Politecnico di Milano

5th School of Engineering

Software Engineering Project

MeteoCal

Requirement Analysis and Specification Document

Nemanja Stolic 814842

Milos Colic 814817

Milan, 9th November, 2014

Contents

1. Introduction

1.1. Purpose

This document has the aim to describe the general functionality of MeteoCal application assigned as a project in the Software Engineering 2 course at Politecnico di Milano. Moreover, it contains all the critical aspects of the application and establishes an agreement between a client and a developer.

This document should be used as a guideline by the software developers; they can always refer back to the RASD to clear misunderstandings and follow the given requirements. The intended audience of this document is all the people actively participating in the Software Engineering 2 course, including professors and tutors. Not only shall the document serve as a reference for the developers to follow the requirements, but also it will serve to the testers to check whether the stated requirements and goals are met or not.

1.2. Scope

The software product that will be delivered is MeteoCal. MeteoCal is a web application intended to help people organizing their calendar, planning their events or finding the events that sound interesting to participate. For instance the application will allow registered users to manage their own calendar or see public calendars published by other registered users, add event to their own calendar, edit or delete their own events and invite other registered users to event. The application will also provide useful weather cast information related to event and notify user in case of bad weather conditions.

The main objectives of MeteoCal are:

- Users should be able to register on the system.
- Users should be able to see their own calendar or other public calendars.
- Users should be able to create a new event or edit/delete own existing events.
- Users who are owners of event should be able invite other registered users to their event.
- Invited users should be able to accept or decline the invitation to an event.
- Users should be able to see details, participants and weather cast data for their own events or other public events.
- The system have to notify user about bad weather conditions in the period of three days before the outdoor event starts.

- At any time the user should be able to reschedule an event.

MeteoCal will provide general functionalities for managing:

- Calendar – the system will manage privacy of calendars and access to them by other registered users.
- Invitations– the system will manage the invitations of the users for the events. Inside the invitation section of event the system will provide search for a user. One user can invite any registered user.
- Events – users can participate in an existing event for which they were invited or create a new event. If one accept an invitation for the event, he/she can cancel it at any time. The system will mark the events which are having bad weather predictions at specified time and it will notify the user it in the period of three days before the start to reschedule the event. The system should propose to event's creator the closest (in time) sunny day (if any).
- Users – the system will manage registration, log in/out of users.

MeteoCal will have the following limitations:

- Each user has only one calendar.
- User's calendar is either public or private.
- User can only accept or decline the invitation for the event.
- User can only schedule an event from the current day and later on.
- Users can schedule an event starting at exact hour, e.g. 1 pm, 2 pm etc..
- The duration of event can be set on an hour basis, e.g. 1 hour, 2hours etc..
- User can't create two different events that are overlapping in time, even if it's a part of the whole period.
- User can't accept to participate to two different events for which times are overlapping.
- User can delete an event only if it's its creator.
- User can edit an event only if it's its creator.
- The event's invitation is sent only once.
- Invitation expires when the event starts.
- User can see only public calendars.
- Private event's details can be seen only by its creator and invited users.

- In public calendars private event's hours are marked as busy hours.
- In case of bad weather conditions, when user is notified to reschedule the event, the system will search for sunny days in the period of next 15 days.
- The notification for rescheduling the event can be shown to the user only once in three days before the start of event.
- User can try to login with incorrect password three times in a half an hour.
- User can change his/her password only by accessing the link sent to his/her email.

The goals are:

- [G1] Allow users to register to the system.
- [G2] Allow users to change the privacy of their calendar.
- [G3] Allow users to create a new event.
- [G4] Allow users to edit or delete their own events.
- [G5] Allow users to invite other registered users to their events.
- [G6] Allow users to accept or decline the invitation to an event.
- [G7] Allow users to remove themselves from participants of events.
- [G8] Allow users to see event's details if they are invited to that specific event.
- [G9] Allow users to see public calendars.
- [G10] Allow users to see event's details if it is public event in the public calendar.
- [G11] Allow users to change their password.

1.3. Definitions and acronyms

1.3.1. Definitions

Keyword	Definitions
Visitor	Visitor can only access login and sign up webpage
User	Visitor becomes user when the system verifies the credentials submitted by the visitor.
Calendar	Collection of time slots to which events can be added. Each user possess only one calendar. Calendar can be private or public.

Event	Collection of entity's details (name, date, begin time, duration, city, street and number, indoor/outdoor, privacy), participants, invited users and weather.
Invitation	The users are invited to participate to an event by receiving an invitation to that event which is created by its owner. Only the creator of the event can send invitation to that event to other users. Invitation consists of two actions: accept or decline. Invitation expires when event starts.
Participant	The user who has confirmed his/her attendance to event to which he/she was invited.
Notification	A message popup which says if there is any important change in the user's calendar.

Table 1: Definitions.

1.3.2. Acronyms and abbreviations

Acronym/Abbreviation	Definitions
XML	Extensible Markup Language
RASD	Requirements Analysis and Specification Document
NFR	Non-functional Requirement
QA	Quality Attributes
FR	Functional Requirement
G	Goal
DBMS	Data Base Management System
AS	Application Server
JEE	Java Enterprise Edition

Table 2: Acronyms and abbreviations

1.4. References

1. IEEE Recommended Practice for Software Engineering Requirements Specification (http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?reload=true&punumber=5841)
2. Alloy model file: "onlinecalendars.als"

1.5. Overview

The document is organized as follows:

1. **Introduction**

   This section provides a synopsis of the software product to be developed.
2. **Overall Description**

   This section describes the general factors that affect the software product and its requirements.
3. **Specific Requirements**

   This section contains all the analysis done to the project requirements. It describes all of the software requirements to a level of detail sufficient to be externally perceivable.
4. **Appendices**

   This section provides supporting information showing how the alloy model contributed to the analysis model and requirement analysis.

2. Overall description

This section describes the general factors that affect the software product and its requirements, and it provides a background for specifying concrete requirements in the next section of this document.

2.1. Product description

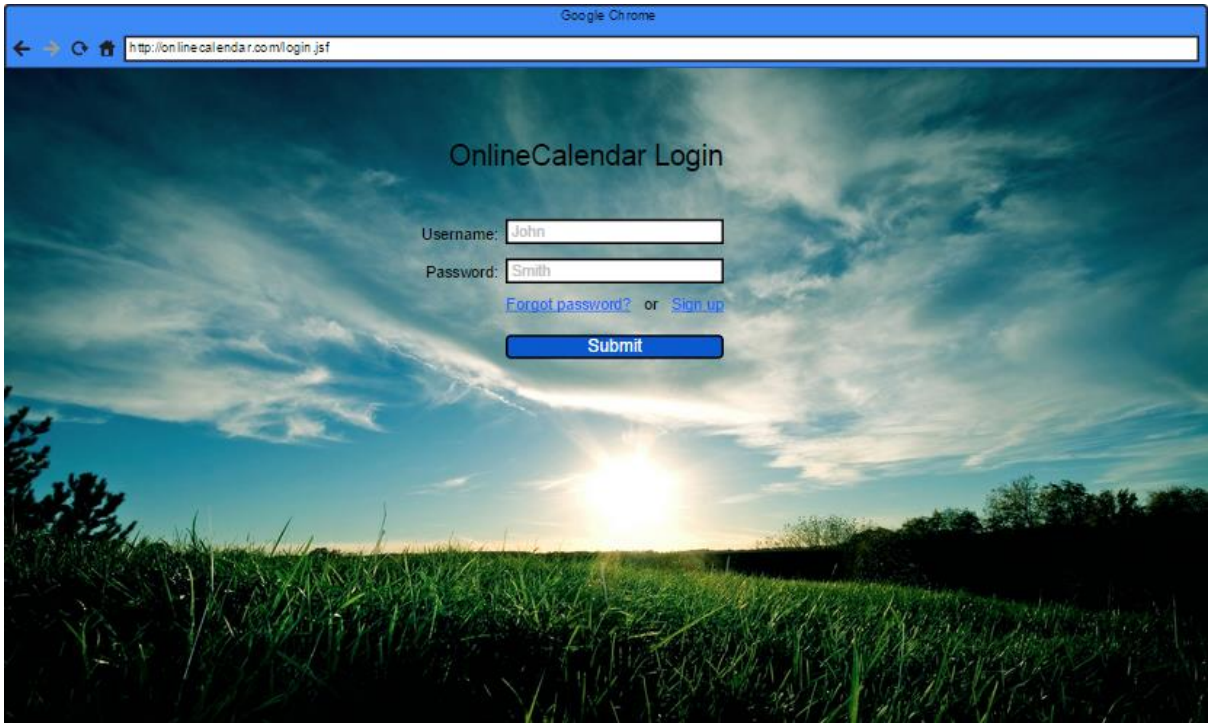
The software product is a completely self-contained system, independent from other systems.

2.1.1. System interfaces

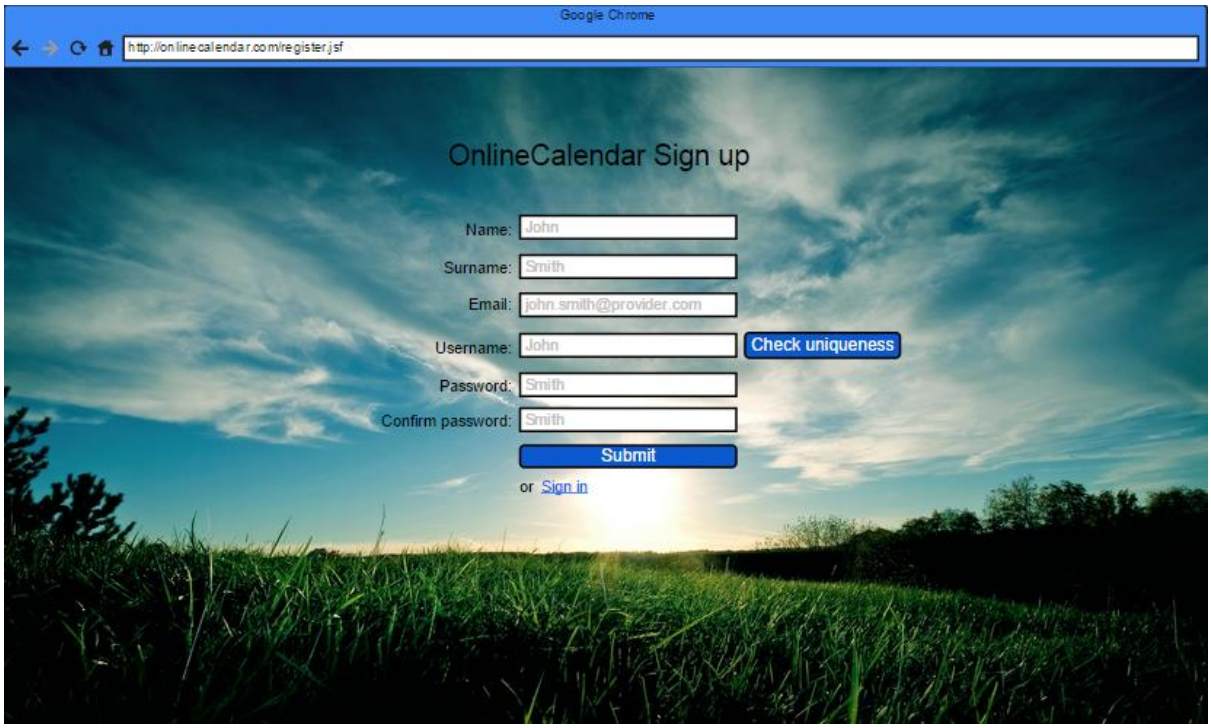
The software product does not provide any external interfaces.

2.1.2. User interfaces

The software product will have similar page layouts as a user interface:

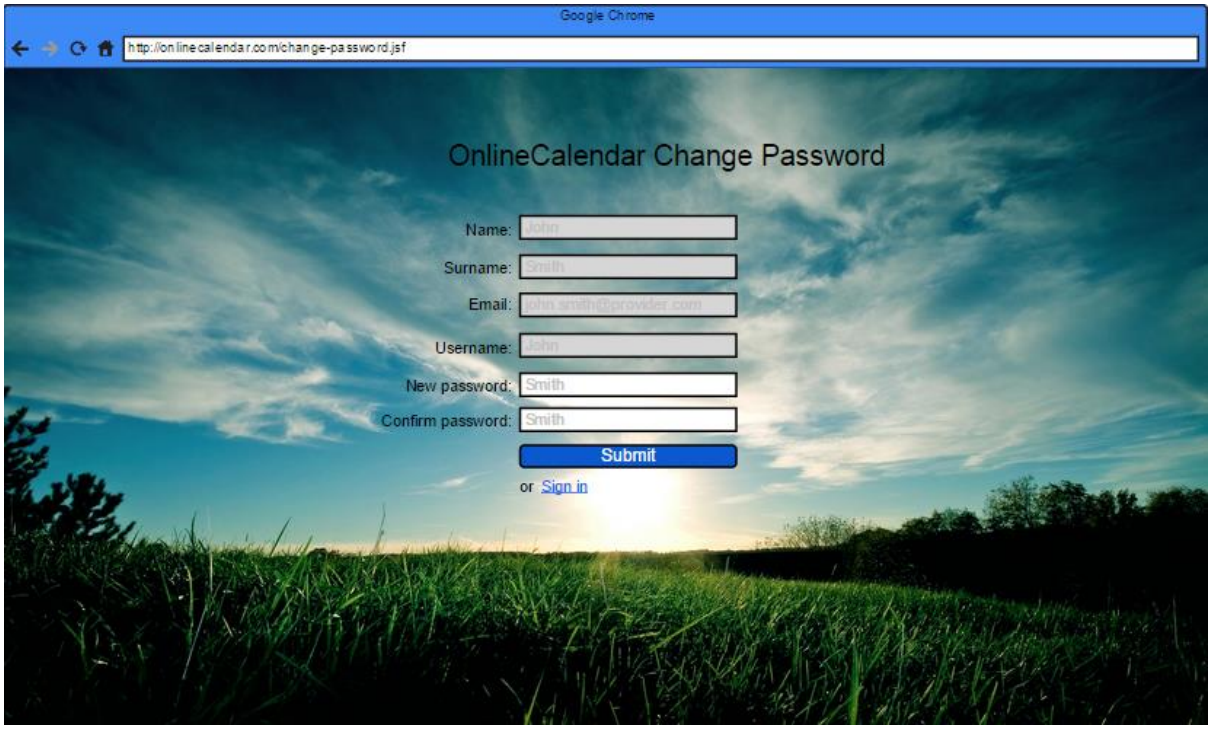


User interface 1: Login page of MeteoCal

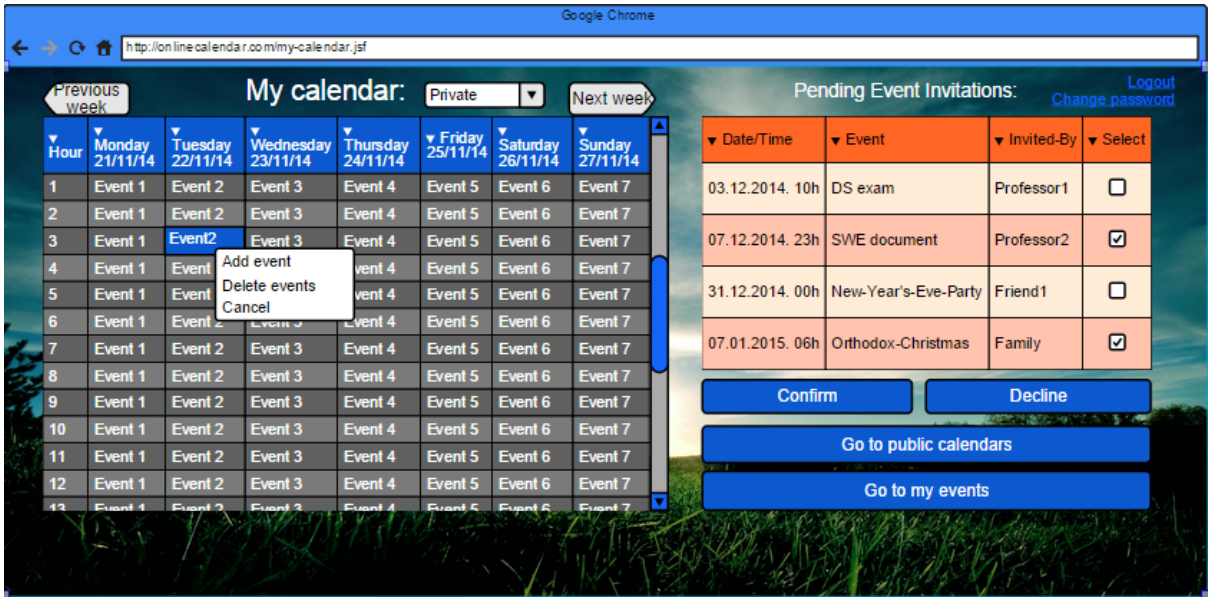


User interface 2: Sign up page of MeteoCal

Milan, 9th November, 2014

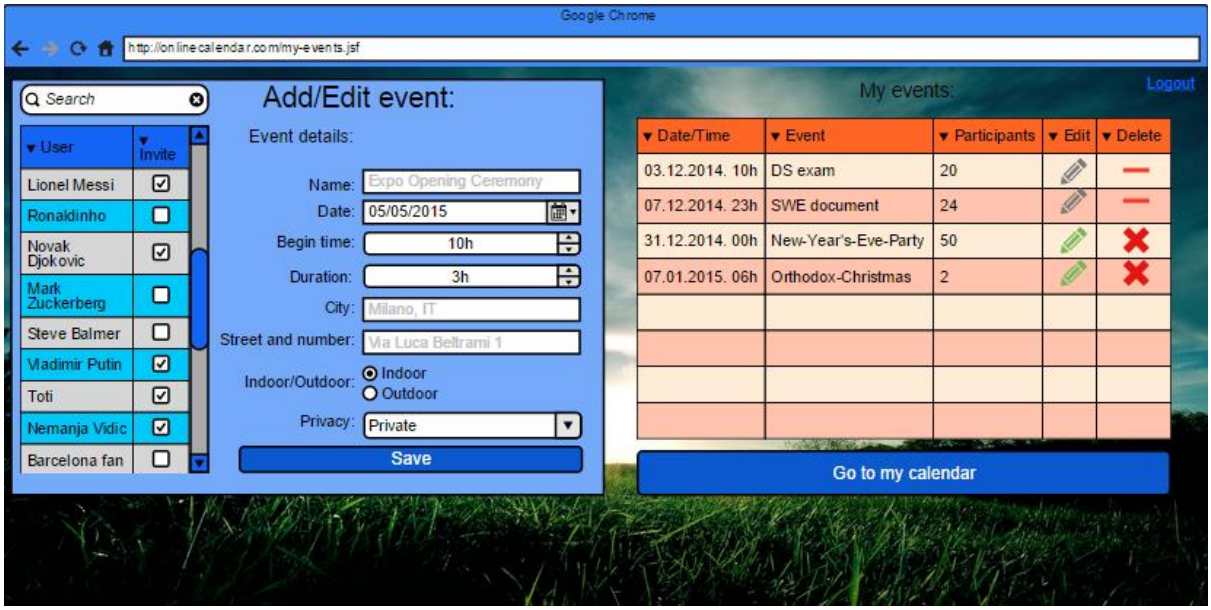


User interface 3: Change password page of MeteoCal

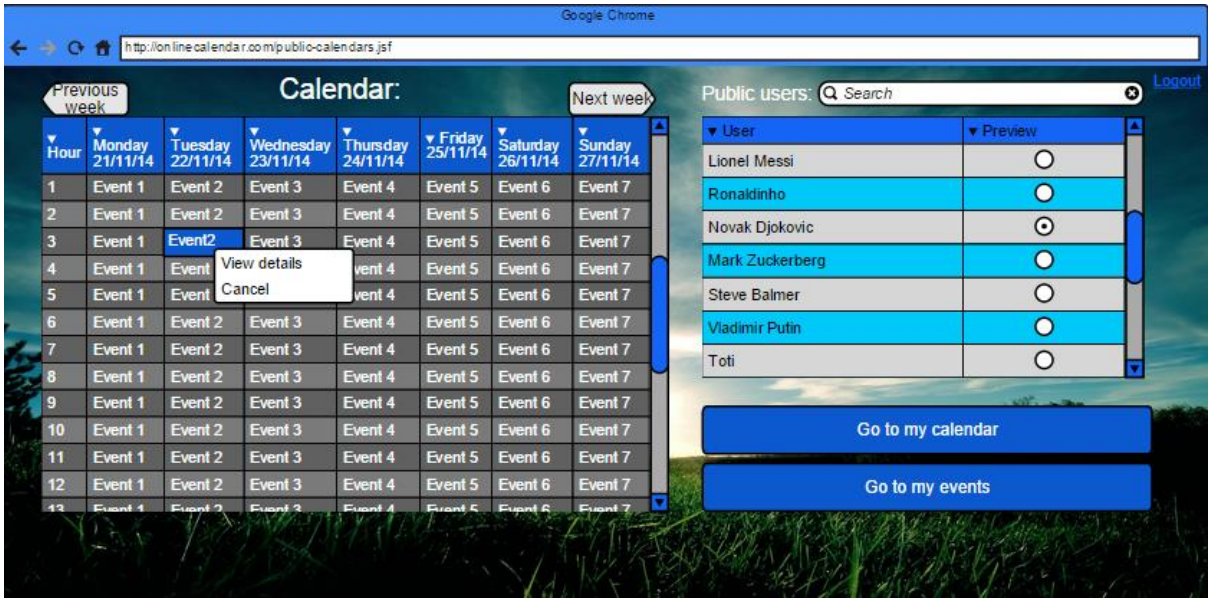


User interface 4: Homepage of MeteoCal

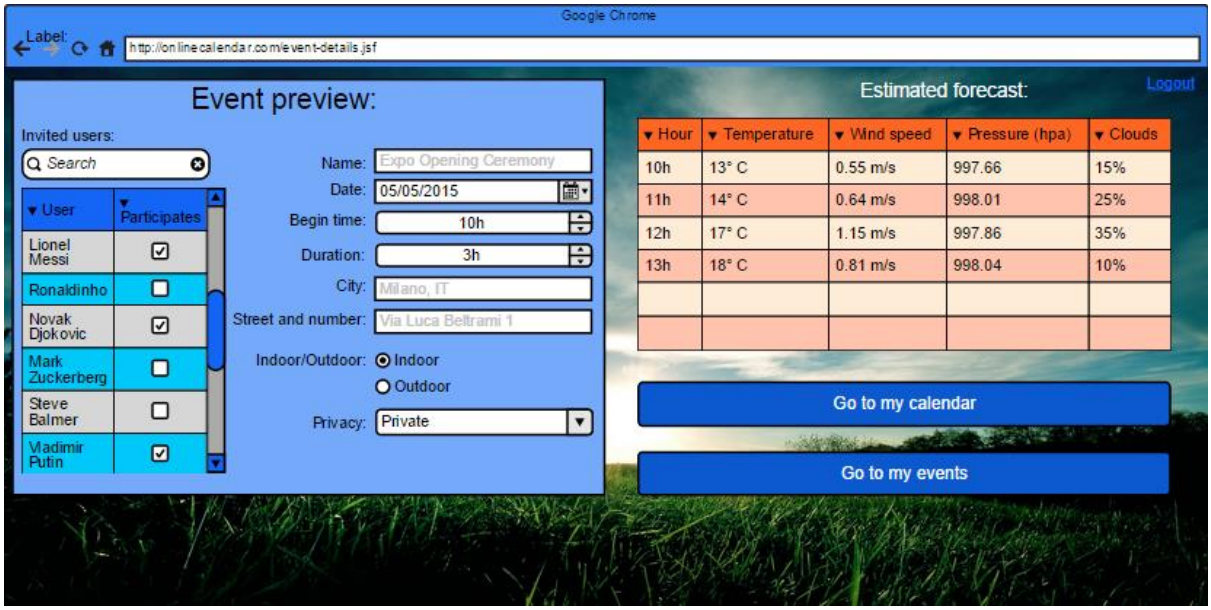
The user can register, login, change his/her password (by clicking on the link sent to his/her email or directly after logging in to the system) and view all the necessary information like the events that he created or the events whose participant he/she is. On the right, there is a list pending invitations that can be either confirmed or declined by the user, also, navigation to own events and public calendars pages.



User interface 5: User's view of the events in his calendar (own and participations).



User interface 6: Page for previewing public calendars.



User interface 7: Preview of an event (with weather cast data added).

2.1.3. Hardware interfaces

The software product does not provide any hardware interfaces.

2.1.4. Software interfaces

- *Database Management system*

  Name: MySQL

  Mnemonic: MySQL

  Specification number: Community Server

  Version number: 5.6.21

  Source: http://dev.mysql.com/downloads/mysql/

- *Application Server*

  Name: GlassFish

  Mnemonic: GlassFish

  Version number: 4.1

  Source: https://glassfish.java.net/download.html

- *Operating System*

  The software product will run on any operating system which supports Java virtual machine, Database Management System and Application Server described above.

2.1.5. Communication interfaces

Protocol	Port	Service
TCP	80	TCP
TCP	3306	MySQL (only if is in a different physical server)

Table 3: Communication interfaces

It is important to note that for the development of the first (current) version of the software we will assume that the Database Management System and Application Server reside on the same physical server.

2.1.6. Memory

The minimum memory requirements are:

- Primary memory: 2GB+
- Secondary memory: 40GB+

Note that the secondary memory is recommended to be physically on a different server from where the software product is installed, as it can exponentially grow without affecting system's performance. For the current production we assume that they are at least installed on the same server.

2.1.7. Operations

A user can interact with the system as a functional user (non-registered user and registered user – citizen and governor). The functional operations for all the users are described in the product functions section.

2.1.8. Site adaption requirements

The product software requires the following conditions to be satisfied in order to run successfully:

- Java Virtual Machine
- Application Server
- Database Management System
- Primary memory required space

- Secondary memory required space

Furthermore, users are required to have installed any of the following web browsers: IE 7.0+, FF 10+, Chrome 20+ or Safari 5+.

2.2. Product functions

This subsection describes a summary of major functions, functional and non-functional requirements of the software product.

2.2.1. General requirements

We have identified 4 main general requirements:

- Managing users
- Managing events
- Managing invitations
- Managing notifications

2.2.2. Functional and non-functional requirements

- Managing users

  Functional requirements:

  [FR1] The system should distinguish users to three categories: citizens, governor, and non-registered.

  [FR2] The non-registered users should be able to register to the system.

  [FR3] The registered users should be able to Login.

  [FR4] The registered users should be able to Logout.

  [FR4] The registered users should be able to change their password.

  [FR5] The registered users should be able to visualize their calendar.

  [FR6] The registered users should be able to visualize public calendars.

  [FR7] The registered users should be able to visualize public event's details in public calendars.

  [FR8] The registered users should be able to visualize the invitations.

  [FR9] The registered users should be able visualize invitation details (event's details to which they are invited).

  [FR10] The registered users should be able to add/delete/update their own events.

[FR11] The registered users should be able to visualize own events and their details.
[FR12] The registered users should be able to visualize weather cast for their own events.
[FR13] The registered users should be able to invite other registered users to their own events.
[FR14] The registered users should be able to accept or decline an invitation to an event.
[FR15] The registered users should be able to remove his/her attendance from event.
[FR16] The registered users should be able to visualize weather cast for the events to which they are invited.

Non-functional requirements:
[NFR1] Users' passwords must be stored securely.
[NFR2] The system must support relatively high number of users.

Managing non-registered users
[FR17] The non-registered users should be able to visualize login page.
[FR18] The non-registered users should be able to visualize sign-up page.

Managing events
[FR19] Events should be able to be created, edited and deleted by their owners.
[FR20] Events have only one owner.
[FR21] Events should be able to be updated by the system when weather cast information changes.
[FR22] Events should be able to be rescheduled by the owners.
[FR23] Event and event's details should be visible to owner and invited registered users.
[FR24] Public events (and their details) inside public calendars should be visible by all registered users.

Non-functional requirements:
[NFR3] Only owners can invite other users participate to an event.

Managing invitations

[FR25] The invitation should be visible to invited registered users.

[FR26] The invitation should be able to be deleted by the system when the event starts.

[FR27] The invitation should not be able to be sent to the same registered user more than once for the same event.

- Managing notifications

Functional requirements:

[FR28] Notifications about bad weather conditions for an event should be sent to owner of the event.

[FR29] Notification about update of event (data updated or rescheduled for another date) is sent to all participants of that event.

[FR30] Notification about deletion of event should be sent to all participants.

Non-functional requirements:

[NFR4] In general, notifications are shown to user only once when he/she logs into the system.

[NFR5] Notifications about bad weather conditions should be shown to owner of event only if he/she login in the period of three days before the event starts.

2.3. User characteristics

The user characteristics are:

- No age restriction.
- Elementary school educational level.
- Knowledge in using a browser.
- Knowledge in using email communication.

2.4. Constraints

The following constraints apply to the software product: must be web based application.

2.4.1. Regulatory policies

The software product does not have to meet any regulatory policies.

2.4.2. Hardware limitations

The software product does not have any hardware limitations.

2.4.3. Interfaces to other applications

The software communicates with weather cast API.

2.4.4. Parallel operation

The software product must support the operation of simultaneous users especially when working with data for the elections and parties.

2.4.5. Audit functions

The software does not perform any audit.

2.4.6. Control functions

The software product does not control any device or any other system.

2.4.7. Higher – order language requirements

The software product requires basic knowledge of HTML, Java and JEE technologies.

2.4.8. Signal handshake protocols

The software product does not manage any handshake protocol.

2.4.9. Reliability requirements

The software product does not require any specific requirements to perform and maintain its functions under normal operation.

2.4.10. Criticality of the applications

The software product requires proper support for concurrent users.

2.4.11. Safety and security considerations

The software product does not require any safety and security considerations.

2.5. Assumptions and Dependencies

The requirements in this document are grounded on the following assumptions:

- The Java virtual machine is already installed on the operating system.
- Users have access to a decent Internet connection.
- The software product provides a governor by default.
- The software product supports any number of citizens and parties.

2.6. Apportioning of requirements

Future releases of the software product may provide support for:

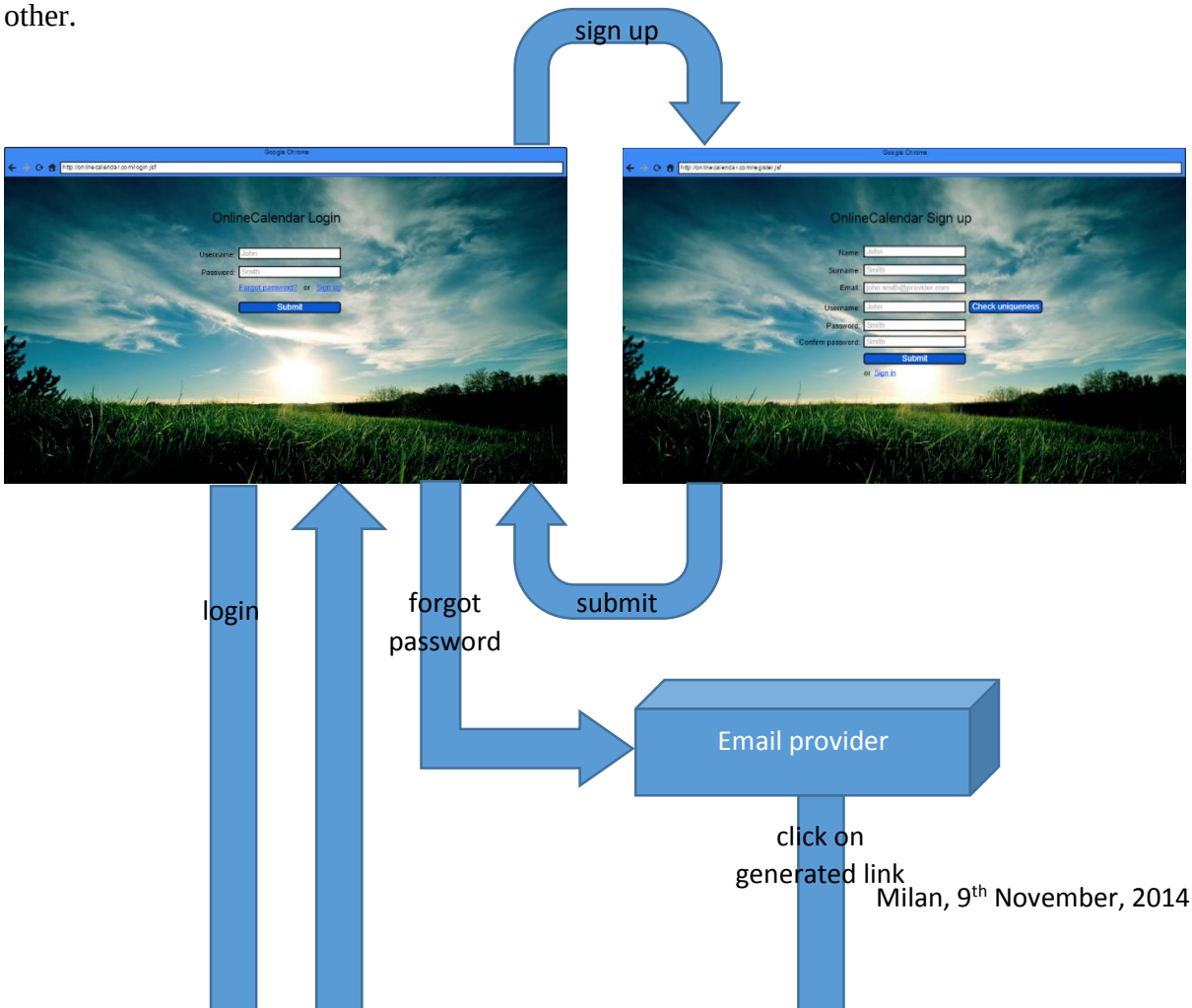
1. Password modifying and resetting.
2. Show statistical analysis of the elections data.
3. Implement a forum for the party members.

3. Specific requirements

3.1. External interface requirements

3.1.1. User interfaces

This story board explains how previously mentioned user interfaces are connected with each other.



