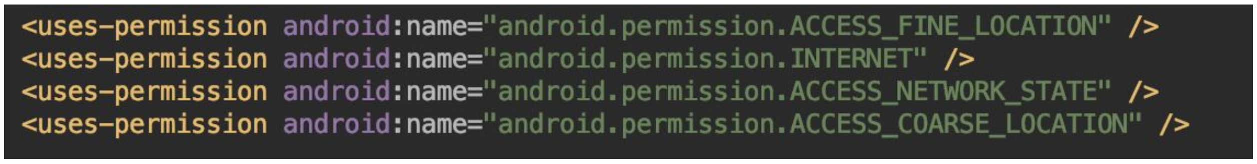
Overview:

I believe that overall, the final product has been successful in solving the problem that it sought out to. It provides the functions that have been listed out in the success criteria, giving the user a variety of options in using it. The algorithms used in this process have remained about the same, with the local data being handled by an object class I created. However, some features of the application have been updated since the original plan. For example, I added an option to log out of the application, which allows the user to return to the welcome page and sign up or sign in. I also added an option that allows the user to name their itineraries so they can retrieve that exact itinerary in the future.

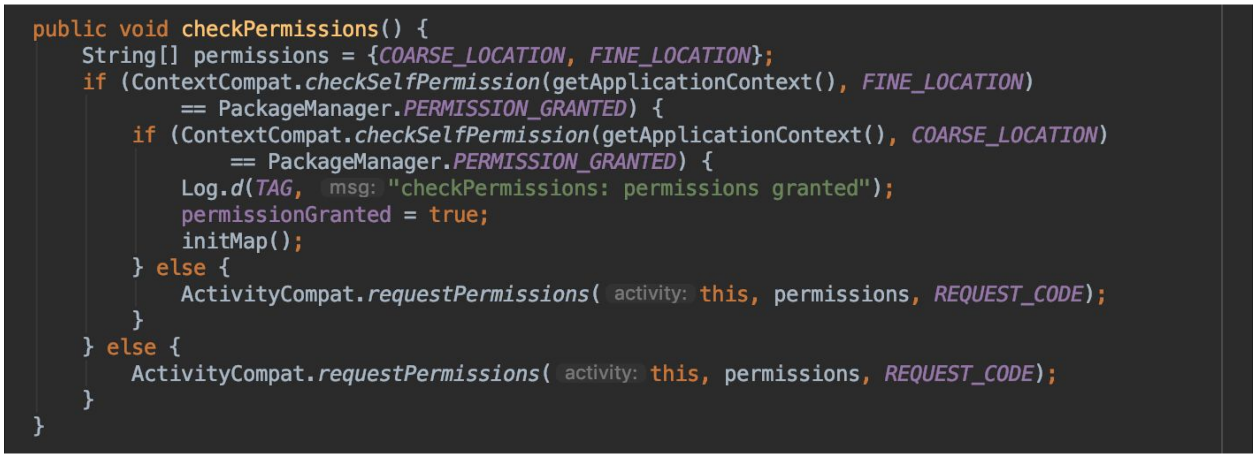
Success Criteria	**Success?**
The program must give at least 10 different landmarks and 5 different hotel options to the user.	After counting the available landmarks and hotels on the map, the program succeeds in this regard.
The program must create an itinerary containing travel times and staying times.	After running the application and testing it by inputting landmarks and times, the program successfully generates an itinerary.
The program must implement Google Maps API for interface with the user.	The program's "MapsActivity" makes use of the Google Maps API to generate an interactive map for the GUI.
The program must have a user sign up system	The "SignupActivity" allows the user to sign up with their email and password.
The program must have a user login system	The "LoginActivity" allows previous users to log into the app.
The program must use the FireBase API to register and store user data	The program makes use of the FirebaseAuth API to store user data and the FirebaseDatabase API to store itineraries.
The program must allow users to store previous itineraries	The program uses the FirebaseDatabase to store user itineraries.
The program must be able to successfully calculate tour times	The program uses a built-in algorithm to calculate the amount of time the tour will last.

