REQUIREMENTS ANALYSIS AND SPECIFICATION DOCUMENTS

Project of Software Engineering 2

WEATHER-CAL

Authors:

PAOLO POLIDORI

MARCO EDEMANTI

Contents

List of Figures

Chapter 1

Introduction

1.1 Purpose

The purpose of this document is to present a description of WheaterCal system. It will explain the features of the system, the interfaces of the system, what the system will do and the constraints under which it must operate. This document is intended for both the stakeholders and the developers of the system.

1.2 Scope

We want to project and implement WheaterCal. The aim of this project is to develop a system that offers an online calendar in which user can schedule their events according to the weather conditions.
A registered user can create, delete and update an event and moreover he should provide information about where and when this event will take place and information about the invited user. Once the event is created the system should provide to its creator the weather forecast information regarding the scheduled day, and most of all

it should notify a bad weather condition one day in advance to all the participants's event.
Also a user is able to make his/her calendar visible to all other registered user showing them only the time slots in which they are busy without letting know the event information unless either the event is public. In addition in case of bad weather, three day before the scheduled data of an event, the system will inform the event's owner and propose to him the closest sunny day.

1.3 Glossary

User	La pagina non subisce variazioni e rimane ferma jjkjkjkkjklkldldldldlldldldldl
Event	La pagina si sposta verso l'alto
Event's Owner	La pagina si sposta verso destra
Participant	La pagina si sposta verso il basso
Calendar	La pagina si sposta verso sinistra

- User
- Event's Owner
- Event
- Calendar
- Participant

1.4 References

IEEE,*IEEE Std 830-1998,IEEE Recommended Practice for Software Requirements Specifications*, IEEE Computer Society 1998

Chapter 2

Overall Description

2.1 System Environment

WeatherCal is a stand alone software made from scratch. It is a Web application that allows the registered users to manage their schedule basing on the weather conditions. The users are also able to invite people to their event and to publicize their event. The system is even capable of notifying the users, as they log on to the system, if the weather forecast for the event is not the desired one. The target of this software is, therefore, to give the users a tool for schedule their events smartly, giving the possibility to change preferences as the weather forecast changes.

2.1.1 Actors

Actors in the system are mainly the people registered into the web application. They will be referred as Registered Users. Whoever is not yet registered on the platform will be an actor for the system too and it will be identified as an Anonymous Users. No administration is required since the system is not intended for illicit purposes and this is a policy of the platform, plus no conflict among users can occur since the only allowed interactions are the invite to an event and a participation to an event (in which other people partecipate).

2.1.2 Scenarios

- Maggie creates an event for her daughter's birthday, which will be in their garden. She invites all her friends telling them to join the event on WeatherCal to be sure that the forecast will be graceful for that day. Three days before the birthday, Maggie logs in two days before the event and she discovers it will be cloud but she still hopes it won't rain on that day. The day before the event Judith, Maggie's friend, is notified that it could be rainy for the birthday as she logs into the system. Anyway Maggie decides she won't posticipate the birthday. In the end that day was rainy, except for the afternoon, exactly for the birthday, and everything went fine.
- John wants to go cycling but right now the forecast is not so optimistic, so he decides to use WeatherCal to find the best day for doing so. The platform suggests him that he should go on Monday afternoon, so he plans to do that. Saturday he logs in and discovers that it will be windy on Monday, infact the system suggests him to go cycling on Wednesday and John chooses to do what the system suggests him. In the end he went cycling and the weather was perfect.
- Beth wants to have a picnic with her friends in two weeks, so she creates an event on WeatherCal when the temperature should be the hottest and there should be no clouds in the sky. Three days before she logs in discovering that for the next two weeks there will be bad weather conditions. She decides so to invite her friends at home for having a launch together.

2.2 Functional Requirements

The system allows the users to partecipate in various scenarios:

- Log on to and log out from the platform.
- Create, Modify, Delete an event.
- Invite other users.
- Manage invitation (accept, refuse).
- Manage event visibility (decide if an event's infos should be accessible only by its participants).
- View other users' profiles.

The system is the leading actor in:

- Warn all the participants to an event which will occur the following day that there will be bad weather conditions.
- Notify to an event's owner three day before the scheduled data that there will be bad conditions for it suggesting the closest available sunny day.

The Anonymous Users are only able to:

- Sign Up in to the Platform.

2.3 Goals

The WeatherCal system should offer and fulfill these main features:

- A simple management for the users' agendas

- A swift way to schedule an event according to the weather forecast
- Invite other registered users to an event
- Preserve the privacy of the users since only the user himself can make their info visibile to everyone.

2.4 Non Functional Requirements

2.4.1 User Interfaces

The WeatherCal system is a Web application intended to be used trough desktop device. Its interface is designed in a simple way so that the user can either easily understand what the system can offer to him and reaches all the main features of the system from a unique view.
Once that a user logs in to the system trough a defined web page he is redirected to the main page. This page is divided in two portion. The left one, which occupies most of the page, displays a monthly calendar in which the user manages its events while the right one shows the weather forecast for the following days and also shows the closest events.
In the figure 2.1 is shown a first sketch of the main user's page.

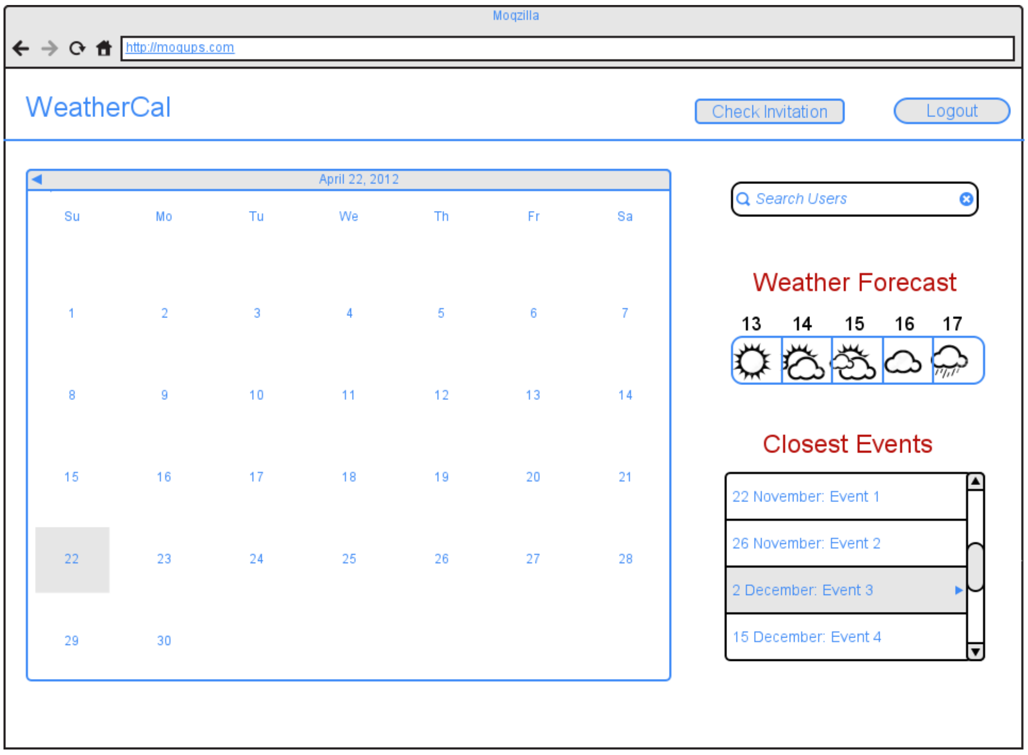


Figure 2.1: Main Page's MockUp

2.4.2 Performance

The system offers the users the possibility to use it without major slowdowns. Multiuser usage is allowed and it can be bore by the server application under reasonable conditions, basing on the server hardware.

