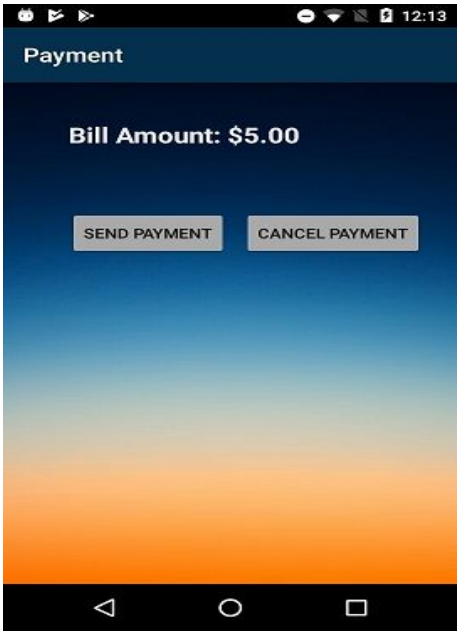


7. Once the Buyer reaches the Seller's destination they can hit the "Met and Pay" button to send the seller the payment, Or they can cancel the transaction.



8. If the buyer chooses to hit cancel, then a message will show up saying "Canceled. Seller will be notified."

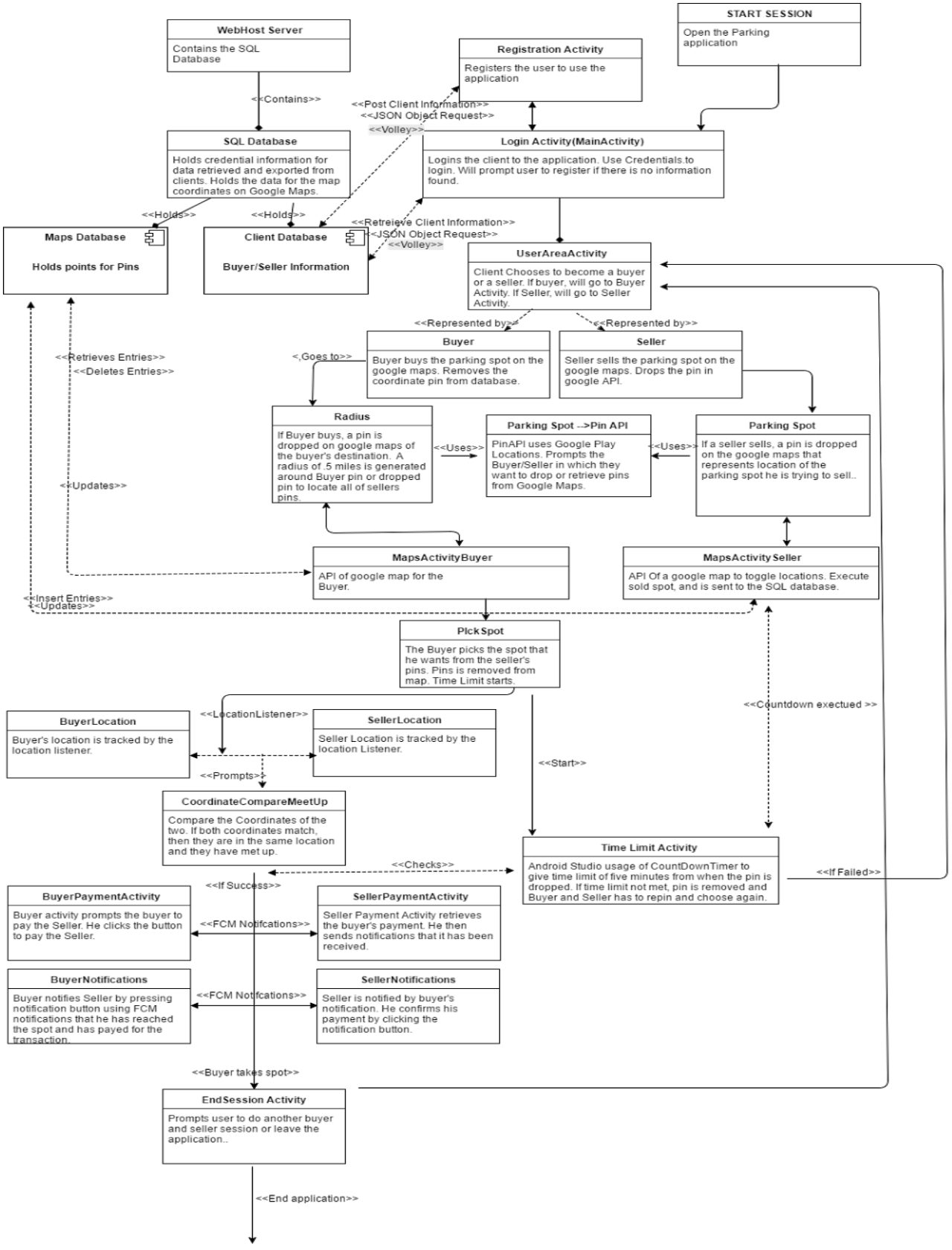
9. Once the user reaches their destination, the application will redirect the user to the payment page. To pay the seller, the buyer can hit the Send Payment button to send the payment. To cancel the payment the Buyer can hit the Cancel Payment button.



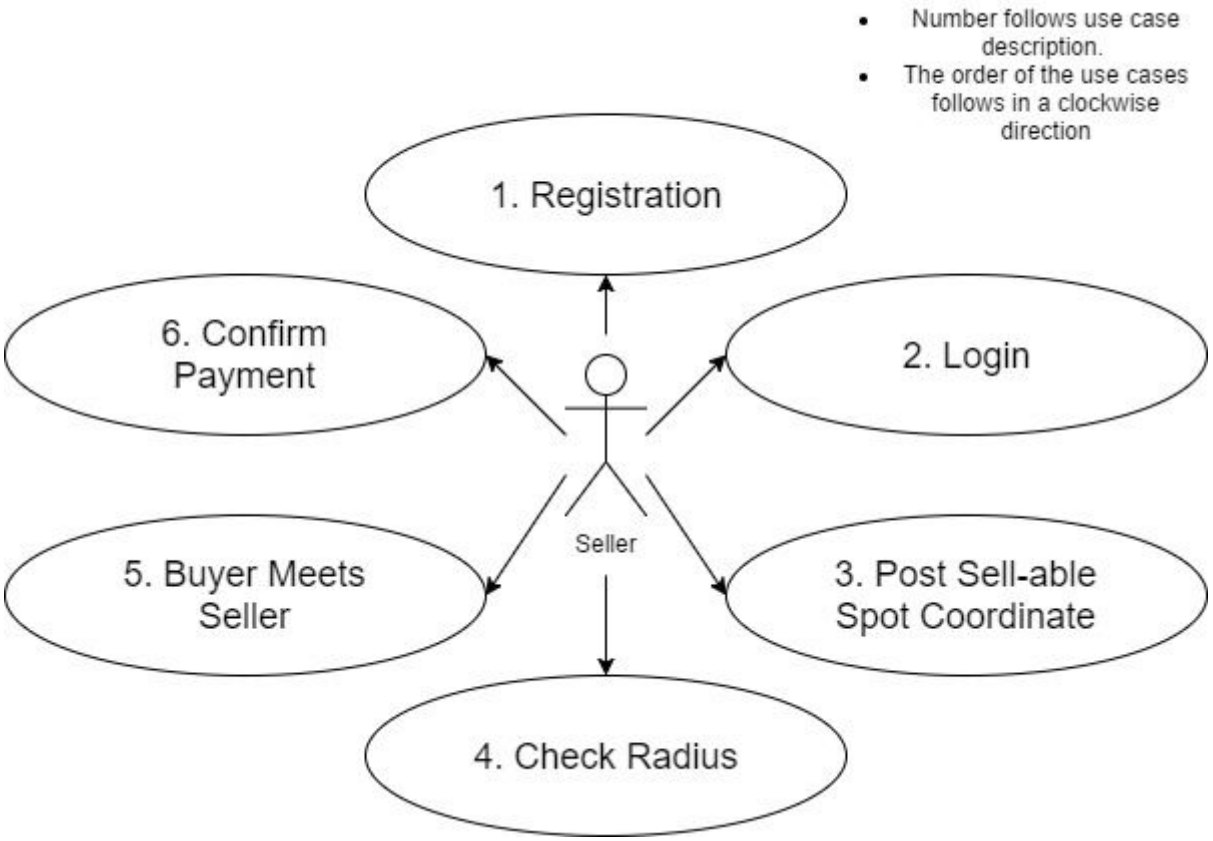
9. If the payment is successful, the application will redirect the user back to the UserAreaActivity page with the notification saying "Seller received Payment. Transaction Successful!"

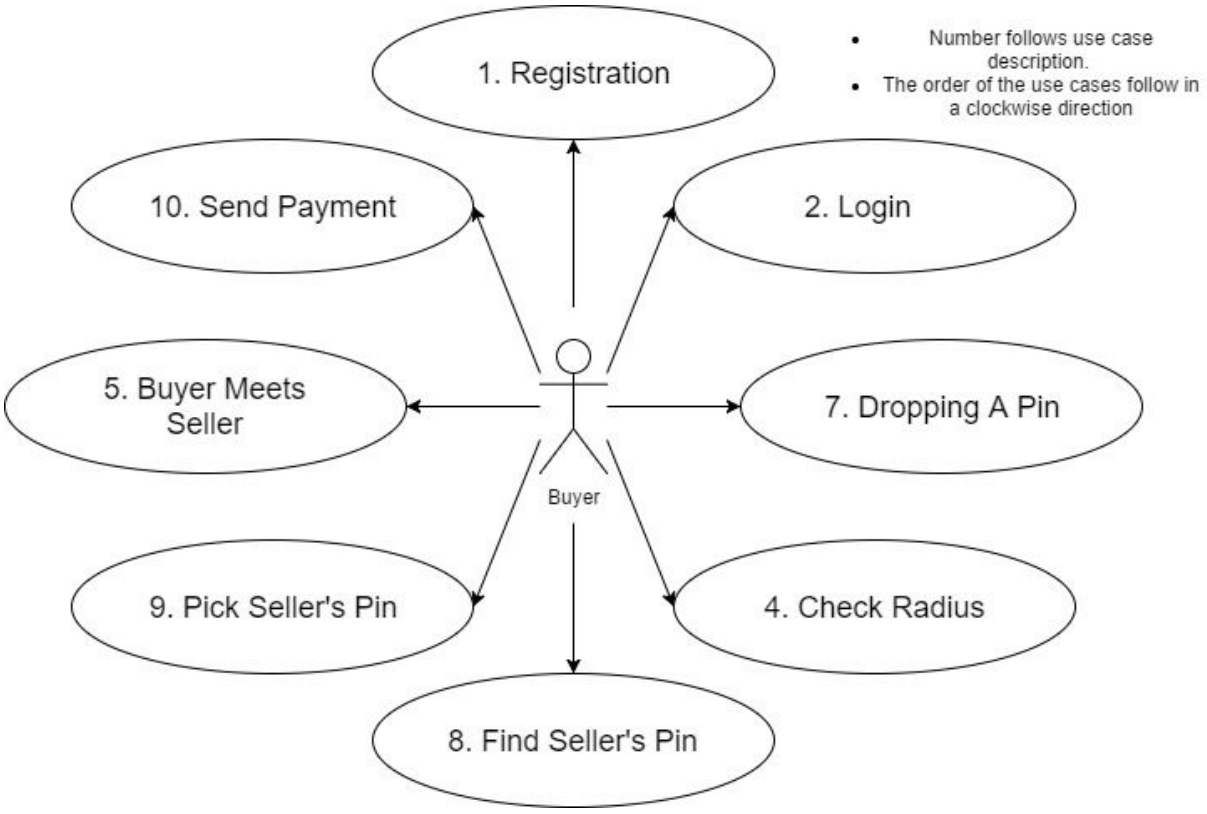
10.If the payment failed or if the user cancels the payment, the user is directed to the UserAreaActivity page with the notification saying “Canceled. Transaction not successful!.”

Conceptual Model



Use Case Diagram





Use Case Descriptions:

Use case #	1.
Use case name	Registration
Summary	User who want to create new account on ParkMe app enters their details on the registration page to create an account.
Actor	Seller/Buyer(Any User)
Precondition	NA
Description	-On Login page click on register me link -User enter the details like name, username, age, password. -Click on register button on Registration page.
Alternative	N.A.
Post-condition	The User account is created for ParkMe application.

Use case #	2.
Use case name	Login
Summary	User enters their username and password to login into the application.
Actor	Seller/Buyer(User)
Precondition	User must have registered account.
Description	- Enter username and password to login - Click on Login button
Post-condition	The user will be logged in the application.

Use case #	3.
Use case name	Post sellable spots coordinates.
Summary	The seller want to post his sellable spot in the application.
Actor	Seller
Precondition	-The user must be logged in to the application. -The user must have selected role as seller on UserAreaActivity page.
Description	- Click “Post Coordinates” after the seller map page. - Timer of 5 minutes will start counting down before the user has to redo the process.
Post-condition	The sellables parking spots details are posted in the application and the buyers will be able to view this spot now.

Use case #	4.
Use case name	Check Radius
Summary	The seller can check the area in which his pin will be visible. The buyer can check the area in which app will find parking spots for him.
Actor	Seller/buyer
Precondition	In case the user is Buyer: Drop a pin In case user is Seller:None(The application takes seller's current location)
Description	- Buyer: Check the radius of the area around their dropped pin only the sellable parking spots in this area will be visible to buyer. - Seller: Check the area in which his sellable parking spot will be visible .
Post-condition	Red circle with 1 mile radius around the pin is displayed in the application.

Use case #	5.
Use case name	Buyer meets seller
Summary	Buyer arrives at the spot that he reserved on the application.
Actor	Seller/Buyer
Precondition	Buyer must have reserved the parking spot (selected a parking spot) from the ParkMe application.
Description	The buyer goes to the location of the seller to meet up.
Post-condition	As soon as the buyer reaches the seller’s location the payment functionality of the application will be enabled for both: Buyer: The button to switch activites from MapsBuyer to

	PaymentBuyer activity “meet and pay” will be enabled. Seller: The button to transfer from SellerMap to SellerPayment activity “meet and pay” will be enabled.

Use case #	6.
Use case name	Confirm payment
Summary	Seller receives a payment from the buyer and he confirms that he has received payment.
Actor	Seller
Precondition	-The buyer and seller have met each other. -The buyer has sent the payment to seller.
Description	The seller receives a payment from buyer and the seller will have to confirm that the seller has received payment by clicking on “Received Payment”
Post-condition	- If the payment is successful, a notification saying “Payment successful” will appear. - If payment falls through, then it will be considered canceled and a message saying Canceled will appear.

Use case #	7.
Use case name	Dropping a pin
Summary	Buyer can drop a pin in the area where is expecting to find a parking spot.

Actor	Buyer
Precondition	The user is logged in the application and selected role as a buyer on UserActivity Page.
Description	- Click “Drop A Pin” on buyer side map. - Click on map wherever you want to drop a pin
Post-condition	The Pin is dropped on the buyer’s side map.

Use case #	8.
Use case name	Find seller’s pin
Summary	Buyer wants to find available parking spot near his desired area.
Actor	Buyer
Precondition	The buyer has dropped a pin in the area where he want to find spots.
Description	- Click “Find Pin” on buyer side map.
Post-condition	All the sellers pin within 1 mile radius from buyers dropped pin will be shown on the map.

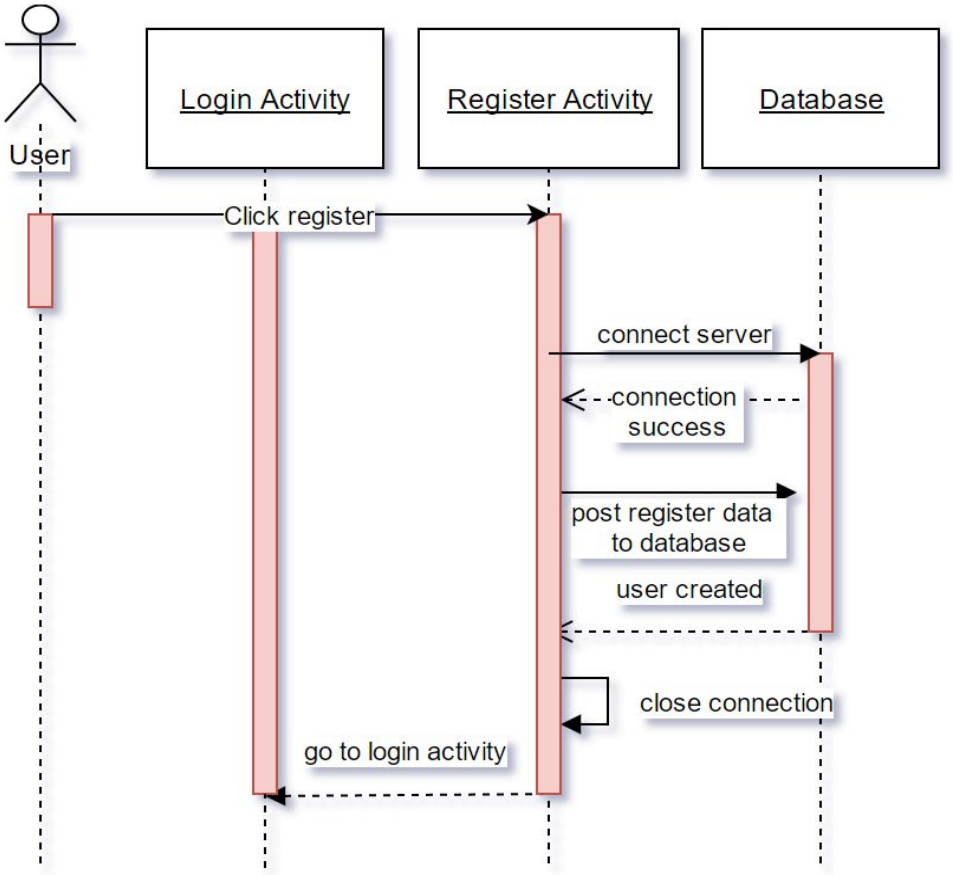
Use case #	9.
Use case name	Select seller’s pin to reserve the seller’s parking spot.
Summary	The buyer will choose the specific sellers pin most suitable for them from the options displayed on map.
Actor	Buyer
Precondition	The buyer must have performed drop a pin to point the location where he want to find parking spot.

	The buyer must have performed find pins action to find all the pins within 1 mile radius from his dropped pin.
Description	- Click “Select Pin” on the buyer’s side map.
Post-condition	The ParkMe application will display the distance from sellers pin to current location, its approximates time to reach and the shortest route suggested by google.

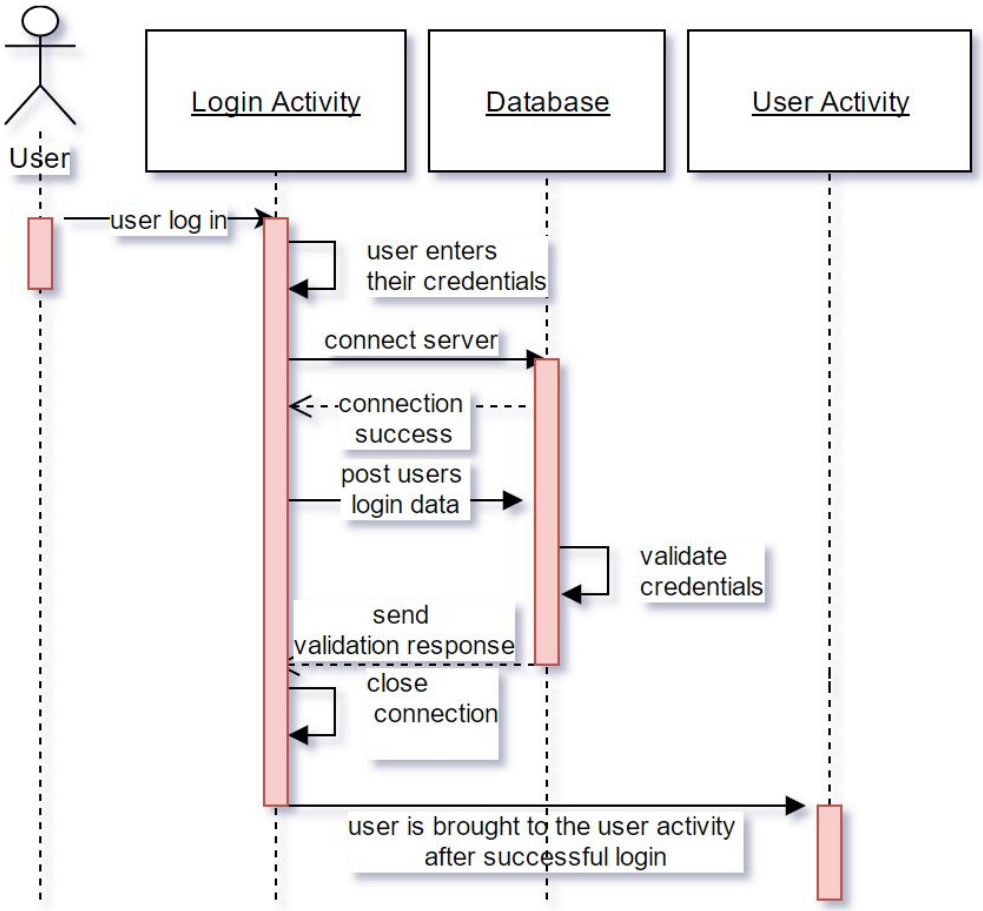
Use case #	10.
Use case name	Send Payment
Summary	Buyer meets up with the seller and starts the procedure to pay him or her.
Actor	Buyer
Precondition	The buyer has selected sellers spot. The buyer must have reached seller’s location.
Description	- Click “Met and pay” to start the payment activity. - Buyer clicks “Send payment” to send $5 to the seller. - **OR** Buyer can click “Cancel” to cancel the whole process completely.
Post-condition	-The seller’s side page will be updated and will be asked for confirm that seller has received payment.

