Politecnico di Milano

Software Engineering 2

MeteoCal

Design Document

Authors:

Andrea Enrico Turri

Andrea Salmoiraghi

Fabiano Riccardi

07/12/2014

Contents

1 Introduction

1.1 Purpose

This document's purpose is to develop the design of the MeteoCal project. After the analysis of the requirements and the explanation of how should the system work made in the RASD document, we are going to describe the software components and the applications that will be used to implement them.
The document will explain the architectural decisions and tradeoffs chosen in the design process and its justifications.

1.2 Document overview

The document is divided into four main parts, as follows:

- **Design considerations**: provides a general description of MeteoCal functionalities and matters related to the overall system and its design, describes the performance requirements of MeteoCal.
- **Software Architecture**: specifies the general architecture, describes the basic structure and interactions of the main subsystems and the technologies chosen to implement the system, as well as identifies the subsystems.
- **Database Design**: specifies in detail the how we modeled the database, from the conceptual model to the logical one.
- **Detailed System Design**: specifies in detail the components of the system through different architectural views and shows interactions between the users and the application. We will show User Experience diagram, Boundary Entity Control diagram and some Sequence diagrams showing interactions among components while performing the main operations.

1.3 Glossary

We refer to the Glossary provided previously in the RASD Document.

2 Design considerations

2.1 Functionalities

The following functional requirements were identified during RASD. These functionalities are grouped by the following functional areas:

- **User management:**
  - Registration to the system
  - Log in
  - Log out
  - Recover password
  - Edit profile information
- **Calendar management:**
  - Set privacy policy
  - Import and export
  - Search for other users' calendar
- **Event management:**
  - Creation of event
  - Updating of event
  - Deletion of event
  - Invite other users
  - Remove invited users
  - Accept / refuse invitations
  - Remove event from calendar after accepting invitation
  - Set privacy policy
  - Search for events
  - View event details
- **Notification management:**
  - Bad weather notifications
  - Event updates notification

2.2 Performance requirements

2.2.1 Reliability

We would like to guarantee more reliability of the system and so we would like to create a distributed DB in order to duplicate the logic and to create backups. Hence we would avoid faults that could affect the entire system, as well as the loss of users' data.
Note that this is considered an advanced feature that we will not implement in the project.

2.2.2 Availability

Since the system has to deal with temporal events, we must guarantee a high percentage of availability, to avoid cases where a person loses appointments due to the malfunctioning of the system.

2.2.3 Security

Since we are dealing with possible private data, security is a very important aspect: we must guarantee the secrecy of all sensitive data (users' data and events' data) and we must avoid that unauthorized people access what they are not supposed to see.

2.2.4 Portability

The application, on server side, must work on every machine running a JEE JVM and a MySQL server.
On the client side we guarantee the maximum portability allowing users to access the application through a browser. We'll try (if there is enough time) to write the output code compatible with the majority of modern browsers, including mobile ones.

3 Architecture Description

3.1 System architecture

The design approach is a JEE Architecture which is based on a client-server 4-tier distributed system, where each tier is described as follows:

- **Client Tier**: this tier is responsible of translating user actions and presenting the output of tasks and results into something the user can understand;
- **Web Tier**: it receives the requests from the client tier and forwards the pieces of data collected to the business tier fro processed data to be sent to the client tier, eventually, formatted.
- **Business Logic Tier**: this tier contains the business logic, it coordinates the application, processes commands, makes logical decisions and evaluations, performs calculations;
- **Persistence Tier**: this tier holds the information of the system data model and is in charge of storing and retrieving information from database.

Note that those tiers are divided into three different levels: client machine (Client tier), JEE server (Web tier and Business tier) and database server (Persistence tier) as shown in the following image. Although the program is projected in this way, in the practical implementation the JEE server and the Database server will run on the same machine (but, of course, they are in different processes).

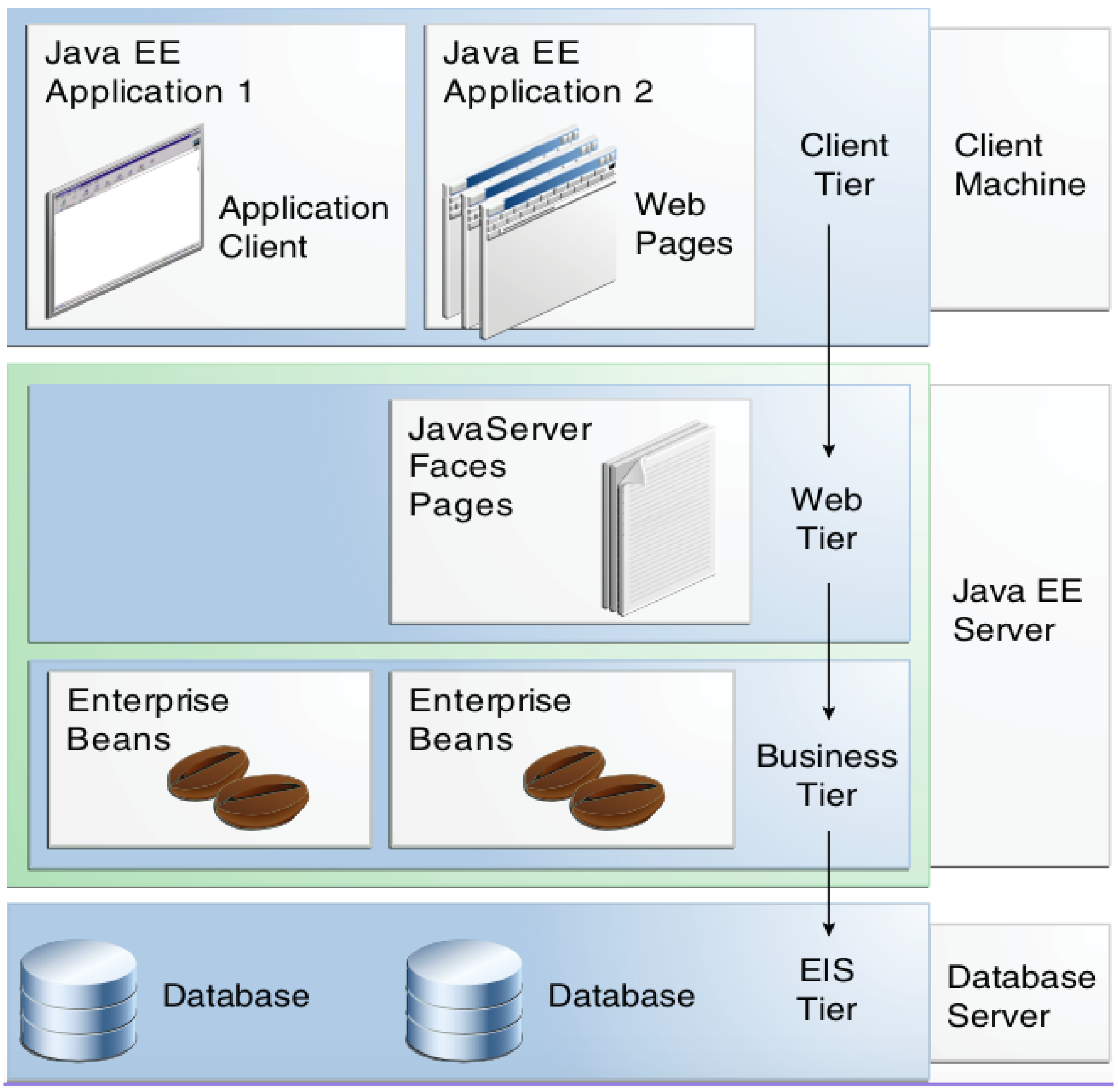

Figure 1: JEE Architecture with different tiers

The division in those tiers is made to support the MVC pattern of developing an application. The team has decided to follow that pattern, that is composed by three elements: Model, View and Controller.

With reference to the BCE diagram, is very easy to identify every component: the model (represents the state of the application) will be composed by the Entity classes, the controller (performs any logic necessary to obtain the correct content for display and decides which view it will pass the request to) will be composed by the Control classes and finally the view (renders the content passed by the controller) will be composed by the Boundary classes.
This pattern is mapped on the JEE architecture as shown in the following picture:

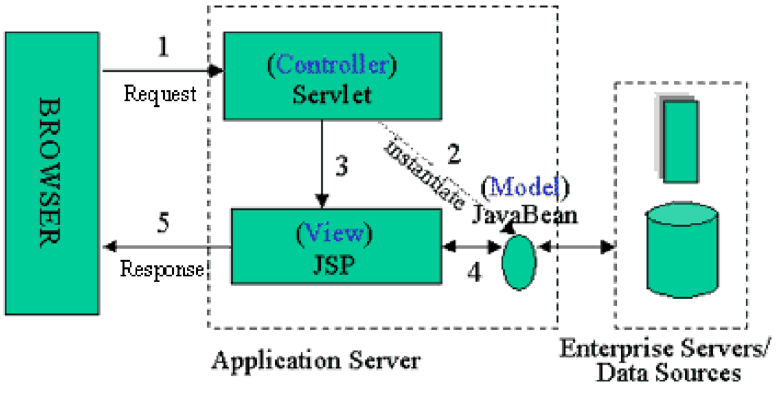

Figure 2: MVC mapped on JEE

The controller will be implemented using Servlets, that will handle HTTP requests and depending from what is requested, can instantiate EJBs (will be used to handle the state of the application, is the model) or forward the request to a JSP page (will handle the presentation logic and will return the response to the browser, is the view).

3.2 System technologies

For each tier described in the previous section, we explain now the technologies that will be used to achieve our goals:

- **Client Tier**: it contains Application Clients and Web Browser since our application is web-based and, as said before, it is the layer that interacts with the user;
- **Web Tier**: it contains the Servlets and Dynamic Web pages containing XHTML to be elaborated;
- **Business Logic Tier**: it contains the Java Beans, that contain the business logic of the application;
- **Persistence Tier**: it contains Java Persistence Entities that wrap the content retrieved from the Database.

