Group 05

GLocate
Software Architecture Document

Version 2.0

Revision History

Date	Version	Description	Author
21/11/2019	1.0	First version	Nguyễn Hoàng Tân
30/11/2019	1.1	- Architectural goals and constraints are filled in more completely - The logical view is extended to fit more closely with MVC	Hồ Minh Trí Nguyễn Hoàng Tân
09/12/2019	2.0	Deployment and Implementation views are added	Hồ Minh Trí

Table of Contents

1. Introduction

2. Architectural Goals and Constraints

3. Use-Case Model

4. Logical View
   - 4.1 Component: User management
   - 4.2 Component: Group management
   - 4.3 Component: Map management
   - 4.4 Component: View
   - 4.5 Component: Controller

5. Deployment View

6. Implementation View

Software Architecture Document

1. Introduction

GLocate is a mobile application for user to share their locations in particular groups and interact with other locations through a map view. It also has a messenger system for better communication between users.

This document elaborates the software architecture document for the system of GLocate. The system architecture is abstracted into many views and components which are explained in detail.

1.1 Purpose

This document provides a comprehensive architectural overview of the system, using a number of different architectural views to depict different aspects of the system. It is intended to capture and convey the significant architectural decisions which have been made on the system.

This document elaborates the architecture of the system through logical, deployment, and implementation view. Only static behavior of the system is described in this document. All the required diagrams and their descriptions are available in this document.

1.2 Scope

The software architecture document applies to each static aspect of the system. The document discusses the class diagrams, package diagrams and other static architecture designs.

1.3 Definitions, Acronyms, and Abbreviations

OOP - Object-Oriented programming
MVC - Model view controller architecture

1.4 References

MVC architecture: https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller

1.5 Overview

The report will present the detailed analysis of the architecture of the GLocate system. The next section covers the architectural goals and constraints. The later sections cover the specific details of the 3 main view (logical view, deployment view, and implementation view) of the system.

2. Architectural Goals and Constraints

2.1 Device requirement

Users with a mobile device that run a modern version of Android or iOS should be able to access GLocate after installing the application on their device

2.2 Security

The information that GLocate contains – the real time locations of users – can be very dangerous if it falls into the wrong hands. Therefore, this information should be kept secure

2.3 Reliability / Performance

- The app should be able to support at least 10 different groups at the same time, with at least 20 users per group.
- There should be at most 5 seconds of lag between when a user moves and when the location of that user is updated in the app.
- The locations of the users should have an error of at most 50 meters.
- The app should work properly in a region with decent GPS and Internet signals.

- The system should be subjected to several testing operations (Unit test, integration testing, system testing) before being deployed in order to make sure that the system is reliable.

2.4 Portability and reuse

- The app should be designed so that it can be extended later, for example to the web. This can be accomplished by using a flexible framework such as React Native.
- To maintain reusability, all the functionalities should be well structured and layered. A good method would be to follow best practices of RUP and adhere to OOP and MVC standards.

