Politecnico Di Milano

Engineering of computing systems

Software Engineering II

A.A. 2014/2015

MeteoCal

Design Document

Professor:

Raffaela Mirandola

Authors:

Alessio Rossotti 823213
Matteo Pasina 837816
Simone Rubiu 835358

December 7, 2014

Contents

1 Introduction

With this document, starting from RASD specifications, we want to describe our architectural and design choices about the system.
Initially we present the entity-relationship model that describes data conceptual design and from this we've obtained the logic model. Next we consider the User Experience and we propose some UX diagrams explaining how the user should navigate in the site and a BCE diagram to show in detail how is divided, following the Model View Controller standard, all the system.

2 Data Design

2.1 Entity-Relationship model

This is the Entity Relationship model that we design from the RASD document. We found two main entities in our system: **User** and **Event**

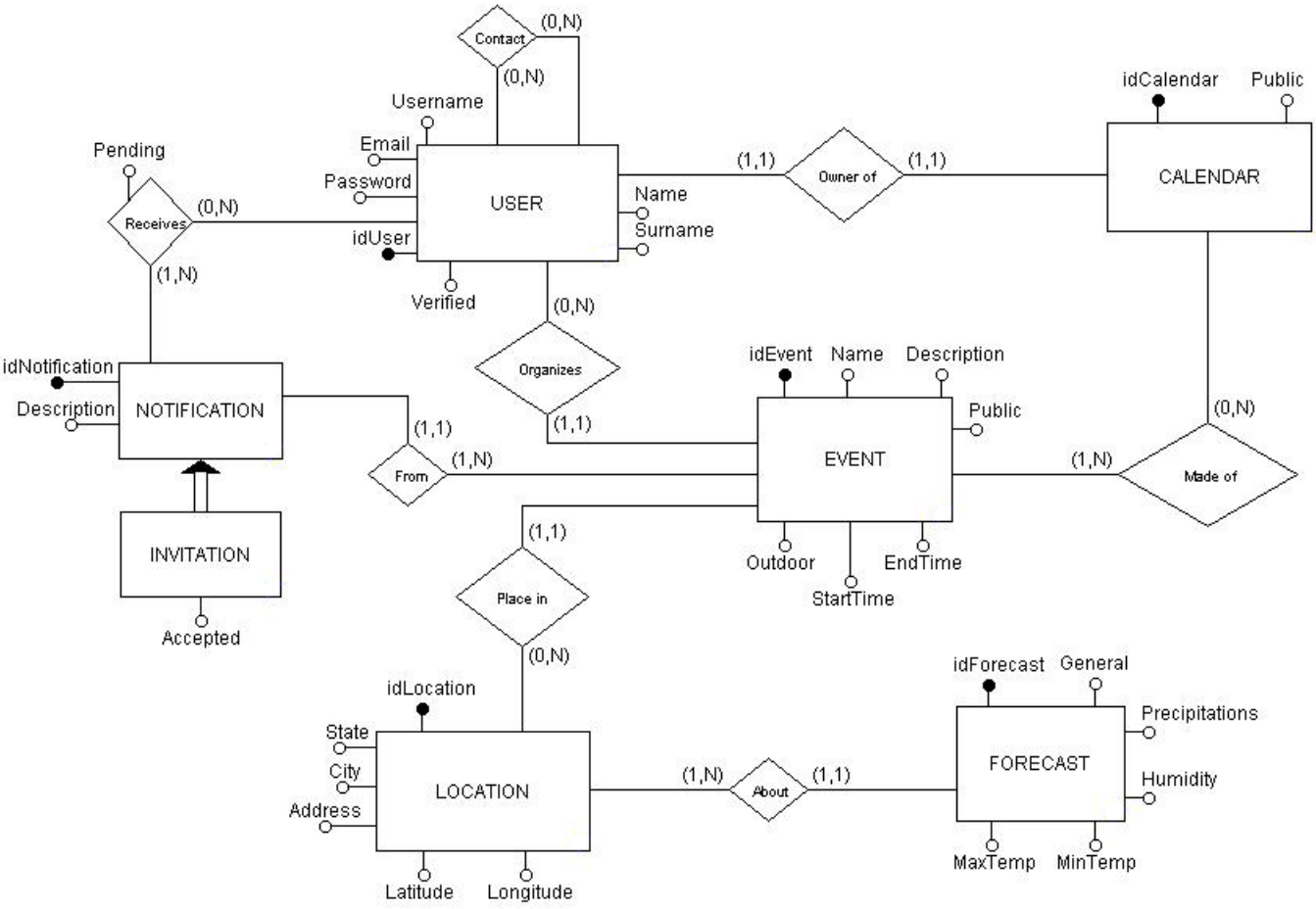

Figure 2.1: Entity-Relationship Model

2.1.1 Entities

- **User:** contains all the registered user. The table is characterized by a primary key, idUser, all the information about the profile like Email, Name, Surname, Username, Password and a boolean, Verified, used to know if a user has verified is email by clicking on a the confirmation link send through the given email.
- **Event:** contains all the information about the event create in MeteoCal. The entity is characterized by a primary key, idEvent, the information about the event like Name, Description, StartTime, EndTime and two booleans Public and Outdoor to specify, respectively, if the event is visible to every one and if the event take place outdoor or indoor.
- **Location:** contains all the place where an event can take place. The table is characterized by a primary key, idLocation, and all information to decribe a location like State, City, Address, Latitude and Longitude.
- **Calendar:** contains all information about the calendar of each user in the system. the entity is characterized by a primary key, idCalendar, and a boolean Public that specify if the calendar is visible to everyone or only to the owner of the calendar.
- **Notification:** contains all the notifications of the system. Each notification is characterized by a primary key, idNotification and a brief Description of the notification.
- **Invitation:** is a sub-entity of notification. The entity got all the attribute of the parent and add a flag, Accepted, that specify if the user accepted or not the invitation.
- **Forecast:** contains the information about the weather. the table is characterized by a primary key, idForecast, a brief description about the forecast, General, and all the information usefull to describe the weather conditions like Precipitations, Humidity, MaxTemp and MinTemp.

2.1.2 Relations

- **Contact:** (User $\Longrightarrow$ User) [0:N]
- **Organizes:** (User $\Longrightarrow$ Event) [0:N] (Event $\Longrightarrow$ User) [1:1]
- **Owner of:** (Calendar $\Longrightarrow$ User) [1:1] (User $\Longrightarrow$ Calendar) [1:1]
- **Made of:** (Calendar $\Longrightarrow$ Event) [0:N] (Event $\Longrightarrow$ Calendar) [1:N]
- **From:** (Notification $\Longrightarrow$ Event) [1:1] (Event $\Longrightarrow$ Notification)[1:N]
- **Receives:** (Notification $\Longrightarrow$ User) [1:N] (User $\Longrightarrow$ Notification) [0:N]
- **Place in:** (Event $\Longrightarrow$ Location) [1:1] (Location $\Longrightarrow$ Event) [0:N]
- **About:** (Location $\Longrightarrow$ Forecast) [1:N] (Forecast $\Longrightarrow$ Location) [1:1]

2.2 Logic model

From the ER diagram above we derive this logical model. This is how the DB will be implemented in the physical system.

- User(**idUser**, Email, Username, Password, Name, Surname, Verified, PublicCalendar)
- UserNotifications(**idNotification**,**idUser**, Pending)
- Notification(**idNotification**, Description, idEvent)
- Event(**idEvent**, Name, Description, Public, Outdoor, StartTime, EndTime, idOrganizer, idLocation)
- Invitation(**idInvitation**, idNotification, Accepted)
- Location(**idLocation**, Latitude, Longitude, Address, State, City)
- Calendar(**idUser**, **idEvent**)
- Forecast(**idForecast**, General, Precipitations, Humidity, MaxTemp, MinTemp, idLocation)
- Contact(**idUser**, **idContact**)

Some consideration about the translation from ER-Diagram to Logic Model:

We noticed that leaving the attribute Public in the entity *Calendar* this was repeated in each tuple of the user owner of the calendar. To avoid this problem we have moved the attribute Public in the entity User, calling PublicCalendar, so the decision to keep the calendar public or private stays close to the user and it is not replicated.

The table *Notification* contains the description of the notification sent by the event.So there is no redundancy because the notification description is stored only once.

The table *UserNotification* is the bridge table between User and Notification, it contains all the notification referred to the user. Each tuple of UserNotification contains a link to User, a link to *Notification*, and a boolean attribute that indicates if the notification has been seen by the user.

The relations *Organizes* and *Place in* are collapsed in the Event table with the attributes idOrganizer that is a idUser and idLocation. While the relation About between Forecast and Location is collapsed in Forecast with the foreign key idLocation.

We kept distinct entities Invitation and Notification. The table Invitation contains a reference to Notification to go back to the Event and the User.

2.2.1 Logic Model Diagram

Now we present the complete diagram of the logic model outcome from mySQL Workbench with the logic model presented above.

