RASD
MeteoCal

Daniele Marangoni, Matteo Montalcini,
Daniele Moro

07/12/2014

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, in order to avoid bad weather condition in case of outdoor activities.
The system will allow registered users to create, delete and update every kinds of events. Whenever a user creates an event, he will have to specify when and where the event will take place and he will be able to invite any number of registered users; only the organizer will be able to update or delete the event, while invited users can only accept or decline the invitation.
The system will associate every event with weather forecast information of the specified place: it provides to notify the organizer of the event three days before the selected date in case of bad weather condition associated to an outdoor event and proposes him the closest sunny day, if any. The system notifies also the participants before an outdoor event if the weather changes.

1.2 Goals

The system has to provide this main features:

- Registration of a person to the system and logging in;
- Managing events;
- Managing invitations;
- The possibility to see information about public events;
- Searching for other users and add to favorites;
- Managing profile;
- System Notification;
- Import/export calendar;

- Managing weather forecasts about outdoor events;
- Inviting new people to sign up.

1.3 Domain Properties

We suppose that these conditions hold in the analyzed world:

- Every registered person is a real one and his personal information inserted in the system is correct;
- Each person creates only real events and, for each one, he chooses a date and a slot time in which he has no other commitment;
- Every event must be classified as "outdoor" or "indoor"; in case of events that can be in some part indoor and in other some part outdoor, the person selects "outdoor" if he wants to receive information about weather;
- A person can move an event only to a date and slot time in which he is free;
- A person can participate only to the events for which he has received an invitation;
- When a person confirms his participation, he is really going to go the event;
- Weather forecasts are provided by an external service and we assume that it provides reliable information.

1.4 Glossary

Calendar: a system to schedule tasks avoid missing events to which we have confirmed our participation.

System: it refers to MeteoCal, the system described in this document and which we are going to develop.

Guest: a person not registered yet.
He can only see information about the features of the application and eventually sign up.

User: a person who is already signed up.
He can log in and he has a profile. He can manage his calendar, invite friends, create an event and use all the functionalities described below (see Functional Requirements).

Weather forecast: the information about the weather condition supplied by a weather service on the web, used by the system for managing events.

Private: is a kind of privacy associated to an event or to a calendar.
The information catalogued as private is visible only to the creator and to the participants. The private privacy of the event is prevalent on the privacy of the user's calendar: if user's calendar is public but the event is private, the event will not be visible to other users.

Public: is a kind of privacy relative to an event or to a calendar.
The information catalogued as public are visible to all the users.

Organizer: the user who organized the event giving information about it and inviting other people.

Event: it is a commitment (e.g., appointment, birthday, celebration of something important . . .) in which are involved the organizer and eventually some other people invited.

Participant: a person who has replied positively to an invitation for an event.

Invitation: request to join at an organized event sent by an organizer to another user/person.

1.5 Assumption

There are few points that aren't described in a specific way, so we'll assume that:

- Only the organizer can update or delete an event, in order to avoid users to modify an event to which they only participate. Doing this, unauthorized modifications are not allowed. The organizer can decide the visibility of an event only upon creation and the invited users can't modify this visibility;
- When a user has already a task and he's invited to another event, he can decide to participate to at most one of the two events. This prevent the presence of inconsistencies in user's calendar because obviously a user can not participate to more than one event at a time;
- Events are organized in typologies, of which the user can find a finite set of possible choices;
- A user can't establish in a precise way wanted weather condition for an outdoor event: for example, a common user can't insert how many millimeters of rain would accept for an outdoor event. So we assume that the system will provide a set of predefined intensity levels for wind, rain/snow and temperature;
- The system updates the weather condition of the events automatically every 12 hours;
- A user can make his calendar public, in the sense that all the other users can see when the owner is busy; there is also the information about which event occupies a slot in case of public events;
- A user can import a calendar: but if this calendar presents any overlapping between events or the adding of the new events will cause overlaps, the import fails. If he wants to import this file, he has to solve the overlaps before retrying the import.
- Since it is not specified the level of the interaction among users that the system has to offer, we decide that:
  - A user can contact other user by using their personal information (e.g. email, phone number) which he can find on user's profile;
  - A user can suggest other people to join the system with an email;
  - *A user can send invitation not only to other users but also to guests, inviting them to register themselves before confirm participation;*

  - *A user can ask information about an event directly using system messaging service (handled by the system itself);*
  - *A user can add other users to his favorites contacts: this will allow him to invite them more easily to future events.*

We specify that the points in *italic* are advanced functionalities that are not explicitly required, but they would improve the system usability. We will implement them at the end of the project when we are sure that all the other functionalities work well.

1.6 Proposed system

We propose a web platform, that will provide users the services described below.
The users will be able to access his profile, create events and send invitations: in this way, it will be easy to organize the own agenda and manage events also because the system will prevent users to confirm participation to overlapping events.
The system will indicate all the information about weather forecasts associating them to the event through their location (especially in case of outdoor events).

1.7 Identify Stakeholders

Our stakeholder is the professor, who gave us this didactic project. The professor want to see the project working at the and of the semester but she also want to see the progress of the project through the various steps of the developing of this software. Hence, it's necessary to redact various documents that follow the developing of the software and our concerning decisions. In this way we can focus on the critical aspects of the software and we have time to reflect upon our choices of making this project as better as possible, evade to start without thinking.

After the identification of the requirements and the specification, it is also necessary to pass through other many steps before the implementation, as the Design Document in which we design our web application. Then, only after the implementation, we'll provide to test the software.

In the realization of the software we have to focus on the functionalities requested by the professor. The users of this application are potentially all the persons who need to schedule their days and their appointments to avoid to forget events, to overlap two or more events which they have managed to be present in and to ensure that outdoor events aren't ruined by weather conditions.

The final scope is to consign a final product that is working at the end of the semester, combining to all the document that are used in a real software development.

1.8 Other consideration about the system

There are some consideration about how MeteoCal should be once terminated:

- **Easy to use:** we try to make it as clear and as intuitive as possible in order to allow also less expert users to exploit it easily, but at the same time without neglecting all the interesting things and functionalities that we want to provide.
- **Easy learning:** the user doesn't have to spend too much time to learn how to use MeteoCal, but it has to be as much intuitive as possible.
- **Stable:** the system has to grant that the data saved can't be lost and all the data about the registration of the new user are collected and consistent.
- **Nice look and feel:** the design of the application will be nice and of good aspect.

2. Actors Identifying

The actors involved in our information system are:

- **Guest:** a person who haven't already had the registration and can only perform this action and get information on the application and its features.
- **User:** a person who's already registered and can use the features made available by the software such as the management of his calendar.

3. Requirements

Starting from the goals and the domain properties, we can now analyze more in detail which features the system has to provide:

- Registration of a person to the system and logging in:
  - The system has to provide functionality that allows people to register themselves to MeteoCal;
  - The system will allow users to log in.
- The possibility to manage events:
  - The system will give to users the possibility to create an event;
  - The system will allow organizer to update an event;
  - The system will guarantee the chance to delete an event they have created;
  - The system has to provide the functionality to confirm participation to an event;
  - The system will permit to delete the participation to an event to which a user has previously confirmed his participation;
  - The system will make possible to define the visibility of an event (private or public) upon creation;
  - The system has to provide a manage time consistency check when a user create an event, in order to avoid conflicts with organizer's existing events.
- The possibility to manage invitations:
  - The system will allow inviting other users to an event when creating an event, but also later;
  - *The system will give the possibility to invite unregistered persons to an event, asking them to register themselves on the application before confirm/deny the participation (advanced functionality);*
  - *The system will guarantee the chance to ask information about an event (advanced functionality).*

- The possibility to see information about public events:
  - The System will allow users to see the information about an event (place, date, participants . . .) only if it's classified as public or if it is private and the user has been invited to.
- Searching for other users and add to favorites:
  - The system will make possible to search of other users;
  - *The system will give the possibility to a user to add other users to his favorites (advanced functionality);*
  - The system will permit to see public events to which a user is going to participate if his calendar is public.
- Managing profile:
  - The system will provide a functionality to update user's profile;
  - The system will give the chance to make user's calendar public/private;
- System Notification:
  - The system will send email notifications to users in many different situations:
    - ∗ invitation to an event;
    - ∗ delay/delete of an event to which a user has previously confirmed his participation;
    - ∗ weather forecast changed in outdoor event;

    Moreover, the organizer will be notified also about bad weather conditions related to an outdoor event which he has created, three days before the date of the event.
- Import/export calendar:
  - The system will allow users to import and export their calendar
- Managing weather forecast about outdoor events:
  - The system will update weather information associated to events periodically;
  - The system will suggest the closest sunny day to the event creator, if any, in case of bad weather forecast, three days before an outdoor event.
- Inviting new people to sign up:
  - The system will make possible, for a user, to invite unregistered persons to join the system.

3.1 Functional Requirements

Defined the main features that the system has to provide, now we can list some functional requirement belonging to the two different actors we have identified:

GUEST: since he is not registered yet, he can only use very few functionalities. He can:

- Search for users of MeteoCal and view their profiles;
- Sign up.

USER: he can use all the features provided by the system, so he can:

- Log in;
- Modify data inserted in his profile;
- Search for other users and view their profiles;
- Establish/Modify the visibility of his calendar;
- Create an event and send invitations to other users/guests for that event;
- Establish the visibility of an event created by himself upon creation;
- Modify, delete or delay an event of which he is the organizer;
- Reply to an invitation of another user;
- Delete his participation from an event to which he has previously confirmed his presence;
- Send a suggestion to sign up to a guest;
- Check the weather forecast of the events he has organized/he is going to participate to.
- See when other users has commitments if they have made public their calendar;
- *Ask for information about an event (advanced functionality);*
- *Reply to information request about an event organized by himself (advanced functionality).*

3.2 Non-Functional Requirements

3.2.1 User Interface

The interface of MeteoCal is thought to be used via web, but it would not be a bad idea to think, in the future, to provide also a mobile version (that, however, we are not going to implement), because it would simplify the managing of the own agenda, since smart phones are becoming very common.

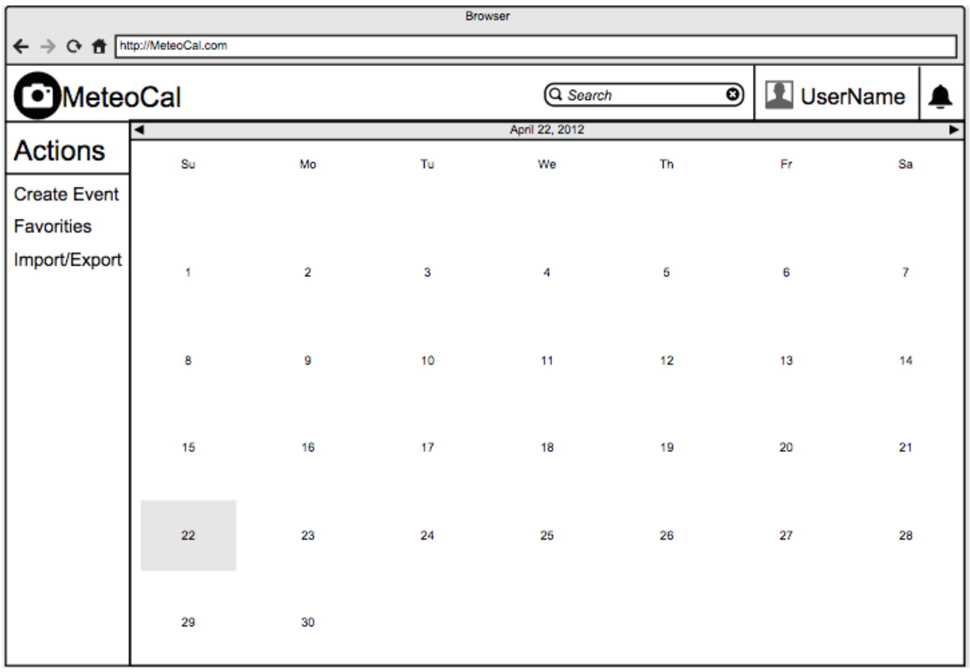
The application has to allow users to see the own profile page after the log in and has to guarantee direct access to all the available functionalities: in particular, it is important that all the functionalities are reachable without performing different passages, but in a very quick and easy way.