3.3 System specification

As said in the previous section, Web and Business Logic tiers are a merged into a unique level running JEE 7 on a Glassfish 4.1 Server and the database is a MySQL 5.6.

3.4 Subsystem Identification

In this section we want to explain in more detail how MeteoCal is composed by subsystems. Every Subsystem can be seen as independent from the others and have an own logic specific function. The main subsystems are:

- **Access management**: it allows to control user/password in login phase, registration and to retrieve password when a user lose it;
- **Event management**: this module has to provide the functions to create, update and delete an event. This subsystem can be divided in 2 subsystem: one for the organizer (advance function to manage the people) and one for the invitate user, which is more simple and it supplies the function to view the event and accept or refuse the invitation;
- **Calendar management**: it provides the function to show a calendar with various behavior and export / import;
- **User settings management**: this module has to provide the basic function to manage own profile, hence modify user's data, show it and change option like privacy and other preferences;
- **Notification management**: this module has to provide the function to show correctly the notification (update the list of notification, sending notification to another user, sending mails etc.).
- **Favorite management**: this module provides the functions to allow the users to manage favorite users;
- **Search management**: this module provides the functionalities to allow the users to search for other users and events;

Every subsystem depends from the others, and everyone is accessible with respect to the privileges that the user had gain using MeteoCal.

4 Data Management

The data of the application is stored into a relational database which has been modeled with the Entity Relationship model and then it has been translated into the logical schema.
In this section we try to explain our choices.

4.1 Conceptual design - ER Model

We first built the ER model to begin to think about the data we need to store and relationships. With the conceptual design the most important entities and the relationship between them were defined.

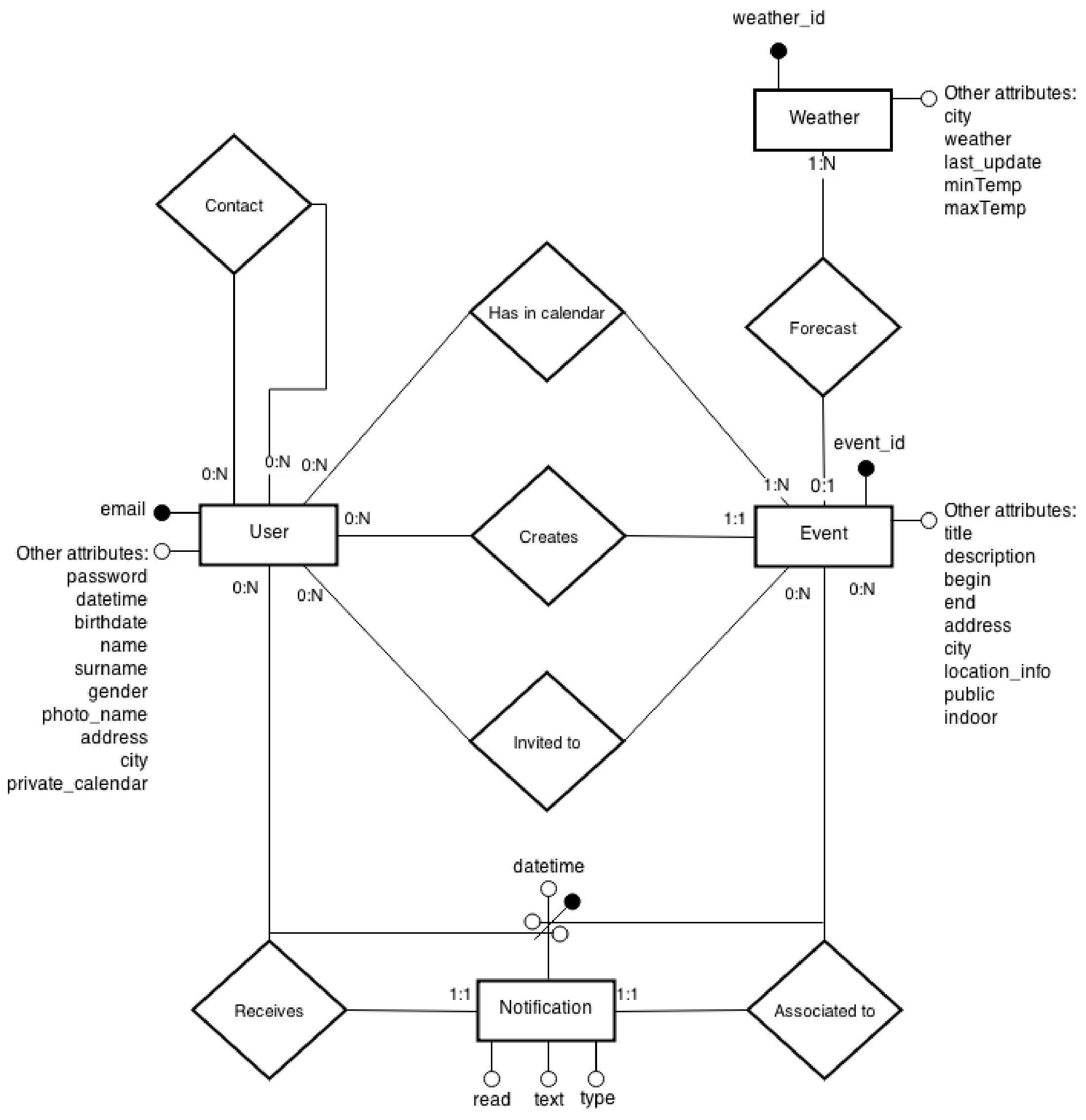

Figure 3: ER Model

4.1.1 Entities

User: In the projected system there are *Guests* and *Registered users*. The first category doesn't need an entity because the system doesn't need to store data for them and the second category comprises all registered users with all the necessary data to perform login and all functionalities described before.
The primary key is the email since we assumed in the RASD that it is the username and it must be unique in the system.
Other attributes are: password, datetime (of registration), birthdate, name, surname, gender, photo, address, city, private calendar flag (to set calendar visibility).

Event: The system should be able to store events, so this entity is necessary to contain every information about them.
The primary key is an ID since it is more efficient to us to develop and gives the system more flexibility that could be useful for future expansions / releases of the system (in which for example there are more than one calendar for every user and there is the possibility to create overlapped events in more than one calendar).
Other attributes are: title of the event, description, begin date/time, end date/time, address, city, location additional info, public flag (for event visibility), indoor/outdoor flag.

Notification: Since the system shows in the main user's page a set of recent notifications, we need an entity to store them.
The primary key is composed by the identifier of the event to which is associated to, by the user that receives it and by a date/time. In this way we keep a different instance of notification for all users that participates to an event, also with the possibility to customize the text.
The notification, of course, contains some text to be displayed to the user, an attribute to identify if it is an update, deletion or bad weather and a flag attribute to determine whether it has been read or not.

Weather: One of the main goals of MeteoCal is to associate weather information to events, so we need an entity to store these data (that may be updated automatically by the server). The primary key is a generic identifier, it contains some attributes to characterize the weather and to identify the city associated to, which can be slightly different from the one inserted by the user (maybe because the weather system doesn't know that city, it can only provide information for a very close city).

4.1.2 Relationships

The entities described above are related to each other thanks to the following relationships:

Creates (one to many): Every user can be the creator of 0 or many events, while an event has one and only one user that is the organizer.

Has in calendar (many to many): Every user can have in his calendar 0 or many events, while an event must have at least one user that participates (i.e. the organizer) or many (i.e. other participating users).

Invited to (many to many): Every user can be invited to join 0 or many events, while an event can have 0 or many invited users (that are different from participating users: these have already accepted).

Contact (many to many): Every user can have in his favorites 0 or many users, while an user can be a contact for 0 or many users.

Receives (one to many): Every user can receive 0 or many notifications, but a notification is associated to one and only one user.

Associated to (one to many): Every event can have 0 or many notifications generated by the system, but a notification is associated to one and only one event (which is also part of the primary key).

Forecast (one to many): Every event can have associated the weather (when available) and an instance of weather can be shared by more than one event.

4.2 From ER to Logical Database

4.2.1 Considerations

The entities described above are related to each other thanks to the following relationships:

- The relationship *Has in calendar* becomes a bridge table between *Event* and *User* since it is a many to many relationship. The table is composed by the two foreign keys with the identifier of the user and of the event.
- The relationship *Creates* is merged into the table representing *Event*, since every event must have one and only one user who is organizer. So the *Event* table has an attribute referencing the identifier of the organizer user.
- The relationship *Invited to* becomes a bridge table between *Event* and *User* since it is a many to many relationship. The table is composed by the two foreign keys with the identifier of the user and of the event.
- The relationships *Receives* and *Associated to* are merged into the table representing *Notification*: this has the primary key composed by the foreign key referencing the event identifier and the user identifier.
- The relationship *Contact* is a many to many relationship which is translated into a bridge table that has the foreign key composed by the identifier of the user who has the favorite and by the identifier of the user who is a contact for the previous user.
- The relationship *Forecast* is a one to many relationship which is translated by adding the foreign key of the weather into the event table.

4.2.2 Translation

In this section we show the logical schema generated:

user(email, password, timestamp, birthdate, name, surname, gender, address, city, photo_name, private_calendar)

event(event_id, *creator*, title, description, begin, end, address, city, location_info, public, indoor,*weather*)

notification(*event_id*, *user_id*, datetime, text, type, read)

invitation(*user_id*, *event_id*)

event_in_calendar(*user_id*, *event_id*)

contacts(*user_id*, *contact_id*)

weather(weather_id, last_update, city, weather, minTemp, maxTemp)

We have underlined the primary keys of each table and we styled in italic the foreign keys that each table contains.