Techniques:

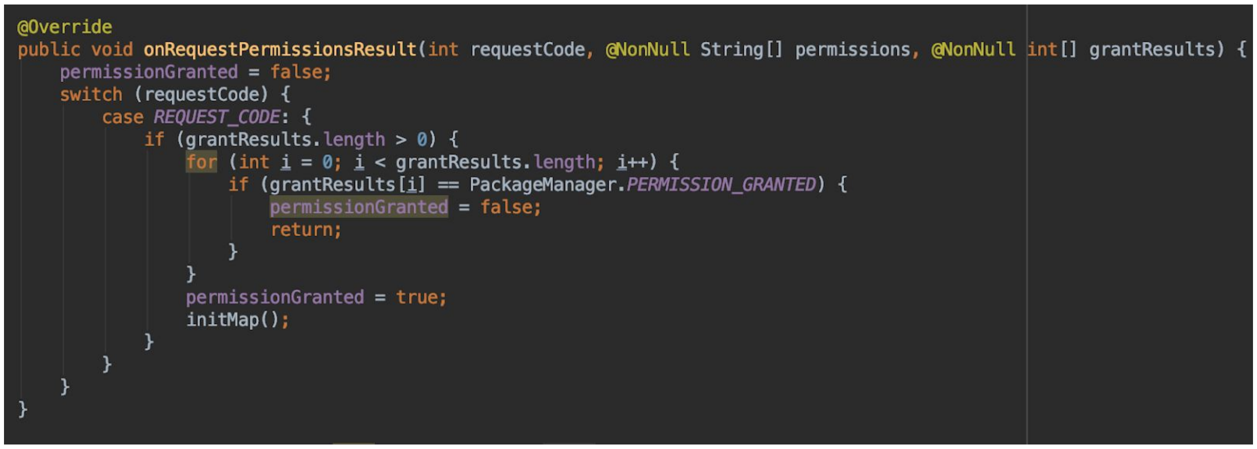


```xml
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
```

The techniques I used to build my program were mainly used in the manipulation of the network and location accesses. These permissions were changed in the android manifest XML file of Android Studio in the project, allowing me to access the user's current location as well as allowing me to check the status of the device's connection to the internet.



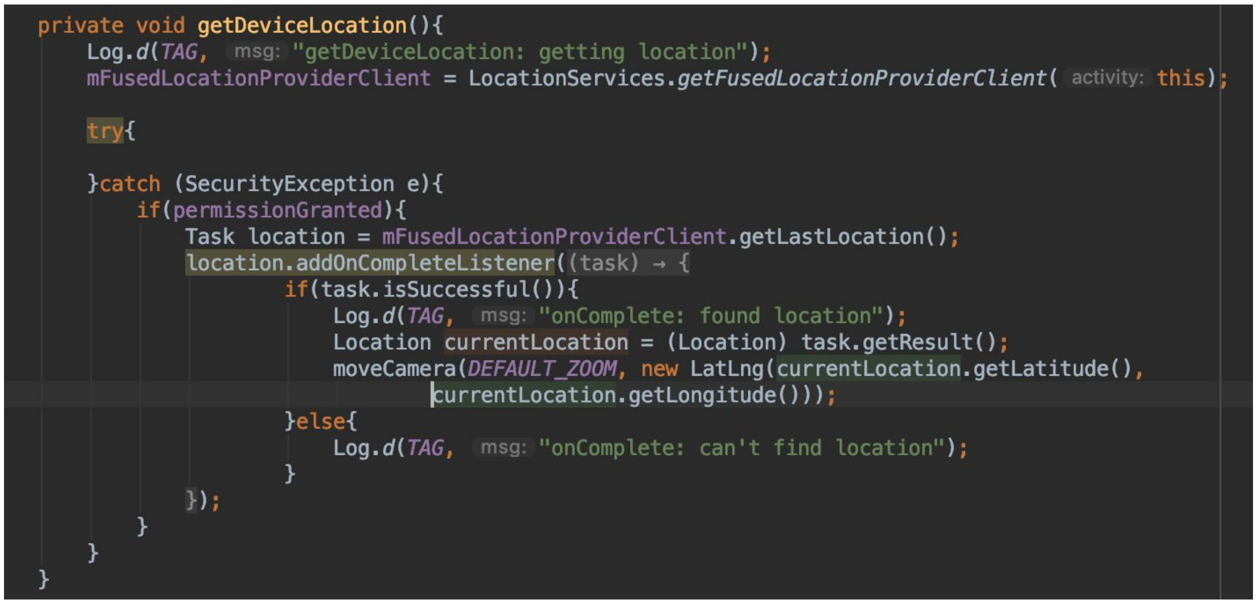
```java
public void checkPermissions() {
    String[] permissions = {COARSE_LOCATION, FINE_LOCATION};
    if (ContextCompat.checkSelfPermission(getApplicationContext(), FINE_LOCATION)
            == PackageManager.PERMISSION_GRANTED) {
        if (ContextCompat.checkSelfPermission(getApplicationContext(), COARSE_LOCATION)
                == PackageManager.PERMISSION_GRANTED) {
            Log.d(TAG, msg: "checkPermissions: permissions granted");
            permissionGranted = true;
            initMap();
        } else {
            ActivityCompat.requestPermissions( activity: this, permissions, REQUEST_CODE);
        }
    } else {
        ActivityCompat.requestPermissions( activity: this, permissions, REQUEST_CODE);
    }
}
```



```java
@Override
public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions, @NonNull int[] grantResults) {
    permissionGranted = false;
    switch (requestCode) {
        case REQUEST_CODE: {
            if (grantResults.length > 0) {
                for (int i = 0; i < grantResults.length; i++) {
                    if (grantResults[i] == PackageManager.PERMISSION_GRANTED) {
                        permissionGranted = false;
                        return;
                    }
                }
                permissionGranted = true;
                initMap();
            }
        }
    }
}
```

