Politecnico di Milano

Scuola di Ingegneria Industriale e dell'Informazione

Computer Science and Engineering

Software Engineering 2 Project – A.Y. 2014/15

Requirements

Analysis

Specification

Document

Authors

Francesco Lattari (838380)

Alessandro Rimoldi (835506)

Summary

1. Introduction

1.1 Description of the given problem

The project that we are going to design and implement is MeteoCal that is a new weather based online calendar whose purpose is to provide a system for creating and managing events. The registered user will be able to create, update, or delete an event with the ability to add more participants and specifying the place, date and time. The special feature of this calendar is to provide weather forecasts during the creation of events that can be of two types, indoor or outdoor, indicating possible bad weather conditions on the dates planned for the second type of events. Finally the organizer can define if the event should be public or private.

1.2 Goals

The system shall implement the following key features:

- Registration of a new user in the system;
- Login and logout of the registered users;
- Allowing a user to search and find other users;
- Allowing a user to invite a friend to join MeteoCal;
- Creating, updating and deleting events;
- Enriching a saved event with weather forecast information if available;
- The possibility to invite other users to an event;
- The possibility to request to the event organizer to join his event;
- The possibility for a user to get out from an event;
- Viewing notifications when the users log into the system;
- Give participants all the necessary information about an event such as the place, date and time;
- Sharing with the participants personal contact information;
- Making your calendar public whose events can be public or private;
- Notifying all event participants one day before the event in case of bad weather conditions for outdoor event;
- Suggesting to the organizer, in case of bad weather, the closest sunny day with a three days' notice.

1.3 Domain properties

These are our suppositions about the analysed world:

- A user should accept an invitation if he can really participate;
- Before posting an event, the organizer should ensure that the location chosen for the event is available and, in case of outdoor event, he should ensure that the weather conditions are favourable to carry out the event;
- An organizer should never create fake events with the only purpose of making fun of other users;
- Only registered users can create and managing events, view the calendar of other users and their personal information;
- Only the organizer is able to update or delete an event while invited users can only accept or decline the invitation or get out from it in a second moment;
- Users should provide truthful information about their contact like a reachable phone number or valid e-mail address.

1.4 Glossary

There are some terms used in the document that have to be explained such as:

- **User:** for user we mean a person who is already registered in the system and can use all its functionality.
- **Guest:** a guest is a person not registered and can only sign in. A guest doesn't have the possibility to search and view other users and their calendar, but he can view the presentation of the service and analyze its functionalities.
- **Event:** an event is for us a thing composed by a place, date, time, description and optionally by a time limit for participants to delete themselves from it and by a maximum number of participants. An event can be of two type: indoor or outdoor and it can also be defined as public or private. The public one has its information visible to other users who can request the participation if it's possible while the latter is externally visible on the calendar but without detailed information.
- **Bad weather condition:** in our interpretation bad weather condition consist in the presence of rain, thunderstorm, snow or low temperature (≤10 °C for example).
- **Notification:** notifications are warnings for the users. They can be of six types: event invitation, participation request, participation confirmed, bad weather alert, event update or deleted and quit from the event (*addressee: Organizer*).

1.5 Assumptions

There are some aspects not clear in the specification of the system so we propose these our assumptions:

- A user can participate in multiple events on the same day with the condition that they do not overlap.
- When a user accept an invitation the system saves the event on his calendar and starts showing it.
- The organizer can establish a time limit for users for removing their participation to the event. After that time, it's not possible getting out from it, even if the user can always not physically participate to the event but this consists in a bad behaviour of the person.
- The organizer can update the event in every moment but he can delete it at most one day before.
- The organizer can also establish a maximum threshold of participants in case of public event and, if it is reached, the system no longer allows to join it.
- The system provides timely weather information before and after the creation of an event.
- The system should allow any registered user to ask the organizer to participate in event declared as public. The organizer will receive the requests and he will decide to accept or to decline it.

1.6 Proposed system

In order to give to users the described functionality we think that a web platform is the most appropriate. Basically because the web solution offers a system more reachable and a faster way for users to keep up to date on the events in which they are enrolled.

All the users will have their personal page with their calendar and they will have a page dedicated to the notifications received. Users will also be able to modify their contact information at any time through an appropriate section and, thanks to a search bar, they will be able to search other users on the system viewing their calendars (if public declared).

In order to help organizer in inviting users in a faster way, the system will suggest to him a list of the users frequently invited.

The subsequent development of a mobile version would provide a service easily accessible through smartphones and tablets.

1.7 Stakeholders

The financial stakeholder of our project is the X software house that commit us this work in order to give to people a new system that can help them in managing their events with the support of a weather forecasts system.
The project stakeholder is our Professor of Software Engineering 2 at Politecnico di Milano who is interested in having a working system in order to evaluate it and understand if we learned the fundamental concepts of the course.
There is also the community stakeholder composed by the people interested in the development of this project. For example the event organizers for which our system could be a valid help in their work, making it easier and faster.
The service will be useful also for that people who are very active and love to have a day without a break. With MeteoCal they will find an easy way to stay up to date on events in their area and they will be able to easily keep track of their commitments.

2. Actors

The actors of *MeteoCal* are essentially two: an *unregistered user (guest)* and a *registered user*, which can be ideally distinguished, depending on the circumstances, in *invited user* and *organizer*. On the basis of this distinction, they can take advantage of specific features. In particular:

- **Unregistered User (Guest)**: a person that has not registered and can only view the list of functionalities provided by the system and convince himself that MeteoCal will be very useful for him. A guest can sign up to MeteoCal not only discovering the service by searching on internet but also receiving by mail an invitation to join to MeteoCal by a registered user.
- **Registered User**: a person that has registered and can join or create events. He can also make his calendar public, allowing the other registered user to see when he's free or busy. In particular, if an event is defined as public, a registered user can see all the details and apply a request to join the event.
  - **Invited User**: a user that has received an invitation to an event and can accept or decline the proposal.
  - **Organizer**: a user that has created an event and can invite any number of registered users. He can also delete the event or update the information

about it, setting when and where it will take place, whether the event will be indoor or outdoor.

3. Requirements

The requirements of MeteoCal are directly linked to the goals and to the domain analysis of the previous sections. In this chapter we are going to describe the functional and the non functional requirements, focusing, in the first case, on the features that each actor can perform, while, in the second case, we will analyze how the MeteoCal service can be improved by some design and technical choices.

3.1 Functional requirements

Here is the functional requirements concerning each defined actor:

- **Guest** → He can:
  - View the description of the service;
  - Sign up.

The system should provide a "sign up" button and a step by step procedure of registration in which the guest has to insert some mandatory information (name, surname, mail and password).

- **Registered User** → Avoiding in this section the conceptual distinction between *Invited User* and *Organizer*, we can simply say that a *Registered User* can:
  - Log in and log out;
  - Invite people to join MeteoCal;
  - Search a user, view his profile and his calendar, if public;
  - View the details of an event and ask to join it, if public;
  - View his own profile and modify it;
  - Receive invitations to events and accept/reject them;
  - Remove himself from an event;
  - Create/Update/Delete an event and invite other users to join it;
  - Receive notification about changes of climatic conditions or information and parameters of an event.

The system should provide a "log in" and "log out" button, a search bar in which the user can find other people and a step by step procedure to guide the user to create events and invite people. There will be also an area in which the user can see and manage the invitations to events and check all the notification. On his own profile a user should have available a "modify profile" button, a link to his own calendar in which can see his events and remove himself from those.

3.2 Non functional requirements

This section points out the general characteristics that the MeteoCal system will provide to its users with respect to issues of quality and usability.

- **Reliability:** MeteoCal must guarantee the persistence of the data entered by users. For example the system will never lost changes to user profile or changes in details of an event.
- **Security:** The system must provide an authentication system that is able to mask to each user all the not allowed operations and, in parallel, to ensure the proper performance of the functions available to authorized users.
- **Accessibility:** The use of the system must be clear and intuitive, and for each type of user every available function must be implemented so as to provide a painless and immediate access.
- **Performance:** The timing of access to the system must be kept reasonably low, and MeteoCal must be quickly responsive to user input.
- **User Interface:** The system uses graphical interface easy to use allowing inexperienced users to work with all the functionality of the system. The interfaces are differentiated according to "role" assumed by the user in that specific case. Each interface will provide the access only to the features that compete to the user who is logged in, hiding and showing the right buttons. This interface uses a graphical structure in which there are buttons, links, windows, dialog boxes and data entry areas.
- **Documentations:** In order to illustrate the development of the project will be made of the following documents:
    - *Requirements Analysis Specification Document* (RASD), addressed to the customer, end users, analysts and developers;
    - *Design Document* (DD), for developers and analysts;
    - Documentation of the source code and test cases.

4. Specification

In this section we explore some specifics regarding MeteoCal. They are summarized in the following points:

- The home page should be structured so that a guest has an overview of the system and gets information regarding its functionality;
- The personal page will be simple and intuitive and the user can have just a click away his own calendar and notifications received;
- During the adding of participants the system suggests to the organizer a list of the users frequently invited in order to allow him to add users in a faster way;
- A user can search for another user by entering the name into the search bar and the system will return a list of users that match the name he is looking for. This research can also be performed via e-mail;
- In order to make the research for other users more intuitive, every user has the possibility to add a personal image to his own profile;
- Any request to participate will have to wait for confirmation of acceptance by the organizer. If the request will remain pending until the event date, it will be automatically deleted from the system;
- Finally, when creating an event, organizer has the option to add a brief description about the event details.

5. Scenarios

To describe some possible scenarios, we imagine that there are three people (actors): *Richard*, who has not yet registered to MeteoCal, *Walter* and *Frank*, who instead are registered users of MeteoCal instead.
Note: the scenarios are not connected to each other, so the events are "reset" each time with the beginning of a new story.

5.1 First scenario

Richard wants to organize a meeting among his old schoolmates but the idea of making a lot of phone calls, sending emails and trying to manage the commitments of everybody annoys him. So he starts searching on internet a way to manage these problems and find a system to organize the event. MeteoCal is among the results shown by the search engine. He reads the features of the service and decides to

have a try. So he registers and invites his schoolmates to join the platform. Once done these steps, Richard creates the event, set the date, the place and marks the event as “indoor” and “private”. Once the event invitation was sent to his schoolmates, he doesn’t have nothing to do except waiting for their response.

5.2 Second scenario

Walter accesses to MeteoCal and receives a notification of invitation to join an event organized by Frank. The event will be held November 5, 2014 at 18:30 at the Wellington Country Park. Walter has no commitments for the date and then decides to participate. At a time when he agrees, there are only three days to the specified date and climatic conditions seem good. The day before the event the weather conditions deteriorate suddenly and is indicated rain. The system alerts everybody about the inconvenience and the organizer decide to cancel the event instead of move it to another day.

5.3 Third scenario

Richard has registered to MeteoCal and decides to enter some personal information to his profile. Then add an avatar, a number of home phone and a cell phone number. In this way it can be more easily found in case he decides to organize an event and his guests want to have some explanations.

5.4 Fourth scenario

Walter is organizing a sporting event at Central Park: a short marathon of 10 kilometres. The race will take place in about a month and Walter sends invitations well in advance, in the hope that a good number of people will join the event, however, the weather forecasts are not yet available. Walter marks the event as public, so that anyone can view the details; he also reports to the guests to bring a document attesting to the good health of the person. About a week before the event, weather conditions worsen, but a light rain would not be a problem. Three days before the race, instead, the system reports thunderstorms. The system suggests to Walter to move the race five days later, when it is expected to overcast skies and mild temperatures. The organizer updates the event date to the suggested one and the system warns the guests of this change. Unfortunately, the day before the race is reported a light rain and Walter decided to not move the event again but update the event information suggesting to bring raincoats.