We can also draw it to give a more clear representation and to give an idea of attributes' data types:

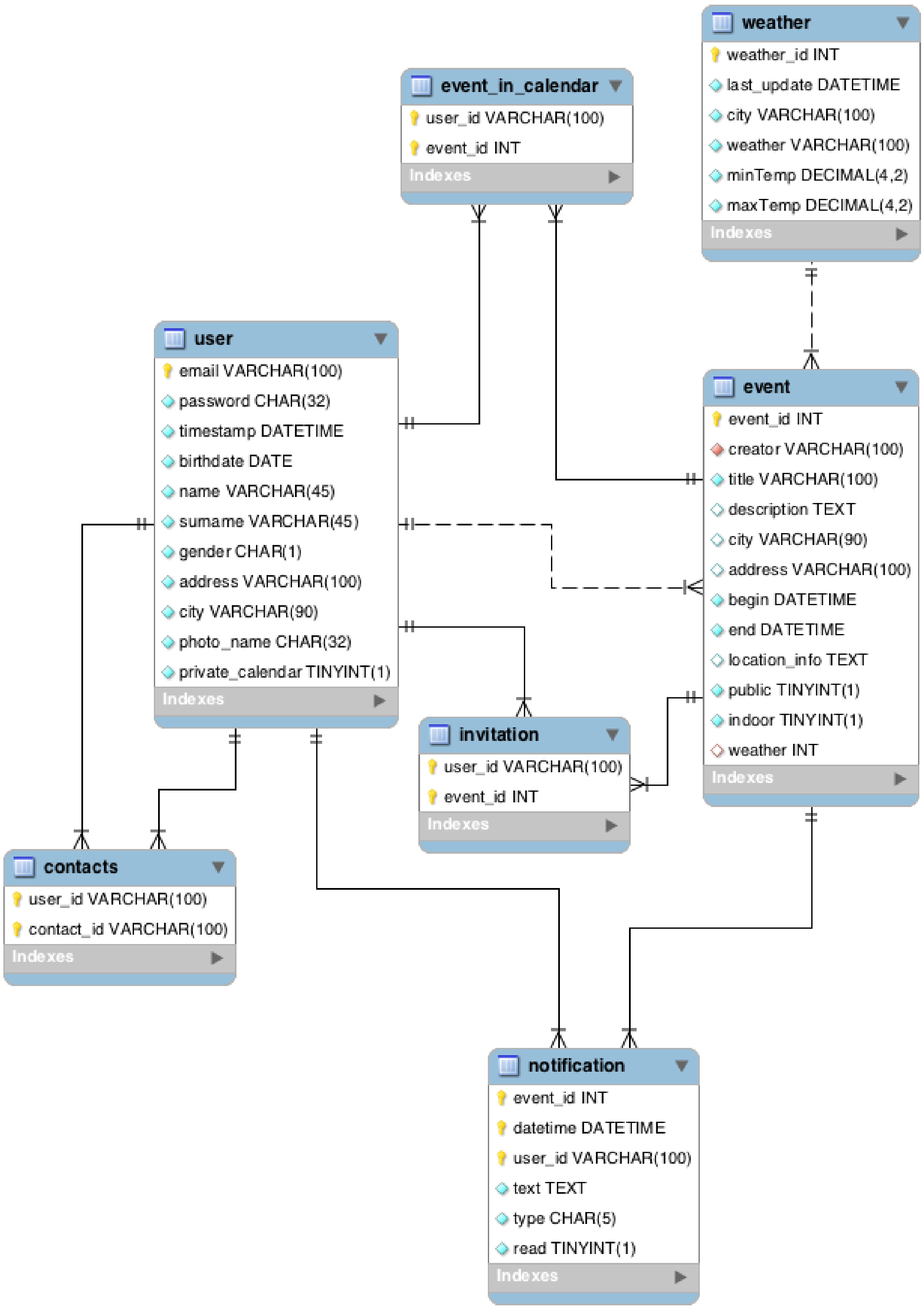

Figure 4: MySQL Workbench logical model

5 User Experience

5.1 Introduction

In this section we are going to describe the User Experience we want to provide to our final users. The UX diagram looks like a class diagram, but the "classes" have appropriate stereotypes:

- «screen»: these classes represent the pages of the application;
- «input form»: these classes represent a part of the page where the user is asked to enter some input and click a button to submit;
- normal classes: these represent the data which can be displayed in the screen.

We decided also to split the UX Diagram in five smaller ones in order to give more readability. The division is made on the basis of functionalities.
Each subsection below is entitled with the name of the functionalities represented in the sub-diagram.

5.2 Assumptions

It is important to remark that this UX diagram is necessary to give an idea of the software-to-be and here we do not distinguish among functionalities that could be used by a type of user or another (e.g. as it is be possible to see below, the Event screen contains the methods to be redirected to the update page, but this, obviously, will be available only to the user that created the event).

As it is possible to see in the diagrams below, there are some pages that are marked with the "$" landmark. This means that they are always reachable from every page of the system.
But it is important also to distinguish between the system home and the other pages marked with "$":

- The home screen is the main page of the system where every guest user is redirected when enters in MeteoCal, it is reachable from every page when the user is not logged in through a method not represented and it is also reachable from every page of logged in users through the *logout()* method which, obviously, logs out the user. The method *logout()* is represented only once in this diagram because of readability, but it must be present in every screen of the system.
- The page to create events, settings, the user's main page and the search page are reachable from every page only for logged in users, the methods to reach them are not shown for readability reasons. Note that the search page is always reachable by filling a search input that will be present in all pages, for this reason it also contains the landmark "$". A logged user can't reach the home unless it logouts from the system.

5.3 UX Diagram

5.3.1 Registration, Login and Password reset

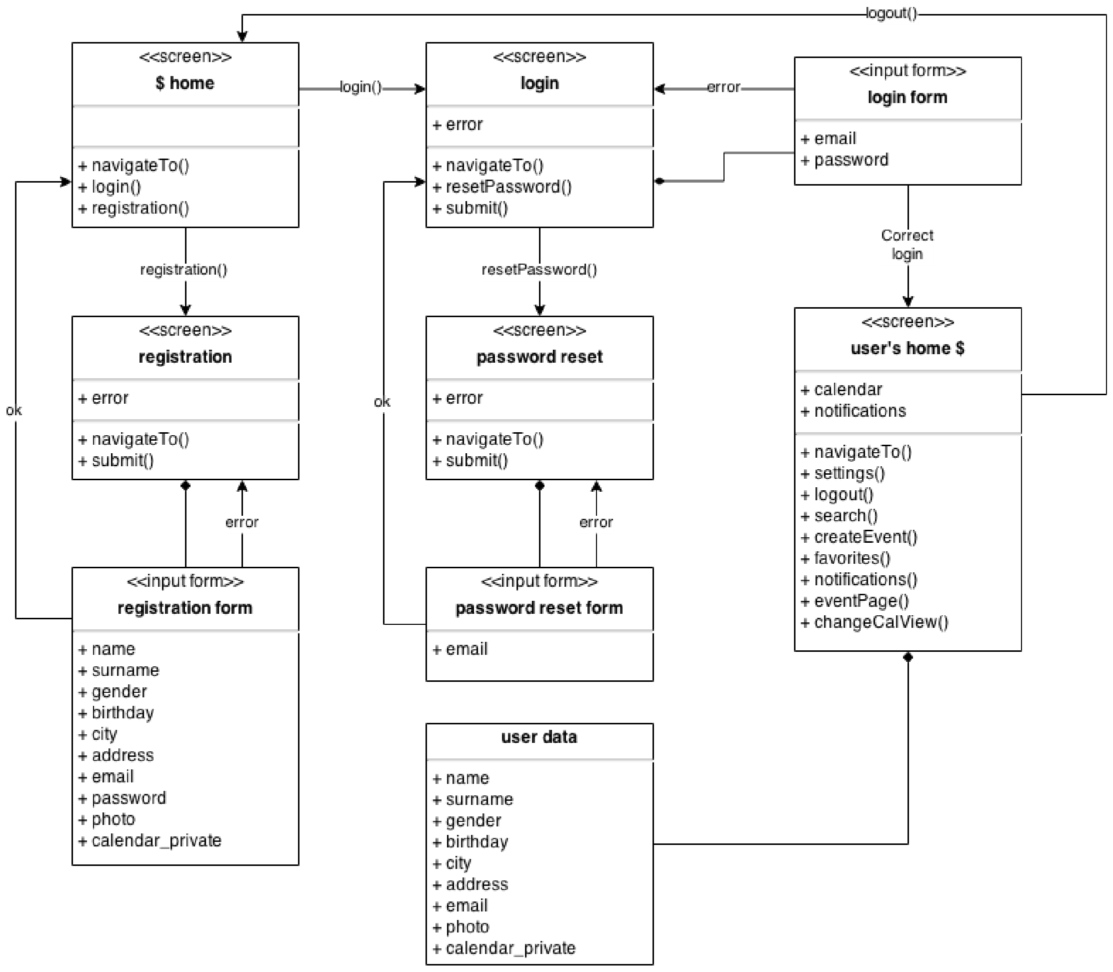

Figure 5: Registration, Login and Password reset UX diagram

When the user enters in MeteoCal, he is redirected to the home screen. Here it is possible to reach the registration screen or the login screen.
Registration screen: it is composed by the form to be filled to become a registered user. If the information provided by the user is not correct (for example type mismatch or already existing user), the screen shows the error, while if everything is fine, the system redirects to the login screen.
Login screen: it is composed by the input form to be filled by the user with authentication information. If the email and the password provided by the user were correct, the system redirects the logged in user to his home page, containing his calendar, his notifications and his personal information. If the authentication goes wrong, the login screen shows a message. From this screen it is reachable the screen to ask a new password: it is composed by a form where the user can enter his registration email to which it will be sent a new password. In case that the inserted email doesn't exist in the system, it is shown a message.

