MeteoCal

Software Enginnering 2 Project

RASD

D. Pensa
M. Pini
A. Pintus

Contents

1 Introduction

1.1 Description

We will project and implement *MeteoCal*, a new weather based online calendar for helping people scheduling their personal events avoiding bad weather conditions in case of outdoor activities.

Users, once registered, should be able to create, delete and update events. An event should contain information about when and where the event will take place, whether the event will be indoor or outdoor. During event creation, any number of registered users can be invited. Only the organizer will be able to update or delete the event. Invited users can only accept or decline the invitation.

Whenever an event is saved, the system should enrich the event with weather forecast information (if available). Moreover, it should notify all event participants one day before the event in case of bad weather conditions for outdoor events. Notifications are received by the users when they log into the system.

1.2 Goals

MeteoCal provides a set of specific features which make this platform an absolute news for online calendars:

- Create a personal calendar and visit other users calendar
- Add events to the calendar and invite other users
- Integrate weather conditions and events
- Ricieve notifications for bad weather condition and modify the events
- Manage calendar and events either publicly or in total privacy

1.3 Domain Properties

The world where we suppose to develop *MeteoCal* should respect these conditions:

- When a user creates an event, it does exist in the real world
- Location and weather conditions informations are given from absolutely trusted internet providers
- If some event location can contain a precise number of participants, the creator user knows exactly this information before creating the event

1.4 Glossary

Here follows some important concepts which must be precisely defined:

- **User**: Someone who has signed up to *MeteoCal*
- **Organizer**: A user who created a particular event
- **Public**: Something that can be seen from any user
- **Private**: Something that is (partially) invisible to other users
- **User Page**: Personal page of each user, containing his/her calendar and some other informations
- **Notification**: Message received by a user directly from the system
- **Address Book**: User's personal list, containing name, surname and e-mail address of people in general, not necessarily other users

1.5 Assumptions

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

- The creator of an event can invite other people at the moment in which the event is created and also until the day the event takes place
- Each user is identified by his e-mail address, and he(she) has exactly one calendar: this means that users can search other users in order to see their calendar
- Calendars and Events can be either public or private, so there can be four situations:
  1. Public calendar with public event: any user can see the calendar and all the event details
  2. Public calendar with private event: any user can see the calendar but the users who are not invited to the event, can only see a "busy" mark on that day of the calendar, but not the event details
  3. Private calendar with public event: no one can see the calendar and of course its events, but if some user participates to the same event (which is public) and has a public calendar, the event will be visible on the calendar of that user
  4. Private calendar with private event: no one can see the calendar nor the event, except for people who are invated
- When a User decides to import an existing Calendar, it will overwrite the current Calendar; it should be written accordingly to a certain syntax and format that will be explained later on.
- To invite someone to an event, he(she) must be signed up with *MeteoCal*: from the moment in which the event has been created, the Organizer can invite other Users, choosing from his(her) Address Book.
- The creator of an event cannot create a new event which is going to be in a period of time overlapped to a pre-existing event in his(her) calendar.
- If a user receives an invitation to an event which is going to be in a period of time overlapped to a a pre-existing event in his(her) calendar, he(she) can choose to accept it but the old event will be deleted. The System will notify the User that the event has been successfully created.
- Each user can add to his(her) Address Book only *MeteoCal* Users: the system is going to check if that User exists or not. This ensures that only *MeteoCal* Users could be invited to events.
- Each user can set his(her) e-mail address as public or private: if it's public other users can add him(her) to his address book with just a click on a button on the user page (without doing it manually)

1.6 Identifying Stakeholders

MeteoCal is going to be a web platform with no limits regarding age, sex or particular abilities in real life, which means that it is not limited to a small target of people who have special interests or abilities. So we could consider the following stakeholders:

- **Companies** that want to organize conferences or other special events with all the employees or simply business dinners.
- **Sportsmen**, who are very interested in wheather conditions while organizing some event
- **Individuals** in general, who want to keep a precise schedule of their events, without the need of being always up-to-date with weather conditions, and of course invite their friends
- **Professor of SE2**, that is the "financial" stakeholder of this project, or simply the one who's interested in developing this new kind of application.

1.7 Proposed System

MeteoCal is going to be a very professional web platform that wants to be an absolute news with respect to the other platforms such as social networks. This means that there is no friendship concept nor wall sharing contents, but it focuses on organizing events and invite other people. There is no meaning in try to compete with Facebook™ or similars, because they offer yet a good way to create events but ignoring weather forecast functionalities that *MeteoCal* is going to offer.

1.7.1 Software Interfaces and Server Requirements

1. **DBMS**
   - Name: MySQL
   - Mnemonic: MySQL
   - Version number: 5.6.14
   - Source: `http://dev.mysql.com/downloads/mysql/`
2. **Application Server**
   - Name: GlassFish
   - Mnemonic: GlassFish
   - Specification number: Open source edition
   - Version number:4.0
   - Source: `https://glassfish.java.net/download.html`
3. **Operating System**: must support MySQL, Glassfish and the JVM (Java Virtual Machine)

1.7.2 Software Interfaces and Client Requirements

There's no limitation on Operating Systems but the web browser must support HTML 4/XHTML, CSS3, Javascript and, of course, the client must be on a stable internet connection.

2 Requirements

It is possible to categorize the actors involved in using *MeteoCal*, in particulare we have:

- **Guest**: someone who wants to access *MeteoCal* but not signed up
- **User**: someone who has signed up to *MeteoCal*
- **Organizer**: a user who created a particular event

2.1 Functional Requirements

Here it is the list of functional requirements for each actor of *MeteoCal*:

- **Guest**:
  1. Sign up to *MeteoCal*
- **User**:
  1. Log in
  2. Accept invite to events
  3. Decline invite to events
  4. See public events details
  5. See public events participants
  6. Create public events
  7. Create private events
  8. Invite users to an event he created
  9. Import calendar
  10. Export calendar
  11. Make the calendar public
  12. Find users
  13. See public calendars
  14. Recover lost password
- **Organizer**, who needs the same requirements of a normal user and also:
  1. Update events he(she) created
  2. Delete events he(she) created

The system must also satisfy these requirements:

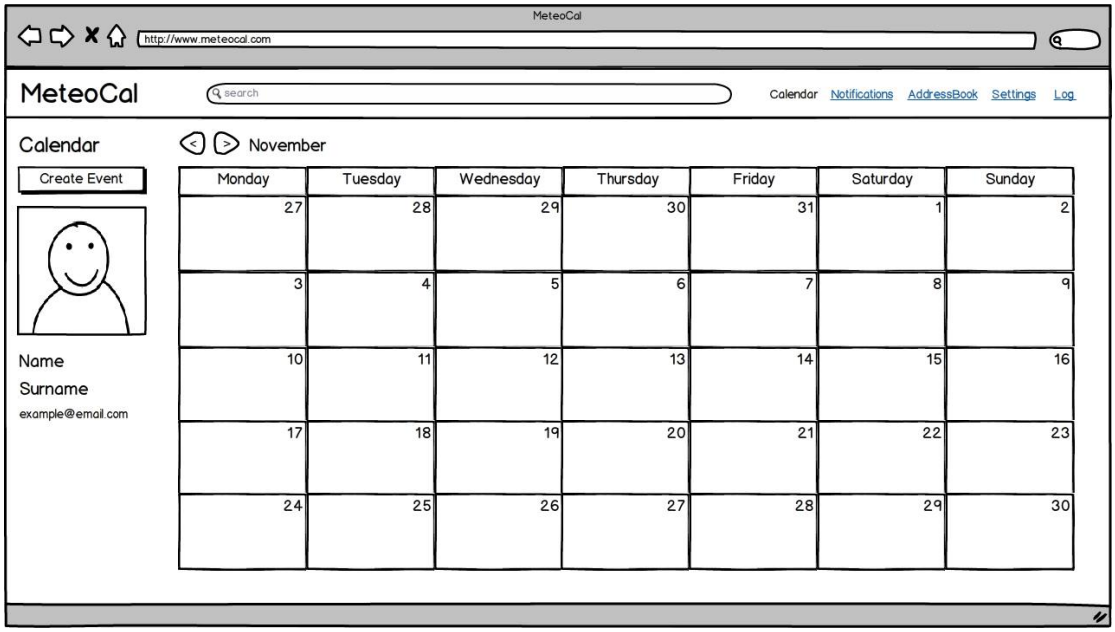
1. An event should contain information about when and where the event will take place, whether the event will be indoor or outdoor
2. During event creation, any number of registered users can be invited
3. Whenever an event is saved, the system should enrich the event with weather forecast information (if available)
4. The system should notify all event participants one day before the event in case of bad weather conditions for outdoor events. Notifications are received by the users when they log into the system
5. If user A has a public calendar, other users will see all the time slots in which A is busy but without seeing the details of the corresponding events, unless they have been defined as public.