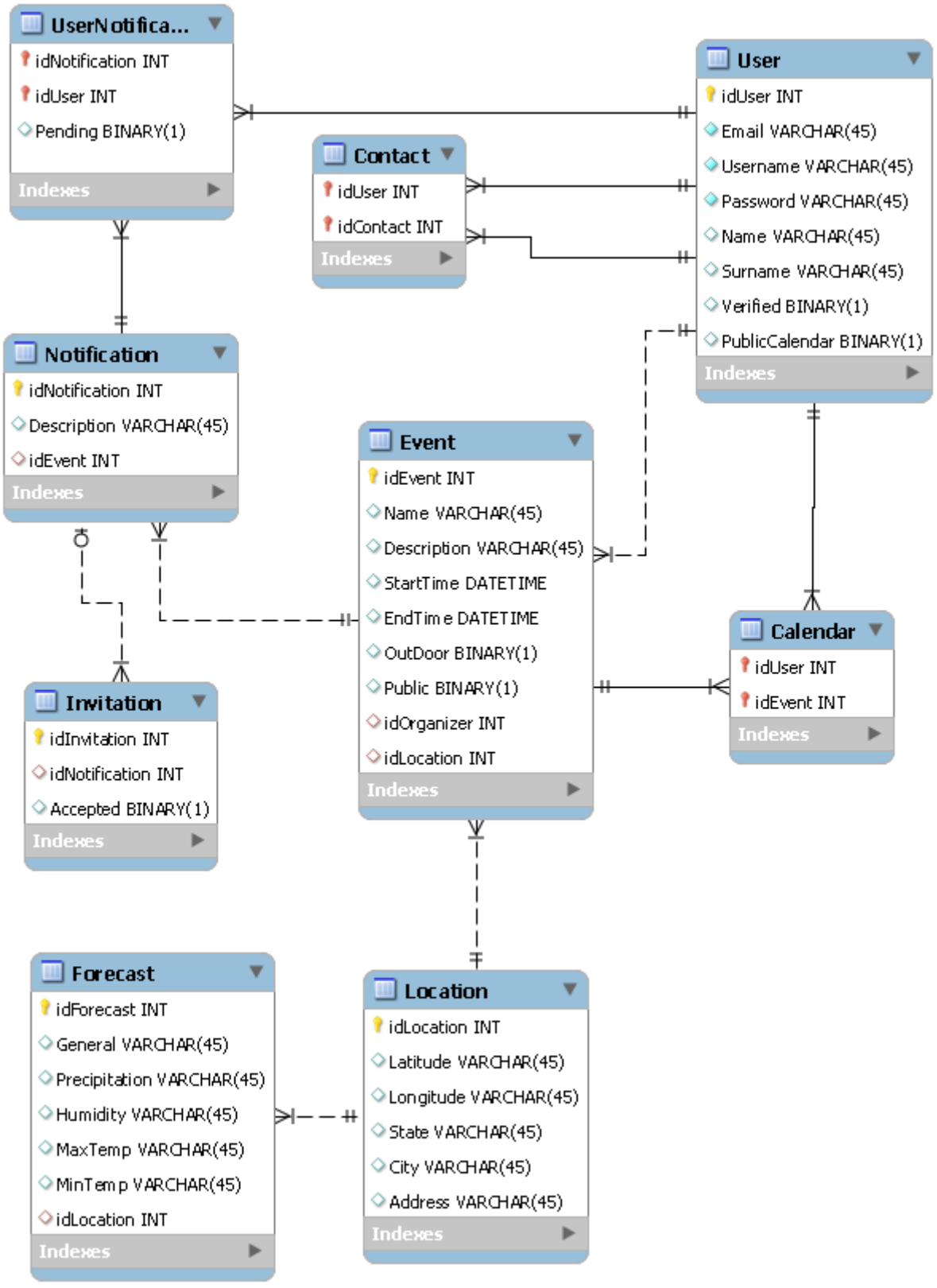

Figure 2.2: Autogenerated Logic model

3 Application Design

3.1 Architectural levels

Our application is divided in 4 tiers, the image below shows graphically how the system should work.

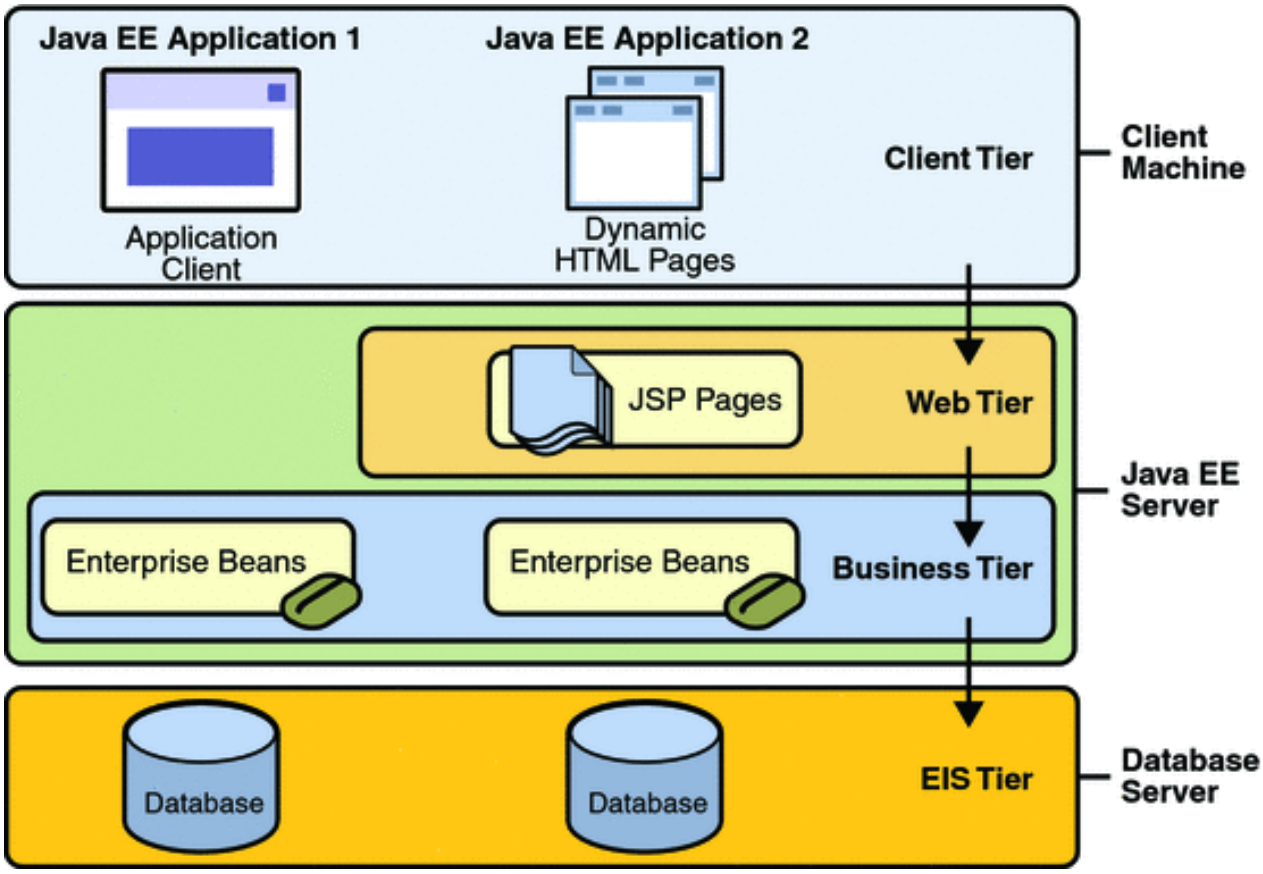

Figure 3.1: JEE Architecture

The standard architecture adopted is the client-server multi-tier divided into four logical levels: Client, Web, Business and Data tier. The user accesses the application via the own terminal that resides on the client tier, the business and the web tier are part of the application server and the data resides in the database tier. In this way, the different logical levels can be distributed on different machines and communicate with each other through the web, allowing to each tier to communicate only with the one immediately previous or next. Each user can accesses the application from the Client tier through any browser. This tier communicates with the Web tier using the HTTP protocol to communicating the user's choices and allowing the user to view HTML response pages from the Web tier that have been created based on the data received from the business tier and the client itself. The Business tier is the heart of the application since it includes the entire application logic and communicates directly with the DBMS, also interacts with an email service , using SMTP protocol, which allows you to send email notification to users. The application data are managed through MVC and represented as Java Entity Beans, application logic and the interaction with the DB are represented as EJB components and to access to the DB the tier makes use of JPA specification thanks to which it is able to extract data from a relational model. The Data tier,finally, is made by a DBMS that can only communicate with the business tier.

3.2 Navigation model

3.2.1 UX Guest

Here we present some user experience diagrams to show how we imagine the navigation through the applcation by the users.

