**Politecnico di Milano**

**5th School of Engineering**



PhoneGuardian

Design and Implementation of Mobile Applications

# **D**esign **D**ocument

**23th May 2015**

Contents

[**D**esign **D**ocument 1](file:///D:\Net%20downloads\Android%20SDK\workspace\Documentation\Design%20Document%20merged.docx#_Toc420679458)

[1. Purpose 4](#_Toc420679459)

[2. Brief introduction 4](#_Toc420679460)

[3. Architectural design 5](#_Toc420679461)

[3.1. Identifying sub-systems 5](#_Toc420679462)

[4. PERSISTENT DATA MANAGEMENT 7](#_Toc420679463)

[4.1. CONCEPTUAL DESIGN 7](#_Toc420679464)

[4.2. LOGICAL DESIGN 9](#_Toc420679465)

[5. Citizen app 10](#_Toc420679470)

[5.1. Class diagrams 10](#_Toc420679471)

[5.1.1. Activities class overview 10](#_Toc420679472)

[5.1.2. Register activity class diagram 11](#_Toc420679473)

[5.1.3. Add event activity class diagram 12](#_Toc420679474)

[5.1.4. Filter activity class diagram 13](#_Toc420679475)

[5.1.5. PG map activity class diagram 14](#_Toc420679476)

[5.1.6. Add comment activity class diagram 15](#_Toc420679477)

[5.1.7. View comment activity class diagram 16](#_Toc420679478)

[5.2. Activities 17](#_Toc420679481)

[6. Authority app 21](#_Toc420679482)

[6.1. Class diagrams 21](#_Toc420679483)

[6.1.1. Activities class overview 21](#_Toc420679484)

[6.1.2. Main activity class overview 22](#_Toc420679485)

[6.1.3. Register activity class overview 23](#_Toc420679486)

[6.1.4. Take job class overview 24](#_Toc420679487)

[6.1.5. Review job activity class diagram 25](#_Toc420679488)

[6.1.6. View job activity class diagram 26](#_Toc420679489)

[6.1.7. View job activity class diagram 27](#_Toc420679490)

[6.1.8. Highscores activity class diagram 28](#_Toc420679491)

[6.2. Activities 29](#_Toc420679494)

[7. Web Server 32](#_Toc420679495)

[8. Sequence diagrams 33](#_Toc420679496)

[8.1. Log In 33](#_Toc420679497)

[8.2. User Logs Out 34](#_Toc420679498)

[8.3. Sign Up 35](#_Toc420679499)

[8.4. Change calendars privacy 36](#_Toc420679500)

[8.5. Next week 37](#_Toc420679501)

[8.6. Accept Invitations 38](#_Toc420679502)

[8.7. Add Event 39](#_Toc420679503)

[8.8. View Events Details 39](#_Toc420679504)

[9. FINAL CONSIDERATIONS 40](#_Toc420679505)

[10. Used tools 41](#_Toc420679506)

[11. Working Hours 41](#_Toc420679507)

# Purpose

This document aims to describe the design and prototyping steps taken for “PhoneGuardian” software system assigned as a project of the “Design and Implementation of Mobile Applications” course at Politecnico di Milano.

# Brief introduction

***PhoneGuardian*** is a software system that consists of three parts: CitizenApp, AuthorityApp and a Web server between them. Main purpose of the system is giving real-time information about events that happened. This information can be seen either from a user of CitizenApp (regular user) or by an user of AuthorityApp (authority staff i.e. policeman, fireman or medical staff on the field).

***CitizenApp*** is an Android application used for reporting emergency events or searching among reported events by applying a filter. Main purpose of the application is to offer insight to dangerous zones of cities. User can also follow reported event and current development of the situation through comments posted by other users.

***AuthorityApp*** is an Android application used for receiving reports about events that happened i.e. the application works in real-time. The user of the application (usually authority staff) can accept a job they want to deal with and after checking situation they can write a review about it and notify other users.

***WebServer*** is a medium that connects *CitizenApp* and *AuthorityApp*. The server deals with data transfer and connects the applications to the database.

# Architectural design

## Identifying sub-systems

We decided to adopt a top-down approach at least at this point of the project. Maybe, once defined the sub-systems, we will adopt a bottom-up approach in order to create more reusable components.

So we think it is now necessary to decompose our system into other sub-systems, in order to make it easier to understand the issues that we found in implementing functionalities and to separate, logically, groups of functionalities and state clearer their interaction.

We separate our systems into these sub-systems:

* CitizenApp (Android application);
* AuthorityApp (Android application);
* Web Server (PHP server);
* Database (MySQL);

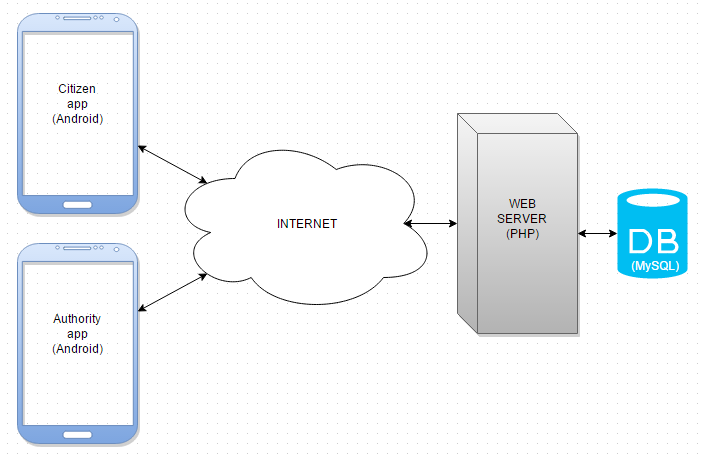


Figure . - System architecture

All the data and control logic are centralized on our Web Server. Each interaction coming from citizens or authority staff must be followed with server’s acknowledgment. Citizens and Authorities cannot communicate directly (although the server may deliver user’s phone number to authority staff, more info about it will be described later on). In that way we gain more control over our system.

# PERSISTENT DATA MANAGEMENT

Our data is stored into a relational database. Database design represented by Entity-Relationship Diagram can be found in the subsection below. Moreover, we will explain in details entities, relations and provide the description for specific parts of each design diagram.

## CONCEPTUAL DESIGN

Conceptual design allows us to start thinking about the data we want to store and about the relations between them.

The most important entities in our system are *User* and *Authority*. Regular user, after completing the procedure of signing up becomes the user of a system. Each *User* can create zero, one or many entities of type *Event*. Thus each *Event* is connected to one *User*, and one *User* can be “the creator” of zero, one or many *Event*s. The relation is identified by the field *phone\_number* which is present in the *User* table. *One-to-many* relation (and also other relations) are presented later in the diagram using Crow’s foot notation.

