MeteoCal

December 7th, 2014

Politecnico Di Milano

Computer Science

Design Document

***Authors*:**

- ***Alessandro Cianferotti***
- ***Alessandro Cimbelli***
- ***Alessandro De Angelis***

Sommario

1 RASD Revision

Foreseeing great potential in the use of Primefaces we decided to use some drop down menu to make the interface more intuitive. For this reason, we had to make some changes to the graphical interfaces described in the previous document (RASD).

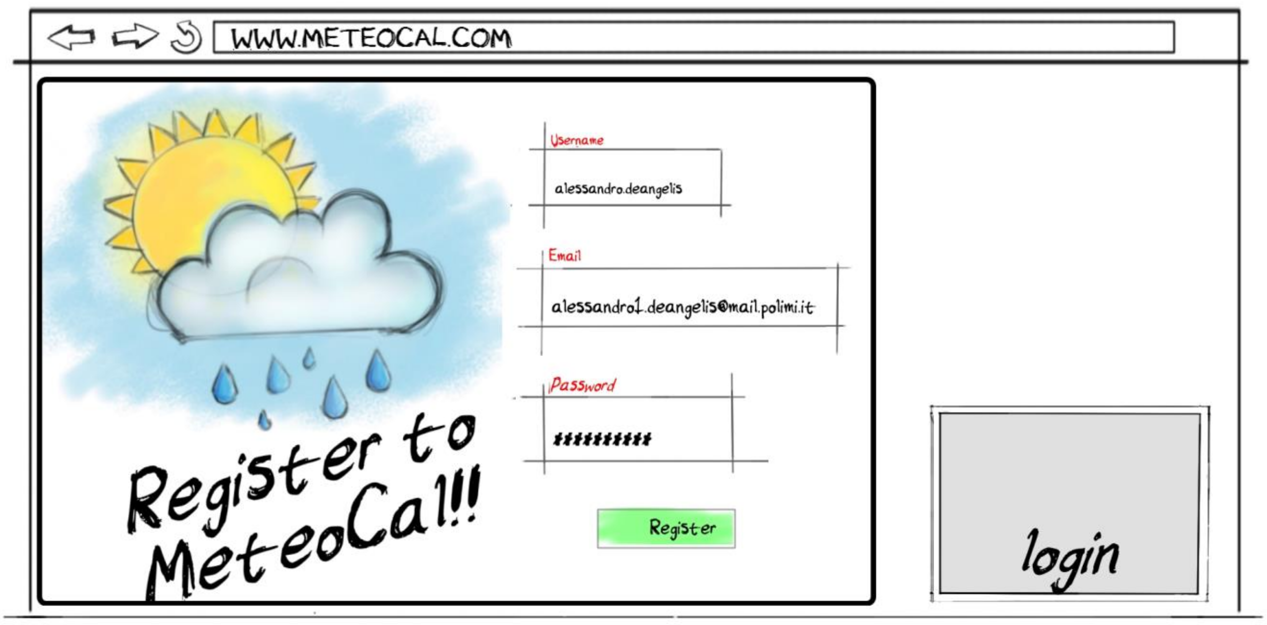


Figure 1 - Home page

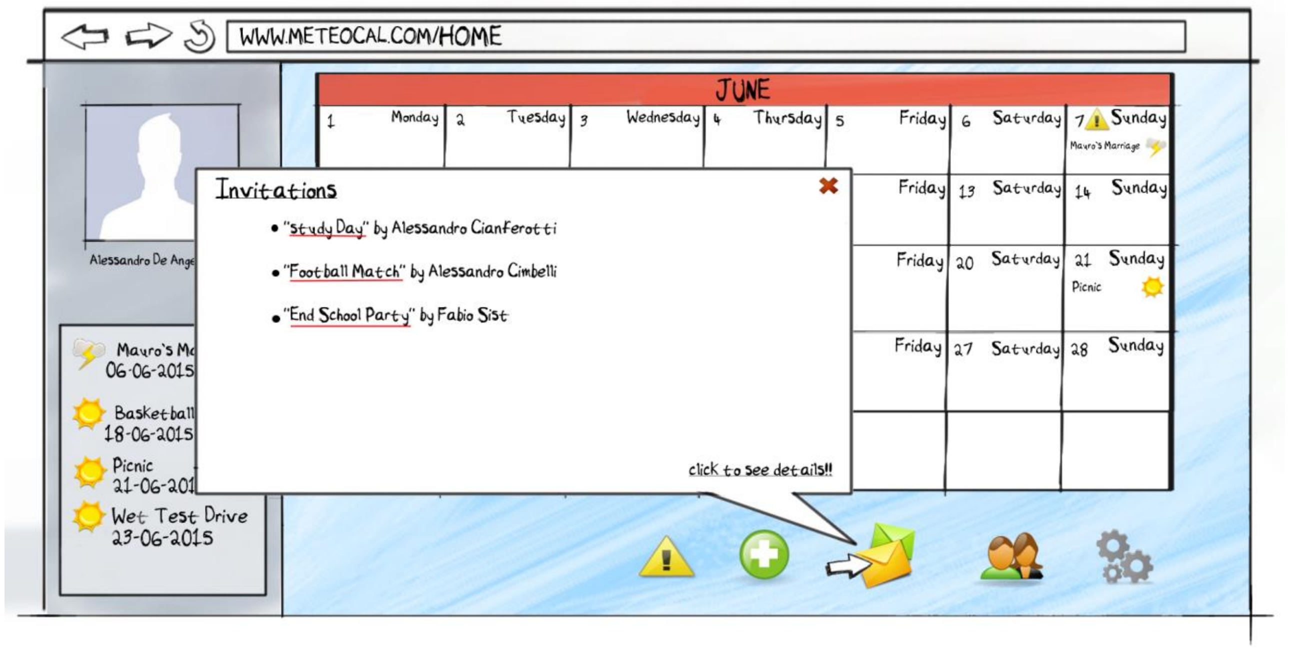
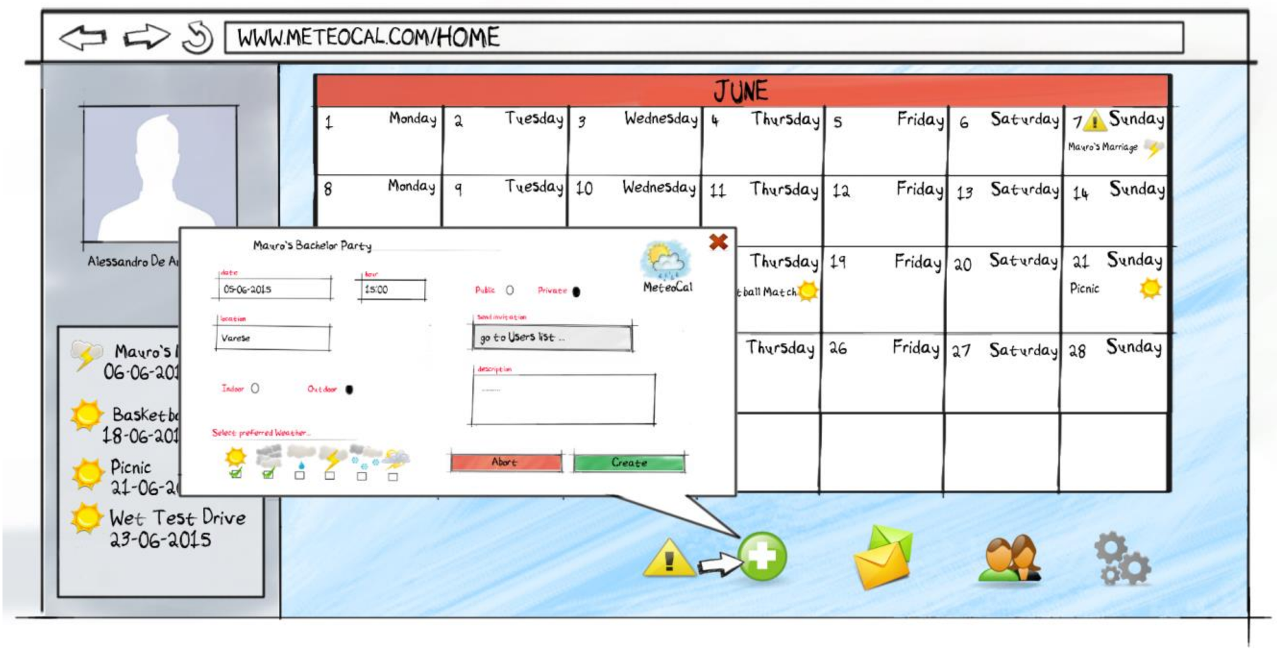


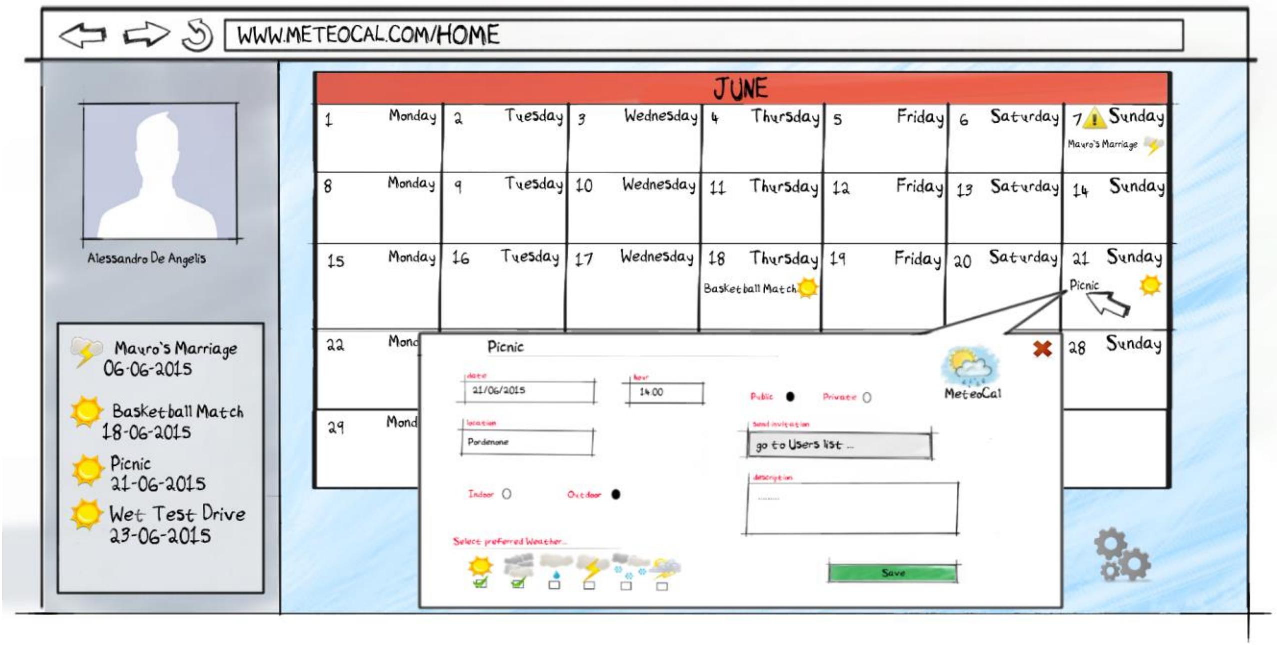
Figure 2 - Invitation



Mauro's Bachelor Party

date: 05-06-2015
hour: 15:00
Public ○ [✓ Private ●]
location: Varese
Send invitation: go to Users list ...
Indoor ○ [✓ Outdoor ●]
description: ....
Select preferred Weather...
✓ ✓ ☐ ☐ ☐ ☐
Abort
Create

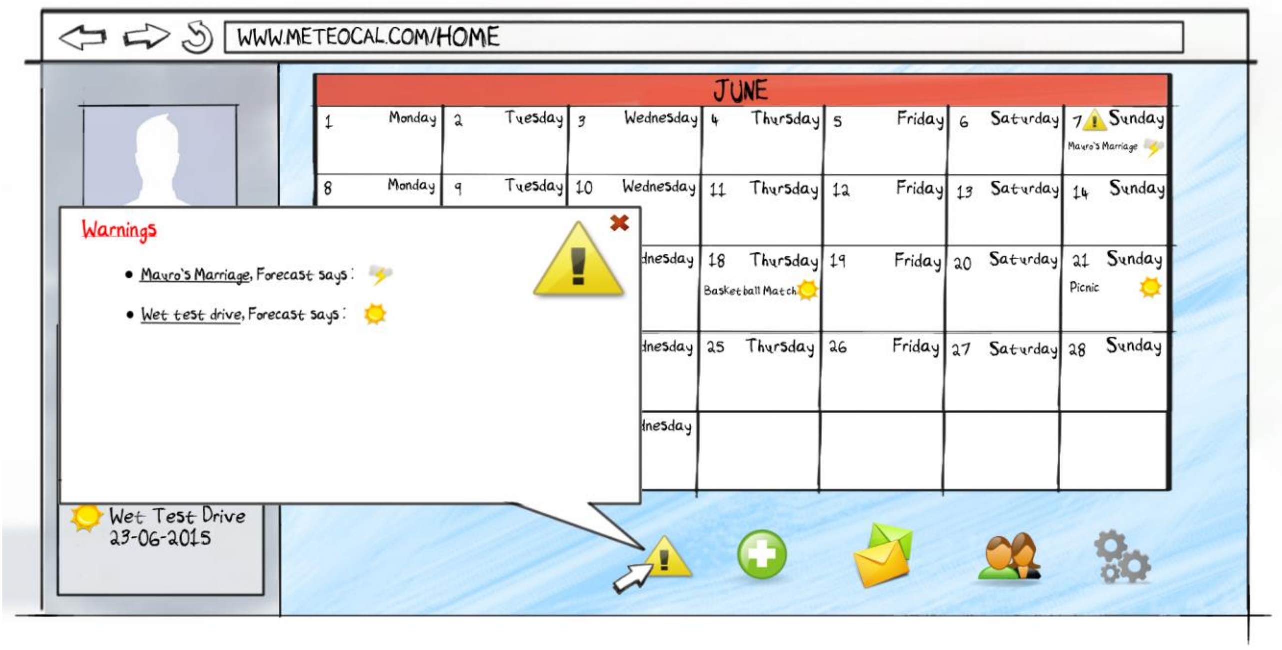
Figure 3- Event creation



Picnic

date: 21/06/2015
hour: 14.00
[✓ Public ●] Private ○
location: Pordenone
Send invitation: go to Users list ...
Indoor ○ [✓ Outdoor ●]
description: ....
Select preferred Weather...
✓ ✓ ☐ ☐ ☐ ☐
Save

Figure 4 - Event update


WWW.METEOCAL.COM/HOME

Warnings
- Mauro's Marriage, Forecast says :
- Wet test drive, Forecast says :

Wet Test Drive
23-06-2015

Figure 5 – Warnings

2 Architecture

2.1 Introduction

In this section we will expose the application's architecture and the technologies chosen for its development.

The Environment chosen is Oracle's Java Enterprise Edition 7 (JEE 7) that includes main characteristics of Java Standard Edition like Object-Orienting ,scalability and reliability with a multi-tier architecture and web services.

For the developing of (UI) we will use JSF (JavaServer Faces) and in particular Java Prime Faces.

JSF is a standardized specification that permit the creation of a customizable and rich user interface.

