SOFTWARE ENGINEERING PROJECT

METEOCAL

DESIGN DOCUMENT

Filippo Pellolio
Andrea Rottigni

Milan, 7-th December 2014

1 INTRODUCTION

1.1 PURPOSE

This software design document describes MeteoCal architecture and design. MeteoCal is a platform that has to be developed as part of software engineering course. The document will explain the architectural and design decisions made to realize as best as possible, according to our opinion, MeteoCal.

1.2 SCOPE

The software product that will be delivered is MeteoCal, which is a weather-based calendar that helps people organizing their events in the most suitable day according to the weather conditions. The final users of the system are:

- **The MeteoCal users:** they can create and manage calendars or events. The can search for other user profile and see their calendar if it is public. They can also search for public events.
- **The MeteoCal administrator:** he can manage default event type, adding new event types or removing them.

1.3 DEFINITIONS, ACRONYMS AND ABBREVIATIONS

- **Guest:** a person who is either not registered or not logged in yet.
- **User:** a person who is registered and already logged in. He can exploit all the functionalities provided by the system to users.
- **Administrator:** the person who is in charge of managing event type in the system.
- **Event:** an event is an activity scheduled in a specific date with a specific owner, its creator, and possibly some participants. It must have a specific event type.
- **Event type:** the type of the event. At each type is associated a weather condition which is the most adapt for performing the event. The can be either default or personalized.
- **Notification:** a message sent by the system to users. There are several types of notification that inform users about events.
- **ER:** entity relationship diagram.
- **BCE:** boundary control entity diagram.
- **JEE:** Java Enterprise Edition.

1.4 OVERVIEW

As already said this document is about MeteoCal architecture and design.

In this first introductory section we provided some general information about the document.

The second section contains a description of the JEE architecture, which is the referential architecture for our project. Then there is a list of all sub-systems in which our system can be divided in order to make easier the analysis and, probably, also the implementation phase.

In the third section we basically provided the structure of the relational database in which persistent data will be stored. Firstly we sketched some ER diagrams and then the logic model of the database.

The fourth section contains some UX diagrams, which give a better understanding on the actions that a user, a guest and the administrator can perform and how they can do such actions.

In the fifth section there are some BCE diagrams. We choose to use this kind of diagrams because they are very similar to model view controller pattern.

The sixth section shows the connection between the actors of MeteoCal and boundaries, controller and entities, contained in the BCE diagrams.

In the last section there is a list of the used tool in order to realize this document.

2 ARCHITECTURE DESCRIPTION

2.1 JEE ARCHITECTURE

We start the description of our application with a short overview on JEE architecture in order to give an initial idea about the structure of our application.

JEE platform uses a multitiered application model as shown in the following picture.

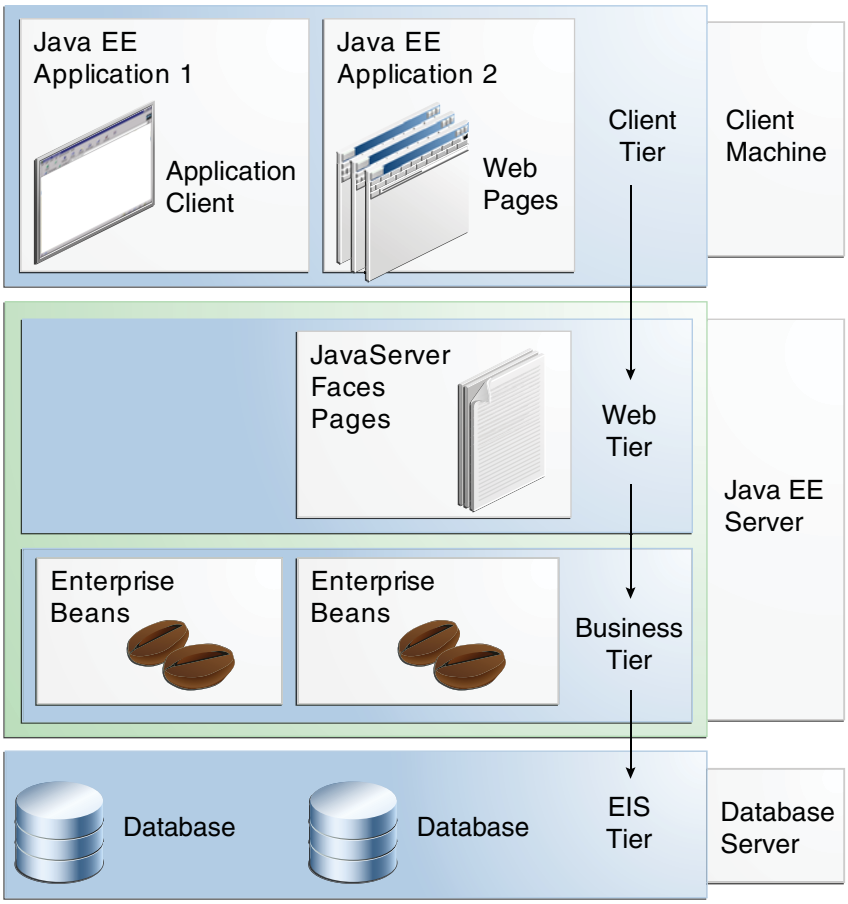


Fig. 1 Multitiered Architecture

In particular, we can identify 4 tiers:

- **Client tier:** a JEE client is either an application client or a web client that interacts directly with the actors. In our project we will develop a web client, which basically consist in two parts: dynamic web pages and a web browser, which display the web pages.
- **Web tier:** It contains either Servlets or Web pages created using JSF, JavaServer Faces technology. Servlets dynamically process requests from the client tier,

forwarding to the business tier the data collected, and produce responses.

- **Business tier:** It is composed by Enterprise JavaBeans, which contain the business logic of the application, and by Java Persistence Entities, which allow retrieving data from the storage.
- **EIS tier:** the EIS tier handles EIS software. In our case it includes a database system where all the relevant data are stored.

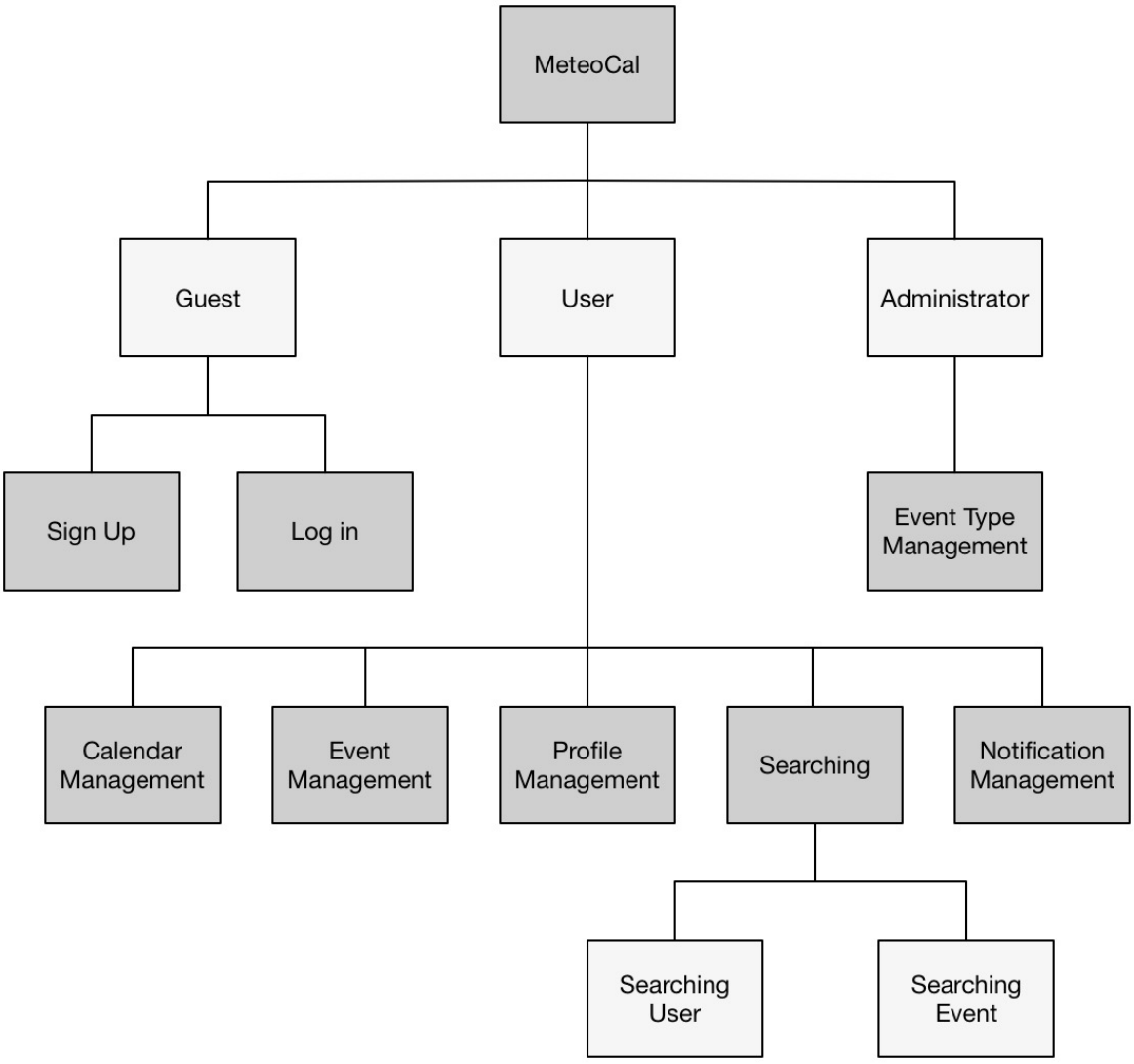
2.2 IDENTIFYING SUBSYSTEMS

We think it may be useful to divide our system in some sub-systems in order to easier understand problems that we will face in the developing process. To obtain the sub-systems we have detect groups of functionality that have something in common, such as operate on similar data.

