MeteoCal Project

Design Document

Authors:
Claudio Sanna
Walter Samà

October 2014 - January 2015

Contents

Chapter 1

Introduction

In this document we will focus our attention on the architecture of the MeteoCal Application; we will analyze the software focusing on the structure of the application, the user side of the application, the logic system that stays at the core of the program and also the composition of the database that we will use. In order to do what we said we will present:

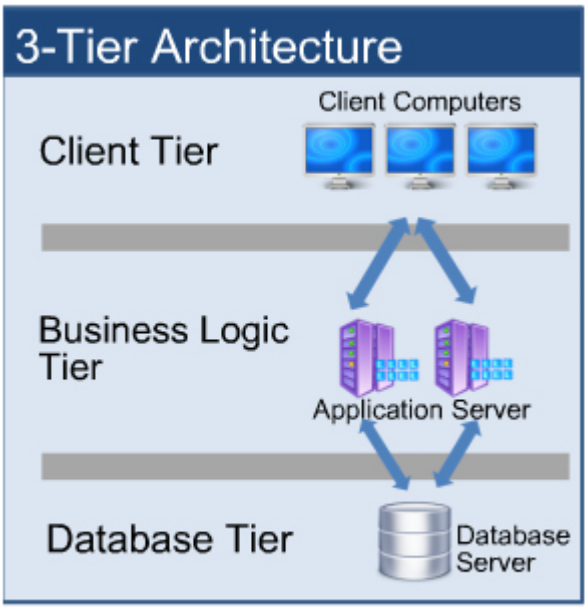
- System architecture, in order to show how we will design the client-server-database system.
- Entity-Relationship diagram with Conceptual and Relational Schema, that will help us to design the database's tables and show connection between the various element.
- Conceptual and Logic view of the database, with the actual tables that we will use for the database specifying the key.
- UX Diagrams, that will be useful to clarify the interaction between the users and the system.
- BCE Diagrams, that will extends the UX diagram introducing the system logic and the connection to the database.
- Sequence diagram, here we will propose some other diagrams and then we clarify and explain in details how the control of the correctness of the data will work.

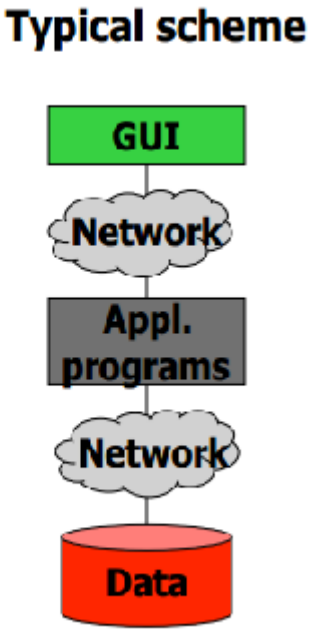
Chapter 2

Architecture Considerations

2.1 Architecture Description

After some considerations we decide to use a standard 3-Tier architecture with a very thin client because we will develop a web based application and the client will only use a web browser to access to it. In the lectures that we follow about architecture we also find out about the possibility to use a service-based system but we don't have much experience and maybe is not the right choice for our application.





In the next page we will tell more about the 3 tier architecture.

- **Client Tier**: A logical layer that represents a local computer, this tier translate the user action in something our application can understand; it also receive the data from the application and it has to translate them into something the user can understand, in our case we will present the result using the user's browser.
- **Application Server Tier**: The logical layer between a user interface or web client and the database. This is typically where the Web server resides, and where business objects are instantiated. The application tier is a collection of business rules and functions that generate, processes and operate upon information. These rules, which can change frequently, are thus encapsulated into components that can be physically separate from the application logic itself.
- **Database Tier**: A logical layer that represents a computer running a DBMS, such as a SQL Server database, it contains all the information that are needed by the users and the application.

All the logical layer that we presented could be in one or more physical machines, depending on how we chose to separate them. In our case we will have all these tiers separate:
Our users could be in every part of the world with separate machine, or maybe they can also have a PC in common.
We don't know very well where the application tier will be, it could be on our computer that will works as a server, or maybe in another server.
The same considerations above could be taken into account for the database tier.

The Tools that we are going to use in the implementation are these:
- **Client Tier** In order to use our application a web browser is needed.
- **Application Tier (Business Logic Tier)** We will use Java Enterprise Edition 7(JEE7) platform that provide enterprise services in the Java language to build our application.
  With Enterprise Java Beans (EJB) we will capture the logic that solves or meets the needs of a particular business domain and persistence entities.
- **Database Tier** We will use MySQL Server to manage the database.

Obviously we will use an IDE to implement all these tiers and the three layer will be built separately.

2.2 Division of the System into Subsystem

In this section we will extract the main functions of our system into different subsystem that will allow us to explain in a better way how the system work incrementing the clarity of the diagrams:

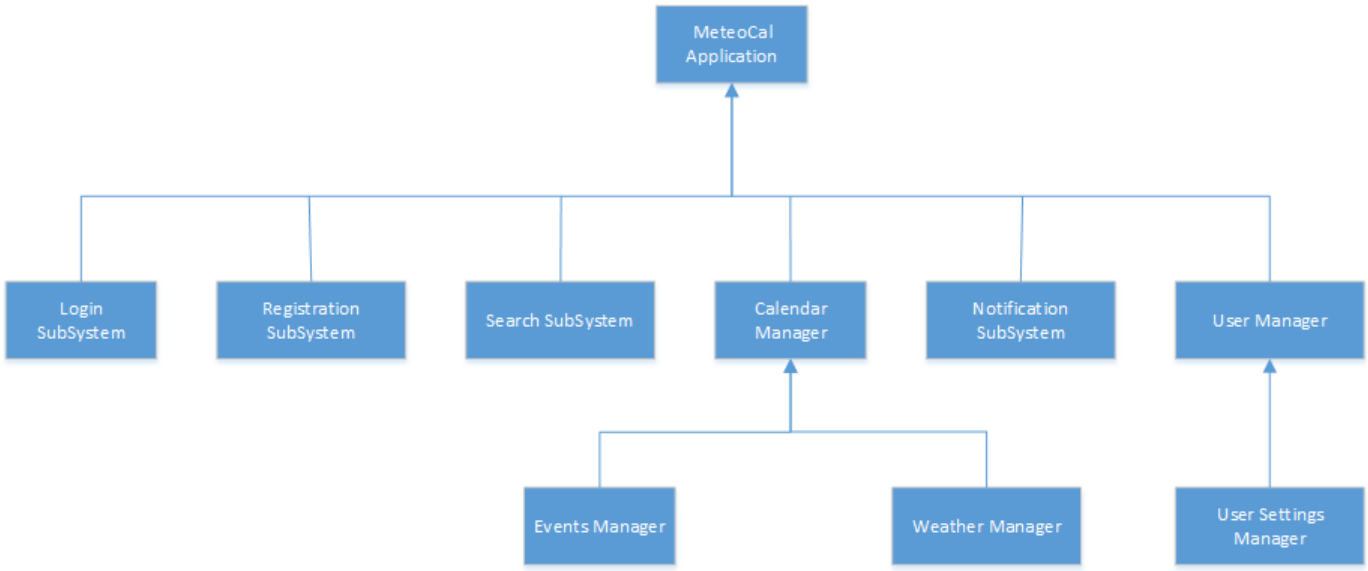


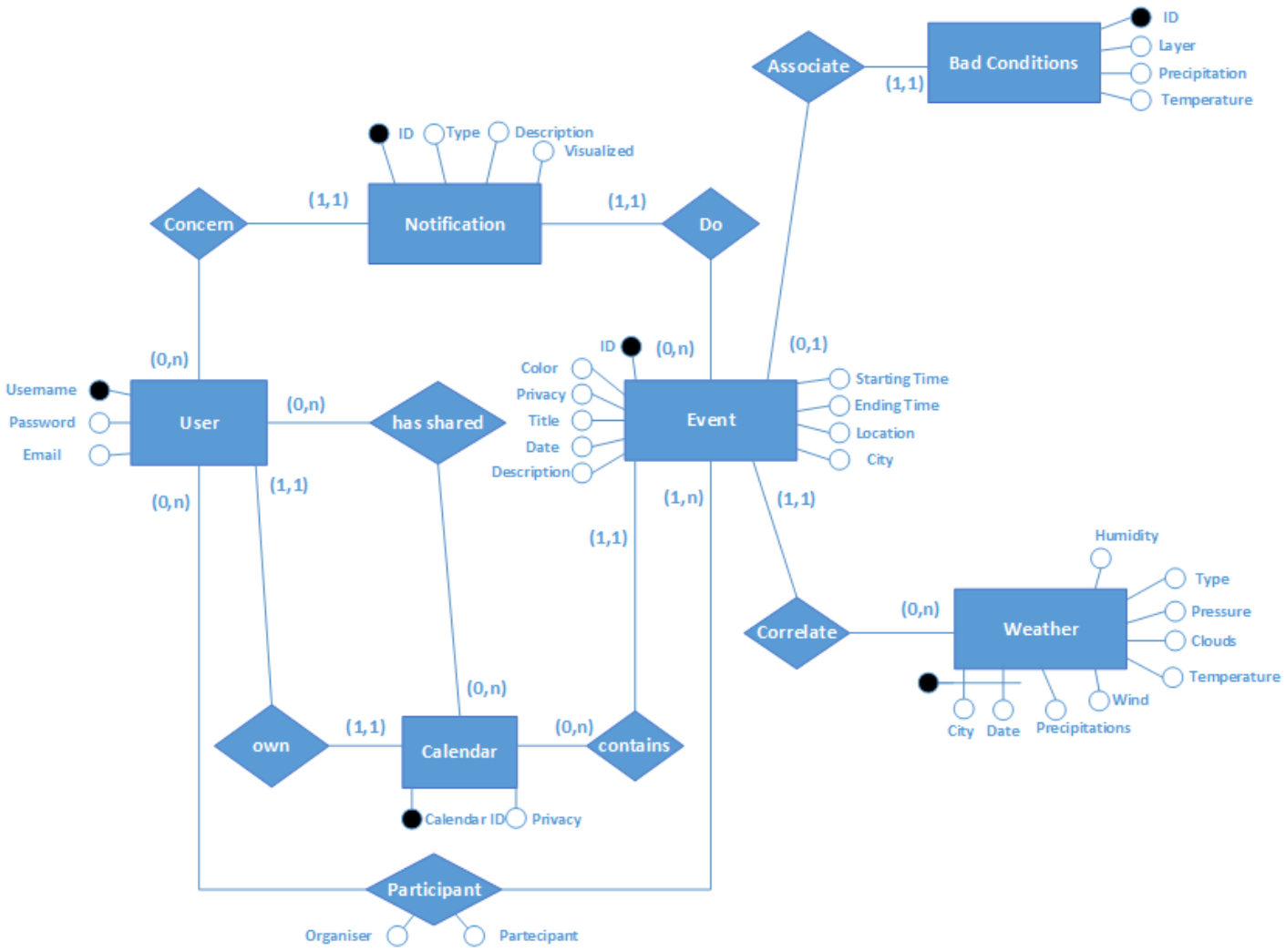
Figure 2.1: Division into Subsystem

Chapter 3

Database management

3.1 Conceptual design

In this section we present the ER diagram of the database that we will implement for the application. We decided to use these entity : *user, notification, event, calendar, bad conditions, weather.*
User will have all the essential information about an user and will be used to check the login and all other functionalities which involves users' information; they key that will be used for this entity is the "Username" so that it will be unique; a constraint on the email will also be set in order to not have the same email for different user name.
Calendar is the entity that connects an event to its owner user; it also contains the information about the calendar "privacy"; this type will be a sort of enumeration type that can only have the values shared, private, public. The key here is a simple integer ID.
The relations **has shared** in the diagram refers to the possibility to share a calendar (as explained in the RASD document), and associate a calendar to one or more users.
The **Notification** entity presents the information of a notice in the attribute "description"; a Boolean type "visualized" will indicates whether if a notification is already seen or not by the user; the "type" attribute is an enumeration one and will indicate the nature of the notice; possible values are: update, invited, accept/decline, badConditions, system. Also in this case an integer ID is used as the key. Finally this entity will store information about the event it refers to and the user owner, as indicated by the relationship "do" and "concern". In case of notification with no event (e.g. system notification) a default value will be set.
Event entity store all the information needed by an event as specified in the RASD document; also in this case the key of the entity is an integer ID. **Event** has many relations with various entity : **do**, associate and invite will be used in a reverse way, so that means that the other element of the relation will have the information of the event that it refers to and no viceversa. For **Calendar** and **Weather** entity the Event will have a direct connection to them including the Calendar ID and presenting among the attributes the city and date that refers to the weather.
Bad conditions contains information about the bad weather settings set by the event creator, as specified in the RASD document. Still an integer ID is used as a key.
Finally we decided to introduce a **Weather** entity that will store the main information of weather; we think that in order to make the application more reliable we will store the information taken by the weather site where we will get the weather forecast. Not all the information will be taken but only the main ones that will be show.
Now we present the schema:


Associate
Bad Conditions
ID
Layer
Precipitation
Temperature
(1,1)
ID
Type
Description
Visualized
Concern
(1,1)
Notification
(1,1)
Do
(0,n)
(0,1)
Username
Password
Email
User
(0,n)
has shared
ID
(0,n)
Color
Privacy
Title
Date
Description
Event
Starting Time
Ending Time
Location
City
(0,n)
(1,1)
(1,n)
(1,1)
(1,n)
Humidity
Correlate
(0,n)
Weather
Type
Pressure
Clouds
Temperature
Wind
City
Date
Precipitations
(0,n)
own
(1,1)
Calendar
(0,n)
contains
Calendar ID
Privacy
Participant
Organiser
Partecipant

3.2 Logical design

3.2.1 From ER to logical structure

In this part we will show the decision that we made to transform the relation and the entity into the final logical structure of the database.
As normal all the entity were transformed into a table and the relations, due to the cardinality, were transformed as follows:

concern : it is transformed into an attribute called *user* incorporated in notification.

own : it is transformed into an attribute called *owner* incorporated in calendar.

hasShared : it is transformed into a new entity called *SharedUsers* with the attribute "username" and "calendarID" that refer respectively to User and Calendar entity.

participant : it is transformed into a new entity called *Participant* with the attribute "user" and "event" that refer respectively to User and Event entity. It also contains the attribute "organiser", which is a Boolean and indicates if the the user is the organiser of the event or not and "participant", which is an enumeration attribute which can have the value accepted, declined and no response. This table is created for each event (even with no invitations) as it contains the organiser.

contains : it is transformed into an attribute called *calendar* incorporated in Event.

do : it is transformed to inan attribute called *eventID* incorporated in Notification.

associate : it is transformed into an attribute called *eventID* incorporated in Bad Conditions.

correlate : it is already incorporated in event, as it already have the city and date attribute. This has been made because a weather is not specified for every event or the weather may not be present, but the attribute are always there with eventually a null value.

3.2.2 Logical structure

Here is the logical database structure :

User(<u>username</u>, password, email)
SharedUsers(<u>username</u>,<u>CalendarID</u>)
Calendar(<u>ID</u>, owner, privacy)
Notification(<u>ID</u>, user, type, description, eventID, visualized)
Event(<u>ID</u>, title, date, startingTime, endingTime, location, city, description, color, privacy, calendar)
Participant(<u>user</u>,<u>event</u>, organiser, participant)
BadConditions(<u>ID</u>, eventID, layer, precipitations, temperature)
Weather(<u>city</u>,<u>date</u>, temperature, precipitations, pressure, humidity, wind, clouds)

Chapter 4

User experience

In this section, we will explain with various Ux Diagrams, the interaction between the users and the system, and also we will tell how the system will work.
We will present various schema of every function in order to keep them more understandable; references thought will be displayed with a * that means that a screen or input forms refers to one in another schema and has already been presented.

4.1 Ux Registration,Login and Home page

In this section we present the home page and the main page of the users page. The first page that a guest encounter is the guest home, where the guest can perform login or signup (through a new screen).
The user home contains the connection to the main functions of the application (except event) such as settings, notifications, search, log out and present the calendar,in a month view, with a brief indication to each event (name, starting time and place); the others two screens, day view and week view, are similar to the user page and present the same features, only the view of the calendar (as the screen say week and day and week view) change and are not reachable with the home button.
The day view is the only one where event can be managed and it will be presented in a more detailed way later.

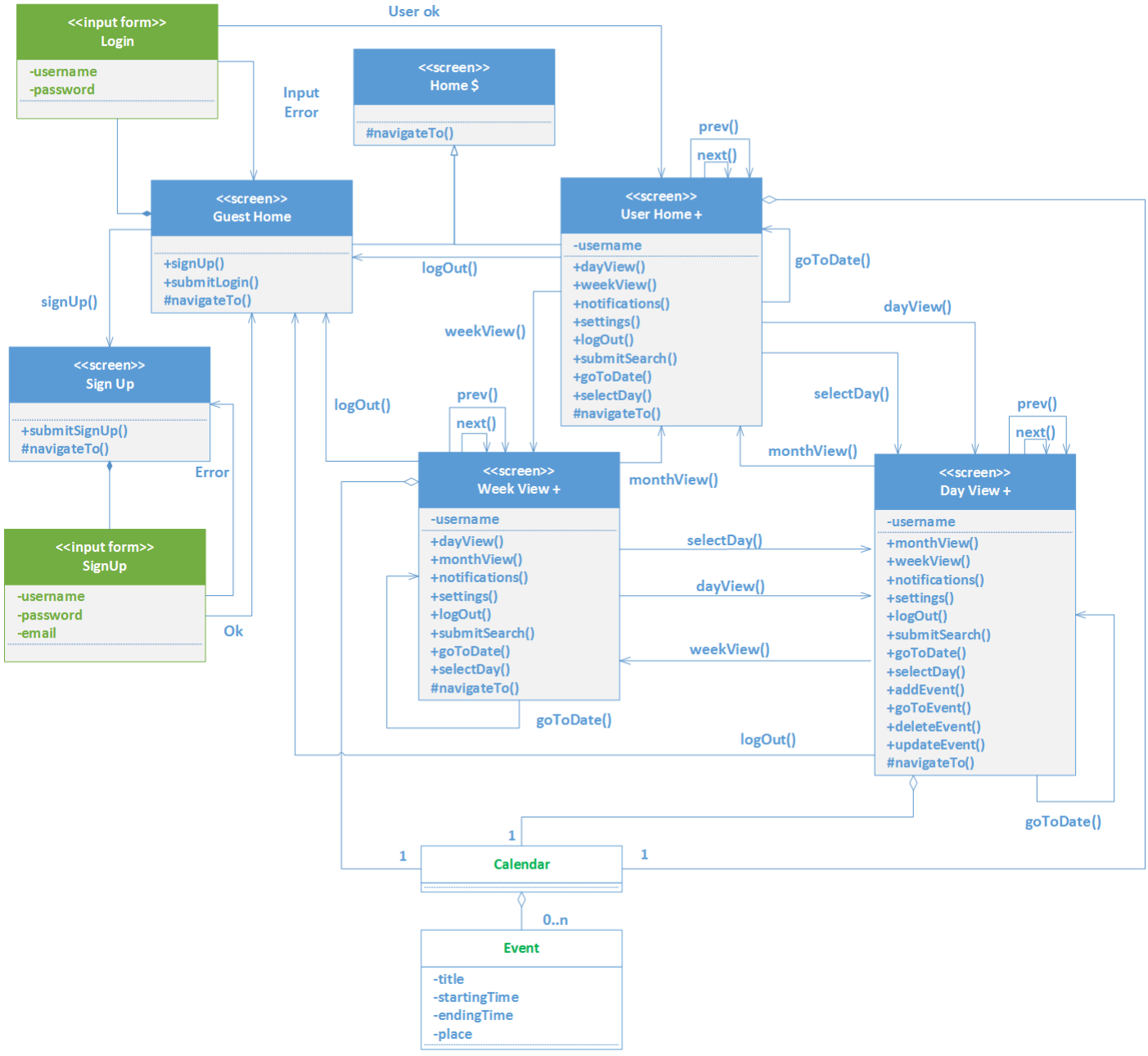


Figure 4.1: Ux Diagram for Registration, Login and User's Home Page subsystem

4.2 Ux Notifications

In this section we present the notifications page. This page can be reached from the home page, the day and week view, and the settings page and viceversa. Two type of notification are presented : the system notification that have only a text, and notification, presented as Notification Event (that contains update, invited, accept/decline, badConditions notifications), which present a connection to the event they refer to.

