MeteoCal Project

Design Document 2.0 [1]

Authors:

Francesco Angelo {francesco.angelo@mail.polimi.it}

Valentina Ceriani {valentina.ceriani@mail.polimi.it}

Umberto Di Fabrizio {umberto.difabrizio@mail.polimi.it}

Prof: Elisabetta Di Nitto

Milan, January 25, 2015

[1] https://code.google.com/p/meteo-cal-angelo-ceriani-difabrizio/

Contents

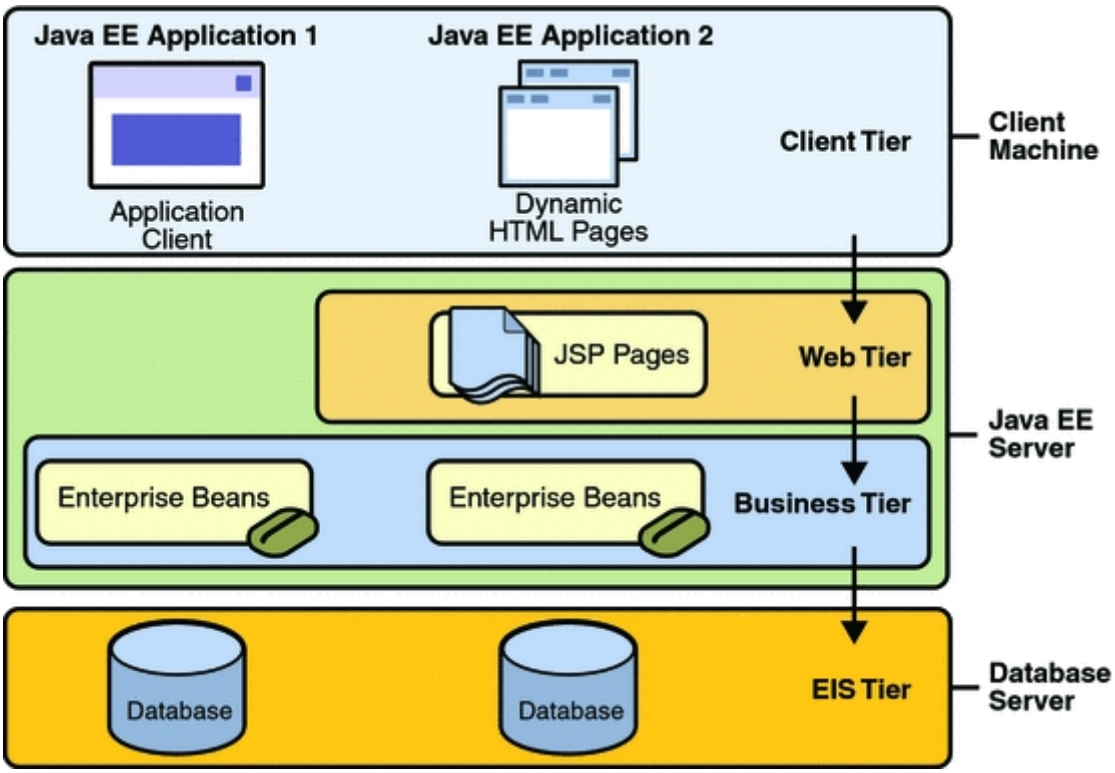
1 Architecture Description

The web-application is based on the Java Enterprise Edition architecture, which is deployed on the GlassFish Server 4.0. The application follows a four-tier architecture:

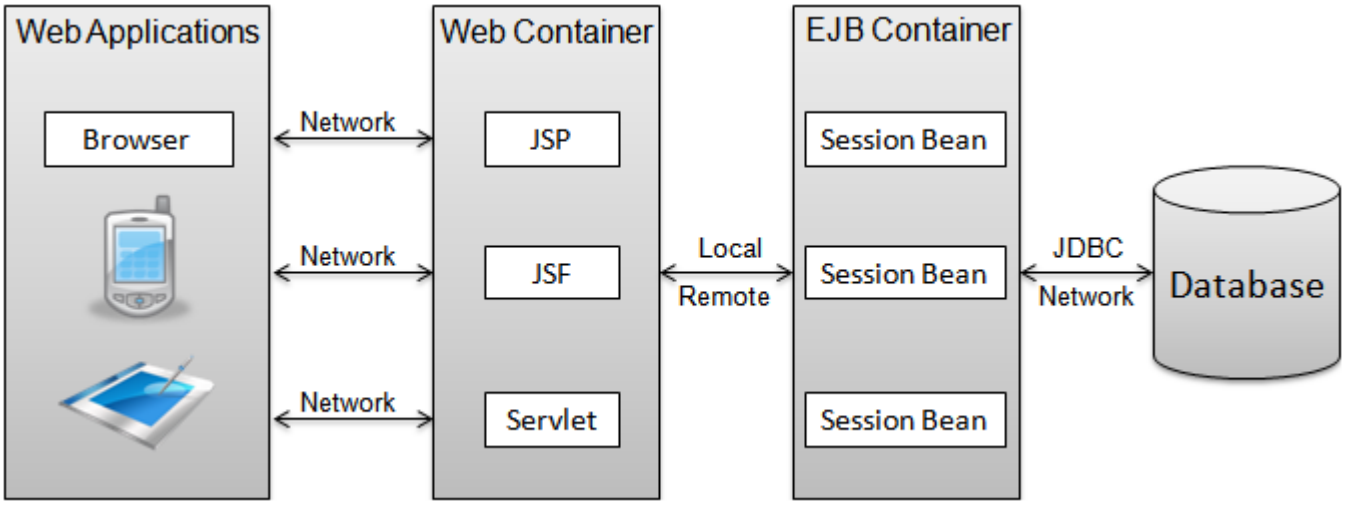
Client tier the user side is a thin client without logic (expect for field pattern validation)that will access the web server through a browser.

JavaEE Server deals with the design of a dynamic view through JSF() pages linked with support Backing Beans, moreover it contains the business logic (EJBs) and the model representation (Entities).The Database server is accessed through the JPA(Java Persistence API) .

EIS tier contains the MySQL server with the data source. Initially the database will reside with the web server on the same machine but the final aim is to configure the database on a physical independent machine.



More generally a Java web application can be seen as shown in the figure below which represent a typical three-tier architecture and its basic components.



2 Persistent Data Management

The persistent data design is explained with particular emphasis on the key implementation's choices.

2.1 Conceptual Design

We decided to have a content oriented approach, starting with the data design.
So reexamining the class diagram, we have identified all the classes of objects or facts involved in the application as entities. The principal subjects are Users, Events and Calendars. In this initial stage, Events and Calendars are generalizations of Private and Public Events and Calendar respectively. In order to express the dependency of the Calendar by the User entity, we have designed Calendar as a weak entity. Notifications are another entity in our application that needs to be stored into the database. Later we have studied all the relationships needed between entities as shown:

Owned the relation of ownership between User and Calendar, that forces a calendar to have exactly one owner

Recipient states who is the receiver of the Notification

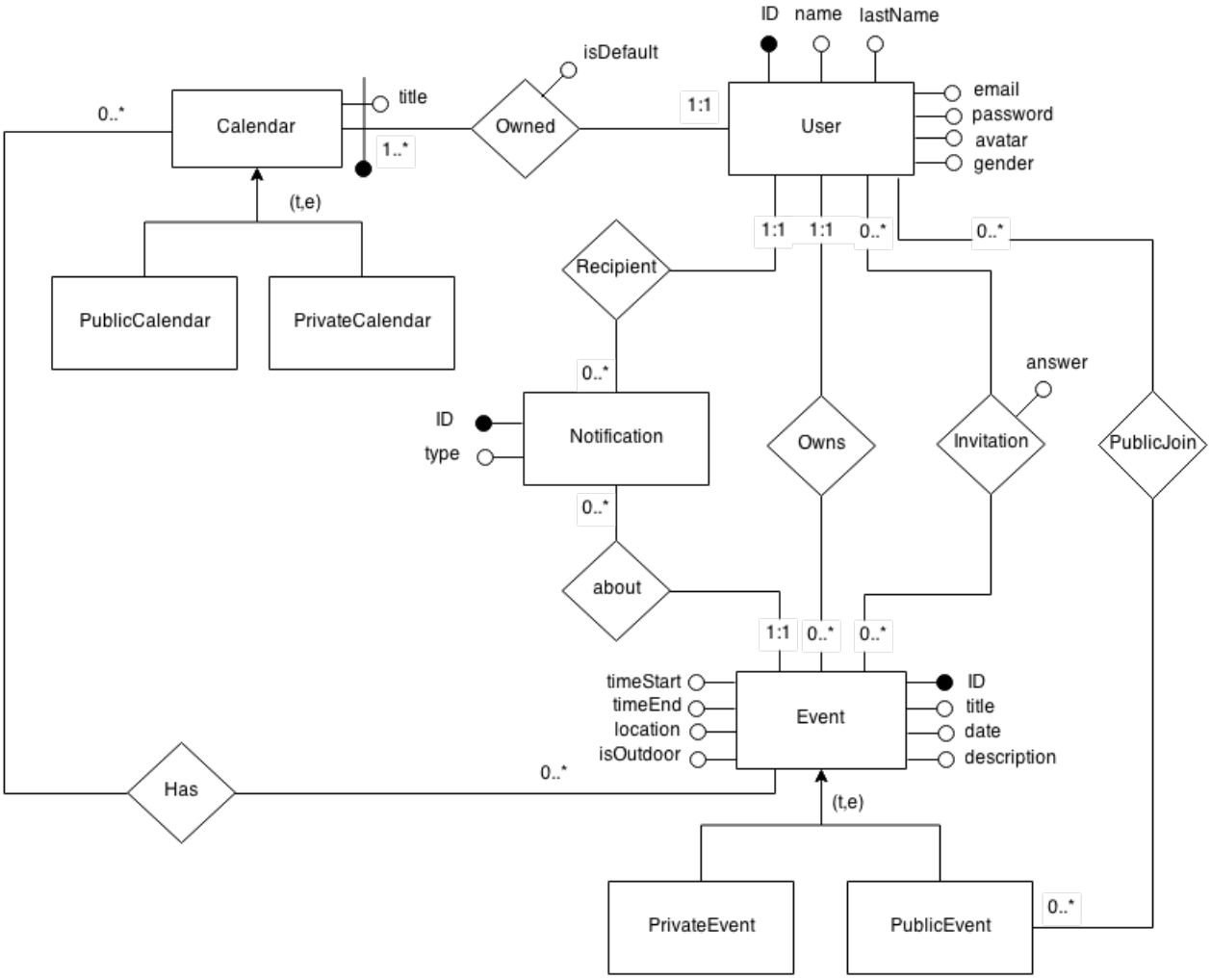
About states the event to which Notification is referred

Invitation means that the User is allowed to take part to the Event

PublicJoin means that the User wants to participate to a public Event without an invitation

Has means that an Event is scheduled in a Calendar

Owns the relation of ownership between User and Event, that forces an event to have exactly one owner


ID
name
lastName
isDefault
title
Calendar
Owned
User
email
password
avatar
gender
0..*
1..*
1:1
(t,e)
PublicCalendar
PrivateCalendar
Recipient
1:1
1:1
0..*
0..*
answer
ID
type
Notification
Owns
Invitation
PublicJoin
0..*
0..*
about
1:1
0..*
0..*
timeStart
timeEnd
location
isOutdoor
Event
ID
title
date
description
Has
0..*
(t,e)
PrivateEvent
PublicEvent
0..*

2.2 Logical Design

During translation into logical Design discussed about ambiguous situations of ER diagram and we chose to implement a single *Calendar* table added with attribute *isPublic* for define the privacy settings of a calendar. Moreover, we decided to split the *Event* table into *Public_Event* and *Private_Event*, because of the relationship involving *User* and *Public_Event* named *Public_Join*.