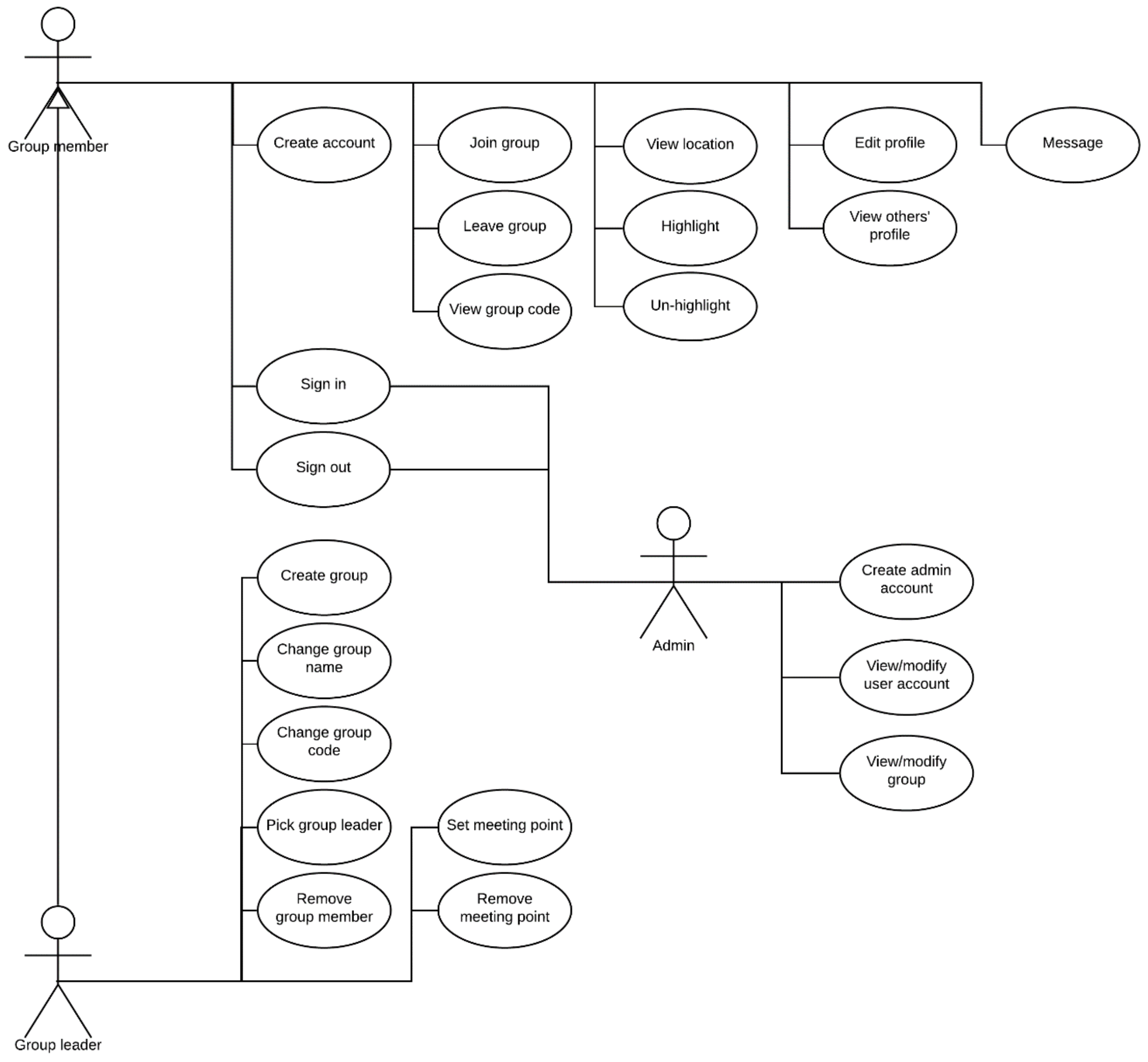
2.5 Development tools:

- GitHub: https://github.com/JG54264/TNT; repository of source code and project artifacts
- Trello: https://trello.com/b/cCezi5E5/apcs1-group05; Kanban board tracking progress of project
- Slack: 17apcs-clc-intro2se.slack.com, channel: apcs1-group05; communications channel
- Facebook: https://www.facebook.com/groups/392715311674005/; communications channel
- Moodle: https://courses.fit.hcmus.edu.vn/ctdb/course/view.php?id=522; place to submit project artifacts and receive project assignments
- Lucidchart: https://www.lucidchart.com/; diagram software used to draw use case model
- draw.io: https://www.draw.io/; diagram software used to draw software architecture and class diagrams
- React Native: https://facebook.github.io/react-native/; framework to develop cross-platform apps
- Firebase: https://firebase.google.com/; mobile application development platform

2.6 Schedule:

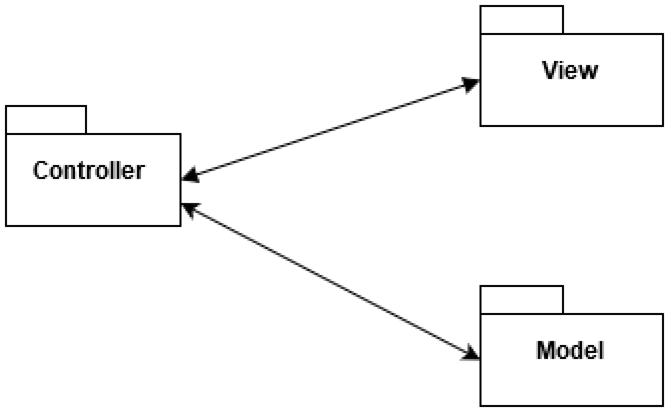
- The app must be finished by the end of the first semester of the 2019-2020 school year.

