# GeoScene

## Augmented Reality and POI Geolocation Scenery viewer Android mobile application

Itay Bouganim, Sahar Vaya, Lior Hassan

Software Engineering final project

Ben-Gurion University of the Negev

# Table of contents

# Chapter 1
# Use-Cases

## 1.1 User Profiles – The Actors

The actors in our system are mostly end users that are not logged nor registered to the external map provider, they may have several different characteristics but the use cases for all of the users are similar. Users that are already registered and logged in to the external map provider using the platform the application provides, will be able to use additional functionality provided by the external map source. Therefore, the system actors that are logged in to the external map provider will have specific unique use-cases.

*Human Actors*

- Users that are not logged in to the external map provider – will be the main expected audience of the application and are predicted to consist mainly of travelers and tourists that would like to gain spatial orientation and get information about their surroundings.
- Users that are logged in to the external map provider – will be users that were previously registered to the external map provider and can use its services through the application interface in order to perform mapping operations (adding, editing or removing locations).
- Users that are associated with a group that can identify locations using triangulation – will be users that can use the application AR triangulation feature (will be elaborated later in the document) in order to identify non-marked locations without pin-pointing their exact location on a map.
- Predefined Organization Administrators – will be predefined users that will be able to add new users to the user group allowed to use the application triangulation feature.

## 1.2 Use-Cases

### 1.2.1 Viewing nearby places using AR

*Primary actors:*
All application users

*Preconditions:*
- Mobile device active internet connection
- Available mobile device GPS signal
- Available mobile device camera
- Given the application permission to use mobile device sensors and camera.
- The applications elevation usage settings are on/off.

*Postconditions (success guaranty):*
- A location marker is shown in AR view using the mobile device camera above every place considering:
  1. Elevation settings on - The user can physically see the location (Currently in the users FoV).
  2. Elevation settings off – All the places that are surrounding the user (in line of sight or not).
- Every location marker shown contains the correct name and additional details of the real-world location located in the location markers orientation.

*Main success scenario (Basic flow):*
1. The user chooses to open the AR view feature of the application.
2. The AR view of the application has initialized.
3. The user looks around with the mobile device camera.
4. Depending on the elevation settings:
   i. Elevation settings on - All the users surrounding places that are currently in his FoV are presented to the user using location markers.
   ii. Elevation settings off - All the users surrounding places are presented to the user using location markers.

*Extensions (Alternate flows):*
1. The user opens the GeoScene application.
2. The user loses GPS signal/Internet connectivity while the application is loading.
3. The application prompts the user to retry and relaunch the application.

*Abstract Example:*
The application is being used by the user inside a crater named “Crater” with 20 other locations surrounding the crater. Since the crater’s elevation is 100m below sea level, considering the elevation settings:
1. Elevation settings on - The user will not be able to see any location markers outside the crater.
2. Elevation settings off – Since the elevation settings of the application is turned off, the user can see 20 location markers, one for each surrounding POI around the crater.

### 1.2.2 Assigning a user to the triangulation group

*Primary actors:*
Predefined Organization Administrator

*Preconditions:*
- Internet connectivity
- The administrator is logged in to the GeoScene web application using global administrator credentials.
- The user the administrator wishes to assign to the triangulation group is registered to the external map provider service.

*Postconditions (success guaranty):*
- The assigned user will be able to use the triangulation feature that is now available for him in the application.

*Main success scenario (Basic flow):*
1. The organization administrator opens the GeoScene web application.
2. The organization administrator chooses to option to assign a user to the triangulation group.
3. The organization administrator inserts the user credentials (i.e. email).
4. The organization administrator approves the assignment of the user to the group.
5. The organization administrator is prompted with a message indicating that the user was assigned successfully.

*Extensions (Alternate flows):*
1. The organization administrator opens the GeoScene web application.
2. The organization administrator chooses to option to assign a user to the triangulation group.
3. The organization administrator inserts the user credentials (i.e. email).
4. The organization administrator is prompted with a message indicating that the user is already in the triangulation group.
5. The organization administrator is re-routed to the assignment page of the web application.

### 1.2.3.a Adding locations to the external map provider – using map

*Primary actors:*
Users that are logged in to the external map provider

*Preconditions:*
- Mobile device active internet connection
- Available mobile device GPS signal
- Available mobile device camera
- Given the application permission to use mobile device sensors and camera.
- The user is registered to the external map provider.
- The user is logged in to the external map provider.

*Postconditions (success guaranty):*
- A request to add new location with the user current real-world coordinates or map coordinates has been sent to the external map provider.
- The new location inserted contains the additional details provided by the adding user.

*Main success scenario (Basic flow):*
1. The user is presented with a menu option that allows him to add a new location at his current real-world position/option to mark a location on the map.
2. After choosing the Add location functionality the user is automatically navigated to the correct screen.
3. The user inserts the location name and additional information in the appropriate text boxes presented in the screen.
4. The user is prompted that the new location add request was sent successfully.

*Extensions (Alternate flows):*
1. The user opens the GeoScene application.
2. The user navigates to the applications login screen.
3. The user inserts his credentials in order to login.
4. The user is presented with a menu option that allows him to add a new location at his current real-world position/option to mark a location on the map.
5. After choosing the Add location functionality the user is automatically navigated to the correct screen.
6. The user inserts the location name and additional information in the appropriate text boxes presented in the screen.
7. The user is prompted that the new location add request was not sent due to network connectivity issues.

### 1.2.3.b Adding locations to the external map provider – using triangulation

*Primary actors:*
Users that are associated with a group that can identify locations using triangulation

*Preconditions:*
- Mobile device active internet connection
- Available mobile device GPS signal
- Available mobile device camera
- Given the application permission to use mobile device sensors and camera.
- The user is registered to the external map provider.
- The user is logged in to the external map provider.
- The user is associated with a group of users that can perform the triangulation functionality.

*Postconditions (success guaranty):*
- If the user approved an existing triangulation (The second user to triangulate that point) A request to add new location with the triangulated coordinates has been sent to the external map provider. If the user starts a triangulation process, then the information regarding his phone sensors bearings and location is saved to the applications caching service.

*Main success scenario (Basic flow):*
1. The user chooses the triangulation feature menu option.
2. An AR view is opened for the user.
3. The user identified the location he wishes to mark, if it was previously marked the user will be presented with the option to approve this location by an existing triangulation request, else, the user will be presented with the option to apply for a new triangulation request.
4. The user chooses to apply/approve according to the request status.
5. The user is prompted that the new location add request was sent successfully if he currently approved it, or the user is prompted that a new triangulation request has been applied.

*Extensions (Alternate flows):*
1. The user opens the GeoScene application.
2. The user navigates to the applications login screen.
3. The user inserts his credentials in order to login.
4. The user chooses the triangulation feature menu option.
5. An AR view is opened for the user.
6. The user identified the location he wishes to mark, if it was previously marked the user will be presented with the option to approve this location by an existing triangulation request, else, the user will be presented with the option to apply for a new triangulation request.
7. The user chooses to apply/approve according to the request status.
8. The user gets an error message due to connectivity issues.

### 1.2.4 Editing previously added locations

*Primary actors:*
Users that are logged in to the external map provider

*Preconditions:*
- Mobile device active internet connection
- Available mobile device GPS signal
- Available mobile device camera
- Given the application permission to use mobile device sensors and camera.
- The user has previously added the location he is trying to edit.

*Postconditions (success guaranty):*
- A previously added location update request is sent with the new information.

*Main success scenario (Basic flow):*
1. The user opens the GeoScene application.
2. The user navigates to the applications login screen.
3. The user inserts his credentials in order to login.
4. The user is presented with a menu option that allows him to view and edit the location previously added by him.
5. The user chooses the location he wished to edit information for from the presented locations.
6. After choosing the Edit location functionality the user is automatically navigated to the correct screen.
7. The user edits the location information in the appropriate text boxes presented in the screen.
8. The user is prompted that the location edit request was sent successfully.

*Extensions (Alternate flows):*
9. The user opens the GeoScene application.
10. The user navigates to the applications login screen.
11. The user inserts his credentials in order to login.
12. The user is presented with a menu option that allows him to view and edit the location previously added by him.
13. The user chooses the location he wished to edit information for from the presented locations.
14. After choosing the Edit location functionality the user is automatically navigated to the correct screen.
15. The user edits the location information in the appropriate text boxes presented in the screen. The user gets an error message due to connectivity issues.

### 1.2.5 Preservation of data about a chosen location to be used on offline mode

*Primary actors:*
All application users

*Preconditions:*

- Mobile device active internet connection
- Available mobile device GPS signal
- Given the application permission to use mobile device sensors and camera.
- The applications elevation usage settings are on/off.

*Postconditions (success guaranty):*

- Data about the chosen area has been saved on the user's device, contains the surrounding POIs and the elevation raster.
- The chosen coordinate appears in the "saved places" window of the application.

*Main success scenario (Basic flow):*

1. The user navigates to the places window using the application's menu.
2. The user chooses the feature of preserve data about location.
3. The user marks a location (specific point) on the map and chooses the location's area size to be analyzed.
4. The user enters the location name to be shown later in the saved places menu.
5. The application downloads the relevant data from the external resources (POIs in the chosen area, elevation raster, and information about the POIs) and saves the data on the user's device under the entered name.

*Extensions (Alternate flows):*

1. The user opens the GeoScene application.
2. The user loses GPS signal/Internet connectivity while the application is loading.
3. The application prompts the user to retry and relaunch the application.

*Abstract Example:*
The application is being used by the user when planning a family trip to "Masada". The user fears that there will be Internet connectivity problems in the area and want to preserve the location information and use it in offline mode. The user marks the relevant location on the providing map and the data is downloaded to his device.

### 1.2.6 Using preservation data about chosen location on offline mode

*Primary actors:*

All application users

*Preconditions:*

- Available mobile device GPS signal
- Given the application permission to use mobile device sensors and camera.
- The application's elevation usage settings are on/off.
- The user preserved data about the location and the location appears in the “places” window.

*Postconditions (success guaranty):*

- A location marker is shown in AR view using the mobile device camera above every place considering:

1. Elevation settings on - The user can physically see the location (Currently in the users FoV).
2. Elevation settings off – All the places that are surrounding the user (in line of sight or not).

- Every location marker shown contains the correct name and additional details of the real-world location located in the location markers orientation.

*Main success scenario (Basic flow):*

1. The user navigates to the places window using the application’s menu.
2. The user presented with all the preserved location in his device and chooses the relevant location.
3. The user is presented with a menu option that allows him to load and use the preserved data.
4. The AR view of the application has been initialized according to the preserved data.
5. The user looks around with the mobile device camera.
6. Depending on the elevation settings:
7. Elevation settings on - All the users surrounding places that are currently in his FoV are presented to the user using location markers.
8. Elevation settings off - All the users surrounding places are presented to the user using location markers.

*Extensions (Alternate flows):*

1. The user opens the GeoScene application.
2. The user loses the GPS signal while the application is loading.
3. The application prompts the user to retry and relaunch the application.

### 1.2.7 delete preservation data about a chosen location

*Primary actors:*
All application users

*Preconditions:*
- Given the application permission to use mobile device sensors and camera.
- The user preserved data about the location and the location appears in the "places" window.

*Postconditions (success guaranty):*
- The desired location's data has been removed from the user's device.
- The desired location's name doesn't appear in the "places" window.

*Main success scenario (Basic flow):*
1. The user navigates to the "places" window using the application's menu.
2. The user presented with all the preserved location in his device and chooses the relevant location.
3. The user is presented with a menu option that allows him to remove the preserved data.
4. The application removes the stored data from the device.

*Extensions (Alternate flows):*
1. The user navigates to the "places" window using the application's menu.
2. The user presented with all the preserved location in his device and chooses the relevant location.
3. The user is presented with a menu option that allows him to remove the preserved data.
4. The application faces accessing issues.
5. The application prompts the user to retry and relaunch the application.

