Politecnico di Milano

Software Engineering 2

MeteoCal

RASD Document

Authors:
Andrea Enrico Turri
Andrea Salmoiraghi
Fabiano Riccardi

16/11/2014

Contents

1 Introduction

1.1 Description of the given problem

We will project and implement MeteoCal, a weather based online calendar for helping people scheduling their personal events avoiding bad weather conditions in case of outdoor activities.
The system that will be developed has to allow the registration of new users and allow them to create, update, delete events, with the possibility to invite other users that can accept or decline the invitation. Events should contain information about when and where the event will take place, whether the event will be indoor or outdoor.
The system has also some other features such as notifications to participants in case of bad weather, import / export of calendars, possibility to make one's calendar or single events public, purpose closest sunny day when registering a new event, avoiding conflicts with existing events.

1.2 Scope

The software-to-be will be developed as a web based application that will let the users create, update, delete their events, with notifications about the weather and inviting other people.

1.3 Glossary: definitions

In this section we define some words that we will use in the documentation of the project.

- **Event**
  An event is a plan in the user's calendar for something that will occur in a certain place during a particular interval of time. An event is characterized by a place, a starting and end time, a flag for outdoor / indoor, a title, a list of participating users.

- **Calendar**
  A calendar is a sort of wall, a notice-board where there are dates and events where the user will participate.

- **Invitation**
  An invitation is a request sent from the user that has organized the event to another user. It can be accepted by the receiver, stay pending until the time of the event.

- **User**
  For user we mean a person who is registered in the system, a user has a calendar, can create events and can use all the functionalities described in the appropriate section of this document.

- **Organizer**
  With reference to an event, the organizer is the user that created it.
- **Invited user**
  With reference to an event, a user that received an invitation to join it.
- **Participant user**
  With reference to an event, an invited user that accepted the invitation.
- **Notification**
  A notification is something, such as an highlighted box in the system, by which notice is given.
- **Email notification**
  A notification sent as an email.
- **Username**
  The identification of a user, used to log into the system and retrieve password.
- **Profile**
  Every user has a profile, an area where are shown user's public events and public calendar.
- **Contact**
  For a user A, a contact is another user B that is in A's address book.
- **Address Book**
  The address book is an area where are listed all the user's contacts.

1.4 Stakeholders

Our "financial" stakeholder is the professor who gave us this didactical project. Our professor needs are to have, within the end of the semester, a product that at least works and we think the main thing that interests the professor is to show that we have understood the development process in all its parts and that we can carry out a project from the beginning to the end passing from all its development phases. So we want to show that we can identify requirements and specifications, design our web application, implement it and then test it, providing all the documents that developers use in developing real software.

The software house X is the company that "virtually" gave us this project to propose a new kind of service. X needs to have a product that works fine and covers all the specification given to us.

We think that our typical users can be:

- occasional users that sometimes can use this tool to plan outdoor events to share with other people
- habitual users that might find this tool useful for example for work purposes.

2 Overall description

2.1 Goals

MeteoCal should provide these main features:

- Registration of a person to the system
- Creation of a new event with the possibility to choose the best place / time w.r.t. the weather conditions
- Browse other users' public calendars and events
- Import and export of calendars
- Making their calendar public / private
- Making their events public / private
- Accept / decline other users' events
- To be notified whenever there will be bad weather conditions
- Update their events
- Delete their events
- Invite other users to join their events

2.2 Domain properties

We suppose that these conditions hold in the analyzed world:

- The weather forecast is supposed to be reliable enough to indicate the right weather in the selected place and time
- The organizer of the event will participate to the event
- If a user accepts an invitation to join an event, it is supposed to participate
- If a user declines an invitation to join an event, it is supposed not to participate

2.3 Assumptions

There are few points that could result not very clear in the specification document, so we will have to assume some facts:

- Every user has only one calendar which is, by default, private. The user can make it public, so that other users can see only available and "busy" time slots, but not the detail of the single events, unless they are public.
- Public events on the user's private calendar are anyway visible.

- If a user doesn't answer to the invitation, this will remain suspended until the end time of the event. After the end time of the event, the user will not be able neither to accept nor to decline.
- A user invited to join a private event can see the details while he doesn't answer to the invitation or if he accepts. If he refuses, he will no longer have access to the event details and he is removed from the list of invited users. If he accepts, the event is added to user's calendar and the user is added to the list of participants and removed from the list of invited users.
- A user that accepted an invitation to an event, can in every moment delete the event from his calendar and so he is removed from the list of participants.
- When the organizer is inviting someone, it is possible to insert the email address of the users to invite which is taken as username by the system.
- Only the organizer of an event can remove a participant or invited user.
- When creating a new event, when the user enters time, date and location, the system checks the weather. If it is possible to get the weather and it is bad, the system proposes to the user an alternative day which is sunny and the closest to the day chosen before.
- *Users that forgot their password, can reset it asking the system to send a new one on their email address.*
- It is possible to export only the calendar of the user, not the ones of others.
- When the organizer updates an event or deletes it, all participant users will be notified. Whenever the event is deleted by the organizer, it will be deleted from the calendar of every participating user.
- The information about a public events include the list of participating users and invited ones.
- The organizer of an event is always a participant.
- A user has associated a profile, which can be searched through a search field. In the profile are present user's basic information and the calendar if it is public: in this way it is possible to see others' calendars.
- *A user can search public events by time and location.*
- When a user imports a calendar, every event in the imported calendar is assumed as a copy, i.e. a new event: there will not be other participants to the events and the system doesn't import the calendar if the there are conflicts with the current calendar of the user.
- As there is the possibility to browse other users' calendars, for us it wasn't so clear if there was a wall in which a user can see calendar of some other random user or only of some "friends", so we decided to define a way to see other users's calendar in the main page and to create an Address Book:

  - *A user can see in the main page the public calendars and public events only of their contacts*
  - *A contact can be added by typing its email address in the Address Book or it is automatically added when a user which is not a contact is invited to join an event*

- *A user can edit his profile information, such as name, picture, email.*

It is important to say that the points in italic are advanced functionalities that we would like to implement to have a more complex system, but, before we want to carry out all the basic set of functionalities in order to have a complete product as requested.
Just to be ready for a possible implementation in the project, we'll carry on these functionalities in all the project's phases. We hope to have enough time to reach this goal.

2.4 Proposed system

We propose a web based platform that will give to registered people the services described below.
Users will be able to have their own calendar, to create events, include weather details in case of outdoor events, receive notifications about bad weather forecasts, browse other's public calendars, import or export calendars, decide to make their events and/or calendar public, accept/decline invitations. Organizers can also update, delete their events and invite other users to join it.
Server will generate different user experiences and so different pages, basing on the actor that is currently using the platform.

2.5 Other considerations about the system

We want to add also other considerations because they are about our thoughts about how MeteoCal should be once developed:

- Easy to use: we will make every effort to create a user interface that allows also new users to be able to use it without using any manual, also the first time. In addition we want that users don't spend too much time to create / update / delete events or join them.

- Nearly stable: we want to guarantee basic functionalities in a stable way, we don't want do have any error or fault in every phase of the processes carried out by the system.

- Have a nice "look and feel": we will concentrate also on this point which we hardly believe that is very important, because, as said by Steve Jobs, *"Design is not just what it looks like and feels like. Design is how it works"*. We will try to make a well designed application, nice to see and that can fit any user, keeping everything as much simple as possible.

3 Actors Identifying

Not registered users: they can visit the home page and register into the system in order to become registered users.

Registered users: they can browse the public calendars of users that decided to make their calendar public, they can create events (they become organizers) or they can be invited to join other's events (they become invited users):

- **Organizers:** their goal is to create events, both private or public, eventually invite other people, update them or remove them, also on the basis of the weather forecasts.
- **Invited:** they can decide to accept or decline an invitation sent by some organizer user. If they accept, they will have the event on their calendar and they can see updates and receive notifications.

4 Requirements

Analyzing goals and assumptions described in the previous chapters, we can derive the requirements, that is what our system will be able to perform.

1. **Registration of a person into the system**
    - The system will provide the registration form and login through username and password.
    - *The system will allow also to reset user's password.*
2. **Events management**
    - The system will allow a user to create an event, suggesting also the closest (to the chosen day) sunny day for the specified location, if it will take place outdoor.
    - The system enables only the organizer user to edit or remove the event. He can also set the visibility to other users, public or private, invite other users to the event and remove invited users in every moment (if they accepted or if they haven't accepted yet).
    - The system will remove the event from the calendar of every participant when the organizer deletes it.
    - *The system will provide also the possibility to search public events by date or place and see their details.*
    - The system will allow the users to join the events for which they received an invitation. The user will also be able to refuse the invitation or to unsubscribe to an already accepted event.
    - Users that accepted invitation and organizer are shown in the event page, in the participants list.
    - Users that still haven't accepted or refused an invitation, are in the list of invited users.
    - The system will apply the following privacy policy for events:

      Private event with user's calendar private The event can be accessed only by participating users, they can see details and participating users.

      Private event with user's calendar public The event can be accessed only by participating users, they can see details and participating users. Users not participating to the event will see the busy time slot in the calendar, but not the details of the event.

      Public event The event can be seen by every user, they can see details and participating users, even when the user's calendar is private.
    - Event details are composed by name, description, location, date, outdoor/indoor, participating and invited users, weather.

- The system will also provide time consistency to the user: by this we mean that the system will avoid time conflicts with existing events, when creating, uploading, accepting events.
- The system will allow invited users to see private event's details, unless they refuse the invitation.

3. **Calendar management**

   - The system will provide to every user one calendar. A calendar can be public or private. In the last case can be seen by other users the periods in which the user is free or busy, event details are hidden, unless the single events are public.
   - The system will allow the users to export their own calendar.
   - The system will allow the users to import a calendar which adds event to the current one, avoiding time overlapping of imported events (in this case the calendar will not be updated with any imported event). Events imported are a "copy", there will not be other participating users and the current user is the organizer of all events.

4. **Users management**

   - *The system will allow the users to search for other users, add a user to the Address Book, remove it, visualize users' profiles and send invitations by entering users' email.*
   - The system will provide a search field to find users by email.
   - *The system will provide a page to edit user's information and upload a picture.*

5. **Notifications**

   - The system will send notifications to users participating to an event if this is updated or deleted by the organizer, or if weather forecasts say that there it will rain in case of outdoor events. These notifications are visualized in the system when the user logs in and they are also sent as an email.
   - The system will be able to check the weather automatically in a specified period of time to send email notifications to users also if they don't log into the system.
   - The system will send email notifications to users invited to join an event.

As we said the the Chapter 2, the requirements written in italic are considered advanced features that will not be implemented unless we will have enough time.

4.1 Functional requirements

After analyzing the features that the system will provide, we can define functional requirements for each actor.

- **Guest**: he can only access the basic functionalities:
  - Sign up
  - Browse homepage
- **Registered user**:
  - Log in
  - Recover password
  - Create an event
  - Search public events
  - Search public calendars (users' profiles)
  - Make the calendar public or private
  - Import a calendar
  - Export the calendar
  - Add/remove users to Address Book
  - Receive notifications
- **Organizer**:
  - Update the event
  - Delete the event
  - Invite other users to the event
  - Make the event public or private
- **Invited user**
  - Accept the invitation
  - Refuse the invitation
  - See event's details

4.2 Nonfunctional requirements

4.2.1 User Interface

The application is thought to be visualizable by the user in every moment, hence we want that it can be used also from smartphones, tablets and any modern browser.

