

POLITECNICO DI MILANO
SOFTWARE ENGINEERING 2

Design Document

Mirjam Škarica

Milan, December 2014.

Table of Contents

1. Introduction.....	1
1.1 Purpose.....	1
1.2 Acronyms	1
2. System Architecture.....	3
2.1 JEE architecture overview.....	3
2.2 Identifying subsystems.....	5
3. Persistent data management.....	7
3.1 Conceptual design.....	7
3.3 Logical Design.....	10
4. User experience.....	12
4.1 Guest Home Page.....	12
4.2 User Home Page.....	13
4.3 User Settings.....	14
4.4. Manage Events Page.....	15
4.5 Notifications Page.....	17
5. BCE diagrams.....	18
5.1 Guest Home Page.....	19
5.2 User Settings.....	20
5.3 Event and Notification Management.....	21

1. Introduction

1.1 Purpose

The purpose of this document is to provide a comprehensive description of the structure of the MeteoCal system. It will state and analyze the design decisions made in order to satisfy all the requirements stated in the Requirements Analysis and Specification Document (RASD). This document is meant mainly as a guideline for developers of the software in question.

1.2 Acronyms

The following are some acronyms and their corresponding terms used throughout the document:

- **CSS:** Cascading Style Sheets
- **DB:** Database
- **DBMS:** Database Management System
- **EIS:** Enterprise Information System
- **ER:** Entity–Relationship
- **HTML:** HyperText Markup Language
- **HTTP:** Hypertext Transfer Protocol
- **JEE:** Java Platform Enterprise Edition
- **JMS:** Java Message Service

- **JSP:** JavaServer Pages
- **MCV:** Model View Controller
- **RASD:** Requirements Analysis and Specification Document
- **UML:** The Unified Modeling Language
- **UX:** User Experience

2. System Architecture

2.1 JEE architecture overview

Developing a system using Java Enterprise Edition (JEE) is one of the requirements imposed by the client. Having that in mind, an overview of JEE architectures given below (*fig 2.1*).

JEE follows the distributed multi-tiered application approach which means the entire application may not reside at a single location, but is distributed. JEE is divided into four tiers:

- **Client tier:** runs on the client machine and provides a dynamic interface to the middle tier, JEE server (*fig 2.1*) by interacting directly with users and communicating with the aforementioned server. The client tier distinguishes two types: application client and web client. The former being a standalone desktop application, and the latter usually a web browser. MeteoCal will be implemented as a web client.
- **Web tier:** runs on the JEE server and comprises of JavaServer Pages (JSP) and Java Servlets. Basic idea is the following. A servlet receives HTTP requests from the client tier and forwards the data to the business tier. After receiving a response from the business tier, dynamic web pages are generated using JSP and are sent back to the client.
- **Business tier:** runs on the JEE server and contains the application's logic. It processes data received from the client and data retrieved

from the database (DB) in order to send a response back to the client.

There are three types of business components in the JEE architecture:

- **Session Beans:** represent a session with a client. Being a transient object, they lose their data when the session terminates
 - **Java Persistence Entities:** entity beans are persistent objects and retain data even after the session. (e.g. they represent a row of data in a database table).
 - **Message-Driven Beans:** used for receiving the receiving the Java Message Service (JMS) messages asynchronously.
- **Enterprise Information System (EIS) tier:** runs on the Database Server and is responsible for storing and retrieving all persistent data.

