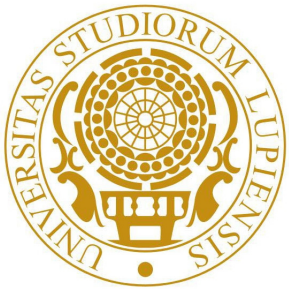
UNIVERSITÀ DEL SALENTO



Facoltà di Ingegneria

Corso di Laurea Magistrale in Computer Engineering

Project of

SOFTWARE ENGINEERING

Professor: **Luca Mainetti**

Students:
Vincenzo Spagnolo
Salvatore Corvaglia

Academic year 2020/2021

Index

1. Description of the System Requirements

A city Major wants to develop a MyAlert system that will allow citizens to report alarms or dangers occurring in the city land and it includes a web app and a mobile app.

The web app provides specific functionalities based on the user that is logged in the system as Manager or as Agent. The mobile app provides further functionalities for the user that is logged in as Agent or as Citizen.

2. Requirements Analysis

The users that interact with the software requested by the city Mayor are:

1. Manager
2. Agent
3. Citizen

1) Manager: he/she manages the alarm/danger schema. The alarms/dangers supported by the system are "car crash", "fire", "flood", "brawl", "illegal dump". Other types of alarms/dangers could by dynamically added by the security manager without the need of modifying the MyAlert database or the MyAlert source code. The security manager creates one or more security agents. The security manager approves the warnings of alert/danger coming from registered citizens and forward the warnings to the security agents for their intervention. The security manager can choose among different forwarding policies: broadcast (to all security agents), automatic selection (to a subset of security agents on the basis of their more recent known GPS positions), manual selection (to a subset of security agents manually selected from a list). The security manager uses the web interface of MyAlert.

2) Agent: he/she logs into MyAlert providing its GPS position through the mobile interface. He/she receives form MyAlert a push notification for any new alarm/danger warning. He/she receive form MyAlert a push notification for any new assigned intervention that he/she must confirm. At the end of an intervention, he/she writes an intervention note (start time of the intervention,

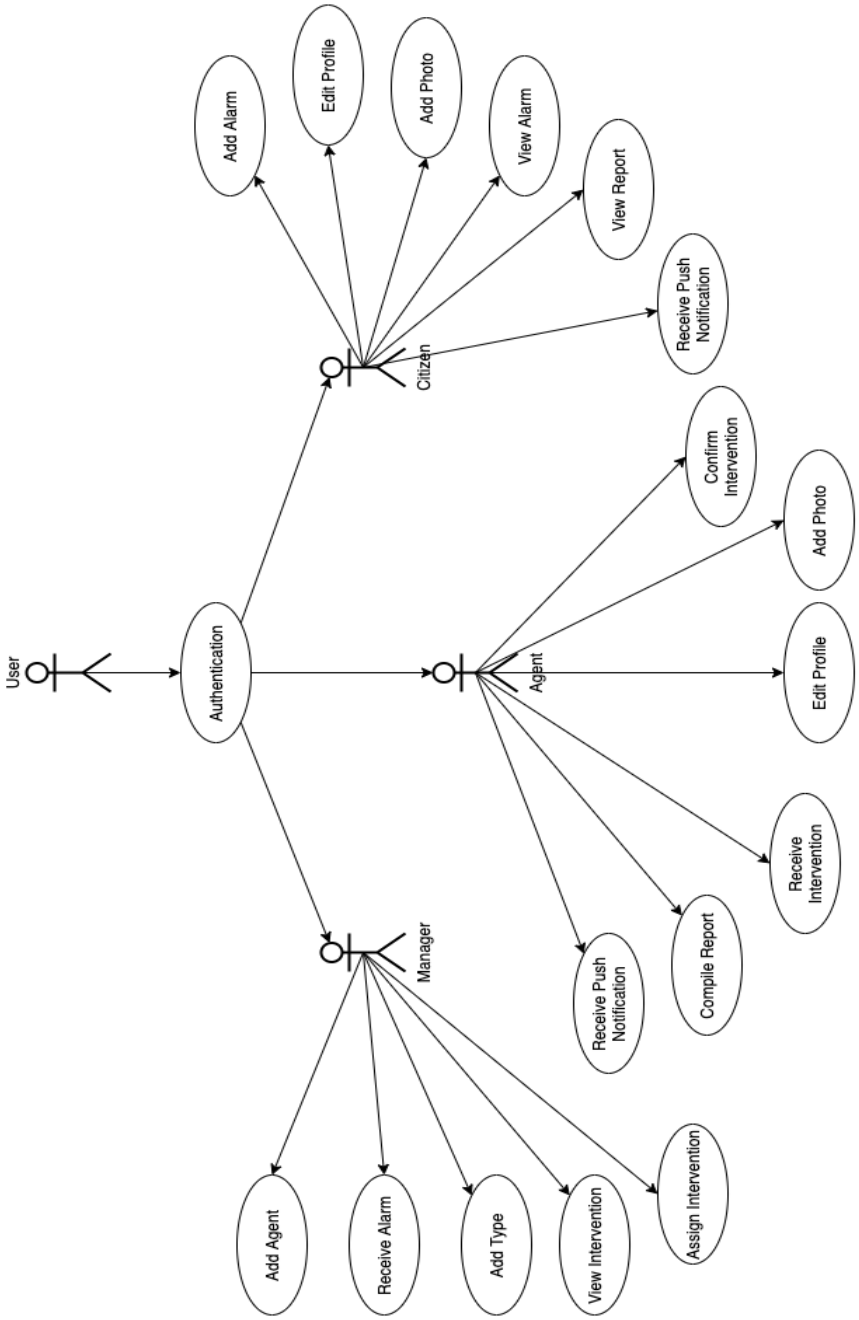
duration of the intervention, short textual description, one or more picture of the intervention, the GPS position for each picture) using the mobile interface. At the end of the working day, he/she completes the intervention note providing a detailed textual report using the web interface.