6. In case of bad weather conditions for outdoor events, three days before the event, the system should propose to its creator the closest (in time) sunny day (if any).
7. The system will provide email notifications (both for invitation and cloudy outdoor events alerts)
8. Manage time consistency when creating an event by avoiding conflicts with existing events
9. Update weather information associated to events periodically, and of course, notify outdoor event participants in case the forecast has changed

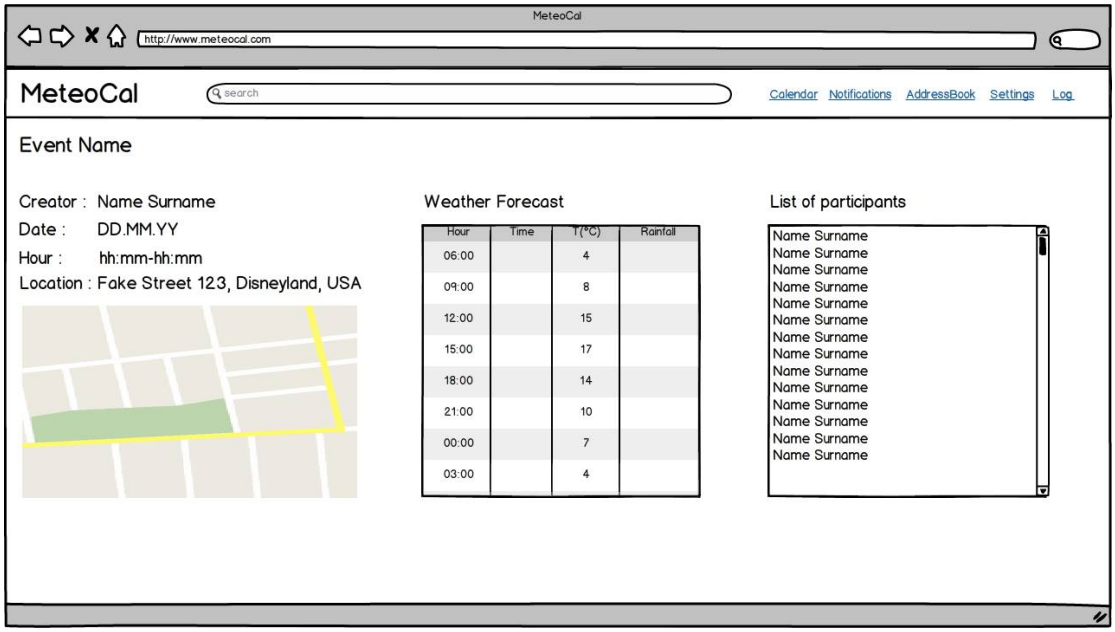
2.2 Non Functional Requirements

2.2.1 User Interface

The basic idea is to create a minimal but very effective user interface. Here we have a pair of possible web pages, just to figured out the idea of what are we looking for. This won't be exactly the final result, because design problems will be discussed later in the development.



Mockup of user interface



2.2.2 Documentation

Here it is the list of document regarding *MeteoCal* development:

- **Project Plan**: to define tasks and development timing
- **RASD**: Requirement Analysis and Specification Document
- **DD**: Design Document, to define the real structure of the web application and its tiers
- **JavaDoc** comments in the source code: to make anyone that wants to develop the platform or do maintenance on it, understand the code.
- **Installation Manual**: a guide to install *MeteoCal*
- **User Manual**: a guide to use *MeteoCal*
- **Testing Document**: a report of our testing of another *MeteoCal* project.

2.2.3 Architectural Considerations

MeteoCal will be developed with the J2EE platform and MySQL database in which all system's information will be stored. What is this information will be clearer watching the Class Diagram given below, because it could be considered as a basis for our database, but the precise ER Diagram will be drawn in the DD. An Internet connection is needed to use *MeteoCal* and also a recent web browser.

2.2.4 Other Considerations

- **Availability**: The system should be online 24h/7d, also during maintenance
- **Security**: Log in phase will be supported by password hashing according to last standards (MD5) and possibly in the future, connection will be under SSH protocol.
- **Maintainability**: There could be server-side modification or upgrades for future features, but never client-side maintenance
- **Portability**: *MeteoCal* only needs a web browser and an internet connection: it supports any kind of device

3 Scenarios Identifying

3.1 Cene Eleganti

Emilio is very sad because his friend Silvio, no more invites him to his "Cene Eleganti"; Emilio and Silvio are both registered to *MeteoCal*, but Emilio doesn't remember his password because it's a long time since his last login. So Emilio decides to use the recovery password option to restore his password. After that he decides to search for Silvio's calendar (which is public) but then he realizes that all Silvio's events are private and he cannot stalk him in anyway.

3.2 Bad Weather for Renato

Renato is a party planner and he needs a software that could help him to schedule every party he organizes. A friend of his suggestes him to sign up *MeteoCal* because of its features related to weather conditions, that would be very useful, especially for outdoor events. First of all he signes up *MeteoCal*, and then starts to use the calendar in order to insert the next party he was going to plan. The party will be on Friday and it will take place in a garden. On Tuesday (three days before the party) he receives an email from *MeteoCal*, notifying that Friday will be rainy, suggesting him to change the date into Sunday (because it's the first sunny day). Renato is an anxious person, so he decides to change the party location, choosing an indoor one. All the participants receive the change notification.

3.3 Lovely Lunch

Nichi wasn't doing fine using "weatherCal" a weather based online calendar, because it does not inform users if the weather forecast of an outdoor event has changed. So he decides to use *MeteoCal*, and after the sign up procedure, he starts to import his previous calendar (the one created with the "weatherCal" software). Then he decides to make his calendar as public, in order to make his friends see if he is busy or not. Two days later he creates a new public event for his birthday, and invites some friends who are *MeteoCal* users for a lunch in a French restaurant .