In order to create a simple platform that a user can exploit immediately after the sign up and the log in, without spending much time in understanding how to use it, we also decide not to burden the user interface by inserting more and more elements, but to maintain it as minimal as possible: moreover, it is not very different from that of the popular social networks, so it's possible, for users, to move in a well-know field.

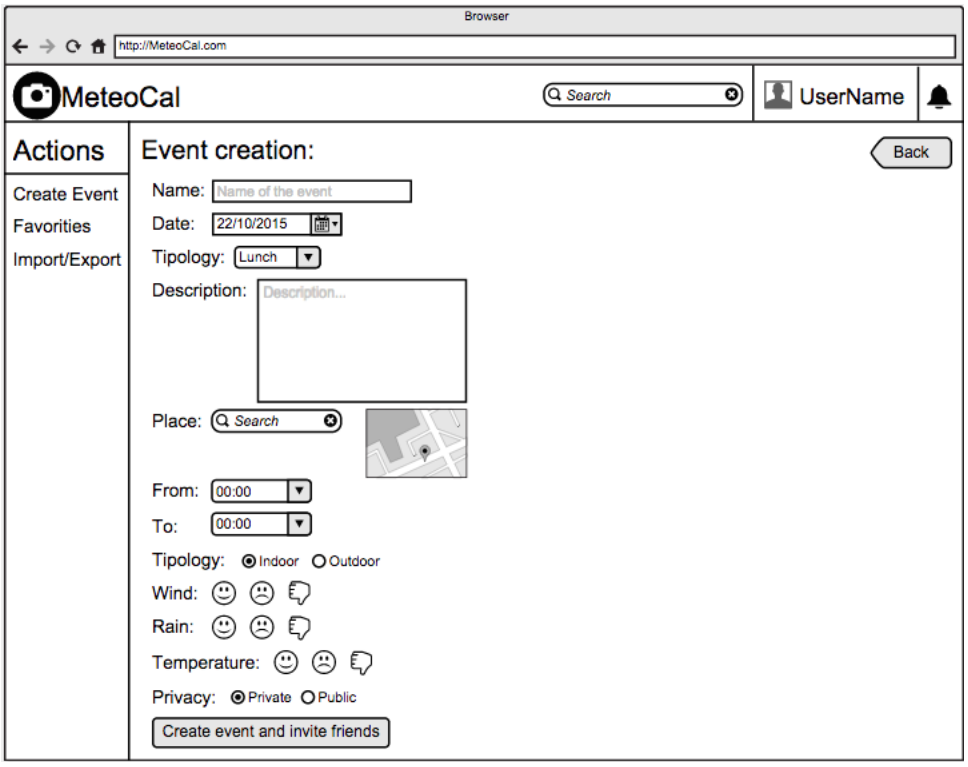
The pages will contain forms to be filled for the specific functionality (like create an event, send invitations, search for a user, modify own profile).

In our platform there are two main pages: we want to explain now how their aspect will be and also their scope.

The first one is the Home Page, from which all the provided functionalities are reachable. As we can see below, on left side a user can find the direct links for create an event, look at his favorites and also import/export a calendar, (while on the other side) on the top the user has the possibility to go to his profile, to read his notifications and also to search for other user. In the center of the page, there is the user's calendar.



The second page is the create event page: all the fields need to be filled, in order to create an event with all the relevant information that another user needs to decide to participate or not. By inserting data about time and location and the weather preferences, the creator can exploit the weather forecast service of the application, while through the privacy information he can decide to make public or private the event he is going to create.



Browser

http://MeteoCal.com

MeteoCal | Search | UserName

Actions

- Create Event
- Favorities
- Import/Export

Event creation: | Back

Name:	Name of the event
Date:	22/10/2015
Tipology:	Lunch
Description:	Description...
Place:	Search
From:	00:00
To:	00:00
Tipology:	[✓ Indoor] Outdoor
Wind:	☺ ☹ 👎
Rain:	☺ ☹ 👎
Temperature:	☺ ☹ 👎
Privacy:	[✓ Private] Public

Create event and invite friends

3.2.2 Documentation

- **RASD:** Requirement Analysis and Specification Document, to clarify the given problem and to specify our goals and how to reach them.
- **DD:** Design Document, through which we define the structure that the system will assume at the end.
- **JavaDoc comments on the source code:** in order to simplify the future work of developers who have to understand the code easily.

3.2.3 Architectural Considerations

We will develop the project in Java with J2EE platform and there will be a database in which all the information of the system will be stored.

An internet connection is needed to run MeteoCal and also a browser.

4. Specification

Now we write some specification in order to clarify how the system will work and guarantee to reach the goals listed before:

- Upon registration, the user has to communicate his name, surname, birthday, email, password, confirmation password, sex, cellphone *(not mandatory)* and city of residence. By inserting this information, it would be possible, for other users, to search for him and to invite him to an event.
  The user has also to decide the privacy of his calendar (public or private).

- At the creation of an event, the user has to insert this information about it: name, date, typology, description, place, slot of time, outdoor/indoor, preferable weather conditions, users invited and the privacy of the event.
  Through the date, slot of time and place information, the system will provide to the organizer the weather forecasts for the event; the system makes possible to choose the preferable weather conditions by selecting from three different levels of intensity about wind, rain/snow, temperature.
  Also the typology of the event can be selected from a predefined list and the system give also the possibility to select the option "Other" and to insert another typology.
  The privacy, once established upon the event creation, can't be changed.

- We have to clarify the concept of privacy of a calendar and of an event.
  If a calendar is "private", all other registered people can't see anything, not even the slots of time in which the user has a commitment.
  If a calendar is "public", it is also possible to see the information about public events, while data about private events are not visible.
  If an event is "private", the information about it are visible only to the invited people and it makes no difference if the organizer's calendar is public or not.
  If an event is "public", all other users can see information about it.

- The organizer can invite a non registered person to an event only if he knows his email, while he can send invitations to other users through their emails or their names.
- When a non registered person receives an invitation to an event, first he has to sign up in MeteoCal, then he has to log in and finally he can accept/refuse the invitation.
- A user can join only in events to which he has been invited.
- A user can't accept an invitation to an event if, at the same time, he has also another commitment. In fact, overlaps are not accepted, so, if he wants to participate to the event for which he has just received the invitation, he has to cancel his participation to the other event and also later he can accept the new invitation.
  The same procedure is applied when an event is delayed and a person has another commitment at the same time to which the event has been delayed: if he wants to participate to the delayed event, he has to cancel his presence to the other one.
- Email notifications are sent when the weather forecast are bad in case of outdoor events, in case of event delay or delete and in case of forecast changing.
- In case of bad weather forecast for outdoor events, three days before the event, the system proposes to the organizer the closest sunny day (if any), and the creator can decide if he wants to move the event to the suggested day or to another one.
- When a user wants to import a calendar, he has to be sure that it is valid, otherwise the system will block the operation, displaying an error message.
  Moreover, an error notification is sent if the adding of the imported events would cause overlaps: in this case, the user is invited to solve by himself the overlaps, so he has to cancel his participation to some events in order not to have any overlaps. Only after this procedure, the import operation is accepted and performed by the system.

5. Scenarios Identifying

We identify some scenarios concerning MeteoCal:

- **User registration**

  Bob is a college student and he is going to graduate, so he decides to organize a big party for the event. Talking with some friends, he discovers that exist a web application, called MeteoCal, which allows users to create an event and send invitations not only to other users, but also to unregistered people, just indicating their emails. It is a very practical applications, because it provides users to know also the web conditions of the place in which the events will take place. Following the advice, once at home he opens is laptop and searches the web for MeteoCal and, after reading some positive comments, he decides to register on the site. As required, he inserts his personal information: name, surname, gender, birthday, email, city of residence and finally a password, but he decides not to insert his mobile phone number, not even so confident about the security of MeteoCal. He submits this data to the system and, after a while, he is notified of the registration.

- **Event creation**

  Once the registration has been successful, Bob immediately start to explore the website. He discovers that he can also see the calendars of the users who decided to make it public and, searching by email, he finds his friends calendars. Once he understands how to use MeteoCal, he creates a new event, called "Bob's Graduation". He inserts all the information required, deciding to organize an aperitif on 30th September at BhangraBar. In order to indicate the exact position, he searches the web and finds the right address (Corso Sempione, 1, 20145 Milano) and he submits that information to the site. He can also check the correctness of the inserted position looking at the map, which exhibit the inserted location. Moreover, he adds a short description of the event and since the Bhangrabar is not completely indoor, but also outdoor, he decides to put constraints on the weather: in case of rain or strong wind, the event should be delayed. Finally, he starts to send all the invitations: only now he realizes that he can find registered users not only through email, but also inserting their names. The only constraint is that unregistered people must be invited by indicating their emails and, once they have received the invitations, they can accept or refuse it only after having registered themselves to MeteoCal. Listed all the people he wants to invite, he confirms the event and, immediately, he receives an email confirmation.

- **Event overlaps, refuse**

  Paul, looking at his email, finds Bob's invitation: initially he is surprised, because for a while he had lost the contacts with him, but anyway he decides to log in MeteoCal to accept the invitation. But when he looks at the event page, he discovers that he cannot accept the invitation, because he has another commitment overlapping the new one: he has to cancel the other event participation before he can accept the new one. But the other event is too important and he can't miss, so he can't go to Bob's party. Disappointed, decides to refuse Bob's invitation, thanking him anyway.

- **Event overlaps, accept**

  Lisa, Bob's best friend, receives, by email, Bob's invitation: she is very happy about it, so she immediately logs in MeteoCal to accepts the invitation, discovering that she can't confirm her participation: in fact, she has another commitment exactly in that day, in particular she has to meet her friends Laura and Lucia to go for a run. So if she wants to accept the invitation, she has to cancel her participation to the other event. But Bob's event is too important and, knowing that she can't miss that party, she calls Laura and Lucia and, apologizing for the inconvenience, tells them that she would not be present for the run, but she would not miss their following meeting, on 1st October. So Lisa opens her calendar, cancels her participation to the run-event and finally accepts Bob's invitation.

- **Registration before confirm participation**

  John receives by email the invitation for Bob's party: he is not registered on MeteoCal, so he can't answer immediately to the invitation. Knowing nothing about MeteoCal, he searches for it on the web, and discovers what it is. In order to answer to the invitation, but also thinking that the application could be very useful, he registers himself on the site and starts to explore it. Once he has found the event on MeteoCal, he reads the information about it. He has already been to BhangraBar and he really likes it, so he is very happy about Bob's choice; moreover, he has no other commitment that day, so he confirms his participation and thanks Bob for the invitation.