### 1.2.8 Registration

*Primary actors:*
All application users

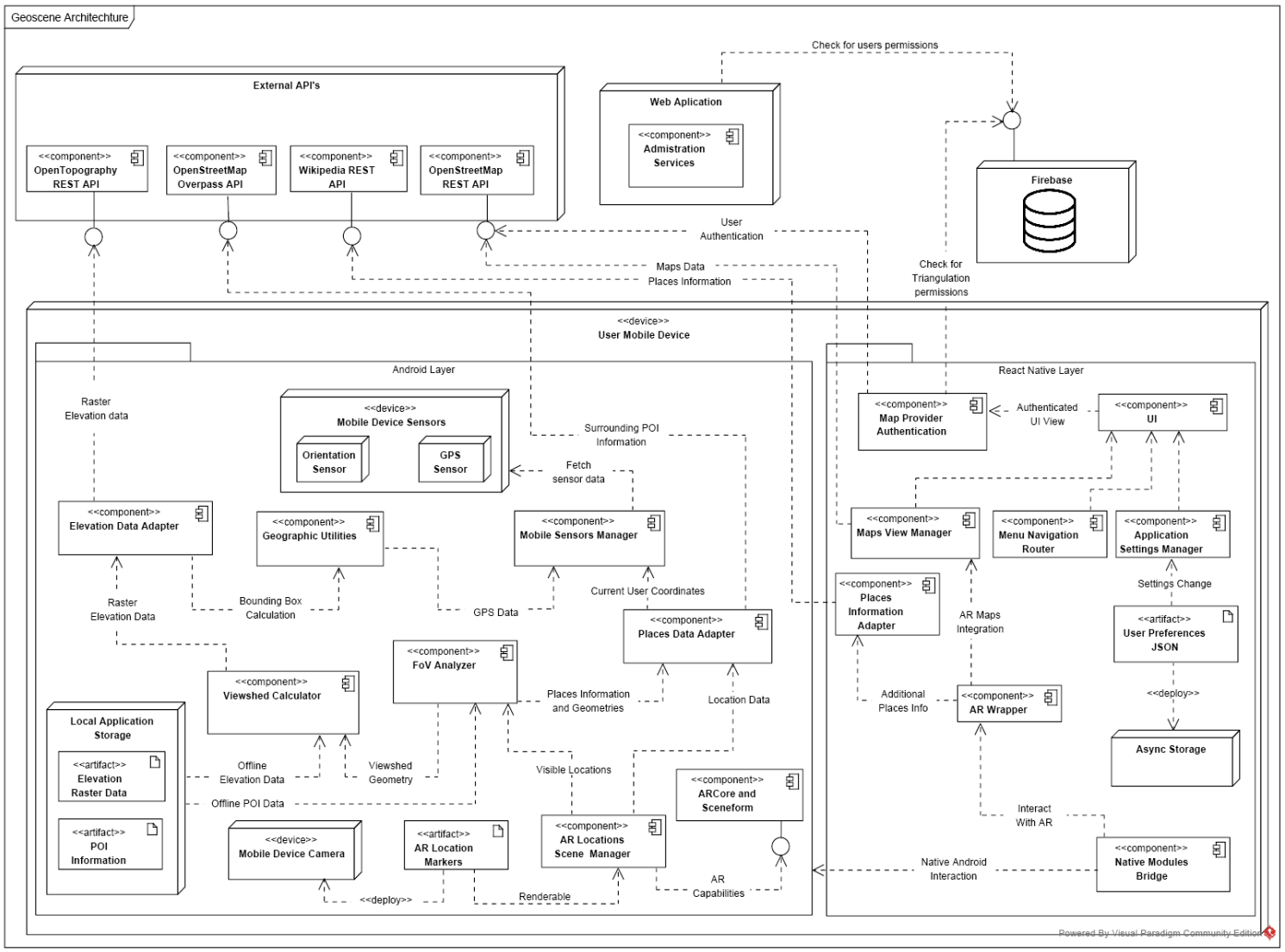
*Preconditions:*
- Mobile device active internet connection

*Postconditions (success guaranty):*
- A new user of the external map provider has been created and saved.

*Main success scenario (Basic flow):*
5. The user opens the GeoScene application.
6. The user navigates to the applications registration screen.
7. The user presented with the external map provider registration system.
8. The user inserts his credentials in order to register.
9. The user is prompted that the registration request was sent successfully.

*Extensions (Alternate flows):*
1. The user opens the GeoScene application.
2. The user loses internet connection while the application is loading.
3. The application prompts the user to retry and relaunch the application.

# Chapter 2
# System Architecture



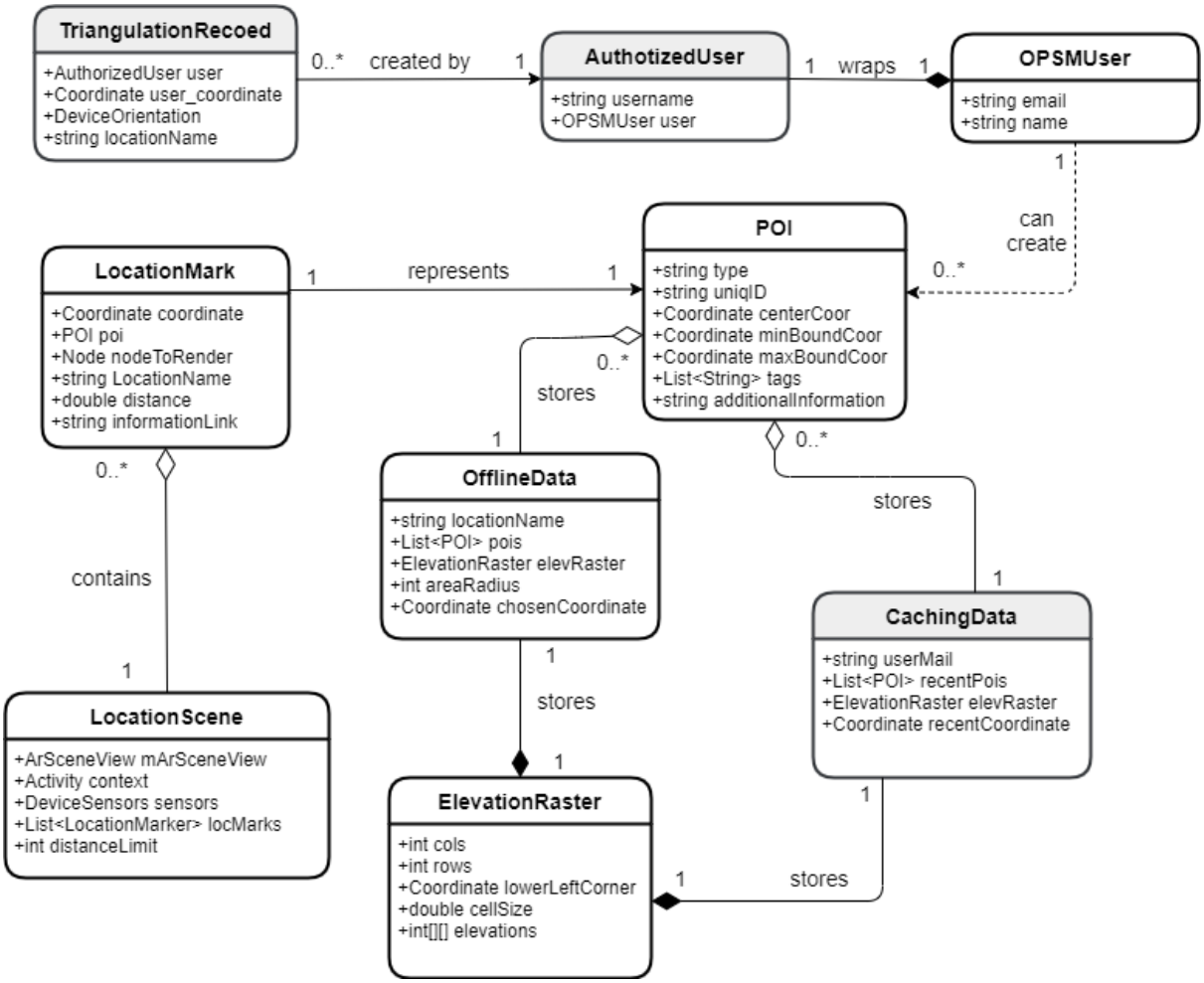
# Chapter 3
# Data Model

## 3.1 Description of Data Objects

The system contains data objects that represent the values and entities of the system. the main data objects are:

- OSMUser – the object represents a registered user. User which registered to the system actually registered to the external map provider, OpenStreetMap. When a user makes a login operation, the system gets the user details from OpenStreetMap and save the relevant details in the OSMUser object. The main attributes are the user's email and name.
- AuthorizedUser – the object represents a user with permission, that assigned to the triangulation enabled group. The object wraps the OSMUser object and has attributes of the OSMUser and username.
- POI – the object represents a point of interest. The object stores information abоIt a POI as obtained from the external map provider query. The attributes are the point type, uniqueID, the center coordinate, area boundary, tags assigned to the point (for example, name, address, website, image, etc.), and additional information (wiki paragraph, additional info added by the creating user, website link, etc.)
- ElevationRaster – the object represents the elevations of the surface that defined by bounding box. The object stores the data that return from the Open Topography external resource. The attributes are the raster size (rows, cols), cell size that represents the area surface samples intervals, a matrix contains the elevation data samples, the coordinate of the point representing the matrix's lower-left corner.
- TriangulationRecord – the object represents a triangulation record created by a user from the triangulation group. The object attributes are the AouthorizedUser created the record, the user coordinate when created the record, the identified location name, and the device sensors and orientation data to be used for the triangulation calculation.
- LocationMarker – the object represents a location mark in the AR view. Each location mark represents POI and contains POI information. The object attributes are a POI to be presented by the mark, the mark location coordinate in the real world, the location name to be written on the mark, the mark distance from the user, link to more information about the location, and Node that uses to render the mark on the user's screen.
- LocationScene – this object represents the real world in the location markers AR space
- OfflineData – the object represents preserved data by the user to be download and store on the user device, in order to be used later in offline mode. The object attributes are the location\area name given by the user, the coordinate that was marked by the user and the desired radius (in km) around the coordinate, list of POIs in the created area and ElevationRaster of the created area,
- CachingData - the object represents a collection of data to be store in the system database and use for caching. the system will save recent data (POIs, POI information, elevation) and will re-load that data from the cache when requested (for example, in case of error or loss of connection). the object attributes are the user identifier (email or username), list of the recent POIs surrounding the user, ElevationRaster of the area, and the user's last coordination.

## 3.2.Data Objects Relationships



## 3.3. Databases

The system uses external non-relational database. each collection in the database contains information depending on the associated data object. The collections data stored as a large JSON tree, with data nodes and their associated keys.

The database supposed to store and manage the following collections:

- **Authorized users**
  This collection contains the users which added to triangulation group. The collection format:

```
{
   username: string
   email: string
   name: string
}
```

- **Caching data**
  This collection uses as cache server, contains the recent data was collected from the application external resources for a logged in user. The collection format:

```
{
   Username: string
   Coordinate: {lat: double, lon: double}
   ElevationRaster: {rows: int,
                     cols: int,
                     lowLeftCorner: {lat: double, lon: double },
                     ElavationMatrix : matrix(int),
                     cellSize: int}
   PoisList: array of objects {locationName: string,
                                   type: string,
                                   lat: double,
                                   lon: double,
                                   Bounds: {maxlat: double,
                                          minLat: double,
                                          maxLon: double,
                                          minLon: double}, description:
                                   string }
}
```

- **Triangulation records**
  This collection contains triangulation samples made by users in the triangulation group. The collection format:

```
{
   recordID: string,
   name: string,
   description: string,
   user: ref (authorizedUser),
   coordinate: {lat: double,
                lon: double},
   status: string
   bearing: double,
   image : jpg,
   screenshot : jpg
}
```

## Transactions

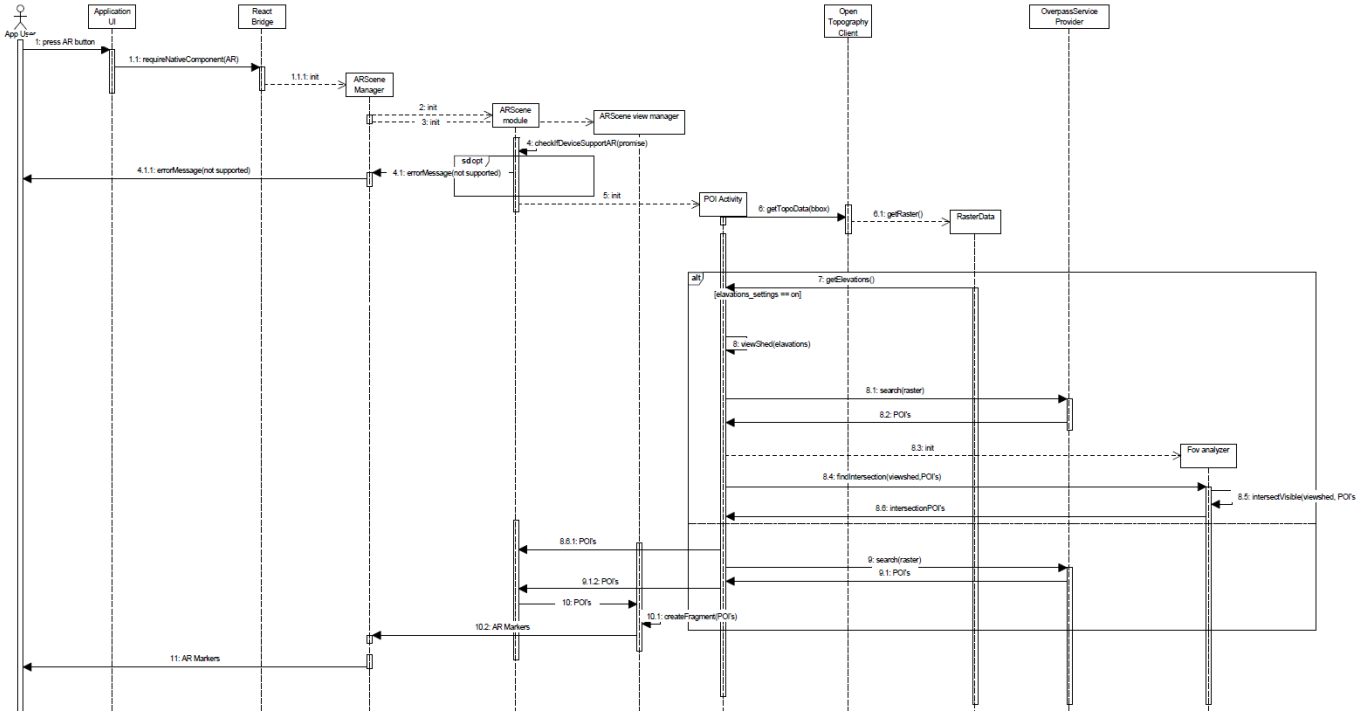
The main transactions in our database are CRUD operations, We will mention just the actions that modifies the database:

- Add new Authorized user:
  the transaction parameters - new user details.
  Adding the new user document to the “authorized users” collection.
- Delete Authorized user:
  The transaction parameters - username.
  Removing the appropriate user from “authorized users” collection.
  Removing the appropriate triangulation records associate with the user from “triangulation records” collection.
- Add caching data:
  the transaction parameters – username, CachingData object attributes.
  Adding the new document to the “caching data” collection.
- Delete caching data:
  the transaction parameters – username.
  Removing the appropriate user’s caching data from the “caching data” collection.
- Add triangulation record:
  the transaction parameters – username, record details.
  Adding the new record document to the “triangulation records” collection with generated id.
- Delete triangulation record:
  the transaction parameters – recordID.
  Removing the appropriate record document from the “triangulation records” collection.