2.2.1 Table Overview

CALENDAR (calendar_title, FK:id_user, isPublic, isDefault)

PUBLIC_EVENT (id_event, title, date, timeStart, timeEnd, location*, description*, is_outdoor, FK: id_owner)

PRIVATE_EVENT (id_event, title, dateStart, dateEnd, timeStart, timeEnd, location*, description*, is_outdoor, FK: id_owner)

USER (id_user, name, surname, email, password, avatar*, gender*)

NOTIFICATION (id_notification, type, FK: id_user, FK: id_event)

INVITATION (answer *, FK: id_user, FK: id_event)

PUBLICJOIN (FK: id_user, FK: id_event)

EVENT_IN_CALENDAR (FK: id_event, FK: calendar_title, FK: id_user) [2]

Table Name	Attributes	Foreign Keys
Calendar	calendar_title, isPublic, isDefault,	id_user
Public_Event	id_event, title, dateStart, dateEnd, timeStart, timeEnd, location*, description*, is_outdoor	id_owner
Private_Event	id_event, title, dateStart, dateEnd, timeStart, timeEnd, location*, description*, is_outdoor	id_owner
User	id_user, name, surname, email, password, avatar*, gender*	
Notification	id_notification, type	id_user, id_event
Public_Join		id_user, id_event
Event_In_Calendar		id_event, calendar_title, id_user

[2] *Calendar_title* has been removed from the set of keys of *Event_In_Calendar* because an event can be only in a calendar per user as stated in the specifications.

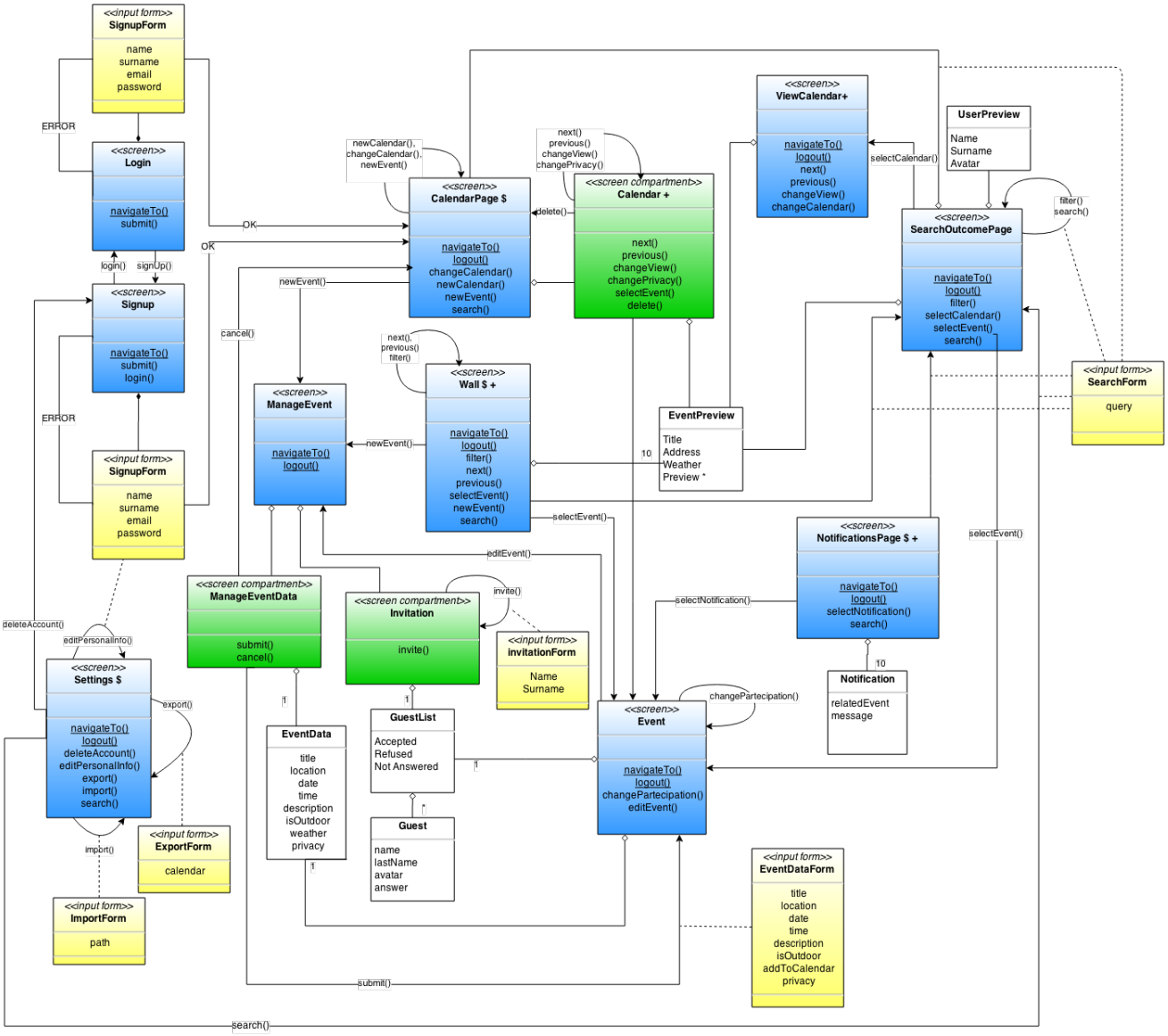
2.2.2 Constraints

- Type values will be defined into NotificationType_Domain
- Answer values will be defined into Answer_Domain
- Email in USER è unique
- DateStart <= dateEnd
- (dateStart−dateEnd) => timeStart<timeEnd
- Notifications are about events related to the user
- Every invitation generate a notification
- Every user owns exactly one calendar marked Default
- The owner of and Event doesn't have an invitation
- The owner cannot do the public join
- If a user joined a public event that user has not an invitation
- Each event that a user participate to, must be at most in one calendar
- If an event is in a calendar, the user must be participating at it

3 User Experience

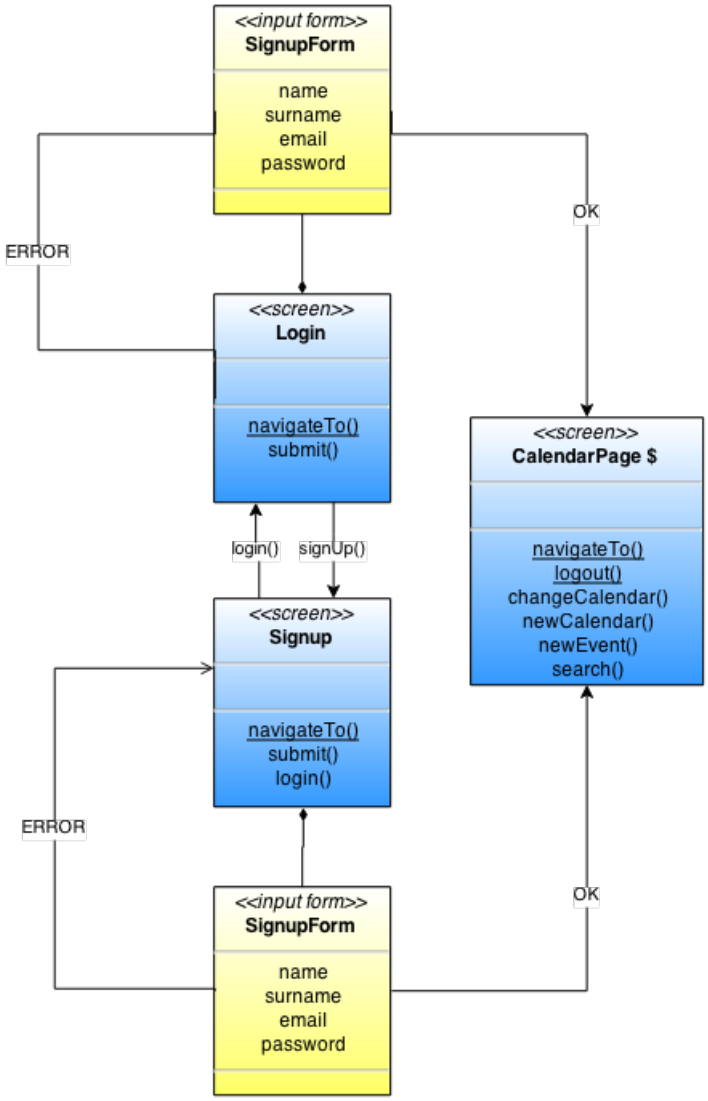
This section present the user experience(UX) diagram whose aim is to convey the general idea of the navigation between pages.
In the following section we first give a peek of our whole navigation model then each navigation sub-system is independently analyzed. The reader can compare the sub-systems with the general model at any time as a reference.