2.4.3 Security

The system relies on HTTP protocol for transferring data over the Internet, so information sent both from the user to the server and vice-versa will be exposed to possible man-in-the-middle attacks. Unless this issue data stored on the server will be accessible only if valid credentials are provided and there is no possibility for a user to bypass privacy constraints, i.e. users will not be able to see non-public events of other users and modify events for which they are not owners.

Chapter 3

System Modeling

3.1 UML Diagrams

3.1.1 Use Case Diagrams

Event Management

The use case in figure 3.1 shows how a user can manage his agenda. After he logs into the platform he will see his calendar and being able to see schedule of his events. This event can be modified and so deleted or if the user is the event's owner he can invite other users.

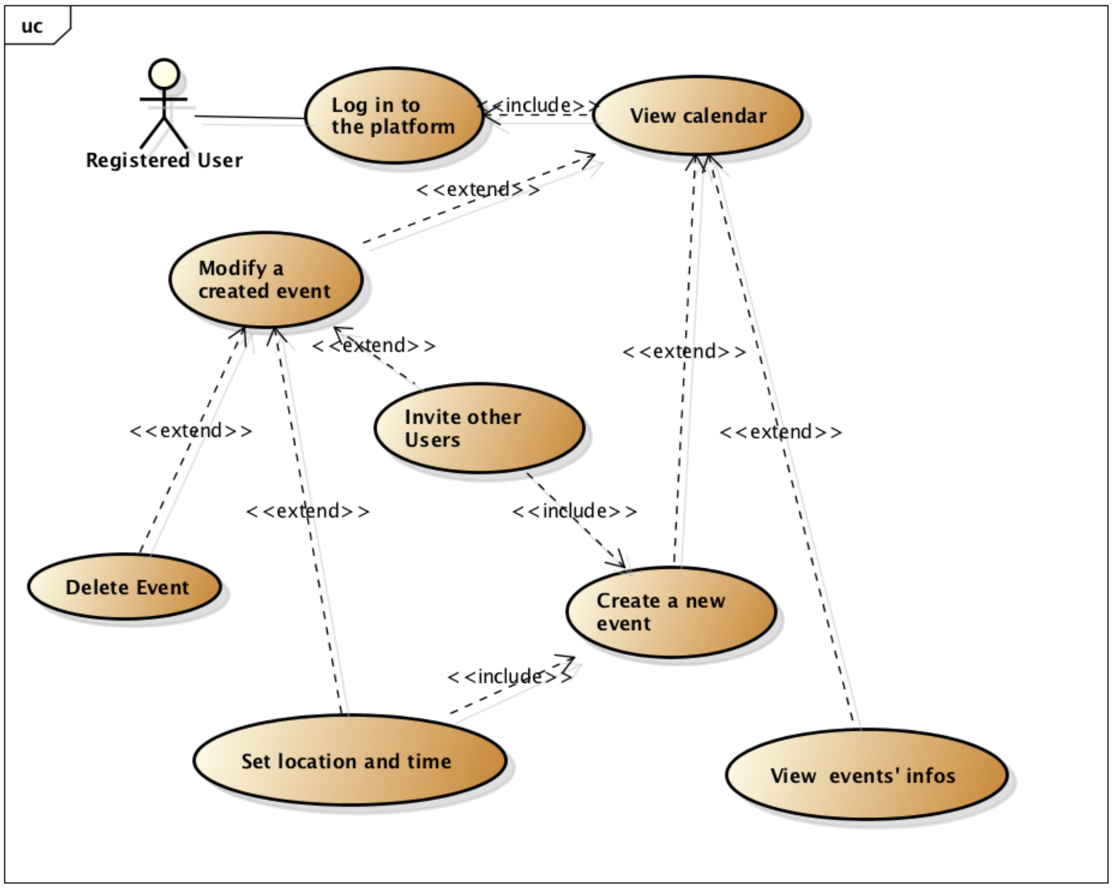


Figure 3.1: Event Management Use Case

The use case in the shows how a user can manage his agenda. After he logs into the platform he will see his calendar and being able to see schedule of his events.This events can be modified and so deleted or if he is the event's owner he can invite other users.This use case show how a user can manage his agenda. After he logs into the platform he will see his calendar and being able to see schedule of his events.This events can be modified and so deleted or if he is the event's owner he can invite other users.

Invitation

The use case in figure 3.2 explains what the user can do when he receive an invitation to an event from an other user. Once he get the notification he can either see the event's info and accept or decline the invitation.

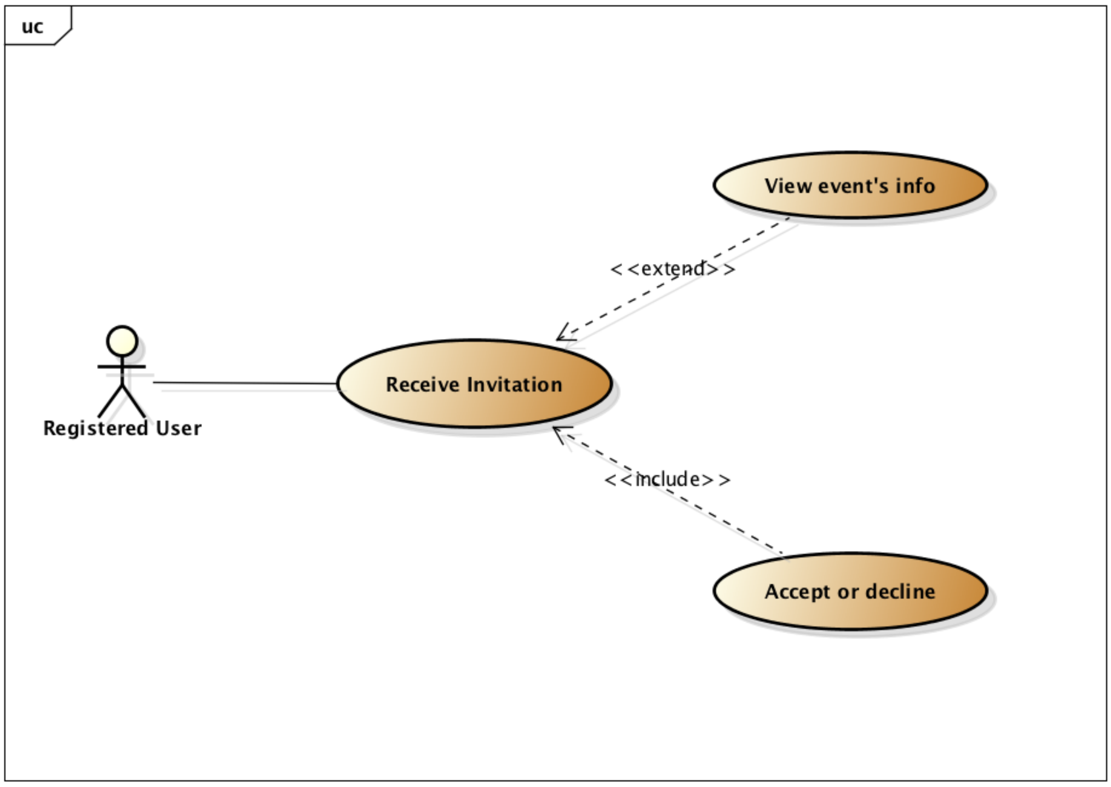


Figure 3.2: Invitation Use Case

View other users' profiles

The use case in figure 3.3 shows how an user can reach the profile of an other user and view his agenda. After he logs in to the platform he will be able to search an user and see his profile, but he will be authorized to see the other user's calendar only if it is set as public by its owner.