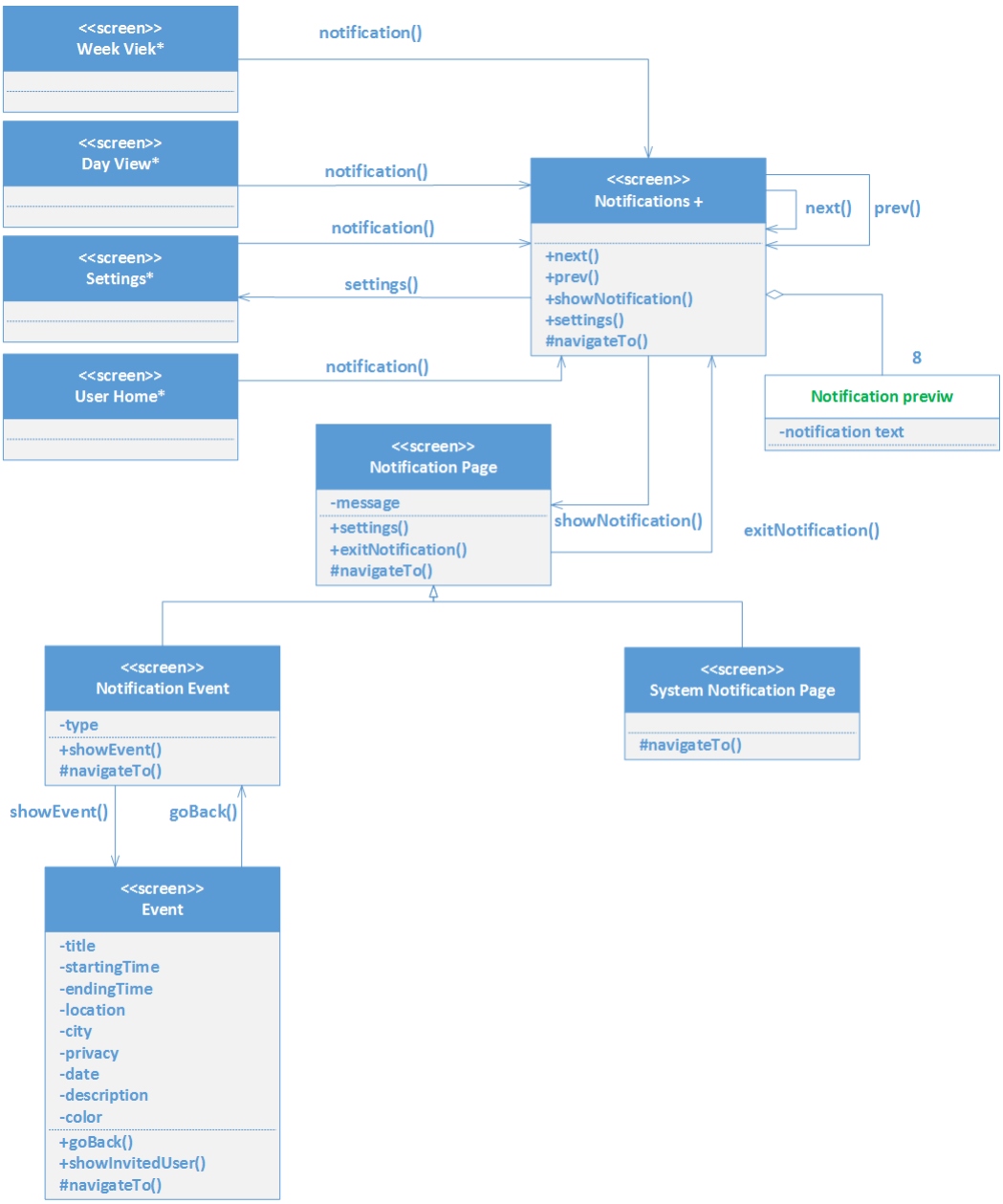


Figure 4.2: Ux Diagram for the Notifications subsystem

4.3 Ux Settings

In this section we present the settings page. This page can be reached from the home page, the day and week view, and the notifications page and viceversa. In this page all the information of the user are shown (email, shared users, calendar privacy) with a form to change them and also a form to change password.

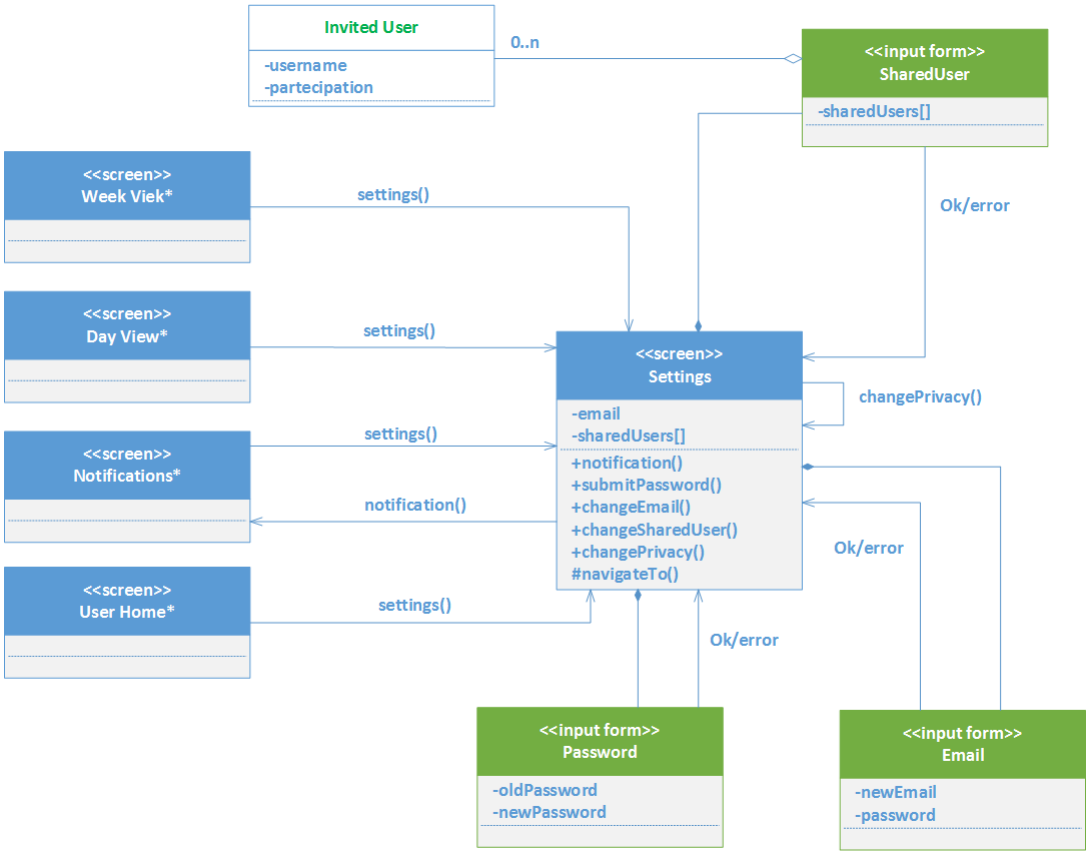


Figure 4.3: Ux Diagram for the User's setting subsystem

4.4 Ux Events

In this section we present the various pages that refers to event managing. All the functions of this type can be performed on the day view screen; from that page three different action can be performed: update, view or add an event. Each of them will open a different screen.
Add event will open a new screen with an input form with all the field for each element that an event has; view event will show the event page with all the information; update event will show a page similar to the Event one, but so that all the information can be changed.

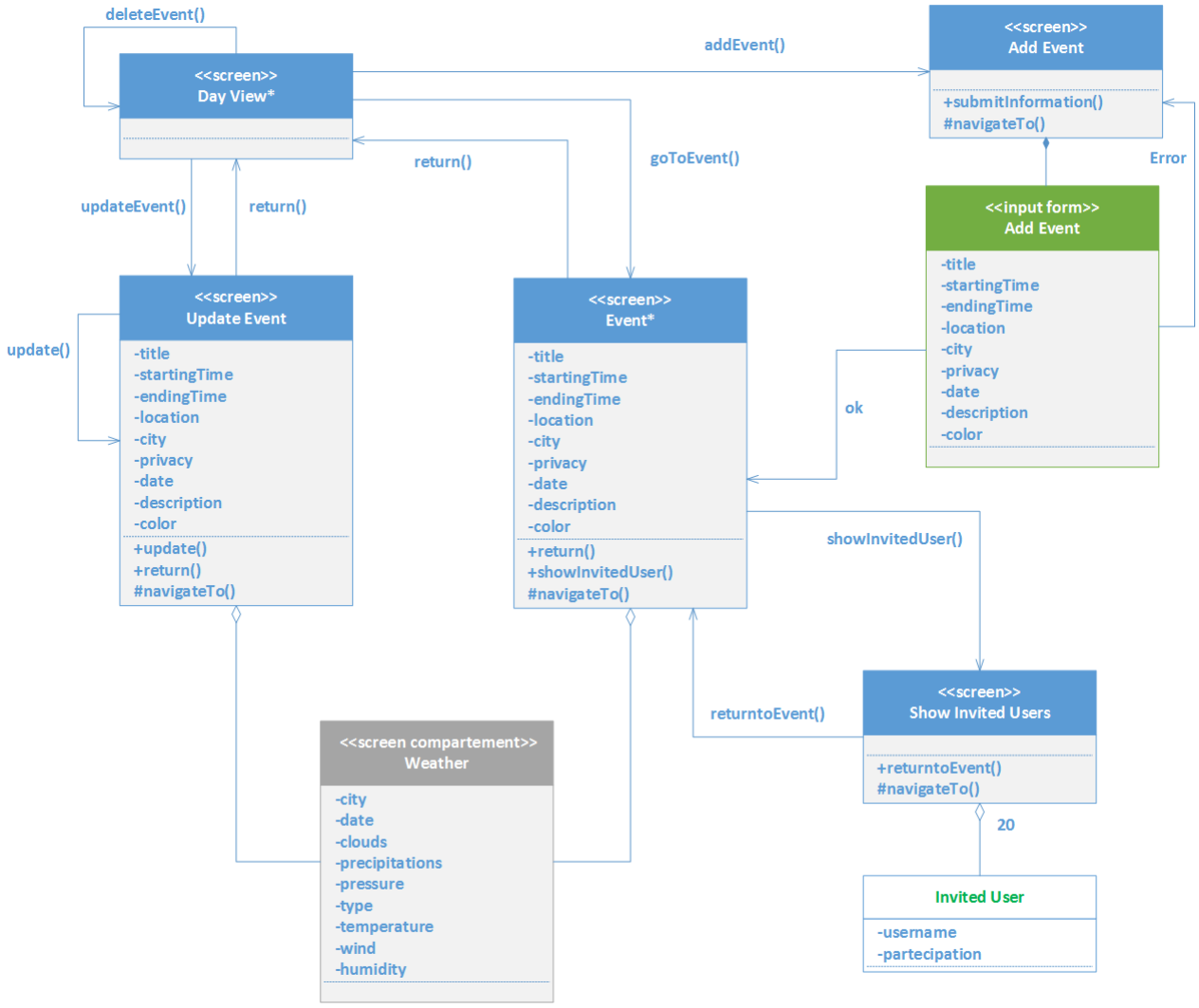


Figure 4.4: Ux Diagram for the Events subsystem

4.5 Ux Search

In this section we present the search page that is generated using the search. All the calendar views (user home, week view, day view) have an input form where the search can be performed; when a search is made a new page will be open and different output can occur: an error message that informs the user that the searched user/email doesn't exist, an error message that informs the user that he has not the access to the calendar or the calendar itself if no error occurs.
From the result page the user can access to the calendar that presents three different pages that are similar to the day view, week view, user home pages presented before; the action that can be performed however are limited and consist in navigate thought the calendar and the events if they are set to public.