We can derive the following sub-systems through the analysis of our system:

- Guest sub-system
    - Sing Up sub-system
    - Log in sub-system
- User sub-system
    - Calendar management sub-system
    - Event management sub-system
    - Profile management sub-system
    - Notification management sub-system
    - Searching sub-system
        - Searching user sub-system
        - Searching event sub-system
- Administrator sub-system
    - Event type management sub-system

To clarify the described subdivision in subsystem we report below a representation.


MeteoCal
Guest
User
Administrator
Sign Up
Log in
Event Type Management
Calendar Management
Event Management
Profile Management
Searching
Notification Management
Searching User
Searching Event

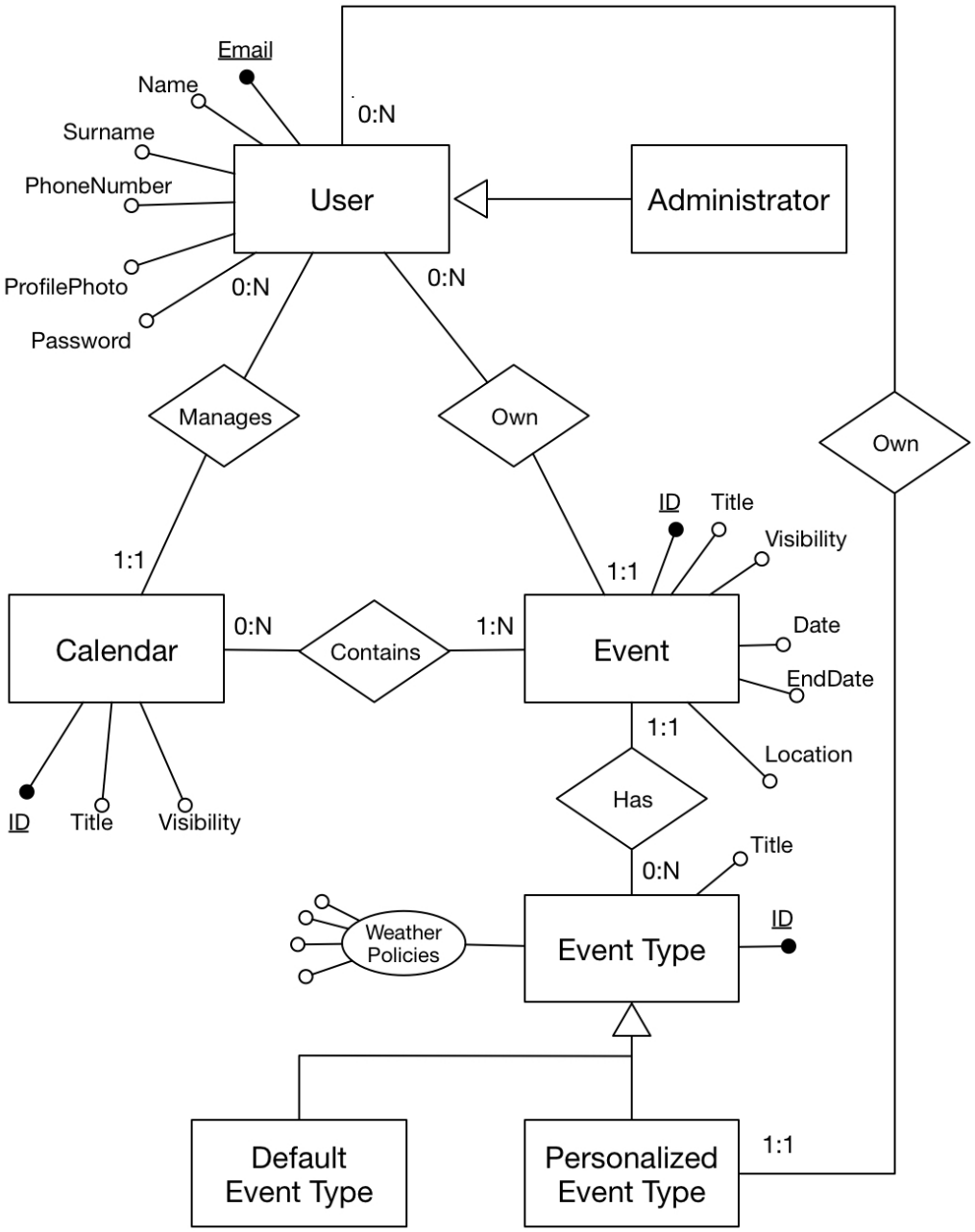
3 PERSISTENT DATA MANAGER

We will organize the data our application has to store in a relational database. In the next section we will represent database, at first conceptually with an ER Diagram, then we will show the real model we used to build our database.

3.1 ER DIAGRAM

The diagrams that will follow represent two parts of the same database, we split them to allow the reader to better understand the whole database: the complete model would have been very difficult to read.

This first model represents the correlation between Users, Calendars and Events:

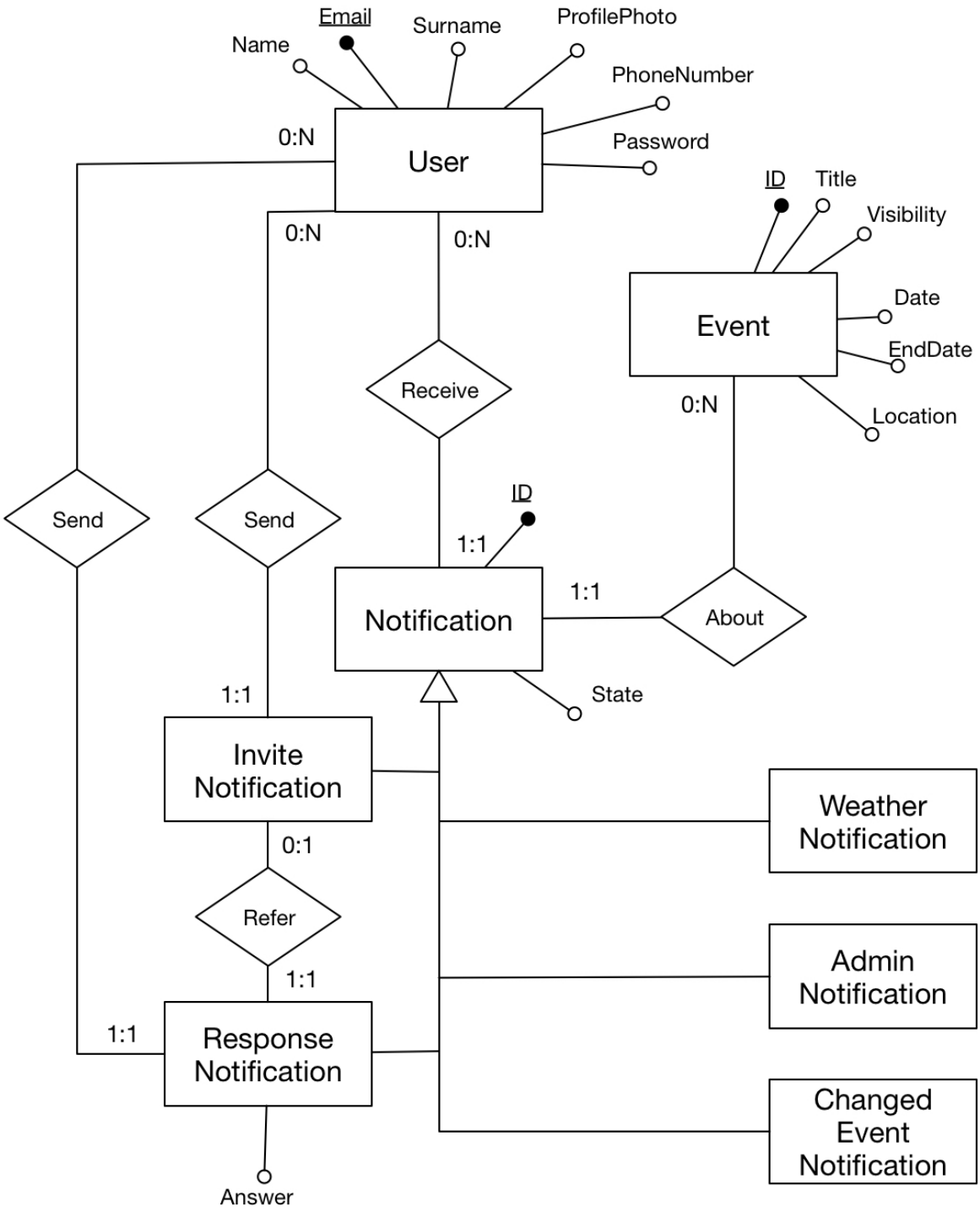


As we explained in the RASD, our model comprehends three types of actors: *Guests, Users* and *Administrators.* Only two of these three actors have something that needs to be stored in the database, in fact, as we already said, when a guest register his data in the database he becomes a user. That's why there isn't any entity that refers to the guest.

Every User manages a number of Calendars that can also be zero; every calendar is associated in a "many to many" relationship with a certain number of events and every event has been created by a user who is its owner. This triangular relationship represents how we manage the user schedule.

Every Event is associated with one and only one type that can be one of the defaults or one created by the same user who created the Event. Every Event Type holds a list of all the possible weather conditions and chooses among those which one are feasible for that type.

In the next page is shown the part of the database in charge of representing the notification for the user, which is a little bit more complex.



Every user receives a variable number of notifications and the state attribute shows if he read them or not. Every notification is about one and only one Event.

A User can invite another user to an Event: the invited one will receive an InviteNotification regarding the chosen Event. When a User receives an InviteNotification he can answer to the invite, doing that he will send a ResponseNotification to the sender of the invite stating his answer.

A ResponseNotification always refers to an InviteNotification.

3.2 LOGICAL DESIGN

What follows is going to be the real implementation of the database we will use in our MeteoCal web application.