These methods make use of the permissions I enabled by checking the device for location permissions before initialising the Google map. By using the ContextCompact object and checking permissions for the application's Fine location and Coarse location, the technique allows the program to make sure all permissions are granted by the device before running the map. The first method, checkPermissions(), checks if the current device has already enabled the permissions. If it hasn't, the method will use the requestPermissions() to directly request the user for the permission to access their location.

The second method, onRequestPermissionsResult(), is an override method that runs when the permission request results are returned. The method checks if the permissions were granted by the user, and if they are the program will create the Google Map. These location permissions are very important to the program because it allows it to find the user's location, which will play a part in the next method:



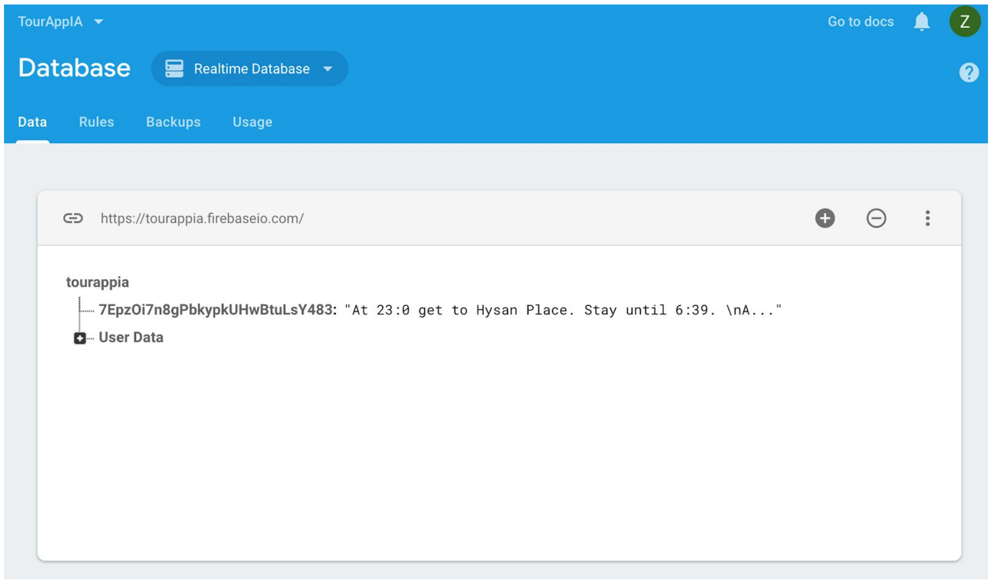
```
private void getDeviceLocation(){
    Log.d(TAG,  msg: "getDeviceLocation: getting location");
    mFusedLocationProviderClient = LocationServices.getFusedLocationProviderClient( activity: this);

    try{

    }catch (SecurityException e){
        if(permissionGranted){
            Task location = mFusedLocationProviderClient.getLastLocation();
            location.addOnCompleteListener((task) → {
                    if(task.isSuccessful()){
                        Log.d(TAG,  msg: "onComplete: found location");
                        Location currentLocation = (Location) task.getResult();
                        moveCamera(DEFAULT_ZOOM, new LatLng(currentLocation.getLatitude(),
                                currentLocation.getLongitude()));
                    }else{
                        Log.d(TAG,  msg: "onComplete: can't find location");
                    }
            });
        }
    }
}
```

This method uses the location permissions previously granted to find the user's current location. As the code shows, the method uses an if() statement to check whether or not the permissions have been granted by the user. The method also uses the Google Location Services Fused Location Provider Client to use GPS and locate the user's current coordinates. This method uses the programming techniques I implemented to play an integral role in the program, as the whole point of the interactive map is to show the available landmarks relative to the user's current location.