5.5 Fifth scenario

Walter is looking at the Frank's calendar to see his commitments on Monday and find out if at the evening is free to suggest a tennis match. Unfortunately Frank is engaged: from 21.00 to 22.00 is playing futsal at the city sports centre. Seven people have already signed up to the event and Water, rather than staying at home doing nothing, decides to send a request to join the game to the organizer.

5.6 Sixth scenario

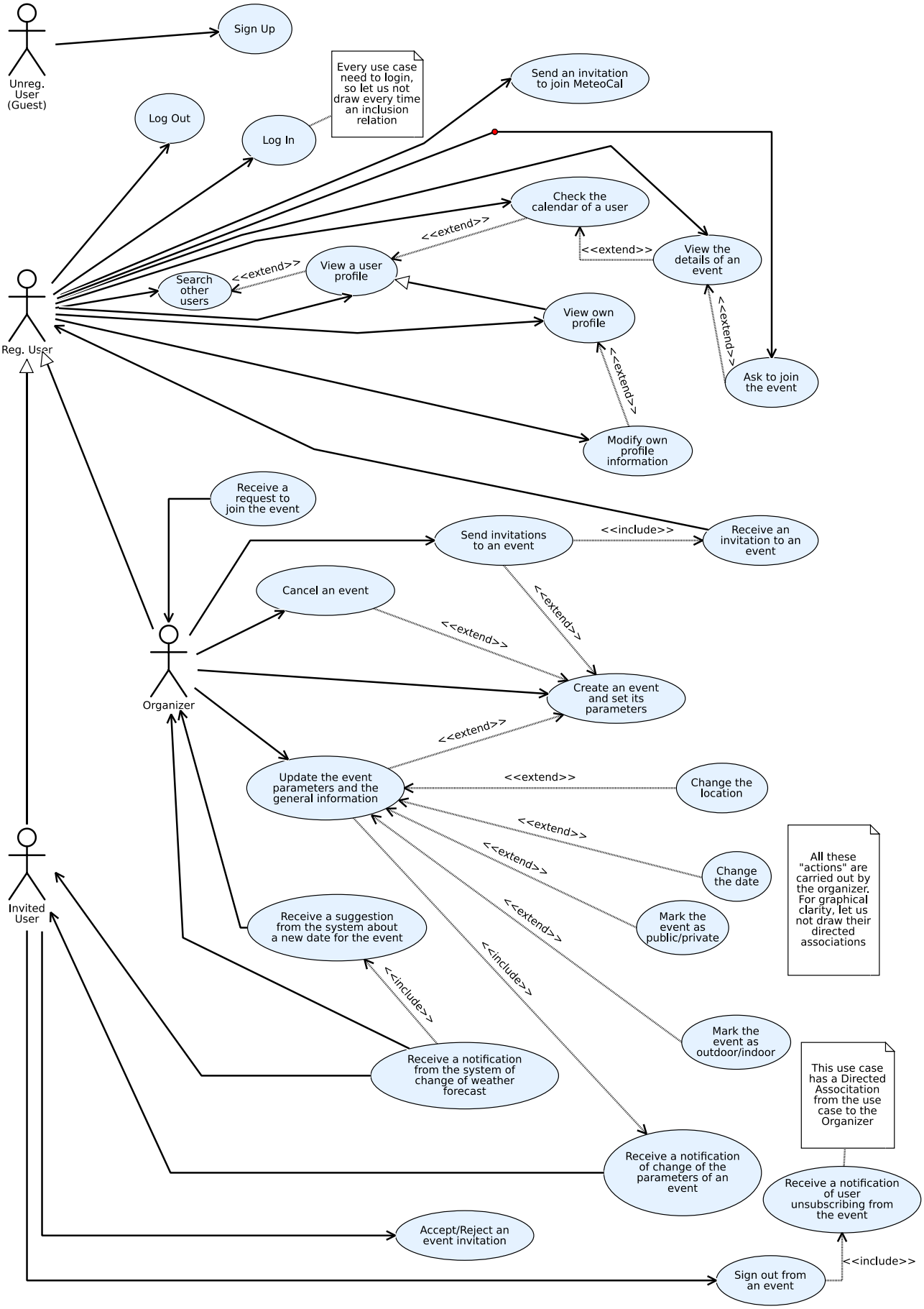
Frank discovers he has another commitment on Sunday and he can't participate to the movie night organized by Walter in his basement. So log on to MeteoCal, select the event from his calendar, cancel his subscription and Walter is notified immediately by the system.

6. UML Models

6.1 Use Case diagram

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

- Sign up;
- Log in;
- Log out;
- Send an invitation of registration to MeteoCal;
- Create an event and set its parameters (date, place, indoor or outdoor, public or private);
- Update the event parameters and the general information;
- Cancel an event;
- Send invitations to an event;
- Receive an invitation to an event;
- Receive a notification of change of the parameters of an event;
- Receive a notification of unsubscribing from an event (*only for the organizer*);
- Receive a notification from the system of change of weather forecast;
- Receive a suggestion from the system about a new date for the event;
- View a user profile;
- Modify own profile information;
- Accept/Reject an event invitation;
- Sign out from an event;
- Check the calendar of a user;
- View the details of an event (if public);
- Apply for registration to an event;


Unreg. User (Guest)
Sign Up
Log Out
Log In
Every use case need to login, so let us not draw every time an inclusion relation
Send an invitation to join MeteoCal
Check the calendar of a user
View the details of an event
Reg. User
Search other users
View a user profile
View own profile
Ask to join the event
Modify own profile information
<<extend>>
<<include>>
Receive a request to join the event
Send invitations to an event
Receive an invitation to an event
Cancel an event
Organizer
Create an event and set its parameters
Update the event parameters and the general information
Change the location
Change the date
Mark the event as public/private
Mark the event as outdoor/indoor
Invited User
Receive a suggestion from the system about a new date for the event
Receive a notification from the system of change of weather forecast
Receive a notification of change of the parameters of an event
Accept/Reject an event invitation
Receive a notification of user unsubscribing from the event
Sign out from an event
All these "actions" are carried out by the organizer. For graphical clarity, let us not draw their directed associations
This use case has a Directed Associtation from the use case to the Organizer

6.2 Use Case description

In this section we will describe some of the most important use cases listed in the first part of the section and drawn in the Use Case Model. In these descriptions, there may be details which in reality will be realized in different ways but in essence the flow of events will be the same.

NAME	Sign Up
ACTORS	Unregistered User (Guest)
ENTRY CONDITIONS	The user doesn't have a MeteoCal account.
FLOW OF EVENTS	• The user is on the MeteoCal home page; • The user clicks on the "sign up" button; • The system initiates the registration procedure; • The guest fulfils the input form and clicks "done"; • The system validates the registration.
EXIT CONDITIONS	The information about the new user are stored into the database. Now the guest can log in.
EXCEPTIONS	The guest doesn't compile the mandatory fields or inserts an email already present in the database. The system shows an error message.

NAME	Log In
ACTORS	Registered User
ENTRY CONDITIONS	The user has already created a MeteoCal account.
FLOW OF EVENTS	• The user is on the MeteoCal home page; • The user clicks on the "log in" button; • The system loads a new page in which the user has to insert his email and password; • The user inserts his credentials; • The user clicks "done"; • The system shows the profile page of the user.
EXIT CONDITIONS	The registered used is now logged in MeteoCal and he can access to all the functionality provided by the system.
EXCEPTIONS	The information inserted in the form by the user are wrong and the system shows an error message.

NAME	Log Out
ACTORS	Registered User
ENTRY CONDITIONS	The user has already performed a log in to the system.
FLOW OF EVENTS	• The user clicks on the “log out” button; • The system closes his session and load the homepage of MeteoCal
EXIT CONDITIONS	The user becomes a guest and he can perform the log in again
EXCEPTIONS	///

NAME	Check the calendar of a user
ACTORS	Registered User
ENTRY CONDITIONS	The user is logged to MeteoCal.
FLOW OF EVENTS	• The user goes on the profile page of another user; • The user clicks on the button “calendar”; • The system shows the user calendar and all his commitments, hiding the information about the event marked as “private”.
EXIT CONDITIONS	///
EXCEPTIONS	The calendar of the user isn’t public.

NAME	Modify own profile information
ACTORS	Registered User
ENTRY CONDITIONS	The user is logged to MeteoCal and is into his profile page.
FLOW OF EVENTS	• The user clicks on “modify” button; • The system loads a page in which the user can modify the profile information, filling again some input forms; • The user modifies what he wants; • The user clicks on “done”; • The system reloads the profile page with all the new information.
EXIT CONDITIONS	The new information are stored into the database.
EXCEPTIONS	• The user leaves the page without clicking on “done” button. All the changes are lost; • The user inserts a bad formed information in an input form. The system will show an error message; • The user has emptied some mandatory fields. The system will show an error message.

NAME	Create an event and set its parameters
ACTORS	Registered User
ENTRY CONDITIONS	The user is logged to MeteoCal.
FLOW OF EVENTS	• The user clicks on “create an event”; • The system loads a page in which the user has to fill some information; • The user inserts the mandatory information: name of the event, date, location, public/private and indoor/outdoor; • The user clicks on “done”;
EXIT CONDITIONS	• The Registered User becomes an “Organizer”; • The system loads a page in which the organizer is invited to send invitations to the other users;
EXCEPTIONS	• The user leaves the page without clicking on “done” button. All the changes are lost; • The user inserts a bad formed information in an input form. The system will show an error message; The user has emptied some mandatory fields. The system will show an error message.

NAME	Cancel an event
ACTORS	Organizer
ENTRY CONDITIONS	The user is logged to MeteoCal and is the organizer of the event that is going to be deleted.
FLOW OF EVENTS	• The user clicks on his calendar; • The user select the designated event; • The user clicks on “delete event” (the button isn’t visible if the user isn’t the organizer of that event); • The system goes back to the calendar and the event is no more visible.
EXIT CONDITIONS	• The event is deleted from the database; • The system notifies all the participants about the annulment.
EXCEPTIONS	The organizer tries to cancel the event when it is no longer possible. The system shows an error message and the request is not completed.

The two use cases “Receive an invitation to an event” and “Accept/Reject an event invitation” are described together because one is the consequence of the other, but we can’t unified them in one use case because it is always possible to accept or reject an invitation at a later date.

NAME	Receive an invitation to an event Accept/Reject an event invitation
ACTORS	Registered User
ENTRY CONDITIONS	The user is logged to MeteoCal.
FLOW OF EVENTS	• The system alerts the user with a notification event invitation; • The user clicks on the “notification” button; • The user clicks on the event; • The system loads a page containing the details of the event and two buttons: “accept” and “decline”; • The user makes his choice; • The system refresh the page hiding the two button.
EXIT CONDITIONS	In case of accepting the invitation: • The user is now registered to the event; • The Registered User becomes an “Invited User”; • The system will alert him about any changes of the event; • The system adds the event to the user calendar.
EXCEPTIONS	• The user leaves the page without making a choice. The system maintains the invitation in memory and the user can perform the selection later; • The user accepts the invitation too late, when the event is full.