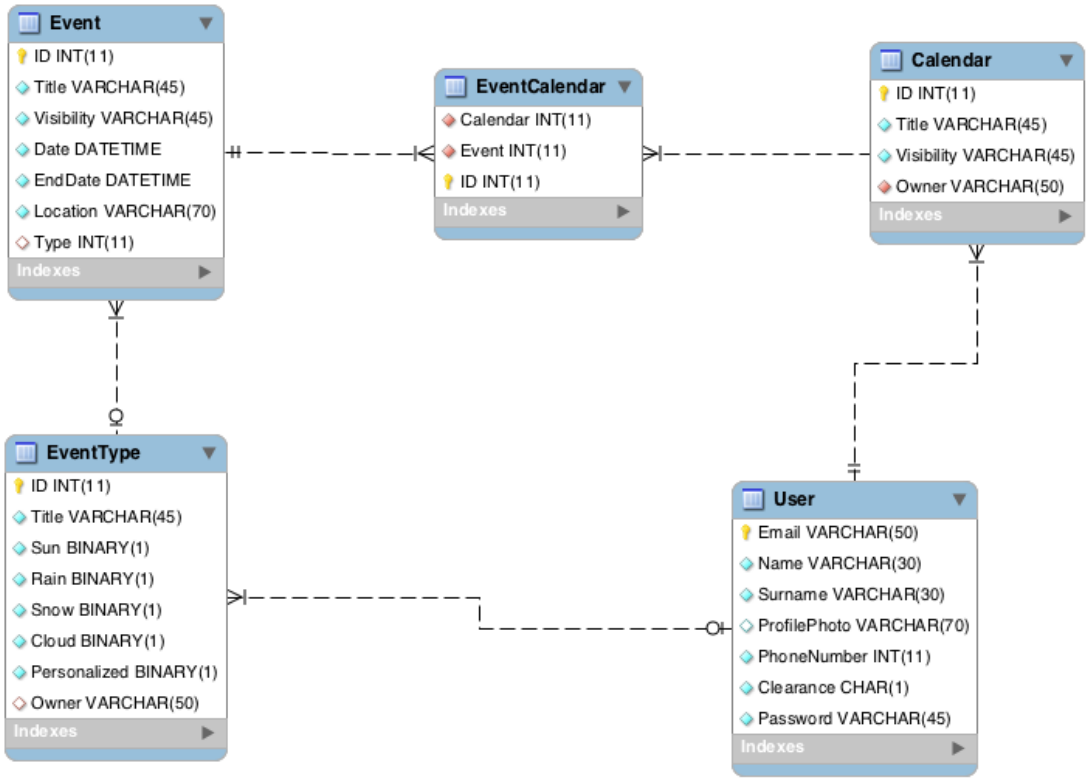
3.2.1 TRANSFORMATION FROM ER TO LOGIC

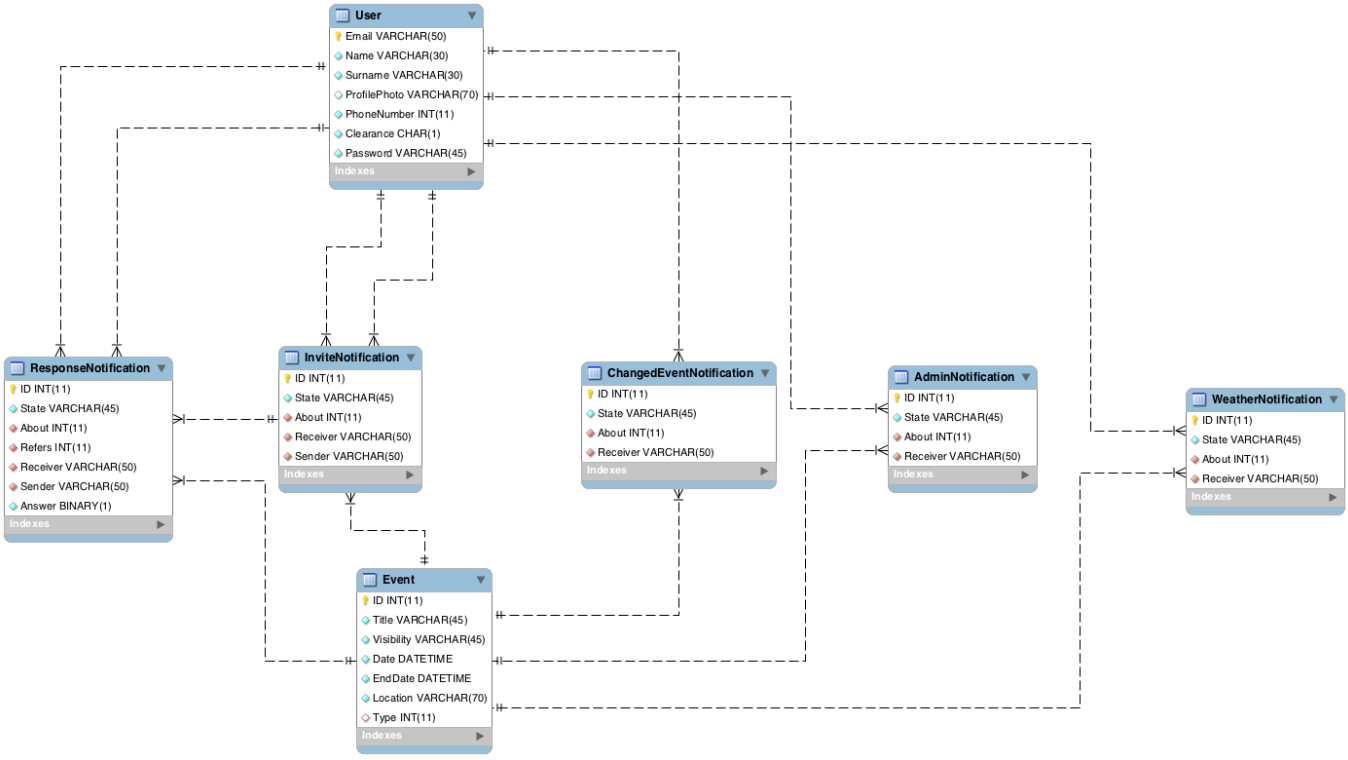
The Logical model can't be the same as the ER since the ER Model is an abstraction that sometimes can't be implemented, so these are the changes we made to make it realizable:

- The Default Event Type and the Personalized Event Type have been merged in the Event Type Table, that now contains a Boolean attribute that states if the Event Type is personalized and a foreign key that refers to the user that created it in the case it is personalized. A trigger will prevent the Default Event Types from having an owner.
- The User and Administrator have been merged in a single User table, an attribute called clearance has been added to distinguish between them: it will be 'U' for the user and 'A' for the administrator.
- The Notification table doesn't exist in the logic design because it doesn't contain any real information. It could be inserted later as a View only for a convenience purpose.
- The Event-Calendar relationship is a "many to many" relationship, so in the implementation it becomes a table that we called EventCalendar.
- The Event Type table has now four Boolean attributes to describe the accepted weather conditions, if an event can be held in a condition this flag will be True in the database. We could anyway change the list of conditions later when we will choose the weather forecast API we're going to use, since it could be more specific.

3.2.2 Logic Model

The Model is again divided in two images to help the readability.