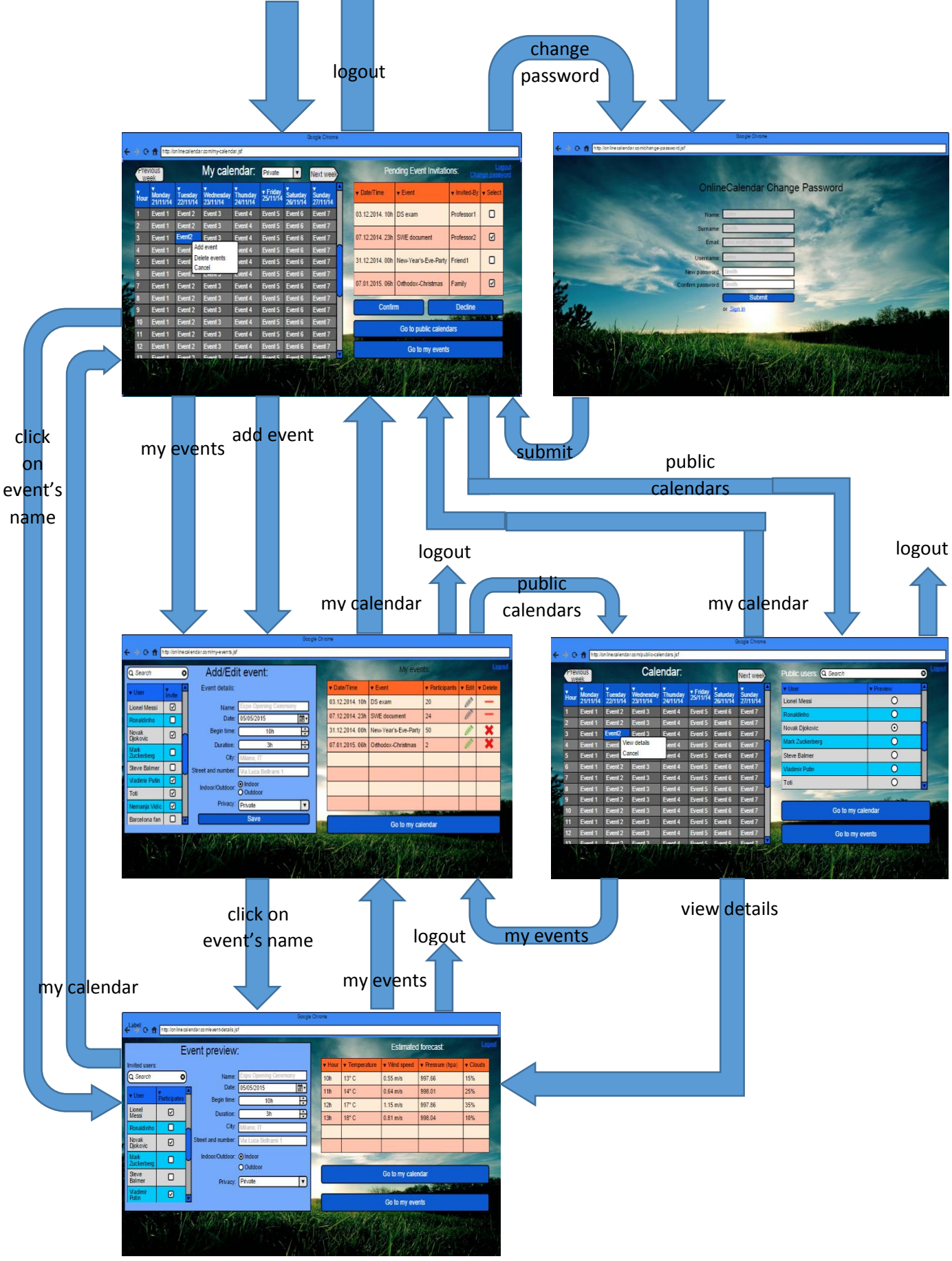


Figure 1: Story board of the user interface.

Milan, 9th November, 2014

3.1.2. Hardware interfaces

The system does not have hardware interfaces.

3.1.3. Software interfaces

The system does not have software interfaces.

3.1.4. Communication interfaces

The system does not have communication interfaces.

3.2. Functional requirements

3.2.1. Scenarios

3.2.1.1. Non-registered user registers in the system

Code	SCS – 001
Description	Describing how non-registered user registers in the system
Goal	User successfully registers to the system
Assumption	1. User is not registered.
Marco wants to register on the new system for management of a personal calendar. He goes to Google and searches for such system, in the result page he finds MeteoCal web application and decides to try it out. When he followed the link he appeared on the log in page of MeteoCal. There he found out the link 'Sign Up' which leads him to register page of the system. On this page he needs to fill in data such as name, surname, username, password and email address. Since the system uses username as a logical ID of an user, it is necessary to check for the uniqueness of username of registering user, if the username is not unique such username needs to be changed. After all the data is correctly inserted into the page form, Marco clicks on the 'Submit'. This action leads him back to log in page. Now he can use his credentials to log into the system.	

3.2.1.2. Registered user logs in on the system

Code	SCS – 002
Description	Registered user wants to log in.
Goal	User successfully logs into the system.
Assumption	1. User is registered
After registering, Marco is again on the log in page of the system. He writes the username and the password in the corresponding fields and clicks on the button "Sign in". **The system shows the log in page again with a note that some of the login info is wrong. Apparently Marco typed wrong the password. He rewrites again the information and clicks on the button for logging in. This time he entered the information correctly, the system shows the my calendar page where he can start using functionalities of MeteoCal.**	

3.2.1.3. Registered user creates a calendar

Code	SCS – 003
Description	Registered user uses MeteoCal to create a calendar
Goal	
Assumption	1. User is registered 2. User logs in the system for the first time.
Alessandro registered and he wants to create his calendar on our system. On his first successful logging in the system a calendar is automatically build by business logic. This calendar is completely empty and it is up to the user to populate it with events as his needs dictate.	

3.2.1.4. Registered user changes privacy of his calendar

Code	SCS – 004
Description	Registered user wants to change his calendar's privacy.
Goal	Calendar's privacy is successfully changed.
Assumption	1. User is registered 2. User is logged in.
Beatrice wants to make her calendar public, and hence visible to all other users of the system. She navigates herself to my calendar page and there finds drop down menu labelled as 'public' or 'private' depending on the current state of the privacy settings of the calendar, when she selects this drop down menu she can choose between 'public' and 'private' in accordance to her needs. After she selects 'public' her calendar is marked as public in our system. In the other case, if Beatrice wants to makes her calendar back to private, the procedure is the same the only difference is in the value selected in the drop down menu.	

3.2.1.5. Registered user creates a new event

Code	SCS – 005
Description	Registered user creates a new event
Goal	User creates event if there are no consistency violations.
Assumption	1. User is registered 2. User is logged in.
Chiara is logged into the system and she wants to create an event for her birthday party and invite her friends Angelo and Beatrice for whom she knows have an account at MeteoCal. Chiara first goes to my calendar page, there she selects date May the 6th, which can be easily done by using two buttons over the calendar display space (previous week button and next week button). When she has selected the date, she searches for the appropriate time of the day, she selects 8:00 pm. She uses right click on the mouse while cursor is over the tile labelled '8:00 pm', in the drop down menu she selects 'add event'. The option 'add event' is greyed out if Chiara already has an event in that time slot, whether the time slot being occupied by other Chiara event or by an event at which	

Chiara has accepted invitation to. If the 'add event' option is viable Chiara will click on it, otherwise she will search for other available time slot. This operation will lead her to my events page, with data fields for date and time already automatically filled in with data corresponding to the time tile she clicked on. Here she can search for Angelo and Beatrice and add them from the list that appears next to the mandatory data for the event. After click on the 'save' button, the event is created, added to her events, invited users are notified by the system, and the weather forecast is obtained by the system in the case she chose her party to be outdoors.

3.2.1.6. Registered user unsubscribes from a party

Registered user edits event	
Code	SCS – 006
Description	User edits event he created at earlier time.
Goal	The event data is changed.
Assumptions	1. The user is a citizen 2. The user is registered and logged in to the system 3. The user is the owner of the event he/she wants to chage
Riccardo has made an event couple of days ago, he remembers that he forgot to invite some of the people using MeteoCal (also any other data can be changed in this way). He navigates himself to my events page, there he searches for his event and clicks on the edit button next to the event name. This operation will lead him to event details page where he can edit any data concerning the event, such as date, time, outdoor/indoors option, etc. as long as it doesn't induce some data consistency violations (system will automatically check for those). There he searches for people he wants to add to existing event, after adding them he clicks on 'save' button. This operation will save the changes to event, also it will make the system notify all the invited users of the event data changed and send invitations to newly added people. If the only change Riccardo does is to invite new people, other event participants won't	

be notified by the system, the notification will be received by all participants only in case of time,date or location changes.

3.2.1.7. Registered user deletes event

Code	SCS - 007
Description	User decides to delete event he created at earlier time.
Goal	The event is deleted from the system
Assumptions	1. User is registered 2. User is logged into the system 3. User is the owner of the event
Angelo created the event a couple of weeks ago, he had some personal issues and now he needs to cancel the event. Angelo navigates to my events page and in the list of all the events, events he created or events he is attending, he finds the event he wants to delete. Next to the name of the event there is a delete button, Angelo clicks on this button and the event is deleted. After deleting the event system also notifies all the participants of the event of the cancelation, and removes the pending invitations for the users that haven't already decided about whether they are participating in the event or not.	

3.2.1.8. Registered user changes event privacy

Code	SCS – 008
Description	User wants make his event details publically available
Goal	Users event is marked as public event
Assumptions	1. The user is registered 2. The user is logged in 3. The user is owner of the event
Luca wants to make his event public. He goes to event details and edits the event's privacy settings. He does this by clicking on the radio button labelled as 'public' and then he clicks on the 'save' button. This operation will make the event public. If Luca's	

calendar is marked as public, event details will be available to all other users of the system along with participant list.

3.2.1.9. Registered user browse through public calendars

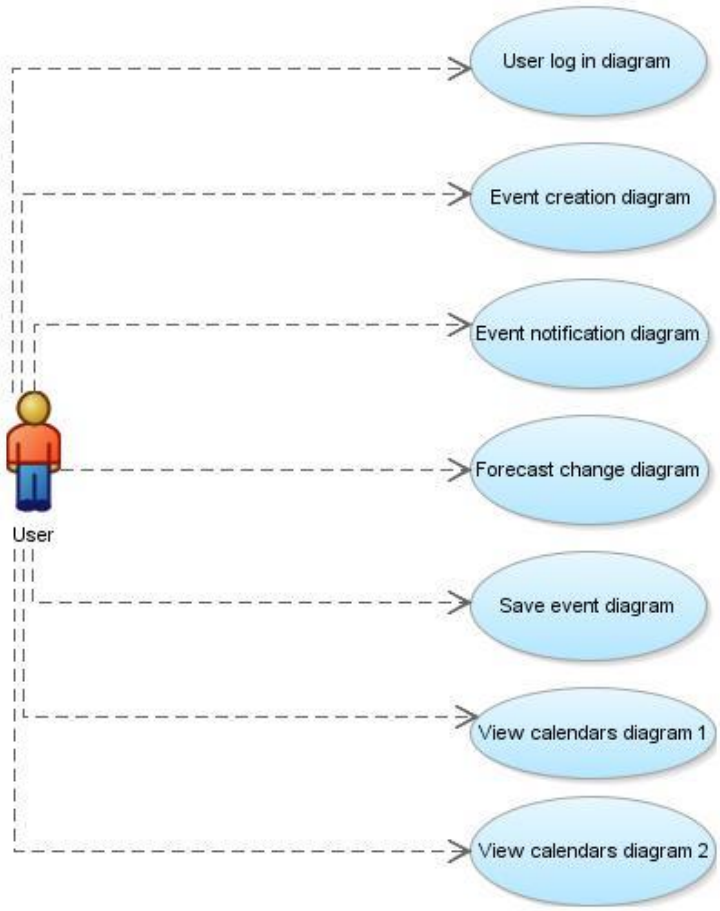
Code	SCS – 009
Description	User wants to look to other user's calendar
Goal	User browse events in the other user's calendar
Assumptions	1 The user is registered 2 The user is logged in
Sergio decides to see the calendar of Pablo for whom he knows has made his calendar public, and hence he can see when Pablo is free. Sergio goes to public calendars page and there he searches for Pablo, in the list of users he selects Pablo's name and Pablo's calendar is presented on the page. From there Sergio can see when Pablo is busy, if Pablo's events are private, or see Pablo's events details if they are marked as public.	

3.2.1.10. Registered user accept/decline event invitation

Code	SCS – 010
Description	User accepts/declines event invitation
Goal	Pending invitation to the user is processed
Assumptions	1. The user is registered 2. The user is logged in 3. The user has a pending invitation
Luca received a notification by the system that he has been invited to an event. He goes to my events page and there he searches for the event he has been notified about. After finding the event, next to its name there are two buttons which are used by Luca to either accept event's invitation or decline it. In the case of accepting this event is added to his calendar and he can see event details, in the case of declining this event is removed from the list of his events, and he will receive no further notification about this event.	

3.2.2. Use case model

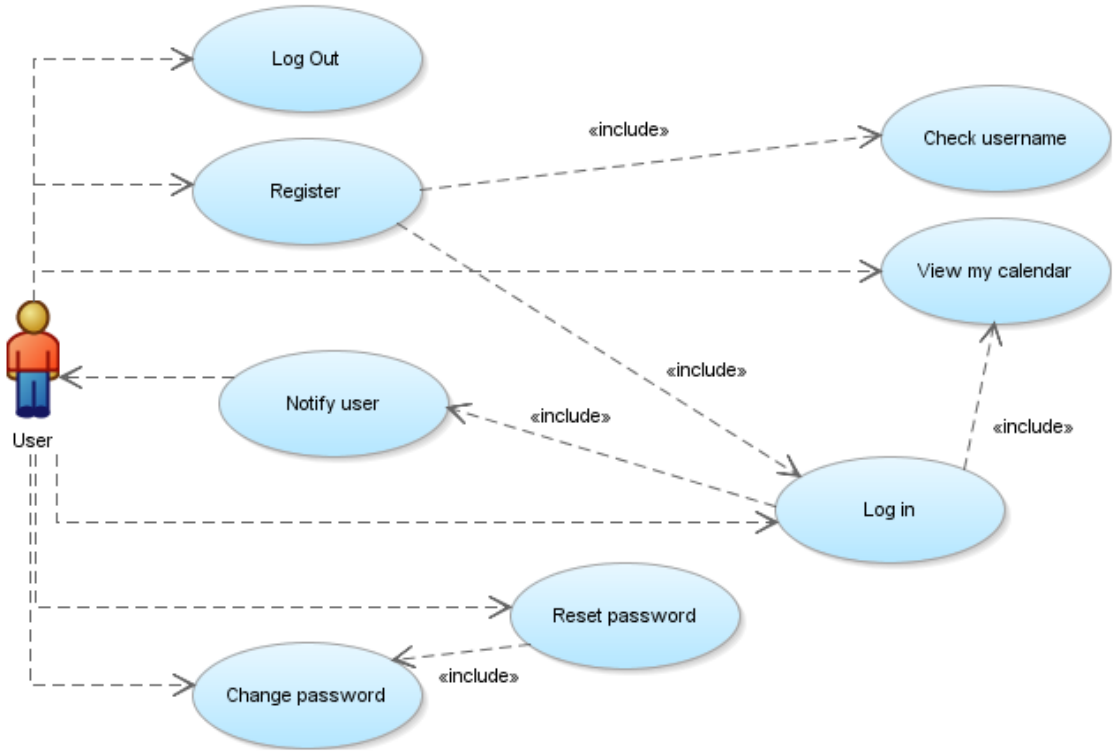
We describe in a detailed way below the main use cases. Some use cases that extends other use cases and some others are omitted because they are really similar to others. It is important to understand that all references to “pages”, “buttons” or “input forms” are only hypothesis to make the situation clearer and to help the reader to draw a visual picture in his mind of what we are talking about, real pages and page structures will be well defined in the Design Document. The software contains one main actor: Registered user (just user from now on). We decided to split the Use Case Diagram into smaller ones because we wanted to make the situation clearer. We can provide some “macro Use Cases” below (this is not a Use Case Diagram, but only a diagram that helps the reader to understand the composition of the diagrams drawn below):



Use Case Diagram 1: General ‘Use case schema’, simplification and division of the real use case diagram in order to present the information in more concise way.

We continue our use case description of the system by exploaring each of the ‘macro use case’ fields in the general use case schema.

3.2.2.1. User log in diagram



Use Case Diagram 2: User log in diagram, it explains in more details the use case of log in and log in related actions in the system.

Name	Log In, USC-001
Actors	User
Entry Conditions	The user is located at log in page of the MeteoCal system
Flow of events	• User opens the log in page of the MeteoCal system • User fills in the username and password fields in the form • User clicks on the ‘Sign in’ button
Exit Conditions	The user has successfully logged into the MeteoCal system
Exceptions	• User is not registered • Provided username is not correct • Provided password is not correct

Name	Log Out, USC-002
Actors	User
Entry Conditions	The user is located at any page of the MeteoCal system
Flow of events	• Use clicks on the 'Log out' link • The system logs out the user • The system generates log in page and presents it to the user
Exit Conditions	The user has successfully logged out of the MeteoCal system
Exceptions	• No exceptions

Name	Register, USC-003
Actors	User
Entry Conditions	The user is located at log in page of the MeteoCal system
Flow of events	• User opens the log in page of the MeteoCal system • User clicks on the 'Sign up' link • User is transferred to register page of the MeteoCal system • User fills in the data needed for registration • User clicks on Submit
Exit Conditions	The user has been successfully registered into the MeteoCal system
Exceptions	• Username not unique • Password fields don't match one to another • Email address already assigned to a user

Name	Check username, USC-004
Actors	User
Entry Conditions	The user is located at register page of the MeteoCal system
Flow of events	• User fills in the data needed for registration • User clicks on the 'Check uniqueness' button • User enters new username if needed
Exit Conditions	The user has chosen the unique username
Exceptions	• No exceptions

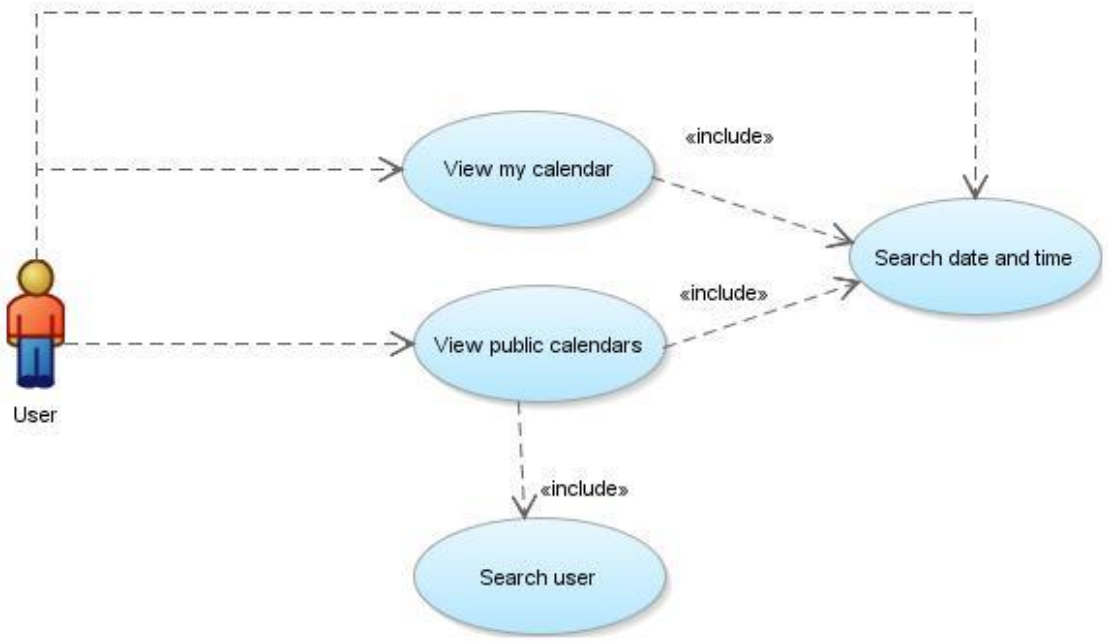
Name	Reset password, USC-005
Actors	User
Entry Conditions	The user is located at log in page of the MeteoCal system
Flow of events	• User clicks on the 'forgot password' link • The system compiles and sends to the user a link to temporary page used for the password reset operation • User opens the email and follows the link to temporary page • User enters new password • User clicks on the 'Submit' button
Exit Conditions	The user has successfully reset his password
Exceptions	• Password fields don't match one to another

Name	Change password, USC-006
Actors	User
Entry Conditions	The user is located at my calendar page of the MeteoCal system
Flow of events	• User clicks on the 'change password' link • The system compiles and sends to the user a link to temporary page used for the password reset operation • User opens the email and follows the link to temporary page • User enters new password • User clicks on the 'Submit' button
Exit Conditions	The user has successfully reset his password
Exceptions	• Password fields don't match one to another

Name	Notify user(Log in), USC-007
Actors	User
Entry Conditions	The user is located at log in page of the MeteoCal system
Flow of events	• User log into the MeteoCal system • The system checks users ID and search for pending notifications intended for the user • The system sends notifications to the user • User browse through notifications
Exit Conditions	The user has successfully received notifications
Exceptions	• No exceptions

Name	View my calendar(Log in), USC-008
Actors	User
Entry Conditions	The user is located at log in page of the MeteoCal system
Flow of events	• User log into the MeteoCal system • The system checks users ID and search for users calendar and events • The system prepares my calendar page with users data embedded inside the page • User browse through his calendar
Exit Conditions	The user has successfully seen his calendar
Exceptions	• No exceptions

3.2.2.2. *View calendars diagram*



Use case diagram 3: View calendars diagram, explains into the details use case diagrams concerning calendars and the relations between them

Name	View my calendar(Public Calendars), USC-009
Actors	User
Entry Conditions	The user is located at public calendars page of the MeteoCal system
Flow of events	• User clicks on the 'Go to my calendar' button • The system checks users ID and search for users calendar and events • The system prepares my calendar page with users data embedded inside the page • User browse through his calendar
Exit Conditions	The user has successfully seen his calendar
Exceptions	• No exceptions

Name	View public calendar, USC-010
Actors	User
Entry Conditions	The user is located at my calendar page of the MeteoCal system
Flow of events	• User clicks on the 'Go to public calendars' button • The system searches for users calendars and events which are marked as public • The system prepares public calendars page with previously collected data embedded inside the page • User browse through public calendars
Exit Conditions	The user has successfully seen public calendars
Exceptions	• No exceptions

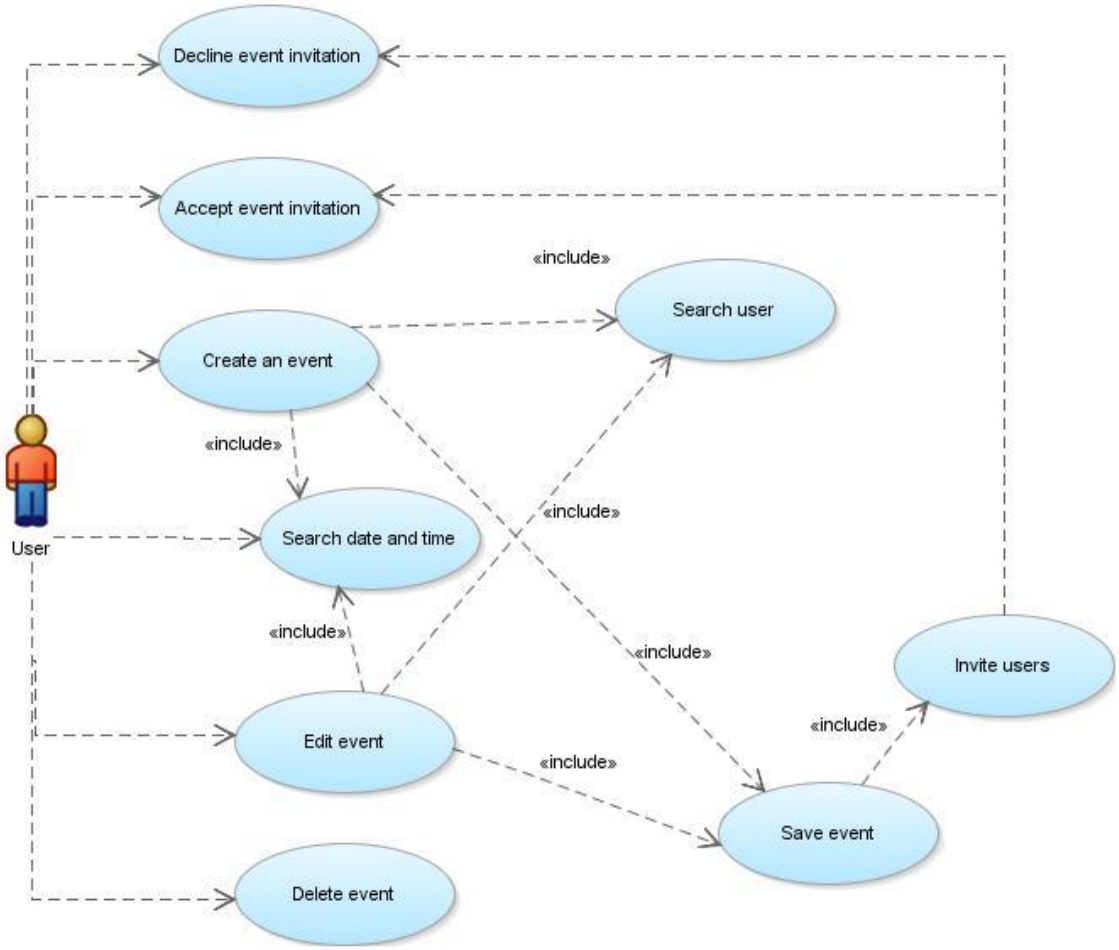
Name	Search users(Public Calendars), USC-011
Actors	User
Entry Conditions	The user is located at public calendars page of the MeteoCal system
Flow of events	• User clicks on the search bar • User enters the name on which he likes to conduct the search • The system searches for users which satisfy the query • The system repopulates the list of users with public calendars based on the results of the query • The system prepares public calendars page with previously collected data embedded inside the page • User clicks on the desired result • Public calendar of selected user has been presented
Exit Conditions	The user has successfully searched for the other user
Exceptions	• No exceptions

Name	Search time and date, USC-012
Actors	User
Entry Conditions	• The user is located at public calendars page of the MeteoCal system • The user is located at my calendar page of the MeteoCal system

Flow of events	• User clicks on :
	  ○ Next week button
	  ○ Previous week button
	• User navigates to desired week
	• User navigates to desired day
	• User navigates to desired time slot

Exit Conditions	The user has successfully searched through the calendars
Exceptions	• No exceptions

3.2.2.3. Event creation diagram



Use case diagram 4: Event creation diagram, explains into the detail use case diagrams related to creation and deletion of event, also it models the inter-dependencies between use cases

Name	Search user(Event Details), USC-013
Actors	User
Entry Conditions	The user is located at event details page of the MeteoCal system
Flow of events	• User clicks on the search bar • User enters the name on which he likes to conduct the search • The system searches for users which satisfy the query • The system repopulates the list of users with public calendars based on the results of the query • User selects users he wants to invite by clicking on the check box next to each result
Exit Conditions	The user has successfully searched for the users
Exceptions	• No exceptions

Name	Invite users, USC-014
Actors	User
Entry Conditions	The user is located at event details page of the MeteoCal system
Flow of events	• User clicks on the 'Save' button • The system checks for newly added users in the event • The system notifies newly added users that they have been invited to event
Exit Conditions	The users have been invited successfully
Exceptions	• No exceptions

Name	Save event, USC-015
Actors	User
Entry Conditions	The user is located at event details page of the MeteoCal system
Flow of events	• User fills in all necessary data in the form • User selects the users he wants to invite to the event • User clicks on the 'Save' button • The system queries the Weather forecast API for the weather forecast related to the event • The system creates/saves the event • The system notifies invited user about the event changes • The system invites users
Exit Conditions	The user has successfully saved the event
Exceptions	• Data consistency errors: the 'Save' button will be greyed out in the case of the interleaving events and other data inconsistencies

Name	Create event, USC-016
Actors	User
Entry Conditions	The user is located at my calendar page of the MeteoCal system
Flow of events	• User searches date and time • User right clicks on the desired time slot • User select 'add event' option from the drop down menu • The system transfers user to event details page • The system fills in date and time fields in the form based on the time slot users has clicked on • User fills in other data in the form and sets privacy • User select other users he wants to invite to the event • User clicks on the 'Save button' • The system saves the event
Exit Conditions	The user has successfully created the event
Exceptions	• Data consistency errors: the 'Save' button will be greyed out in the case of the interleaving events and other data inconsistencies

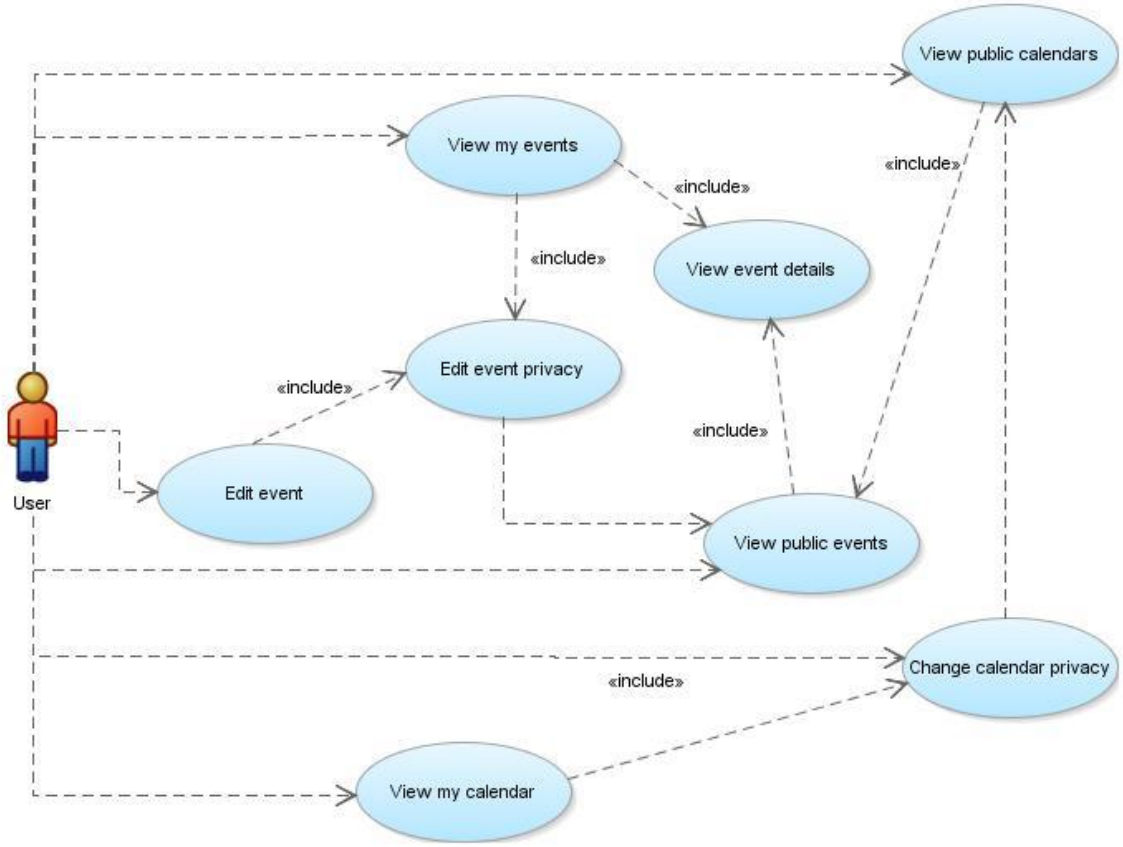
Name	Edit event, USC-017
Actors	User
Entry Conditions	The user is located at my events page of the MeteoCal system
Flow of events	• User navigates through the list of his events • User selects the event he wants to edit • User clicks on 'Edit' button next to event name • The system transfers user to event details page • The system loads event data into the form based on the event users has clicked on • User edits data of the event • User select other users he wants to invite/remove to/from the event • User clicks on the 'Save button' • The system saves the event
Exit Conditions	The user has successfully edited the event
Exceptions	• Data consistency errors: the 'Save' button will be greyed out in the case of the interleaving events and other data inconsistencies

Name	Delete event(My Calendar), USC-018
Actors	User
Entry Conditions	The user is located at my calendar page of the MeteoCal system
Flow of events	• User searches date and time • User selects the event he wants to delete • User right clicks on the event in his calendar • User selects 'delete event' option from the drop down menu • The system checks the ownership of the event • If the user is the owner the event is deleted from the system and all participants are notified of event's deletion • If the user is not the owner the event is just removed from the user's event list, and the user is treated as he have been declined the invitation
Exit Conditions	The user has successfully deleted the event
Exceptions	• No exceptions

Name	Delete event(My Events), USC-019
Actors	User
Entry Conditions	The user is located at my events page of the MeteoCal system
Flow of events	• User navigates through the list of his events • User selects the event he wants to delete • User clicks on 'Delete' button next to event name • The system checks the ownership of the event • If the user is the owner the event is deleted from the system and all participants are notified of event's deletion • If the user is not the owner the event is just removed from the user's event list, and the user is treated as he have been declined the invitation
Exit Conditions	The user has successfully deleted the event
Exceptions	• No exceptions

Name	Accept/Decline event invitation, USC-020
Actors	User
Entry Conditions	The user is located at any page of the MeteoCal system
Flow of events	• User receives notification from the system • User navigates to my events page • On my events page user browse for new event that he hasn't respond to yet • Next to the event name there are buttons 'Accept' and 'Decline' • User clicks on 'Accept'/'Decline' button and correspondingly to the choice the event is added/removed to his list of events
Exit Conditions	The user has successfully accepted/declined the event invitation
Exceptions	• No exceptions

3.2.2.4. View calendars 2 diagram



Use case diagram 5: View calendars 2 diagram, adds necessary connections between use case diagrams of calendars with those of events