3) Citizen: he/she logs into the system providing its GPS position through the mobile interface. He/she informs about alarms/dangers compiling mainly automatically a "warning form" composed by his/her name, the timestamp of the alarm, one or more picture of the alarm/danger, the GPS position for each picture. He/she receives a push notification when a given own warning is taken in charge form a security agent. He/she receives a push notification when the intervention for a given own warning is closed by a security agent; in this case, he/she can read the intervention note. He/she interacts with MyAlert only through the mobile interface.

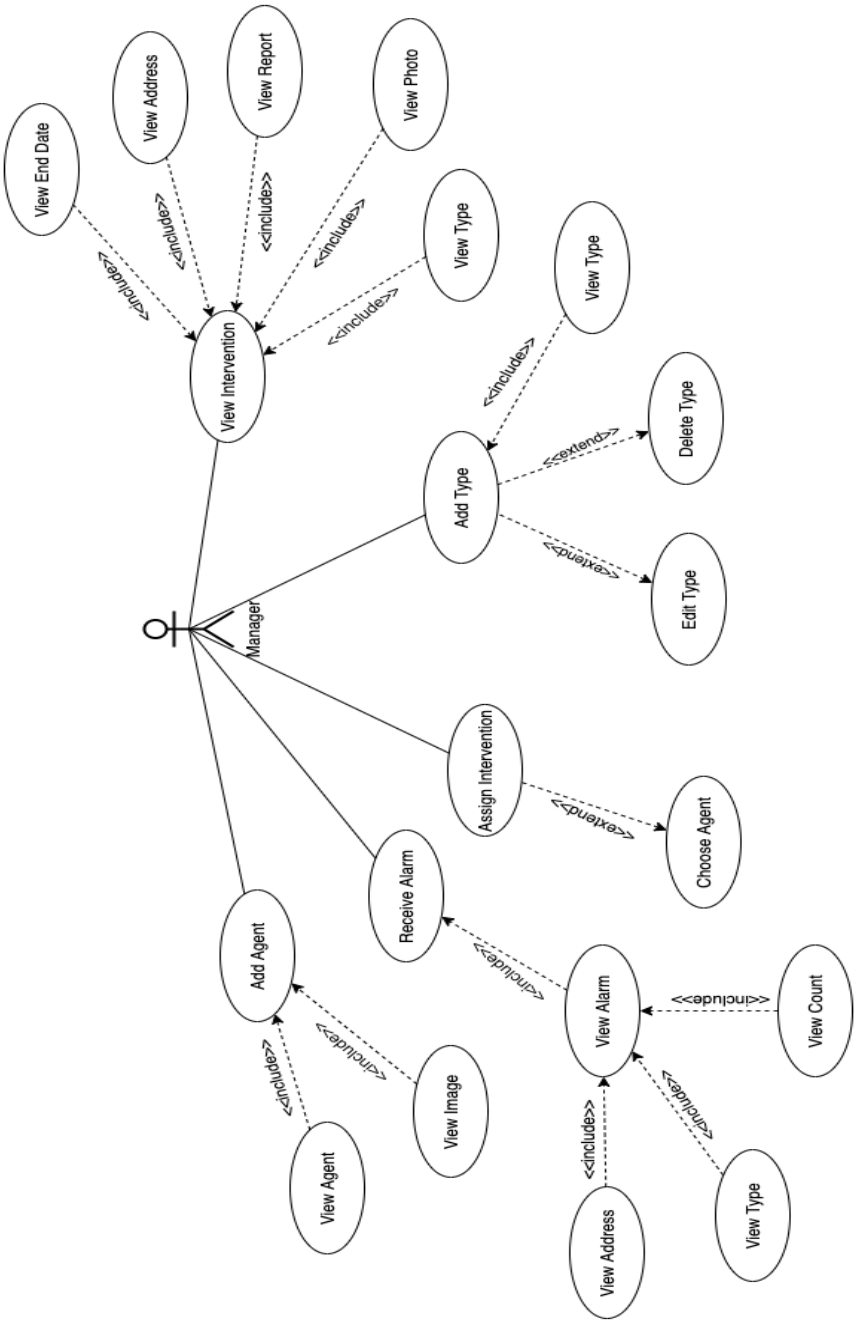
Furthermore, there is a Guest User that he/she interacts with the public part of MyAlert through the mobile interface to get information about the service.