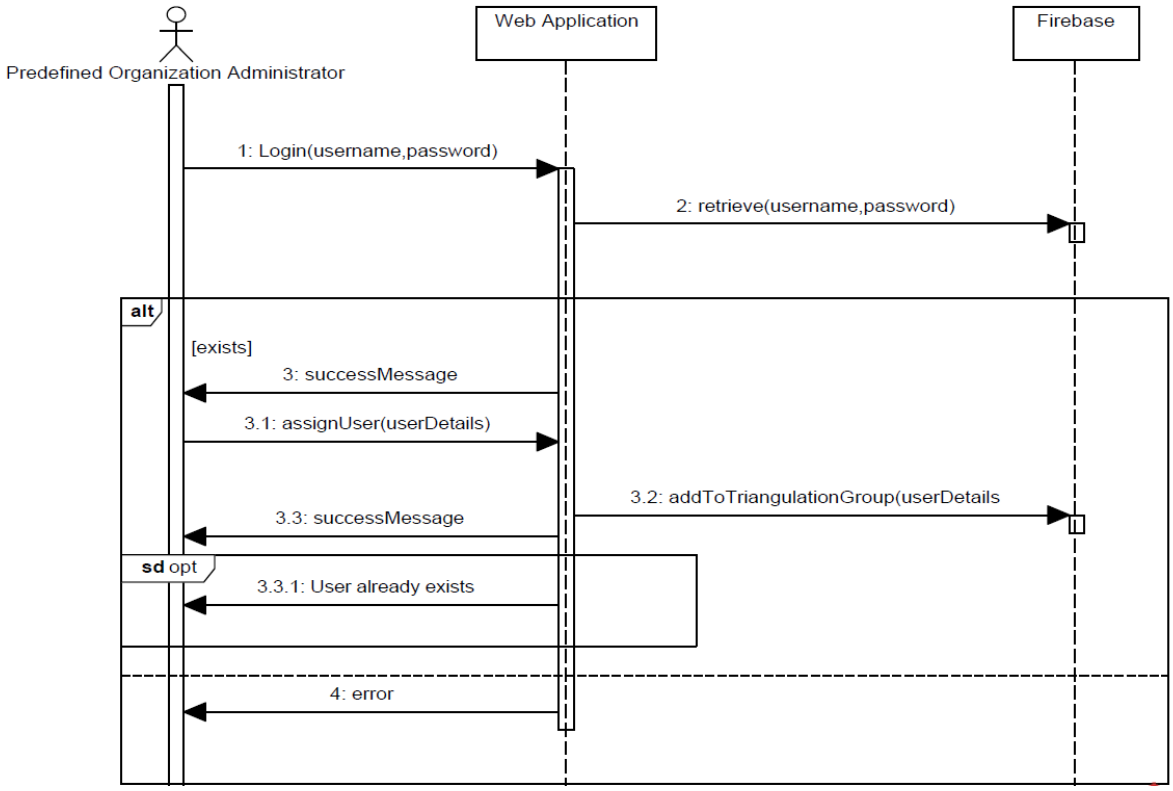
# Chapter 4
# Behavioral Analysis

## 4.1. Sequence Diagrams

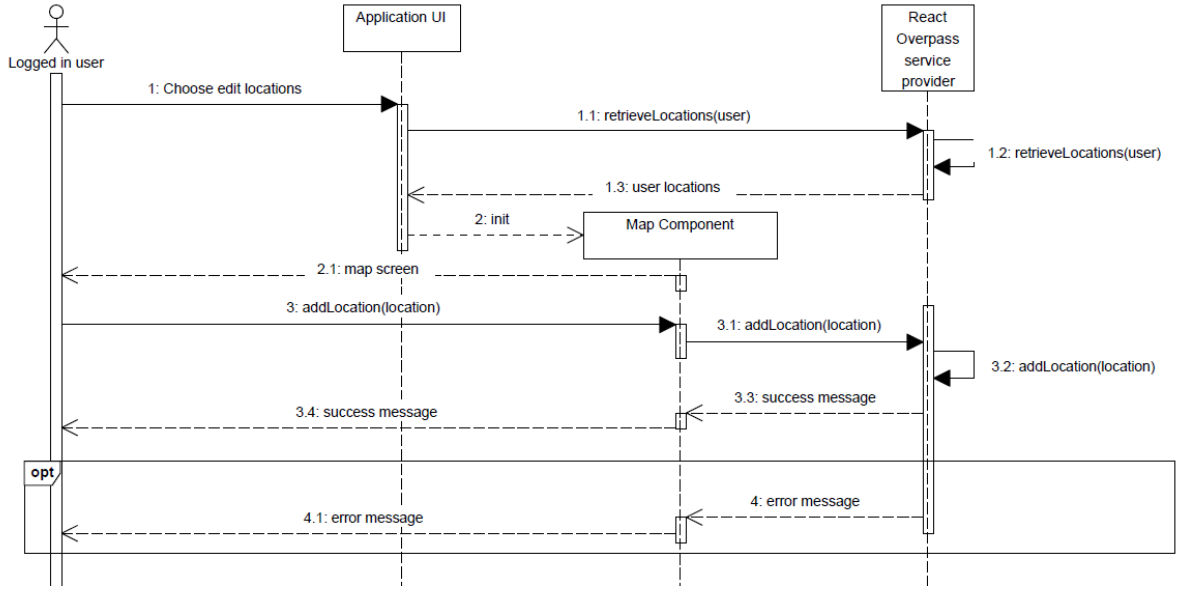
### 4.1.1. Viewing nearby places using AR



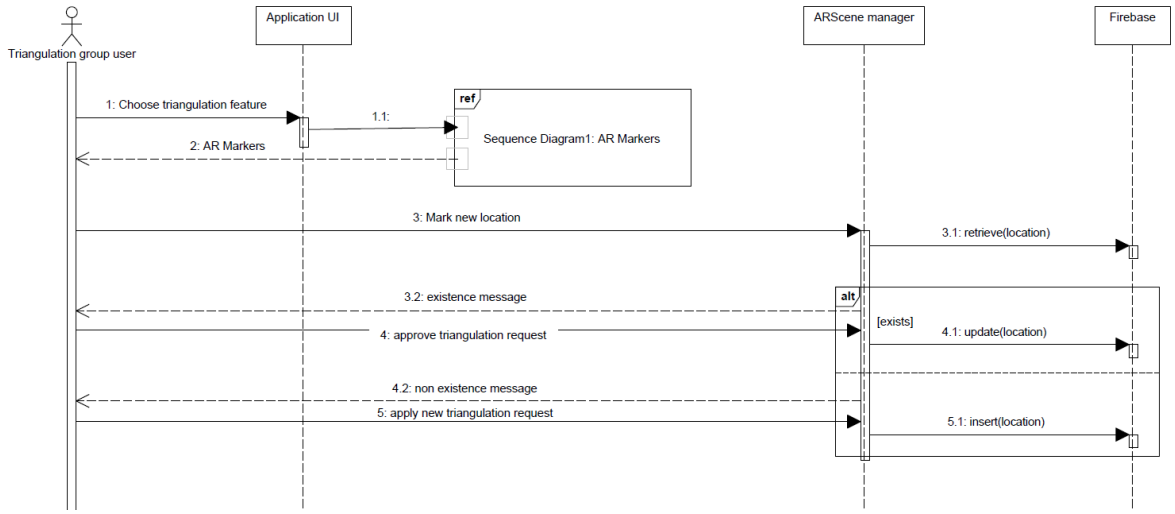

### 4.1.2. Assigning a user to the triangulation group



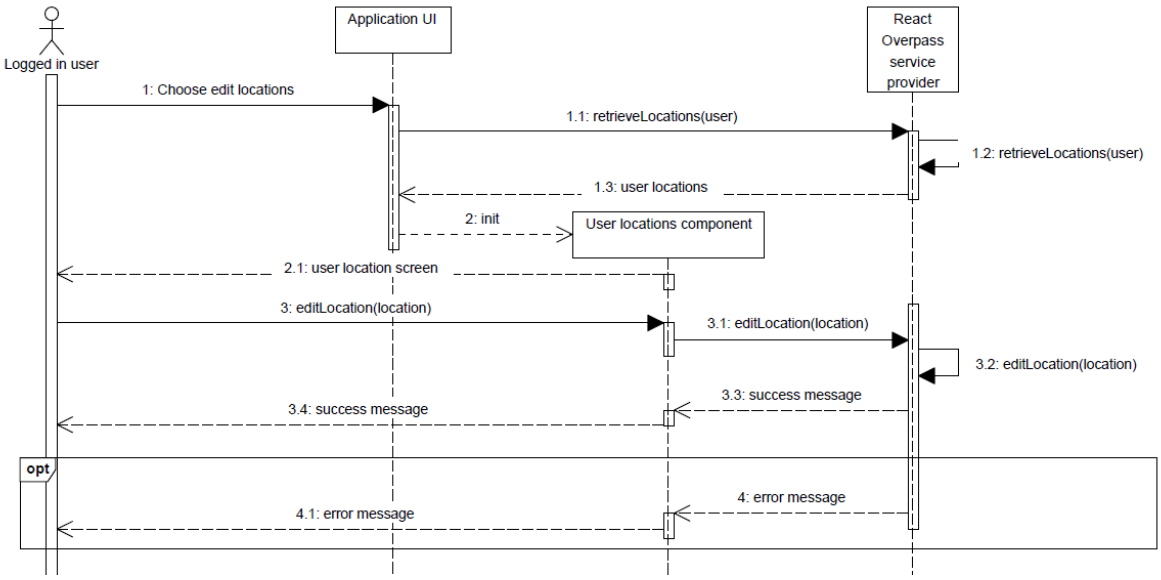
### 4.1.3. Adding locations to the external map provider - using map