JAVA 7 API

2.2 Design Pattern

Oracles Designed for JEE a 4-Tier Architecture (From Oracles Documentation):

- Client tier

  It’s the component placed on the client Machine.

  Can be of two types:

  - Web Client (sometimes called a thin client)

    This type of client usually do not query databases, execute complex business rules, or connect to legacy applications with the aid of a web browser, which renders the pages received from the server.

  - Application Client that provides a way for users to handle tasks that require a richer user interface than can be provided by a mark-up language and communicates directly with the Business tier

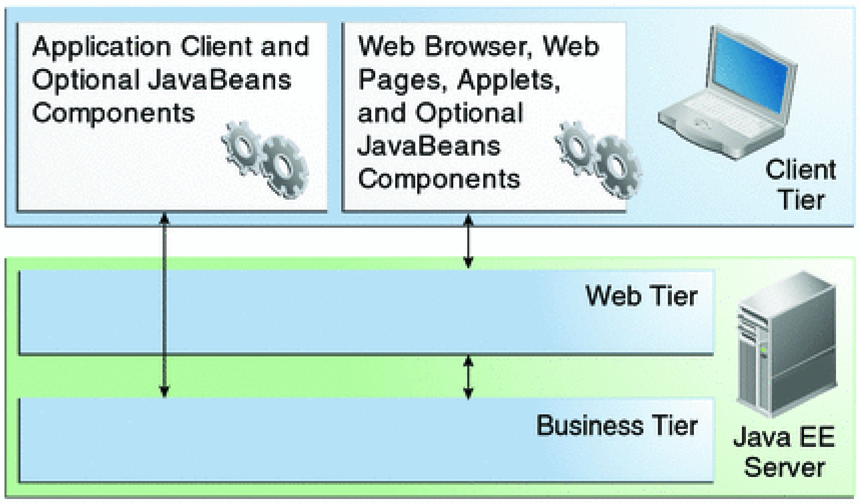

Application Client and Optional JavaBeans Components
Web Browser, Web Pages, Applets, and Optional JavaBeans Components
Client Tier
Web Tier
Business Tier
Java EE Server

Figure 6 - Client tier

- Web tier

  Java EE web components are either servlets or web pages created using JavaServer Faces technology and/or JSP technology (JSP pages). Servlets are Java programming language classes that dynamically process requests and construct responses. JSP pages are text-based documents that execute as servlets but allow a more natural approach to creating static content. JavaServer Faces technology builds on servlets and JSP technology and provides a user interface component framework for web applications.

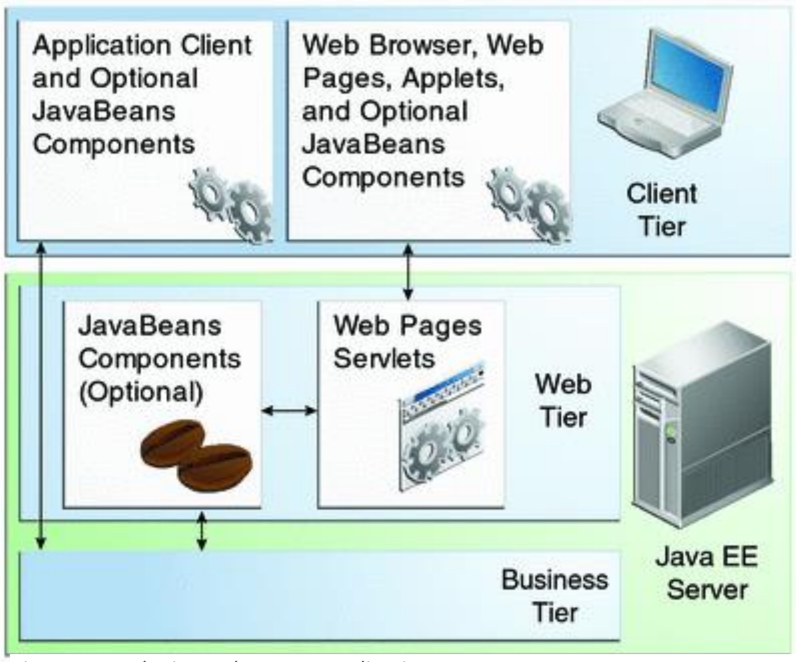
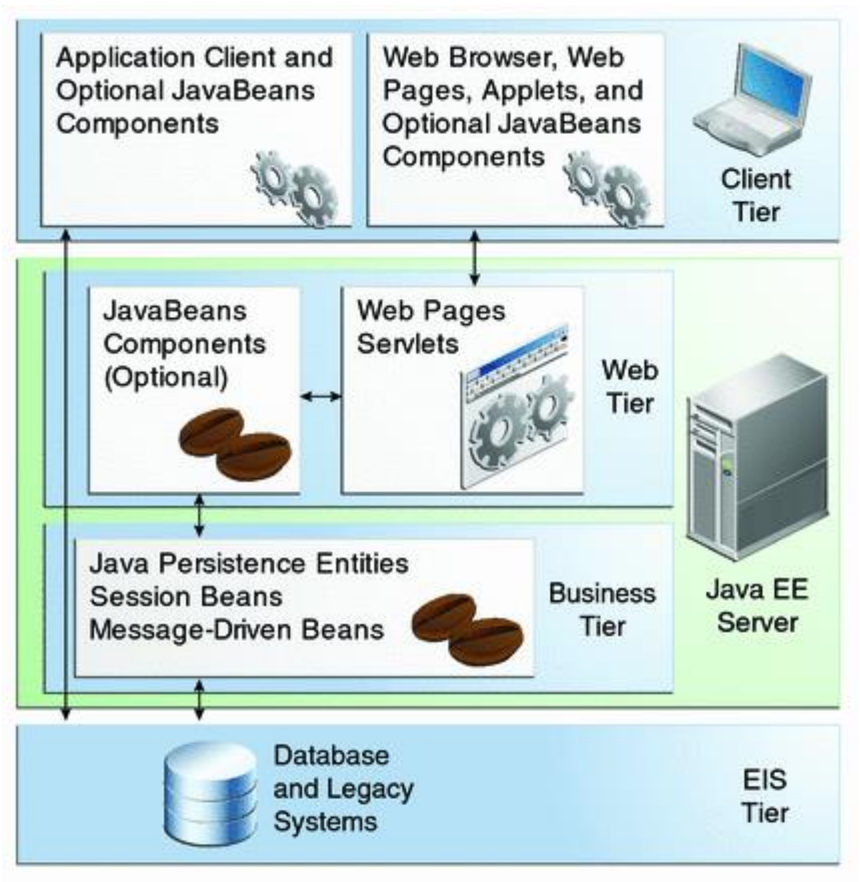

Application Client and Optional JavaBeans Components
Web Browser, Web Pages, Applets, and Optional JavaBeans Components
Client Tier
JavaBeans Components (Optional)
Web Pages Servlets
Web Tier
Java EE Server
Business Tier

Figure 7 - Web Tier and Java EE Applications

- Business tier

  Business code, which is logic that solves or meets the needs of a particular business domain such as banking, retail, or finance, is handled by enterprise beans running in either the business tier or the web tier.

- Enterprise Information System Tier
  The enterprise information system tier handles EIS software and includes enterprise infrastructure systems, such as enterprise resource planning (ERP), mainframe transaction processing, database systems, and other legacy information systems. For example, Java EE application components might need access to enterprise information systems for database connectivity.


Application Client and Optional JavaBeans Components
Web Browser, Web Pages, Applets, and Optional JavaBeans Components
Client Tier
JavaBeans Components (Optional)
Web Pages Servlets
Web Tier
Java Persistence Entities Session Beans Message-Driven Beans
Business Tier
Java EE Server
Database and Legacy Systems
EIS Tier
Figure 8 - Overview

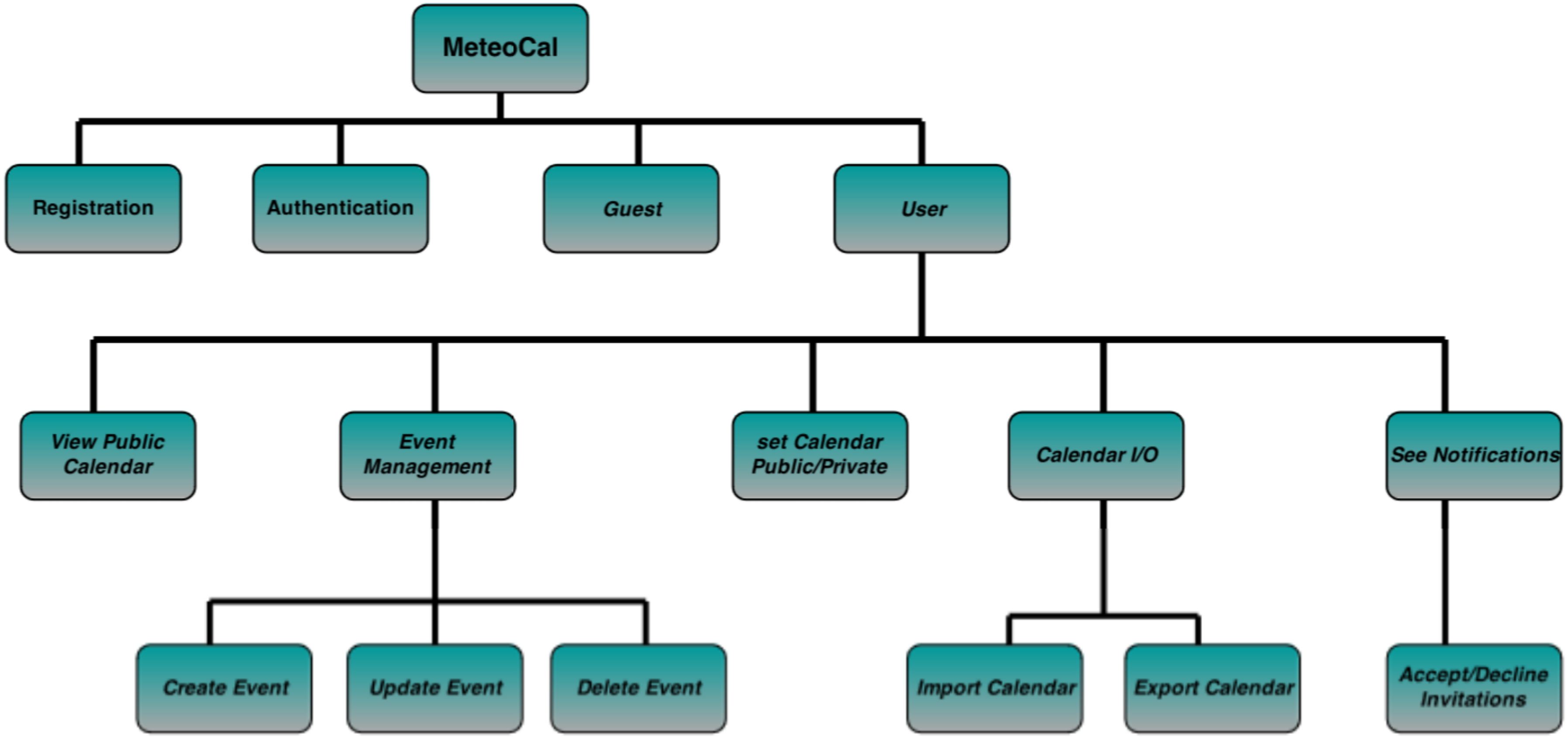
2.3 Used Technologies

Web Tier	Java Server Faces
Business Logic Tier	JEE 7 With GlassFish Open Source Edition 4.1
Persistence Tier / Database	MySQL

2.4 Sub Systems

We separate our systems into these sub-systems:

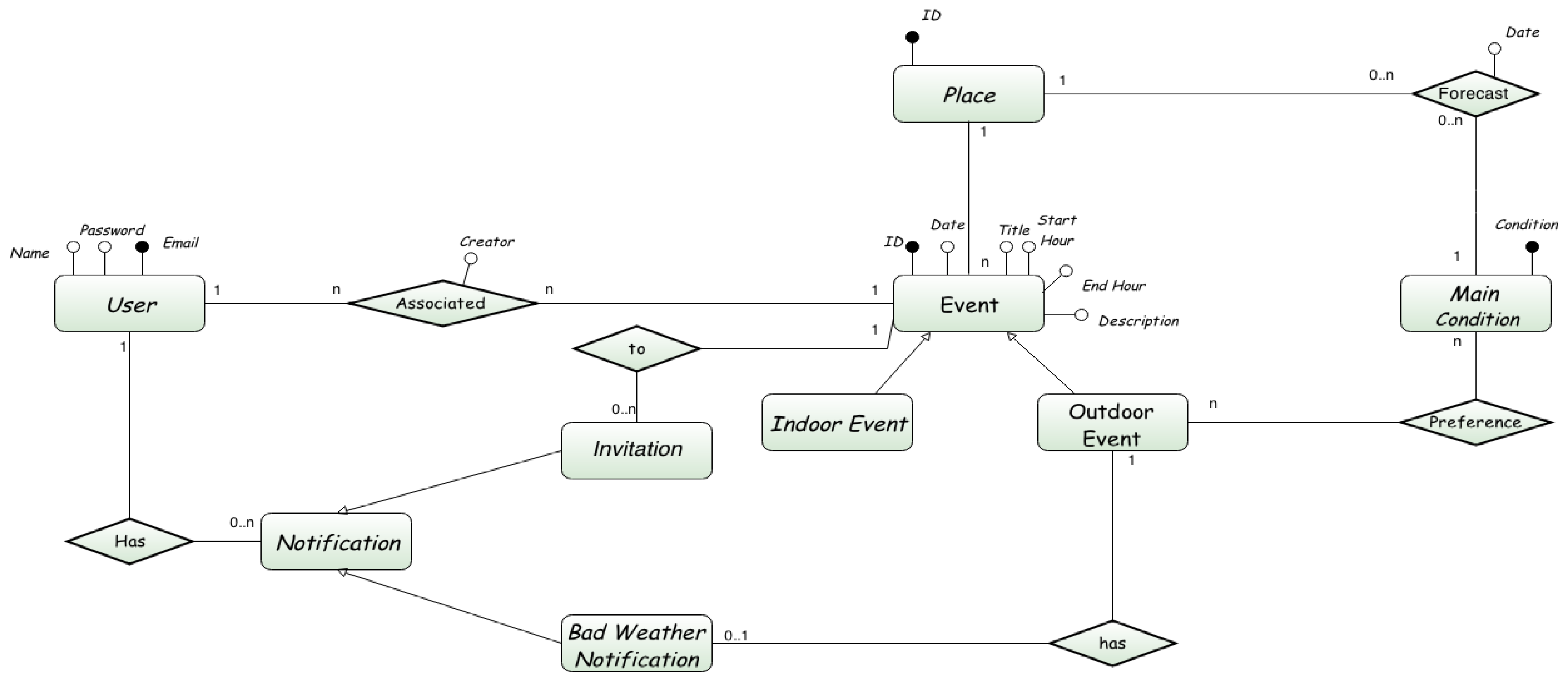
- Registration Sub-System
- Authentication Sub-System
- Guest Sub-System
- User Sub-System
  - Event Management Sub-System
  - Calendar View Sub-System
  - Calendar Management Sub-System
  - Notification Sub-System


MeteoCal
Registration
Authentication
Guest
User
View Public Calendar
Event Management
set Calendar Public/Private
Calendar I/O
See Notifications
Create Event
Update Event
Delete Event
Import Calendar
Export Calendar
Accept/Decline Invitations

3 *Persistent Data Management*

The data is stored in a relational database. In this Section will be provided an Entity Relational model (ER) and a logical Model with detailed motivations for entities and chosen relations.

