**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.docx#_Toc420181238)

[1. Purpose 3](#_Toc420181239)

[2. Brief introduction 3](#_Toc420181240)

[3. Conceptual design 3](#_Toc420181241)

[4. Architectural design 3](#_Toc420181242)

[4.1. IDENTIFYING SUB-SYSTEMS 3](#_Toc420181243)

[5. PERSISTENT DATA MANAGEMENT 6](#_Toc420181248)

[5.1. CONCEPTUAL DESIGN 6](#_Toc420181249)

[5.2. LOGICAL DESIGN 8](#_Toc420181250)

[5.2.1. TRANSLATION TO LOGICAL MODEL 8](#_Toc420181251)

[6. Used tools 11](#_Toc420181252)

[7. Working Hours 11](#_Toc420181253)

# Purpose

This document aims to describe the design and prototyping steps taken for “Big Gym” web application assigned as part of a project of the “Hypermedia Applications (Web and Multimedia)” course at Politecnico di Milano.

# Brief introduction

Good description of an app

The document provides detailed web application development procedure. Starting from the conceptual design, upgrading it to logical design and, finally, page design. After page design is introduced, it is going serve as the basis for interactive mockup development and will be described at the end of this document. In order to declutter and keep the document on point, we have left out the legend i.e. the meaning of all the symbols used in schemas and decided to treat it as a given. Explanation of all the symbols used can be found in the course's lesson slides.

# 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:

* Citizen app (Android application);
* Authority app (Android application);
* Web server (PHP server);
* Database (MySQL);

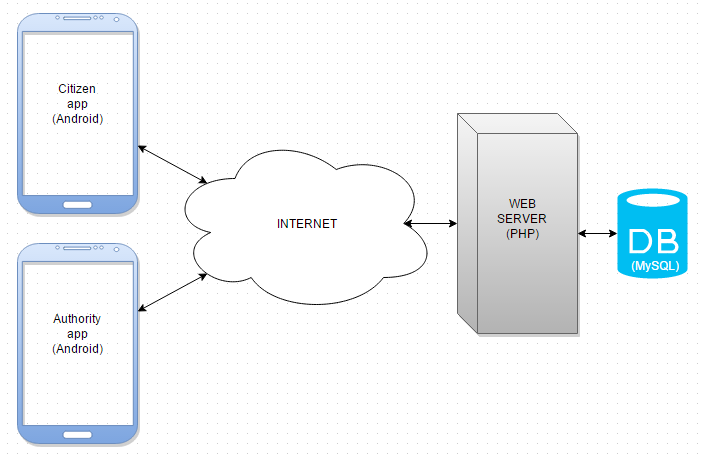


Figure - System architecture

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.

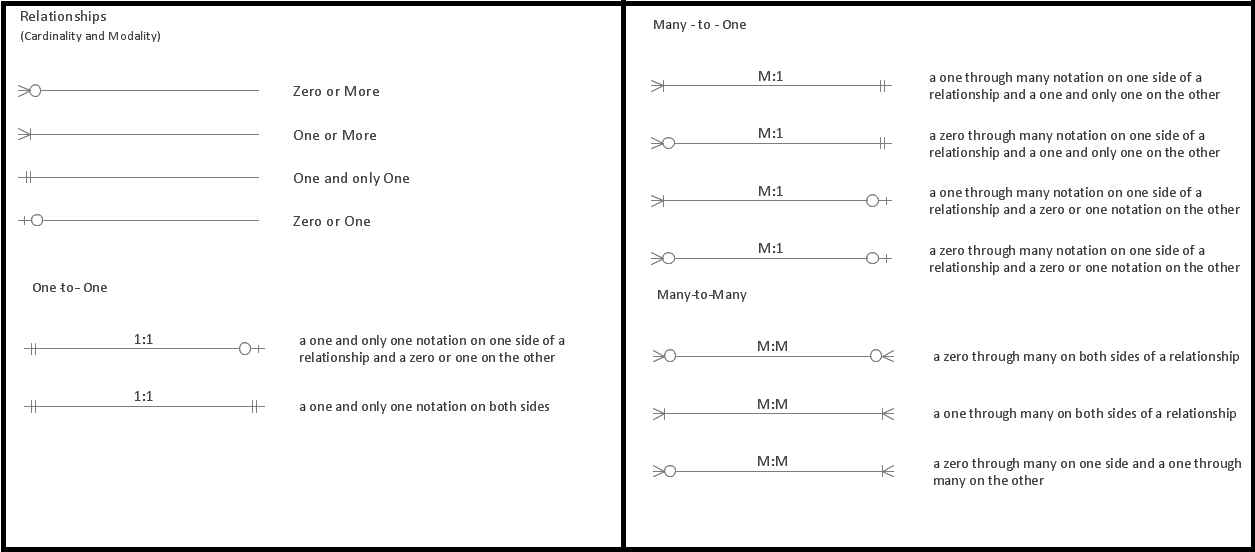
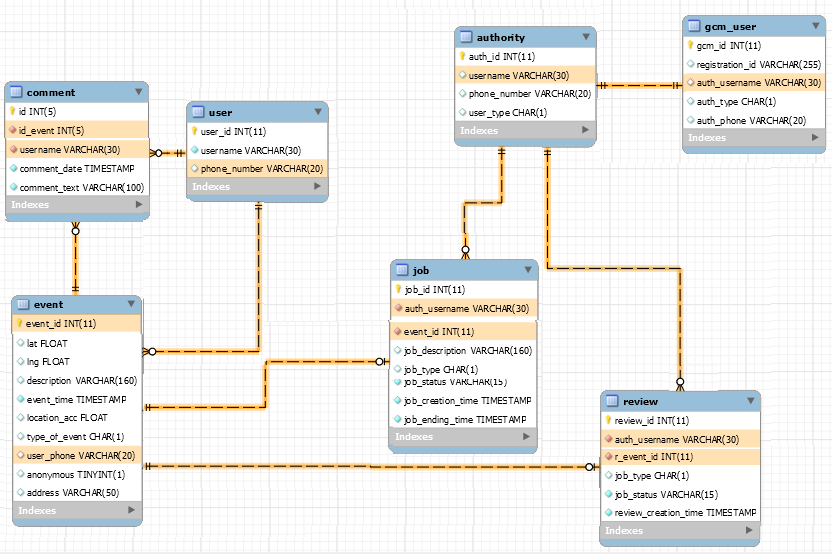