3.4 Beautiful

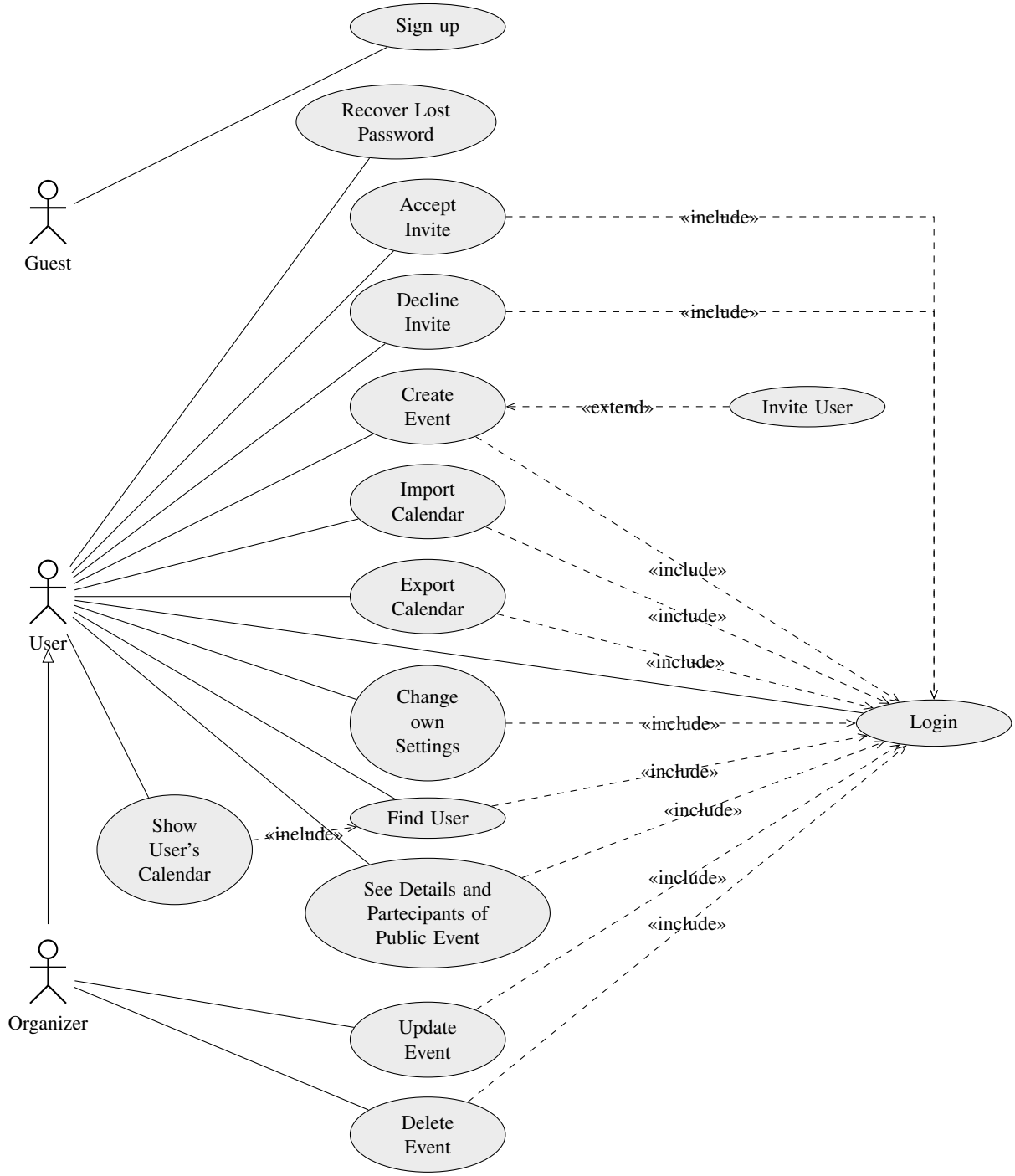
Maria Elena and Matteo are going to get married and they want to create a new event using *MeteoCal* for their marriage party. They create the event and send an invitation to their friends registered to *MeteoCal*. A week later Maria Elena decides to change the date of the party because in that day the place when they wanted to do the party is busy, so she changes the location into "Leopolda Station in Florence" and all the participants receive the notification. Some days later, Maria Elena catches another woman in bed with Matteo, and obviously she cancels the marriage and the party.

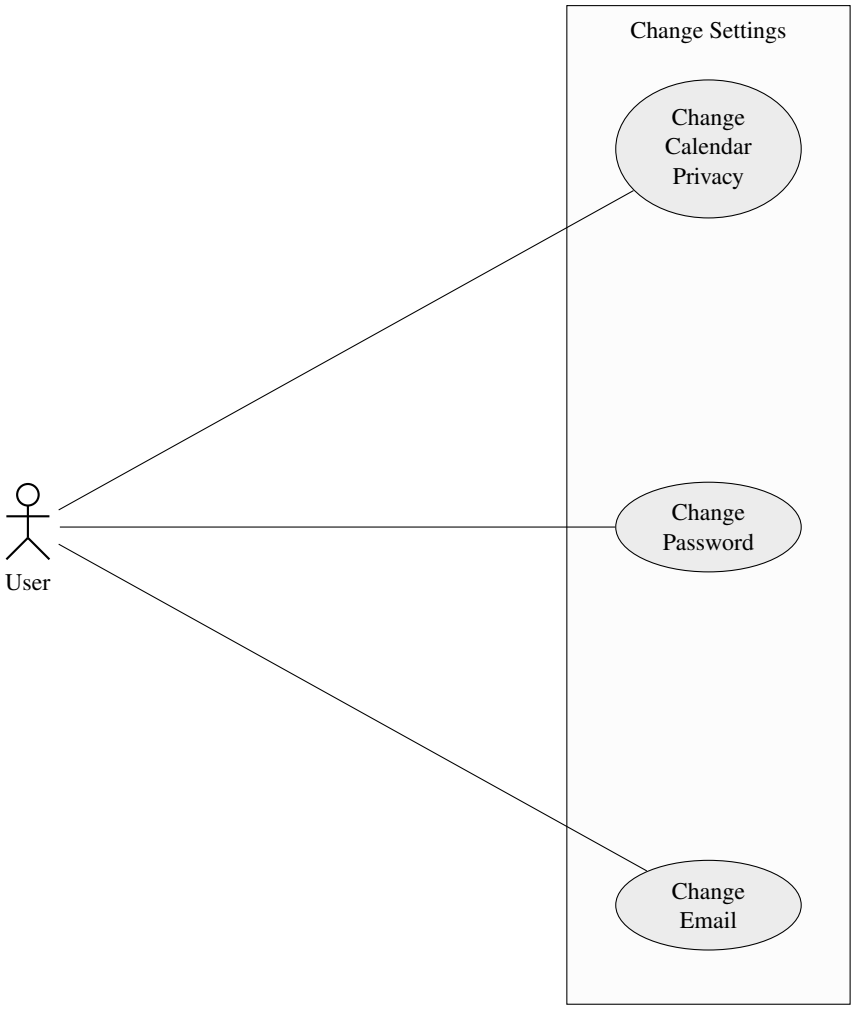
3.5 V-Day

Beppe wants to add GianRoberto (a colleague of his) into his AddressBook, but he doesn't know whether he is signed up to *MeteoCal*. So he uses the research function of *MeteoCal* inserting the GianRoberto's email (he had got it written in his papery address book) in order to check it. Once he finds it in *MeteoCal*, he add the contact to his AddressBook from GianRoberto's page. The next day Beppe, who has an empty Calendar, imports his own Calendar from his PC (which is already filled with his Events) and invites GianRoberto to the "V-Day" Event.

4 UML Models

4.1 Use Case Diagram





SignUp

Description	Guest sign up into the system
Goal	Guest wants to sign up to *MeteoCal* Application
Actors	Guest
Entry Condition	• Guest has not already registered • Guest is on HomePage
Exit Contition	The guest's information is stored into the database system
Flow of Events	1. Fulfill sign up's form 2. Click on "Sign Up" 3. The system verifies the inserted email 4. The system shows message of correct results of operation

Exceptions	**Wrong email:** email hasn't provider (@provider.com) **Email already exist:** email is already in the database In each case system shows an error message

Login

Description	User login into the system
Goal	User wants to use *MeteoCal* Application
Actors	User
Entry Condition	User is on home page
Exit Contition	User successfully logged into *MeteoCal* and he is on his personal page
Flow of Events	1. The user opens the login page of *MeteoCal* 2. The system show the requested login page 3. The user enters his e-mail address and password in the input form provided 4. The user clicks the button "log in" 5. The system verify inserted credentials 6. The system shows the user's personal page
Exceptions	**Wrong credentials:** the system displays an error message

Event Creation

Description	User creates new event
Goal	User wants to create new event
Actors	User
Entry Condition	User is on his userpage
Exit Contition	User successfully created event
Flow of Events	1. The users clicks on "New Event" button 2. The system show the requested event creation page 3. The user enters all requested data in the provided forms 4. The user clicks on "Create" button 5. The system checks timing of event 6. The system checks in the user's calendar that he doen't has another event at the same time 7. The system send invitations to the existing users 8. The users see confermation message on his userpage

Exceptions	**Wrong time or date:** the system asks to re-insert data **Time consistency failed:** the system show the relative error message **No existing user:** the system show which email aren't related to registered user