Sequence Diagrams:

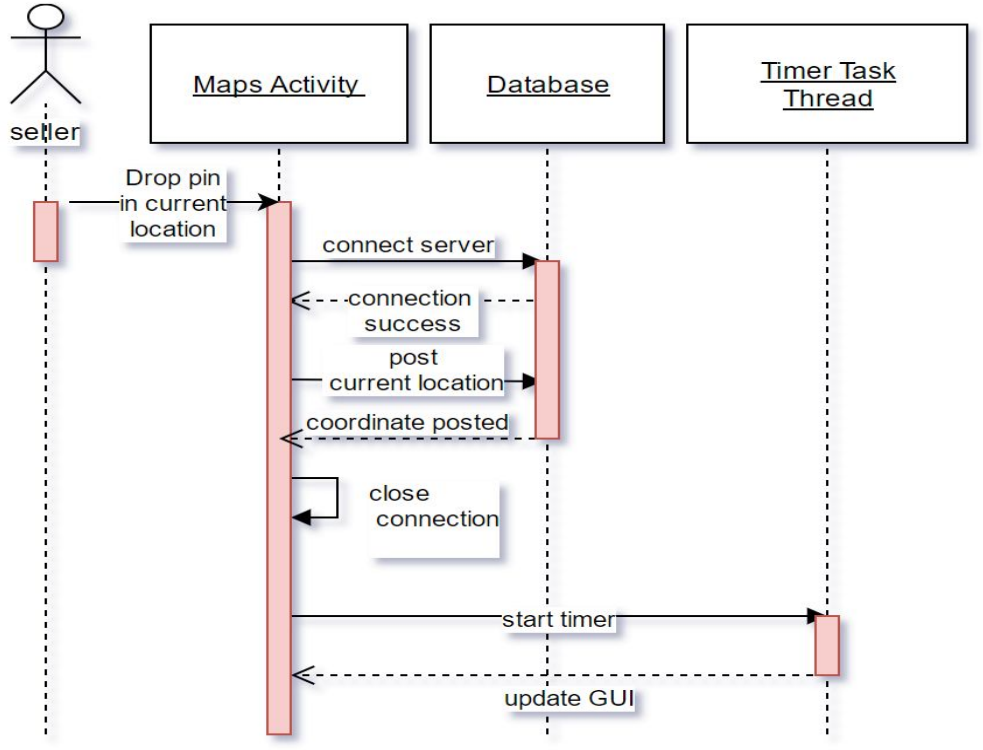
1. **Seller as a user sequence diagrams:**