Figure: Crow’s foot notation

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

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 (s)he creates 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 (her)himself.

## 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 diagram

description

# Authority app

class diagram

description

# PHP Server

description

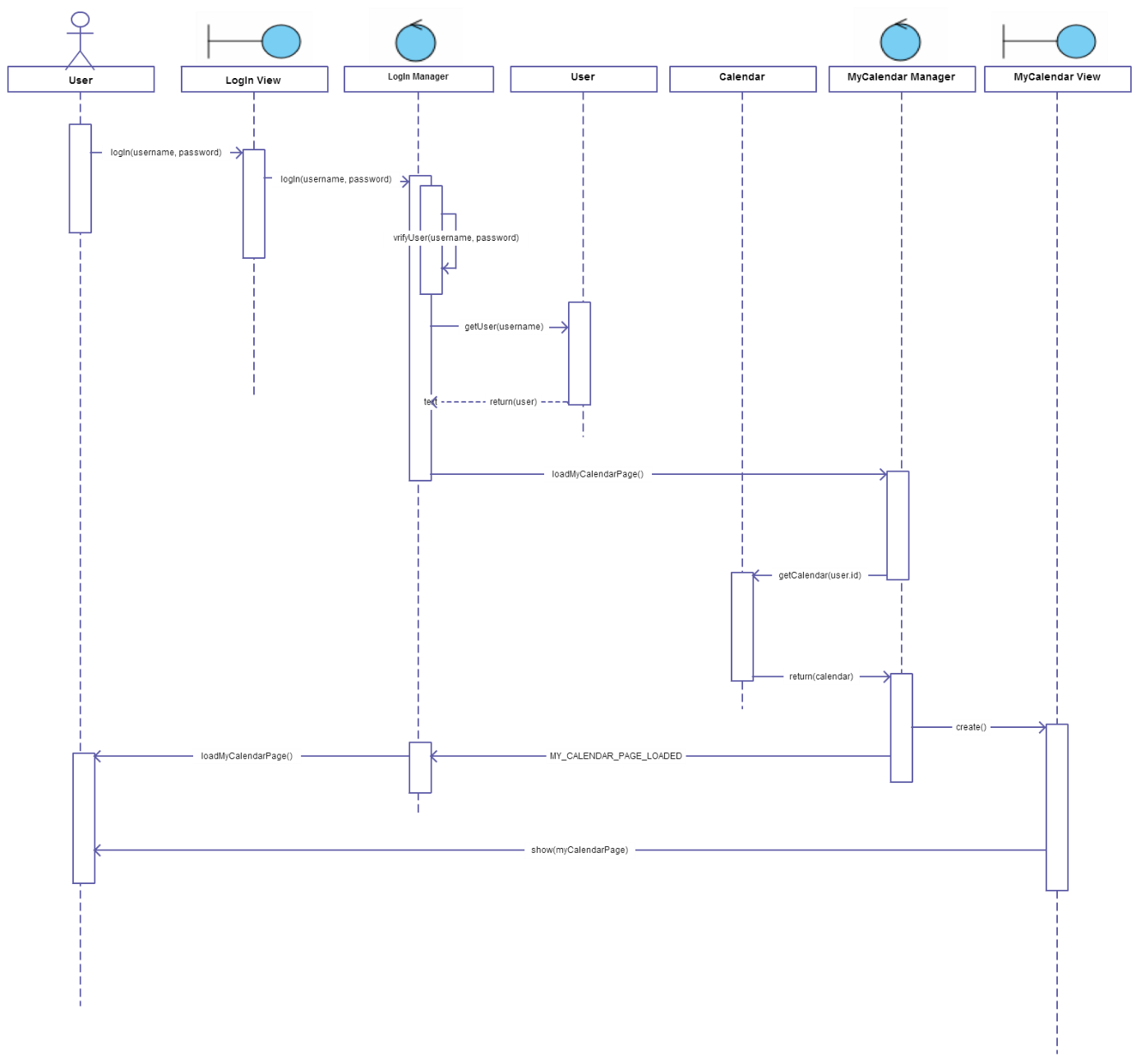
# SEQUENCE DIAGRAMS

We provide some sequence diagram to let the reader better understand BCE diagrams described above. All the methods used are the methods listed into the BCE in boundaries, controls and entities.

## Log In

A user:

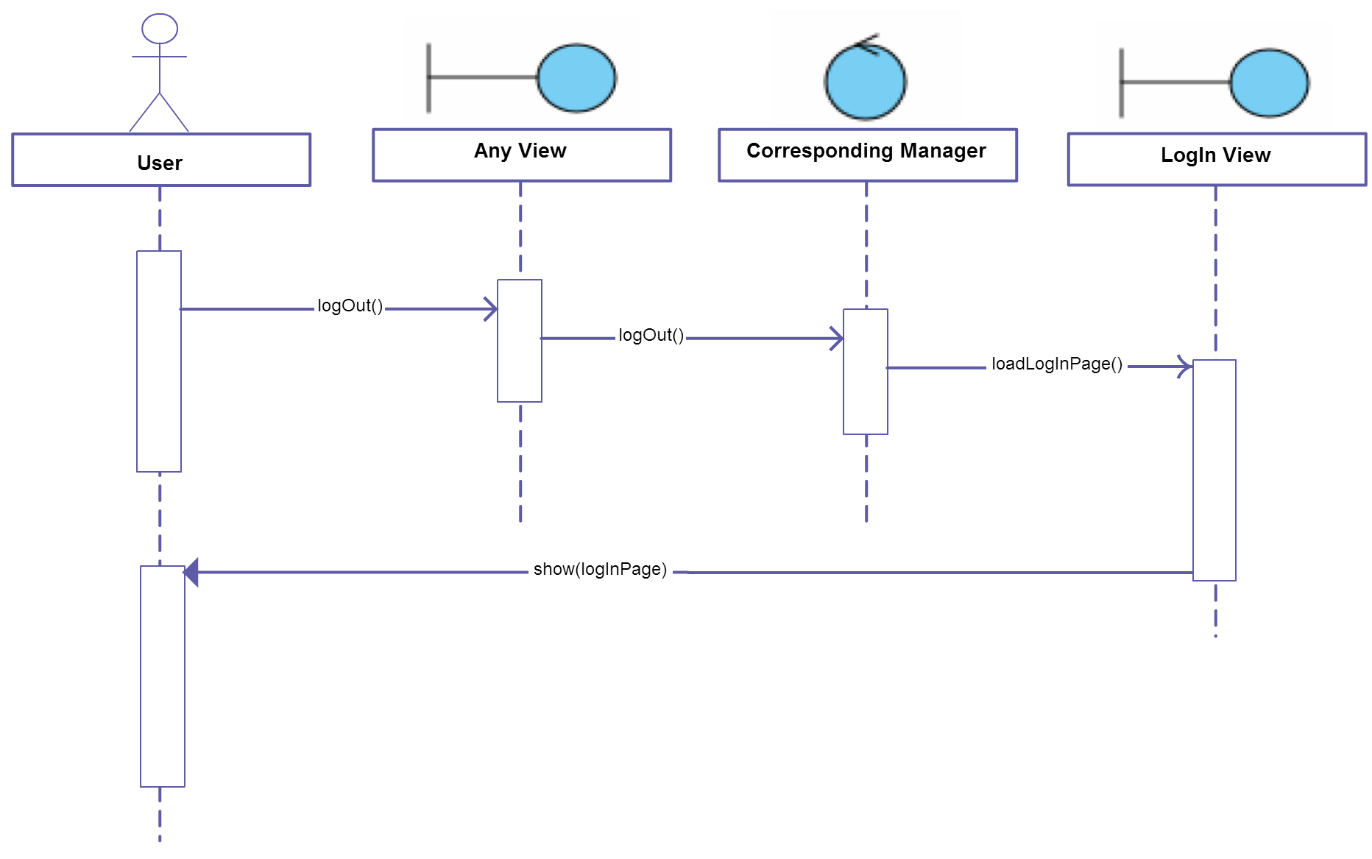
* Logs in.