One of our goals is to build a very user friendly interface, minimal and easy to use. We want that the button "Create event" would be always visible, so as a bar where it will be possible to search (events, users) and buttons also to go to settings and logout pages.

Then we will prevent events and calendars that are private from being visualized from other users than the owners. The same for all reserved information.

When the user logs into the system, he will be redirected to his calendar. The page is composed also by a notification area and allows to reach directly the main functionalities of the system, so the usage of the platform can be quick and easier.

The pages will contain forms to be completed for the needed functionalities and also checks to the inputs, in order to always control the flow of the user experience.

Our platform will contain four main pages which we want to focus on in order to understand its usability.

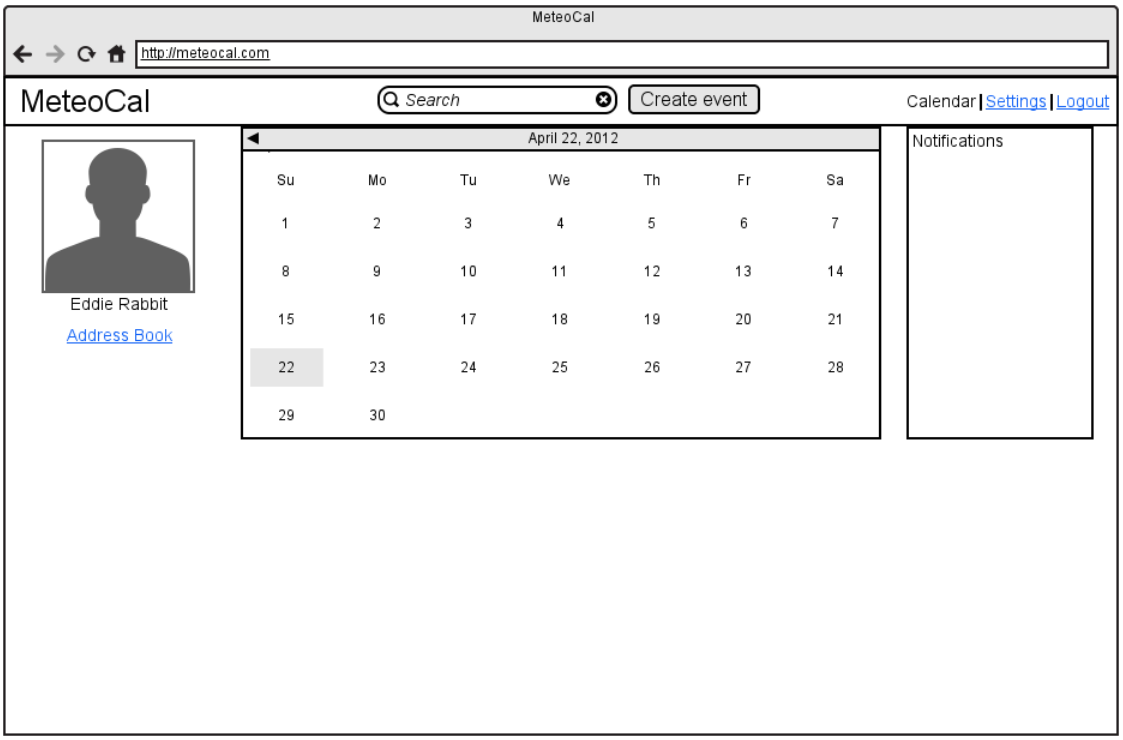


Figure 1: Calendar page

The first page is the calendar page in which current user's information will be displayed, in addition to the notification area. Figure 1 shows a sketch of this page.

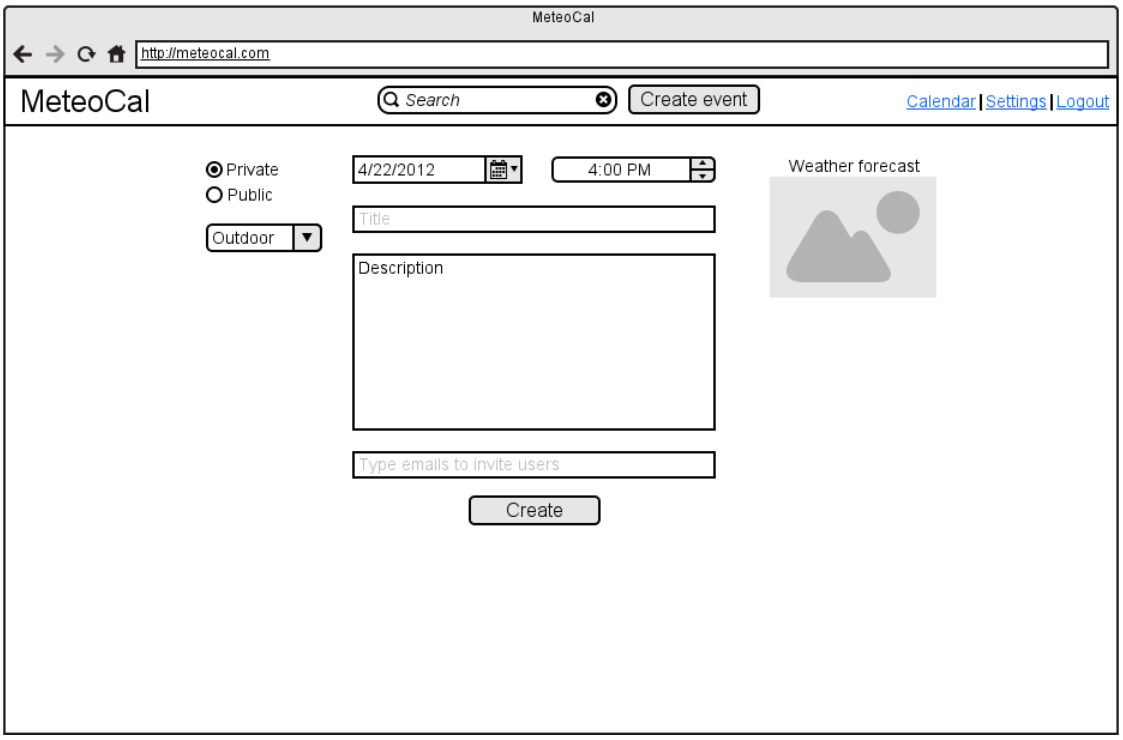


Figure 2: Creation of event

The second page is the one devoted to the creation of an event. It will be displayed when the user clicks on the "Create event" button and it is composed mainly by a form in which are collected all information necessary for the creation of the event. Figure 2 shows a sketch of this page.

Figure 3: Event details

The third page is the event page. This page will be available only to the organizer, the participating users and other users if the event is public: the page will be a little different. The organizer will be able to click on a "Edit event" button to update it or even delete it, he will also have a bar in which he can type the email of other users to be invited, while others will be able only to see information. The sketch is shown in Figure 3. The fourth page is the profile page in which will be displayed public information of a user.

4.2.2 Documentation

With the purpose to explain our project, we wrote the following documentation:

- **RASD**: Requirement Analysis and Specification Document, to understand better and clarify the whole project, useful also for every other stakeholder.
- **DD**: Design Document, to define the real structure of our web application and its tiers.
- **JavaDOC**: documentation to explain the code, useful for software maintenance or system updates.
- **User Manual**: for the users, it will be composed mainly by pictures, but it will also include some text in a tutorial form.
- **Installation manual**: guide for the people that will install the system on new machines, for the developers and the server administrators. The users shouldn't have any problem, since it is a Web Application which is accessible from any modern browser, without requiring any additional software.

4.2.3 Architecture

The system will need one or more dedicated servers, accessible from remote connections and running JavaEE. It will also be necessary a Database to store the users' information.
What is this information will be clarified below, looking at the class diagram, also if we will be more precise in the Design Document.
An internet connection is required to access the application.

5 Scenarios Identifying

Creation of an event	
Participating actors	Steve: organizer user
Flow of events	Steve wants to organize for the day after a gathering of cyclists. It then accesses the calendar MeteoCal and log in. The system shows the page with your calendar and Steve clicks on "Create Event", he enters as location Mavericks, November 15th, 2014 at 3:00 PM, type outdoor, public, title, description. But the system detects that rain is expected, therefore proposes to Steve to change the date with that of 17th November, the closest sunny day. So Steve confirms the date proposed by the system. Then Steve can invite all his friends cyclists enrolled in MeteoCal by entering their email address in the dedicated field. When selected them all, he creates the event which is now added to his calendar. The system then shows the event page with description, title, information about the date, time, location, weather conditions, users who have accepted the invitation and invited users who have not yet responded.

Invitation to events	
Participating actors	Jony: invited user
Flow of events	Jony has been invited to a gathering of cyclists to be held in Mavericks on November 17th, 2014 at 3:00 PM. He sees the invitation reading his email, where access to the system by clicking on "View invitation." Then, the browser is opened and Jony is redirected to the login page, he logs in, then he is redirected to the event page, where he can see the title, description, date, time, location, weather conditions, users who have accepted the invitation and invited guests who have not yet responded. He decided to accept the invitation, then he clicks on "Accept invitation" and the event is added to his calendar. Jony then, while he is on the page of his calendar, he receives a notification for another invitation to a concert of a rock band, then he clicks on "View event" and he is redirected on the event's page. He realizes, however, that the date coincides with the birthday of his girlfriend and that he had already scheduled another appointment, so he clicks on "Reject invitation" and he is redirected to his calendar in which this event has not been added.

Bad weather notification	
Participating actors	Eddie: organizer user
Flow of events	Eddie, some weeks ago, had created an outdoor event for the day of November 5th, 2014, to celebrate the 30th wedding anniversary. The system, however, has found that there will be rain and it then sends an email to all participants with the notice and to Eddie also an email with a button to make a possible change. Then Eddie clicks on the "Edit Event", then the browser is opened with the login page, he logs in and then he is redirected to the page to edit the event. The farm where Eddie organized the event also has an indoor space, so he edits the event as "indoor" and he also changes the description. When he confirms, all participating users receive an email to notify them of the change. Eddie is returned to the event page that has been updated.

Registration and import of calendar	
Participating actors	Brigitte: new user
Flow of events	Brigitte has heard of a new service: MeteoCal. Intrigued, she went on the home page and she decided to sign up and she clicks on the "Register" button. By completing the registration form and confirming, she can now use the system. She is redirected immediately to her empty calendar. She clicks on "Settings" and she sees that from here she can import an existing calendar, so she uploads a file that contains all her appointments that will be loaded into the system.

Time consistency	
Participating actors	Paola: registered user
Flow of events	Paola is inserting in the calendar her events of the week, but at some point the page to create an event a message is shown to warn her that the event can not be saved because it overlaps with the time of the swimming pool. Because what she is entering now is an important medical examination, she decides to cancel the swimming pool, so she goes back to the calendar, she selects the event, she clicks on it and she removes it by clicking on "Edit Event" and then "Delete Event". She will have to confirm her choice. Now she is redirected to the calendar, she clicks "Create Event" and she completes the insertion of the medical examination.

Password recovery	
Participating actors	Brigitte: registered user
Flow of events	Brigitte, after signing up to MeteoCal, after a tiring day, in the evening, she decides to begin entering her events into MeteoCal, so she accesses to the site, but she cannot log in because she forgot her password. So she can obtain a new password by clicking on "Reset Password", she enters her email address and a new password is sent on her email. Brigitte reads the email, she writes the new password somewhere, then she returns to the system and she can finally log in to go back to enter her events.