Data Structures:

For this application, the main local data is stored in the form of Arraylists. As the Landmarks are their own object class, the Landmarks can then be easily stored and retrieved by storing them in an ArrayList. I chose to use an ArrayList because Arraylists provide an easy method to find objects within using the .get() method and can easily be added to by using the .add() method. This will allow me to easily enter large amounts of data into the ArrayList without needing to re-order to objects within or change anything. Another reason I chose to use an ArrayList instead of a linked list or tree or Array because the ArrayList is a dynamic data structure which will allow me to store large amounts of data without needing to resize the data structure or use complicated methods to copy the data from one structure to another.

In terms of user data stored non-locally, I used the Google Firebase Database to store the data that needs to be saved outside of the application. I used Firebase to store the user credentials on signup, and use it to check credentials when logging users in. I also used the Firebase Database to store the user's itineraries under their unique UID and logged it under a name that they choose. I chose this data structure because it allows me to store user data non-locally, allowing me to save the user data even if he ends the application. This allows me to implement functions such as user sign-up and sign-in.



```
touruappia
```

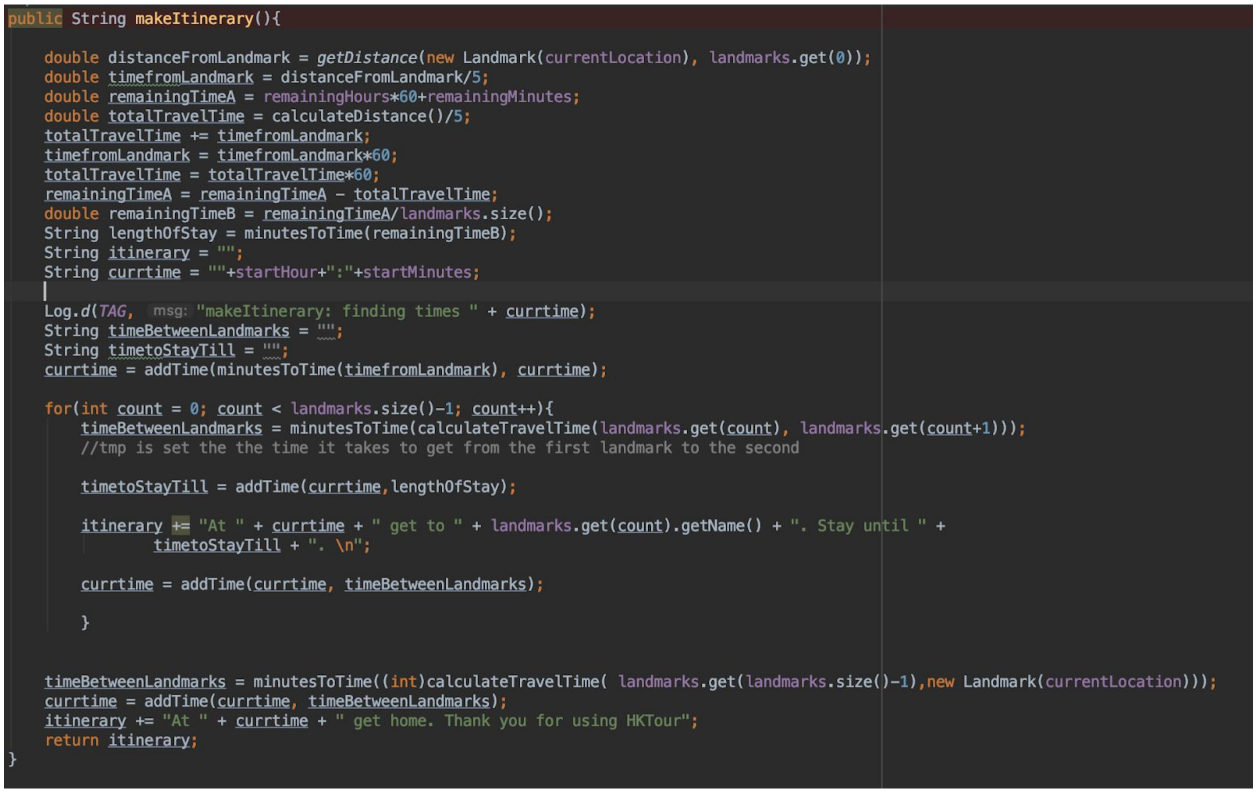
touruappia

7EpzOi7n8gPbkypkUHwBtuLsY483: "At 23:0 get to Hysan Place. Stay until 6:39. \nA..."