3.1 The whole picture



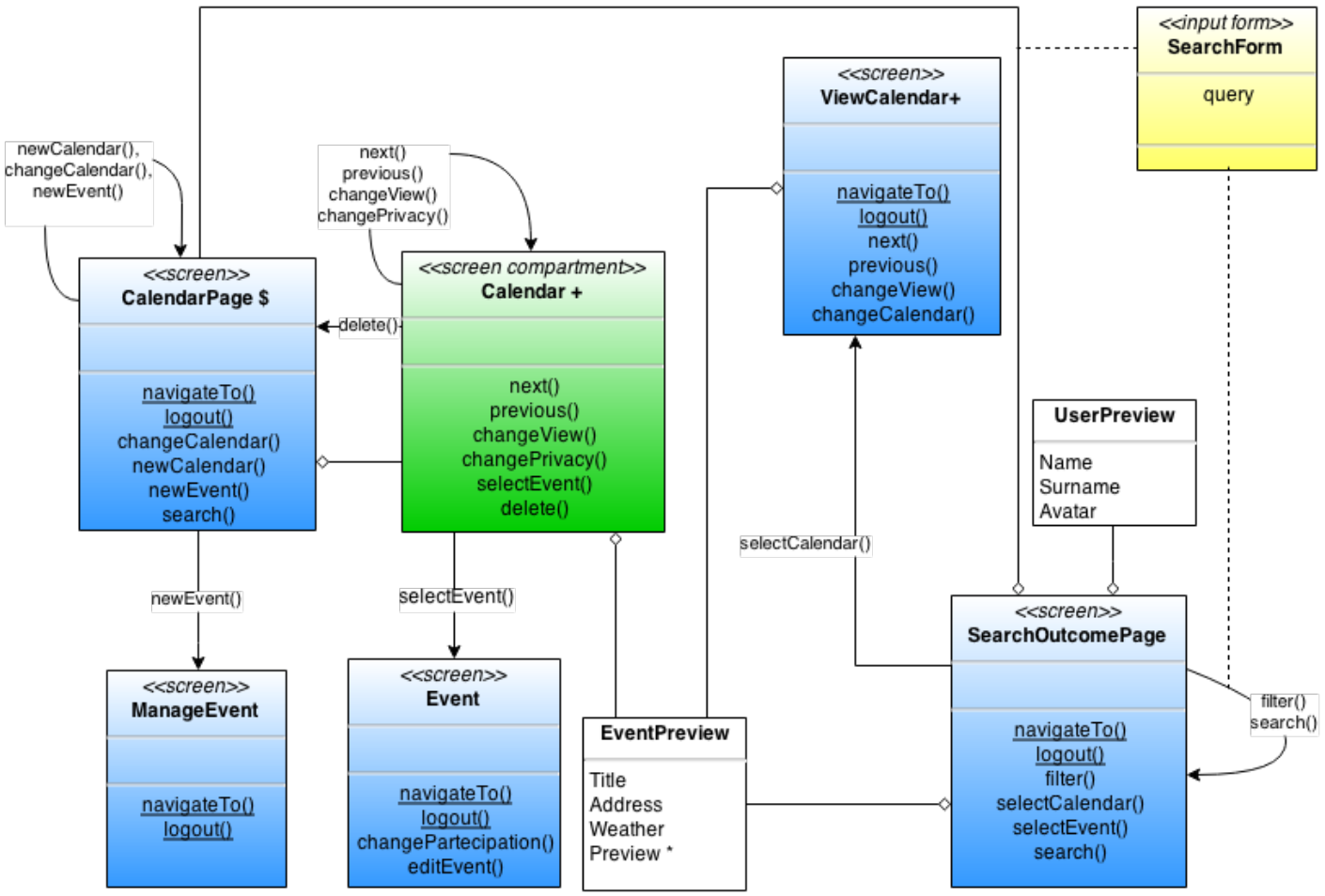
3.2 Login

Any user that browse to our site is redirected to the signUp page, there he can submit the form to signUp or if he already has the credential he can click on LogIn to be forwarded to the login page.



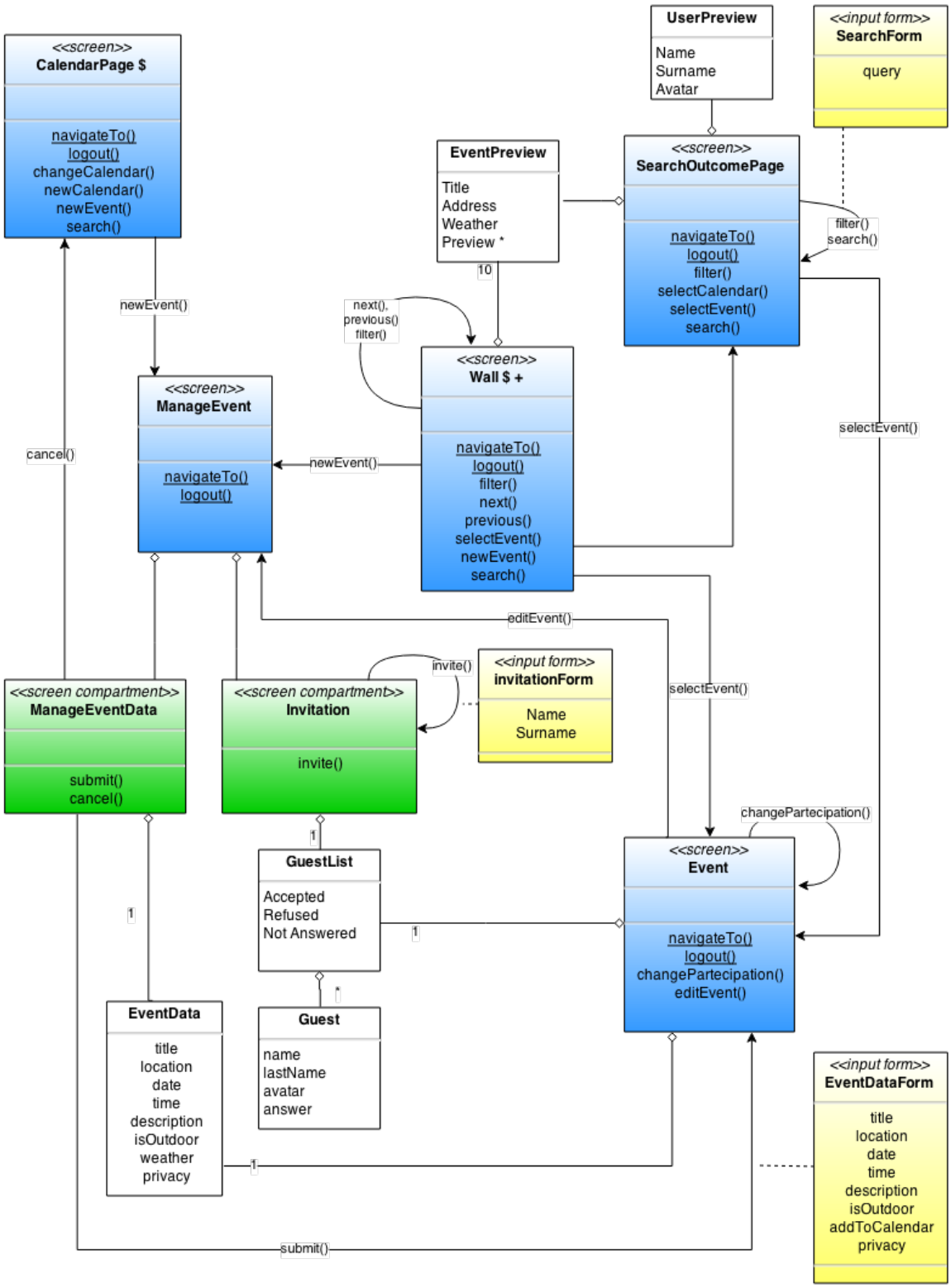
3.3 Calendar

The calendar navigation is quite straight-forward: the calendar page of the user is composed by one or more calendar whose view can be customized (weekly,monthly) as well as the calendar's preferences.
The user can then select an event on his calendar to reach the event Page, create a new Event or a new calendar. From this page as well as the other main pages of the system the user can submit a general search to find a particular user or an event and the results are displayed in the SearchOutcomePage where they can be filtered.



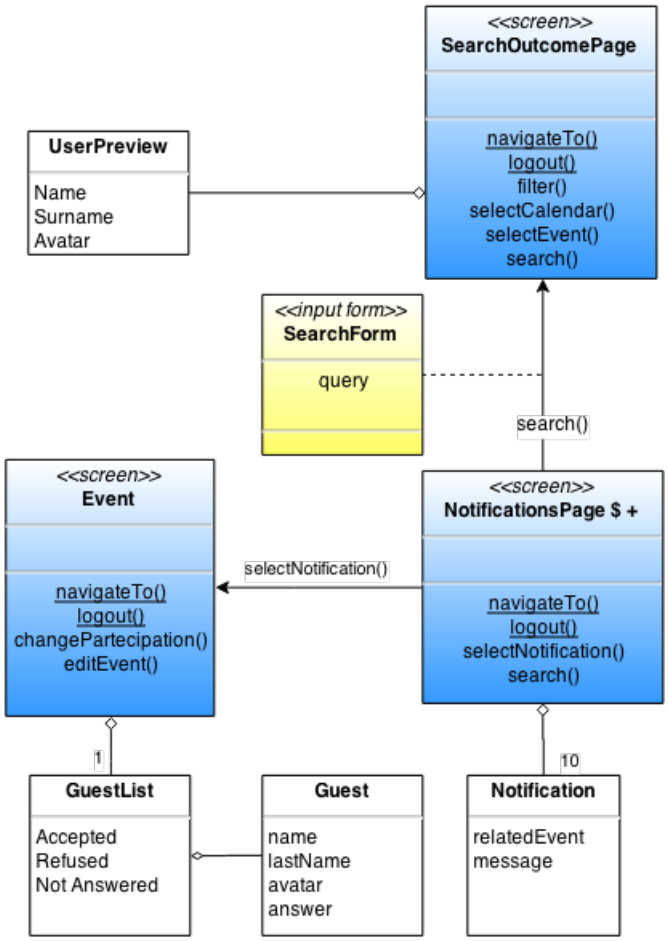
3.4 Event

The wall is composed by events. The user can select any event and he is forwarded to the EventPage, once there he can change his participation to the event and only the owner can edit it through the Manage Event page, which is the same page used for the creation of an event. Of course any time an event is created or edited a form must filled up.



