Politecnico di Milano

Development and Implementation of Mobile Applications

WIT

Design Document

Autori:

Jacopo Rigoli

Mattia Zannin

03/09/2015

Indice

1 Presentation of the Project

1.1 Scope

The goal of the WIT project is to create an application that can help tourists to retrieve information about touristic points of interest around them in the simplest way possible.

A user that wants to discover information about a place in front of him only needs to point the device to its direction and the app will automatically retrieve all relevant information.

The application also provide a page with some basic information about the city that the tourist is visiting, like the weather and the most searched monuments.

All the results retrieved by the application are stored in a diary that automatically recreates all the trips of the user.

1.1.1 System architecture

The application has been developed following the tiers paradigm:

- Client tier

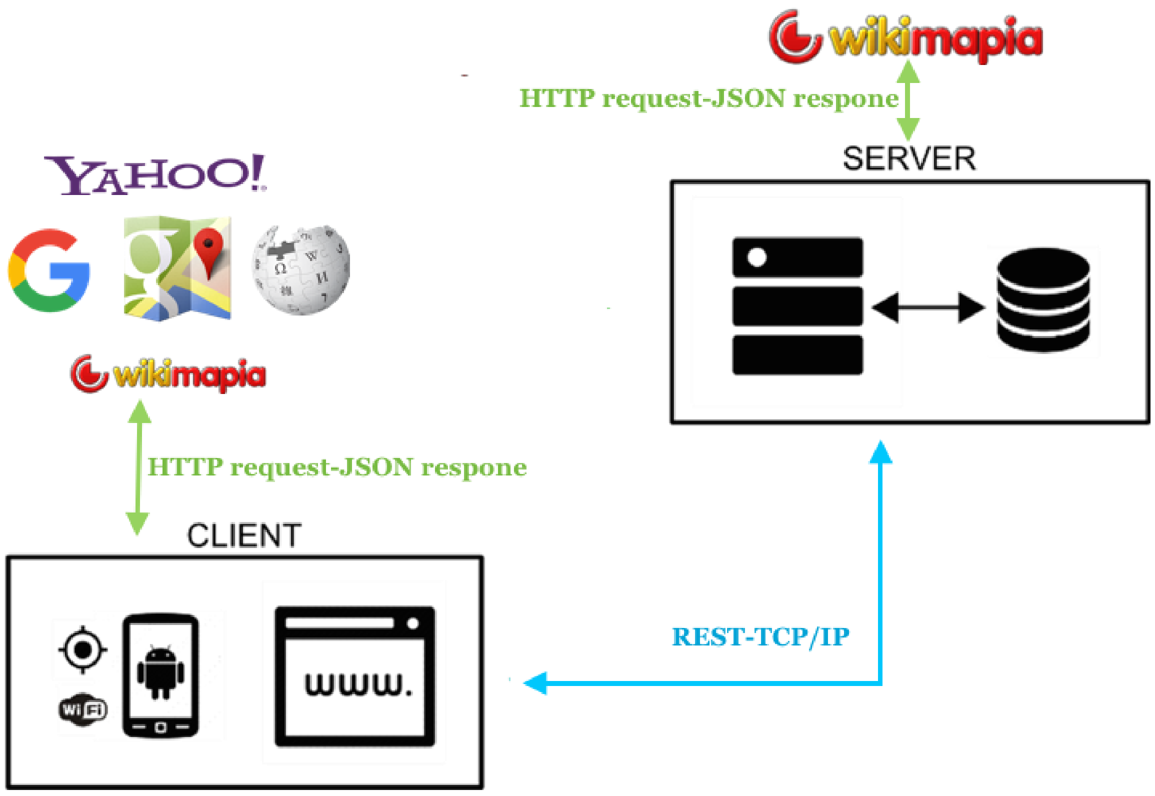
Client side of the application is a mobile app developed for Android with Android with Java programming language and XML user interface.

- Server tier

The server is a php server hosted on altervista (Apache HTTP server) developed following the RESTful paradigm.

- Data tier

It's composed by a MySql database, accessible only from the server tier and by a SQLite database on the client tier.



1.1.2 Stakeholders

The Stakeholders of this application are tourists. They can use the application either by logging with Facebook or not.

Unlogged tourist: Scans monuments to retrieve information, visualizes information about the current city, see position of the saved monuments on the map, use the diary, upload an own photo of a monument if no one is present.

Logged tourist: same functionality of the unlogged user plus: can share the founded monuments on Facebook, can see the list of friends that have seen the same scanned monument, can save a backup of the diary on the server and downloaded it on another device (not implemented yet).

1.1.3 System functionalities

- Account:

Allow the user to login with Facebook. It is not mandatory to use the app, is required only for social functionalities.

- Scan:

If there are a network connection and a gps signal, the scan functionality send a request to the server that responds with a list of POIs (point of interest) near the user position.

An Algorithm using the device's orientation return the POI in front of the user, if it exists. If a POI is founded, a request is sent to Wikipedia to obtain more information about it. The final result is then presented to the user, saved in the local db and in the server's one.

- Info:

If there are a network connection and a gps signal the Info functionality shows some informations about the city that the user is visiting. In particular it displays the current weather conditions and a list with the five most searched (scanned with the app) monuments of the city. For each monument it shows the distance from the user's position and the monument informations.

- My Monuments:

Shows, from the most to the less recent, all the saved monuments.

- Diary:

Automatically divides the saved monument into trips. A trip groups monuments from the same city that are scanned in the same period of time. If a user visit more cities the cities' trips are grouped into a single state's trip.

- Google Maps integration:

The application uses Maps to display the position of the monuments and the path from the user's position. This functionality is useful to see the position of the best 5 monuments of a city, and if a user that see a saved monument doesn't remember its location.

- Take a picture:

This functionality, if the result of a scan doesn't contain a picture, allows the user to take a picture by himself of the searched monument. Than the picture will be uploaded to the server and if it will be approved it will added to the information of the monument.

- Facebook integration:

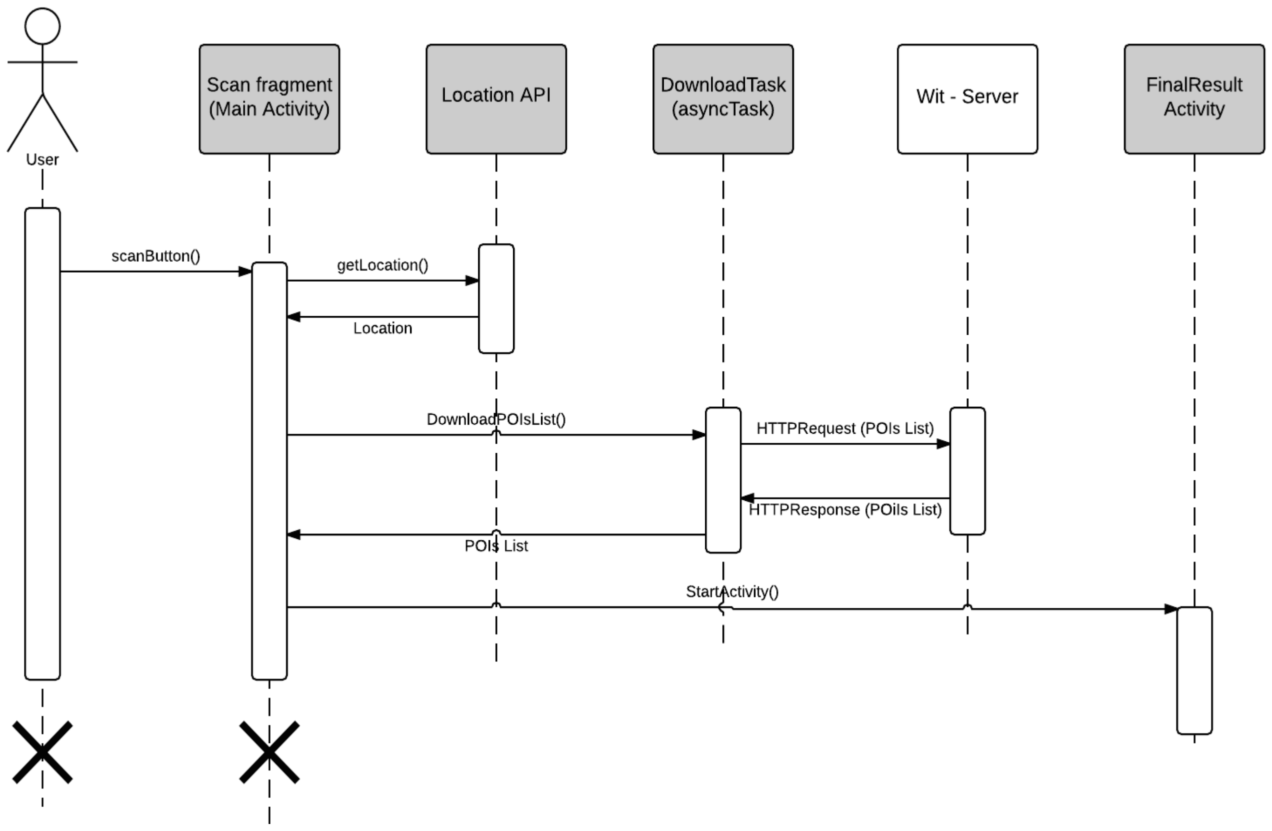
If the user is logged can decide to share the founded monument on his Facebook timeline, also when a monument is founded the user can see a list of friends that scanned the same monument with the application.

- Settings:

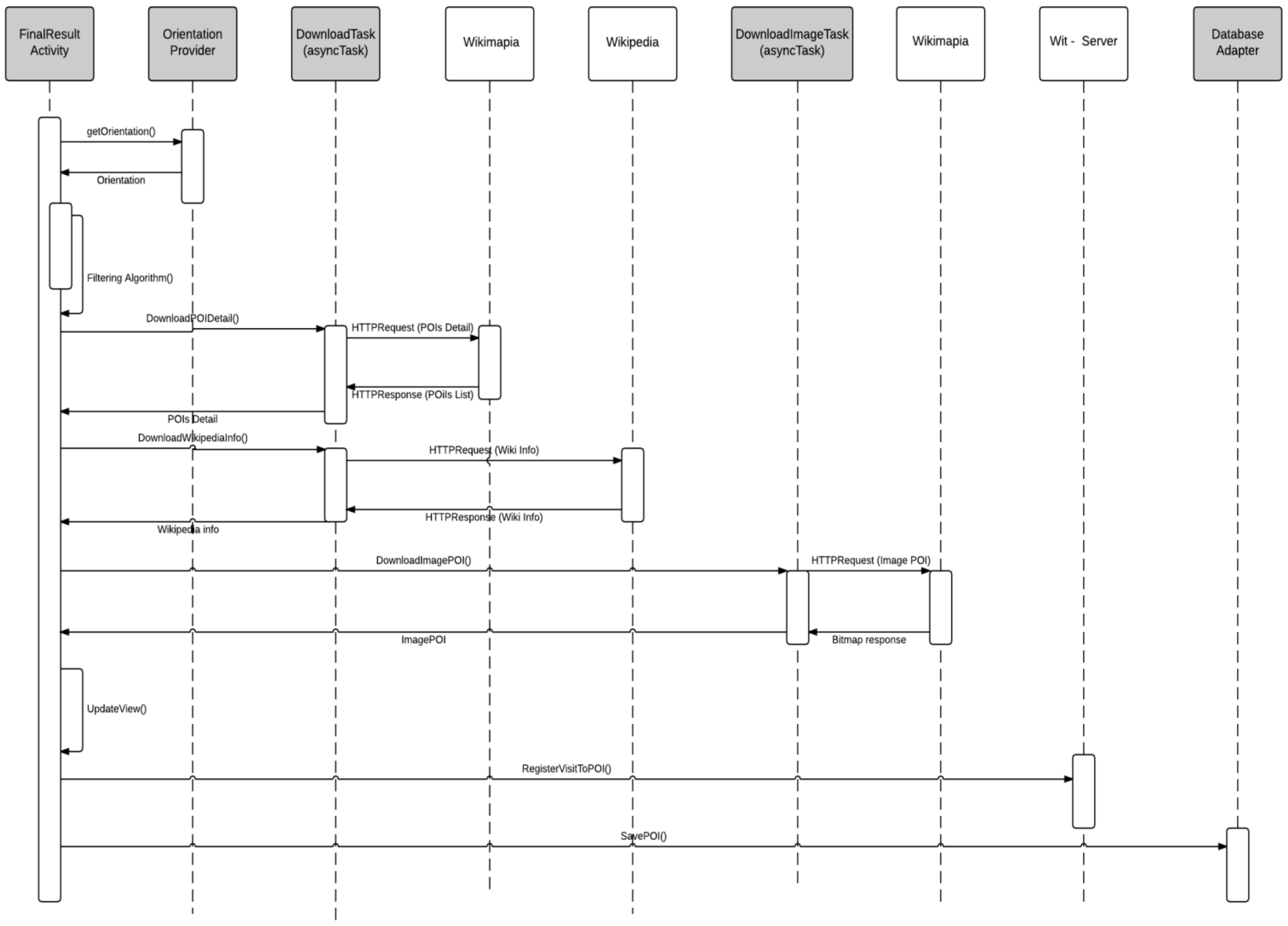
  Allow to able or disable the Facebook sharing functionality.