NAME	Update the event parameters and the general information
ACTORS	Organizer
ENTRY CONDITIONS	The user is logged to MeteoCal and is the organizer of the event that is going to be modified.
FLOW OF EVENTS	• The user clicks on his calendar; • The user select the designated event; • The user clicks on “modify event” (the button isn’t visible if the user isn’t the organizer of that event); • The system loads a page in which the user can modify one or more of these elements: date, location, public/private and indoor/outdoor; • The user press “done”; • The system goes back to page containing the details of the event just updated.
EXIT CONDITIONS	• The system saves the new details of the event in the database; • The system alerts all the participants about the changes.
EXCEPTIONS	• The user leaves the page without clicking on “done” button. All the changes are lost; • The user inserts a bad formed information in an input form. The system will show an error message; • The user has emptied some mandatory fields. The system will show an error message.

NAME	Sign out from an event
ACTORS	Invited User
ENTRY CONDITIONS	The user is logged to MeteoCal and is an invitee of the event from which is going to sign out.
FLOW OF EVENTS	• The user goes to the page containing his calendar; • The user selects the designated event; • The user clicks on “sign out”; • The system loads the refreshed page containing the calendar. The event is no more visible.
EXIT CONDITIONS	• The system saves in the database the new list of participants that no longer contains the invitee. • The Invited User becomes a Registered User; • The system warns the organizer about the change in participants list.
EXCEPTIONS	///

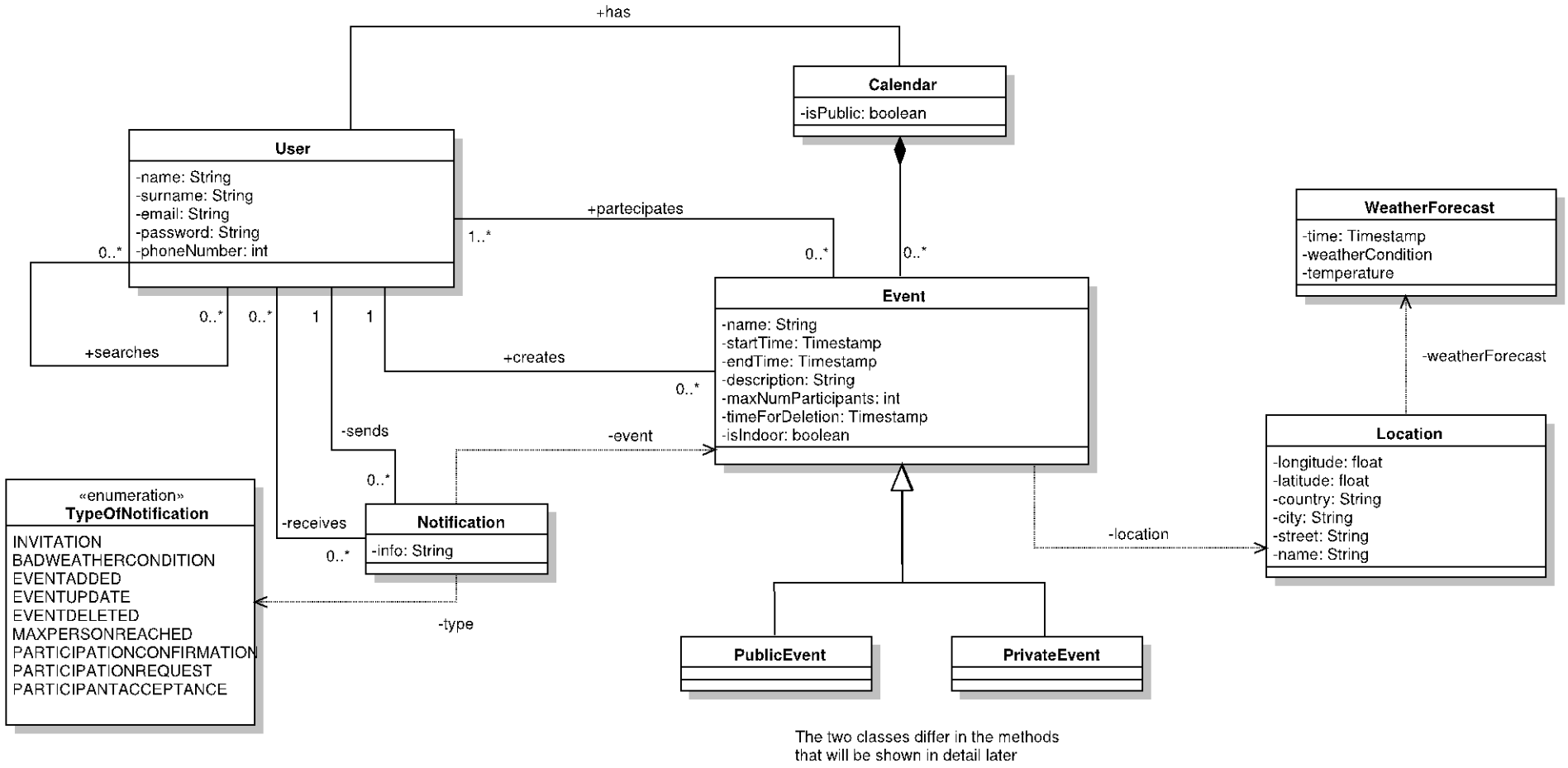
NAME	Receive a suggestion from the system about a new date for the event
ACTORS	Organizer
ENTRY CONDITIONS	The user is logged to MeteoCal and is the organizer of the event. The climatic conditions of the event are changed (deteriorated) and the organizer has already received the notification by the system for this change.
FLOW OF EVENTS	• The user clicks on the notification containing the suggestion received by the system; • The user decides if the date is appropriate or not: if it is, the organizer clicks on “accept”, otherwise goes to the event page and click “modify event” or “delete event” (*see the use case already described about changes in event details*).
EXIT CONDITIONS	• In case the user accepts the suggestion, the system save the new date into the database and alerts all the participant about the change; • In the other case, we can’t define here the exit conditions.
EXCEPTIONS	///

This use case has two different behaviours depending on whether the actor is any invited user or the organizer: in the first case the notification comes one day before the event, in the second case three days before, but the flow of events is the same.

NAME	Receive a notification from the system of change of weather forecast
ACTORS	Invited User and Organizer
ENTRY CONDITIONS	The climatic conditions of the event are changed (deteriorated) and missing three or less days to the event
FLOW OF EVENTS	First path: • The organizer logs in to MeteoCal; • The system notify him that the climatic conditions of the event are changed. The flow of events follows the steps described in "Receive a suggestion from the system about a new date for the event" use case; Second path (missing one day to the event): • The user logs in to MeteoCal; • The system notify him that the climatic conditions of the event are changed; • The user can sign out from the event going on the details page of the event (see the specific use case);
EXIT CONDITIONS	///
EXCEPTIONS	///

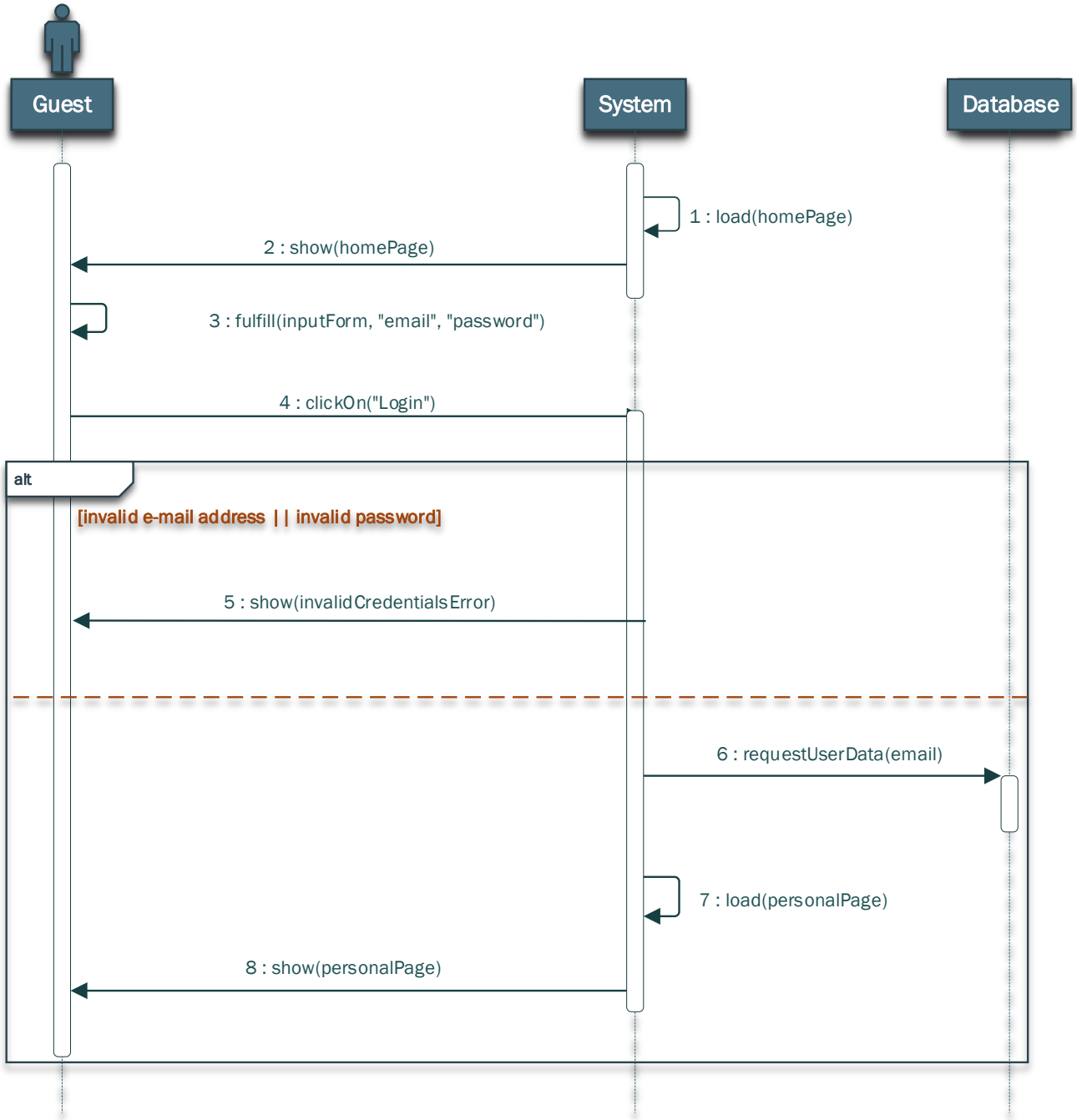
6.3 Class diagram

The one presented here is a very general idea of how it could present the class diagram, created solely on the basis of the specifications and use cases. A more detailed and faithful to implementation version will be provided later.