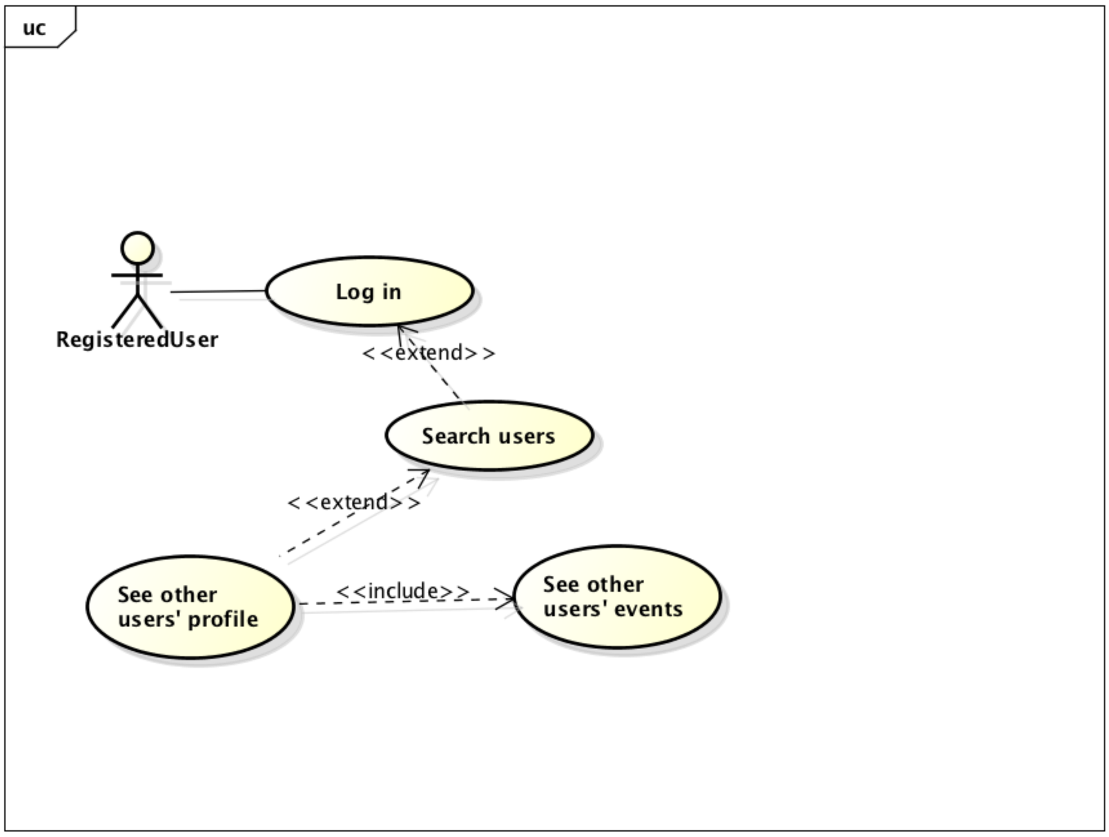


Figure 3.3: See Other User Profile Use Case

Bad Weather

The use case in figure 3.3 explains how an user can behave when he get notified by the platform in case of bad weather condition. Whenever he receives a bad weather notifications the user can act in two different way. If he's the event's owner he can delete the event or modify the event's location while if he's an event's participant he can only manage his event's participation.

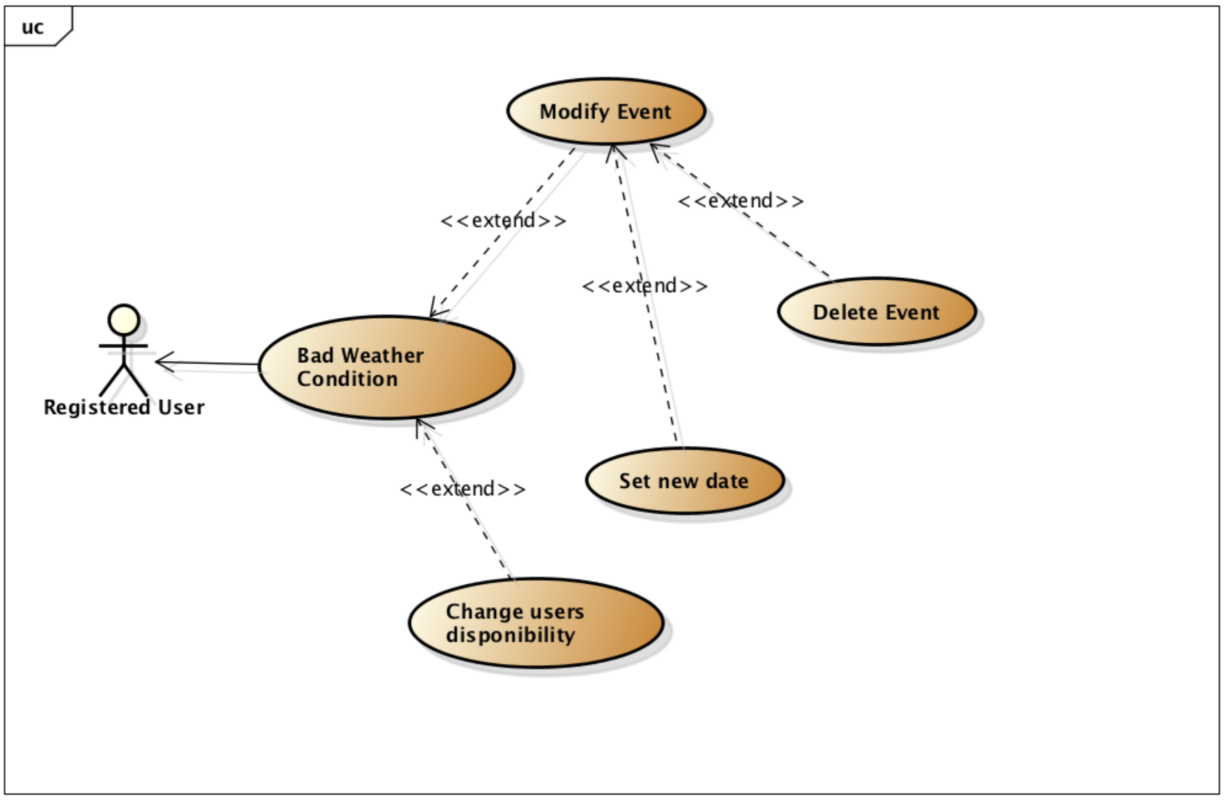


Figure 3.4: Bad Weather Use Case

3.1.2 Class Diagrams

Once that the requirements of our platform have been defined we are able to compose the class diagram of the system,as shown in figure 3.5.