Calendar visibility	
Participating actors	Ann: registered user
Flow of events	Ann works for a company that organizes outdoor events, then uses MeteoCal to manage the timing between members of society, taking advantage of the opportunity to be notified in advance for any adverse weather conditions. She also wants to make public the calendar, so that other MeteoCal users can see when the company is already involved in other events and when it is free, so it is possible to decide whether to entrust to Ann's company the task of organizing the event. In order to to this,Ann goes to MeteoCal, she logs in and she is redirected to her calendar. Then she clicks on "Settings" link, which redirects to the settings page. Here she clicks on "Edit visibility Calendar" and then she chooses "Public". She confirms, then she is redirected to the settings page and now her schedule is visible to other users.

Search calendars	
Participating actors	Mark: registered user
Flow of events	Mark wants to know when Ann, another member of MeteoCal that the public calendar, will be free for a meeting. So Mark looks for Ann's calendar on MeteoCal: he logs in, he is redirected to his calendar and he enters Ann's email in the search bar. So he clicks on the result that he is displayed, and he is redirected to Ann's profile. Now he can see when Ann is free and he decides when they could see each other.

Address book	
Participating actors	Brigitte: registered user
Flow of events	Brigitte often creates outdoor parties with friends, so she always uses MeteoCal to create events and to inquire about the weather. To accelerate the process to invite other users, Brigitte logs into the system, she clicks on "Address Book" link. Here there is the list of her friends, she types in the text field the email addresses of her friends (the ones registered on MeteoCal) and so they are saved in the system. Now she goes back to her calendar by clicking on the logo "MeteoCal", she clicks on "Create Event" and she enters all the data. Below, there is already a list of contacts that may be invited by clicking on them, without typing their email address.

Removal of event where the user was invited	
Participating actors	Steve: registered user
Flow of events	Steve, some days ago, accepted an invitation to an event for a party next Saturday night. Unfortunately, he found out that that day he will have to work for an urgent project and he won't be able to go there. So he opens MeteoCal, he logs in, he searches the event in his calendar, he clicks on it. So Steve is redirected to the page of the event, he clicks "Remove event" and he clicks "Ok" in the confirmation message. So he is redirected to his calendar in which has been removed this event and so other participants, while visualizing event's details, can see that Steve won't be at the party.

6 UML Modeling

6.1 Use case

We can derive some use cases from the scenarios identified in previous paragraph:

- Registration
- Log in
- Recover password
- Creation of event, invitation
- Update of event
- Search public events
- Accept/Refuse an invitation to an event
- Make calendar public or private
- Import and export of calendar
- Search for users (calendars)
- Add/Remove contact to address book

We decided to describe small Use Cases diagrams instead of a unique and big one. In this way we are trying 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):

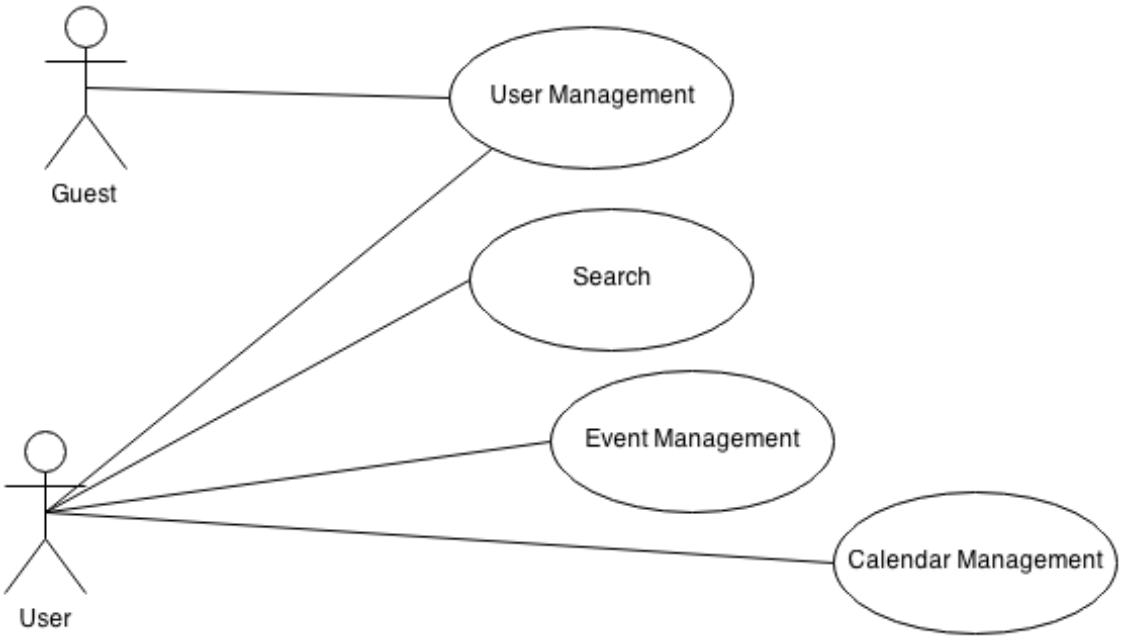


Figure 4: Macro use cases

In the following subparagraphs we will 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", "links", "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.

6.1.1 User management

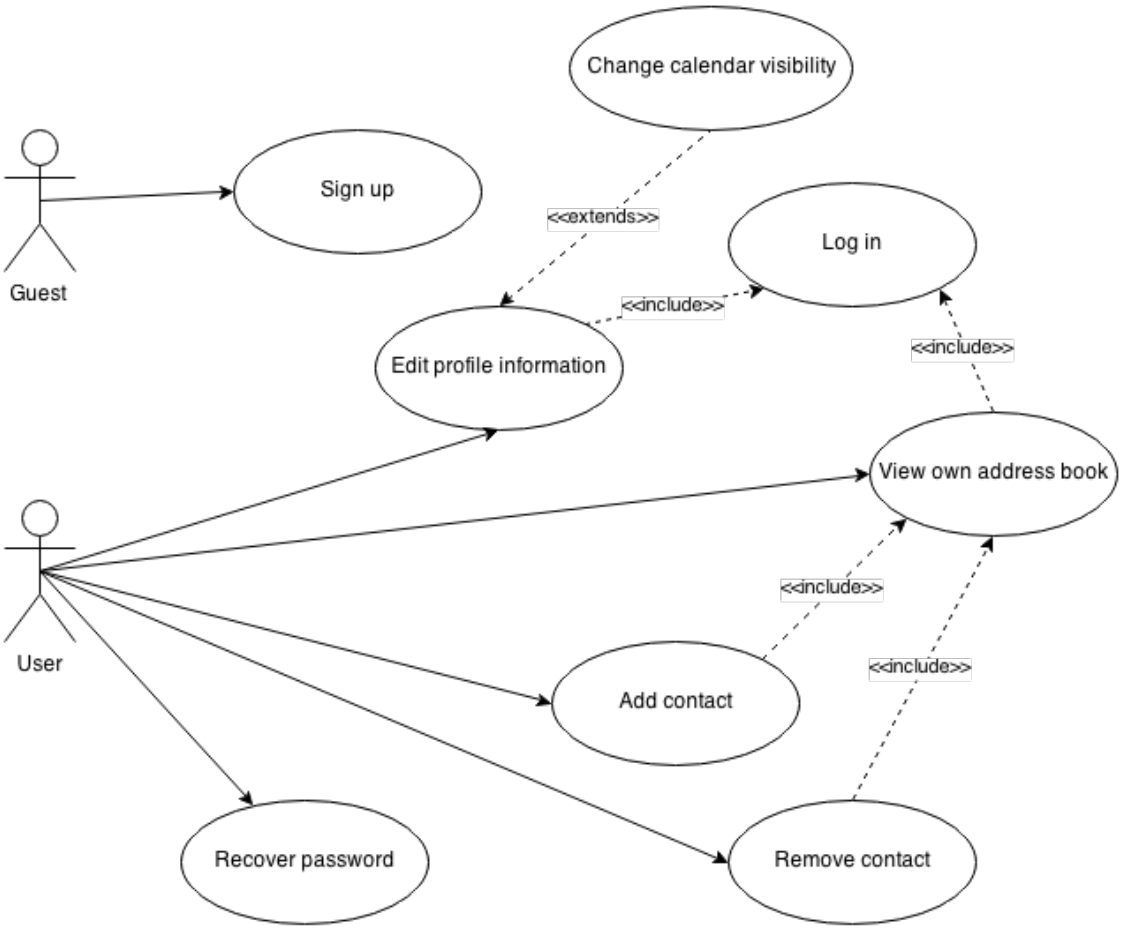


Figure 5: User management use cases

Name	**Registration**
Actors	Unregistered user
Entry conditions	None.
Flow of events	• The user reaches the registration page, so it is displayed the registration form; • The user types his data in the registration form in order to fill it; • The user clicks on the "Register" button; • The system shows a message to inform the new user that has been successfully registered and redirects to the login page.
Exit conditions	The user has successfully registered to the system.
Exceptions	The information inserted in the form is missing, is inconsistent or is wrong: it is displayed an error message and it is still shown the form.

Name	**Log In**
Actors	Registered user
Entry conditions	The user has successfully registered to the system.
Flow of events	• The user reaches the login page, so it is displayed the login form; • The user types his email address and password in the login form; • The user clicks on the "Login" button; • The system redirects the user to the main page, with user's calendar.
Exit conditions	The user has accessed in his private area.
Exceptions	The information inserted in the form is not correct: it is displayed an error message and it is still shown the form.

Name	**Password recovery**
Actors	Registered user
Entry conditions	The user has successfully registered to the system.
Flow of events	• The user reaches the login page, so it is displayed the login form; • The user clicks on the "Password reset" link and he is redirected to the password reset page; • The user types his email address in the input form; • The user clicks on the "Reset" button; • The system sends an email to the user with the new password and shows a message to inform the user about this; • The system redirects the user to the login page.
Exit conditions	The user has received an email with a new password.
Exceptions	The email address inserted in the form doesn't match any user: it is displayed an error message and it is still shown the form.

Name	**Add user to Address Book**
Actors	Registered user
Entry conditions	The user has logged in the system, he is on the main page with his calendar.
Flow of events	• The user clicks on the "Address Book" link; • The user clicks is redirected to the address book page; • The user types the email address of the user he wants to add in the input form; • The user clicks on the "Add" button; • The system reloads the page;
Exit conditions	The user has added a user to his address book.
Exceptions	The email address inserted in the form doesn't match any user: it is displayed an error message.

6.1.2 Event management

Figure 6: Event management use cases

Name	**Creation of an event**
Actors	Registered user
Entry conditions	The user has logged in the system, he is on the main page with his calendar.
Flow of events	• The user clicks on "Create event" button and he is redirected to the page with the input form to create an event; • The user fills the form with the event information (title, description, visibility...); • If the event is "Outdoor", the system checks the weather for the time and location specified: if there the forecast says that there will be bad weather, the system proposes the closest sunny day; • If the user wants to invite other users, he can type their email in the field and press "Add" or he can just click on the contacts he wants to invite; • The user clicks on "Create event" to confirm and he is redirected to the page of the event;
Exit conditions	The new event is added to user's calendar and he is the organizer.
Exceptions	The information inserted in the form is missing, is inconsistent, is wrong or if there is a time conflict with another event: it is displayed an error message and the system remains on the page with the form.