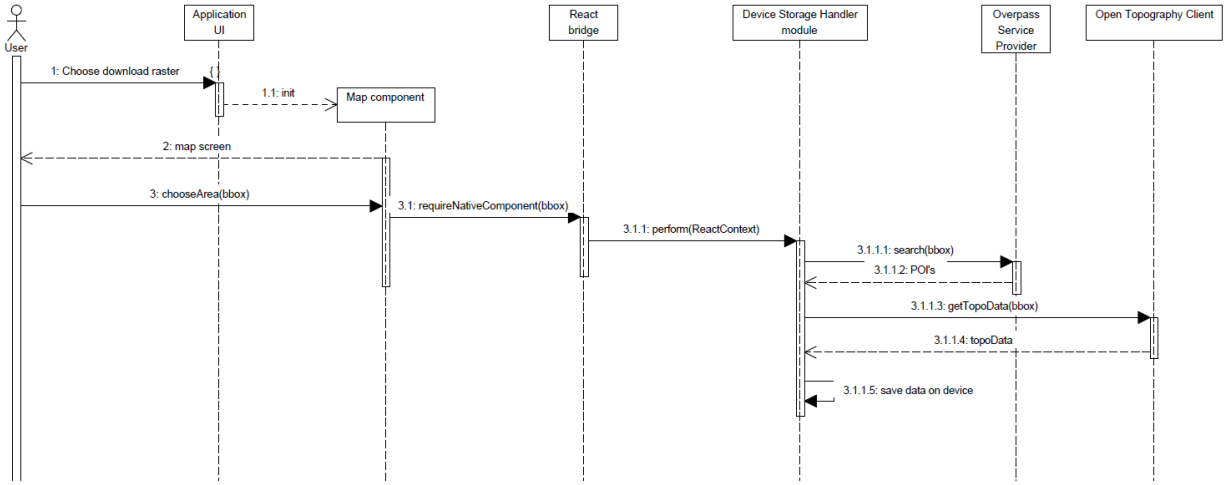

### 4.1.4. Adding locations to the external map provider - using triangulation



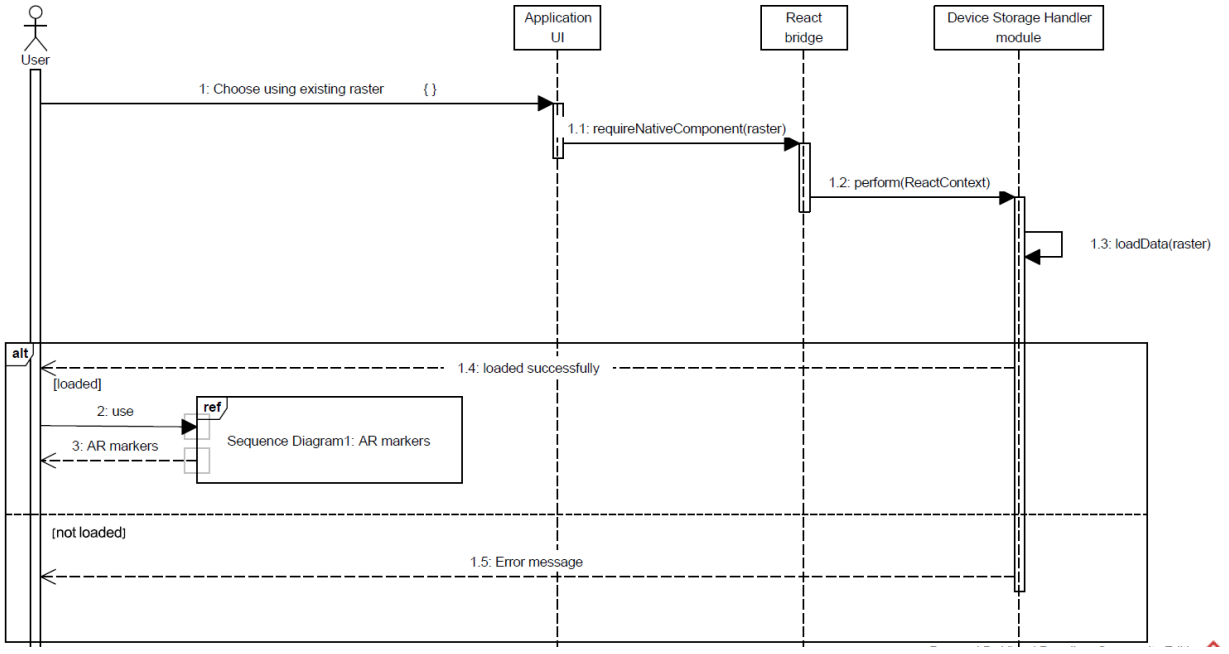
### 4.1.5. Editing previously added locations



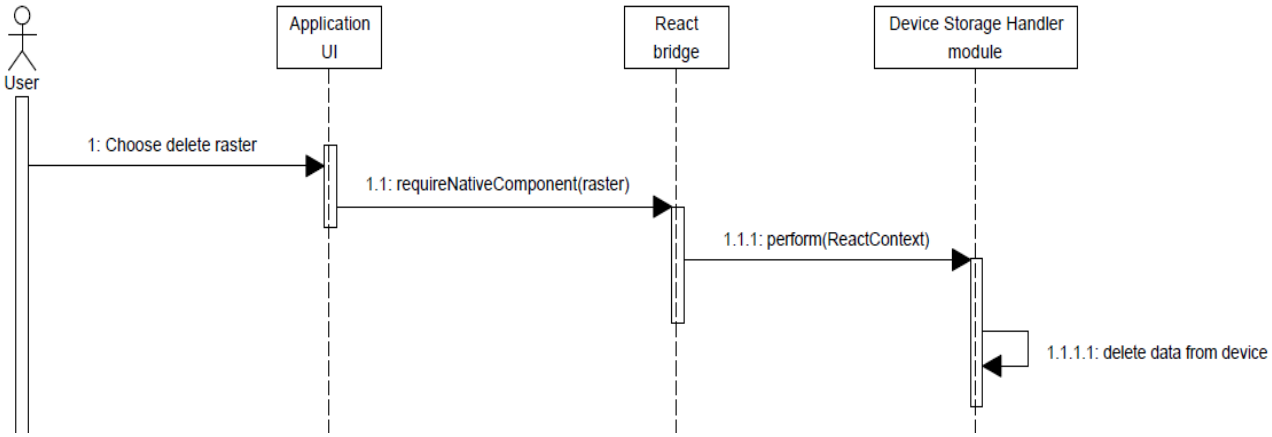

### 4.1.6. Preservation of data about chosen location to be used on offline mode




### 4.1.7. Using preservation data about chosen location on offline mode



### 4.1.8. delete preservation data about chosen location




## 4.2. Events

### 4.2.1. The application losing GPS signal

When losing GPS signal, the application should write appropriate message to the user and provide a way to calibrate the device compass in order to improve the device location accuracy.in addition, since WIFI spots can improve the GPS signal, the application will suggest the user to search for nearby WIFI signals.

### 4.2.2. The application losing internet connectivity

When losing internet connectivity, the application should write appropriate message to the user and suggest the user to improve his location.

### 4.2.3. System failure

Upon receiving failure signal, the application should write appropriate message to the user. The application will cache data from latest requests or operation to be recovered when the user will load the application after the failure.

### 4.2.4. The user pointing the camera over a wall in a close place

When the user using the application in a close place, the application should write appropriate message to the user. In addition, application should not display to the user the nearby places and should not perform unnecessary HTTP requests to retrieve topographic data and data about nearby places.

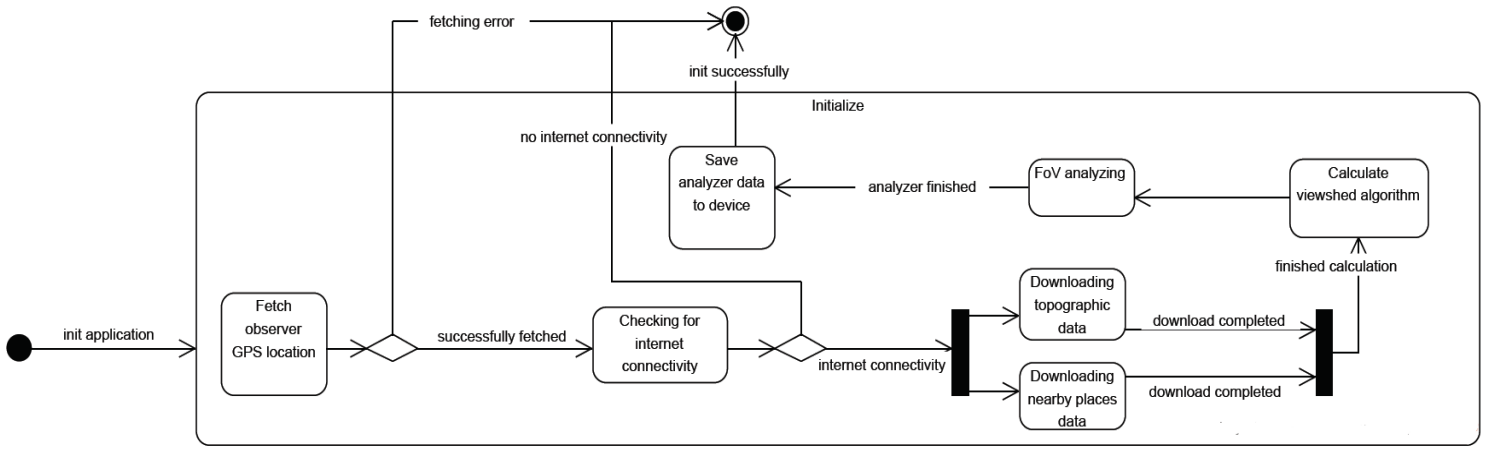
### 4.2.5. The user is in movement during the use of the application

Upon movement of the user during the use of the application, the application should track the GPS signal. After defined distance, the application should retrieve new topographic data and data about the nearby places. After receiving the new data by using HTTP requests to the external resources, the application should recalculate the users FoV. After recalculating the users FoV, the application should display to the user new AR markers for each place in his new FoV.

## 4.3. States

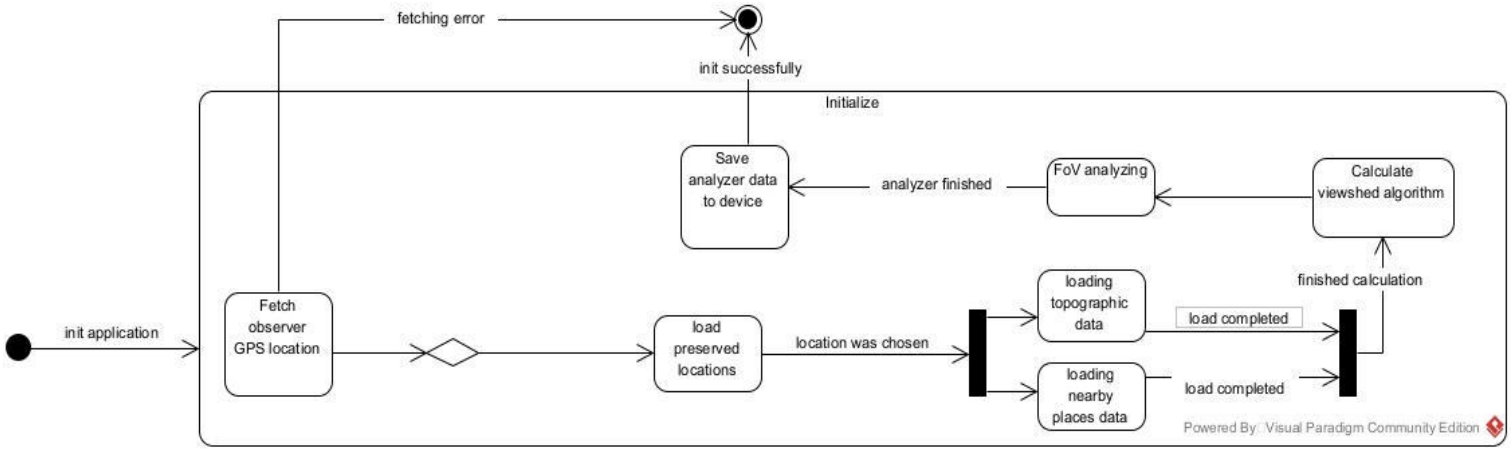
### 4.3.1. Application initialization state

In order to avoid HTTP requests response time while using the application, the application will perform the HTTP requests during the initialization. in addition, the viewshed calculation will be in the initialization too. During the initialization, the application should check for GPS signal and internet connectivity. The initialization process will be as follows:



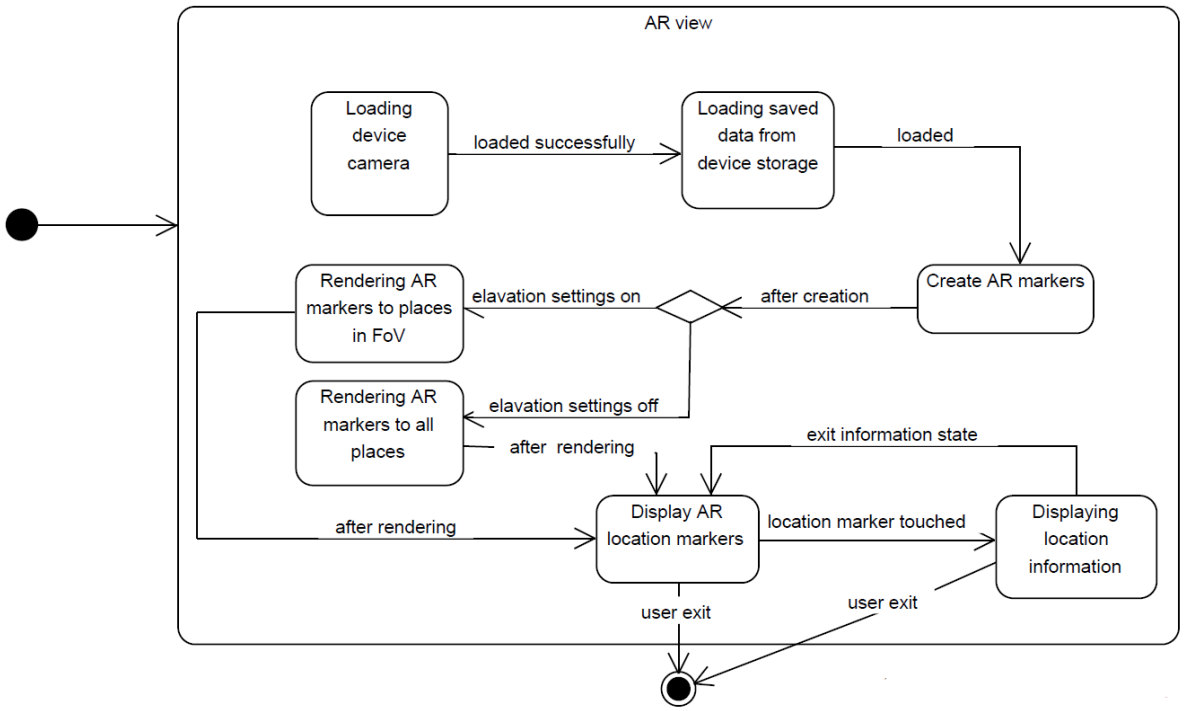

#### 4.3.1.1 Application initialization state – offline mode

while using the application on offline mode, the application uses preserved data downloaded to the user device. in addition, the viewshed calculation will be in the initialization too. During the initialization, the application should check for GPS signal and preserved location data. The initialization process will be as follows:



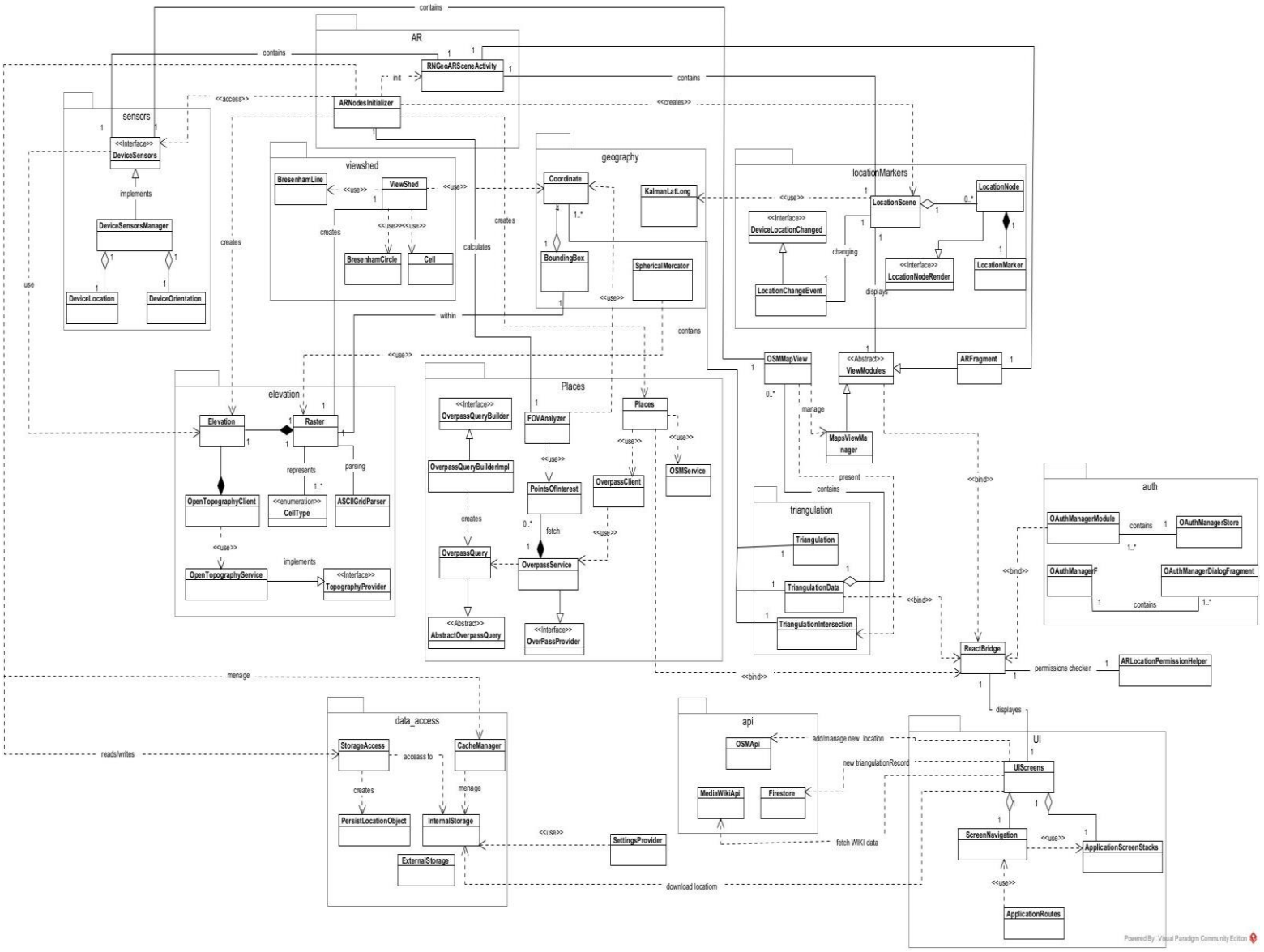
### 4.3.2. AR view state

When the user chooses to view all the point of interest in AR mode, the application will load the saved data from the mobile device. The saved data could be the data that stored during the initialization process or the raster data that the user downloaded while he choose to use offline mode(before choosing AR mode, the user should choose if he want to use the raster data that he already downloaded). After loading the required data, the application will create the AR location markers and render the AR markers on the camera frame. The AR view process will be as follows:



# Chapter 5
# Object-Oriented Analysis

## 5.1. Class Diagram



## 5.2. Class Description

### 5.2.1. Viewshed class:

- This class is responsible to Implementing the Viewshed algorithm: Identify the areas in the user's field of view, by analyzing the elevations data of the user's surroundings and applying the viewshed algorithm.
- Main methods in this class:
    - calculateViewshed:
      Pre-condition:
      1. ASCII Grid elevation data parsed into Raster Data type.

      Post-condition:
      1. Identify the areas in the user's field of view.

      Invariants:
      1. Context Raster: inv: Raster.elevations != NULL
- Method calculateViewshed should consider a distance price. A POI can be very far from the observer location, so that, if the algorithm considers only the elevations, this POI can be classified as visible to the observer. Therefore, the algorithm will consider elevations and distance.

### 5.2.2. FOVAnalyzer class:

- The main algorithm in this class is responsible to intersect between the output of the viewshed algorithm and the observer surrounding point of interests. the output of the algorithm will be all the observer surrounding point of interests that are in his field of view. This is the main process in the application before the location markers will appear in the user's mobile screen.
- Main methods in this class:
    - intersectVisiblePlaces:
      Pre-condition:
      1. Viewshed algorithm already computed
      2. Query for the nearby points of interest was executed and returned the result successfully. (points_of_interest !=NULL)

      Post-condition:
      1. visiblelocations !=NULL

      Invariant :
      1. Context visiblelocations: inv : !pointsOfInterest.isEmpty() and viewshed.intersect(any(pointsOfInterest)) **implies** !visiblelocations.isEmpty()

5.2.3. ARNodesInitializer class:

• This class is responsible to initialize the application AR location markers, the location scene is created in this class and all the location markers added to the location scene. It also sends requests to the external resource's providers (places, elevations), receives their response and delegate the response for the following calculation. This class handle and execute asynchronously all the calculations that required to create the appropriate location markers (viewshed, FoV analyzing, fetch and parse elevation data and fetch all the places data).

- Main methods in this class:
  - initializeLocationMarkers:
    Pre-condition:
    1. Device sensors initialize
    2. AR scene initialize
    3. Locationscene.locationMarkers.isEmpty() = True OR Locationscene.cache.isEmpty() = False

    Post-condition:
    1. Visible places locations markers added to the location scene.

    Invariant:
    1. Context Locationscene: inv : isExist(visibleLocation) **implies** Locationscene.locationMarkers.isEmpty() = False

5.2.4 Triangulation class:

- This class responsible for all the triangulation calculations, including the intersections calculation of two triangulation records. This class's calculations takes into account the spheres of the earth and performs calculations accordingly.
- Main function in this class:
  - triangulate:
    Creates new intersection coordinate if exist, else returns null.

5.2.5 OSMMapView class:

• This class is responsible for the map view. the class manages the indications shown on the map for example the device orientation, triangulations overlays, the area bounding circle and the observer location sign.

- Main methods in this class:
  - setShowTriangulationData:
    If the passed argument is true, the application's map view will present all the relevant triangulation overlays lines, The user's view direction line and indicates intersections.
    Pre-condition:
    1. Device sensors initialized.
    2. MapView initialized,
    3. MapView.mapController initialized,

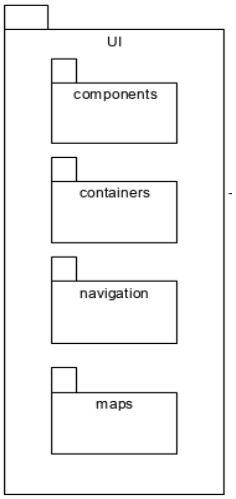
    Post-condition:

    1. map.getOverlays contains lineOfSight
    2. Intersection != Null implies map.overlays contains triangulationLines && map.overlays contains triangulationPoints && map.overlays contains triangulationViewrs

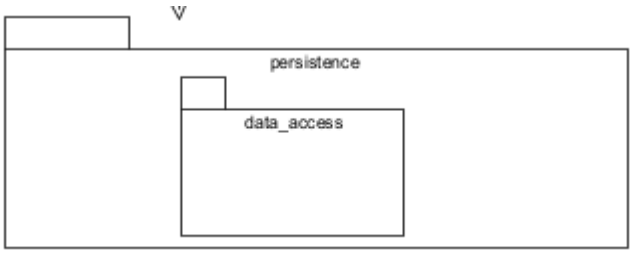
## 5.3. Packages

The packages used to organize the project better and help us to quickly locate classes which improve the efficiency. The application will include many classes, which are best organize into a hierarchy of packages :

*UI package*
Includes all UI components, containers, navigation bars, maps and everything that the user interacts with in the application.

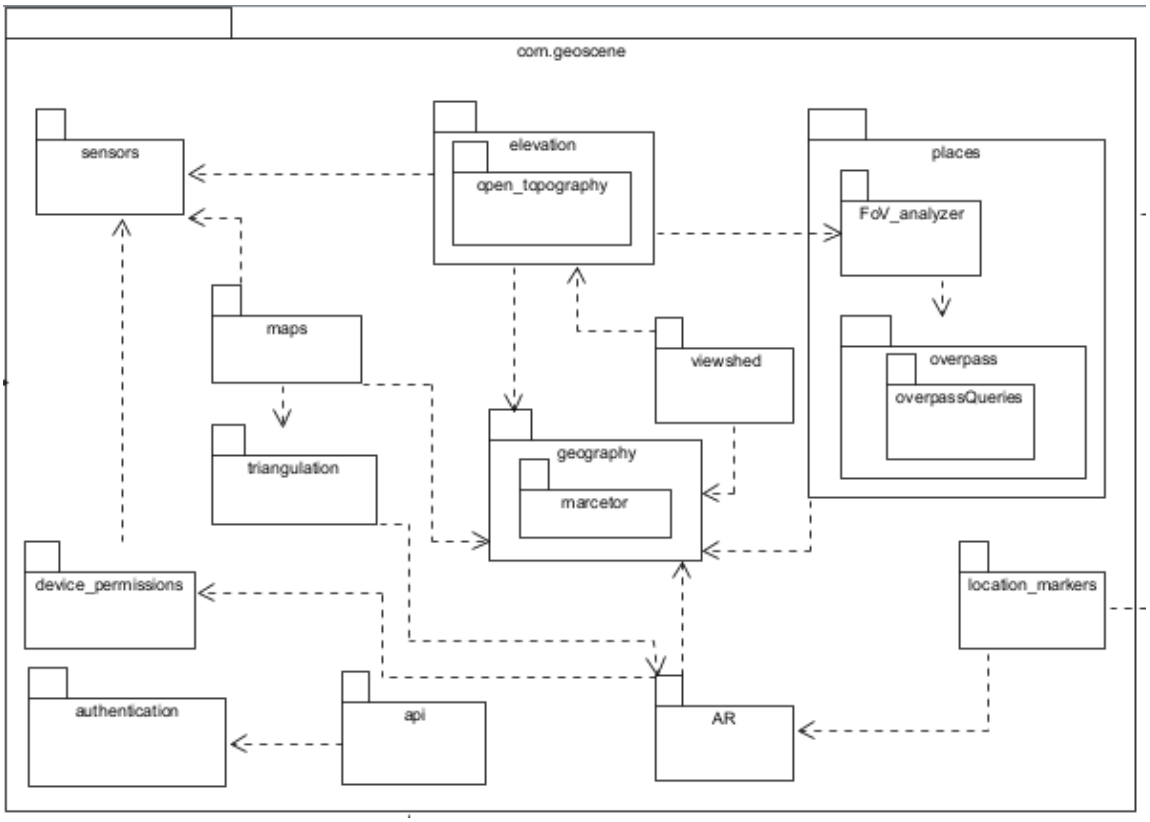



*external_data package*
This package includes all the classes that should interacts with Firebase DB of the application, the mobile device storage and the cache. All of the requests to save and load data to\from a storage will be handled in this package.