1.1.3.1 Sequence diagrams of the main functionalites

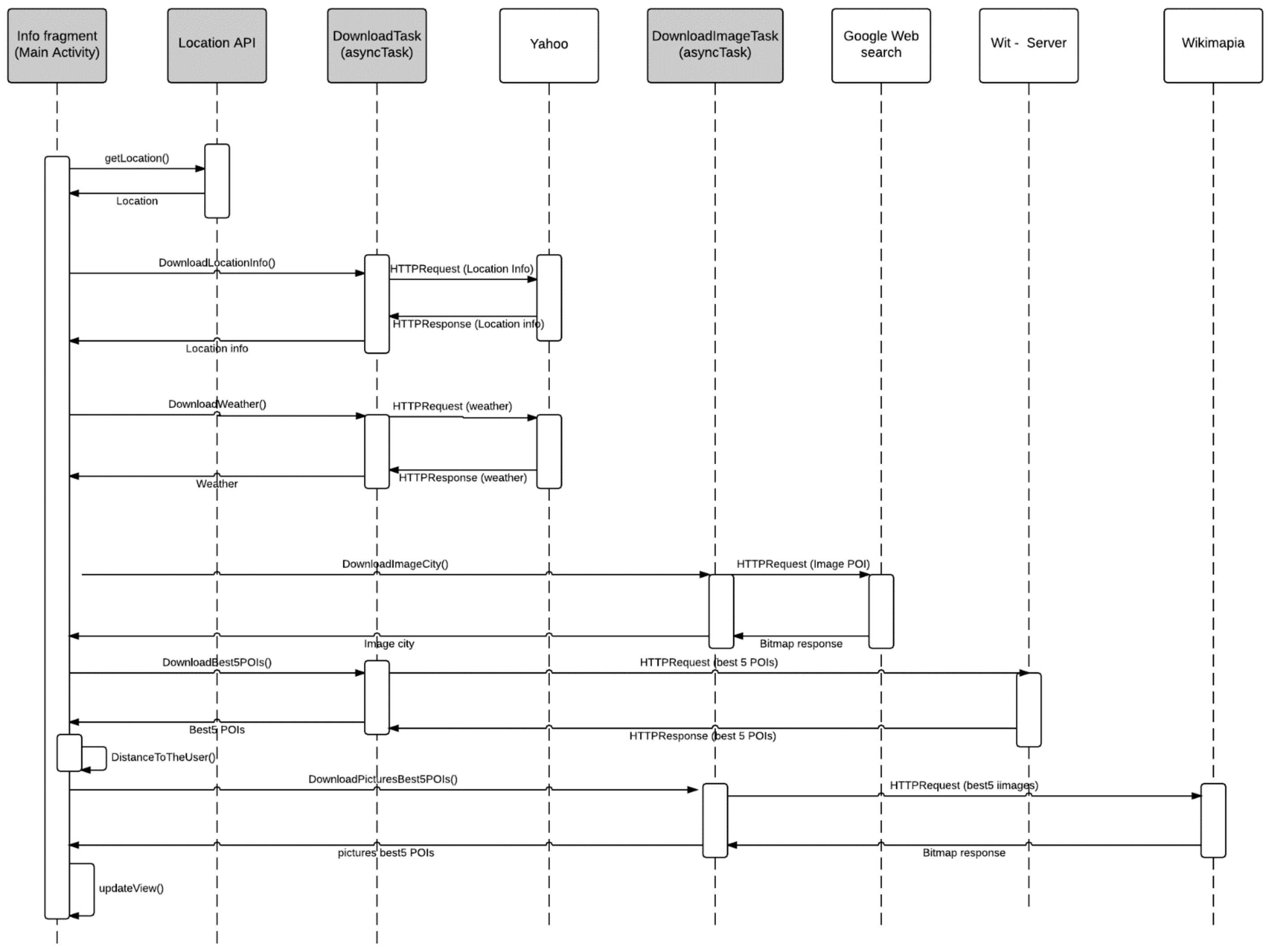
- Scan Activity



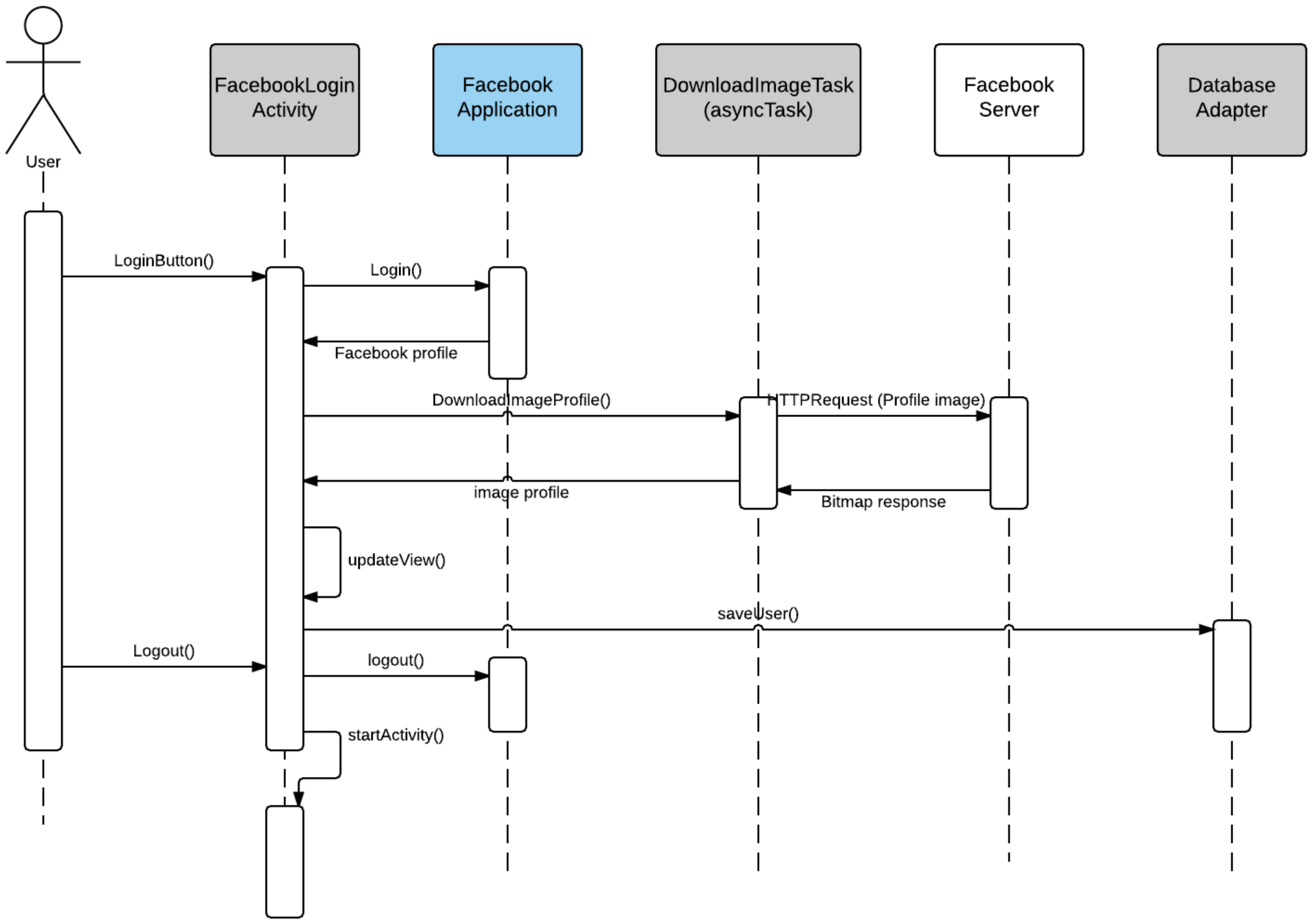
- Final result



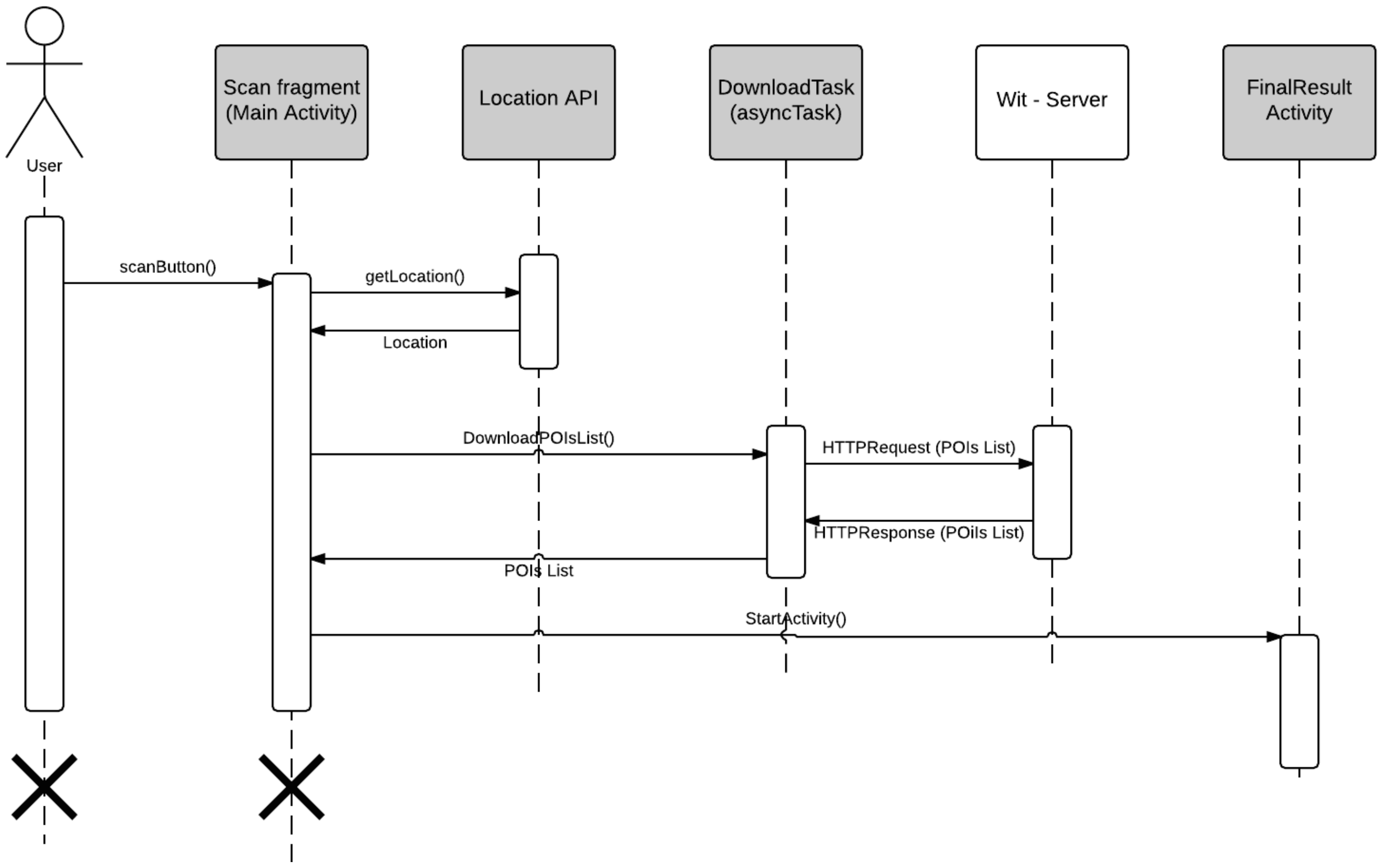
- Info


Info fragment (Main Activity)
Location API
DownloadTask (asyncTask)
Yahoo
DownloadImageTask (asyncTask)
Google Web search
Wit - Server
Wikimapia
getLocation()
Location
DownloadLocationInfo()
HTTPRequest (Location Info)
HTTPResponse (Location info)
Location info
DownloadWeather()
HTTPRequest (weather)
HTTPResponse (weather)
Weather
DownloadImageCity()
HTTPRequest (Image POI)
Bitmap response
Image city
DownloadBest5POIs()
HTTPRequest (best 5 POIs)
HTTPResponse (best 5 POIs)
Best5 POIs
DistanceToTheUser()
DownloadPicturesBest5POIs()
HTTPRequest (best5 iimages)
Bitmap response
pictures best5 POIs
updateView()

- Login


User
FacebookLogin Activity
Facebook Application
DownloadImageTask (asyncTask)
Facebook Server
Database Adapter
LoginButton()
Login()
Facebook profile
DownloadImageProfile()
HTTPRequest (Profile image)
image profile
Bitmap response
updateView()
saveUser()
Logout()
logout()
startActivity()

- Take a picture



1.1.4 Goals

- Precision:

  The application must be precise, it has to find the right monument that the user is scanning.

- Speed:

  The application should not be too slow to find the result and download the information.

2 Overall Description

2.1 Product Functions

2.1.1 Functional Requirements

- Login with Facebook
- Find information about a monument
- See information about the current city
- View monument's location
- View the summary of a trip in the diary
- View information about a saved monument
- Upload a picture of a monument
- Share a monument on Facebook (Logged user)
- See friend list that see the same monument (Logged user)

2.1.2 Non Functional Requirements

- Server online 24/7.
- Manage cuncurrent requests.

2.2 User Characteristics

Unlogged user: A tourist that is not logged to the application with Facebook.

Logged user: A tourist interested in the social functionality that is logged with Facebook.

2.3 Requirements to be implemented

- Gamification:

  Encourage users to collect POIs of various type to achieve trophies.

- Offline mode:

  Let the user to previous download from home a package with all the info of a particular city, so he can visit it without an internet connections.

- Push notifications between two users when a user's friends scan a monument that the other has recently scanned.

2.4 Data sources

- Wikimapia:

  Wikimapia is an open-content collaborative mapping project that combine the characteristic of Google Maps and Wikipedia with the aim to mark and describe all geographical objects in the world. Its contents are reliable and generated by users. Is available worldwide and counts over 20 millions of places, described in 35 languages.

  Wikimapia is used to obtained all the POIs around a user when he perform a "Scan action".