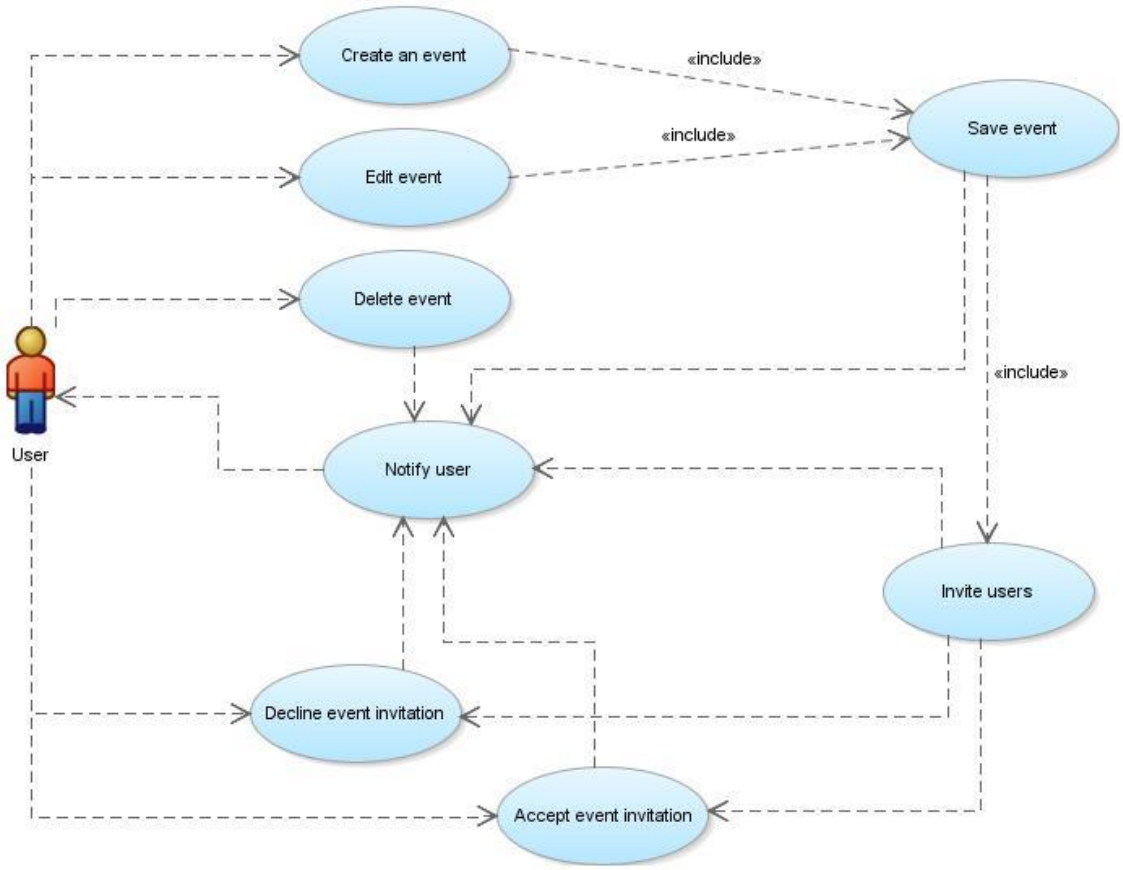
Name	Change calendar privacy, USC-021
Actors	User
Entry Conditions	The user is located at my calendar page of the MeteoCal system
Flow of events	• User clicks on the drop down menu located over the calendar presentation portion of the page • The menu presents two choices Public/Private • User selects one and clicks on it • If the user has clicked on the 'Public' option system marks his calendar as public and this calendar will be available in public calendars page • If the user has clicked on the 'Private' option system marks his calendar as private and this calendar can only be seen by the user (owner).
Exit Conditions	The user has successfully changed the privacy of his calendar
Exceptions	• No exceptions

Name	Edit event privacy, USC-022
Actors	User
Entry Conditions	The user is located at event details page of the MeteoCal system
Flow of events	• User clicks on the drop down menu located in the bottom of the form that contains the events details • The menu presents two choices Public/Private • User selects one and clicks on it • If the user has clicked on the 'Public' option system marks his event as public and this event's details will be available in public calendars page if the encapsulating calendar is also marked as public • If the user has clicked on the 'Private' option system marks his event as private and this events details can only be seen by the user (owner).
Exit Conditions	The user has successfully changed the privacy of his event
Exceptions	• No exceptions

Name	View my events, USC-023
Actors	User
Entry Conditions	The user is located at my calendar page of the MeteoCal system
Flow of events	• User clicks on the 'Go to my events' button • System transports the user to the my events page • User browse through his events in the list located on the right hand side of the screen
Exit Conditions	The user has successfully seen his event list
Exceptions	• No exceptions

Name	View public events, USC-024
Actors	User
Entry Conditions	The user is located at public calendars page of the MeteoCal system
Flow of events	• User searches for the other user's calendar • User searches for the time and date • User finds in the navigated week events marked in different color • User right clicks on the desired public event • The system transfers the user to event details page (all editing options are disabled in this view)
Exit Conditions	The user has successfully seen public events and is able to go to public events details
Exceptions	• No exceptions

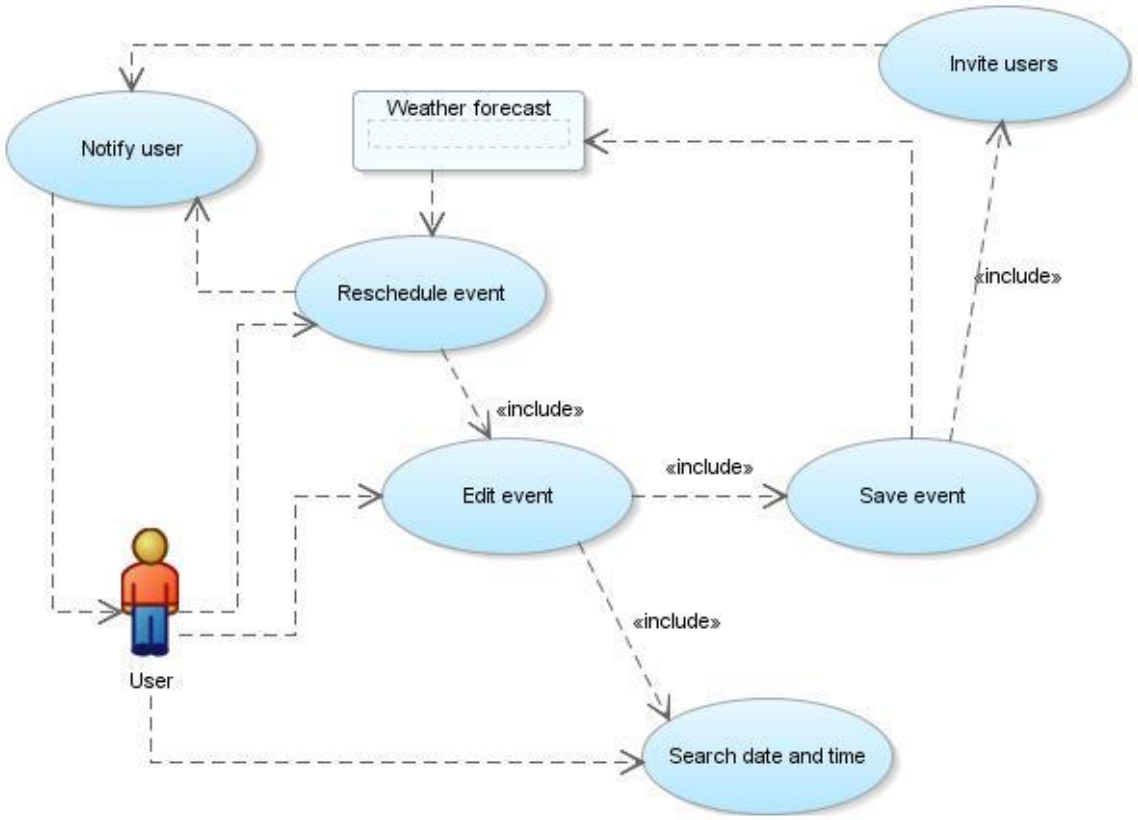
3.2.2.5. Event notification diagram



Use case diagram 6: Event notification diagram models in more details the inter-dependencies of the event related use cases

This diagram doesn't induce any new use case that hasn't been already explained. Only purpose of this diagram is the modelling of the include dependencies and logical dependencies in the system. For example, each time user commits 'Invite users' use case, other system users that were invited to event will commit 'Accept'/'Decline' use case.

3.2.3.6. Forecast change diagram

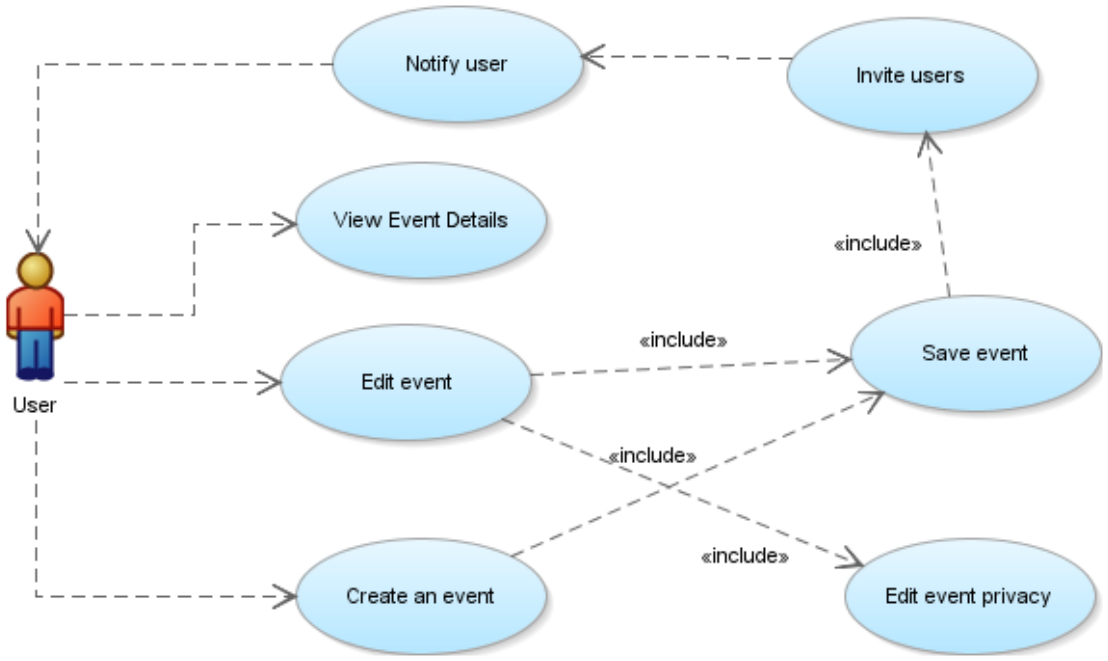


Use case diagram 7: Forecast change diagram models the intra-dependencies of the MeteoCal system and the outside weather forecast API, and the use cases related to this particular aspect of the system

Weather forecast block inside this diagram is not a use case it is simple abstraction of the outside element, which can be seen as a topic of interest to our system and its users. The arcs connecting the weather forecast API to some other use cases represent the logical dependencies. For example, after any event is saved the system subscribes to the API in attempts to fetch the forecast for that day, or if the forecast is bad it can activate the 'Reschedule event' use case.

Name	Reschedule event, USC-025
Actors	User
Entry Conditions	• The user is located at any page of the MeteoCal system • The user is the owner of the event E1 • The event E1 is scheduled to happen in 3 days • The event E1 is outdoor event
Flow of events	• The system receives the forecast for the event, and it's bad weather • User receives the notification, along with suggested best new date in next 15 days • User is transferred to edit event page • User edits event's date and/or location of the event for 'indoors'
Exit Conditions	The user has successfully rescheduled the event
Exceptions	• There are no sunny days in next 15 days: user can either change the event to be 'indoors' or cancel the event

3.2.3.7. Save event diagram



Use case diagram 8: Save event diagram explains relations between user and storing event data into the system.

Again, like in the case with 'Event notification diagram', this diagram doesn't induce any new use cases, it has been depicted with the sole purpose of showing relations of some use cases in more simple diagram, with some other use cases being omitted.

Name	View event details, USC-026
Actors	User
Entry Conditions	• The user is located at my events page of the MeteoCal system • The user is located at public calendars page of the MeteoCal system

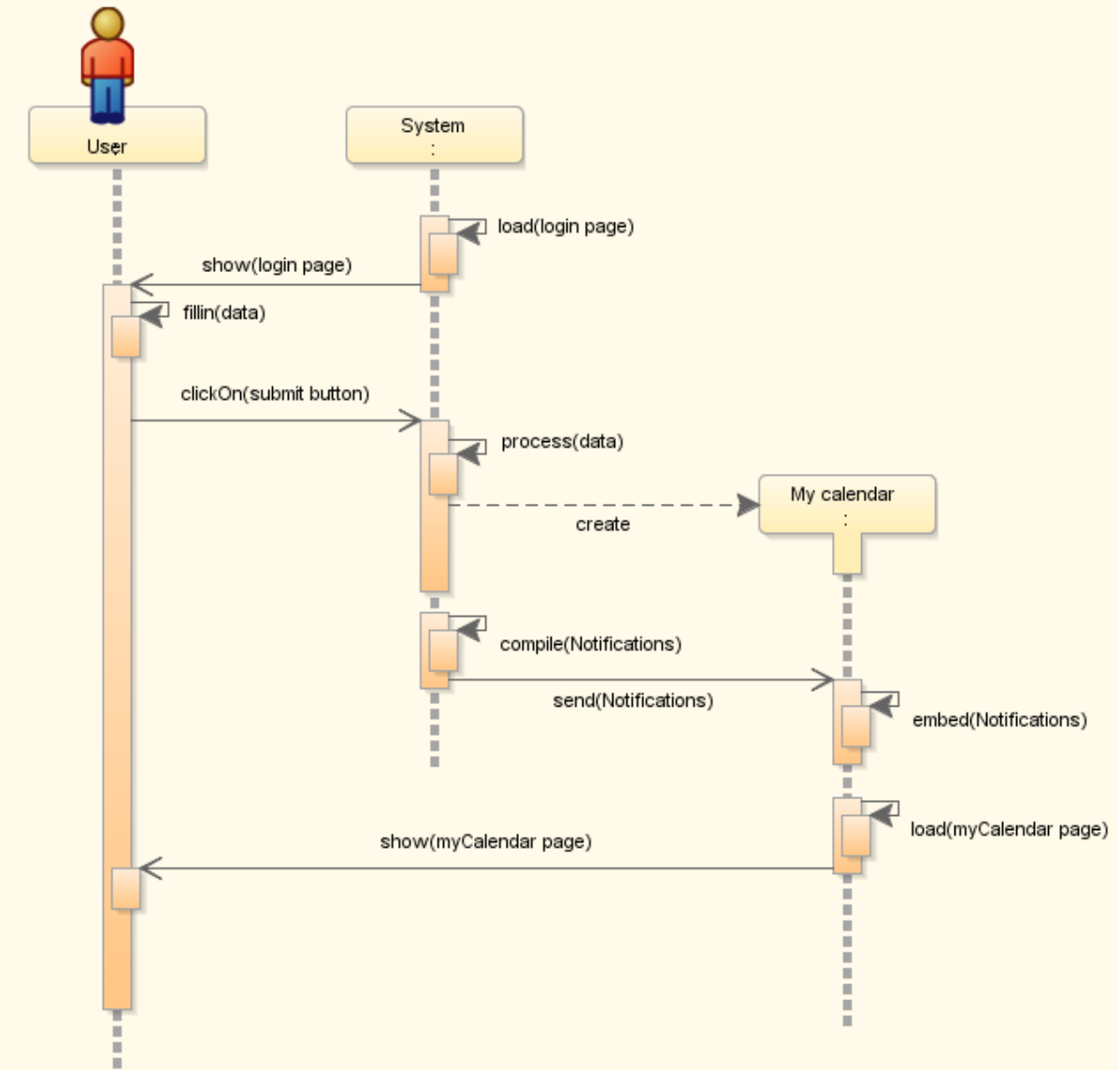
Flow of events	• My events page case:
	  ○ User browse through the list of his events
	  ○ User clicks on the 'details' button next to the name of the event
	  ○ User is transferred to event details page
	• Public calendars page case:
	  ○ User searches for other user
	  ○ User searches for date and time
	  ○ User right clicks on desired event
	  ○ User selects 'event details' option
	  ○ User is transferred to event details page

Exit Conditions	The user has successfully seen event details
Exceptions	• No exceptions

3.2.4. Sequence model

In this section we introduce sequence diagrams that correspond to use cases described in previous section. We made simplifications for some frequent sequences of events, which will be explained in detail when they are depicted. Also if we have used simplified event sequences inside of a sequence diagram, we will present small table of references to make the understanding of diagrams as easy as possible.

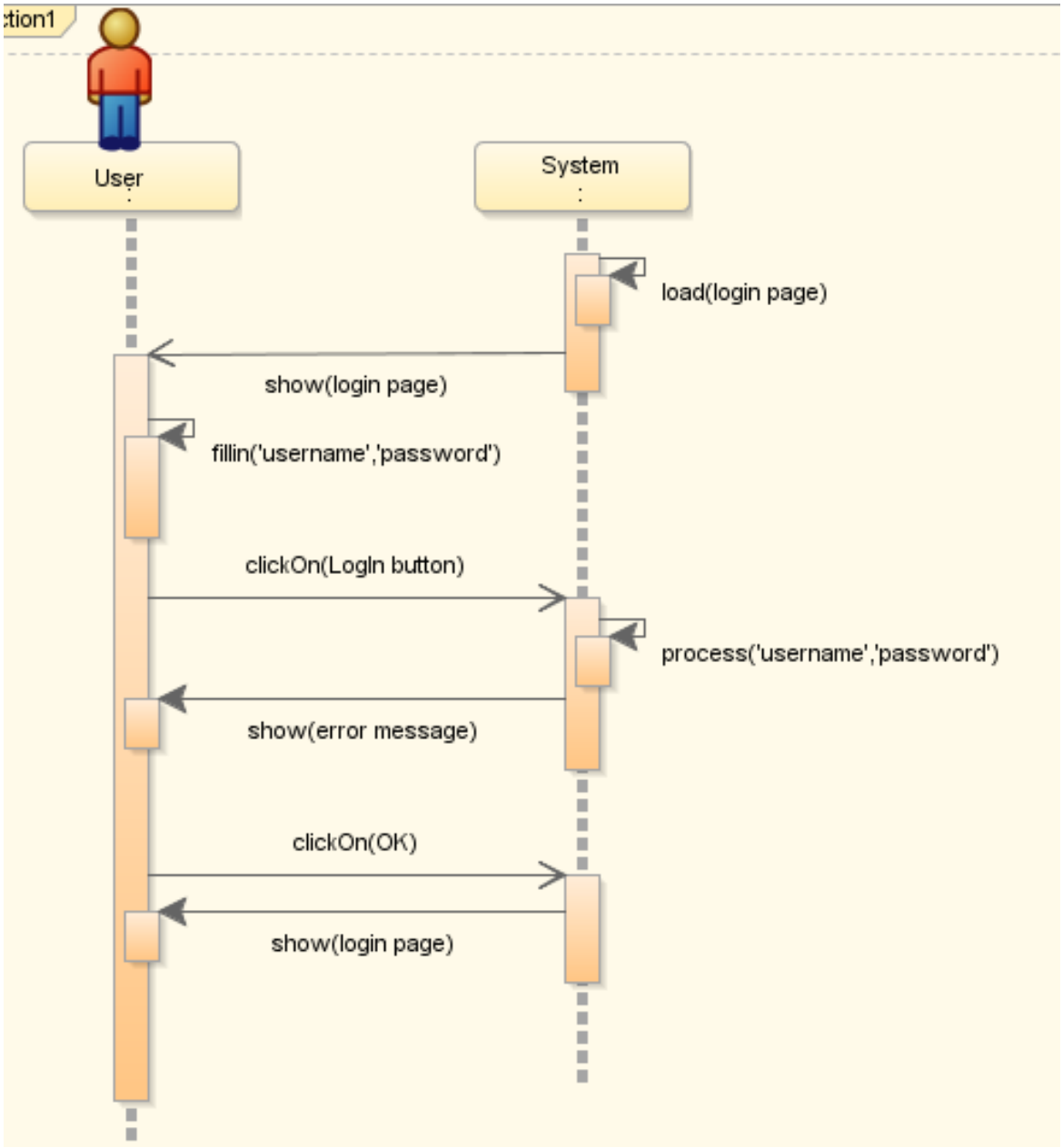
3.2.4.1. Log In-Success diagram



Sequence diagram 1: Log in sequence diagram in the case the log in operation has been successful

Use case diagram reference	• SD-001
Name	• Log In-Success diagram
Use case diagram reference	• USC-001(Log In) • USC-007(Notify(Log In))
Simplification reference	• No simplifications

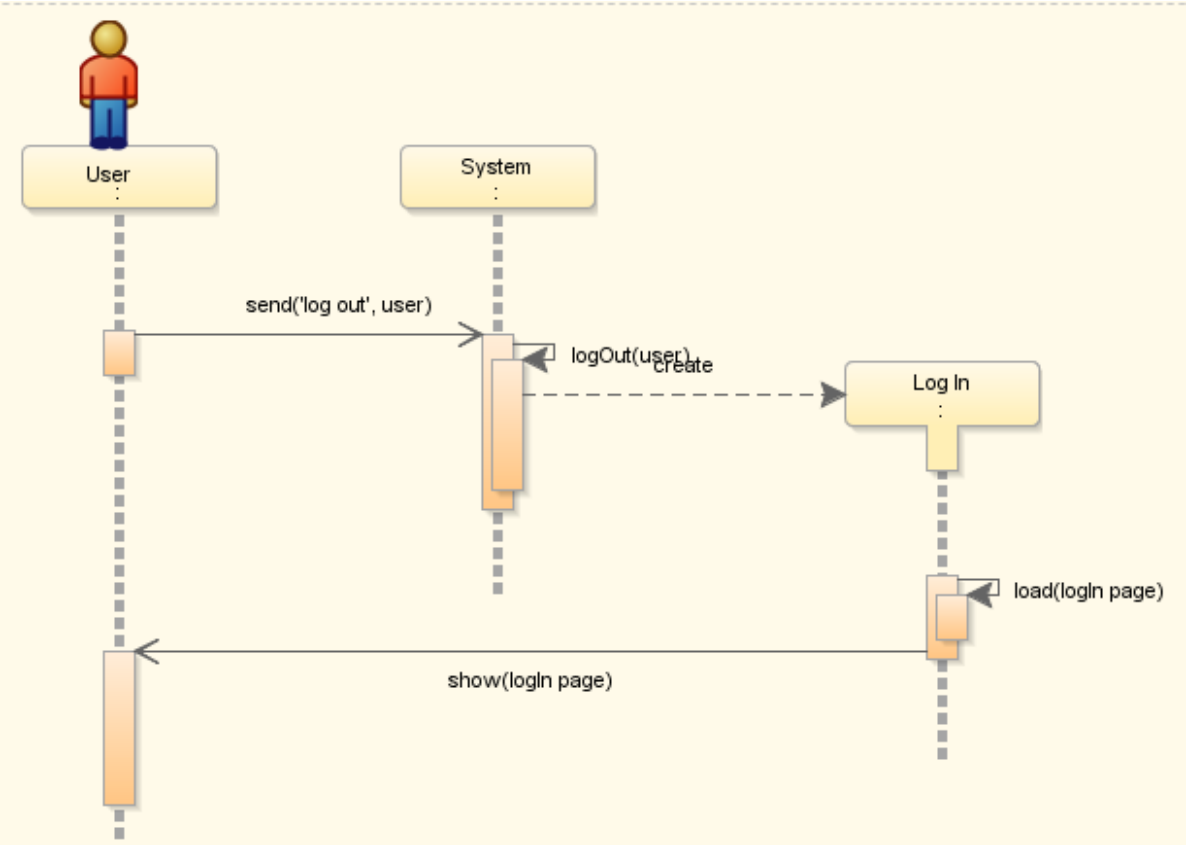
3.2.4.2. Log In-Error diagram



Sequence diagram 2: Log in sequence diagram in the case that the log in operation hasn't been successful

Code	• SD-002
Name	• Log In-Error diagram
Use case diagram reference	• USC-001(Log In)
Simplification reference	• No simplifications

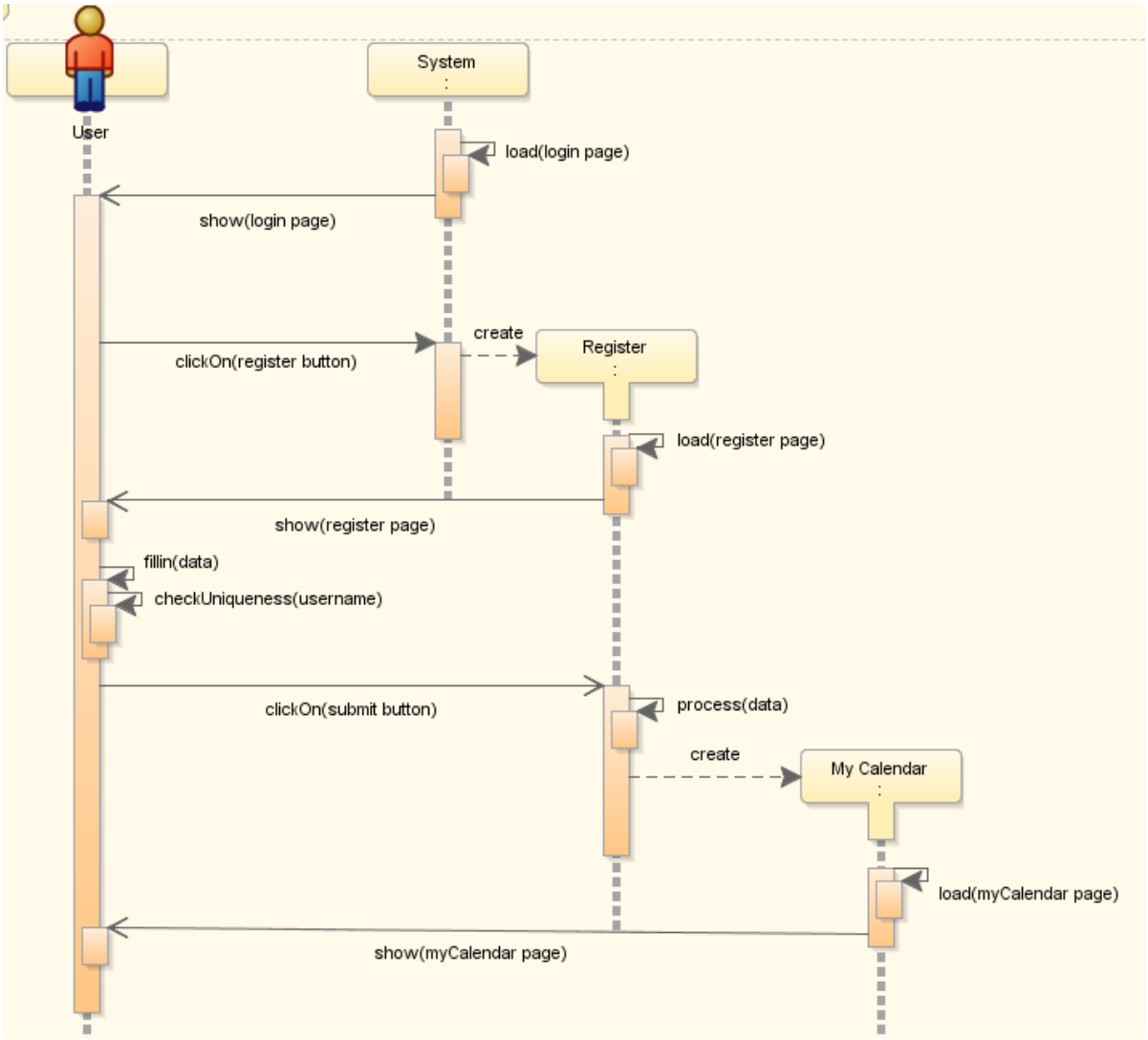
3.2.4.3. *Log Out diagram*



Sequence diagram 3: Log out sequence diagram that depicts log out operation performed by the user

Code	• SD-003
Name	• Log Out diagram
Use case diagram reference	• USC-002(Log Out)
Simplification reference	• No simplifications

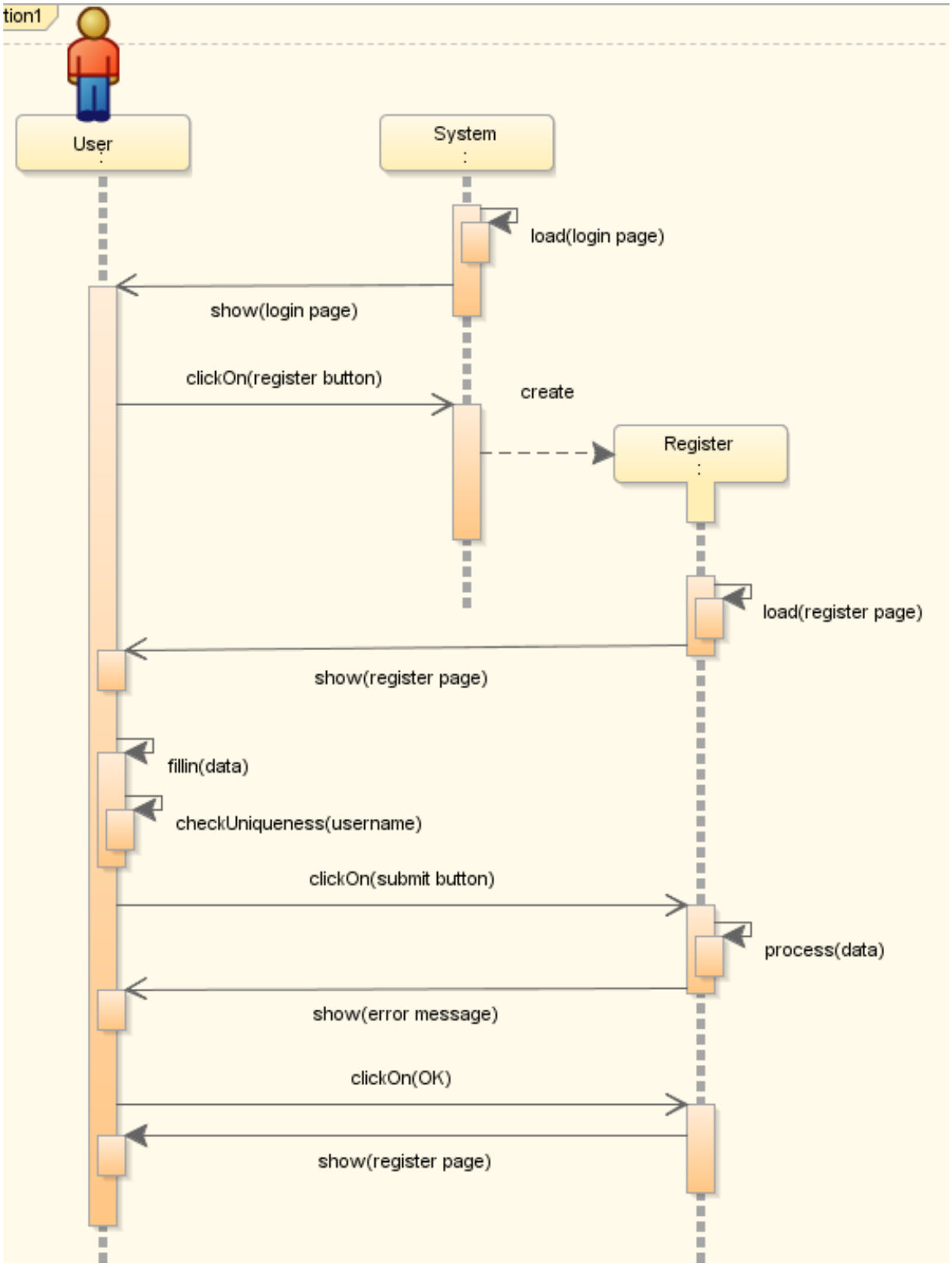
3.2.4.4. Registration-Success diagram



Sequence diagram 4: Registration diagram in the case when registration operation has been successful

Code	• SD-004
Name	• Registration-Success diagram
Use case diagram reference	• USC-003(Register) • USC-004(Check Username)
Simplification reference	• No simplifications

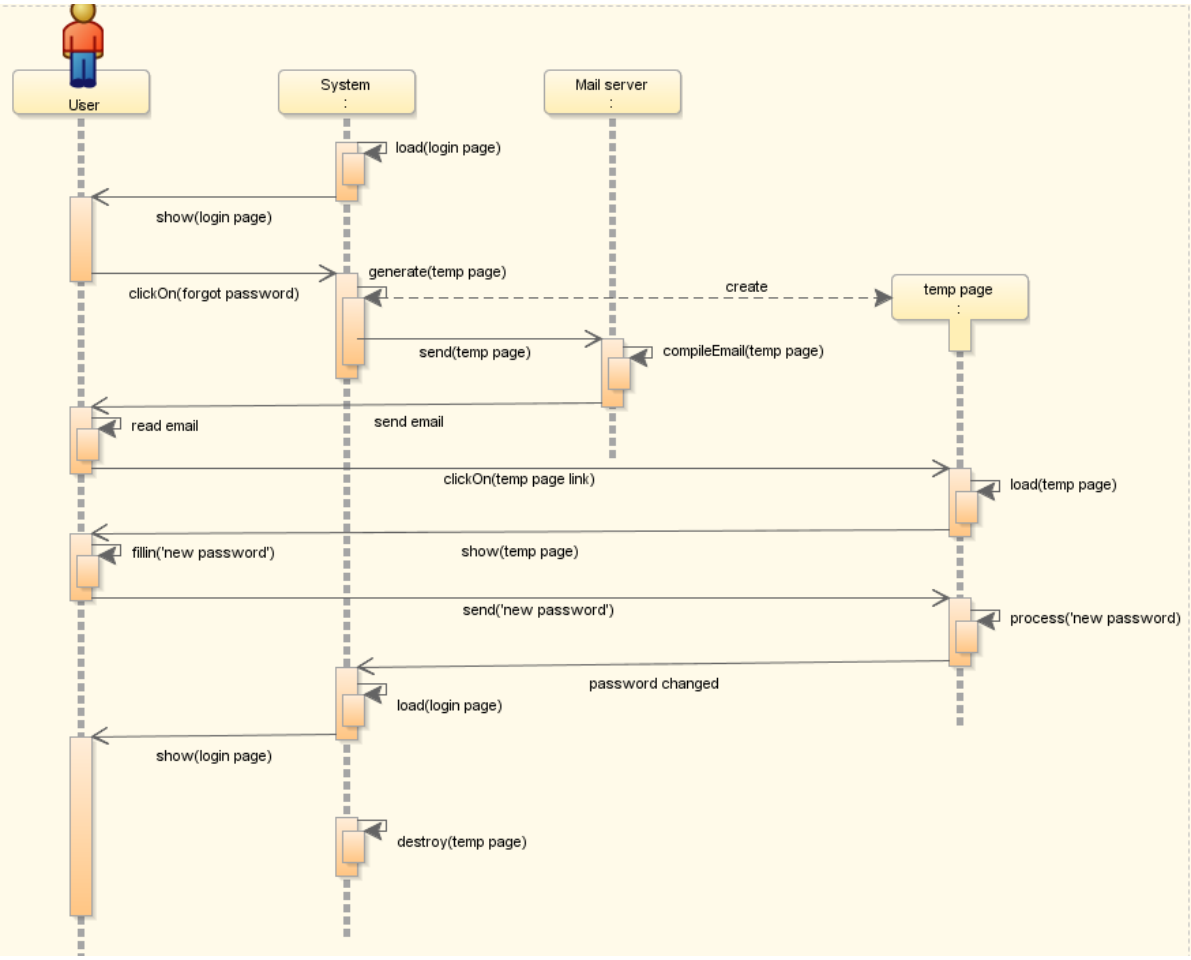
3.2.4.5. *Registration-Error diagram*



Sequence diagram 5: Registration sequence diagram in the case when registration hasn't been successful