- **Add invitation**

  Five days before the party, Bob receives a good news: two of his cousins who live abroad, happy for the result reached by Bob, have decided to come back to see his graduation ceremony. Bob decides to invite to the party his cousins, so he opens his laptop and, after logging in MeteoCal, sends two new invitations for the organized event.

- **Bad weather condition, delay event**

  On 29th September Laura, through MeteoCal application, is notified about a change in the weather conditions for 1st October, exactly the day in which she has to go for a run with Lisa and Lucia: rain and wind are expected. Thinking that it's not a good idea running under the rain, she looks for another possible date: in particular, she follows the system suggestion, which indicates her the closest sunny days, and chooses to go for a run on 3rd October, hoping that Lisa and Lucia would be free for that date. So she postpones another time the date, and attends for Lisa's confirm.

- **Notify delayed event, no problem**

  Lisa is notified by MeteoCal that an event to which she is going to participate has been modified: in particular, Laura has postponed the date for their run. She looks her calendar and, verified that she has no other commitment for the 3rd October, she realized that no problem arise for her through this change.

- **Notify delayed event, some problem**

  Lucia is notified by MeteoCal about the modification of the event for the 1st October: Laura has postponed the event on the 3rd October, exactly the day in which she has planned to go to the gym (overlapping specified also by the notification). Thinking that she can go to the gym in another day, she decides to participate to the delayed event and, in order to apply her decision, she cancels that event from her calendar and finally she can accept for the second time Laura's invitation.

- **Ask for event information**

  Lisa has never been to BhangraBar and is very curious to know which kinds of foods are served there. But even she decides to request some information to Bob, Lisa discovers that she has no money left in the mobile phone: remembering the possibility to ask for information about an event through MeteoCal, she opens her laptop, logs in the application and, after having found the event, she asks for information to Bob through the provided functionality. Then she closes her laptop, hoping not to have to wait long for an answer.

6. UML Models

6.1 Use Case Diagrams

From the scenarios identified in the previous chapter, we can derive some use case that we decide to split in some macro categories to make the situation clearer:

- **Registration/Login**
  - Log in;
  - Sign up.
- **Event and Invitation Managing**
  - Create event;
  - Delete event;
  - Update event;
  - Reply to an invitation;
  - Delete participation to an event;
  - Invite friend/guest to an event.
- **Profile Managing**
  - Modify profile information;
  - Make user's calendar public;
  - Add user to favorites;
  - Search for a user;
  - Import/Export calendar;
  - Invite friend to sign up.
- **Message Managing**
  - Ask information about an event;
  - Reply to a question about an event.

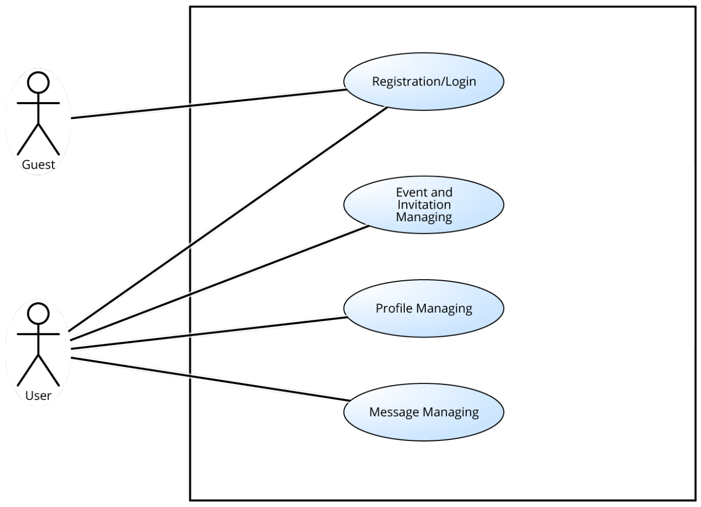


Figure 6.1: This diagram helps to read and to understand the composition of the following diagram.

6.1.1 Registration/Login

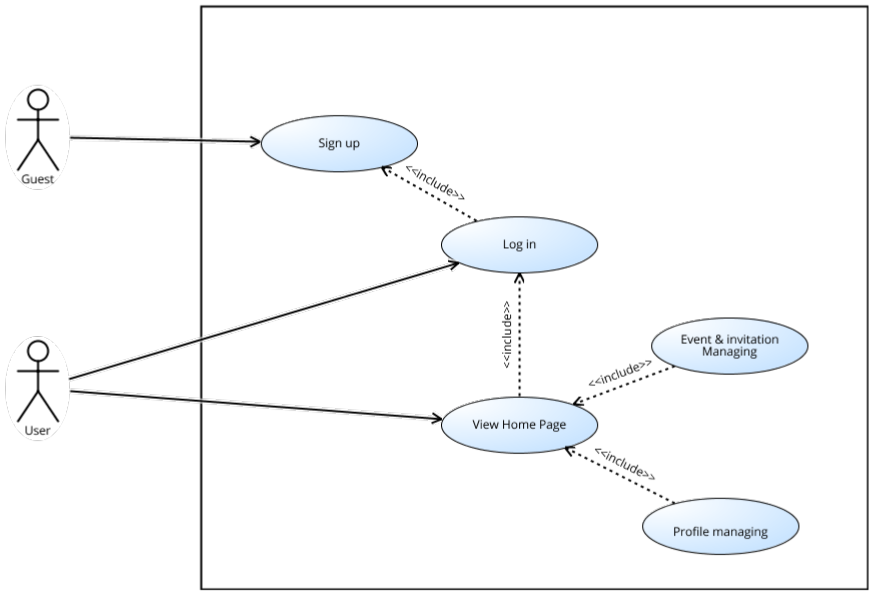


Figure 6.2: Diagram of registration and log-in use cases.

Name	**Log in**
Actors	User
Entry Conditions	The user has successfully signed up to the system.
Flow of events	• The user opens the home page of the platform; • the system shows him the page; • the user enters his email and password in the input form provided; • the user clicks on the log in button; • the system shows the user's profile page.
Exit Conditions	No exit condition
Exceptions	The information (username and/or password) inserted by the user is wrong, an error message is shown.

Name	**Sign up**
Actors	Guest
Entry Conditions	The guest hasn't already signed up and he views MeteoCal home page, or he received an invitation to sign up.
Flow of events	If he received an email invitation, he clicks on the link on the email and he is redirected to the home page of meteoCal. After: • The guest clicks on the sign up button; • The system shows him a page which contains all the input form for all the data required; • The guest fulfills the input forms provided by the system and clicks the sign up button; • The system redirects the new user to MeteoCal, and then confirms the registration.
Exit Conditions	The information about the new user is stored into the database in the way to grant his log in. The log in is now possible.
Exceptions	The guest inserts a not valid password, or doesn't fill some mandatory fills, or the mail has been already used by a user. In this case the page is reloaded showing an error message.

6.1.2 Event and Invitation Managing

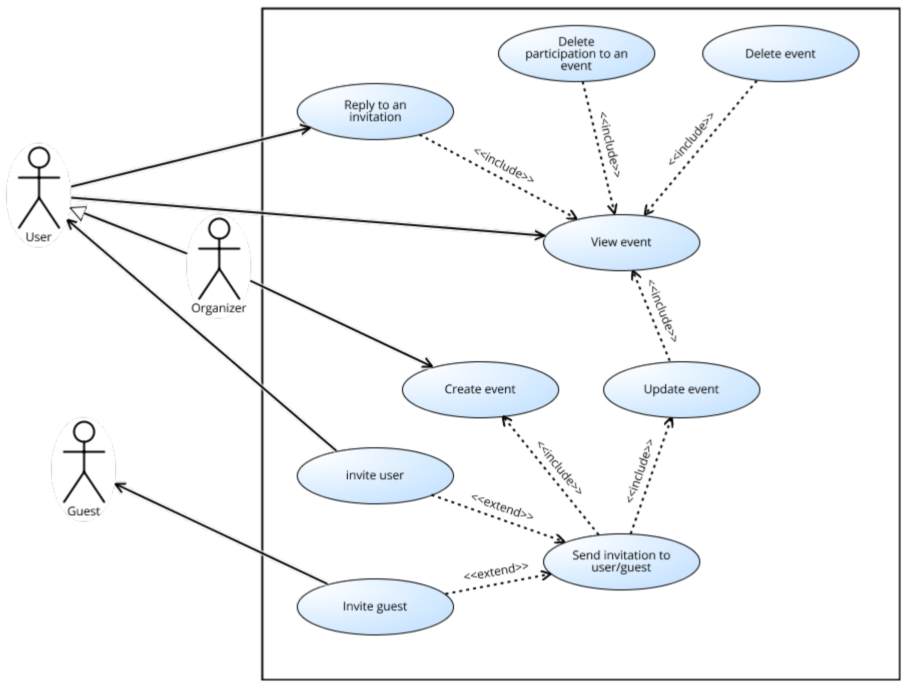


Figure 6.3: Diagram of event and invitation managing.

Name	**Create Event**
Actors	User
Entry Conditions	The user has already logged in.
Flow of events	• The user clicks on the button "New event"; • The system shows a page where there is a list of input forms to be completed with the information of the new event; • The user fulfills all the input forms, selecting the date, writing the name of the event, the description (not mandatory), indicating if the event is public or private, selecting the location by inserting the address and selecting the correct position on the map, selecting if the event is outdoor or indoor, and selecting the typology ecc..; • The user adds (if he wants) invited users; • The user clicks on the submit button; • The system reloads the profile page containing the new information, and sends the notification for the invite to the other users.
Exit Conditions	The data is now stored and profile page is reloaded with the new event. The system sends requests to users invited.
Exceptions	The user doesn't fill some mandatory fields or selects a date in which he has already another commitment: in this case the system shows an error message.

Name	**Delete event**
Actors	User
Entry Conditions	The user has already logged in and created an event.
Flow of events	• The user selects an event from own created events; • The system generates the page with all the information about the event; • The user clicks on the button “Cancel”; • The system shows a confirmation message with the possibility to abort or confirm; • The user clicks on confirm; • The system shows a page with the confirm of the action and sends to all the invited users a notification.
Exit Conditions	The system deletes the event from the database and sends to all the invited users a warning notification about the cancellation.
Exceptions	The user selects to abort the cancellation. The user switches off the platform before confirmation.

Name	**Update event**
Actors	User
Entry Conditions	The user has already logged in and created an event.
Flow of events	• The user selects an event from own created events; • The system generates the page with all the information about the event; • The user clicks on the button "modify event"; • The user modifies some information of the event (name, date, location, typology, description, visibility of the event); • The user clicks the "modify" button; • The system shows a confirmation message with the possibility to abort or confirm; • The user clicks on "confirm" button; • The system reloads the page containing updated information of the event, and sends the modify notification to the invited users.
Exit Conditions	The updated information about the event are stored and are visibile from the page related to the event.
Exceptions	The user selects to abort the modification. The guest inserts doesn't fill some mandatory fills: in this case the page is reloaded showing an error message.

Name	**Reply to an invitation**
Actors	User
Entry Conditions	The user has received an invitation to an event.
Flow of events	• The user clicks on the event or on the invite notification; • The system shows the event page with all the information; • The user clicks on “Accept” or “Refuse” button; • The system adds the user to the list of the users who have accepted/refused the invitation, according to the previous answer and displays a confirm message.
Exit Conditions	The system stores the information about the user’s answer, by adding him to the list of the users who have accepted/refused the invitation.
Exceptions	The user clicks on “Back” button: in this case, no changes are performed by the system to the list of invitation. The user has an overlapping commitment, the system shows an error message and doesn’t do anything.