## User Logs Out

A user:

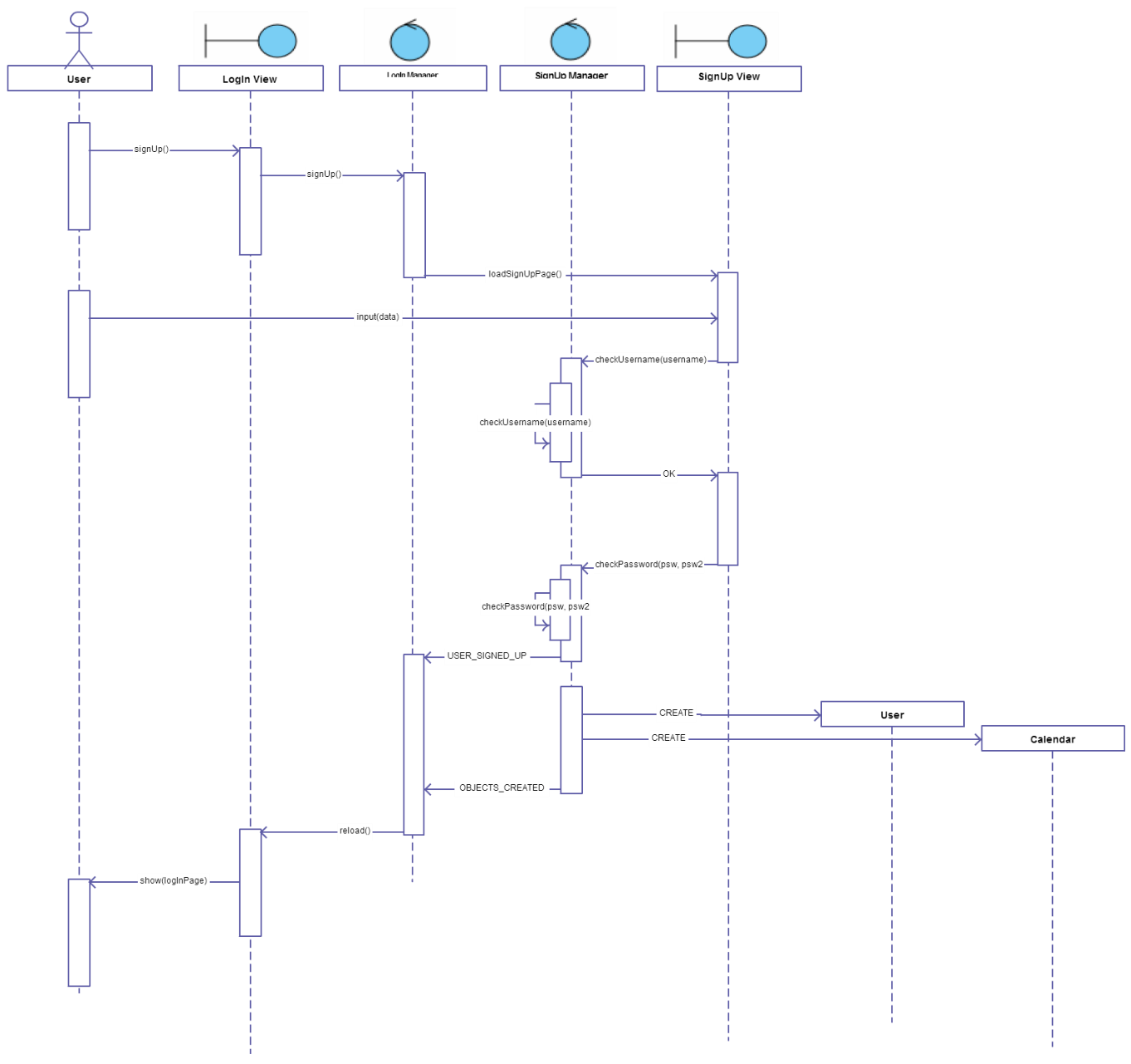
* Logs out of the system



## Sign Up

A user:

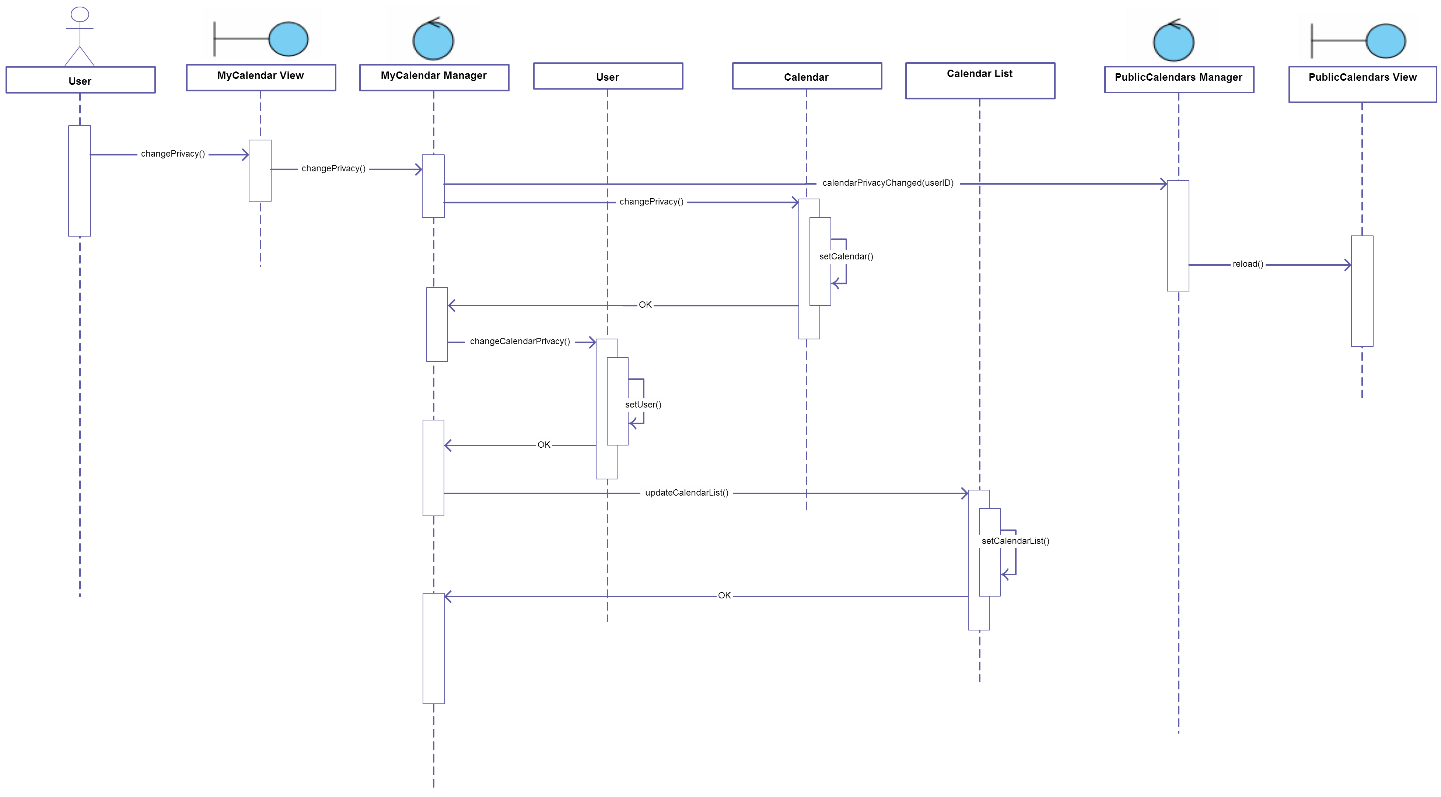
* Signs up as a new user of the system



## Change calendars privacy

A user:

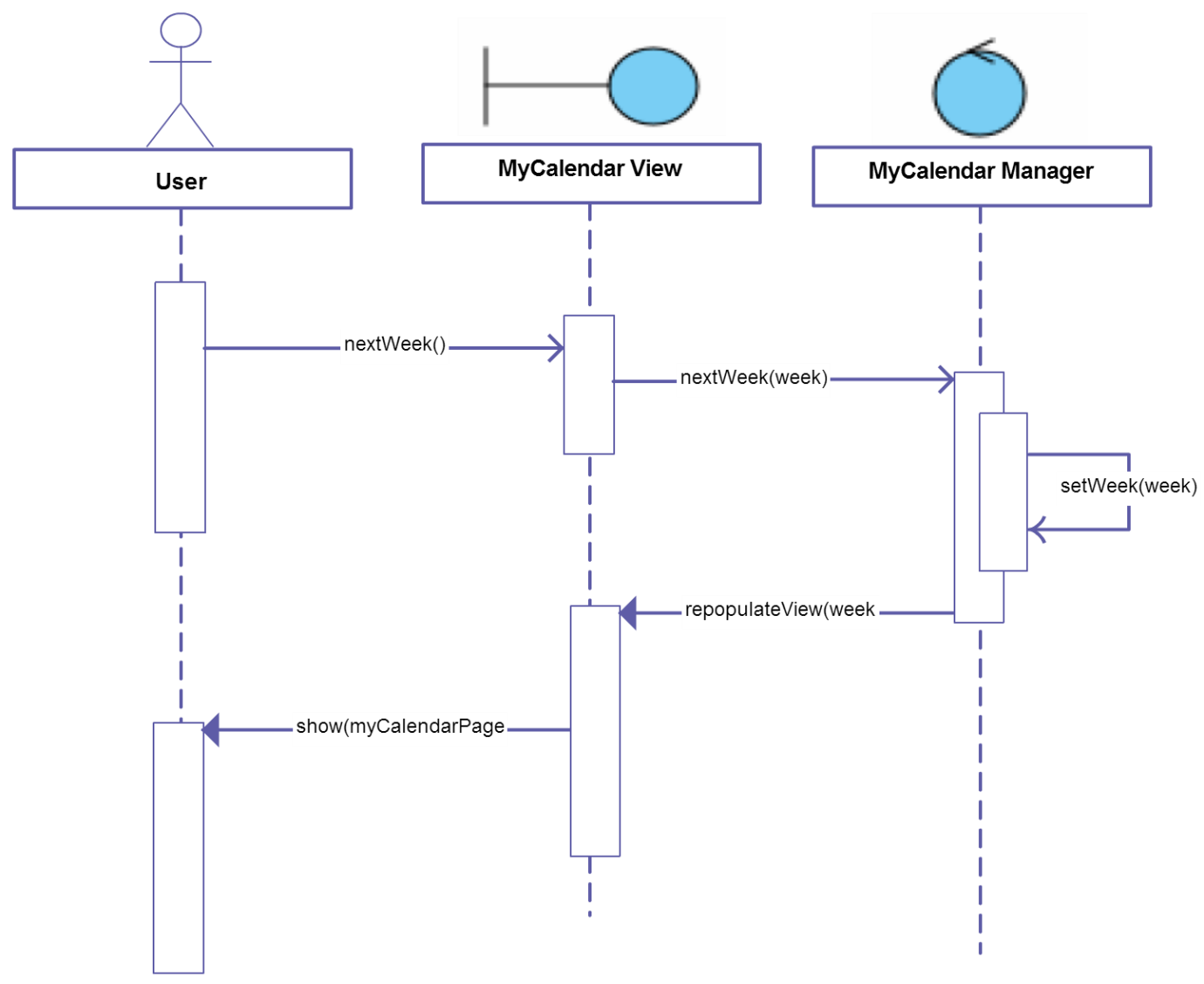
* Changes privacy setting of his calendar



## Next week

A user:

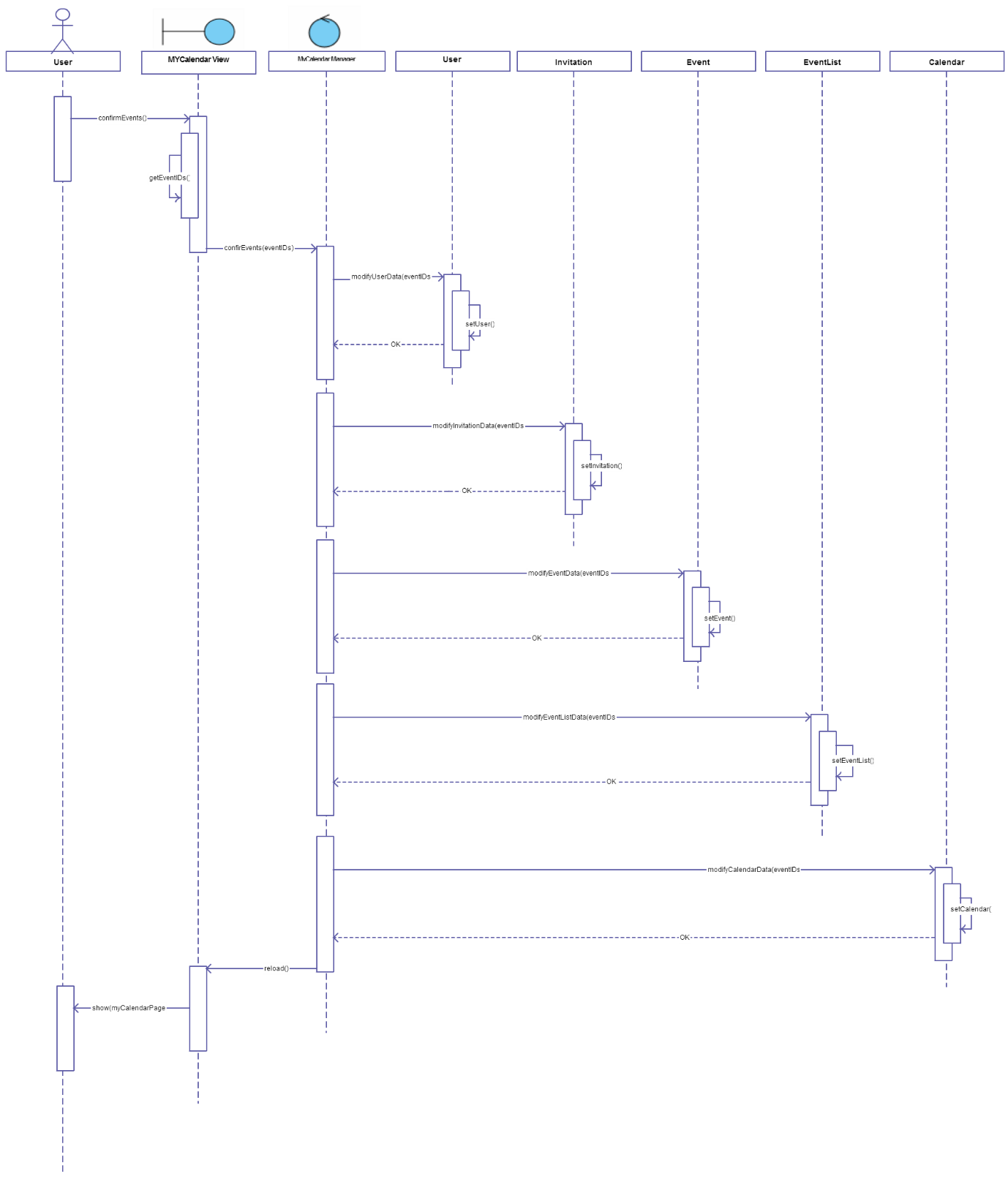
* User navigates through the calendar he is currently seeing