5.3.2 Settings

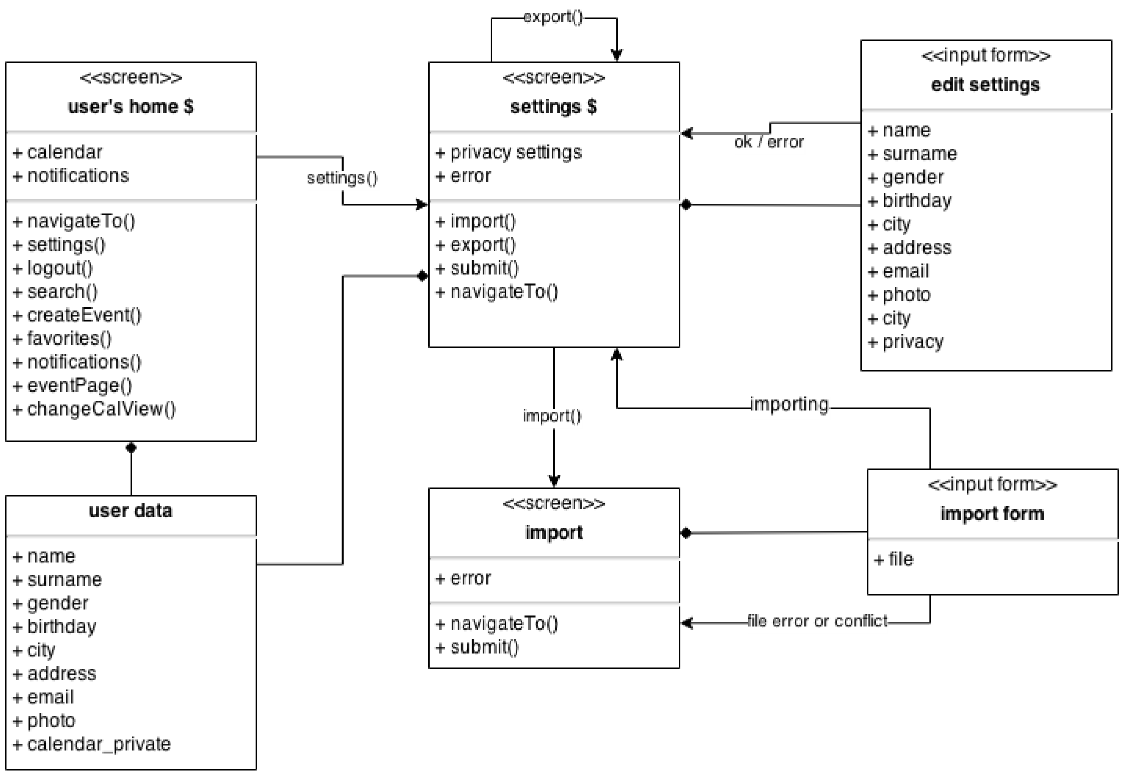

Figure 6: Settings UX diagram

From now on, in all UX diagrams we will show, we assume that the user has already logged in and he is redirected to his home page.
Settings screen contains an input form where it is possible to modify user's information and settings about the calendar visibility. From here it is possible to export the calendar or to go to the import screen where there is an input form where it is possible to select the file to be imported: in case of error, a message on the import screen is displayed, otherwise the user is redirected to the settings screen.

5.3.3 Search

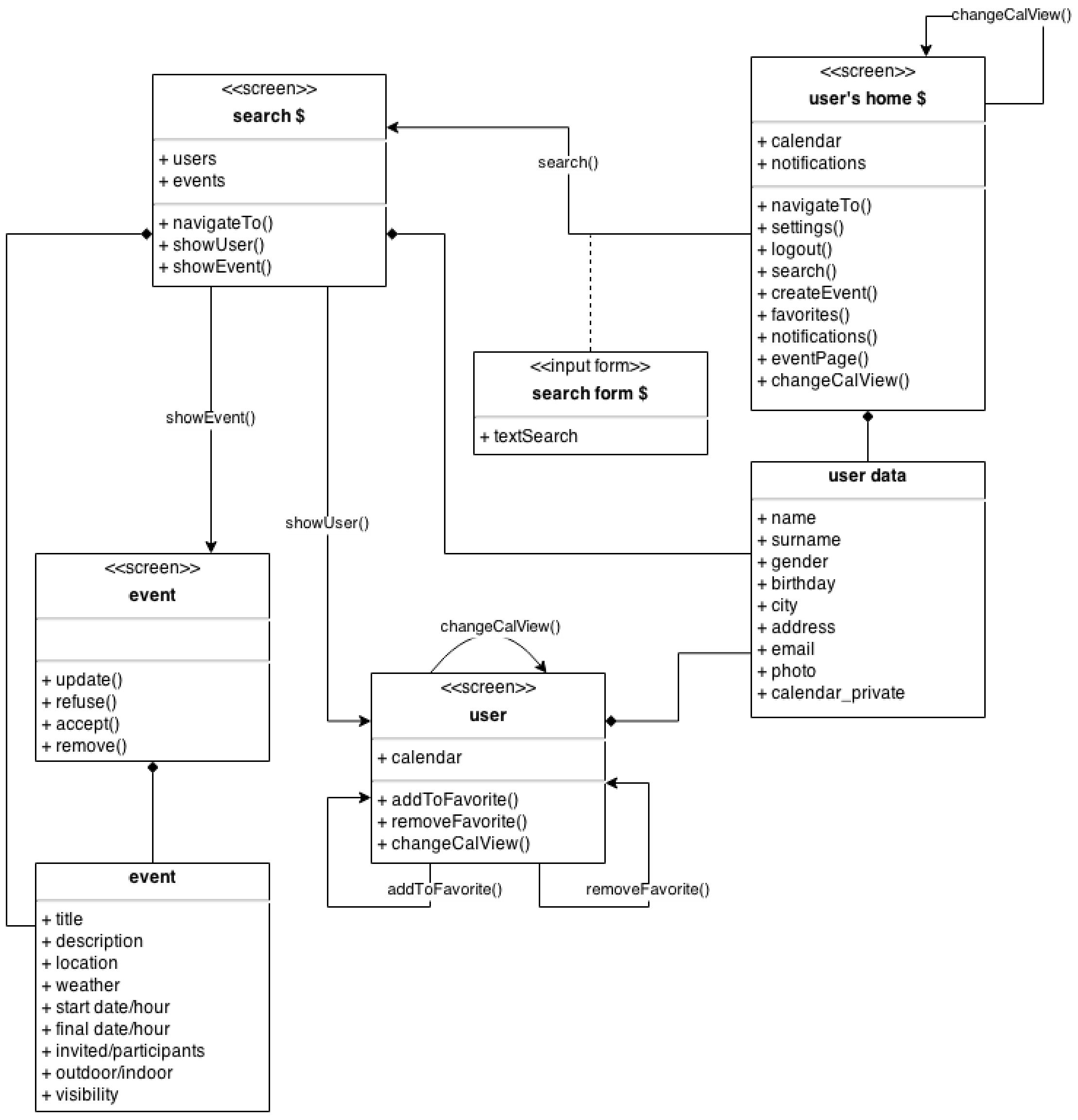

Figure 7: Search UX diagram

In this sub-diagram it is possible to see the User Experience that the system provides to the users about the search of events or users with their calendars.
In the user's home screen and in the user's profile screen there is a method to change the calendar view.
It is possible to reach the search screen by fulfilling the search form provided in every page. The search screen displays the results that can be users and/or events. From here it is possible to reach the user's profile page in which there is his calendar and there is the possibility to add/remove this user to/from favorites. From search screen it is possible to reach a public event page where there is all its information available.