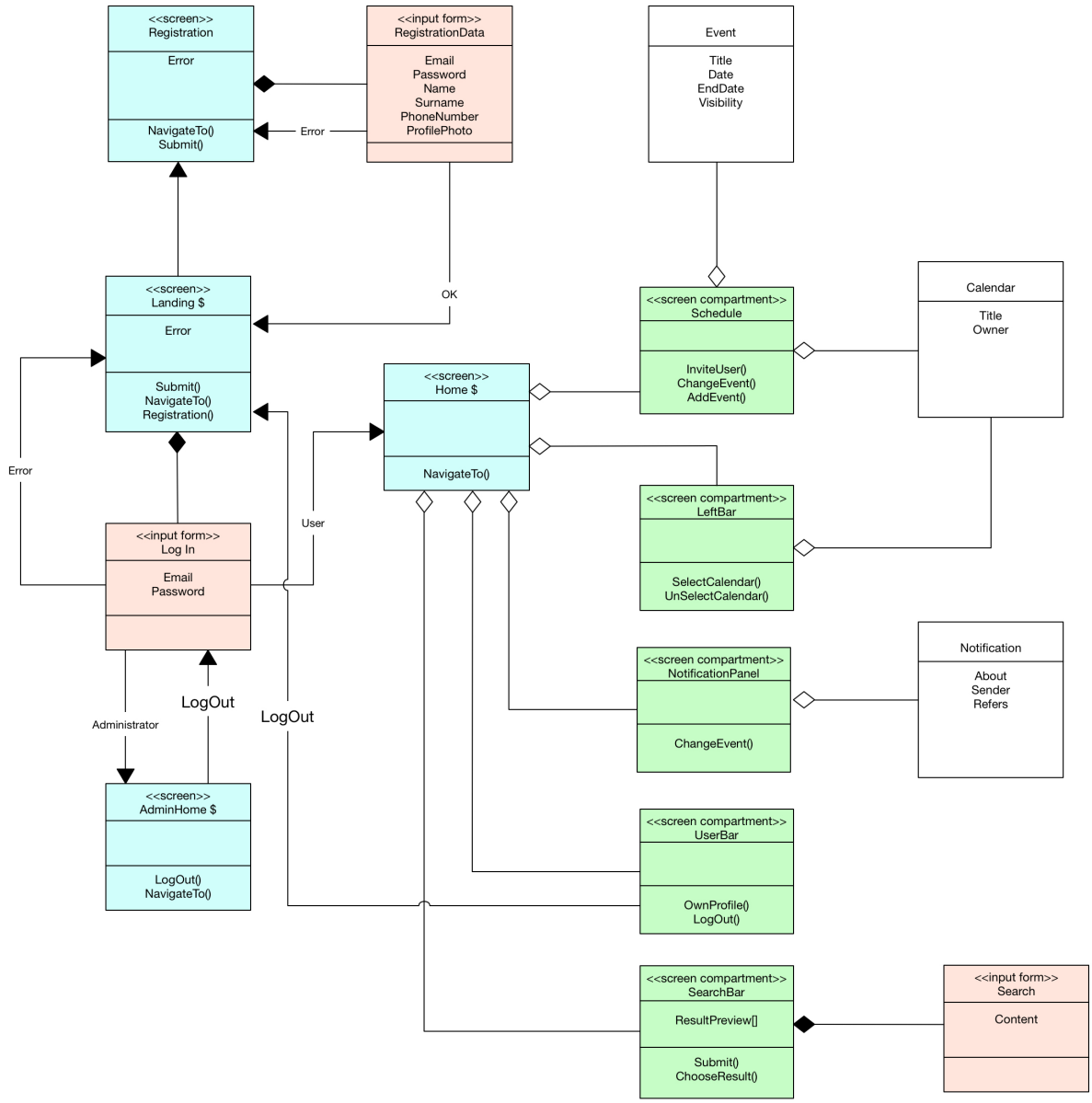


- Calendar(ID,Title, Visibility, *Owner*)
- Event (ID, Title, Visibility, Date, EndDate, Location, *Type*)
- EventCalendar(ID, *Calendar*, *Event*)
- EventType(ID, Title, Sun, Rain, Snow, Cloud, Personalized, *Owner*)
- AdminNotification(ID, State, *About*, *Receiver*)
- ChangedEventNotification(ID, State, *About*, *Receiver*)
- InviteNotification(ID, State, *About*, *Receiver*, *Sender*)
- ResponseNotification(ID, State, Answer, *About*, *Refers*, *Receiver*, *Sender*)
- WeatherNotification(ID, State, *About*, *Receiver*)

4. USER EXPERIENCE

In this chapter we will describe the functionality of our project with a UX diagram, again it will not be formally correct and complete because we decided to split it in three parts in order to help the user to understand it.

4.1 LANDING UX DIAGRAM



As you can see there are three pages marked with the $ symbol, this symbol means that the page is reachable from every page in the site, this is not exactly correct in this case, in fact these three pages are relative to the three actors in our system and the “always reachable”

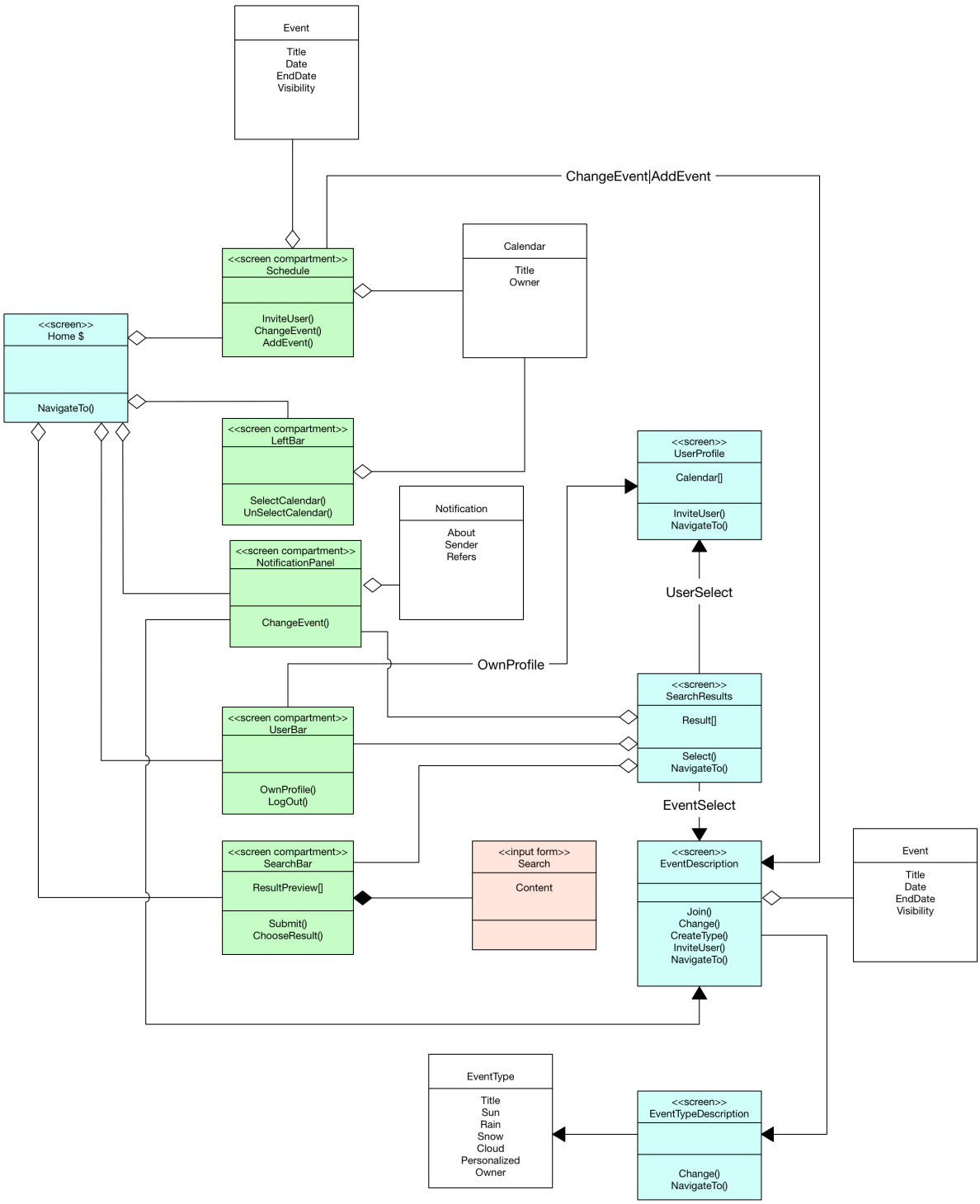
relationship is valid only for one page for every actor.

A guest will see the “Landing” page immediately when he visit the site, from this page he can click on a link to navigate to a page where he will be able to register a new user in the site. After the registration is completed without errors he will be able to log in using the input form in the landing page. The System will evaluate the credentials inserted and redirect the new actor to his proper homepage.

A User lands in the “Home” page; this page is composed of five components that allows the user to perform all the actions he is allowed to perform. Their functionalities will be explained in detail in the next UX diagram.

The Administrator lands in the “AdminHome” page; this page will be explained in detail in the Admin UX diagram later.

4.2 USER UX DIAGRAM



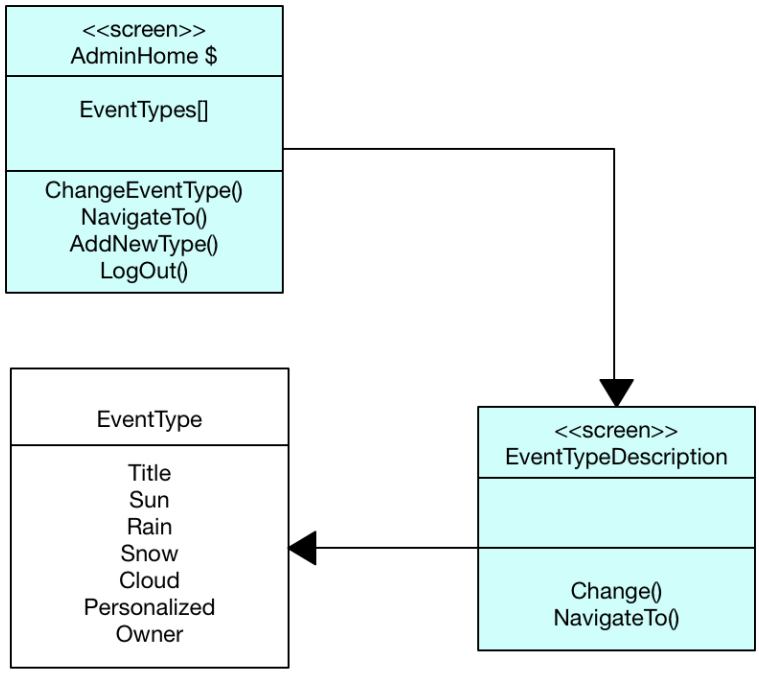
This is by far the biggest diagram of the three exposed, and it's not even complete: some arrows are intentionally not shown to avoid the formation of an unreadable web of lines, these missing arrows will be anyway explained in the following description.

The Home page is composed by five main components:

1. **The "Schedule" component:** This is the section of the page where the calendar and the events on it are really shown to the user, clicking on the box that represents one of this event the user will be able to change the information about it (if he's allowed to) or invite other users to it. Clicking on an empty time or date in this section the user will be allowed to create a new event, either providing the information about it in a little popup panel or by being redirected to the event information page.
2. **The "LeftBar" component:** In this section the user can select or deselect the calendars he wants to see, by doing so the schedule component will be updated and it will show only the events contained in the selected calendars.
3. **The "NotificationPanel" component:** This section is hidden by default, if new notifications are available for the user a little icon will show up to alert him, then, by clicking on it, the notification panel will appear, showing the user the notifications that has just arrived for him. By clicking on a notification the user will be sent to the event information page related to the same event of the notification. This Relation is not shown in the UX Diagram for readability, there should be an arrow going from the notification panel to the EventInformation screen.
4. **The "UserBar" component:** The userbar component is really simple, it will be divided in two section: the first one is a link to the user profile page displayed as the name of the username with his profile photo beside, the second section is a LogOut button that will allow the user to log out from the current session.
5. **The "SearchBar" component:** The searchbar will be displayed as a text input box in the top bar of the page, the user can write the content he wants to search in the box and a little preview of the results will be shown, he can now either click on the search button, that will send him to the search results page, or select one of the results directly from the preview (The arrows of this behavior are also not shown in the Diagram).

Every page the user can visit should be linked to the components 3,4 and 5, because they are part of the default look for the user in every page, doing so would have generated an incredible amount of lines in the diagram, so we decided to link them only to the SearchResult page to show how it should be for every other page.

4.3 ADMINISTRATOR UX DIAGRAM



The Administrator diagram is quite simple since he's not allowed to access many functionalities. His home homepage consists in a list of all the default event types, an "Add new Default Type" button and a log out button. By selecting one of the event types he will be allowed to change it.