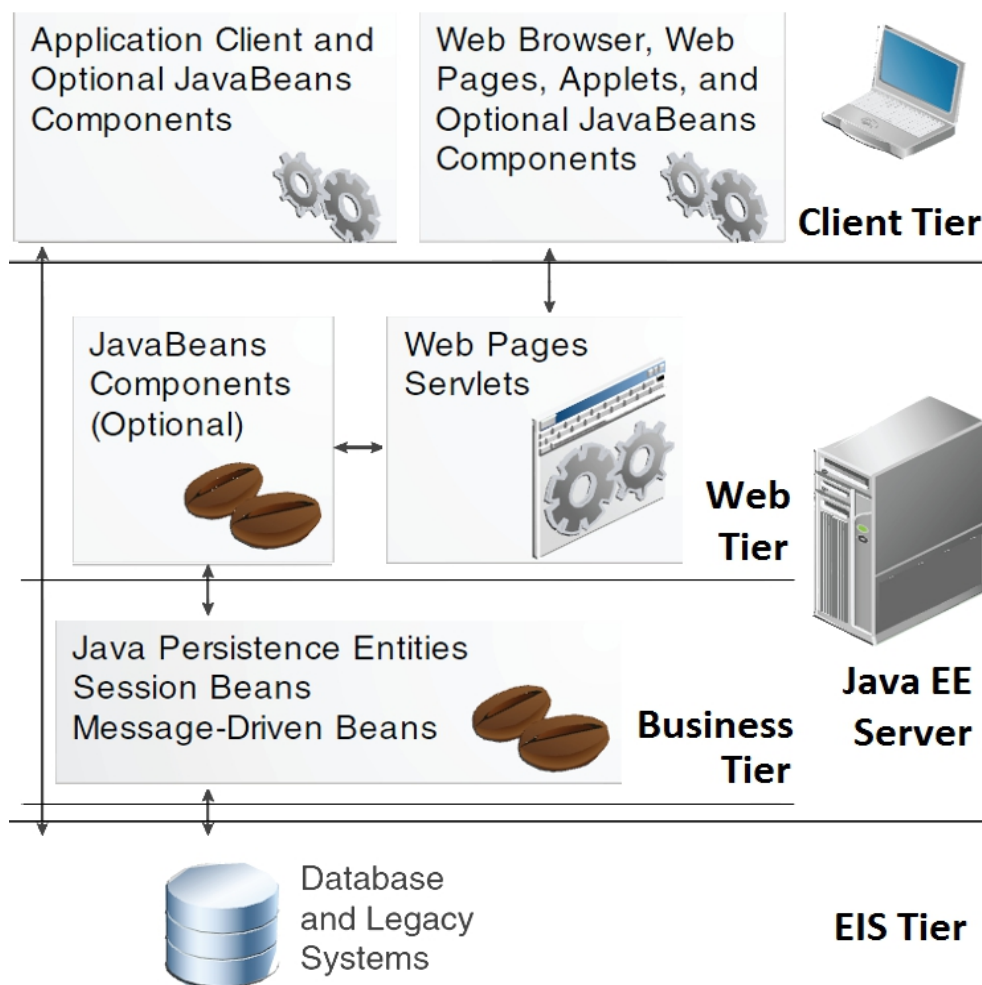


Fig 2.1

2.2 Identifying subsystems

MeteoCal system is broken-down into smaller subsystems by using a top-down approach. This is done in order to distinguish logically separate components so their role in the entire system is more understandable, making their functionality easier to identify and implement.

Subsystems (*Fig 2.2*) are derived from the functional requirements stated and described in the RASD document.

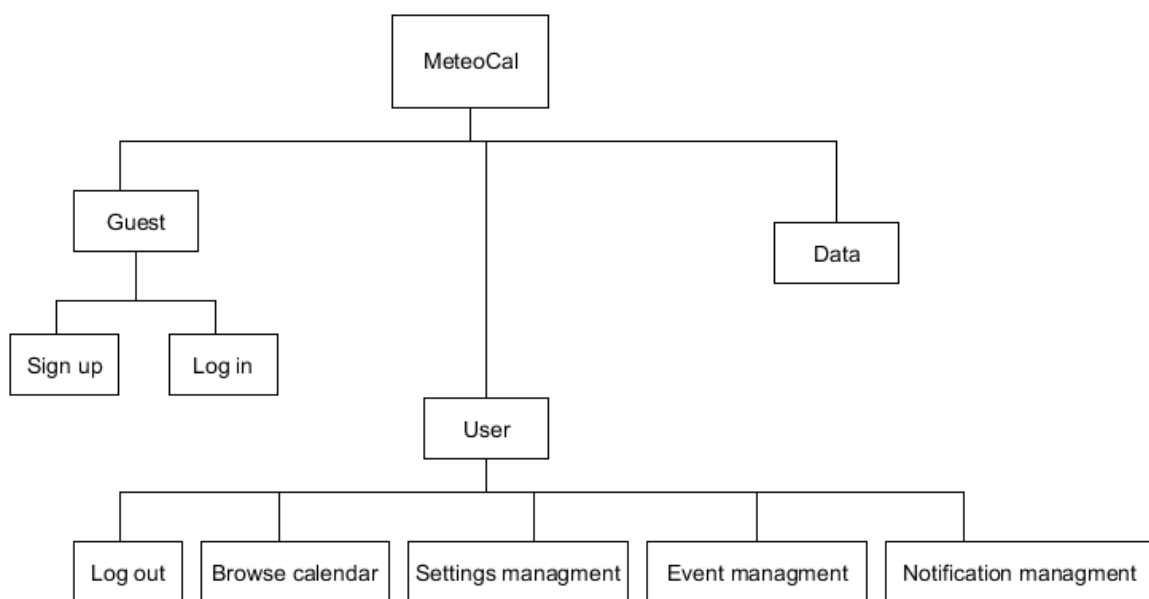


Fig 2.2

Brief description of each subsystem's functionality is given below:

- **Guest subsystem** allows clients access to the application :
 - Log in: allows existing users to access the application
 - Sign up: allows new clients to register and log in
- **User subsystem** allows logged in clients (users) the full use of MeteoCal system's functionalities:

- Log out
- Browse calendar
- Event management: allows users to create events, and the possibility to update and delete them
- Notification management: depending on the type of notification received (weather or invitation), user can view it and in the case of an invitation notification, he can choose to accept or decline it
- Setting management: allows users to change their settings, such as email or password.
- **Data subsystem:** is the subsystem where all the persistent data is stored (like the weather information)

3. Persistent data management

MeteoCal system's data will be stored into a relational database. Different diagrams of the DB schema are provided in order to identify and gain a deeper understanding of the system's underlying physical structure.

3.1 Conceptual design

Conceptual view of system's data communicates a clear picture regarding the entities we want to store and relationships between. The entity-relationship diagram, *fig 3.2*, is derived from Class diagram stated in the RASD document, Chapter 5.2. Some minor changes in the reference to the class diagram have been introduced for the sake of simplicity. They will be pointed out in the short explanation of the ER diagram bellow.

Event stores all information regarding an event. The class diagram suggests the inheritance relationship between *Event*, *OutdoorEvent* and *IndoorEvent* should be modeled using a disjoint constraint (*fig 3.1*). However, since the two classes differ minimally, instead of the constraint, a boolean attribute *is_outdoor* is added to the event entity to differentiate the two types of events.