5.3.4 Event creation and update

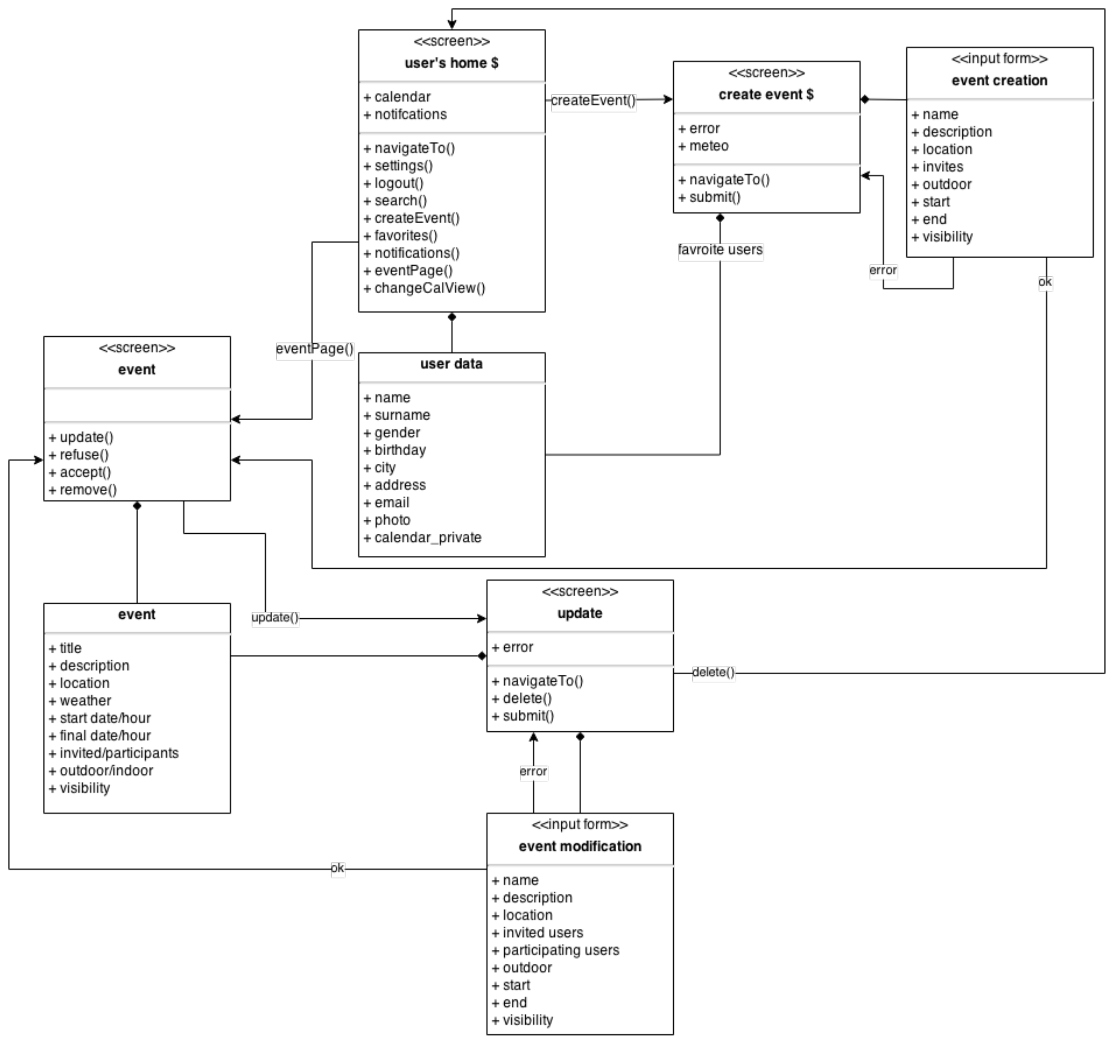

Figure 8: Event creation and update UX diagram

In this part of the UX diagram we start from the user's home to create an event or to update an event shown in user's calendar.
With reference to the creation of an event, from every page for logged in users there is the possibility to reach the event creation screen, containing the input form with all data necessary to create the event. The system also checks the correctness of the inserted data, the overlapping of events for the user, the weather condition: in case of any problem shows an error to the user. If everything is fine, it redirects to the event screen with all the information inserted before.
From the user's calendar contained in his home page, for the organizer user, it is possible to reach the event page where it is possible to reach the update page. Here there is a form to edit event's information (which is checked as for the creation). From the update screen it is possible to delete the event (not to be confused with *remove()* that removes the event from user's calendar, but it is not available to the organizer user).

5.3.5 Favorite users and notifications

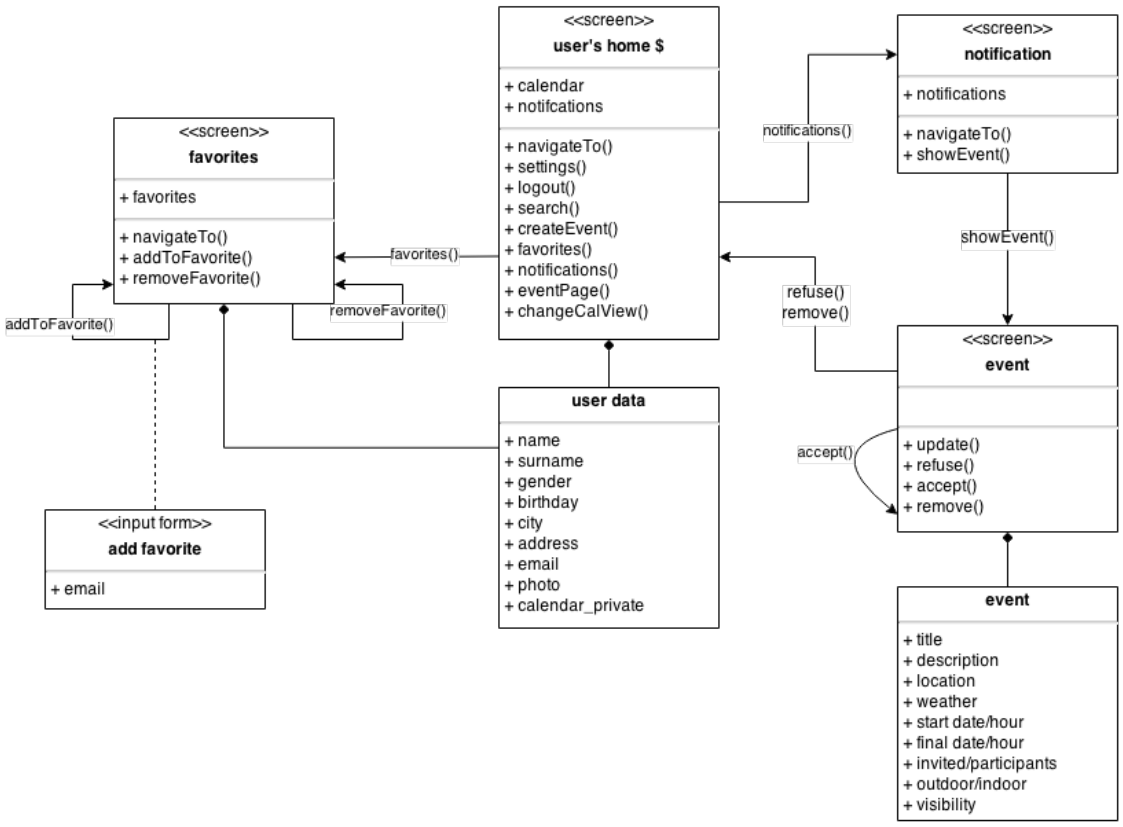

Figure 9: Favorite users and notifications UX diagram

In this part of the UX diagram we show on the left the favorite users screen and on the right the notification screen.
From the favorites screen, reachable from user's home, it is possible to add a user as a favorite by inserting his email in the input form provided, or to remove a favorite user.
In the notification screen are shown user's (recent) notifications. From them, it is possible to reach the screen of the event they are associated to.
In the event screen, when the user receives the invitation, it is possible to accept it or refuse it (in this last case the system returns to user's home). It is also possible, from this screen, a method to remove an event that we have already accepted (it will be removed from user's calendar, but this is not available for the organizer that has to delete the event from the update screen).

6 Boundary Entity Control

6.1 Introduction

We show the Boundary Entity Control diagram in order to give another design schema which is very close to the Model View Controller design pattern which will be used to implement the application.
In fact the boundary stereotype can be mapped to the view, the user interface described above in the User Experience diagram, while the control represents the controller of the application and the entity maps to the model.

Boundaries are derived from the main "sections" of the system, representing the login / registration area, the user's settings area and the main system area. They group together some screens shown in the UX diagram.

All the methods of the screens are written into the appropriate boundaries (maybe some names have been changed to make it more understandable, such as all *submit()* methods to be distinguished). There is a method *showSomething()* for every screen in the UX diagram.
Finally it is important to say that entities do not represent the ER Diagram, but only a conceptual view of the entities used in the BCE.
This time, we decided to give an unique view of the BCE diagram because it is still well-readable and it may help to understand as the whole system will work.

6.2 BCE Diagram

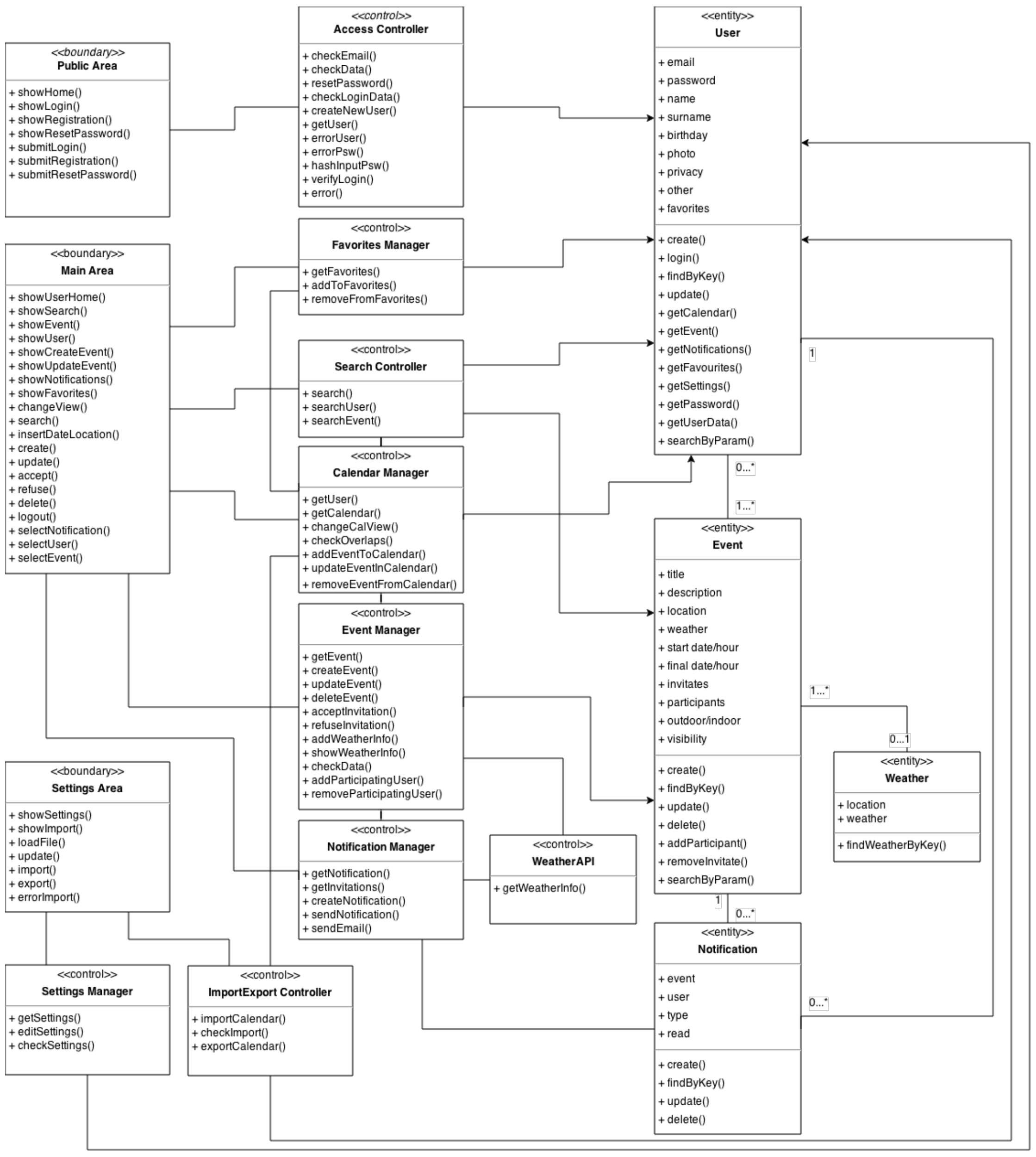

Figure 10: BCE Diagram

We chose to divide the functionalities of the program in more subsystems. To do this we have analyzed the functions to be implemented and we have found three main areas that may represent the cores of the application: Public Area, Main Area, Settings Area. These areas are our boundary classes of the system.