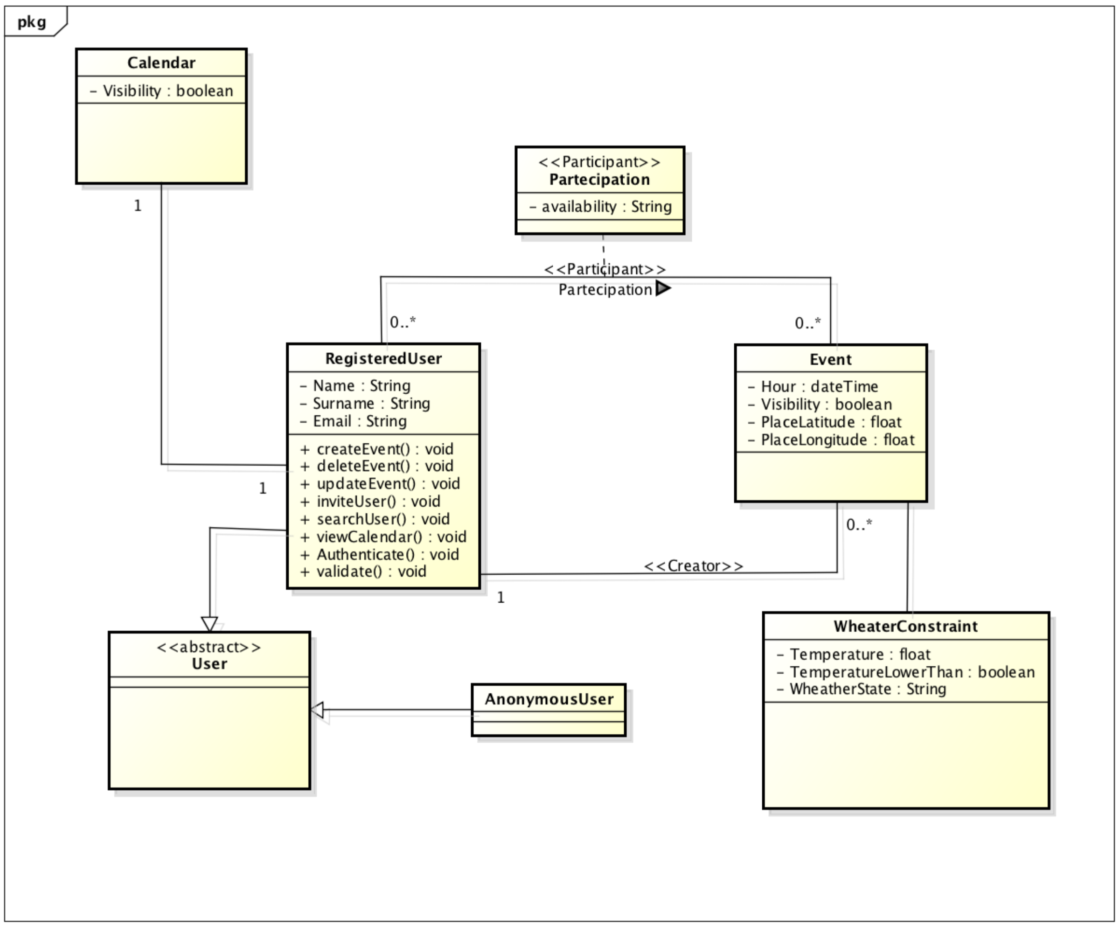


Figure 3.5: Class Diagram

3.1.3 State Diagrams

Invitation

In figure 3.6 the state diagram of an invitation to an event is shown.

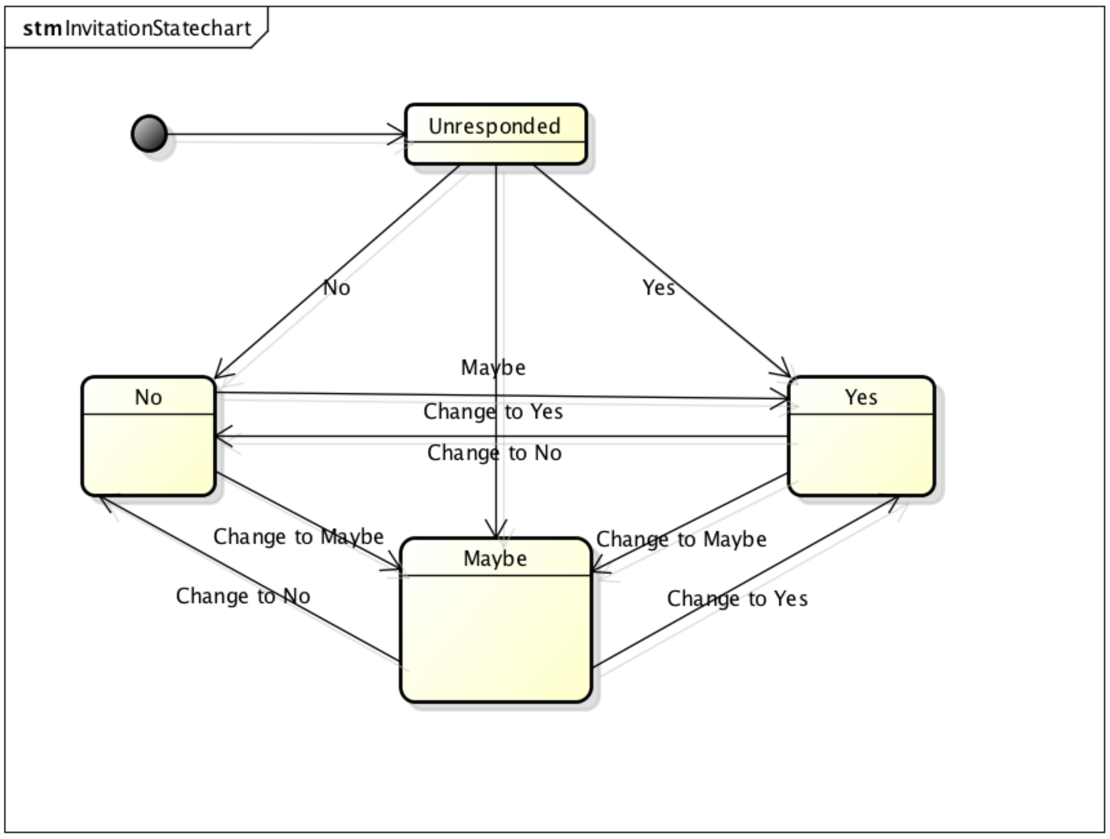


Figure 3.6: Invitation State Diagram

As the event owner invites a user, an invitation is created, having an unknown participation, which is defined by the first state. When an invitation is in this state means that an invitation for a user to an event has been made, but he didn't respond to the invitation or he don't even seen that yet. As the user responds to the notification, the state of the invitation becomes definite, so in can be "Yes", "No" or "Maybe". The user can then change that state among these three states, until the event has ended. Infact, after the end of the event the modification of the participation will be disallowed.