3.1 Conceptual Design



3.1.1 Choices and Motivations

- Entities

  - The only actor present in the ER is the User. For the others, Guest and Forecast Provider, no data must be stored for services provided by the system. We decided to use Email as primary key to make easy login control.
  - The main entity is Event, characterized by a unique ID and some important information like date, title, starting and ending hour and description made by the User creator. We have decided to create two types of Events, Indoor and Outdoor. For the moment, Indoor Event has no difference with the generic Event, but this generalization will make possible further expansions of the System .Outdoor Event instead permit the creator User to choose preferences about the Weather.
  - The Event is associated with a Place, characterized for the moment only by an ID.
  - The system stores data sent by the forecast provider in DB, avoiding in this way problems with continuous connections.
  - The Entity Main Condition represent the homonym Enum Class in Class Diagram (Requirements Analysis and Specification Document, chapter 7.4).
  - The Entity Notification is of two types: Invitation of other Users for Events and Bad Weather Notification for Outdoor Events.

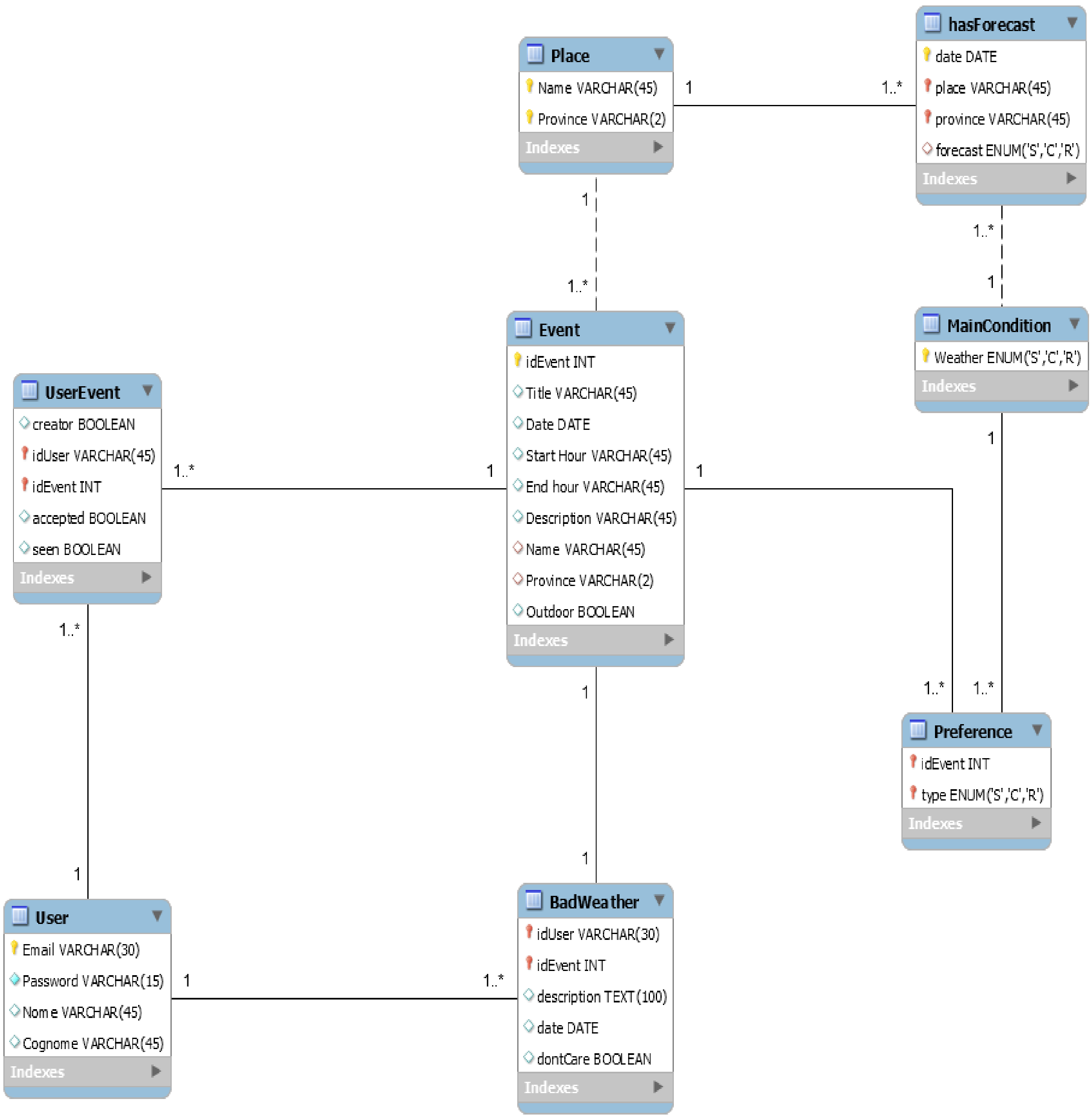
- Relations

  - User-Events: the n-m relation requires a bridge entity that represents the Calendar class in Class Diagram. This entity contains a Boolean variable for knowing if the user is the creator of the Event.
  - Event-Place: Event Entity does not have a direct link with Forecast, in this way we reduced considerably the number of relations.
  - Place-Main Condition: the Bridge entity forecast, containing the date of the prevision, represents the Forecast class in the Class Diagram.

- Outdoor Event-Preference: With this relation the system make the Users to select weather Conditions for their Outdoor Event.
- Outdoor Event-Bad Weather Notification: this Relation is created by the system to alert Users about the difference between preferences and forecast.
- Notification-User, Invitation-Events: Notifications (Bad Weather Notification and Invitations) must have a link with the user to notify. In addition the invitation must be linked to the event to which is related.

3.2 Logical Model

In this section we want to represent the logical model of the database. We have obtained this schema translating the ER model.



First of all we have solved the m-n association:

- UserEvent: between User and Event. It contains the primary key of these two entities. It has also three Boolean fields to describe if the user is the creator or it has been invited and, in case of invitation, if the user has seen the notification.
- hasForecast: between Place and MainCondition. Of course it contains the primary key of Place and MainCondition and in addition it has a Date field to distinguish the different forecast for the same Place.

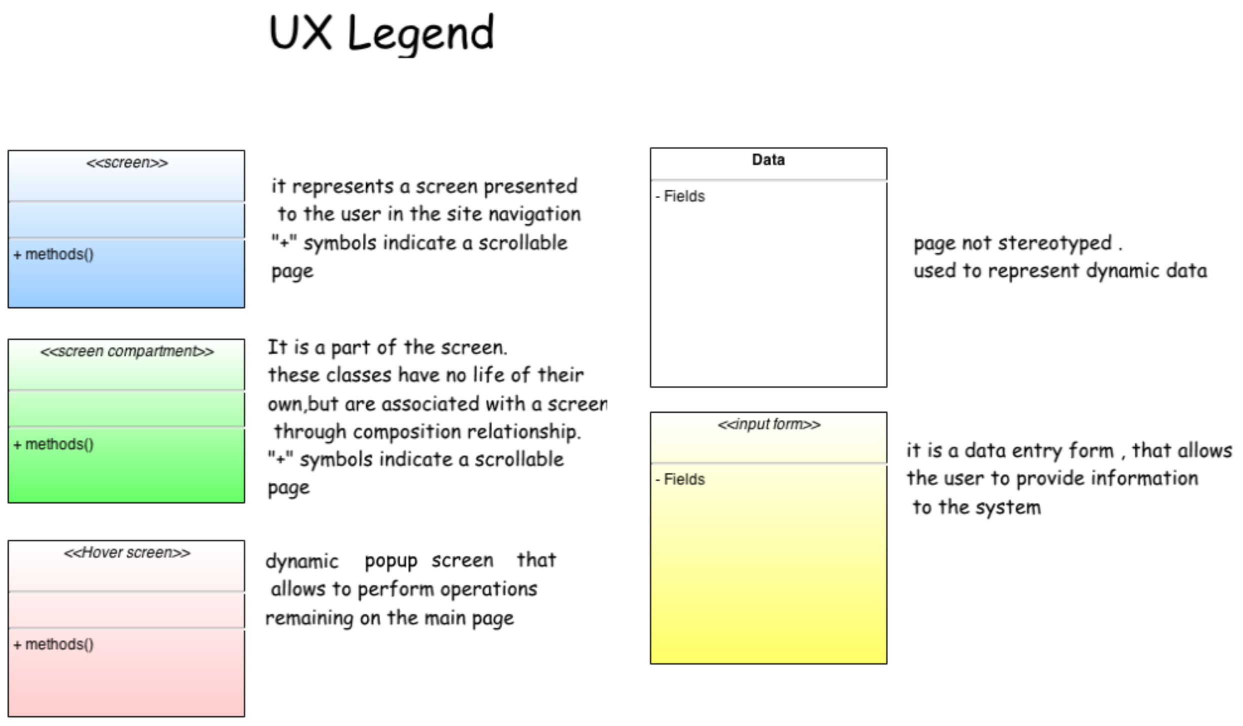
After this step we have solved the hierarchy problems: one for the event and one for the notification.

As regards for the event, we have taken this way: Indoor and Outdoor events have almost the same fields (only preferences different the event), so we have solved the hierarchy collapsing entity Outdoor and entity Indoor in one single entity named Event. Event contains all the properties of Indoor and Outdoor and a "flag" field that describes the event type.

As regards for the notification, we have solved the problem in a different way. We have deleted the entity Invitation and, taking advantage of the entity UserEvent, we have added to its few fields to describe why that event appears in a user's calendar. Instead, we still have the entity Bad Weather.

4 UX Diagrams

In this paragraph we want to describe the User Experience (UX) given by our system to its users. We decided also to split the UX Diagram in functionalities as for Use Cases in the RASD Document in order to better understand the whole Diagram.


UX Legend
<<screen>>
+ methods()
it represents a screen presented to the user in the site navigation "+" symbols indicate a scrollable page
<<screen compartment>>
+ methods()
It is a part of the screen. these classes have no life of their own,but are associated with a screen through composition relationship. "+" symbols indicate a scrollable page
<<Hover screen>>
+ methods()
dynamic popup screen that allows to perform operations remaining on the main page
Data
- Fields
page not stereotyped . used to represent dynamic data
<<input form>>
- Fields
it is a data entry form , that allows the user to provide information to the system

4.1 Overview

The following Diagram represents the different home pages according to the kind of user that can interact with the system .They represent the root of our website, in fact starting from them the user's experience can enrich all the possible screens of the website. The guest screen is the first page available in the website, starting from it users can perform their authentication while guests can fill the registration's fields to become new user. Notice that "userhome" screen is marked with the landmark *$*, it means that it's reachable from every page of the system. Some part of the diagram can be only a hint to its real development, this to guarantee an easier comprehension of this diagram and the followings:

- **Authentication Panel** and **Registration Panel** are in all guest's pages;
- **User Panel, Buttons** and **Calendar** are in all user's pages;

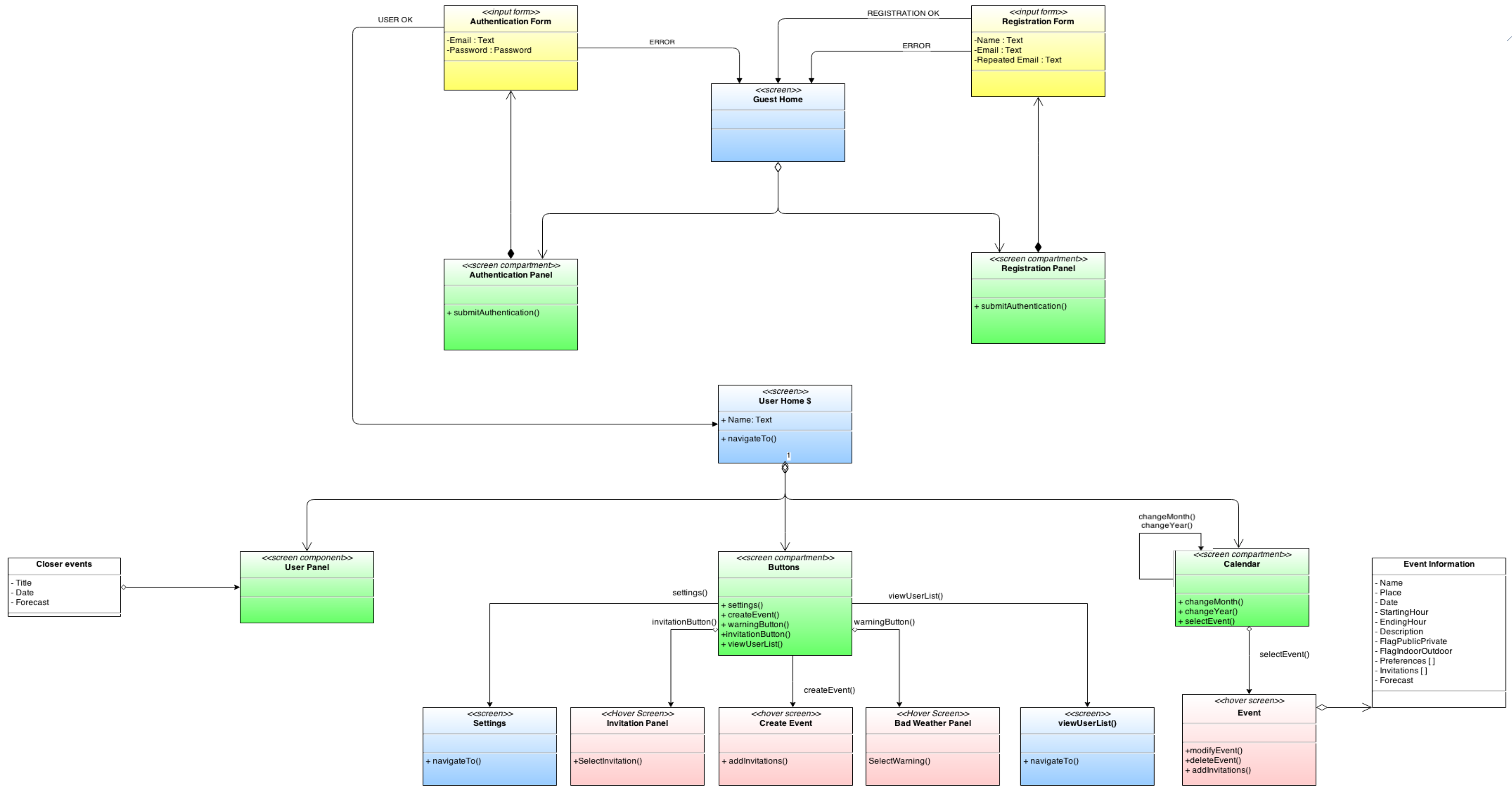
In the specific:

- Guest

The Guest's panel is the common component that can be used in a website to provide the input form through which is possible to get an authentication or a registration according to the to the guest's necessities.