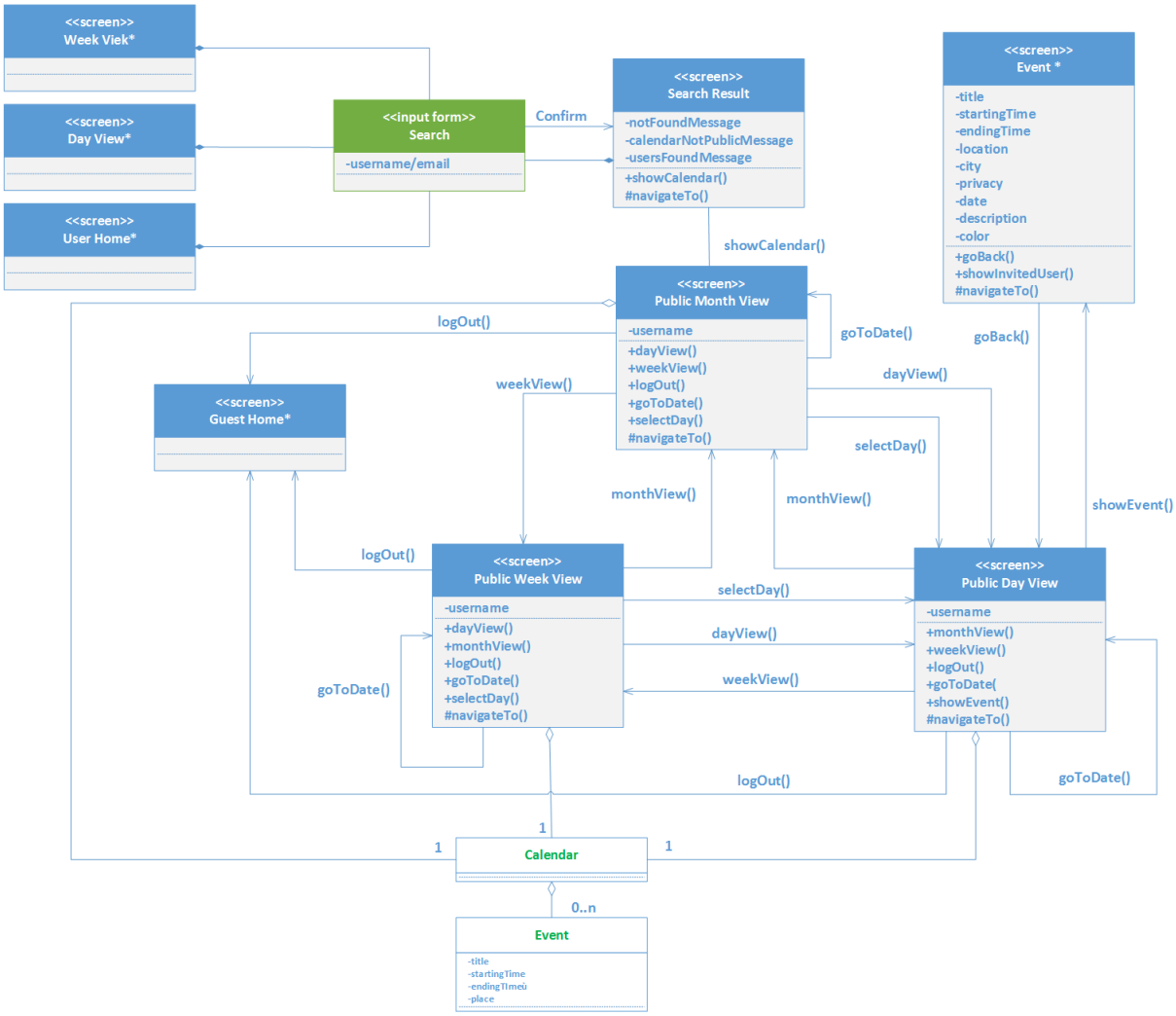


Figure 4.5: Ux Diagram for the Search subsystem

Chapter 5

BCE Diagrams

In this section, we will explain using BCE diagrams, the interaction between the different parts of the internal application. We will present before the organization of all the entities, in order to make more clear the next schema (they will not present the connection between all the entities as they are presented in the entities section).

5.1 BCE Entities

Here is presented the whole entities schema; it reflects the relations shown in the ER and all the schema presented in the logical view.

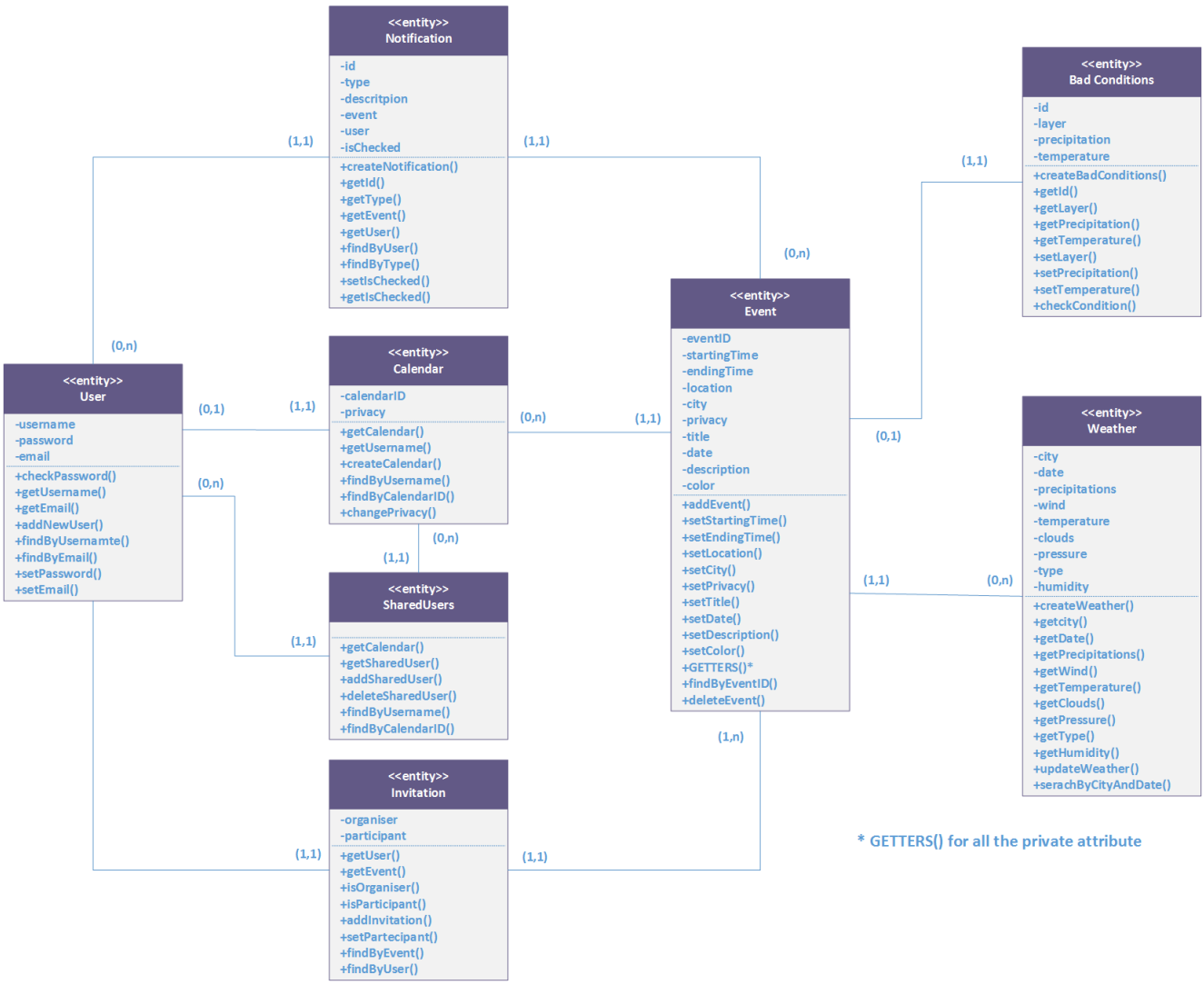


Figure 5.1: BCE Entities that will be used in other diagrams

5.2 BCE Main Page Management

In this section the main page (user and guest) structure is shown. There is the system structure of the login and signup manager and the main functions of all the calendar pages.