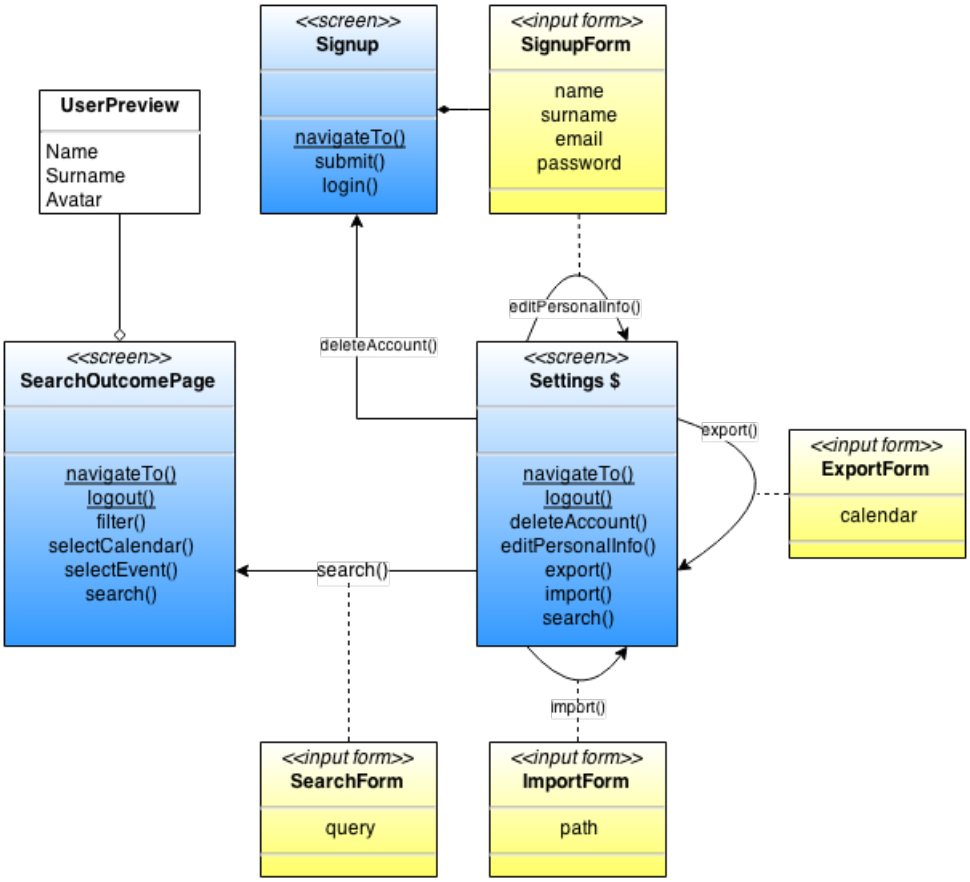
3.5 Notification

The notification page can be accessed at any time from any page, once the user is forwarded to that page he can select a particular notification and he will be forwarded to the event page that generated the notification. Also from this page it is possible to start a search.



3.6 Settings

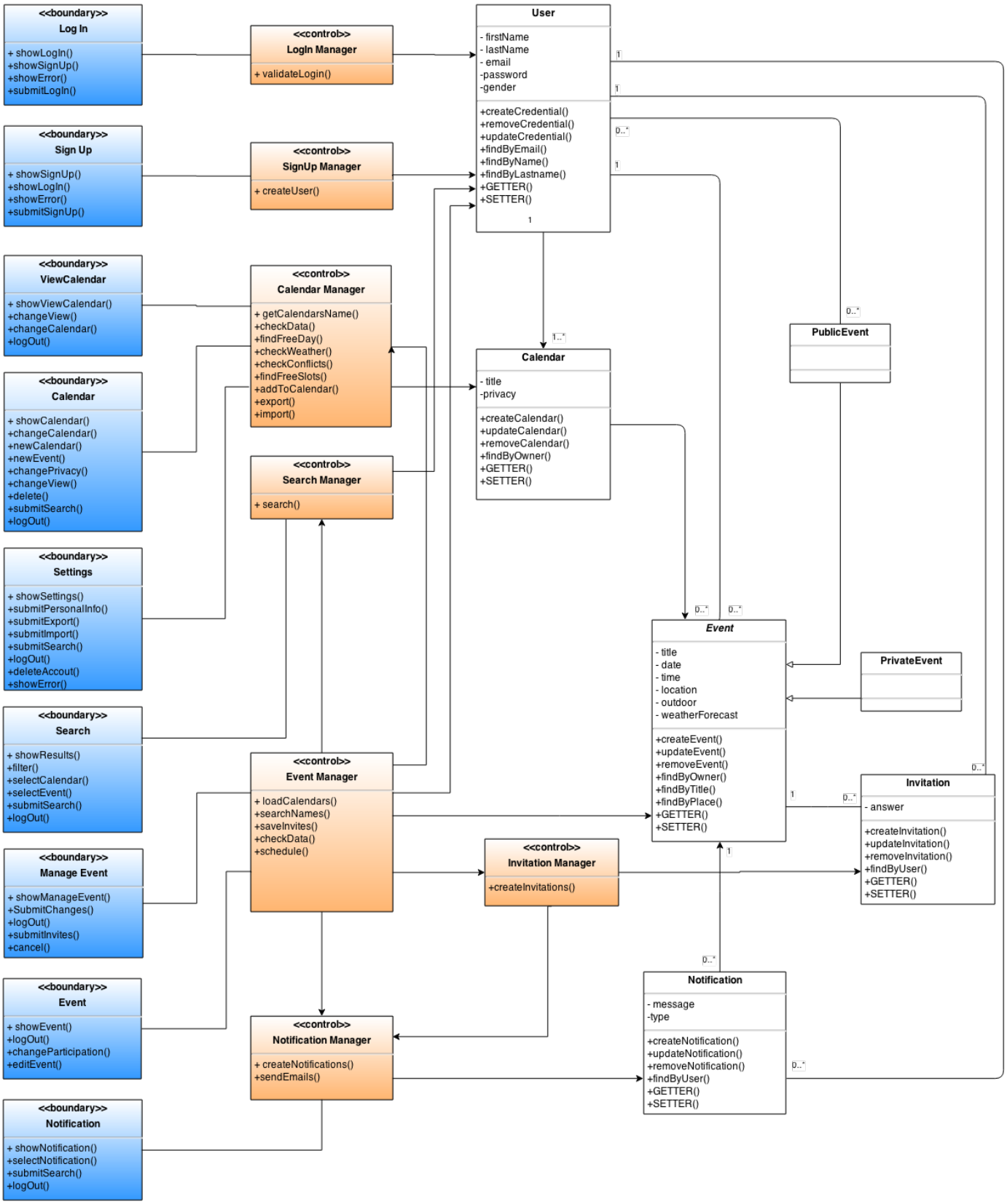
From the setting page the user can edit all his personal info, export a particular calendar or import an old one. If the user decide to delete his account he will be forwarded to the sign-up page.



4 BCE Diagrams

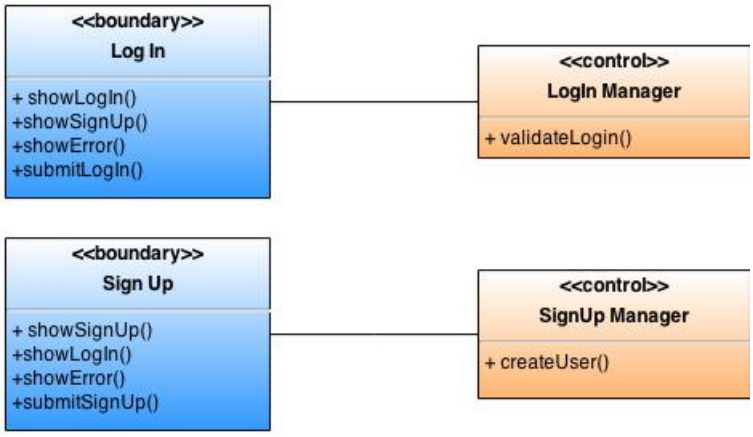
The BCE diagram to show a simplified yet comprehensive look of the model-view-component structure that we are going to implement.
Also in this case the reader can have a quick look at the whole model and use it as a reference to better understand the following focuses on each component.