3. Use-Case Model



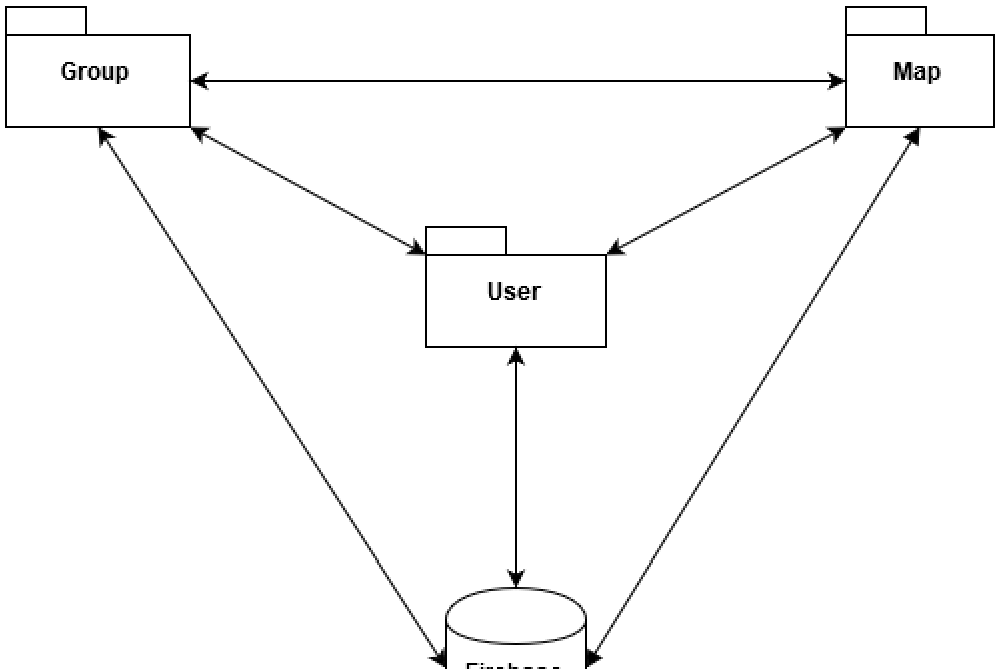
4. Logical View

Our app follows the Model-View-Controller (MVC) architecture so at the most basic level, it can be divided into three components:



- Model: the central components of the app, contains the data and methods that operate on the data.
- View: the representation of the data to the user, along with interfaces such as buttons that allow the user to interact with the app.
- Controller: accepts inputs from the View component and converts to commands for the Model component; accepts data from the Model component to supply to the View component.

  The Model component can be further divided into:

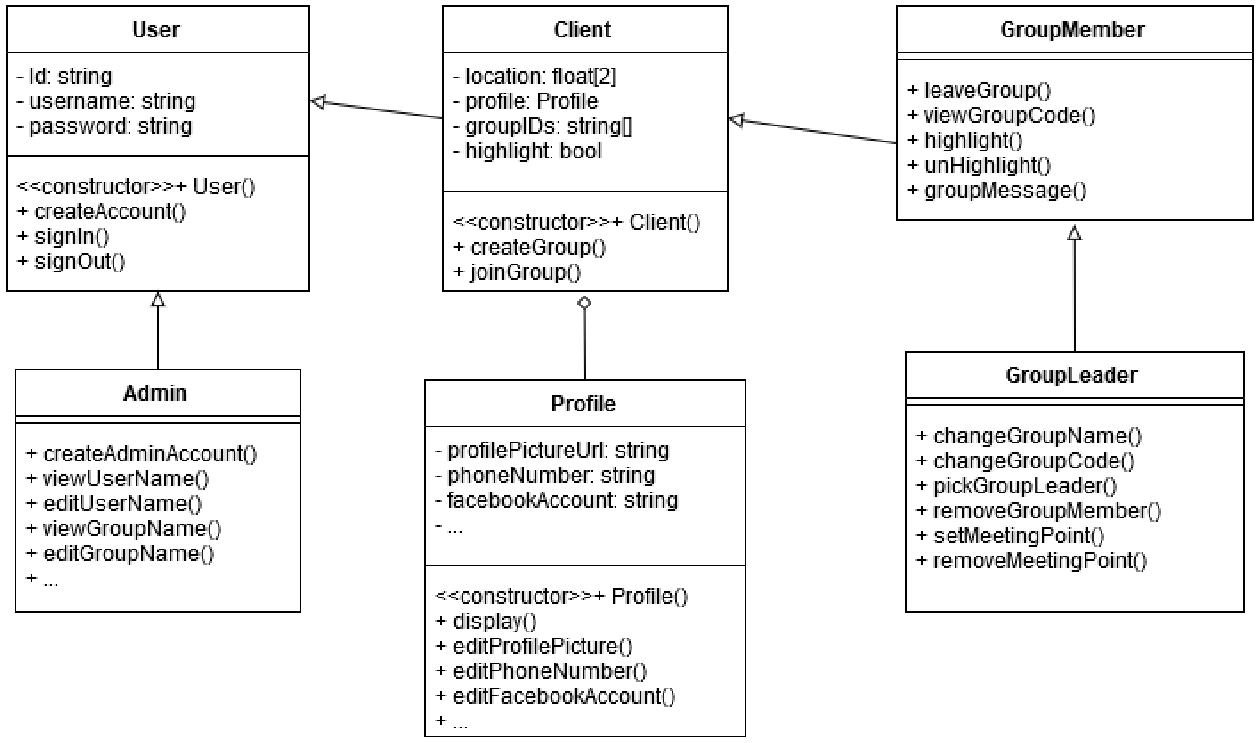


- The components that are local to the device, which are: User management, Group management, Map management. These components communicate with each other using ordinary function calls
- The remote Firebase database. The database is connected to the local components over the Internet using protocols such as HTTP.

- **Language**: except for the remote Firebase database, which is configured using the interface of Firebase, all the other components are local to the device and is implemented using React Native. This means that they are mostly programmed in Javascript, although some native features might be programmed in Java or Swift.

4.1 Component: User management

This component contains data involving the users of the system. It also provides functionalities to manage the users, such as: user sign in and sign out, profile viewing and editing, etc.

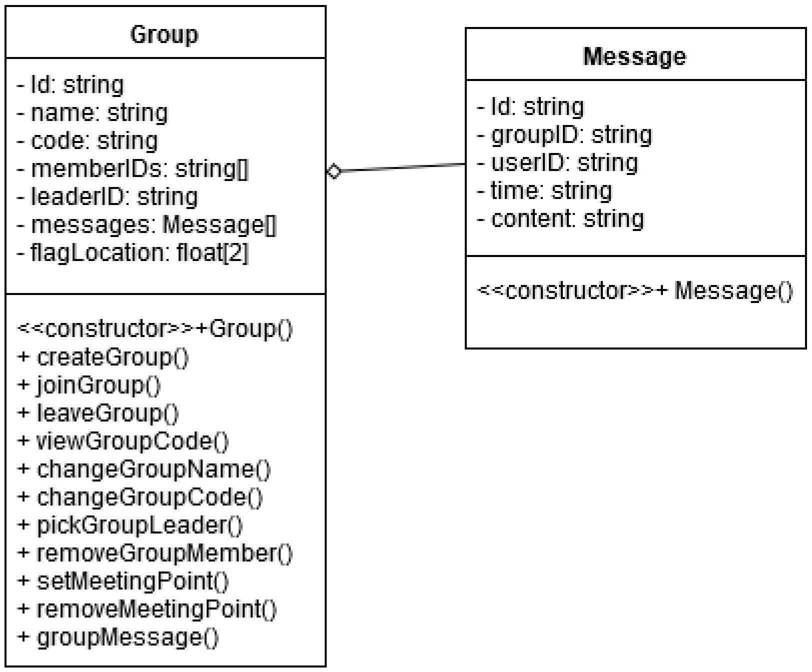


Explanations of the classes:

- User: the most basic user class, contains information to support sign in / sign out. Has two subclasses: Client and Admin.
- Client: a standard user that uses the location tracking feature of the app (as opposed to an admin). Each client object has a reference to the Profile object of that client. Each client is in a number of groups and depending on whether they are a member or a leader of the group, the appropriate functionality is supported by the GroupMember or GroupLeader class.
- Profile: contains the profile details of a user and the functionalities to manage these details.
- GroupMember: contains the functionalities that a group member can do.
- GroupLeader: subclass of GroupMember, contains the additional functionalities that a group leader can do.
- Admin: subclass of User, contains functionalities to support the duties of an admin, such as: view and modify user accounts, view and modify groups, etc.