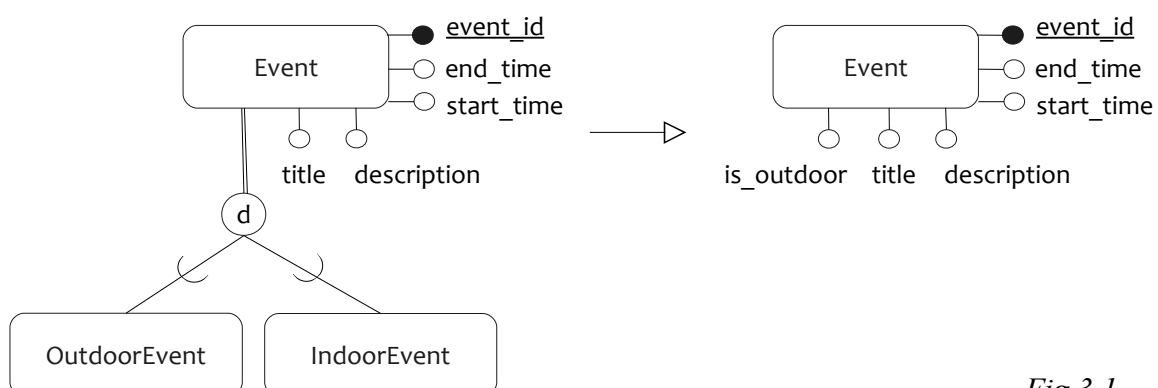


Fig 3.1

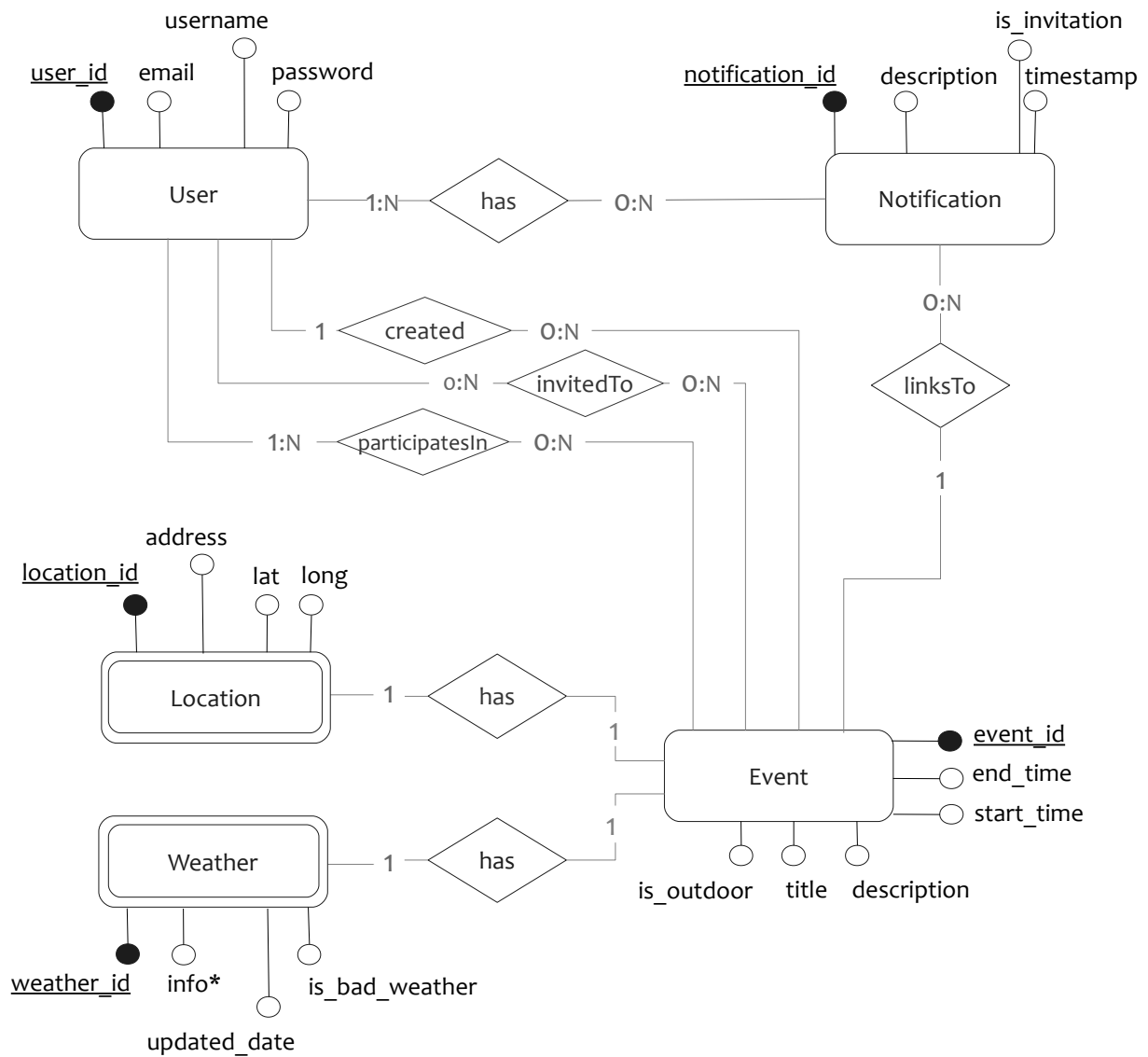


Fig 3.2 Entity-Relationship diagram

User stores all information regarding a user.

Location stores all information about an event's location. With attributes *lat* and *long* representing location's latitude and longitude, respectively.

Weather stores available weather forecast information given the event's time and location. Attribute *info** is introduced to avoid unnecessary clutter in the ER diagram. *Info** is a substitution for the following boolean attributes: *sun*, *rain*, *thunderstorm*, *clouds*, *snow* and one float attribute *temperature*.

Notification stores all notifications users receive. Two possible types are weather and event notification. Taking the class diagram into consideration, their relationship should be modeled using a disjoint constraint, but is instead differentiated much in the same way **Event** entity is, by introducing a boolean attribute *is_invitation*.

Relations:

Event has Location: each event has exactly one location, and each location is associated with exactly one event.

Event has Weather: each event has exactly one weather forecast information, and each weather forecast information is associated with exactly one event.

User created Event: a user can create zero or more events, but each event has to be created by exactly one user.

User invitedTo Event: a user can be invited to zero or more events, an event can have zero or more users invited

User participatesIn Event: a user can participate in zero or more events, but each event has to have at least one user participating (the user who created the event is automatically considered a participant)

User has Notification: each user can have zero ore more notifications, each notification is associated with one or more users

Notification linksTo Event: each notification links to exactly one event, but each event can produce zero or more notifications

3.3 Logical Design

The logical view of system's data is the table schema of its database. It is derived by translating the ER diagram by applying the following transformations ¹:

Entities:

- Each entity turns into a table
- Each attribute turns into a column in the table
- The primary key of each entity becomes the primary key of table

Relations:

- **1:N relation:** take the primary key of the relation on the “1” side and set it as a foreign key for the table on the “N” side
- **1:1 relation:** there is a choice which table will receive the primary key of the other as its foreign key. Usually, if the dependent entity can be determined, it receives the foreign key.
- **N:N relation:** each such relation turns into a separate table and the keys of the two entities are set as the primary key

The model obtained after this process is the following:

Table name	Attributes (*Primary keys are denoted by underlining the attribute name)
USER	<u>user_id</u> , email, username, password
EVENT	<u>event_id</u> , creator, title, end_time, start_time, description, is_outdoor
LOCATION	<u>location_id</u> , event_id, is_outdoor, address, latitude, longitude
WEATHER	<u>weather_id</u> , event_id, update_date, is_bad_weather, sun, rain, thunderstorm, clouds, snow, temperature
NOTIFICATION	<u>notification_id</u> , event_id, description, is_invitation, timestamp
PARTICIPATION	<u>user_id</u> , <u>event_id</u>
INVITATION	<u>user_id</u> , <u>event_id</u>
USER_ NOTIFICATION	<u>user_id</u> , <u>notification_id</u>

¹ Reference material: T.Haigh, [teaching material](#)
Connolly, Begg, Database Systems: A Practical Approach to Design, Implementation and Management