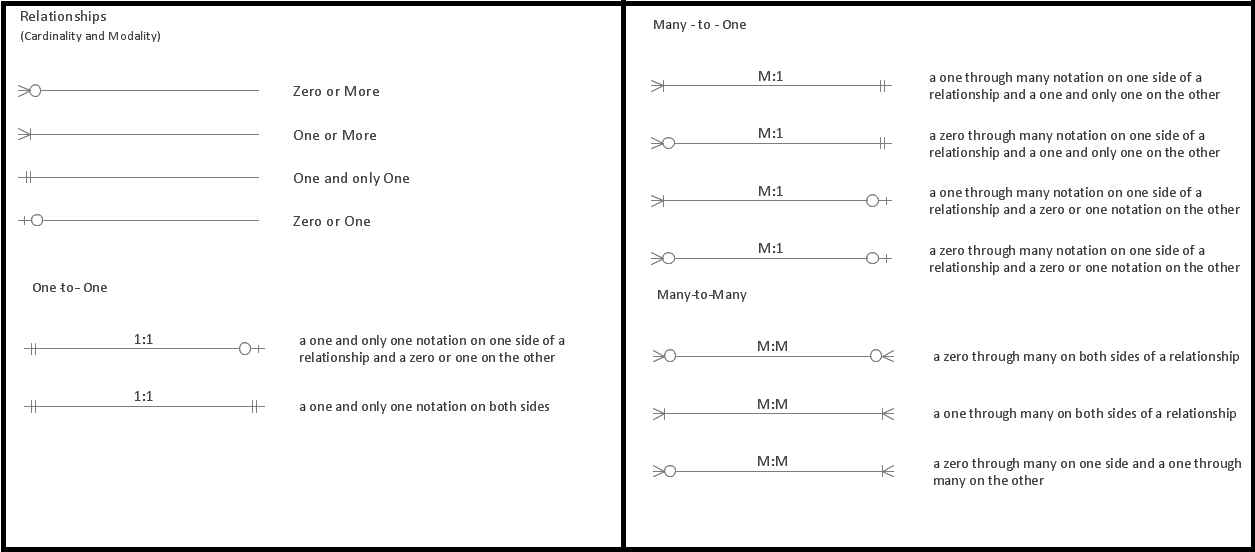


Figure 4.1. Crow’s foot notation

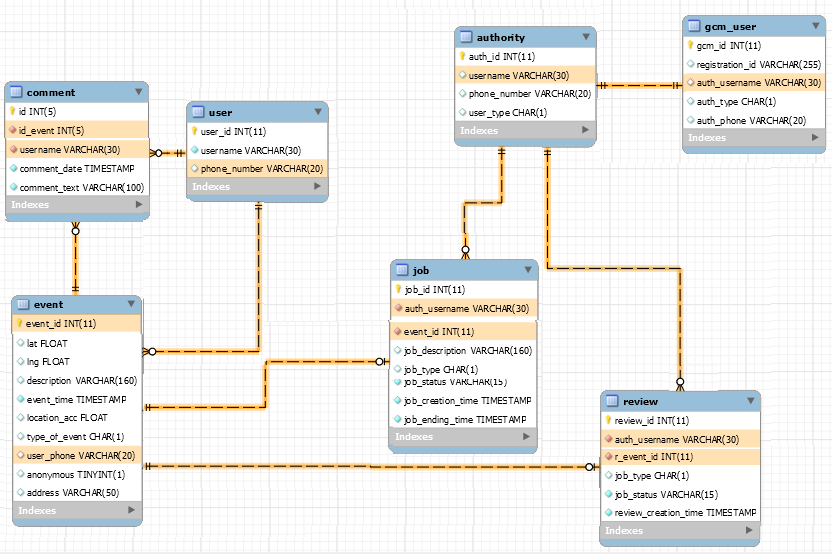
In the following diagram our system’s conceptual design is presented: 

Figure 4.2. Database design

Each *User* may post zero or more *Comment*s related to one specific entity Event. Each *Event* may contain zero or many *Comment*s posted by different users, but each Comment has to be connected to its unique creator of type *User*. Hence, the relations are *One-to-many* between *User* and *Comment* and *One-to-many* between *Event* and *Comment*.

On the other side of the schema there is a simple entity called *Authority*. Each *Authority* must also be *Gcm\_user* (Google Cloud Messaging User), and *Gcm\_user* must be identified by a single *Authority* username. Hence, *One-to-one* relation is added between them.

Each *Authority* may take an Event for investigation, and by doing that they create an entity of type *Job* in the database. After taking the *Job*, *Authority* must review the job, creating an entity *Review* in the database. Hence, *One-to-many* relation between *Authority* and *Job*, and *One-to-many* relation between *Authority* and *Review*.

*Job* and *Review* entities, from the creation time, must be connected to a single *Event*. Hence, *One-to-one* relation is added to the database.

Note: Although an *Authority* may take many *Jobs* and make many *Review*s by the database design, in software implementation we limited the amount of *Job*s to be taken to one. The *Job* needs to be reviewed in order to allow the *Authority* to take another *Job*. In this way we prohibited Authorities from greedy actions, for instance – taking all the jobs for themselves.

## LOGICAL DESIGN

Logical Design has the aim to better represent the database structure of our system, but, in order to build this model from the ER diagram drawn above, we have to perform some transformations.

After conceptual design we needed to create a real structure of the database, so we used a forward engineering approach to generate tables from Entity-Relations diagram. In order to achieve that we used a tool *MySqlWorkbench* which supports “Forward Engineering” transformation. In this way the time necessary to create the database was reduced effectively.

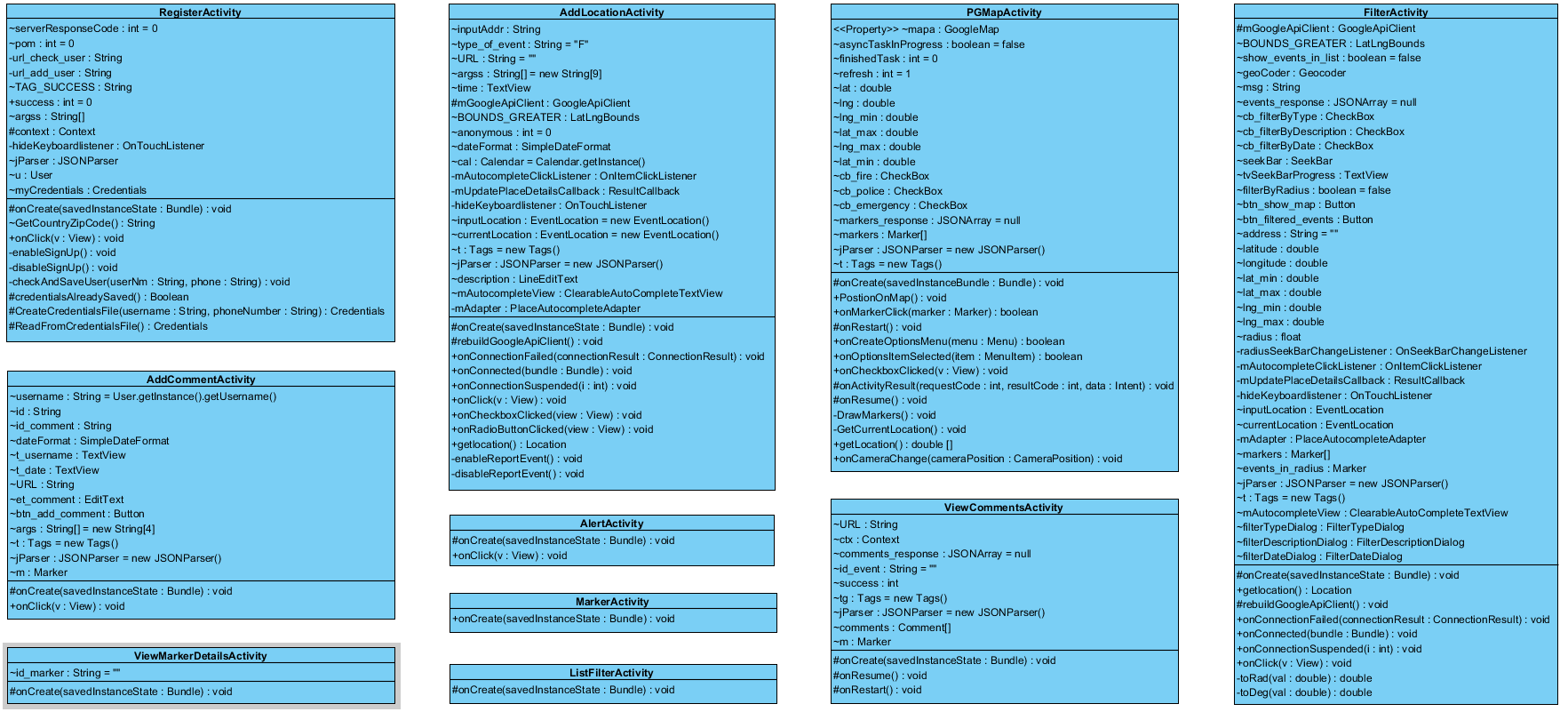


# Citizen app

## Class diagrams

This section will contain Citizen app's class diagrams. For the sake of readability and simplicity the class diagram has been broken down into smaller pieces. First diagram will just show an overview of main classes, namely Activity classes, and all others will go into more detail regarding each individual Activity.