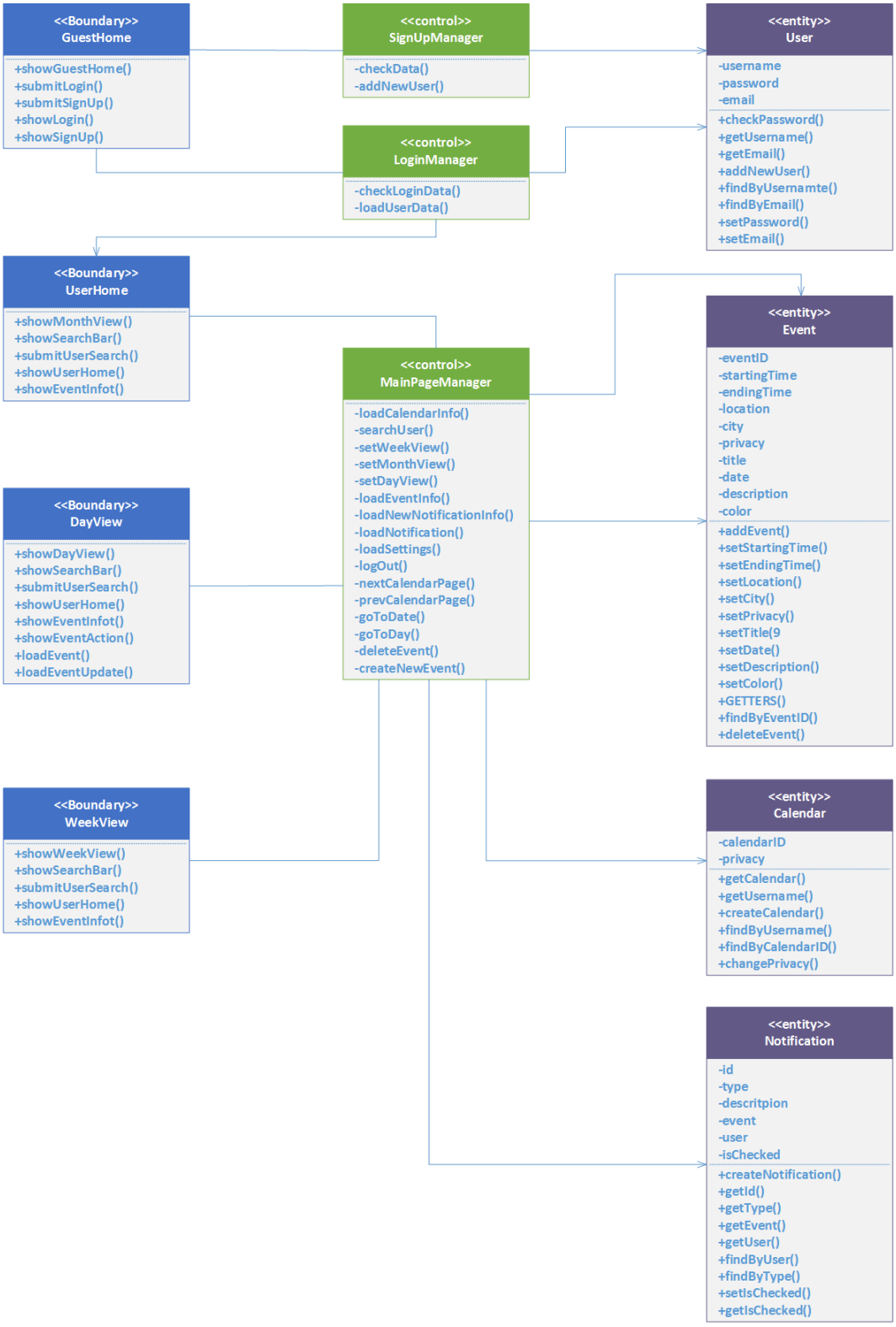


Figure 5.2: BCE diagram for the main page management

5.3 BCE Events Management

In this section we present the event pages structure. The event info manager manages all the functions that regards every event; it has all the methods to load and show all event info, to update the and to create or delete events.

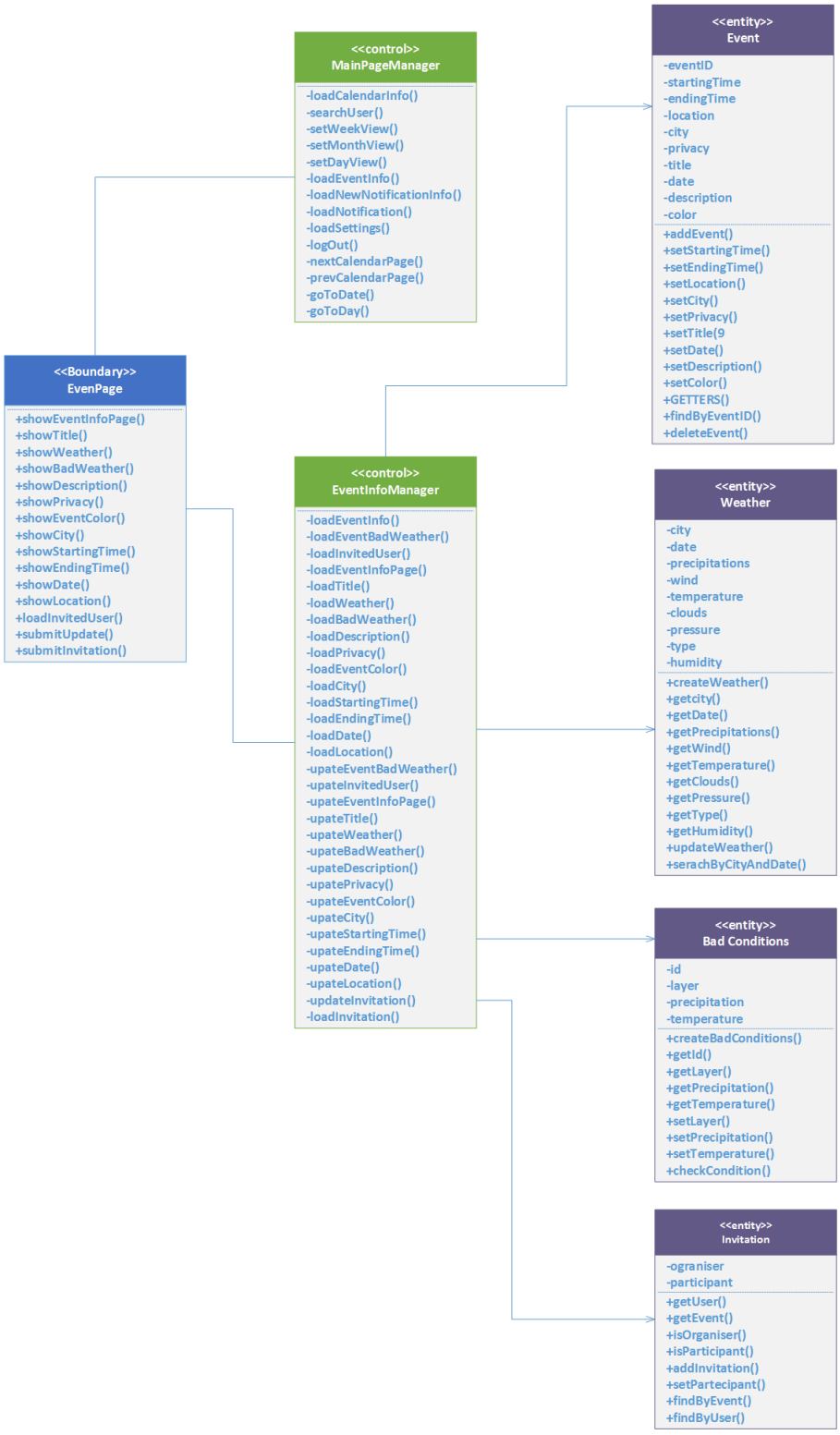


Figure 5.3: BCE diagram for the events management

5.4 BCE Notifications and Settings

We will explain what these Subsystem will do and then, in order to give more space to the diagrams, we will draw them.

5.4.1 BCE Notifications Management

In this section the notification system structure is shown. Two controller manages the loading and showing of respectively events and notifications; the notification manager, that has no direct interact with the users, is the part of the system that manages the notification creation and sending to every users.

5.4.2 BCE Settings Management

In this section we present the settings system structure. The control manages all the functions to load and update all the information about the user.