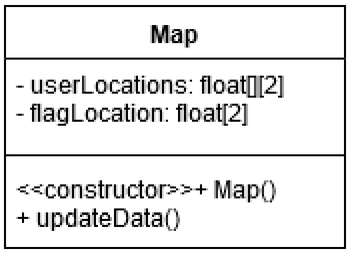
4.2 Component: Group management

This component contains data involving the groups of the system and the members of them. Some functionalities provided include: adding and removal of members from group, changing of group name and group code, etc. Each Group object contains an array of Message objects, which are the messages posted by the members of that group to the group's message thread.



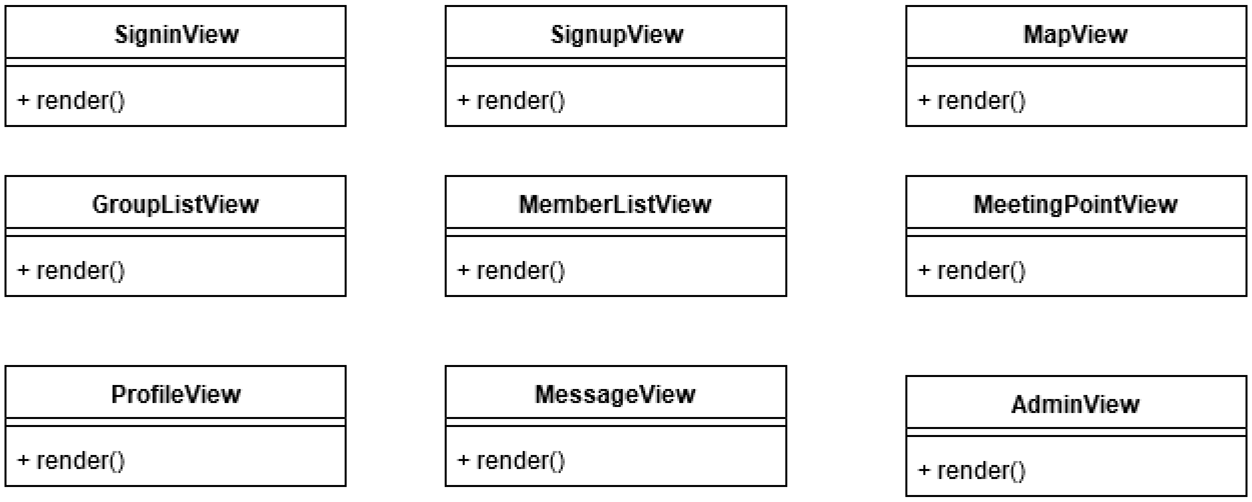
4.3 Component: Map management

This component manages the data involving the real time locations of the users, in order to display on the map.



4.4 Component: View

This component contains the classes related to the front-end of the app. There is roughly one class for each screen of the app. Each class has one render() function to return the view that is displayed to the user.