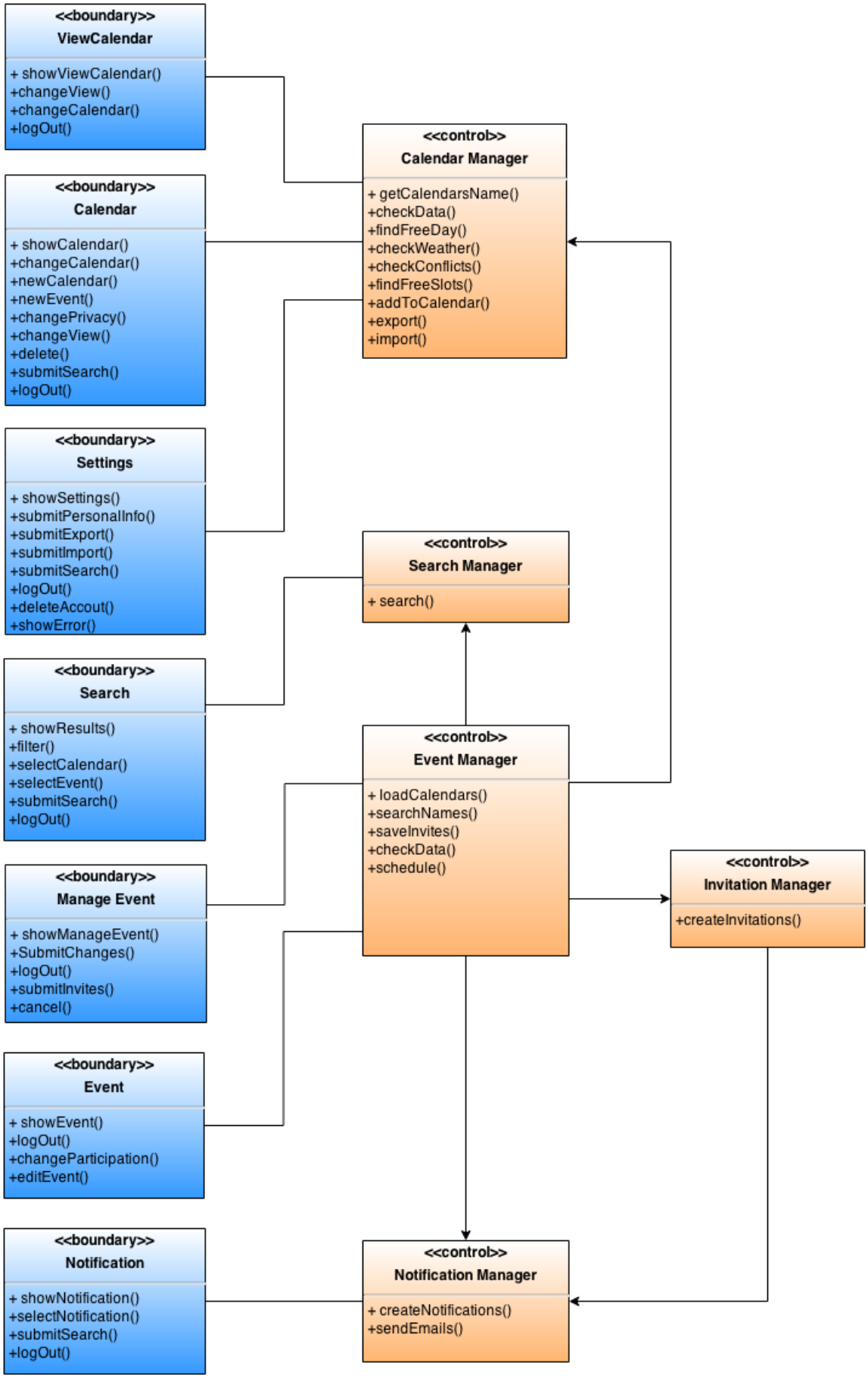
4.1 The whole picture



4.2 Boundary and Control Focus

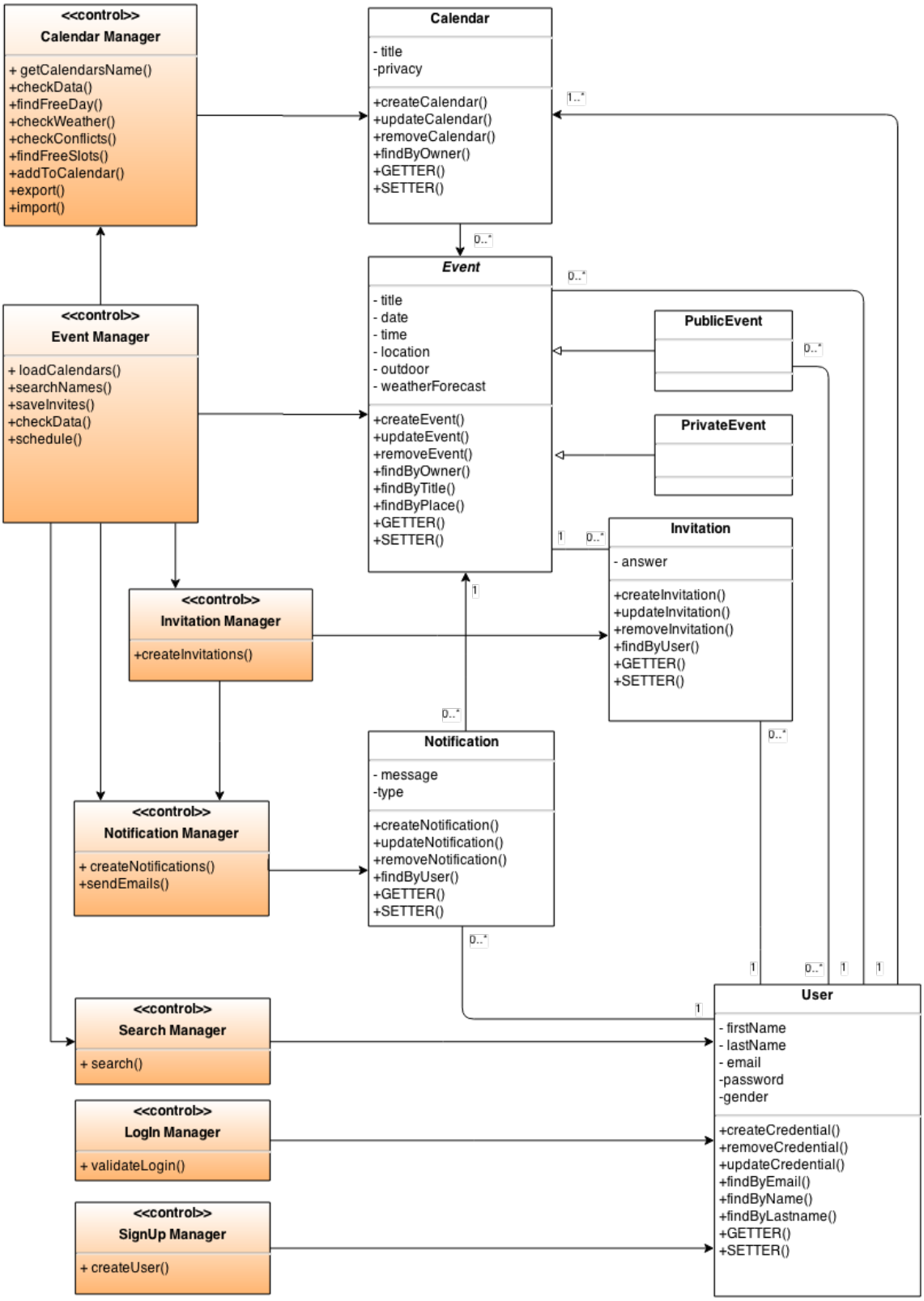
The structure we decided to adopt is very decentralized, with this we mean that basically any Boundary has his own Control. The aim of this approach is to divide the business logic in his fundamental parts and then build the whole system by combining them.
In this way we will a more extendable behavior of the system as a whole and a deeper control of each phase infact each boundary "talks" to his controller which is then in charge of calling the other controllers to achieve the goal.
Moreover it will result easier for the team to divide the interest areas during the coding phase.




<<boundary>>
ViewCalendar
+ showViewCalendar()
+changeView()
+changeCalendar()
+logOut()
<<boundary>>
Calendar
+ showCalendar()
+changeCalendar()
+newCalendar()
+newEvent()
+changePrivacy()
+changeView()
+delete()
+submitSearch()
+logOut()
<<boundary>>
Settings
+ showSettings()
+submitPersonalInfo()
+submitExport()
+submitImport()
+submitSearch()
+logOut()
+deleteAccout()
+showError()
<<boundary>>
Search
+ showResults()
+filter()
+selectCalendar()
+selectEvent()
+submitSearch()
+logOut()
<<boundary>>
Manage Event
+ showManageEvent()
+SubmitChanges()
+logOut()
+submitInvites()
+cancel()
<<boundary>>
Event
+ showEvent()
+logOut()
+changeParticipation()
+editEvent()
<<boundary>>
Notification
+ showNotification()
+selectNotification()
+submitSearch()
+logOut()
<<control>>
Calendar Manager
+ getCalendarsName()
+checkData()
+findFreeDay()
+checkWeather()
+checkConflicts()
+findFreeSlots()
+addToCalendar()
+export()
+import()
<<control>>
Search Manager
+ search()
<<control>>
Event Manager
+ loadCalendars()
+searchNames()
+saveInvites()
+checkData()
+schedule()
<<control>>
Invitation Manager
+createInvitations()
<<control>>
Notification Manager
+ createNotifications()
+sendEmails()

4.3 Control and Entity Focus

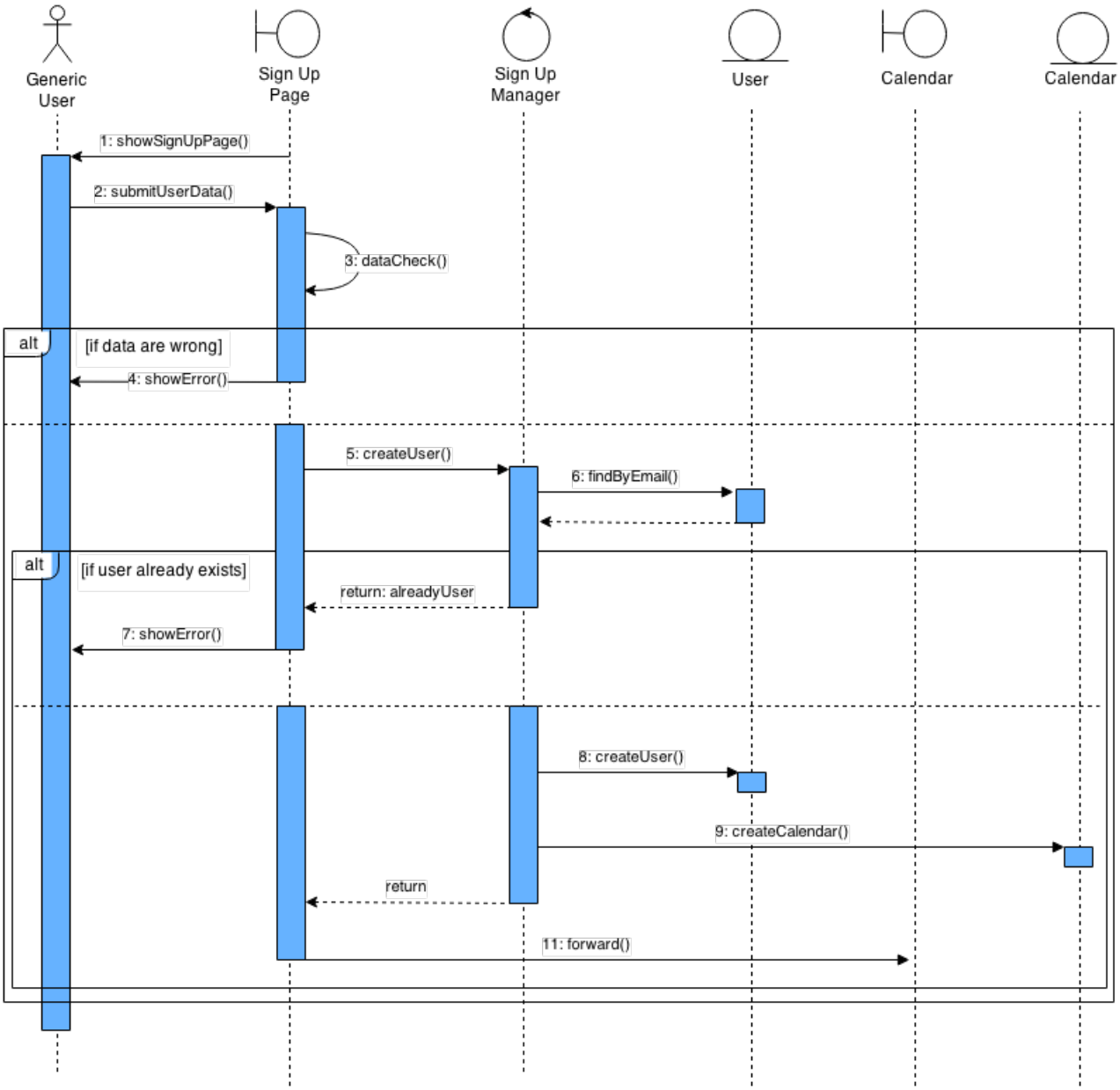
The controller are the means to connect the boundaries to the database information stored in the entities.Again the main idea is to build a loosely coupled structure to reach the maximum flexibility possible.