Email Changing

Description	User change his email
Goal	User wants to change email he use for the access to *MeteoCall*
Actors	User
Entry Condition	User is on UserPage
Exit Contition	User successfully changed his email and see confirmation message on his personal page
Flow of Events	1. The user clicks on "settings" 2. The system shows the requested settings page 3. The user fulfills changeEmail form 4. The user clicks on "change" button 5. The system verifies the inserted email 6. The system shows the userpage with a confirmation message
Exceptions	**Wrong email:** email hasn't provider (@provider.com) **Email already exist:** email is already in the database In each case system shows an error message

Accept/Decline Invitation

Description	User accepts or decline the event invitation
Goal	User wants (or not) to partecipate to the event
Actors	User
Entry Condition	The system has just loaded the userpage
Exit Contition	User successfully added to event partecipants (or the invite was completly destroyed)
Flow of Events	1. The users clicks on “Notifies” button 2. The system shows user’s notifies 3. The users clicks on selected notify 4. The system verifies if the user has another event on the same time 5. The system show choosing Page 6. The user clicks on “Accept” or “Refuse” 7. In case the user has accepted the invite, the system add him to event partecipant and add the event to his calendar 8. In case the user has not accepted the invite, the system removes the invitation 9. In each case the user will be redirect to his personal page
Exceptions	**User has another event:** the system shows a warning notifing the user that if he accept the event, this will be overwritten

Export Calendar

Description	User exports his calendar
Goal	User wants to save own calendar
Actors	User
Entry Condition	The user is on userpage
Exit Contition	User successfully saved his calendar on file
Flow of Events	1. The users clicks on “Settings” button 2. The system shows the requested settings page 3. The users clicks “Export Calendar” button 4. The system starts download operation
Exceptions	There are no exceptions

Import Calendar

Description	User imports his calendar
Goal	User wants to use his offline calendar on *MeteoCal*
Actors	User

Entry Condition	The user is on userpage
Exit Contition	User successfully overwrite his *MeteoCal*'s calendar with an other offline one
Flow of Events	1. The users clicks on "Settings" button 2. The system shows the requested settings page 3. The users clicks "Choose Offline Calendar" button 4. The browser shows popup windows with user's folders 5. The user chooses a file 6. The user clicks on "Load" button 7. The system starts upload operation 8. The system verifies the correctly structure of the file 9. The system overwrites user's calendar
Exceptions	**File not Valid:** the system shows an error

Change calendar's privacy

Description	User change his calendar's privacy
Goal	User wants to make his calendar public(private)
Actors	User
Entry Condition	The user is on userpage
Exit Contition	User successfully changes his calendar's privacy
Flow of Events	1. The users clicks on "Settings" button 2. The system shows the requested settings page 3. The users clicks "MakePublic" (or "MakePrivate") button 4. The system updates user's info on the database 5. The system reloads settings page
Exceptions	There are no exceptions

Show an user calendar

Description	User watch an other user's calendar
Goal	User wants to see another calendar
Actors	User
Entry Condition	The user is on userpage
Exit Contition	User successfully see another calendar
Flow of Events	1. The user types email of an other user in the search box 2. The system shows him the user's calendar

Exceptions	**No email found:** The system shows an error **Email found but secret:** The system shows an error

See Details of Event

Description	User see all details of a public event
Goal	User wants to see all details of a public event
Actors	User
Entry Condition	The user is on any userpage
Exit Contition	User successfully see all detail of a public event
Flow of Events	1. The users clicks on name of event he wants to see details 2. The system verifies that the event is public 3. The system shows the event page with all information
Exceptions	**Event is Private:** the system shows an error

Update Event

Description	Organizer update the event that he has created
Goal	Organizer wants to update the event
Actors	Organizer
Entry Condition	Organizer is on the event page
Exit Contition	• Organizer successfully modify the event • the partecipants receive the notification of the update

Flow of Events	1. The organizer click on “Modify” button 2. The system shows modifyEvent page 3. The organizer changes the settings he wants to modify as if he is creating new event 4. When the organizer ended the modify, clicks on “Save Modify” button 5. The system checks timing of event 6. The system checks in the user’s calendar that he had not another event at the same time 7. The system update the event in the database 8. The system sends a notification with the new info to all partecipants 9. The system redirect the organizer on the event page
Exceptions	**Wrong time or date:** the system asks to re-insert data **Time consistency failed:** the system show the relative error message

Delete Event

Description	Organizer delete the event that he has created
Goal	Organizer wants to delete the event
Actors	Organizer
Entry Condition	Organizer is on the event page
Exit Contition	• Organizer successfully delete the event • the partecipants receive the notification of deletion
Flow of Events	1. The organizer click on “Modify” button 2. The system shows modifyEvent page 3. The organizer clicks on “Delete Event” button 4. The system delete the event in the database 5. The system sends notification of the deletion to all partecipants 6. The system redirect the organizer on his personal page
Exceptions	There is no exceptions

Recover Lost Password

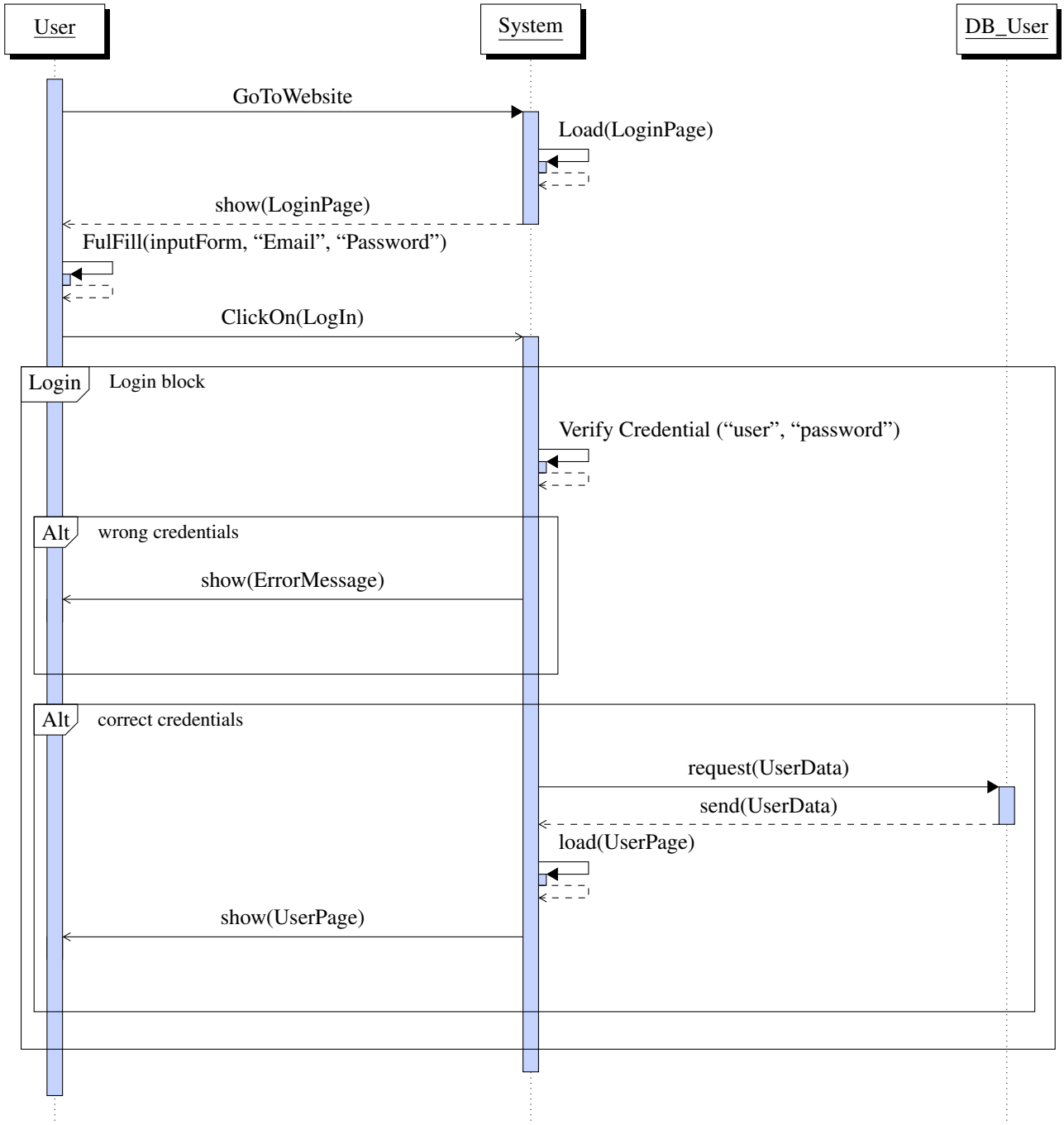
Description	User losts his password

Goal	Guest wants to recover his password to login in *MeteoCal* Application
Actors	User
Entry Condition	User is on HomePage
Exit Contition	The user has recovered his password
Flow of Events	1. The user inserts his email in the login form 2. The user clicks on “Lost Password?” button 3. The system sends him an email with a link to change the password (if the email exist in the database) 4. The user clicks on that link 5. The system shows PasswordRecovery page 6. The user inserts a new password in the provided form 7. The user clicks on “Change Password” button 8. The system redirect user on the homepage 9. The system shows message of correct results of operation
Exceptions	**Email Not Exist:** the system shows an error message