Code	• SD-005
Name	• Registration-Error diagram
Use case diagram reference	• USC-003(Register) • USC-004(Check Username)
Simplification reference	• No simplifications

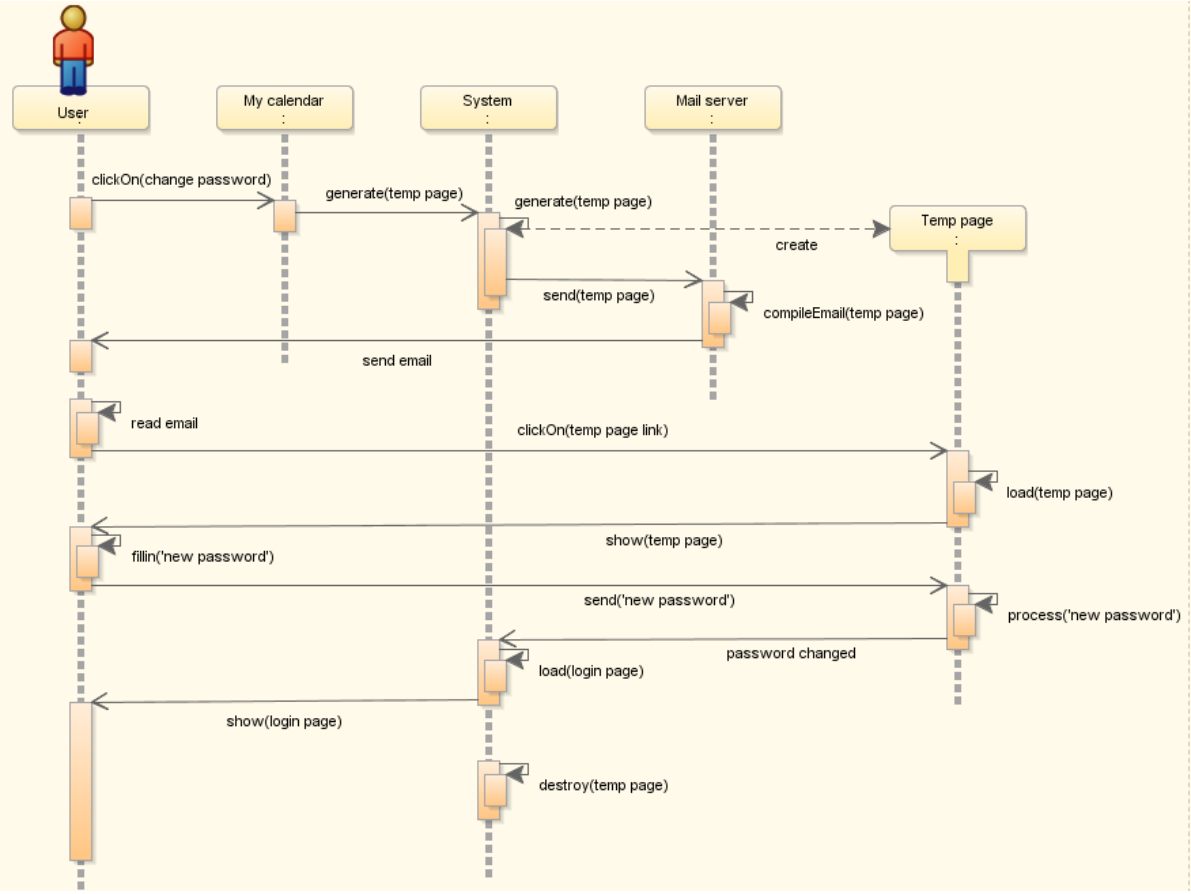
3.2.4.6. Reset password diagram



Sequence diagram 6: Reset password sequence diagram explains the flow of function calls in the system while 'Reset password' use case has been performed, also it references USC-006 due to 'include' relation between USC-005 and USC-006

Code	• SD-006
Name	• Reset Password diagram
Use case diagram reference	• USC-005(Reset Password) • USC-006(Change Password)
Simplification reference	• No simplifications

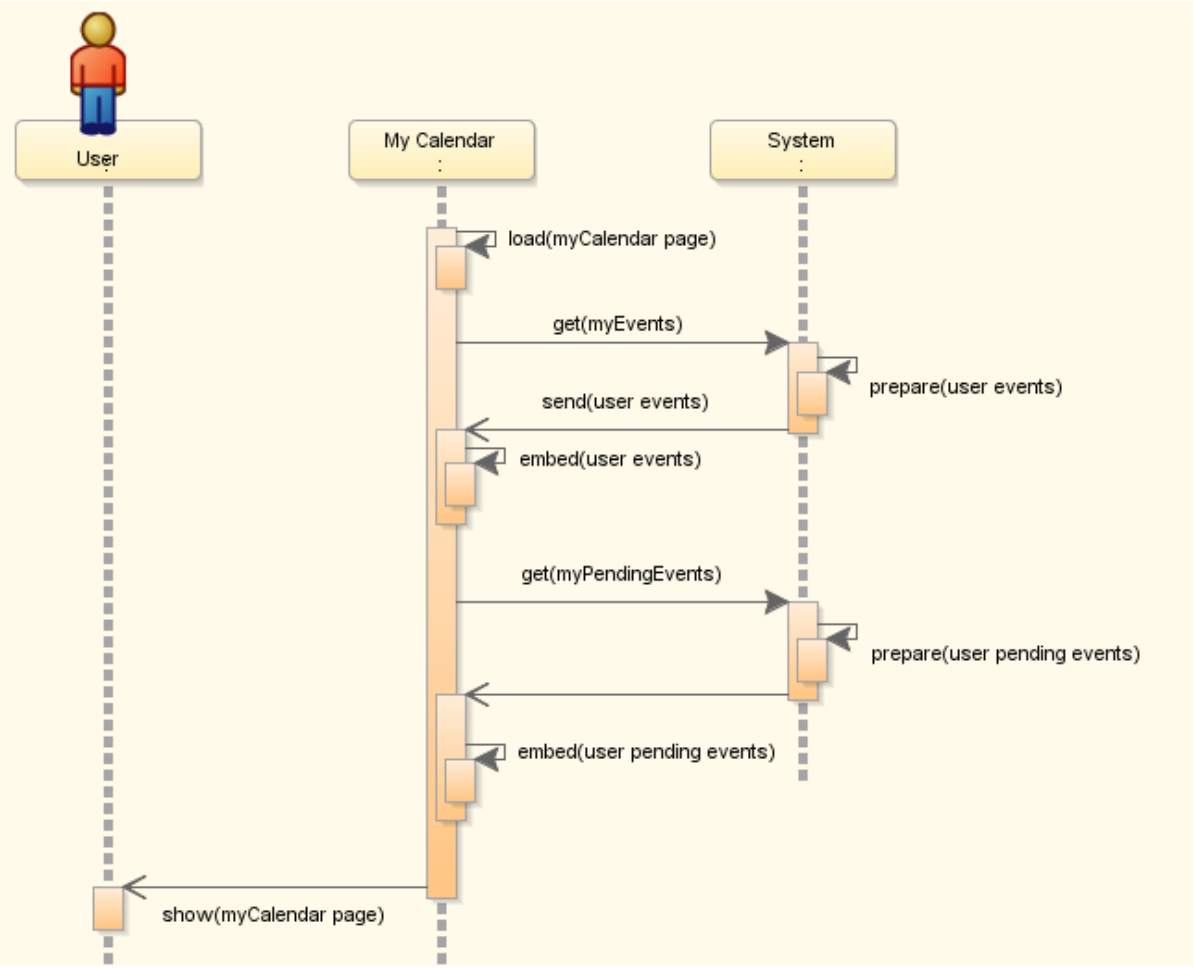
3.2.4.7. Change password diagram



Sequence diagram 7: Change password diagram explains function calls sequence when already logged in user desires to change his password

Code	• SD-007
Name	• Change Password diagram
Use case diagram reference	• USC-006(Change Password)
Simplification reference	• No simplifications

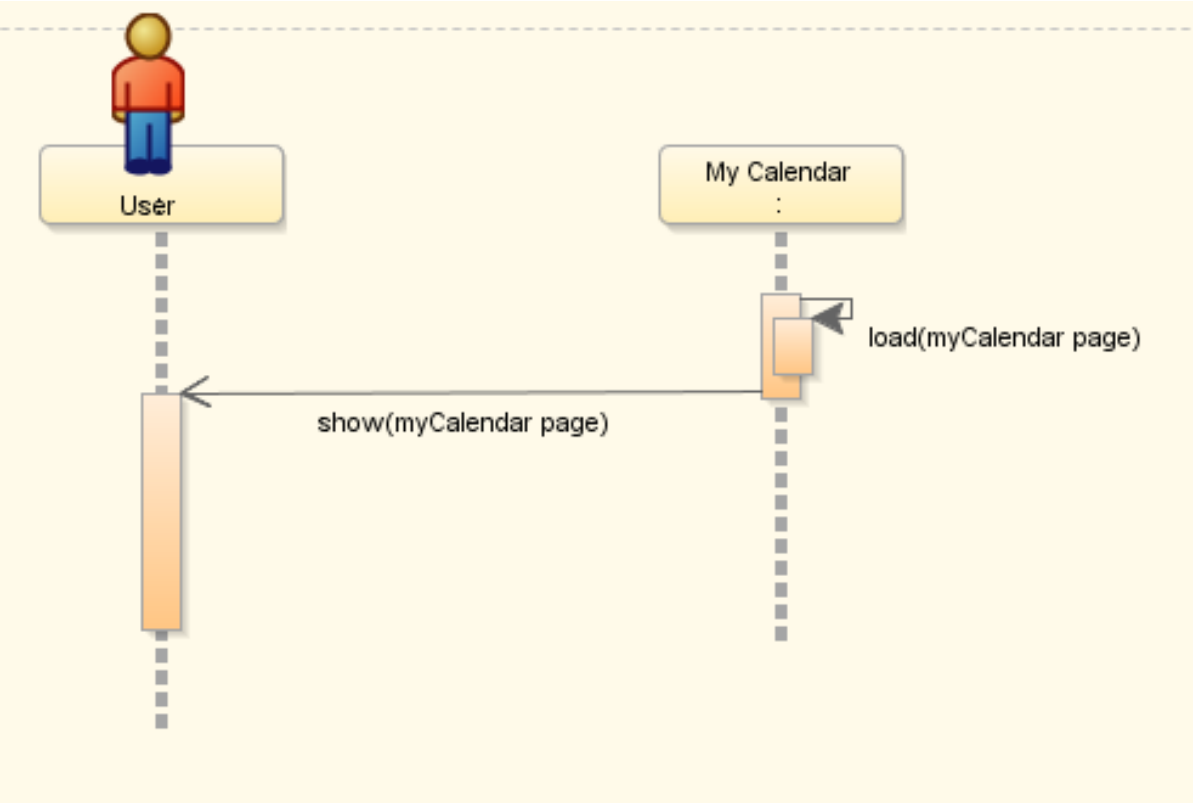
3.2.4.8. Generate My Calendar Page diagram



Sequence diagram 8: Generate My Calendar Page sequence diagram represent detailed description of creation of my calendars page in the system

Code	• SD-008
Name	• Generate My Calendar Page diagram
Use case diagram reference	• USC-008(View my calendar(Log in)) • USC-009(View my calendar(Public calendars))
Simplification reference	• No simplifications

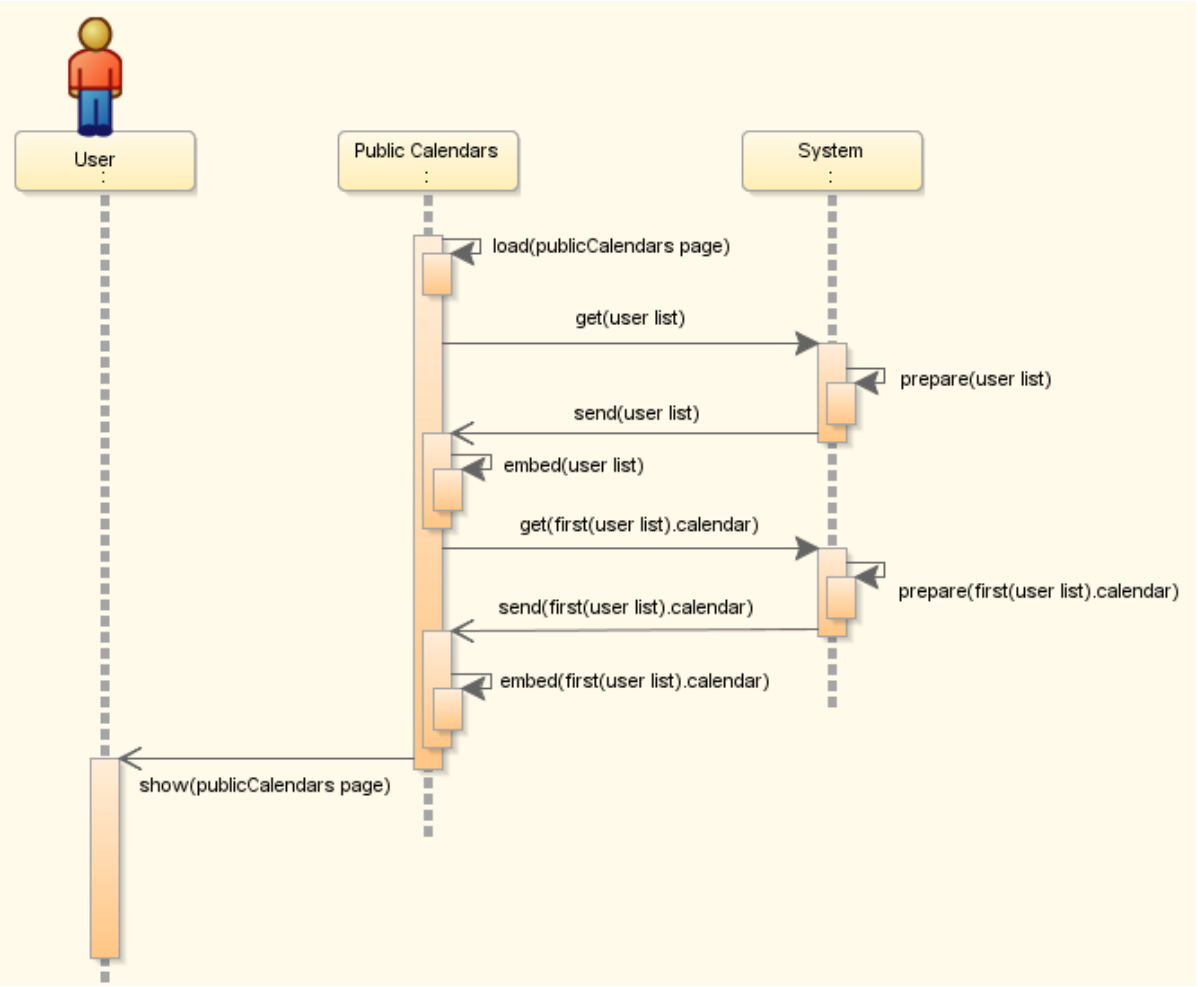
3.2.4.9. *Generate My Calendar Page-Simplified diagram*



Sequence diagram 9: Generate My Calendar Page-Simplified diagram, substitution diagram for the full representation of Generate My Calendar Page diagram, which will be used as a component in other sequence diagrams for more concise representation

Code	• SDS-001
Name	• Generate My Calendar Page-Simplified diagram
Sequence diagram reference	• SD-008

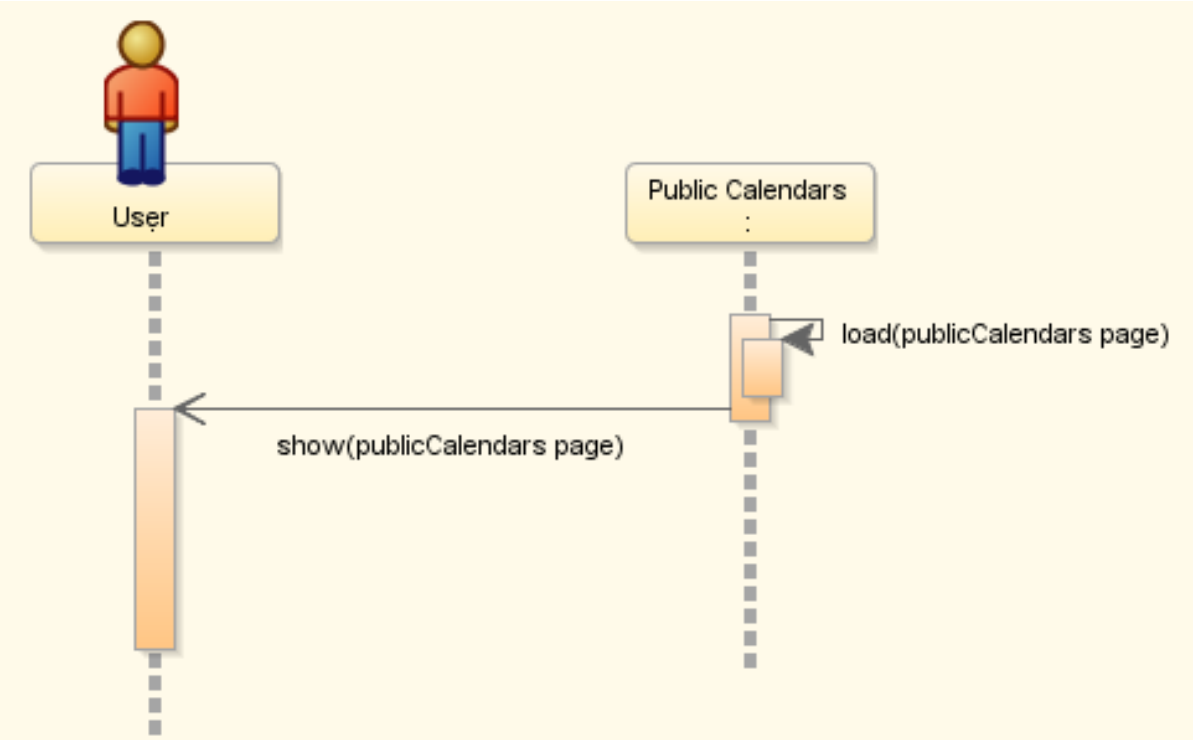
3.2.4.10. Generate Public Calendars Page diagram



Sequence diagram 10: Generate Public Calendars Page diagram explains detailed function call sequence for the creation of public calendars page

Code	• SD-009
Name	• Generate Public Calendars Page diagram
Use case diagram reference	• USC-010(View Public Calendars)
Simplification reference	• No simplifications

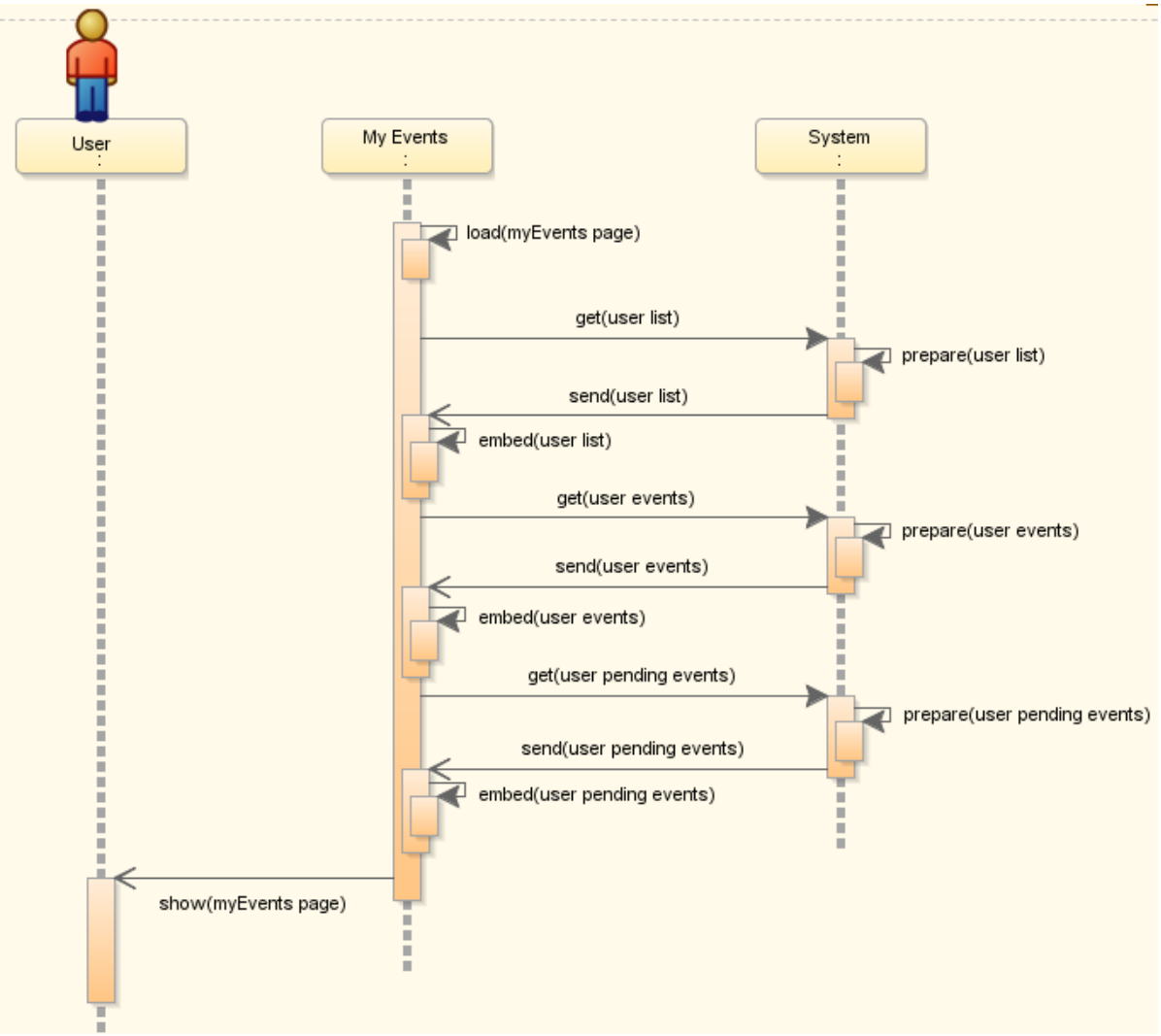
3.2.4.11. Generate Public Calendars Page-Simplified diagram



Sequence diagram 11: Generate Public Calendars Page-Simplified diagram, substitution diagram for the full representation of Generate Public Calendars Page diagram, which will be used as a component in other sequence diagrams for more concise representation

Code	• SDS-002
Name	• Generate Public Calendars Page-Simplified diagram
Sequence diagram reference	• SD-009

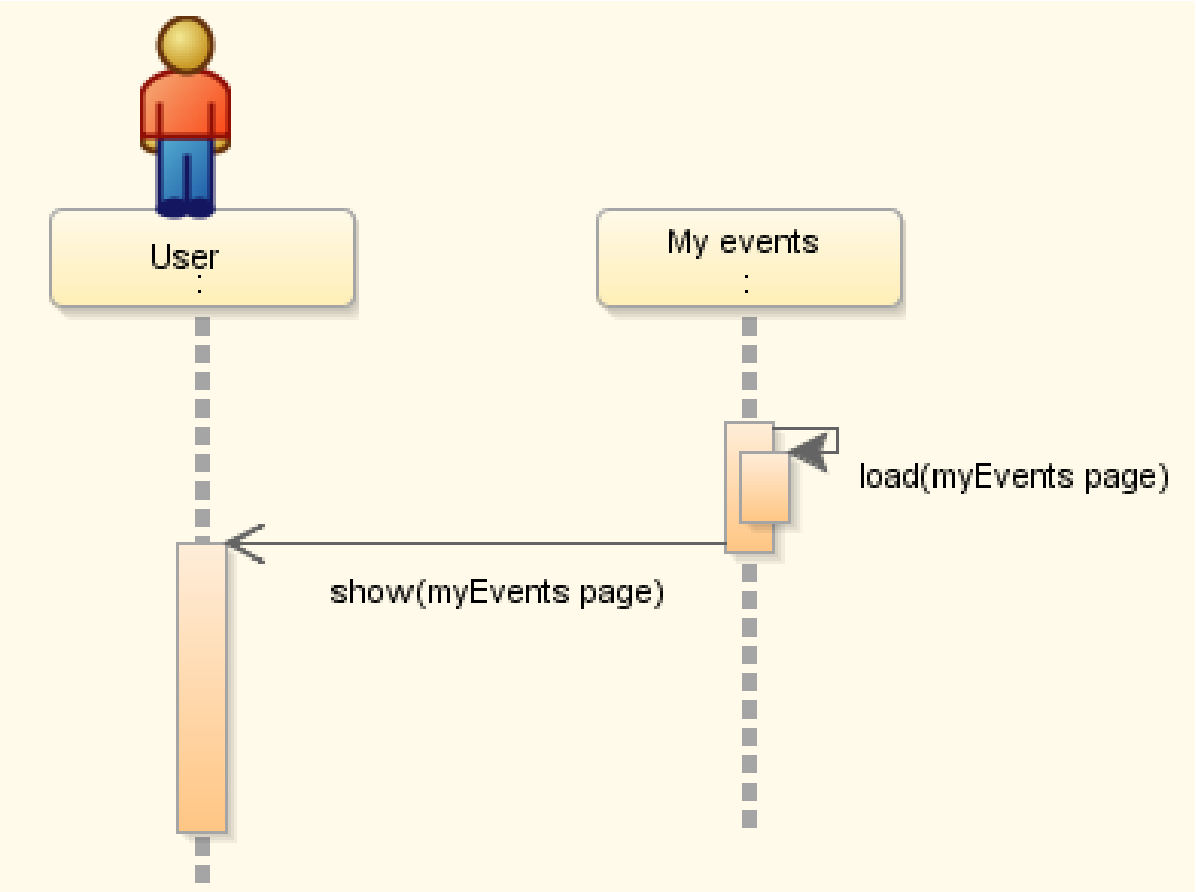
3.2.4.12. Generate My Events Page diagram



Sequence diagram 12: Generate My Events Page diagram explains detailed function call sequence for the creation of my events page

Code	• SD-010
Name	• Generate My Events Page diagram
Use case diagram reference	• No use case
Simplification reference	• No simplifications

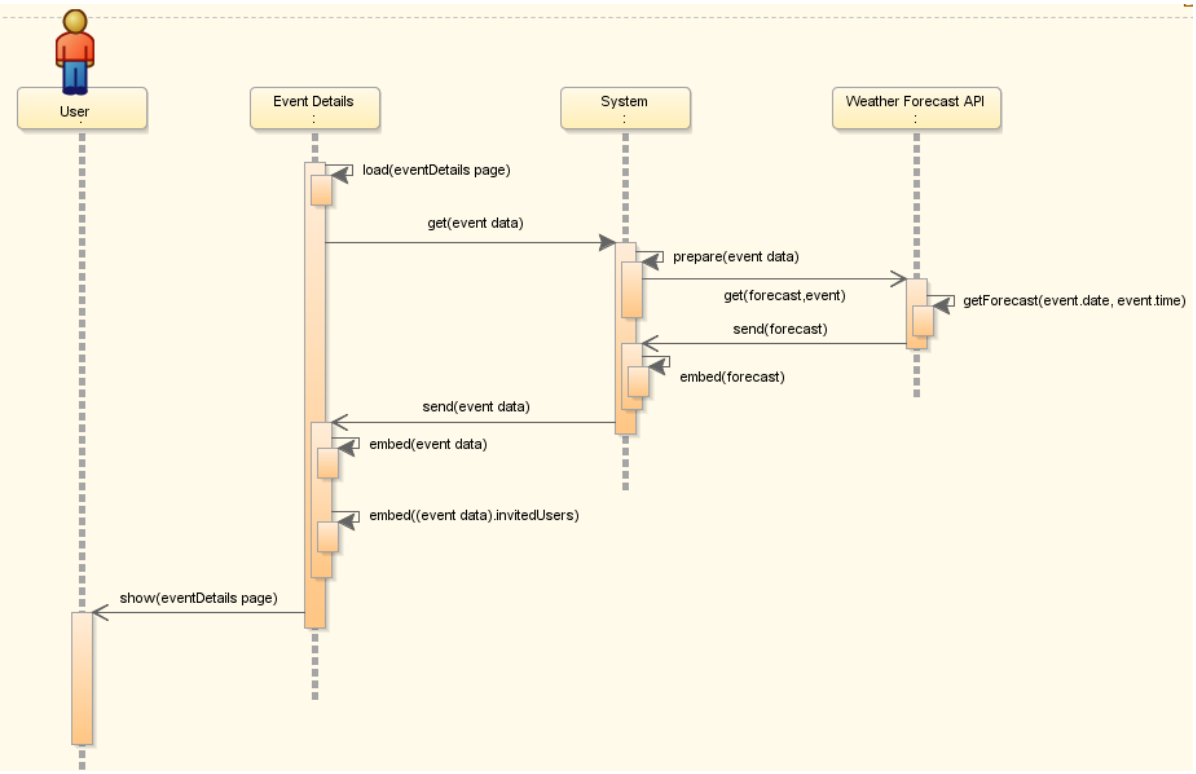
3.2.4.13. Generate Public Calendars Page-Simplified diagram



Sequence diagram 13: Generate My Events Page-Simplified diagram, substitution diagram for the full representation of Generate My Events Page diagram, which will be used as a component in other sequence diagrams for more concise representation

Code	• SDS-003
Name	• Generate My Events Page-Simplified diagram
Sequence diagram reference	• SD-010

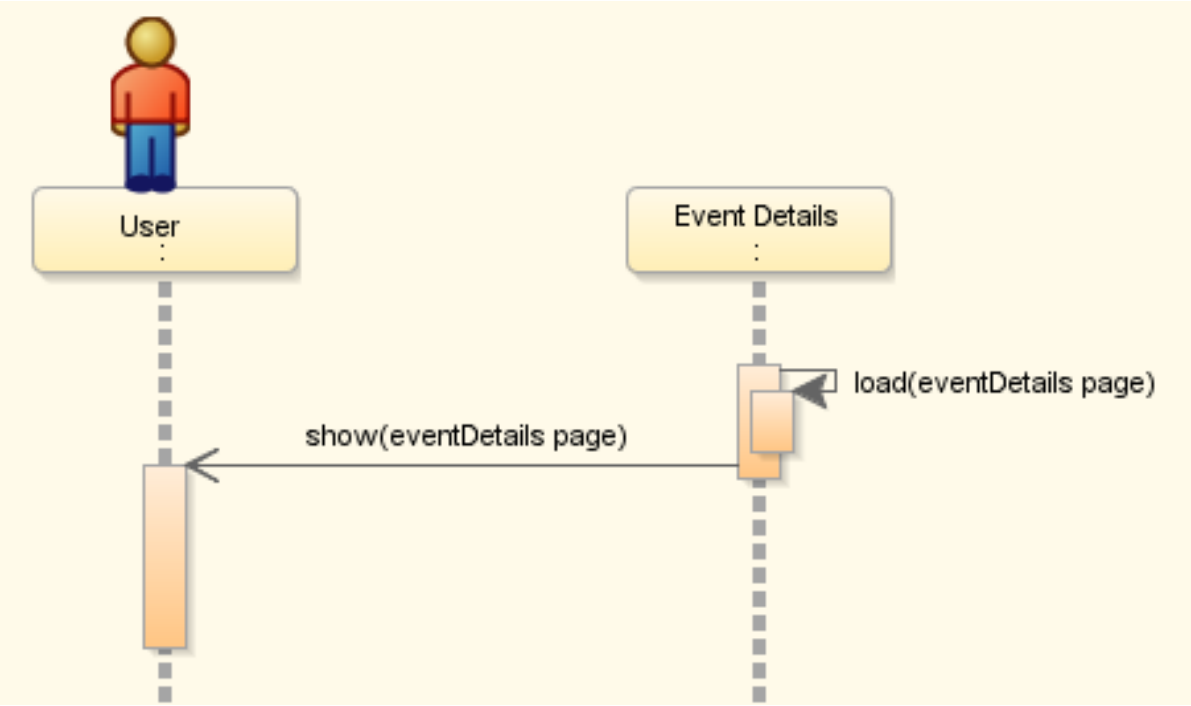
3.2.4.14. *Generate My Events Page diagram*



Sequence diagram 14: Generate Event Details Page diagram explains detailed function call sequence for the creation of event details page

Code	• SD-011
Name	• Generate Event Details Page diagram
Use case diagram reference	• No use case
Simplification reference	• No simplifications