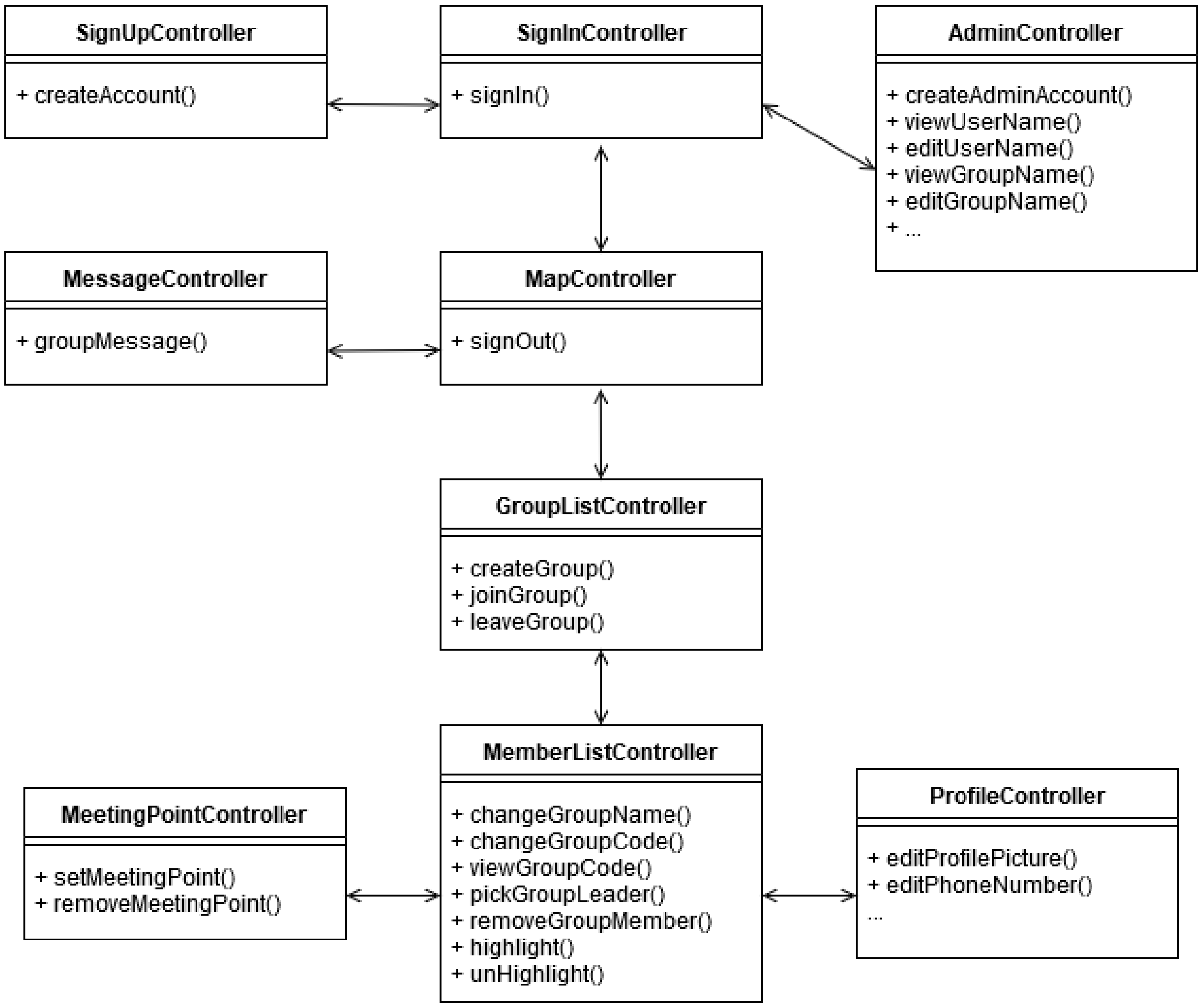


4.5 Component: Controller

This component contains the classes that bridge between the Model component and the View component. There is also roughly one class for each screen of the app.

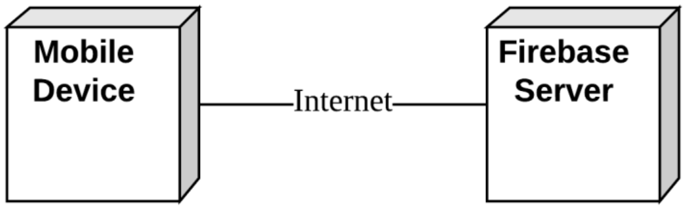
Each Controller class has an association with the corresponding View class. It also has associations with the Model classes that has functions with the same name as it.

For example: AdminController has an association with AdminView. And because createAdminAccount(), viewUserName(), editUserName(),… are in the class Admin from the Model component, AdminController has an association with class Admin.



5. Deployment View

Below is the Deployment View of our app.