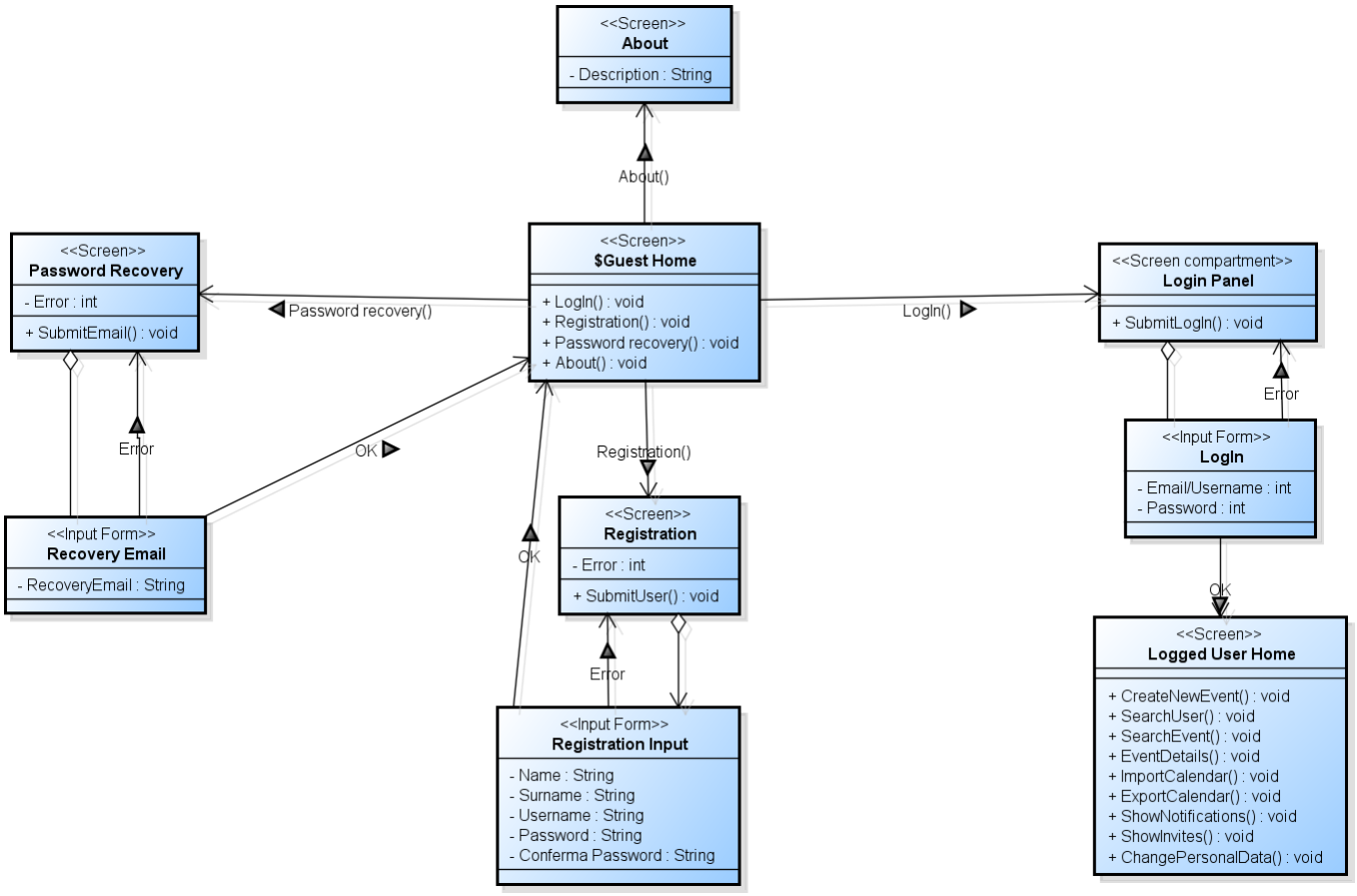

Figure 3.1: UX Guest

The above diagram shows how a guest user can move in tha main page. He can only register or login if he is already registered. He can also recover his password. When he clicks on register he goes in the register page and he's asked to give name, surname, username and password then he is sent again in the home where he can log in. In the about page there is a description of what is MeteoCal.

3.2.2 UX Logged User

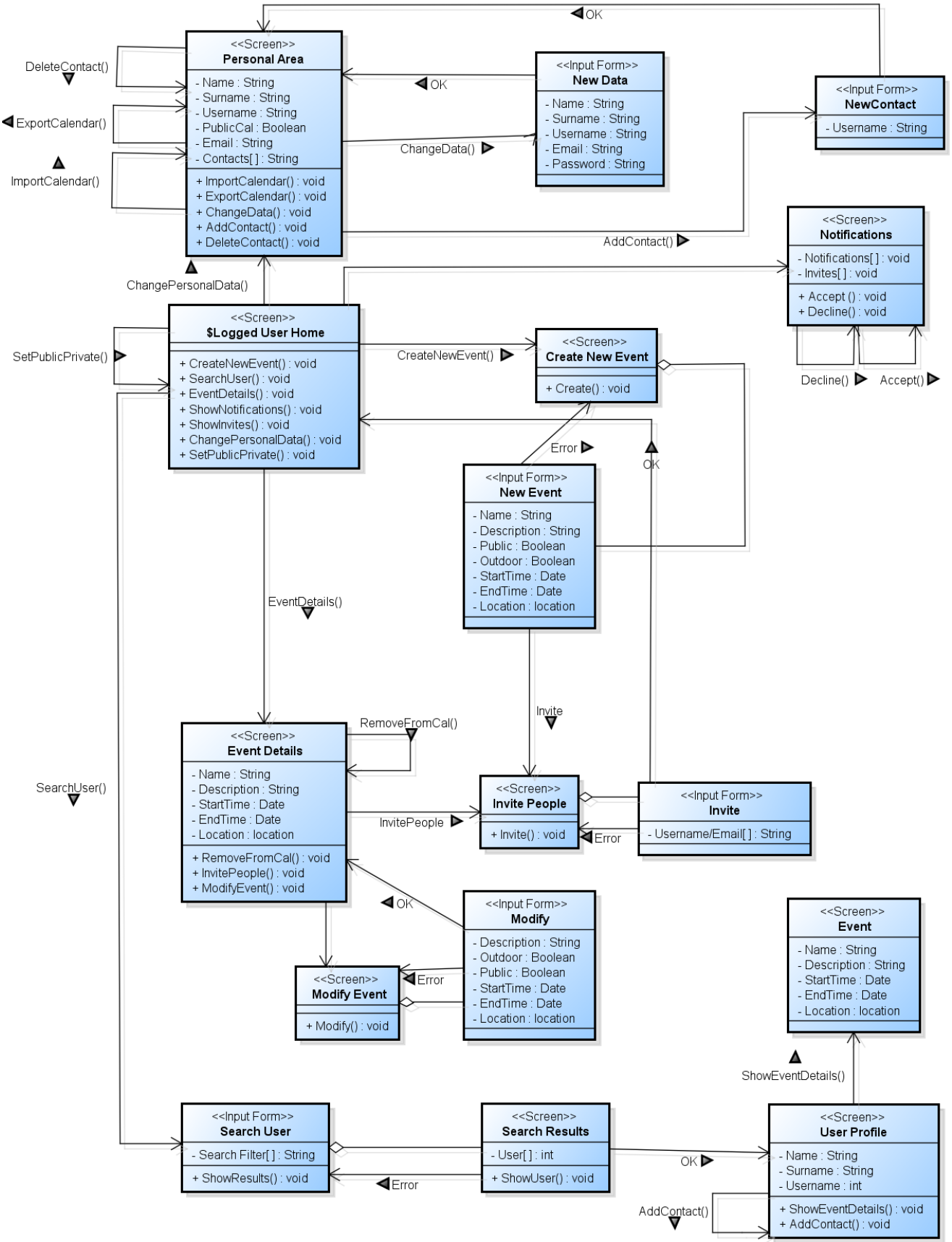

Figure 3.2: UX User

The previous diagram shows what a logged user can do. The user will see his home with his calendar. If he want to create a new event he clicks on new event and is sent to the create new event page. When he creates an event he is asked if he wants to invite people writing username or email of the users he wants to invite but he can also skip this passage. In the home page there's a search bar for finding users. In a page of another user profile a user can see his calendar if it's public and the events with the details if he clicks on them. He can also add the user in the contact list. If the user wants to modify an own event he must open the details of that event and then click on modify. There are two screens of event details, one for an own event that can be modified and one for events in which you are not the organizer.

3.3 BCE model

In this section we present the BCE model, a particular UML schema, that divides the components belonging to the three different logic levels of the application server.
The BCE model is very useful for implementing the Model-View-Controller (MVC) pattern thanks to the presence of the fundamental stereotypes Boundary-Control-Entity.