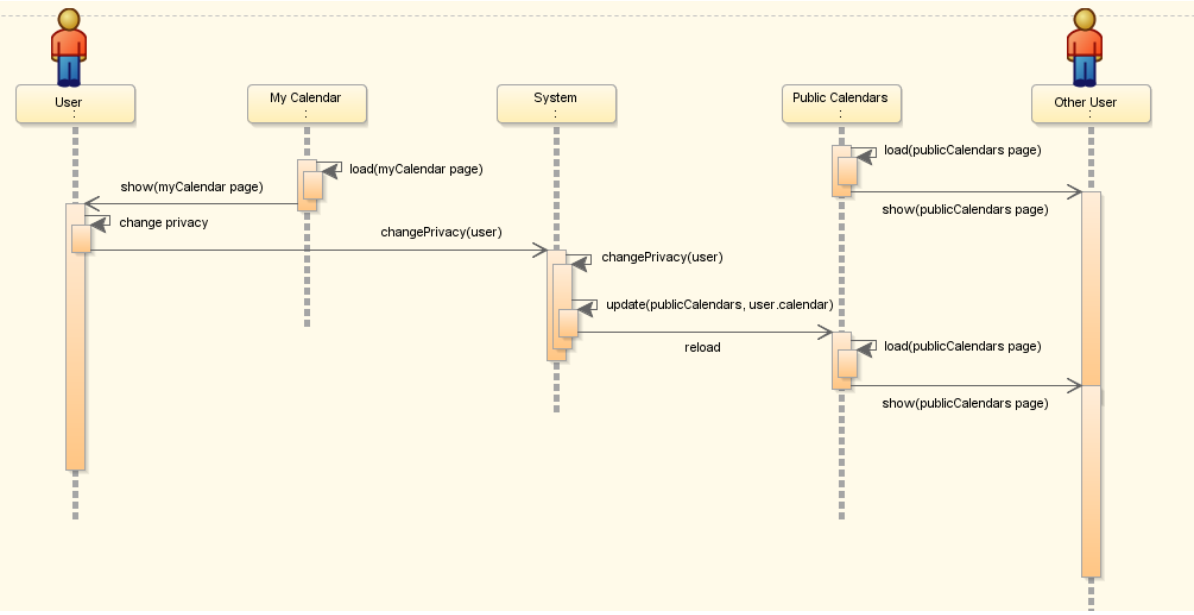
3.2.4.15. Generate Event Details Page-Simplified diagram



Sequence diagram 15: Generate Event Details Page-Simplified diagram, substitution diagram for the full representation of Generate Event Details Page diagram, which will be used as a component in other sequence diagrams for more concise representation

Code	• SDS-004
Name	• Generate Event Details Page-Simplified diagram
Sequence diagram reference	• SD-011

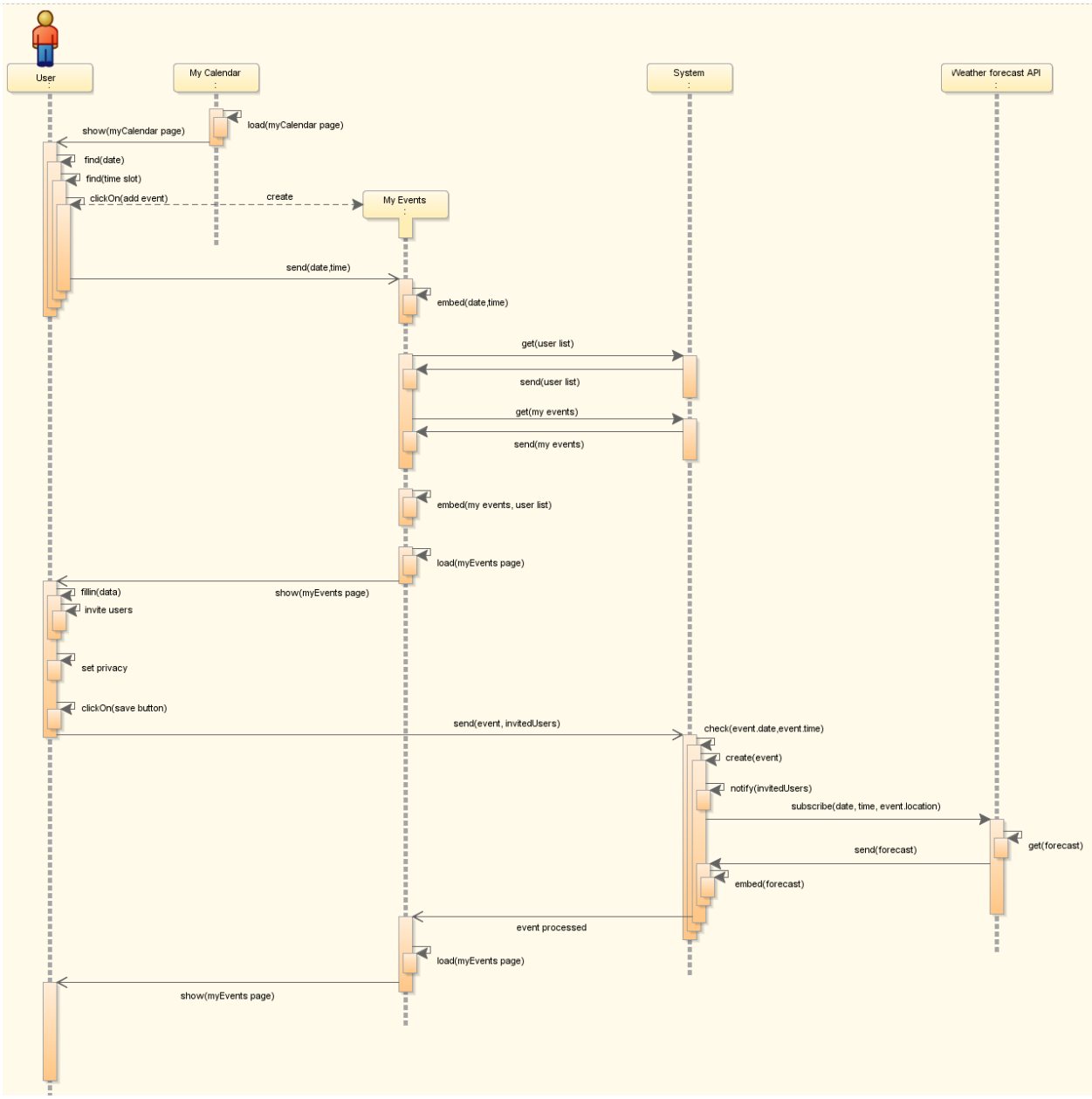
3.2.4.16. *Change Calendar's Privacy diagram*



Sequence diagram 16: Change Calendar's Privacy sequence diagram explains the interaction between system components when calendar's privacy has been changed

Code	• SD-012
Name	• Change Calendar's Privacy diagram
Use case diagram reference	• USC-021(Change Calendar Privacy)
Simplification reference	• SDS-001(Generate My Calendar Page-Simplified) • SDS-002(Generate Public Calendars Page-Simplified)

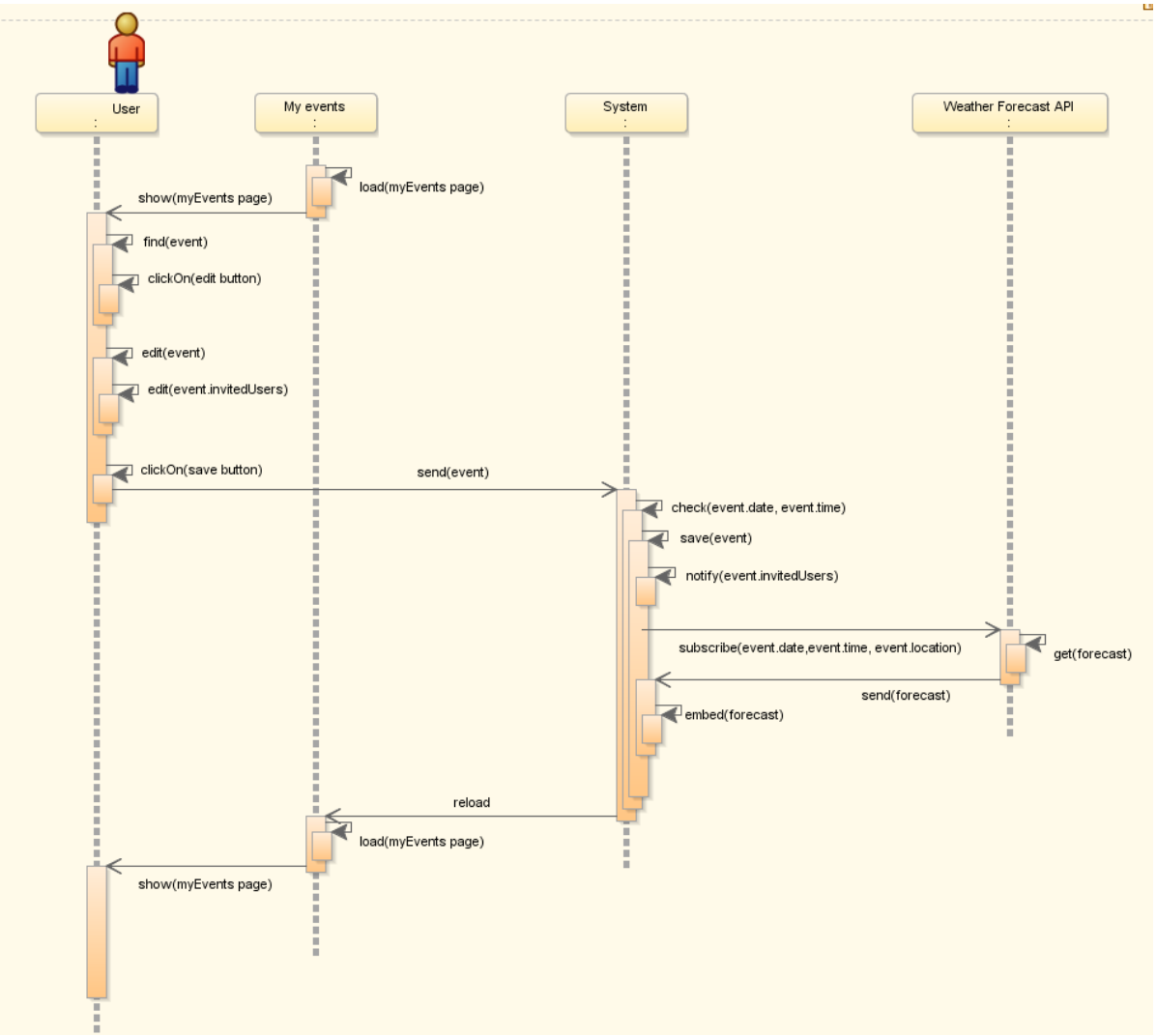
3.2.4.17. *Create Event diagram*



Sequence diagram 17: Create Event sequence diagram depicts in details function call sequence performed in the system when 'create event' use case has been performed by the user

Code	• SD-013
Name	• Create Event diagram
Use case diagram reference	• USC-013(Search User(Event Details)) • USC-014(Invite Users) • USC-015(Save Event) • USC-016(Create Event) • USC-017(Edit Event) • USC-022(Edit Event Privacy) • USC-023(View My Events)
Simplification reference	• SDS-001(Generate My Calendar Page-Simplified) • SDS-003(Generate My Events Page-Simplified)

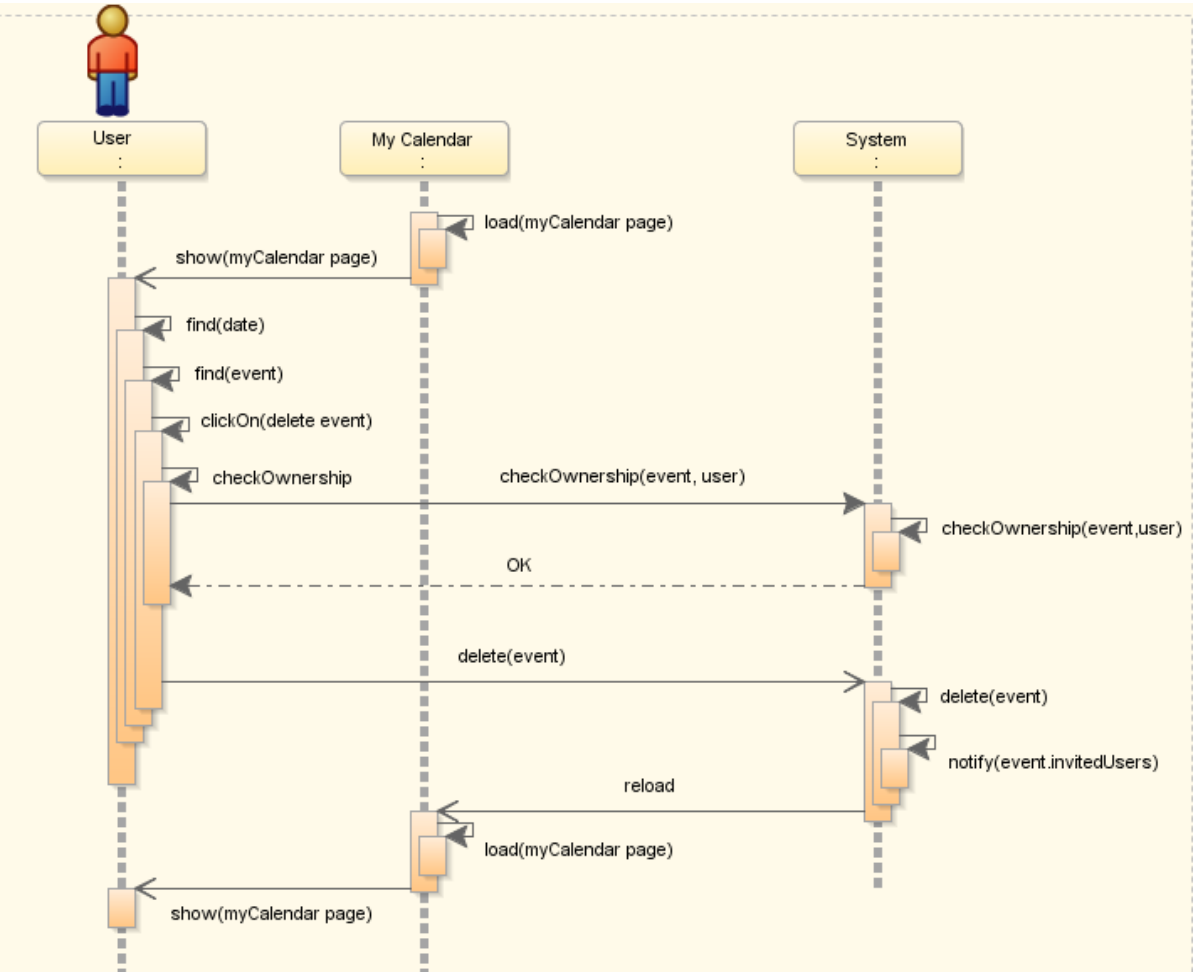
3.2.4.18. *Create Event diagram*



Sequence diagram 18: Edit Event sequence diagram depict detailed description of function call sequence performed by the system when 'Edit event' use case has been performed by user

Code	• SD-014
Name	• Edit Event diagram
Use case diagram reference	• USC-013(Search User(Event Details)) • USC-014(Invite Users) • USC-015(Save Event) • USC-017(Edit Event) • USC-022(Edit Event Privacy) • USC-023(View My Events)
Simplification reference	• SDS-003(Generate My Events Page-Simplified)

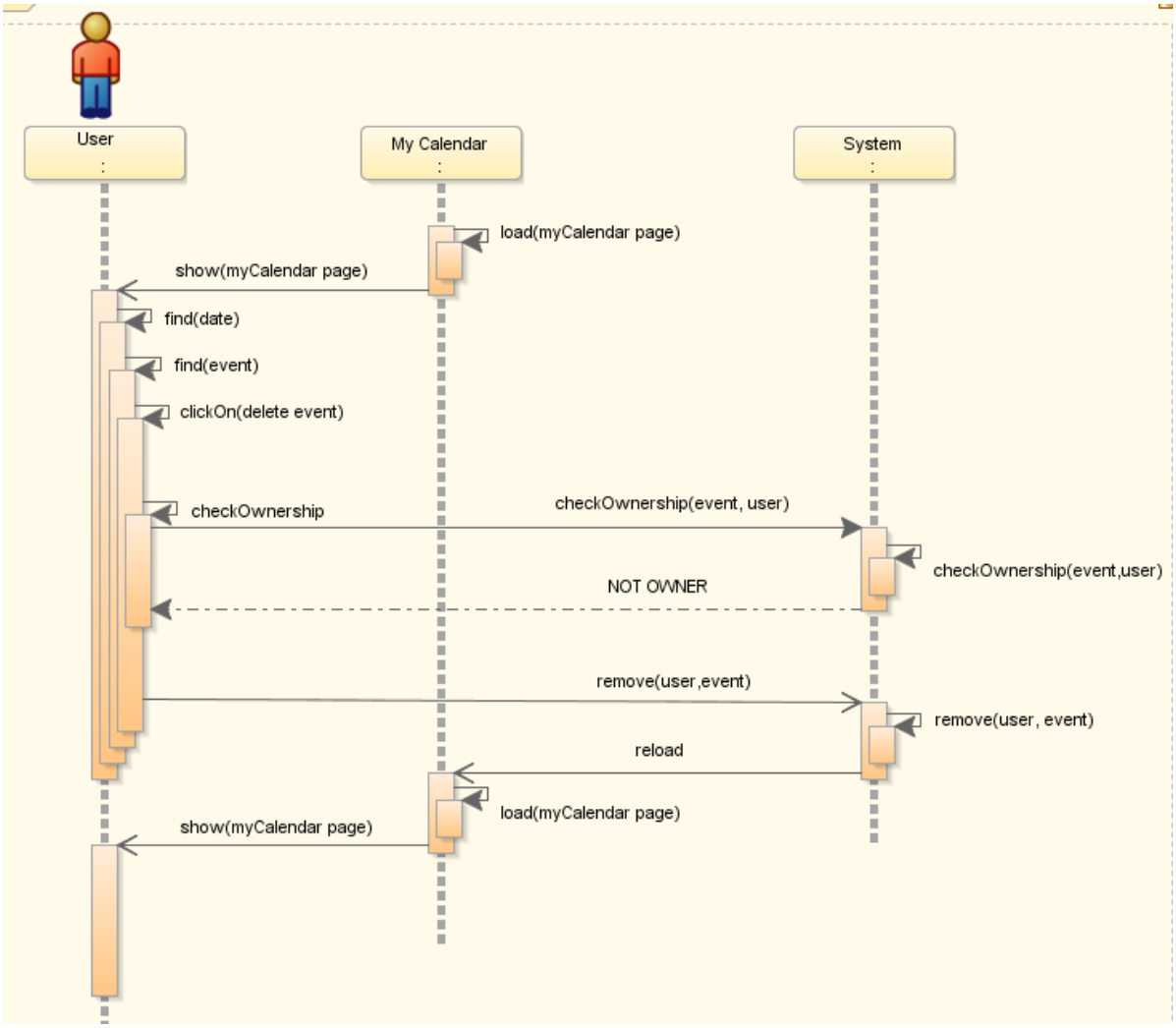
3.2.4.19. Delete Event-My Calendar-Owner diagram



Sequence diagram 19: Delete Event-My Calendar-Owner sequence diagram represent detailed description of inter-communication inside of the system when 'delete event' use case has been performed by the user from my calendars page and the user is the owner of the event

Code	• SD-015
Name	• Delete Event-My Calendar-Owner diagram
Use case diagram reference	• USC-012(Search time and date) • USC-018 Delete event(My Calendar)
Simplification reference	• SDS-001(Generate My Calendar Page-Simplified)

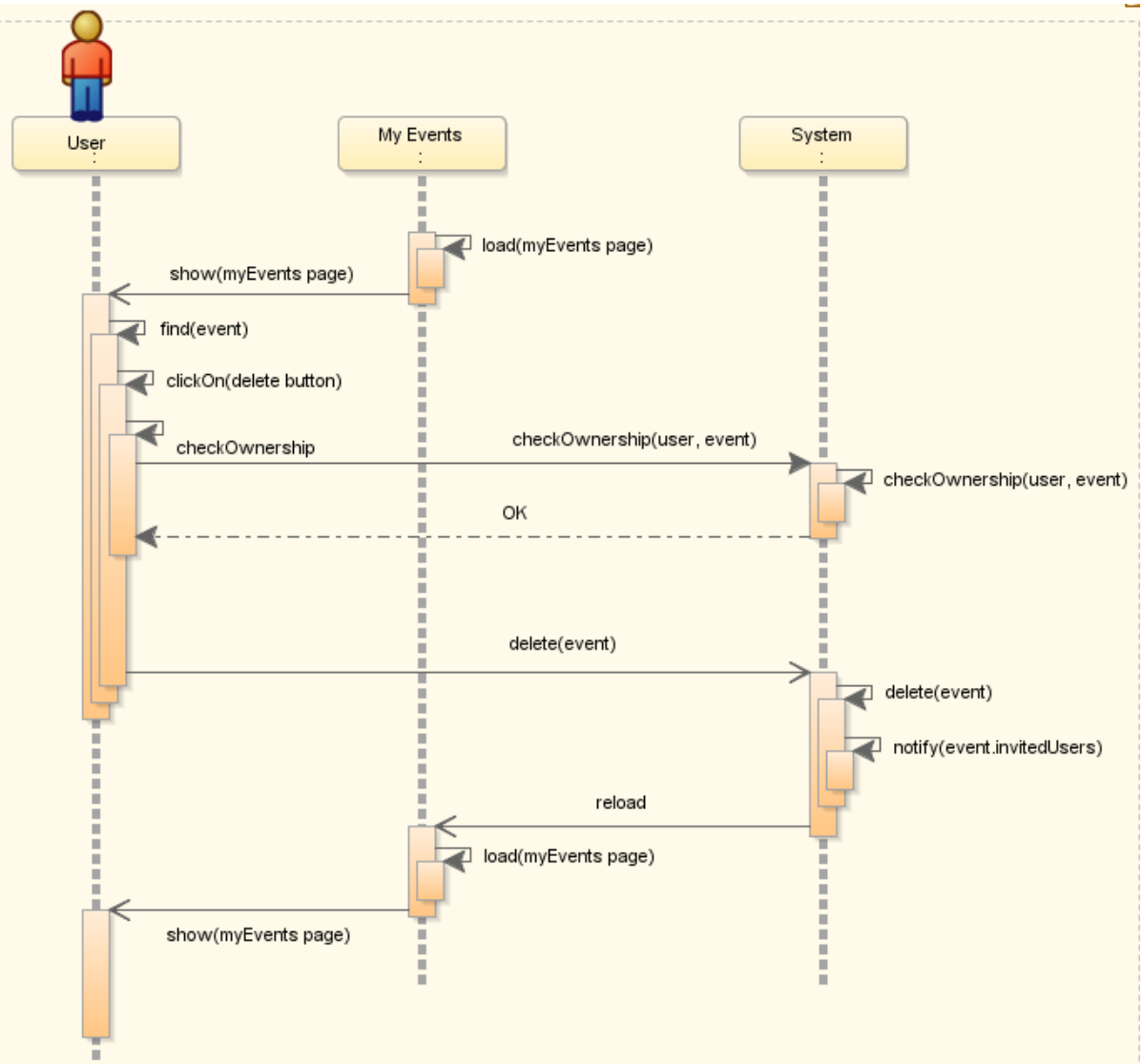
3.2.4.20. Delete Event-My Calendar-Owner diagram



Sequence diagram 20: Delete Event-My Calendar-Not Owner sequence diagram represent detailed description of inter-communication inside of the system when 'delete event' use case has been performed by the user from my calendars page and the user is not the owner of the event

Code	• SD-016
Name	• Delete Event-My Calendar-Not Owner diagram
Use case diagram reference	• USC-012(Search time and date) • USC-018 Delete event(My Calendar)
Simplification reference	• SDS-001(Generate My Calendar Page-Simplified)

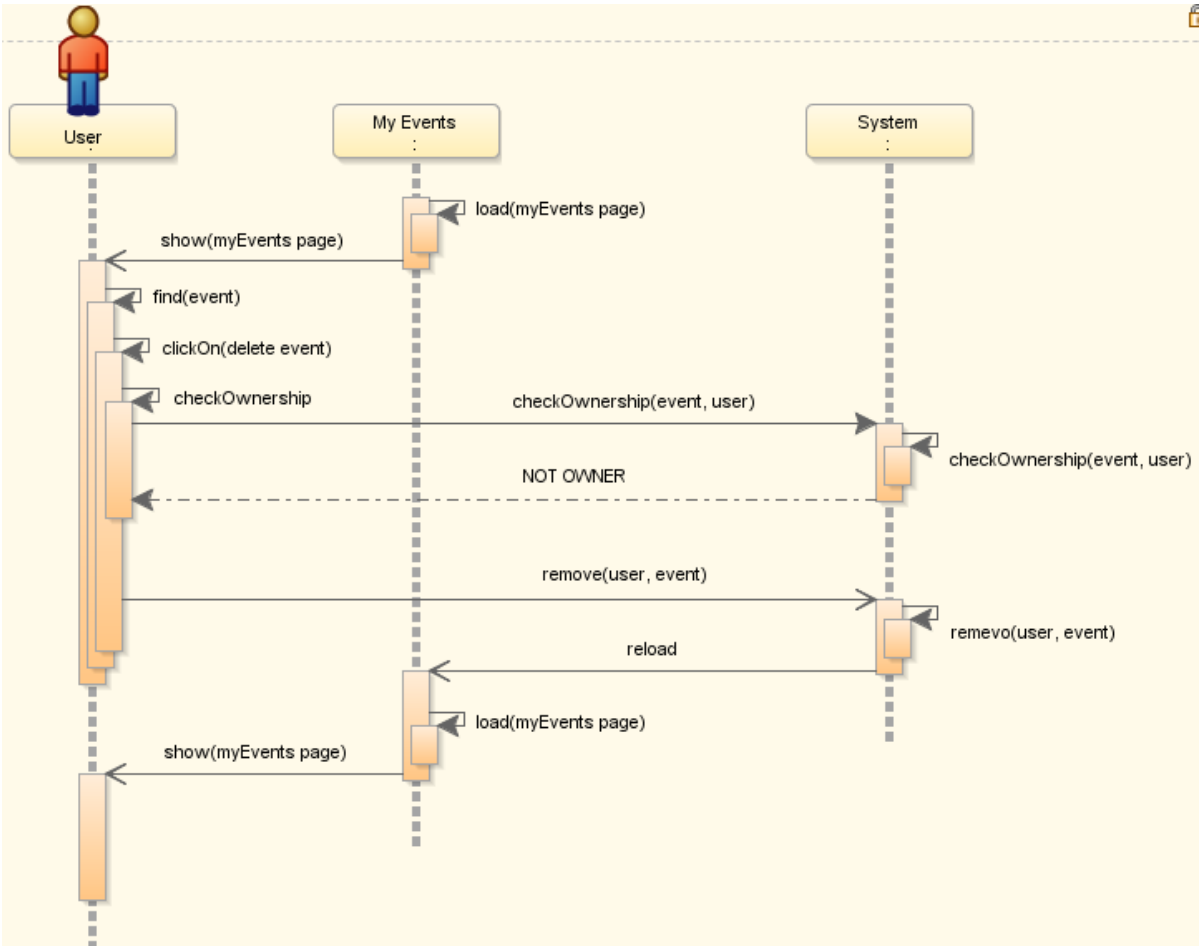
3.2.4.21. *Delete Event-My Events-Owner diagram*



Sequence diagram 21: Delete Event-My Events-Owner sequence diagram represent detailed description of inter-communication inside of the system when 'delete event' use case has been performed by the user from my events page and the user is the owner of the event

Code	• SD-017
Name	• Delete Event-My Events-Owner diagram
Use case diagram reference	• USC-019 Delete event(My Events)
Simplification reference	• SDS-003(Generate My Events Page-Simplified)

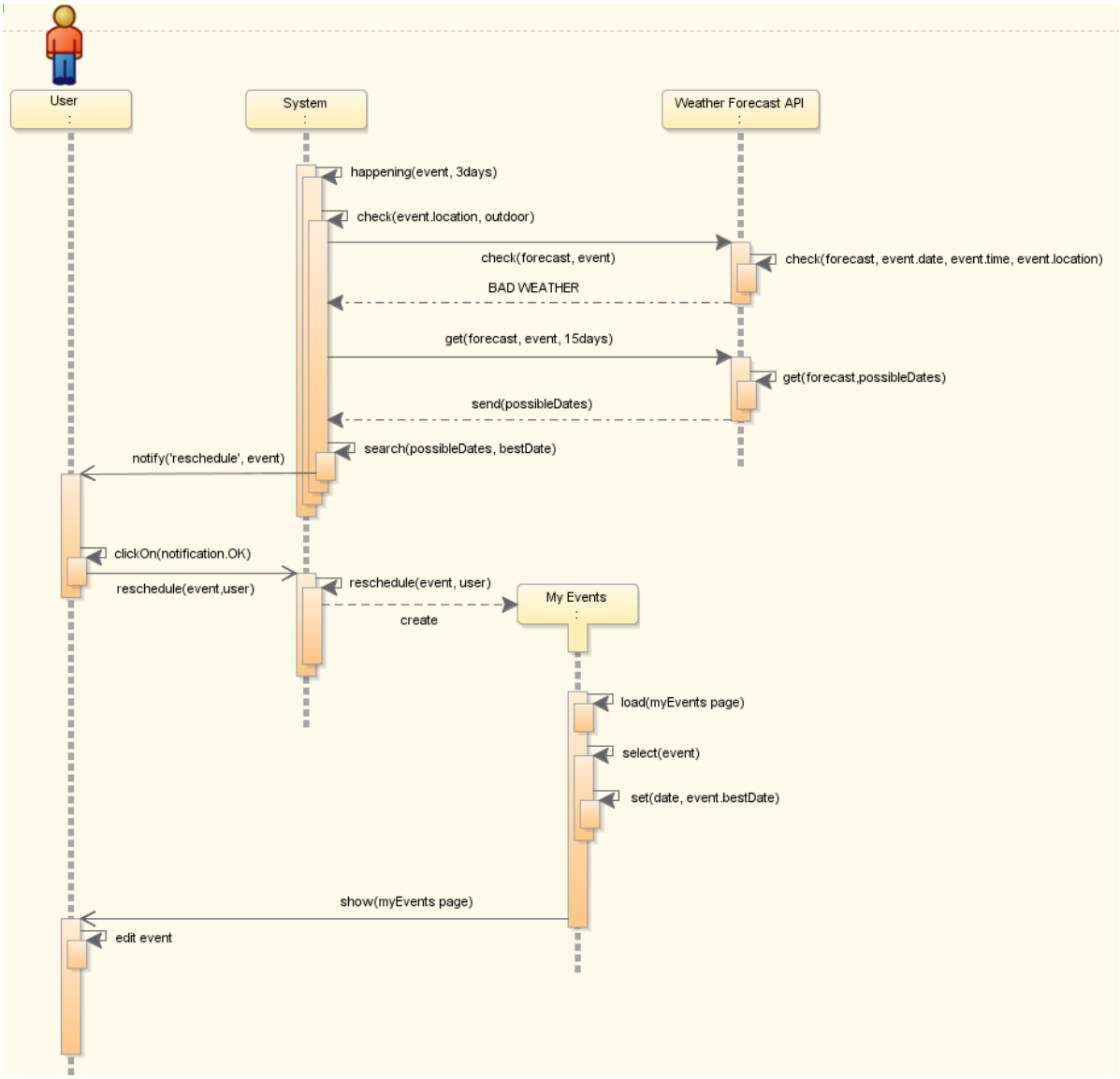
3.2.4.22. *Delete Event-My Events-Not Owner diagram*



Sequence diagram 22: Delete Event-My Events-Not Owner sequence diagram represent detailed description of inter-communication inside of the system when 'delete event' use case has been performed by the user from my events page and the user is not the owner of the event

Code	• SD-018
Name	• Delete Event-My Events-Not Owner diagram
Use case diagram reference	• USC-019 Delete event(My Events)
Simplification reference	• SDS-003(Generate My Events Page-Simplified)

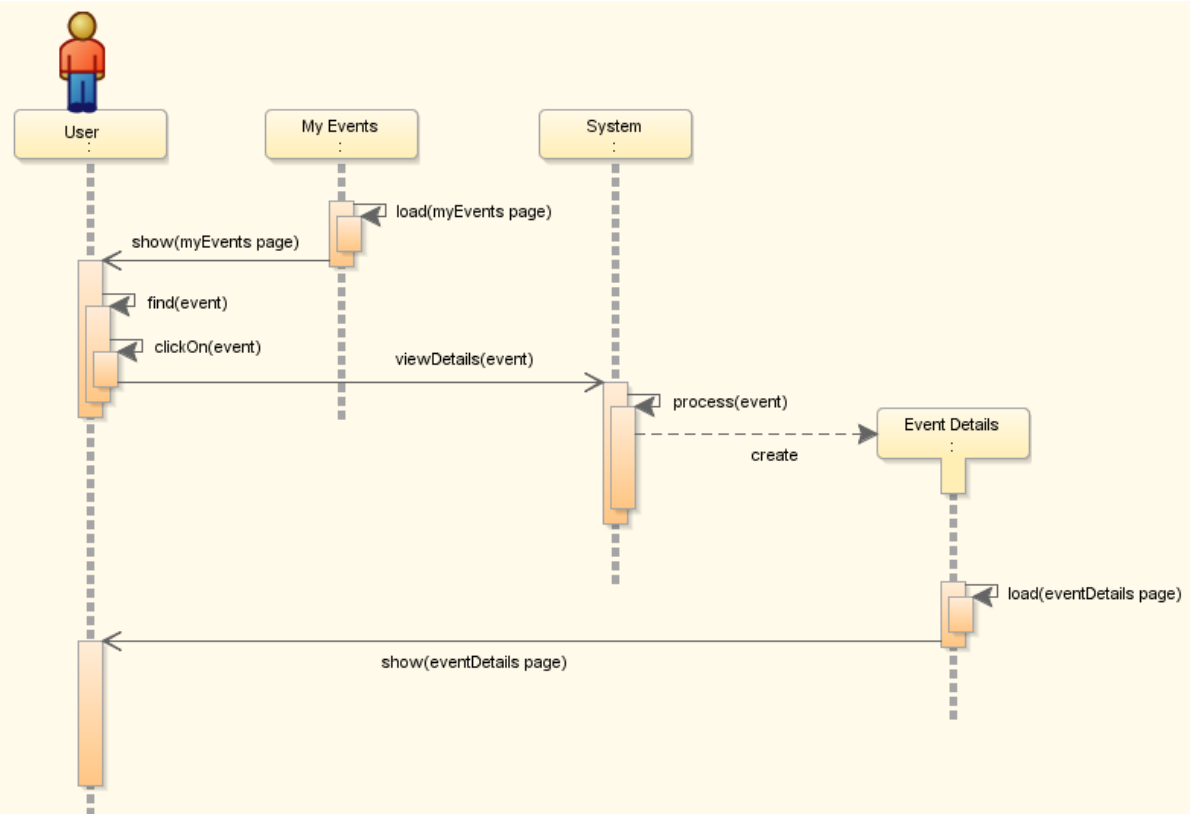
3.2.4.23. *Reschedule Event diagram*



Sequence diagram 23: Reschedule event sequence diagram explains into the detail the inter-communication of the system components when the event fulfils conditions to be rescheduled by the user