Name	**Delete partecipation to an event**
Actors	User
Entry Conditions	The user has already logged in and confirmed the participation to an event.
Flow of events	• The user selects an event that has already confirmed his participation; • The system generates the page with all the information about the event; • The user clicks on the button “Cancel participation”; • The system shows a confirmation message with the possibility to abort or confirm; • The user clicks on confirm button; • The system shows a page with the confirm of the action and sends a notification about that to the owner of the event.
Exit Conditions	The system deletes the participation of the user to the event, and notifies the organizer of the event about that.
Exceptions	The user selects to abort the cancellation. The user switches off the platform before confirmation.

Name	**Invite friend/guest to an event**
Actors	User
Entry Conditions	The user has already created an event.
Flow of events	• The user clicks on the event; • The system shows the event page with all the information; • The user clicks on "Send new Invitations" button; • The user searches and selects the users and guests he wants to invite, searching by their emails (or also by their name and surname if they are users) and then clicks on "Send Invitations" button; • The system adds the selected users and guests to the list of invited person and sends them an invite notification.
Exit Conditions	The system stores the information about the new invited person and send them a new notification.
Exceptions	The user clicks on "Back" button: in this case, no changes are performed by the system to the list of invitation.

6.1.3 Profile Managing

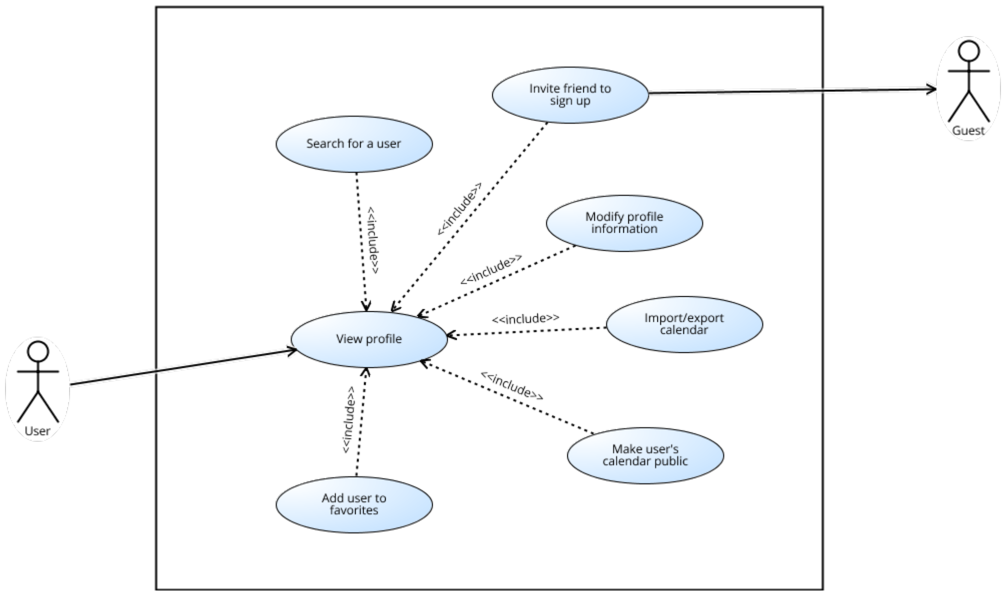


Figure 6.4: Diagram of profile managing use cases.

Name	**Modify profile information**
Actors	User
Entry Conditions	The user has successfully logged in and he is looking at the home page.
Flow of events	• The user clicks the “my profile” button; • The system generates the page with all the information about his profile; • The user modifies some of the inserted data (he can modify email, password, city of residence, mobile phone number); • The user clicks the “modify” button; • The system shows a confirmation message with the possibility to abort or confirm; • The user clicks on “confirm” button; • The system shows the updated profile page of the user.
Exit Conditions	The system updates the user’s profile in the database.
Exceptions	The user selects to abort the modification. The user inserts a not valid password, or, doesn’t fill some mandatory fills, or the mail has been already used by a user. In this case the page is reloaded showing an error message.

Name	**Make user's calendar public**
Actors	User
Entry Conditions	The user has successfully logged in.
Flow of events	• The user is looking at the home page; • The user clicks on the button "Option"; • The system shows him the options concerning to his calendar; • The user modifies the privacy of his calendar; • The user clicks the "Confirm" button; • The system shows a confirmation message with the possibility to abort or confirm; • The user clicks on "Confirm" button; • The system shows the updated options.
Exit Conditions	The system updates calendar's options.
Exceptions	The user selects to abort the modification: in this case the options are restored to its previous state.

Name	**Add user to favorites**
Actors	User
Entry Conditions	The user has already logged in.
Flow of events	• The user clicks on “Favourites” button; • The system generates the page with the list of favourites (if any) and the input form to search and after add a user; • The user fills the input form with the mail or the name and surname of the user that he wants to add; • The system generates the list of the user that matches with the input inserted by the user; • The user selects the other user that he wants to add; • The system regenerates the list of favorites with the new user added and confirms the addition.
Exit Conditions	The system adds the selected user to the actor’s favorites users.
Exceptions	The user fills the input form with incorrect data: the system will display an error message.

Name	**Search for a user**
Actors	User
Entry Conditions	The user has already logged in.
Flow of events	• The user types the name and surname of a user, or his email in a search field to have a quick search; • The user clicks “search” button; • The system shows the result, in case of homonymy it shows also the email.
Exit Conditions	No exit condition.
Exceptions	The research gives no results. The user clicks “search” without entering any information.

Name	**Import/Export Calendar**
Actors	User
Entry Conditions	The user has already logged in.
Flow of events	• The user clicks on "Import/Export" button; • The system shows the page with "Import" button and "Export" button; • The user selects one of the two options; • If the user selected "Import", he has to choose the file he wants to import and he has to click the "Confirm" button; if the user selected "Export", he only has to click "Confirm" button; • The system applies the user's choice;
Exit Conditions	The system exports or imports the calendar.
Exceptions	The system blocks the import of a calendar if that is not valid and displays an error message. The user clicks on "Back" button: in this case, the system doesn't make any modification.

Name	**Invite friend to sign up**
Actors	User
Entry Conditions	The user has already logged in.
Flow of events	• The user clicks on the button “Invite friends”; • The system shows him a new page in which there are two fields to compile: the email to which we have to send the invite and a message to attach; • The user types the email and an optional message; • The user clicks on the button “Send invitation”; • The system shows a page with the confirm of the action;
Exit Conditions	The system sends the invite to the specified email with the specified message.
Exceptions	The user doesn’t fill the field relating to the email or the mail has been already used by a user. In this case the page is reloaded showing an error message.

6.1.4 Message Managing

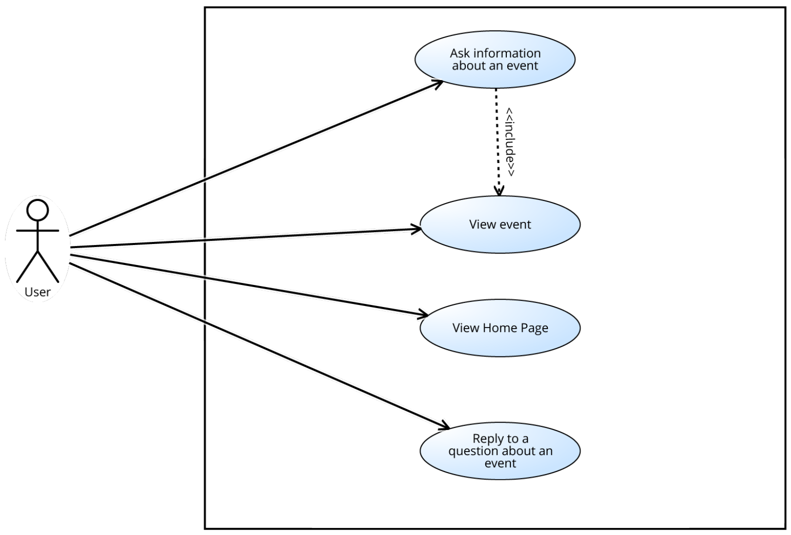
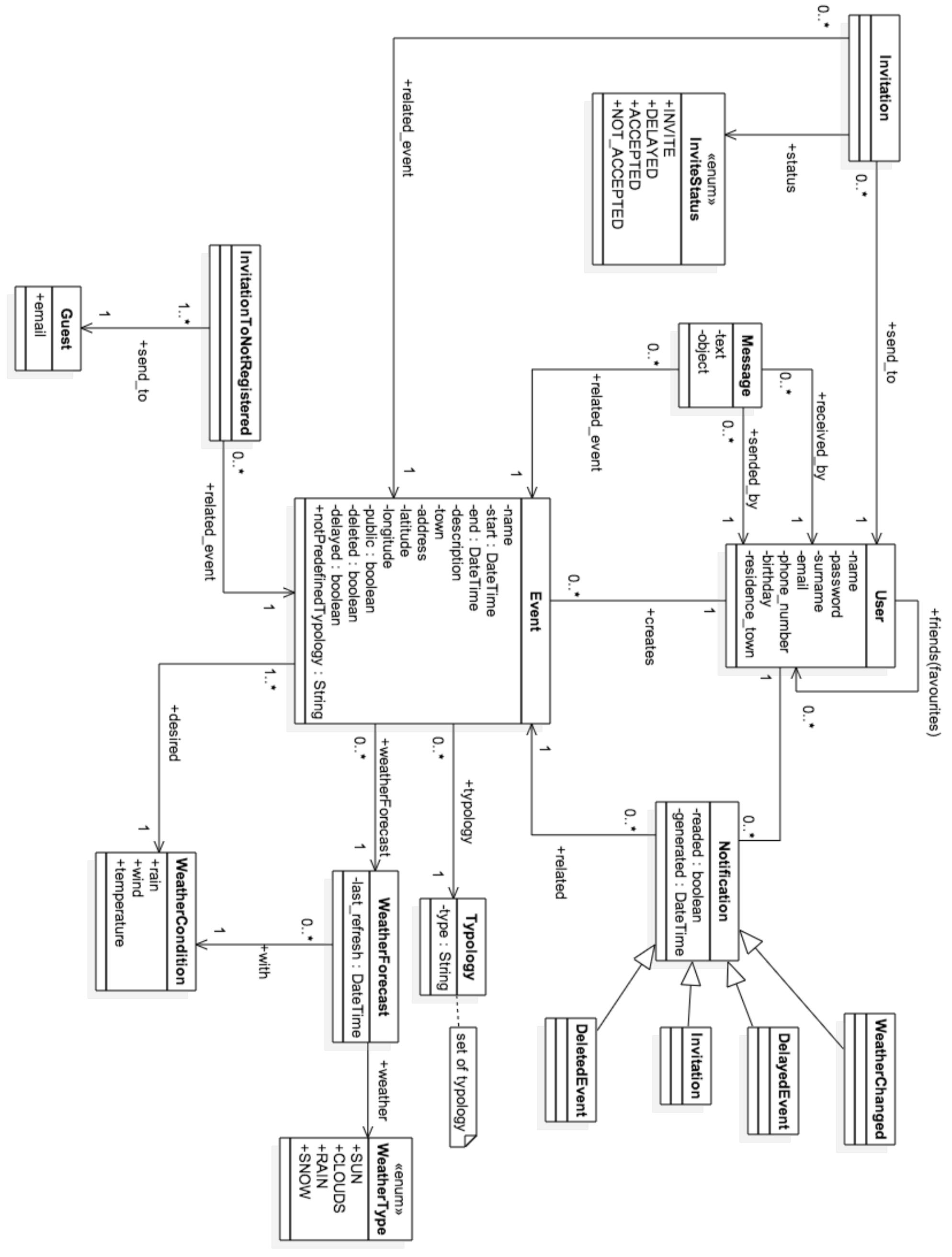

Ask information about an event
<<include>>
View event
View Home Page
Reply to a question about an event
User

Figure 6.5: Diagram of message managing use cases.