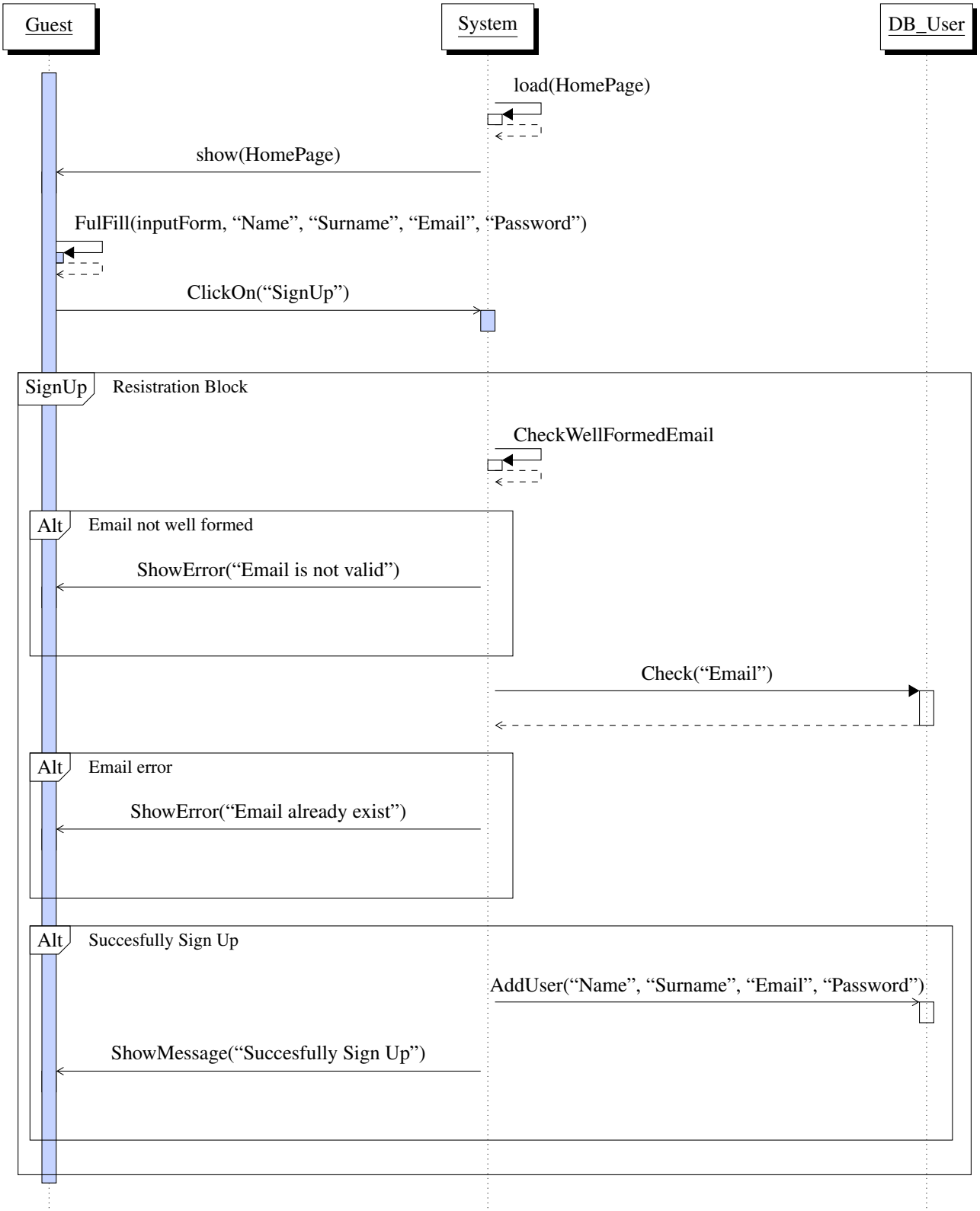
4.2 Sequence Diagram

The following diagrams aren't perfectly in accordance with standards, due to libraries we have used in latex.