- Wikipedia:

  Wikipedia is a free-access, free-content Internet encyclopedia. Is used download addictional information about the founded POIs.

- Yahoo PlaceFinder:

  PlaceFinder is a RESTful Web service that given latitude and longitude of a place gives some geographical information like the name of the city and country of the place, and a unique id called woeid that identifies the place on the earth. Is used to find the current city the user is visiting.

- Yahoo Weather:

  Yahoo Weather is a web service that provide up-to-date weather information for any location. Is used to display the weather of the city the user is visiting.

- Google Web Search:

  Google is the most used search engine on the internet. We use it to find the images about the cities visited by a user.

3 Application

The application is a native android application that supported devices with android 4.1 or higher.

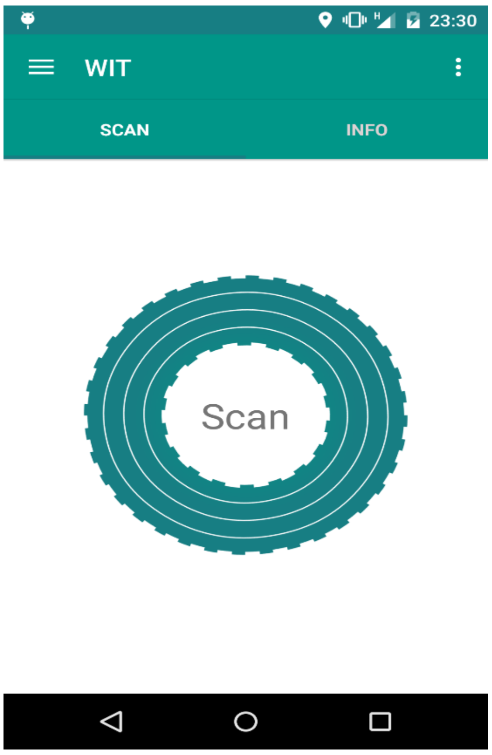
In order to run the application it requires an internet connection and a gps signal available.

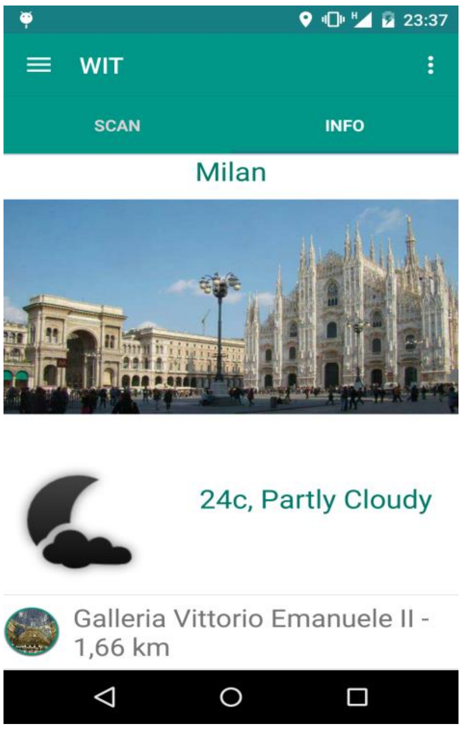
The android user permission that the application required are:

- “android.permission.INTERNET”
- “android.permission.ACCESS_NETWORK_STATE”
- “android.permission.ACCESS_FINE_LOCATION”
- “android.permission.ACCESS_COARSE_LOCATION"
- “android.permission.WRITE_EXTERNAL_STORAGE”
- “com.google.android.providers.gsf.permission.READ_GSERVICES”
- "it.polimi.dmw.wit.permission.MAPS_RECEIVE"

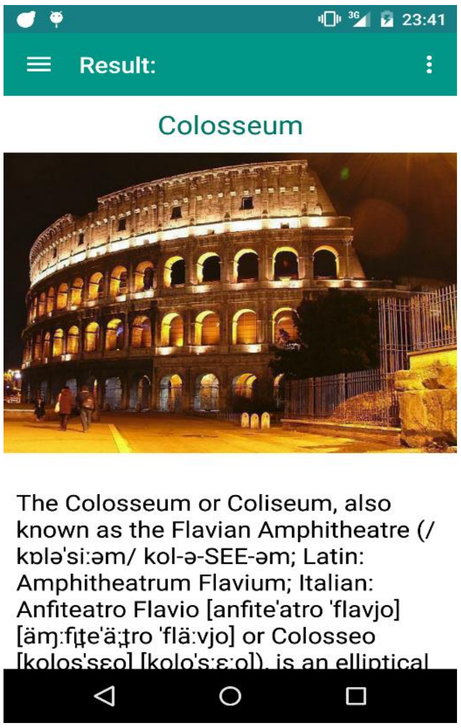
3.1 User manual

The main activity of the application consist in two fragments: one called Scan in which there is a button for the scanning activity, the other is called Info and contain the information about the city a user is current visiting.

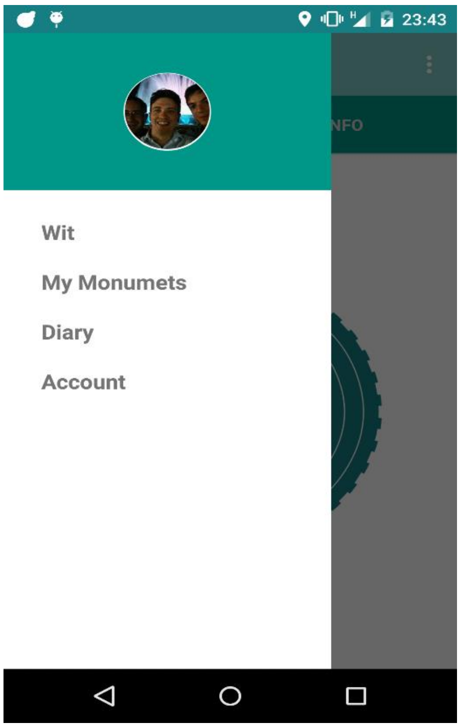




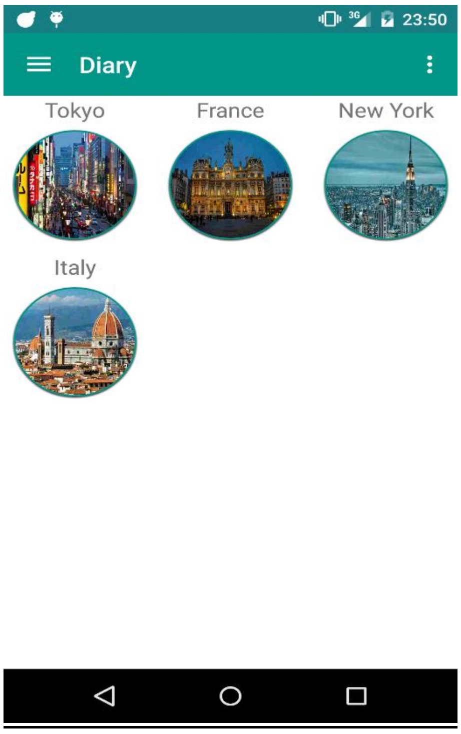
The final result activity displayes all the information about a monument: name picture if is present and a description. There is also a button to take a picture if the picture is not present. If the user is logged to Facebook and one or more friends have scanned the same monument his or their photo and name appears at the bottom of the page. A slider menu is used to navigate between the activities of the applications.

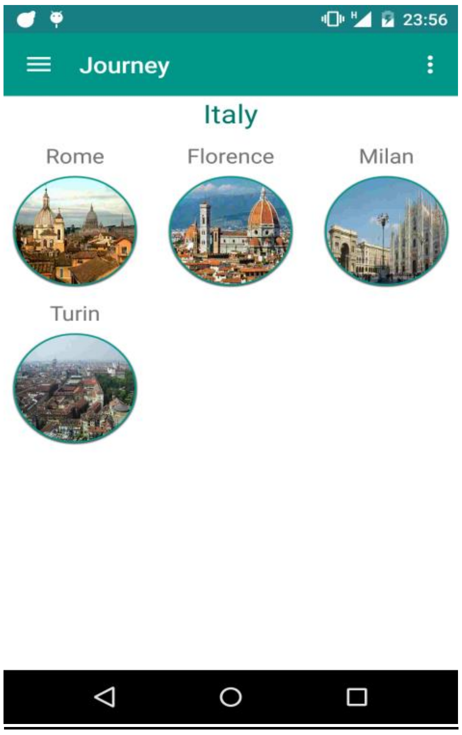


The Colosseum or Coliseum, also known as the Flavian Amphitheatre (/ kɒlə'siːəm/ kol-ə-SEE-əm; Latin: Amphitheatrum Flavium; Italian: Anfiteatro Flavio [anfite'atro 'flavjo] [äɱ:fi̞te'ä:t̪ro 'flä:vjo] or Colosseo [kolos'sɛo] [kolo's:ɛ:o]), is an elliptical



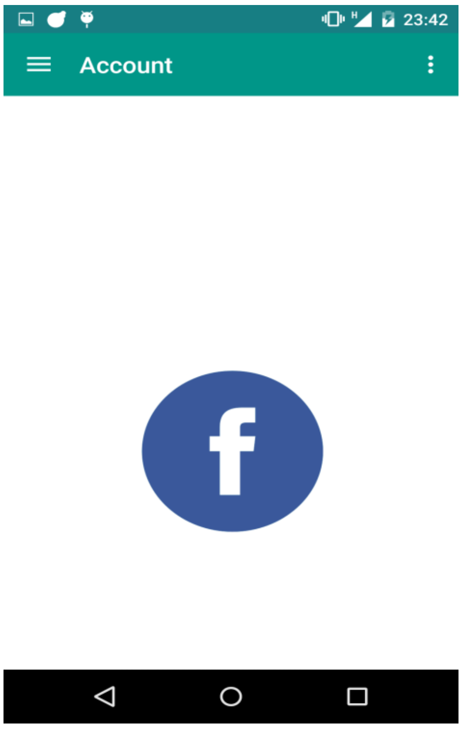
The "My Monuments" activity builds a list with all the saved monuments from the newest to the oldest. The diary activity visualizes all the user's trips. If a trip includes more than one city an intermediate activity is displayed with a list with all the cities of the trip. The trip detail activity displays all the seen monuments in chronological order.

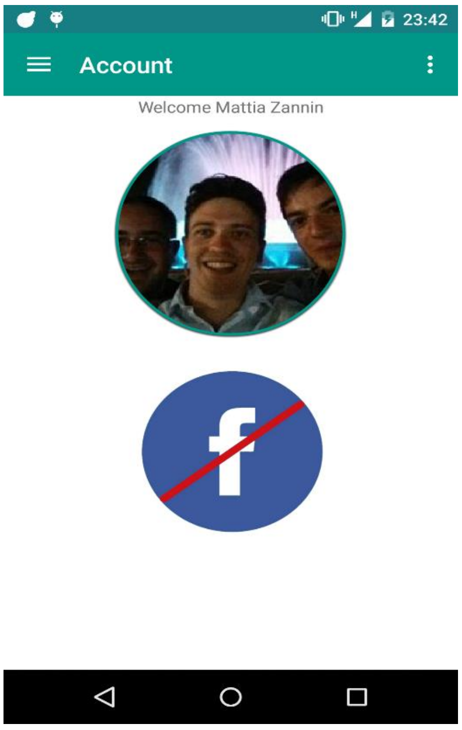


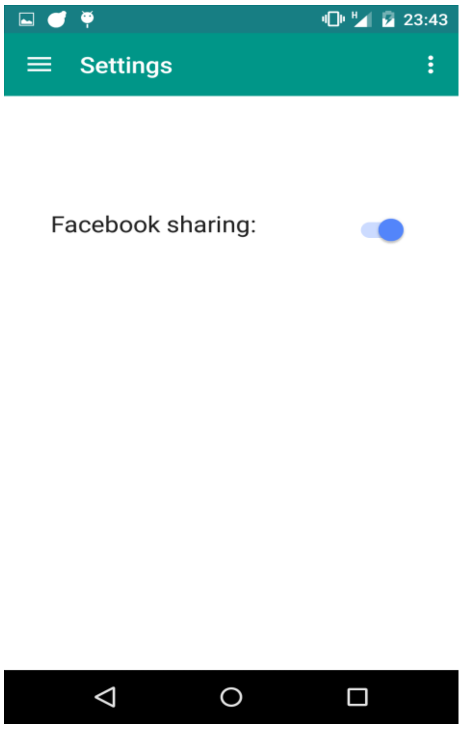


The account activity has a button to login/logout to Facebook.

The setting activity have a switch button to enable/disable the Facebook sharing functionality.



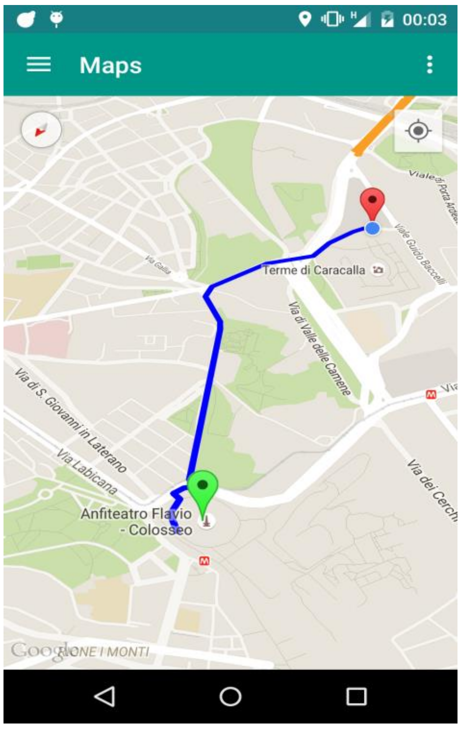


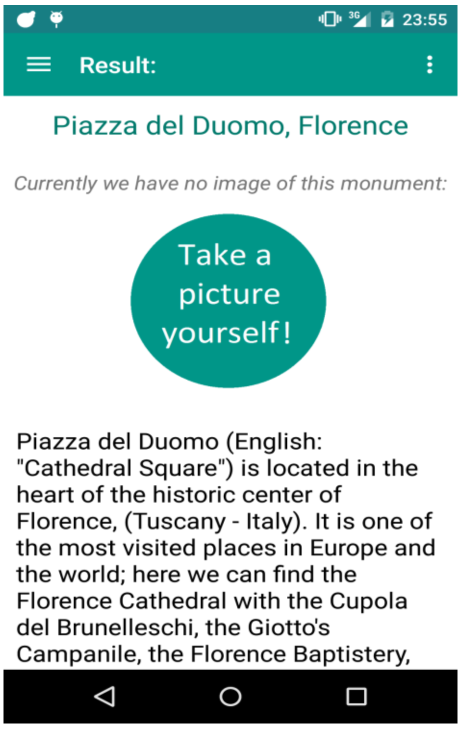


Facebook sharing:

If in the final result activity the monuments has no image, the user can upload one by himself.