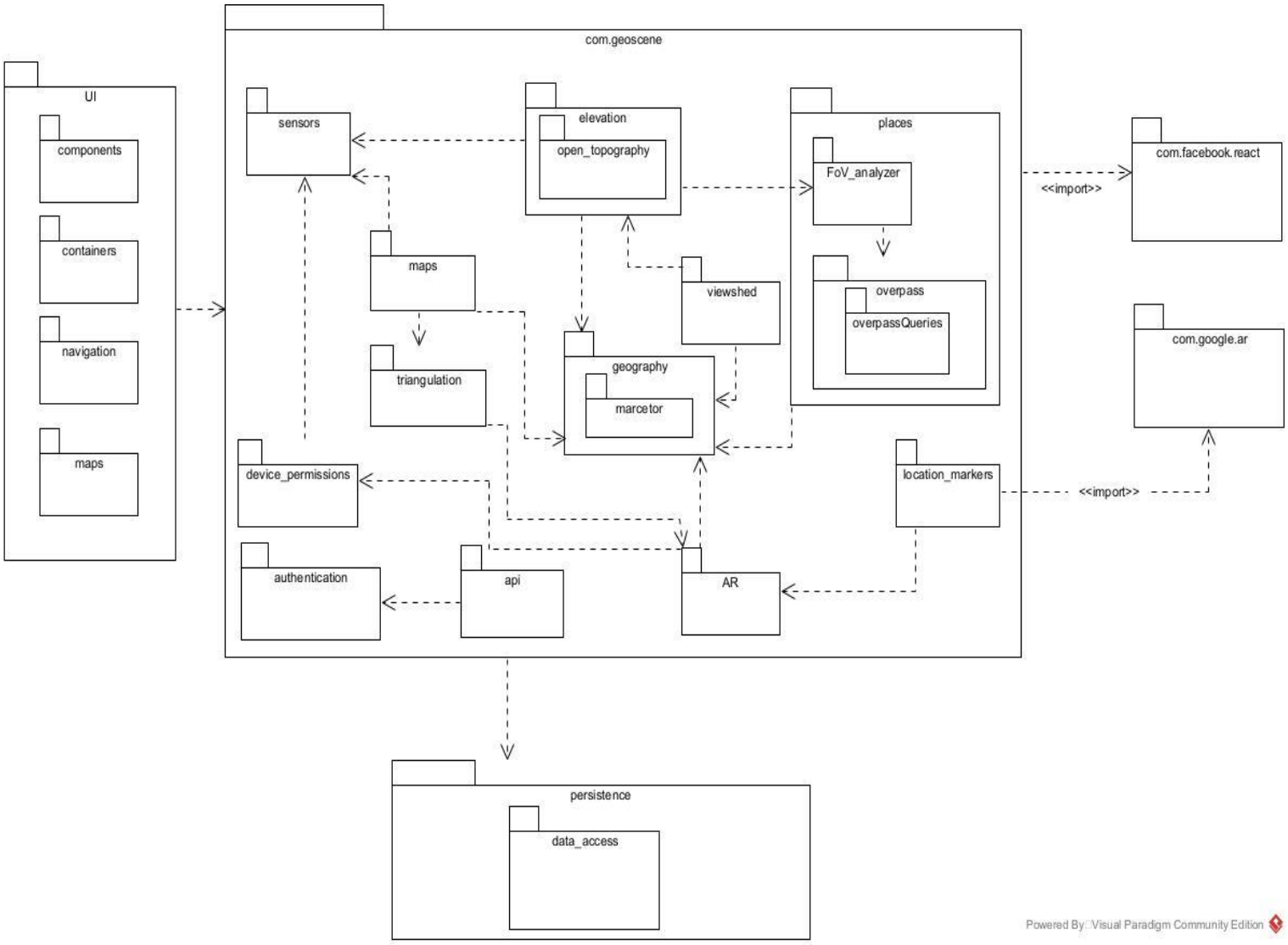


*com.geoscene package*
This package includes many sub packages:

- Sensors – includes all the classes that interacts with the mobile device sensors (location sensor, orientation sensor etc.)
- Geography - includes topographic and geometry classes and calculations, also includes the sub-package:
    - Marcetor - includes all the geographical calculations (distance between coordinates, Mercator projection and bounding box calculations)
- Permissions – includes the classes that checks for the mobile device permissions that the user needs in order to use the application.
- AR – includes all the classes that are responsible to interact with Google's ARCore and put the AR objects on the mobile device camera.
- Location_markers – includes all the classes that are responsible to create the AR location markers .
- Viewshed – includes all the classes that take part of the viewshed calculation.
- Maps – includes all the classes relates to the maps view in the application and its functionality.
- Api – includes all the connections and requests to the API resources (OSM, WIKI, Firestore, etc)
- Triangulation – includes all the triangulation classes that take part of the triangulation calculations.
- Elevation – includes all the elevation raster calculation classes, also include the sub-package:
    - Open_topo - including all the classes that provides interface between the application and the Open Topography API.
- Places – include the sub packages:
    - FOV_analyzer – includes all the classes that are responsible of calculating the field of view analyzing after the viewshed calculations.
    - Overpass – including all the classes that provides interface between the application and the OpenStreetMap overpass API. This package include overpass queries package which include all the classes that are used to build the queries to overpass API



All the packages have relationships between each other, and some packages use other packages.
The package diagram for the application :



## 5.4 Unit Tests

| *# Req.* | *# Test* | *Component* | *Function* | *Input* | *Expected Output* | *Description* | *PASSED* |
|---|---|---|---|---|---|---|---|
| Fetching and analyzing data | | | | | | | |
| 1 | 1.1. | DeviceLocation (Android) | getDeviceLocation | - | Device location coordinates and altitude | Test that device GPS sensor values are returned successfully and that the location information that was returned matches the actual device location and altitude. | |
| 1 | 1.2. | DeviceOrientation (Android) | getOrientation | - | Device orientation in degrees from the azimuth (clockwise) | Tests that the device orientation (bearing) returns the correct azimuth in degrees (0-360) and that the device tilt does not affect the calculated orientation. | |
| 1 | 1.3. | DeviceOrientation (Android) | getDeviceOrientation | - | Whether the device is currently in Landscape orientation or Portrait orientation. | Tests that the correct device orientation is returned (Landscape/Portrait). | |
| 2 | 2.1 | FOVAnalyzer (Android) | intersectVisiblePlaces | Elevation Raster including the calculated viewshed, nearby POIs information, The visible places setting is active. | The nearby POIs that intersect with the calculated viewshed. | Tests that the method returns the correct intersection of the calculated viewshed and the bounding boxes of the nearby POIs fetched from the external places provider. Tests that if at least one raster tile is visible according to the viewshed that intersects on the POIs bounding box, it is returned. | |
| 2 | 2.2 | FOVAnalyzer (Android) | intersectVisiblePlaces | nearby POIs information, The visible places setting is inactive. | The nearby POIs unfiltered. | Tests that the method returns all the POIs fetched from the external places provider without filtering them by elevations. | |
| 2.1 | 2.1.1 | BoundingBox (Android) | getBoundingBox | Center coordinate of the bounding box, half side of the bounding box in KM. | Pair of two coordinates representing the bounding box (south, west) and (north, east) | Tests the WGS84 bounding box retuned center is the given center coordinates. Tests that the size of the bounding box is represented by the two output coordinates equals to 2 * half side input value. | |
| 2.1 | 2.1.2 | ViewShed (Android) | calculateViewshed | Elevation raster, observer coordinates, observer GPS altitude. | Two-dimensional array representing the visible raster tile according to the viewshed algorithm. | Tests that the method returns the visible raster tiles according to the observer coordinates the user defined bounding box size and the surrounding elevation tiles and the calculated slopes. Tests that places that are visible according to the viewshed | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | | algorithm but are outside of the pre-defined reduction range are filtered according to the constant distance price provided. | |
| 2.1 | 2.1.3 | BresenhamLine (Android) | calculateBresenhamLine | 2 points that the line should be drawn between. | List of elevation raster tiles that the calculated rasterized line crosses. | Tests that the methods return the correct cells that compose the rasterized line between the two input points. | |
| 2.1 | 2.1.4 | BresenhamCircle (Android) | calculateBresenhamCircle | Observer coordinates, raster dimensions (array size), circle radius. | List of elevation raster tiles that compose the perimeter of the rasterized circle drawn with center as the observer coordinated and input radius. | Tests that the cells returned form a closed circle. Tests that the center of the returned cells that compose the circle perimeter is the observer coordinates. Tests that the returned circle raster cells correspond to the input radius. | |
| 2.2 | 2.2.1 | SettingsScreen (React-Native) | setBoundingRadius | UI Slider numeric value as radius in KM | The current active circular radius in which the elevation tiles and places will be retrieved. | Tests that the parameter that define the circular radius in KM has been correctly set to the current numeric value (using the android Native Module). | |
| 3.1 | 3.1.1 | FOVAnalyzer (Android) | setMockElevation | UI Slider numeric value as elevation added to the real elevation in meters. | The current user elevation values including the mock elevation set, the elevation according to the elevation raster tile and the device GPS altitude difference. | Tests that the current user elevation value is changed (added) according to the numeric mock elevation values the user has set. | |
| 3.2 | 3.2.1 | Elevation (Android) | getObserverElevation | Sensor location altitude, observer raster cell elevation | The current value of the observer elevation including the raster tile elevation the observer is at and the difference from that elevation value and the device GPS measured altitude. | Tests that observer elevation value in meters is set according to the elevation raster tile and the device altitude measured by the GPS. Tests that the elevation is set only by the raster tile elevation if the GPS does not currently hold a valid altitude value or the GPS in the device does not support altitude values. | |
| 4 | 4.1 | StorageManager (Android) | preserveBoundingBox | Center point coordinates (latitude longitude), radius from the center in KM determining the bounding box, data given name, data given description | Preserved data key generated if the entire data in the chosen bounding box was stored in the device storage successfully and null otherwise. | Tests that by giving a center point and a radius, the necessary data is fetched and stored for later use in the device. Tests that by giving an invalid data (negative radius, invalid center coordinate, empty name, etc.) the data is not fetched and stored to the device. | |
| 4 | 4.2 | StorageManager (Android) | storeElevationData | Elevation raster, | True if the entire data in | Tests that the elevation data is | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | preserved data key | the given raster was stored in the device storage successfully and False otherwise. | stored in the device correctly under the given preserved data key. | |
| 4 | 4.3 | StorageManager (Android) | storePlacesData | POIs information, preserved data key | True if the entire POI data was stored in the device storage successfully and False otherwise. | Tests that the POI data is stored in the device correctly under the given preserved data key. | |
| 4 | 4.4 | StorageManager (Android) | retrievePreservedData | Preserved data key | Pre-stored elevation and POI data if the input key is valid, otherwise throws error. | Tests that the entire elevation and POI data is retrieved correctly by giving a valid preserved data key. Tests that an error is thrown by giving an invalid key. | |
| 5 | 5.1 | CacheHandler | checkCacheAndRetrieve | Current observer location, previous observer location, distance tolerance, has settings changed | Elevation and POI data if the input parameter passes the necessary checks, otherwise null. | Tests that data is retrieved from cache if the distance between the current observer location and the previous observer location is under the input distance tolerance else if the setting has changed or the distance is above the tolerance check that null is returned. | |
| Displaying information | | | | | | | |
| 6/6.1 | 6.1 | AsyncARLocationsInitializer | generateLocationMarkers | Visible POI list | Location marker object list | Test that for each visible POI given as input, the output contains a Location marker object containing the appropriate POI information. | |
| 6.2 | 6.2.1 | LocationScene | checkLocationMarkersOverlap | 2 different visible Location Nodes | True if the given Location Nodes overlap (UI representation intersects) and False otherwise. | Tests that returns True for overlapped places given two different location nodes that overlap (are in the same general direction) and returns False for 2 places that does not interact. | |
| 6.2 | 6.2.2 | LocationScene | offsetOverlapping | 2 different visible Location Nodes | New height values for the given input locations. | Tests that by giving two locations that their UI representation overlaps, a new height value for the UI representation is returned and that the Location Nodes does not overlap given the new height. | |
| 9 | 9.1 | LocationNode | scaleAndRotate | 2 LocationNodes with distance values $d_1, d_2$ | Scale values for $d_1 and\ d_2$ | Tests that given $d_1 < d_2$ then the scale value returned for $d_1$ Is less than the scale value for $d_2$ (and that if equal the same scale value is returned). | |
| 10 | 10.1 | MapFragment | getDisplayedBoundingBox | Observer coordinates | Map displayed bounding box | Tests that the observer coordinates are places in the center of the returned bounding | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | | box which represents the currently visible map portion. | |
| 11 | 11.1 | MapFragment | getMapOrientation | Device bearing | True if the difference between the map orientation (bearing) and the device orientation is less than a constant number of degrees, False otherwise. | Tests that given the current device azimuth (bearing) in degrees, the test returns true if difference between the device orientation and the map orientation is less than a pre-defined constant value. | |
| Adding/Editing features | | | | | | | |
| 12 | 12.1 | AuthenticationManager | externalLogin | User details object | True if currently assigned user details equal to the given input details. | Tests that by using a Stub on the external map provider authentication system, the currently assigned user for the application is assigned correctly. | |
| 12.1/12.2 | 12.1.1 | UserLocationsManager | addLocation | Location details (name, description and coordinates) | The list of locations fetched from the external provider stub. | Tests that the current active user is assigned. Tests that after inserting a new location using a Stub as the external map provider, the returned list of location after fetching includes the new location added. | |
| 12.3 | 12.3.1 | Triangulation | addLocationTriangulation | Device bearing in degrees, observer coordinates, name, description | Triangulation record added fetched from application DB. | Tests that the current active user is assigned and contains permission to perform triangulation. Tests that the added triangulation records is present in the application DB. | |
| 12.3 | 12.3.2 | Triangulation | fetchNearbyTriangulationRecords | Observer coordinates | List of Triangulation records. | Tests that all triangulation records that the observer that the coordinated they were tagged from are in range of the observer pre-defined visible radius. Tests that If no triangulation records have been tagged in the observers defined range an empty list return. | |
| 12.3 | 12.3.3 | Triangulation | performTriangulation | 2 Coordinates and 2 bearing degree values | True if the generated triangulated coordinate is in distance that is less than a pre-defined threshold from the real location coordinate. | Tests that the algorithm that perform the triangulation calculation returns a coordinate that is in an acceptable distance from a real mapped location (under a pre-defined application threshold). | |
| 12.3 | 12.3.4 | Triangulation | approveTriangulationRecord | Triangulation record id | True if the coordinate approved is present in the external map provider stub, False otherwise. | Tests that the triangulation record identified by the triangulation record id is not present in the application DB. Tests by using an external map provider stub that the | |