User Data

Algorithmic Thinking:

The way the algorithms handle the data in this application has remained constant throughout the entire design process. In order to create a functioning GUI, most of the application's code and data is passed through activities and not classes. However, in order to allow for the application to run smoothly, the activities cannot handle too much data or the application will lag heavily. Therefore, I used an external class called "Evaluate" to handle all of the complex calculations and algorithms. The most important method in the Evaluate class is the "createItinerary()" method, which makes use of all the information passed into the class to generate the itinerary for the user.



```java
public String makeItinerary(){

    double distanceFromLandmark = getDistance(new Landmark(currentLocation), landmarks.get(0));
    double timefromLandmark = distanceFromLandmark/5;
    double remainingTimeA = remainingHours*60+remainingMinutes;
    double totalTravelTime = calculateDistance()/5;
    totalTravelTime += timefromLandmark;
    timefromLandmark = timefromLandmark*60;
    totalTravelTime = totalTravelTime*60;
    remainingTimeA = remainingTimeA - totalTravelTime;
    double remainingTimeB = remainingTimeA/landmarks.size();
    String lengthOfStay = minutesToTime(remainingTimeB);
    String itinerary = "";
    String currtime = ""+startHour+":"+startMinutes;

    Log.d(TAG,  msg: "makeItinerary: finding times " + currtime);
    String timeBetweenLandmarks = "";
    String timetoStayTill = "";
    currtime = addTime(minutesToTime(timefromLandmark), currtime);

    for(int count = 0; count < landmarks.size()-1; count++){
        timeBetweenLandmarks = minutesToTime(calculateTravelTime(landmarks.get(count), landmarks.get(count+1)));
        //tmp is set the the time it takes to get from the first landmark to the second

        timetoStayTill = addTime(currtime,lengthOfStay);

        itinerary += "At " + currtime + " get to " + landmarks.get(count).getName() + ". Stay until " +
                timetoStayTill + ". \n";

        currtime = addTime(currtime, timeBetweenLandmarks);

        }


    timeBetweenLandmarks = minutesToTime((int)calculateTravelTime( landmarks.get(landmarks.size()-1),new Landmark(currentLocation)));
    currtime = addTime(currtime, timeBetweenLandmarks);
    itinerary += "At " + currtime + " get home. Thank you for using HKTour";
    return itinerary;
}
```

As shown in the screenshot, the main algorithm in the class Evaluate actually makes use of many smaller private methods to handle the large number of calculations that are required to generate this itinerary. The algorithm takes the information already given to the class (such as start time, end time, current location) and uses private methods to calculate the total distance, tour time, and travel time. This allows the algorithm to equally divide the remaining time between the number of landmarks. The algorithm then uses a simple for loop to iterate through the given Arraylist of landmarks, adding to the "itinerary" string after making the necessary calculations.

Outside Sources:

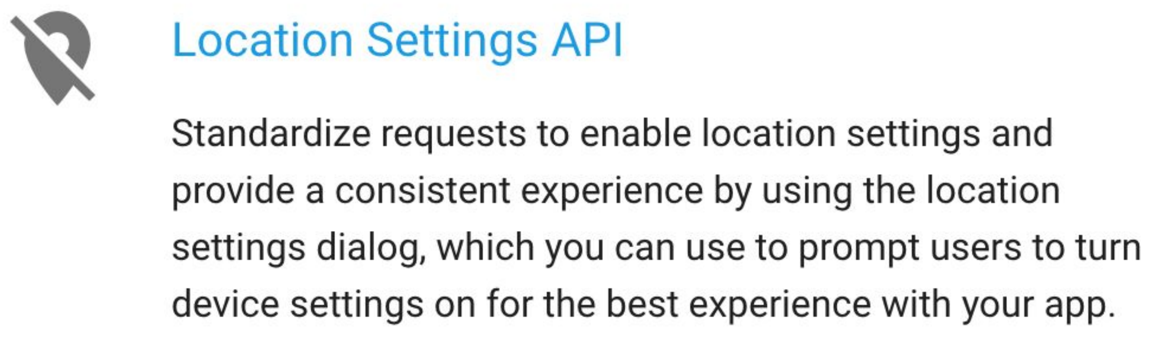
https://www.youtube.com/channel/UCoNZZLhPuuRteu02rh7bzsw
CodingwithMitch - Android Google Maps API Series