User can check the position of a POI in the info activity.


00:03
Maps
Terme di Caracalla
Via di Valle delle Camene
Via di S. Giovanni in Laterano
Via Labicana
Anfiteatro Flavio - Colosseo
Via dei Cerchi
Google



Piazza del Duomo, Florence

Currently we have no image of this monument:

Piazza del Duomo (English: "Cathedral Square") is located in the heart of the historic center of Florence, (Tuscany - Italy). It is one of the most visited places in Europe and the world; here we can find the Florence Cathedral with the Cupola del Brunelleschi, the Giotto's Campanile, the Florence Baptistery,

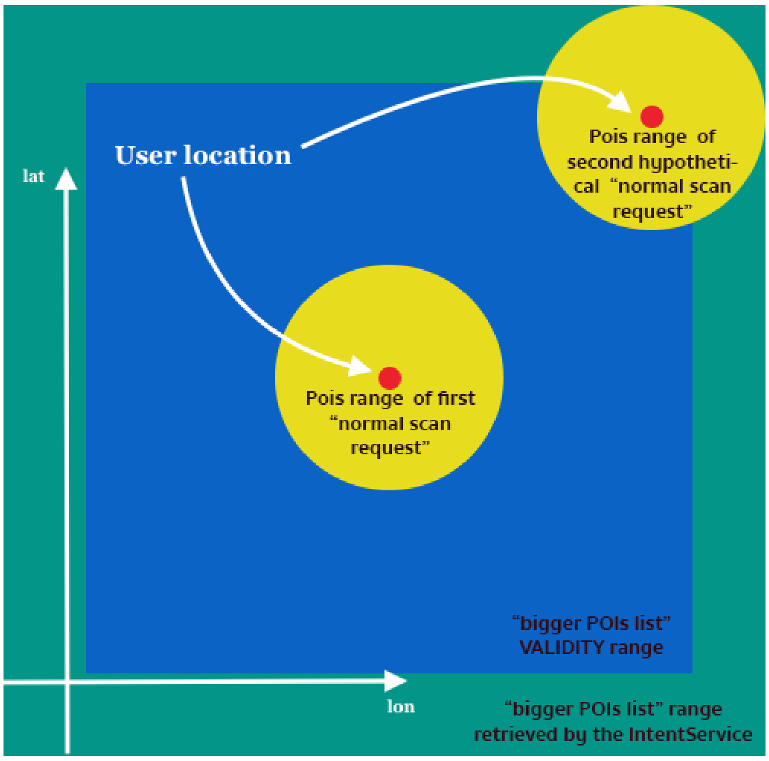
3.2 Caching & optimizations

When performing the scan action, often the request for the POIs list to the server is the bottle neck of the operation, especially in cities with a huge number of monuments.

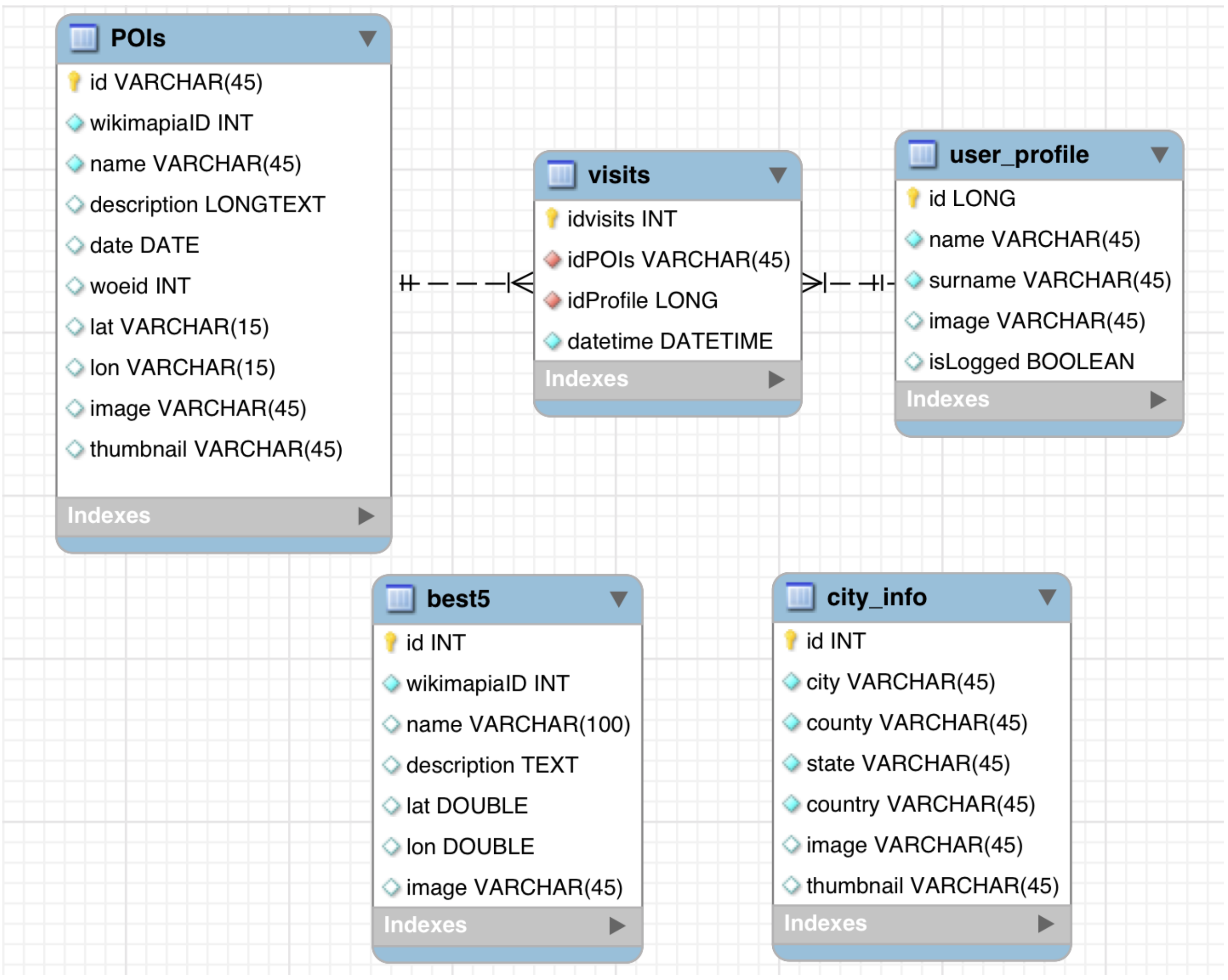
(Due to the fact that the server has to send to wikimapia one request every 100 monuments to obtain more informations about them, some optimizations are also done server side)

So, potentially, one user's scan can correspond to even 5 requests to wikimapia, this is not admissible because wikimapia's keys have a limited number of requests each hour.

We solved this problem client side, by sending a requests to the server in a parallel thread (IntentService) for a POIs list with an higher radius (wrt the normal send by the scan activity) and we store the Json answer in the internal memory of the phone.
So in the scan action, if the user isn't gone too far and the normal POIs list range is contained in the bigger Json, no server request is send for the POIs list.


User location
Pois range of second hypothetical "normal scan request"
Pois range of first "normal scan request"
lat
lon
"bigger POIs list" VALIDITY range
"bigger POIs list" range retrieved by the IntentService

3.3 Local Database:

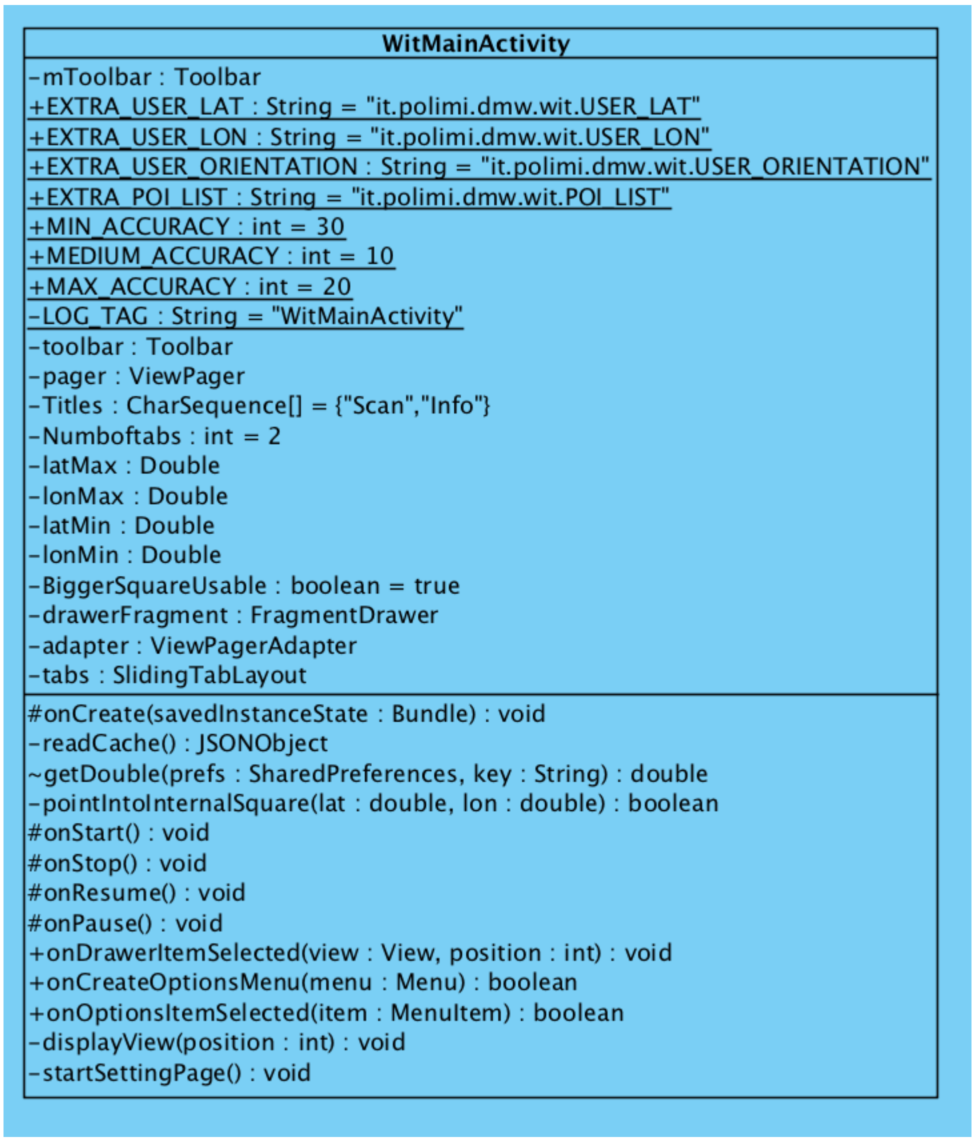


3.4 Class diagram

Classes are divided into two packages: one called “activities” that contains all the android activities and one called “utilities” that contains all the other support classes.