3. Use Cases Description

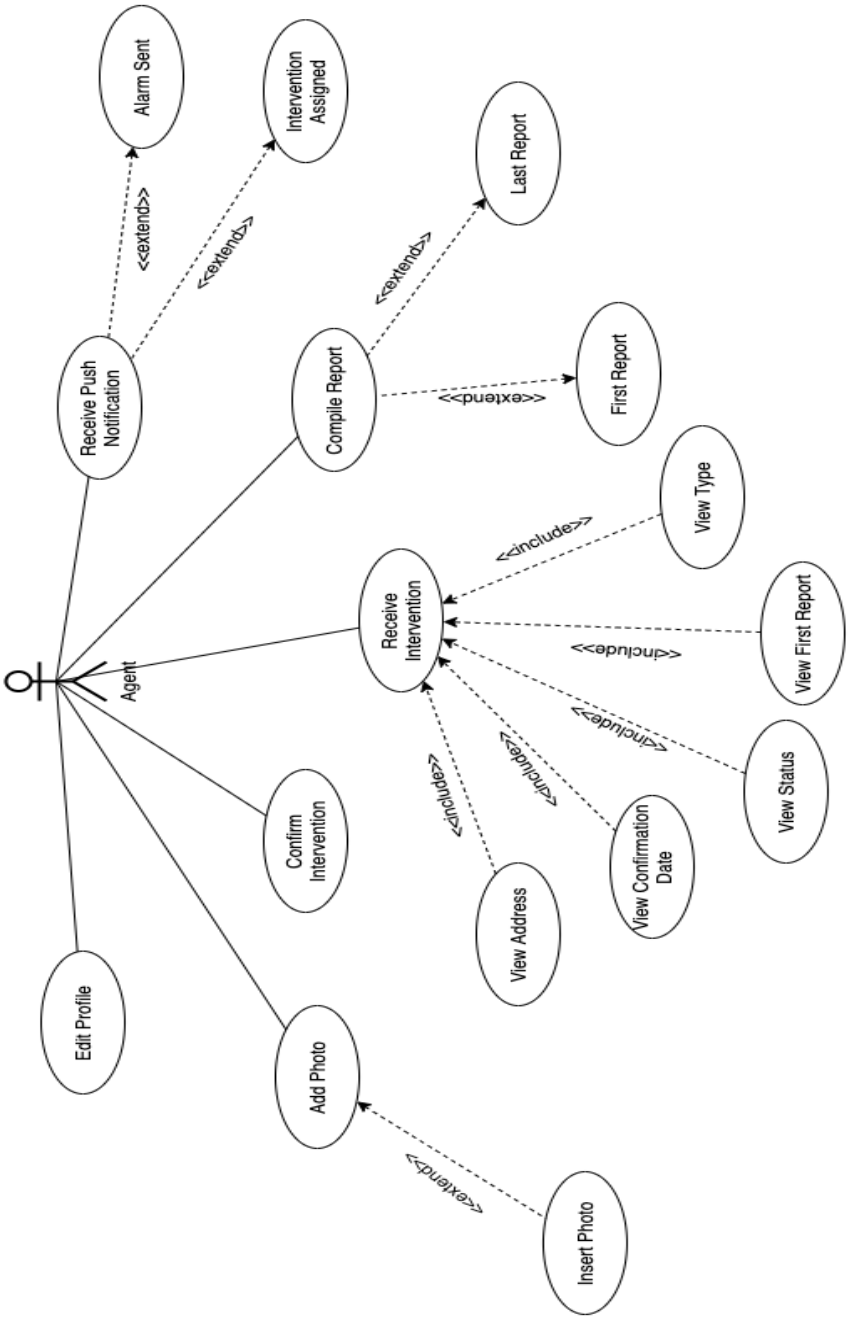
3.1 User



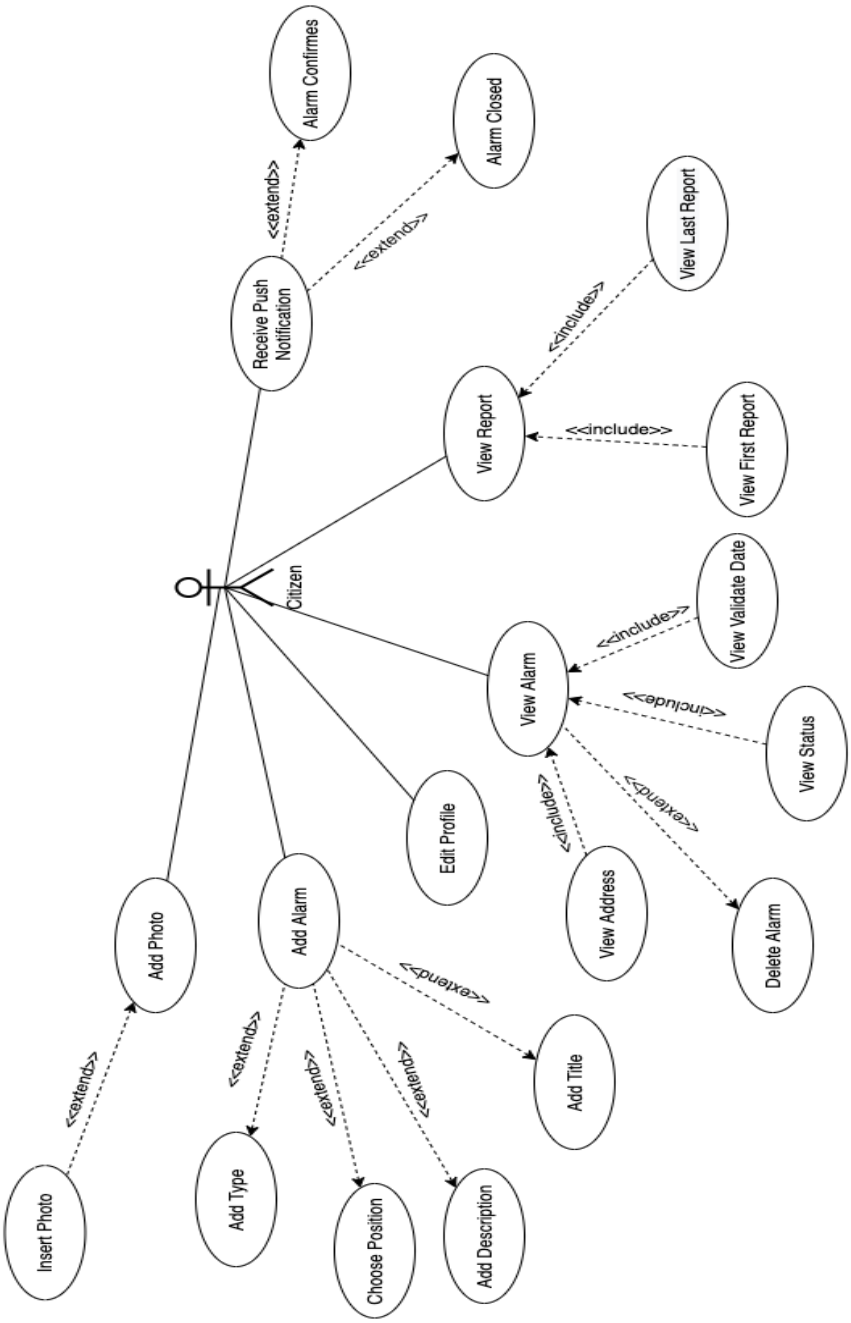
3.2 User Manager



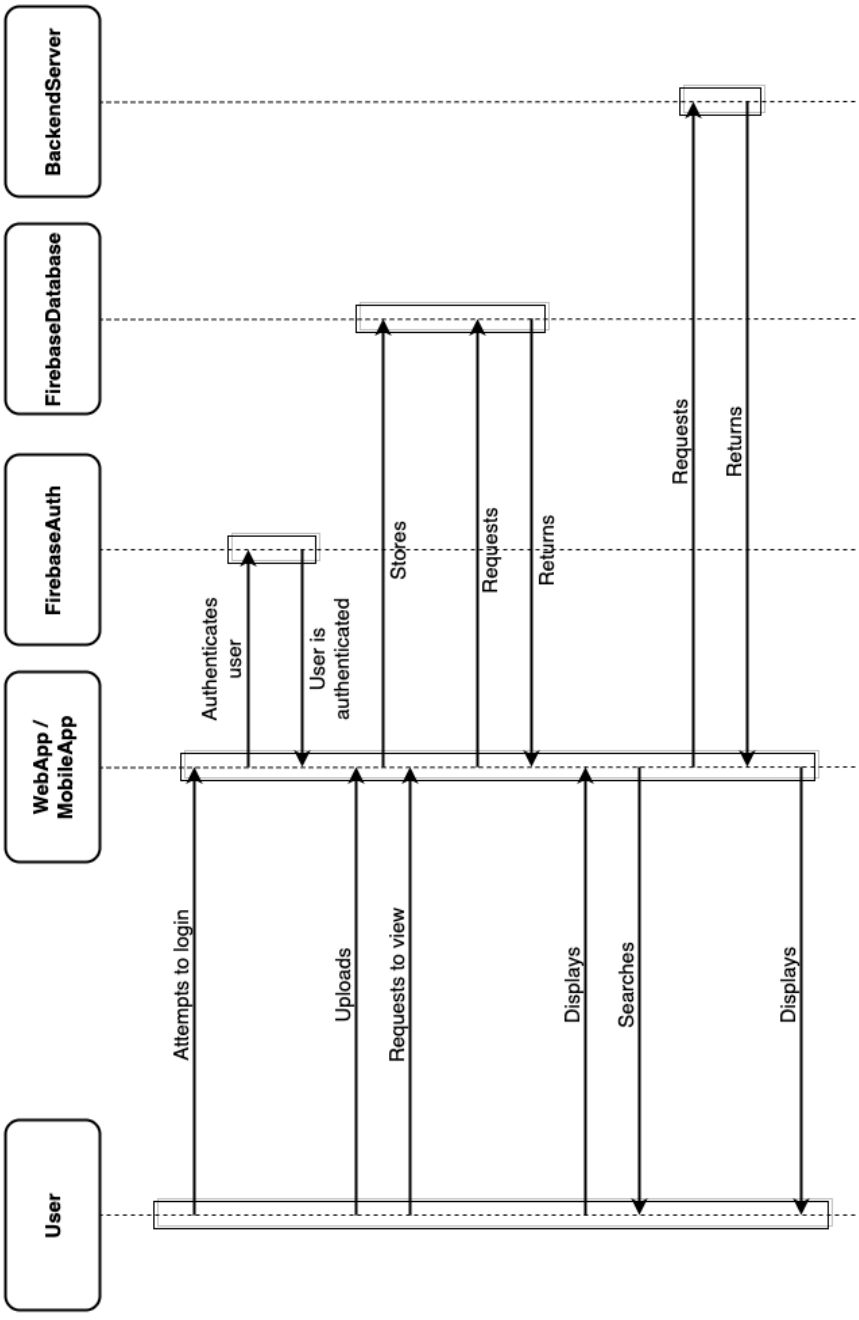
3.3 User Agent



3.4 User Citizen

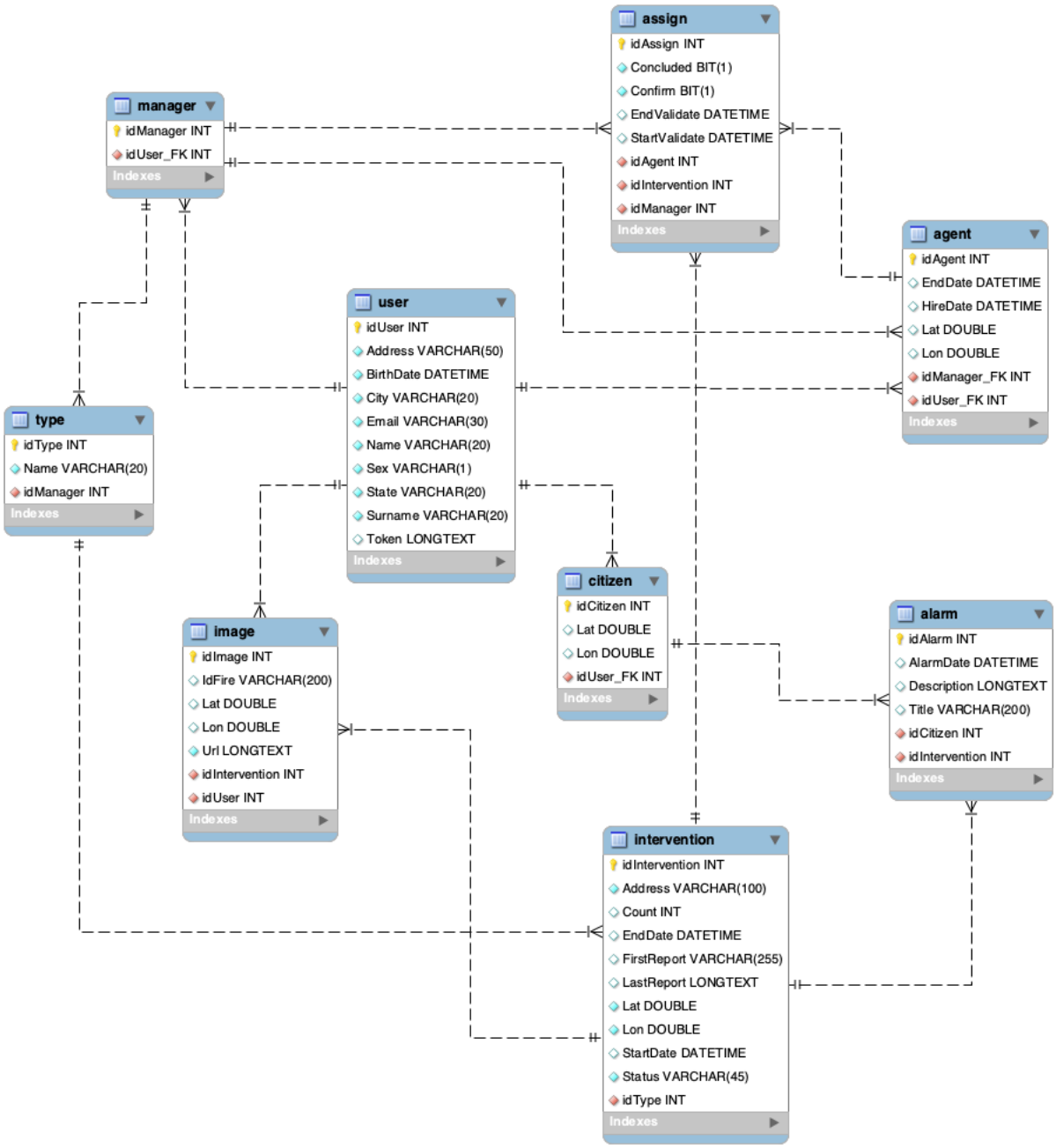


4. Sequence Diagram Authentication



5. Database Description

Entity-Relationship (ER) Diagram

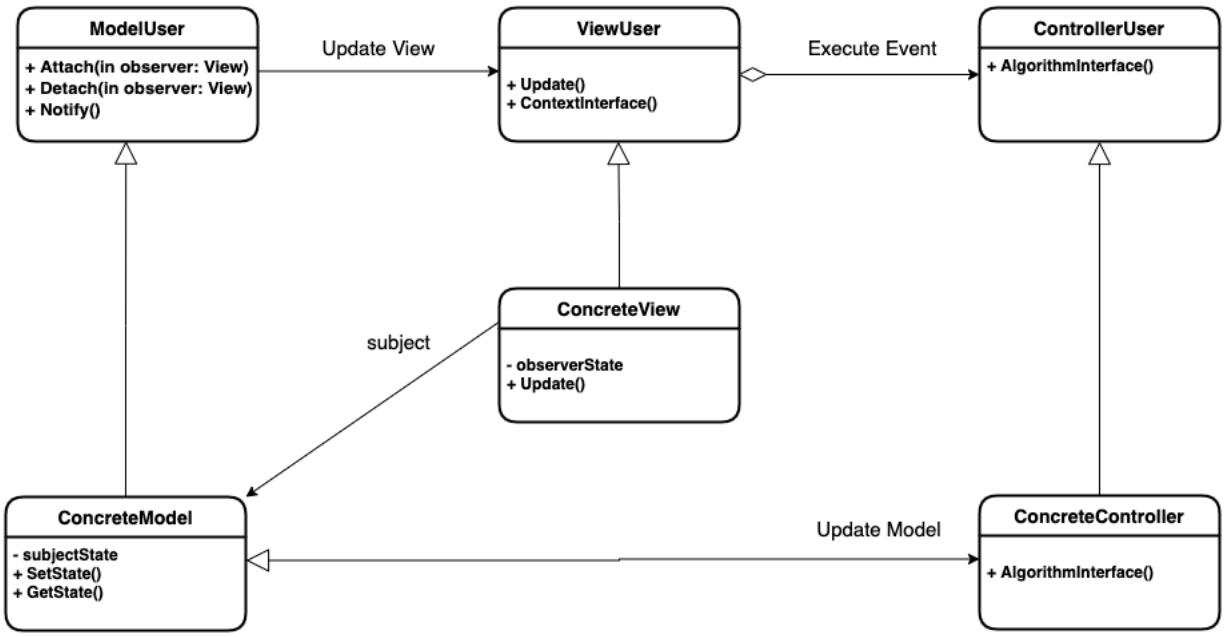


The table "user" represents the generic user who is logged in the system. The user foreign key is used to specify the type of the user, in fact: Manager, Agent and Citizen are taken from the table user. In the "type" table we find the type of alarm

that is reported by the citizen and that is created by the Manager, Manager who can obviously create, modify and delete the type of alarm from his own web interface and manually select the agents and also automatically with a maximum range of 150 mt. the position of the signaled alarm. In the “agent” and “citizen” table we have the latitude and longitude that are taken when the user logs in to allow their location. In the "alarm" table we find the title and description that citizens describe based on the alarm they report, as well as the date on which it was created. The "image" table contains the latitude and longitude of the images that are sent from the device of the citizen who obviously allowed his position. Also, you can view the image which is saved on Firebase Storage via url. In the "intervention" table we find, in addition to latitude and longitude already declared before, address of the related intervention, count which indicates the number of alarms of the same type received by citizens in a maximum range of 50 mt., start and end date which indicate respectively the beginning and the end of the intervention and finally status which represents the status of the intervention which can be of three types: signaled, assigned and closed. For last, the "assign" table indicates when the assignment is confirmed and declared concluded, the confirmation itself and the two reports.