- User

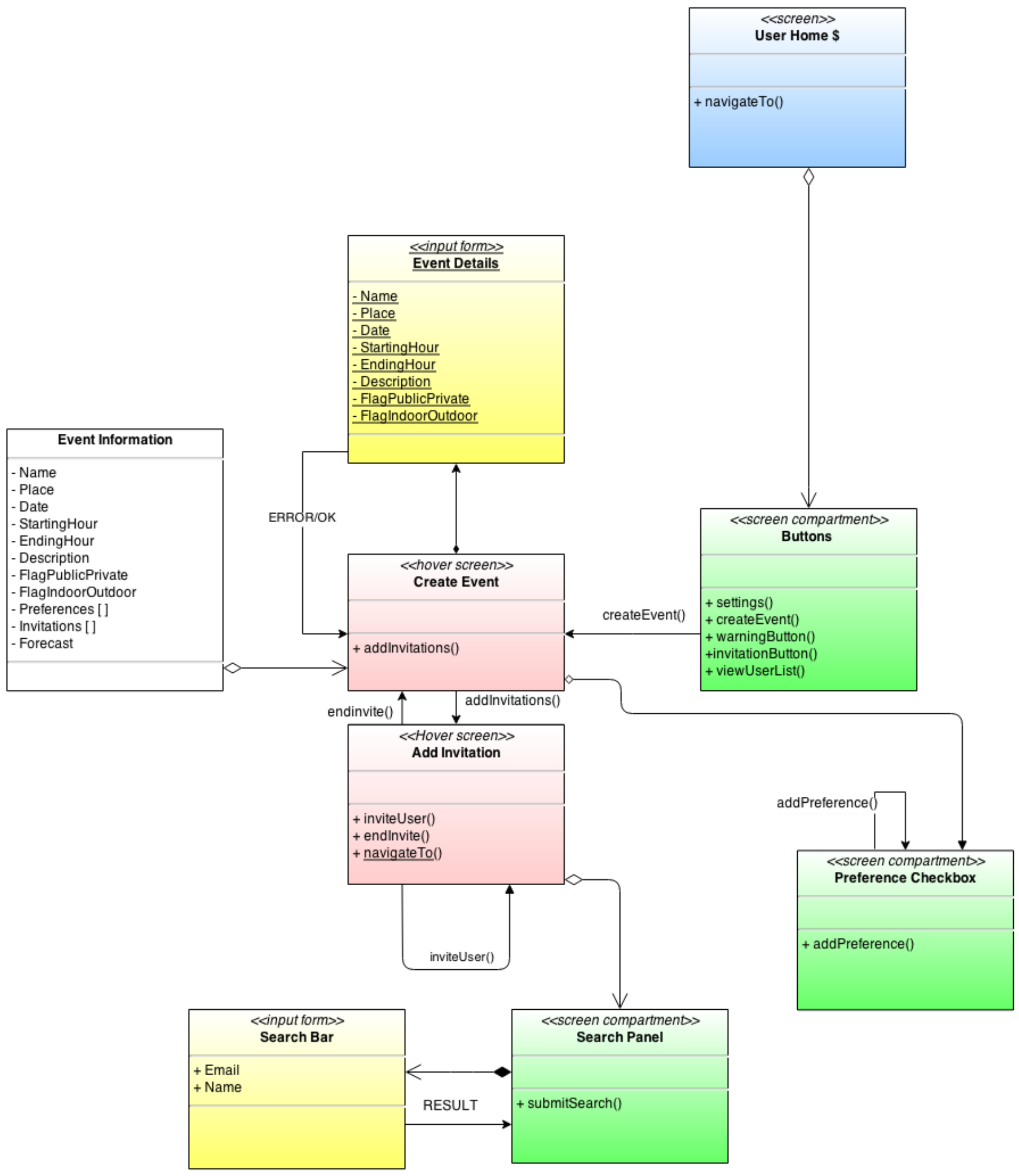
The Buttons component represent a component full of buttons through which is possible to reach specific functionalities. The functionalities are represented intuitively by the screen's name and the will be deepened in their appropriate UX. The Calendar component will show the presence of event in the displayed days and through each of them will be directly managed the event creation, modification and elimination. Each day will also provide the detail of events that are going to take place in that day.



USER OK
<<input form>>
Authentication Form
-Email : Text
-Password : Password
ERROR
REGISTRATION OK
ERROR
<<screen>>
Guest Home
<<input form>>
Registration Form
-Name : Text
-Email : Text
-Repeated Email : Text
<<screen compartment>>
Authentication Panel
+ submitAuthentication()
<<screen compartment>>
Registration Panel
+ submitAuthentication()
<<screen>>
User Home $
+ Name: Text
+ navigateTo()
1
Closer events
- Title
- Date
- Forecast
<<screen component>>
User Panel
<<screen compartment>>
Buttons
+ settings()
+ createEvent()
+ warningButton()
+invitationButton()
+ viewUserList()
settings()
invitationButton()
viewUserList()
warningButton()
createEvent()
changeMonth()
changeYear()
<<screen compartment>>
Calendar
+ changeMonth()
+ changeYear()
+ selectEvent()
selectEvent()
Event Information
- Name
- Place
- Date
- StartingHour
- EndingHour
- Description
- FlagPublicPrivate
- FlagIndoorOutdoor
- Preferences [ ]
- Invitations [ ]
- Forecast
<<screen>>
Settings
+ navigateTo()
<<Hover Screen>>
Invitation Panel
+SelectInvitation()
<<hover screen>>
Create Event
+ addInvitations()
<<Hover Screen>>
Bad Weather Panel
SelectWarning()
<<screen>>
viewUserList()
+ navigateTo()
<<hover screen>>
Event
+modifyEvent()
+deleteEvent()
+ addInvitations()

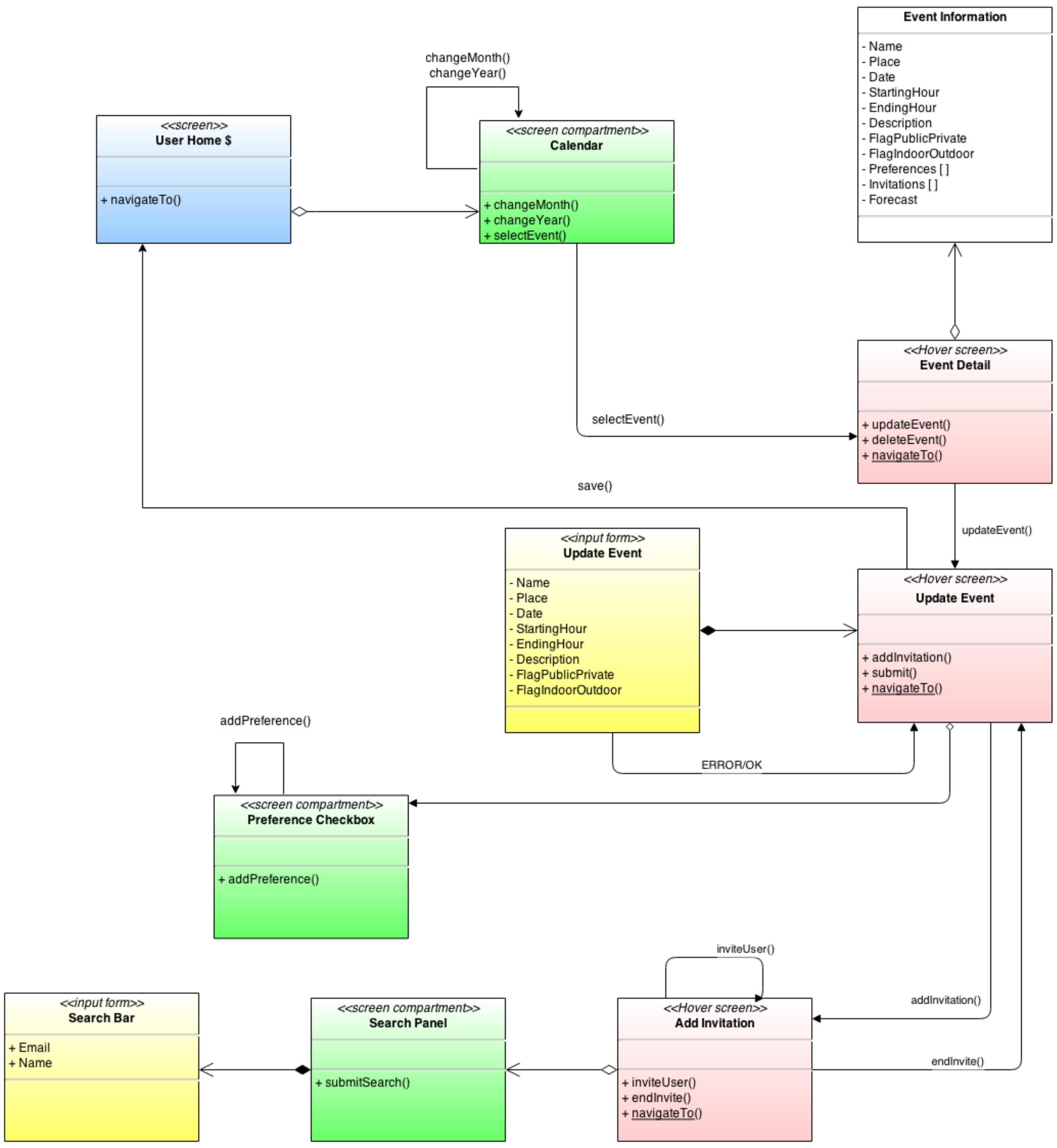
4.2 Create Event

This UX diagram represents how the user interacts with the system for the creation of an event.
In the "user home" the user can select a day from the calendar and, through the hover screen "create event" opened by the system, he can create an event. In this script there is also a checkbox in which the user can select the weather preference. Furthermore there is a button that provide the possibility to make some invitations through the "hover screen" Add invitation.



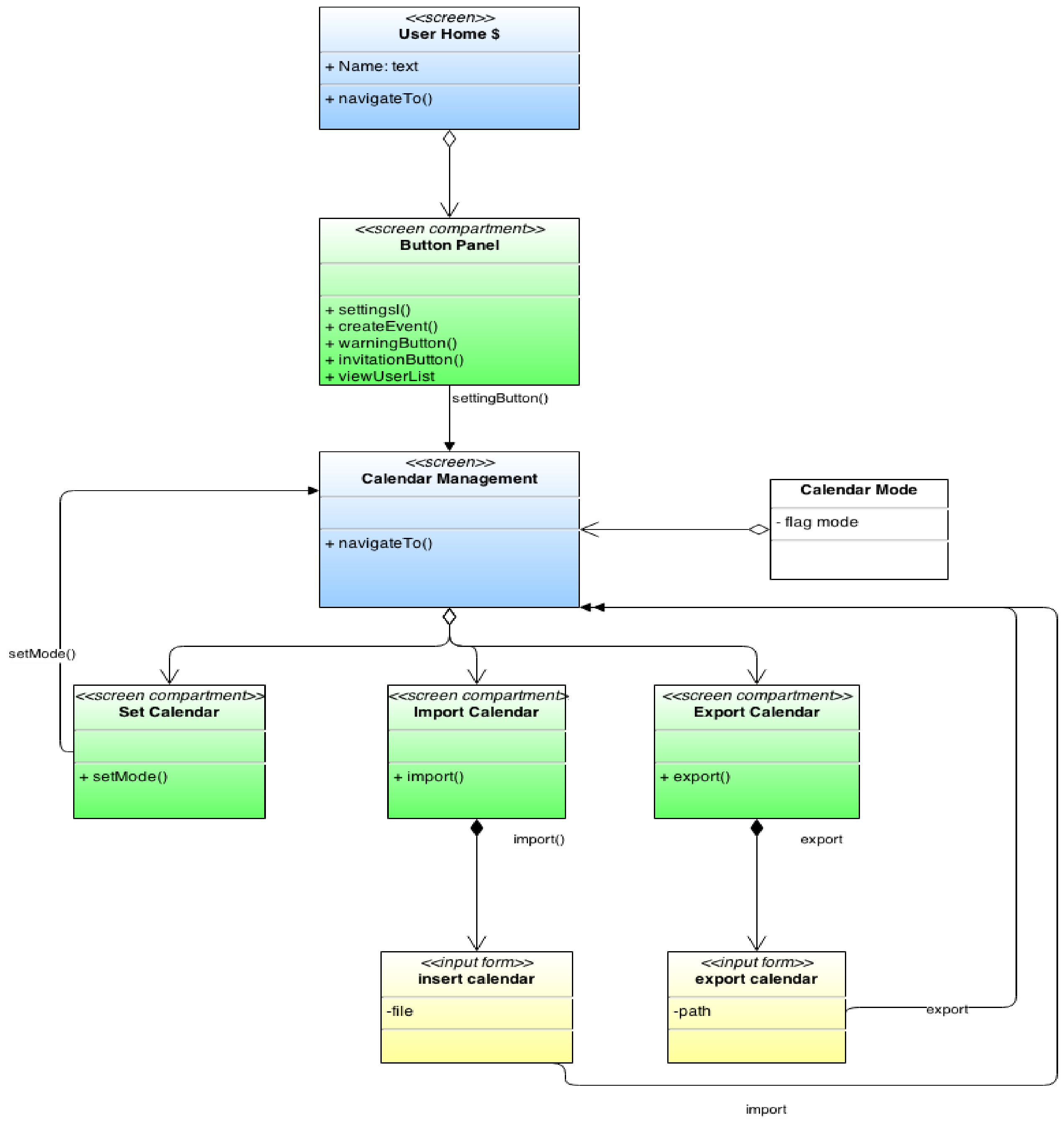
4.3 Event management

The event management will be accessible directly from the calendar compartment. In our idea the user will be able to manage his events choosing them from the day's tail of the calendar which contains them. Once chosen a "hover screen" will be appear with all event's details divided into fields that the user can easily modify. Furthermore to add more invitations will be provided simple interface in which the user can select users.