5 BCE DIAGRAMS

In this section there are three Boundary-Control-Entity diagrams. We decide to adopt this kind of design diagram because it is very similar to the design pattern Model View Controller. In fact we can easily think that boundaries correspond to the view, controls correspond to the controller and entities correspond to the model.

In order to realize these BCE diagrams we have, of course, exploited all the precedence considerations, but, in particular, BCE is strictly related to use case diagrams, which are in the RASD document, User experience diagrams, and ER diagrams. In fact, for each entity in the ER diagram there is a corresponding entity in the BCE. For all the pages, indicated with the label screen, in the UX diagram, there is, in the BCE, a corresponding method showX, where X is the name of the page, and each method, shown in the UX, is reported here, sometimes with a different name to avoid ambiguity.

We decide to make three BCE diagrams to make more them more readable. However the subdivision is not casual. In fact Each BCE diagram correspond to one of the three main sub-systems, that are:

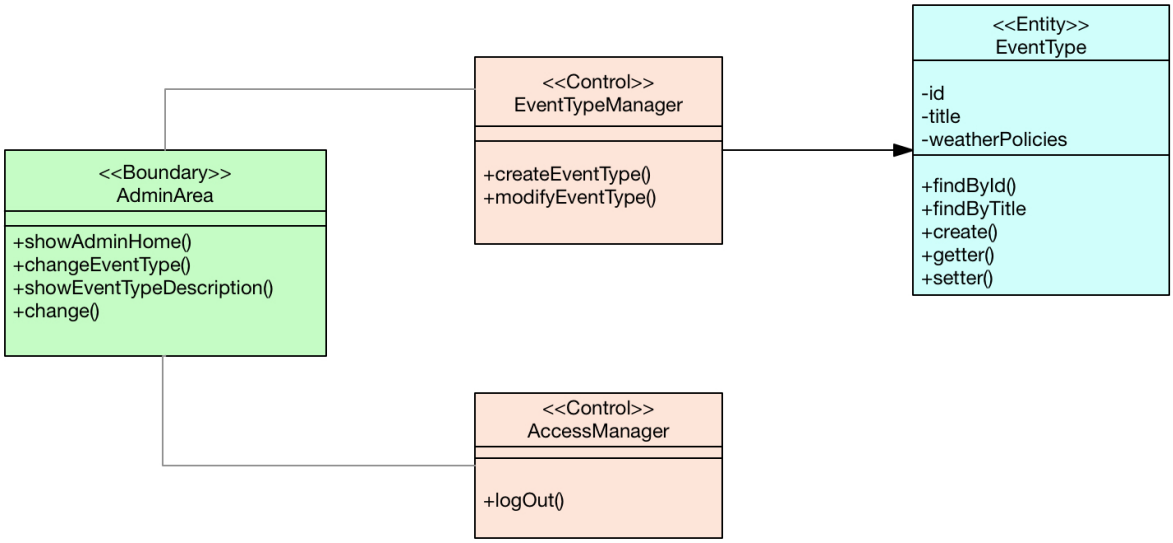
- Administrator
- Guest
- User

5.1 ADMINISTRATOR BCE

The boundary AdminArea represents the interface to the administrator and it contains all the basic functionalities that the administrator can exploit.

Since, for the time being, the administrator can only manage default event type there is a unique specific controller, the EventTypeManager that can interact with EventType entity. But there is also an AccessManager, which is used also by common user. Here we provide a brief description of the controller used:

- **EventTypeManager:** this controller allows the administrator managing default event type. The createEventType method allows the administrator creating new default event type, while the modifyEventType allows modifying an already existent event type, including the deletion.
- **AccessManager:** this controller allows the administrator to log out.



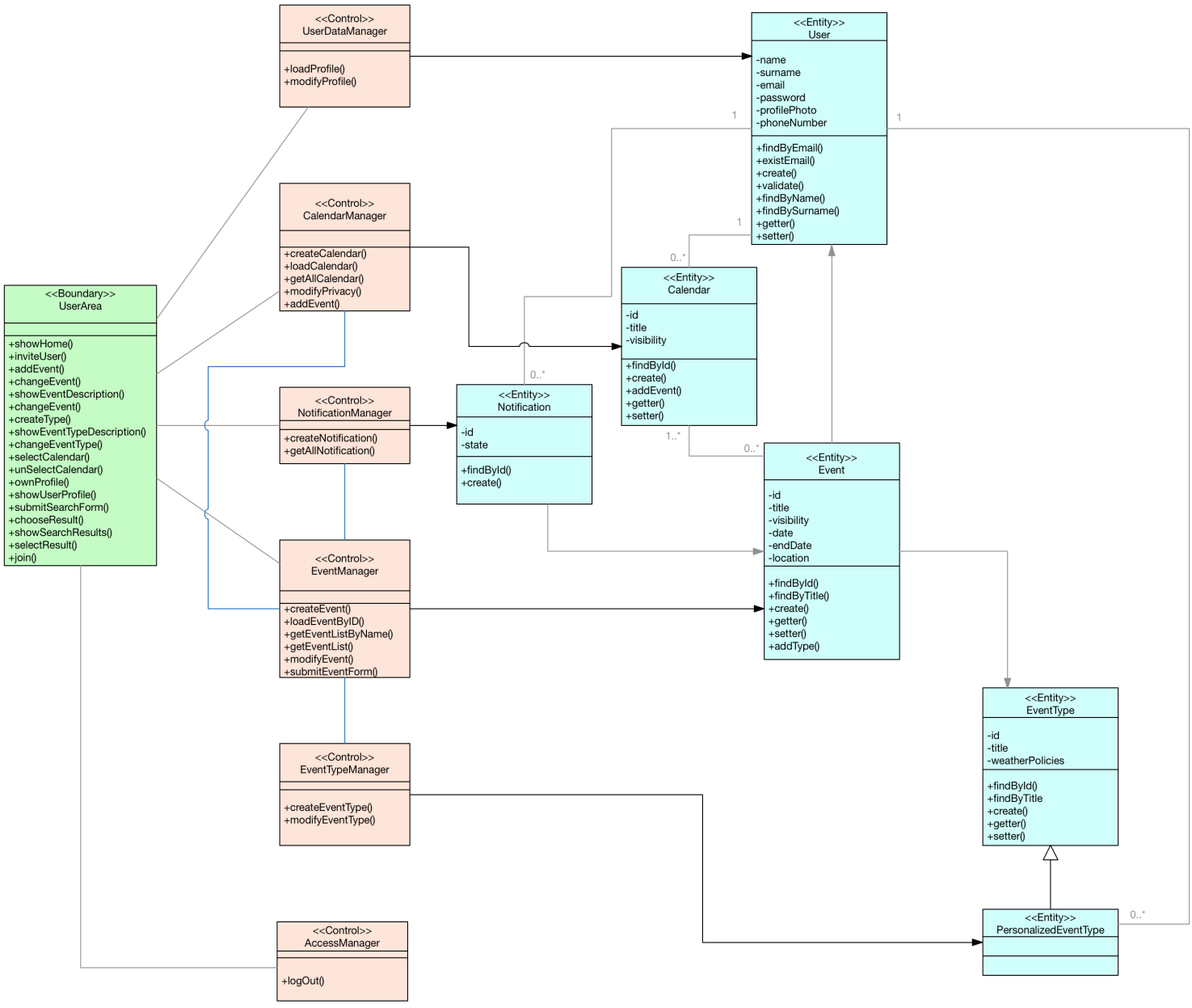
5.2 USER BCE

Since the User is the most important actor that can interact with the system and that can perform the larger number of activity, the BCE diagram of the user is the biggest one. There is a unique boundary, the UserArea, that is, the interface to the user. In this boundary there are all the main functionalities that the user can exploit. We decide to exclude something in order to make the diagram more readable.

As already said, the entities here correspond to the entities of the ER diagram. In the user BCE there are almost all the entities. We have decided to exclude all the different kind of notifications and other details because are not relevant at this level of the analysis.

There are several controllers:

- **UserDataManager:** this controller manages what concerns the user's profile, such as loading the user information, in order to show the user his profile or other user's profile, or modify some profile information
- **EventManager:** this controller manages all the operation on events: creation, modification and a lot of other activity, such as invite people, select the event date etc., that are summarized in the submitEventForm() method. There is an edge that connects the EventManager with the CalendarManager because when user creates an event, he must select a calendar where puts that event and the calendar manager is the controller in charge of adding events to calendars. Another edge connects the EventManager with the EvenTypeManager because, as said before, an user can create his personalized event type while creating an event, and, in order to do such thing, he needs the functionalities provided by this controller. There is a last edge that connects the EventManager with the NotificationManager. In fact when an user decides to invite some other users to an event, a notification is created automatically.
- **CalendarManager:** this is the controller in charge of managing calendar allowing the creation and the modification. It allows loading the calendar with all its details to show to the user.
- **EventTypeManager:** it provides the same functionalities of the admin one, but this can operate only with the PersonalizedEventType entity.
- **NotificationManager:** this controller allows loading all the notification that the user as received, interacting with the notification entity.
- **AccessManager:** it allows the user to log out. It is the same controller of the admin.