Name	**Ask information about an event**
Actors	User
Entry Conditions	The user has already logged in and has received an invitation to an event.
Flow of events	• The user clicks on an event • The system generates the page with all the information about the event; • The user clicks on the button “Ask information about the event”; • The system shows him a new page in which there is an empty text field; • The user writes the question about the event in the text field; • The user clicks on the button “Send question”; • The system shows an advice with the confirm of the action.
Exit Conditions	The system sends the question to the creator of the event with the specified message from the user.
Exceptions	The user doesn’t fill the message field: in this case the system shows an error message. The user aborts the operation clicking on “Back” button.

Name	**Reply to a question about an event**
Actors	User
Entry Conditions	The user has already logged in and has received a question about a created event.
Flow of events	• The user clicks on the notification of the question; • The system generates the page with the question and an empty text field; • The user writes the answer to the user; • The user clicks on the button "Send"; • The system shows an advice with the confirm of the action.
Exit Conditions	The system sends the answer to the user that has generated the question.
Exceptions	The user doesn't fill the message field and clicks on the button "Send": in this case the system shows an error message. The user aborts the operation clicking on "Back" button.

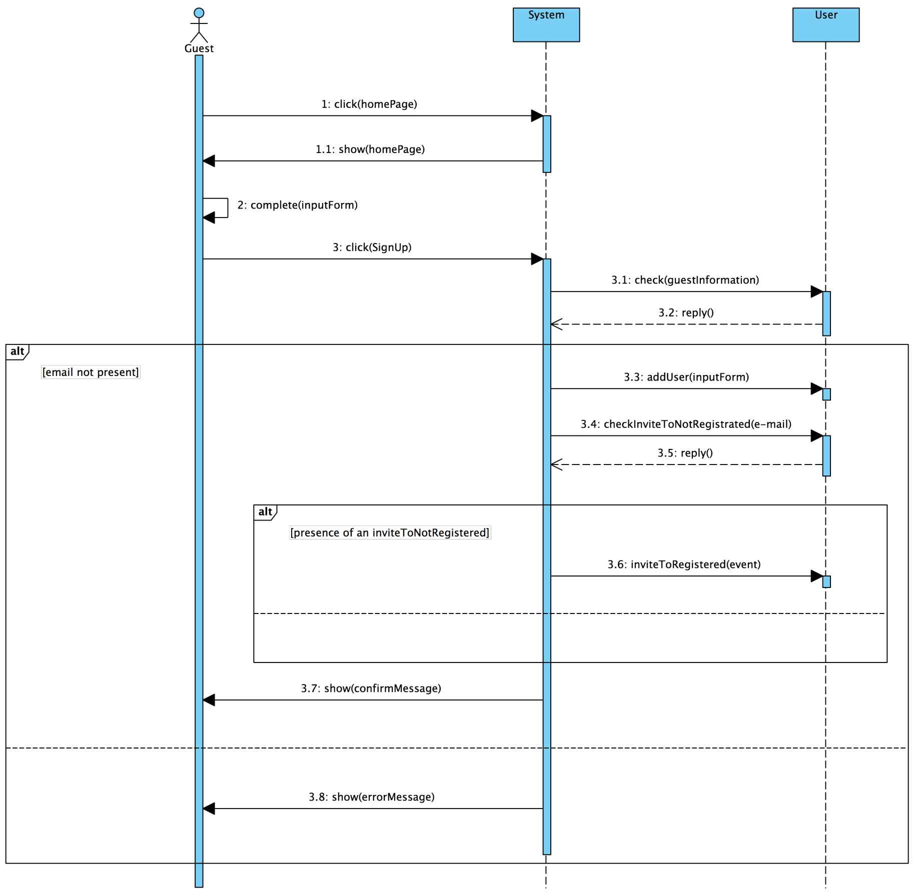
6.2 Class Diagram



6.3 Sequence Diagrams

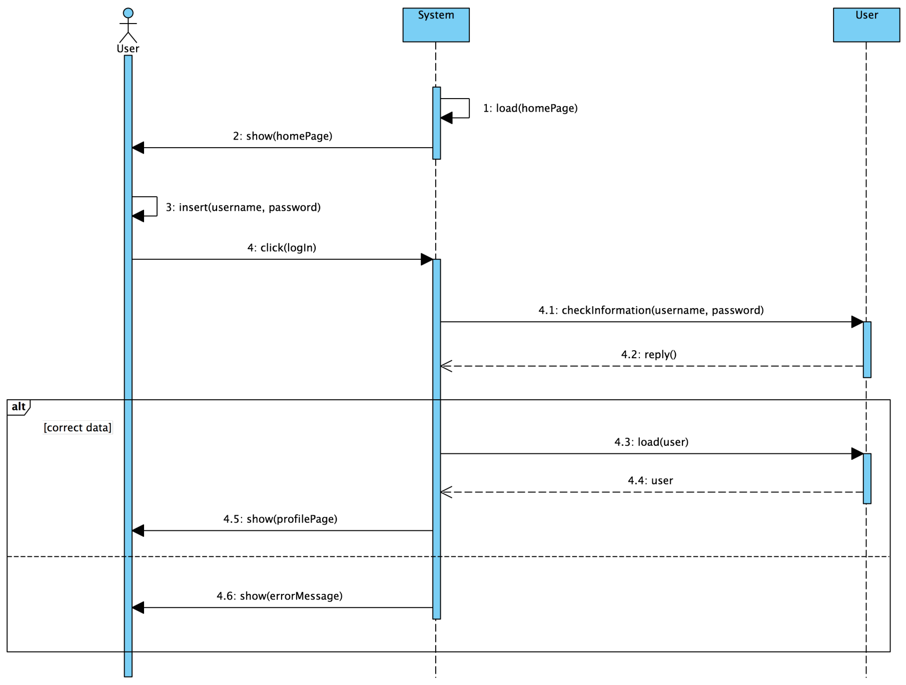
6.3.1 Sign Up

The user isn't already registered at the application.



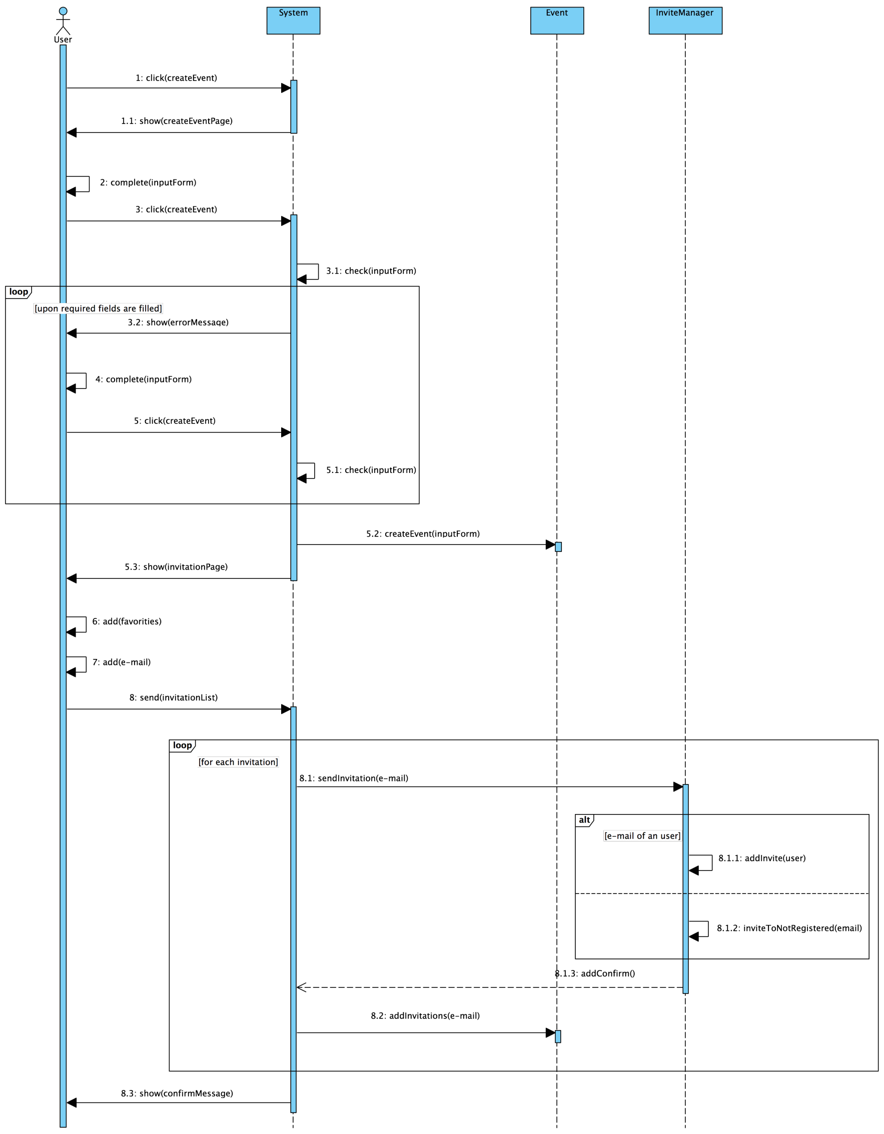
6.3.2 Log in

The user is already registered at the application and he log in.



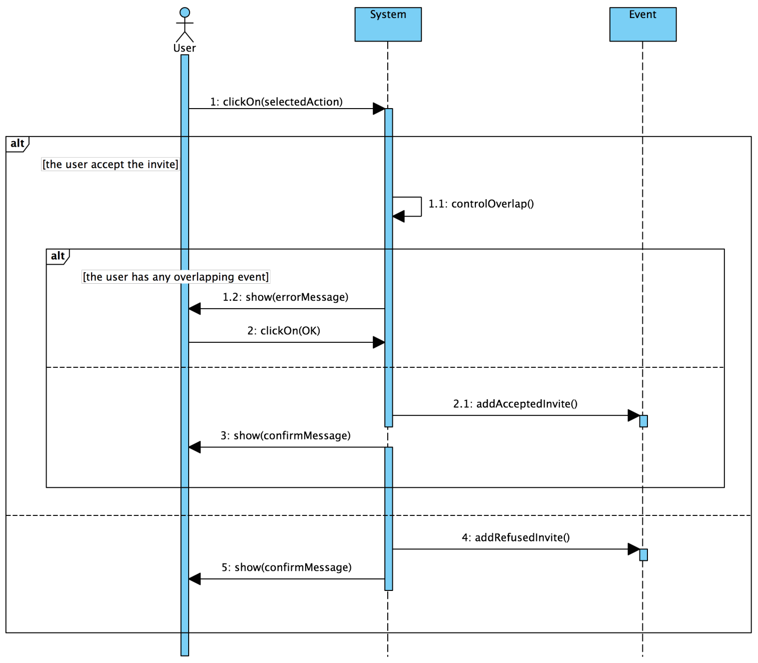
6.3.3 Create event

We suppose that the user has already logged in. The agent ”inviteManager” is the part of the system which deals with invitations.