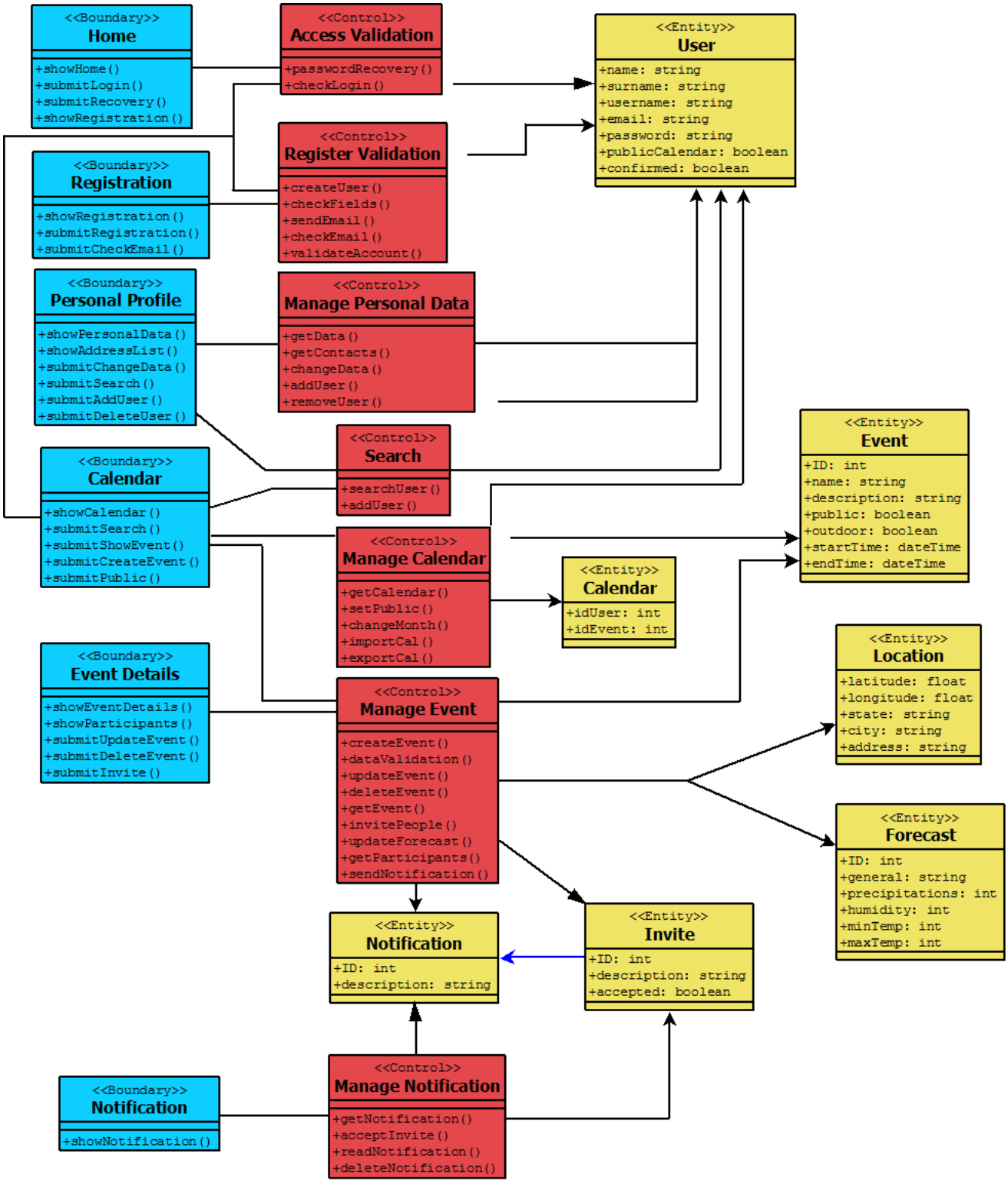

Figure 3.3: BCE Model

3.4 Sequence diagrams

3.4.1 Registration and Login

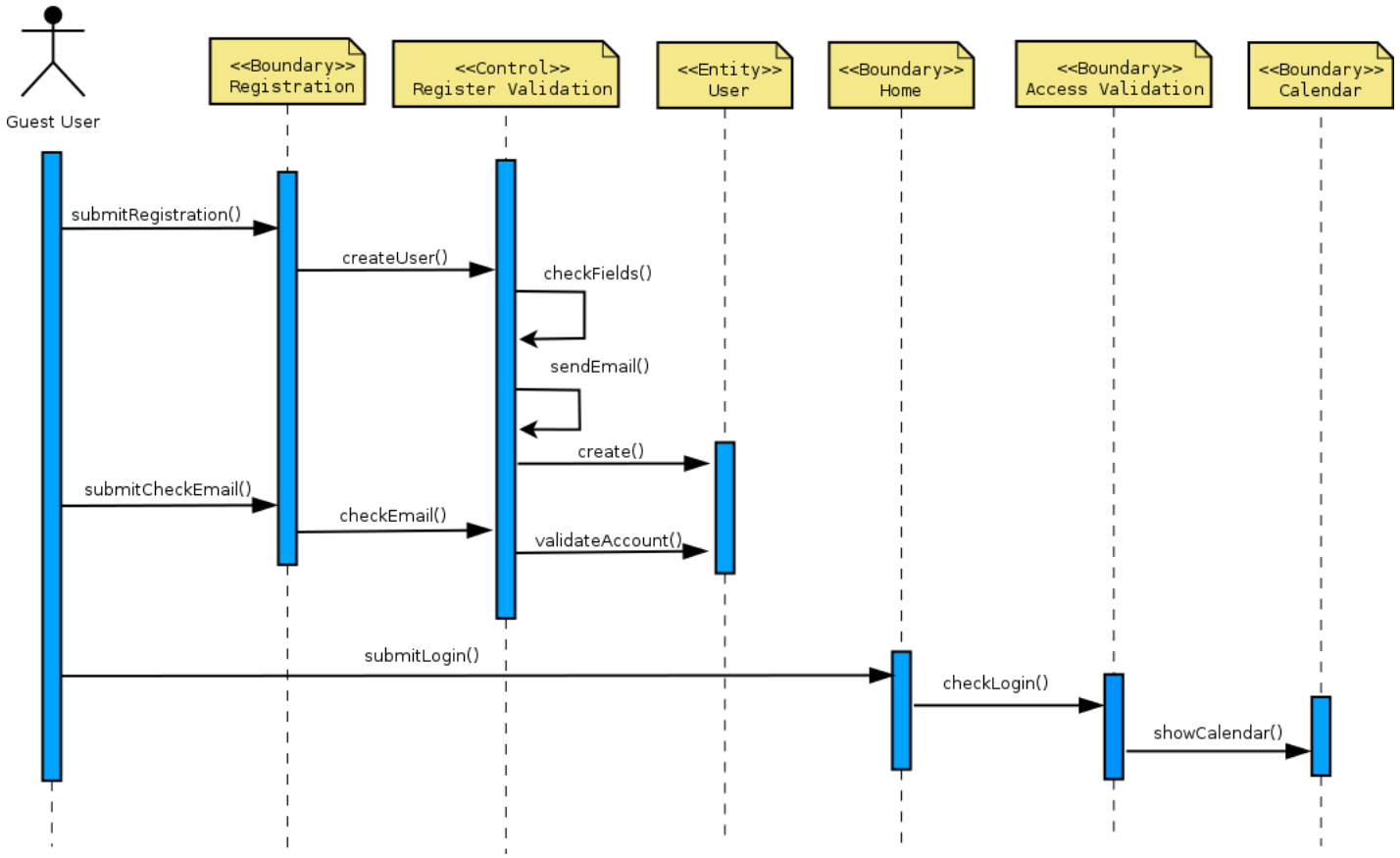

Figure 3.4.1: Registration and Login

3.4.2 Create Event

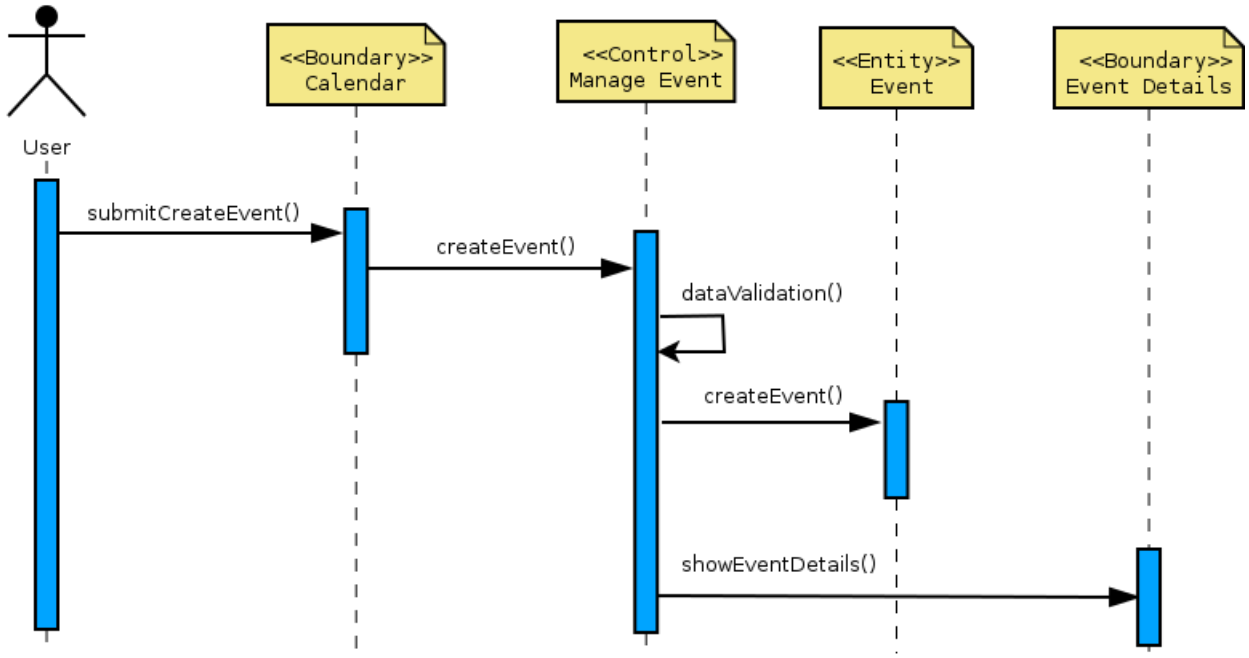

Figure 3.4.2: Create Event

3.4.3 Update Event and Notification

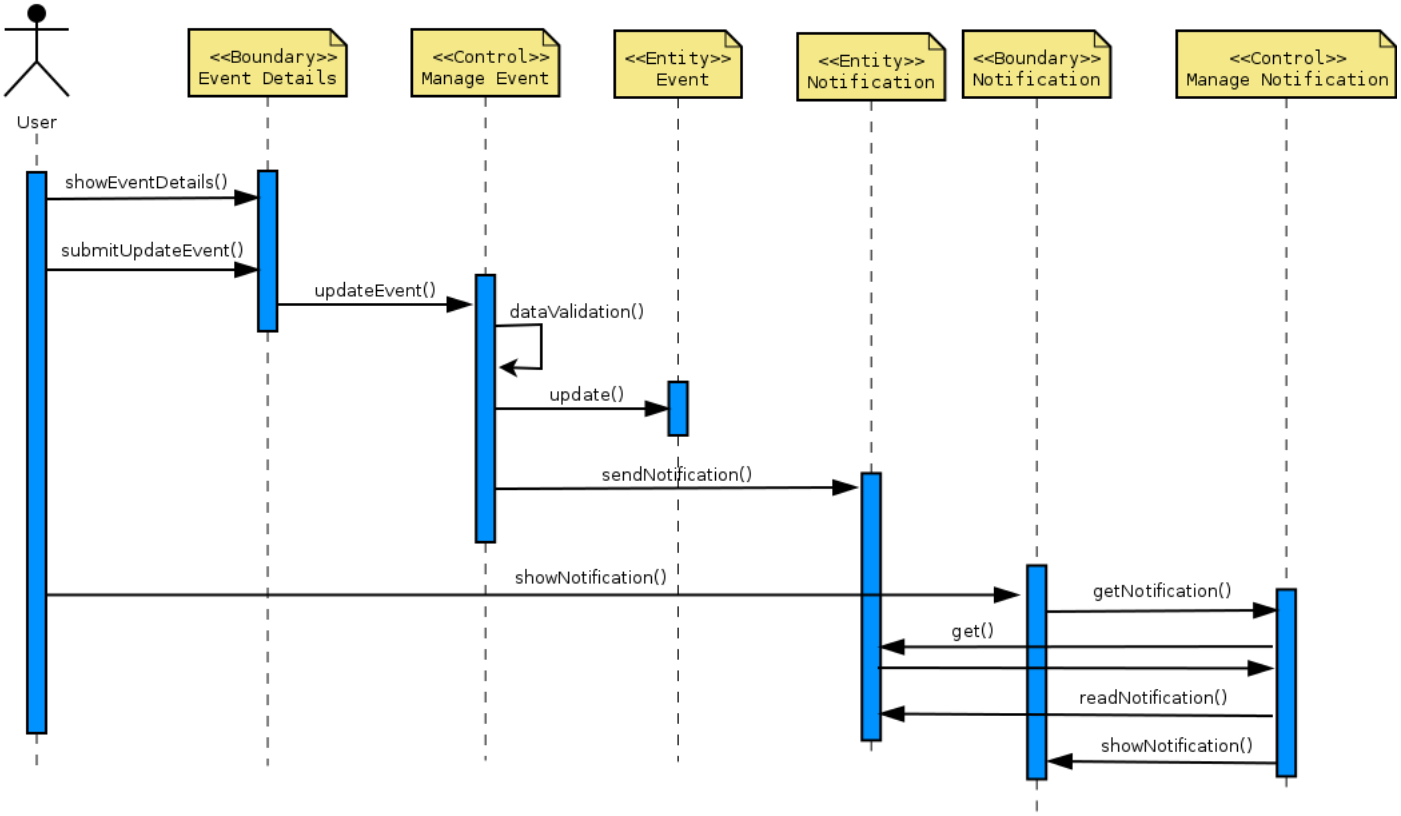

Figure 3.4.3: Update Event and Notification