The **Public Area** contains the functionalities that the system offer to a guest user, like registration, password reset and login.
The **Main Area** is the real core of the system. A logged user works mainly in this area and he can use the features related to the calendar: view his calendar, create

events, update events or search an user to view his public calendar.
The **Settings Area**, instead, is the area dedicated to the management of user and calendar settings. This area is accessible by a logged user too, but we have decide to separate that from the main area because the functionalities offered here are not directly related to the main features of the application.

Each area is served by different controllers that provide methods used to reach the goal of the system. We have tried to create more controllers, each dedicated to a subsystem of the application. For example the *Access Controller* will be implement the code for registration and login of users, the *Search Controller* will be provide the search engine for the users/events, the *Calendar Manager* will be provide the methods to manage calendar and events, and so on.

Furthermore the controllers have the main function to communicate with the entity classes, which are nothing more than the entity that we have shown in the ER diagram (it's not the same model, so the representation may result quite different).

7 Sequence Diagrams

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.

7.1 Log in

Here we show a sequence diagram where a user attempts to log in.

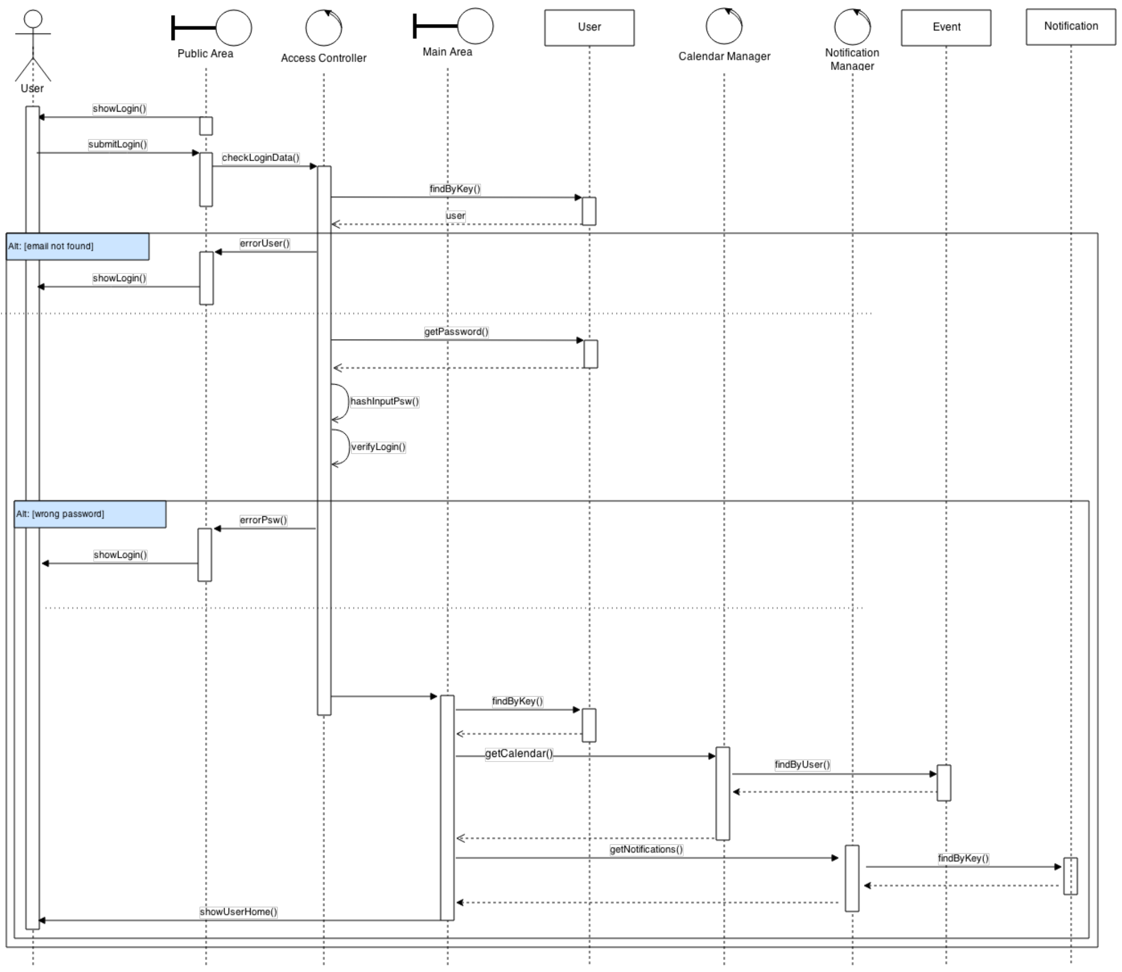

Figure 11: Login Sequence Diagram

7.2 Registration

Here we show a sequence diagram where an unregistered user attempts to register to MeteoCal.

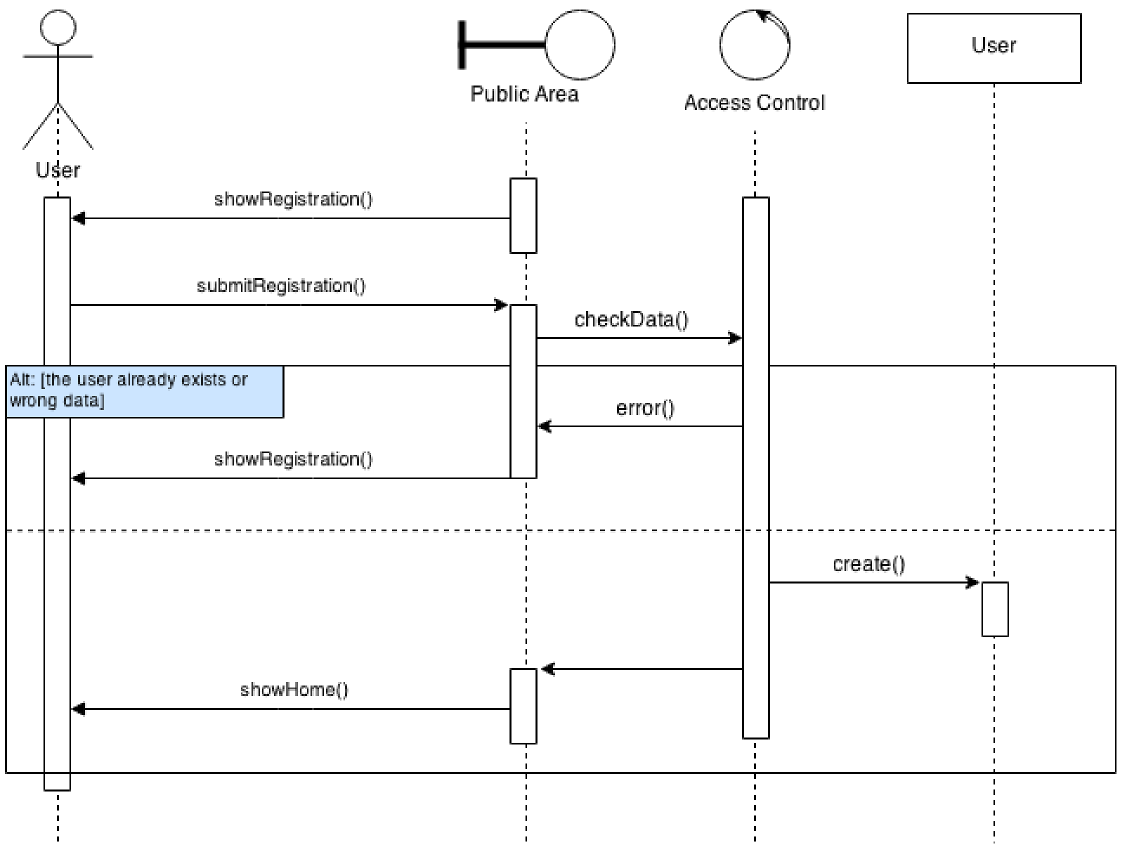

Figure 12: Registratrion Sequence Diagram

7.3 Search

Here we show a sequence diagram where a user searches for a public event or the calendar (profile) of another user.