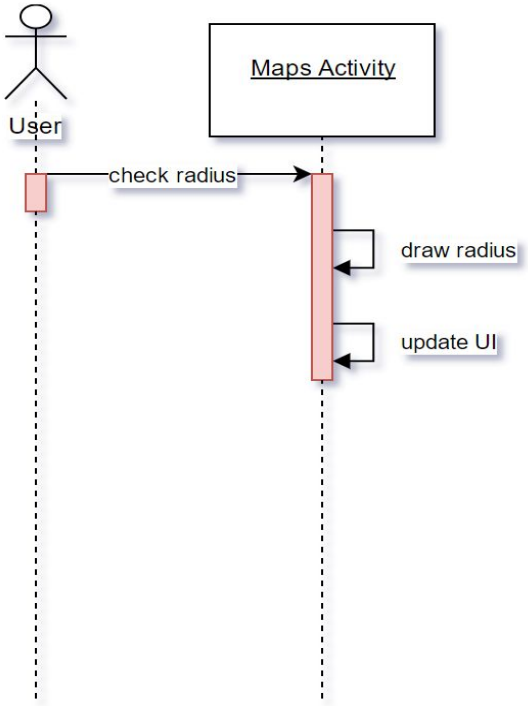
Seller: Register diagram



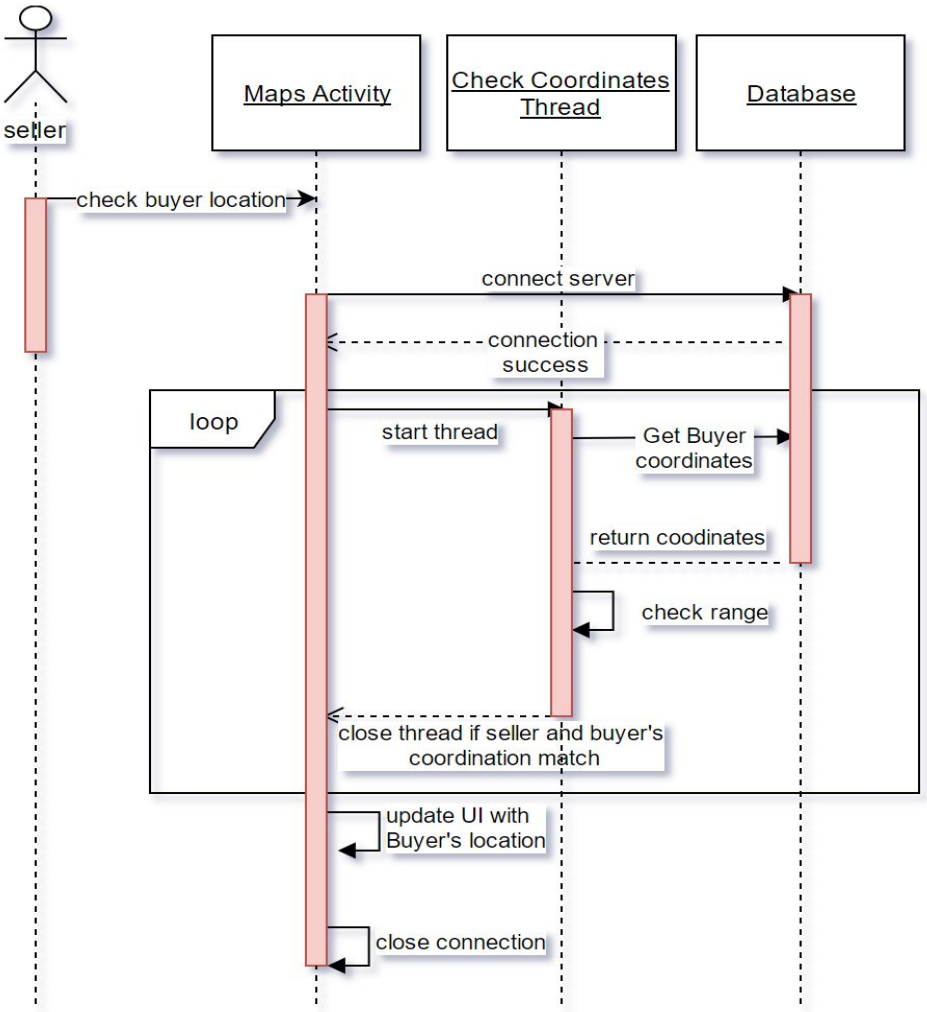
Seller: Login Diagram



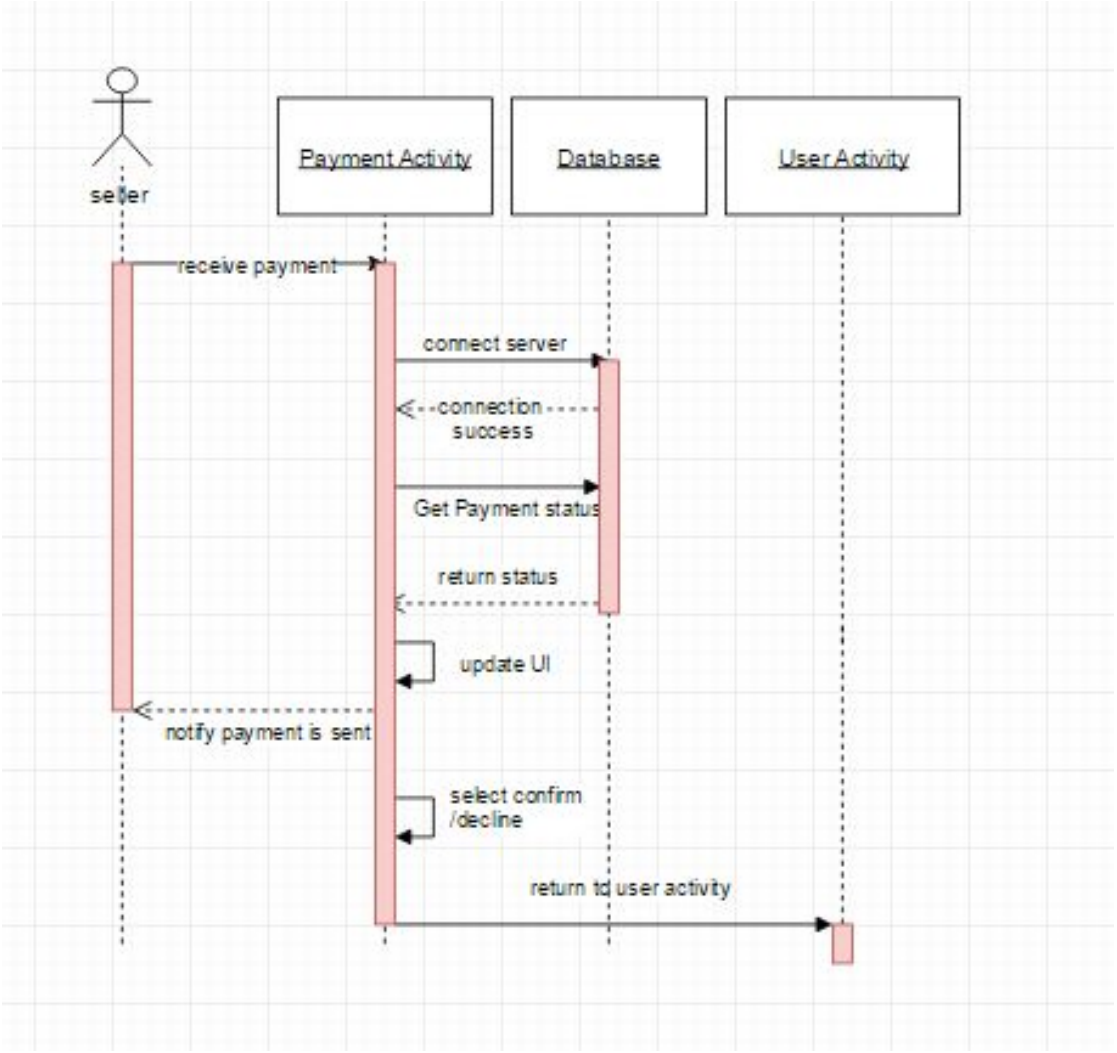
Seller: Post Coordinates Diagram



Seller: Check Radius Diagram

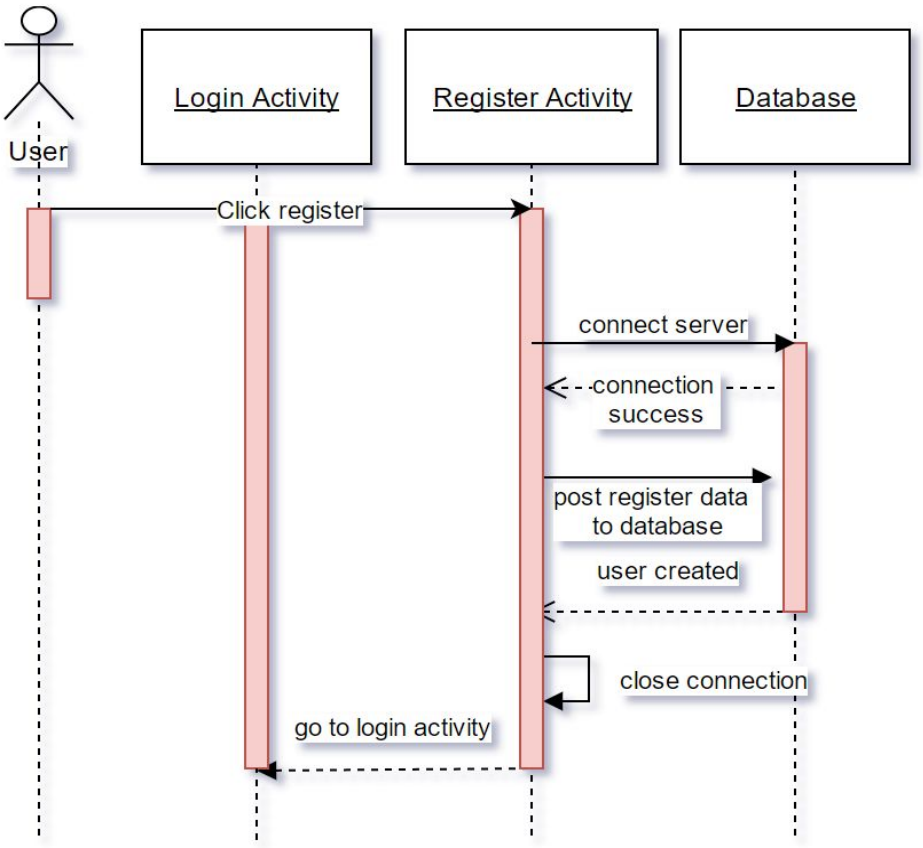


Seller: Buyer Meets Seller Diagram

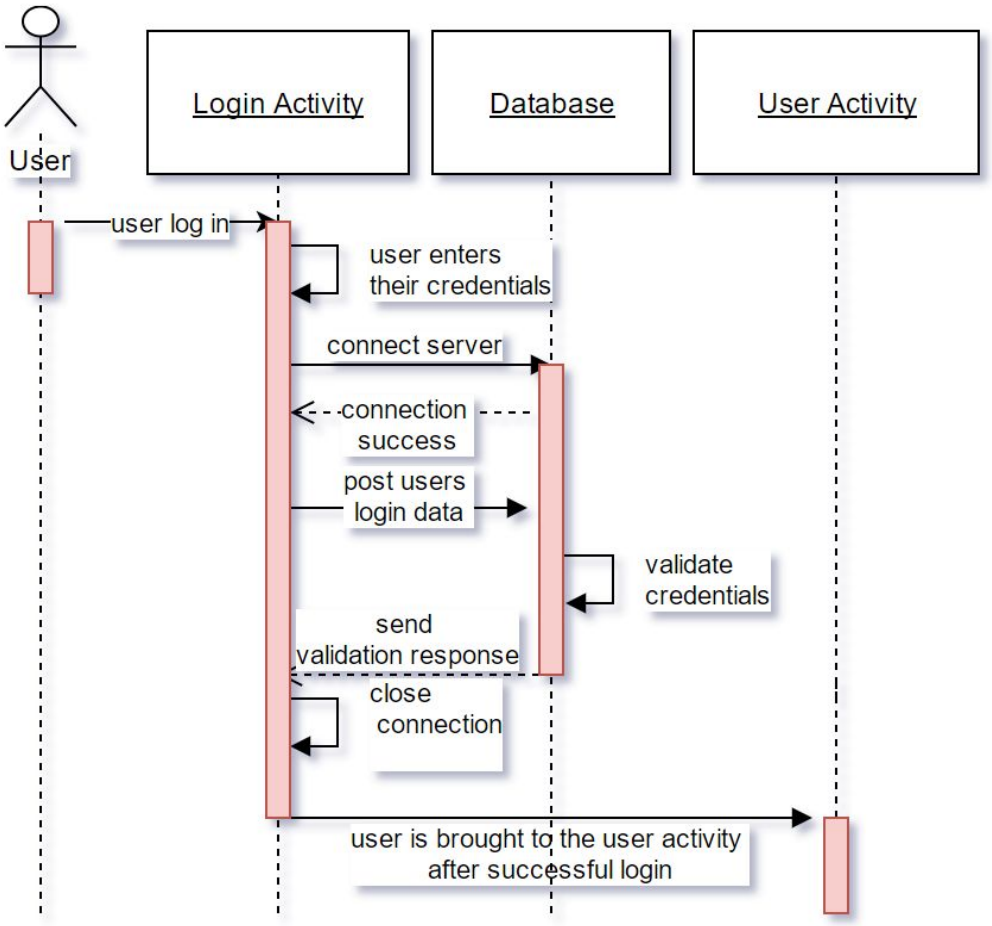


Seller: Confirm Payment Diagram

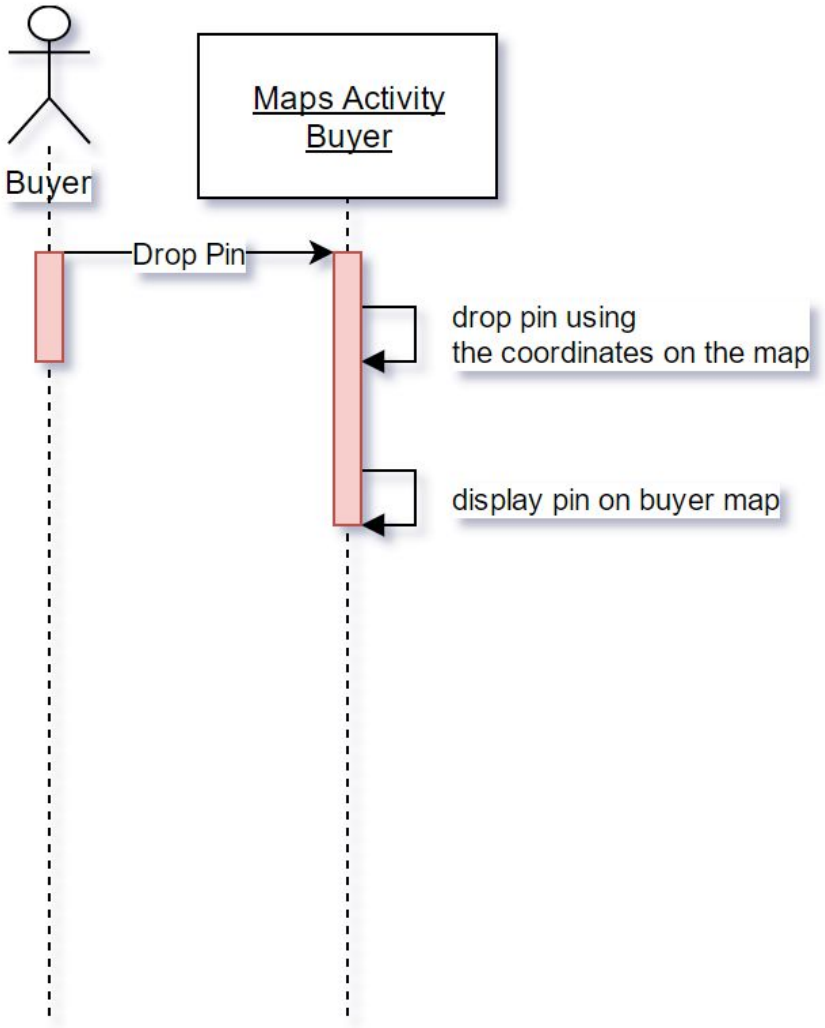
2. Buyer as user use case diagrams



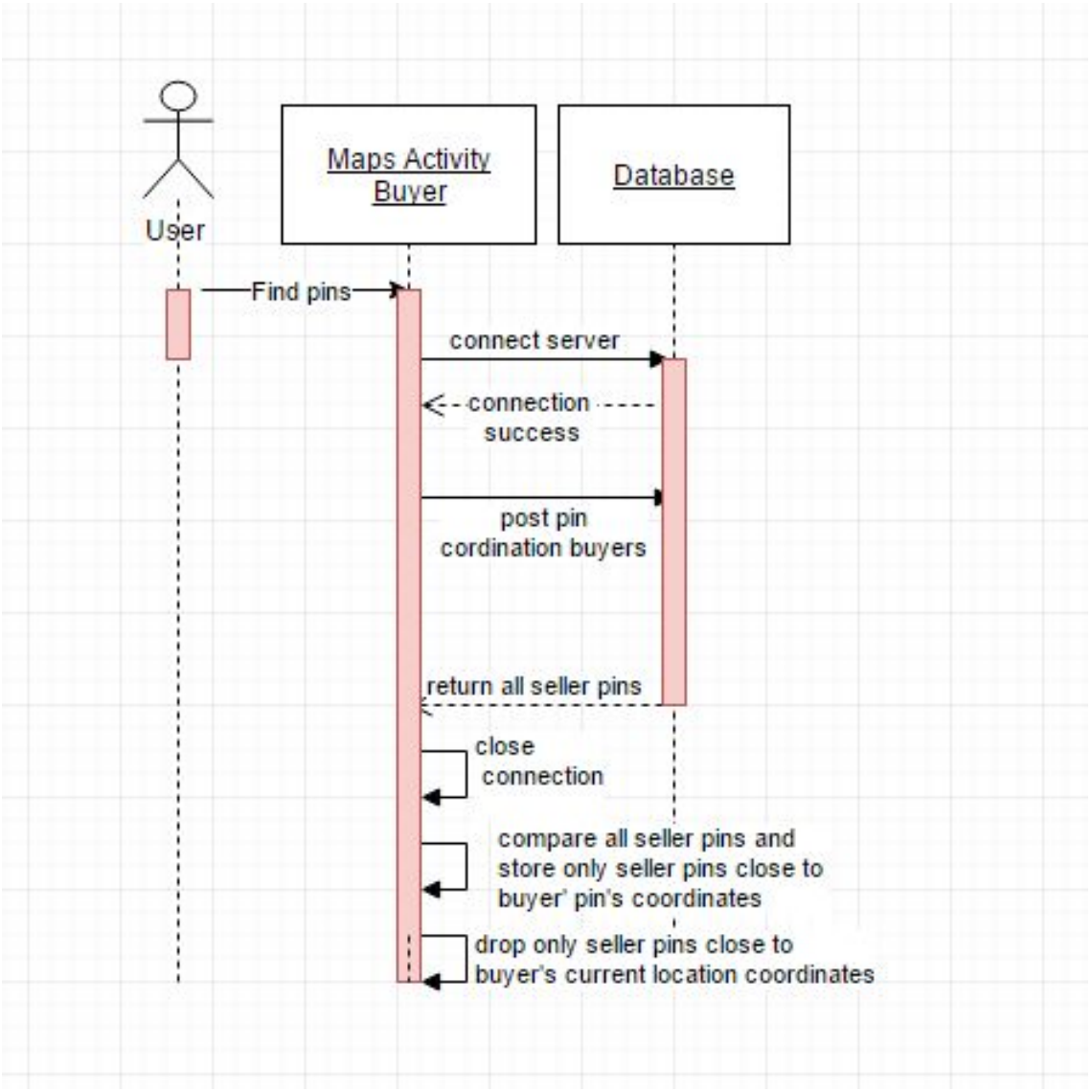
Buyer: Register diagram



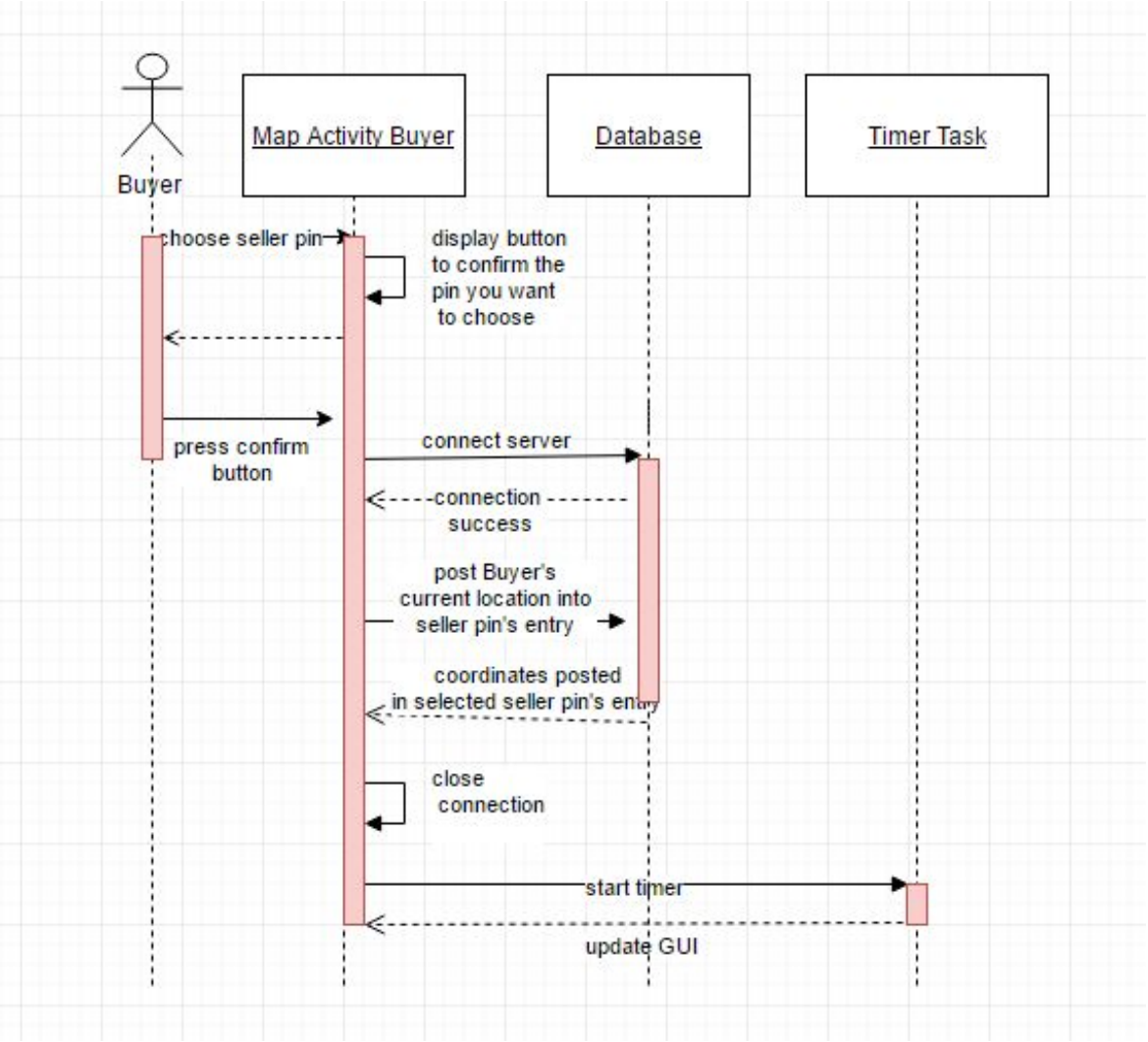
Buyer: Login Diagram



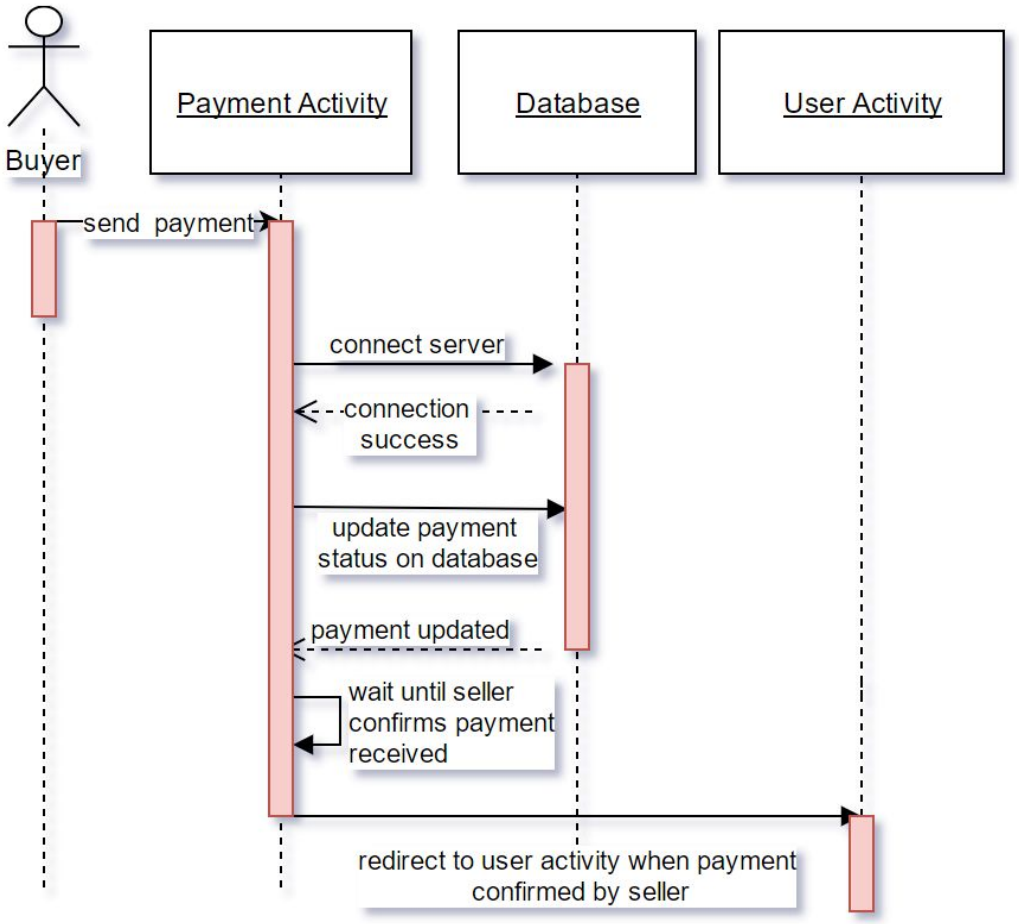
Buyer: Drop a Pin Diagram



Buyer: Find Pins Diagram



Buyer: Pick Pins Diagram



Buyer: Send Payment Diagram

Design Overview

1.Model View Controller:

- Introduction of Model View Controller:

Model View Controller commonly known as MVC is an architecture pattern used for object oriented programming. Basically MVC has 3 type of objects, Model which is an application object. The view is used for presentation and user sees the view object. The controller is responsible for controlling how the user interface is reacting to user actions.

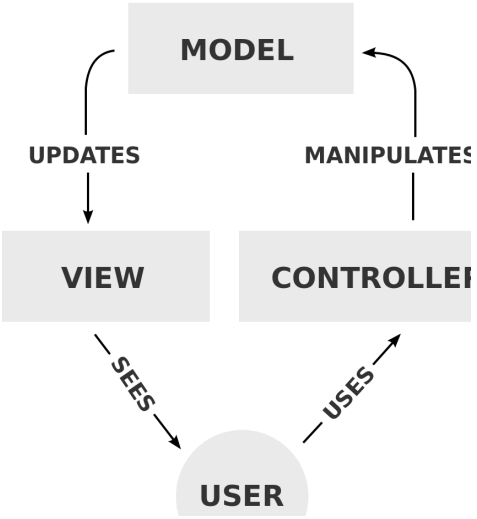


Figure1: Design Pattern: Model view Controller

- MVC pattern in ParkMe Application:

In ParkMe application, **Model** is database of ParkMe application which is used to store the user credentials, the location coordinates for map locations where the parking is available. In Android application each front end screen is specified by xml file which are basically **View** from Model View controller architecture. Figure2 shows activity_main.xml file from ParkMe application.

```xml
<?xml version="1.0" encoding="utf-8"?>
<android.support.design.widget.CoordinatorLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.example.joseph.parkmeapplication.MainActivity">

    <android.support.design.widget.AppBarLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:theme="@style/AppTheme.AppBarOverlay">

        <android.support.v7.widget.Toolbar
            android:id="@+id/toolbar"
            android:layout_width="match_parent"
            android:layout_height="?attr/actionBarSize"
            android:background="?attr/colorPrimary"
            app:popupTheme="@style/AppTheme.PopupOverlay" />

    </android.support.design.widget.AppBarLayout>

    <include layout="@layout/content_main" />

    <android.support.design.widget.FloatingActionButton
        android:id="@+id/fab"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="bottom|end"
        android:layout_margin="16dp"
        app:srcCompat="@android:drawable/ic_dialog_email" />

</android.support.design.widget.CoordinatorLayout>
```

Figure 2: View: activity_main.xml from ParkMe application

Controller. Each front end activity has associated java class with it which acts as a controller and controls the logic of that activity. The Activity could be thought like a bundle of both controller and view because it has something like a outlet which is connected to the specific parts of the view. And in this way, the controller can get information from view and call the model to do some proper actions. Figure 3 shows controller for MainActivity(MainActivity.Java) which is loading its view (activity_main.xml)

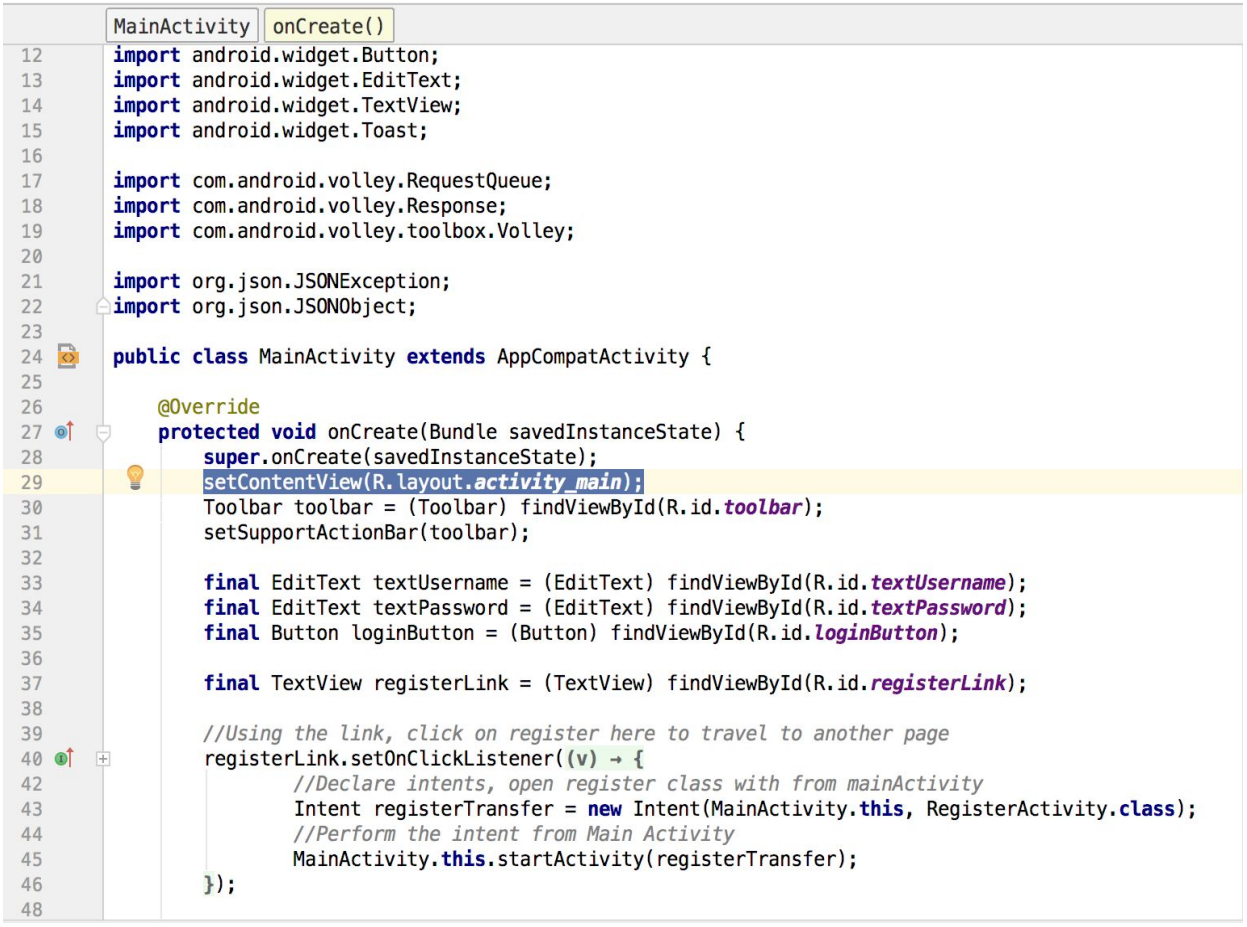


Figure 3: Controller: Mainactivity.java controller loading its view activity_main.xml

2. Mediator Pattern:

- Introduction to Mediator Pattern:

The mediator pattern is a behavioural pattern which manages the communication between the classes in the program. In mediator pattern, the communication between the objects is encapsulated within a mediator object. Object/classes do not communicate with each other directly, instead communicate through a mediator. Mediator promotes loose coupling by keeping objects from referring to each other explicitly, and it lets you vary their interaction independently.

- Reasons for introducing mediator pattern:

If we use the mediator pattern the objects don't communicate with each other directly which promotes loose coupling and better structure of the code.

- Mediator Pattern in ParkMe Application:

The mediator pattern is not implemented in ParkMe application yet but we plan to implement it in future so the switching between the 2 activities happens with the help of a mediator. In future for ParkMe application we want to have a mediator

object which will hold the reference for each activity. When one activity will want to switch to another, it will inform the mediator object. The mediator will switch to the desired activity based on the signal sent.

3.Observer pattern:

- Introduction to Observer pattern:

The Observer pattern defines a one-to-many dependency between objects. When one object changes state, all of its dependents are notified and updated automatically.

- Reason for Introducing Observer pattern:

Observer pattern can be used to respond to user input.

- Observer pattern in ParkMe Application:

The best part that represents the observer pattern is when a pin is dropped from the Buyer's map. The buyer pin is the user input that is saved on the map, and is repeatedly checking for seller pins in the radius circle. The buyer pin is constantly updated, and can check new locations for different seller pins that is within the buyer pin's radius. Figure 4 is the seller pins located depending on the radius of the dropping of pins from the Buyer. Figure 5 is assigning new radius from the dropping of the pins.

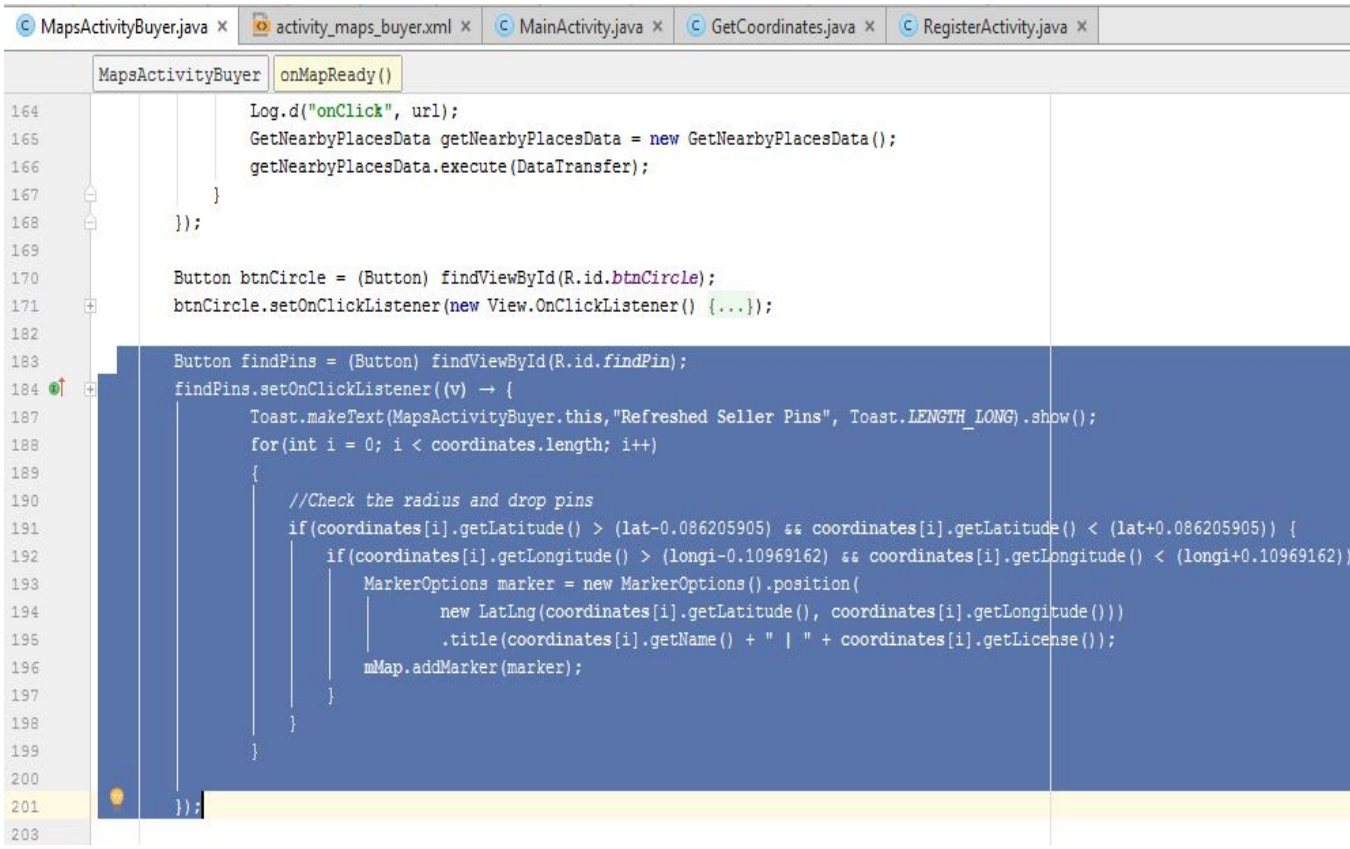


Figure 4: FindPins button to locate the seller pins. It is dependant on where the pin is dropped.