6. Design Pattern

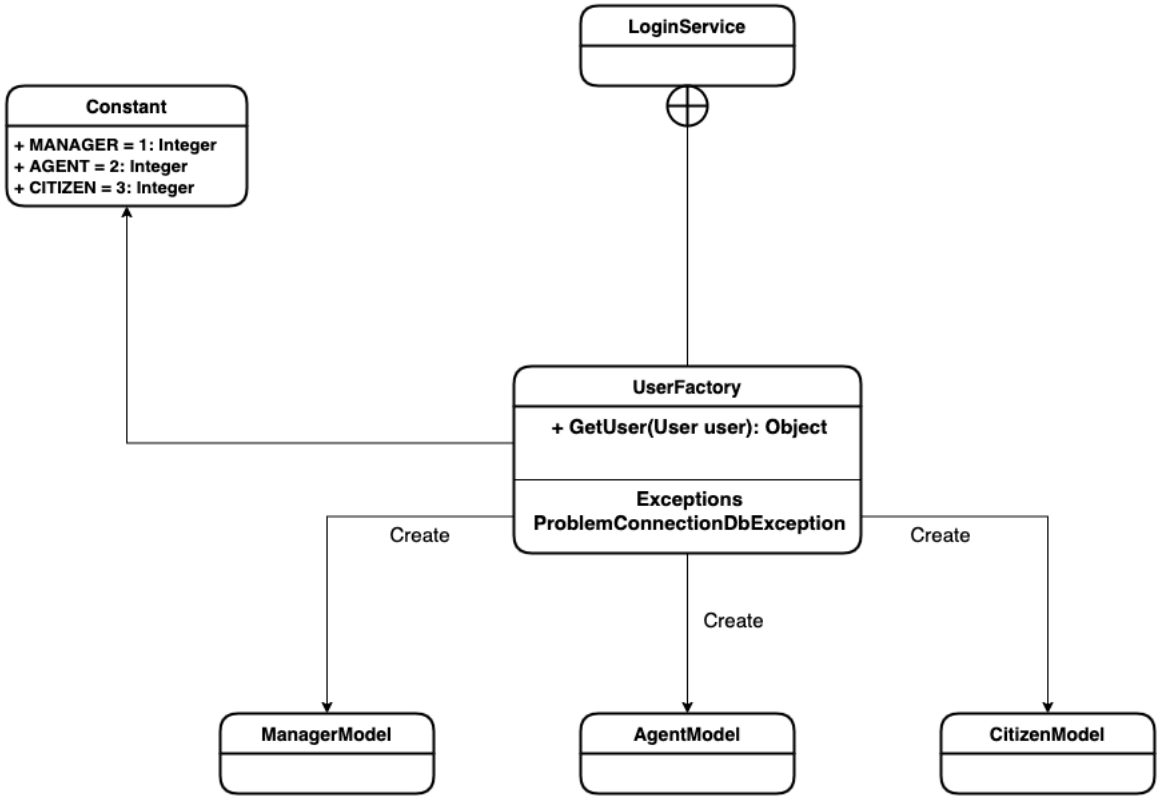
6.1 MVC



The Model View Controller (MVC) design pattern specifies that an application consist of a data model, presentation information, and control information.

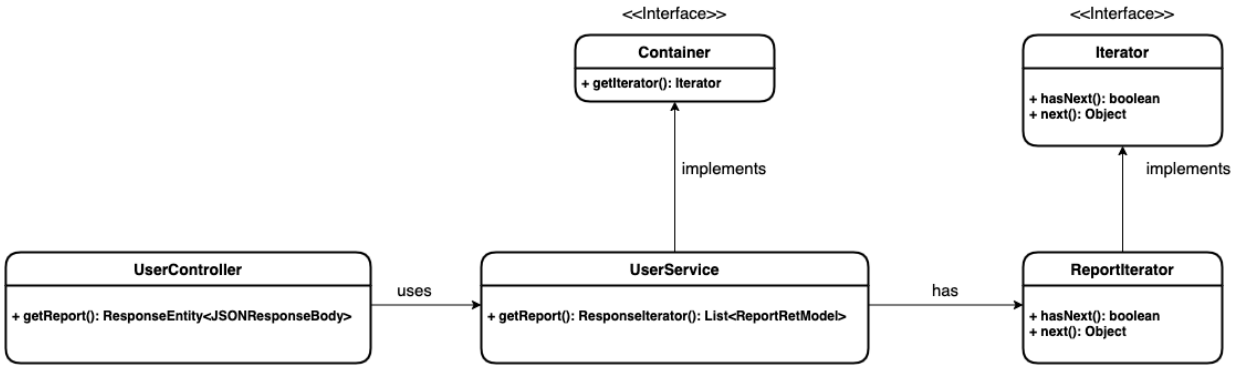
The pattern requires that each of these be separated into different objects. MVC is more of an architectural pattern, but not for complete application. MVC mostly relates to the UI / interaction layer of an application.

6.2 Abstract Factory



In class-based programming, the factory method pattern is a creational pattern that uses factory methods to deal with the problem of creating objects without having to specify the exact class of the object that will be created.

6.3 Iterator



Iterator pattern is a behavioral pattern and it is commonly used in Java and .Net programming environment. It is used to access the elements of a collection object in a sequential manner, without knowing the underlying representation.

7. Test Description

7.1 Jacoco

JaCoCo is an open-source toolkit for measuring and reporting Java code coverage. JaCoCo is distributed under the terms of the Eclipse Public License. It was developed as a replacement for EMMA, under the umbrella of the EclEmma plugin for Eclipse.

JaCoCo offers instructions, line and branch coverage.
In contrast to Atlassian Clover and OpenClover, which require instrumenting the source code, JaCoCo can instrument Java bytecode using two different approaches:
- like JCov on the fly while running the code with a Java agent (computational process with communicative and autonomy functions)
- like Cobertura and JCov prior to execution (offline)
And can be configured to store the collected data in a file, or send it via TCP. Files from multiple runs or code parts can be merged easily. Unlike Cobertura and EMMA it fully supports Java 7, Java 8, Java 9, Java 10 and Java 11.

- JCov is the tool which has been developed and used with Sun JDK (and later Oracle JDK) from the very beginning of Java: from the version 1.1. JCov is capable of measuring and reporting Java code coverage. JCov is distributed under the terms of the GNU Public License (version 2, with the Classpath Exception). JCov has become open-source as a part of OpenJDK code tools project in 2014.

Test Summary

75	0	0	2.217s	100%
tests	failures	ignored	duration	successful

Packages Classes

Package	Tests	Failures	Ignored	Duration	Success rate
it.myalert	1	0	0	0.276s	100%
it.myalert.restcontroller	29	0	0	1.334s	100%
it.myalert.services	45	0	0	0.607s	100%

MyAlarm

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
it.myalert.entities		34%		0%	124	250	224	420	70	196	0	9
it.myalert.restcontroller		56%		41%	59	114	132	326	14	56	0	10
it.myalert.servimplement		58%		30%	28	87	51	124	18	77	0	9
it.myalert.firebase		36%		100%	2	6	16	28	2	5	0	2
it.myalert.converter		95%		63%	11	42	2	172	0	27	0	9
it.myalert		37%		n/a	1	2	2	3	1	2	0	1
it.myalert.DTO		100%		n/a	0	134	0	203	0	134	0	10
it.myalert.configuration		100%		n/a	0	3	0	8	0	3	0	1
it.myalert.exeption		100%		n/a	0	9	0	18	0	9	0	9
Total	2.062 of 5.353	61%	201 of 276	27%	225	647	427	1.302	105	509	0	60

We used Jacoco to get a test summary of the overall code of the project, regarding the executed tests.