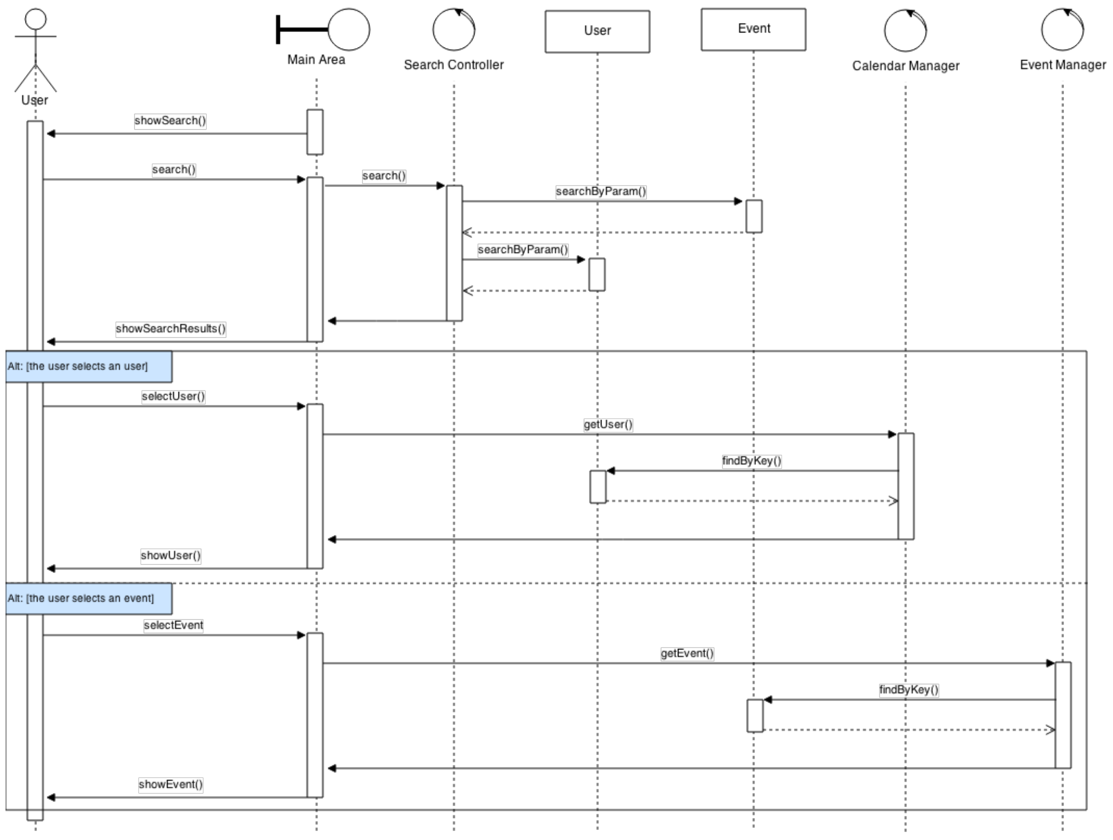

Figure 13: Search Sequence Diagram

7.4 Creation of an event

Here we show a sequence diagram where a user creates an event. We have highlighted that whenever he inserts the date/time and location attributes, the system updates the weather in order to propose the closest sunny day to the user. We also show how the system avoids the overlapping of events and the creation of notifications for invited users.

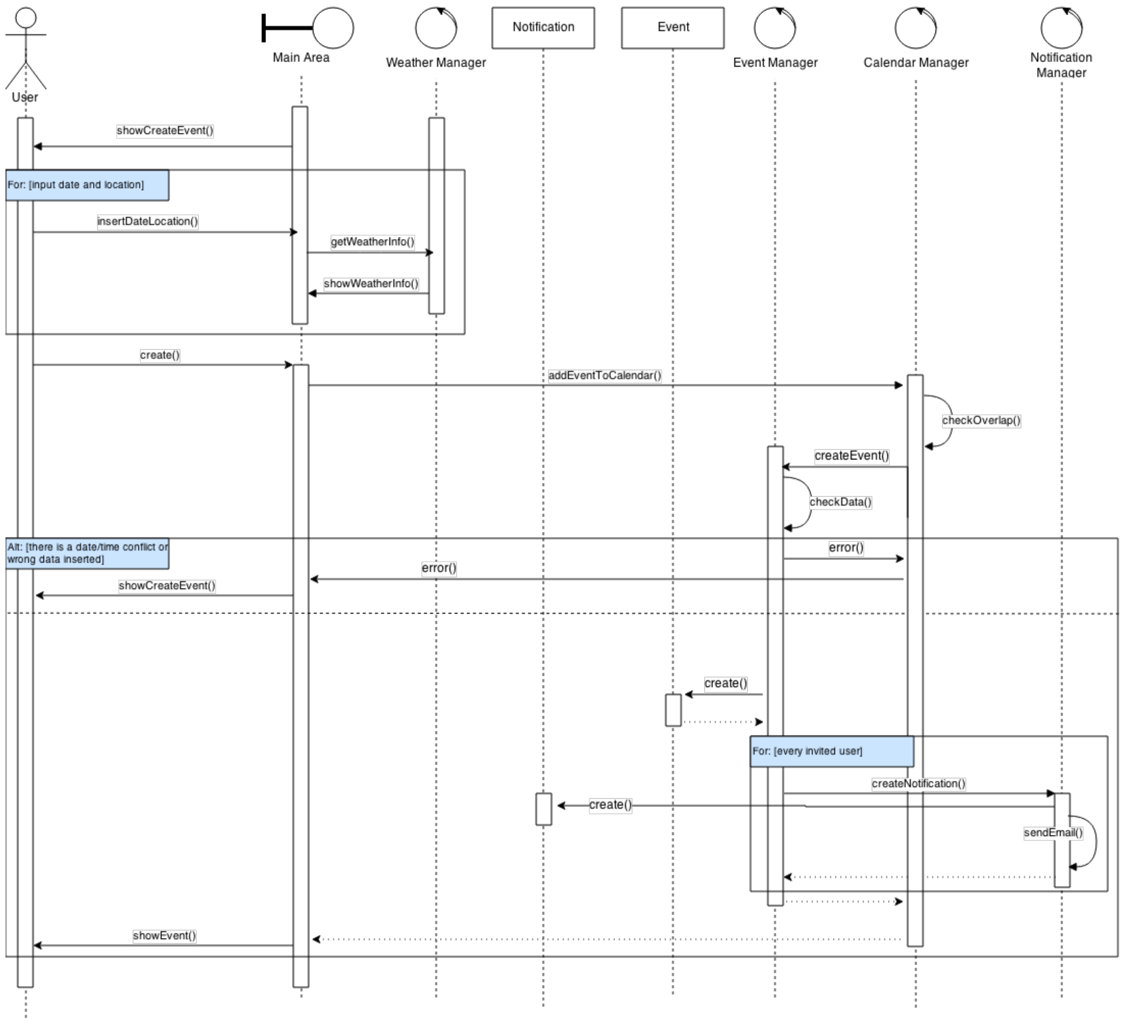

Figure 14: Event Creation Sequence Diagram

7.5 Updating of an event

Here we show a sequence diagram where a user updates an existing event.

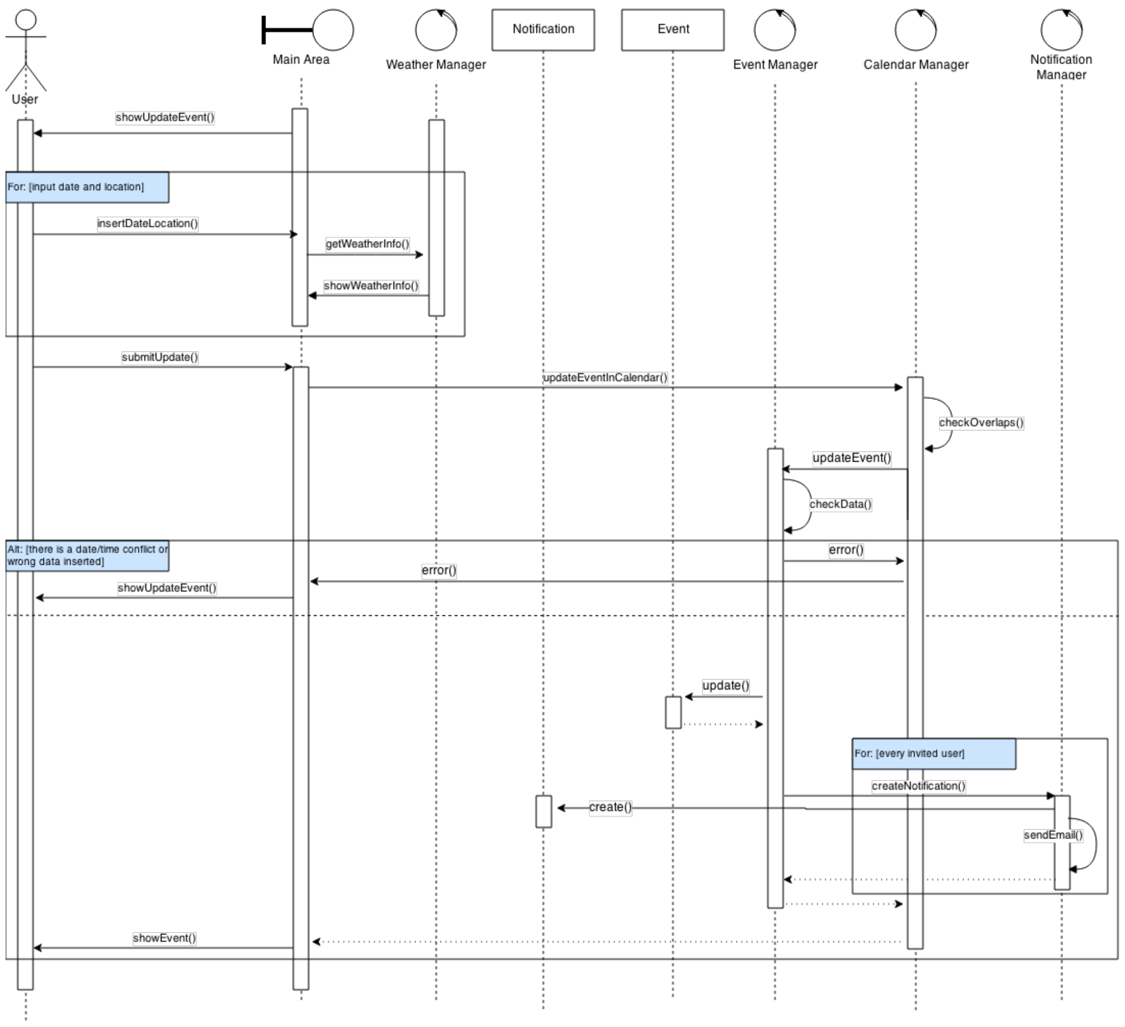

Figure 15: Event Updating Sequence Diagram

7.6 Acceptation of an invitation

Here we show a sequence diagram where a user accepts an invitation to join an event from the notification.