3.1.4 Sequence Diagrams

User Registration

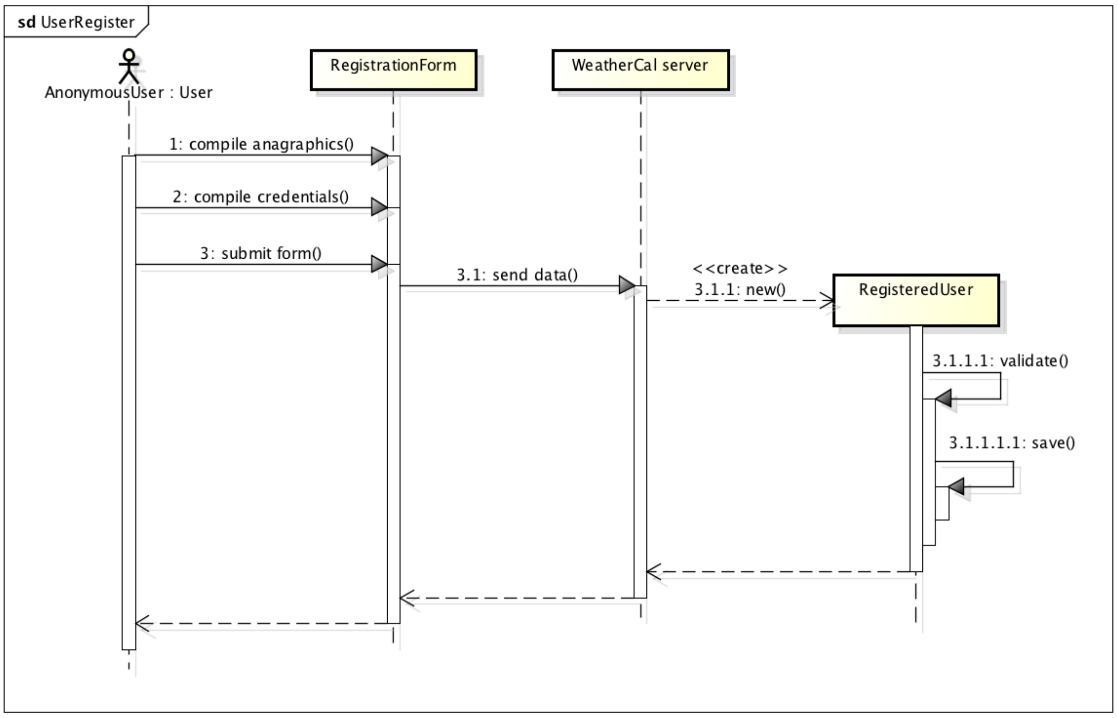


Figure 3.7: Sequence diagram of user registration

Login

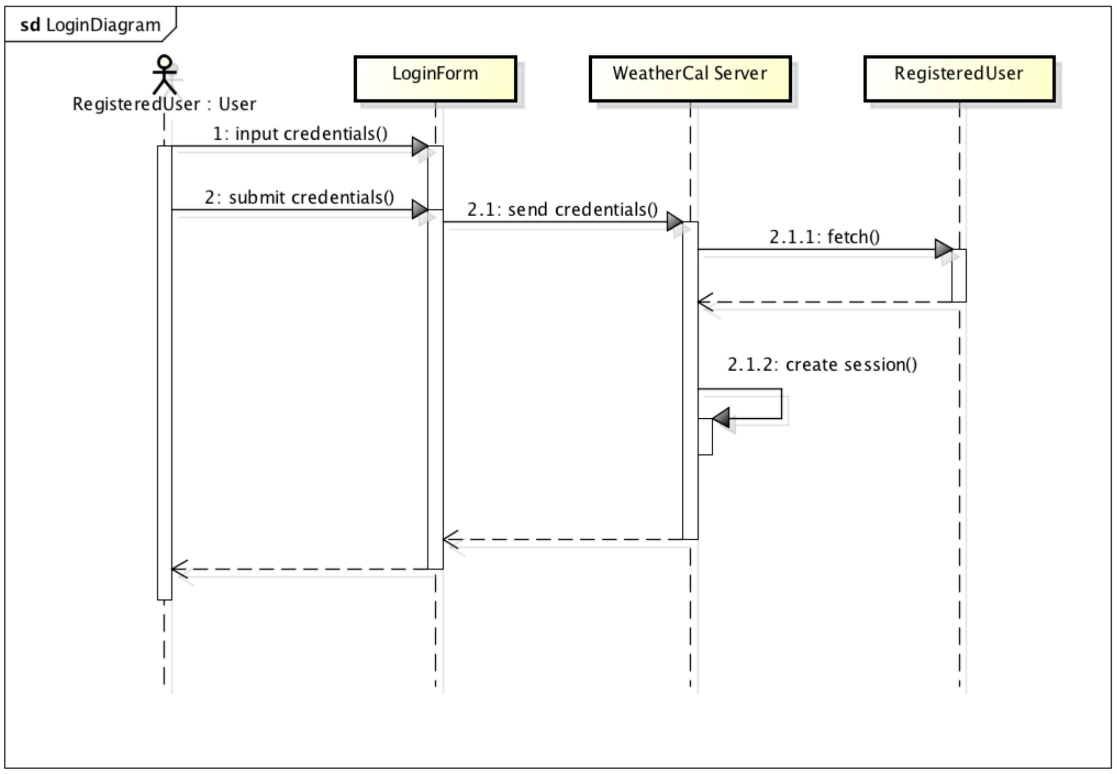


Figure 3.8: Login Sequence Diagram

Event creation

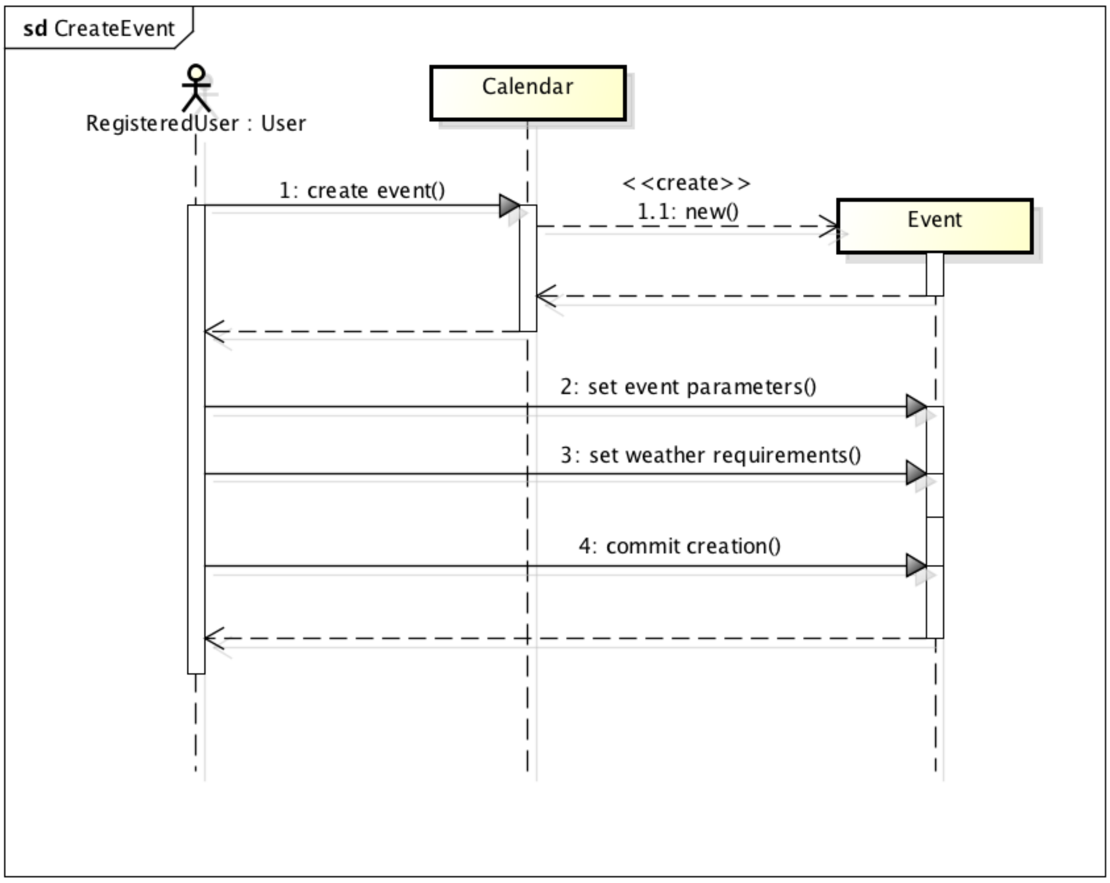


Figure 3.9: Create Event Sequence Diagram

Event modification

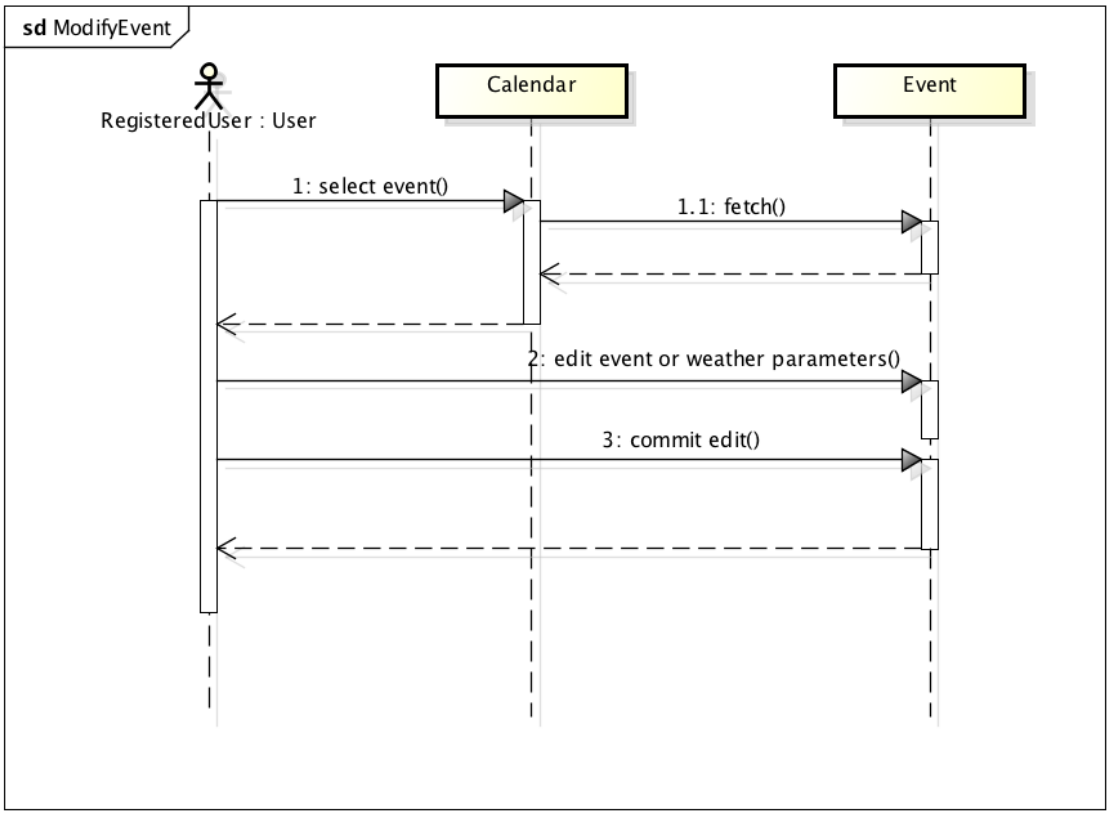


Figure 3.10: Sequence Diagram of event's modification

Event deletion

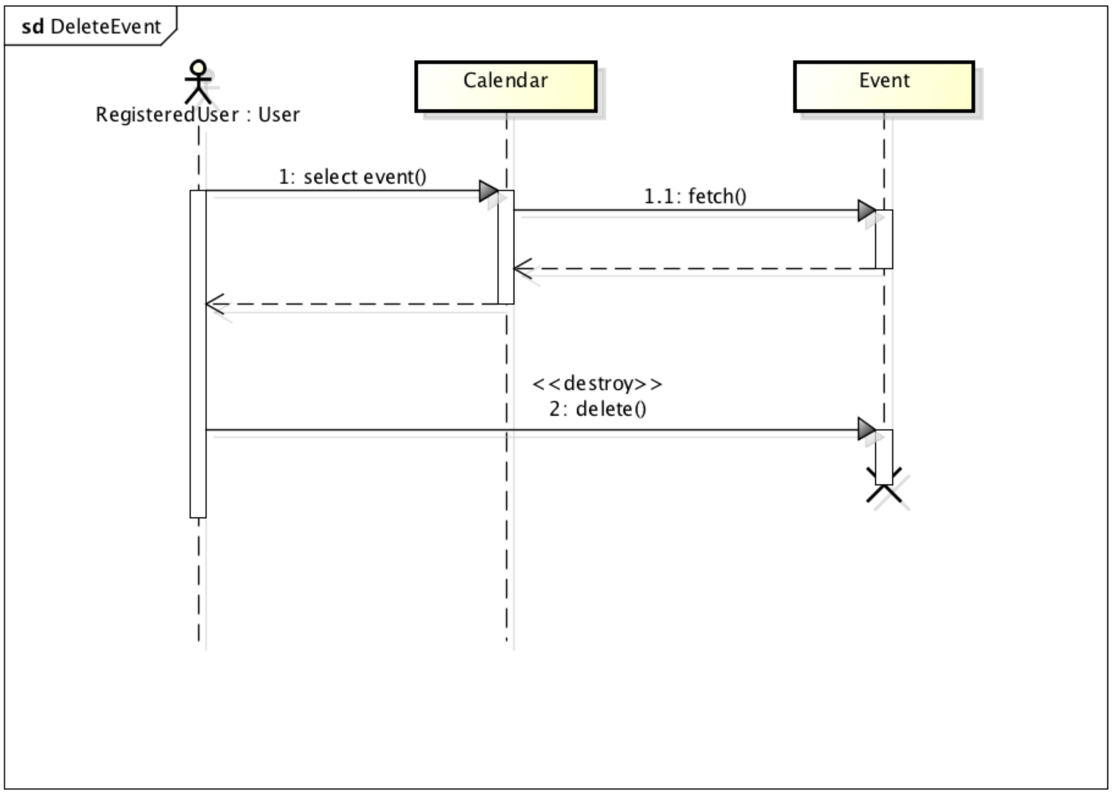


Figure 3.11: Sequence Diagram of event's deletion

Invitation

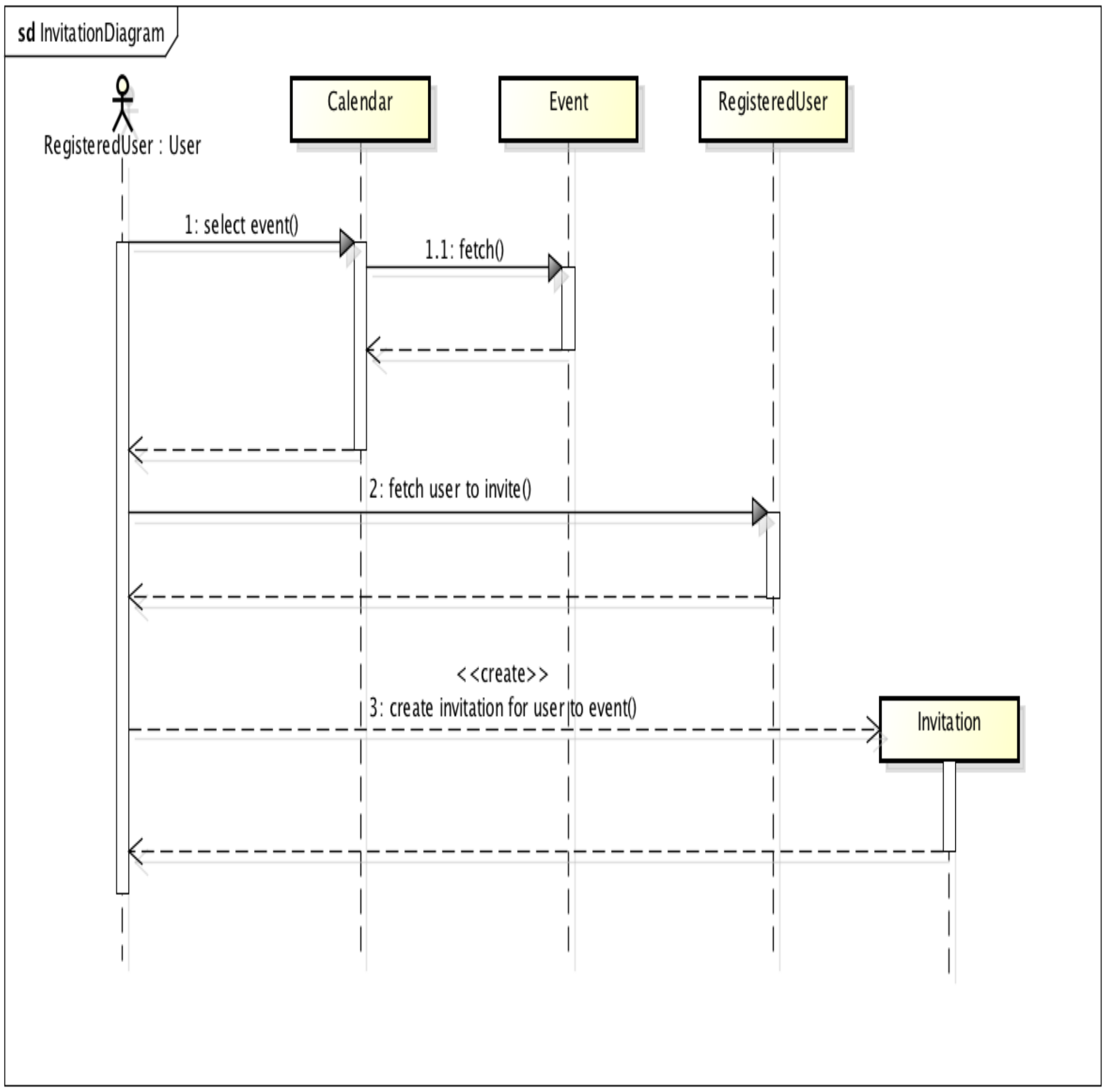


Figure 3.12: Sequence Diagram of an invitation

Invite notification

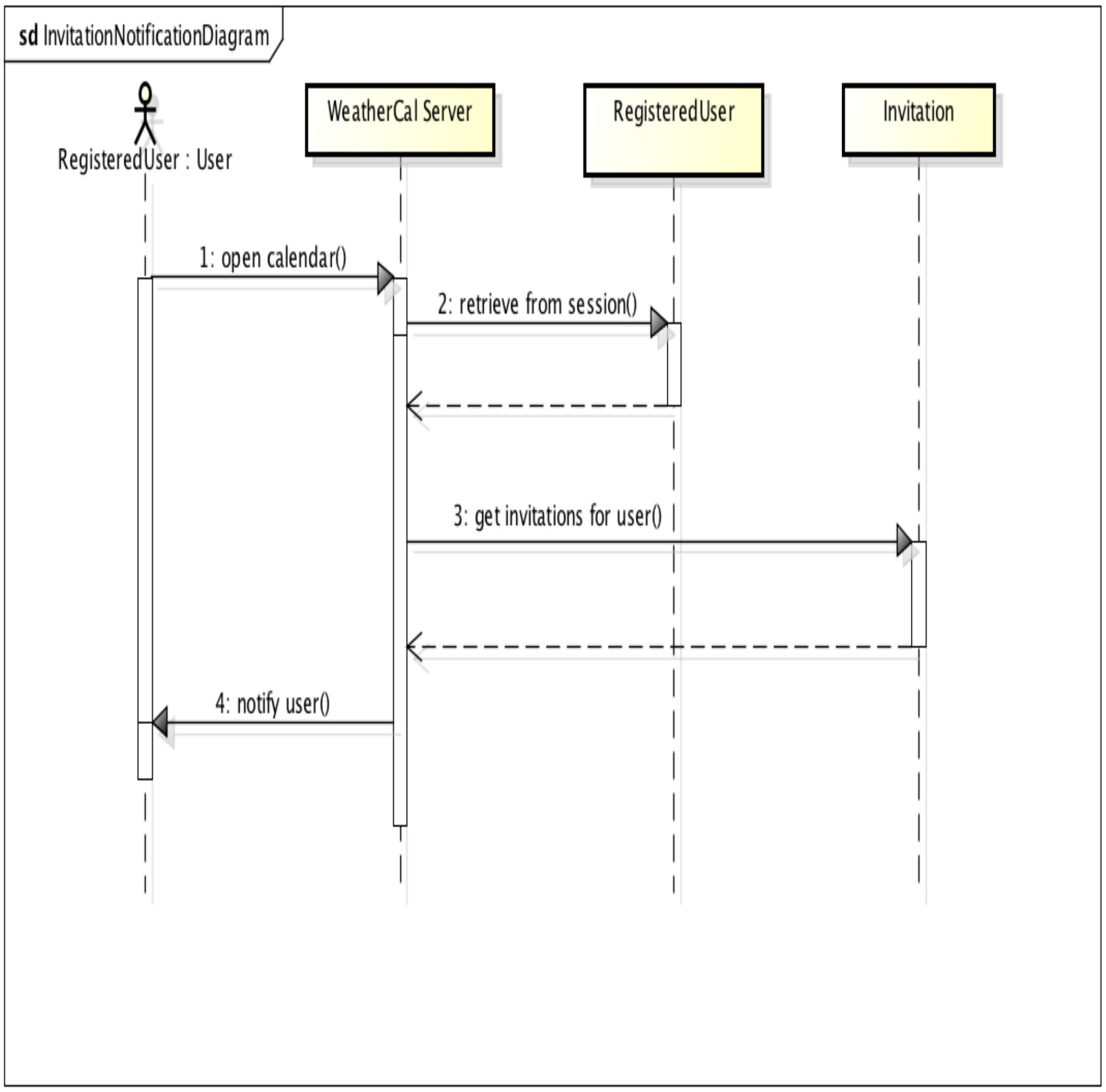


Figure 3.13: Sequence Diagram of an invitation's notification

Modify participation

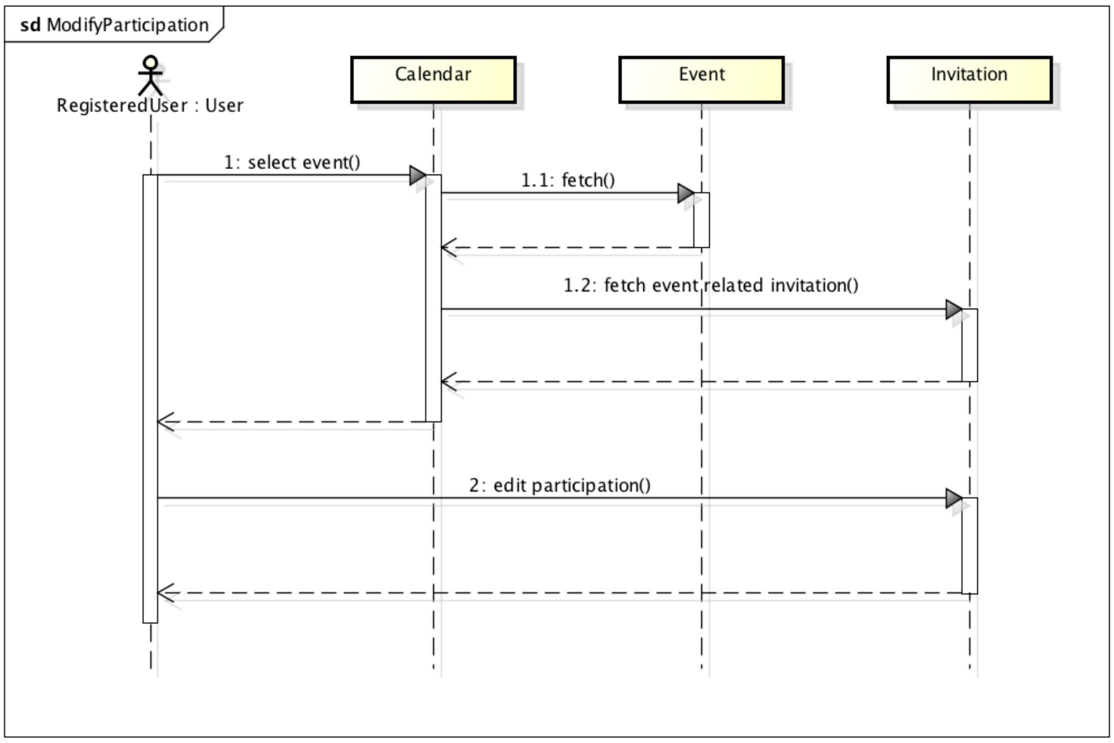


Figure 3.14: Modify participation Sequence Diagram

3.2 Alloy

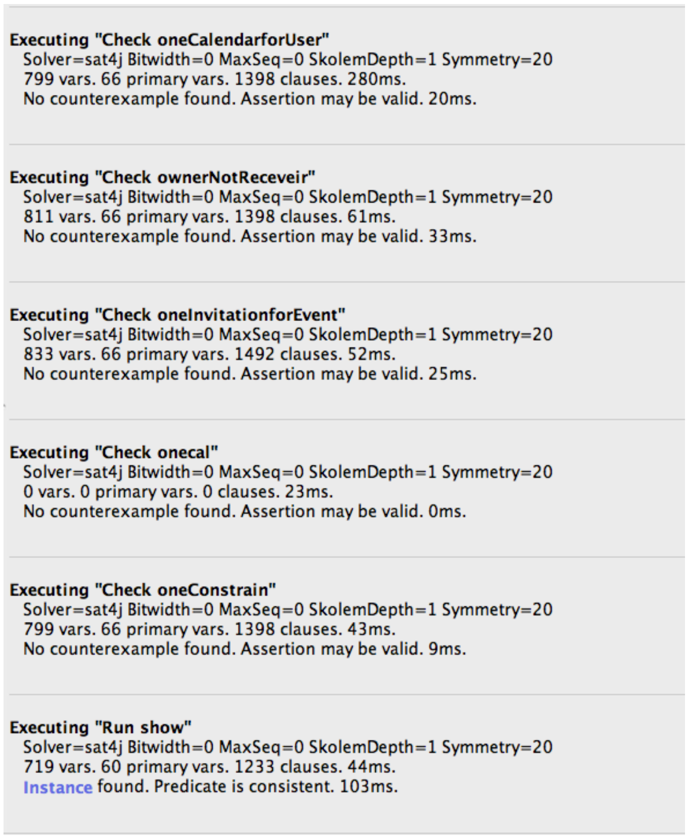
We use Alloy Analyzer to figure out if our Class Diagram was designed in a consistent way so that even our system will be implemented in a consistent style.