7.2 JUnit and Mockito

On the other hand, we could have computed the results using mathematical calculations. For example, the coverage is computed as follows:

Packages	Classes	Methods	Lines
it.myalert.entities	100% (9/9)	(70/196)	(224/420)
it.myalert.restcontroller	100% (10/10)	(14/56)	(132/326)
it.myalert.servimplement	100% (9/9)	(18/77)	(51/124)
it.myalert.firebase	100% (2/2)	(2/5)	(16/28)
it.myalert.converter	100% (9/9)	(0/27)	(2/172)
it.myalert.	100% (1/1)	(1/2)	(2/3)
it.myalert.DTO	100% (10/10)	(0/134)	(0/203)
it.myalert.configuration	100% (1/1)	(0/3)	(0/8)
it.myalert.exeption	100% (9/9)	(0/9)	(0/18)

We used JUnit, which is an open-source framework designed for the purpose of writing and running tests in the Java programming language. JUnit, originally written by Erich Gamma and Kent Beck, has been important in the evolution of test-driven development, which is part of a larger software design paradigm known as Extreme Programming (XP).

And we used also Mockito, which is a mocking framework, JAVA-based library that is used for effective unit testing of JAVA applications. Mockito is used to mock interfaces so that a dummy functionality can be added to a mock interface that can be used in unit testing.

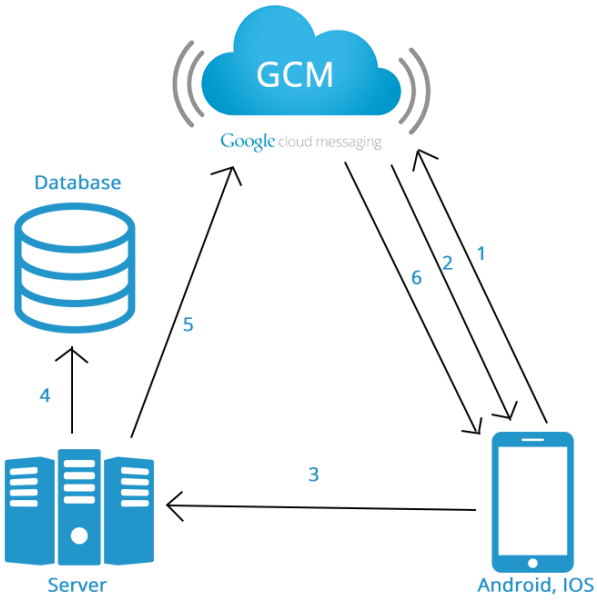
8. Used Tools

We used the following tools for the development of the project.

- Firebase
- Postman
- Spring Tool Suite (STS)
- IntelliJ IDEA
- WebStorm
- WampServer
- MySQL Server
- MySQL Workbench

8.1 Push Notification with FCM

For the management of push notifications, we used the FCM service (Firebase Cloud Messaging).



Firebase Cloud Messaging provides a reliable and battery-efficient connection between the server and devices that allows you to deliver and receive messages and notifications.

8.2 Firebase Authentication

We used Google's Firebase service in the Ionic and Angular Application to use Firebase Authentication service.

We created the login and registration components with which a user can register and log with email and password to be redirected to the reference dashboard.

9. Sprint Backlog

Week 1

Activity	07/01/21	08/01/21	09/01/21	10/01/21	11/01/21
Project structure, Initial settings	4				
Database SQL		2			
Set workspace, Generation entities			2		
Firebase Settings, Firebase services				2	
Authentication, Verification, Utils, Shared, Configuration CORS					5

Week 2

Activity	14/01/21	15/01/21	16/01/21	17/01/21	18/01/21
Firebase Login and Registration Authentication	4	2			
Generic BE Manager		3	1		
Generic BE Agent			1		
Generic BE Citizen				2	
BE: common utils					2

Week 3

Activity	21/01/21	22/01/21	23/01/21	24/01/21	25/01/21
DTO	3	3	2		
DAO				2	2
Services	4	3	3		
Service Implement		3	2		
Rest Controller				2	2

Week 4

Activity	28/01/21	29/01/21	30/01/21	31/01/21	01/02/21
Generic FE Manager	3				
Generic FE Agent		3	3		
Generic FE Citizen			1	2	
FE: common utils				4	3
Setting of Firebase Cloud Messaging (FCM)					4

Week 5

Activity	04/02/21	05/02/21	06/02/21	07/02/21	08/02/21
Mobile App: Agent Push Notification	3	3	2	2	
Mobile App: Citizen Push Notification		2			
Android Emulator in Android Studio			3	3	
FE/Mobile App: CSS/HTML styling			3	3	
Bug Fixing					3

Week 6

Activity	11/02/21	12/02/21	13/02/21	14/02/21	15/02/21
Testing: Mockito, Junit, Jacoco	2	2			
Documentation BE		2	3	3	
Documentation FE				2	
Documentation Mobile App					2
Last tests of the project & emulator					2

10. Burndown Chart