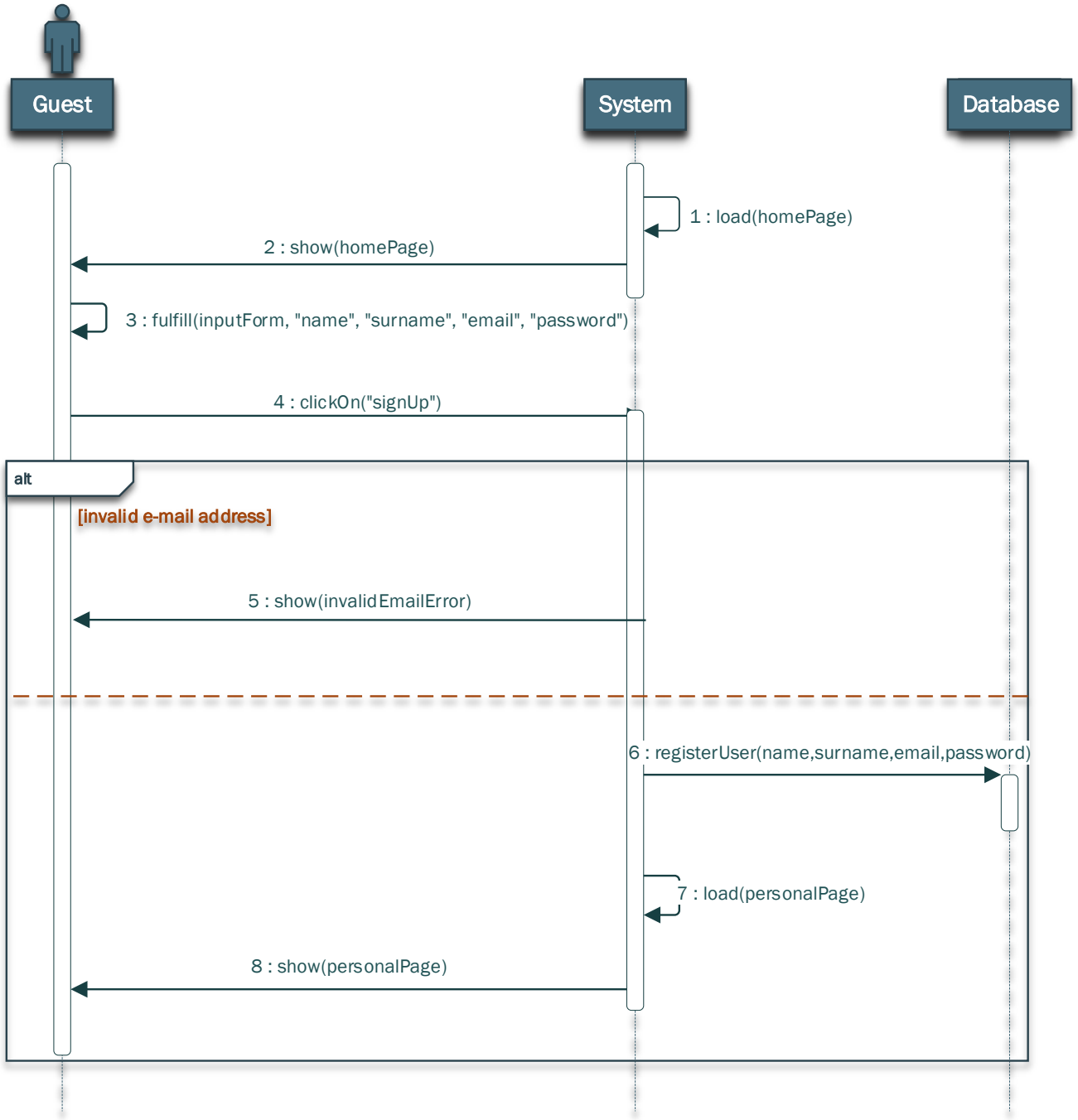


6.4 *Sequence diagrams*

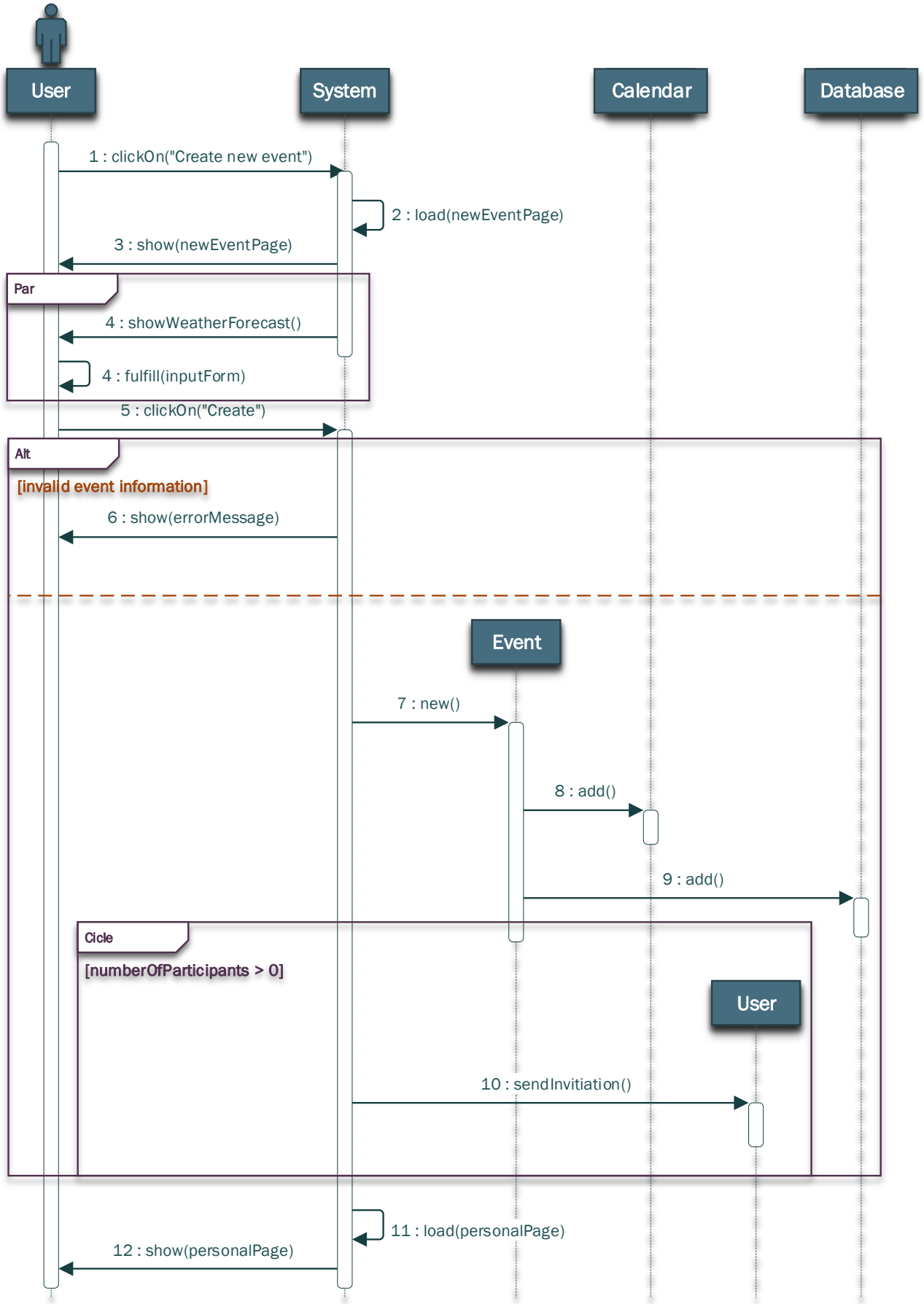
6.4.1 Log In



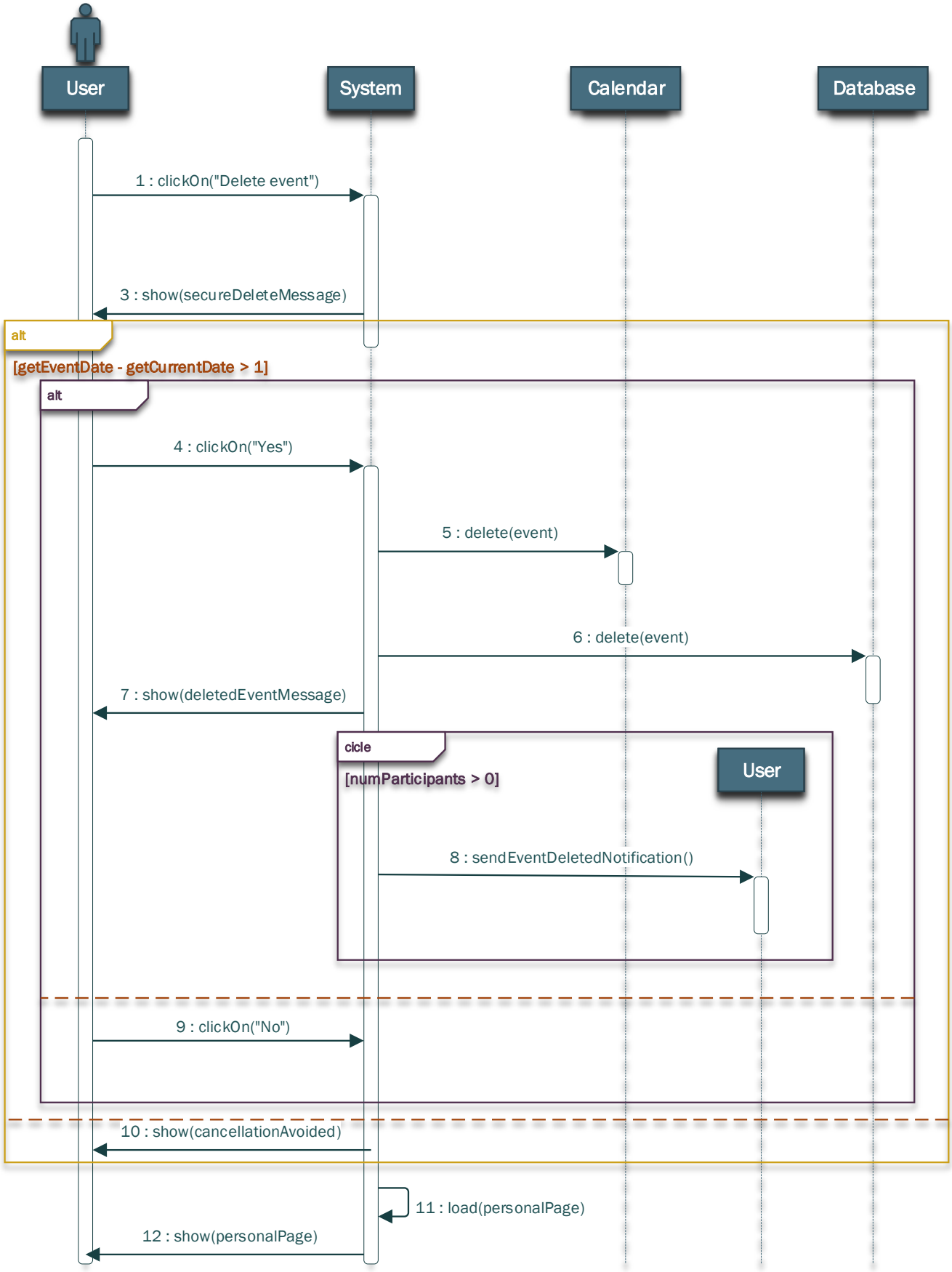
6.4.2 Sign Up


Guest
System
Database
1 : load(homePage)
2 : show(homePage)
3 : fulfill(inputForm, "name", "surname", "email", "password")
4 : clickOn("signUp")
alt
[invalid e-mail address]
5 : show(invalidEmailError)
6 : registerUser(name,surname,email,password)
7 : load(personalPage)
8 : show(personalPage)