4.4 Import, export and set public/private

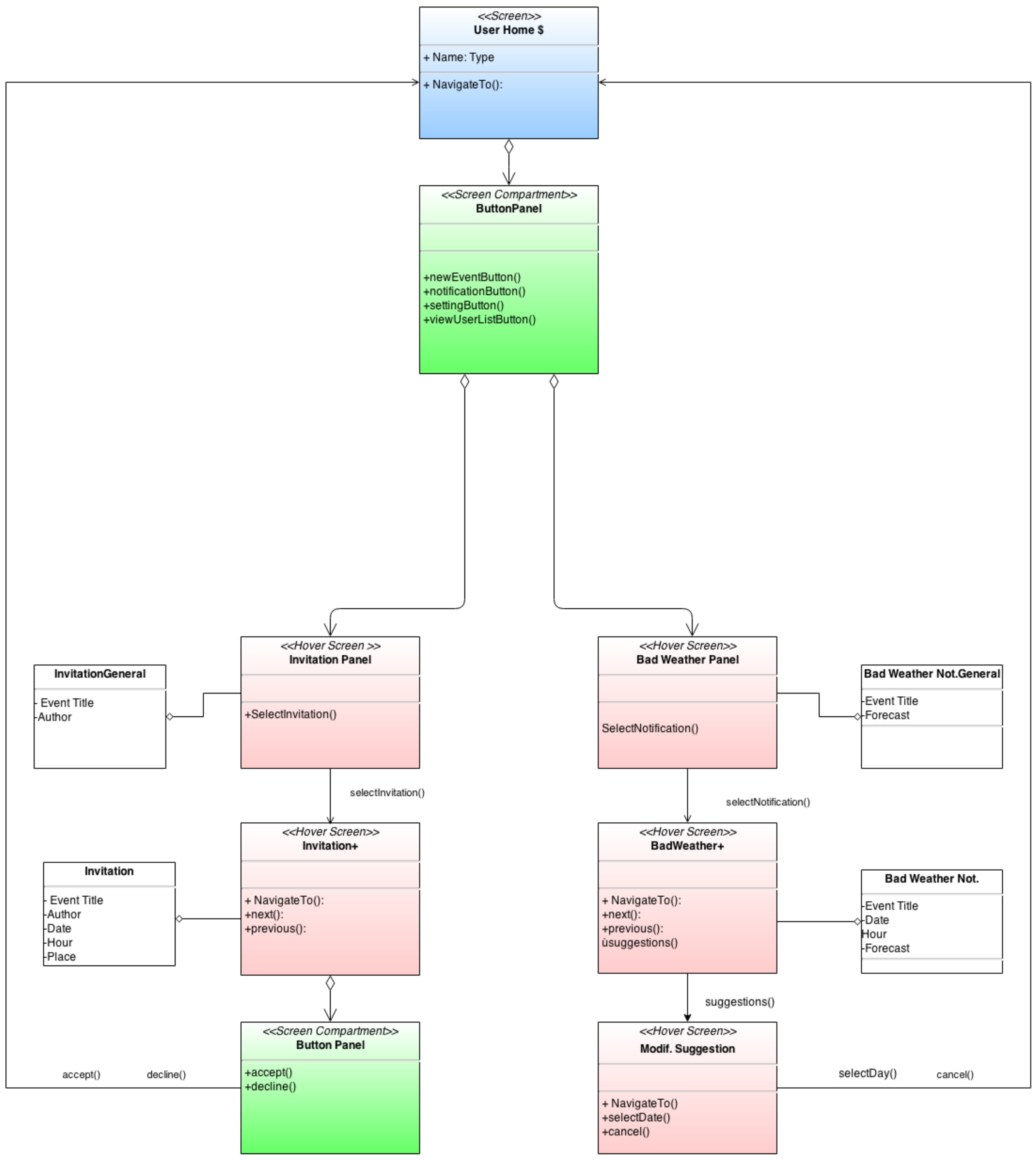
This UX diagram represents the interaction with the calendar. From the user home, the user can reach the settings page using the relative button. When the user is in the settings page, he can see how the calendar is set (public or private mode) and in case he can change the status. Furthermore, there are two more functionalities: it is possible to import an existing calendar and to export the own calendar. Each functionality is reachable through three different buttons.



4.5 Notification management

This UX diagram represents how the user interacts with the system in the management of notifications. From "user home” screen ,using the relative buttons, he can interact with the system through the windows popup (hover screen , see legend) allowing to manage invitations , accepting or rejecting through its buttons, or manage his own events that have a bad weather warning with the possibility to select the following days that match his preferences . This functionality is provided automatically by the system.

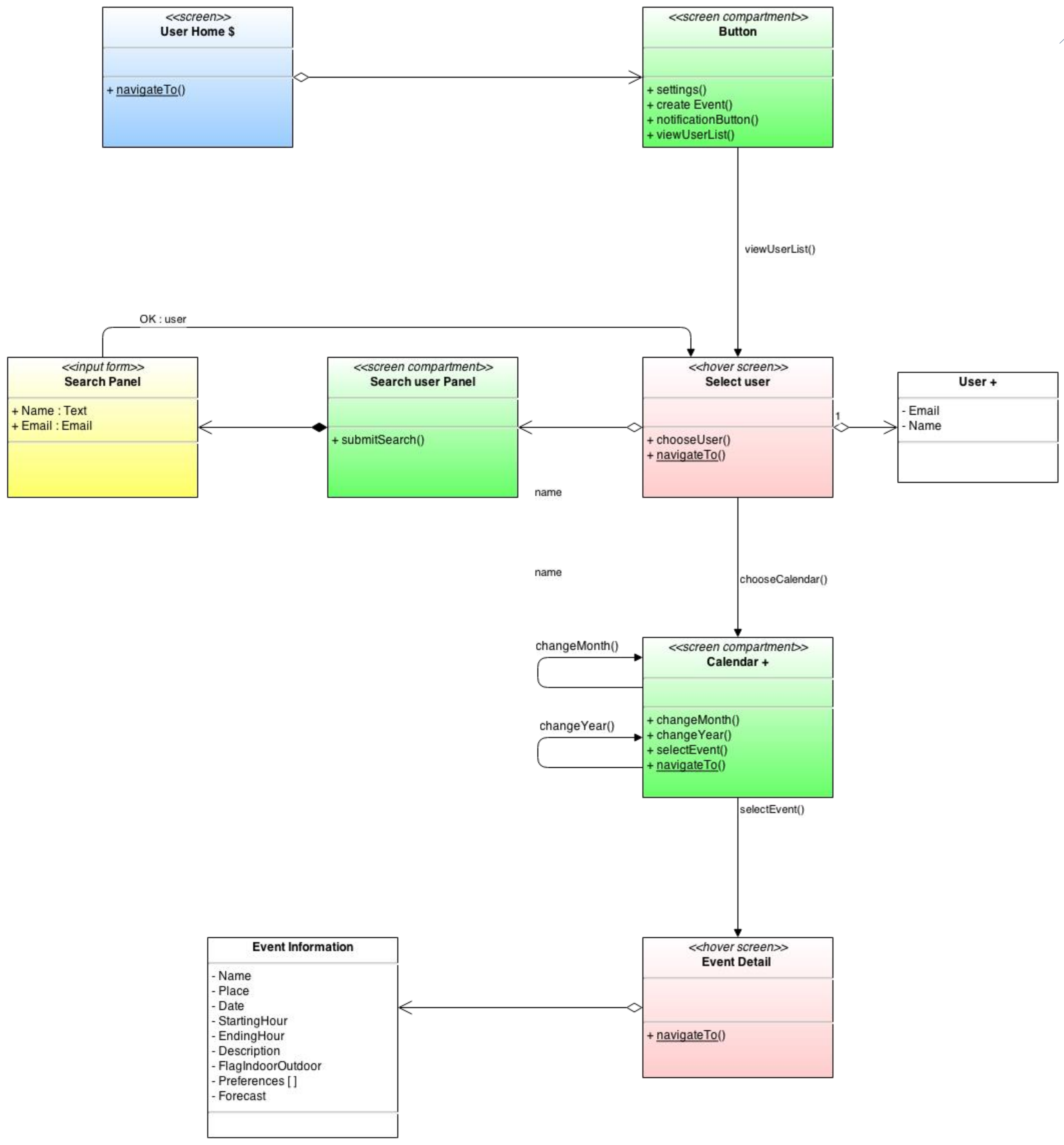
(Diagram next page)


<<Screen>>
User Home $
+ Name: Type
+ NavigateTo():
<<Screen Compartment>>
ButtonPanel
+newEventButton()
+notificationButton()
+settingButton()
+viewUserListButton()
InvitationGeneral
- Event Title
-Author
<<Hover Screen >>
Invitation Panel
+SelectInvitation()
<<Hover Screen>>
Bad Weather Panel
SelectNotification()
Bad Weather Not.General
-Event Title
-Forecast
selectInvitation()
selectNotification()
Invitation
- Event Title
-Author
-Date
-Hour
-Place
<<Hover Screen>>
Invitation+
+ NavigateTo():
+next():
+previous():
<<Hover Screen>>
BadWeather+
+ NavigateTo():
+next():
+previous():
ùsuggestions()
Bad Weather Not.
-Event Title
-Date
Hour
-Forecast
suggestions()
<<Screen Compartment>>
Button Panel
+accept()
+decline()
<<Hover Screen>>
Modif. Suggestion
+ NavigateTo()
+selectDate()
+cancel()
accept()
decline()
selectDay()
cancel()

4.6 View public Calendar

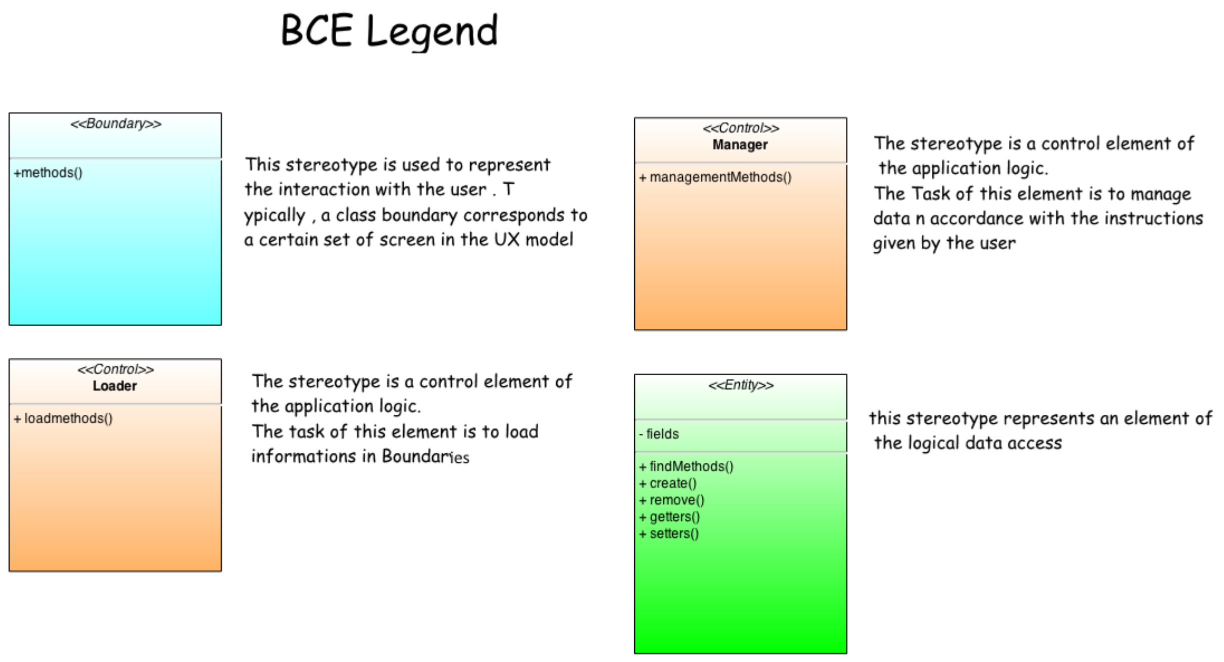
This UX show the interfaces through which the user can see the public calendar of the other users enrolled into the website. Starting from the user home there will be a button that if pressed will be able to show the list of user who set their calendar as public. Once chosen the user will be possible to navigate into his calendar watching the details of the user's events which must be set as public.

(Diagram next page)


<<screen>>
User Home $
+ navigateTo()
<<screen compartment>>
Button
+ settings()
+ create Event()
+ notificationButton()
+ viewUserList()
viewUserList()
OK : user
<<input form>>
Search Panel
+ Name : Text
+ Email : Email
<<screen compartment>>
Search user Panel
+ submitSearch()
<<hover screen>>
Select user
+ chooseUser()
+ navigateTo()
1
User +
- Email
- Name
name
name
chooseCalendar()
changeMonth()
changeYear()
<<screen compartment>>
Calendar +
+ changeMonth()
+ changeYear()
+ selectEvent()
+ navigateTo()
selectEvent()
Event Information
- Name
- Place
- Date
- StartingHour
- EndingHour
- Description
- FlagIndoorOutdoor
- Preferences [ ]
- Forecast
<<hover screen>>
Event Detail
+ navigateTo()

5 BCE Diagram

We decided to give a further design schema of MeteoCal using the Boundary-Control-Entity pattern, because it is very close to the Model-View-Controller pattern and UML defines a standard to represent it.. All the methods of the screens are written into the appropriate boundaries (maybe some names changed a bit only to make them more understandable in this context). At the bottom of the page we insert a legend trying to make the diagrams easier to understand.



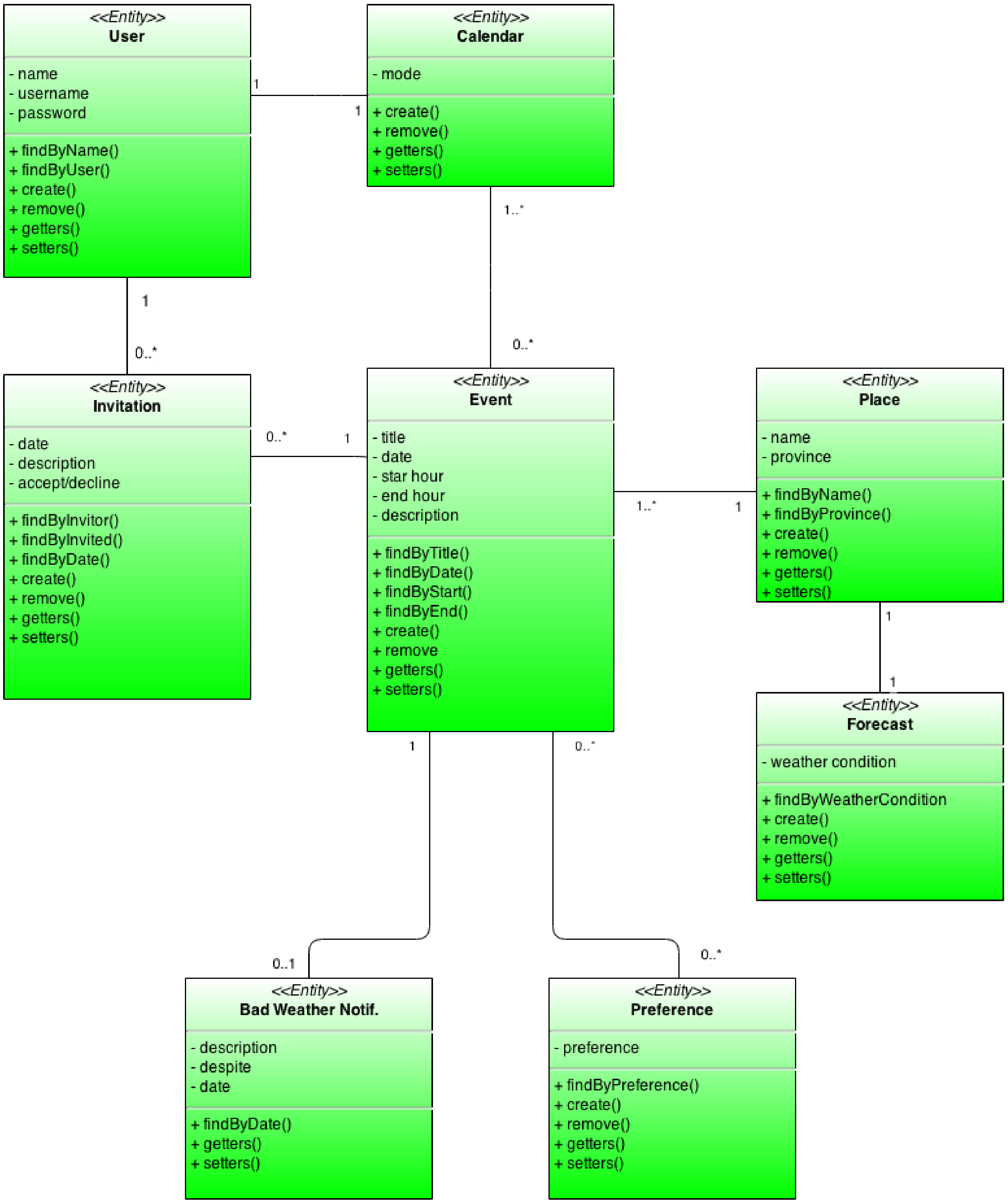
This stereotype is used to represent the interaction with the user . T ypically , a class boundary corresponds to a certain set of screen in the UX model