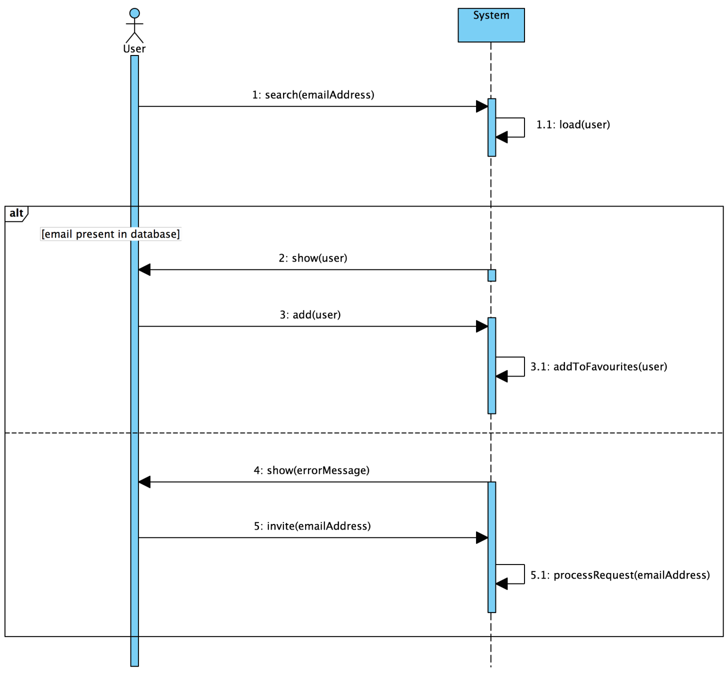
6.3.4 Accept/refuse invitation to an event

We suppose that the user has already logged in and he receives an invitation to an event.



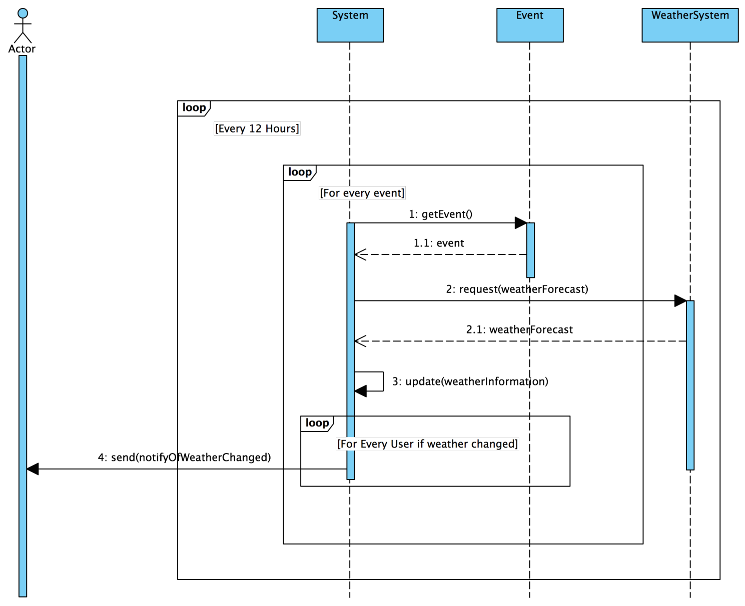
6.3.5 Add to favorities

We suppose that the user has already logged in and he decides to add in his "Favourite" section a friend by inserting his email.



6.3.6 Weather refresh

This diagram refers to the automatic system that refresh the weather forecast of the events every predefined period of time (in this example 12 hours). The weather system indicated refers to a part of our system dedicated to take weather forecast information from the external system.


Actor
System
Event
WeatherSystem
loop
[Every 12 Hours]
loop
[For every event]
1: getEvent()
1.1: event
2: request(weatherForecast)
2.1: weatherForecast
3: update(weatherInformation)
loop
[For Every User if weather changed]
4: send(notifyOfWeatherChanged)

7. Alloy Modeling

In order to verify if our class diagram can be consistent, we used Alloy Analyzer: we wrote the code and then we generated some worlds, checking their correctness.

Since we want to build the worlds as clear as possible, we decided not to include all the classes that are in the class diagram: in particular, we decided to omit the invite notifications and the weather conditions because they are the result of some different events (invite notifications are sent not only at the event creation, but also later and also when an event is delayed and this delay cause an overlap between two events for a participant; weather conditions changes and this modification can't be observed in alloy) and we think that it would not be completely explicative if we insert them in the alloy code, because we could not see from it the dynamics of the events.

We also decided not to include all the attributes for each signature, because it would increase the dimension of the generated worlds with information that are not essential to verify (e.g., user's name and surname, ...) and the user's password, because it is not related to the other parts analyzed. The password is subjected to characters control but it wouldn't join anything else but the user inside the model.

We have introduced a simplification about the date of an event: we decided not to create a signature with all the date fields (year, day, hour, ...), but we approximate it by using an int: this allow us to check the date constraints among the events.

7.1 Alloy code

```
open util/boolean

sig Guest {
  email: one Email
}

sig User {
  email: one Email
}

sig Email {}

sig InvitationToUser {
  relatedEvent: one Event,
  receiver: one User,
  status: one String
}

sig Event {
  organizer : one User,
  dateStart: one Int,
  dateEnd: one Int,
  typology: one Typology,
  delayed : one Bool,
  deleted : one Bool,
  weatherConditionChanged : one Bool
}

sig Message {
  event : one Event,
  messageSender : one User,
  messageReceiver : one User
}

sig Typology {}

sig InvitationToNotRegistered {
  relatedEvent: one Event,
  receiver: one Guest
}

abstract sig Notification {
  relatedEvent : one Event,
  receiver: one User
}

sig DelayNotification extends Notification {}

sig WeatherChangedNotification extends Notification {}

sig DeleteNotification extends Notification {}
```

```

//FACT

fact aDifferentEmailForEachPerson {
//All users and guests must have a different email
  all em : Email | no disjoint u1,u2 : User | u1.email = em &&
      u2.email = em
  all em : Email | no disjoint g1, g2 : Guest | g1.email = em
      && g2.email = em
  all em : Email | no u : User, g : Guest | u.email = em && g.
      email = em
}

fact aSingleInvitationForASingleEvent {
//A user can't receive two different invitations for the same
    event
  all e : Event | no disjoint i1, i2 : InvitationToUser | some
      u : User | i1.relatedEvent = e && i2.relatedEvent = e &&
      i1.receiver = u && i2.receiver = u
  all e : Event | no disjoint i1, i2 :
      InvitationToNotRegistered | some g : Guest | i1.
      relatedEvent = e && i2.relatedEvent = e && i1.receiver =
      g && i2.receiver = g
}

fact fixedStateForInvitationStatus {
//The status of an Invitation for user can only be "accepted"
    or "refused" or "noAnswer"
  all i : InvitationToUser | i.status = "accepted" or i.status
      = "refused" or i.status = "noAnswer"
}

fact noOverlappingEventsForAUser {
// A user can't confirm his participation to two overlapping
    events.
  all u : User | no disjoint e1, e2 : Event | some i1, i2 :
      InvitationToUser | e1.dateStart < e2.dateStart && e2.
      dateStart < e1.dateEnd && i1.relatedEvent = e1 && i2.
      relatedEvent = e2 && i1.status = "accepted" && i2.status
      = "accepted" && i1.receiver = u && i2.receiver = u
}

fact atLeastAnInvitationForAGuest {
//A guest is a person who is not registered but has received at
     least one invitation to an event.
  all g : Guest | some i : InvitationToNotRegistered | i.
      receiver = g
}

fact endAfterStart {
// No event can finish before starting
  all e : Event | e.dateStart < e.dateEnd
}

```

```
fact invitationProperties {
// Self-invitation is not permitted
  all i : InvitationToUser | no u : User | i.relatedEvent.organizer = u && u = i.receiver
}

fact senderAndReceiverDifferentPerson {
//A user can't send a message to himself
  all m : Message | some disjoint u1, u2 : User | m.messageSender = u1 && m.messageReceiver = u2
}

fact messageFromOnlyInterested {
//Only users who are going to partecipate to the event or who has not answered yet can send message to the event organizer.
  all m : Message | some u : User, i : InvitationToUser, e : Event | m.event = e && i.relatedEvent = e && i.receiver = u && i.status != "refused" && (m.messageSender = u or m.messageReceiver = u)
}

fact messageBetweenUserAndOrganizer {
//For every message, the organizer must be the sender or the receiver
  all m : Message | some e : Event | m.event = e && (m.messageSender = e.organizer or m.messageReceiver = e.organizer)
}

fact delayNotificationForDelayedEvent {
//A delay notification is sent only if the related event is delayed
  all d : DelayNotification | some e : Event | e.delayed.isTrue && d.relatedEvent = e
}

fact deleteNotificationForDeletedEvent {
//A delete notification is sent only if the related event is deleted
  all d : DeleteNotification | some e : Event | e.deleted.isTrue && d.relatedEvent = e
}

fact weatherChangedNotificationForWeatherChangedInEvent {
//A weather changed notification is sent only if the forecast weather conditions of the related event change
  all w : WeatherChangedNotification | some e : Event | e.weatherConditionChanged.isTrue && w.relatedEvent = e
}

fact EventStateProperty {
//We are observing the situation in a precise instant, so we assume that only one of the delayed, deleted and weather
```