Name	**Acceptance of invitation**
Actors	Registered user
Entry conditions	The user is registered and has received an invitation to join an event and he hasn't accepted yet.
Flow of events	• The receives an email with the invitation and a link to the see event details; • The user clicks on the "View invitation" link and the browser opens and shows the login page; • The user logs in the system; • The user is redirected to the page of the event, with a button to accept and a button to refuse; • The user clicks on the "Accept invitation" button; • The system adds the current user to the list of participating users;
Exit conditions	The user is added to the list of participating users of the event, the event is added to user's calendar and the user can view event's details.
Exceptions	None

Name	**Update an event**
Actors	Registered user
Entry conditions	The user has logged in the system, he is on the main page with his calendar and he is the organizer of the event he is going to edit.
Flow of events	• The browses on his calendar and clicks on the event to update; • The user is redirected to the page with the details of the event; • The user clicks on the "Edit Event" link and he is redirected to the page with the input form already filled with current data of the event; • The user modifies the data in the input form; • The user confirms the update by clicking "Save" button; • The user is redirected to the page with event details;
Exit conditions	The event is updated with the data inserted by the user.
Exceptions	The information inserted in the form is missing, is inconsistent, is wrong or the new time overlaps with another event: it is displayed an error message and the system remains in the page with the form.

6.1.3 Search

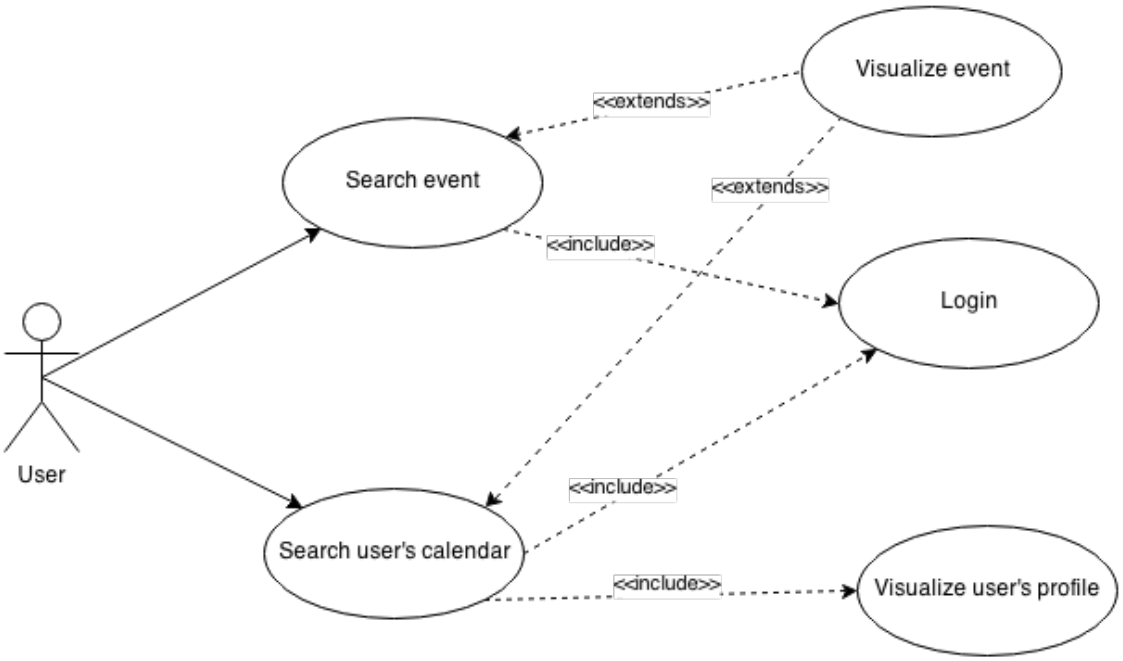


Figure 7: Search use cases

Name	Search for a user's calendar
Actors	Registered user
Entry conditions	The user has logged in the system, he is on the main page with his calendar.
Flow of events	• The user types the email address of another user in the search bar and it is displayed a result; • The user clicks on the result and he is redirected to the profile page of the researched user; • The user can see the public calendar of the researched user;
Exit conditions	The user can see the calendar of another user.
Exceptions	If the email address typed doesn't match a user, there are no results displayed. If the calendar of the researched user is private, it will be hidden.

Name	Search event
Actors	Registered user
Entry conditions	The user has logged in the system, he is on the main page with his calendar.
Flow of events	• The user types the title, location or date/time in the search bar and the system displays some events; • The user clicks on an event (public) and he is redirected to the page of the researched event; • The user can see the page of the event (public) with all its information;
Exit conditions	The user visualizes the page of the searched event
Exceptions	If the data typed doesn't match any event, there are no results displayed.

6.1.4 Calendar management

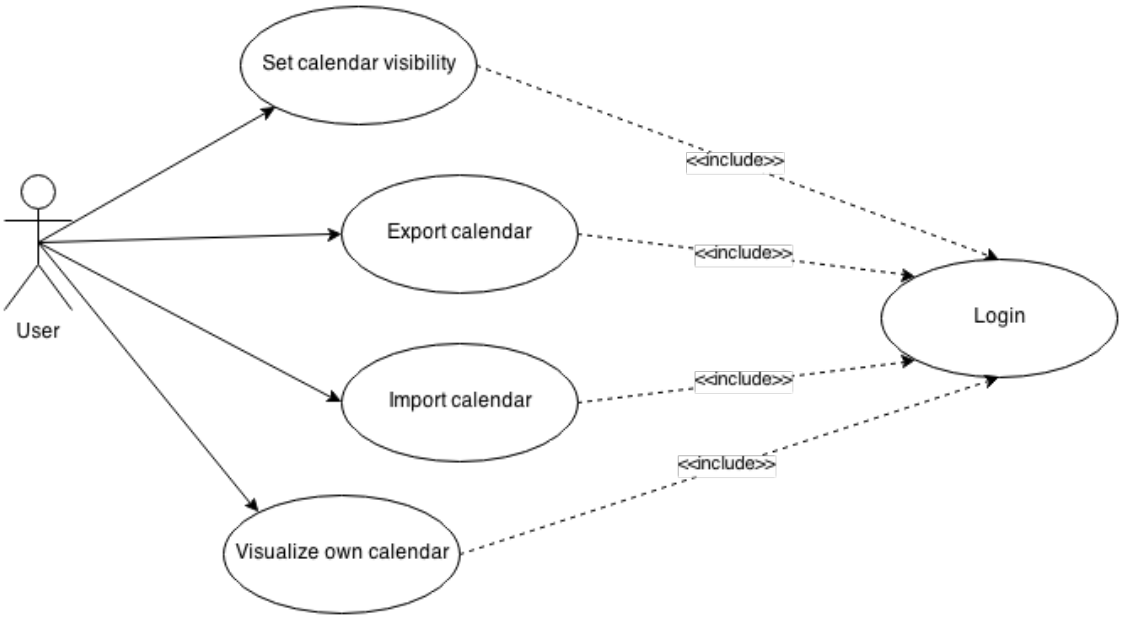


Figure 8: Calendar management use cases

Name	Making calendar public (private)
Actors	Registered user
Entry conditions	The user has logged in the system, he is on the main page with his calendar which is private.
Flow of events	• The user clicks on the "Settings" link and the browser opens and shows the settings page; • The user clicks on the "Edit visibility Calendar" link and he is redirected to the page that shows the current visibility status and the form to change it; • The user selects "Public" ("Private") and clicks on the "Save" button; • The user clicks is redirected to the settings page;
Exit conditions	User's calendar is now public.
Exceptions	None

Name	Export calendar
Actors	Registered user
Entry conditions	The user has logged in the system, he is on the main page with his calendar which is private.
Flow of events	• The user clicks on the "Settings" link and the browser opens and shows the settings page; • The user clicks on the "Export calendar" buttons; • The download of the file starts;
Exit conditions	The file with the calendar is saved on the user's client.
Exceptions	None

Name	**Import calendar**
Actors	Registered user
Entry conditions	The user has logged in the system, he is on the main page with his calendar which is private.
Flow of events	• The user clicks on the "Settings" link and the browser opens and shows the settings page; • The user clicks on the "Import calendar" buttons and he is asked to select a file; • The user selects the file to import and confirms; • The user confirms to upload the file by clicking on "Upload file"; • The user is redirected to his calendar, now with all events imported;
Exit conditions	The user's calendar is filled with events imported.
Exceptions	If the file is not valid or if there are time conflicts with other events a message is shown and no event is added to calendar.

6.2 Class diagram

Now that we have refined our use cases we can draw a class diagram of our system:

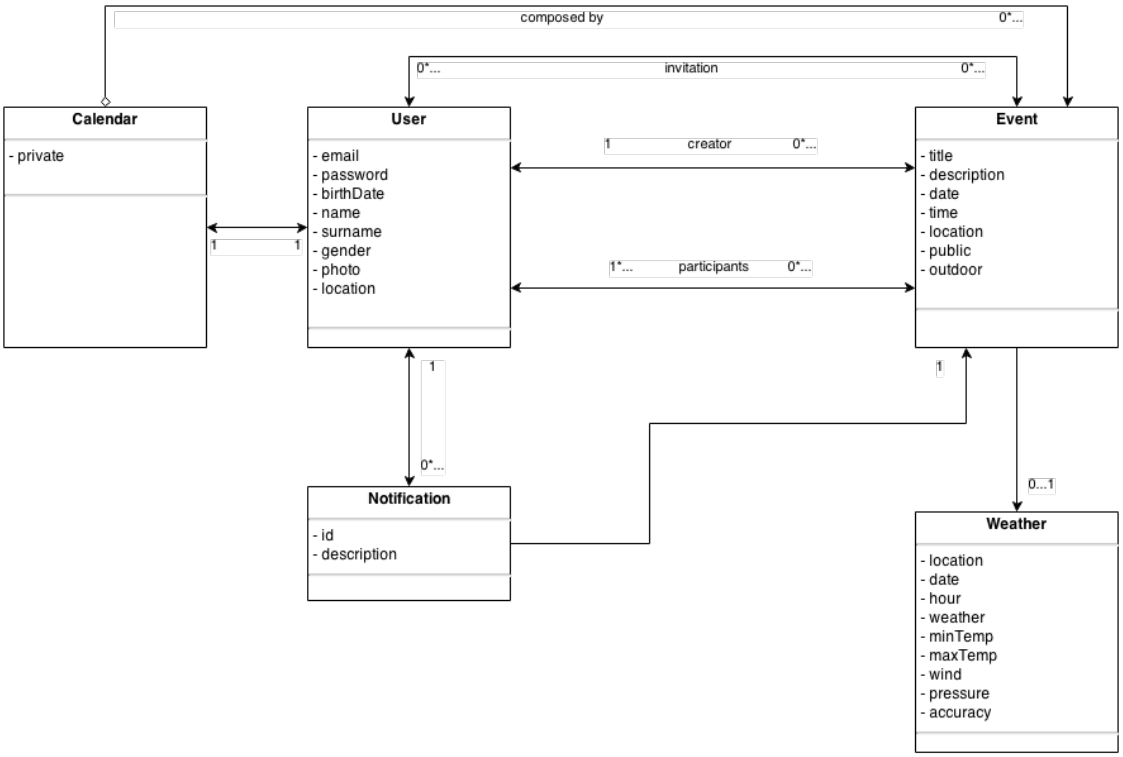
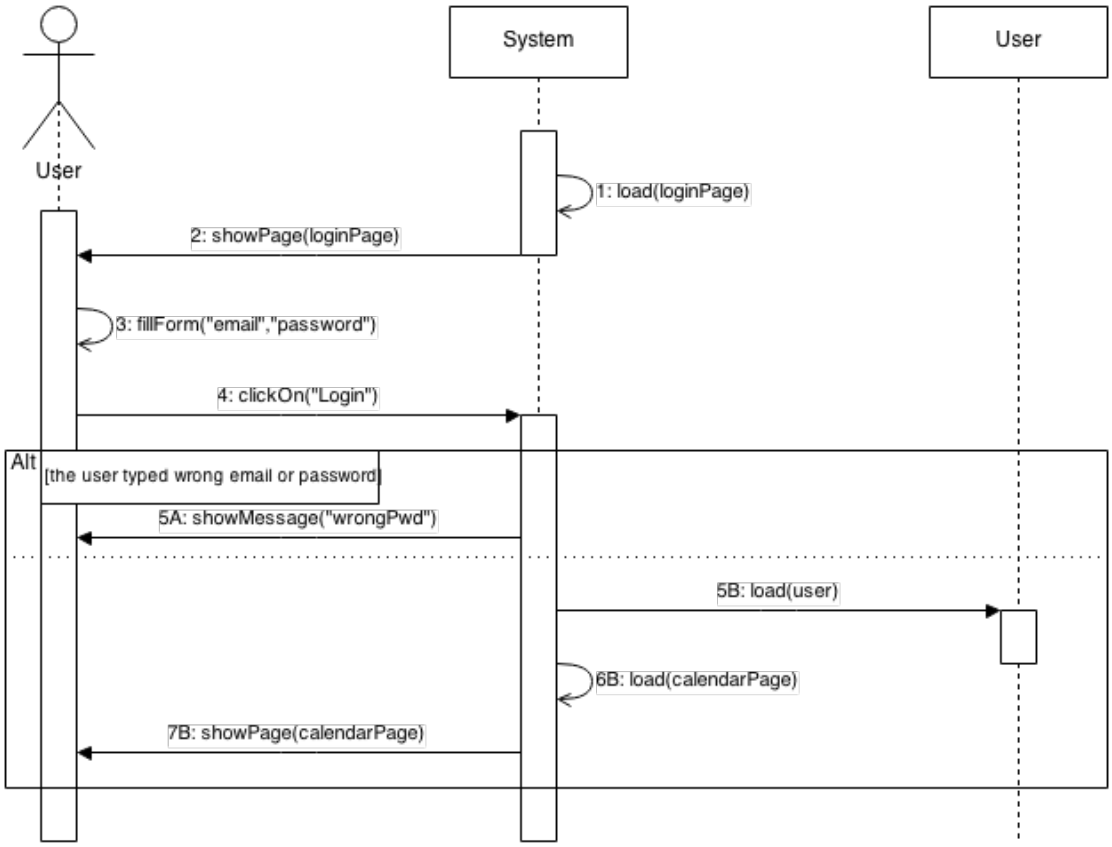