3.4.4 Invite People

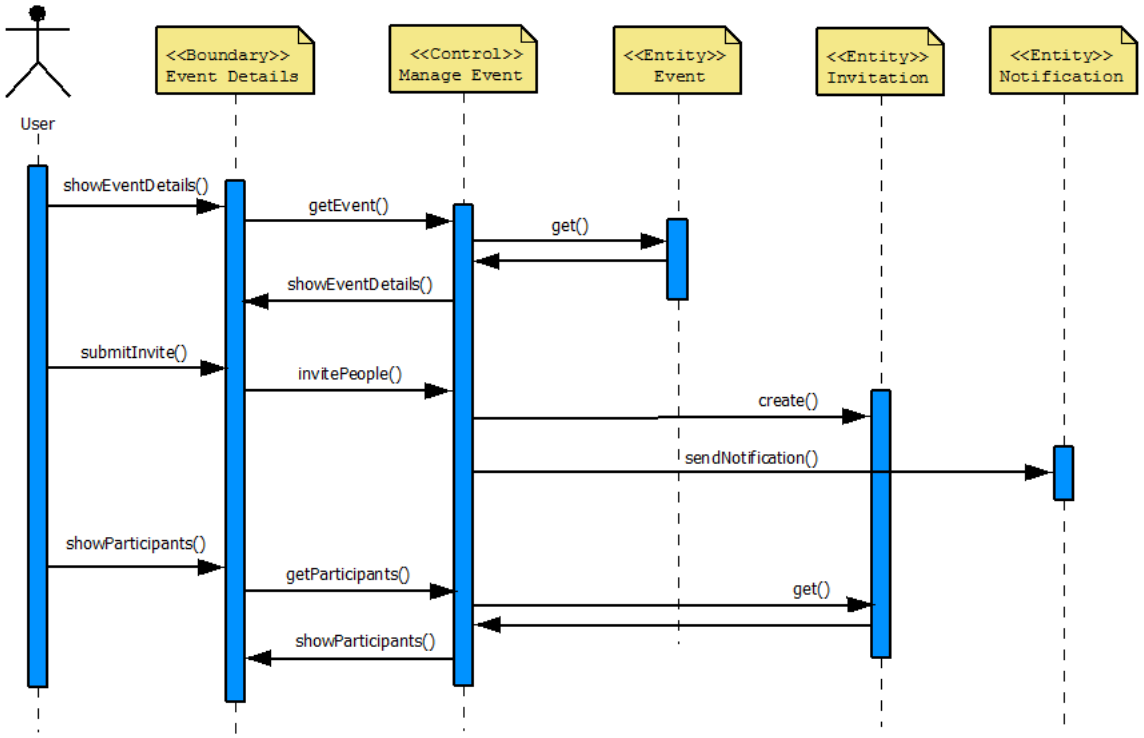

Figure 3.4.4: Invite People

4 Page sketch

In this section we present some interfaces relative to the main application screens.
With these sketches we want to show how the contents are divided in the different pages and we want to give an idea of how it will be the navigation through the pages of the application.