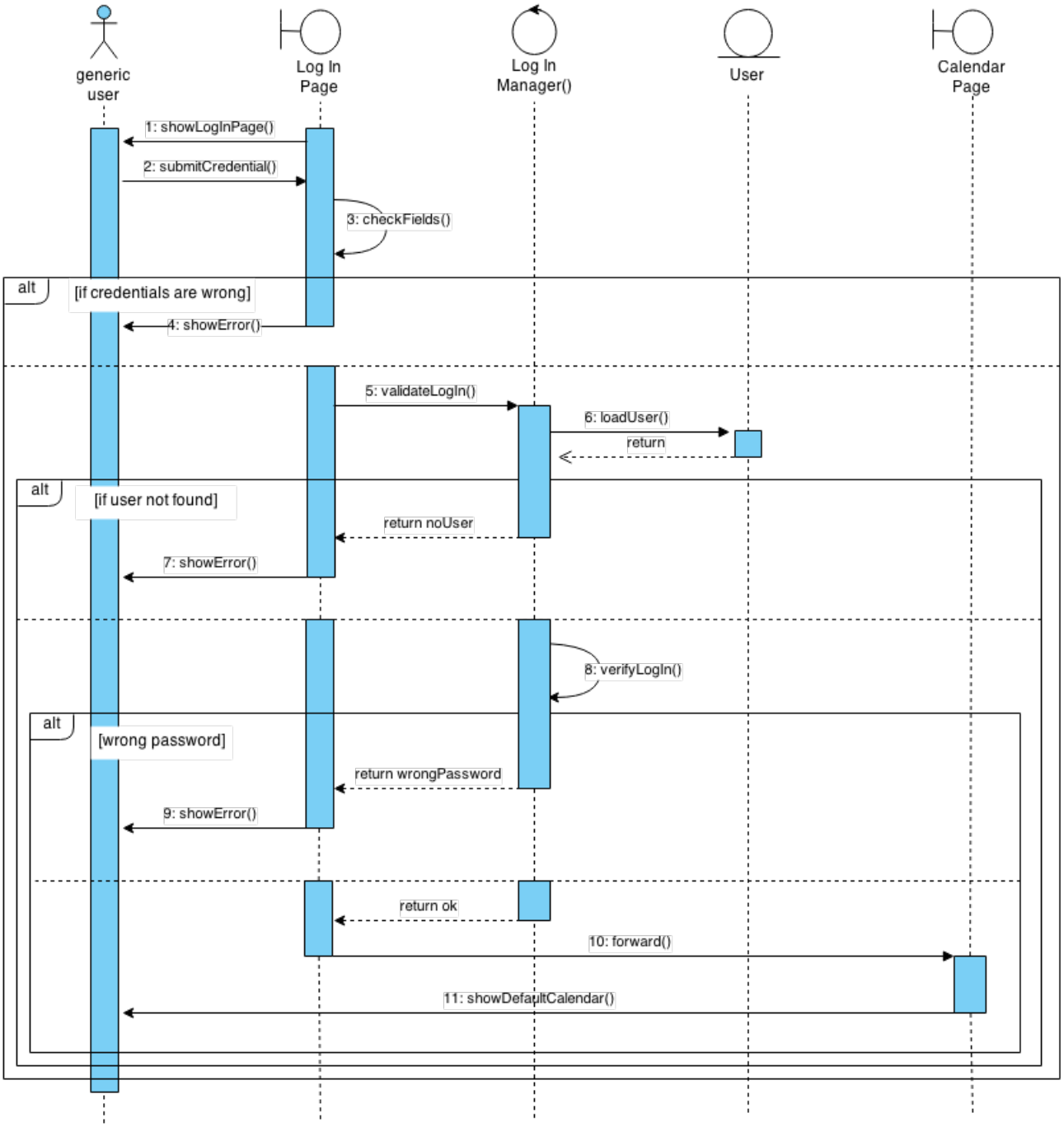
5 Sequence Diagrams

Here all the specification gathered during the previous phase (RASD) and the structure shown in the present document are combined to present a plain representation of the application flows, sketching some detailed sequence diagrams.

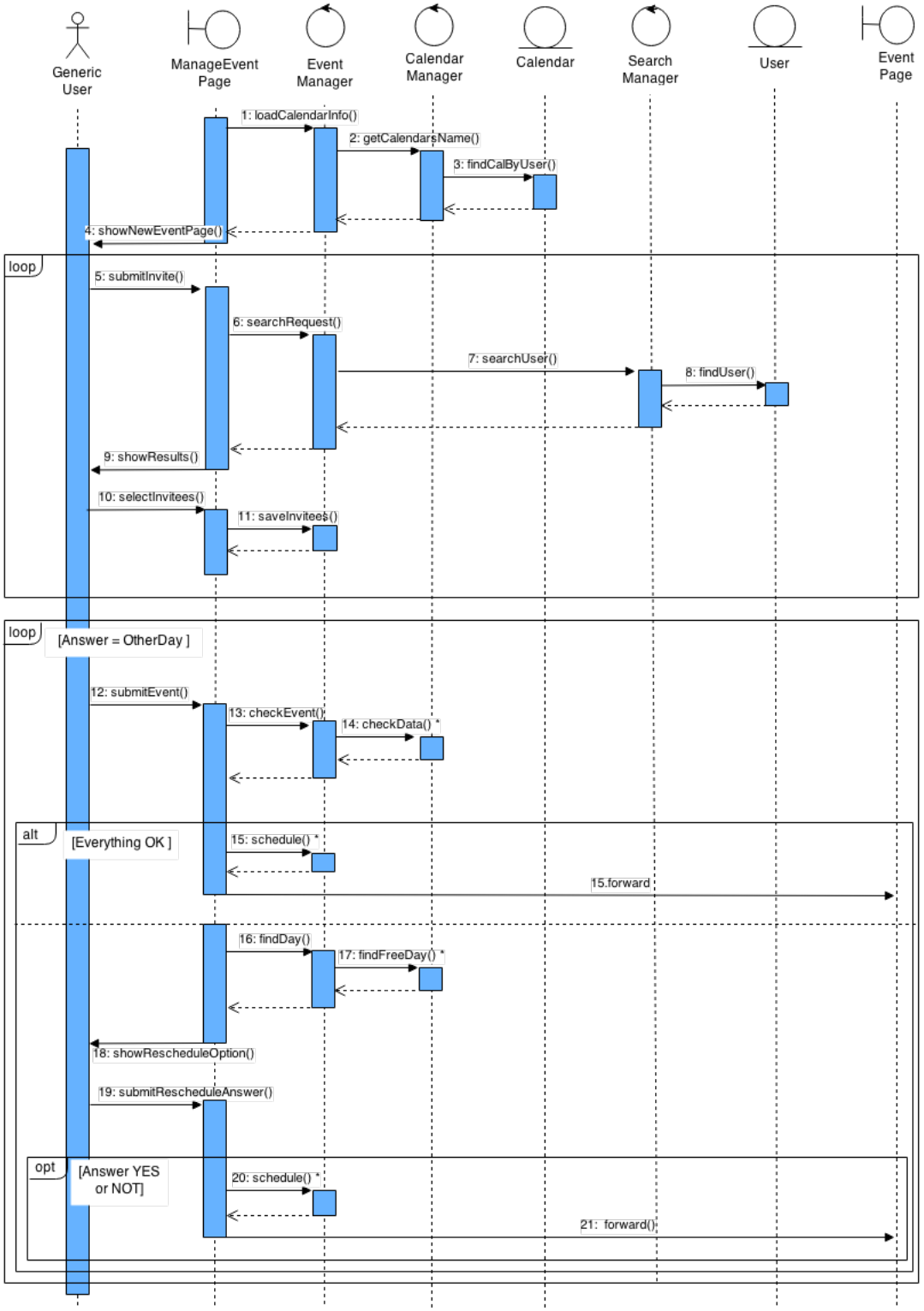
5.1 Sign Up



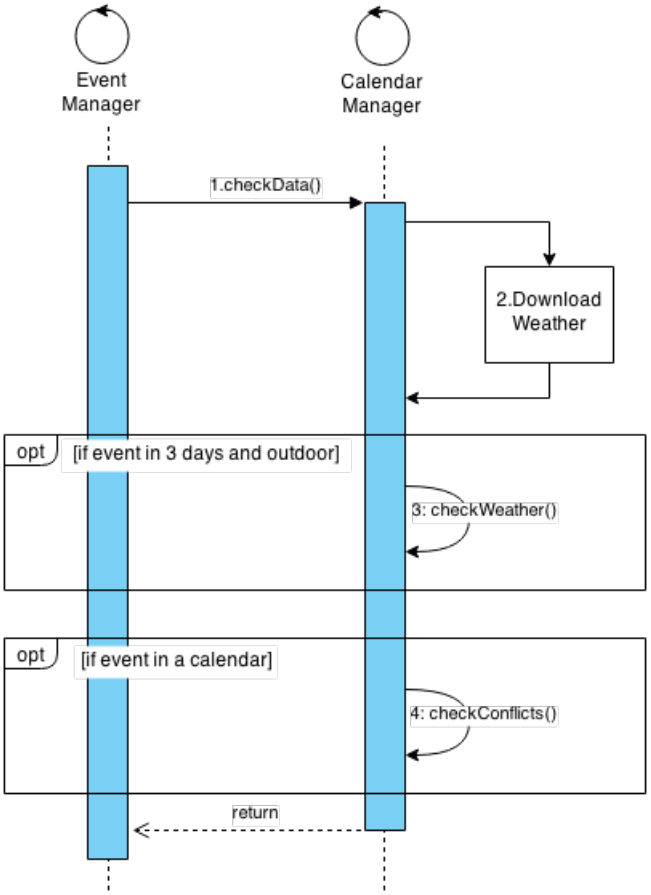
5.2 Log In


generic user
Log In Page
Log In Manager()
User
Calendar Page
1: showLogInPage()
2: submitCredential()
3: checkFields()
alt
[if credentials are wrong]
4: showError()
5: validateLogIn()
6: loadUser()
return
alt
[if user not found]
return noUser
7: showError()
8: verifyLogIn()
alt
[wrong password]
return wrongPassword
9: showError()
return ok
10: forward()
11: showDefaultCalendar()