The stereotype is a control element of the application logic.
The Task of this element is to manage data n accordance with the instructions given by the user

The stereotype is a control element of the application logic.
The task of this element is to load informations in Boundaries

this stereotype represents an element of the logical data access

5.1 Entity overview



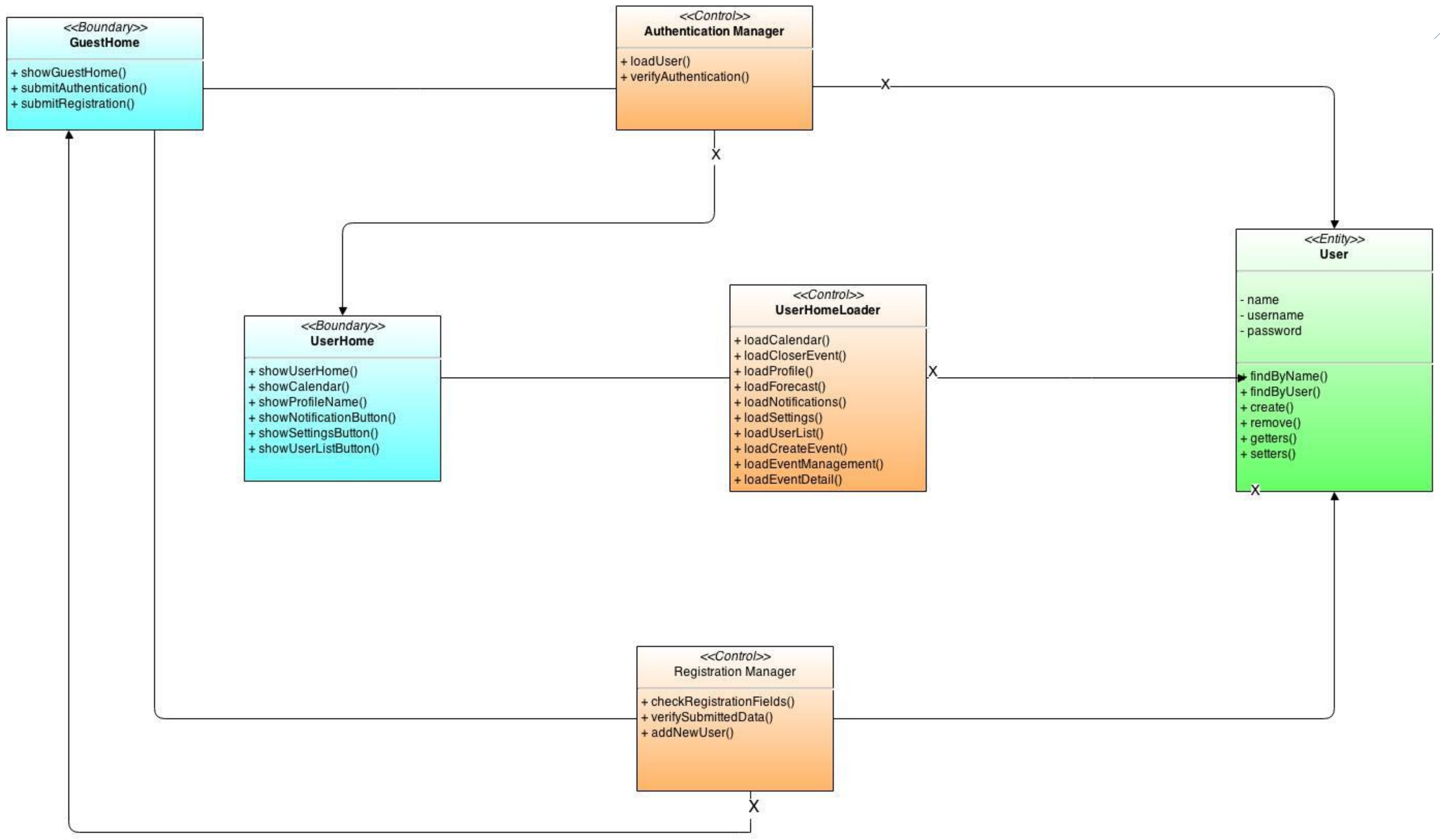
5.2 Registration and Authentication

The BCE diagram is characterized by two boundary “userhome” and “guesthome” which represent the interface with the correspondent actor-users: “user” and “guest”. The objective of the boundaries is to represent the functionalities that its user can exploit in his homepage.

In the following lines will be described a concise description of the controllers involved into the diagram:

- ***RegistrationManager:*** this controller manages the verification of the data submitted at the registration time. In case of successful registration attempt, it will provide to the creation of a new user.
- ***AuthenticationManager:*** this controller manages the verification of authentication fields. The logic is that firstly, it examines if the fields are filled correctly. Then the controller can load the correspondent user finding it by username and then check the password. This mechanism will be better examined later in the Sequence Diagrams section.
- **UserHomeLoader:** this controller is responsible of the general management of the user’s home. It will load all the data associated to the home’s components like the calendar and all its events, which is the perfect example.

(Diagram next page)

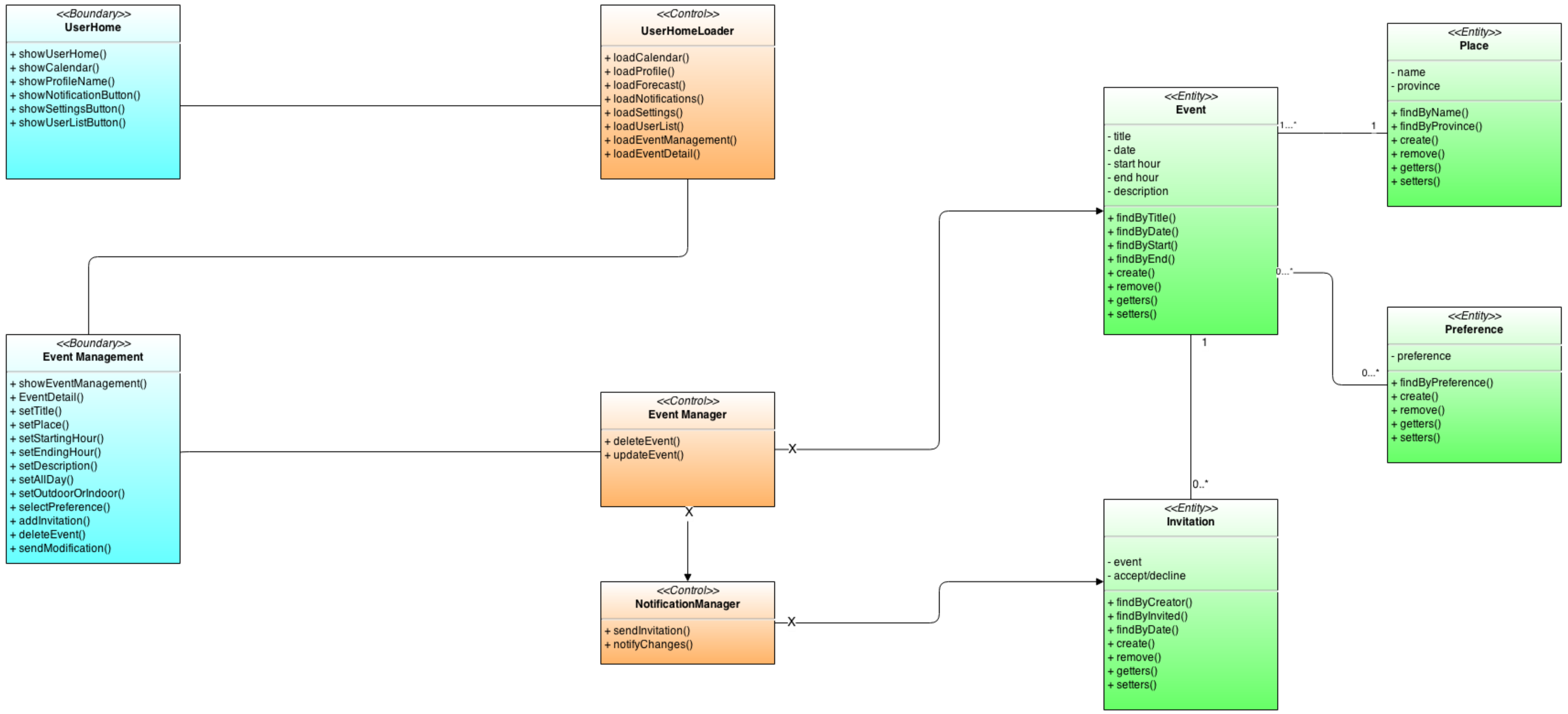

<<Boundary>>
GuestHome
+ showGuestHome()
+ submitAuthentication()
+ submitRegistration()
<<Control>>
Authentication Manager
+ loadUser()
+ verifyAuthentication()
X
X
<<Entity>>
User
- name
- username
- password
+ findByName()
+ findByUser()
+ create()
+ remove()
+ getters()
+ setters()
X
<<Boundary>>
UserHome
+ showUserHome()
+ showCalendar()
+ showProfileName()
+ showNotificationButton()
+ showSettingsButton()
+ showUserListButton()
<<Control>>
UserHomeLoader
+ loadCalendar()
+ loadCloserEvent()
+ loadProfile()
+ loadForecast()
+ loadNotifications()
+ loadSettings()
+ loadUserList()
+ loadCreateEvent()
+ loadEventManagement()
+ loadEventDetail()
X
<<Control>>
Registration Manager
+ checkRegistrationFields()
+ verifySubmittedData()
+ addNewUser()
X

5.3 Event management

In the following lines are described the controllers involved within the eventManagementBoundary's scope, should be notice that the UserHomeLoader's role has been already written in the previous pages but it is useful to load data into the event management boundary:

- ***Event Manager:*** this controller is involved in the creation, modification or elimination of an event. This controller also has an interaction with the Notification Manager. For example in case of invitation by the creator the event manager during the data processing will inform The notification manager that at this point will be able to satisfy his obligation.
- ***Notification Manager:*** this controller manages all the notifications and offers methods to deliver the notifications to their recipient.

(Diagram next page)

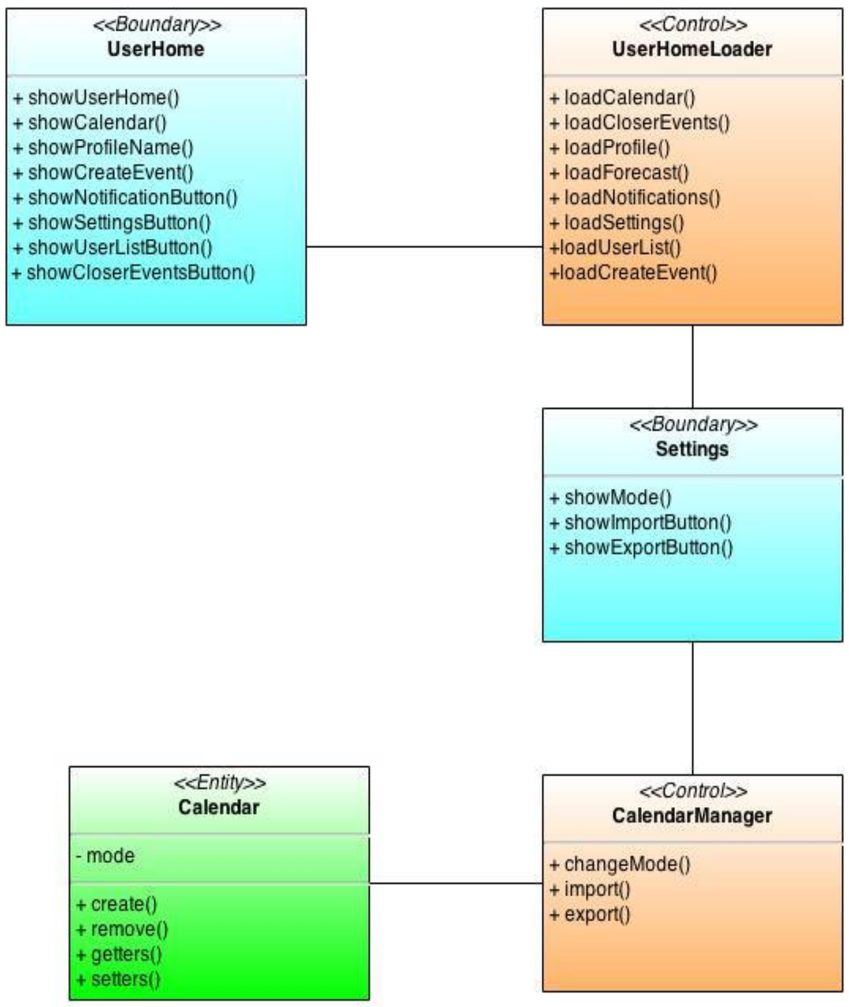


<<Boundary>>
UserHome
+ showUserHome()
+ showCalendar()
+ showProfileName()
+ showNotificationButton()
+ showSettingsButton()
+ showUserListButton()
<<Control>>
UserHomeLoader
+ loadCalendar()
+ loadProfile()
+ loadForecast()
+ loadNotifications()
+ loadSettings()
+ loadUserList()
+ loadEventManagement()
+ loadEventDetail()
<<Entity>>
Place
- name
- province
+ findByName()
+ findByProvince()
+ create()
+ remove()
+ getters()
+ setters()
<<Entity>>
Event
- title
- date
- start hour
- end hour
- description
+ findByTitle()
+ findByDate()
+ findByStart()
+ findByEnd()
+ create()
+ remove()
+ getters()
+ setters()
1...*
1
0...*
0...*
1
0..*
<<Entity>>
Preference
- preference
+ findByPreference()
+ create()
+ remove()
+ getters()
+ setters()
<<Boundary>>
Event Management
+ showEventManagement()
+ EventDetail()
+ setTitle()
+ setPlace()
+ setStartingHour()
+ setEndingHour()
+ setDescription()
+ setAllDay()
+ setOutdoorOrIndoor()
+ selectPreference()
+ addInvitation()
+ deleteEvent()
+ sendModification()
<<Control>>
Event Manager
+ deleteEvent()
+ updateEvent()
X
X
<<Control>>
NotificationManager
+ sendInvitation()
+ notifyChanges()
X
<<Entity>>
Invitation
- event
- accept/decline
+ findByCreator()
+ findByInvited()
+ findByDate()
+ create()
+ remove()
+ getters()
+ setters()

5.4 Import, Export, set public/private

This BCE diagram represents how the system handles the upload / download of calendars and your availability to make his public calendar from the "Settings" boundary.
In particular the controls involved are:

- ***UserHomeLoader:*** already descripted in previous pages.
- ***CalendarManager:*** the task of this control is to set the Boolean that makes calendar public or not, import XML file provided by the user and store the user calendar in a specified file.