Activities:

MainActivity:

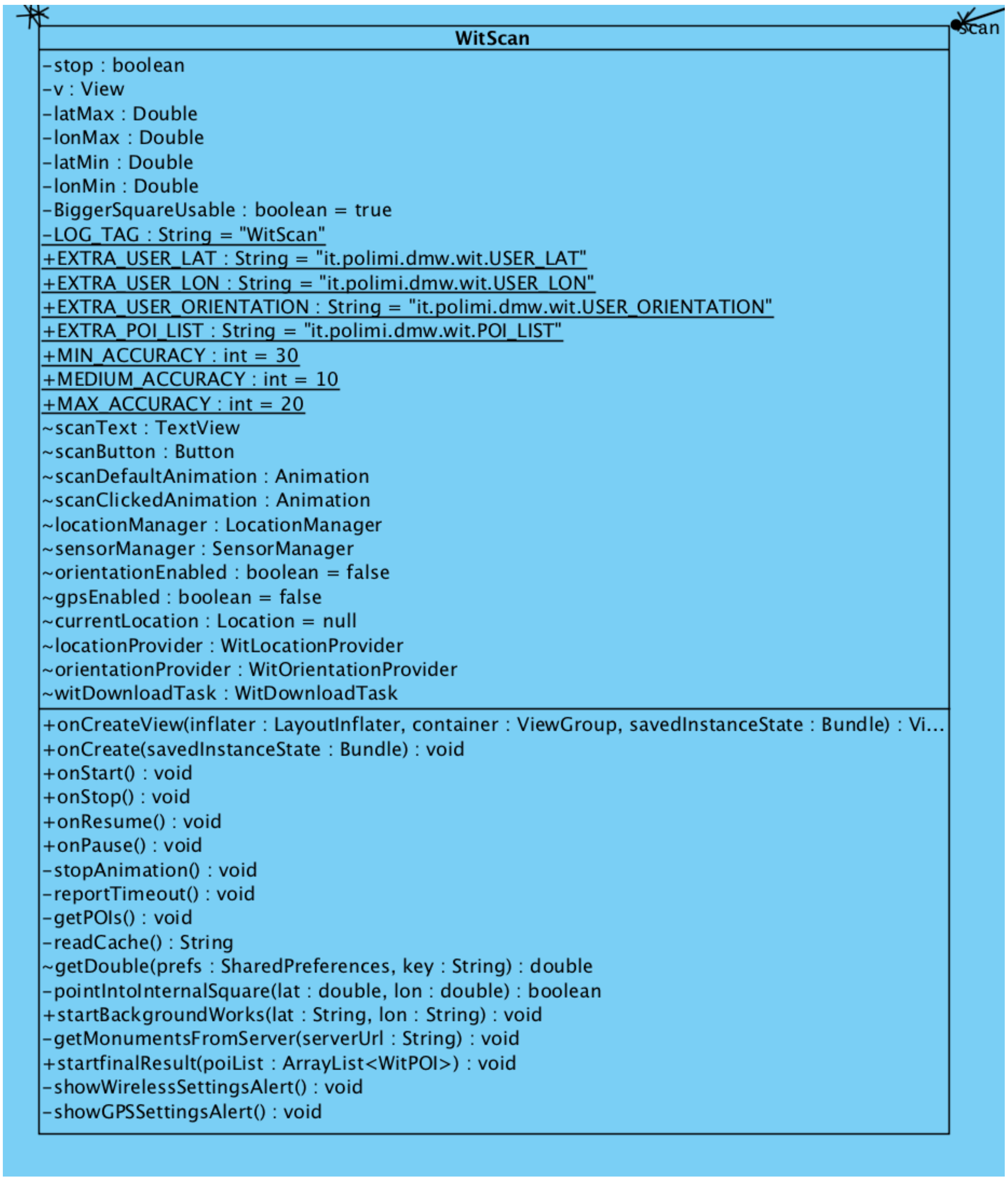


```
WitMainActivity
-mToolbar : Toolbar
+EXTRA_USER_LAT : String = "it.polimi.dmw.wit.USER_LAT"
+EXTRA_USER_LON : String = "it.polimi.dmw.wit.USER_LON"
+EXTRA_USER_ORIENTATION : String = "it.polimi.dmw.wit.USER_ORIENTATION"
+EXTRA_POI_LIST : String = "it.polimi.dmw.wit.POI_LIST"
+MIN_ACCURACY : int = 30
+MEDIUM_ACCURACY : int = 10
+MAX_ACCURACY : int = 20
-LOG_TAG : String = "WitMainActivity"
-toolbar : Toolbar
-pager : ViewPager
-Titles : CharSequence[] = {"Scan","Info"}
-Numboftabs : int = 2
-latMax : Double
-lonMax : Double
-latMin : Double
-lonMin : Double
-BiggerSquareUsable : boolean = true
-drawerFragment : FragmentDrawer
-adapter : ViewPagerAdapter
-tabs : SlidingTabLayout

#onCreate(savedInstanceState : Bundle) : void
-readCache() : JSONObject
~getDouble(prefs : SharedPreferences, key : String) : double
-pointIntoInternalSquare(lat : double, lon : double) : boolean
#onStart() : void
#onStop() : void
#onResume() : void
#onPause() : void
+onDrawerItemSelected(view : View, position : int) : void
+onCreateOptionsMenu(menu : Menu) : boolean
+onOptionsItemSelected(item : MenuItem) : boolean
-displayView(position : int) : void
-startSettingPage() : void
```

This is the starting activity of the application and it is composed of two fragments Scan and Info.

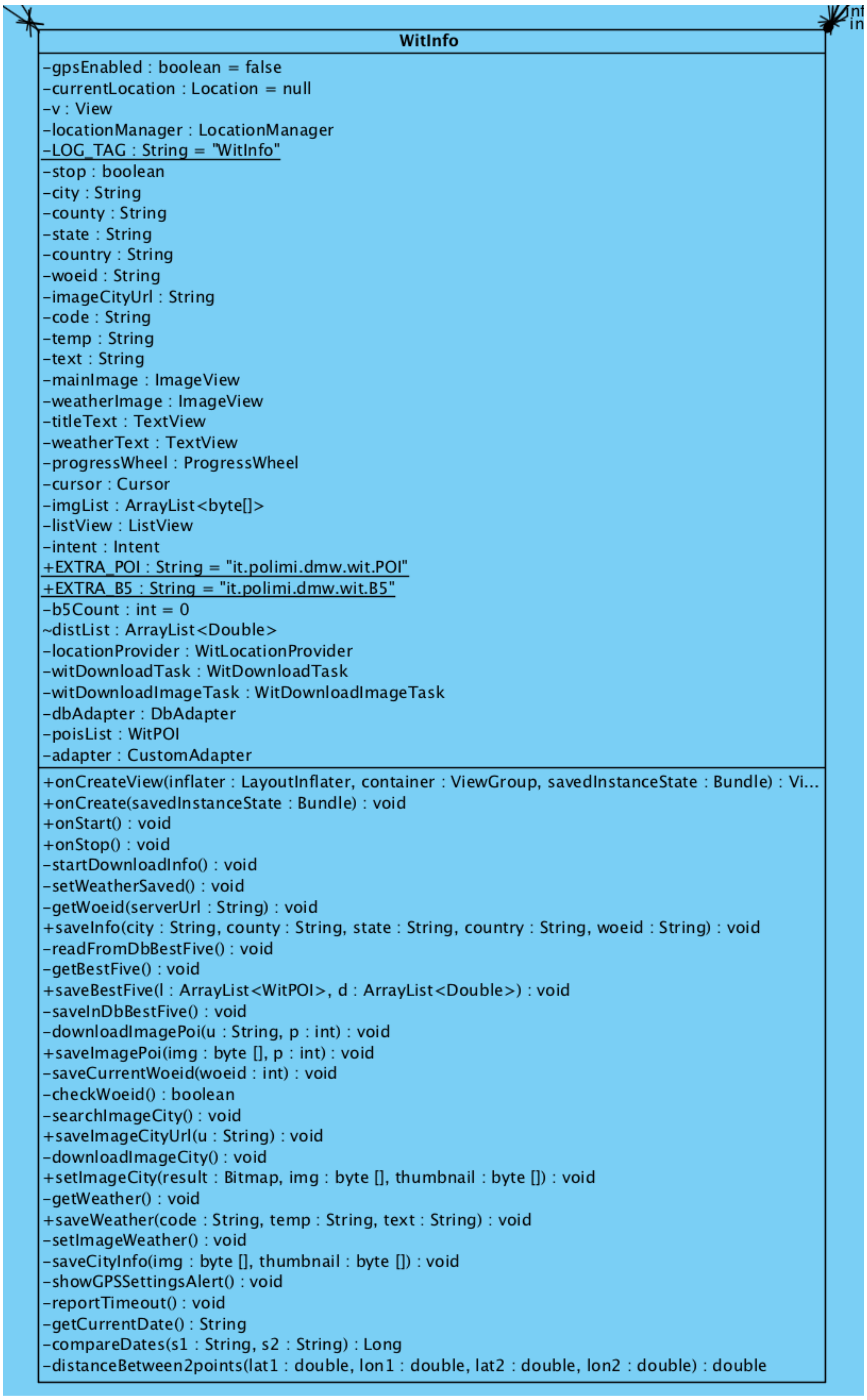
Scan:



```
WitScan
-stop : boolean
-v : View
-latMax : Double
-lonMax : Double
-latMin : Double
-lonMin : Double
-BiggerSquareUsable : boolean = true
-LOG_TAG : String = "WitScan"
+EXTRA_USER_LAT : String = "it.polimi.dmw.wit.USER_LAT"
+EXTRA_USER_LON : String = "it.polimi.dmw.wit.USER_LON"
+EXTRA_USER_ORIENTATION : String = "it.polimi.dmw.wit.USER_ORIENTATION"
+EXTRA_POI_LIST : String = "it.polimi.dmw.wit.POI_LIST"
+MIN_ACCURACY : int = 30
+MEDIUM_ACCURACY : int = 10
+MAX_ACCURACY : int = 20
~scanText : TextView
~scanButton : Button
~scanDefaultAnimation : Animation
~scanClickedAnimation : Animation
~locationManager : LocationManager
~sensorManager : SensorManager
~orientationEnabled : boolean = false
~gpsEnabled : boolean = false
~currentLocation : Location = null
~locationProvider : WitLocationProvider
~orientationProvider : WitOrientationProvider
~witDownloadTask : WitDownloadTask
+onCreateView(inflater : LayoutInflater, container : ViewGroup, savedInstanceState : Bundle) : Vi...
+onCreate(savedInstanceState : Bundle) : void
+onStart() : void
+onStop() : void
+onResume() : void
+onPause() : void
-stopAnimation() : void
-reportTimeout() : void
-getPOIs() : void
-readCache() : String
~getDouble(prefs : SharedPreferences, key : String) : double
-pointIntoInternalSquare(lat : double, lon : double) : boolean
+startBackgroundWorks(lat : String, lon : String) : void
-getMonumentsFromServer(serverUrl : String) : void
+startfinalResult(poiList : ArrayList<WitPOI>) : void
-showWirelessSettingsAlert() : void
-showGPSSettingsAlert() : void
```

The Scan fragment contains the “Scan” button that start the scanning process of a POI. It starts a timeout Thread that periodically check if the gps service can obtain a gps position enouth accurate. If a position is obtained it send a request to the server to obtained the list of POIs near the current location. The list than is send using an Intent to the FinalResultActivity. If the timeout happens an errore message is shown to the user.

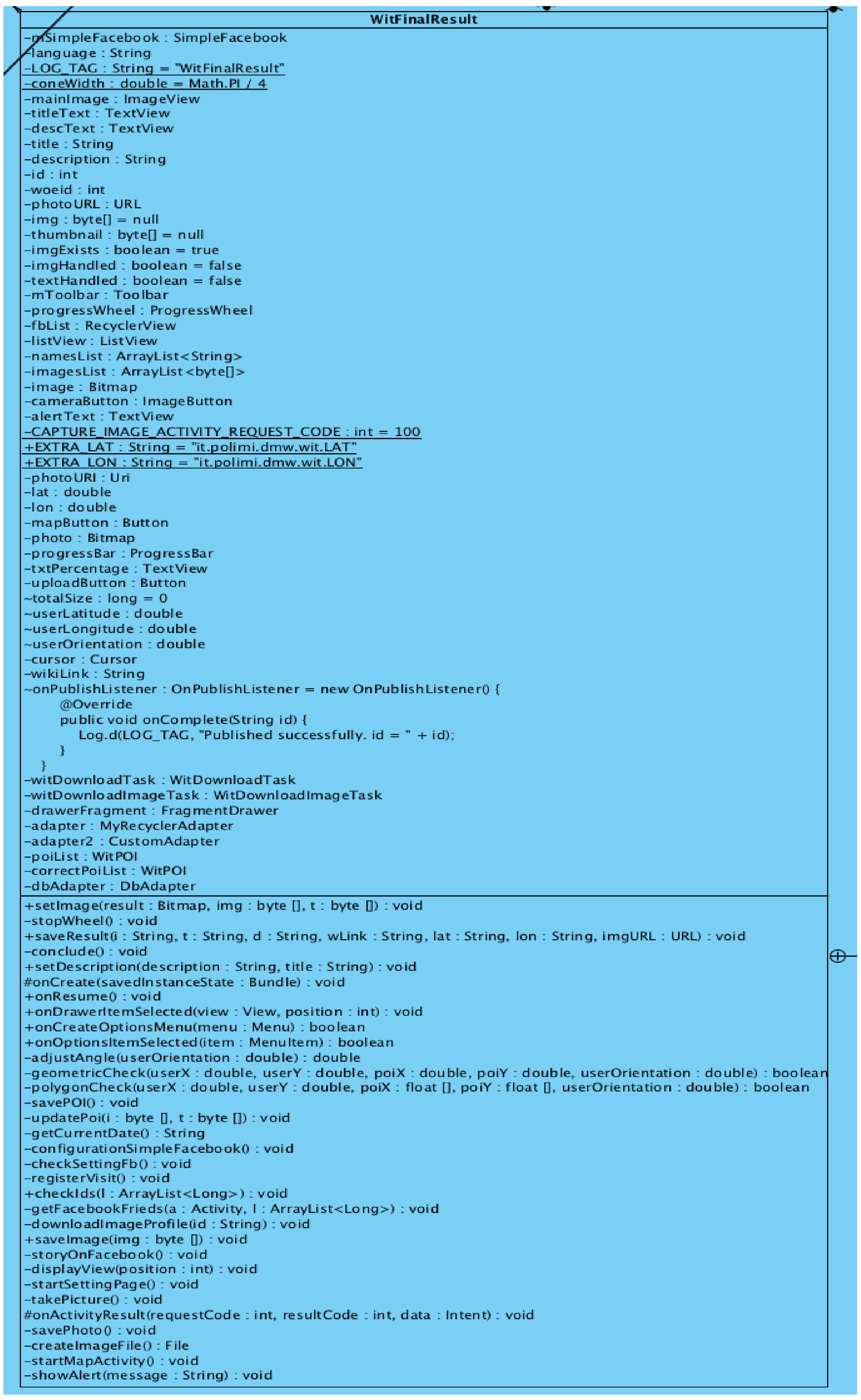
Info:



```
WitInfo
-gpsEnabled : boolean = false
-currentLocation : Location = null
-v : View
-locationManager : LocationManager
-LOG_TAG : String = "WitInfo"
-stop : boolean
-city : String
-county : String
-state : String
-country : String
-woeid : String
-imageCityUrl : String
-code : String
-temp : String
-text : String
-mainImage : ImageView
-weatherImage : ImageView
-titleText : TextView
-weatherText : TextView
-progressWheel : ProgressWheel
-cursor : Cursor
-imgList : ArrayList<byte[]>
-listView : ListView
-intent : Intent
+EXTRA_POI : String = "it.polimi.dmw.wit.POI"
+EXTRA_B5 : String = "it.polimi.dmw.wit.B5"
-b5Count : int = 0
~distList : ArrayList<Double>
-locationProvider : WitLocationProvider
-witDownloadTask : WitDownloadTask
-witDownloadImageTask : WitDownloadImageTask
-dbAdapter : DbAdapter
-poisList : WitPOI
-adapter : CustomAdapter
+onCreateView(inflater : LayoutInflater, container : ViewGroup, savedInstanceState : Bundle) : Vi...
+onCreate(savedInstanceState : Bundle) : void
+onStart() : void
+onStop() : void
-startDownloadInfo() : void
-setWeatherSaved() : void
-getWoeid(serverUrl : String) : void
+saveInfo(city : String, county : String, state : String, country : String, woeid : String) : void
-readFromDbBestFive() : void
-getBestFive() : void
+saveBestFive(l : ArrayList<WitPOI>, d : ArrayList<Double>) : void
-saveInDbBestFive() : void
-downloadImagePoi(u : String, p : int) : void
+saveImagePoi(img : byte [], p : int) : void
-saveCurrentWoeid(woeid : int) : void
-checkWoeid() : boolean
-searchImageCity() : void
+saveImageCityUrl(u : String) : void
-downloadImageCity() : void
+setImageCity(result : Bitmap, img : byte [], thumbnail : byte []) : void
-getWeather() : void
+saveWeather(code : String, temp : String, text : String) : void
-setImageWeather() : void
-saveCityInfo(img : byte [], thumbnail : byte []) : void
-showGPSSettingsAlert() : void
-reportTimeout() : void
-getCurrentDate() : String
-compareDates(s1 : String, s2 : String) : Long
-distanceBetween2points(lat1 : double, lon1 : double, lat2 : double, lon2 : double) : double
```