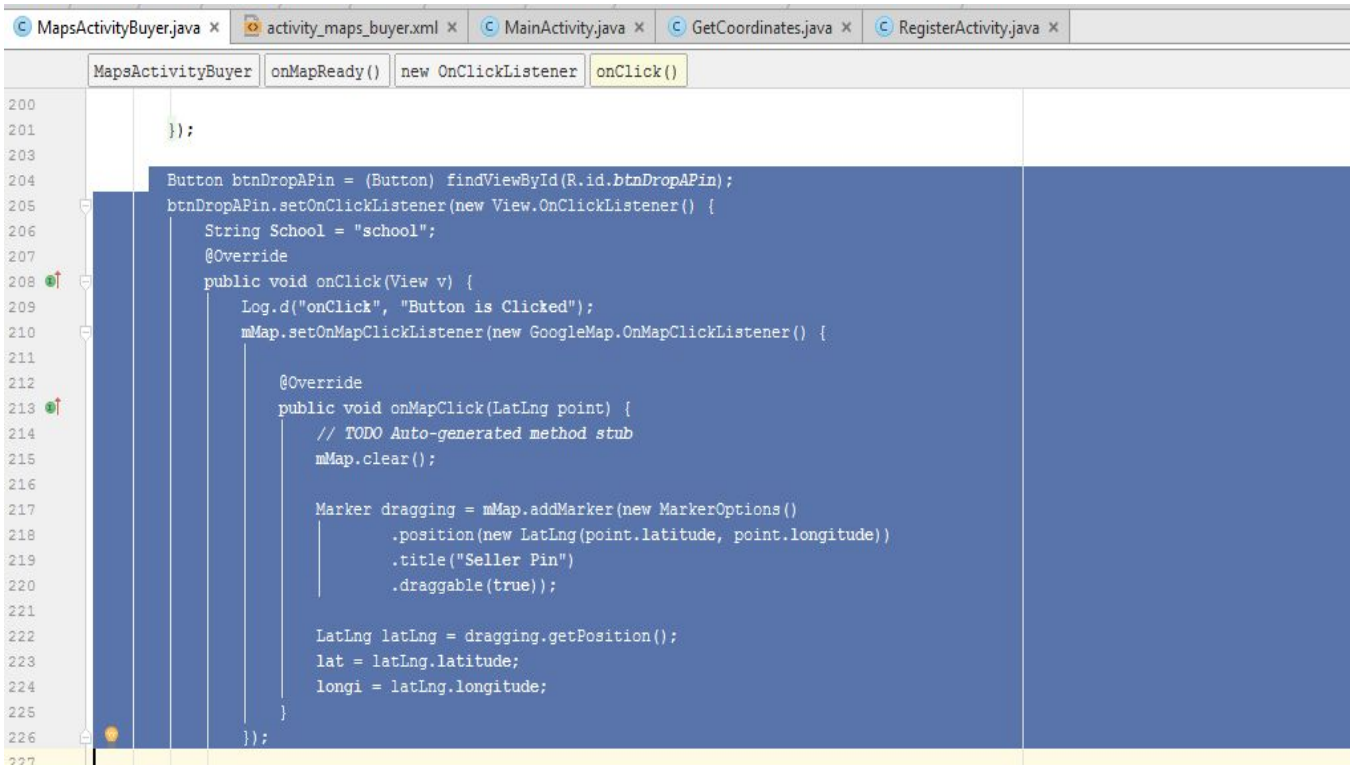


Figure 5: The user defines where the pin is dropped. This effects where pin is located on the buyer's map.

4.Template method Pattern:

- Introduction to Template method pattern:

Template method design pattern defines the program skeleton of an algorithm. One or more steps can be overridden by subclasses to allow differing behaviors.

- Reason for Introducing Template method pattern:

The Template method pattern is used to allow subclasses implement varying behavior (through method overriding)

- Template method Pattern in ParkMe Application:

The base class "AppCompatActivity" defines the method which had to be overridden. For example: the method Oncreate() needs to be overridden by the classes which extent the base class, like RegisterActivity, MainActivity, UserAreaActivity etc. Figure 6 shows Mainactivity extending from AppCompatActivity and implementing the Oncreate method which is overridden.

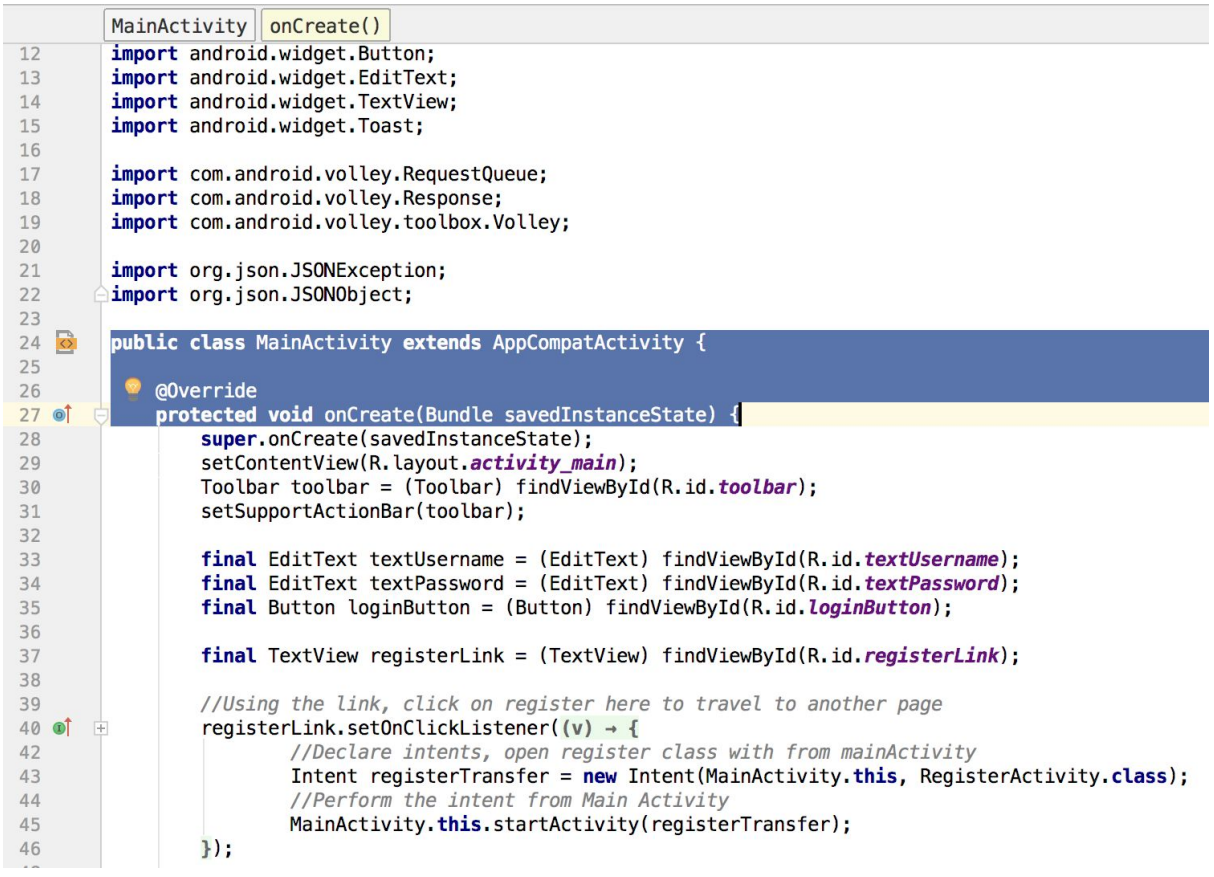


Figure 6: Mainactivity extending AppCompactActivity and Overriding OnCreate()

Description of Threads:

The FindingCoordinates Thread is Separated from the UI Thread

In MapsActivity, two threads are used. In Android applications, threads must be used to separate tasks that are CPU-intensive. This is because the main thread, the UI thread, is primarily used for updating the UI. If the UI thread is busy handling other tasks for more than five seconds, an Application Not Responding (ANR) dialog will pop up.

In MapsActivity, once the seller has sold their spot, the application will constantly communicate with a remote database to grab the coordinates of the buyer. Once the buyer's coordinates are close to the seller's coordinates, the buyer's location will be displayed on the map. Since the application is constantly connecting with the database over a network connection, the application cannot wait for the network response before displaying the buyer's coordinates to the screen. This will make the UI updates slow and it can also cause an ANR dialog if the network connection is slow.

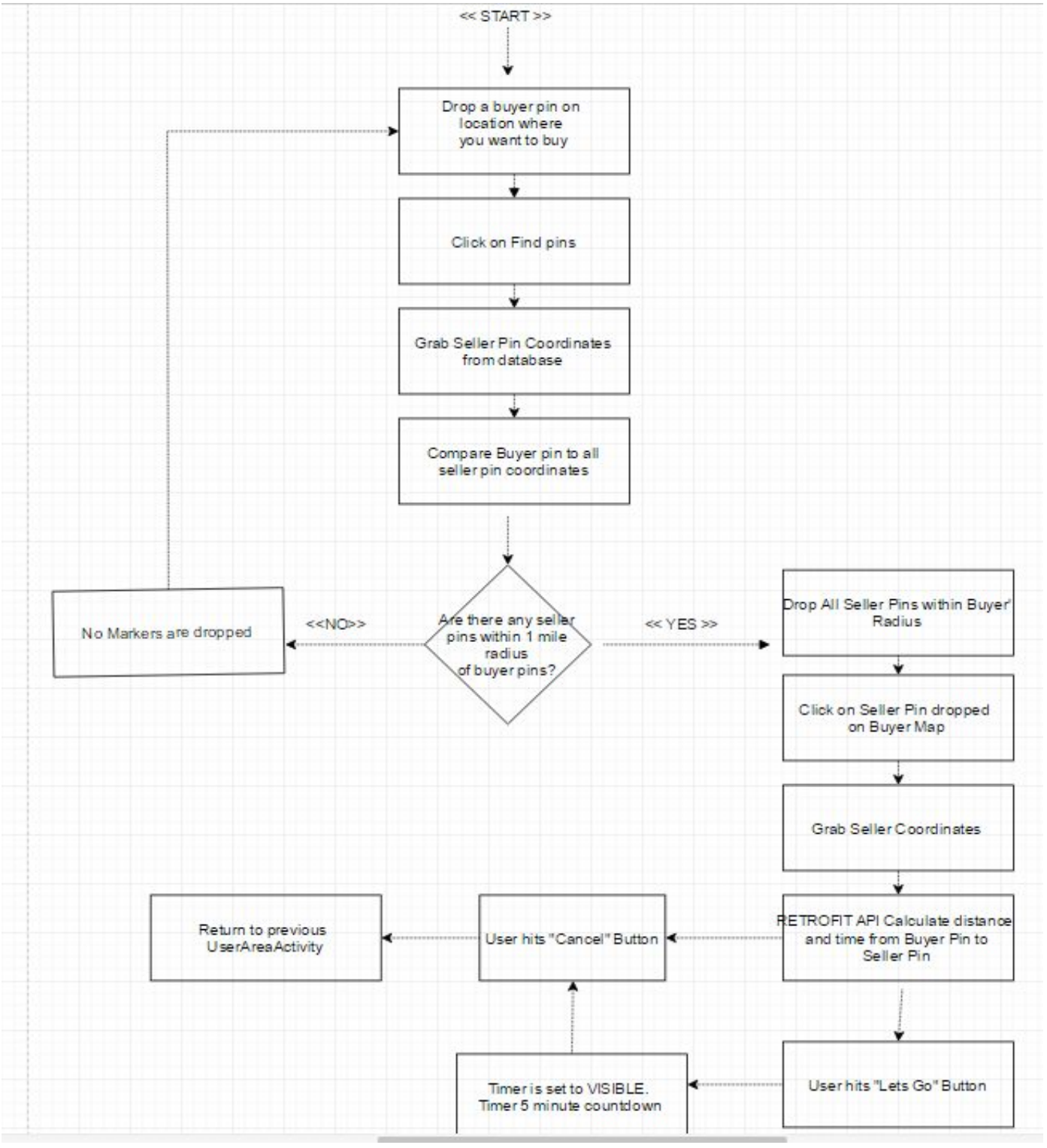
To have the application communicate with the database and update the UI to the user in parallel, a worker thread is created. The thread will constantly connect with the database to grab the buyer's coordinates. Once the thread gets a response with the buyer's coordinates, this is sent to UI thread. If the coordinates are close to the seller's coordinates, the buyer's location will be displayed on the map.

The TimerTask Thread

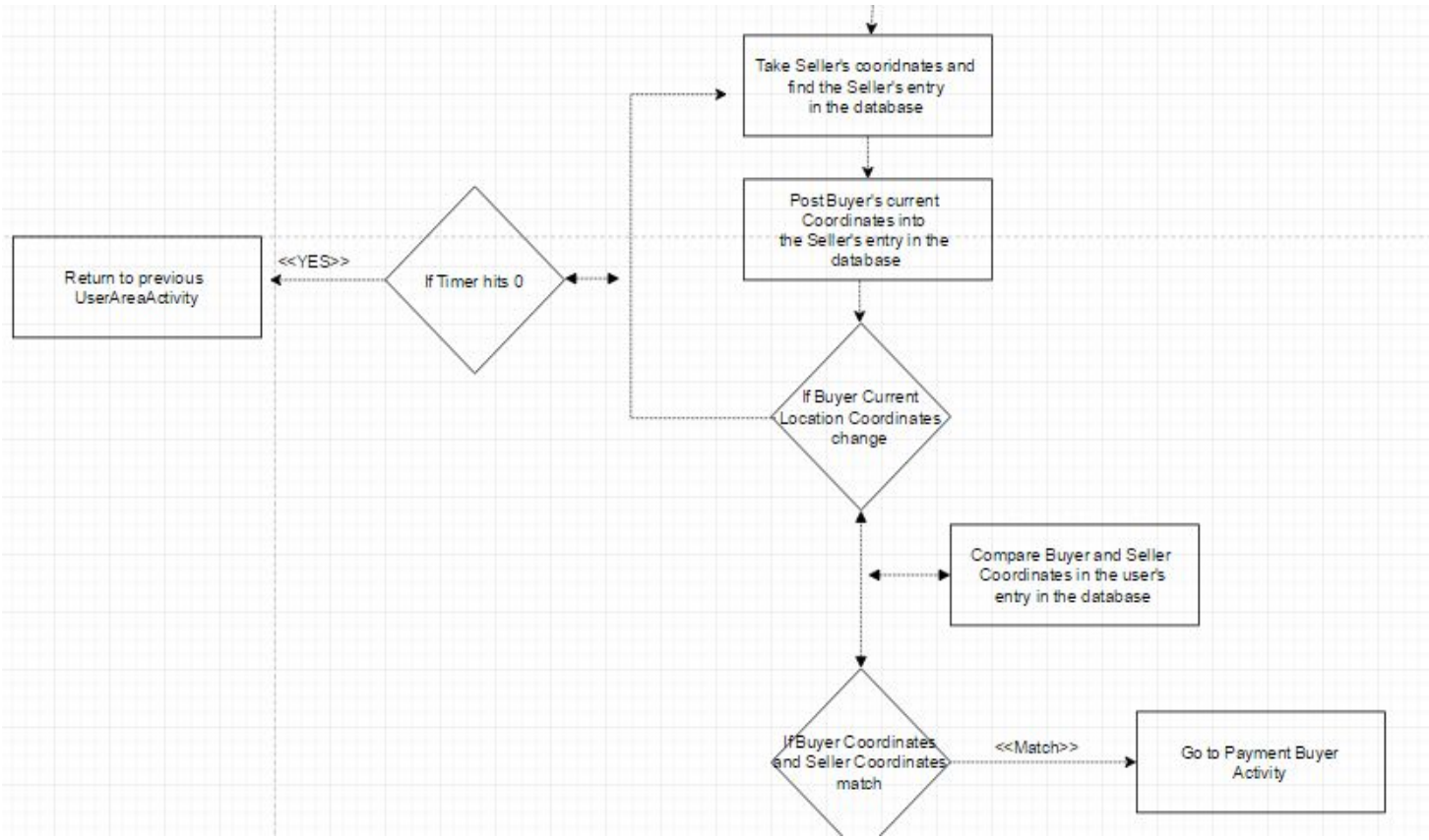
In MapsActivity, once the seller has put their spot up for sale, a five minute countdown timer starts to indicate the amount of time the seller has to sell their spot. The timer is run on a separate thread since it is constantly running and being updated. The status of the timer is sent back to the UI thread to be displayed on the activity.

External Documentation:

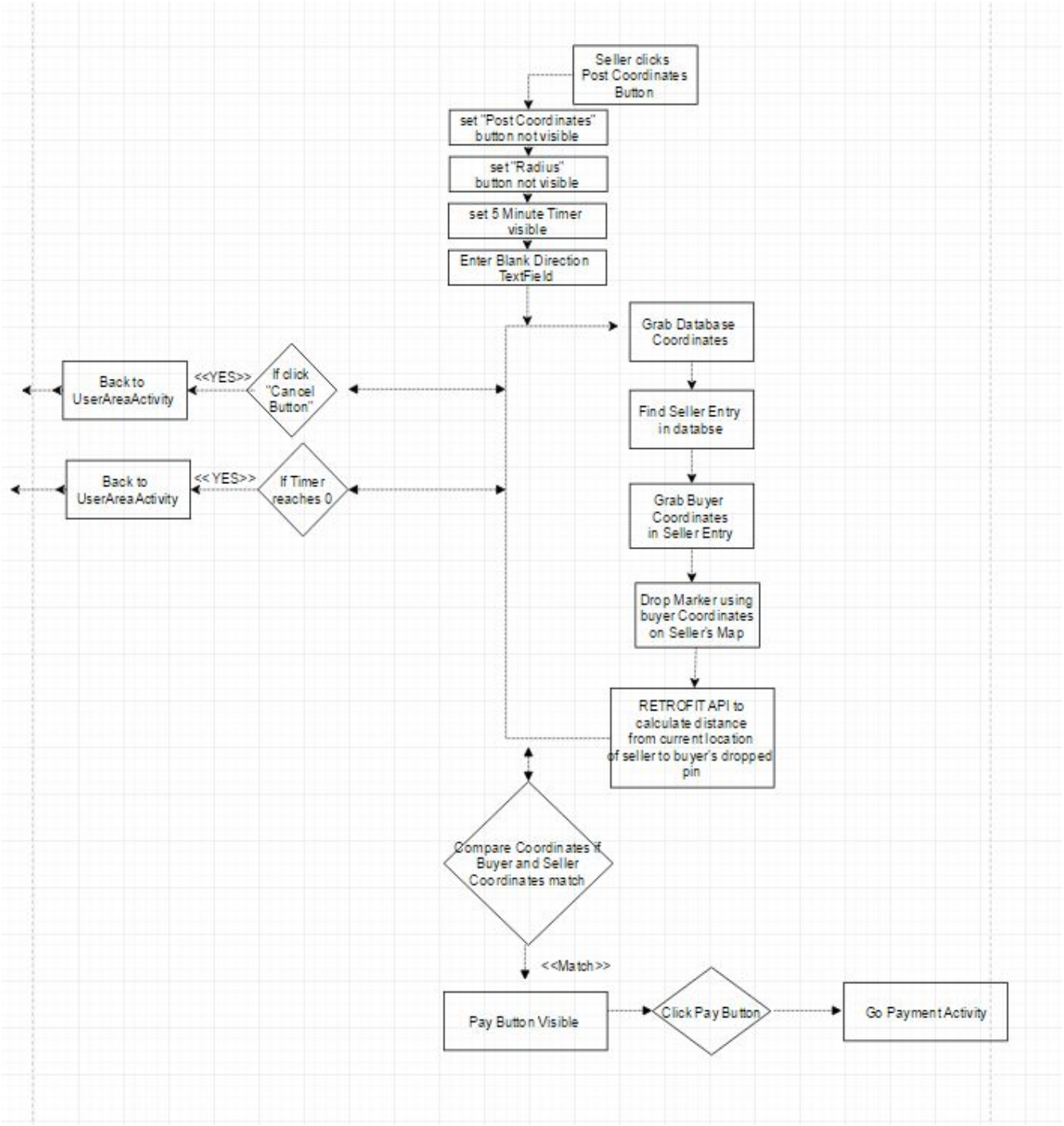
1.Flow Diagrams



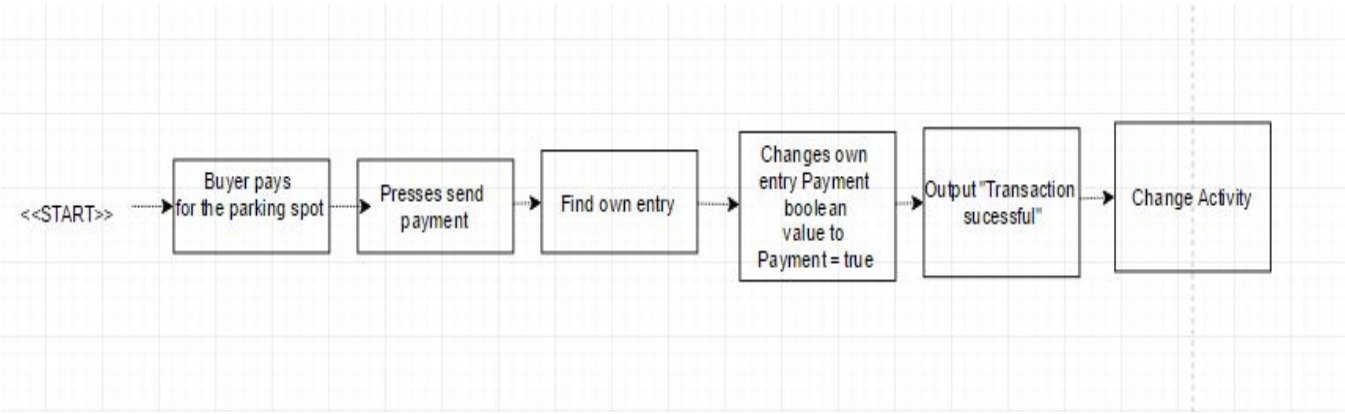
<<"Timer Set to visible" connects/points to "Take Seller's Coordinates">>