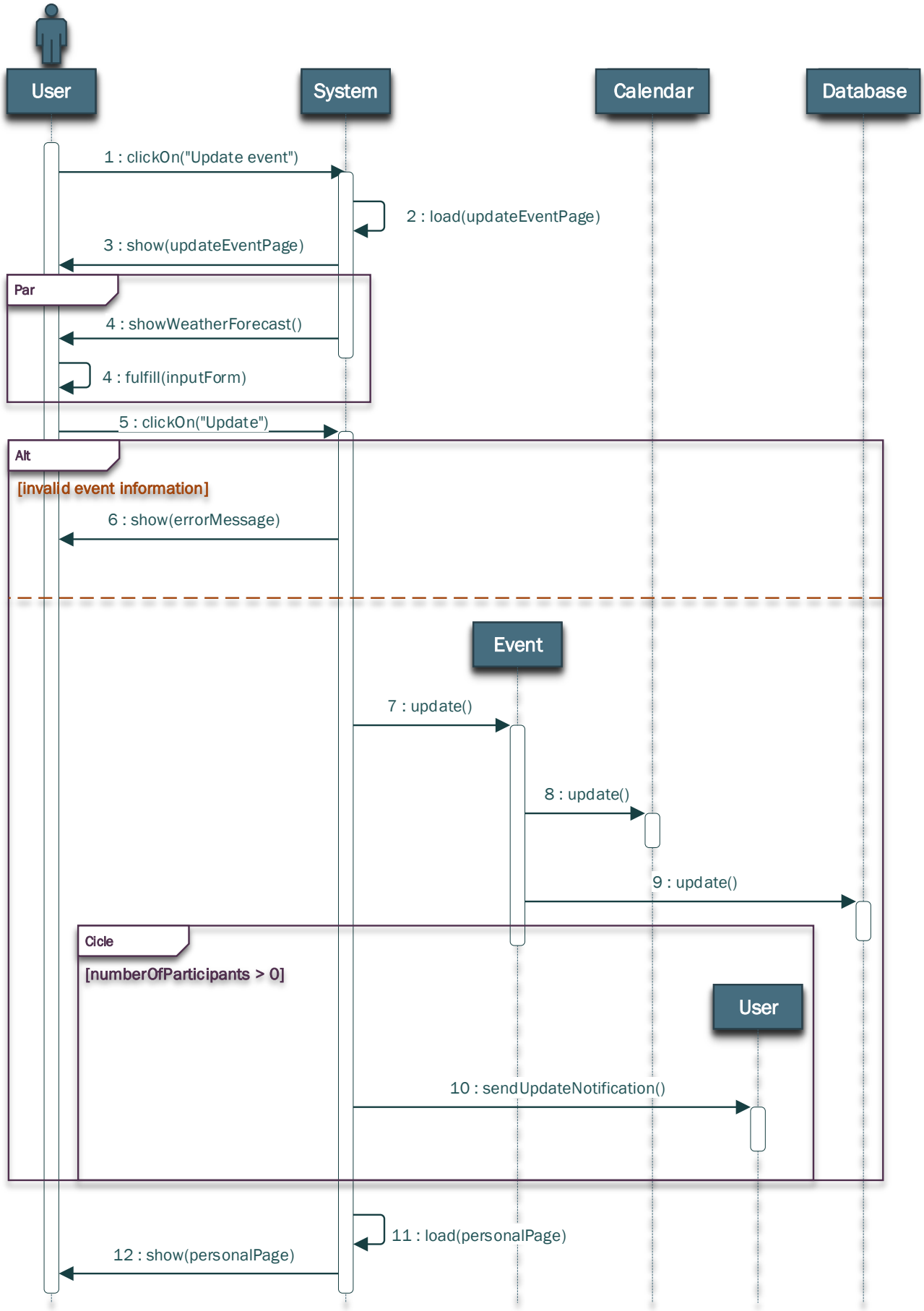
6.4.3 Create Event



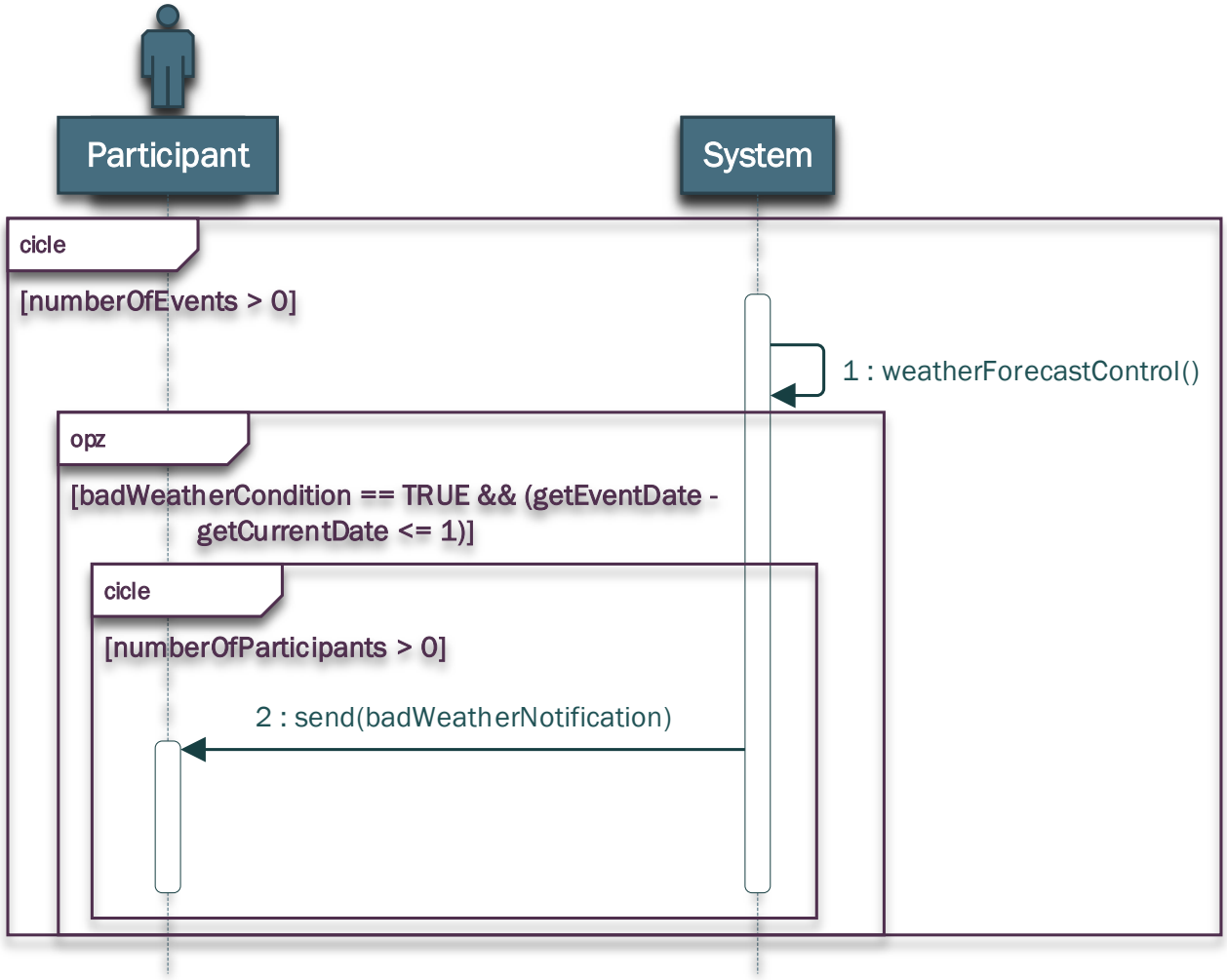
6.4.4 Delete Event



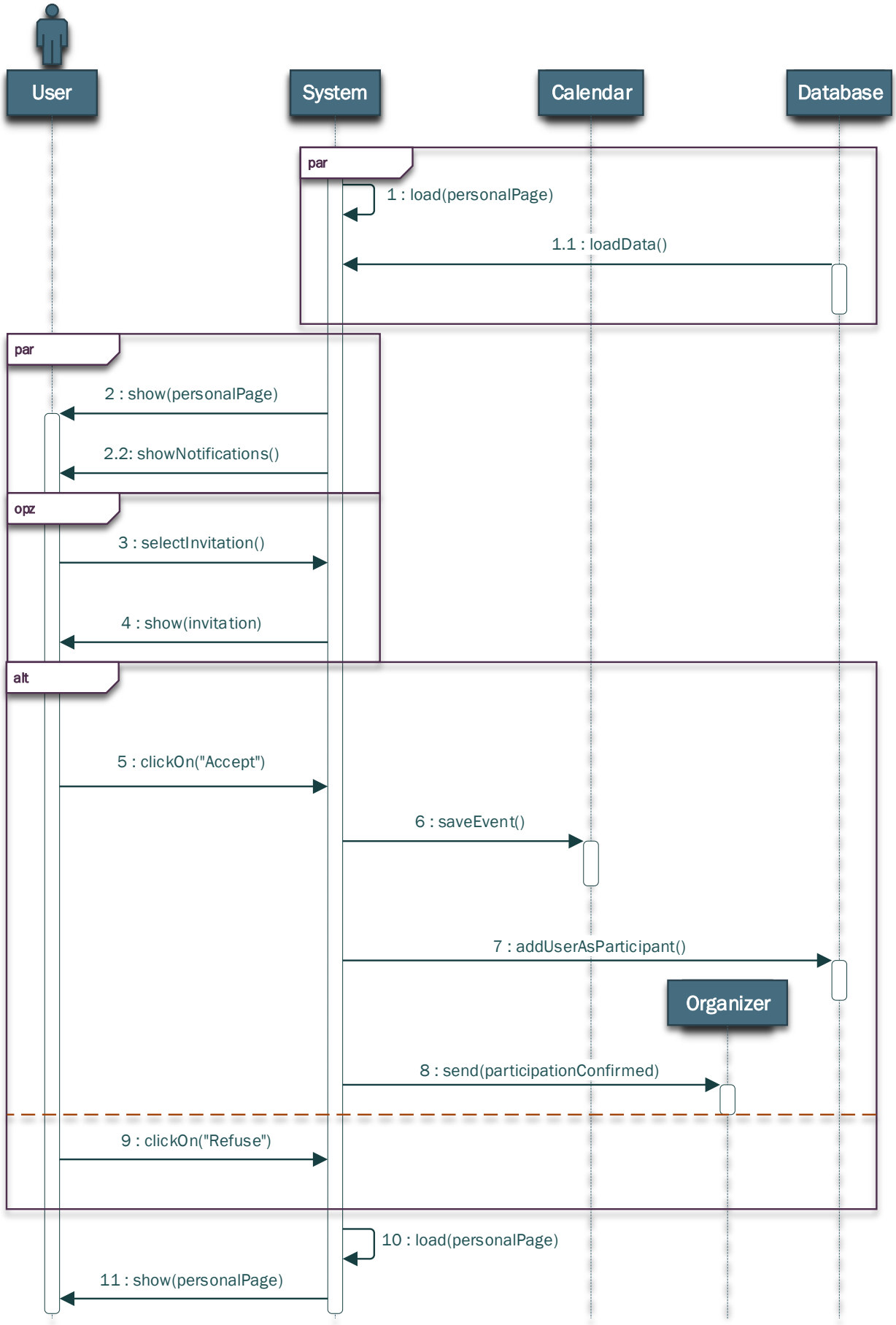
6.4.5 Update Event



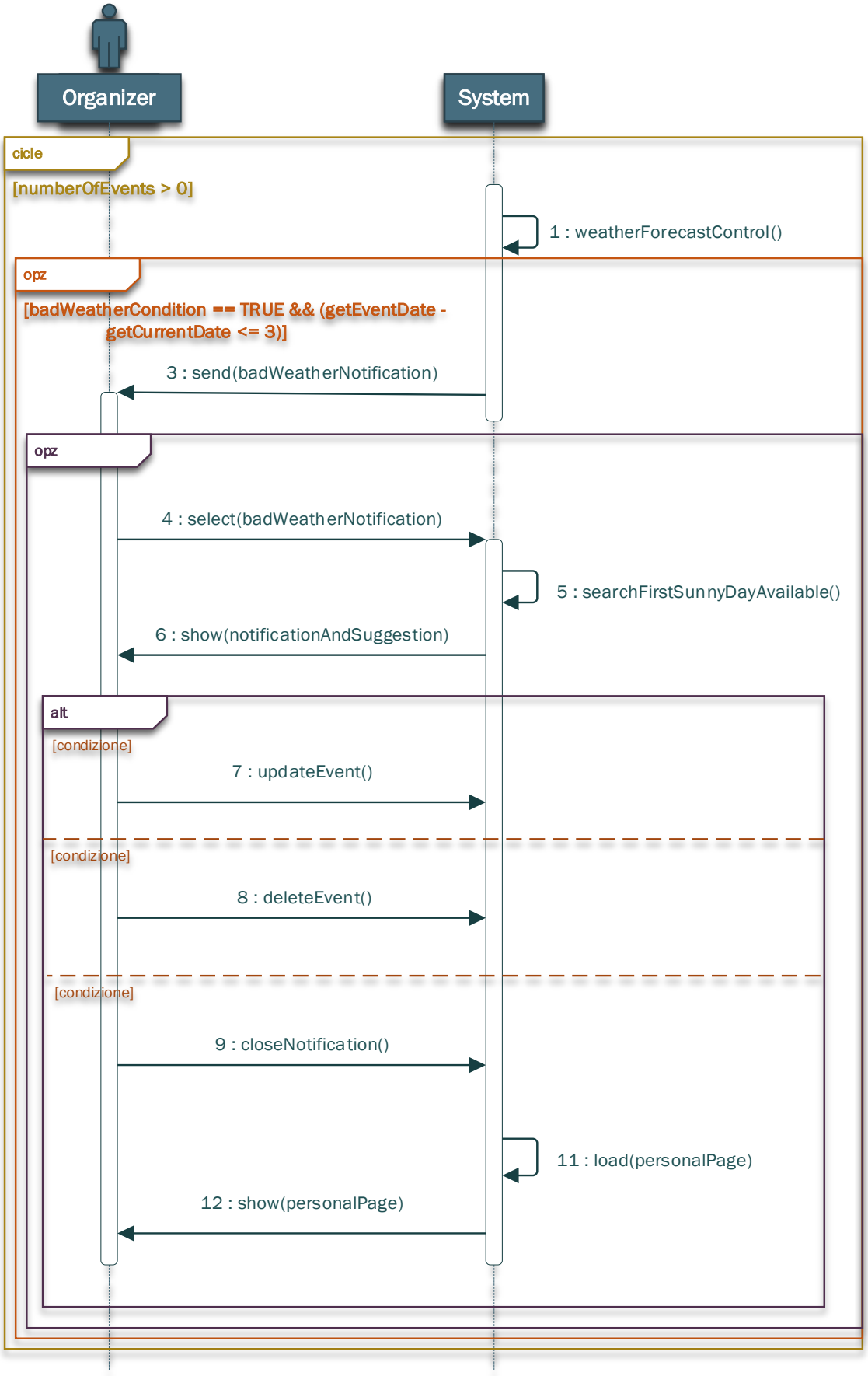
6.4.6 Bad weather notification



6.4.7 Event Confirmation

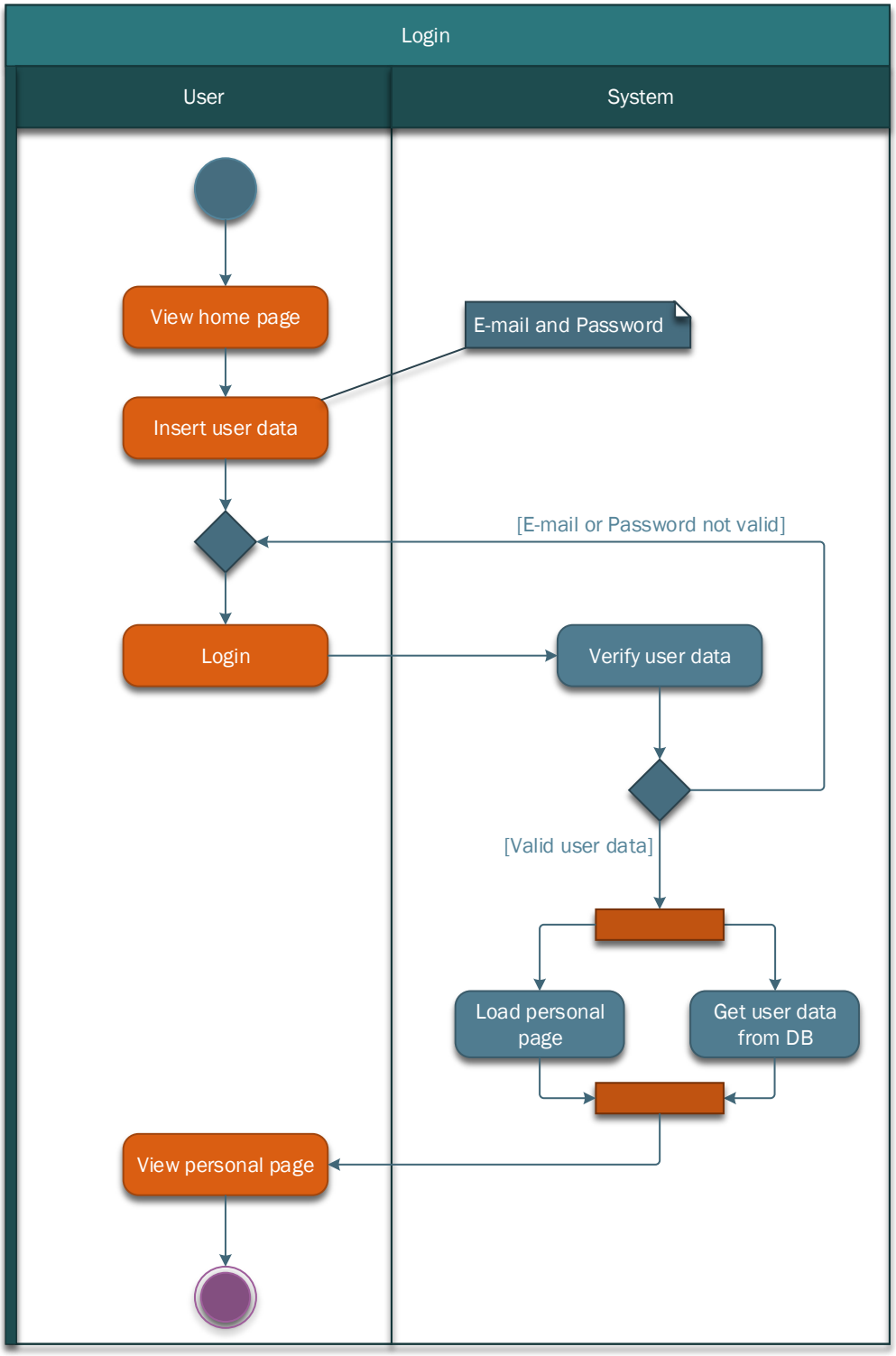


6.4.8 Bad weather notification to the organizer

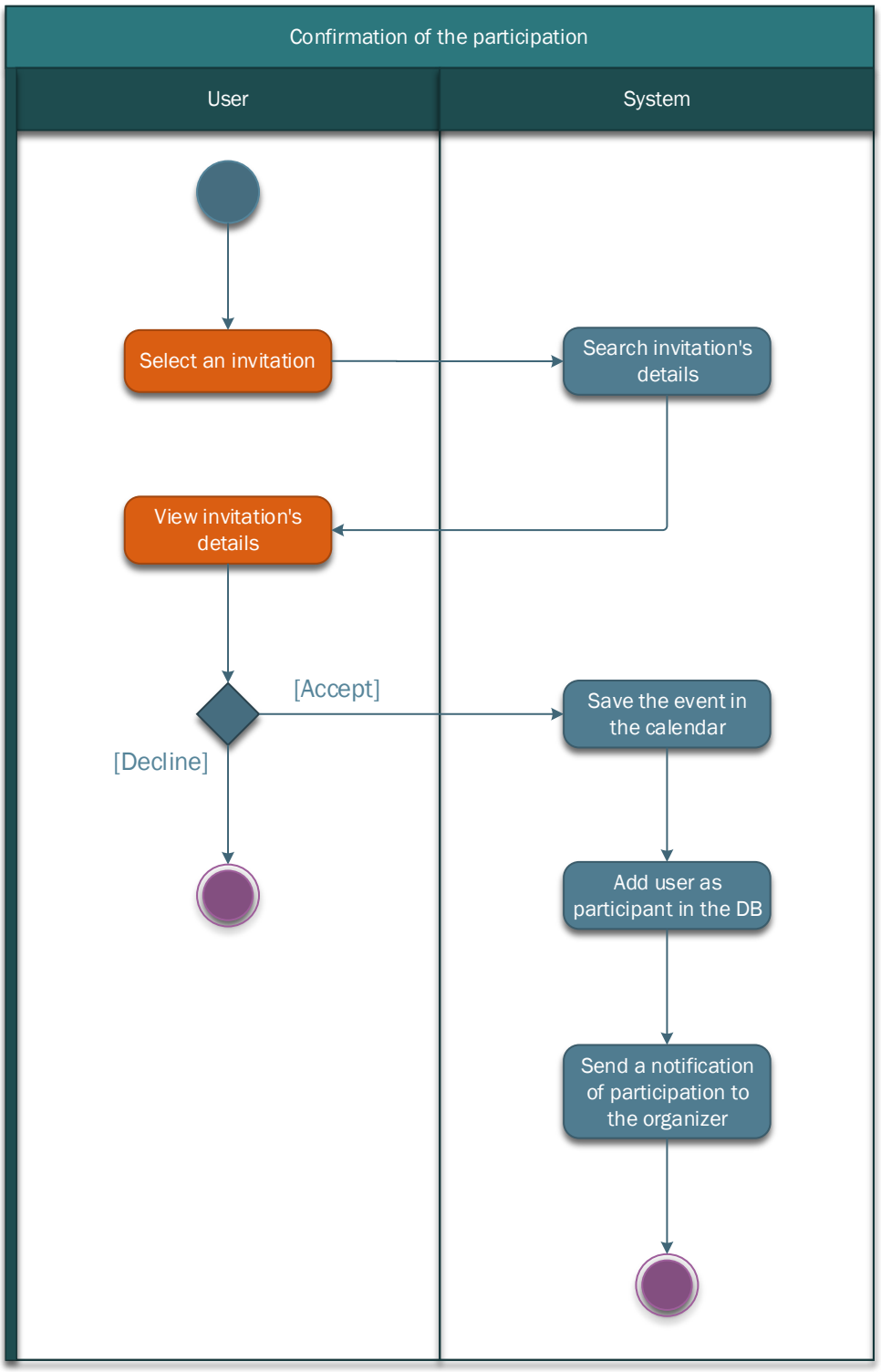


6.5 Activity diagrams

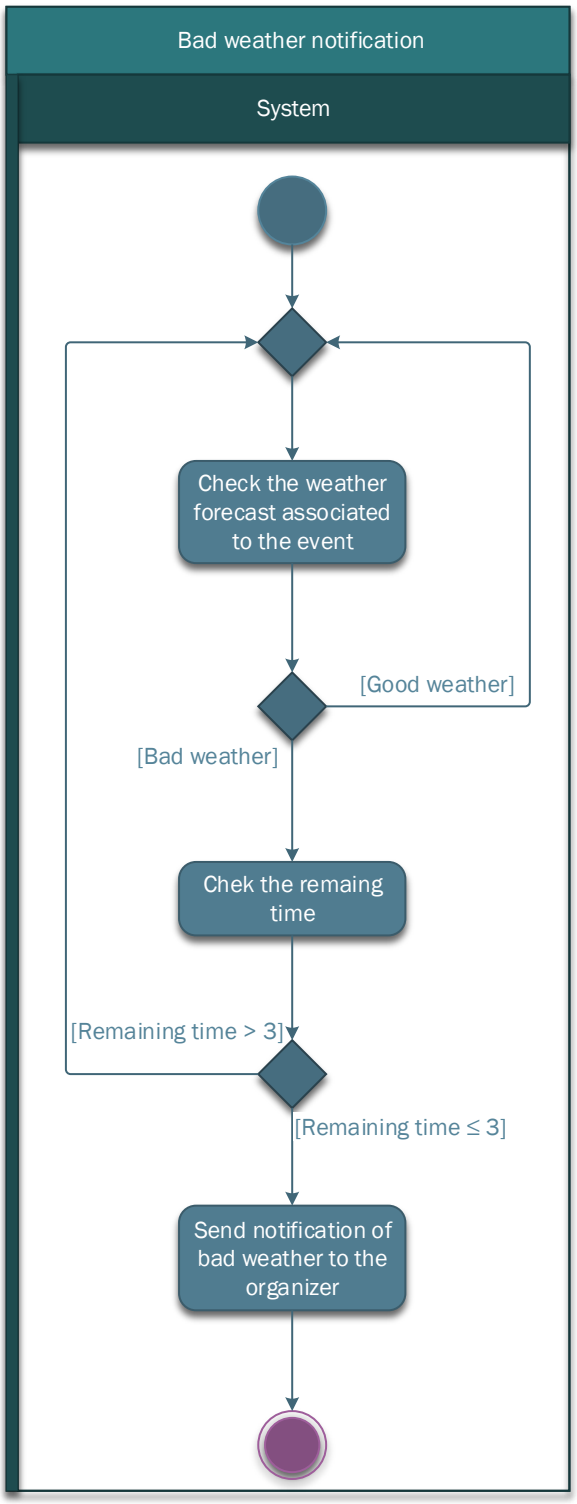
6.5.1 Log In



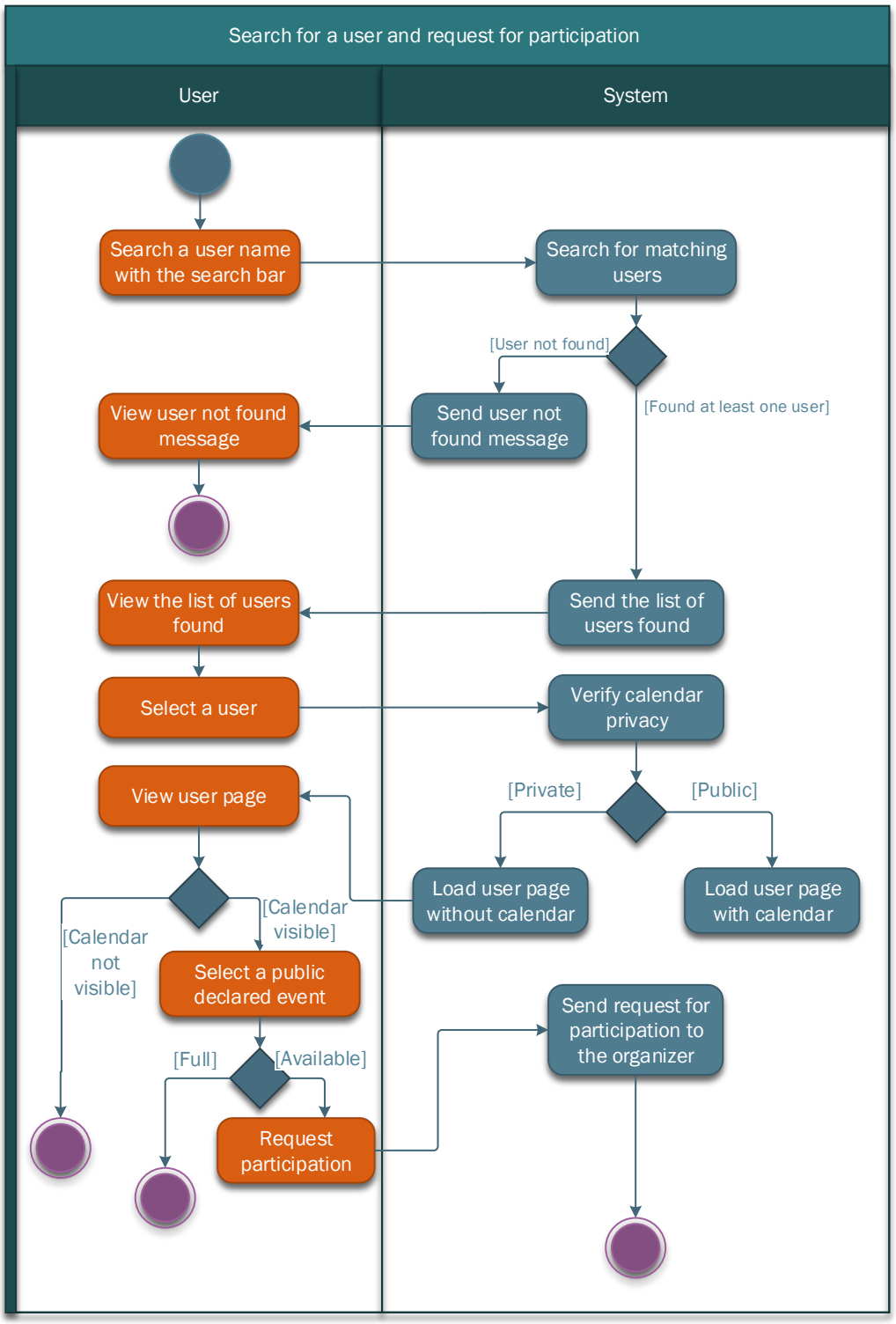
6.5.2 Confirmation of participation



6.5.3 Bad weather notification to the organizer



6.5.4 Search for a user and request for participation



7. Alloy

7.1 Alloy code

In this chapter we will briefly analyze the model designed for the project and see if it is consistent. For lack of time and simplicity, not all constraints have been translated and analyzed with alloy, especially the "time controls" of events (when receiving notification of bad weather, when it is possible to cancel the event, etc.).

```
//SIGNATURES

sig Name {
}

sig Surname {
}

sig  Email {
}

sig Password {
}

sig Phone {
}

sig TimeStamp {
}

sig Weather {
}

abstract sig Privacy {
}

one sig Public, Private extends Privacy {
}

abstract sig Type {
}

sig Invitation, BadWeatherCondition, EventDeleted, EventUpdated,
PartecipationConfirmation, PartecipationRequest extends Type {
}

abstract sig TypeOfPlace {
}

one sig Indoor, Outdoor extends TypeOfPlace {
}
```

```
sig Place {
	relatedWeather : Weather
}

sig Event {
	startTime : one TimeStamp,
	endTime : one TimeStamp,
	eventLocation : one Place,
	placeType : one TypeOfPlace,
	privacy : one Privacy,
	organizer : one User,
	participants : set User,
}

sig Calendar {
	owner : one User,
	events : set Event,
	privacy : one Privacy
}

sig Notification {
	dispatchTime : one TimeStamp,
	sender : lone User,
	addressee : one User,
	typeOfNotification : one Type,
	linkedEvent : one Event
}

sig User {
	name : one Name,
	surname : one Surname,
	email : one Email,
	password : one Password,
	phone : lone Phone,
	calendar : one Calendar,
	notifications : set Notification
}

//FATCS

//In the system can’t exist two users with the same data
fact userRules {
	all n : Name, s : Surname, e : Email, p1 : Password, p2 : Phone, c :
	Calendar | one u : User | u.name=n and u.surname=s and u.email=e and
	u.password=p1 and u.phone=p2 and u.calendar=c
	all c : Calendar | one u : User | c = u.calendar