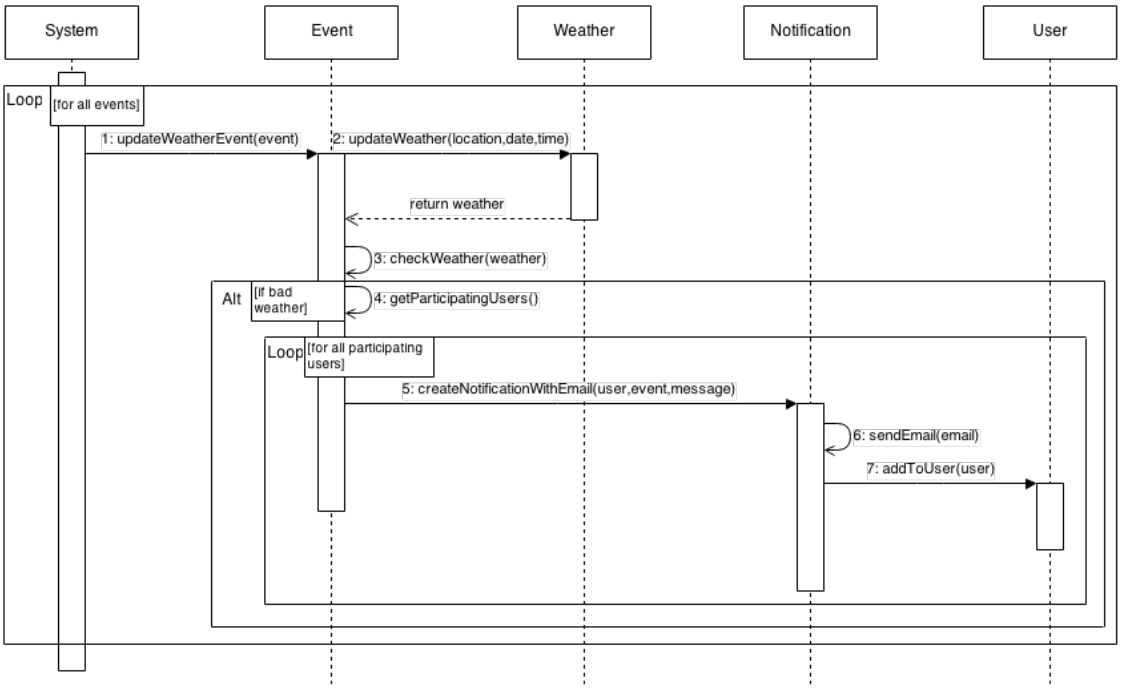
Figure 9: Class diagram

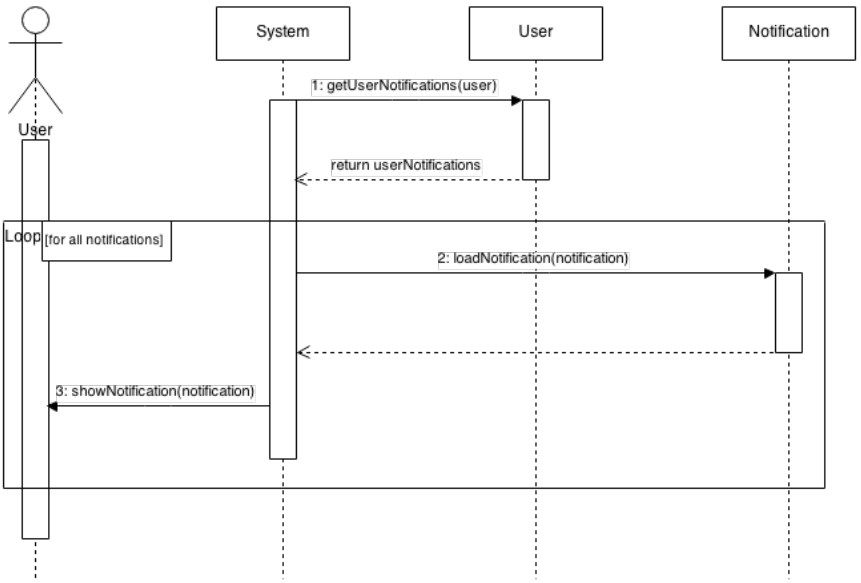
6.3 Sequence diagrams

6.3.1 Login

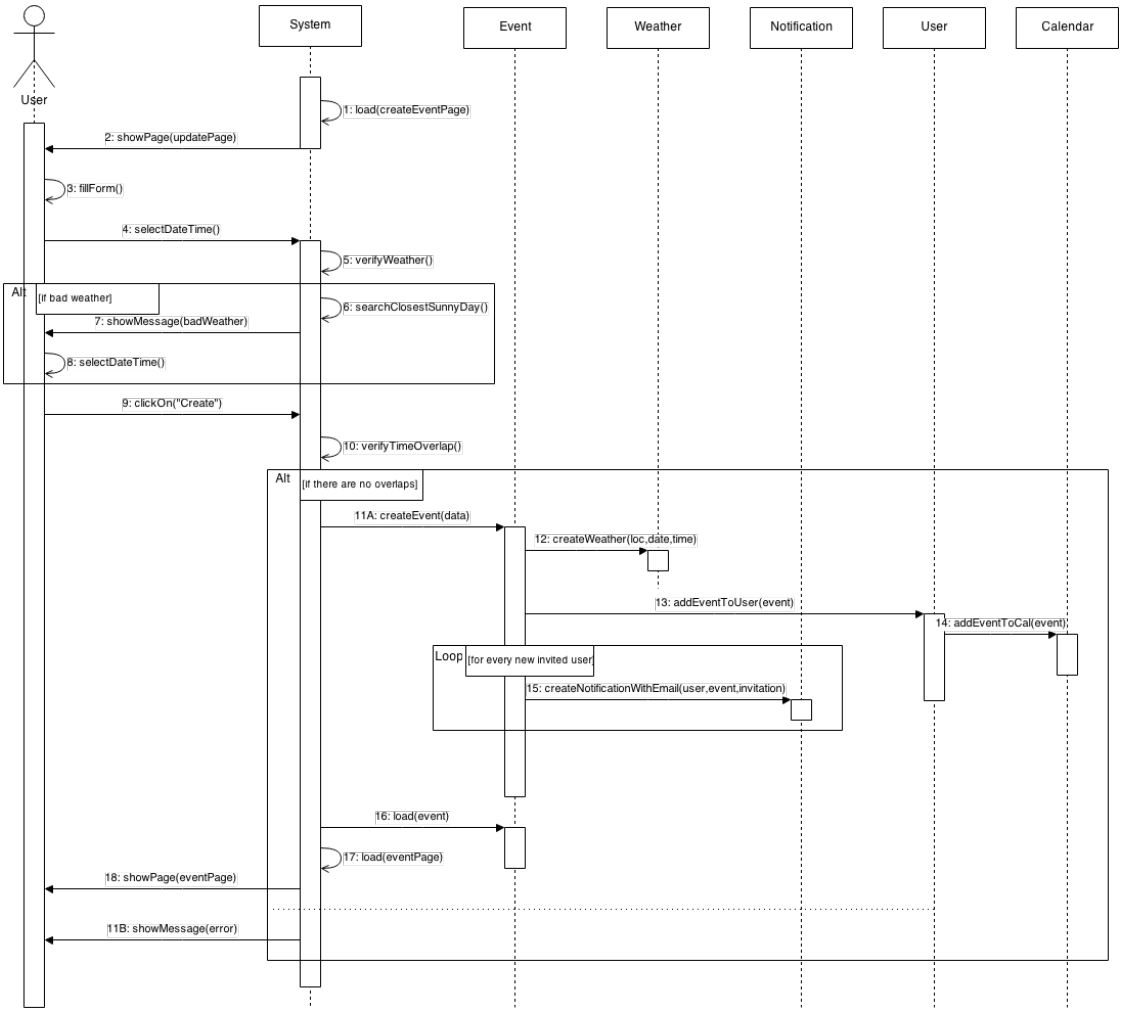


6.3.2 Generation of a notification and visualization

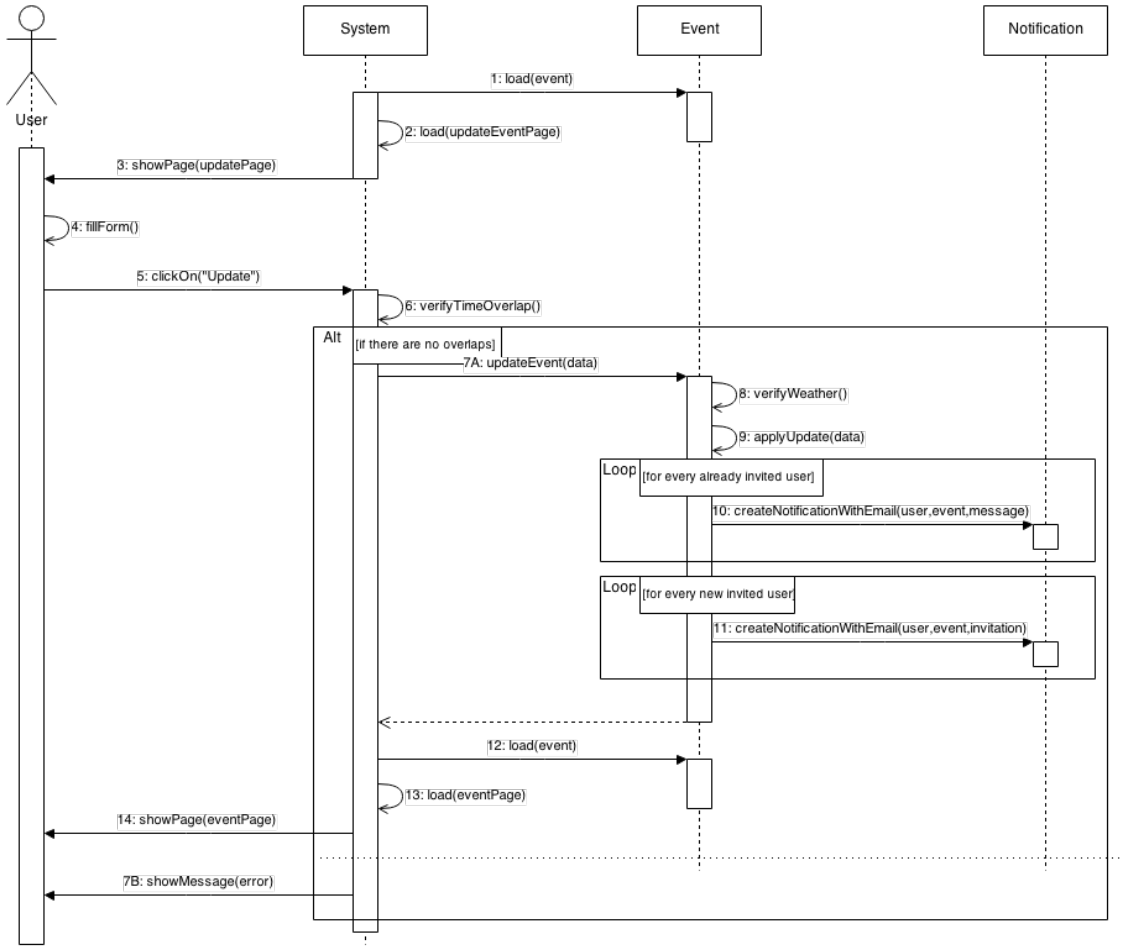




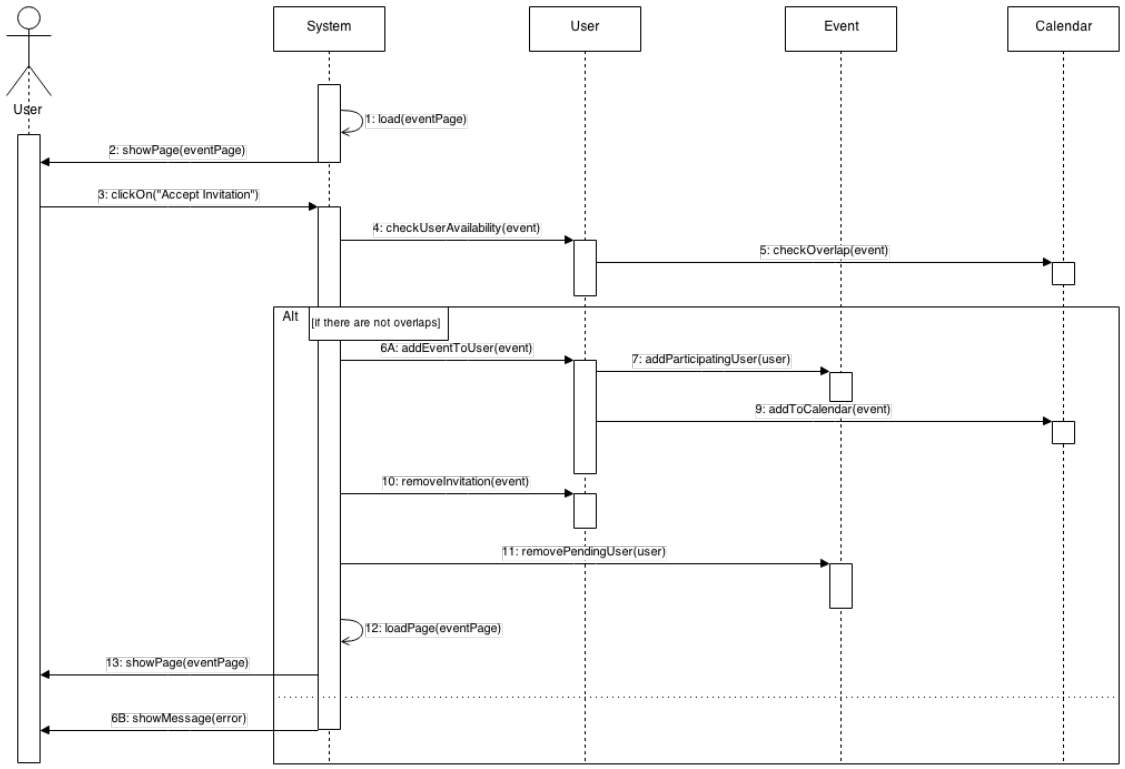
6.3.3 Creation of an event



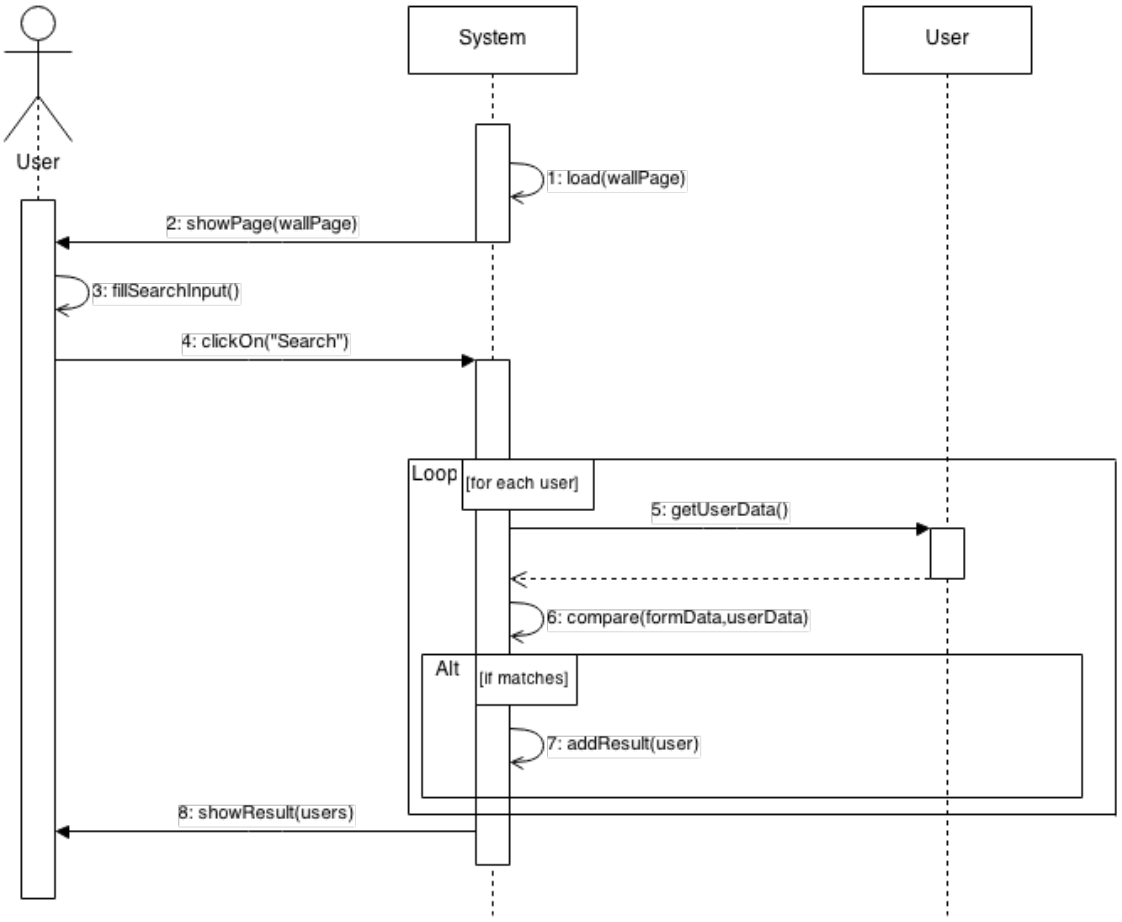
6.3.4 Update of an event



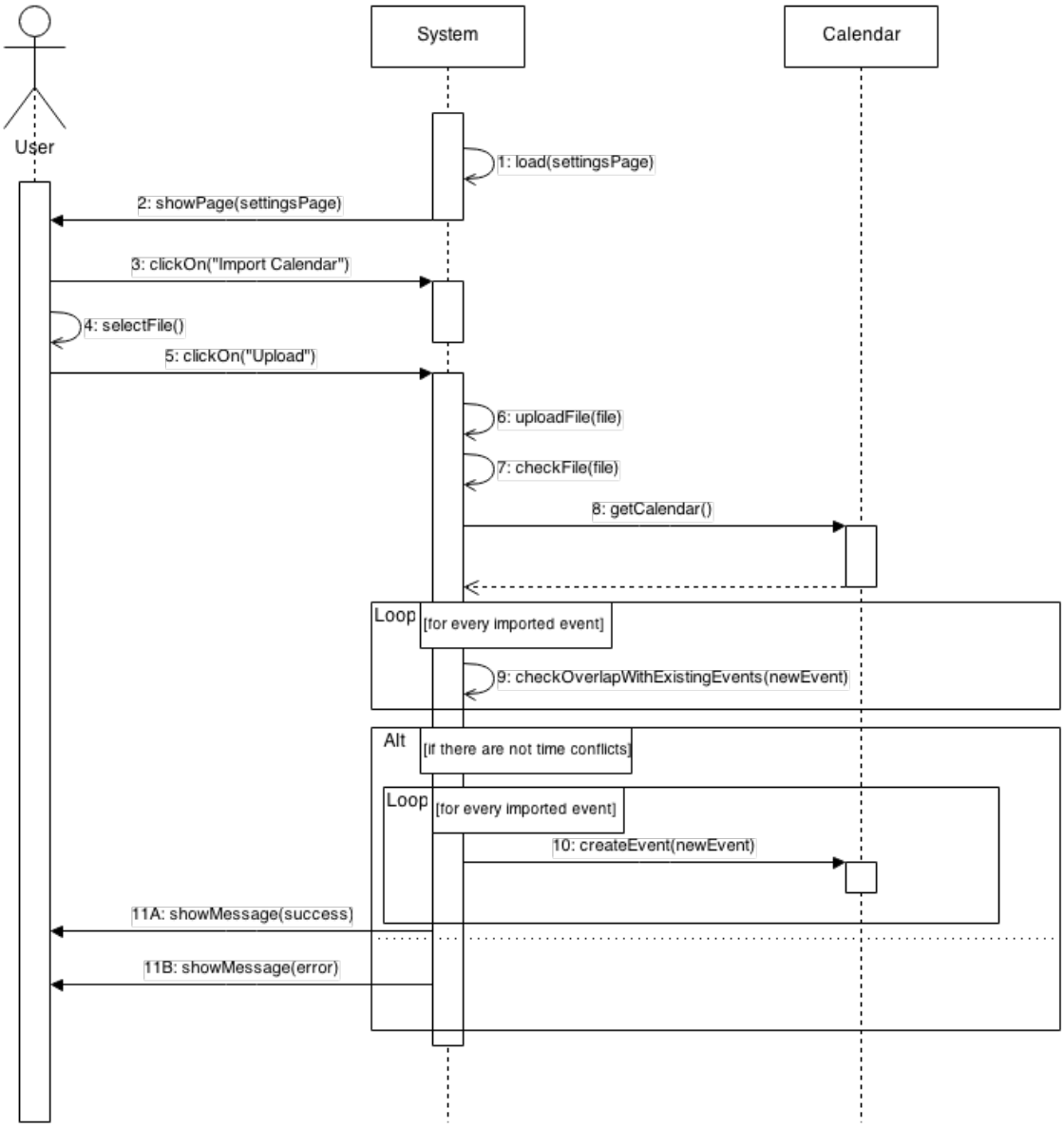
6.3.5 Acceptance of invitation



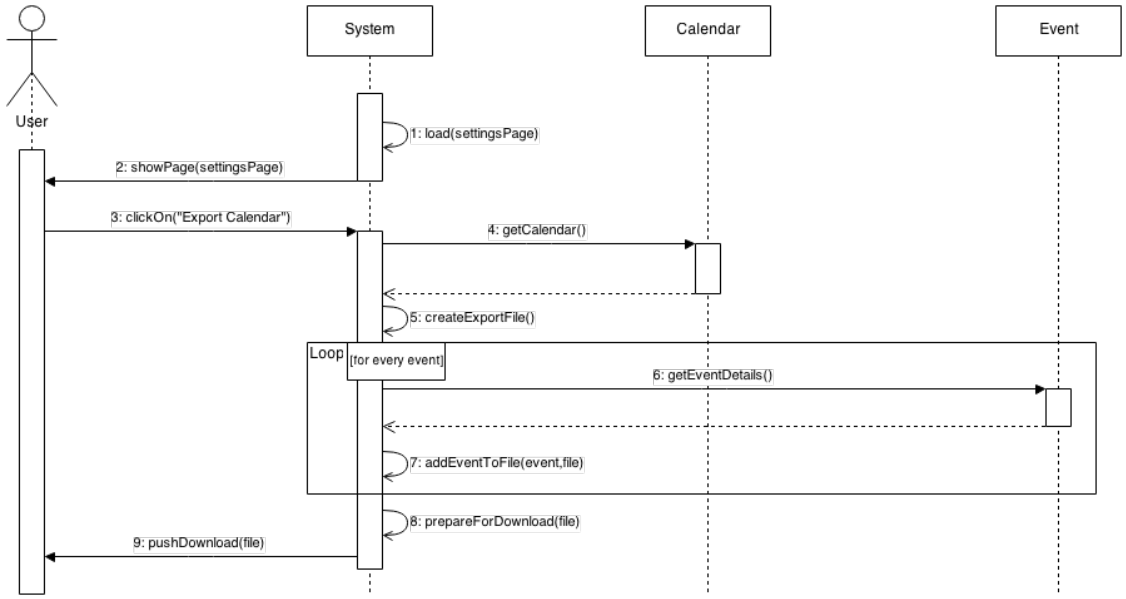
6.3.6 Search user profile (calendar)



6.3.7 Import calendar



6.3.8 Export calendar

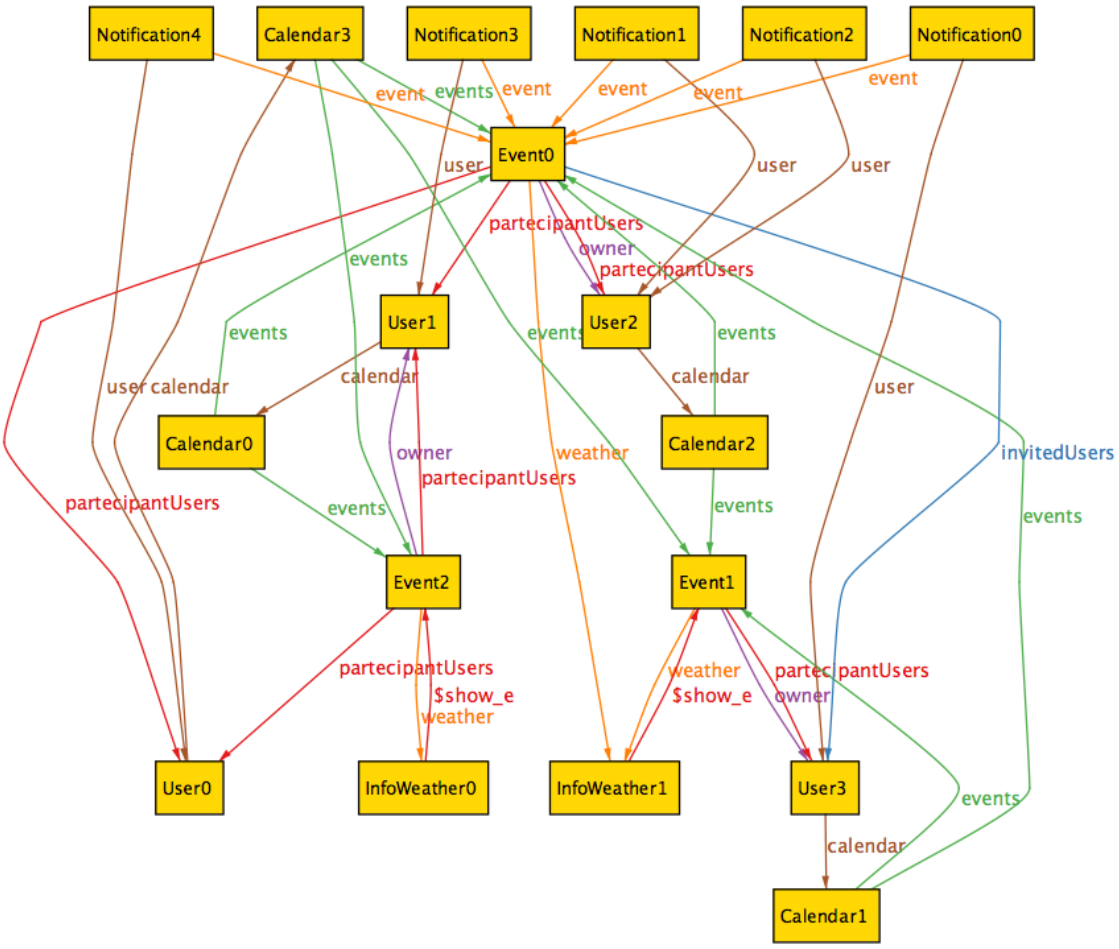


7 Alloy Modeling

In this paragraph we try to understand if our Class Diagram can be consistent using Alloy Analyzer. We report in the next subsections the code used and some World generated by our predicates just to let understand that our model is consistent.

7.1 Generated World

In this paragraph we report some worlds generated by Alloy Analyzer in order to make understand that our model is consistent.
We report below a general world generated. Further specific representations are reported below in order to better understand some characteristics of the model.



7.1.1 Code