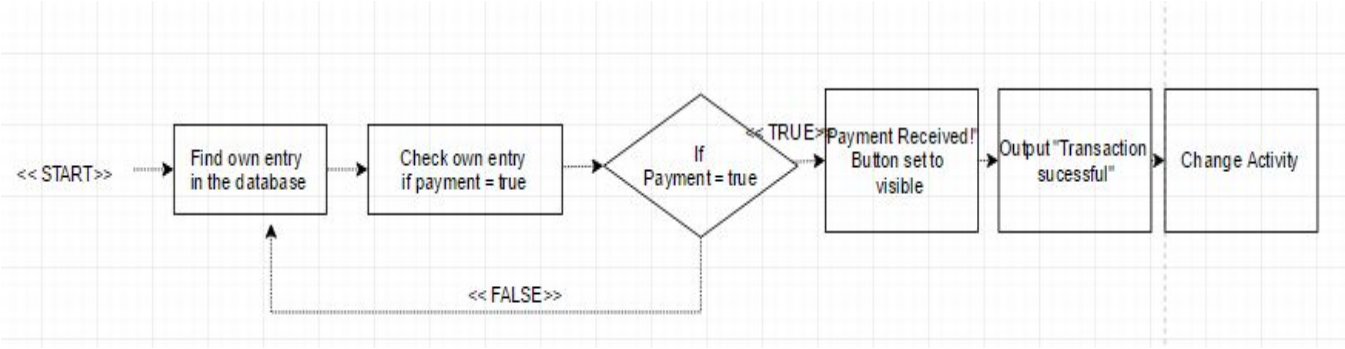
Flow Diagram 1 : *Map Buyers Algorithm*



Flow Diagram 2: *Map Seller Algorithm*



Flow Diagram 3 : *Payment Buyer Algorithm*



Flow Diagram 4 : *Payment Seller Algorithm*

2. Package Diagrams and Descriptions:

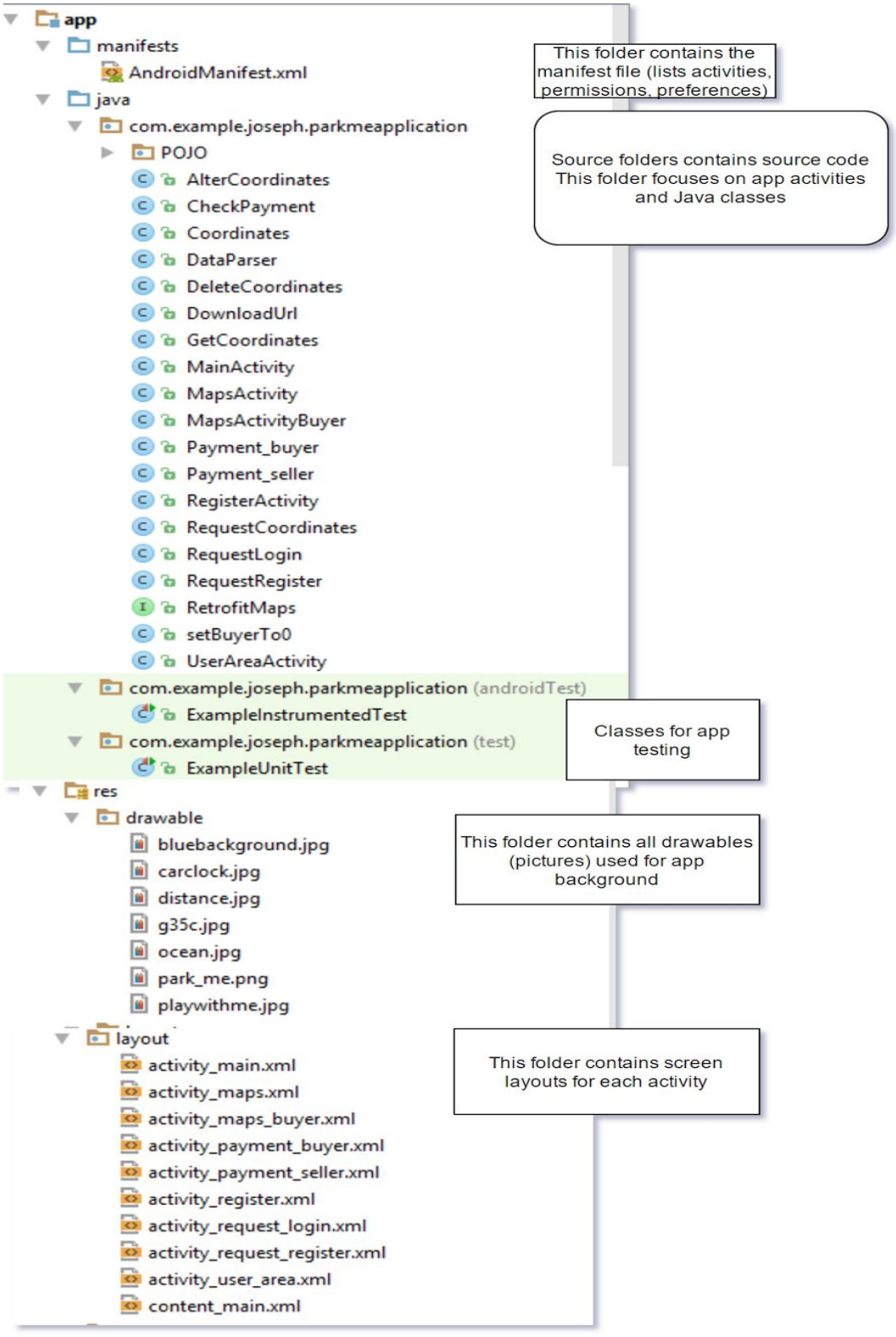


Fig: Package Diagram

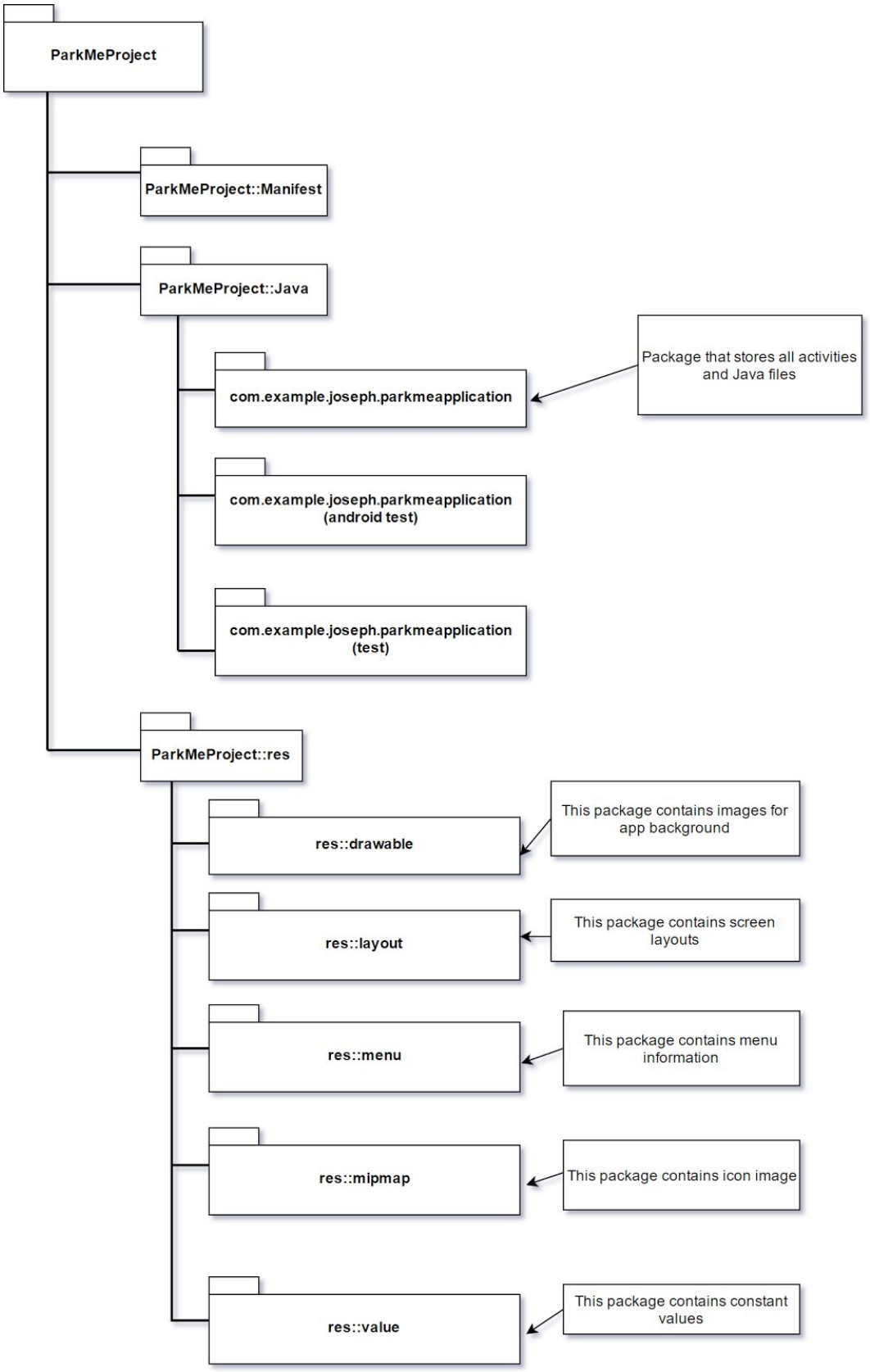


Fig: Package Structure

3.Class Diagrams and Descriptions:

Overall Class Diagram:

If Overall diagram below is too small: this link expands the overall class diagram

V

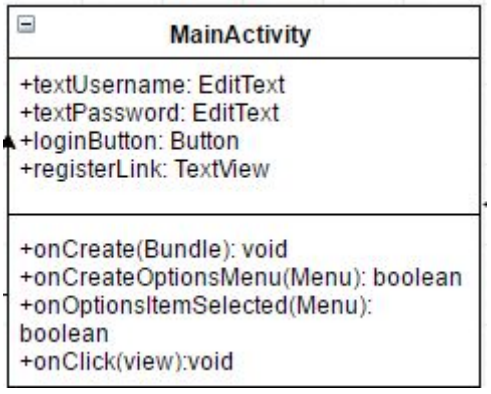
V

https://www.draw.io/?state=%7B%22ids%22:%5B%220B9ns_s1H_tNeWUdJVjZkZlZQZE0%22%5D,%22action%22:%22open%22,%22userId%22:%221130368878129721139 22%22%7D#G0B9ns_s1H_tNeWUdJVjZkZlZQZE0


<< START >>
MainActivity
+textUsername: EditText
+textPassword: EditText
+loginButton: Button
+registerLink: TextView
+onCreate(Bundle): void
+onCreateOptionsMenu(Menu): boolean
+onOptionsItemSelected(Menu): boolean
UserAreaActivity
+welcomeMessage: TextView
+buttonbuyer:button
+buttonSeller: button
+ onCreate(Bundle):
RequestLogin
+ LOGIN_REQUEST_URL: String
+ parameters: Map<String, String>
+ RequestLogin(String, String, Response.Listener<String>)
+ getParams(): Map<String, String>
RegisterActivity
+textName: EditText
+textUsername: EditText
+textPassword: EditText
+textAge: EditText
+onCreate(Bundle
RequestRegister
+textName: EditText
+textUsername: EditText
+ RequestRegister(String, String, int, String, Response.Listener<String>)
+ getParams(): Map<String,String>
DeleteCoordinates
+ DeleteCoordinates(String, String, double, Response.Listener<String>)
+ getParams(): Map<String, String>
RequestCoordinates
+ REGISTER_REQUEST_URL: String
+ parameters: Map<String, String>
+ RequestCoordinates(String, String, double, double, Response.Listener<String>
+ getParams(): Map<String, String>
AlterCoordinates
+ REGISTER_REQUEST_URL: String
+ AlterCoordinates(String, String, double, double, double, double, String, Response.Listener<String>)
+getParams(): Map<String, String>
DeleteBuyerCoordinates
+ RequestLogin(String, String, Response.Listener<String>)
+ getParams(): Map<String, String>
GetCoordinates
+ArrayAdapter<String>
+address: String
+listView: listView
+line: String
+result: String
+data: String[]
+is: InputStream
+coordinates: Coordinates[]
+ getData(): void
+ returnCoordinates(): Coordinates[]
Coordinates
+ userId: int
+ latitude: double
+ longitude: double
+ name: String
+ licenses: String
+ selected: int
+ BuyerLatitude: double
+ BuyerLongitude: double
+ BuyerName: String
+ getUserId(): int
+ setUserId(): void
+ getLatitude(): double
+ setLatitude(): void
+ getLongitude(): double
+ setLongitude(): void
+ getName(): String
+ setName(): void
+ getLicense(): String
+ setLicense(): void
+ getBuyerLatitude(): void
+ setBuyerLatitude(): double
+ getBuyerLongitude(): void
+ setBuyerLongitude(): double
+ getBuyerName():String
+ setBuyerName():void
+ getSelected(): int
+ setSelected(int): void
+ getPayment(int): void
+setPayment(int): void
MapsActivityBuyer
+ mMap: GoogleMap
+ CountdownTimer: countDownTimer
+latitude: double
+longitude: double
+lat: double
+longi: double
+name: String
+username: String
+license: int
+PROXIMITY_RADIUS: int
+mGoogleApiClient: GoogleApiClient
+mLastLocation: Location
+mCurrLocation: Marker
+mLocationRequest: LocationRequest
+cancel, btnCircle, btnRestaurant, payButton: LocationRequest
+timer: TextView
+myTimer: Timer
+myTimerTask: MyTimerSeller
+origin, dest: LatLng
+line: PolyLine
+calculateDistance: TextView
+counter: int
+ onCreate(Bundle): void
+ CheckGooglePlayServices(): boolean
+ onMapReady(GoogleMap): void
+ deleteEntry():void
+ popButton():void
+ deselectButton():void
+MapDrop(double, double, String):void
+ build_retrofit_and_get_response(string):void
+ decodePoly(String): List
+ buildGoogleApiClient(): void
+ onConnected(Bundle): void
+ onConnectionSuspend(int): void
+ onLocationChanged(Location): void
+ onConnectionFailed(ConnectionResult): void
+ checkLocationPermission(): boolean
+onKeyDown(int, KeyEvent): boolean
+ onRequestPermissionsResult(int, String, int[]): void
+ compareCoordinates(): void
MapsActivitySeller
+ mMap: GoogleMap
+ CountdownTimer: countDownTimer
+latitude: double
+longitude: double
+lat: double
+longi: double
+name: String
+username: String
+license: int
+PROXIMITY_RADIUS: int
+mGoogleApiClient: GoogleApiClient
+mLastLocation: Location
+mCurrLocation: Marker
+mLocationRequest: LocationRequest
+cancel, btnCircle, btnRestaurant, payButton: LocationRequest
+timer: TextView
+myTimer: Timer
+myTimerTask: MyTimerSeller
+origin, dest: LatLng
+line: PolyLine
+calculateDistance: TextView
+counter: int
+ onCreate(Bundle): void
+ CheckGooglePlayServices(): boolean
+ onMapReady(GoogleMap): void
+ deleteEntry():void
+ popButton():void
+ deselectButton():void
+ MapDrop(double, double, String):void
+ build_retrofit_and_get_response(string):void
+ decodePoly(String): List
+ buildGoogleApiClient(): void
+ onConnected(Bundle): void
+ onConnectionSuspend(int): void
+ onLocationChanged(Location): void
+ onConnectionFailed(ConnectionResult): void
+ checkLocationPermission(): boolean
+onKeyDown(int, KeyEvent): boolean
+ onRequestPermissionsResult(int, String, int[]): void
+ compareCoordinates(): void
Payment_buyer
+ name, username, license, name2, username2: String
+ alterLatitude, alterLongitude, BuyerLatitude, BuyerLongitude: double
+ license2: int
+ BuyerName: String
onCreate(Bundle): void
Payment_seller
+ myTimer: Timer
+ paymentWait: PaymentWait
+ name, age, username, license: String
+ latitude, longitude: double
+ payButton, cancelButton: Button
+ onCreate(Bundle): void
+ setButton(): void
+ deleteEntry(): void
SetPaymentTrue
+ REGISTER_REQUEST_URL: String
+ CheckPayment(String, String, double, double, double, double, String, Response.Listener<String>)
+ getParams(): Map<String, String>

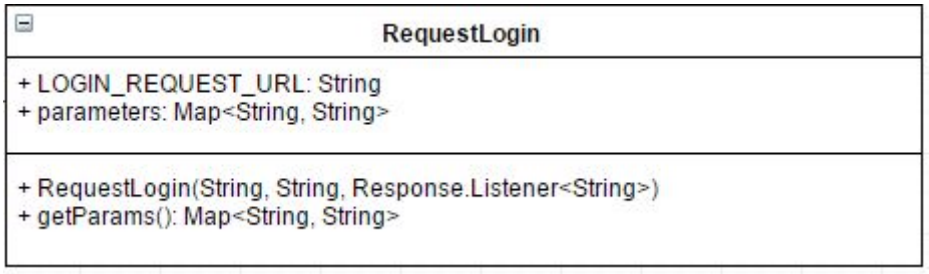
Individual Class Diagrams and Descriptions:

MainActivity/LoginActivity



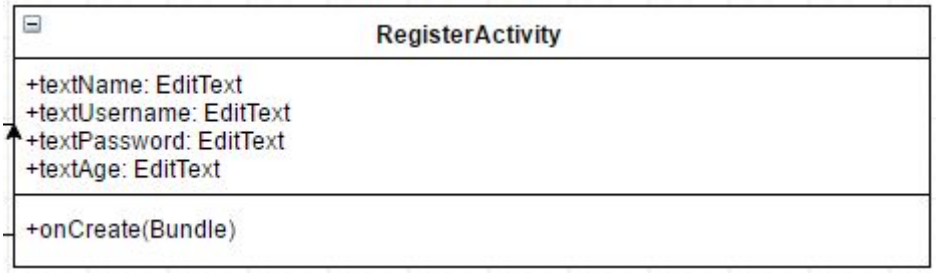
MAINACTIVITY CLASS: This is the first activity that starts when the user starts the application. A login prompt is issued with username and password field as well as login button. There is a link on the bottom that allows you to register. When the login button is pressed the username and password are submitted to the SQL database with the help of the requestLogin class. If the login is successful, the user is redirected to UserAreaActivity.

RequestLogin:



REQUESTLOGIN CLASS: This class is used by the main/login activity to communicate with the login php script on the server using JSONObject requests. Map<String string> parameters is created to store username and password strings.Parameters is then passed to the server where it is used to request login.

RegisterActivity:

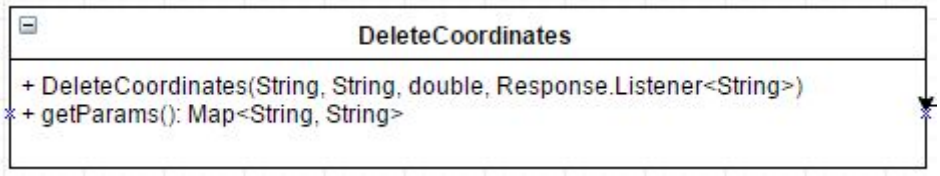


REGISTERACTIVITY CLASS: This class is responsible for registering a user. Username, password, name, and age strings are prompted of the user and stored into the database after successful user registration. When the register button is pressed the user data is stored in the database. If there were no errors in the user data then the user will be redirected to the MainActivity for login using the login information entered.

RequestRegister:

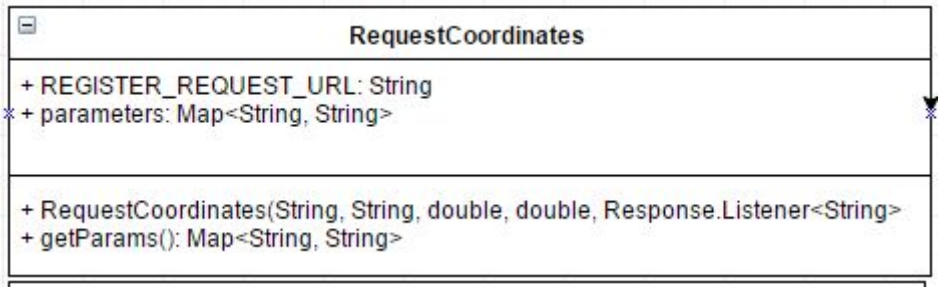
REQUESTREGISTER CLASS: This class is used by RegisterActivity to communicate with the registration PHP file from the server. Map<String string> parameters is created to store user data, ie, username, password, name, and age. Parameters are then passed to the server where it is used to register.

DeleteCoordinates:



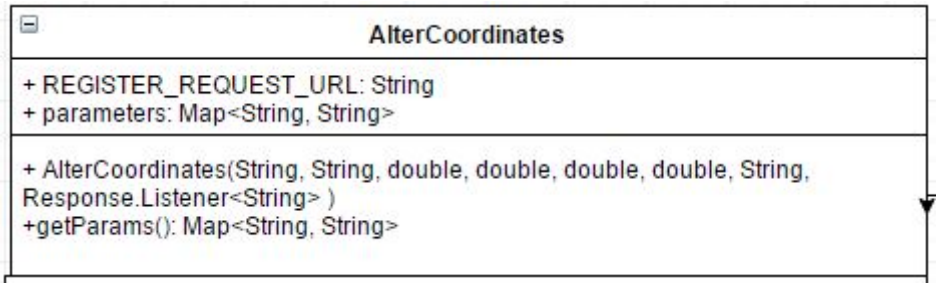
DELETECOORDINATES CLASS: This class communicates with a PHP file to allow the user to delete coordinates from the database. It takes the coordinates name and the coordinate location, find it in the database, and delete those fields.

RequestCoordinates:



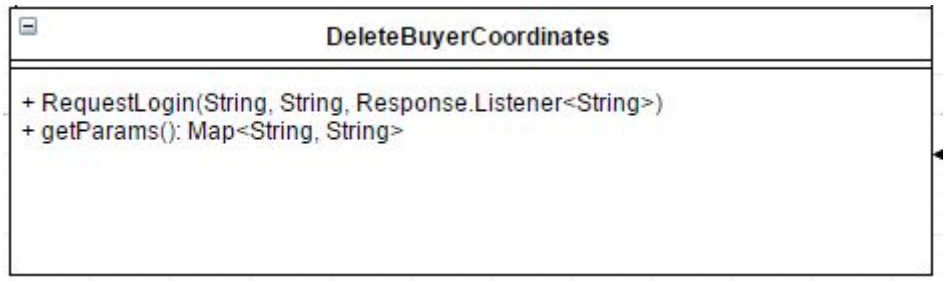
REQUESTCOORDINATES CLASS: This class is responsible for communicating with the PHP file on the server using JsonObject requests. The server is contacted to access coordinates. Map<String string> parameters is created to store strings of data including name, license, latitude, longitude, once stored parameters is returned.