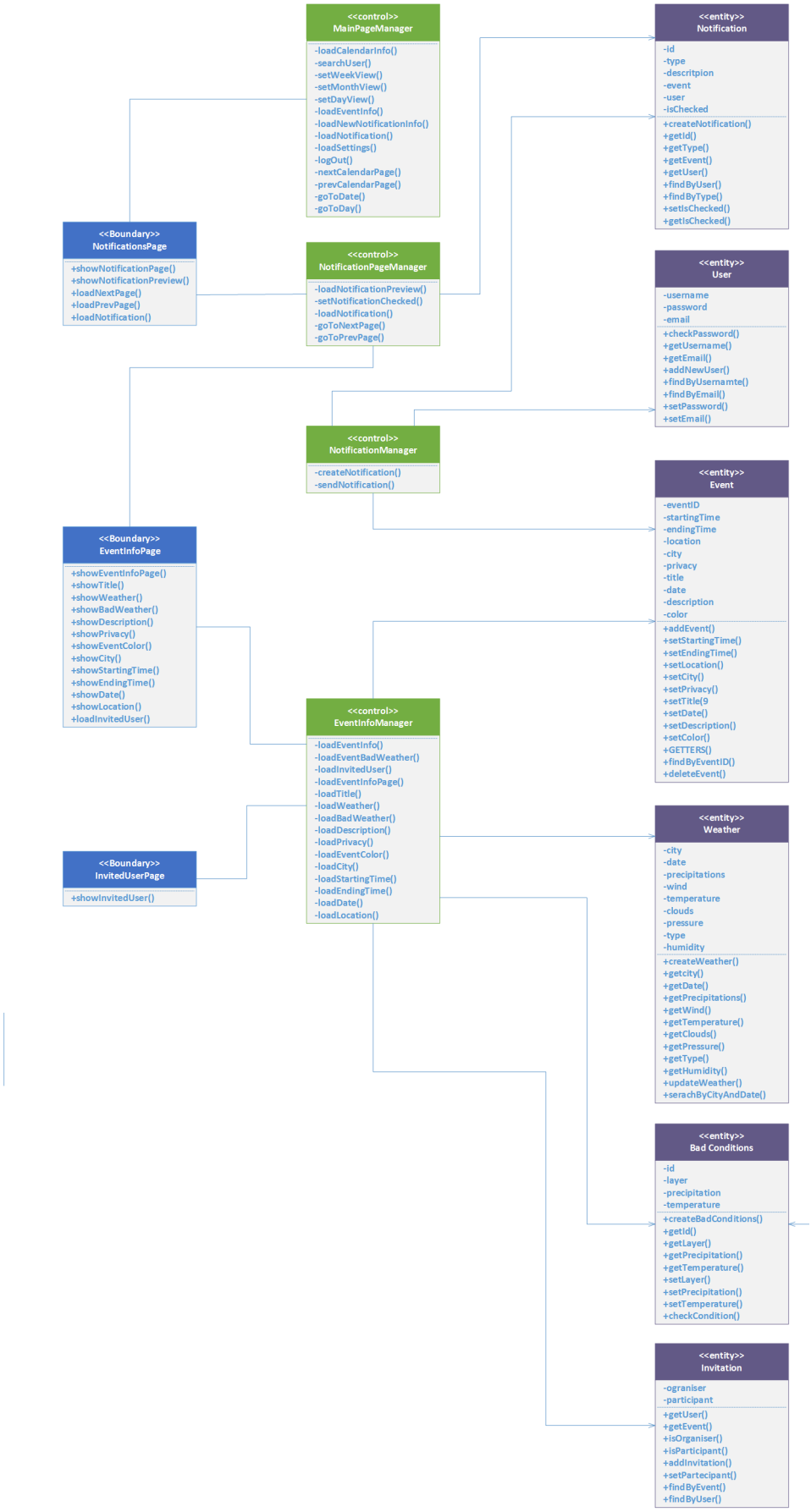


Figure 5.4: BCE diagram for the notifications management

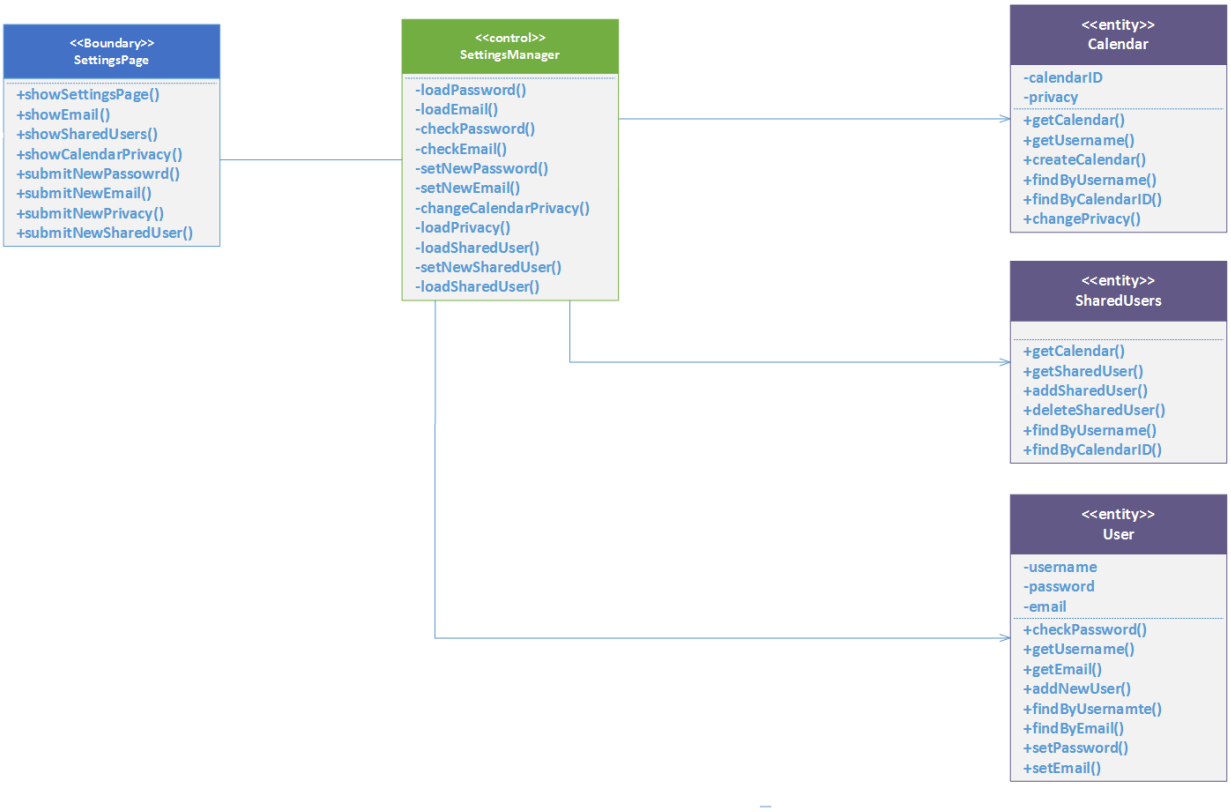


Figure 5.5: BCE diagram for the settings management

5.5 BCE Search Management

In this section we present the search system structure. Two controls manages respectively the search engine and the public calendar/event loading and showing. We will divide the schema in two parts to make it more readable.

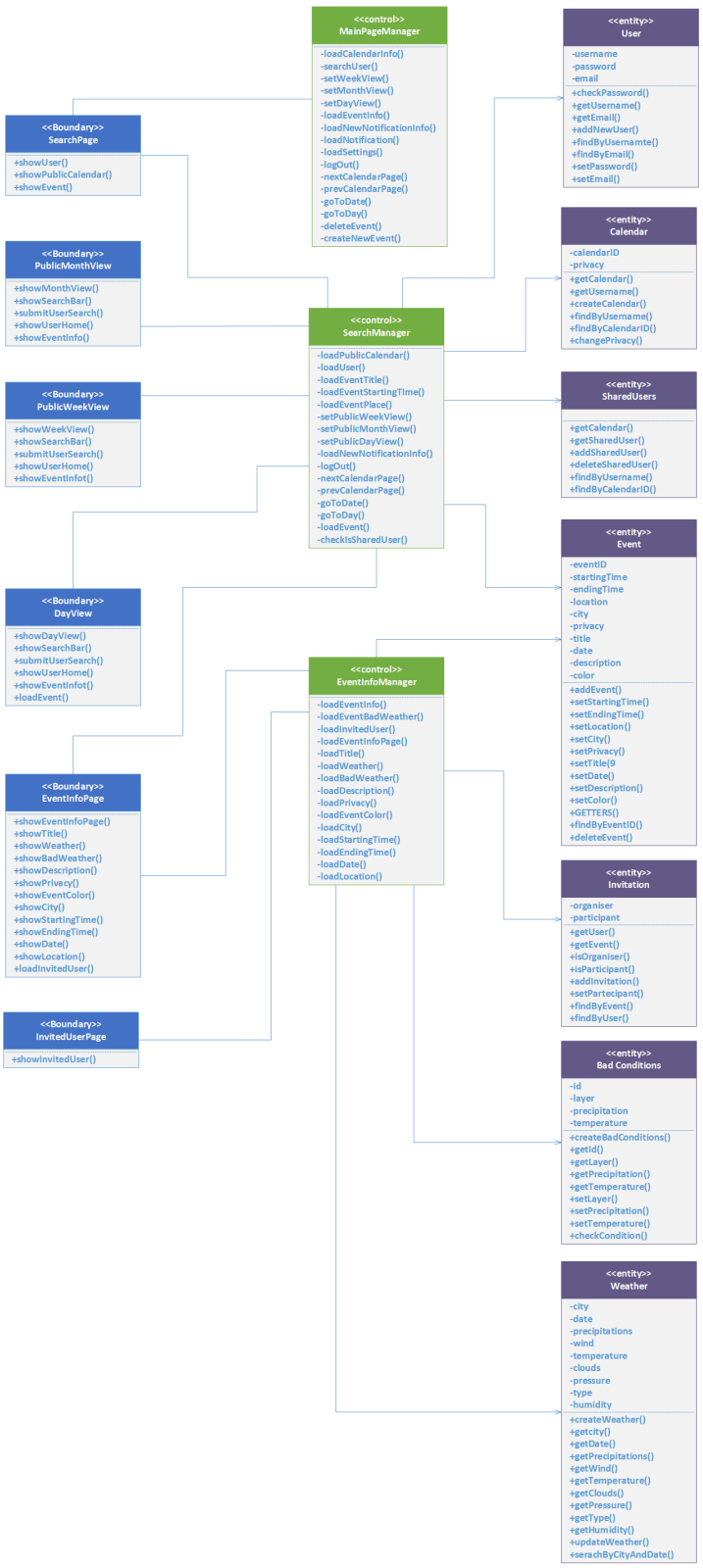


Figure 5.6: BCE diagram for the search management

Part 1 :

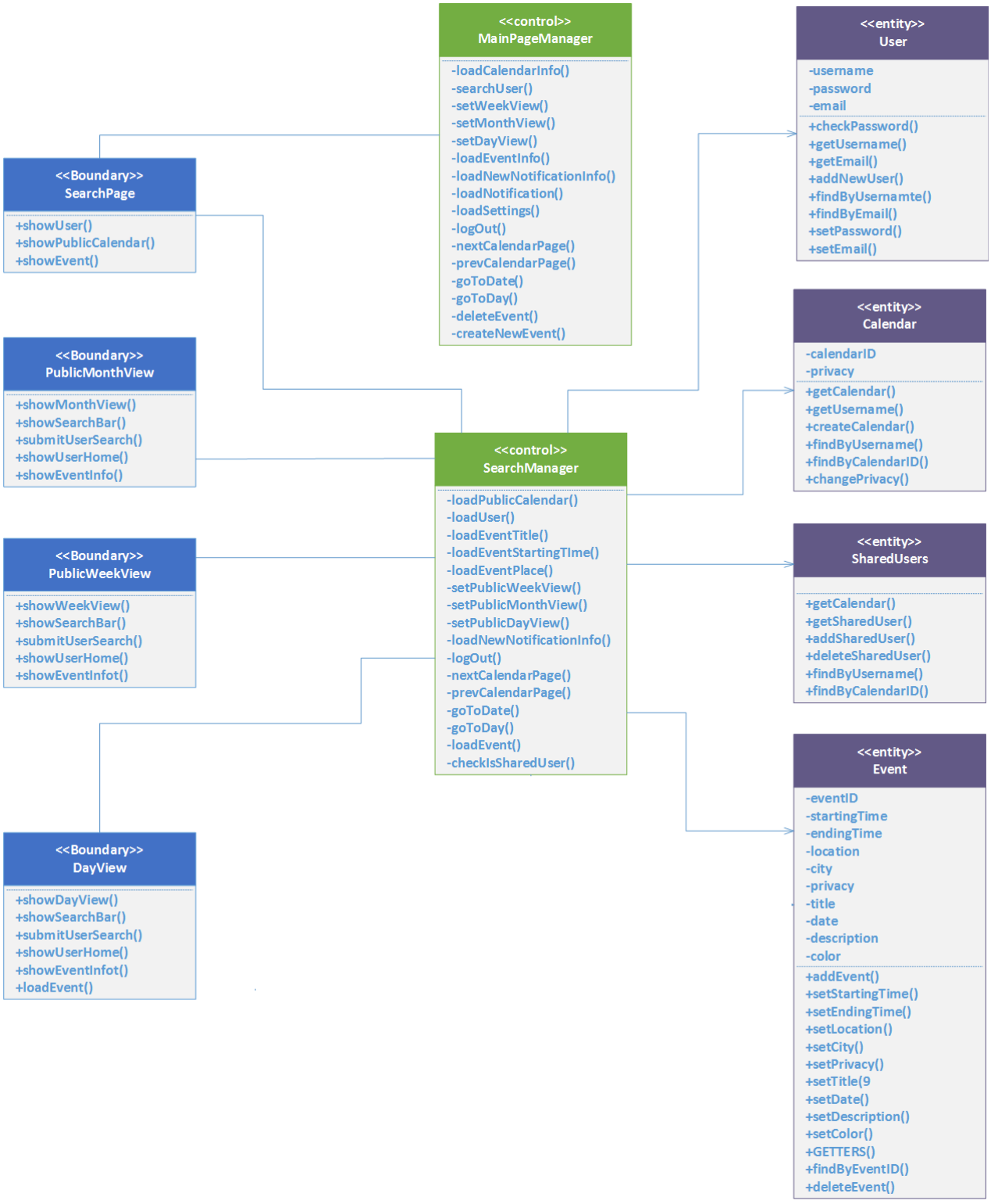


Figure 5.7: BCE diagram for the search management P1

Part 2 :