```
/************************************ MODEL ************************************/

sig User {
    calendar: one Calendar
}

sig Calendar {
```

```
    events: set Event
}{
    // A calendar must have an user
    one u: User | this = u.calendar
}

sig Event {
    owner: one User,
    invitedUsers: set User,
    partecipantUsers: set User,
    weather: lone InfoWeather
}{
    // The owner is also a partecipant
    owner in partecipantUsers
    // An invited user can not be a partecipating user
    #(invitedUsers & partecipantUsers) = 0
    // The invited users must receive an invite notification for the event
    all u: User | u in invitedUsers implies (Notification.user = User and Notification.event =
this)
    // All events must be in the calendars of partecipant users
    all u: User | u in partecipantUsers iff this in u.calendar.events
}

sig InfoWeather {
}{
    // The weather info must be linked to at least a user
    some e: Event | e.weather = this
}

sig Notification {
    event: one Event,
    user: one User
}{
    // A notification for an event can only be sent to an user linked to that event
    user in (event.partecipantUsers+event.invitedUsers)
}


/****************************** ASSERTIONS ****************************/

assert oneCalendarOneUser {
    all c: Calendar | one u: User | u.calendar = c
}
//check oneCalendarOneUser

assert ownerInPartecipantUsers{
    no e: Event | e.owner not in e.partecipantUsers
}
//check ownerInPartecipantUsers

assert invitedUserNotInPartecipantUsers{
    all u: User | all e: Event | u in e.invitedUsers implies u not in e.partecipantUsers
}
//check invitedUserNotInPartecipantUsers

assert invitedUsersHaveOneNotification{
    no u: User | u in Event.invitedUsers and u not in Notification.user and Notification.event =
Event
}
//check invitedUsersHaveOneNotification

assert eventInPartecipantUsersCalendar {
    all e: Event | all u: User | u in e.partecipantUsers implies e in u.calendar.events
}
//check eventInPartecipantUsersCalendar

assert infoWeatherLinkedToAtLeastOneEvent {
    no i: InfoWeather | i not in Event.weather
}
//check infoWeatherLinkedToAtLeastOneEvent
```

```
assert eventNotificationOnlyToEventUsers {
   no n: Notification | n.user not in (n.event.partecipantUsers+n.event.invitedUsers)
}
//check eventNotificationOnlyToEventUsers


/****************************** SHOW ****************************/

pred show() {
}

run show for 4 but exactly 4 User, exactly 3 Event, 2 InfoWeather, exactly 5 Notification
```