https://www.youtube.com/user/SimplifiedCoding
Simplified Coding - Android Firebase API Tutorial Series

https://www.youtube.com/watch?v=FhobmAHGVwI
This video about the User Registration function of Firebase

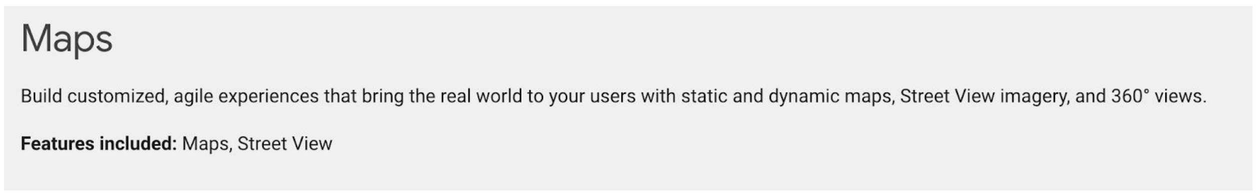
Tools:

The external libraries I used were all provided by Google and integrated into Android Studio by installing dependencies. The first library I used was the Google Maps API, which allowed me to display the Google Map in my Maps Activity. This was useful to me because I could display a physical representation of the different landmarks available to the user on an interactive Map. I also used the Google Location Library which comes along with the Google Maps API, which allowed me to use Google Location Services to find the user's current position and display in relation to the other landmarks. These libraries played a key role in forming my GUI for the user and making the overall user experience more pleasant by providing good graphics and interactive activities.



Location Settings API

Standardize requests to enable location settings and provide a consistent experience by using the location settings dialog, which you can use to prompt users to turn device settings on for the best experience with your app.

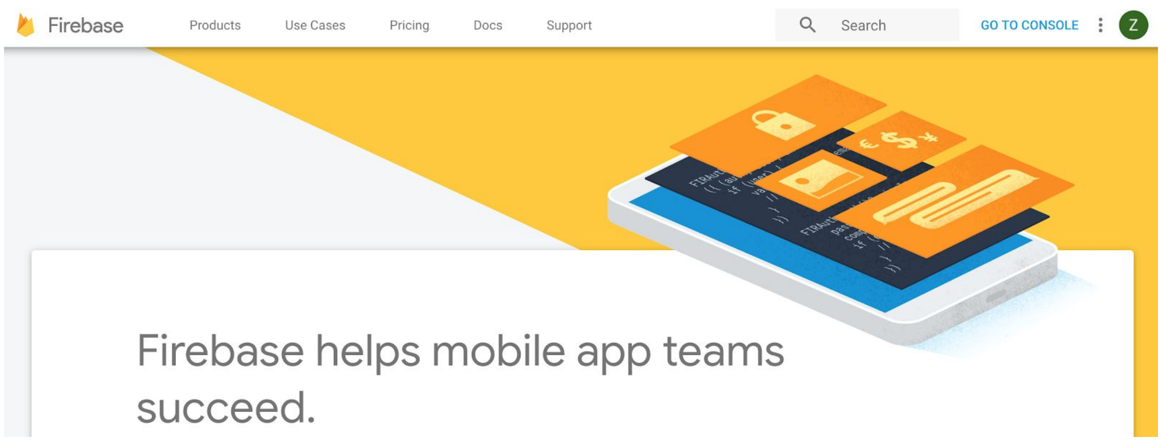


Maps

Build customized, agile experiences that bring the real world to your users with static and dynamic maps, Street View imagery, and 360° views.

Features included: Maps, Street View

Another external library that I used was the Google Firebase API. I used two specific functions of the Firebase API, which were the FirebaseAuth and FirebaseDatabase. As stated before, the Firebase API allowed me to implement functions such as user registration and user log-in, which is a function of the FirebaseAuth API. This allowed me to create users automatically using the createNewUser() method, which allowed me to easily register the users of my application with their email and chosen password.



Firebase helps mobile app teams succeed.