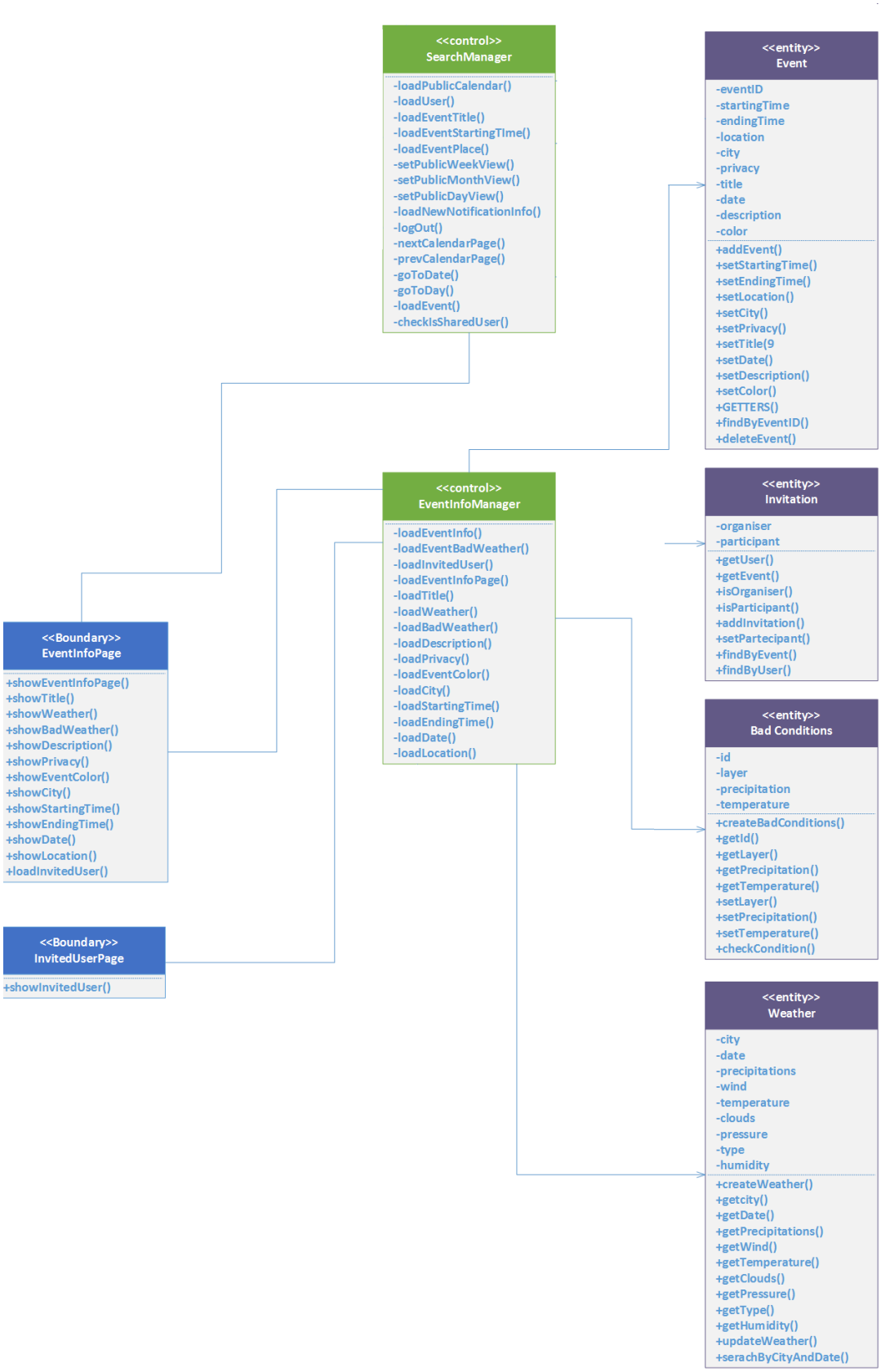


Figure 5.8: BCE diagram for the search management P2

Chapter 6

Sequence diagram

After talking with some professors we thought that just copying the sequence diagrams we made in the RASD document, would be useless, the only difference that we present is that the "Store Notification" action here will be extend with an interaction with our database. In this section we will:

- Show in details how the "Control Data Consistency" (explained at the beginning of the sequence diagrams section, and presented in the first sequence diagram, that we presented in the RASD document) will work in all the diagrams we have already drawn.
- Propose two other sequence diagram, that we couldn't draw in the RASD, where we will show how the bad conditions system and how the search system will work.

6.1 Bad Conditions System - Sequence Diagram

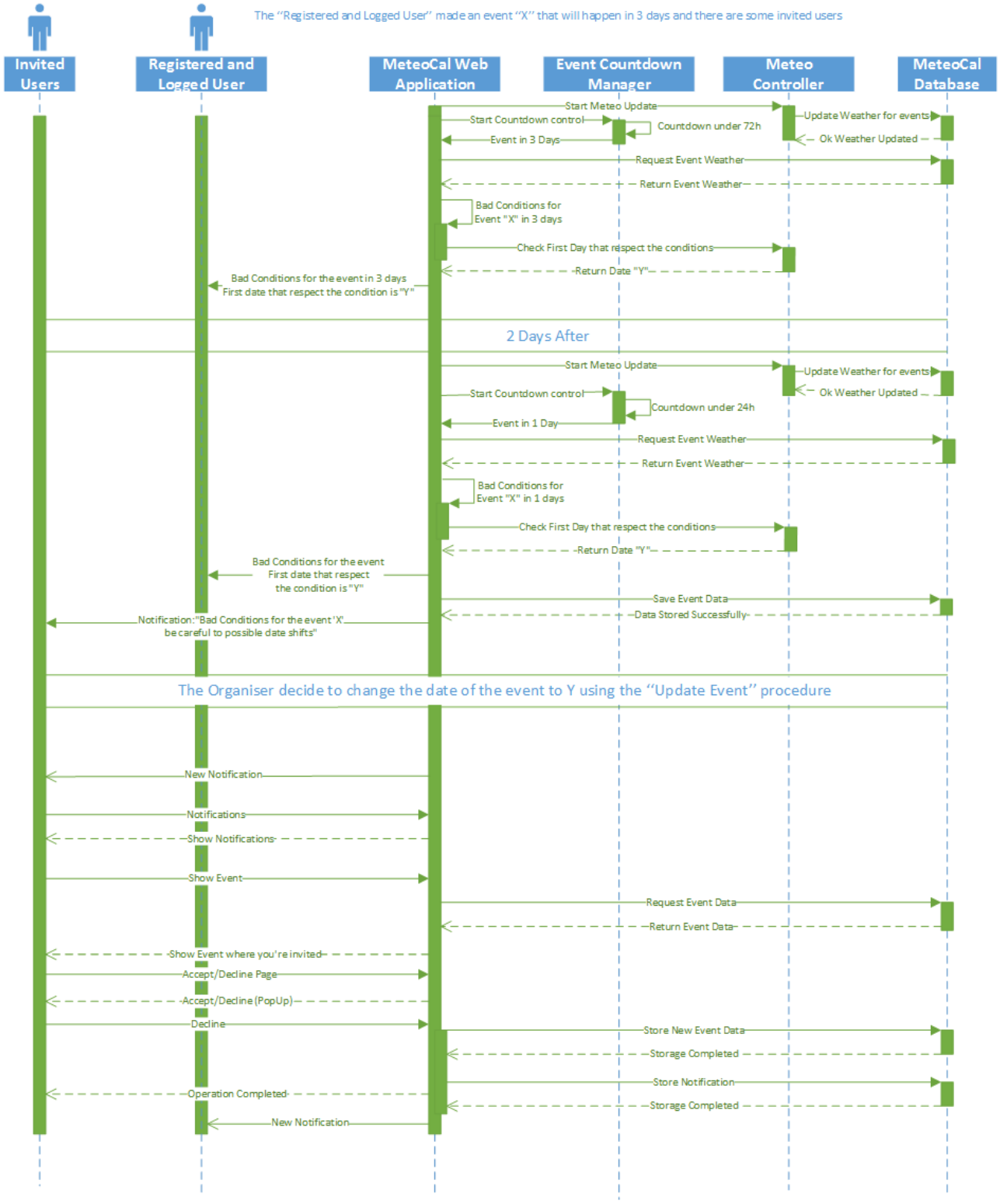


Figure 6.1: Sequence diagram for the bad conditions system

Other Considerations:

- If only one of the fields in the bad conditions, specified by the user, is violated then the bad conditions became true and the user will be warn if the time is running out for the event to begin (3 days before and 1 days before).