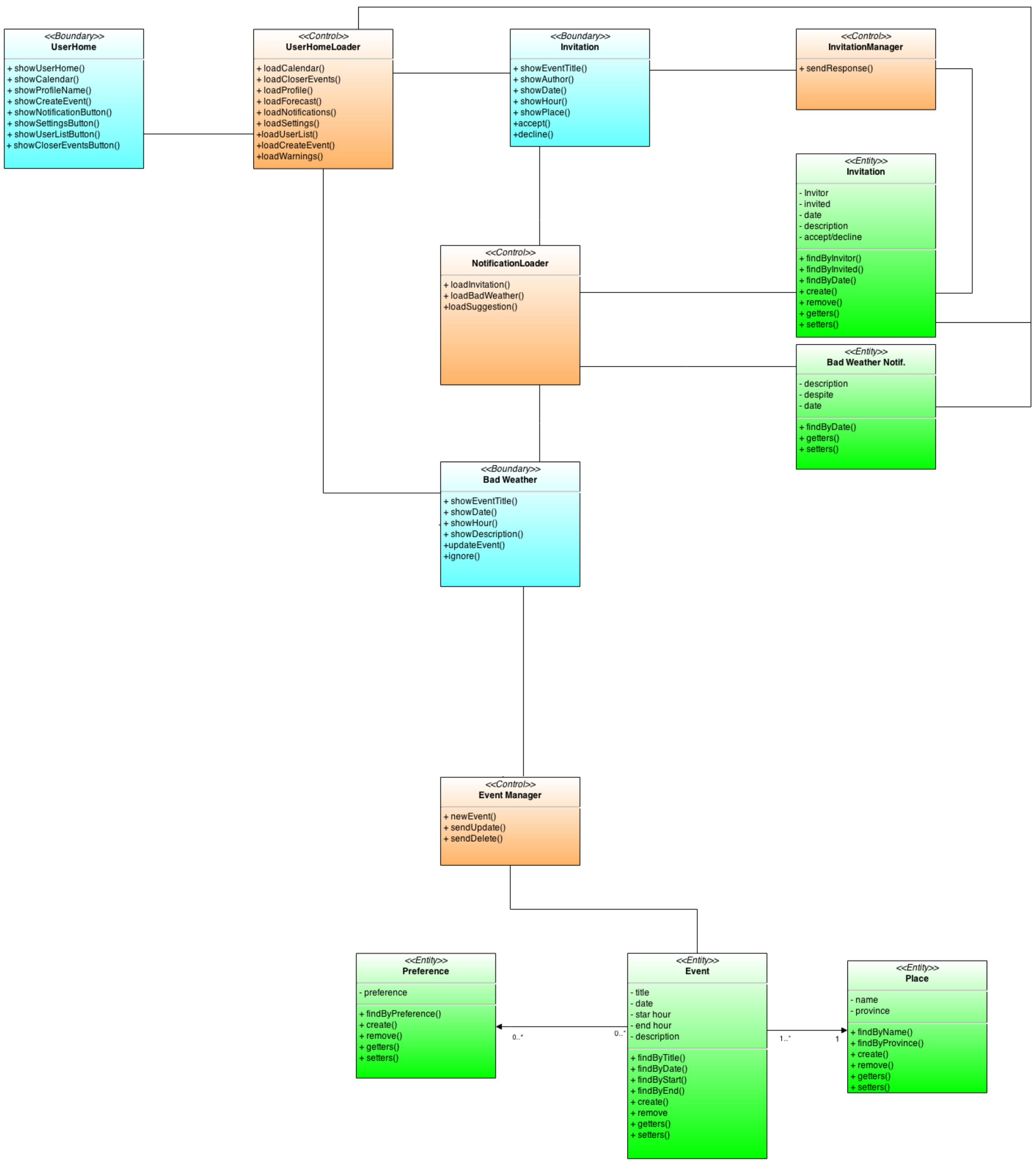
5.5 Notification management

This BCE diagram represents the management of notifications by the system (invitations and warnings). The boundary "UserHome" the user can select through the buttons the two boundary “invitation" and “bad Weather" and related management operations.

The Controls Involved in this diagram are:

- ***UserHomeLoader:*** already descripted in previous pages
- ***NotificationLoader:*** the role of this control is load invitations and Warnings ad in the latter case generate suggestions to the user for other days with the same weather specified during the creation of the event.
- ***InvitationManager:*** the role of this control is to transmit user’s decision about the selected invitation to the system
- ***Event Manager:*** the role of this control is to transmit user’s decision about event warning to the system

  (Diagram next page)



<<Boundary>>
UserHome
+ showUserHome()
+ showCalendar()
+ showProfileName()
+ showCreateEvent()
+ showNotificationButton()
+ showSettingsButton()
+ showUserListButton()
+ showCloserEventsButton()
<<Control>>
UserHomeLoader
+ loadCalendar()
+ loadCloserEvents()
+ loadProfile()
+ loadForecast()
+ loadNotifications()
+ loadSettings()
+loadUserList()
+loadCreateEvent()
+loadWarnings()
<<Boundary>>
Invitation
+ showEventTitle()
+ showAuthor()
+ showDate()
+ showHour()
+ showPlace()
+accept()
+decline()
<<Control>>
InvitationManager
+ sendResponse()
<<Entity>>
Invitation
- Invitor
- invited
- date
- description
- accept/decline
+ findByInvitor()
+ findByInvited()
+ findByDate()
+ create()
+ remove()
+ getters()
+ setters()
<<Control>>
NotificationLoader
+ loadInvitation()
+ loadBadWeather()
+loadSuggestion()
<<Entity>>
Bad Weather Notif.
- description
- despite
- date
+ findByDate()
+ getters()
+ setters()
<<Boundary>>
Bad Weather
+ showEventTitle()
+ showDate()
+ showHour()
+ showDescription()
+updateEvent()
+ignore()
<<Control>>
Event Manager
+ newEvent()
+ sendUpdate()
+ sendDelete()
<<Entity>>
Preference
- preference
+ findByPreference()
+ create()
+ remove()
+ getters()
+ setters()
0..*
0..*
<<Entity>>
Event
- title
- date
- star hour
- end hour
- description
+ findByTitle()
+ findByDate()
+ findByStart()
+ findByEnd()
+ create()
+ remove
+ getters()
+ setters()
1..*
1
<<Entity>>
Place
- name
- province
+ findByName()
+ findByProvince()
+ create()
+ remove()
+ getters()
+ setters()

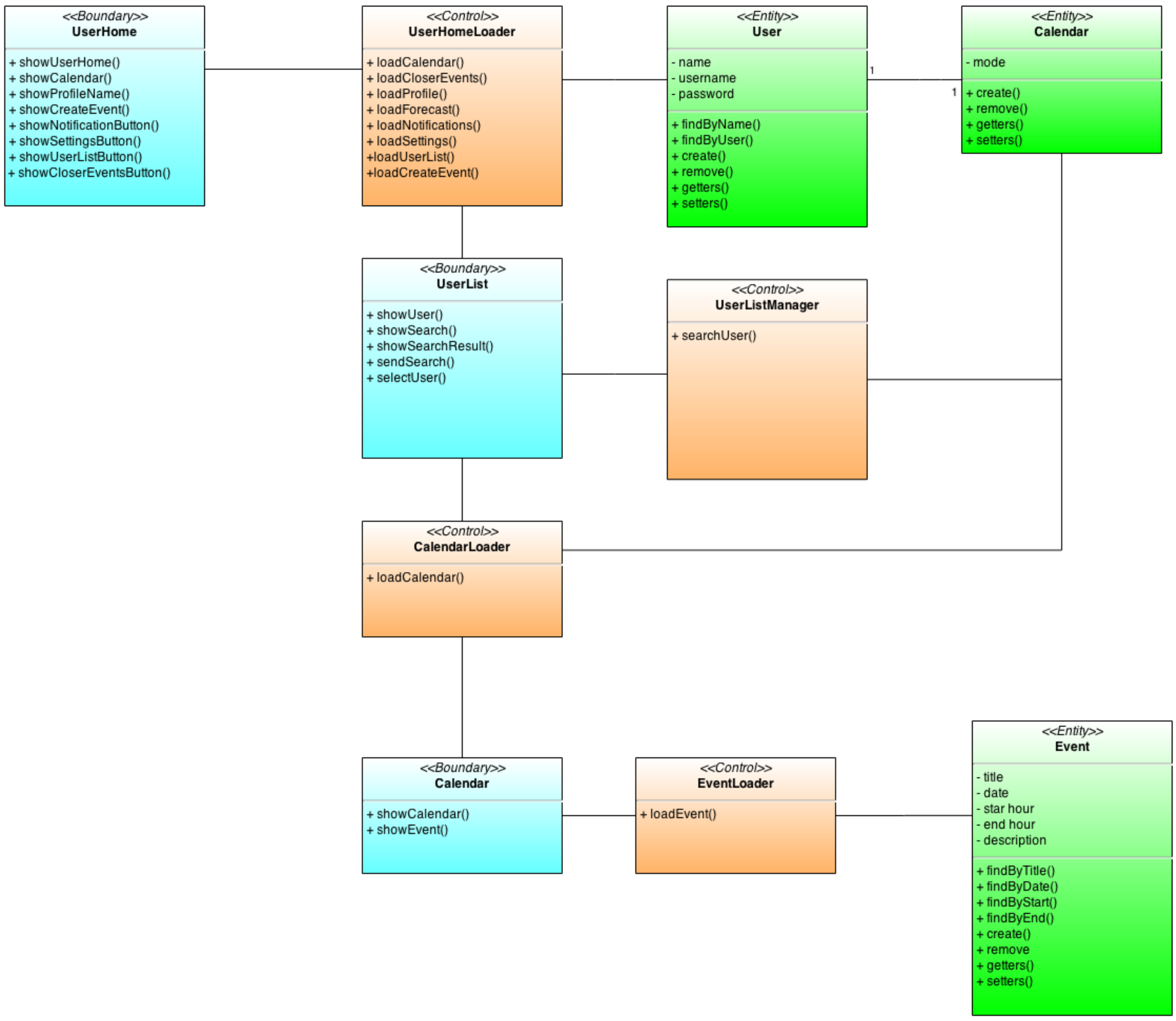
5.6 View public calendar

The BCE diagram is characterized by three boundaries “userHome”, “userList” and “calendar” that represent the interface that the system provides to user to view the public calendar.

This diagram shows four different controllers:

- ***UserHomeLoader:*** this controller is responsible of the general management of the user’s home. It will load all the data associated to the home components like the calendar and all its events, which is the perfect example.
- ***UserListManager:*** this controller is responsible to provide the list of the user. It is linked to calendar entity to identify the public calendars.
- ***CalendarLoader:*** when a user selects a user, this controller load the relative calendar.
- ***EventLoader:*** this controller load all the event details

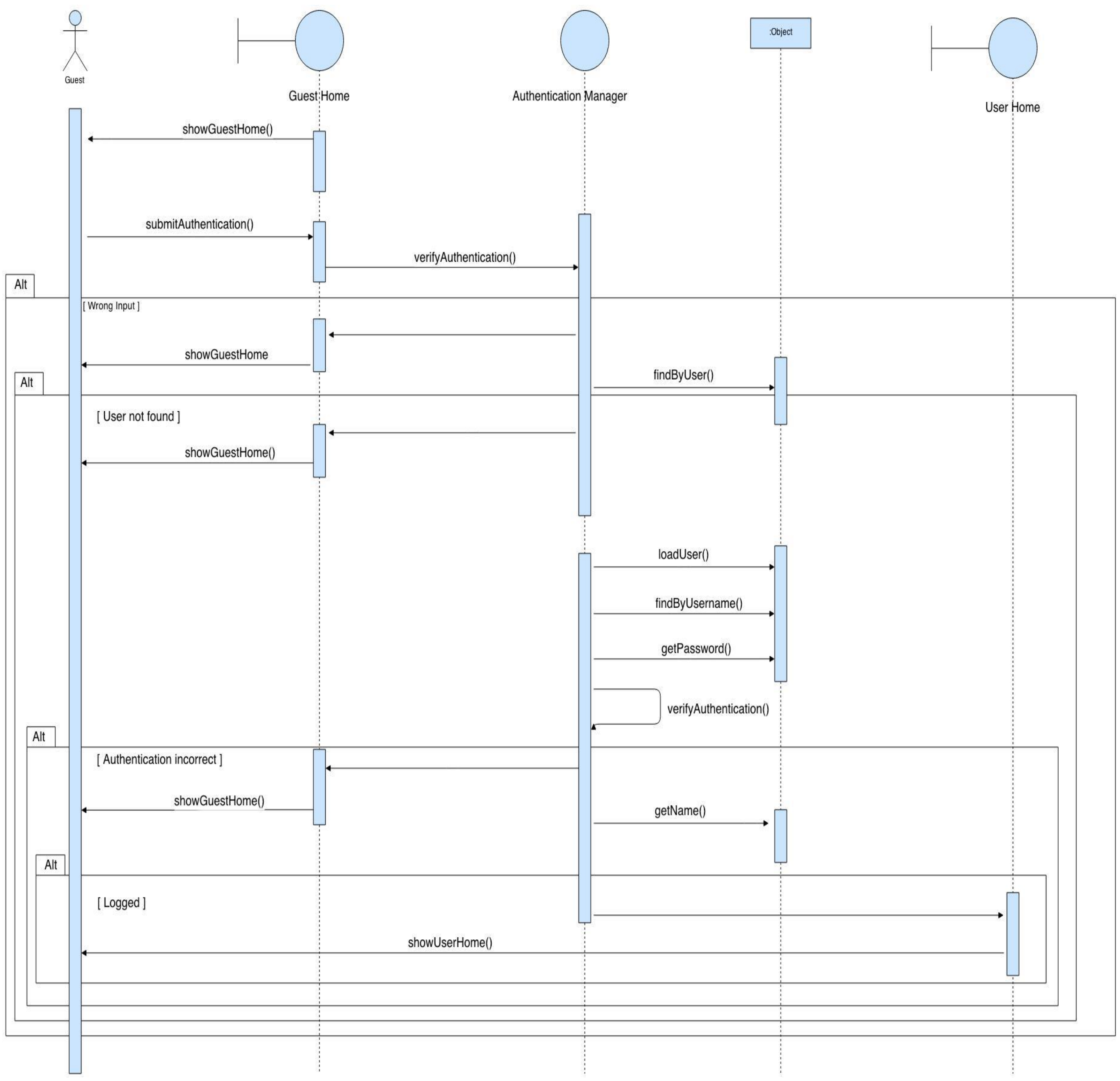
(Diagram next page)


<<Boundary>>
UserHome
+ showUserHome()
+ showCalendar()
+ showProfileName()
+ showCreateEvent()
+ showNotificationButton()
+ showSettingsButton()
+ showUserListButton()
+ showCloserEventsButton()
<<Control>>
UserHomeLoader
+ loadCalendar()
+ loadCloserEvents()
+ loadProfile()
+ loadForecast()
+ loadNotifications()
+ loadSettings()
+loadUserList()
+loadCreateEvent()
<<Entity>>
User
- name
- username
- password
+ findByName()
+ findByUser()
+ create()
+ remove()
+ getters()
+ setters()
1
1
<<Entity>>
Calendar
- mode
+ create()
+ remove()
+ getters()
+ setters()
<<Boundary>>
UserList
+ showUser()
+ showSearch()
+ showSearchResult()
+ sendSearch()
+ selectUser()
<<Control>>
UserListManager
+ searchUser()
<<Control>>
CalendarLoader
+ loadCalendar()
<<Boundary>>
Calendar
+ showCalendar()
+ showEvent()
<<Control>>
EventLoader
+ loadEvent()
<<Entity>>
Event
- title
- date
- star hour
- end hour
- description
+ findByTitle()
+ findByDate()
+ findByStart()
+ findByEnd()
+ create()
+ remove
+ getters()
+ setters()