The info fragment display information about the city where the user is current located. If it could obtained a gps position it send latitude and longitude to Yahoo PlaceFinder. The response data are used to set up the weather, the image of the city with a request to Google Web Search, and the five most searched POIs of the city with a request to our server.

WitFinalResult:



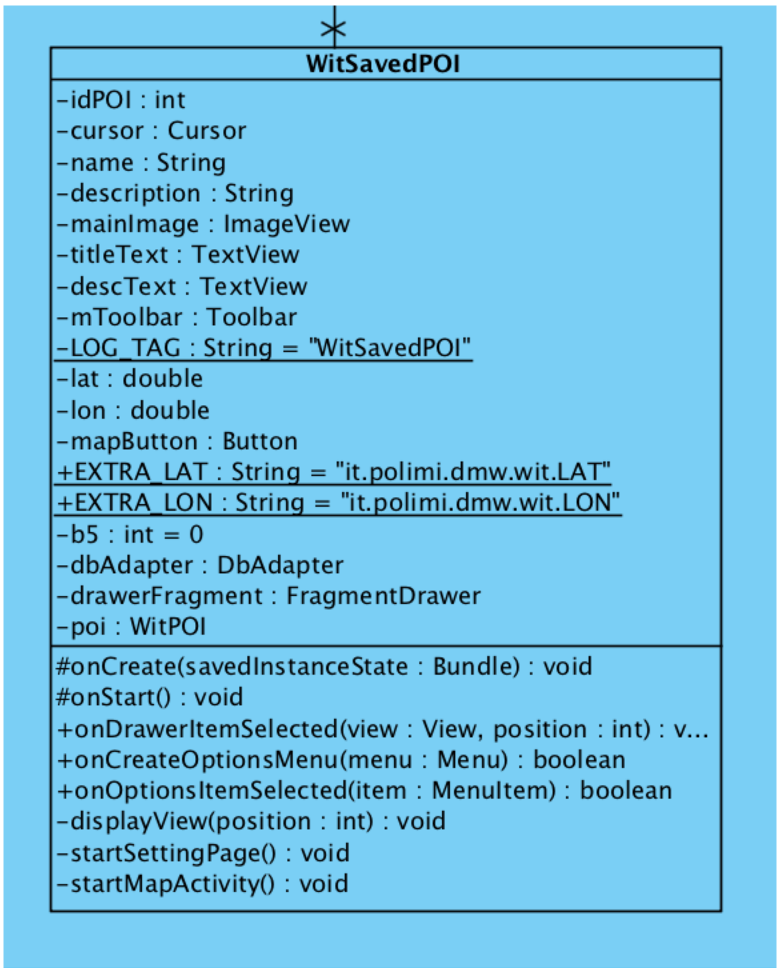
```
WitFinalResult
-mSimpleFacebook : SimpleFacebook
-language : String
-LOG_TAG : String = "WitFinalResult"
-coneWidth : double = Math.PI / 4
-mainImage : ImageView
-titleText : TextView
-descText : TextView
-title : String
-description : String
-id : int
-woeid : int
-photoURL : URL
-img : byte[] = null
-thumbnail : byte[] = null
-imgExists : boolean = true
-imgHandled : boolean = false
-textHandled : boolean = false
-mToolbar : Toolbar
-progressWheel : ProgressWheel
-fbList : RecyclerView
-listView : ListView
-namesList : ArrayList<String>
-imagesList : ArrayList<byte[]>
-image : Bitmap
-cameraButton : ImageButton
-alertText : TextView
-CAPTURE_IMAGE_ACTIVITY_REQUEST_CODE : int = 100
+EXTRA_LAT : String = "it.polimi.dmw.wit.LAT"
+EXTRA_LON : String = "it.polimi.dmw.wit.LON"
-photoURI : Uri
-lat : double
-lon : double
-mapButton : Button
-photo : Bitmap
-progressBar : ProgressBar
-txtPercentage : TextView
-uploadButton : Button
~totalSize : long = 0
~userLatitude : double
~userLongitude : double
~userOrientation : double
-cursor : Cursor
-wikiLink : String
~onPublishListener : OnPublishListener = new OnPublishListener() {
        @Override
        public void onComplete(String id) {
            Log.d(LOG_TAG, "Published successfully. id = " + id);
        }
    }
-witDownloadTask : WitDownloadTask
-witDownloadImageTask : WitDownloadImageTask
-drawerFragment : FragmentDrawer
-adapter : MyRecyclerAdapter
-adapter2 : CustomAdapter
-poiList : WitPOI
-correctPoiList : WitPOI
-dbAdapter : DbAdapter
+setImage(result : Bitmap, img : byte [], t : byte []) : void
-stopWheel() : void
+saveResult(i : String, t : String, d : String, wLink : String, lat : String, lon : String, imgURL : URL) : void
-conclude() : void
+setDescription(description : String, title : String) : void
#onCreate(savedInstanceState : Bundle) : void
+onResume() : void
+onDrawerItemSelected(view : View, position : int) : void
+onCreateOptionsMenu(menu : Menu) : boolean
+onOptionsItemSelected(item : MenuItem) : boolean
-adjustAngle(userOrientation : double) : double
-geometricCheck(userX : double, userY : double, poiX : double, poiY : double, userOrientation : double) : boolean
-polygonCheck(userX : double, userY : double, poiX : float [], poiY : float [], userOrientation : double) : boolean
-savePOI() : void
-updatePoi(i : byte [], t : byte []) : void
-getCurrentDate() : String
-configurationSimpleFacebook() : void
-checkSettingFb() : void
-registerVisit() : void
+checkIds(l : ArrayList<Long>) : void
-getFacebookFrieds(a : Activity, l : ArrayList<Long>) : void
-downloadImageProfile(id : String) : void
+saveImage(img : byte []) : void
-storyOnFacebook() : void
-displayView(position : int) : void
-startSettingPage() : void
-takePicture() : void
#onActivityResult(requestCode : int, resultCode : int, data : Intent) : void
-savePhoto() : void
-createImageFile() : File
-startMapActivity() : void
-showAlert(message : String) : void
```

This activity use a geometric algorithm in combinations with the orientation of the device to identify from the list of POIs the closest one in front of the user if exist. In the positive case it send a request to Wikimapia and Wikipedia to download all the additional information of the founded POI. Than it display it and saved in the database. If there isn't any pictures link in the received JSON it gives the possibility to take a pictures with the device's camera and upload to our server.

If the user is logged to Facebook and has enabled the option it publish a post with the POI's detail on the user timeline.

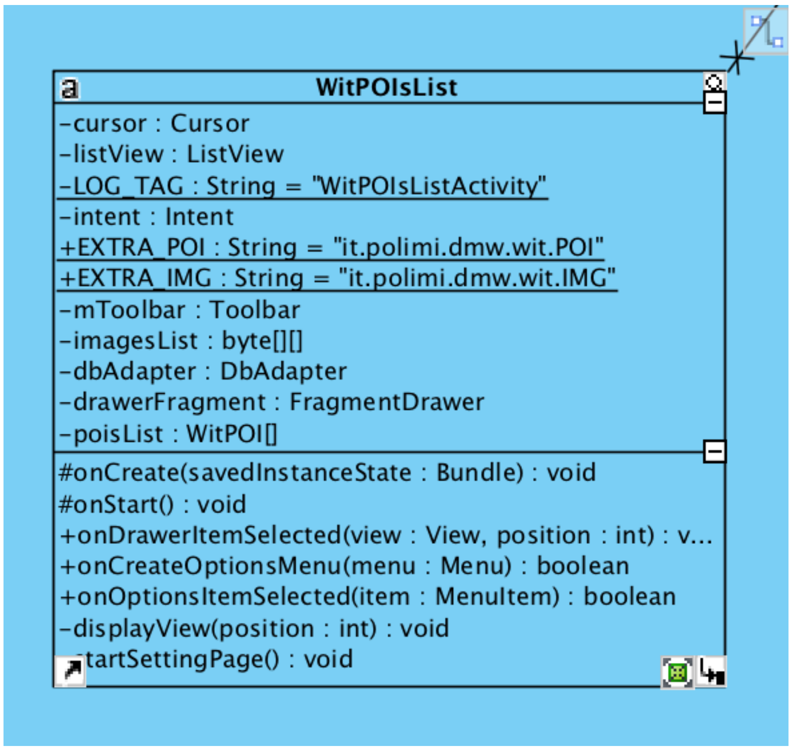
Finally it send a notification to the server to register the visit to the monument. The server respond with a list of ids of users that have seen the monument and if some of them are Facebook's friends of the user and the user is logged to Facebook, they are displayed at the end of the page.

WitSavedPOI:


WitSavedPOI
-idPOI : int
-cursor : Cursor
-name : String
-description : String
-mainImage : ImageView
-titleText : TextView
-descText : TextView
-mToolbar : Toolbar
-LOG_TAG : String = "WitSavedPOI"
-lat : double
-lon : double
-mapButton : Button
+EXTRA_LAT : String = "it.polimi.dmw.wit.LAT"
+EXTRA_LON : String = "it.polimi.dmw.wit.LON"
-b5 : int = 0
-dbAdapter : DbAdapter
-drawerFragment : FragmentDrawer
-poi : WitPOI
#onCreate(savedInstanceState : Bundle) : void
#onStart() : void
+onDrawerItemSelected(view : View, position : int) : v...
+onCreateOptionsMenu(menu : Menu) : boolean
+onOptionsItemSelected(item : MenuItem) : boolean
-displayView(position : int) : void
-startSettingPage() : void
-startMapActivity() : void

It display the details of a POI saved in the database.