Code	• SD-019
Name	• Reschedule event diagram
Use case diagram reference	• USC-015(Save Event) • USC-017(Edit Event) • USC-025(Reschedule Event)
Simplification reference	• SDS-003(Generate My Events Page-Simplified)

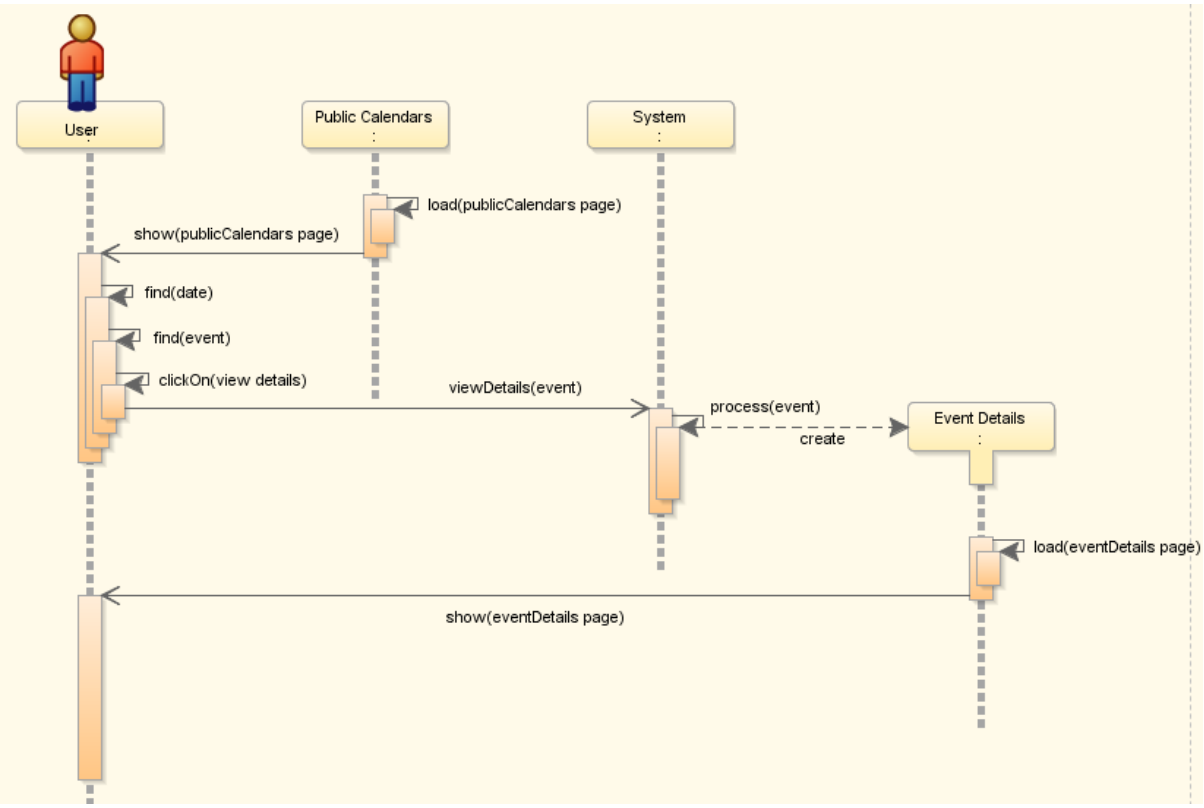
3.2.4.24. *View Event Details-My Events diagram*



Sequence diagram 24: View Event Details-My Events sequence diagram represents sequence of function calls circulating the system when 'View Event Details' use case has been performed by the user

Code	• SD-020
Name	• View Event Details-My Events diagram
Use case diagram reference	• USC-023(View My Events) • USC-026(View Event Details)
Simplification reference	• SDS-003(Generate My Events Page-Simplified) • SDS-004(Generate Event Details Page-Simplified)

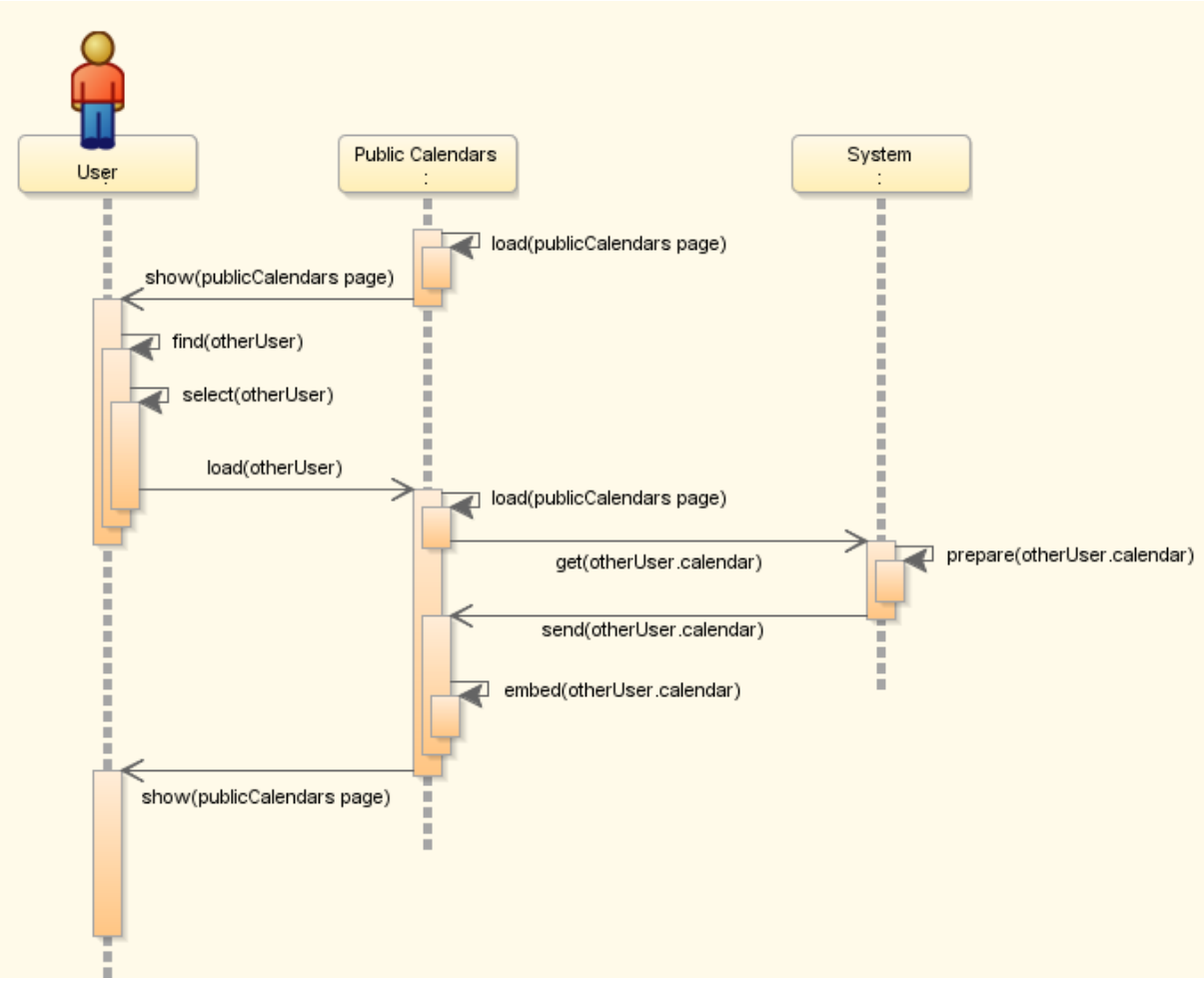
3.2.4.25. *View Event Details-Public Calendars diagram*



Sequence diagram 25: View Event Details-Public Calendars sequence diagram represents sequence of function calls circulating the system when 'View Event Details' use case has been performed by the user

Code	• SD-021
Name	• View Event Details-Public Calendars diagram
Use case diagram reference	• USC-010(View public calendar) • USC-026(View Event Details)
Simplification reference	• SDS-002(Generate Public Calendars Page-Simplified) • SDS-004(Generate Event Details Page-Simplified)

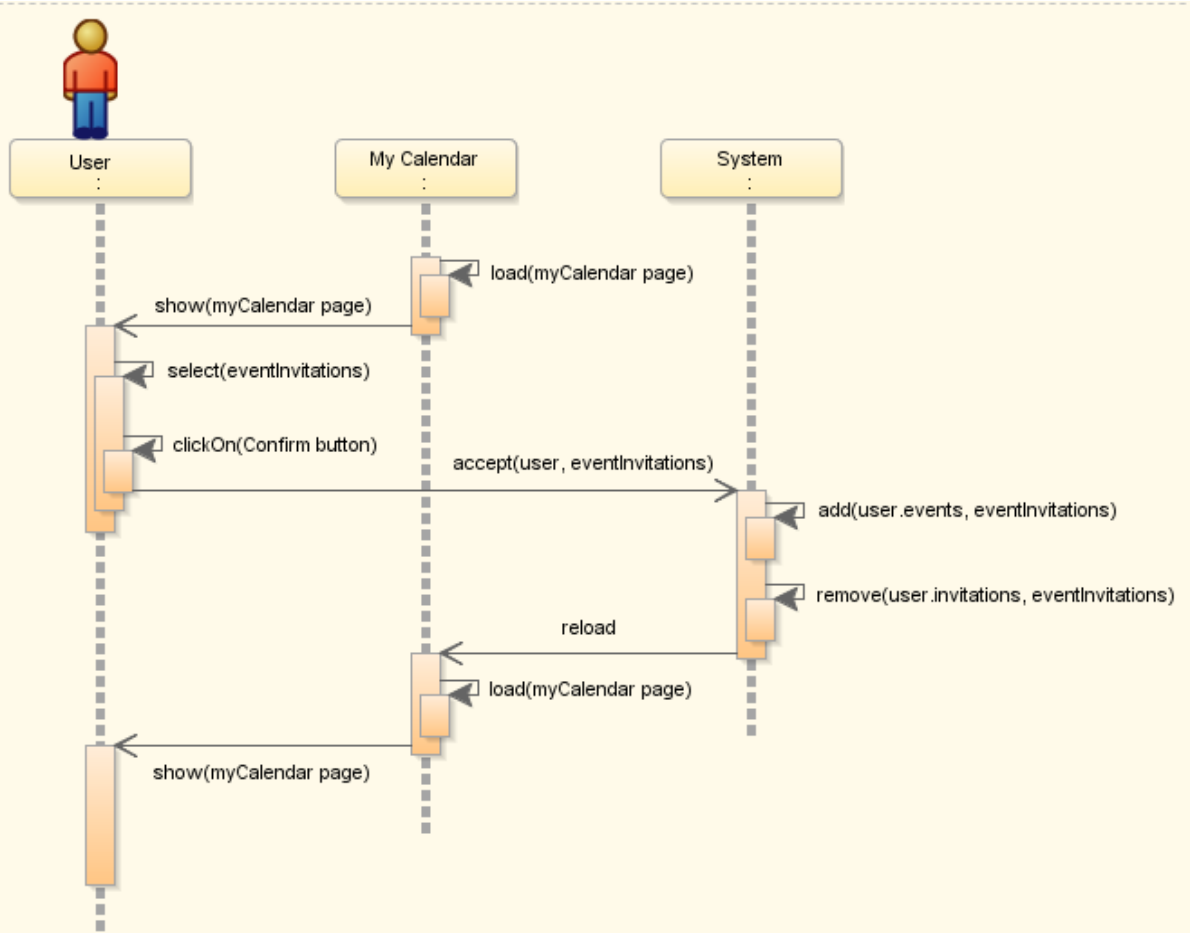
3.2.4.26. Search Public Calendars diagram



Sequence diagram 26: Search Public Calendars sequence diagram represents exchanged function calls when 'search public calendars' use case has been performed, and also describes how the data on the public calendars page is repopulated based on the user interaction

Code	• SD-021
Name	• Search Public Calendars diagram
Use case diagram reference	• USC-010(View public calendar) • USC-011(Search users(Public Calendars))
Simplification reference	• SDS-002(Generate Public Calendars Page-Simplified)

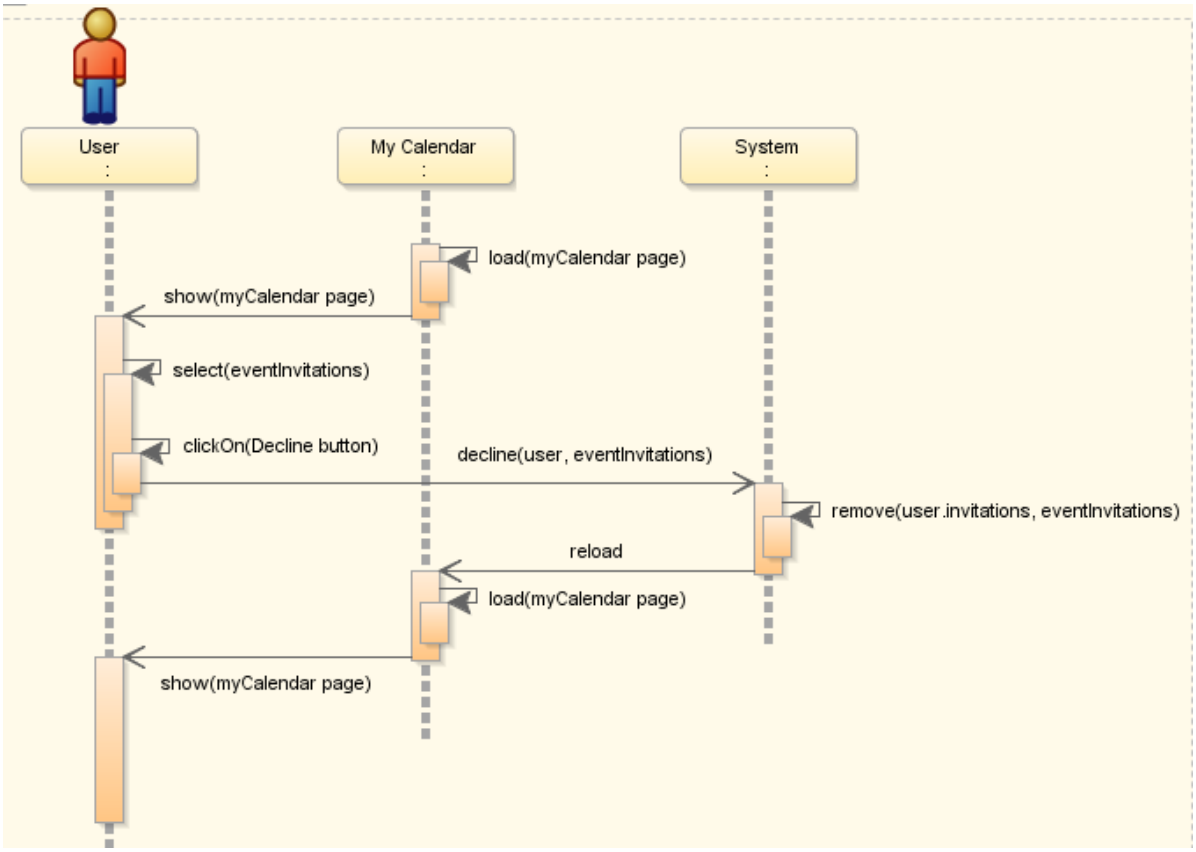
3.2.4.27. View Event Details-Public Calendars diagram



Sequence diagram 27: Accept Invitation sequence diagram depicts the detailed inter-communication between system components when 'accept invitation' use case is performed

Code	• SD-022
Name	• Accept Invitation diagram
Use case diagram reference	• USC-008(View my calendar(Log in)) • USC-020(Accept/Decline event invitation)
Simplification reference	• SDS-001(Generate My Calendars Page-Simplified)

3.2.4.28. View Event Details-Public Calendars diagram

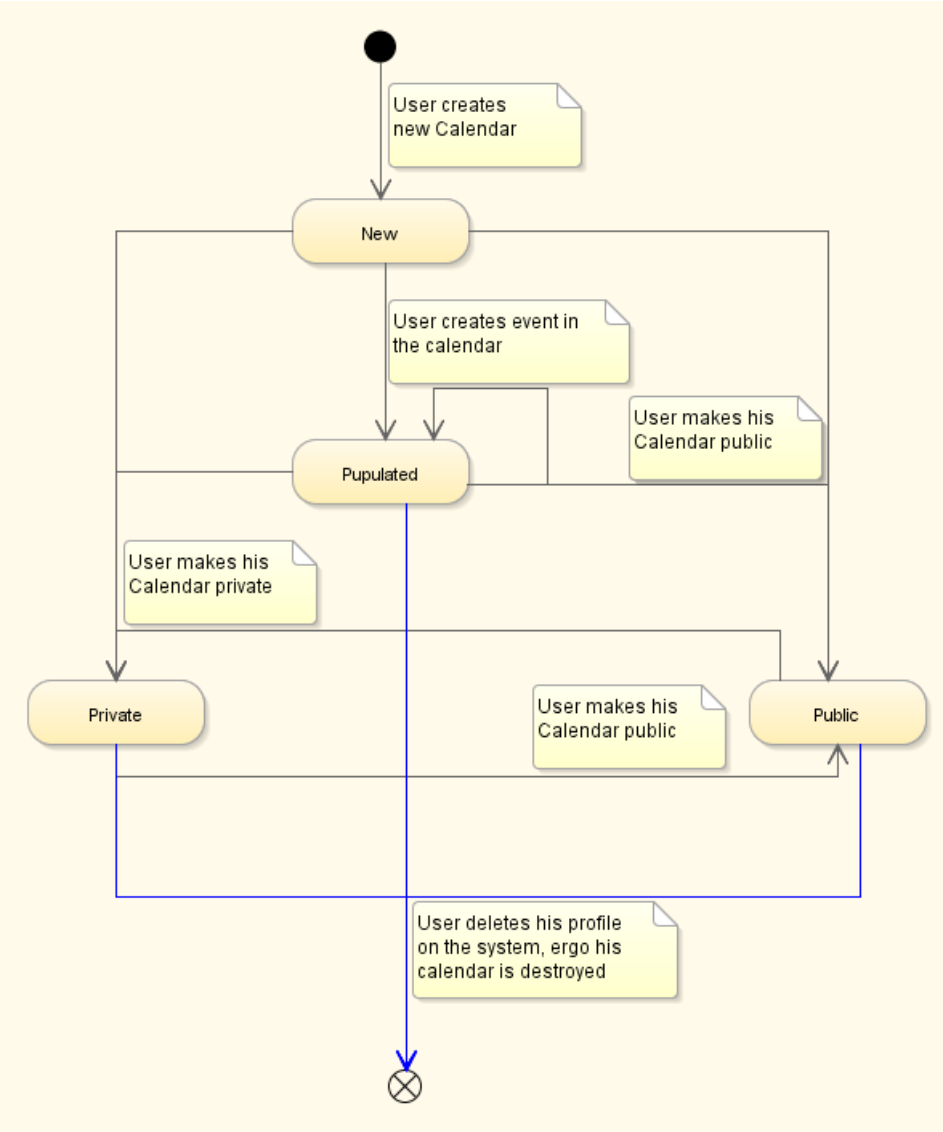


Sequence diagram 28: Decline Invitation sequence diagram depicts the detailed inter-communication between system components when 'decline invitation' use case is performed

Code	• SD-023
Name	• Decline Invitation diagram
Use case diagram reference	• USC-008(View my calendar(Log in)) • USC-020(Accept/Decline event invitation)
Simplification reference	• SDS-001(Generate My Calendars Page-Simplified)

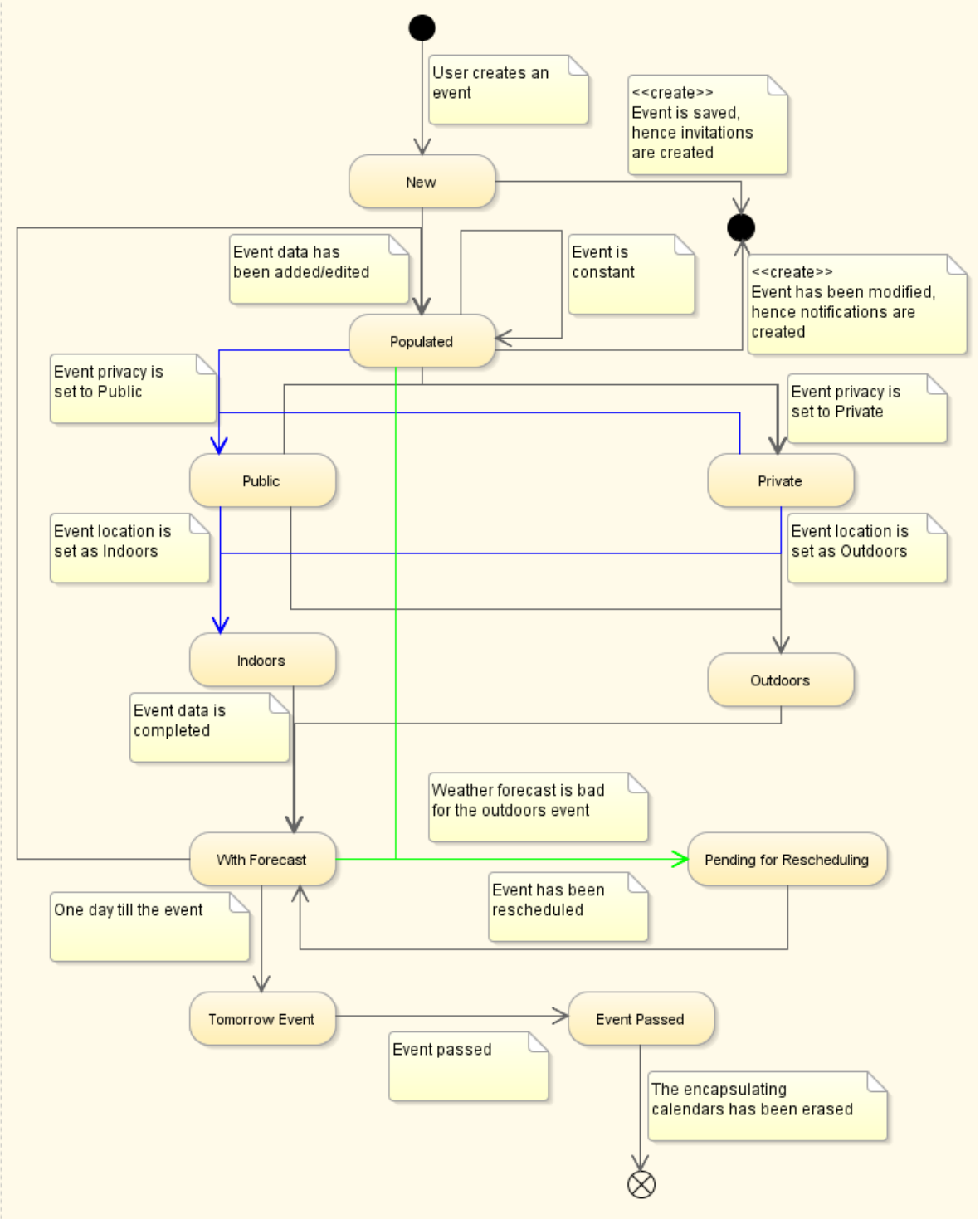
3.2.5. State chart model

3.2.5.1. Calendar state chart diagram



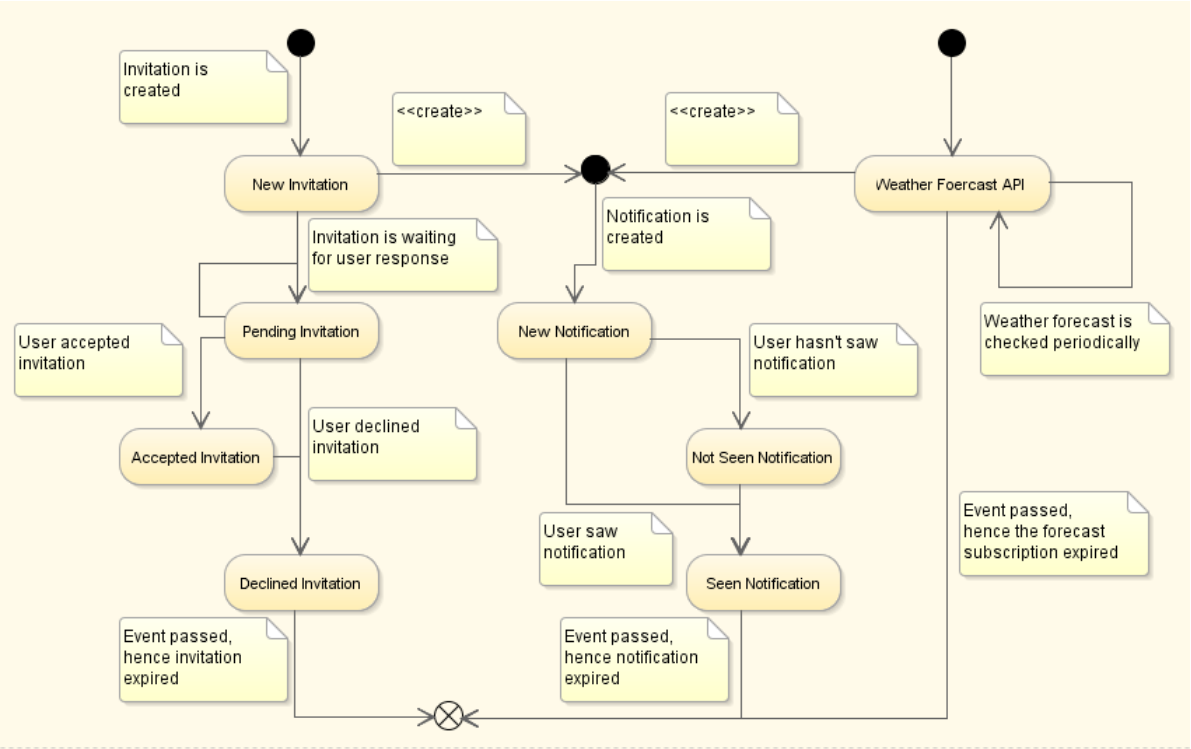
State chart diagram 1: Calendar state chart diagram depicts all the possible states that the calendar object can take through his life time in the system, different coloring of the arcs are just to avoid wrong understanding of intersecting arcs

3.2.5.2. *Event state chart diagram*



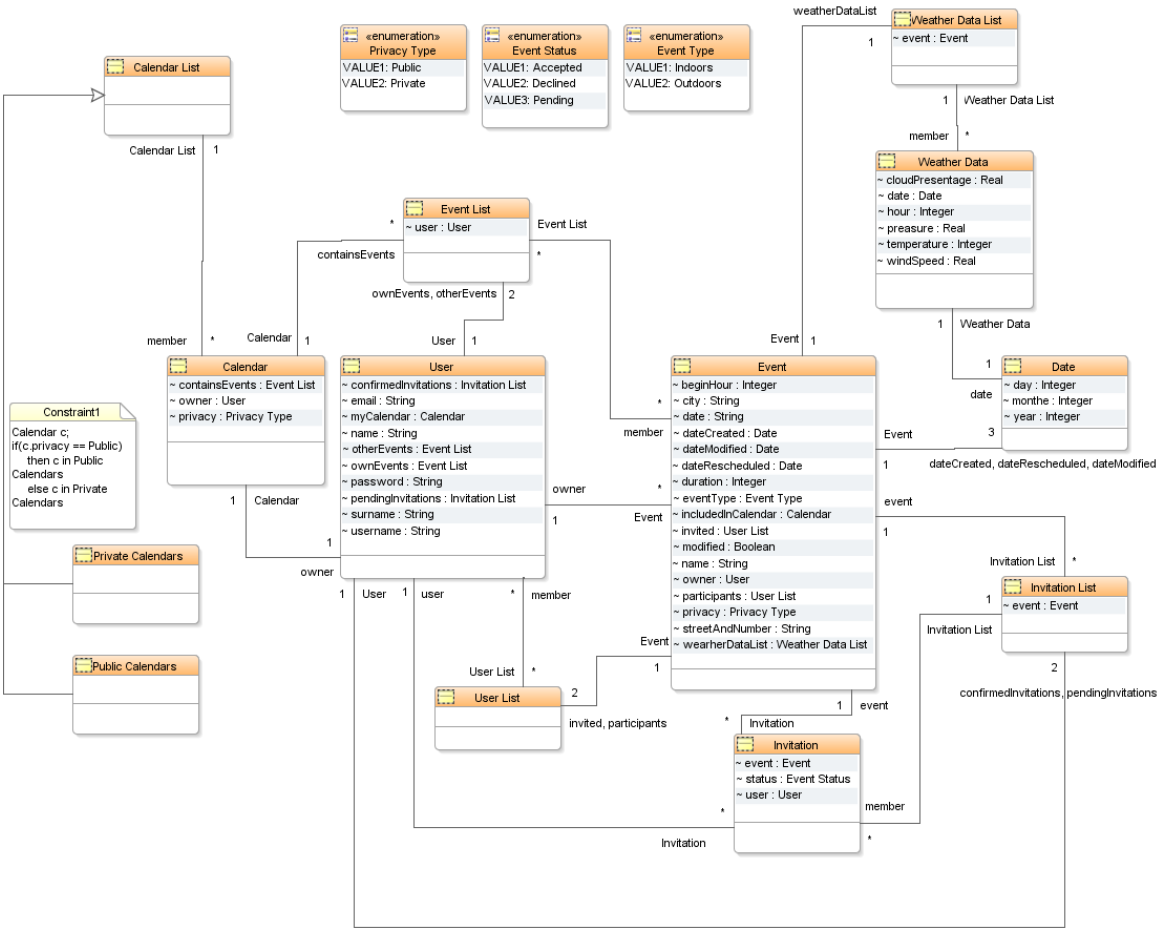
State chart diagram 2: Event state chart diagram depicts all the possible states that the event object can take through his life time in the system, different coloring of the arcs are just to avoid wrong understanding of intersecting arcs, also in this diagram we depicted the creation point of the notification object since its creation is strictly dictated by the event object

3.2.5.3. *Notification/Invitation state chart diagram*



State chart diagram 3: Notification/Invitation state chart diagram depicts all the possible states that the notification and invitation objects can take through their life times in the system, also in this diagram we depicted Weather forecast API high level abstraction of state diagram due to its influence to creation of notification object

3.2.6. Architecture model

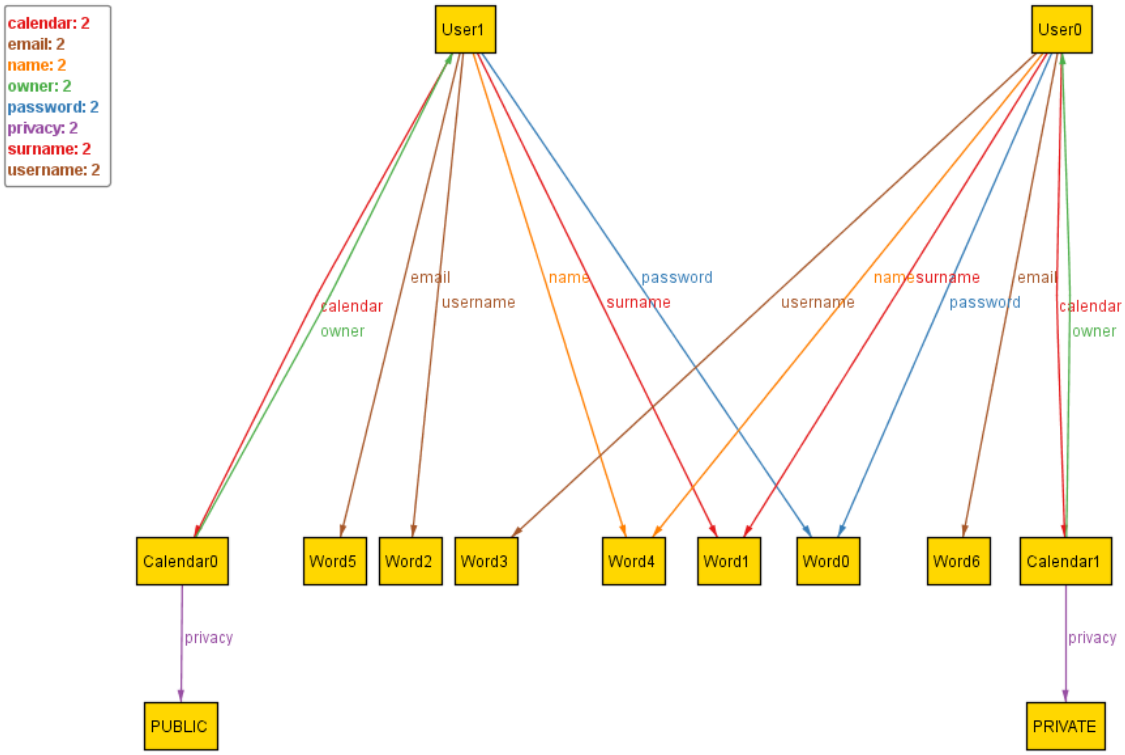


Class diagram 1: This diagram represents object hierarchy in the MeteoCal system, in this diagram we also depicted enumerations needed in certain cases.