## Accept Invitations

A user:

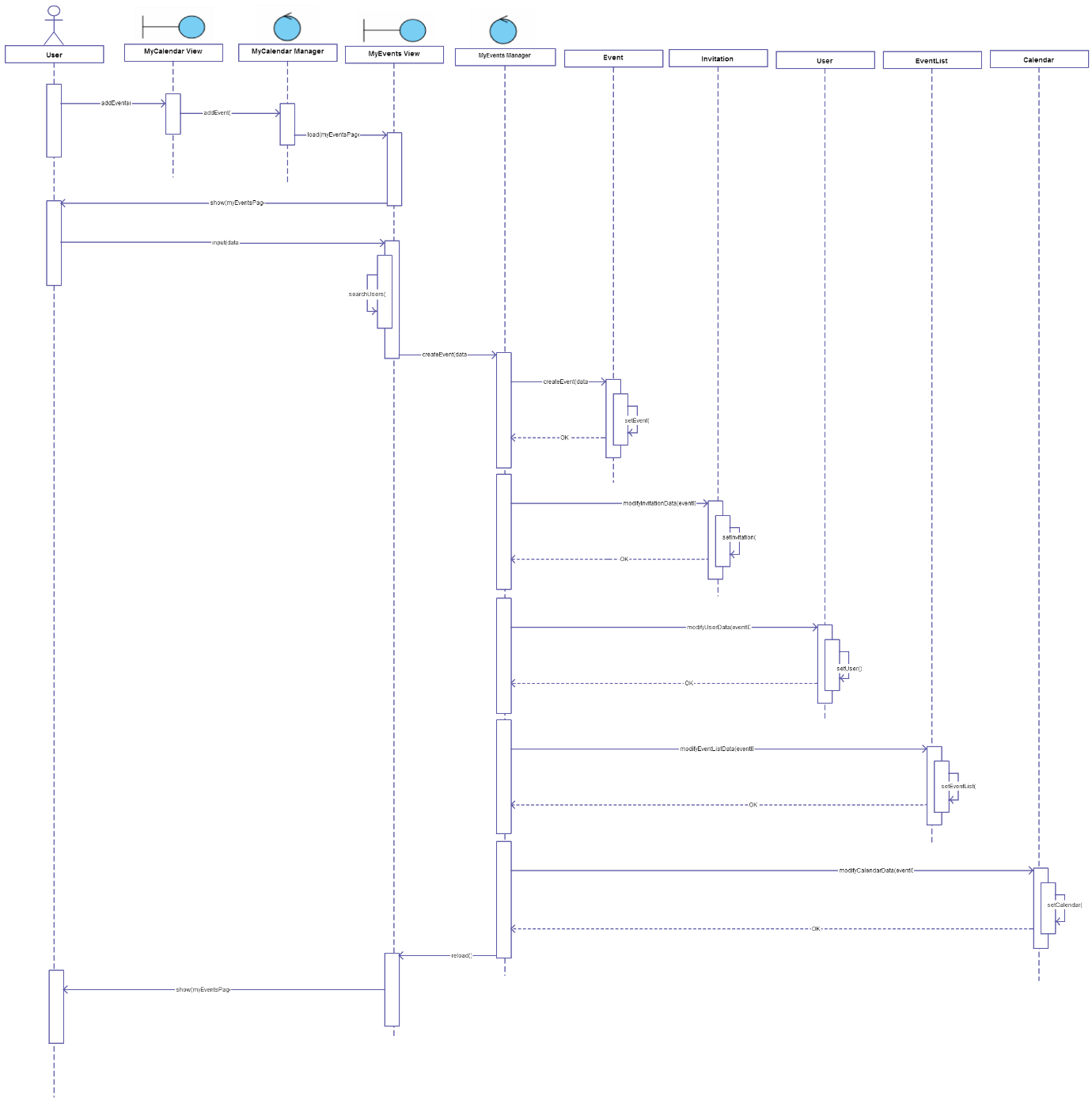
* User accepts invitations to the events he has selected