7.1.2 Result of Analyzer

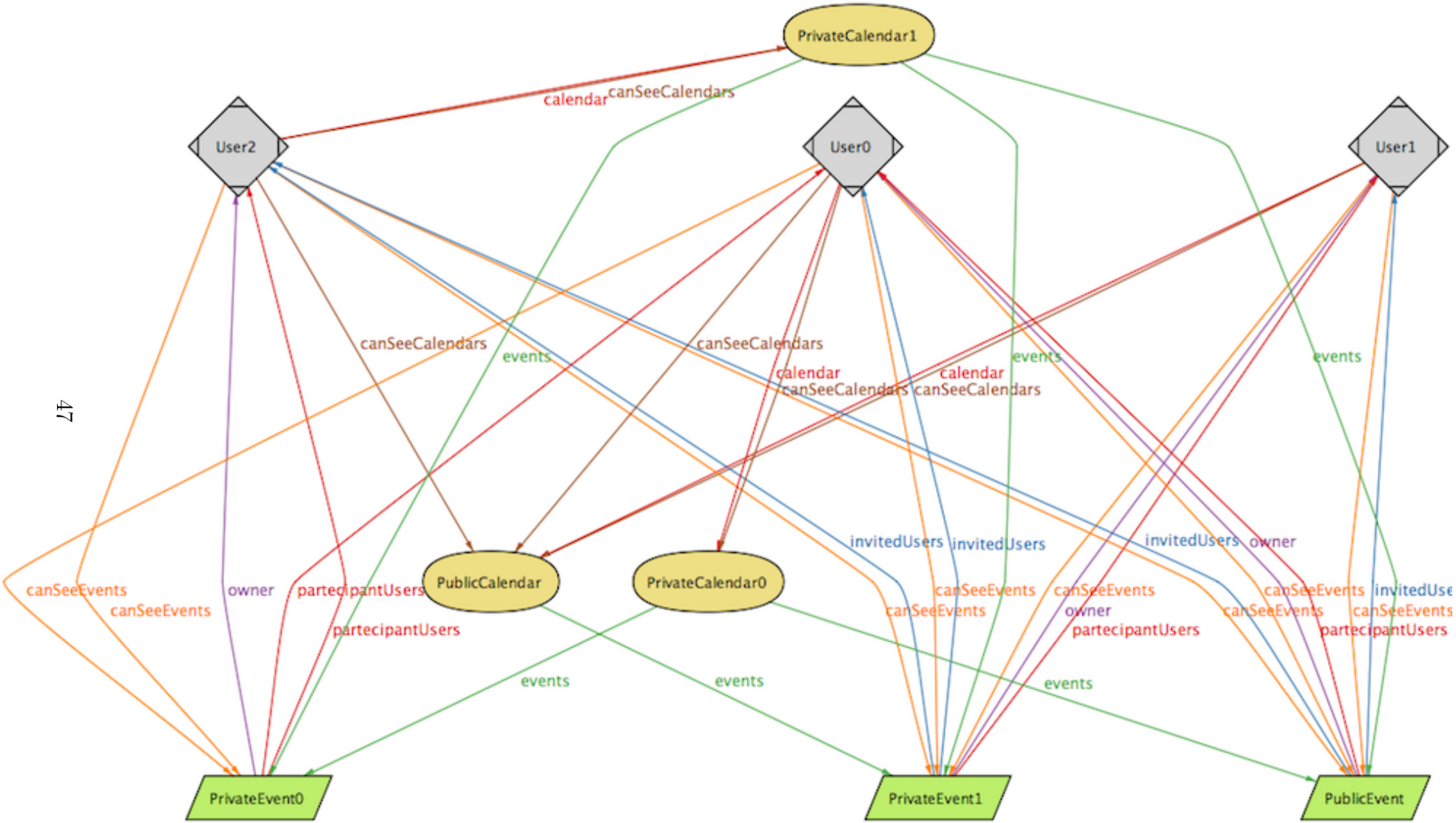
Here an example of result given by the Analyzer:

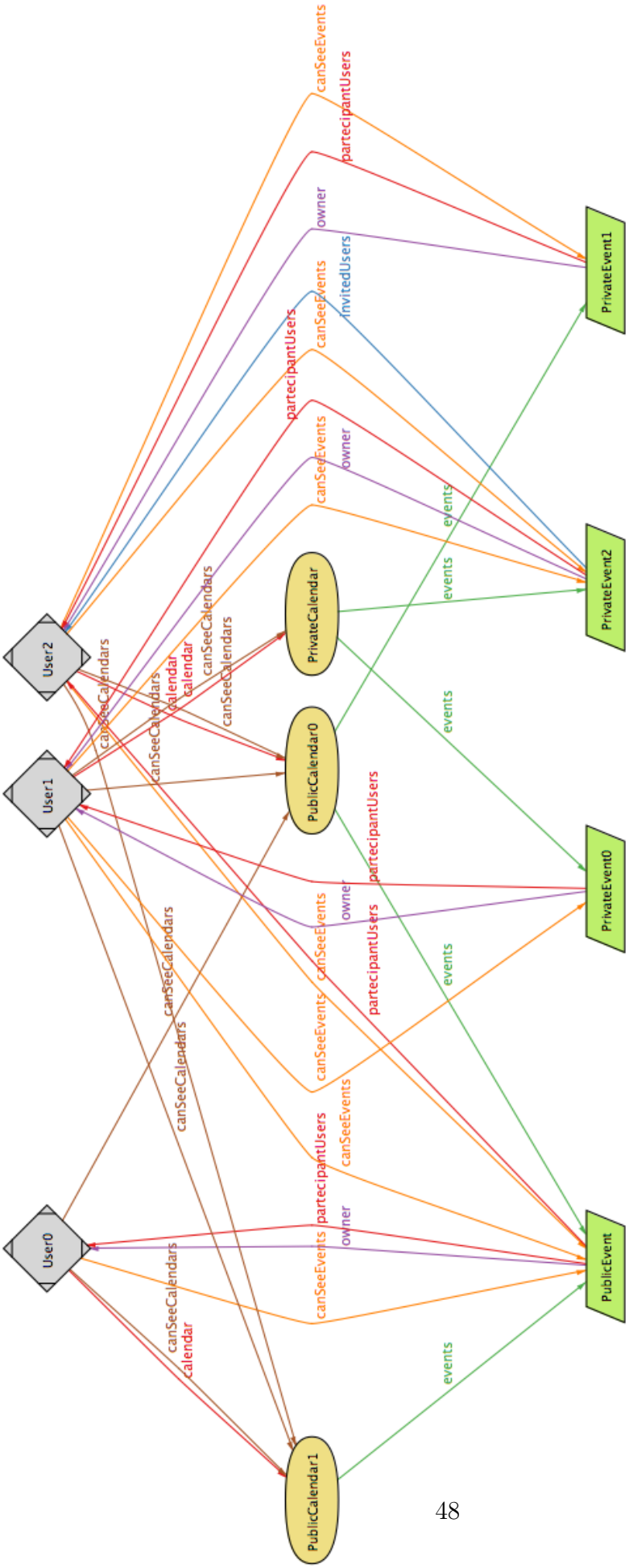
```
8 commands were executed. The results are:
  #1: No counterexample found. oneCalendarOneUser may be valid.
  #2: No counterexample found. ownerInPartecipantUsers may be valid.
  #3: No counterexample found. invitedUserNotInPartecipantUsers may be valid.
  #4: No counterexample found. invitedUsersHaveOneNotification may be valid.
  #5: No counterexample found. eventInPartecipantUsersCalendar may be valid.
  #6: No counterexample found. infoWeatherLinkedToAtLeastOneEvent may be valid.
  #7: No counterexample found. eventNotificationOnlyToEventUsers may be valid.
  #8: Instance found. show is consistent.
```

7.2 Visibility

In this paragraph we are showing how we model the behavior of the calendar with respect to the visibility (of entire calendar and of events), because we think it is an important aspect of what we are going to develop.
Here we can see, for example, that:

- A user can have one calendar and the calendar belongs only to that user;
- If the user is invited to join an event, the he received the notification;
- A user can see public calendars;
- A user can not see other private calendars;
- A user can see the details of all his events;
- A user can see the details of public events;
- A user that organized an event, is also a participant;
- All events must be in the calendars of partecipant users.


PrivateCalendar1
User2
User0
User1
calendar
canSeeCalendars
canSeeCalendars
events
canSeeCalendars
calendar
canSeeCalendars
calendar
canSeeCalendars
events
events
invitedUsers
invitedUsers
invitedUsers
owner
PublicCalendar
PrivateCalendar0
canSeeEvents
canSeeEvents
owner
partecipantUsers
partecipantUsers
canSeeEvents
canSeeEvents
canSeeEvents
owner
partecipantUsers
canSeeEvents
canSeeEvents
partecipantUsers
events
events
events
PrivateEvent0
PrivateEvent1
PublicEvent


User0
User1
User2
PublicCalendar0
PublicCalendar1
PrivateCalendar
PublicEvent
PrivateEvent0
PrivateEvent1
PrivateEvent2
canSeeCalendars
calendar
events
canSeeEvents
owner
partecipantUsers
invitedUsers

7.2.1 Code

```
/********************************** MODEL **********************************/

sig User {
   calendar: one Calendar,
   // The calendars that the user can ideally see
   canSeeCalendars: set Calendar,
   // The events that the user can ideally see
   canSeeEvents: set Event
}{
   // The user can always see his calendar
   calendar in canSeeCalendars
   // The user can see public calendars also
   all c: PublicCalendar | c in canSeeCalendars
   // The user can not see other private calendars
   all c: PrivateCalendar | c ≠ calendar implies c not in canSeeCalendars
   // The user can see all his events
   calendar.events in canSeeEvents
   // The user can see all public events of public calendar
   // The user can see also public events of private calendar (if he can reach them in someway to de
   all e: PublicEvent | e in canSeeEvents
   // The user can also see private events in which he is invited
   all e: PrivateEvent | this in (e.partecipantUsers + e.invitedUsers) iff e in canSeeEvents
}

abstract sig Calendar {
   events: set Event
}{
   // A calendar must have an user
   one u: User | this = u.calendar
}

sig PrivateCalendar extends Calendar{
}

sig PublicCalendar extends Calendar {
}

abstract sig Event {
   owner: one User,
   invitedUsers: set User,
   partecipantUsers: set User
}{
   // The owner is also a partecipant
   owner in partecipantUsers
   // An invited user can not be a partecipating user
   #(invitedUsers & partecipantUsers) = 0
   // All events must be in the calendars of partecipant users
   all u: User | u in partecipantUsers iff this in u.calendar.events
}

sig PrivateEvent extends Event {
}

sig PublicEvent extends Event {
}


/****************************** ASSERTIONS ******************************/

assert userCanSeeHisCalendar {
   no u: User | u.calendar not in u.canSeeCalendars
}
//check userCanSeeHisCalendar

assert userCanSeeAllPublicCalendar {
   no c: PublicCalendar | all u: User | c not in u.canSeeCalendars
}
//check userCanSeeAllPublicCalendar
```

```
assert userCanNotSeeOtherPrivateCalendar {
   no c: PrivateCalendar | all u: User | c in u.canSeeCalendars and c ≠ u.calendar
}
//check userCanNotSeeOtherPrivateCalendar

assert userCanSeeHisEvents {
   no u: User | u.calendar.events not in u.canSeeEvents
}
//check userCanSeeHisEvents

assert userCanSeePublicEvents {
   no e: PublicEvent | all u: User | e not in u.canSeeEvents
}
//check userCanSeePublicEvents

assert userCanSeeLinkedEvents {
   no e: Event | User in (e.partecipantUsers + e.invitedUsers) and e not in User.canSeeEvents
}
//check userCanSeeLinkedEvents


/****************************** SHOW ******************************/

pred show() {
   #PublicCalendar > 0
   #PrivateCalendar > 0
   #PublicEvent > 0
   #PrivateEvent > 0
}

run show for 4 but exactly 3 Calendar, exactly 4 Event
```

7.2.2 Result of Analyzer

Here an example of result given by the Analyzer:

```
7 commands were executed. The results are:
  #1: No counterexample found. userCanSeeHisCalendar may be valid.
  #2: No counterexample found. userCanSeeAllPublicCalendar may be valid.
  #3: No counterexample found. userCanNotSeeOtherPrivateCalendar may be valid.
  #4: No counterexample found. userCanSeeHisEvents may be valid.
  #5: No counterexample found. userCanSeePublicEvents may be valid.
  #6: No counterexample found. userCanSeeLinkedEvents may be valid.
  #7: Instance found. show is consistent.
```

8 Used Tools

In this paragraph we show the tools used to create this RASD Document:

- TeXShop, LaTeX editor: to redact and format this document;
- Moqups (moqups.com): to create sketches;
- Draw.io: to create UML Use Cases, Class Diagram, Sequence Diagrams;
- Alloy Analyzer: to prove consistency of our world.