	all e : Email | one u : User | e = u.email
	all p : Phone | one u : User | p = u.phone
}

fact calendarRule{
	//A user is the owner of a calendar if and only if he has that calendar
	all c : Calendar, u : User | (c.owner=u) iff (u.calendar=c)
```

```
	//If a user participates to an event, then the event is in his calendar or
	he is the organizer of that event
	all u : User, e : Event | (u in e.participants or e.organizer=u) iff (e in
	u.calendar.events)
}

fact eventRules {
	//TRIVIAL: The organizer can create an event but not participate
	all e : Event | e.organizer in e.participants or !(e.organizer in
	e.participants)
	//End time and start time can't be the same
	all e : Event | e.startTime!=e.endTime
}

fact notificationRules {
	//A notification has only one user
	all n : Notification | one u : User | n.addressee=u
	//If a user can view a notification then the notification was addressed to
	him
	all n : Notification, u : User | n in u.notifications iff n.addressee=u
	//If a user receives a notification that isn't an invitation then he is a
	participant of the corresponding event
	all n : Notification | (n.typeOfNotification=EventUpdated and
	n.typeOfNotification=EventDeleted) iff (n.addressee in
	n.linkedEvent.participants)
	//The sender of a notification is not equal to the addressee
	all n : Notification | !(n.addressee=n.sender)
	//If a user accepts an invitation to join an event then the participants'
	list contains him
	all n : Notification | (n.typeOfNotification=PartecipationConfirmation)
	implies (n.sender in n.linkedEvent.participants)
	//If a user requests an event participation then the corresponding event
	is public and the calendar that contains that event is public too
	all n : Notification | (n.typeOfNotification=PartecipationRequest) iff
	(n.linkedEvent.privacy= Public and (some c : Calendar | n.linkedEvent in
	c.events && c.privacy=Public))
	//If a user recieves an event deleted notification then his calendar
	doesn't contain the event anymore and he is not a participant
	all n : Notification | (n.typeOfNotification=EventDeleted) implies
	((n.addressee in n.linkedEvent.participants) and (all c : Calendar |
	!(n.linkedEvent in c.events)))
	//If a user receives an event invitation then he is not already within the
	participants
	all n : Notification | (n.typeOfNotification=Invitation) implies
	!(n.addressee in n.linkedEvent.participants)
	//When the notification is about the bad weather conditions then the