## Add Event

A user:

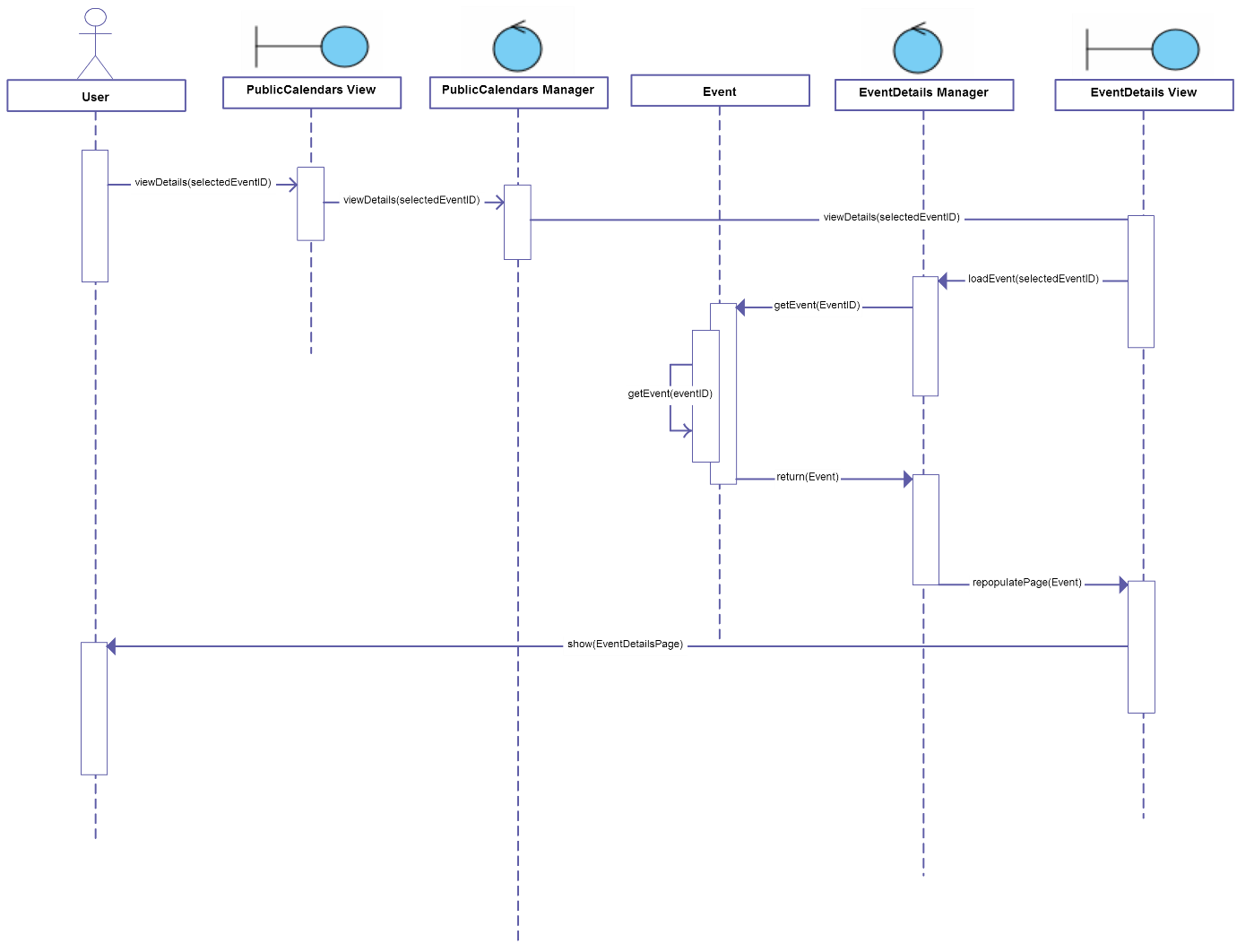
* User creates a new event



## 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 |