AlterCoordinates:



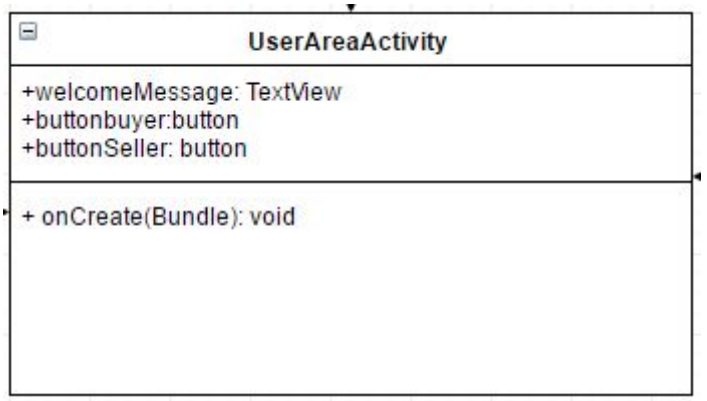
ALTERCOORDINATES CLASS: This class communicates with the PHP file on the server that allows the user to alter the coordinates on the database. This is used when the buyer is moving and is constantly updating the database with his current location. This is needed to change the coordinates in our server.

DeleteBuyerCoordinates:



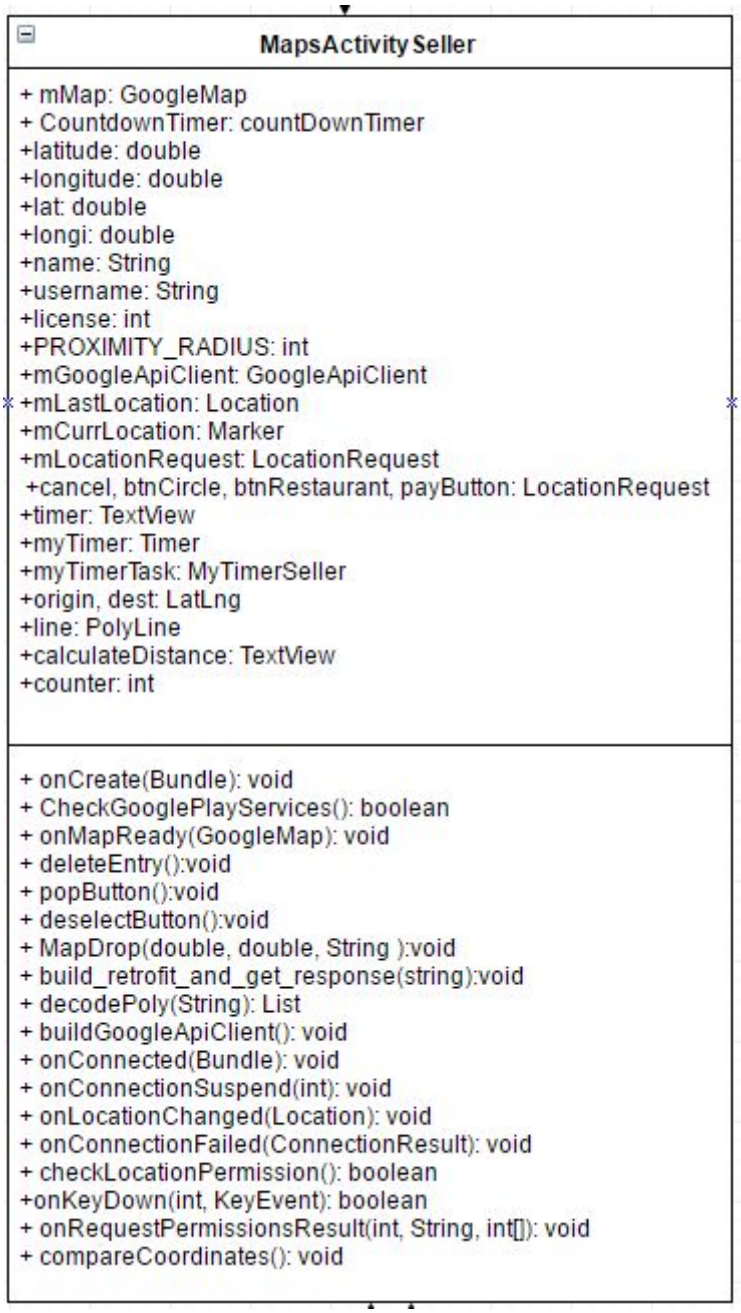
DELETEBUYERCOORDINATES CLASS: This class is to talk with the PHP file in the server. Whenever the buyer cancels his transaction, it deletes the his own buyer coordinates from the database in the server.

UserAreaActivity:



USERAREAACTIVITY CLASS: On UserActivity page if the user chooses to be a seller he will be redirected to the MapsSeller activity. If the user chooses to be a buyer, he will be redirected to the MapsBuyer activity.

MapsActivitySeller:

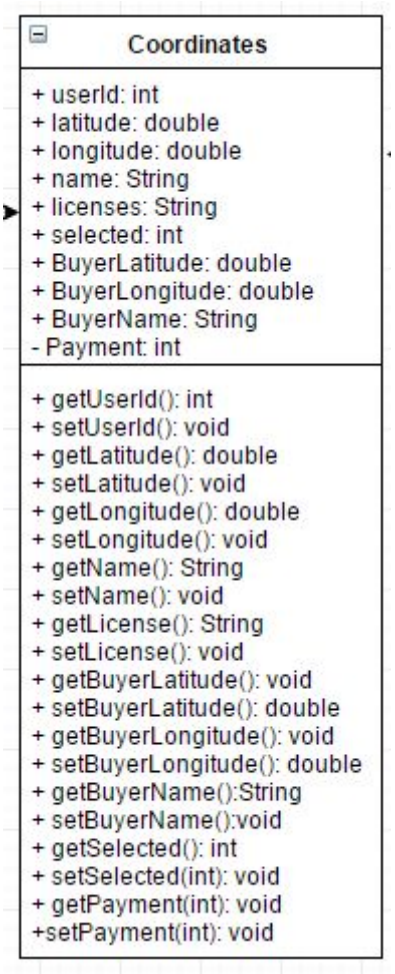


MAPSACTIVITYSELLER CLASS: MapsActivity Seller plays the role of the seller. The maps activity seller uses google maps API. This activity is used to post coordinates of current location on database, to compare coordinates with buyers, and play the ultimate functionality of the seller role in the application.

MapsActivityBuyer:

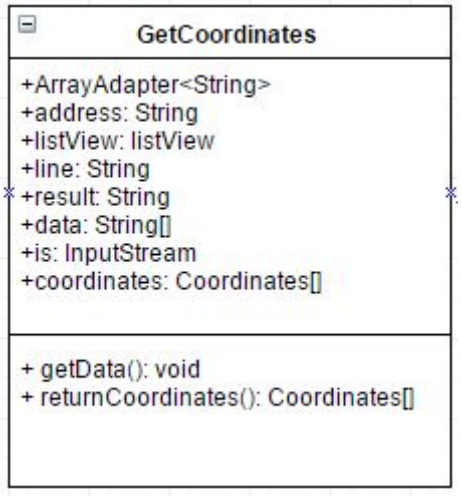
MapsActivityBuyer: If the user has selected a role to be a buyer he will be taken to MapsActivityBuyer class. This class uses Google Maps API to display the map. We then use the map to drop and retrieve pins and compare coordinates with Seller.

Coordinates:



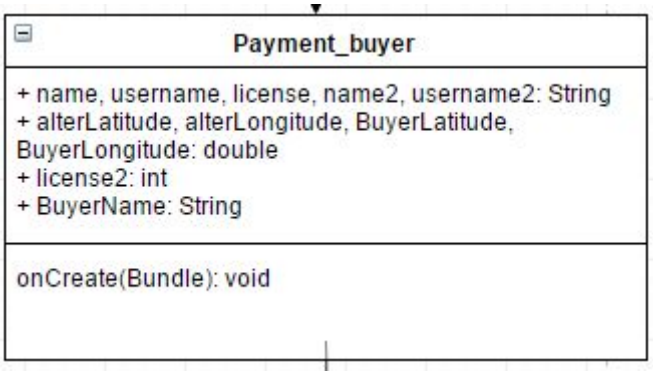
COORDINATES CLASS: The coordinates class holds the values of all the information from the SQL database after fetching.

GetCoordinates:



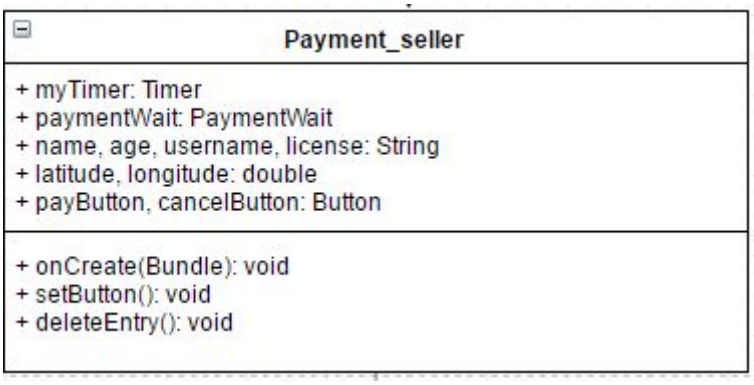
GETCOORDINATES CLASS: This class uses the Coordinates class to store and retrieve locations of markers dropped. The Get coordinates class is the basic connection from and to the SQL database.

Payment_buyer:



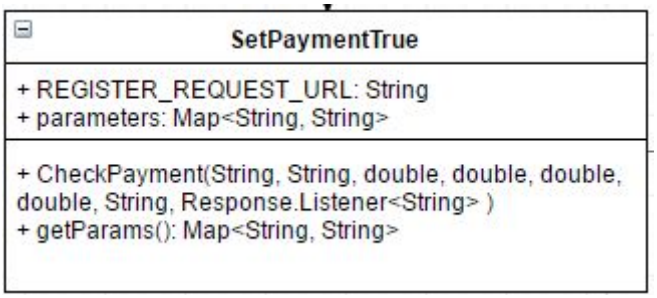
PAYMENTACTIVITYBUYER CLASS: The PaymentActivityBuyer class is responsible for facilitating transaction information from the buyer to the seller. The class is used to send a payment that notifies the database that transactions has been paid.

Payment_seller Activity:



PAYMENTACTIVITYSELLER CLASS: This class is responsible for notifying the seller that a buyer has paid for the spot. Once the Seller has received payment, a button is prompted and the seller can confirm that he has received payment.

SetPaymentTrue:



SETPAYMENTTRUE CLASS: This class is involved with checking and validating payment and making sure that the buyer and seller are in close proximity to each other so that they can initiate **payment**.

Class Hierarchy

The class hierarchy shows the structure of our project. We override methods that are included in both the Parent and Child classes.

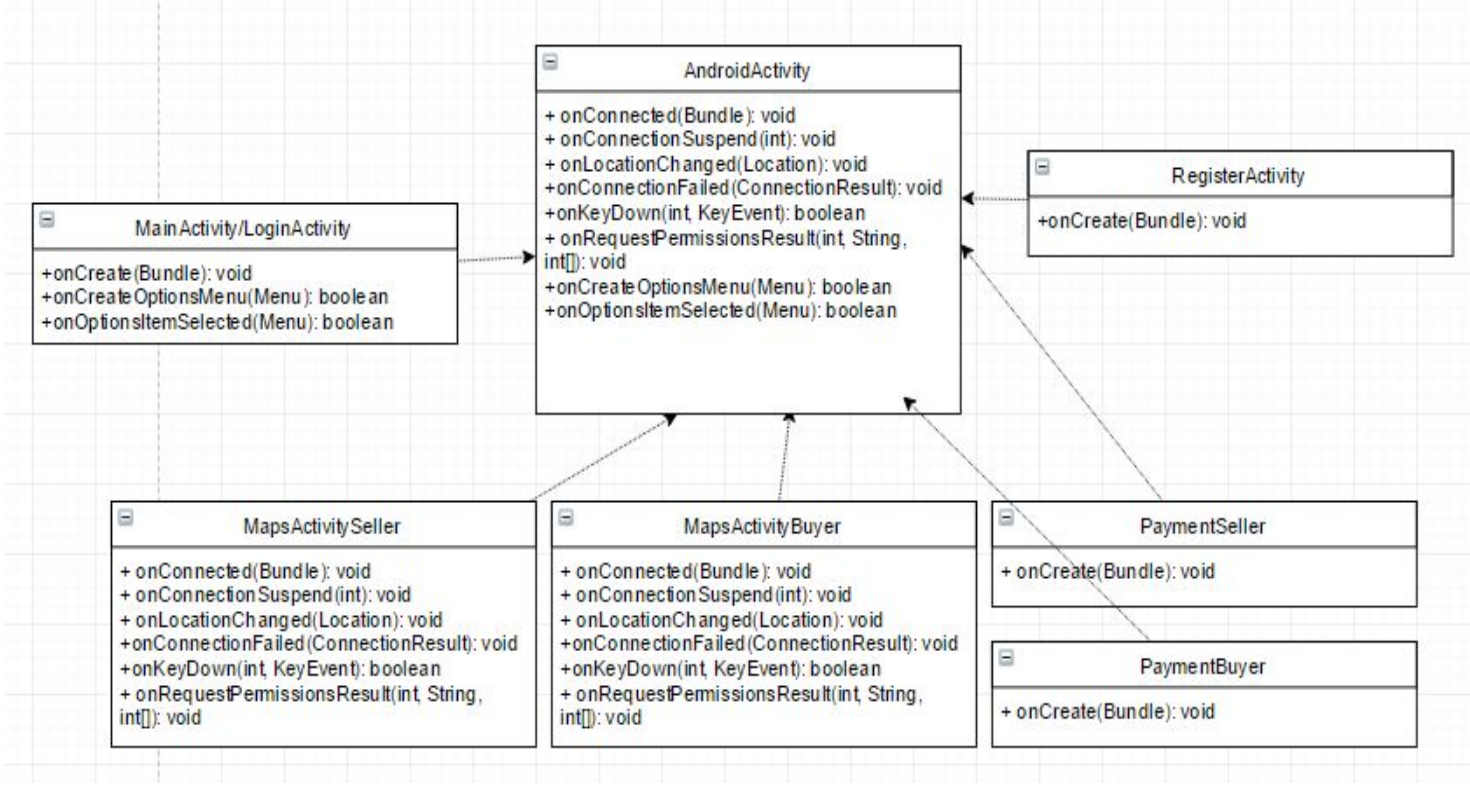


Fig: *Parent and Child of activities*

In our MapsActivityBuyer and MapsActivity Seller, the majority of the methods extend from Android Activity and override methods that have to deal with the location of the phone and networks due to the activities having to deal with the location of the phones.

Activities that are shown on the application each have their own onCreate method that overrides the Android Activity.

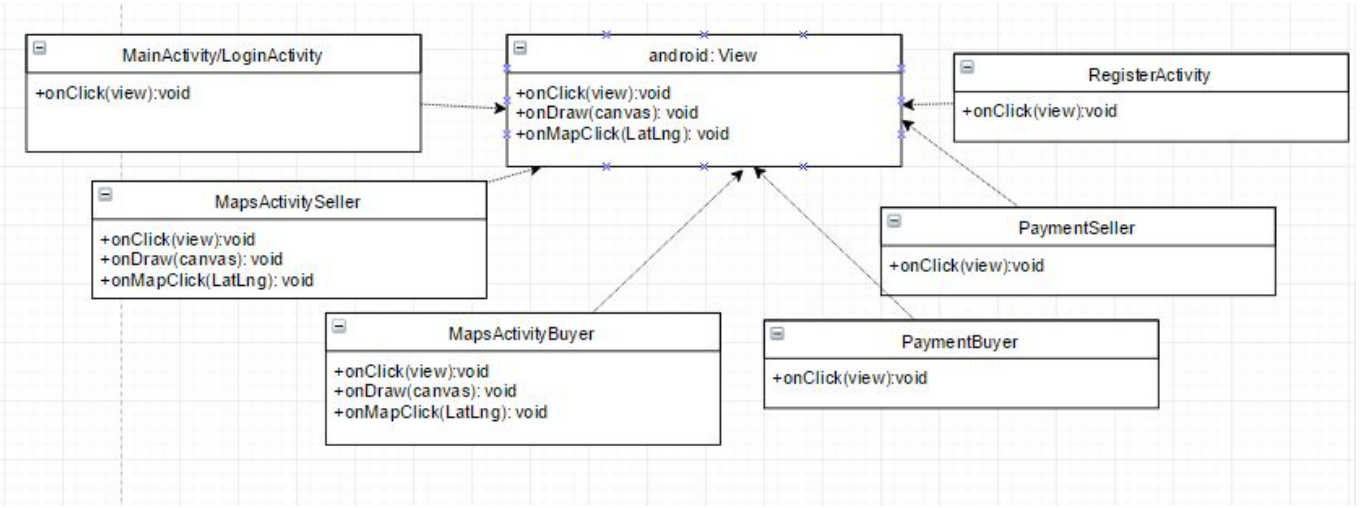


Fig: *Parent and Child of Views*

In MapsBuyer and MapsSeller, drawing a radius on the canvas overrides canvas view.

All activities that have to deal with button clicks automatically override the functionality of clicking on android views.

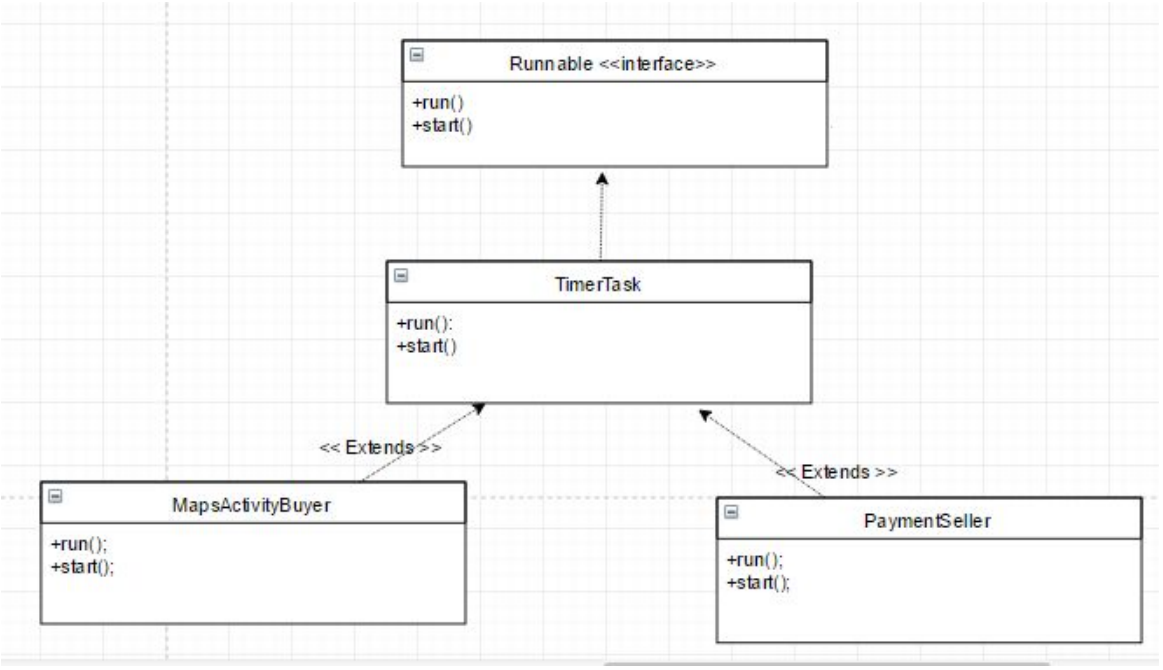
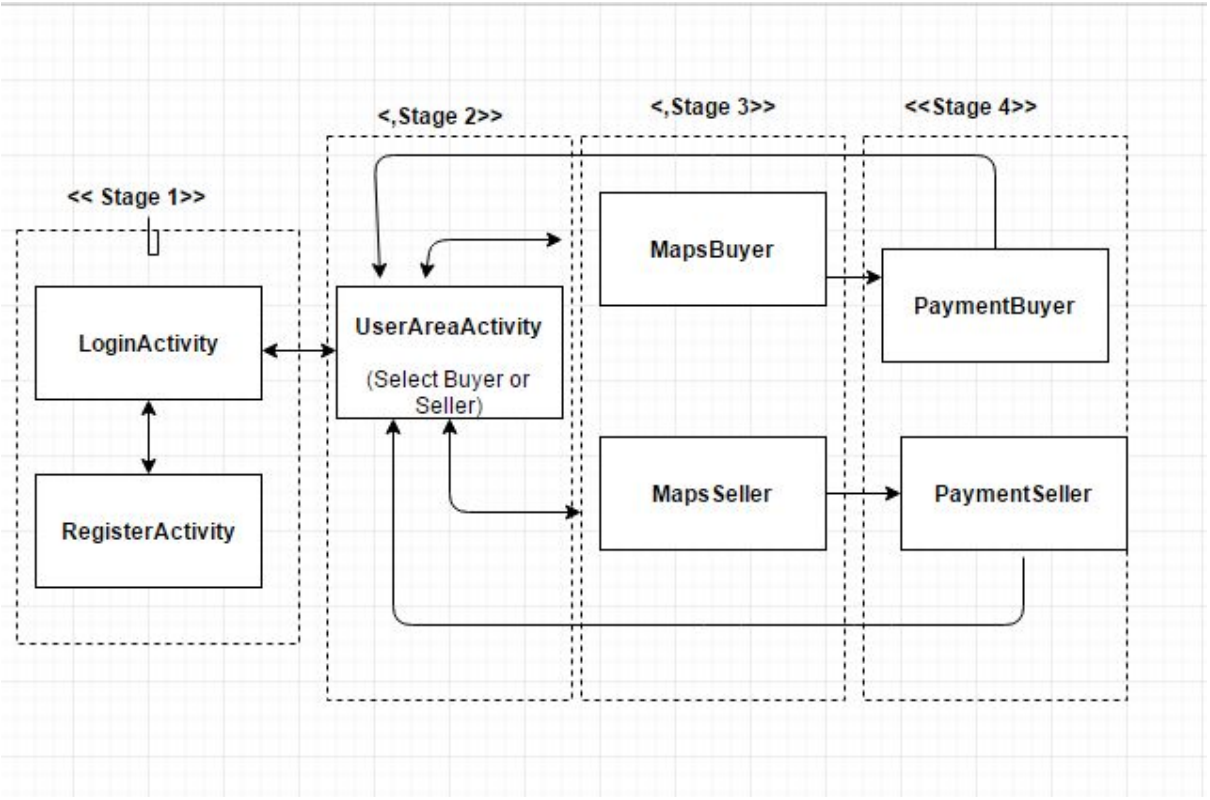


Fig: *TimerTask Activity*

MapsActivityBuyer and PaymentSeller both extend from TimerTask class that overrides runnable.