### Activities class overview

Fig 5.1 -

### Register activity class diagram

Fig 5.2 -

### Add event activity class diagram

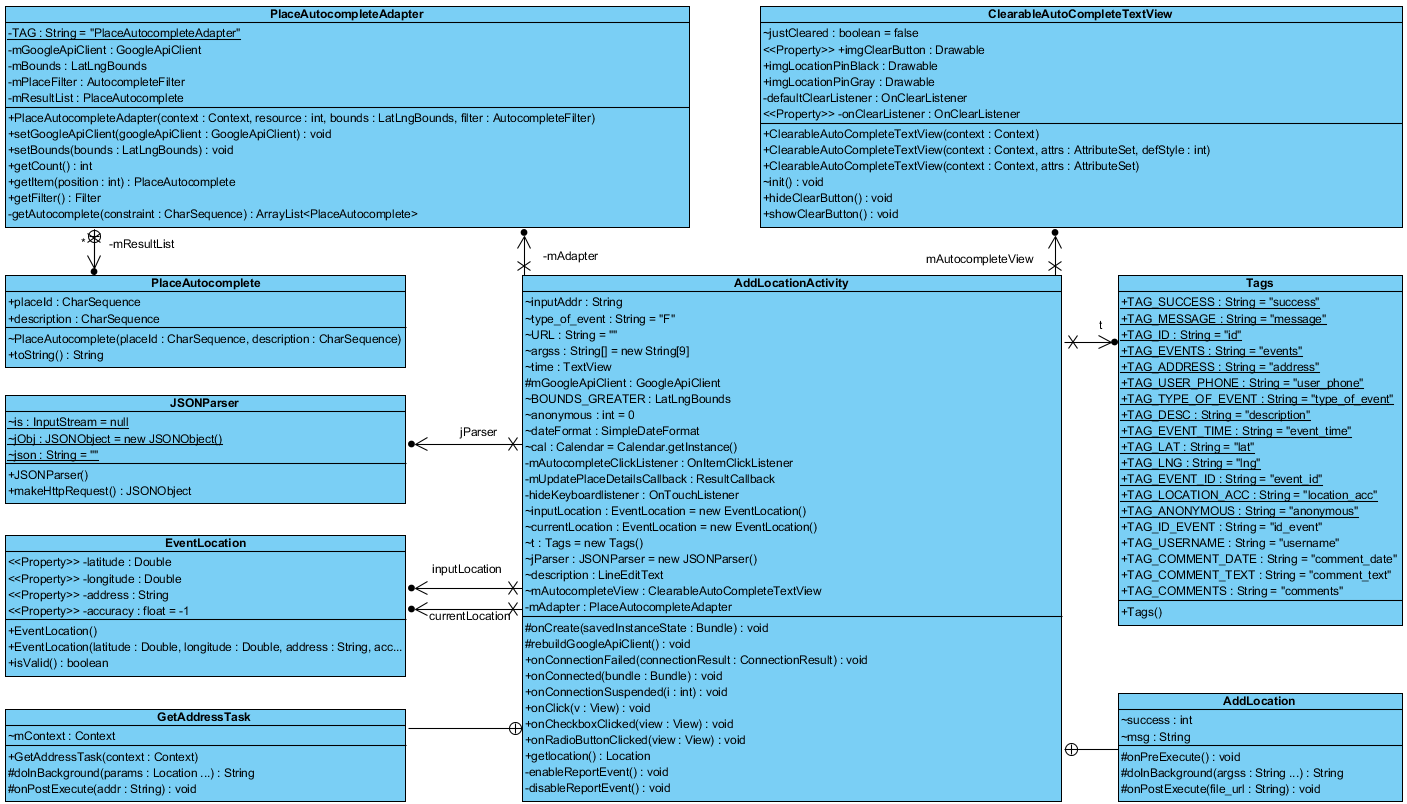


Fig 5.3 -

### Filter activity class diagram

Fig 5.4 -

### PG map activity class diagram

Fig 5.5 -

### Add comment activity class diagram

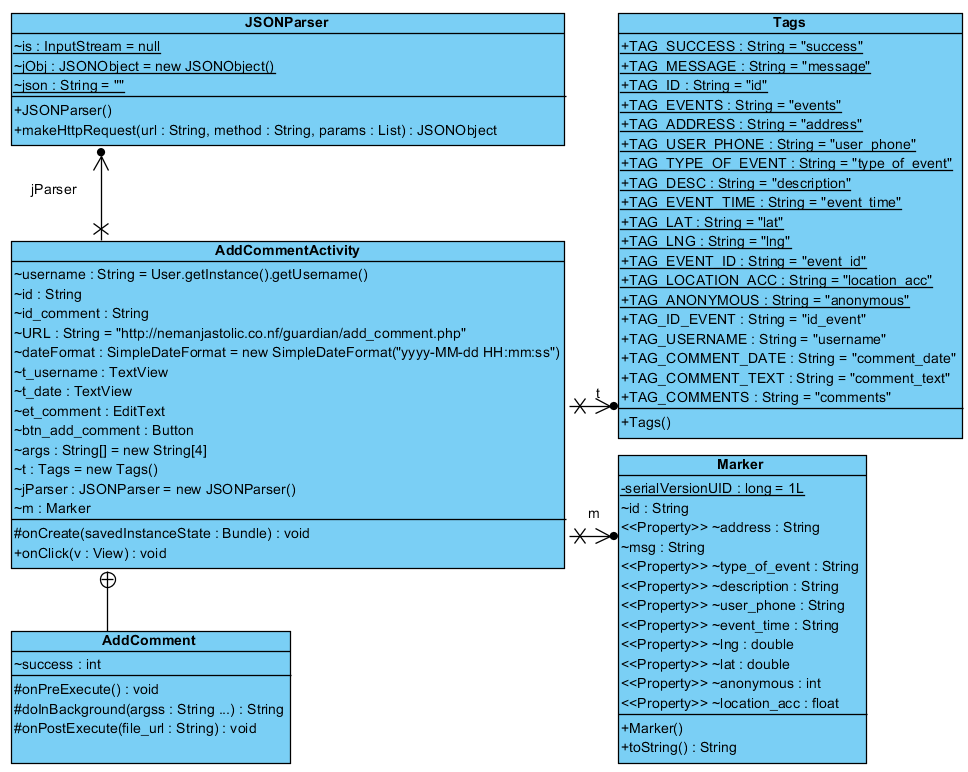


Fig 5.6 -

### View comment activity class diagram

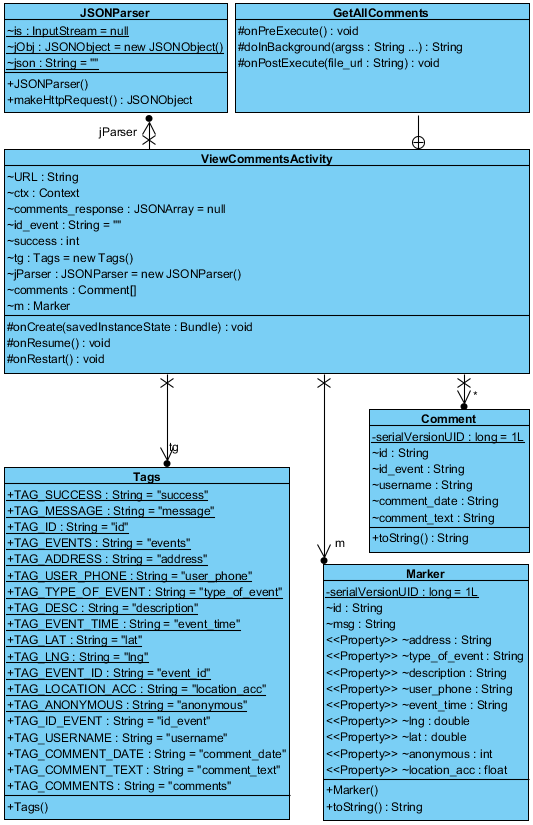


Fig 5.7 -



## Activities

|  |  |
| --- | --- |
| D:\FAX\I year\Mobile\PhoneGuardian\Documentation\Screenshots\CitizenApp\1 register_activity.png  Figure 5.2.1. Register Activity | D:\FAX\I year\Mobile\PhoneGuardian\Documentation\Screenshots\CitizenApp\2 alert_acitivty.png  Figure 5.2.2. Main Activity |
|  |  |