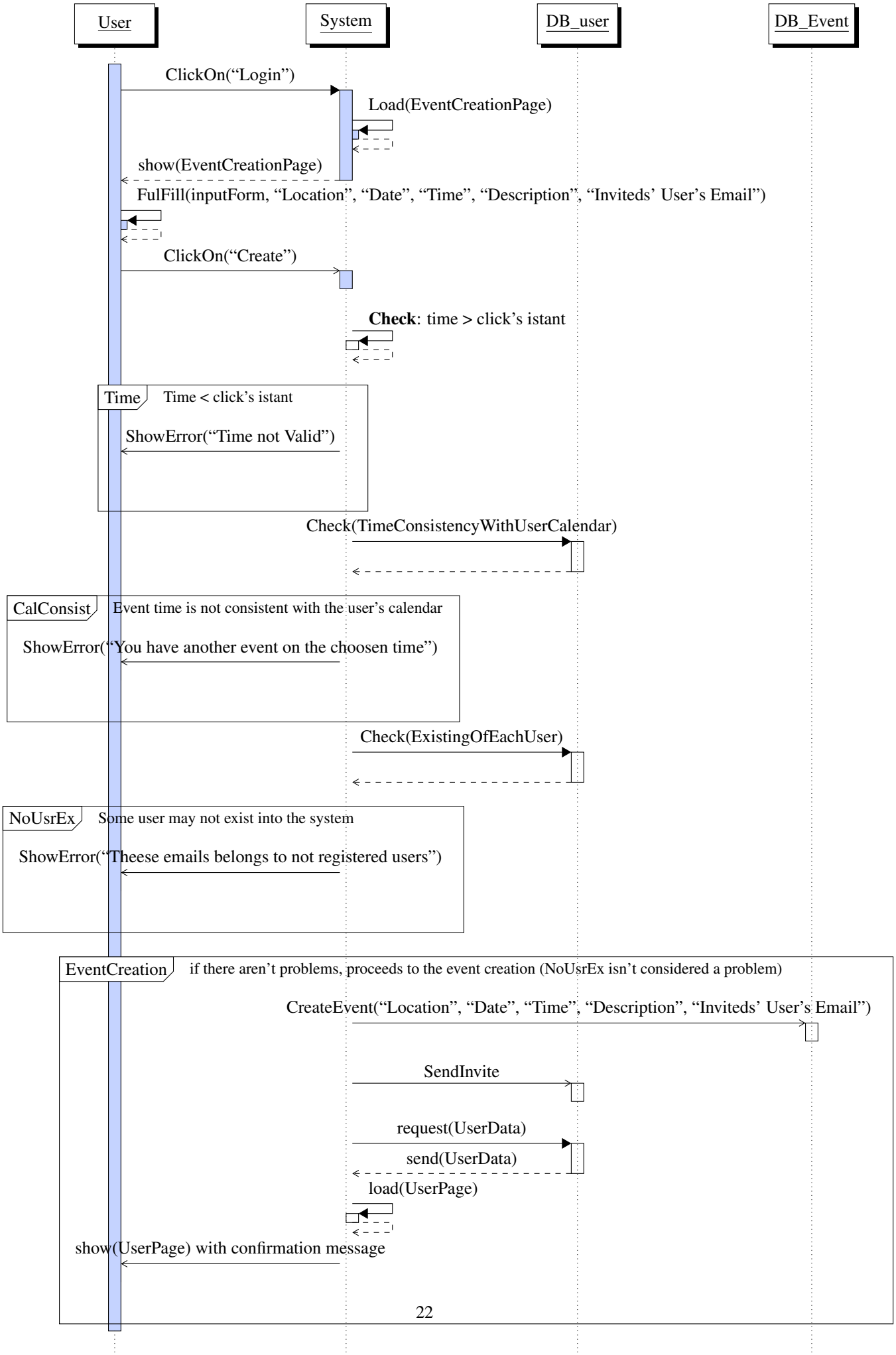
Login



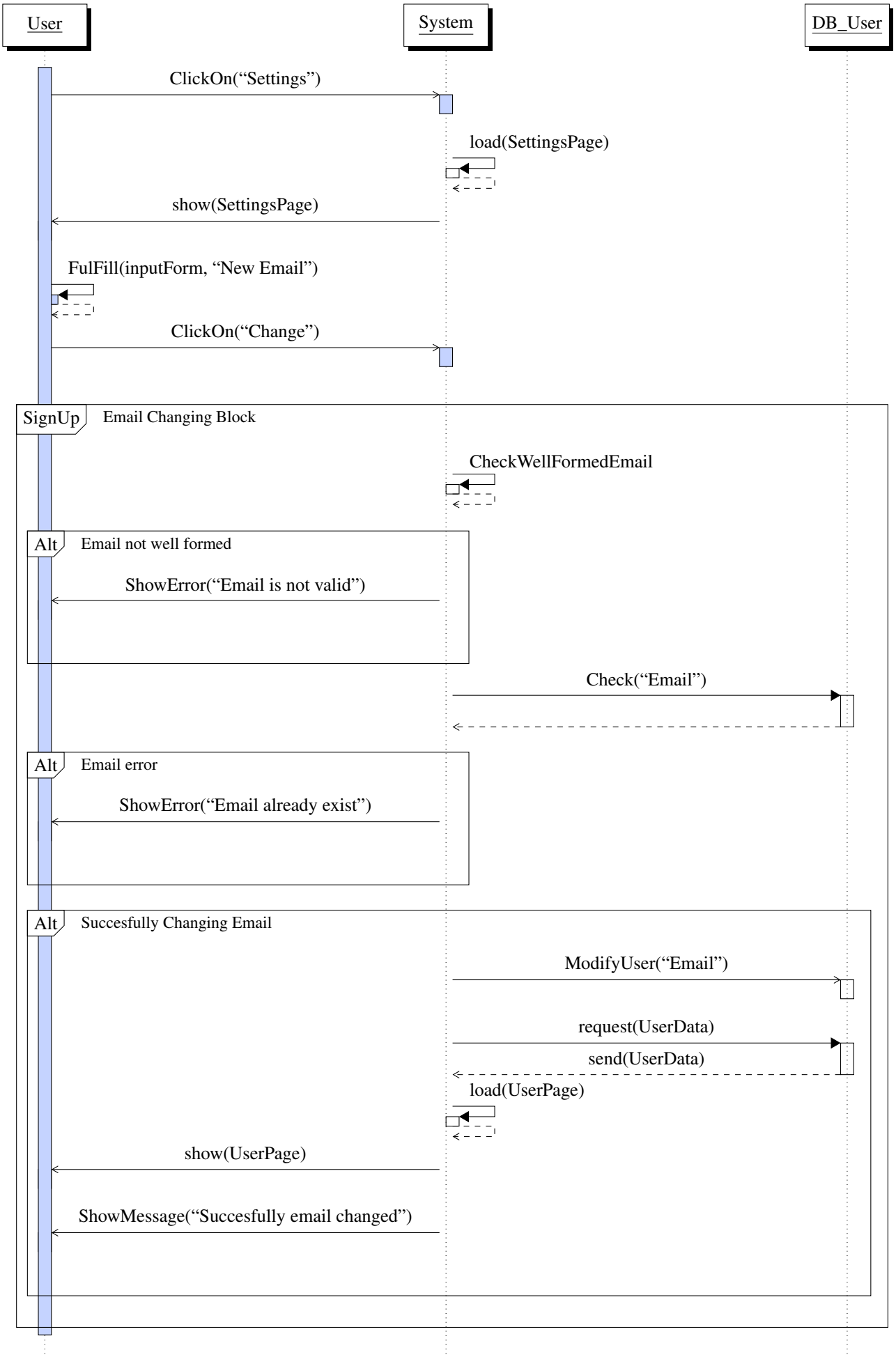
SignUp


Guest
System
DB_User
load(HomePage)
show(HomePage)
FulFill(inputForm, “Name”, “Surname”, “Email”, “Password”)
ClickOn(“SignUp”)
SignUp
Resistration Block
CheckWellFormedEmail
Alt
Email not well formed
ShowError(“Email is not valid”)
Check(“Email”)
Alt
Email error
ShowError(“Email already exist”)
Alt
Succesfully Sign Up
AddUser(“Name”, “Surname”, “Email”, “Password”)
ShowMessage(“Succesfully Sign Up”)