3.2.7. Alloy model

```
module OnlineCalendar

//STATIC SETS
enum PrivacyStatus {PRIVATE, PUBLIC}


//ABSTRACT SIGNATURES
abstract sig Word {}


//SIGNATURES
sig User
{
   name: one Word,
   surname: one Word,
   email: one Word,
   username: one Word,
   password: one Word,
   calendar: one Calendar
}

sig Calendar
{
   owner: one User,
   privacy: one PrivacyStatus
}

//FACTS

//the users can have the same names, same surnames, same passwords, but no same username, email or calendar
fact noTwoUsersWithSameData
{
   no disj u1, u2 : User | u1.email = u2.email
   no disj u1, u2 : User | u1.username = u2.username
   no disj u1, u2 : User | u1.calendar = u2.calendar
   no disj u1, u2 : User | u1.name = u2.surname
   no disj u1, u2 : User | u1.name = u2.email
   no disj u1, u2 : User | u1.name = u2.password
   no disj u1, u2 : User | u1.name = u2.username
   no disj u1, u2 : User | u1.surname = u2.name
   no disj u1, u2 : User | u1.surname = u2.email
   no disj u1, u2 : User | u1.surname = u2.username
   no disj u1, u2 : User | u1.surname = u2.password
   no disj u1, u2 : User | u1.email = u2.name
   no disj u1, u2 : User | u1.email = u2.surname
   no disj u1, u2 : User | u1.email = u2.username
   no disj u1, u2 : User | u1.email = u2.password
   no disj u1, u2 : User | u1.username = u2.name
   no disj u1, u2 : User | u1.username = u2.surname
   no disj u1, u2 : User | u1.username = u2.email
   no disj u1, u2 : User | u1.username = u2.password
}
```

```
//no user can have same attribute values
fact userDataUniqueness
{
   all  u : User | u.name != u.surname
   all  u : User | u.name != u.email
   all  u : User | u.name != u.username
   all  u : User | u.name != u.password
   all  u : User | u.surname != u.email
   all  u : User | u.surname != u.username
   all  u : User | u.surname != u.password
   all  u : User | u.email != u.username
   all  u : User | u.email != u.password
   all  u : User | u.username != u.password
}

//relate one calendar to one user
fact userCalendarRelation
{
   all u: User, c: Calendar |  u = c.owner implies u.calendar = c
}
fact numOfCalendarOwners
{
   all c: Calendar | #(c.owner) = 1
}

pred show()
{
   #User = 2
}
run show for 10
```

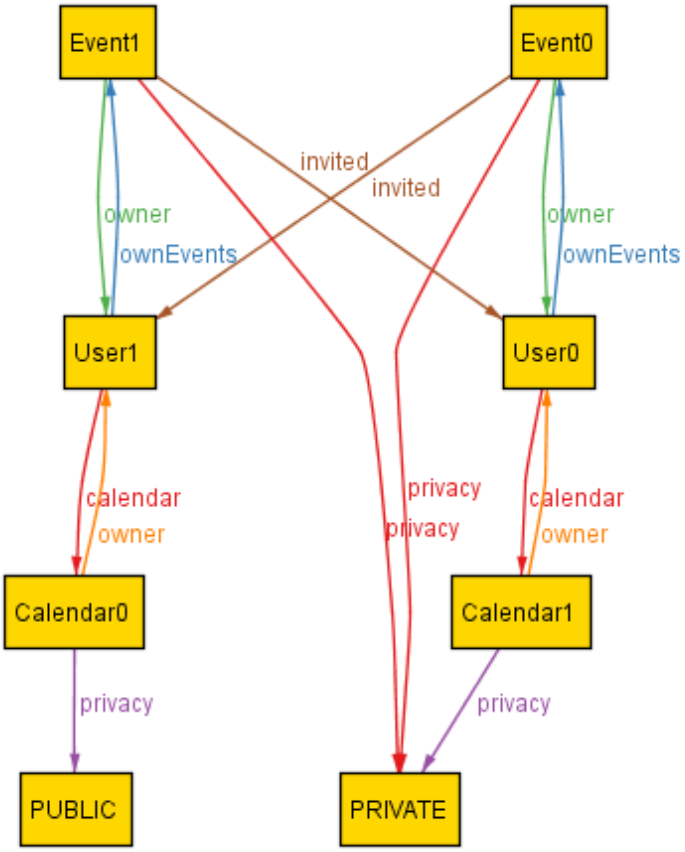


Alloy diagram 1: User-Calendar one-to-one relation

It's time to create an *Event*, connect the owner of the event to that event, and connect weather data list to the same event. Each event has one owner and one WeatherDataList. For now, we will comment out WeatherDataList to show ownership relations.

```
//added enumerations
enum Boolean {TRUE, FALSE}
enum EventType {INDOOR, OUTDOOR}

//ABSTRACT SIGNATURES
abstract sig Word {}
abstract sig Date {}

//SIGNATURES
//removed some fields that affected the visibility of a current concept
sig User
{
   calendar: one Calendar,
   pendingInvitation: set Event,
   ownEvents: set Event
}
//added signature
sig Event
{
   owner: one User,
   name: one Word,
   date: one Date,
   beginHour: one Int,
   duration: one Int,
   city: one Word,
   streetAndNumber: one Word,
   eventType: one EventType,
   privacy: one PrivacyStatus,
   invited: set User,
   //participants: set User,
   //weatherDataList: one WeatherDataList,
   dateCreated: one Date,
   modified: one Boolean,
   dateModified: lone Date,
   //dateRescheduled: lone Date
}



//FACTS

//relate one created event to one user
fact userEventRelation
{
   all u: User, e: Event |  e.owner = u implies e in e.owner.ownEvents
   all u: User, e: Event |  (e.owner != u) implies (e not in u.ownEvents)

}


pred show()
{
   #User = 2
   #Event = 2
   #(Event.owner)>1
}
run show for 10
```

Milan, 9th November, 2014



Alloy diagram 1: Each event must have only one owner

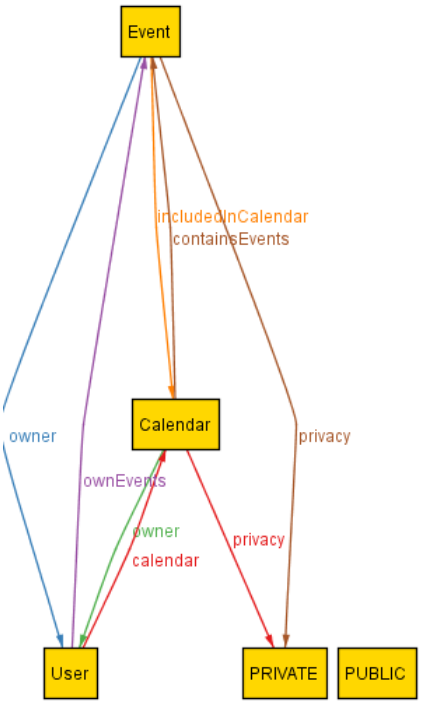
1st Note: For simplicity, other attributes of Event are omitted.

2nd Note: Users are invited to their own events. Later we will add that those invitations are automatically accepted.

Now, we will include created event in owner's calendar. Only the changed code is shown below:

```
//SIGNATURES

sig User
{//a lot of attributes from previous version of user are omitted for simplicity
   calendar: one Calendar,
   pendingInvitation: set Event,
   //confirmedInvitation: set Event,
   //declinedInvitation: set Event,
   ownEvents: set Event,
   //otherEvents: set Event
}

sig Calendar
{
   owner: one User,
   privacy: one PrivacyStatus,
   containsEvents: set Event
}
```

```
sig Event
{
   owner: one User,
   //name: one Word,
   //date: one Date,
   //beginHour: one Int,
   //duration: one Int,
   //city: one Word,
   //streetAndNumber: one Word,
   //eventType: one EventType,
   privacy: one PrivacyStatus,
   invited: set User,
   includedInCalendar: Calendar
   //participants: set User,
   //weatherDataList: one WeatherDataList,
   //dateCreated: one Date,
   //modified: one Boolean,
   //dateModified: lone Date,
   //dateRescheduled: lone Date
}
fact EventCalendarOwnerRelation
{
            all e: Event, u: User | e.owner = u implies e in e.owner.calendar.containsEvents
}
pred show()
{
   #User = 1
   #Event = 1
   #(Event.owner)>0
}
run show for 10
```



Alloy diagram 2: Owner's events included in his/her calendar

Milan, 9th November, 2014

Next, we will add invitations to events from different users, and create three subtypes of response *confirmedInvitation*, *declinedInvitation*, *pendingInvitation*). Invitation to own event is going to automatically create the *confirmedInvitation*. Code (note – for simplicity some attributes of signatures and facts are omitted):

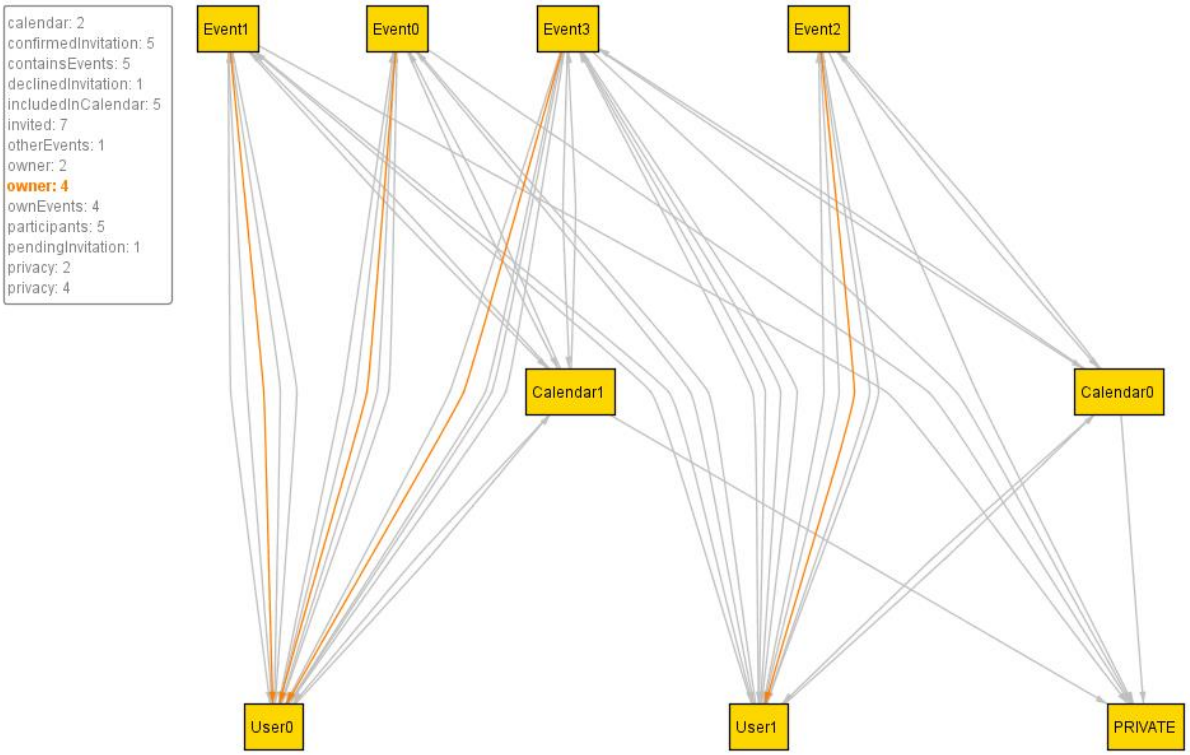
```
module OnlineCalendar

//STATIC SETS
enum PrivacyStatus {PRIVATE, PUBLIC}
enum Boolean {TRUE, FALSE}
enum EventType {INDOOR, OUTDOOR}

//ABSTRACT SIGNATURES
abstract sig Word {}
abstract sig Date {}

//SIGNATURES DEFINITIONS
sig User
{
   calendar: one Calendar,
   pendingInvitation: set Event,
   confirmedInvitation: set Event,
   declinedInvitation: set Event,
   ownEvents: set Event,
   otherEvents: set Event//the events from other users to for which the user confirmed the invitation
}
sig Calendar
{
   owner: one User,
   privacy: one PrivacyStatus,
   containsEvents: set Event
}
sig Event
{
   owner: one User,
   //name: one Word,
   //date: one Date,
   //beginHour: one Int,
   //duration: one Int,
   //city: one Word,
   //streetAndNumber: one Word,
   //eventType: one EventType,
   privacy: one PrivacyStatus,
   invited: set User,
   includedInCalendar: set Calendar,
   participants: set User,
   //weatherDataList: one WeatherDataList,
   //dateCreated: one Date,
   //modified: one Boolean,
   //dateModified: lone Date,
   //dateRescheduled: lone Date
}
//FACTS
//the users can have the same names, same surnames, same passwords, but no same username, email or calendar
fact noTwoUsersWithSameData
{//a lot of rules omitted for simplicity
   no disj u1, u2 : User | u1.calendar = u2.calendar
}
```

```
//relate one calendar to one user
fact userCalendarRelation
{
  all u: User, c: Calendar | u = c.owner implies u.calendar = c
}
fact numOfCalendarOwners
{
  all c: Calendar | #(c.owner) = 1
}

//relate one created event to a user who is its owner
fact userEventRelation
{
  all u: User, e: Event |  (e.owner  = u) implies (e in e.owner.ownEvents)
    all u: User, e: Event | (e.owner != u) implies (e not in u.ownEvents)
}


//if user is the owner of an event, that user is automatically invited
fact ownerAutomaticallyInvited
{
    all u:User, e: Event | (e.owner=u) implies (e.owner in e.invited)
}

//if user is not invited to an event, that event must not be contained inside any set of user's invitations
fact notInvitedNoResponse
{
    all  e: Event, u: User | (u not in e.invited) implies (e not in u.confirmedInvitation)
    all  e: Event, u: User | (u not in e.invited) implies (e not in u.declinedInvitation)
    all  e: Event, u: User | (u not in e.invited) implies (e not in u.pendingInvitation)
}

//if owner is invited to his/her event, invitation is automatically confirmed
fact ownerInvitationAutomaticallyConfirmed
{
  all u:User, e: Event | (e.owner=u) implies (e in e.owner.confirmedInvitation)
}

//if user has confirmed the invitation for an event, that event should be contained inside his/her calendar
fact EventCalendarOwnerRelation
{//notice that now we don't add just e.owner into user's calendar
  all e: Event, u: User | (e in u.confirmedInvitation) implies (e in u.calendar.containsEvents)
  all e: Event, u: User | (e in u.calendar.containsEvents) implies (e in u.confirmedInvitation)
  all e: Event, u: User | (e in u.calendar.containsEvents) implies (u.calendar in e.includedInCalendar)
  all e: Event, u: User | (u.calendar in e.includedInCalendar) implies (e in u.calendar.containsEvents)
}

//if invitation to an event is confirmed, it cannot be declined or pending at the same time
fact disjointOtherInvitationIfConfirmed
{
    all  e: Event, u: User | (e in u.confirmedInvitation) implies (e not in u.declinedInvitation)
    all  e: Event, u: User | (e in u.confirmedInvitation) implies (e not in u.pendingInvitation)
}

//if invitation to an event is declined, it cannot be confirmed or pending at the same time
fact disjointOtherInvitationIfDeclined
{
    all  e: Event, u: User | (e in u.declinedInvitation) implies (e not in u.confirmedInvitation)
    all  e: Event, u: User | (e in u.declinedInvitation) implies (e not in u.pendingInvitation)
}
```

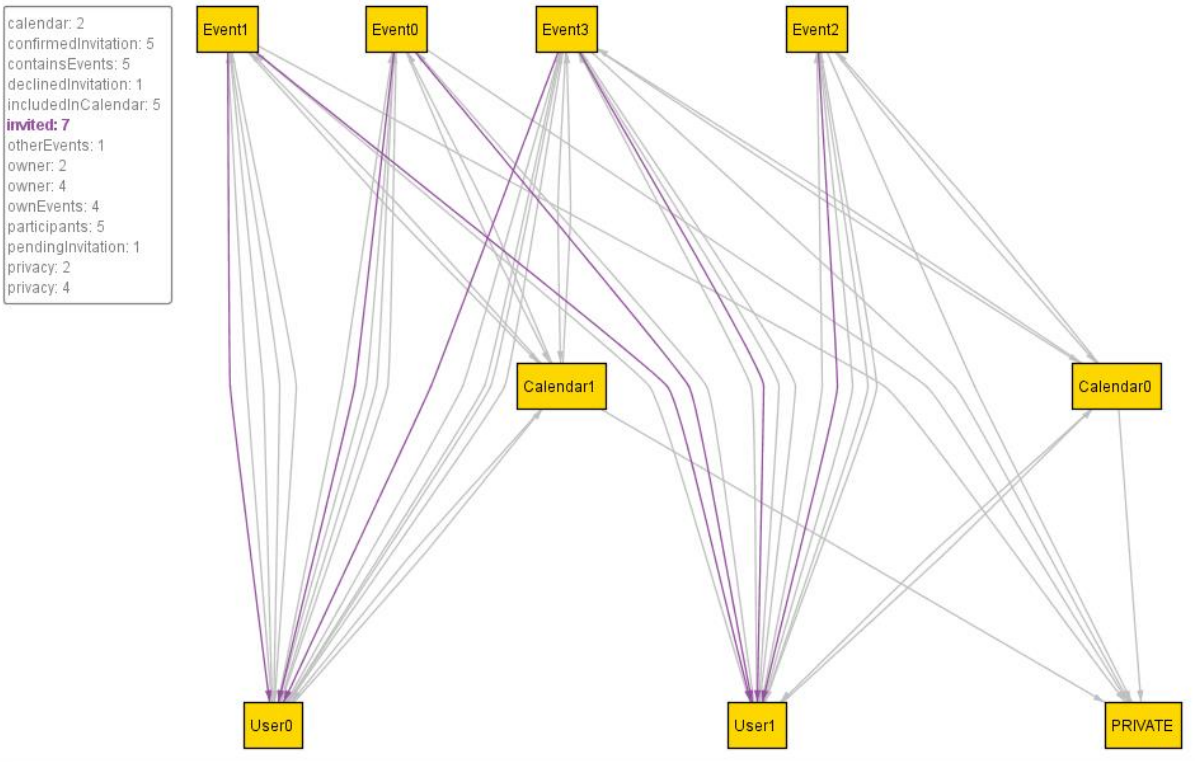
Milan, 9th November, 2014

```
//if invitation to an event is pending, it cannot be confirmed or declined at the same time
fact disjointOtherInvitationIfPending
{
      all  e: Event, u: User | (e in u.pendingInvitation) implies (e not in u.confirmedInvitation)
      all  e: Event, u: User | (e in u.pendingInvitation) implies (e not in u.declinedInvitation)
}


//if the user is invited to an event, his/her response must be either confirmed, declined or pending
fact oneInvitationResponseMustBePresent
{
   all  e: Event, u: User | ((u in e.invited) and (e not in u.confirmedInvitation) and (e not in u.declinedInvitation)) implies (e
in u.pendingInvitation)
   all  e: Event, u: User | (e  in u.pendingInvitation) implies ((u in e.invited) and (e not in u.confirmedInvitation) and (e not
in u.declinedInvitation))
   all  e: Event, u: User | ((u in e.invited) and (e not in u.pendingInvitation) and (e not in u.declinedInvitation)) implies (e  in
u.confirmedInvitation)
   all  e: Event, u: User | (e  in u.confirmedInvitation) implies ((u in e.invited) and (e not in u.pendingInvitation) and (e not
in u.declinedInvitation))
   all  e: Event, u: User | ((u in e.invited) and (e not in u.pendingInvitation) and (e not in u.declinedInvitation)) implies (e  in
u.confirmedInvitation)
   all  e: Event, u: User | (e  in u.confirmedInvitation) implies ((u in e.invited) and (e not in u.pendingInvitation) and (e not
in u.declinedInvitation))
}


//if user was invited to an event of which he/she is not owner, and he confirmed invitation, that event goes to otherEvents
set
fact userEventRelation
{
   all u: User, e: Event |  ((e in u.confirmedInvitation) and (e not in u.ownEvents)) implies (e in u.otherEvents)
   all u: User, e: Event |  (e in u.otherEvents) implies ((e in u.confirmedInvitation) and (e not in u.ownEvents))
}


//if the user has confirmed the invitation to an event of another user, the user is participant of that event
fact Participants
{
   all e: Event, u: User | (e in u.confirmedInvitation) implies (u in e.participants)
   all e: Event, u: User | (u in e.participants) implies (e in u.confirmedInvitation)
}


pred show()
{
   #User = 2
   #Event = 4
   #(Event.owner)>1
   #(invited)>6
   #(User.confirmedInvitation)>2
   #(User.declinedInvitation)>0
   #(User.pendingInvitation)>0
}
run show for 15
```

Milan, 9th November, 2014

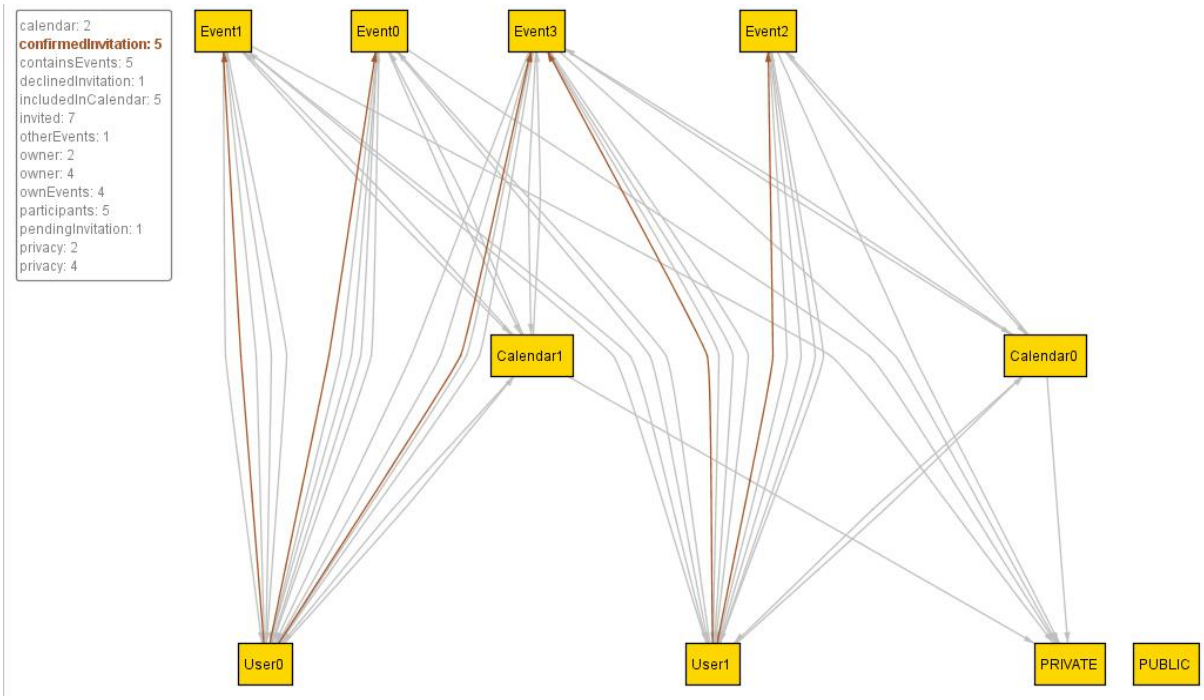


Alloy diagram 3: Ownership relation (which events are created by which user)



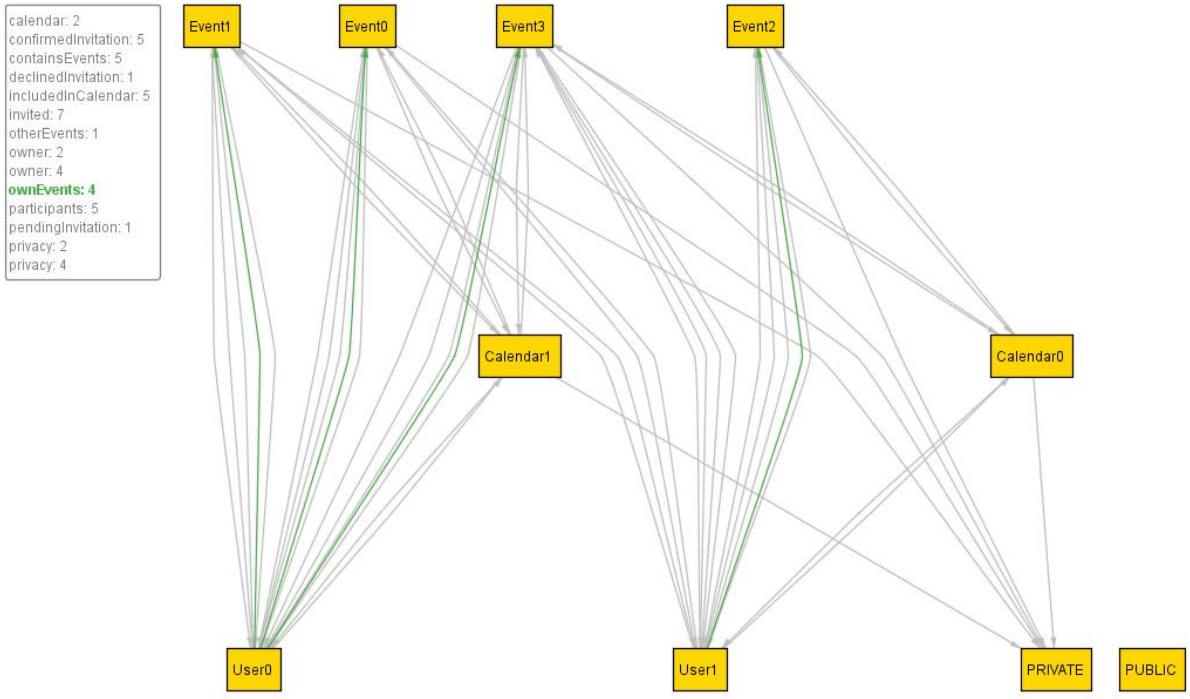
Alloy diagram 4: Invitations to events

Note: Owners are automatically invited to their own events

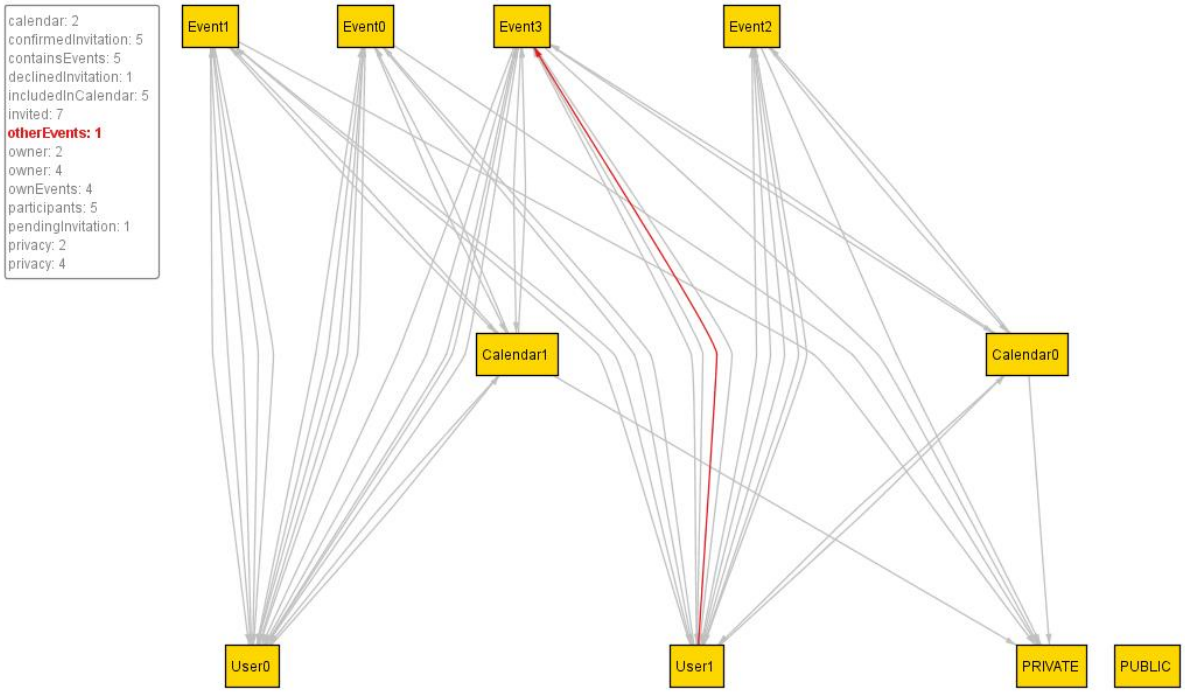


Alloy diagram 5: Confirmed invitations to events

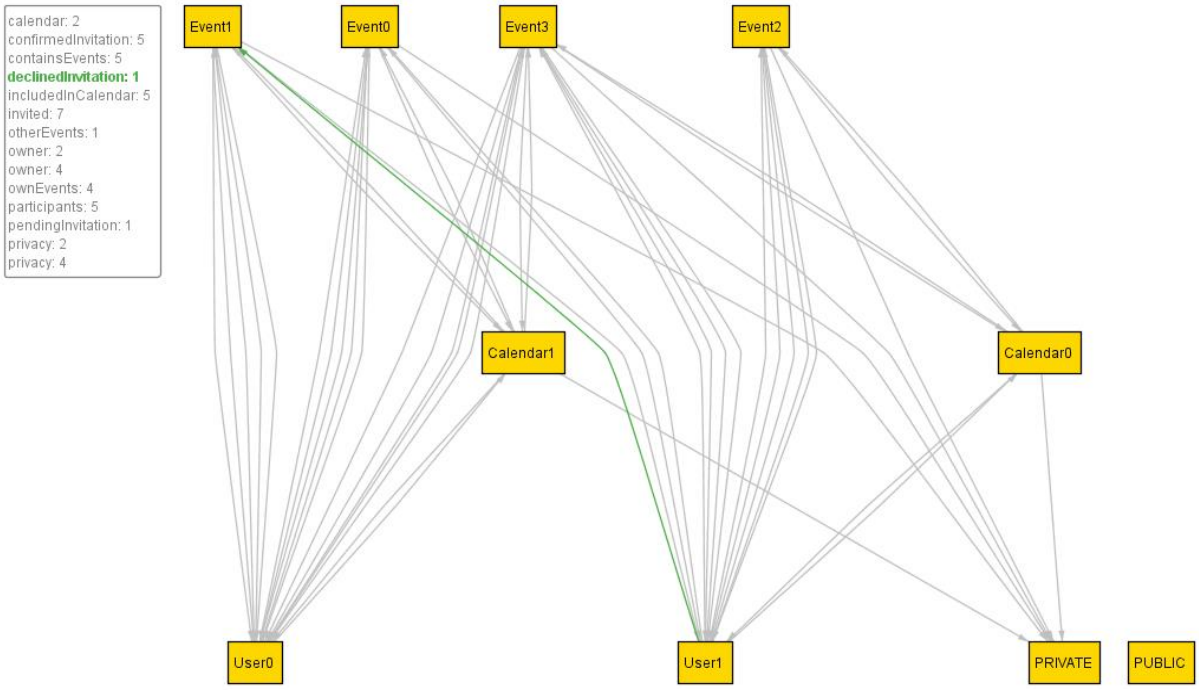
Note: The invitations to owners are automatically accepted and those events are placed inside own events showed in next figure.