	notification sender is the system and not a user
	all n : Notification | !(n.sender in User) implies
	(n.typeOfNotification=BadWeatherCondition)
	//Bad weather notifications are received by the event participants
	all n : Notification | ((n.typeOfNotification=BadWeatherCondition) implies
	(n.addressee in n.linkedEvent.participants))
	//Bad weather notifications are received by the event organizer
```

```
	all n : Notification | ((n.typeOfNotification=BadWeatherCondition) implies
	(n.addressee=n.linkedEvent.organizer))
	//The organizer can't do a request to join one of his own events
	all n : Notification | (n.typeOfNotification=PartecipationRequest) implies
	(n.sender!=n.linkedEvent.organizer)
}

//ASSERTIONS

assert noUserDuplication {
	no disj u1, u2 : User | u1.name=u2.name and u1.surname=u2.surname and
	u1.email=u2.email and u1.password=u2.password and u1.phone=u2.phone and
	u1.calendar=u2.calendar
}
check noUserDuplication for 4

assert eventInTheCalendar {
	all u : User | all e : Event | (u=e.organizer) implies (e in
	u.calendar.events)
}
check eventInTheCalendar for 4

assert notificationSenderNotEqualToAddressee {
	all n : Notification | n.addressee!=n.sender
}
check notificationSenderNotEqualToAddressee for 4

assert userSeesOnlyHisNotifications {
	all n : Notification, u : User | (n.addressee=u) iff (n in
	u.notifications)
}
check userSeesOnlyHisNotifications for 4

assert participantsListIsAlwaysWellComposed {
	all n : Notification | (n.typeOfNotification=PartecipationConfirmation)
	implies (n.sender in n.linkedEvent.participants)
	all n : Notification | (n.typeOfNotification=EventDeleted) implies
	#(n.linkedEvent.participants)=0
}
check participantsListIsAlwaysWellComposed for 4

assert aUserCantSeePrivateEventOrPrivateCalendar {
	all n : Notification | (n.typeOfNotification=PartecipationRequest) implies
	(n.linkedEvent.privacy=Public)
}
check aUserCantSeePrivateEventOrPrivateCalendar for 4

//PREDICATES

//A simply world with two Users
pred world1 {
	#User=2
	#Event=0
	#Place=0
```

```
	#TimeStamp=0
	#Notification=0
}
run world1

//A world with one user and two Events. The organizer participates in only one
event.
pred world2 {
	#User=1
	#Event=2
	one e : Event | #(e.participants)=0
}
run world2

pred show {
}
run show
```

7.2 Report

Executing "Check eventInTheCalendar for 4"
Solver=sat4j Bitwidth=0 MaxSeq=0 SkolemDepth=1 Symmetry=20
77539 vars. 424 primary vars. 183509 clauses. 1204ms.
No counterexample found. Assertion may be valid. 151ms.