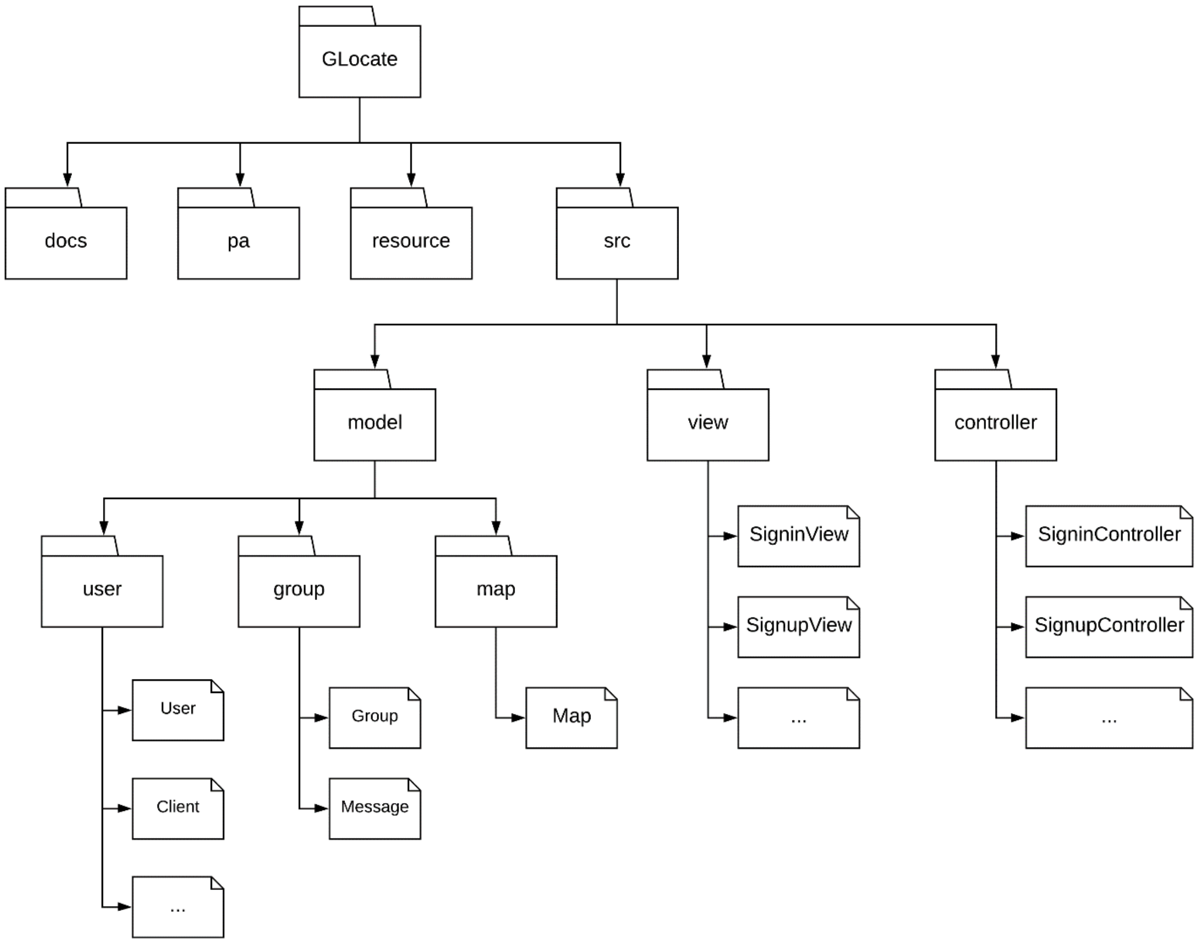
The Deployment View consists of two nodes:

- Mobile device:
  - The device on which the app is used.
  - The device should be running a modern version of either **iOS** or **Android**.
  - The device should be able to connect to the Internet and to GPS.
  - Most of the app's processing is run directly on the device.

- Firebase server:
    - A server containing the database of the app.
    - The server is provided and maintained by the Firebase platform.
    - The server is connected to the mobile devices over the Internet, using protocols such as HTTP.
    - The server does not run any code but only accept read and write requests on the database from the mobile devices and carry out these requests.

6. Implementation View

Below is the Implementation View of our app, with emphasis on the src directory:



The root directory GLocate consists of four subdirectories:

- docs: Used to store documentation, including Vision Document, Software Development Plan, etc.
- pa: Used to store the submissions, with one subdirectory for each PA.
- resource: Used to store the resources of the app, such as images.
- src: Used to store the source code of the app. Because our app follows the MVC architecture, the source code files can be easily divided into three subdirectories, one for each component of the MVC architecture:

- model: Used to store the classes that contain the data of the app and methods that operate on the data. Each class is put into one file. To make the directory structure more organized, we group the related class files into subdirectories. There are three subdirectories:
    - user: Contains classes involving the users of the app, such as User, Client, etc.
    - group: Contains classes involving the groups of the app, which are Group and Message.
    - map: Contains the Map class, which manages the data displayed on the map.

- view: Used to store the classes related to the front-end of the app. There is roughly one class for each screen of the app and each class is put in one file. Examples are SigninView, SignupView, etc.

- controller: Used to store the classes that bridge between the classes in the model and the view directories. There is also roughly one class for each screen of the app and each class is put in one file. Examples are SigninController, SignupController, etc.