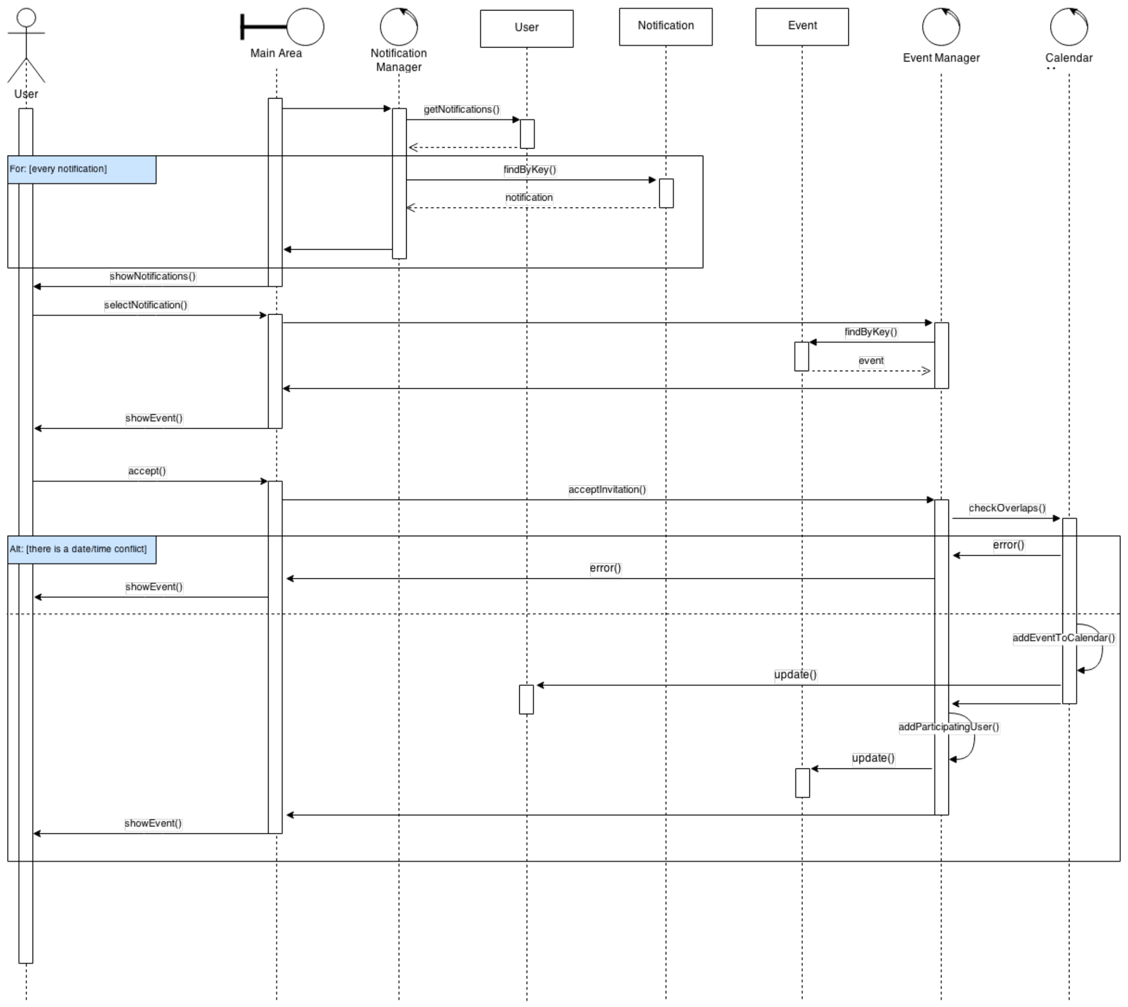

Figure 16: Acceptation of Invitation Sequence Diagram

7.7 Refutation of an invitation

Here we show a sequence diagram where a user refuses an invitation to join an event from the notification.

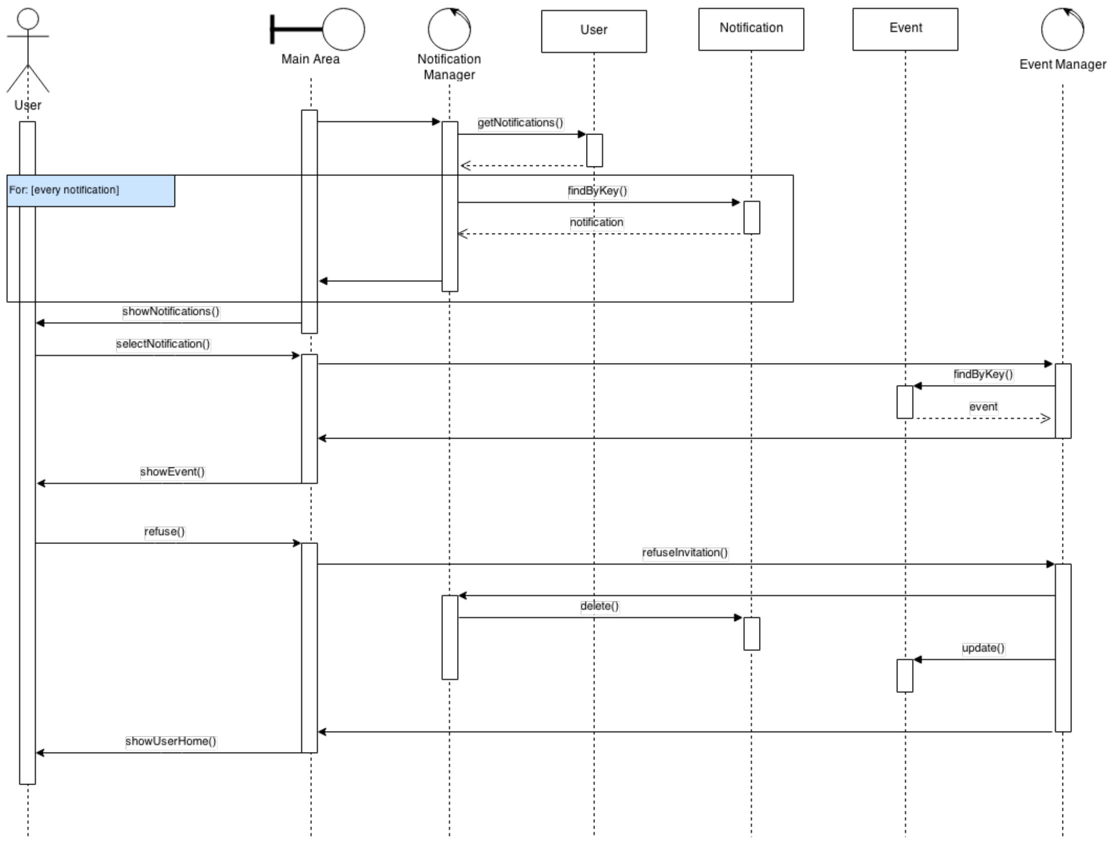

Figure 17: Refutation of Invitation Sequence Diagram

7.8 Import of event

Here we show a sequence diagram where a user imports a calendar and the system checks the overlapping of other events.

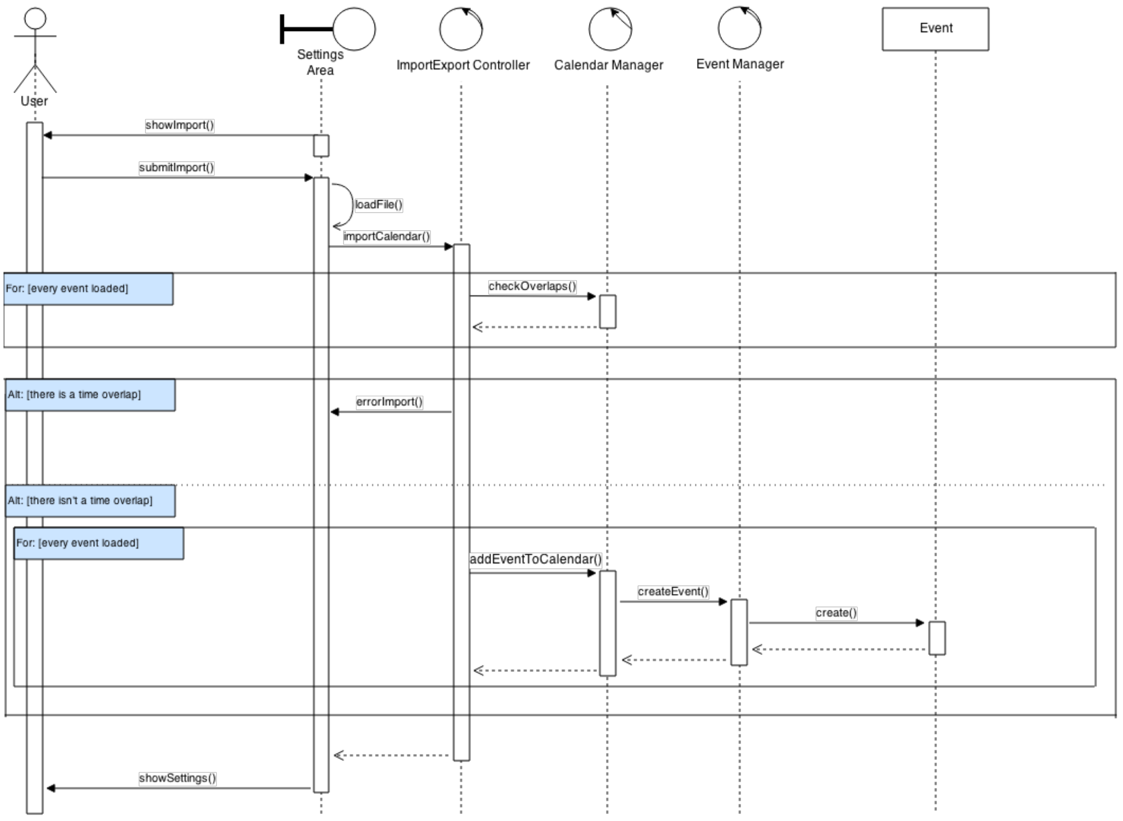

Figure 18: Import Sequence Diagram