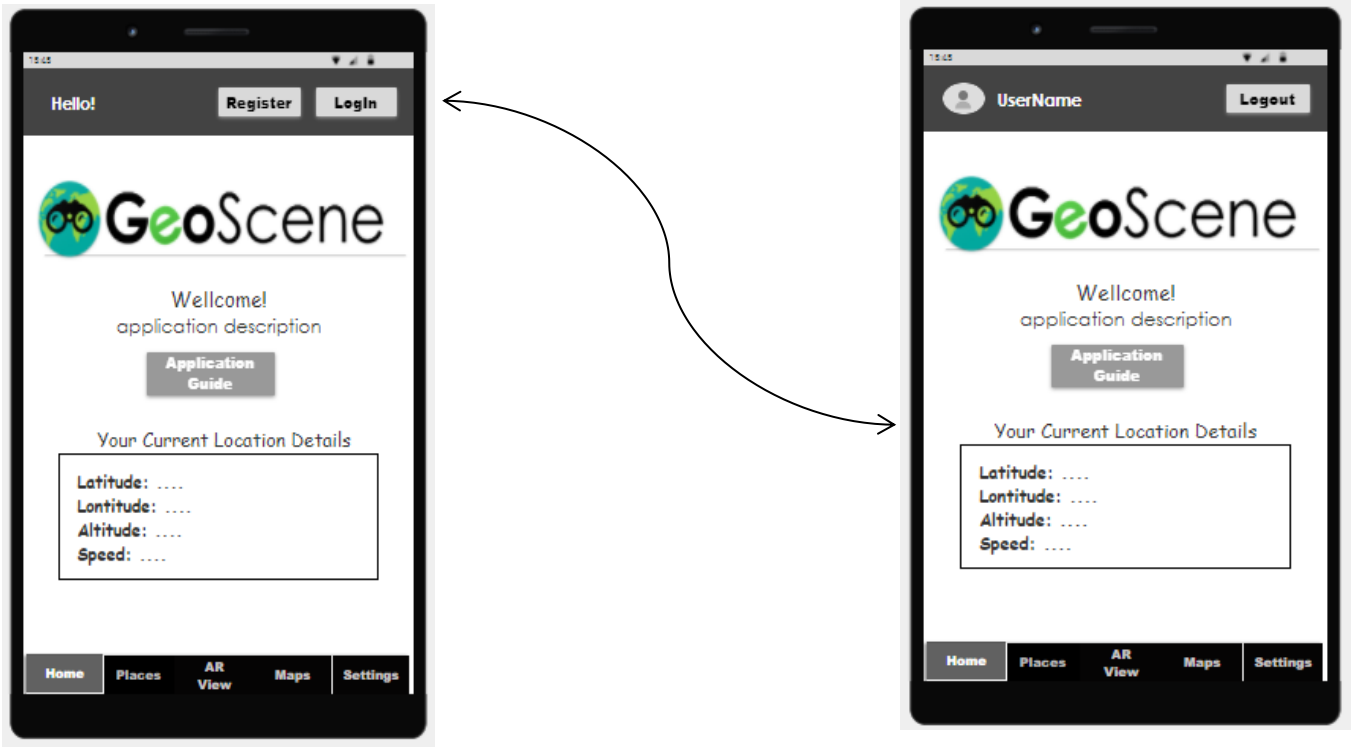
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | | new location has been added according to the triangulated coordinate generated by the triangulation algorithm. | |
| 12.4 | 12.4.1 | UserLocationsManager | editLocation | Changeset id that contains the location and the new location information values. | True if the location fetched from the external map provider stub contains the details given as input. | Tests that the new information given as the new location information for a location added in the OSM input changeset id contains the new information by returning it from the map provider stub after the change. | |
| 12.5 | 12.5.1 | UserLocationsManager | removeLocation | Changeset id that contains the location | True if the location represented by the changeset id has been removed from the external map provider stub. | Tests that the location represented by the changeset id has been removed from the externa map provider stub. | |

# Chapter 6
# User Interface Draft

The application contains several main screens which are designed to make navigation easier for the user and make the functionality of the application easily accessible.

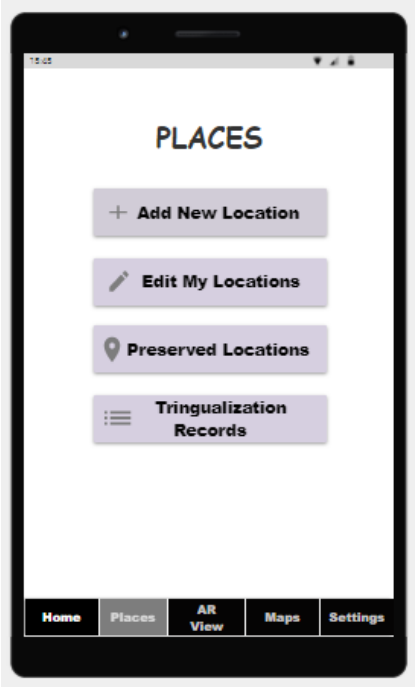
**The "Home" Page Draft:**

The application main window, Easy to navigate by clicking the "Home" option in the bottom menu.
The register and login buttons at the top will open the external map provider relevant pages. After the user logged in the top panel will represent the username and image. In this page the user can see his current location details and the application description.
This page contains button that opens interactive application guide screen.

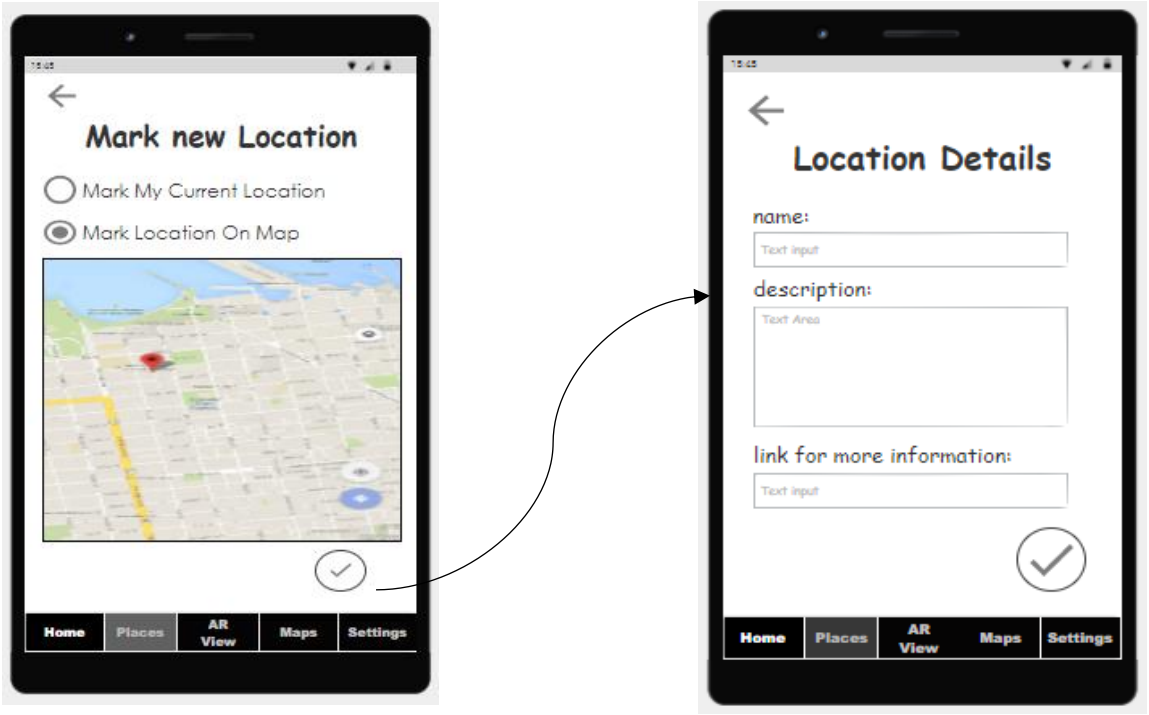



**The "Places" Page Draft:**

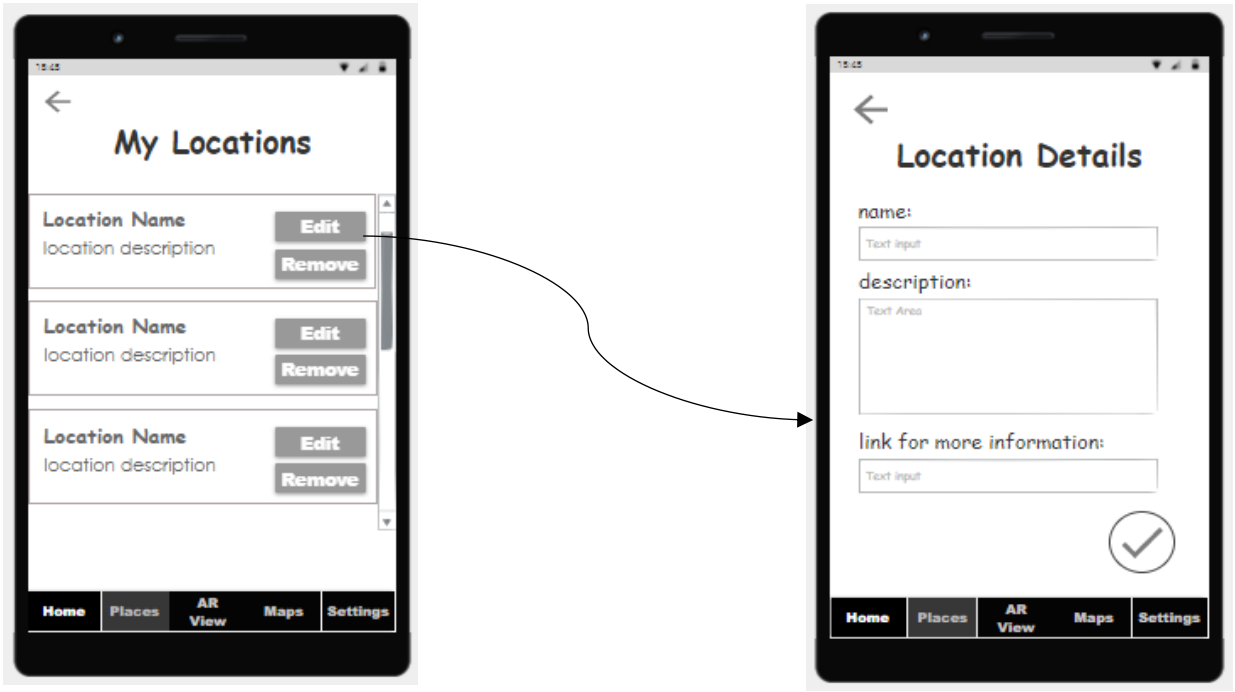
Easy to navigate by clicking the "Places" option in the bottom menu.
From this screen the user can navigate to 4 more screens.



**The "Add Location" Page Draft:**
The user navigates to this page by clicking the "add new location" button in "Places" page.
The functionality of adding new location to the External Map Provider is divided over two pages. On the first page, the user chooses whether the coordinate of the location he wants to add is his current coordinate or he wants to search for the point on a map.
After marking the location, the user pass to the other page, in this page the user should insert the location details – name, description. And link for more information website (Wikipedia, etc.). the output from this page will be success or error message.