Interesting In-Depth Algorithm for Successful Transaction



Four stages for successful transaction

Broad Organization of Project (Specifications for each stage later in this section):

Stage 1:

The user will be prompted with the Login Activity in which the user can enter his credentials. If the user has not signed up with the application yet, he will click the "Register Me" link to go into RegisterActivity to register his credentials into the database.

Stage 2:

After putting his credentials into the Login, the user will be prompted to the UserAreaActivity where the user can act as a role of a Buyer or a Seller.

Stage 3:

If the user selects to become a buyer, the user is prompted to the MapsBuyer. If the user selects to be a seller, the user is prompted to MapsSeller. Once the Buyer locates and meets up with the Seller (more information about how

this works later on in the external documentation), depending on Buyer or Seller they will go into their own payment activities.

Stage 4:

If the user is using the MapsBuyerActivity the user will go into the PaymentBuyerActivity. If the user is using the MapsSellerActivity the user will go into the PaymentSellerActivity. Once the payment transaction has been successful, both buyer and seller will be taken into the UserAreaActivity in which they can choose to become a buyer or seller again.

Descriptions and algorithms of classes

Stage 1:

Login/Registration Activity:

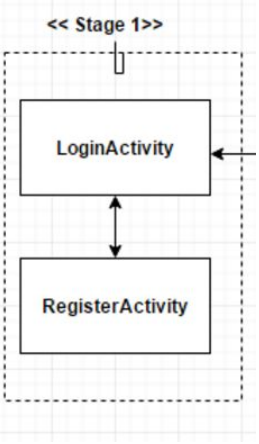


***Figure A**: First stage of the application*

This is the first stage in which the user is prompted with when logging into the application. The user is prompted to enter his credentials in order to log into the application. Below is the picture that shows the user logging into his credentials in the login screen.

***Figure B**: The user enters the username and his password*

When the login button is pressed, it then checks the PHP file that checks the database. We take the username and password input, and we create a new volley request to launch the information to our Login.php script in the server.

```
public class RequestLogin extends StringRequest {
    private static final String LOGIN_REQUEST_URL = "http://sfsuse.com/~kkwok/Files/Login.php";
    private Map<String, String> parameters;

    //Constructor to ask for name, username, age, password, listener
    public RequestLogin(String username, String password, Response.Listener<String> listener)
    {
        //Pass through into volley
        super(Method.POST, LOGIN_REQUEST_URL, listener, null);
        parameters = new HashMap<>();
        parameters.put("username", username);
        parameters.put("password", password);
    }

    public Map<String,String> getParams() { return parameters; }

}
```

***Figure Snippet C**: Android class to set the parameters of credentials to pass to the Login PHP file in server*

```
RequestLogin requestLogin = new RequestLogin(username, password, responseListener);
RequestQueue rq = Volley.newRequestQueue(MainActivity.this);
rq.add(requestLogin);
```

***Figure Snippet D**: Execute the volley command to talk with PHP script*

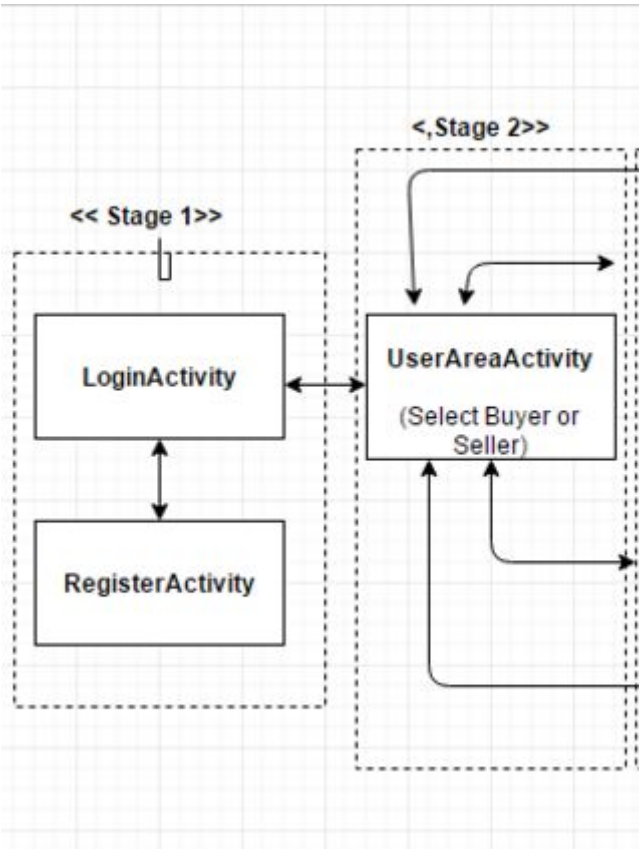
If the user has input the right credentials, then he would be logged into the application.

```
      Response.Listener<String> responseListener = new
Response.Listener<String>() {
          public void onResponse(String response) {


      if (success)
      {
         //Create a new intent
         Intent intent = new Intent(MainActivity.this, UserAreaActivity.class);
         //Pass in the variables for the intent
         String name = jsonResponse.getString("name");
         int age = jsonResponse.getInt("age");
         intent.putExtra("name", name);
         intent.putExtra("username", username);
         intent.putExtra("age", age);
         //Start the intent and pass the activities
         MainActivity.this.startActivity(intent);
      }
         }
      };
```

***Figure Snippet E:** ResponseListener listens to the VolleyRequest sent online. If Login.php script on the server outputs "success", this means that the credentials tha the user inputted are in the database. The user is then sent to the next activity.*

Stage 2: UserAreaActivity



***FigureF**: Stage 2 User Area Activity*

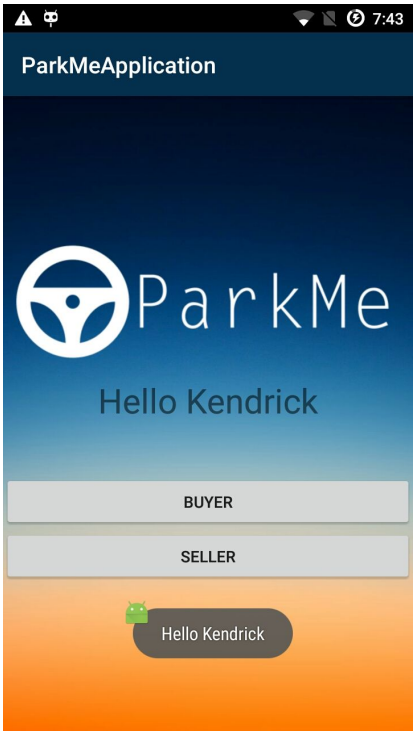


***Figure G**: User is prompted to become a Buyer or a Seller.*

The user is prompted to become a buyer or a seller. Depending on which button is clicked, the user is prompted to different activities.

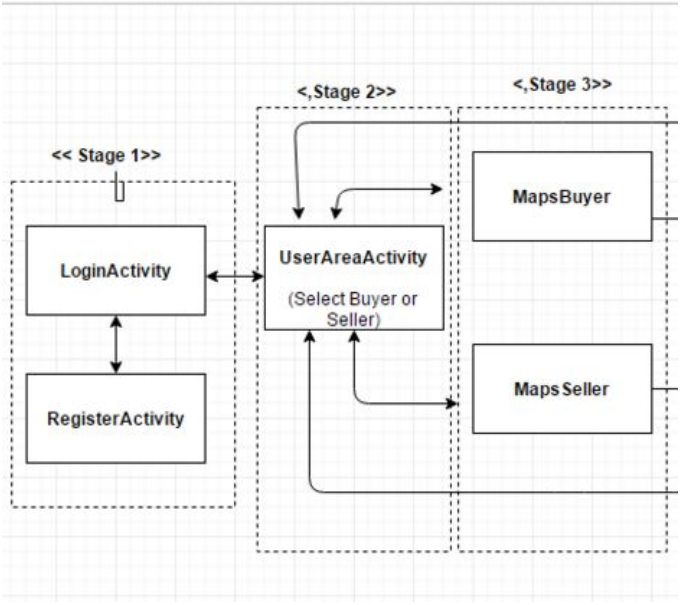
```
BuyerButton.setOnClickListener(new View.OnClickListener() {
   public void onClick(View v) {
        //Declare intents, open register class with from mainActivity
        Intent registerTransfer = new Intent(UserAreaActivity.this,
        MapsActivityBuyer.class);
        //Pass in the variables
        registerTransfer.putExtra("name", name);
        registerTransfer.putExtra("username", username);
        registerTransfer.putExtra("age", age);
        //Perform the intent from Main Activity
        UserAreaActivity.this.startActivity(registerTransfer);
        finish();
   }
});
```

***Figure Snippet H:** If buyer button is clicked, he would be launched into MapsActivityBuyer*

```
buttonSeller.setOnClickListener(new View.OnClickListener() {
   public void onClick(View v) {
        //Declare intents, open register class with from mainActivity
        Intent registerTransfer = new Intent(UserAreaActivity.this,
MapsActivitySeller.class);
        registerTransfer.putExtra("name", name);
        registerTransfer.putExtra("username", username);
        registerTransfer.putExtra("age", age);
        //Perform the intent from Main Activity
        UserAreaActivity.this.startActivity(registerTransfer);
   }
                                        });
```

***Figure Snippet I:** If Seller button is clicked, he would be launched into MapsActivitySeller.*

Stage 3: MapsSeller / MapsBuyer



***FigureJ**: This is stage 3 of the application*

MapsActivitySeller

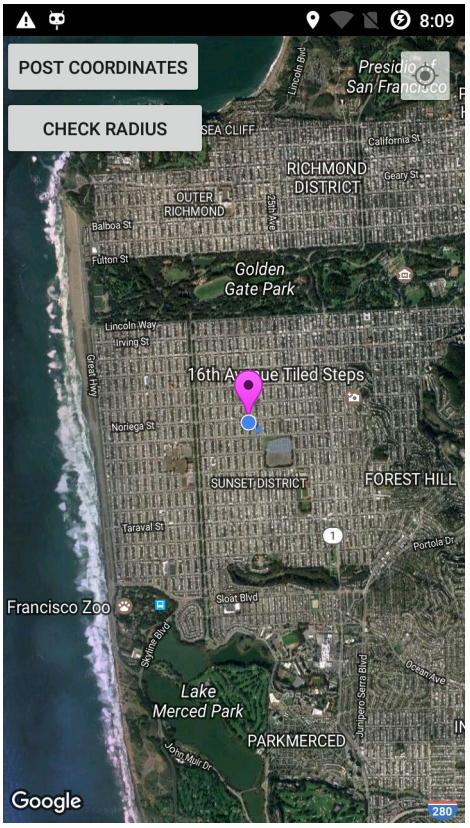


Figure K: *MapsActivity seller. The red marker shows the current location of the user.*

If the user has logged in to become a seller, the user then goes into MapsActivitySeller. In this activity, the pin grabs the current location and puts it on the map.

```
mLocationRequest.setInterval(1000);
mLocationRequest.setFastestInterval(1000);
```

Figure Snippet L*: Sets the interval every 1 second to grab current location.*

```
public void onLocationChanged(Location location) {
//Find the origin of the current location
origin = latLng;
//Configure the marker on the current location
MarkerOptions markerOptions = new MarkerOptions();
markerOptions.position(latLng);
markerOptions.title("Current Position");
//Change the color to mangenta
markerOptions.icon(BitmapDescriptorFactory.defaultMarker(BitmapDescriptorFactory
.HUE_MAGENTA));
//Drop the marker on the map
mCurrLocationMarker = mMap.addMarker(markerOptions);
}
```

Figure M: *This method is called every time the location is changed (The first call of this method is when the GoogleMaps is displayed)*

When the PostCoordinates button is pressed, the user's name, latitude, and longitude is put into volley to be inserted into the database.

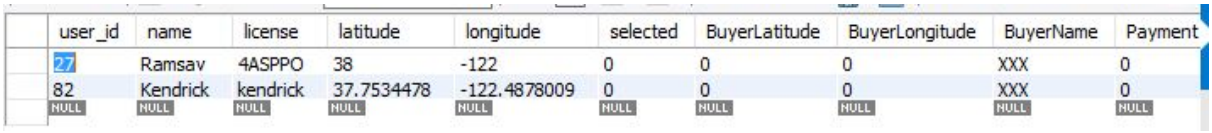
```
AlterCoordinates ac = new AlterCoordinates(alterName, username, alterLatitude,
alterLongitude,
       latitude, longitude, name1, getResponse2);
//Instantiate a request queue
RequestQueue queue2 = Volley.newRequestQueue(MapsActivityBuyer.this);
//Execute the queue
queue2.add(ac);
```

Figure N: *User's information is put into AlterCoordinates. To execute the user's information, a request queue is initiated and volley launches it to the server.*

user_id	name	license	latitude	longitude	selected	BuyerLatitude	BuyerLongitude	BuyerName	Payment
27	Ramsav	4ASPPO	38	-122	0	0	0	XXX	0
82	Kendrick	kendrick	37.7534478	-122.4878009	0	0	0	XXX	0
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

***Figure O**: SQL Databse on the server. "Kendrick" entry is put into the SQL database with the latitude and longitude of the seller's pin.*

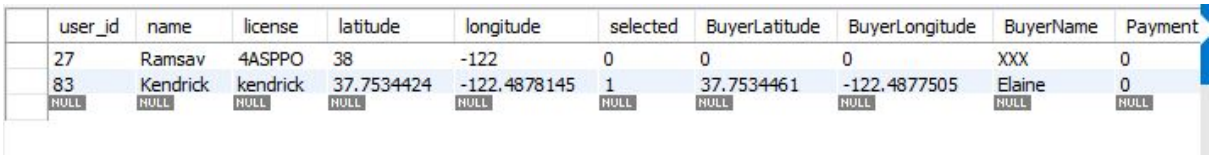
Now that we have grabbed all the data from the SQL database. We can repeatedly check on the database to see if anything has been changed. Recall that the SQL Database for the "Kendrick" entry looks like this.



user_id	name	license	latitude	longitude	selected	BuyerLatitude	BuyerLongitude	BuyerName	Payment
27	Ramsav	4ASPPO	38	-122	0	0	0	XXX	0
82	Kendrick	kendrick	37.7534478	-122.4878009	0	0	0	XXX	0
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

***Figure P**: Before the buyer is selected*

After a Buyer has selected a pin, (algorithm showed in MapsActivityBuyer in this section) the database will be changed to this.



user_id	name	license	latitude	longitude	selected	BuyerLatitude	BuyerLongitude	BuyerName	Payment
27	Ramsav	4ASPPO	38	-122	0	0	0	XXX	0
83	Kendrick	kendrick	37.7534424	-122.4878145	1	37.7534461	-122.4877505	Elaine	0
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

***Figure Q**: SQL Database shows when the seller pin has been selected. Selected, BuyerLatitude, BuyerLongitude, and BuyerName in "Kendrick" entry has been changed.*

The seller uses TimerTask thread to repeatedly check the database to see if a buyer has selected their spot. The TimerTask is called every five seconds.

```
myTimer.scheduleAtFixedRate(myTimerTask, 0, 5000); //(timertask,delay,period)
```

***Figure Snippet R**: This TimerTask is scheduled to be called every five seconds*

A coordinates class uses volley to hold all the data from the database. By calling gc.getData(), all the data from the database is held into the coordinates array.

```
//Instantiate getCoordinate class that use GSON to grab SQL data
GetCoordinates gc = new GetCoordinates();
gc.getData();
coordinates = gc.returnCoordinates();
```

***Figure S**: All the data from the database is held into the coordinates array.*

A TimerTask method is called every five seconds and is used to **compare the coordinates together**. If the longitude and latitude of the Buyer and Seller taken

from the database are within an interval of .00050, a button will pop up that will prompt the user to go into the payment activity.

```
public class MyTimerSeller extends TimerTask {
       @Override
       //Run the thread
       public void run () {
           //Grab the coordinates from the SQL datbabase
           GetCoordinates gc = new GetCoordinates();
           gc.getData();
           coordinates = gc.returnCoordinates();

           //Loop through the whole database data and find matches
           for (int i = 0; i < coordinates.length; i++) {

             //If the latitude of seller and coordinates of buyer is within
//interval of .0050
     if (coordinates[i].getLatitude() > (coordinates[i].getBuyerLatitude() -
0.00050) && coordinates[i].getLatitude() < (coordinates[i].getBuyerLatitude() +
0.00050)) {

       //If the longitude of seller and coordinates of buyer is within interval
//of .0050

       if (coordinates[i].getLongitude() > (coordinates[i].getBuyerLongitude() -
0.00050) && coordinates[i].getLongitude() < (coordinates[i].getBuyerLongitude()
+ 0.00050)) {

         //If the names match in the database
         if (coordinates[i].getName().equals(name)) {
                          //Set the payButton to be visible
                         payButton.setVisibility(View.VISIBLE);
                     }
                 }
             }
           }
```

Figure T: *Data is parsed into coordinates. Compare the coordinates of buyer and seller, and if the buyer and seller are within a .0005 radius, then a button "Let me Receive Payment" will show that prompts the user to the next activity.*

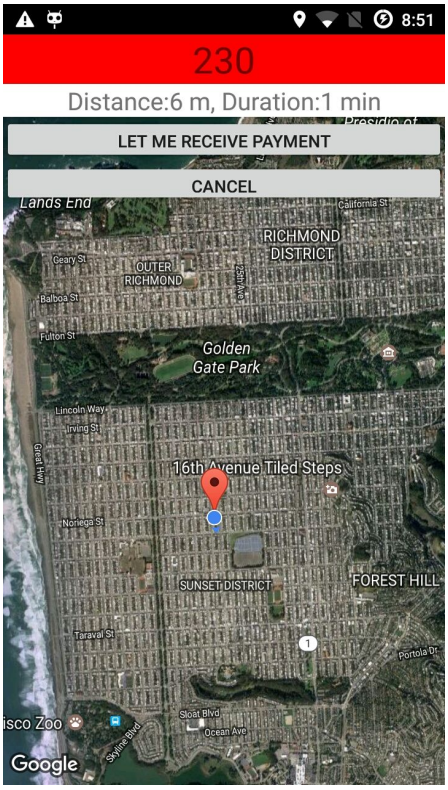


Figure U: *When the Buyer's and Seller's coordinates are within a certain radius, a button "Let me Receive Payment" will show that prompts the user to the next activity.*

MapsActivityBuyer

The user is trying to buy the pin from the seller.

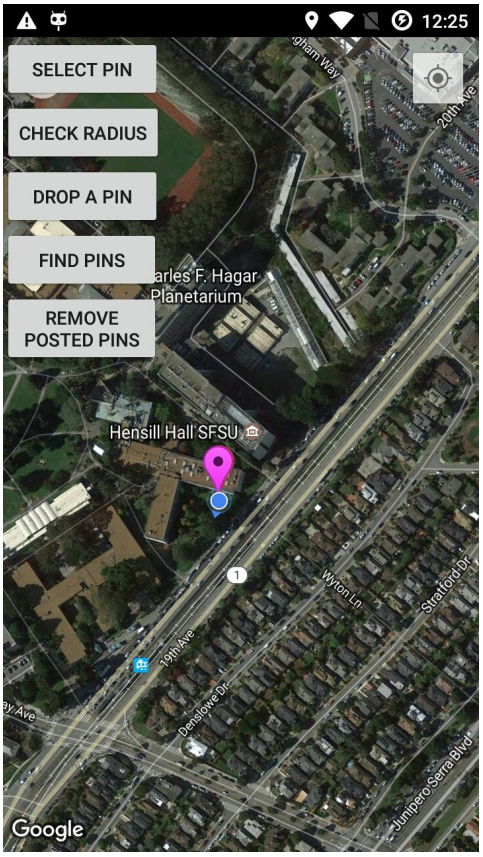


Figure V: *The Maps Buyer screen*

The alogrithm to find the seller's pin is simple. Recall how we were able to retrieve all the coordinates from the database from the server.