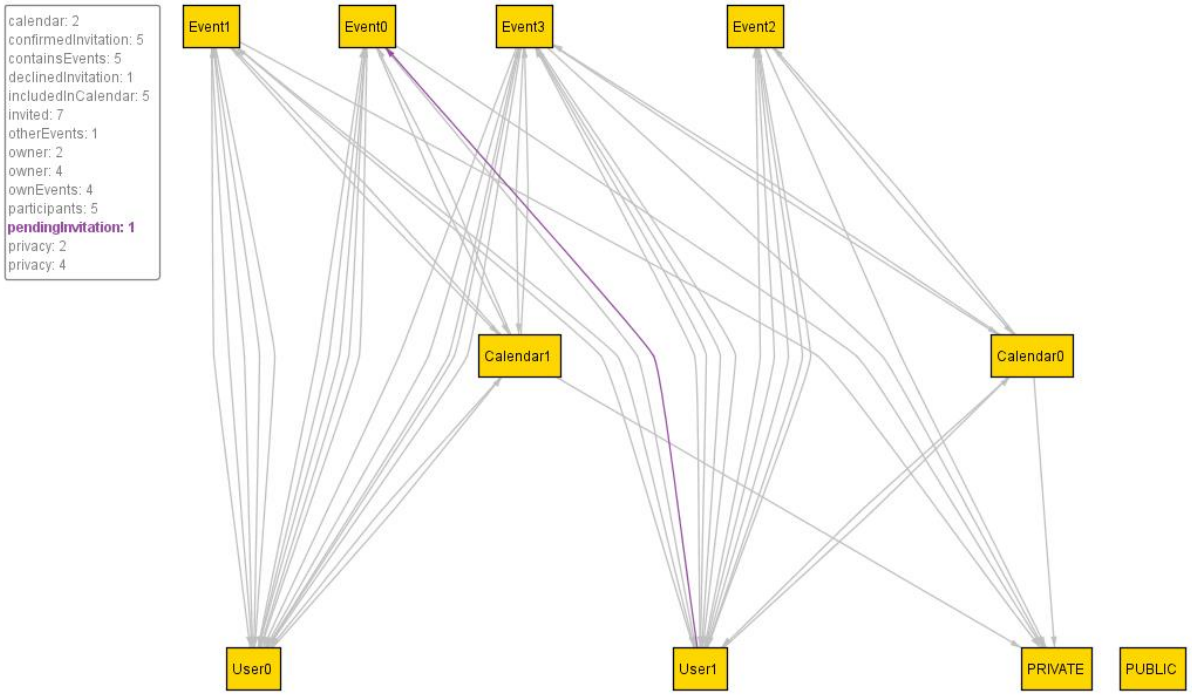
Alloy diagram 6: Own events (User0 owns Event1, Event0, Event3. User1 owns Event2)



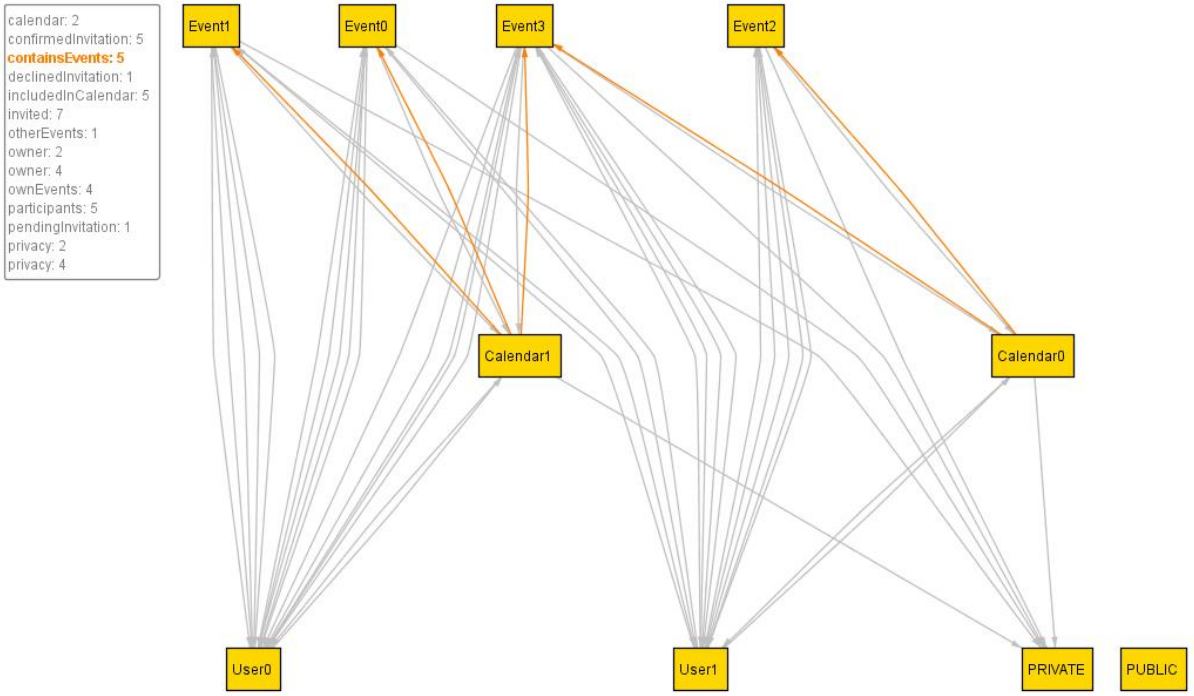
Alloy diagram 7: Other events (the events of other users, to which our user was invited and has confirmed the invitation)



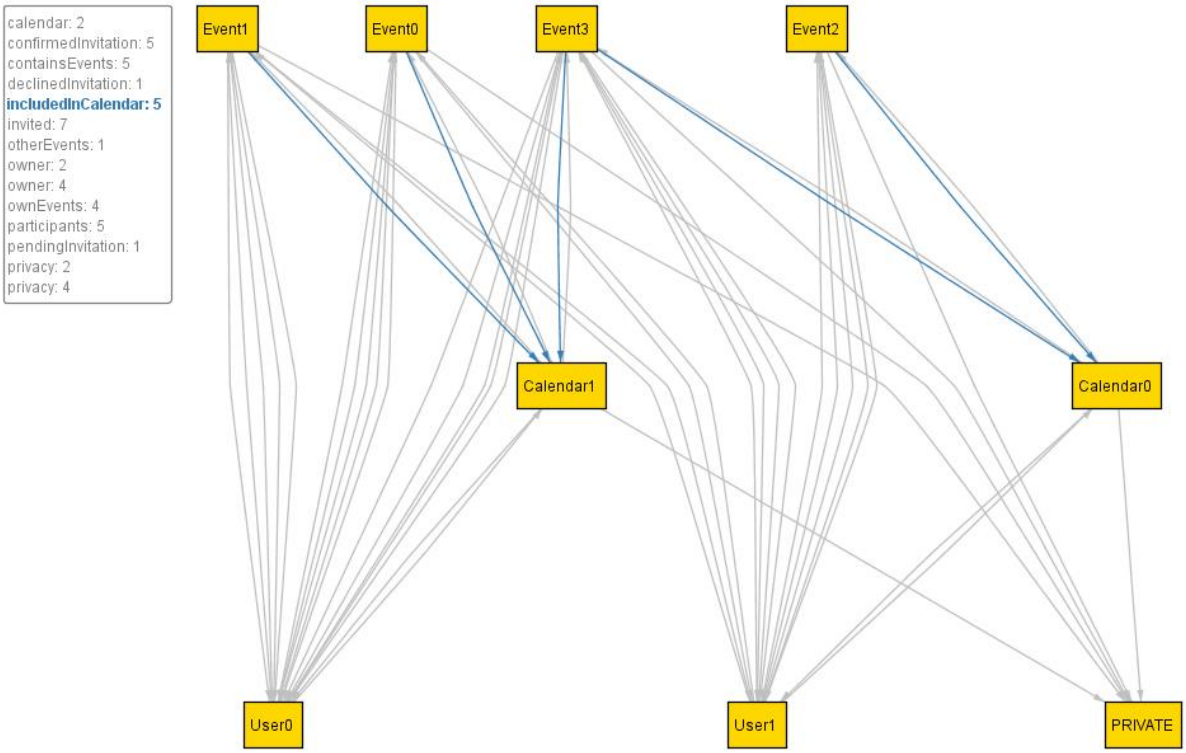
Alloy diagram 8: Declined invitations to events by users (Remember that User0 was the owner of Event1, and that User1 was invited to that event)



Alloy diagram 9: Pending invitations (the ones that are neither confirmed, neither declined)



Alloy diagram 10: Confirmed invitations to events lead to adding those events to users' calendars (both own events and events of other users to which one is participating are added)

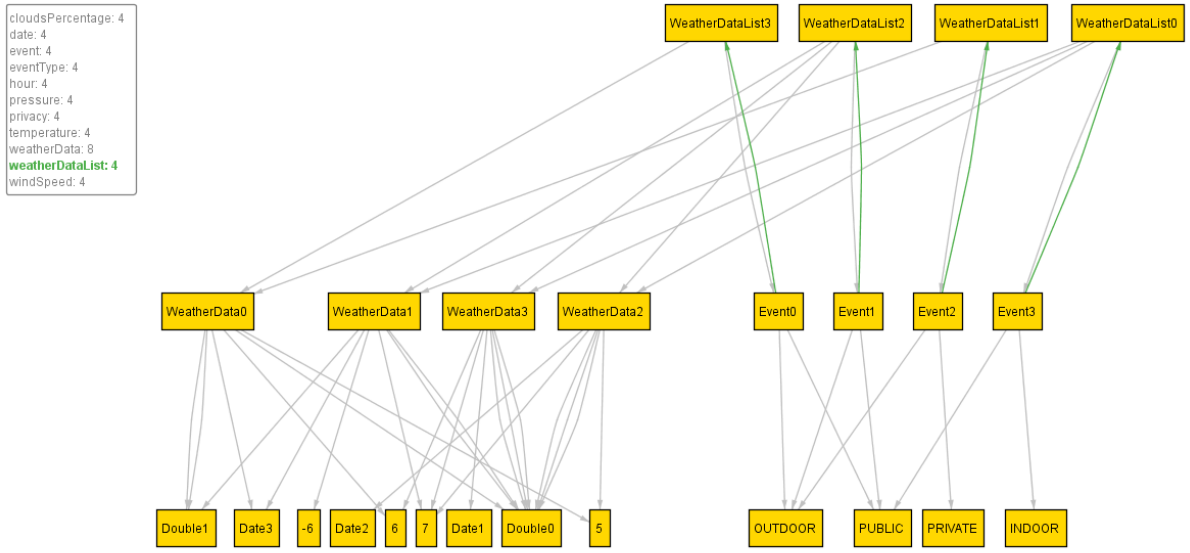


Alloy diagram 11: Events hold the information about calendars to which they are added.

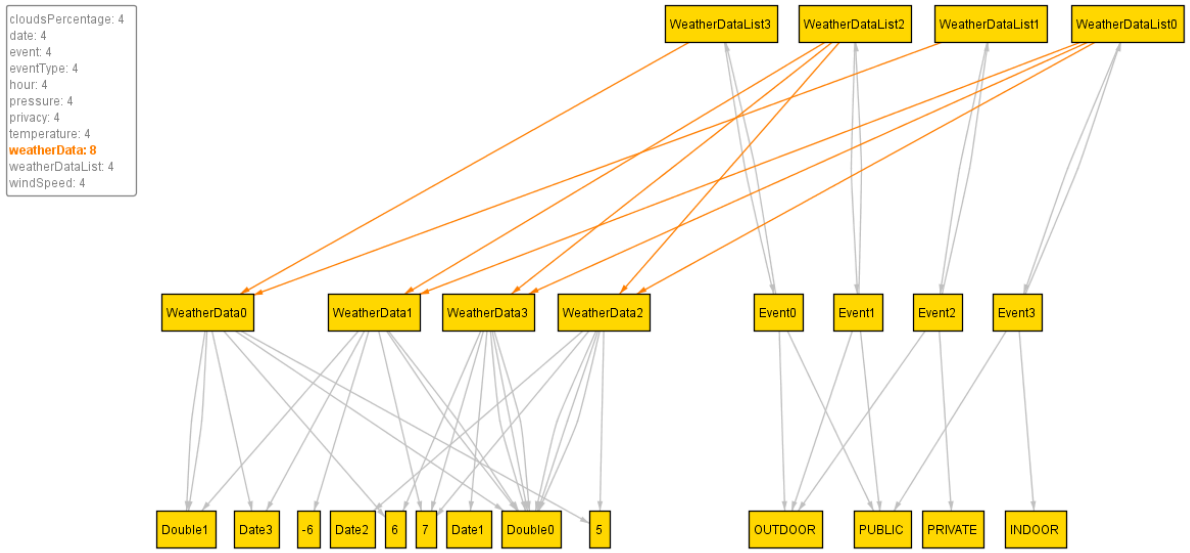
Next, we add condition that every event must have weather cast data attached to it. Code (simplified):

```
module OnlineCalendar

//STATIC SETS
enum PrivacyStatus {PRIVATE, PUBLIC}
enum Boolean {TRUE, FALSE}
enum EventType {INDOOR, OUTDOOR}

//ABSTRACT SIGNATURES
abstract sig Word {}
abstract sig Date {}
abstract sig Double {}

//SIGNATURES

sig Event
{
   eventType: one EventType,
   privacy: one PrivacyStatus,
   weatherDataList: one WeatherDataList
} { #weatherDataList=1 }


sig WeatherData
{
   date: one Date,
   hour: one Int,
   temperature: one Int,
   windSpeed: one Double,
   pressure: one Double,
   cloudsPercentage: one Double
}
```

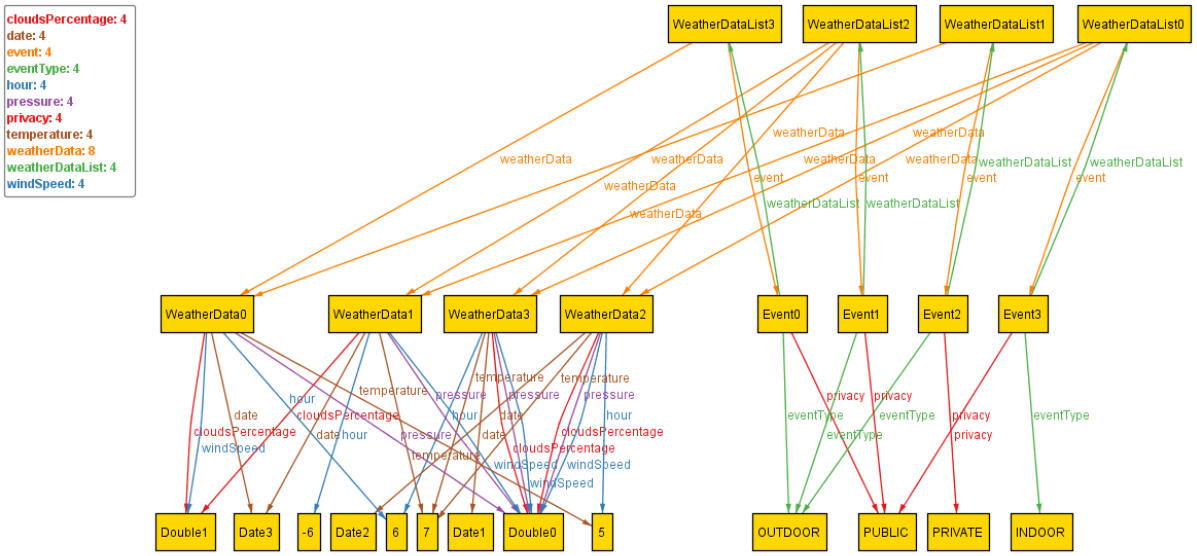
```
sig WeatherDataList
{
   weatherData: set WeatherData,
   event: one Event
} { #weatherData>0}
//FACTS
//relate one weatherdatalist to one event
fact eventWeatherListRelation
{
   all e: Event, w: WeatherDataList | e = w.event implies e.weatherDataList = w
   all e: Event, w: WeatherDataList | e.weatherDataList = w implies e = w.event
}
pred show()
{
   #Event = 4
}
run show for 4
```



Alloy diagram 12: Each event has one weather data list, that is, predicted weather conditions for created event.



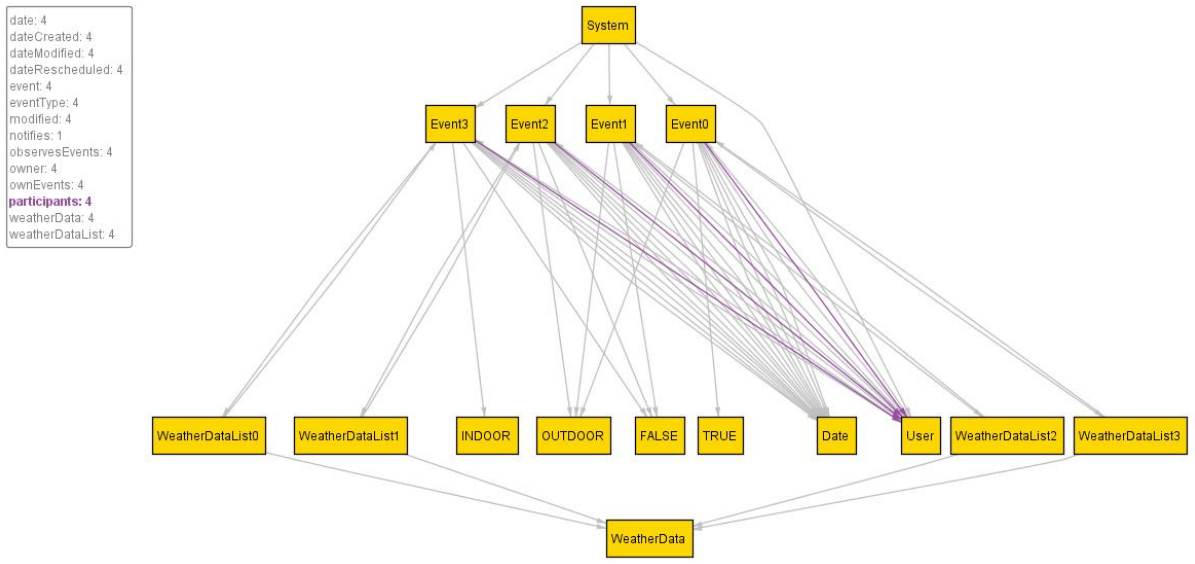
Alloy diagram 13: Weather data may be shared among different events, because they might overlap in hours.



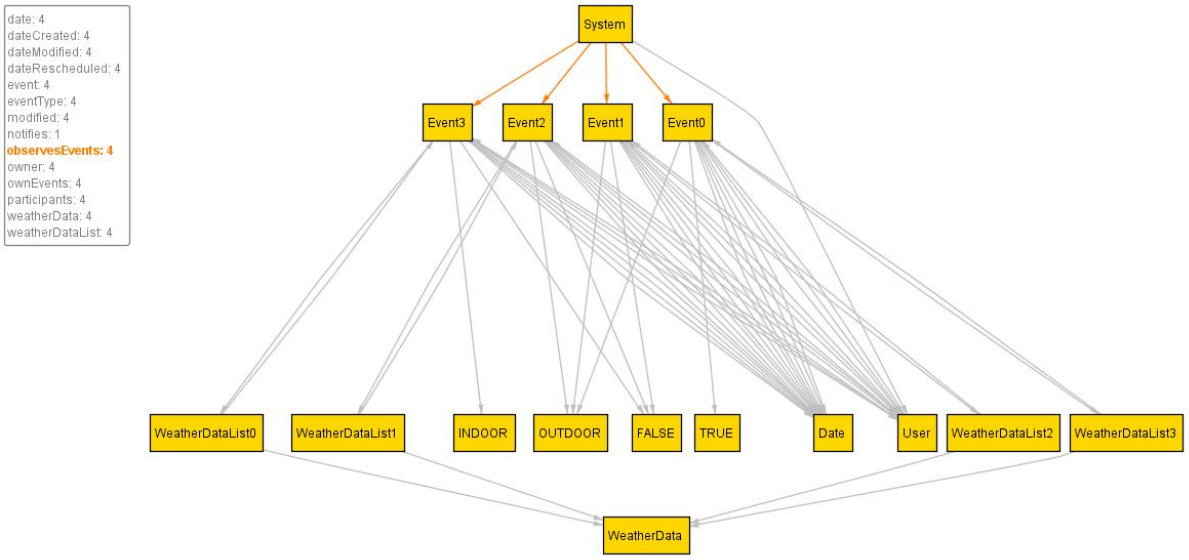
Alloy diagram 14: On the left -Weather data may share simple data values like integers, double numbers, etc. that represent attributes values of WeatherData. No limitations are introduced here. On the right – Each event has its own type (indoor or outdoor) and also its privacy.

Now we will create signature *System* that will observe the user's events, and notify all the participants about modification of the event:

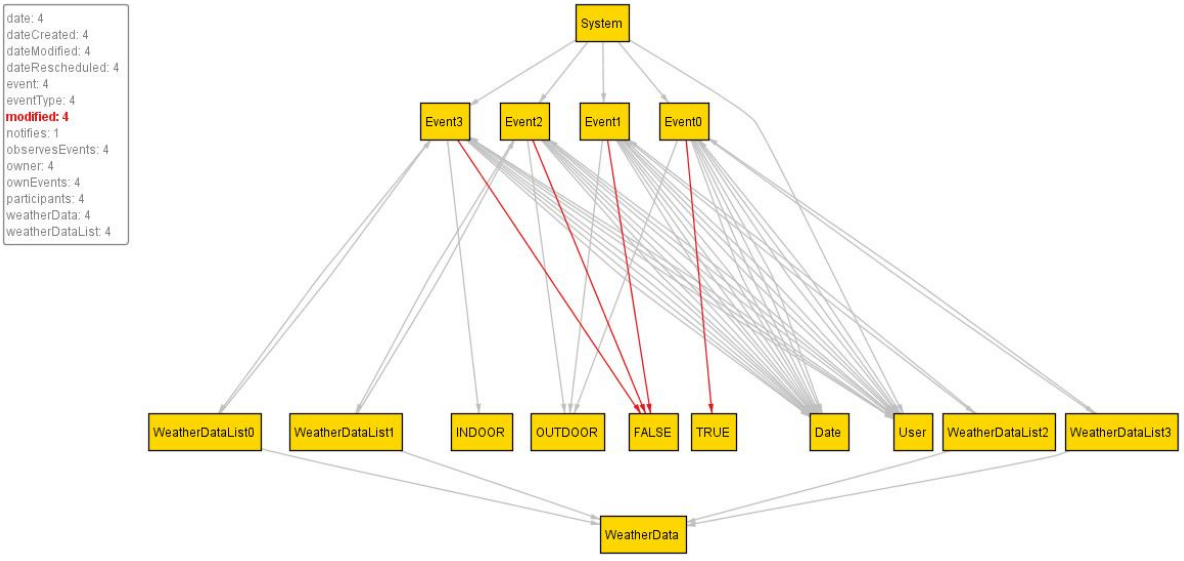
```
one sig System
{
   notifies: set User,
   observesEvents: set Event
} {#observesEvents = #Event }
fact eventWeatherListRelation
{
   all e: Event, w: WeatherDataList | e = w.event implies e.weatherDataList = w
   all e: Event, w: WeatherDataList | e.weatherDataList = w implies e = w.event
}
fact eventModified
{
      all e: Event, u: User,s:System | ((e.modified = TRUE) and (u in e.participants)) implies (u in s.notifies)
}
```



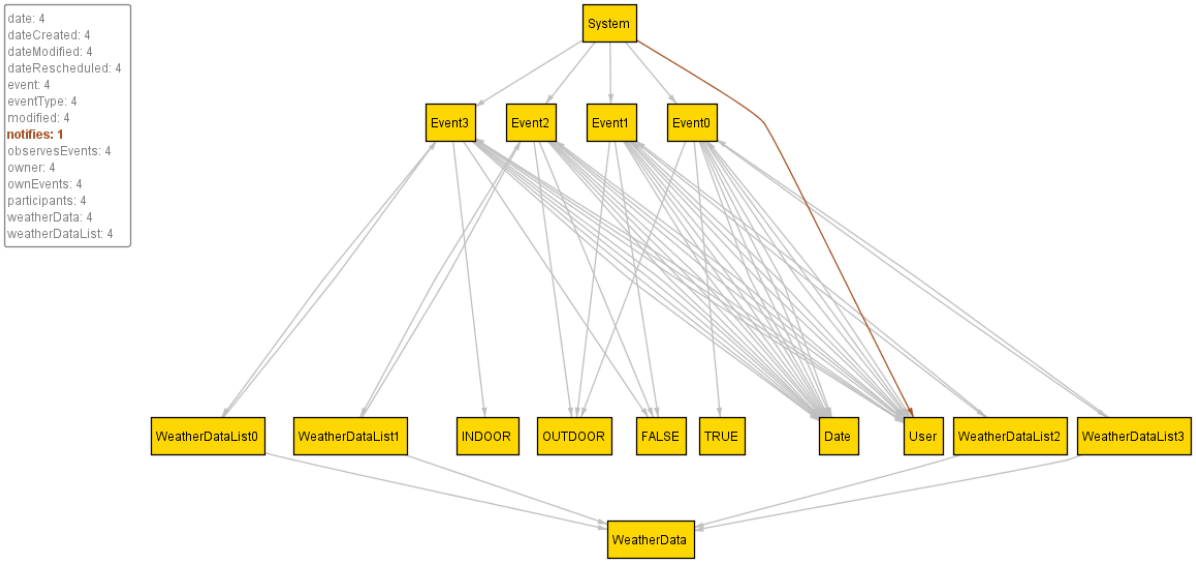
Alloy diagram 15: The user participate to some events in the system.



Alloy diagram 16: System observes all events in the system.



Alloy diagram 17: Some of the event in which our user participate is being modified.



Alloy diagram 18: System notifies all participants about modifications of event.

The final model:

```
module OnlineCalendar

//STATIC SETS
enum PrivacyStatus {PRIVATE, PUBLIC}
enum Boolean {TRUE, FALSE}
enum EventType {INDOOR, OUTDOOR}

//ABSTRACT SIGNATURES
abstract sig Word {}
abstract sig Date {}
abstract sig Double {}


//SIGNATURES DEFINITIONS
sig User
{
   name: one Word,
   surname: one Word,
   email: one Word,
   username: one Word,
   password: one Word,
   calendar: one Calendar,
   pendingInvitation: set Event,
   confirmedInvitation: set Event,
   declinedInvitation: set Event,
   ownEvents: set Event,
   otherEvents: set Event
}

sig Calendar
{
   owner: one User,
   privacy: one PrivacyStatus,
   containsEvents: set Event
}
```

```
sig Event
{
   owner: one User,
   name: one Word,
   date: one Date,
   beginHour: one Int,
   duration: one Int,
   city: one Word,
   streetAndNumber: one Word,
   eventType: one EventType,
   privacy: one PrivacyStatus,
   invited: set User,
   includedInCalendar: set Calendar,
   participants: set User,
   weatherDataList: one WeatherDataList,
   dateCreated: one Date,
   modified: one Boolean,
   dateModified: lone Date,
   dateRescheduled: lone Date
} { #weatherDataList=1 }
sig WeatherData
{
   date: one Date,
   hour: one Int,
   temperature: one Int,
   windSpeed: one Double,
   pressure: one Double,
   cloudsPercentage: one Double
}
sig WeatherDataList
{
   weatherData: set WeatherData,
   event: one Event
} { #weatherData>0}
one sig System
{
   notifies: set User,
   observesEvents: set Event
} {#observesEvents = #Event }
//FACTS
//the users can have the same names, same surnames, same passwords, but no same username, email or calendar
fact noTwoUsersWithSameData
{
   no disj u1, u2 : User | u1.email = u2.email
   no disj u1, u2 : User | u1.username = u2.username
   no disj u1, u2 : User | u1.calendar = u2.calendar
   no disj u1, u2 : User | u1.name = u2.surname
   no disj u1, u2 : User | u1.name = u2.email
   no disj u1, u2 : User | u1.name = u2.password
   no disj u1, u2 : User | u1.name = u2.username
   no disj u1, u2 : User | u1.surname = u2.name
   no disj u1, u2 : User | u1.surname = u2.email
   no disj u1, u2 : User | u1.surname = u2.username
   no disj u1, u2 : User | u1.surname = u2.password
   no disj u1, u2 : User | u1.email = u2.name
   no disj u1, u2 : User | u1.email = u2.surname
   no disj u1, u2 : User | u1.email = u2.username
   no disj u1, u2 : User | u1.email = u2.password
   no disj u1, u2 : User | u1.username = u2.name
   no disj u1, u2 : User | u1.username = u2.surname
   no disj u1, u2 : User | u1.username = u2.email
   no disj u1, u2 : User | u1.username = u2.password
}
```

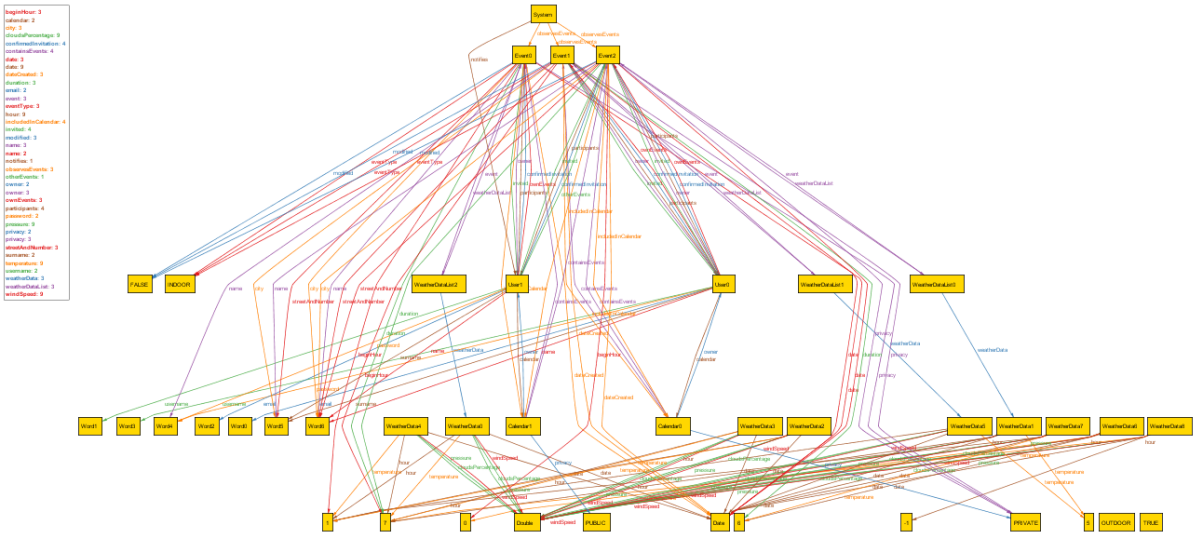
Milan, 9th November, 2014

```
fact userDataUniqueness
{
   all  u : User | u.name != u.surname
   all  u : User | u.name != u.email
   all  u : User | u.name != u.username
   all  u : User | u.name != u.password
   all  u : User | u.surname != u.email
   all  u : User | u.surname != u.username
   all  u : User | u.surname != u.password
   all  u : User | u.email != u.username
   all  u : User | u.email != u.password
   all  u : User | u.username != u.password
}
//relate one calendar to one user
fact userCalendarRelation
{
   all u: User, c: Calendar |  u = c.owner implies u.calendar = c
}
fact numOfCalendarOwners
{
   all c: Calendar | #(c.owner) = 1
}
//relate one created event to a user who is its owner
fact userEventRelation
{
   all u: User, e: Event |  (e.owner  = u) implies (e in e.owner.ownEvents)
   all u: User, e: Event |  (e.owner != u) implies (e not in u.ownEvents)
}
//if user is the owner of an event, that user is automatically invited
fact ownerAutomaticallyInvited
{
      all u:User, e: Event | (e.owner=u) implies (e.owner in e.invited)
}
//if user is not invited to an event, that event must not be contained inside any set of user's invitations
fact notInvitedNoResponse
{
      all  e: Event, u: User | (u not in e.invited) implies (e not in u.confirmedInvitation)
      all  e: Event, u: User | (u not in e.invited) implies (e not in u.declinedInvitation)
      all  e: Event, u: User | (u not in e.invited) implies (e not in u.pendingInvitation)
}
//if owner is invited to his/her event, invitation is automatically confirmed
fact ownerInvitationAutomaticallyConfirmed
{
   all u:User, e: Event | (e.owner=u) implies (e in e.owner.confirmedInvitation)
}
//if user has confirmed the invitation for an event, that event should be contained inside his/her calendar
fact EventCalendarOwnerRelation
{
   all e: Event, u: User | (e in u.confirmedInvitation) implies (e in u.calendar.containsEvents)
   all e: Event, u: User | (e in u.calendar.containsEvents) implies (e in u.confirmedInvitation)
   all e: Event, u: User | (e in u.calendar.containsEvents) implies (u.calendar in e.includedInCalendar)
   all e: Event, u: User | (u.calendar in e.includedInCalendar) implies (e in u.calendar.containsEvents)
}
//if invitation to an event is confirmed, it cannot be declined or pending at the same time
fact disjointOtherInvitationIfConfirmed
{
      all  e: Event, u: User | (e in u.confirmedInvitation) implies (e not in u.declinedInvitation)
      all  e: Event, u: User | (e in u.confirmedInvitation) implies (e not in u.pendingInvitation)
}
//if invitation to an event is declined, it cannot be confirmed or pending at the same time
fact disjointOtherInvitationIfDeclined
{
      all  e: Event, u: User | (e in u.declinedInvitation) implies (e not in u.confirmedInvitation)
```

Milan, 9th November, 2014

```
    all  e: Event, u: User | (e in u.declinedInvitation) implies (e not in u.pendingInvitation)
}
//if invitation to an event is pending, it cannot be confirmed or declined at the same time
fact disjointOtherInvitationIfDeclined
{
    all  e: Event, u: User | (e in u.pendingInvitation) implies (e not in u.confirmedInvitation)
    all  e: Event, u: User | (e in u.pendingInvitation) implies (e not in u.declinedInvitation)
}
//if the user is invited to an event, his/her response must be either confirmed, declined or pending
fact oneInvitationResponseMustBePresent
{
  all  e: Event, u: User | ((u in e.invited) and (e not in u.confirmedInvitation) and (e not in u.declinedInvitation)) implies (e
in u.pendingInvitation)
  all  e: Event, u: User | (e  in u.pendingInvitation) implies ((u in e.invited) and (e not in u.confirmedInvitation) and (e not
in u.declinedInvitation))
  all  e: Event, u: User | ((u in e.invited) and (e not in u.pendingInvitation) and (e not in u.declinedInvitation)) implies (e  in
u.confirmedInvitation)
  all  e: Event, u: User | (e  in u.confirmedInvitation) implies ((u in e.invited) and (e not in u.pendingInvitation) and (e not
in u.declinedInvitation))
  all  e: Event, u: User | ((u in e.invited) and (e not in u.pendingInvitation) and (e not in u.declinedInvitation)) implies (e  in
u.confirmedInvitation)
  all  e: Event, u: User | (e  in u.confirmedInvitation) implies ((u in e.invited) and (e not in u.pendingInvitation) and (e not
in u.declinedInvitation))
}
//if user was invited to an event of which he/she is not owner, and he confirmed invitation, that event goes to otherEvents
set
fact userEventRelation
{
  all u: User, e: Event |  ((e in u.confirmedInvitation) and (e not in u.ownEvents)) implies (e in u.otherEvents)
  all u: User, e: Event |  (e in u.otherEvents) implies ((e in u.confirmedInvitation) and (e not in u.ownEvents))
}
//if the user has confirmed the invitation to an event of another user, the user is participant of that event
fact Participants
{
  all e: Event, u: User | (e in u.confirmedInvitation) implies (u in e.participants)
  all e: Event, u: User | (u in e.participants) implies (e in u.confirmedInvitation)
}
//relate one weatherdatalist to one event
fact eventWeatherListRelation
{
  all e: Event, w: WeatherDataList |  e = w.event implies e.weatherDataList = w
  all e: Event, w: WeatherDataList |  e.weatherDataList = w implies e = w.event
}
fact eventModified
{
    all e: Event, u: User,s:System | ((e.modified = TRUE) and (u in e.participants)) implies (u in s.notifies)
}
pred show()
{
  #User = 2
  #Event = 3
}
run show for 10
```

Milan, 9th November, 2014



Alloy diagram 19: Final model

Note: To see a whole picture in details refer to attached alloymodel.png

3.3. Design constraints

The software product must be designed and implemented in JEE and its related technologies.

3.4. Performance requirements

The software product requires that each web page shall load in less than 15 seconds.

3.5. Standard compliance

The software product must be developed following recommended standards in order to be easily readable and updatable.

3.6. Software system attributes

3.6.1. Reliability

The system shall assure the integrity of the citizens, the governor and the parties.

3.6.2. Availability

The system should be available 24 hours per day, 7 days per week, and 365 days per year.

3.6.3. Security

The software product must encrypt users' password. This can be achieved using cryptographic techniques.

3.6.4. Maintainability

The database of the software product should be backed up periodically, so that in case of failure existing data can be reconstructed easily.

3.6.5. Portability

The software product can be installed on any operating system which supports Java Virtual Machine and its dependent components.

3.7. Other requirements

The software product must provide understandable messages in text form in the event of an error and instruct the user on what to do.

4. Appendices

4.1. Alloy

- Alloy model file: onlinecalendar.als
- Alloy complete diagram: alloymodel.png

Name	**Working hours**
Stolic Nemanja	30 hours
Milos Colic	30 hours

Milan, 9th November, 2014