Below are reported the code used to represent our system and some models generated by the Alloy Analyzer tool.

3.2.1 Code

```
//SIGNATURE
sig RegisteredUser{
calendar:one Calendar
}
sig Invitation{
Receiver: some RegisteredUser,
event:one Event
}
sig WeatherConstrain{
}
sig Calendar{
}
sig Event{
 Owner:one RegisteredUser,
 constrain:one WeatherConstrain
}
//FACTS
fact calendarConstrain{
//the number of Calendar must be equals to the RegisteredUser
#Calendar=#RegisteredUser
//two or more RegisteredUser cannot refer to the same calendar
no disj r1,r2:RegisteredUser | r1.calendar=r2.calendar
}
fact eventConstrain{
//the number of Invitation must be equals to Event
#Event=#Invitation
//each event refer to a different weather constrain
#WeatherConstrain=#Event
no disj e1,e2:Event | e1.constrain=e2.constrain
}
fact invitationConstrain{
//two or more Invitation cannot refer to the same event
no disj r,p:Invitation | p.event=r.event //and p.Receiver = r.Receiver
//the owner of an event cannot receive the invitation of its event
all  disj e:Event , i1: Invitation | i1.event=e implies e.Owner not in
    i1.Receiver
}
//ASSERTION
assert oneCalendarforUser{
no disj r1,r2:RegisteredUser | r1.calendar=r2.calendar
}
//check oneCalendarforUser
assert  ownerNotReceveir{
all  disj e:Event , i1: Invitation | i1.event=e implies e.Owner not in
    i1.Receiver
}
//check ownerNotReceveir
assert oneInvitationforEvent{
no disj r,p:Invitation | p.event=r.event and p.Receiver = r.Receiver
}
```

```
//check oneInvitationforEvent
assert onecal{
#Calendar=#RegisteredUser
}
//check cal
assert oneConstrain{
no disj e1,e2:Event | e1.constrain=e2.constrain
}
//check oneConstrain
pred show(){}
run show
```

Listing 3.1: Alloy Analyzer code label



```
Executing "Check oneCalendarforUser"
  Solver=sat4j Bitwidth=0 MaxSeq=0 SkolemDepth=1 Symmetry=20
  799 vars. 66 primary vars. 1398 clauses. 280ms.
  No counterexample found. Assertion may be valid. 20ms.

Executing "Check ownerNotReceveir"
  Solver=sat4j Bitwidth=0 MaxSeq=0 SkolemDepth=1 Symmetry=20
  811 vars. 66 primary vars. 1398 clauses. 61ms.