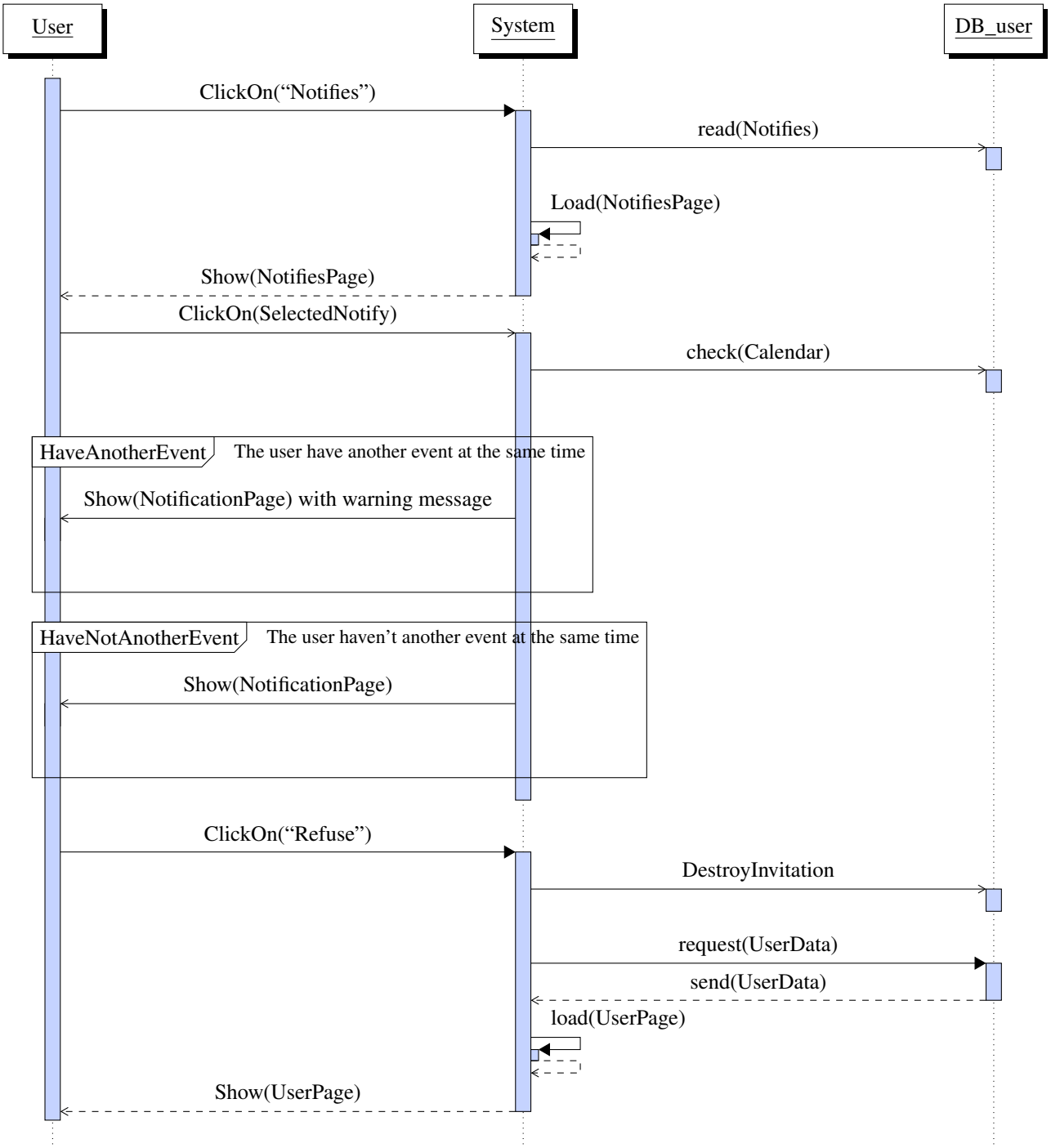
Event Creation



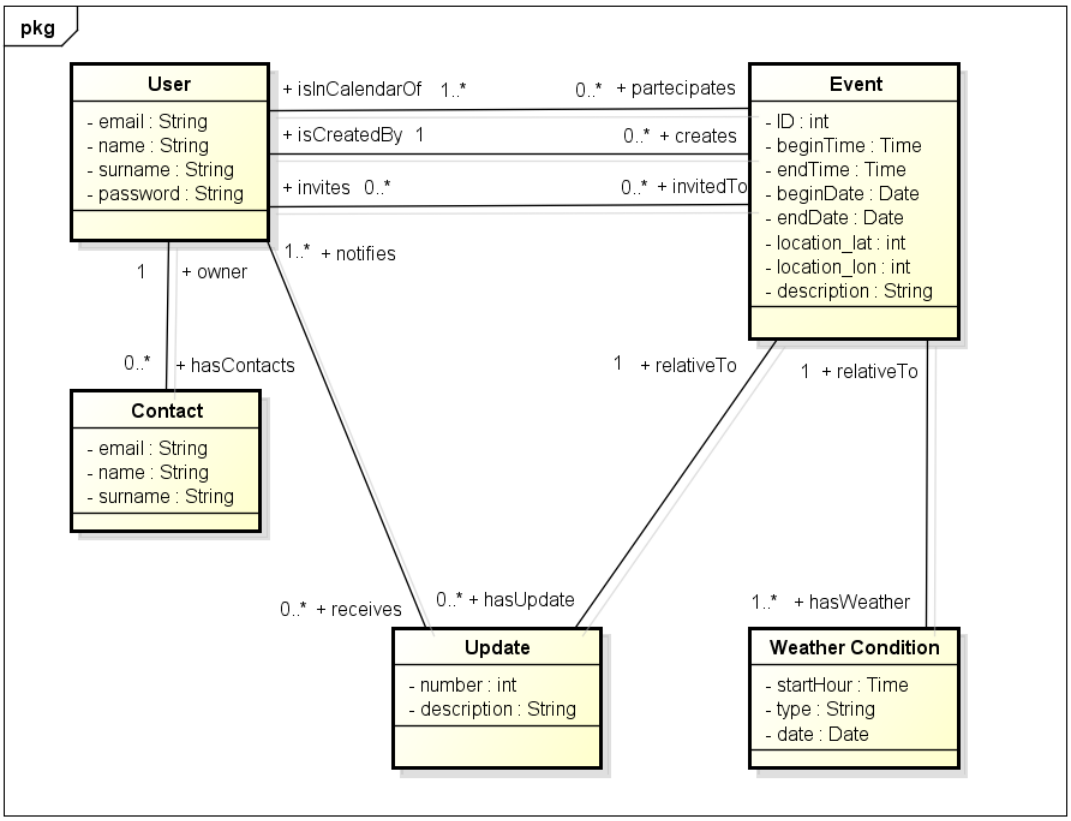
Email Changing



Decline Invitation



4.3 Class Diagram



5 Alloy Modelling

Listing 1: MeteoCal Alloy

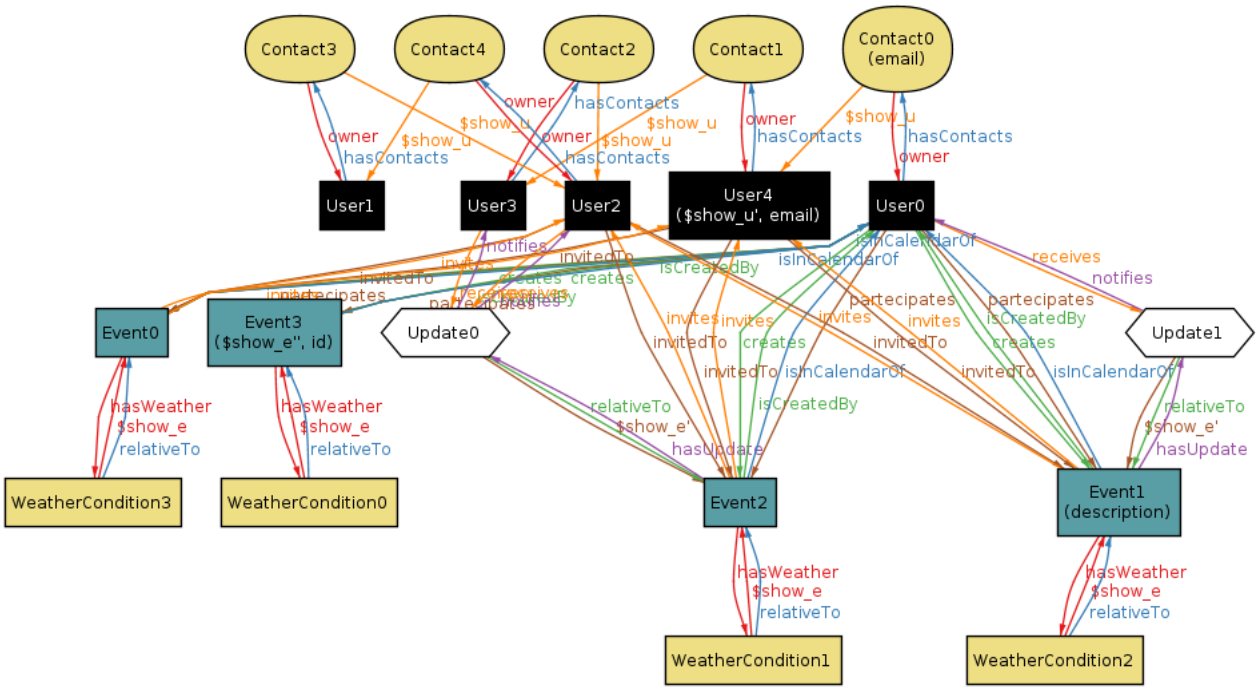
```
module MeteoCal

sig Text{}
sig Email{}
sig ID{}


sig User{
    email: one Email,
    name: one Text,
    surname: one Text,
    password: one Text,
    partecipates: set Event,
    creates: set Event,
    invitedTo: set Event,
    hasContacts: set Contact,
    receives: set Update
}

sig Contact{
    email: one Email,
    name: one Text,
    surname: one Text,
    owner: one User
}

sig Event{
    id: one ID,
    beginTime: one Int,
    endTime: one Int,
    beginDate: one Int,
    endDate: one Int,
    location_lat: one Int,
    location_lon: one Int,
    description: one Text,
    hasWeather: one WeatherCondition,
    isInCalendarOf: some User,
    isCreatedBy: one User,
    invites: set User,
    hasUpdate: set Update
}

sig Update{
    number: one Int,
    description: one Text,
    relativeTo: one Event,
    notifies: some User
}

sig WeatherCondition{
    type: one Text,
    relativeTo: one Event
}

//FACTS

fact inverseRelationship{
    all u: User | all e: Event | e in u.partecipates <=> u in e.isInCalendarOf
    all u: User | all e: Event | e in u.creates <=> u in e.isCreatedBy
    all u: User | all e: Event | e in u.invitedTo <=> u in e.invites

    all u: User | all c: Contact | c in u.hasContacts <=> u=c.owner

    all u: User | all up: Update | up in u.receives <=> u in up.notifies