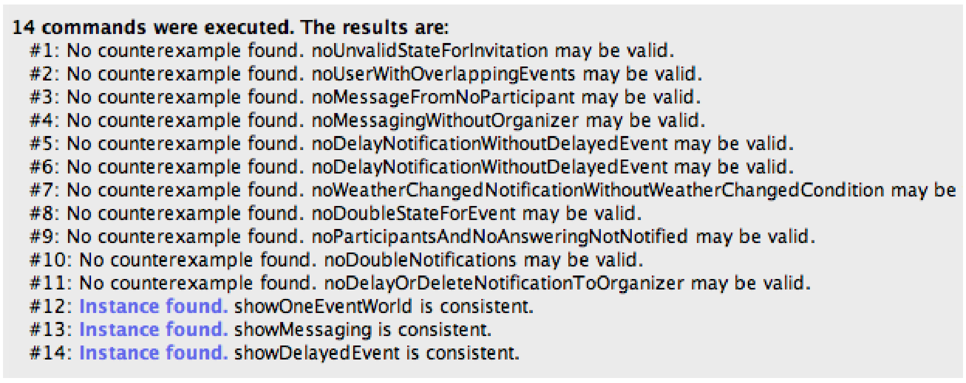
```
    condition changed states is selected in that instant.
  all e : Event | e.delayed.isTrue implies ( ! e.deleted.isTrue
      && ! e.weatherConditionChanged.isTrue)
  all e : Event | e.deleted.isTrue implies ( ! e.delayed.isTrue
      && ! e.weatherConditionChanged.isTrue)
  all e : Event | e.weatherConditionChanged.isTrue implies ( !
     e.delayed.isTrue && ! e.deleted.isTrue)
}

fact allParticipantsAndNoAnsweringsAreNotified {
//When something happens relatively to an event, all users who
    are going to partecipate or who has not answered yet must
    be notified.
  all i : InvitationToUser | ( i.relatedEvent.delayed.isTrue &&
      (i.status = "accepted" or i.status = "noAnswer") )
     implies (one d:DelayNotification | d.receiver = i.
     receiver && d.relatedEvent = i.relatedEvent)
  all i : InvitationToUser | ( i.relatedEvent.delayed.isTrue &&
      (i.status = "refused") ) implies (no d:DelayNotification
      | d.receiver = i.receiver && d.relatedEvent = i.
     relatedEvent)
  all i : InvitationToUser | (i.relatedEvent.deleted.isTrue &&
     (i.status = "accepted" or i.status = "noAnswer") )
     implies (one d:DeleteNotification | d.receiver = i.
     receiver && d.relatedEvent = i.relatedEvent)
  all i : InvitationToUser | (i.relatedEvent.deleted.isTrue &&
     (i.status = "refused") ) implies (no d:DeleteNotification
      | d.receiver = i.receiver && d.relatedEvent = i.
     relatedEvent)
  all i : InvitationToUser | (i.relatedEvent.
     weatherConditionChanged.isTrue && (i.status = "accepted"
     or i.status = "noAnswer") ) implies (one d:
     WeatherChangedNotification | d.receiver = i.receiver && d
     .relatedEvent = i.relatedEvent)
  all i : InvitationToUser | (i.relatedEvent.
     weatherConditionChanged.isTrue && (i.status = "refused")
     ) implies (no d:WeatherChangedNotification | d.receiver =
      i.receiver && d.relatedEvent = i.relatedEvent)
}

fact aSingleNotificationToEachUser {
//Only one notification has to be sent to a user for a
    determinate situation
  all u : User | no disjoint d1, d2 : DelayNotification | d1.
     relatedEvent = d2.relatedEvent && d1.receiver = u && d2.
     receiver = u
  all u : User | no disjoint d1, d2 : DeleteNotification | d1.
     relatedEvent = d2.relatedEvent && d1.receiver = u && d2.
     receiver = u
  all u : User | no disjoint d1, d2 :
     WeatherChangedNotification | d1.relatedEvent = d2.
     relatedEvent && d1.receiver = u && d2.receiver = u
}

```

```
fact organizerNotNotifiedAboutDeleteOrDelay {
//The organizer must not be notified about a delay or delete of an event, because he performs directly these decisions.
  all d : DelayNotification | no u : User | d.relatedEvent.organizer = u && d.receiver = u
  all d : DeleteNotification | no u : User | d.relatedEvent.organizer = u && d.receiver = u
}

//ASSERT

assert noPersonWithSameEmail {
  no disjoint u1, u2 : User | some em : Email | u1.email = em && u2.email = em
  no disjoint g1, g2 : Guest | some em : Email | g1.email = em && g2.email = em
  no u : User, g : Guest | some em : Email | u.email = em && g.email = em
}

check noPersonWithSameEmail

assert noMoreInvitationForASingleEventToAPerson {
  no disjoint i1, i2 : InvitationToUser | some e : Event | some u : User | i1.relatedEvent = e && i2.relatedEvent = e && i1.receiver = u && i2.receiver = u
  no disjoint i1, i2 : InvitationToNotRegistered | some e : Event | some g : Guest | i1.relatedEvent = e && i2.relatedEvent = e && i1.receiver = g && i2.receiver = g
}

check noMoreInvitationForASingleEventToAPerson

assert noUnvalidStateForInvitation {
  no i : InvitationToUser | i.status != "accepted" && i.status != "refused" && i.status != "noAnswer"
}

check noUnvalidStateForInvitation

assert noUserWithOverlappingEvents {
  no u : User | some disjoint e1, e2 : Event | some i1, i2 : InvitationToUser | e1.dateStart < e2.dateStart && e2.dateStart < e1.dateEnd && i1.relatedEvent = e1 && i2.relatedEvent = e2 && i1.status = "accepted" && i2.status = "accepted" && i1.receiver = u && i2.receiver = u
}

check noUserWithOverlappingEvents

assert noGuestWithoutInvitation {
  no g : Guest | no i : InvitationToNotRegistered | i.receiver = g
}
```

```

check noGuestWithoutInvitation

assert noEndBeforeStart {
  no e : Event | e.dateStart > e.dateEnd
}

check noEndBeforeStart

assert noSelfInvitation {
  no u : User | some i : InvitationToUser | i.relatedEvent.
      organizer = u && i.receiver = u
}

check noSelfInvitation

assert noSelfMessaging {
  no m : Message | some u : User | m.messageSender = u && m.
      messageReceiver = u
}

check noSelfMessaging

assert noMessageFromNoParticipant {
  no m : Message | some i : InvitationToUser, e : Event | one u
       : User | m.event = e && i.relatedEvent = e && i.receiver
       = u && i.status = "refused" && (m.messageSender = u or m
      .messageReceiver = u)
  no m : Message | some e : Event | all i : InvitationToUser |
      one u : User | i.relatedEvent = e && m.event = e && u not
       in i.receiver && u != e.organizer && m.messageSender = u
}

check noMessageFromNoParticipant

assert noMessagingWithoutOrganizer {
  no m : Message | some e : Event | m.event = e && m.
      messageSender != e.organizer && m.messageReceiver != e.
      organizer
}

check noMessagingWithoutOrganizer

assert noDelayNotificationWithoutDelayedEvent {
  no d : DelayNotification | some e : Event | !e.delayed.isTrue
       && d.relatedEvent = e
}

check noDelayNotificationWithoutDelayedEvent

assert noDeleteNotificationWithoutDeletedEvent {
//A delete notification is sent only if the related event is
    deleted
  no d : DeleteNotification | some e : Event | !e.deleted.
```

```
     isTrue && d.relatedEvent = e
}

check noDelayNotificationWithoutDelayedEvent

assert
    noWeatherChangedNotificationWithoutWeatherChangedCondition
    {
  no w : WeatherChangedNotification | some e : Event | !e.
      weatherConditionChanged.isTrue && w.relatedEvent = e
}

check
    noWeatherChangedNotificationWithoutWeatherChangedCondition

assert noDoubleStateForEvent {
  no e : Event | e.delayed.isTrue && (e.deleted.isTrue or e.
      weatherConditionChanged.isTrue)
  no e : Event | e.deleted.isTrue && (e.delayed.isTrue or e.
      weatherConditionChanged.isTrue)
  no e : Event | e.weatherConditionChanged.isTrue && ( e.
      delayed.isTrue or e.deleted.isTrue)
}

check noDoubleStateForEvent

assert noParticipantsAndNoAnsweringNotNotified {
  no u : User | some e : Event, i : InvitationToUser | e.
      delayed.isTrue && i.receiver = u && i.relatedEvent = e &&
       (i.status = "accepted" or i.status = "noAnswer")  &&  (
      all d : DelayNotification | d.relatedEvent = e && u not
      in d.receiver)
  no u : User | some e : Event, i : InvitationToUser | e.
      deleted.isTrue && i.receiver = u && i.relatedEvent = e &&
       (i.status = "accepted" or i.status = "noAnswer")  && (
      all d : DeleteNotification | d.relatedEvent = e && u not
      in d.receiver)
  no u : User | some e : Event, i : InvitationToUser | e.
      weatherConditionChanged.isTrue && i.receiver = u && i.
      relatedEvent = e && (i.status = "accepted" or i.status =
      "noAnswer")  && (all d : WeatherChangedNotification | d.
      relatedEvent = e && u not in d.receiver)
}

check noParticipantsAndNoAnsweringNotNotified

assert noDoubleNotifications {
  no u : User | some disjoint d1, d2 : DelayNotification | d1.
      relatedEvent = d2.relatedEvent && d1.receiver = u && d2.
      receiver = u
  no u : User | some disjoint d1, d2 : DeleteNotification | d1.
      relatedEvent = d2.relatedEvent && d1.receiver = u && d2.
      receiver = u
  no u : User | some disjoint d1, d2 :
      WeatherChangedNotification | d1.relatedEvent = d2.
      relatedEvent && d1.receiver = u && d2.receiver = u
}

check noDoubleNotifications

assert noDelayOrDeleteNotificationToOrganizer {
  no d : DelayNotification | some u : User | d.relatedEvent.
      organizer = u && d.receiver = u
  no d : DeleteNotification | some u : User | d.relatedEvent.
      organizer = u && d.receiver = u
}

check noDelayOrDeleteNotificationToOrganizer

pred showOneEventWorld{}

//RUN1
run showOneEventWorld for 6 but exactly 4 User, 1 Event, 1
    InvitationToNotRegistered, 2 Message

pred showMessaging [e : Event, i : InvitationToUser] {
  #Typology <= #e.typology
  some i : InvitationToUser | i.status = "noAnswer"
  some i : InvitationToUser | i.status = "accepted"
  some i : InvitationToUser | i.status = "refused"
}

//RUN2
run showMessaging for 5 but exactly 1 Event, 1
    InvitationToNotRegistered, 4 Message

pred showDelayedEvent [e : Event, i : InvitationToUser ]{
some i : InvitationToUser | i.status = "noAnswer"
some i : InvitationToUser | i.status = "accepted"
some i : InvitationToUser | i.status = "refused"
e.delayed.isTrue
}

//RUN3
run showDelayedEvent for 6 but 1 Event
```

Report:



```
14 commands were executed. The results are:
  #1: No counterexample found. noUnvalidStateForInvitation may be valid.
  #2: No counterexample found. noUserWithOverlappingEvents may be valid.
  #3: No counterexample found. noMessageFromNoParticipant may be valid.
  #4: No counterexample found. noMessagingWithoutOrganizer may be valid.
  #5: No counterexample found. noDelayNotificationWithoutDelayedEvent may be valid.
  #6: No counterexample found. noDelayNotificationWithoutDelayedEvent may be valid.
  #7: No counterexample found. noWeatherChangedNotificationWithoutWeatherChangedCondition may be
  #8: No counterexample found. noDoubleStateForEvent may be valid.
  #9: No counterexample found. noParticipantsAndNoAnsweringNotNotified may be valid.
  #10: No counterexample found. noDoubleNotifications may be valid.
  #11: No counterexample found. noDelayOrDeleteNotificationToOrganizer may be valid.
  #12: Instance found. showOneEventWorld is consistent.
  #13: Instance found. showMessaging is consistent.
  #14: Instance found. showDelayedEvent is consistent.
```