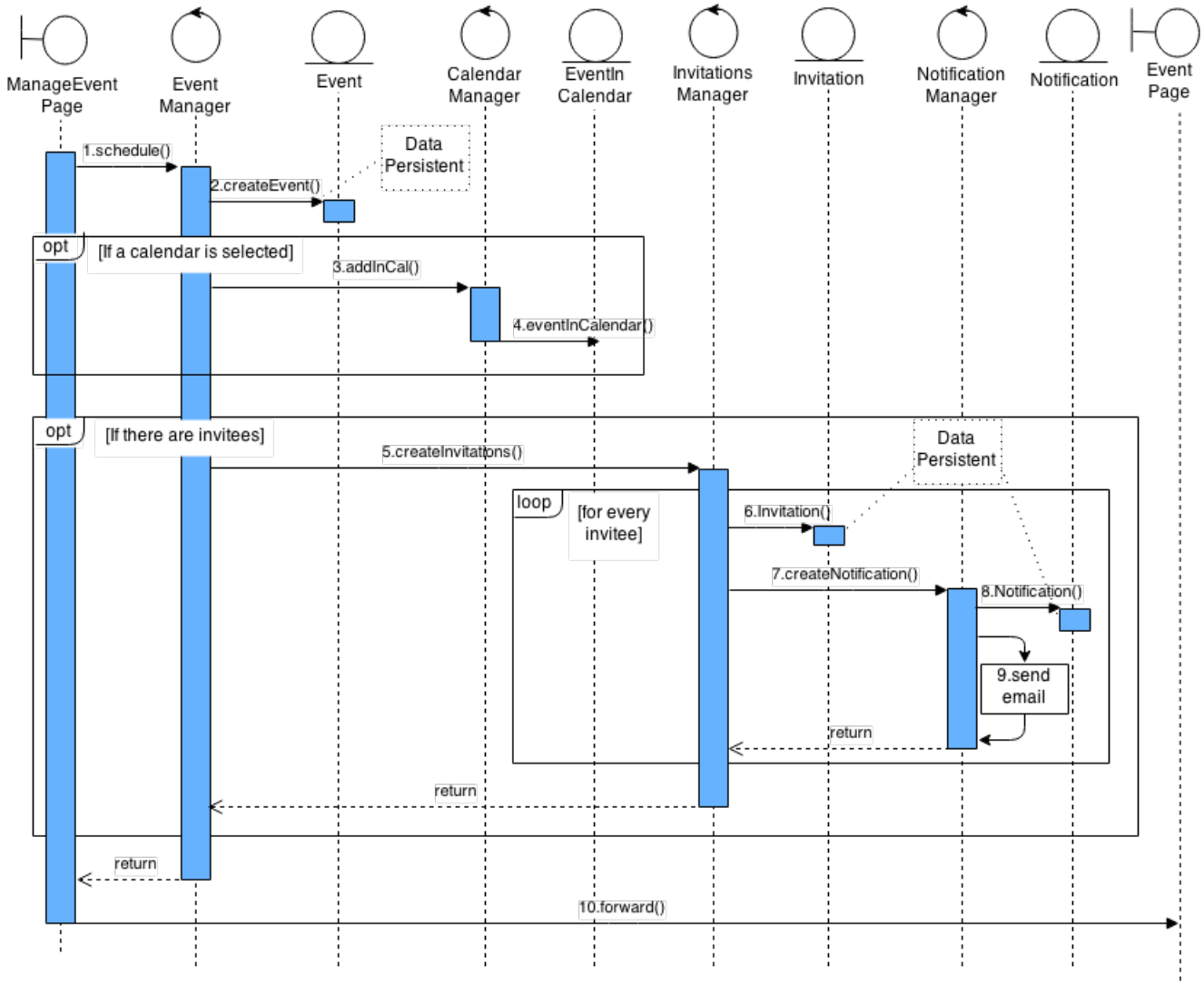
5.3 New Event


Generic User
ManageEvent Page
Event Manager
Calendar Manager
Calendar
Search Manager
User
Event Page
1: loadCalendarInfo()
2: getCalendarsName()
3: findCalByUser()
4: showNewEventPage()
loop
5: submitInvite()
6: searchRequest()
7: searchUser()
8: findUser()
9: showResults()
10: selectInvitees()
11: saveInvitees()
loop
[Answer = OtherDay]
12: submitEvent()
13: checkEvent()
14: checkData() *
alt
[Everything OK]
15: schedule() *
15.forward
16: findDay()
17: findFreeDay() *
18: showRescheduleOption()
19: submitRescheduleAnswer()
opt
[Answer YES or NOT]
20: schedule() *
21: forward()

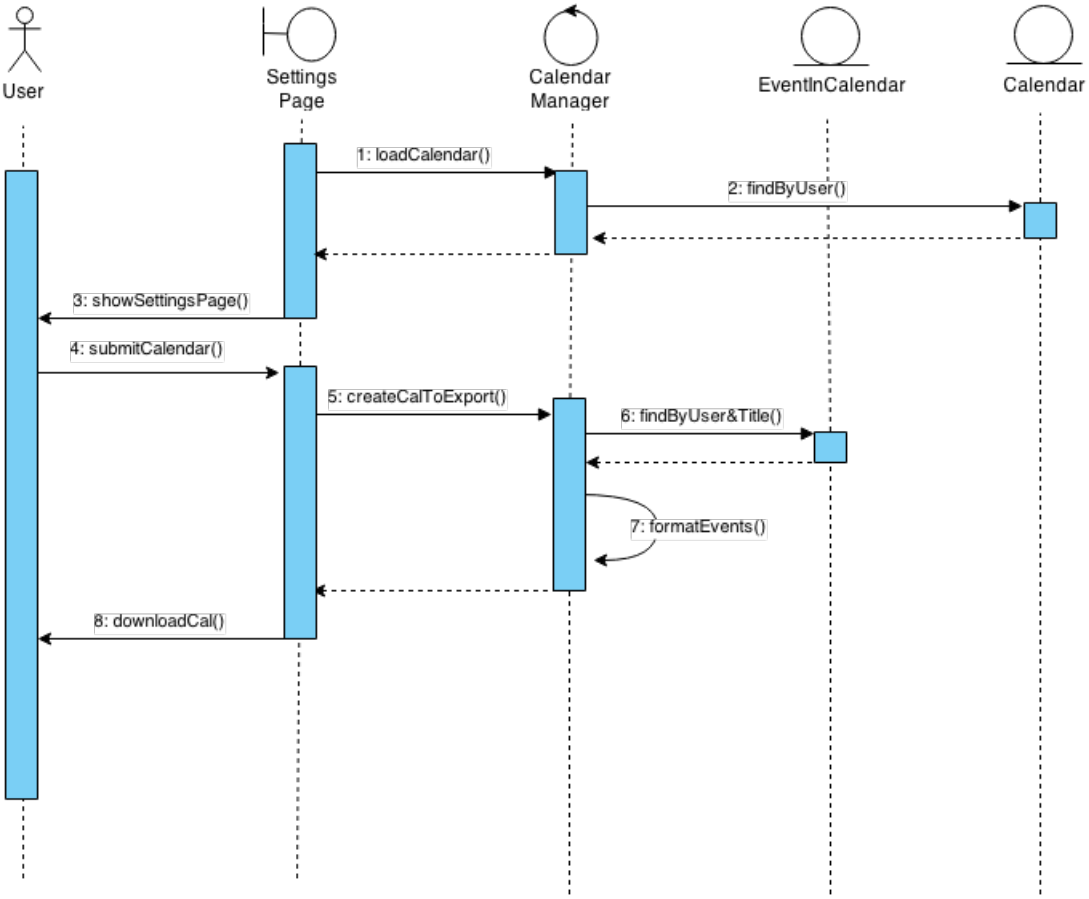
5.4 Check Date



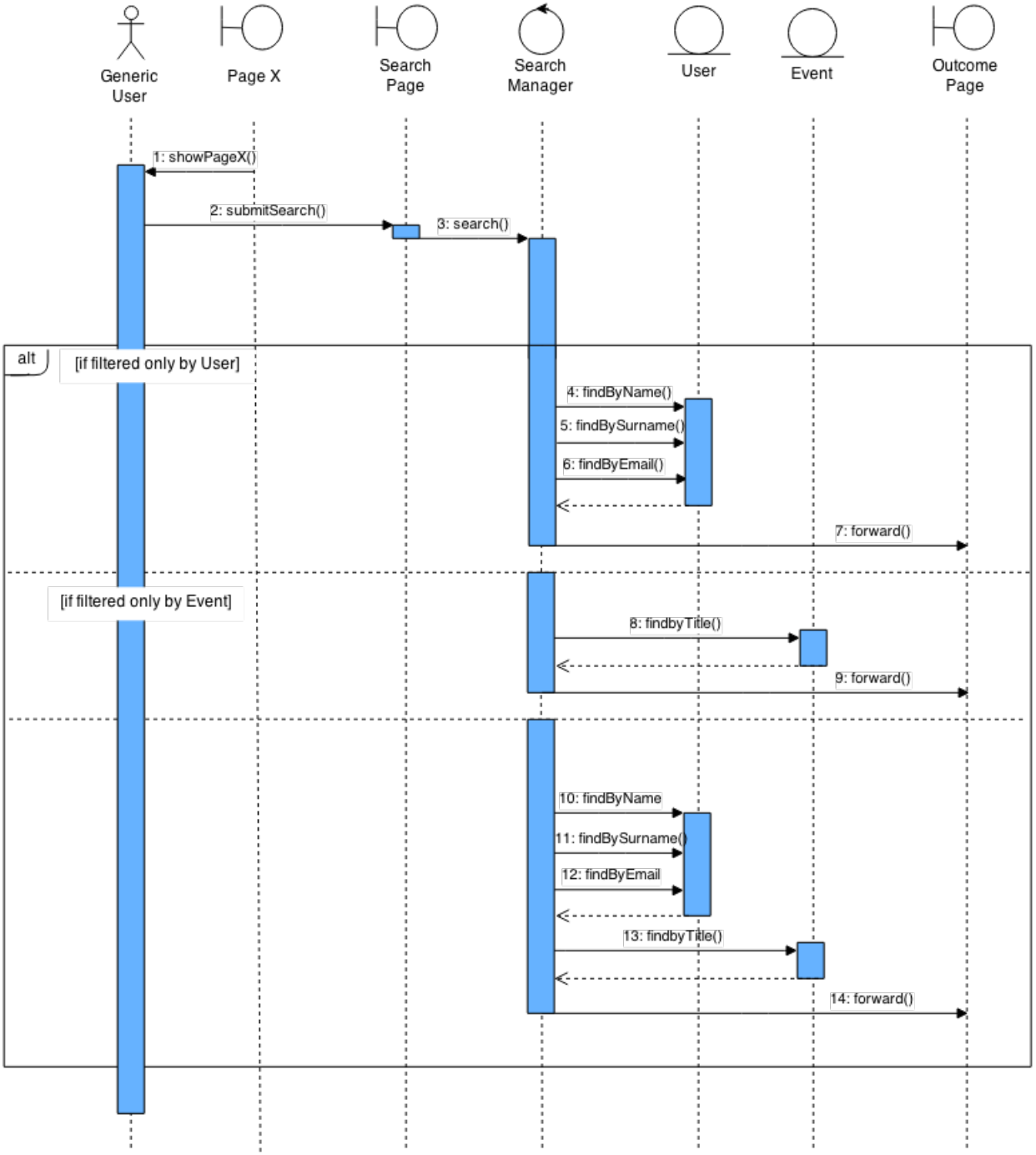
5.5 Schedule


ManageEvent Page
Event Manager
Event
Calendar Manager
EventIn Calendar
Invitations Manager
Invitation
Notification Manager
Notification
Event Page
1.schedule()
2.createEvent()
Data Persistent
opt
[If a calendar is selected]
3.addInCal()
4.eventInCalendar()
opt
[If there are invitees]
5.createInvitations()
Data Persistent
loop
[for every invitee]
6.Invitation()
7.createNotification()
8.Notification()
9.send email
return
return
return
10.forward()