  No counterexample found. Assertion may be valid. 33ms.

Executing "Check oneInvitationforEvent"
  Solver=sat4j Bitwidth=0 MaxSeq=0 SkolemDepth=1 Symmetry=20
  833 vars. 66 primary vars. 1492 clauses. 52ms.
  No counterexample found. Assertion may be valid. 25ms.

Executing "Check onecal"
  Solver=sat4j Bitwidth=0 MaxSeq=0 SkolemDepth=1 Symmetry=20
  0 vars. 0 primary vars. 0 clauses. 23ms.
  No counterexample found. Assertion may be valid. 0ms.

Executing "Check oneConstrain"
  Solver=sat4j Bitwidth=0 MaxSeq=0 SkolemDepth=1 Symmetry=20
  799 vars. 66 primary vars. 1398 clauses. 43ms.
  No counterexample found. Assertion may be valid. 9ms.

Executing "Run show"
  Solver=sat4j Bitwidth=0 MaxSeq=0 SkolemDepth=1 Symmetry=20
  719 vars. 60 primary vars. 1233 clauses. 44ms.
  Instance found. Predicate is consistent. 103ms.
```

Figure 3.15: Assertion verified

3.2.2 Model generated

In this section are reported some of the various scenarios representing the model of our system.

Interaction between all users

No interaction between the users

Interaction only between a few users