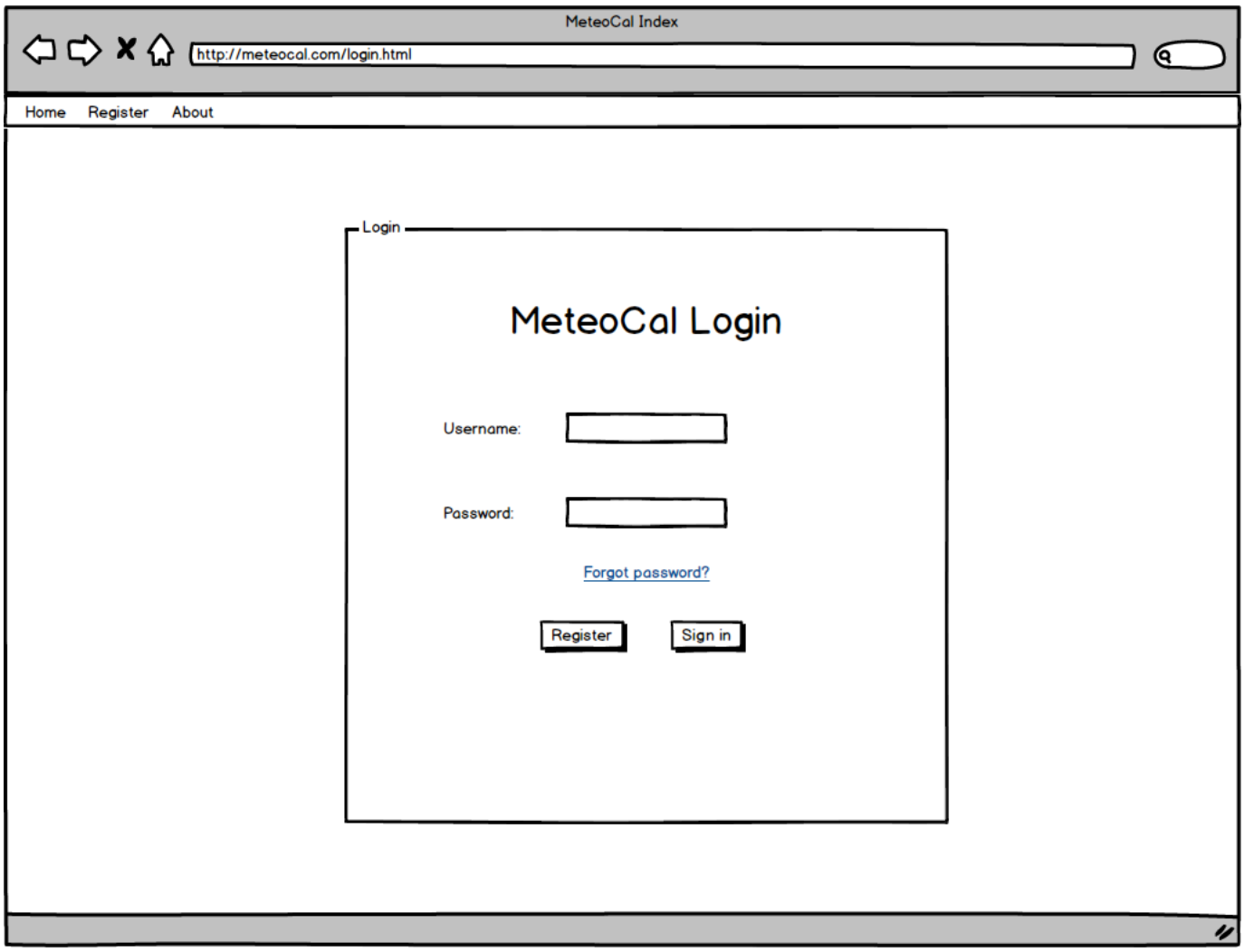

Figure 4.1: Homepage

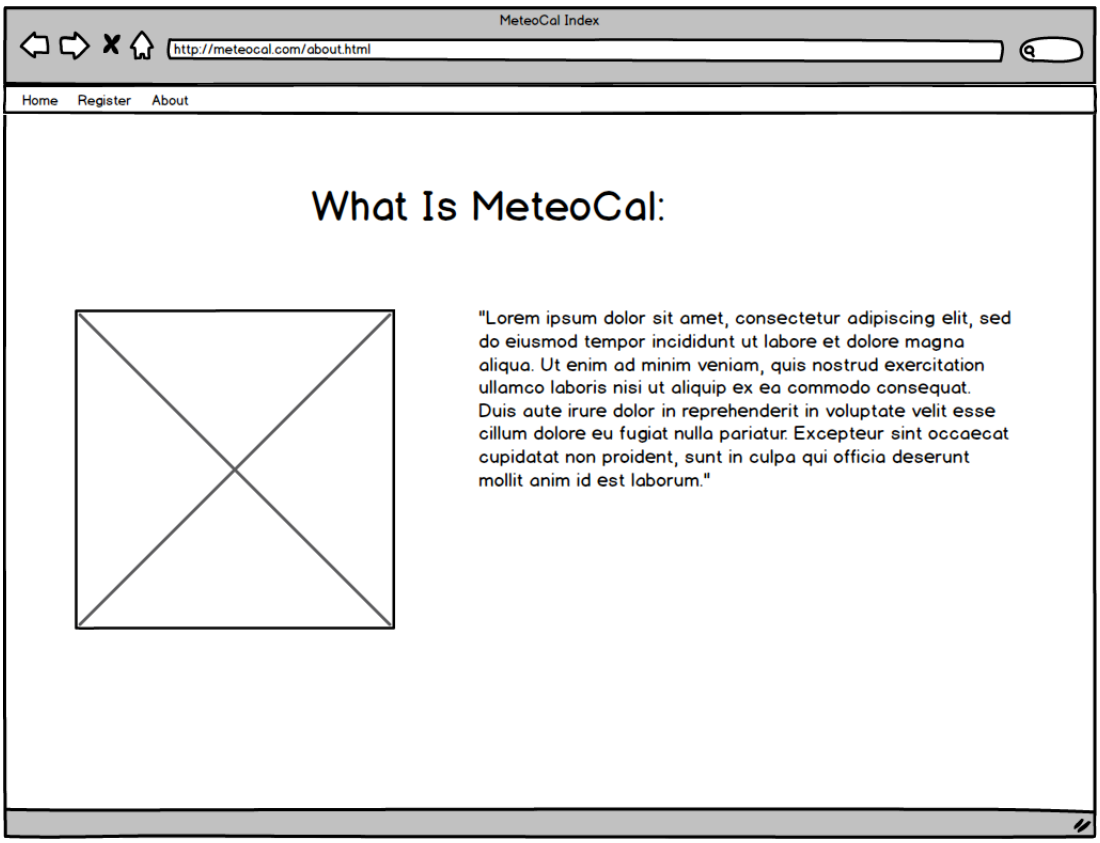

Figure 4.2: About

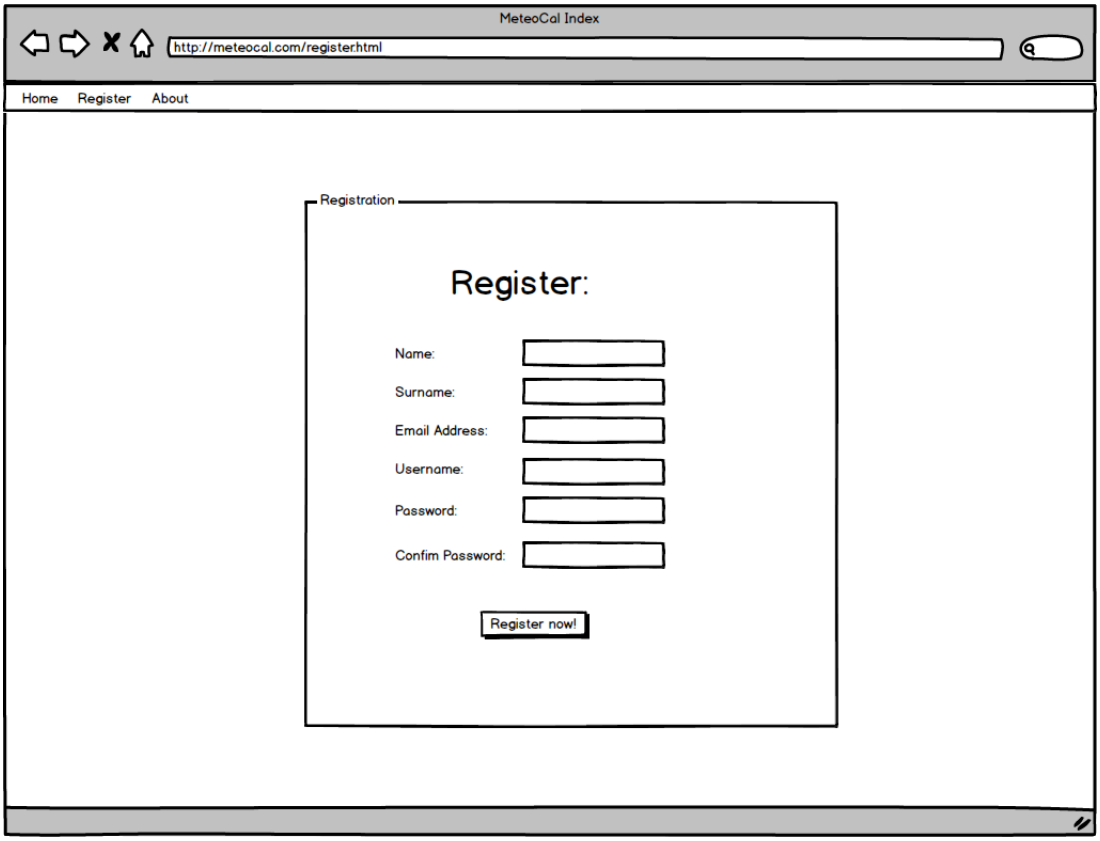

Figure 4.3: Registration

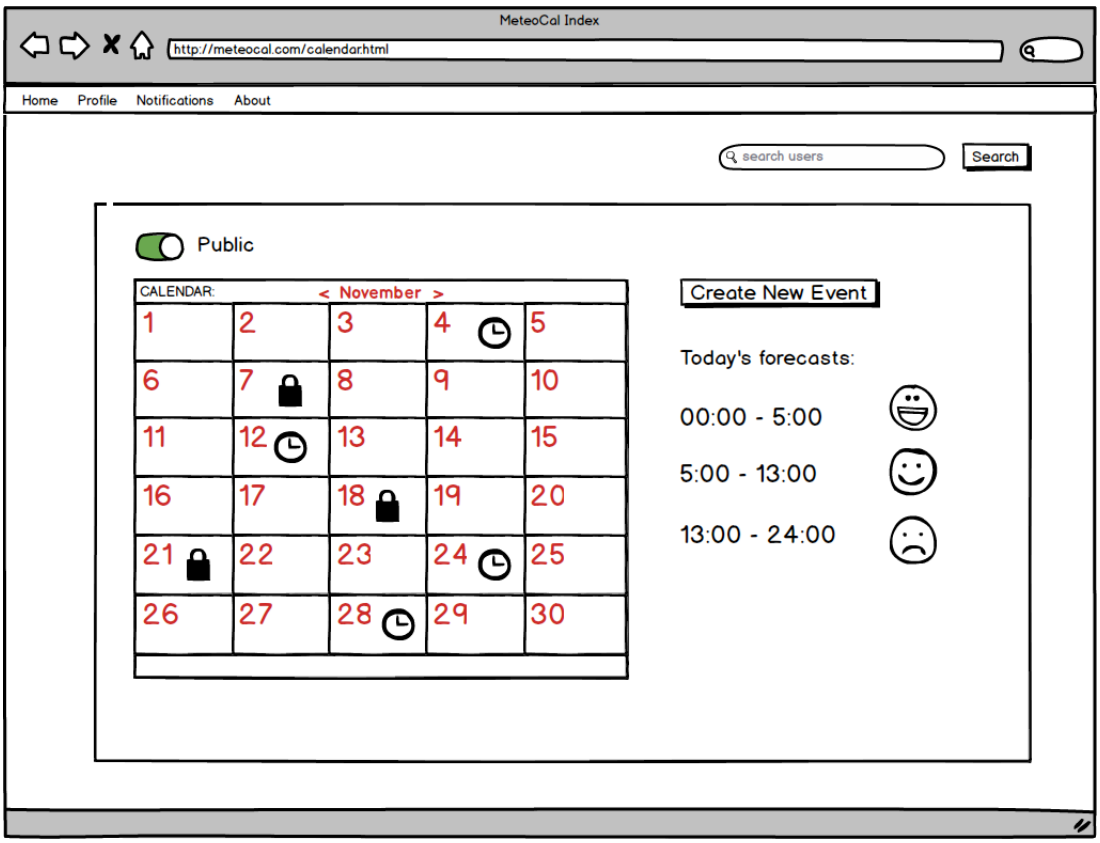

Figure 4.4: Calendar

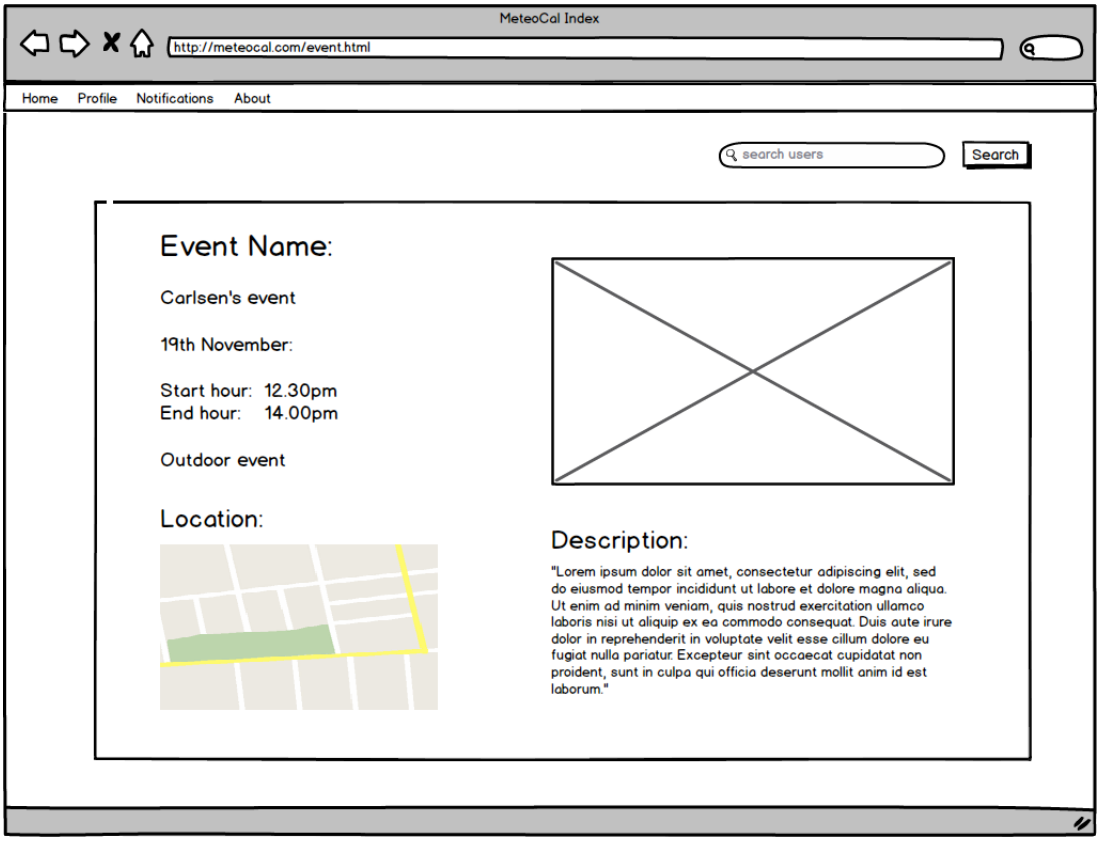

Figure 4.5: Event Details

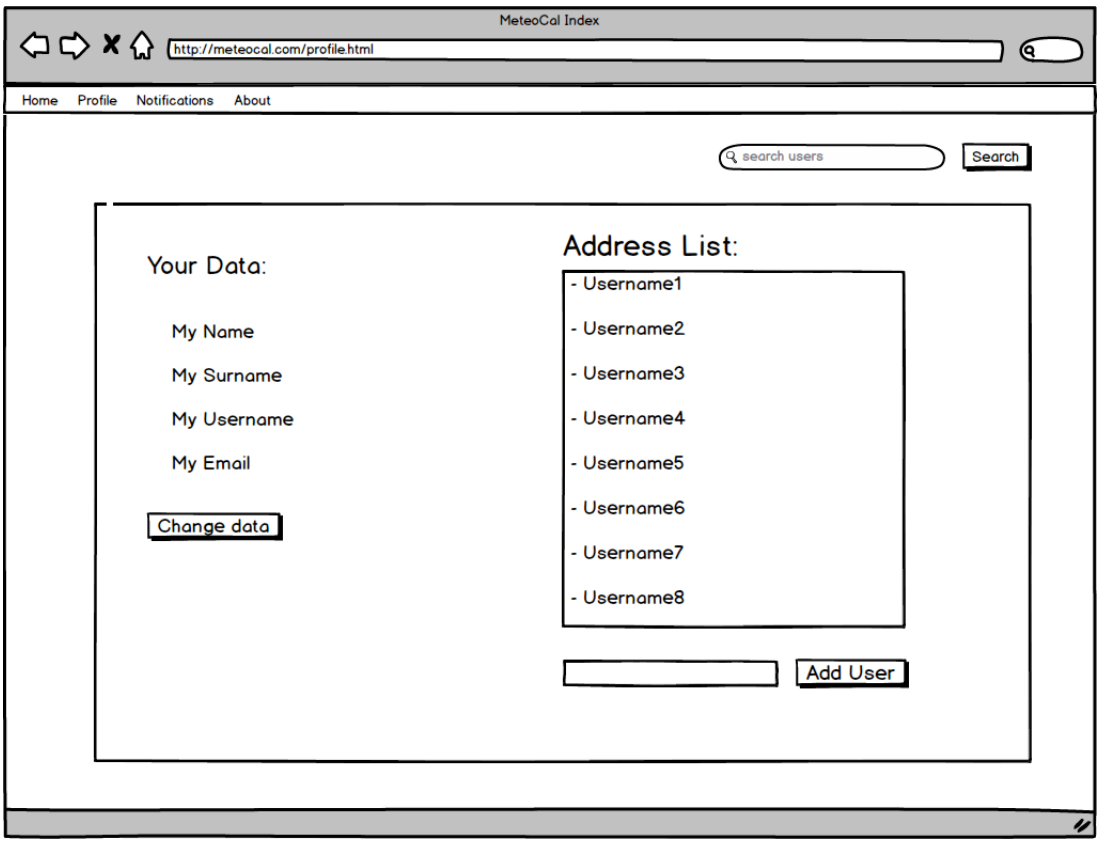

Figure 4.6: Personal Profile