A coordinates class uses volley to hold all the data from the database. By calling gc.getData(), all the data from the database is held into the coordinates array.

```
//Use coordinates class to grab data from the SQL database
GetCoordinates gc = new GetCoordinates();
gc.getData();
coordinates = gc.returnCoordinates();
```

Figure W: *All the data from the database is held into the coordinates array.*

We drop all the Seller pins on the maps, but we only set the ones within the radius to be seen. To set the parameters within the radius, we check in the coordinates class that

holds all of our seller pins if they are close to the current location(or within the radius) of the buyers. If they are within the radius and are close enough, then the pin is dropped.

```
//Loop through the coordinates class that holds all SQL data
for (int i = 0; i < coordinates.length; i++) {
   //Check the radius and drop pins

//If the latitude of seller and coordinates of buyer is within interval of .0050
   if (coordinates[i].getLatitude() > (lat - 0.086205905) &&
coordinates[i].getLatitude() < (lat + 0.086205905)) {

 //If the longitude of seller and coordinates of buyer is within interval of 0.1096916

if (coordinates[i].getLongitude() > (longi - 0.10969162) &&
coordinates[i].getLongitude() < (longi + 0.10969162)) {

              //Customize the marker to drop in certain coordinates
           MarkerOptions marker = new MarkerOptions().position(
                   new LatLng(coordinates[i].getLatitude(),
coordinates[i].getLongitude()))
                   .title(coordinates[i].getName() + " | " +
coordinates[i].getLicense());

marker.icon(BitmapDescriptorFactory.defaultMarker(BitmapDescriptorFactory.HUE_GREEN));
        //Add the green marker to the map
            mMap.addMarker(marker);
         }
    }
}
```

Figure X: *This shows that if Seller pin is within Buyer pin's radius, they are made visibile onto the google map and dropped onto the screen*

When the Buyer finds the Seller Pin that they want and presses "Select Pin, the Buyer grabs the Seller pin's coordinates.

```
SelectPin.setOnClickListener(new View.OnClickListener() {

   @Override
   public void onClick(View v) {

       mMap.setOnMarkerClickListener(new GoogleMap.OnMarkerClickListener() {
           @Override
           public boolean onMarkerClick(Marker marker) {

             //Assign marker of the Seller Pin's coorrdinates from touch
               LatLng position = marker.getPosition();

               //Grab the seller pin's coordinates
               SellerLatitude = position.latitude;
               SellerLongitude = position.longitude;
               mMap.clear();
               MarkerOptions marker1 = new MarkerOptions().position(
                       new LatLng(position.latitude, position.longitude))
                       .title("Potential Seller");
```

)};

Figure Snippet Y: Shows the Buyer grabbing the pin latitude and longitude of the Seller's pin

Once the Buyer has the Seller's coordinates, the buyer will start a timer task thread that pushes his current location every five seconds and repeatedly checks the database of the seller's pin to see if the seller pin and buyer pin's coordinate match.

```
//Method always called when current location is changed
public void onLocationChanged(Location location) {

    mLastLocation = location;

    if (mCurrLocationMarker != null) {
        mCurrLocationMarker.remove();
    }

    //Place current location marker
    BuyerLatitude = location.getLatitude();
    BuyerLongitude = location.getLongitude();
}
```

***Figure Snippet Z**: Every single time the Buyer coordinate's changed, onLocation method is called and BuyerLatitude and BuyerLongitude is assigned the Buyer's current location.*

```
public class MyTimerBuyer extends TimerTask {

            @Override
        public void run() {

            //Class takes BuyerLatitude, BuyerLongitude, and BuyerNamee as parameters to
push into the database
 AlterCoordinates ac = new AlterCoordinates{BuyerLatitude, BuyerLongitude, Buyername,
getResponse2);

            //Instantiate request queue to make a request
             RequestQueue queue2 = Volley.newRequestQueue(MapsActivityBuyer.this);
            //Execute the request
             queue2.add(ac);
}}
```

***Figure Snippet A2**: TimerTask repeatedly pushing the Buyer current location coordinates in the database using volley.*

We take the pins from the database using coordinates class again, and compare the buyer's coordinates with the seller's coordinates. If they match, then a button pops up that allows you to go to the payment activity.

//Loop through the coordinates class, going thorugh the whole database data

```
for (int i = 0; i < coordinates.length; i++) {

       //If the latitude of seller and coordinates of buyer is within interval
//of .0050
   if (coordinates[i].getLatitude() > (coordinates[i].getBuyerLatitude() -
0.00050) && coordinates[i].getLatitude() < (coordinates[i].getBuyerLatitude() +
0.00050)) {
       //If the longitude of seller and coordinates of buyer is within interval
//of .0050
        if (coordinates[i].getLongitude() > (coordinates[i].getBuyerLongitude() -
0.00050) && coordinates[i].getLongitude() < (coordinates[i].getBuyerLongitude()
+ 0.00050)) {
       //If the names match to verify it is the same name in the database
            if (coordinates[i].getBuyerName().equals(name1)) {
       //Make the pay button visible onto the screen
                payButton.setVisibility(View.VISIBLE);
            }
        }
   }
}
```

***Figure Snippet C2**: Button appears only when the coordinates grabbed from database and buyer's coordinates match*

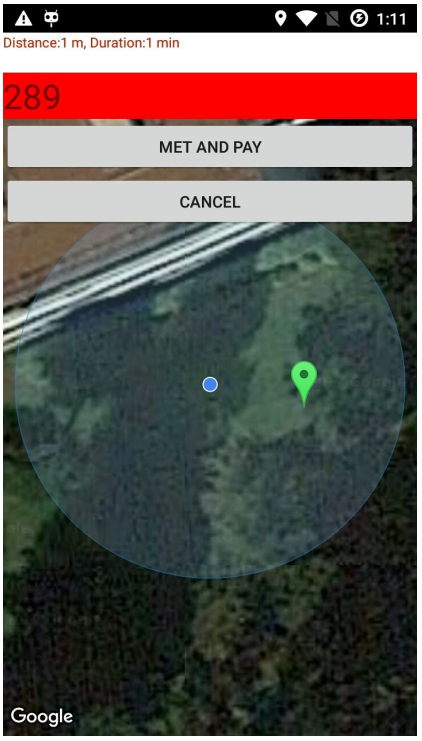


***Figure Snippet D2**: Shows the Met And Pay button that allows the user to go into the next activity*

Stage 4: PaymentBuyer / PaymentSeller

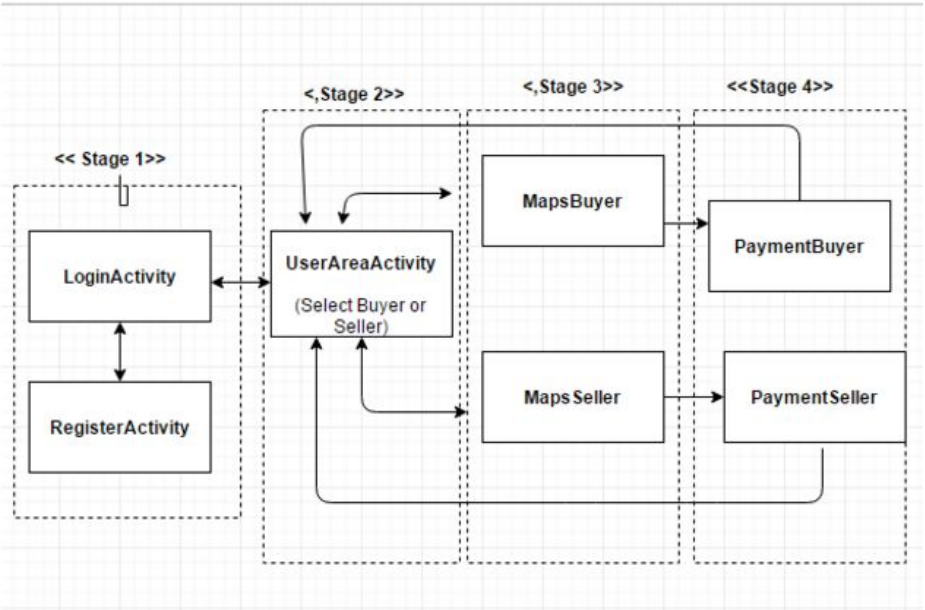


***Figure E2**: Shows the fourth stage of the organization of this project*

PaymentBuyer

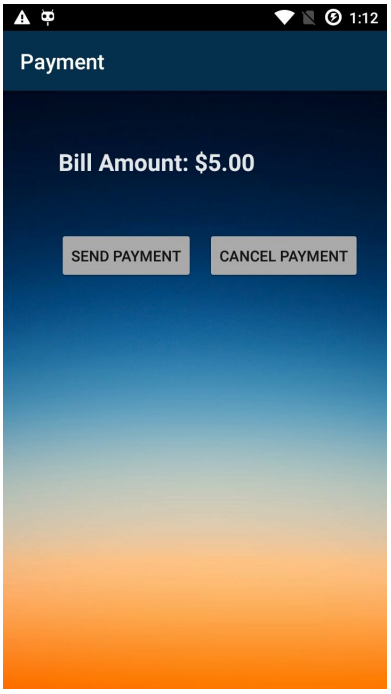


***Figure F2:** Payment activity to send the payment*

When User clicks the send payment, the user changes the database payment from 0 to 1.

```
//Pay button in Buyer Activity
payButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Toast.makeText(Payment_seller.this, "Transaction successful!",
Toast.LENGTH_LONG).show();

        //Pass CheckPayment class to send to database
        CheckPayment cp = new CheckPayment(payment, responseListener);
        //Instantiate request queue to send to PHP file to query
        RequestQueue queue2 = Volley.newRequestQueue(Payment_buyer.this);
        //Execute the request
        queue2.add(cp);

        //Declare a new intent
        Intent intent = new Intent(Payment_buyer.this, UserAreaActivity.class);
        //Put the varaibles to be passed in the intent
        intent.putExtra("name", name);
        intent.putExtra("username", username);
        intent.putExtra("age", license);
        //Execute the intent
        Payment_buyer.this.startActivity(intent);
        myTimer.cancel();
        //Finish and kill the activity
        finish();
    }
});
```

Figure G2: *Code that shows what happens when Buyer clicks that payment has been received*

user_id	name	license	latitude	longitude	selected	BuyerLatitude	BuyerLongitude	BuyerName	Payment
27	Ramsav	4ASPPO	38	-122	0	0	0	XXX	0
83	Kendrick	kendrick	37.7534424	-122.4878145	1	37.7534212	-122.4878222	Elaine	0
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figure H2: *Shows the database before payment has been sent*

user_id	name	license	latitude	longitude	selected	BuyerLatitude	BuyerLongitude	BuyerName	Payment
7	Ramsav	4ASPPO	38	-122	0	0	0	XXX	0
3	Kendrick	kendrick	37.7534424	-122.4878145	1	37.7534212	-122.4878222	Elaine	1
ULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	

Figure I2: *Shows that payment in "Kendrick" entry has been sent*

The user is then redirected back to stage 2 (UserAreaActivity) with a notification that shows that the transaction has been successful.

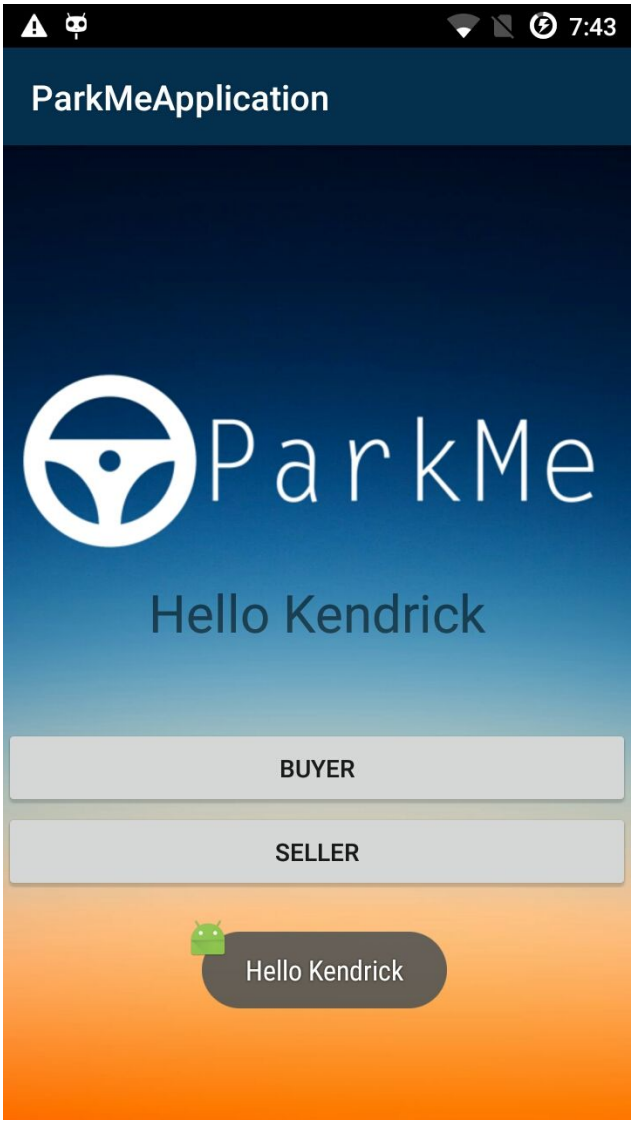


Figure J2: *Buyer is taken back to the UserAreaActivity*

PaymentSeller

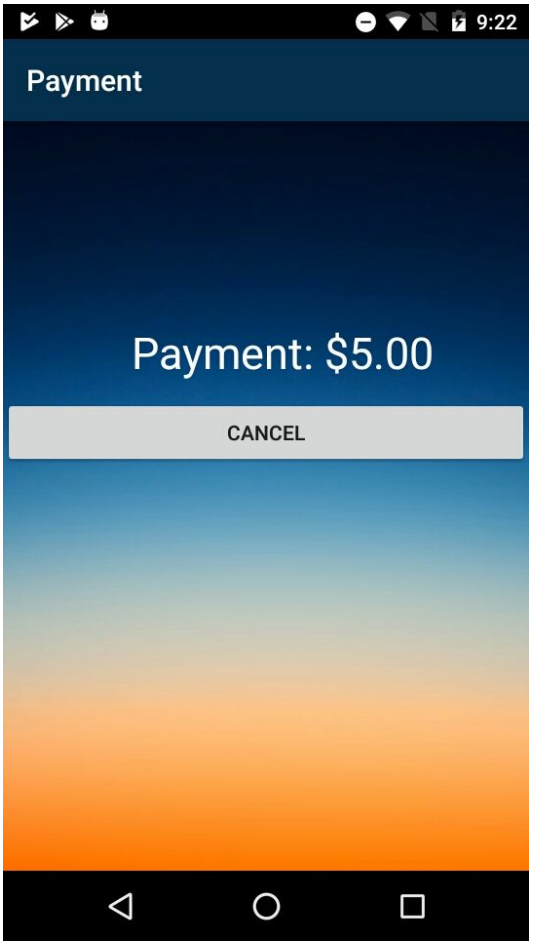


***Figure K2**: Initial Payment Seller Activity*

As you can see, the payment seller activity initially only has the cancel button. How does the Seller receive the notification that the buyer has payed for the spot? Recall **Figure L3** in the buyer activity.

```
//Pass CheckPayment class to send to database
CheckPayment cp = new CheckPayment(payment, responseListener);
//Instantiate request queue to send to PHP file to query
RequestQueue queue2 = Volley.newRequestQueue(Payment_buyer.this);
//Execute the request
queue2.add(cp);
```

***Figure Snippet L2:**Volley Request that changes the payment from 0 to 1*

From the payment buyer activity, when the buyer pays for the spot he sends a volley request to change the database payment query from 0 to 1. Recall this from **Figure H2.**

user_id	name	license	latitude	longitude	selected	BuyerLatitude	BuyerLongitude	BuyerName	Payment
7	Ramsav	4ASPPO	38	-122	0	0	0	XXX	0
3	Kendrick	kendrick	37.7534424	-122.4878145	1	37.7534212	-122.4878222	Elaine	1
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	

FigureH2: *Shows that payment in "Kendrick" entry has been sent*

For the Payment Seller activity, the Seller will create a timer task that periodically checks the database if the payment field for his entry has been set to 1.

```
        public class PaymentWait extends TimerTask {


        //Run the thread five seconds
           @Override
           public void run() {

        //Grab the SQL Data and store it into coordinates
                GetCoordinates gc = new GetCoordinates();
                gc.getData();
                coordinates = gc.returnCoordinates();


                for (int i = 0; i < coordinates.length; i++) {

        //If condition to find the Seller entry && if the payment field has been set to 1
        if (coordinates[i].getLatitude() == sellerLatitude &&
coordinates[i].getLongitude() == sellerLongitude && coordinates[i].getPayment() == 1) {

                        //Make the pay button visible
                        payButton.setVisibility(View.VISIBLE)
                    }
                }
           }
        }
```

Figure Snippet M2: *Shows the code to make button visible if Seller has received notification*

If the payment field for the entry of the Seller has been set to 1, then that means that the buyer has paid. The Payment seller will repeatedly check if the Buyer has payed for the spot. Another timer task will be called every five seconds to check if the payment is equal to 1. If the payment is equal to 1, then a button will pop up

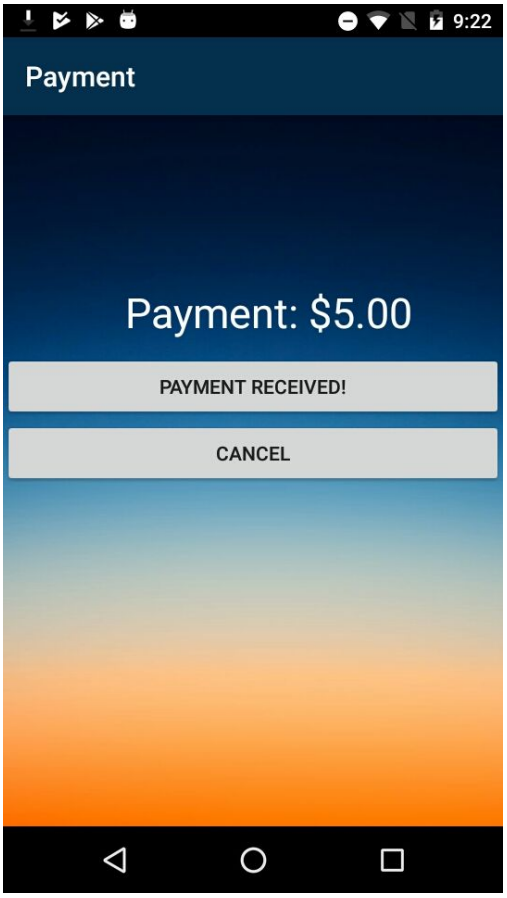


Figure N2: *Button pops up on screen when notification has been received*

After Seller has clicked that he has received the payment, then he will be taken back to the UserAreaActivity.

//Declare pay button

```
payButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Toast.makeText(Payment_seller.this, "Transaction successful!",
Toast.LENGTH_LONG).show();

        //Declare a new intent
        Intent intent = new Intent(Payment_seller.this, UserAreaActivity.class);
        //Put the varaibles to be passed in the intent
        intent.putExtra("name", name);
        intent.putExtra("username", username);
        intent.putExtra("age", license);
        //Execute the intent
        Payment_seller.this.startActivity(intent);
        myTimer.cancel();
        //Finish and kill the activity
        finish();
    }
});
```

Figure O2: *Code that shows what happens when Seller clicks that payment has been received*

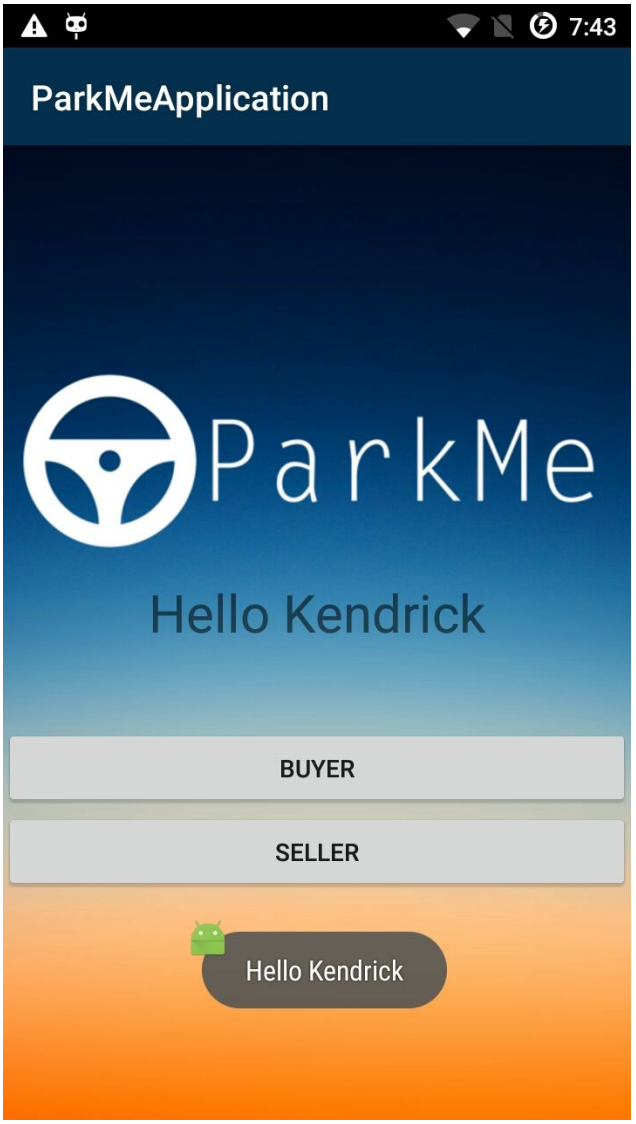


Figure P2: *Seller is taken back to the UserAreaActiv*