WitPOIsList:



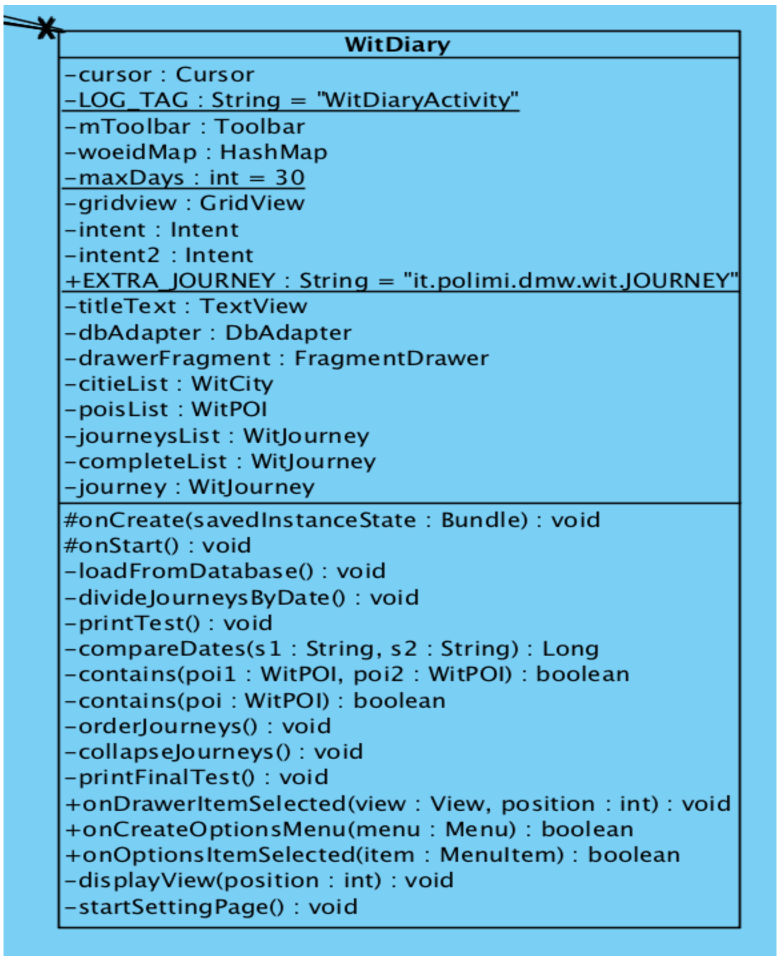
WitPOIsList

-cursor : Cursor
-listView : ListView
-LOG_TAG : String = "WitPOIsListActivity"
-intent : Intent
+EXTRA_POI : String = "it.polimi.dmw.wit.POI"
+EXTRA_IMG : String = "it.polimi.dmw.wit.IMG"
-mToolbar : Toolbar
-imagesList : byte[][]
-dbAdapter : DbAdapter
-drawerFragment : FragmentDrawer
-poisList : WitPOI[]

#onCreate(savedInstanceState : Bundle) : void
#onStart() : void
+onDrawerItemSelected(view : View, position : int) : v...
+onCreateOptionsMenu(menu : Menu) : boolean
+onOptionsItemSelected(item : MenuItem) : boolean
-displayView(position : int) : void
-startSettingPage() : void

It display the list of all the saved POIs

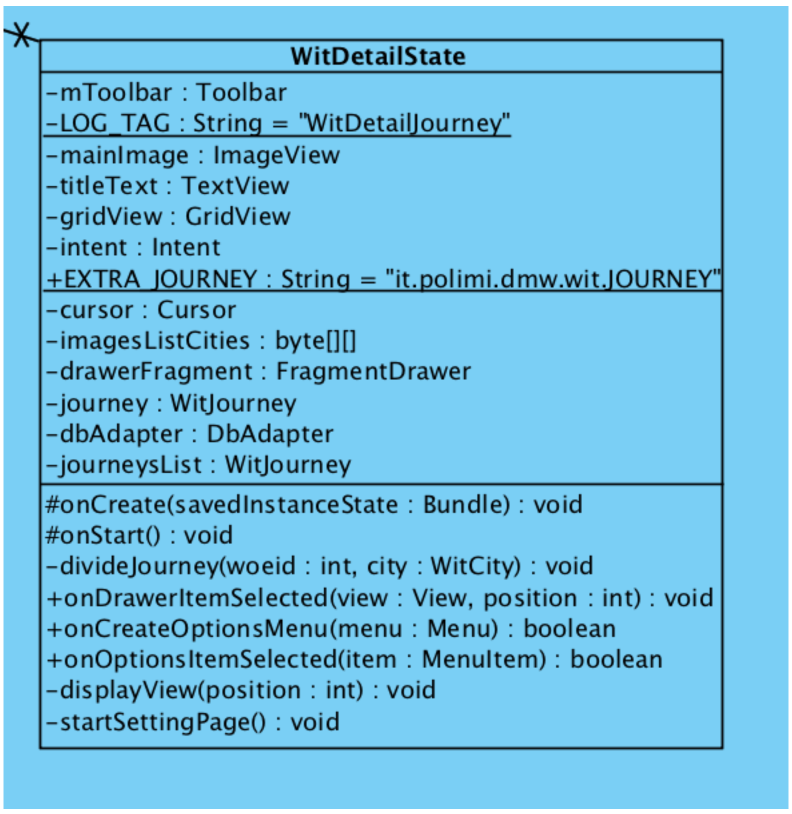
WitDiary:



```
WitDiary
-cursor : Cursor
-LOG_TAG : String = "WitDiaryActivity"
-mToolbar : Toolbar
-woeidMap : HashMap
-maxDays : int = 30
-gridview : GridView
-intent : Intent
-intent2 : Intent
+EXTRA_JOURNEY : String = "it.polimi.dmw.wit.JOURNEY"
-titleText : TextView
-dbAdapter : DbAdapter
-drawerFragment : FragmentDrawer
-citieList : WitCity
-poisList : WitPOI
-journeysList : WitJourney
-completeList : WitJourney
-journey : WitJourney
#onCreate(savedInstanceState : Bundle) : void
#onStart() : void
-loadFromDatabase() : void
-divideJourneysByDate() : void
-printTest() : void
-compareDates(s1 : String, s2 : String) : Long
-contains(poi1 : WitPOI, poi2 : WitPOI) : boolean
-contains(poi : WitPOI) : boolean
-orderJourneys() : void
-collapseJourneys() : void
-printFinalTest() : void
+onDrawerItemSelected(view : View, position : int) : void
+onCreateOptionsMenu(menu : Menu) : boolean
+onOptionsItemSelected(item : MenuItem) : boolean
-displayView(position : int) : void
-startSettingPage() : void
```

It display the list of all user’s trips.

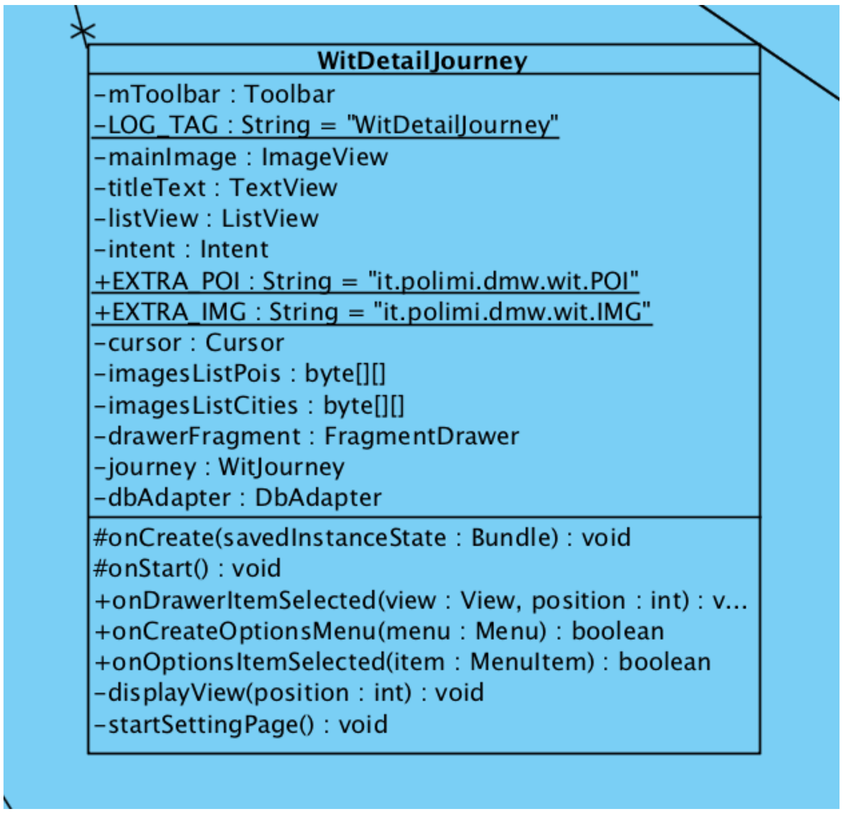
WitDetailState:



```
WitDetailState
-mToolbar : Toolbar
-LOG_TAG : String = "WitDetailJourney"
-mainImage : ImageView
-titleText : TextView
-gridView : GridView
-intent : Intent
+EXTRA_JOURNEY : String = "it.polimi.dmw.wit.JOURNEY"
-cursor : Cursor
-imagesListCities : byte[][]
-drawerFragment : FragmentDrawer
-journey : WitJourney
-dbAdapter : DbAdapter
-journeysList : WitJourney
#onCreate(savedInstanceState : Bundle) : void
#onStart() : void
-divideJourney(woeid : int, city : WitCity) : void
+onDrawerItemSelected(view : View, position : int) : void
+onCreateOptionsMenu(menu : Menu) : boolean
+onOptionsItemSelected(item : MenuItem) : boolean
-displayView(position : int) : void
-startSettingPage() : void
```

It display all the cities visited in the same country in a trip.

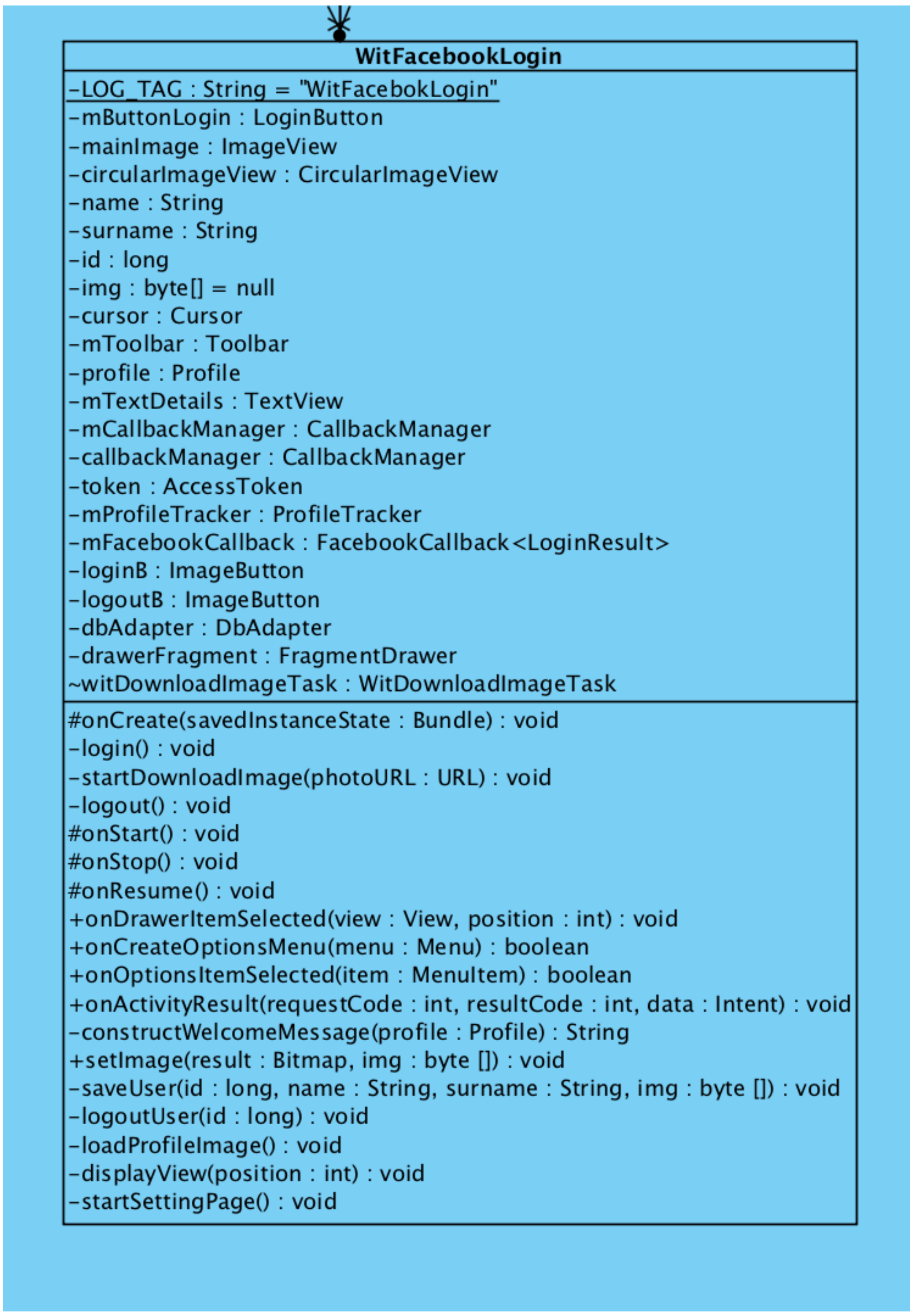
WitDetailJourney:



```
WitDetailJourney
-mToolbar : Toolbar
-LOG_TAG : String = "WitDetailJourney"
-mainImage : ImageView
-titleText : TextView
-listView : ListView
-intent : Intent
+EXTRA_POI : String = "it.polimi.dmw.wit.POI"
+EXTRA_IMG : String = "it.polimi.dmw.wit.IMG"
-cursor : Cursor
-imagesListPois : byte[][]
-imagesListCities : byte[][]
-drawerFragment : FragmentDrawer
-journey : WitJourney
-dbAdapter : DbAdapter
#onCreate(savedInstanceState : Bundle) : void
#onStart() : void
+onDrawerItemSelected(view : View, position : int) : v...
+onCreateOptionsMenu(menu : Menu) : boolean
+onOptionsItemSelected(item : MenuItem) : boolean
-displayView(position : int) : void
-startSettingPage() : void
```

It display the details of a trip.