**The "Edit My Locations" Page Draft:**
The user navigates to this page by clicking the "Edit My Locations"" button in "Places" page.
In this page the user can view all the locations that was added by him. The user can edit location details by clicking the "edit" button, the page "location details will be open with the previous data in the relevant fields, the user can edit the data, the output from this page will be success or error message.
By clicking the "remove" button the user can remove the location, the output from this page is success or error message.

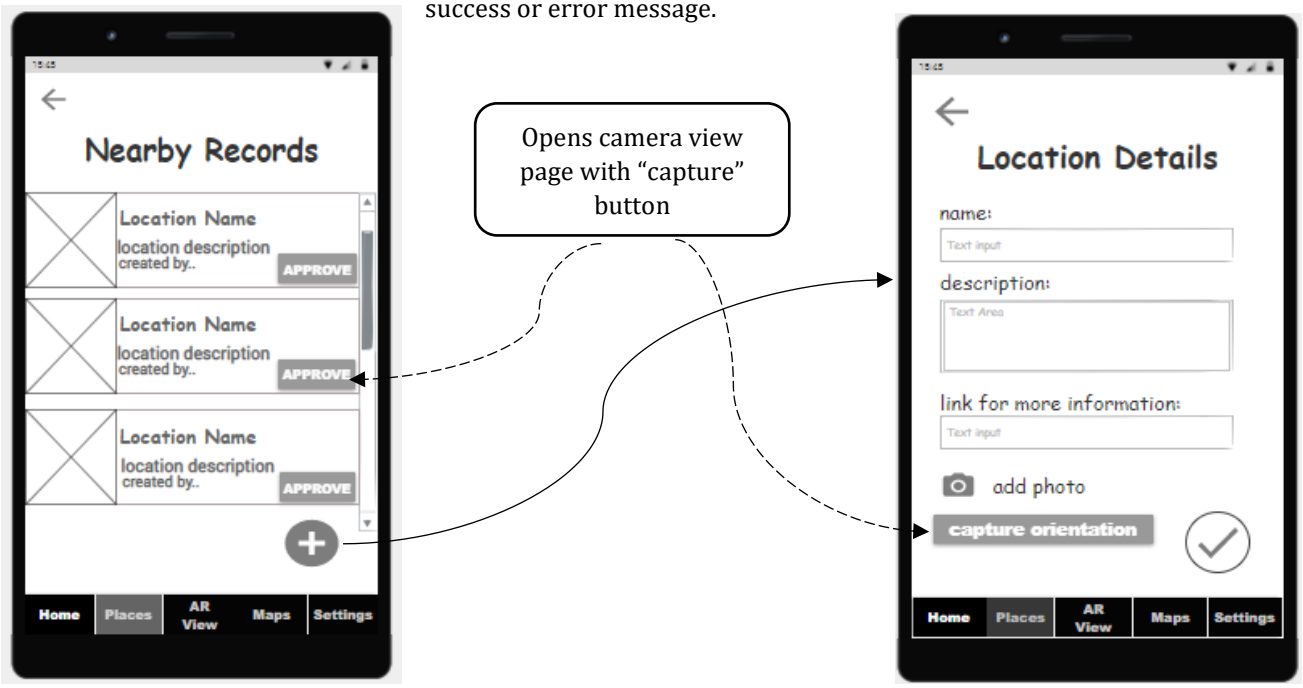


**The "Preserved Locations" Page Draft:**

The user navigates to this page by clicking the "preserved locations" button in "Places" page.
When user preserve location, the surrounding POIs data and the elevation data is download and store on the user device in order to be used in offline mode. In this page the user can view all his preserved locations, remove location, or use the stored data. For each preserved location, the user can see the location name, short description and image of the location on map.
In this page the user can preserve new location by clicking the "+" button. A new page will be open, the user should mark the requested location on map, specify radius to be explored around the marked location, insert the location name and description. The output from this page is success or error message.


**The "Triangulation Records" Page Draft:**

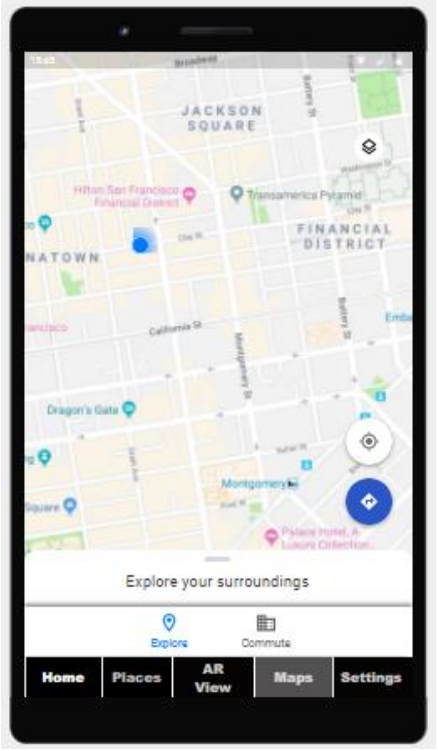
The user navigates to this page by clicking the "Triangulation Records"" button in "Places" page.
In this page the user can view all the nearby triangulation requests that are waiting for approvement. The user can see the name, description and image for each record, and the username created the request. In order to approve the request, the user clicks the "approve" button which opens camera view page with "capture" button, the user direct the camera to the relevant location and click "capture" button to capture orientation. The output from this page is success or error message.
In order to add new triangulation, record the user can click the "+" button, which opens new record page. In this page the user inserts the name of the requested location, description, link for mor information, and photo of the requested location. In order to capture the location orientation, the user click the "capture orientation" button, a new camera view page opens with "capture" button. The output from this page is success or error message.



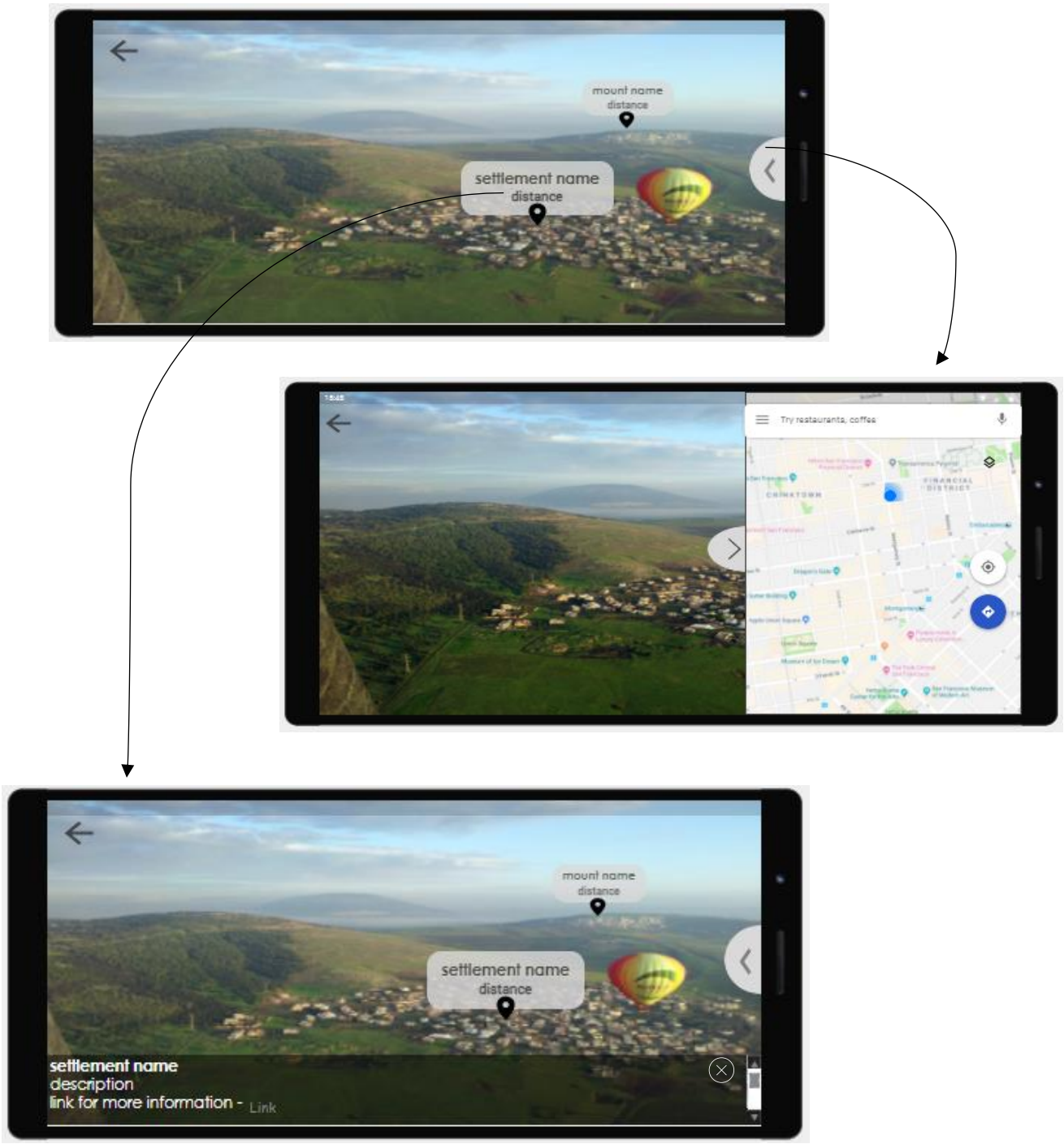
**The "Settings" Page Draft:**
Easy to navigate by clicking the "Settings" option in the bottom menu.
In this page the user can change the application settings:

- Turn on \ off the viewshed mode, which determines if the surrounding POIs will be filtered according to the user point of view.
- Specify the visible areas circular radius to display POIs in.
- Setting the ground relative height to increase their FoV depending on their non-topographic elevation or choose the auto option in order to determine the relative height using the device sensors.


**The "Maps" Page Draft:**
Easy to navigate by clicking the "Maps" option in the bottom menu.
In this page the user can view his location on map, as well as POIs around him in the specified radius.



**The “AR View” Page Draft:**
Easy to navigate by clicking the “AR View” option in the bottom menu.
This page presents the user with an augmented reality camera display with signs indicating points of interest in the selected radius environment. By swiping left of the arrow button"<" , the user will be able to see the POIs around him also on a side-by-side map.
Clicking on one of the signs opens a panel that contains extended information about the clicked POI.



# Chapter 7
# Testing

the application constructs from two main subsystems, while testing the application we test each part separately and the integration of the parts.

**Java Android part:**

in this part, we test our native modules (AR, maps integration and external resources API bridges). in order to do so, we use different kinds of testing levels:

- unit-testing
- integration testing
- API testing

In order to test the android side we will use JUnit framework.
Unit Test and Integration Tests, tests and modelized the communication between the application objects and received data from the different services.

the API tests validate the functionality, reliability, performance, and security of the external resource's interfaces.
Expected results:

- All unit tests pass
- All integration tests pass
- API tests pass - which means all requests pass successfully and the received information is validated.

**React Native part:**

The react native part of the application is responsible mainly for the UI side of the application and it serves as a medium between the user and the android native modules which are responsible for the core functionality of the application. Thus, we will mainly perform E2E test in order to validate this part. We will use the following frameworks in order to perform the tests of this part:

- To test the logical side of any computations perform we will use JEST (JavaScript Testing Framework)
- To perform the E2E tests we will use Detox and Mocha.

**End to End tests:**

our end to end testing simulates a user's experience and actions in our application. this part contains different tests that cover all the possible scenarios. in this part the main kinds of tests are:

- acceptance test
- sub-system integration tests

Expected Results:

- pass all acceptance test
- pass all sub-system integration tests
- pass automated tests

The acceptance tests and integration tests check all the scenario flows described in the use-cases section and validates application acts as excepted while simulates possible runs.

## Non-functional requirements testing methods

- Performance – like presented in the requirements document, we performed measurements that were measured for a high usage case including different data sizes, and for average real-world speeds for mobile 3G, 4G, and 5G technologies measured by cellular service providers. our test plan contains tests that validate and enforce the measurement results. the tests measures time when loading data received from external resources under different conditions, and check if the time is reasonable according to the requirements.
  In addition, we will check the performance under different tiers of devices in order to validate the required running times of analyzing and rendering as stated in the non-functional requirements specification.
- Reliability and Stability – in order to check the stability and the reliability of the application we will perform E2E tests to simulate different scenarios and expect that no crashes occur. In addition to test the cache DB functionality we will perform stress tests using Apache JMeter in order to simulate http requests and check the responses.
- Security – as part of our unit tests we will test the permissions validation and encryption of the different components that are involved in any user operation.
- Environmental and Portability – as predefined in the application android manifest, the requirements are met, therefore, there is no need to perform tests for this section.
- Availability - in order to use the application, the user must enable some device permissions (GPS, Camera), and have a network connection. our test plane contains tests that check the application handles appropriately in case of unable permissions or loss of network connection.
- Useability – usability testing will be performed in order to check the satisfaction of the application UX.