Executing "Check notificationSenderNotEqualToAddressee for 4"
Solver=sat4j Bitwidth=0 MaxSeq=0 SkolemDepth=1 Symmetry=20
77483 vars. 420 primary vars. 183455 clauses. 1220ms.
No counterexample found. Assertion may be valid. 32ms.

Executing "Check userSeesOnlyHisNotifications for 4"
Solver=sat4j Bitwidth=0 MaxSeq=0 SkolemDepth=1 Symmetry=20
77517 vars. 424 primary vars. 183539 clauses. 1307ms.
No counterexample found. Assertion may be valid. 70ms.

Executing "Check participantsListIsAlwaysWellComposed for 4"
Solver=sat4j Bitwidth=0 MaxSeq=0 SkolemDepth=1 Symmetry=20
77460 vars. 416 primary vars. 183553 clauses. 1294ms.
No counterexample found. Assertion may be valid. 46ms.

Executing "Check aUserCantSeePrivateEventOrPrivateCalendar for 4"
Solver=sat4j Bitwidth=0 MaxSeq=0 SkolemDepth=1 Symmetry=20
77495 vars. 420 primary vars. 183475 clauses. 1242ms.
No counterexample found. Assertion may be valid. 79ms.

Executing "Run world1"
Solver=sat4j Bitwidth=0 MaxSeq=0 SkolemDepth=1 Symmetry=20
13225 vars. 252 primary vars. 28938 clauses. 176ms.
Instance found. Predicate is consistent. 54ms.

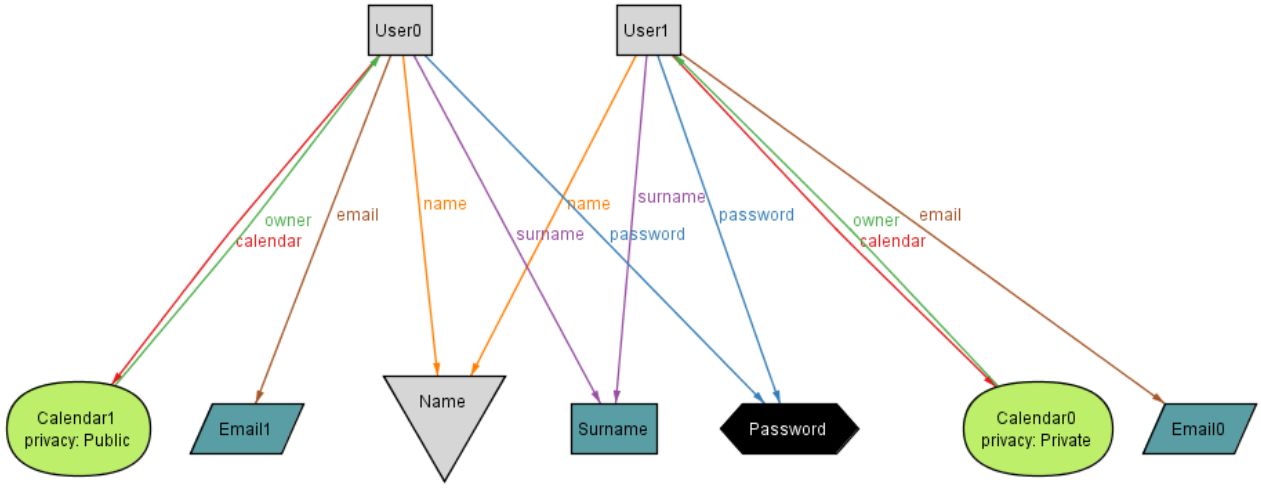
Executing "Run world2"
Solver=sat4j Bitwidth=0 MaxSeq=0 SkolemDepth=1 Symmetry=20
13233 vars. 252 primary vars. 28981 clauses. 169ms.
Instance found. Predicate is consistent. 44ms.

Executing "Run show"
Solver=sat4j Bitwidth=0 MaxSeq=0 SkolemDepth=1 Symmetry=20
13190 vars. 252 primary vars. 28823 clauses. 188ms.
Instance found. Predicate is consistent. 46ms.

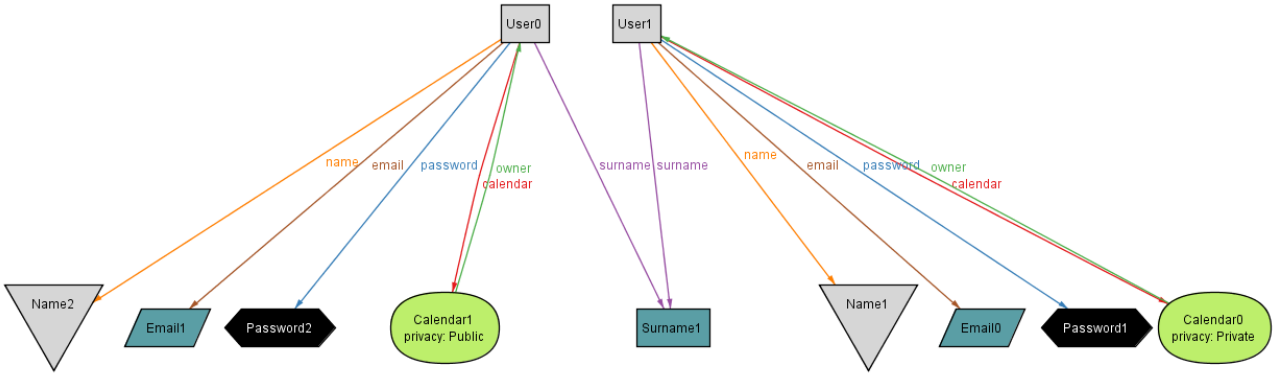
9 commands were executed. The results are:
#1: No counterexample found. noUserDuplication may be valid.
#2: No counterexample found. eventInTheCalendar may be valid.
#3: No counterexample found. notificationSenderNotEqualToAddressee may be valid.
#4: No counterexample found. userSeesOnlyHisNotifications may be valid.
#5: No counterexample found. participantsListIsAlwaysWellComposed may be valid.
#6: No counterexample found. aUserCantSeePrivateEventOrPrivateCalendar may be valid.
#7: Instance found. world1 is consistent.
#8: Instance found. world2 is consistent.
#9: Instance found. show is consistent.

7.3 Worlds generated

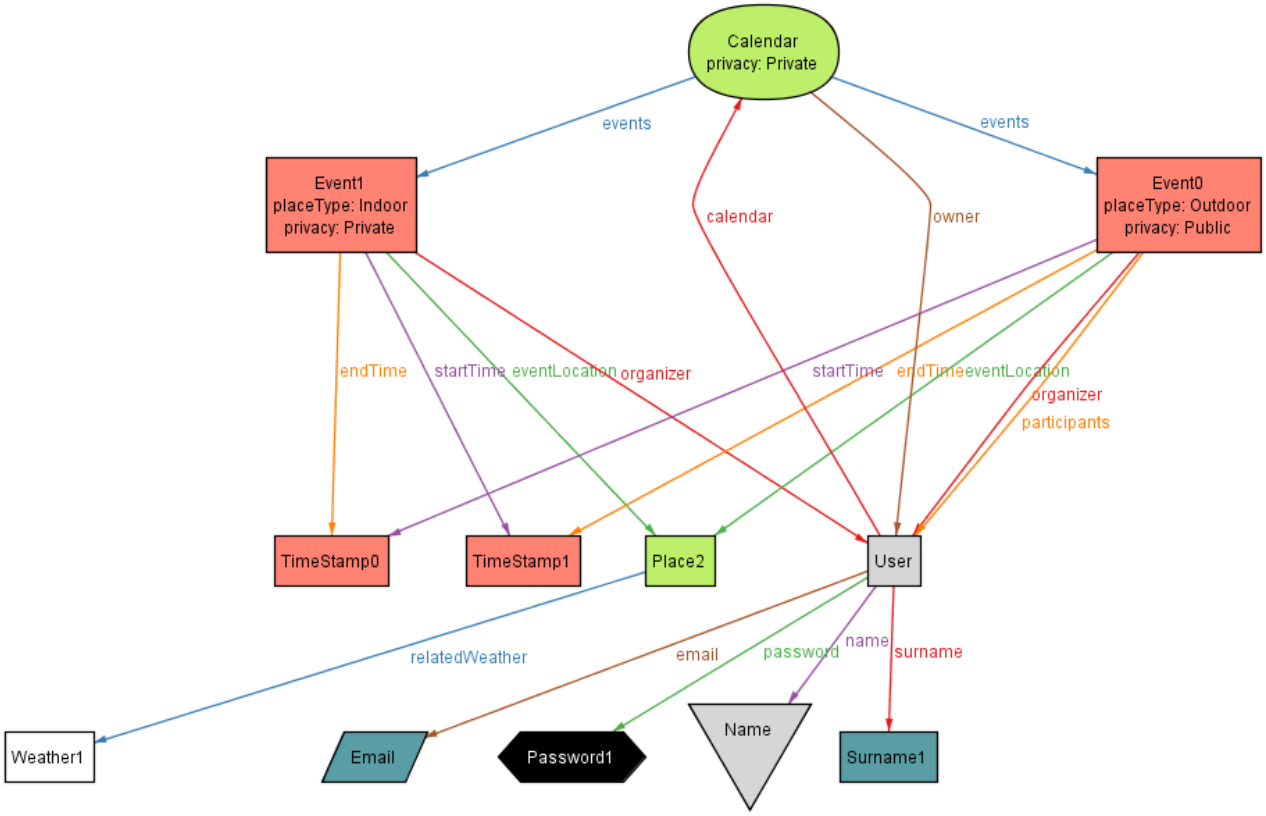
The first world is a simple example of how the system does not allow duplication of user data. The users in the system will be identified primarily for their email and each member will have one and only one calendar. Other data, however, will be reasonably duplicate: it is possible that there are users with the same name or who choose the same password for access.



In this case instead the majority of the data are different.



In this world, there are one user and two events and the user's organizer of both, but only in one case he attends the event.



Finally, this is an example of a complete world.