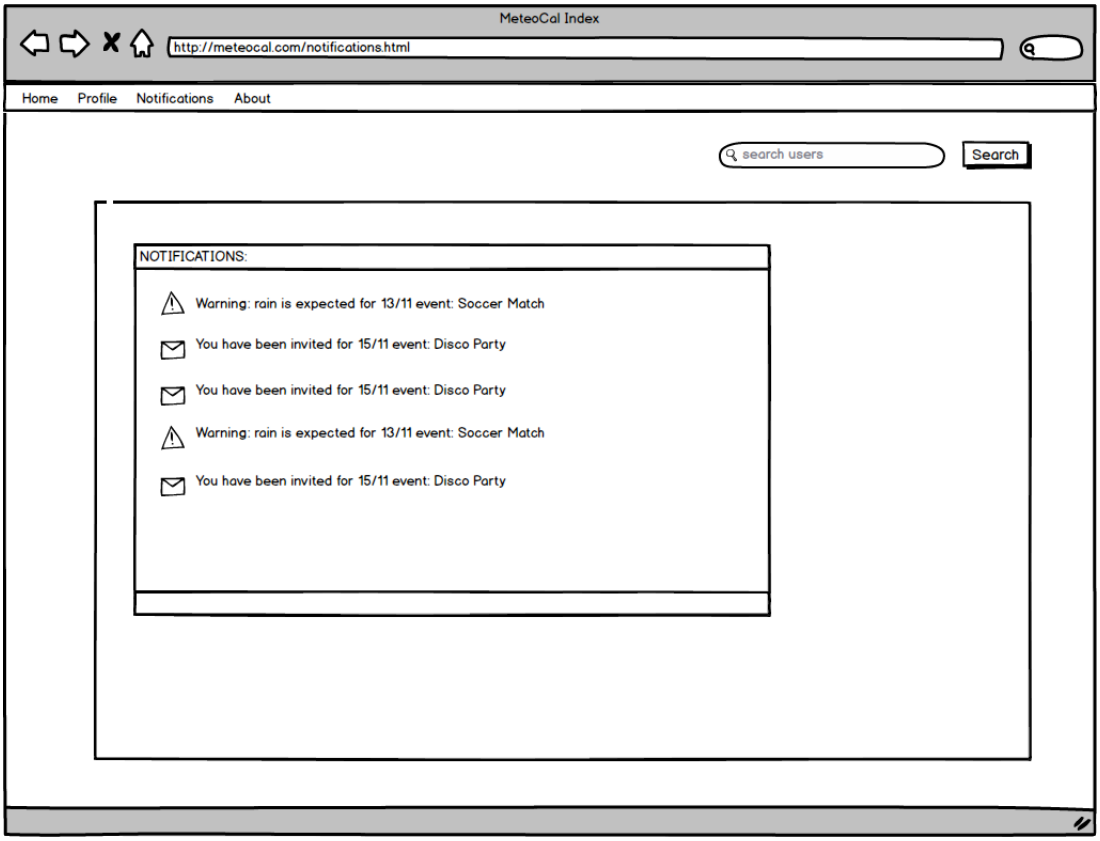

Figure 4.7: Notifications

5 RASD adjustment

We have adjusted the class diagram in the following way:

- The class Forecast is no longer connected directly to the event but is related to Location as it depends directly to.
- The class Invitation has become a subclass of Notification that is no longer a composition on User.