```

```
66      all u: Update | all e: Event | u in e.hasUpdate <=> e=u.relativeTo
67
68      all e: Event | all wc: WeatherCondition | wc=e.hasWeather <=> e=wc.relativeTo
69  }
70
71  fact unique{
72      all disj u1,u2: User | u1.email != u2.email
73
74      all disj e1,e2: Event | e1.id != e2.id
75
76      all disj c1, c2: Contact | all u: User | c1 in u.hasContacts and c2 in u.
        hasContacts implies c1.email != c2.email
77
78      all disj up1,up2: Update | all u: User | up1 in u.receives and up2 in u.receives
        implies up1.number != up2.number
79
80  }
81
82  fact existance{
83      all c: Contact | some u: User | c.email = u.email
84
85      all wc: WeatherCondition | some e: Event | wc in e.hasWeather
86
87      all u: Update | some e: Event | u in e.hasUpdate
88
89      all e: Email | some u: User | e=u.email
90
91      all i: ID | some e: Event | i=e.id
92
93
94  }
95
96  fact {
97      //each contact in each user has different email
98      all c: Contact | all u: User | c in u.hasContacts implies c.email != u.email
99
100     //Start Event < End Event
101     all e: Event | e.beginTime < e.endTime or e.beginDate < e.endDate
102
103     //creator has event in calendar
104     all e: Event | all u: User | u=e.isCreatedBy implies e in u.partecipates
105
106     //creator is not invited to his event
107     all e: Event | all u: User | u=e.isCreatedBy implies e not in u.invitedTo
108
109     //partecipation implies invite or creation
110     all u: User | all e: Event | (e in u.partecipates and u != e.isCreatedBy) implies
        e in u.invitedTo
111
112     //contact same email => same name, surname
113     all c: Contact | all u: User | c.email = u.email implies (c.name=u.name and c.
        surname=u.surname)
114 }
115
116 //PREDICATES
117
118 pred addEvntToCalendar[u,u': User, e: Event]{
119     u'.partecipates=u.partecipates+e
120 }
121
122 pred removeEvntFromCalendar[u,u': User, e: Event]{
123     u'.partecipates=u.partecipates-e
124 }
125
126 //ASSERTS
127
128 assert eventInAtLeastOneCalendar{
129     all e: Event | some u: User | e in u.partecipates
130 }
```

```
131 //check eventInAtLeastOneCalendar
132
133 assert chkAddEvntToCalendar{
134     all e: Event |  all u, u': User |
135         addEvntToCalendar[u, u', e] implies e in u'.partecipates
136 }
137 //check chkAddEvntToCalendar
138
139 assert chkRemoveEvntFromCalendar{
140     all e: Event |  all u, u': User | (addEvntToCalendar[u,u',e] and
    removeEvntFromCalendar[u, u', e]) implies u=u'
141 }
142 //check chkRemoveEvntFromCalendar
143
144
145 //SHOWS
146
147 pred showContacts(){#Event=0}
148 //run showContacts for 5
149
150 pred showEvent(){#Contact=0}
151 run showEvent for 3
152
153 pred show(){}
154 run show for 5
```

5.1 Generated World

In this paragraph we report some worlds generated by *Alloy Analizer* in order to make understand that our model is consistent.
We report below a general world generated, and three asserts verified with the *Alloy Analizer*. Further specific representations are reported below in order to better understand the characteristics of the model.




```
Executing "Run show"
  Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
  6626 vars. 429 primary vars. 13401 clauses. 38ms.
  Instance found. Predicate is consistent. 29ms.
```

Assert Verification

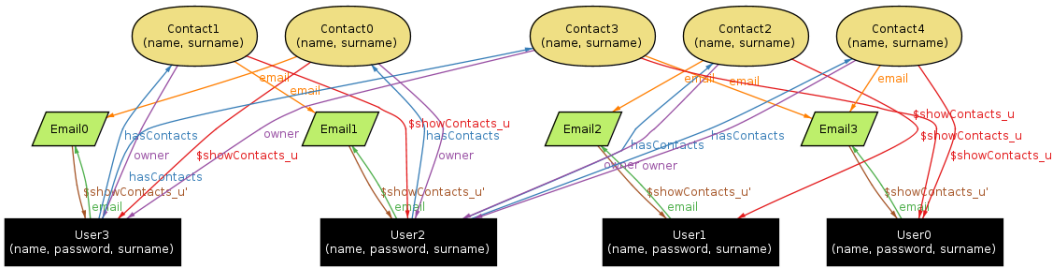
```
Executing "Check chkAddEvntToCalendar"
  Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
  6723 vars. 438 primary vars. 13589 clauses. 30ms.
  No counterexample found. Assertion may be valid. 2ms.
```

```
Executing "Check chkRemoveEvntFromCalendar"
  Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
  6741 vars. 438 primary vars. 13622 clauses. 42ms.
  No counterexample found. Assertion may be valid. 2ms.
```

```
Executing "Check eventInAtLeastOneCalendar"
  Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
  6656 vars. 432 primary vars. 13442 clauses. 35ms.
  No counterexample found. Assertion may be valid. 9ms.
```

5.1.1 Contact World

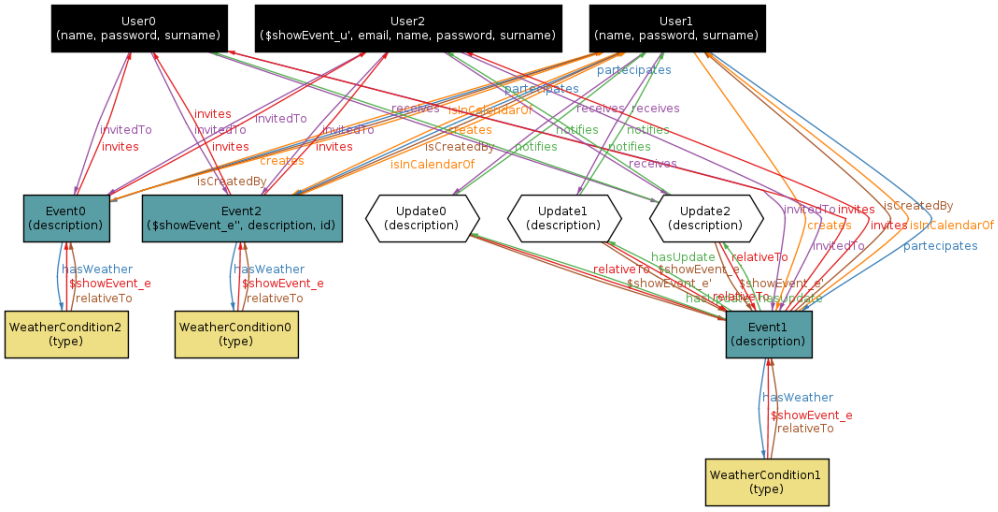
In this section is reported an example of smaller world about how contacts work.



Executing "Run showContacts for 5"
Solver=sat4j Bitwidth=4 MaxSeq=5 SkolemDepth=1 Symmetry=20
14459 vars. 1350 primary vars. 29135 clauses. 63ms.
Instance found. Predicate is consistent. 27ms.

5.1.2 Event World

In this section is reported an example of smaller world about how events work.



Executing "Run showEvent for 3"
Solver=sat4j Bitwidth=4 MaxSeq=3 SkolemDepth=1 Symmetry=20
6424 vars. 630 primary vars. 13401 clauses. 21ms.
Instance found. Predicate is consistent. 17ms.

6 Working Hours Schedule

	Pensa	**Pini**	**Pintus**	**h/sect**
Problem Analysis	2	2	2	**6**
Non Functional Requirements	6	4	5	**15**
Functional Requirements	6	4	3	**13**
Alloy Specifications	4	9	8	**21**
h/pers	**18**	**19**	**18**	**55**

7 Changing History

2014/11/21:

- Layout
- Class Diagram
- Alloy