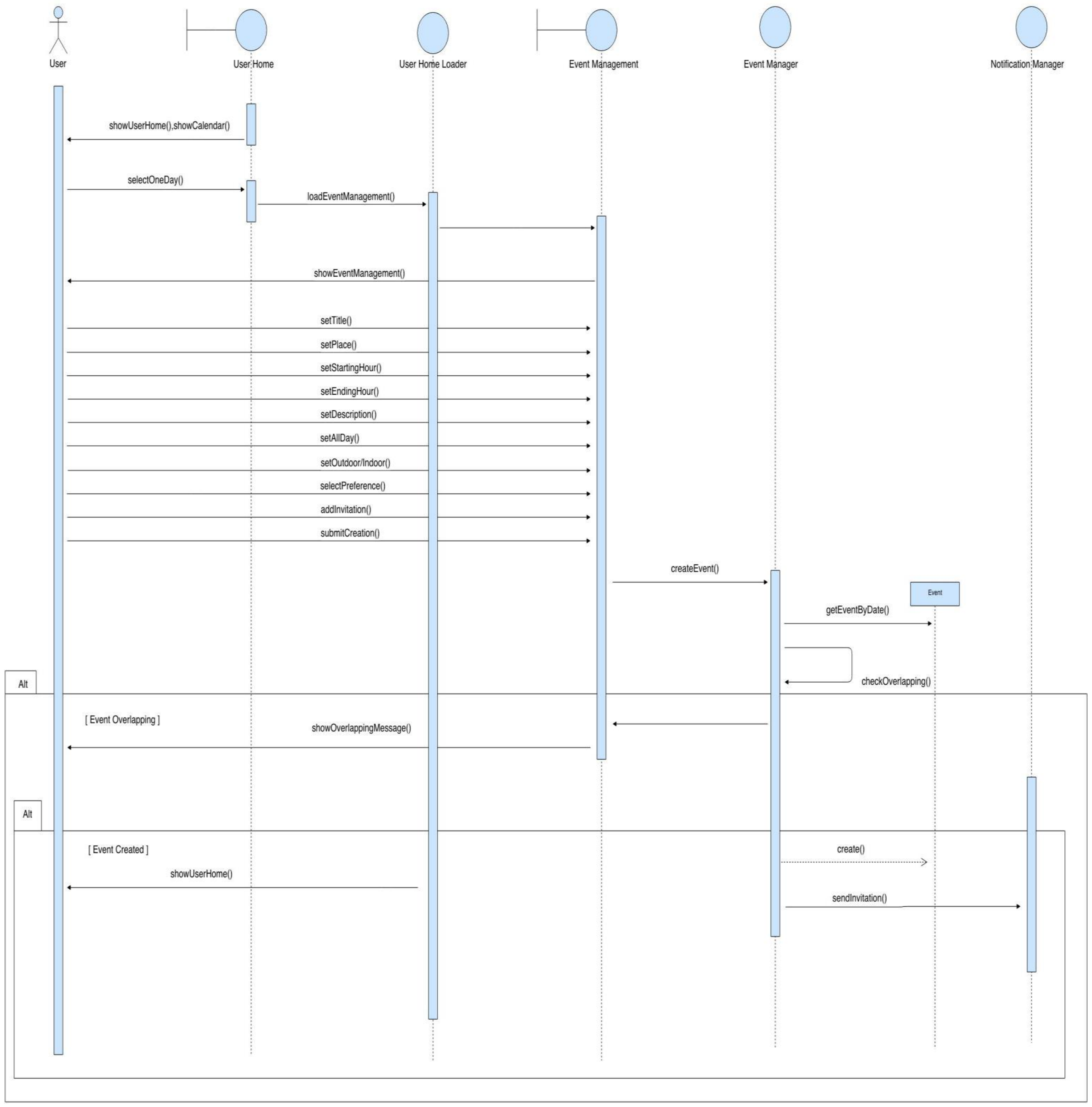
6 *Sequence Diagram*

We provide some sequence diagram to let the reader better understand BCE diagrams described above. All the methods used are the methods listed into the BCE in boundaries, controls and entities.

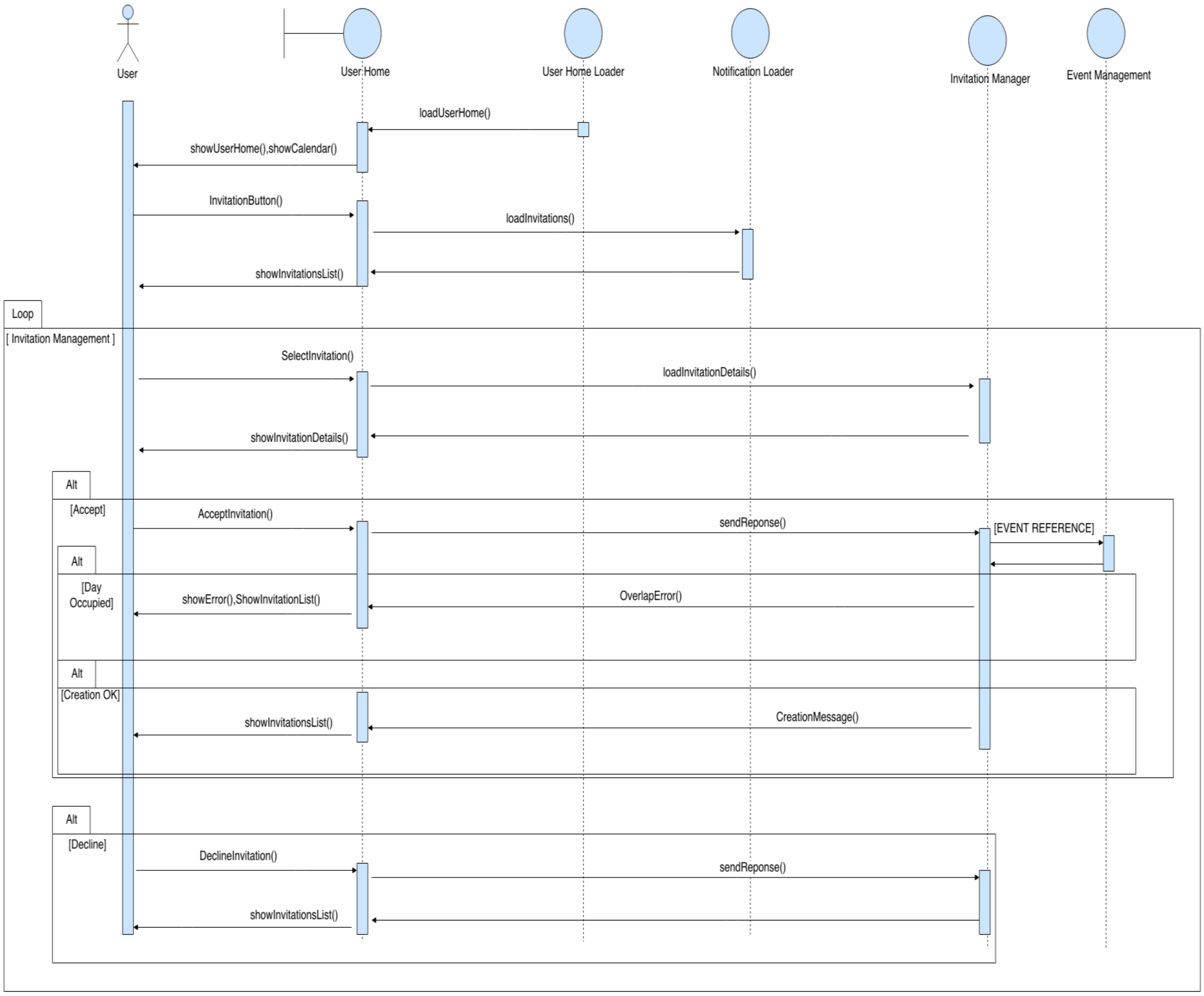
6.1 Authentication



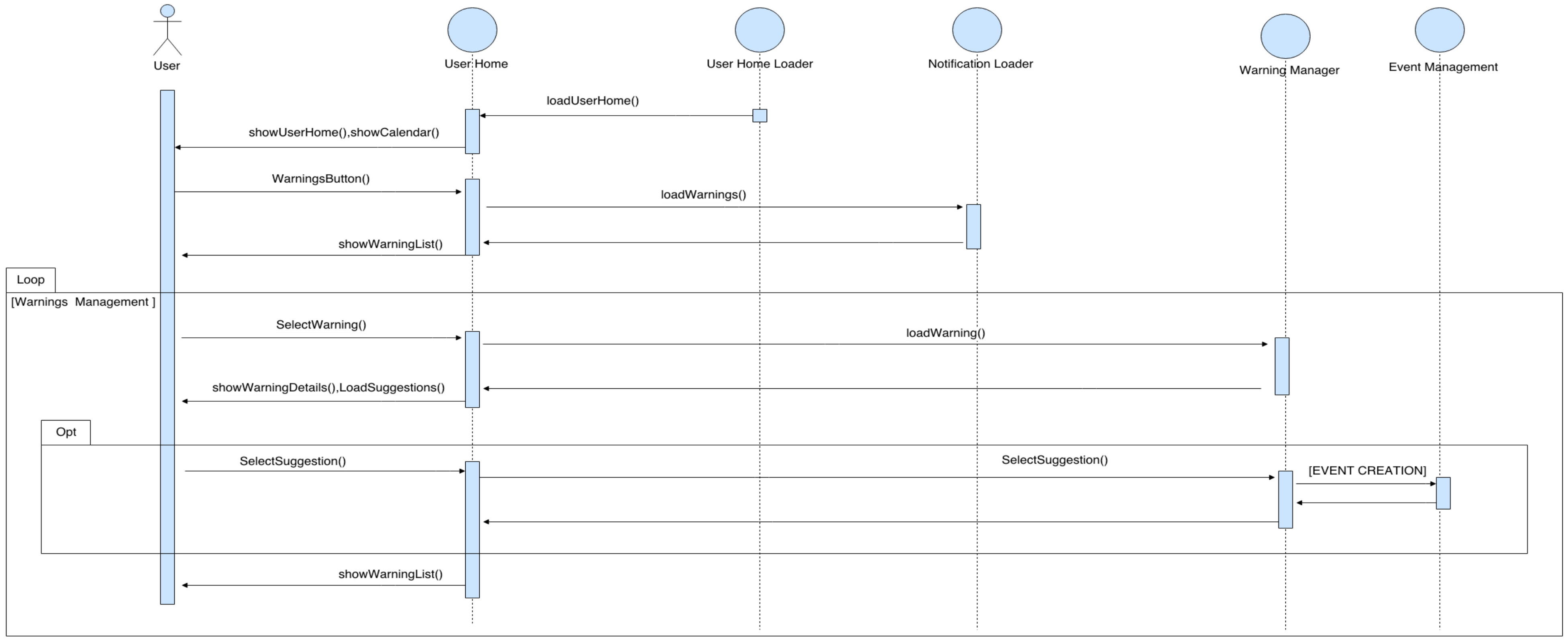
6.2 Create event



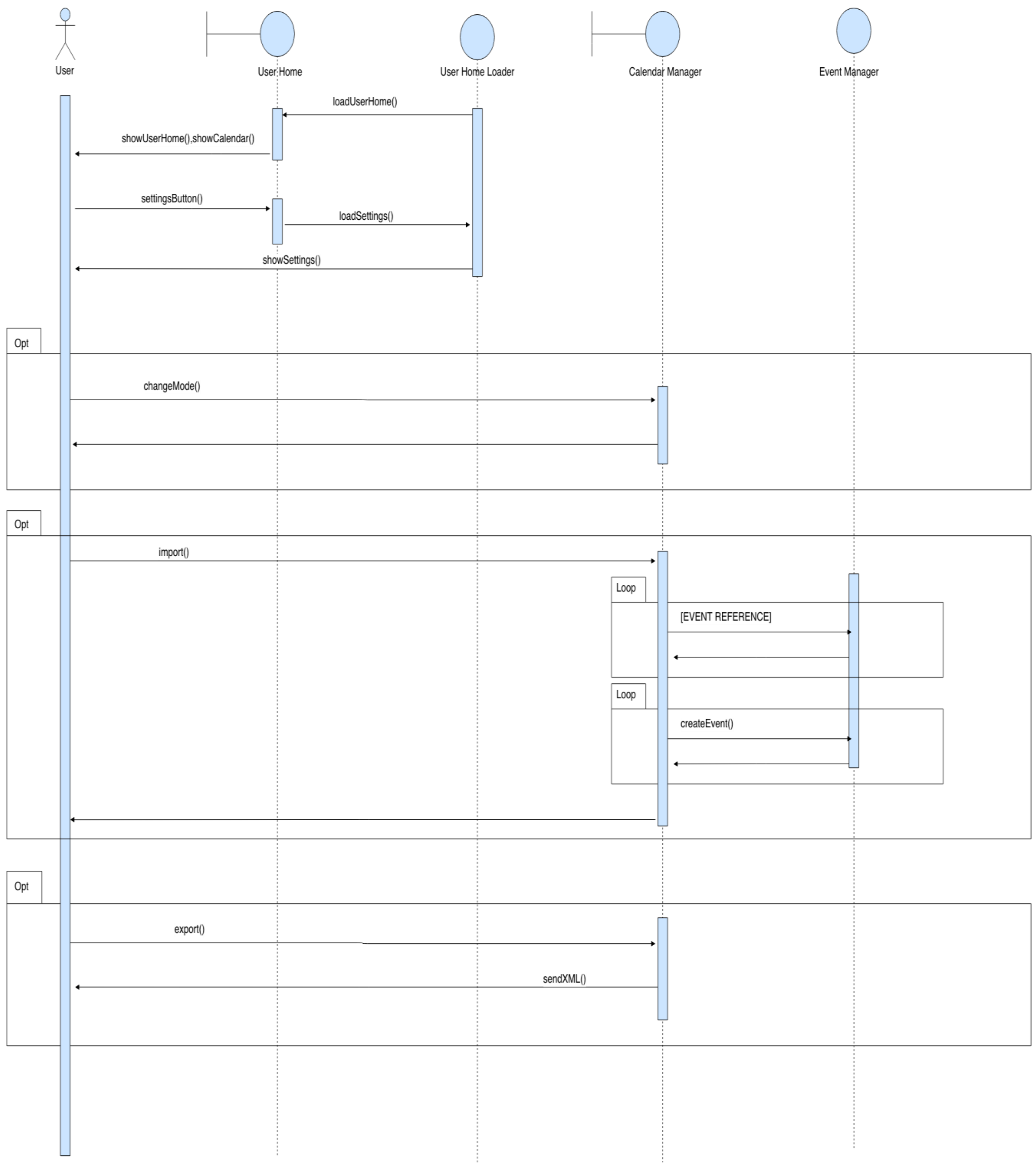
6.3 Invitation



6.4 Bad weather notification



6.5 Import, export, set public/private



7 ***Final consideration***

As we don’t know the Java development environment 2EE we decided to hold us to a higher level of abstraction, in order to describe the functionalities of the system.
In the specific we know that:

- the UX diagrams represent the xhtml pages used by the user to interface with the system
- the BCE diagrams provide a division of the levels of the system
- the sequence diagrams show the interaction between the system component in relation to the different functionalities