Here we present the new class diagram:

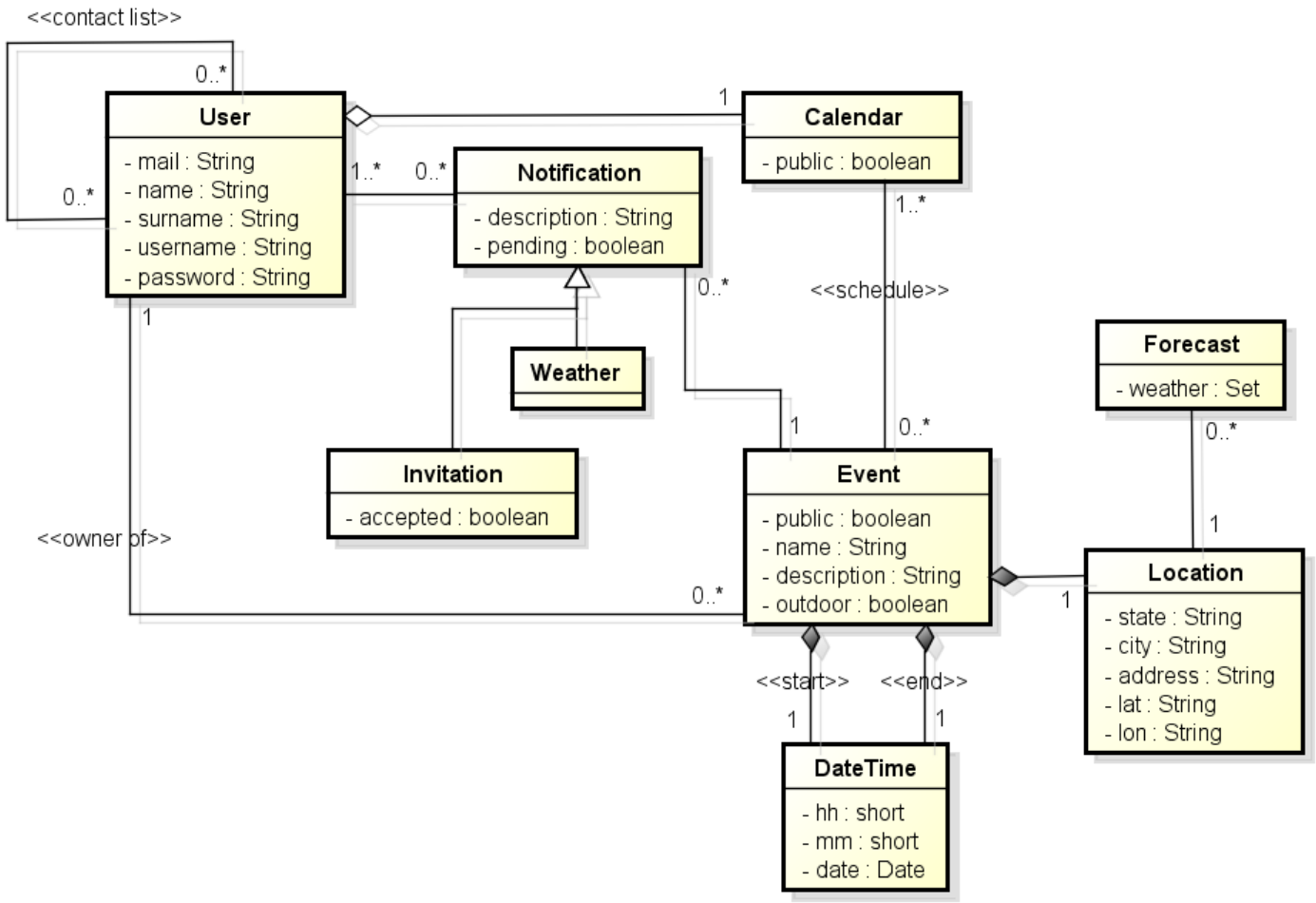

Figure 5: Class Diagram

6 Division of the roles

	Rossotti	Pasina	Rubiu	Total hours
General Description	2	1	1	4
Data Design (ER and Logic models)	4	3	5	12
Navigation Model (UX Diagrams)	5	5	4	14
BCE Diagram	6	7	7	20
Sequence Diagrams	6	6	6	18
Page Sketch	2	3	2	7
RASD adjustment	1	1	1	3
Total hours:	26	26	26	78