5.6 Export


User
Settings Page
Calendar Manager
EventInCalendar
Calendar
1: loadCalendar()
2: findByUser()
3: showSettingsPage()
4: submitCalendar()
5: createCalToExport()
6: findByUser&Title()
7: formatEvents()
8: downloadCal()

5.7 Search


Generic User
Page X
Search Page
Search Manager
User
Event
Outcome Page
1: showPageX()
2: submitSearch()
3: search()
alt
[if filtered only by User]
4: findByName()
5: findBySurname()
6: findByEmail()
7: forward()
[if filtered only by Event]
8: findbyTitle()
9: forward()
10: findByName
11: findBySurname()
12: findByEmail
13: findbyTitle()
14: forward()

6 Tools

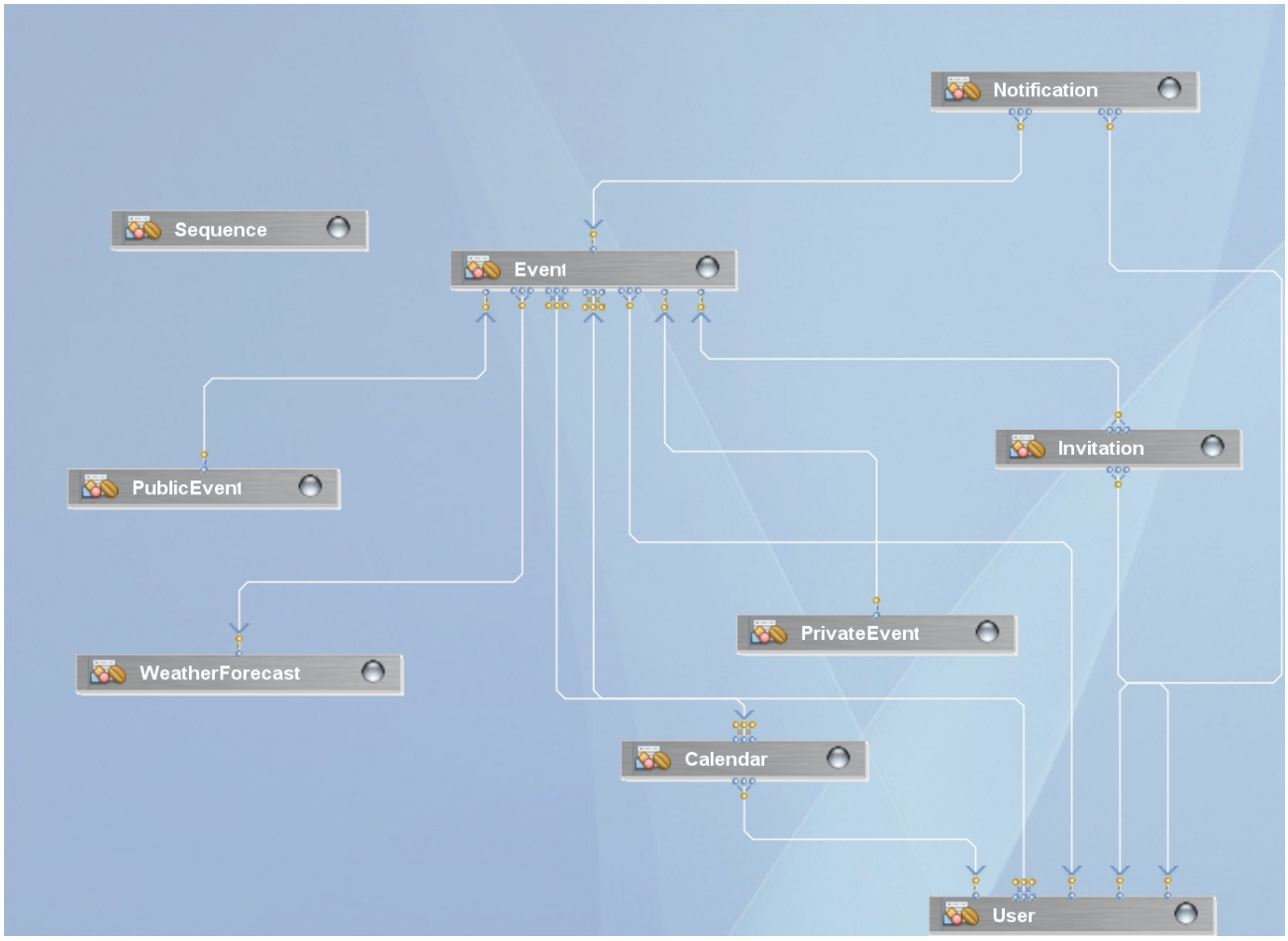
- Draw.io: Graphs editing
- TexMaker Latex editor: Edit this pdf file

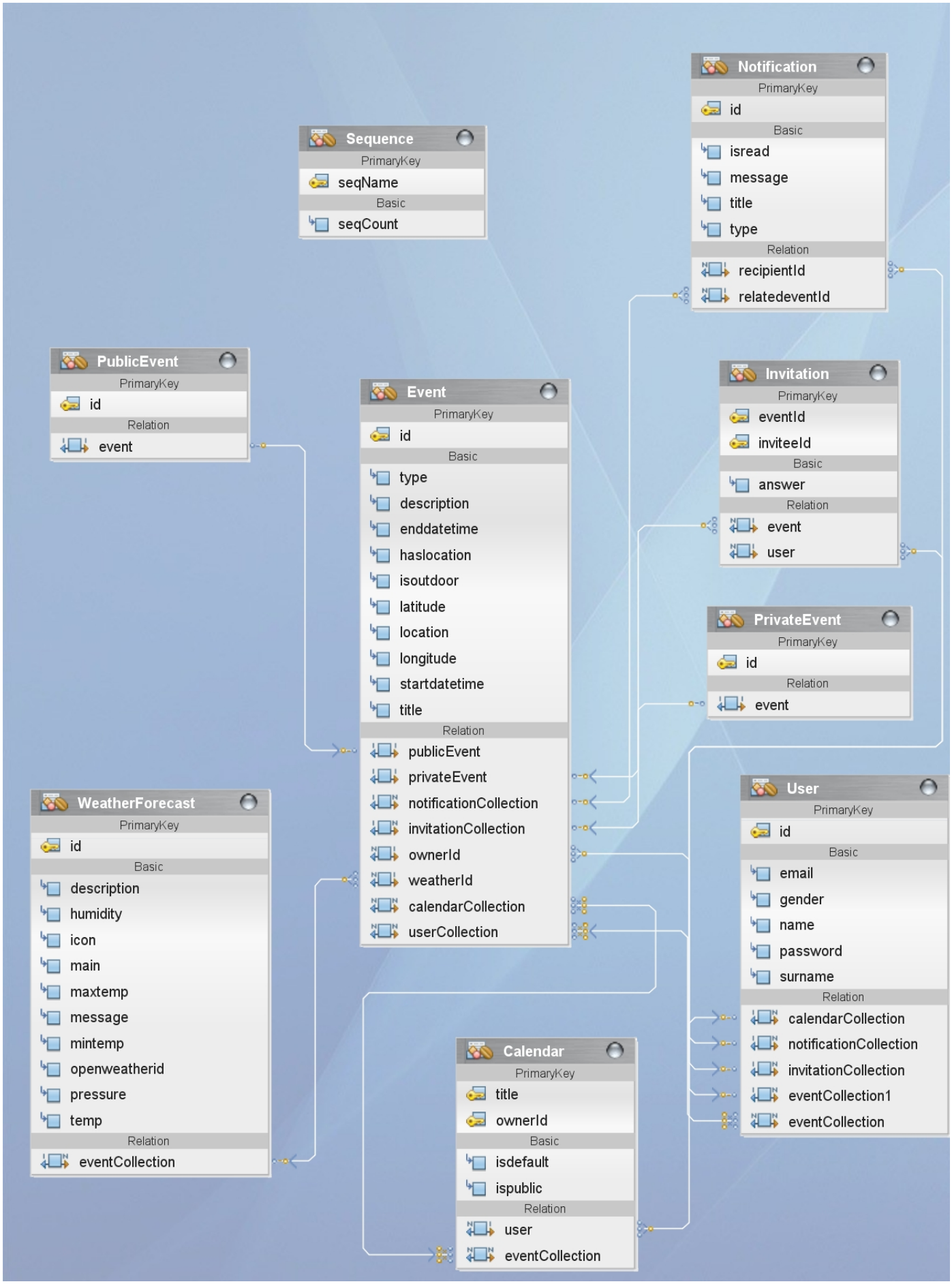
7 Working Hours

Group Member	Total Hours
Francesco Angelo	25
Valentina Ceriani	25
Umberto Di Fabrizio	26

8 Final notes

The only review about the Design Document is about the ER diagram that we did improve during the development phase. Here we report the auto generated diagram from our database.




Notification
PrimaryKey
id
Basic
isread
message
title
type
Relation
recipientId
relatedeventId
Sequence
PrimaryKey
seqName
Basic
seqCount
PublicEvent
PrimaryKey
id
Relation
event
Event
PrimaryKey
id
Basic
type
description
enddatetime
haslocation
isoutdoor
latitude
location
longitude
startdatetime
title
Relation
publicEvent
privateEvent
notificationCollection
invitationCollection
ownerId
weatherId
calendarCollection
userCollection
Invitation
PrimaryKey
eventId
inviteeId
Basic
answer
Relation
event
user
PrivateEvent
PrimaryKey
id
Relation
event
WeatherForecast
PrimaryKey
id
Basic
description
humidity
icon
main
maxtemp
message
mintemp
openweatherid
pressure
temp
Relation
eventCollection
User
PrimaryKey
id
Basic
email
gender
name
password
surname
Relation
calendarCollection
notificationCollection
invitationCollection
eventCollection1
eventCollection
Calendar
PrimaryKey
title
ownerId
Basic
isdefault
ispublic
Relation
user
eventCollection