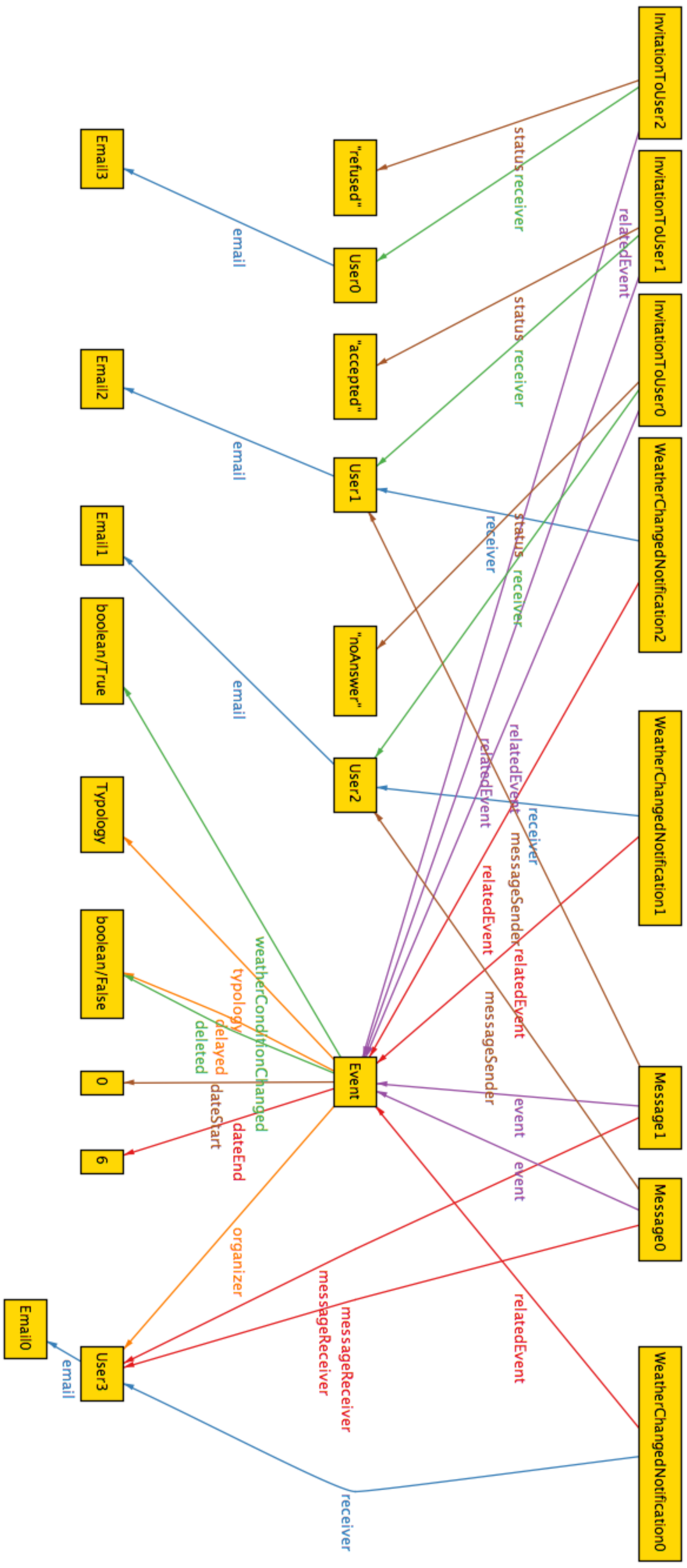
7.2 Worlds Generated

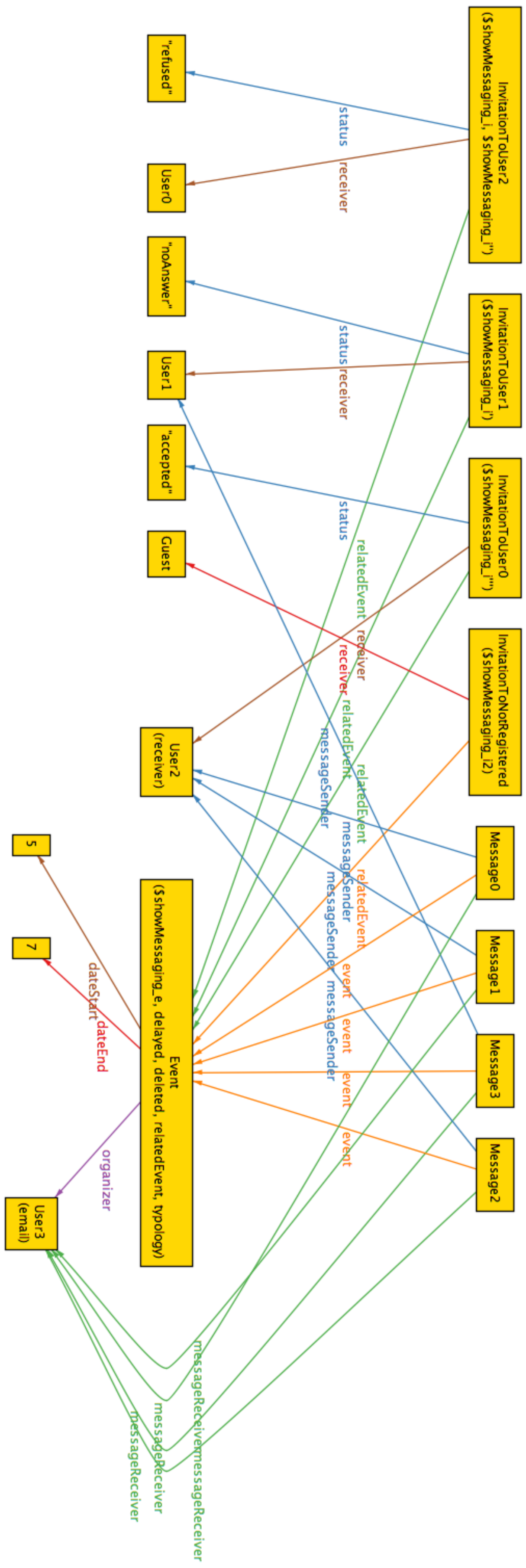
We generated some worlds, which can be seen below. It is possible to check that:

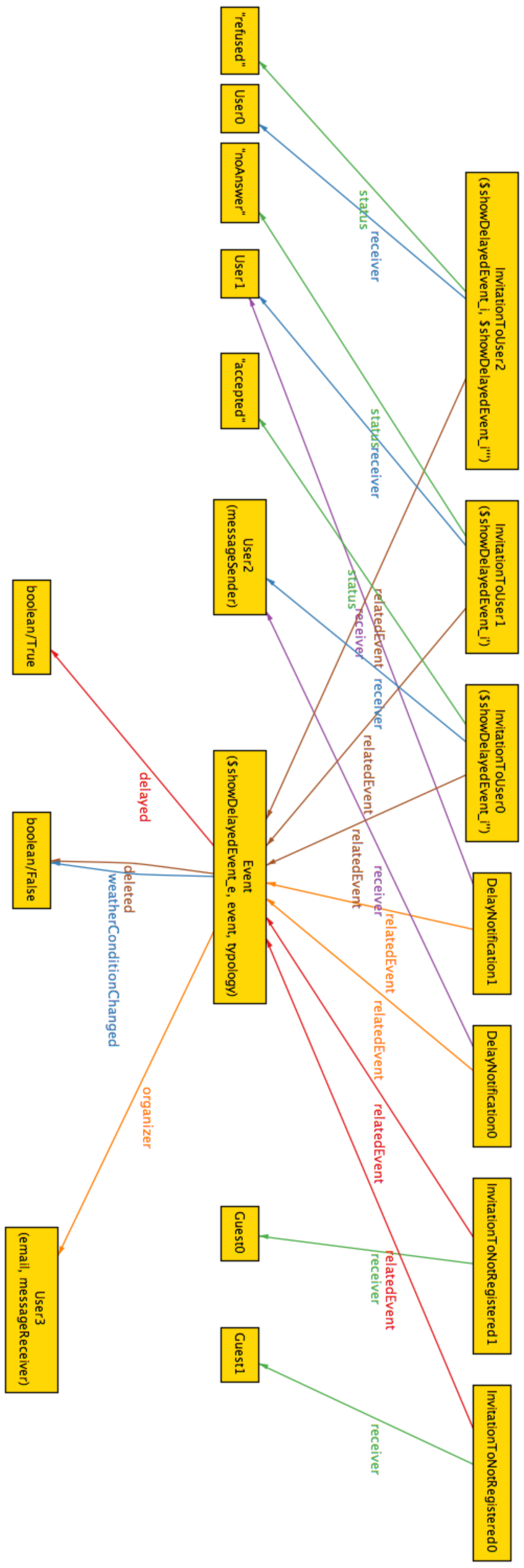
- Every user/guest has a personal email, which is different from all the other ones.
- Every event has exactly one user as organizer and can be many invitations related to it, but also some notifications, in case of delay or delete of an event or in case of a modification in weather conditions.
- A user can't receive an invitation to an event of which he is the organizer.
- Delay notifications, delete notifications and weather changed notifications are sent only to users who has positively answered to the invitation or hasn't replied yet, while no notifications are sent to users who are not going to participate (so they are not interested in knowing anything else about the event) and to guest (they are not registered, so they can't exploit MeteoCal services). Obviously, no delay or delete notification is sent to the organizer, because he is the person who decides to apply that modification to the event, so it is not necessary to inform him about these changes.
- A user can receive at most one notification for a single modify related to an event.
- A user/guest can receive at least one invitation for the same event.
- Every invitation must be related to an event and has exactly one receiver, who can be a guest or a user: in the second case, also an

invitation state (exactly one) is associated to the invitation, in order to understand if the user is going to participate or not, or if he hasn't decided yet; a guest can't answer to an invitation without signing up before, so every state related to an invitation not to registered can't be different from the "noAnswer" one, so it would be useless to insert that information.

- Only users who are interested in the event (so they have answered positively or they have not replied yet to the invitation) can send a message to the organizer asking for other information. There can't be a message in which the organizer is not the sender neither the receiver.


InvitationToUser2
InvitationToUser1
InvitationToUser0
WeatherChangedNotification2
WeatherChangedNotification1
Message1
Message0
WeatherChangedNotification0
"refused"
User0
"accepted"
User1
"noAnswer"
User2
Event
Email3
Email2
Email1
boolean/True
Typology
boolean/False
0
6
User3
Email0
status
receiver
relatedEvent
email
receiver
messageSender
messageReceiver
event
weatherConditionChanged
typology
delayed
deleted
dateStart
dateEnd
organizer


InvitationToUser2 ($showMessaging_i, $showMessaging_i'')
InvitationToUser1 ($showMessaging_i')
InvitationToUser0 ($showMessaging_i''')
InvitationToNotRegistered ($showMessaging_i2)
Message0
Message1
Message3
Message2
"refused"
User0
"noAnswer"
User1
"accepted"
Guest
User2 (receiver)
Event ($showMessaging_e, delayed, deleted, relatedEvent, typology)
5
7
User3 (email)
status
receiver
relatedEvent
messageSender
event
dateStart
dateEnd
organizer
messageReceiver


InvitationToUser2 ($showDelayedEvent_i, $showDelayedEvent_i'')
InvitationToUser1 ($showDelayedEvent_i')
InvitationToUser0 ($showDelayedEvent_i''')
DelayNotification1
DelayNotification0
InvitationToNotRegistered1
InvitationToNotRegistered0
"refused"
User0
"noAnswer"
User1
"accepted"
User2 (messageSender)
Event ($showDelayedEvent_e, event, typology)
Guest0
Guest1
boolean/True
boolean/False
User3 (email, messageReceiver)
status
receiver
relatedEvent
delayed
deleted
weatherConditionChanged
organizer

8. Conclusion

As we said before, we have analyzed also advanced requirements (which are not expressly requested) because we think that they would improve the usability of MeteoCal. Since we don't have much time to develop the system, in the following steps of development we will analyze only the expressly requested requirements and so we will not include some of the advanced features (e.g. messaging, send invitations to participate to event to guests, send invitation to sign up in the system to guests, add users to favourites).

9. Used Tools

The tools we used to create this RASD document are:

- TexStudio: to redact and format this document;
- Signavio Platform: to create Use Case Diagram;
- VisualParadigm: to create Sequence Diagram;
- Moqups: to create the UI sketches;
- Alloy 4.2;
- StarUML: to create the Class Diagram.