Figure 5.2.1. presents user register procedure. The user needs to type in username and his phone number. The application picks up his Country code based on his location. After submitting the information, the application tries to verify the phone number. In case of invalid phone number, the user receives a toast message stating that the phone number entered is incorrect.

Figure 5.2.2. presents main activity. If the user already has an account for this application, the register activity will not start; the application will check his credentials and immediately start the main activity. From this activity user can either report an emergency event or check the map for events nearby.

|  |  |
| --- | --- |
| C:\Users\milica13629\AppData\Local\Microsoft\Windows\INetCache\Content.Word\3 add_location_activity.png  Figure 5.2.3. Add Event Activity | C:\Users\milica13629\AppData\Local\Microsoft\Windows\INetCache\Content.Word\4 pgmap_activity.png  Figure 5.2.4. Map Activity |
| C:\Users\milica13629\AppData\Local\Microsoft\Windows\INetCache\Content.Word\5 marker_activity.png  Figure 5.2.5. Event Details Activity | |

Figure 5.2.3. presents an activity for adding new events. The application suggest to the user to add and event to his current location or to type in an address of the event; application is doing address matching using Google Places API and suggests to the user the best match; also the application is updating the time and date to current time and date. User needs to choose a type of event and type in a description of the emergency situation that he saw. Also he has an option of adding an event as an anonymous user or he could reveal his phone number to the appropriate authorities.

Figure 5.2.4. presents a map with pins fitting the screen around the current location of the user. Zooming in and out the viewport reduces or increases the number of pins shown on the map. The user can filter the pins by clicking the icons representing different types of reported events or he can refresh the map clicking on the appropriate button. Tapping on desired pin the user can get insight to more detail information about the reported event (shown on Figure 5.2.5.); also user is able to make more complex queries by clicking on filter button which takes him to the Filter Activity (shown on Figure 5.2.8.).

|  |  |
| --- | --- |
| C:\Users\milica13629\AppData\Local\Microsoft\Windows\INetCache\Content.Word\6 add_comment_activity.png  Figure 5.2.6. Add Comment Activity | C:\Users\milica13629\AppData\Local\Microsoft\Windows\INetCache\Content.Word\7 view_comments_activity.png  Figure 5.2.7. View Comments Activity |
| C:\Users\milica13629\AppData\Local\Microsoft\Windows\INetCache\Content.Word\8 search_activity.png  Figure 5.2.8. Filter Activity | C:\Users\milica13629\AppData\Local\Microsoft\Windows\INetCache\Content.Word\9 list_of_markers_activity.png  Figure 5.2.9. List of events Activity – after applying filter |

Figure 5.2.6. presents an activity for posting comments on the event which was selected by tapping a pin on the map. Comments are saved in a database and are connected with the selected event. Any user can check comments posted for any reported event. The list of comments and authority review together with the date of posting and usernames are shown in figure 5.2.7.

Figure 5.2.8. presents the activity for creating more complex queries for filtering the events. User can filer events based on his current address or he could pick an address, also he can set a radius around the typed address he is interested in. Filter includes choosing a type of event, description of reported event and date span he is interested in. The results of applied filter can be shown on a map or in a list.

Figure 5.2.9. presents list of events based on the applied filter created in previous activity. In the list user can see basic information about events.

# Authority app

## Class diagrams

This section will contain Citizen app's class diagrams. Much like the Citizen app's section, for the sake of readability and simplicity the class diagram has been broken down into smaller pieces. First diagram will just show an overview of main classes, namely Activity classes, and all others will go into more detail regarding each individual Activity.