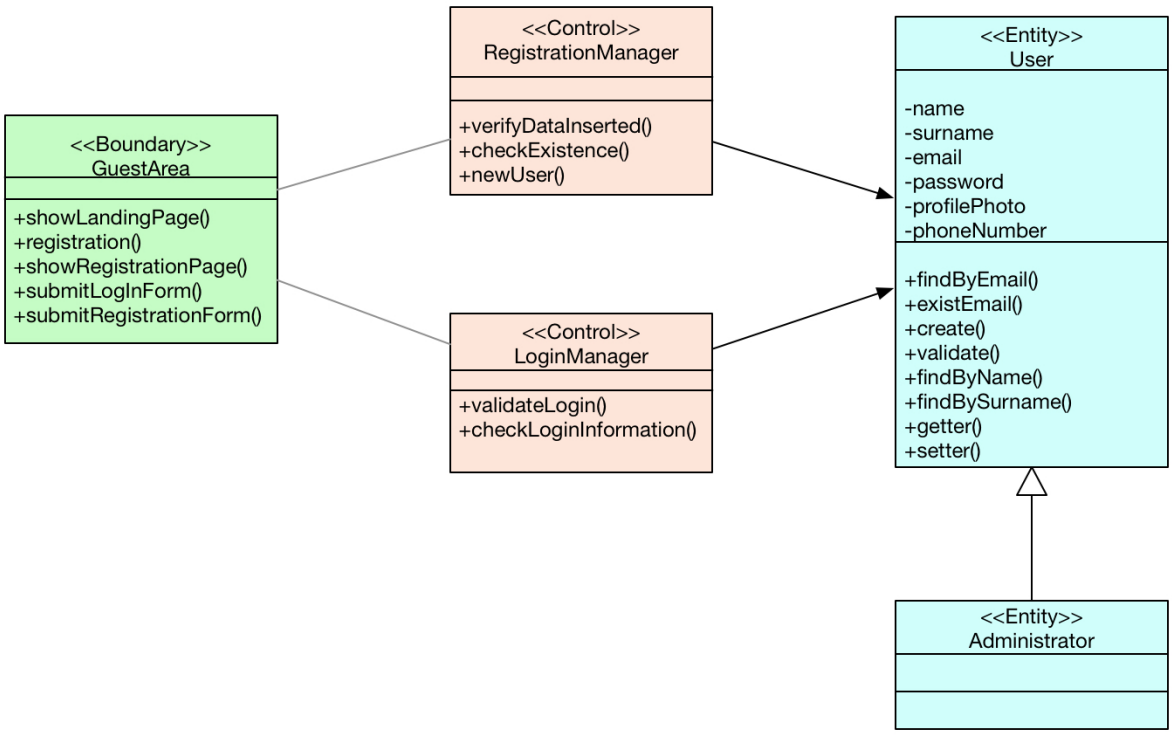

<<Control>>
UserDataManager
+loadProfile()
+modifyProfile()
<<Entity>>
User
-name
-surname
-email
-password
-profilePhoto
-phoneNumber
+findByEmail()
+existEmail()
+create()
+validate()
+findByName()
+findBySurname()
+getter()
+setter()
1
1
<<Control>>
CalendarManager
+createCalendar()
+loadCalendar()
+getAllCalendar()
+modifyPrivacy()
+addEvent()
1
0..*
<<Boundary>>
UserArea
+showHome()
+inviteUser()
+addEvent()
+changeEvent()
+showEventDescription()
+changeEvent()
+createType()
+showEventTypeDescription()
+changeEventType()
+selectCalendar()
+unSelectCalendar()
+ownProfile()
+showUserProfile()
+submitSearchForm()
+chooseResult()
+showSearchResults()
+selectResult()
+join()
<<Entity>>
Calendar
-id
-title
-visibility
+findById()
+create()
+addEvent()
+getter()
+setter()
0..*
<<Control>>
NotificationManager
+createNotification()
+getAllNotification()
<<Entity>>
Notification
-id
-state
+findById()
+create()
1..*
0..*
<<Entity>>
Event
-id
-title
-visibility
-date
-endDate
-location
+findById()
+findByTitle()
+create()
+getter()
+setter()
+addType()
<<Control>>
EventManager
+createEvent()
+loadEventByID()
+getEventListByName()
+getEventList()
+modifyEvent()
+submitEventForm()
<<Entity>>
EventType
-id
-title
-weatherPolicies
+findById()
+findByTitle
+create()
+getter()
+setter()
<<Control>>
EventTypeManager
+createEventType()
+modifyEventType()
<<Entity>>
PersonalizedEventType
0..*
<<Control>>
AccessManager
+logOut()

5.3 GUEST BCE

The last BCE is about the guest sub-system. As remembered over and over again, a guest can only sign up or log in. So the boundary GuestArea offers the interface to do such a things.

There are two controllers one devoted to the registration to the system and one to the login. Here a more precise description of the two controllers:

- **RegistrationManager:** this is the controller in charge of registering a guest to the system. It performs all the necessary activity requested to perform a correct registration: it checks if the data inserted by the guest are valid, it checks if the guest is already registered to the system and, if all the constraints are respected it created a new user.
- **LoginManager:** this controller allows a guest to access to the platform, becoming a user. It checks the correctness of the log in fields and it validates the log in with respect to the password inserted.



6 SEQUENCE DIAGRAMS

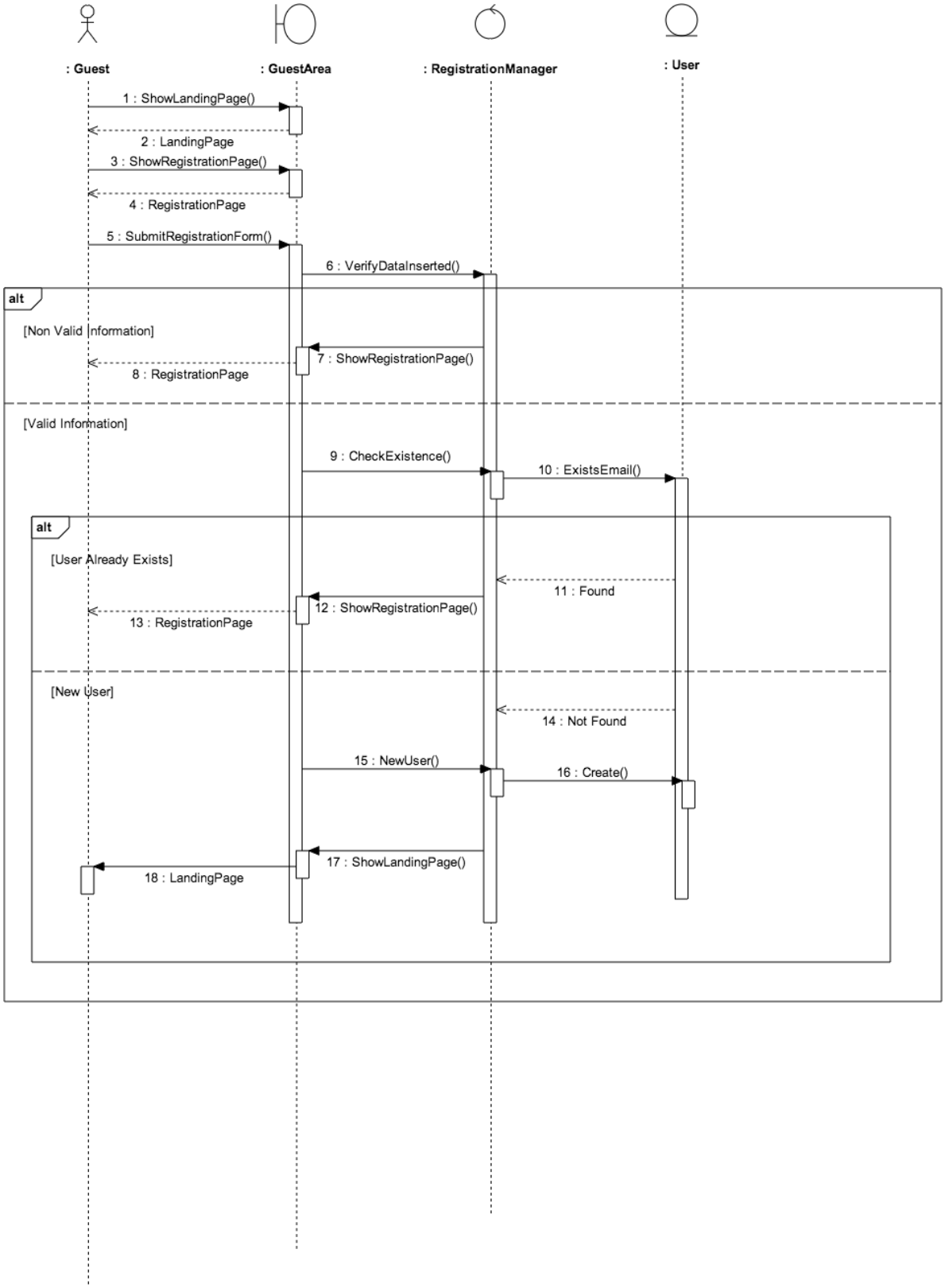
In this section we provide some sequence diagrams to let the reader better understand our BCE and UX diagrams.

Sometimes they are not formally correct, but this is because our purpose is to make easier read the diagrams and we don’t want to fill them with formal details that may make them very complex.