WitFacebookLogin:



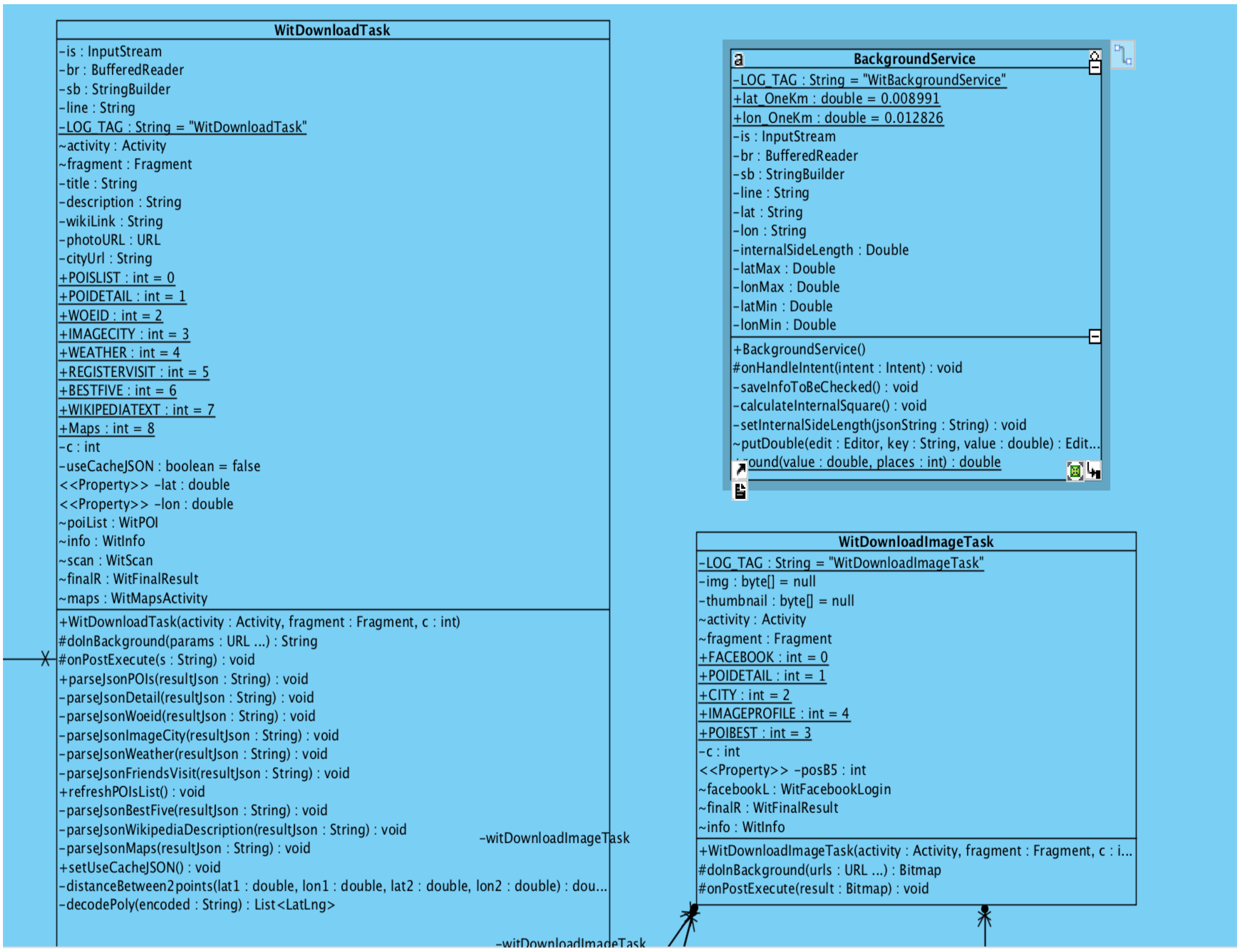
WitFacebookLogin

```
-LOG_TAG : String = "WitFacebokLogin"
-mButtonLogin : LoginButton
-mainImage : ImageView
-circularImageView : CircularImageView
-name : String
-surname : String
-id : long
-img : byte[] = null
-cursor : Cursor
-mToolbar : Toolbar
-profile : Profile
-mTextDetails : TextView
-mCallbackManager : CallbackManager
-callbackManager : CallbackManager
-token : AccessToken
-mProfileTracker : ProfileTracker
-mFacebookCallback : FacebookCallback<LoginResult>
-loginB : ImageButton
-logoutB : ImageButton
-dbAdapter : DbAdapter
-drawerFragment : FragmentDrawer
~witDownloadImageTask : WitDownloadImageTask
```

```
#onCreate(savedInstanceState : Bundle) : void
-login() : void
-startDownloadImage(photoURL : URL) : void
-logout() : void
#onStart() : void
#onStop() : void
#onResume() : void
+onDrawerItemSelected(view : View, position : int) : void
+onCreateOptionsMenu(menu : Menu) : boolean
+onOptionsItemSelected(item : MenuItem) : boolean
+onActivityResult(requestCode : int, resultCode : int, data : Intent) : void
-constructWelcomeMessage(profile : Profile) : String
+setImage(result : Bitmap, img : byte []) : void
-saveUser(id : long, name : String, surname : String, img : byte []) : void
-logoutUser(id : long) : void
-loadProfileImage() : void
-displayView(position : int) : void
-startSettingPage() : void
```

This activity manages the login/logout to Facebook.

WitdownloadTask, WitdownloadImageTask, BackroundService



WitDownloadTask

```
-is : InputStream
-br : BufferedReader
-sb : StringBuilder
-line : String
-LOG_TAG : String = "WitDownloadTask"
~activity : Activity
~fragment : Fragment
-title : String
-description : String
-wikiLink : String
-photoURL : URL
-cityUrl : String
+POISLIST : int = 0
+POIDETAIL : int = 1
+WOEID : int = 2
+IMAGECITY : int = 3
+WEATHER : int = 4
+REGISTERVISIT : int = 5
+BESTFIVE : int = 6
+WIKIPEDIATEXT : int = 7
+Maps : int = 8
-c : int
-useCacheJSON : boolean = false
<<Property>> -lat : double
<<Property>> -lon : double
~poiList : WitPOI
~info : WitInfo
~scan : WitScan
~finalR : WitFinalResult
~maps : WitMapsActivity
```

```
+WitDownloadTask(activity : Activity, fragment : Fragment, c : int)
#doInBackground(params : URL ...) : String
#onPostExecute(s : String) : void
+parseJsonPOIs(resultJson : String) : void
-parseJsonDetail(resultJson : String) : void
-parseJsonWoeid(resultJson : String) : void
-parseJsonImageCity(resultJson : String) : void
-parseJsonWeather(resultJson : String) : void
-parseJsonFriendsVisit(resultJson : String) : void
+refreshPOIsList() : void
-parseJsonBestFive(resultJson : String) : void
-parseJsonWikipediaDescription(resultJson : String) : void
-parseJsonMaps(resultJson : String) : void
+setUseCacheJSON() : void
-distanceBetween2points(lat1 : double, lon1 : double, lat2 : double, lon2 : double) : dou...
-decodePoly(encoded : String) : List<LatLng>
```

BackgroundService

```
-LOG_TAG : String = "WitBackgroundService"
+lat_OneKm : double = 0.008991
+lon_OneKm : double = 0.012826
-is : InputStream
-br : BufferedReader
-sb : StringBuilder
-line : String
-lat : String
-lon : String
-internalSideLength : Double
-latMax : Double
-lonMax : Double
-latMin : Double
-lonMin : Double
```

```
+BackgroundService()
#onHandleIntent(intent : Intent) : void
-saveInfoToBeChecked() : void
-calculateInternalSquare() : void
-setInternalSideLength(jsonString : String) : void
~putDouble(edit : Editor, key : String, value : double) : Edit...
+round(value : double, places : int) : double
```

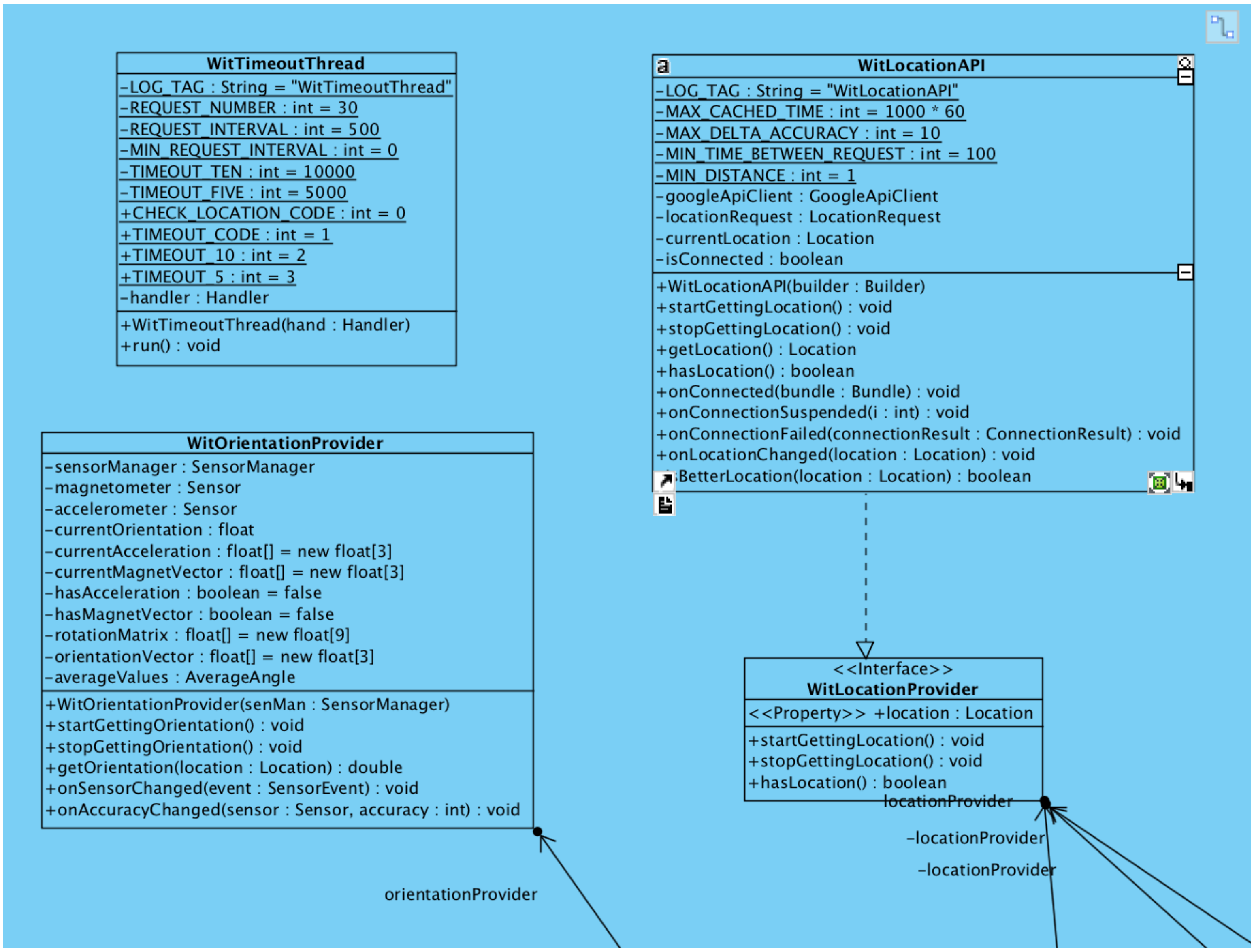
WitDownloadImageTask

```
-LOG_TAG : String = "WitDownloadImageTask"
-img : byte[] = null
-thumbnail : byte[] = null
~activity : Activity
~fragment : Fragment
+FACEBOOK : int = 0
+POIDETAIL : int = 1
+CITY : int = 2
+IMAGEPROFILE : int = 4
+POIBEST : int = 3
-c : int
<<Property>> -posB5 : int
~facebookL : WitFacebookLogin
~finalR : WitFinalResult
~info : WitInfo
```

```
+WitDownloadImageTask(activity : Activity, fragment : Fragment, c : i...
#doInBackground(urls : URL ...) : Bitmap
#onPostExecute(result : Bitmap) : void
```

These classes opens a new thread in background to manage all the download.

WitlocationApi, WitOrientation Provider:


WitTimeoutThread
-LOG_TAG : String = "WitTimeoutThread"
-REQUEST_NUMBER : int = 30
-REQUEST_INTERVAL : int = 500
-MIN_REQUEST_INTERVAL : int = 0
-TIMEOUT_TEN : int = 10000
-TIMEOUT_FIVE : int = 5000
+CHECK_LOCATION_CODE : int = 0
+TIMEOUT_CODE : int = 1
+TIMEOUT_10 : int = 2
+TIMEOUT_5 : int = 3
-handler : Handler
+WitTimeoutThread(hand : Handler)
+run() : void
WitLocationAPI
-LOG_TAG : String = "WitLocationAPI"
-MAX_CACHED_TIME : int = 1000 * 60
-MAX_DELTA_ACCURACY : int = 10
-MIN_TIME_BETWEEN_REQUEST : int = 100
-MIN_DISTANCE : int = 1
-googleApiClient : GoogleApiClient
-locationRequest : LocationRequest
-currentLocation : Location
-isConnected : boolean
+WitLocationAPI(builder : Builder)
+startGettingLocation() : void
+stopGettingLocation() : void
+getLocation() : Location
+hasLocation() : boolean
+onConnected(bundle : Bundle) : void
+onConnectionSuspended(i : int) : void
+onConnectionFailed(connectionResult : ConnectionResult) : void
+onLocationChanged(location : Location) : void
BetterLocation(location : Location) : boolean
WitOrientationProvider
-sensorManager : SensorManager
-magnetometer : Sensor
-accelerometer : Sensor
-currentOrientation : float
-currentAcceleration : float[] = new float[3]
-currentMagnetVector : float[] = new float[3]
-hasAcceleration : boolean = false
-hasMagnetVector : boolean = false
-rotationMatrix : float[] = new float[9]
-orientationVector : float[] = new float[3]
-averageValues : AverageAngle
+WitOrientationProvider(senMan : SensorManager)
+startGettingOrientation() : void
+stopGettingOrientation() : void
+getOrientation(location : Location) : double
+onSensorChanged(event : SensorEvent) : void
+onAccuracyChanged(sensor : Sensor, accuracy : int) : void
<<Interface>>
WitLocationProvider
<<Property>> +location : Location
+startGettingLocation() : void
+stopGettingLocation() : void
+hasLocation() : boolean
locationProvider
-locationProvider
-locationProvider
orientationProvider

These classes interface with the device’s sensor: the gps and the orientation sensor.