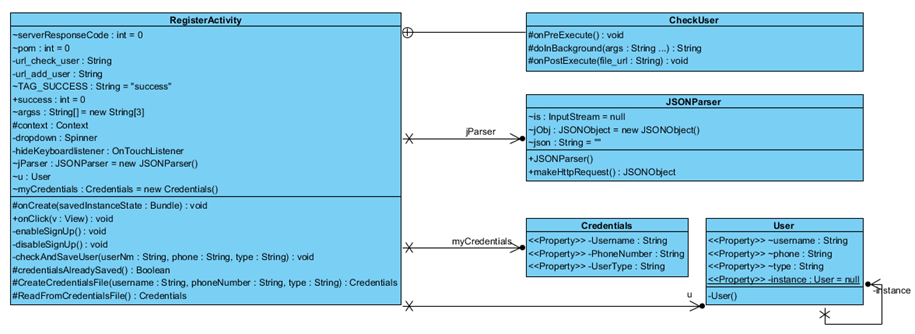
### Activities class overview

Fig 6.1 -

### Main activity class overview

Fig 6.2 -

### Register activity class overview



*Fig 6.3 -*

### Take job class overview

Fig 6.4 -

### Review job activity class diagram

Fig 6.5 -

### View job activity class diagram

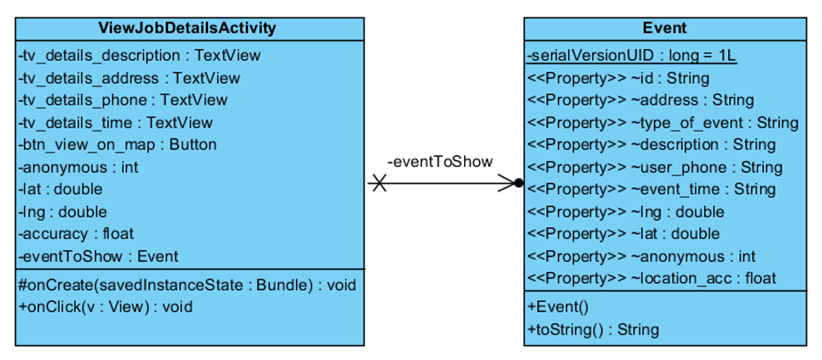


Fig 6.6 –

### View job activity class diagram

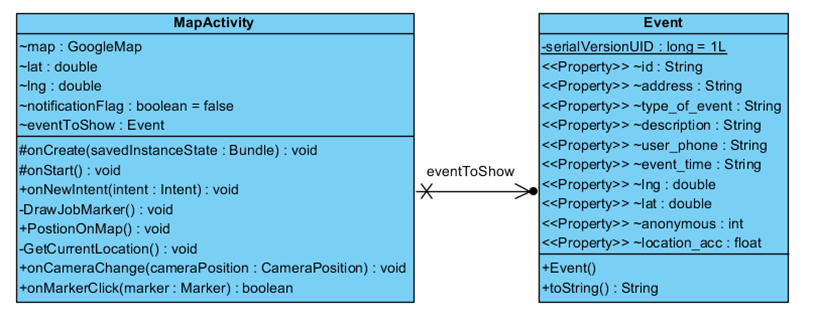


Fig 6.7 -

### Highscores activity class diagram

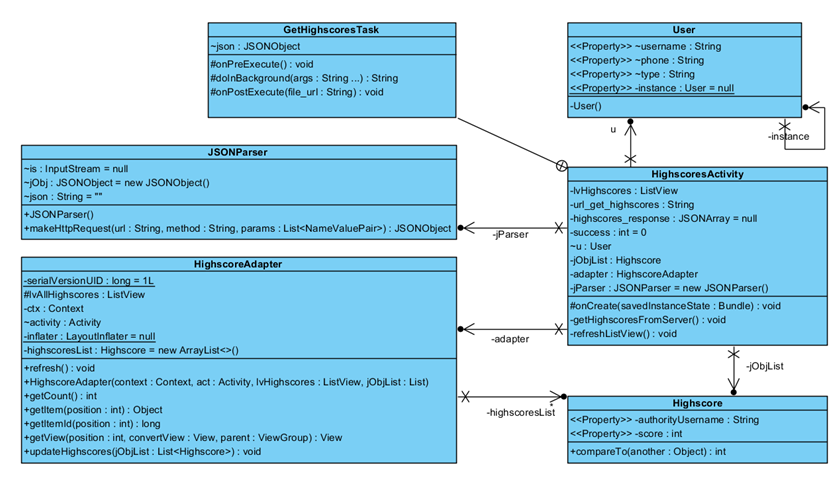


Fig 6.7 -



## Activities

|  |  |
| --- | --- |
| C:\Users\milica13629\AppData\Local\Microsoft\Windows\INetCache\Content.Word\1 register_activity.png  Figure 6.2.1. Register Activity | C:\Users\milica13629\AppData\Local\Microsoft\Windows\INetCache\Content.Word\2 main_activity.png  Figure 6.2.2. Main Activity |

Figure 6.2.1 presents register activity for authorities; it has the same functionalities as above presented register activity for regular user. The only difference is that they have to specify the work unit they are in i.e. fire brigade staff, emergency staff or police staff.

Figure 6.2.2 presents the main activity of AuthorityApp; the user has few possibilities: check for available jobs, review already taken job or look at current highscores table.

|  |  |
| --- | --- |
| C:\Users\milica13629\AppData\Local\Microsoft\Windows\INetCache\Content.Word\3 take_job_activity.png  Figure 6.2.3. Job List Activity | C:\Users\milica13629\AppData\Local\Microsoft\Windows\INetCache\Content.Word\4 job_details.png  Figure 6.2.4. Job Details Activity |

Figure 6.2.3 presents the list of available jobs for authority personnel; the job unit is an event that is not taken for investigation by any other authority. When the job is taken by a single authority (“Take job” clicked) it is removed from the offer for all other authorities. Also, the job list is sorted based on the user’s current location (from closest event that is reported up to the farthest). At any time the authority personnel can return back to this screen and return back the job to the offer by clicking on “Cancel current job”.

Figure 6.2.4 presents the detailed information of previously chosen job (“Details” button clicked in Figure 6.2.3.); it shows information about the event that happened such as date, address, description and phone number of user of CitizenApp if the event is not reported anonymously.

Figure 6.2.5 presents previous event on a map (“Show on map” button clicked in Figure 6.2.4.) showed as a pin, and current authority’s location show as circle.

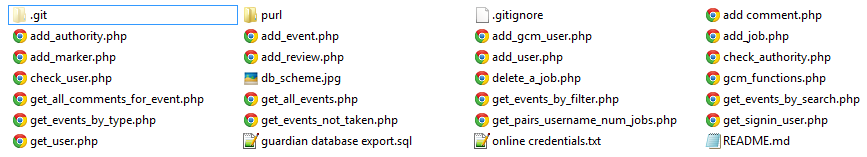
Figure 6.2.6 presents review activity where an authority can write a review of a job that (s)he took after finishing the necessary work. This review will be shown in a comment list of the solved job in CitizenApp; the users of CitizenApp will be able to see an official review given by an appropriate authority staff.

Figure 6.2.7 presents an already explained functionality of CitizenApp i.e. reporting an emergency event; purpose of this screenshot is to give an insight to the Notification Service (figure 6.2.8) implemented in AuthorityApp that notifies appropriate authorities about the event just reported, providing a real-time functionality of our system. The authority staff receives a notification about the reported event. Tapping on the notification opens a map - showing the reported event.

|  |  |
| --- | --- |
| C:\Users\milica13629\AppData\Local\Microsoft\Windows\INetCache\Content.Word\5 map_acitivty.png  Figure 6.2.5. Map Activity | C:\Users\milica13629\AppData\Local\Microsoft\Windows\INetCache\Content.Word\6 review_job.png  Figure 6.2.6. Review Job Activity |
| C:\Users\milica13629\AppData\Local\Microsoft\Windows\INetCache\Content.Word\7 CitizenApp add_location_activity.png  Figure 6.2.7. Add Event Activity – CitizenApp | C:\Users\milica13629\AppData\Local\Microsoft\Windows\INetCache\Content.Word\8 Notification_service.png  Figure 6.2.8. Notification Service |

# Web Server

Our web server, implemented in PHP technology, consists of one root folder named “guardian” and simple php files placed inside of it. The root folder is placed online at the address <http://www.nemanjastolic.co.nf/guardian/> , in order to enhance teamwork and debugging process.



*Figure 7.1. PHP folder structure*

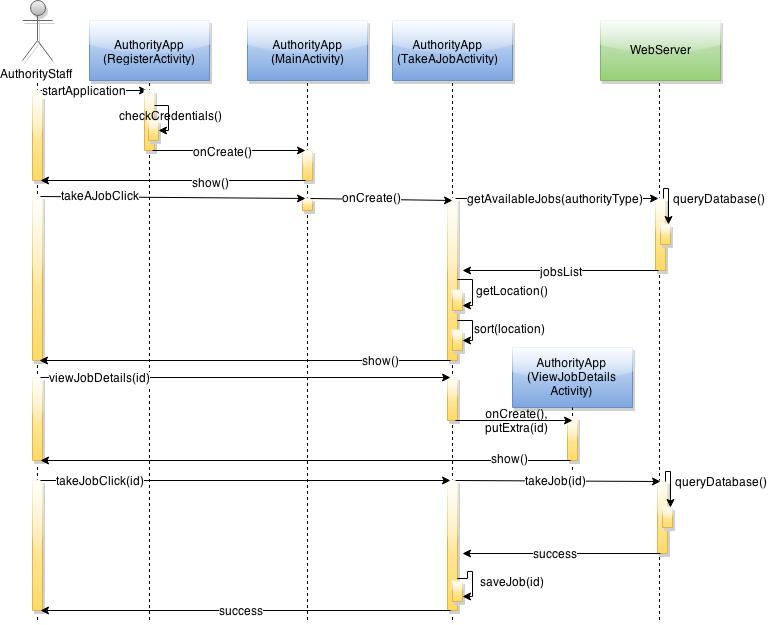
Besides simple php files that mostly just access the database and return the result of a query, there is also a folder named purl. “PURL” is a custom [GitHub](https://github.com/azayarni/purl) library for replacing the CURL functionality which our server didn’t provide us for free.

Also, an export of the MySQL database in form of the sql file is provided.