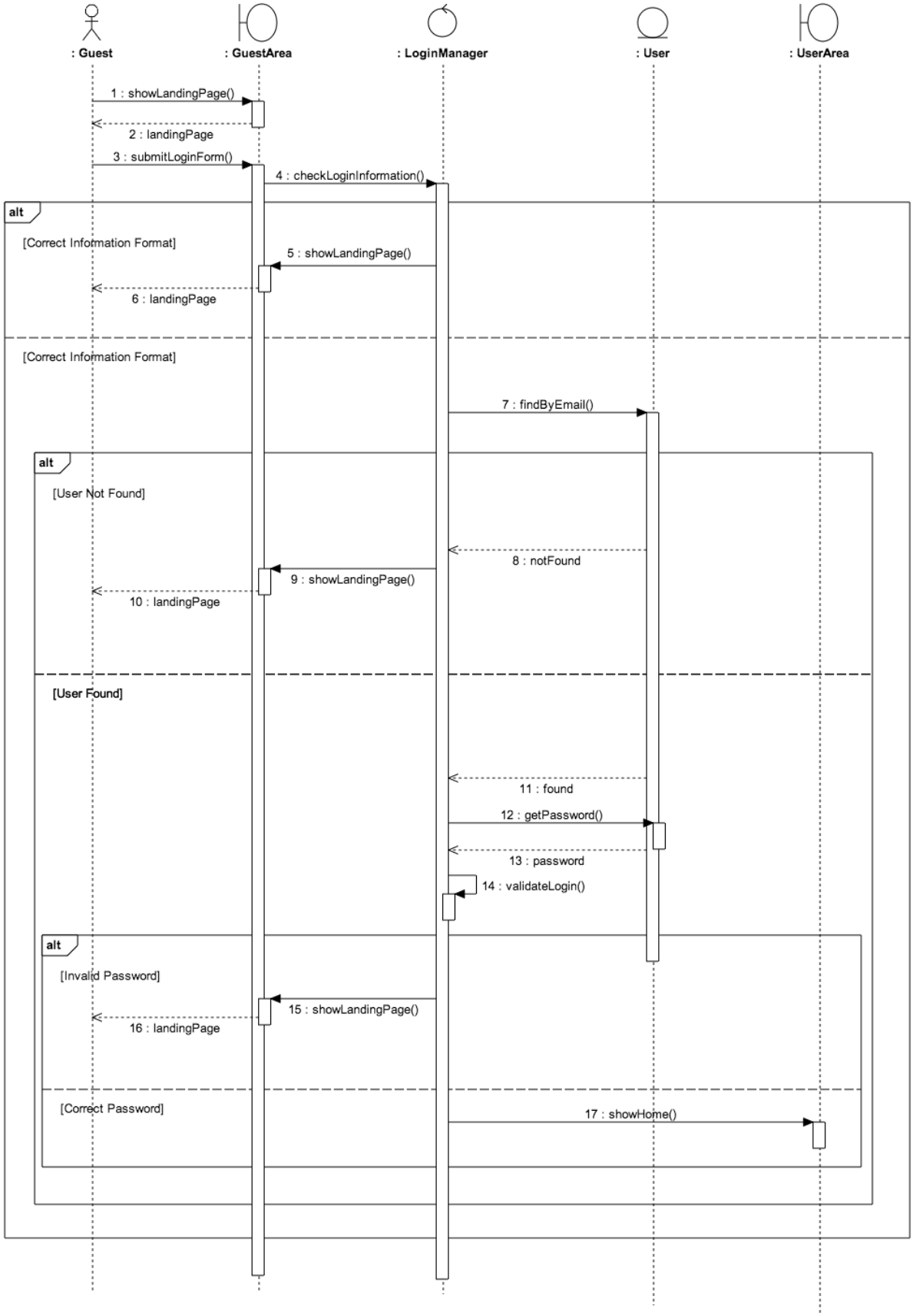
These are the diagrams that we sketched:

- Registration
- Log in
- View Profile
- Create Event
- Create Event Type
- Modify Event
- Postpone Event Because Of Notification
- Search And Add a Public Event

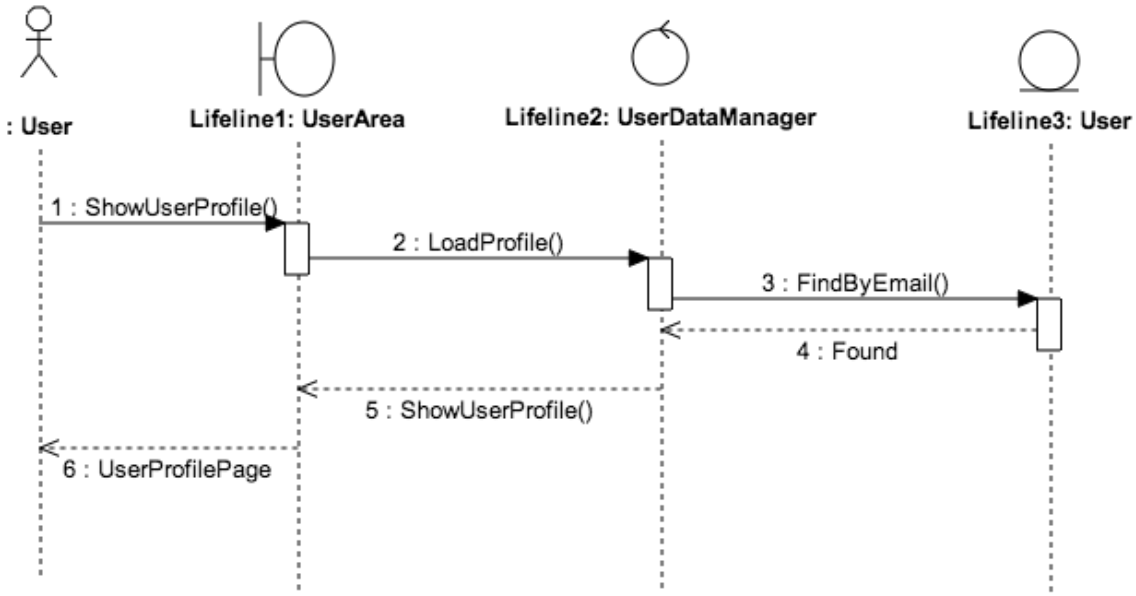
6.1 REGISTRATION



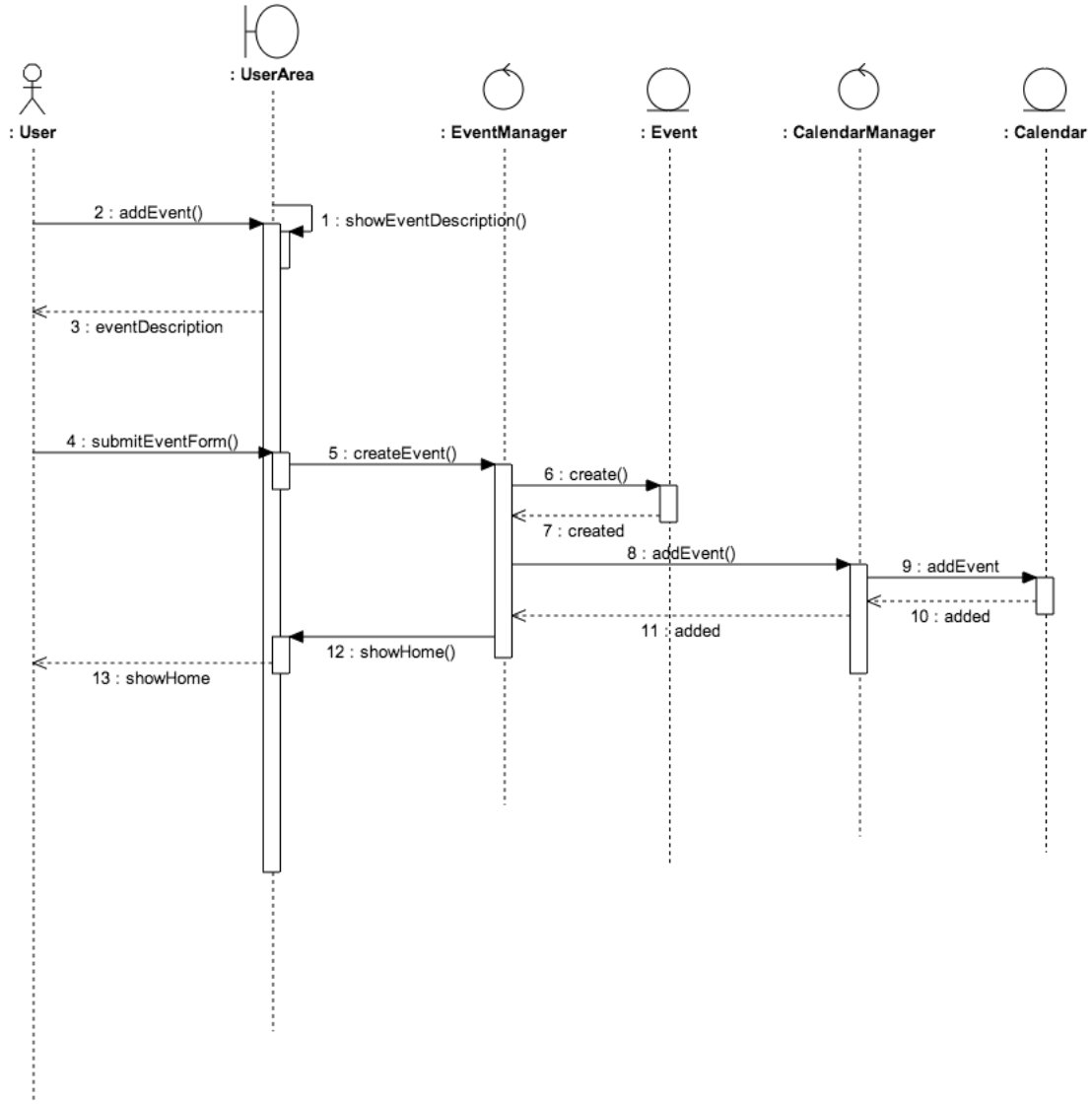
6.2 LOG IN


: Guest
: GuestArea
: LoginManager
: User
: UserArea
1 : showLandingPage()
2 : landingPage
3 : submitLoginForm()
4 : checkLoginInformation()
alt
[Correct Information Format]
5 : showLandingPage()
6 : landingPage
[Correct Information Format]
7 : findByEmail()
alt
[User Not Found]
8 : notFound
9 : showLandingPage()
10 : landingPage
[User Found]
11 : found
12 : getPassword()
13 : password
14 : validateLogin()
alt
[Invalid Password]
15 : showLandingPage()
16 : landingPage
[Correct Password]
17 : showHome()