6.2 Search System - Sequence Diagram

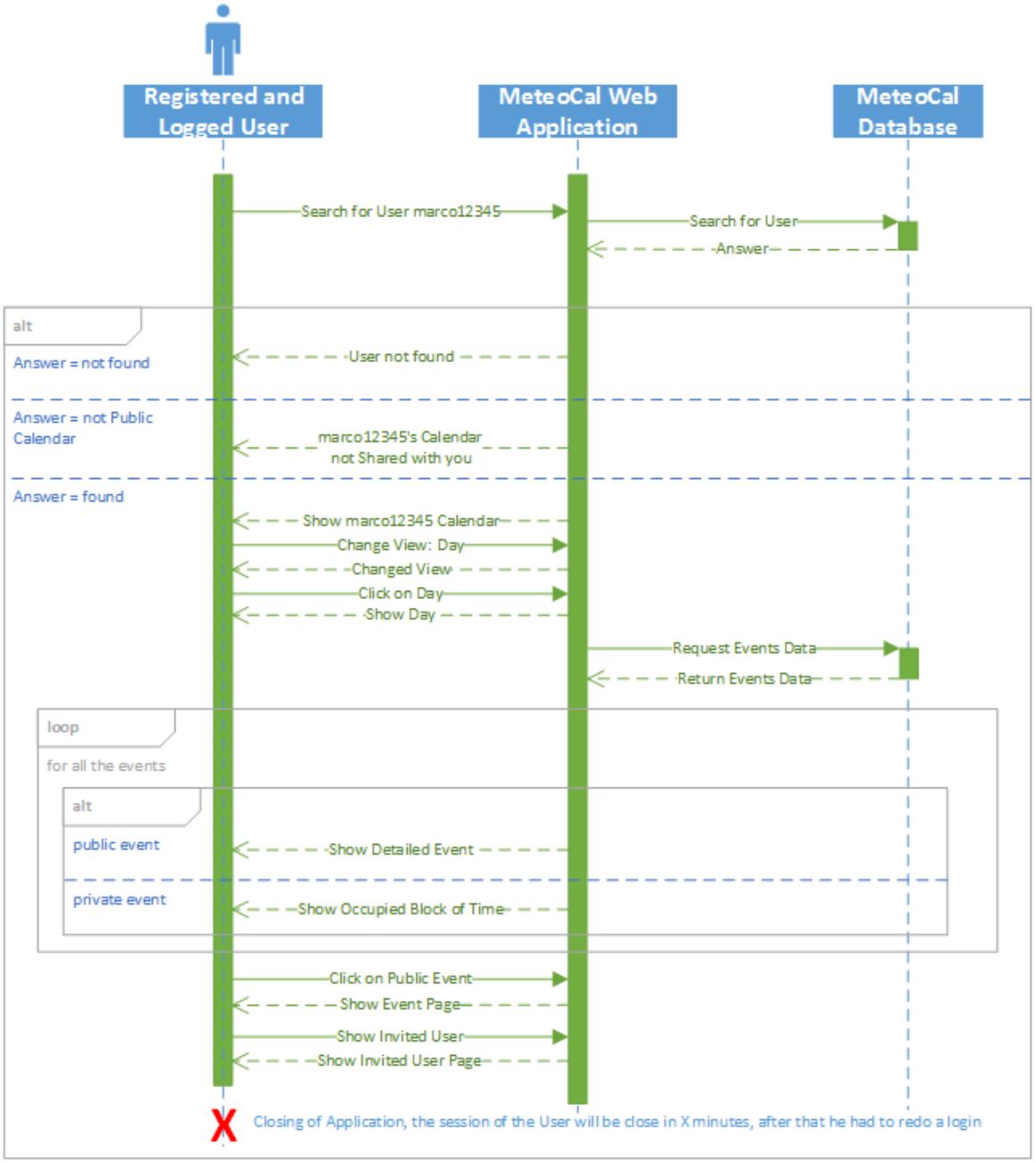


Figure 6.2: Sequence diagram for the search system

Other Considerations:

- The procedure that we presented for the visualization of the invited users, from another user's event, is obviously the same one of the one in which the user that made the research is invited, and also the same if the user is the organiser of one of his events.

6.3 Control Data Consistency - CDC

We have already said that in all the diagrams we drew we use this control in order to avoid redundancy, because in all the cases if the data insert by the user are not correct, we simply have to do a loop and ask again for the input; so here we will explain in details what kind of input we need to be correct, and how the system will respond to these wrong input.

6.3.1 Login CDC

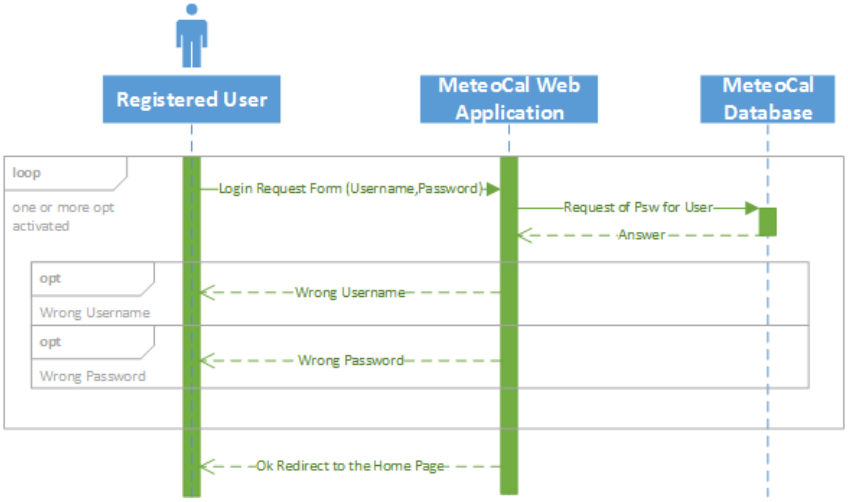


Figure 6.3: Sequence diagram that show the Login CDC

6.3.2 Registration CDC

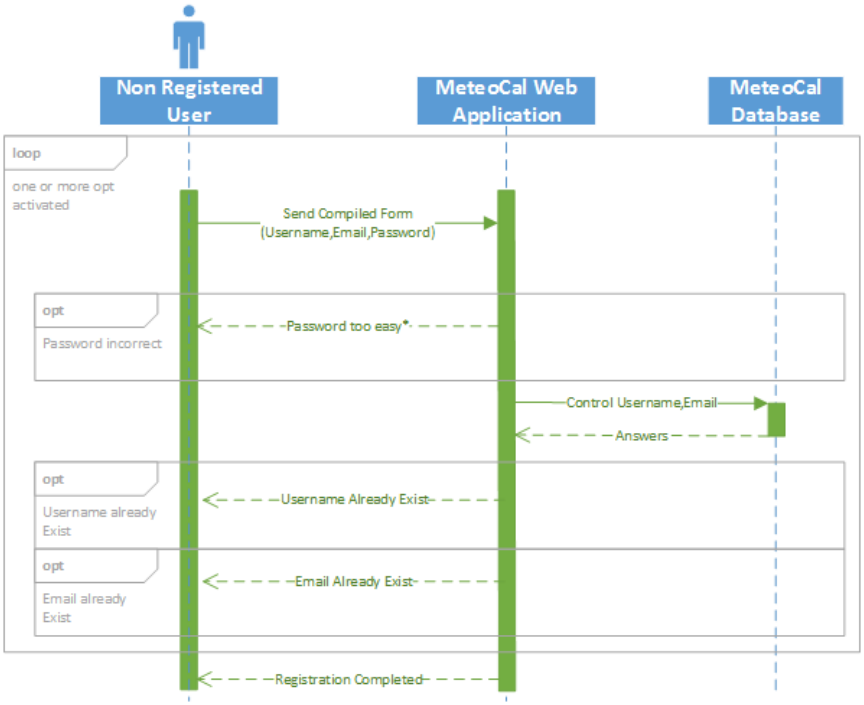


Figure 6.4: Sequence diagram that show the Register CDC

6.3.3 Create/Update Event CDC

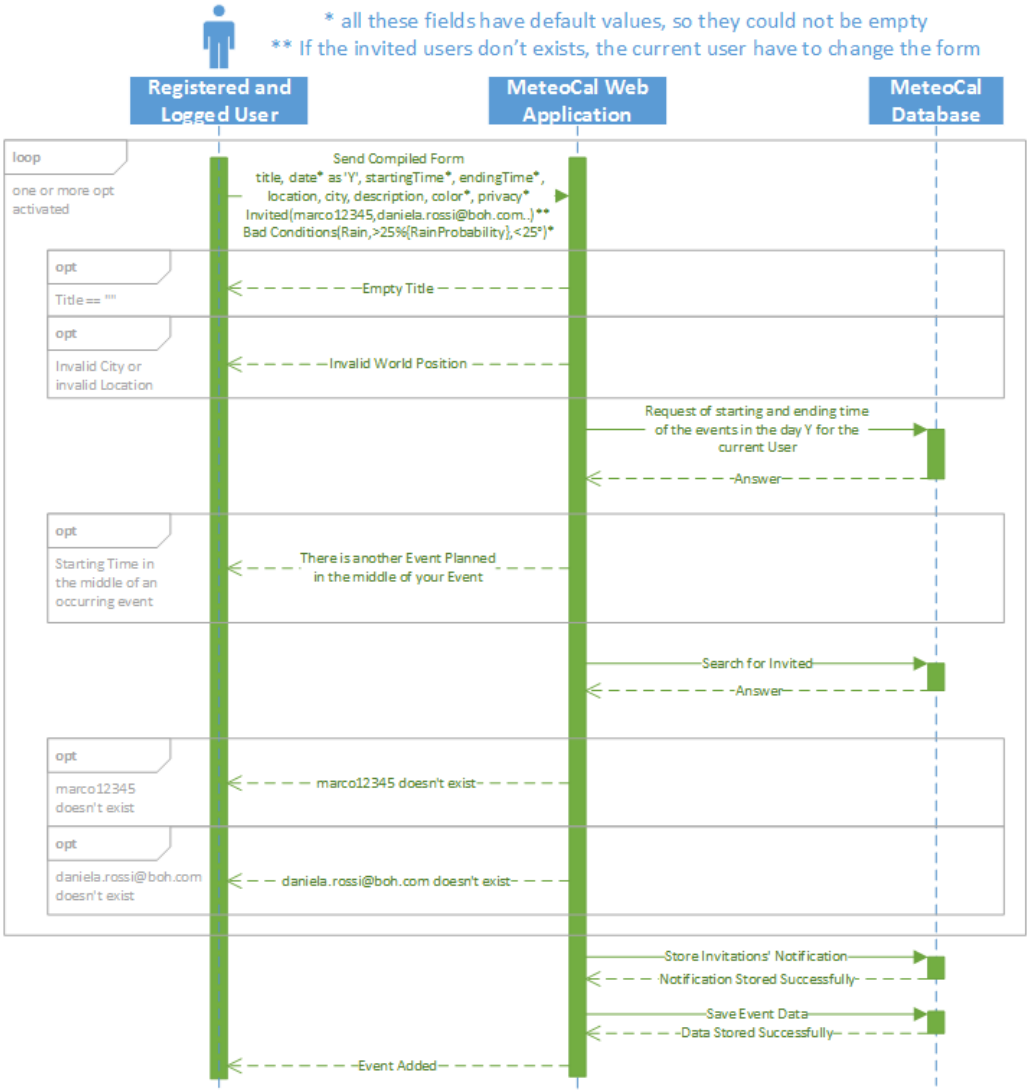


Figure 6.5: Sequence diagram that show the Create and Update CDC

6.4 Other Considerations

We here specify some other things:

- An invited user can accept or decline whenever he wants, and he can always change his idea.
- If the Organiser try to Invite some Users that does not exists, the system will tell to the user: "User doesn't exist" and then the system will send to the user the compiled form without invite the other users until the form is corrected and resubmitted.
- Only the Organiser can change the Event, the invited could only see its details.
- The System, once a day, will call some subroutines that will update the Weather and control the Countdown for the events to begin.
- From now on, we take the decision to allow only 1 Event at a certain time, we don't allow overlaps. In future the application could be updated with the insertion of other types of Event that allow overlaps or simply we won't do the overlapping control.
- When a User Log in the application, the Notifications will be loaded from the database, and them will be shown to the user.

Chapter 7

Used Tools

These are the software used to create this Design Document:

- **Share Latex Web Platform**: used to redact and format this document.
- **Microsoft Visio 2013**: used to create all the Ux,BCE and Sequence Diagrams;

Bibliography

[1] Notes from information systems course, 2014.

[2] Rafaela Mirandola. Design document slides, 2014.

[3] Damian Andrew Tamburri. Software architecture styles slides, 2014.