# Sequence diagrams

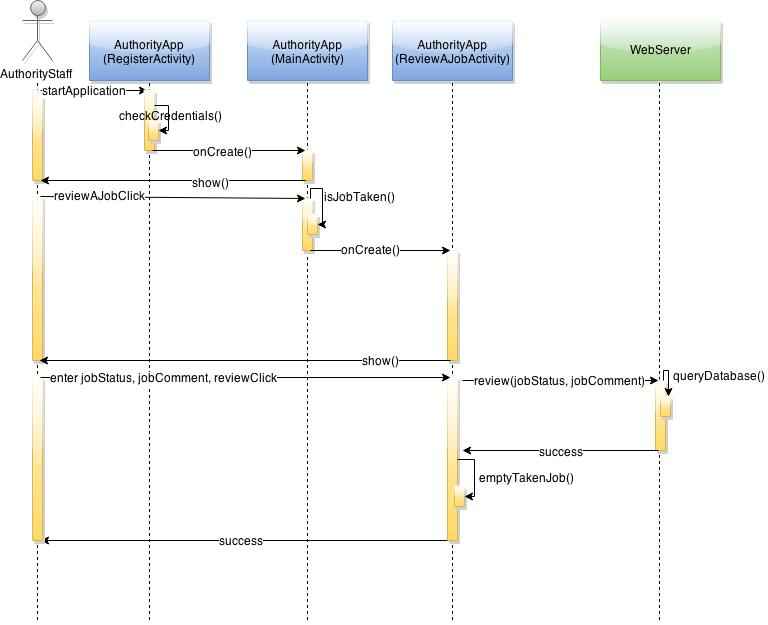
We provide some sequence diagram to let the reader better understand functionalities described above. We ask the reader not to take names of the methods strictly. For example, in *“View available jobs, view details, take a job”* sequence diagram it should have been written *getAvailableJobsFromServer()* instead of *getAvailableJobs(authorityType)*. Also *sort(jobs)* is done inside *JobAdapter* object contained in *TakeAJobActivity*, not exactly in *TakeAJobActivity* itself. The reasons for this kind of representation are readability and better understanding of application’s lifecycle.

## View available jobs, view details, take a job



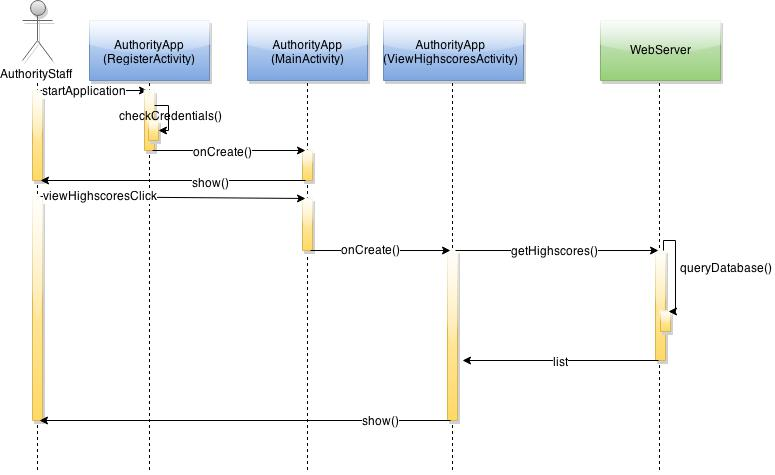
## Review a job

A user must take a job before reviewing it. Also, the user can’t take another job until (s)he finishes already taken one. The job review procedure is shown.



## View highscores

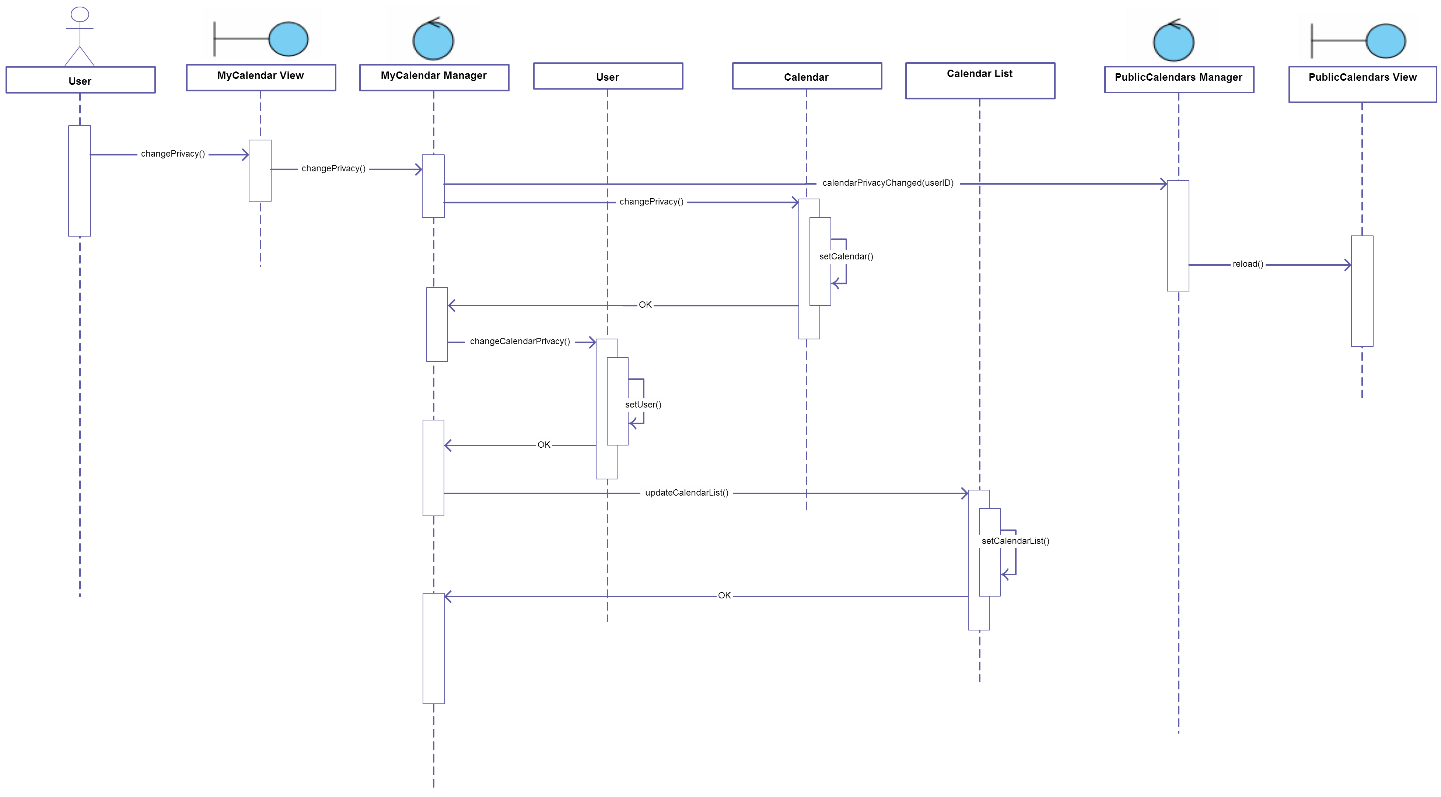
The username and number of jobs finished are provided by the server as a sorted list.



## Get notified about reported events

A user:

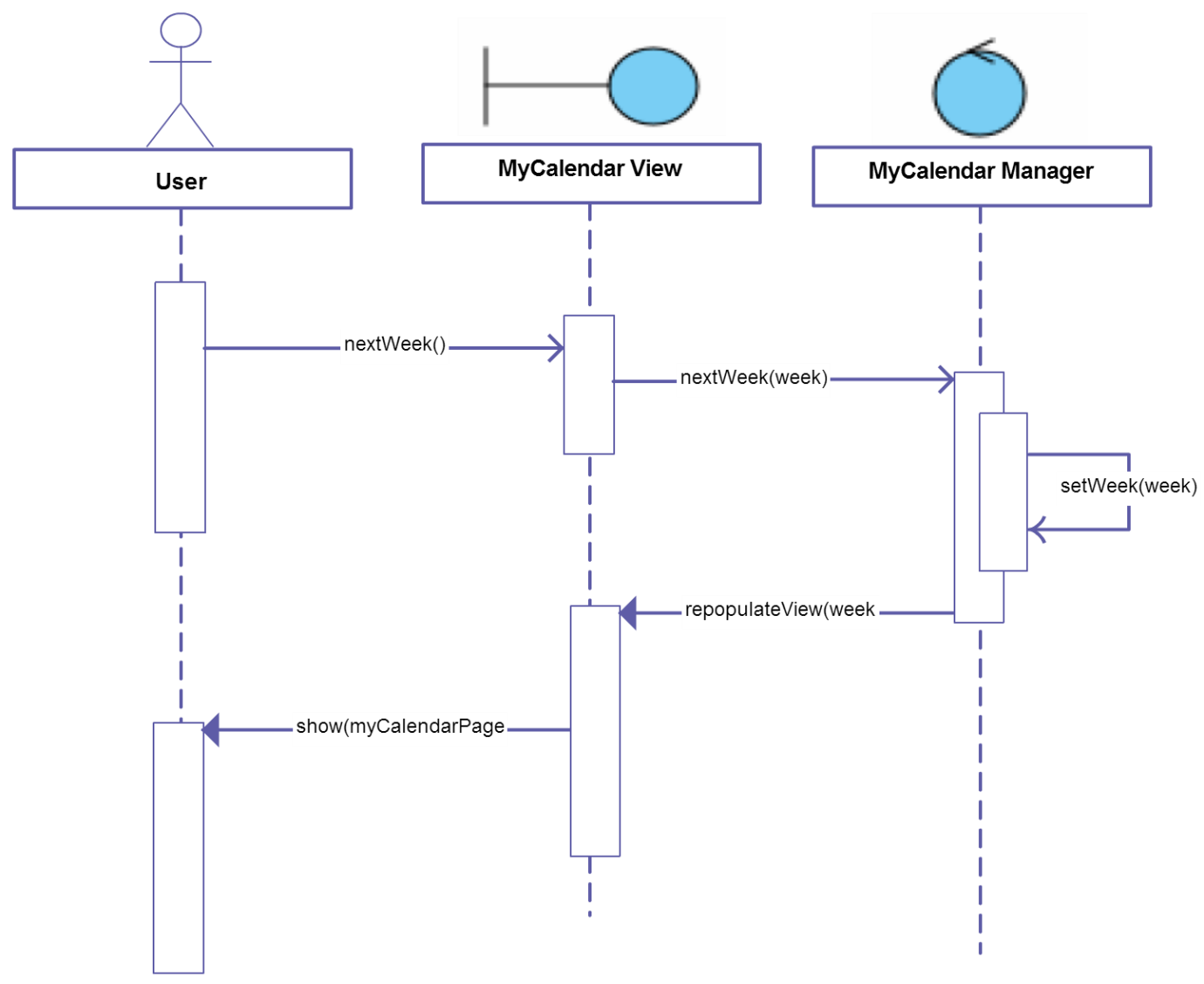
* Changes privacy setting of his calendar



## Next week

A user:

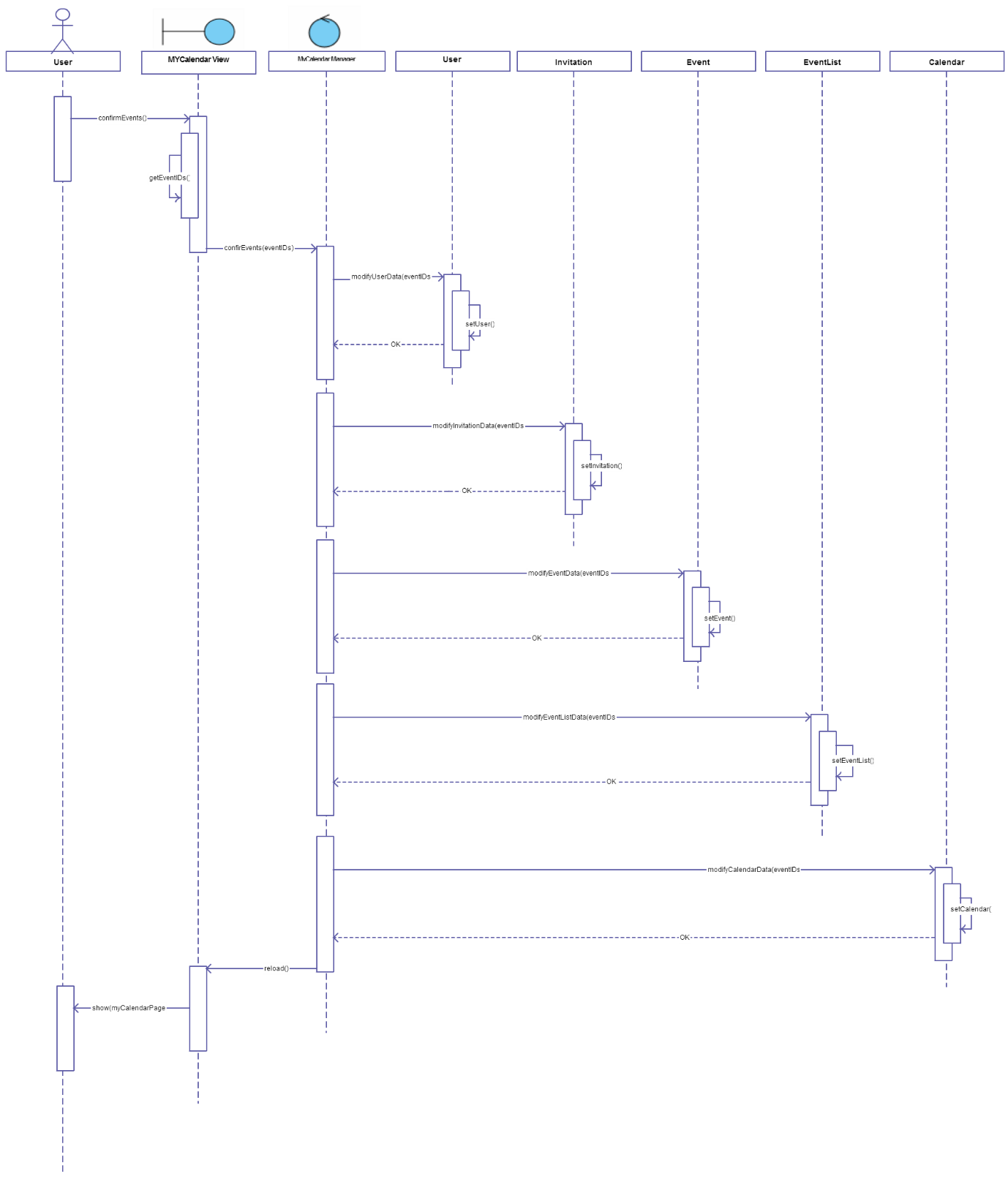
* User navigates through the calendar he is currently seeing



## Accept Invitations

A user:

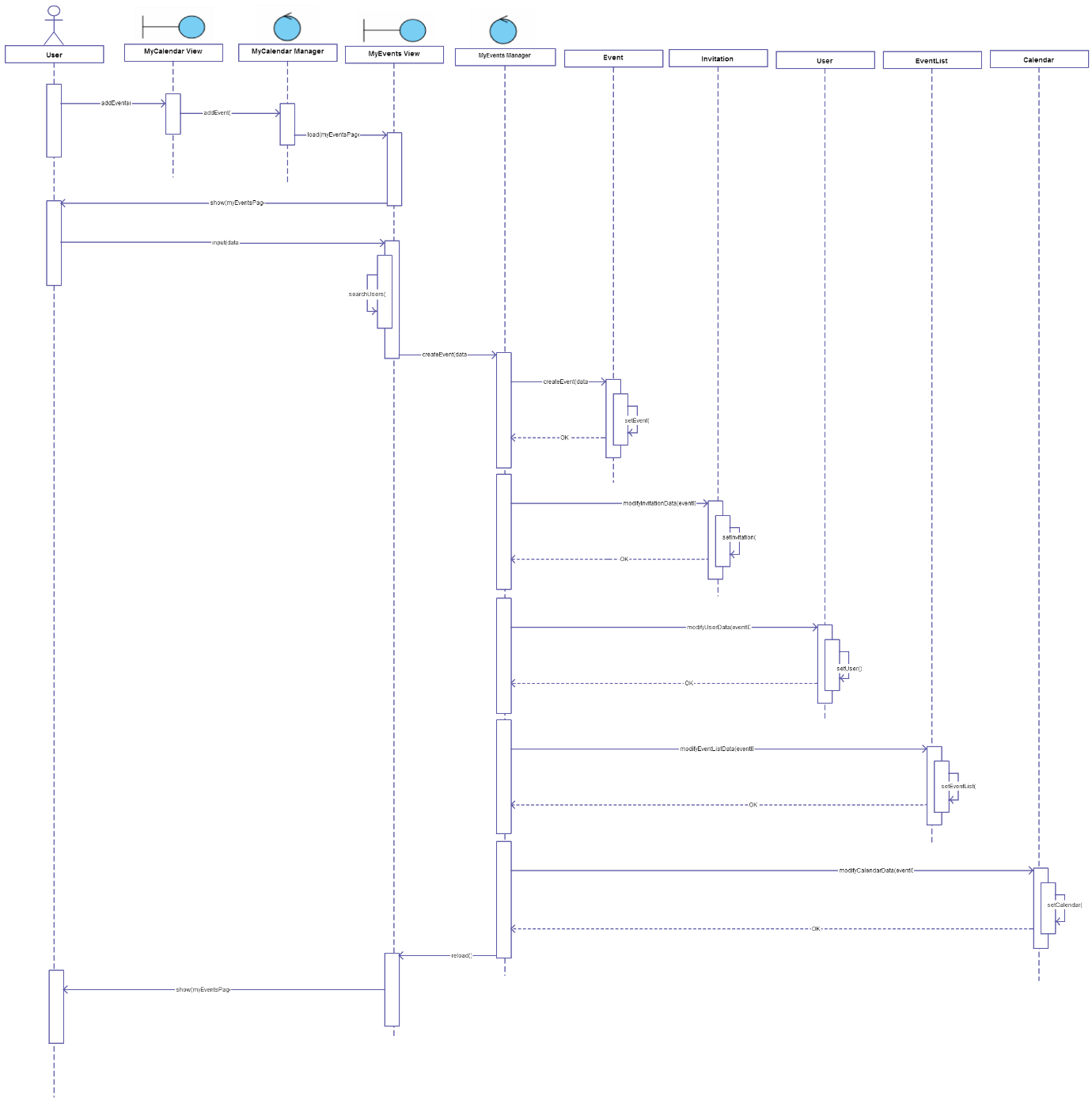
* User accepts invitations to the events he has selected



## Add Event

A user:

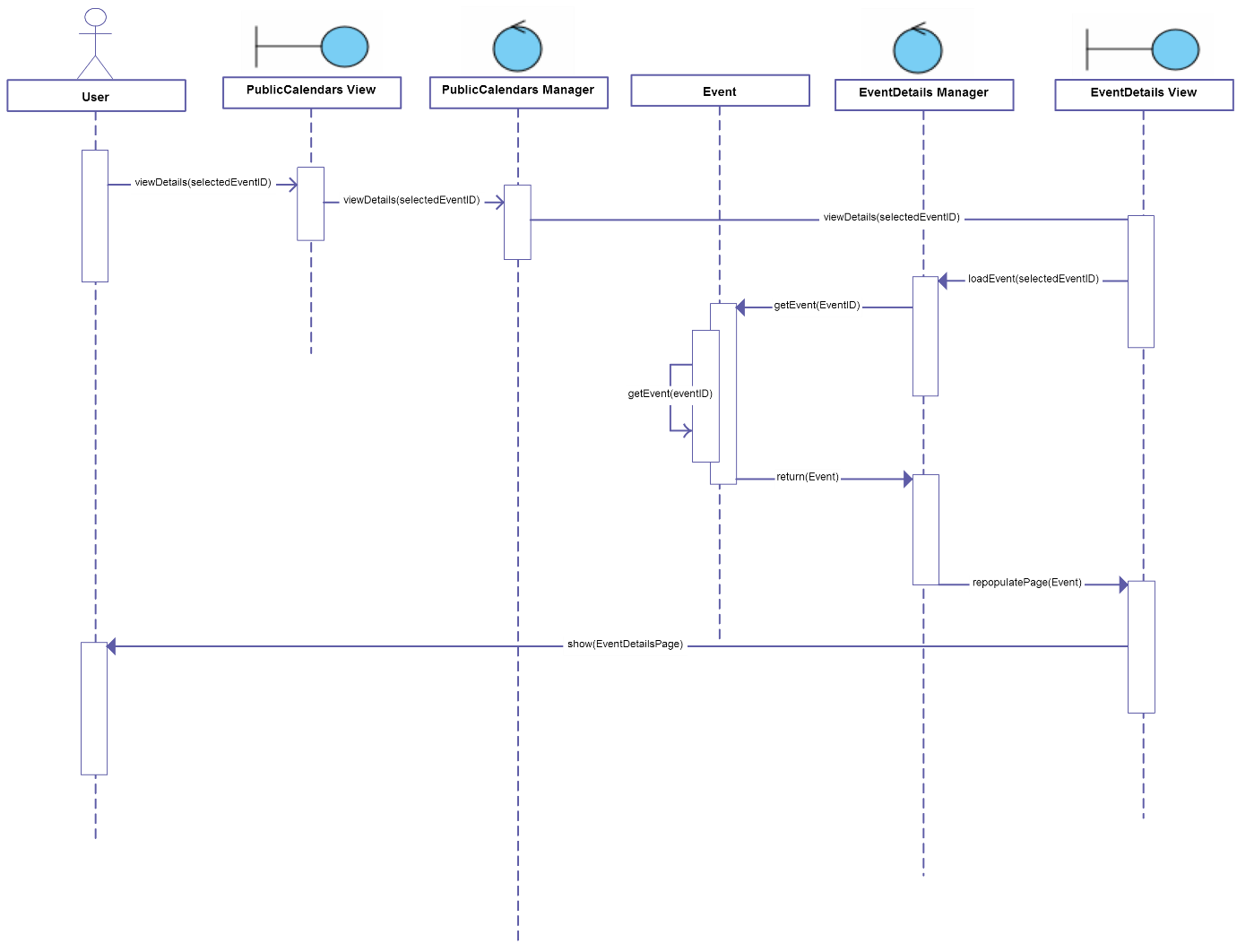
* User creates a new event



## View Events Details

A user:

* User previews details of the event he has selected



# FINAL CONSIDERATIONS

We decided not to draw any detailed diagram, because we think that a standard detailed diagram (with Server Page, Client Page, HTML Form and Control stereotypes) wouldn’t have added meaning to our Design Document. In fact, with this diagram, we only had to have Server Pages if Client Pages (the same thing as Screens in the UX Diagram) are built dynamically, and we know that almost a large part of our pages will be dynamic.

Moreover, it is not clear if Server Pages and Controls represent directly Servlets or Beans.

For this reason we think that the standard detailed diagram couldn’t bring us to a more specific knowledge of the implementation of our project.

Eventually, we drew UX Diagrams and BCE Diagrams instead, that are diagrams very detached from the architecture that lays under the project, but we decided not to draw more specific diagrams (such as Deployment View and Run-Time View), because we don’t know so much JEE architecture to go into details. We know, in fact, that from now on we have to take a very big effort to understand the architecture well and to start implementing our project.

# Used tools

* Microsoft Office Word
* Adobe Illustrator
* Evolus Pencil

# Working Hours

|  |  |
| --- | --- |
| **Name** | **Working hours** |
| Nemanja Stolic | 20 hours |
| Mirjam Skarica | 20 hours |
| Milica Jovanovic | 20 hours |