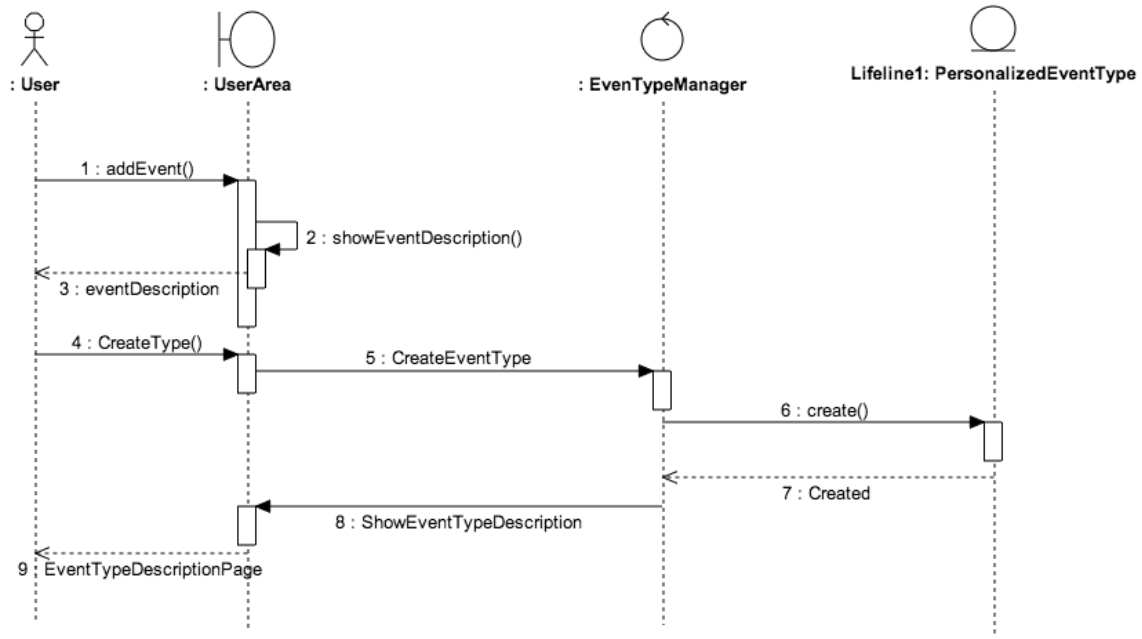
6.3 VIEW PROFILE



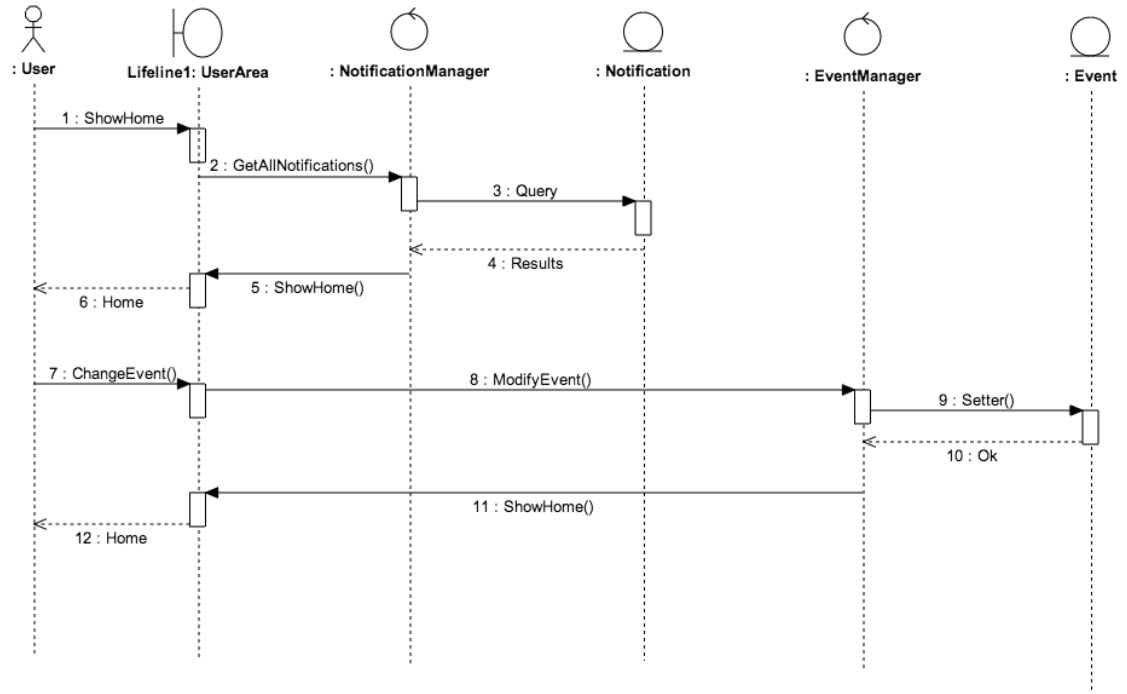
6.4 CREATE EVENT



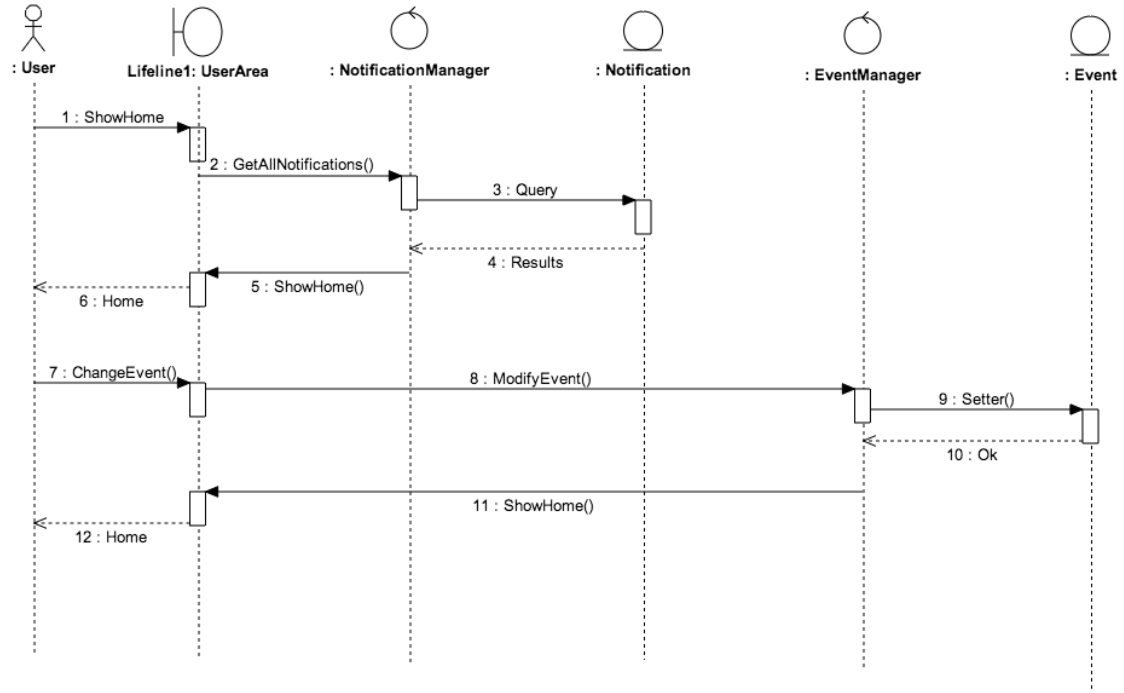
6.5 CREATE EVENT TYPE



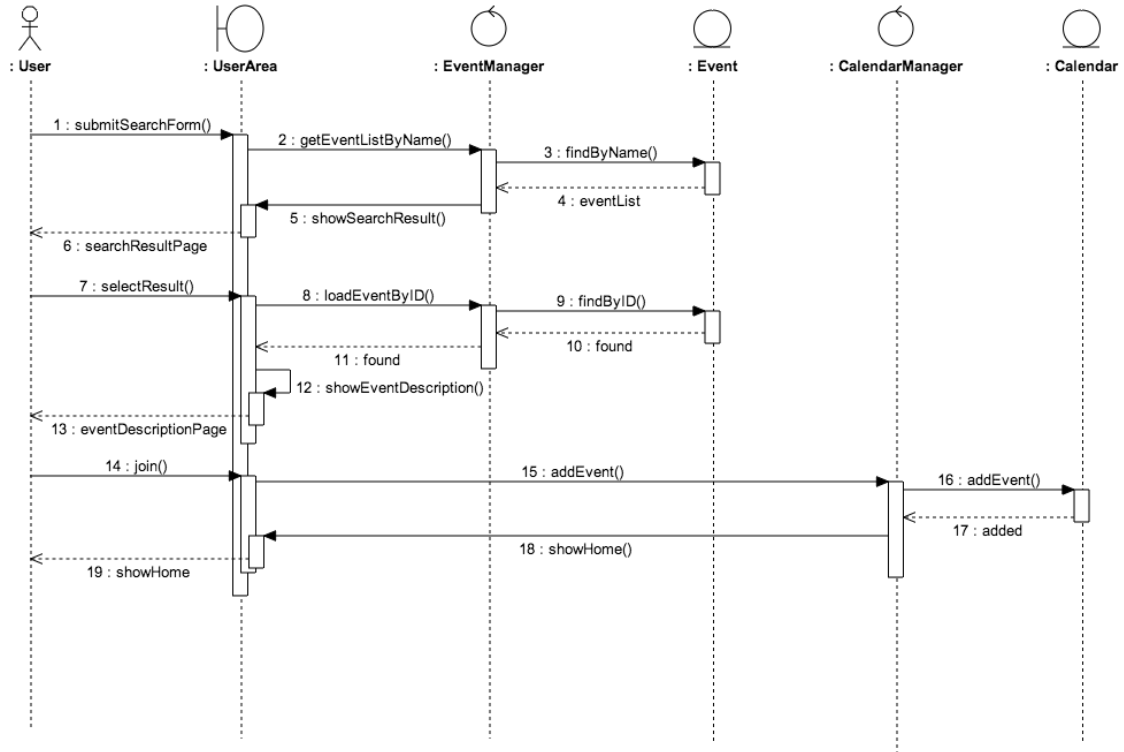
6.6 MODIFY EVENT



6.7 POSTPONE EVENT BECAUSE OF NOTIFICATION



6.8 SEARCH AND ADD A PUBLIC EVENT



7 USED TOOLS

This is the list of the tools that we have exploited to realize this document:

- Microsoft Office Word 2011: to redact this document.
- StarUML: to create sequence diagrams.
- OmniGraffle: to sketch ER diagrams, UX diagrams and BCE diagrams.
- MySQLWorkbench: to realize the logic model of the database.