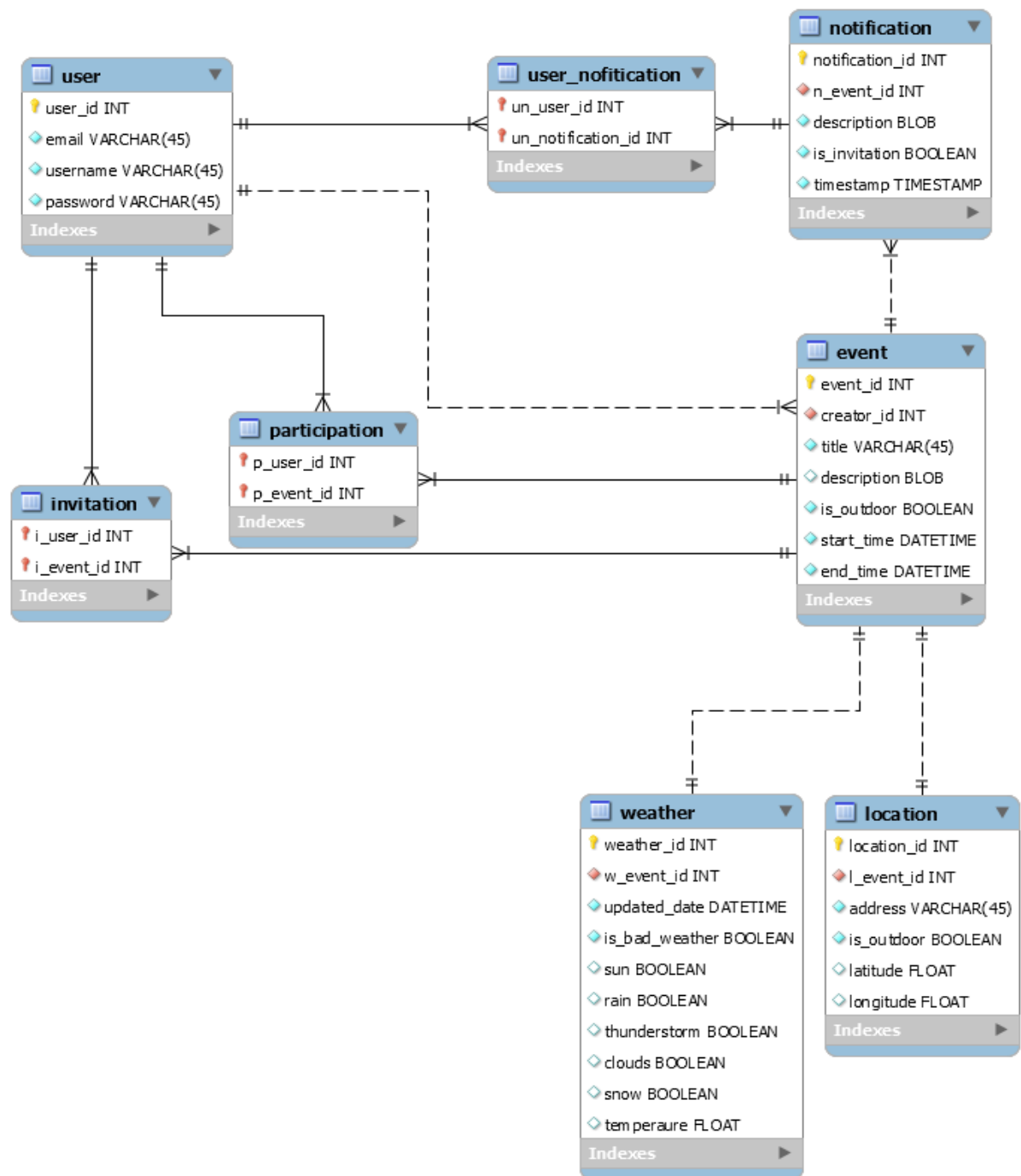


Figure 3.3 Logical view

4. User experience

Depending on design choices, every application has a unique way of interacting with a user. This section will depict graphically MeteoCal's user experience (UX).

In The following stereotypes are used in all diagrams:

<<screen>> represents a dynamically created page that contains all other elements as well as links to other pages.

<<screen component>> represents a modular building element with a specific purpose that can be attached to any part of a page, thus extending it's functionality.

<<input form >> is a type of a <<screen component>> containing input fields for user to fill out and submit it to the system for further processing.

4.1 Guest Home Page

The UX diagram in fig 4.1 represents MeteoCal's home page. The guest homepage consists of two smaller components, specifically, two input forms for the two different types of clients previously identified in the RASD document. The *signUp form* for registering a new user, and the *logIn form* for an already registered user. After filling out the appropriate form the client submits it. If the submission was successful the client is redirected to his personal home page, otherwise he is redirected back to the guest home page.

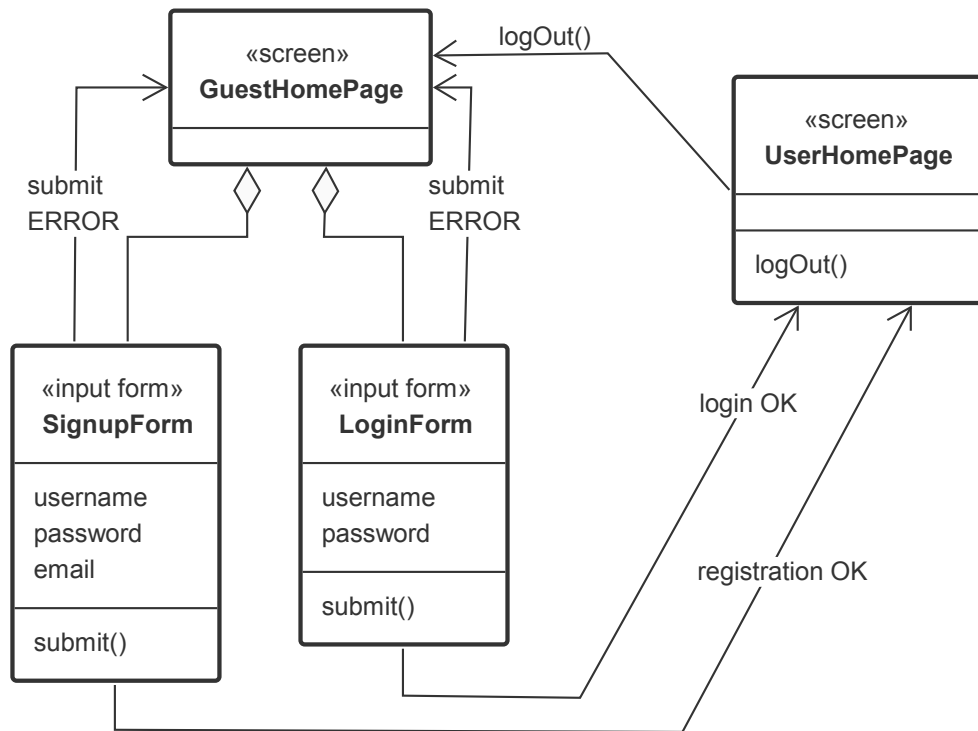


Fig 4.1

4.2 User Home Page

The diagram in *fig 4.2* represents the already logged in user's personal home page from which he can access all of MeteoCal's functionality like browse his calendar, check notifications, manage events and his settings. Each of these functionalities will be explained in more detail in their respective diagrams. It is important to mention that all diagrams following this one should contain the *NavigationBar* component, but will be omitted to reduce clutter. From the *NavigationBar* user can, at any point and from any page, return to his homepage, manage his settings and log out.

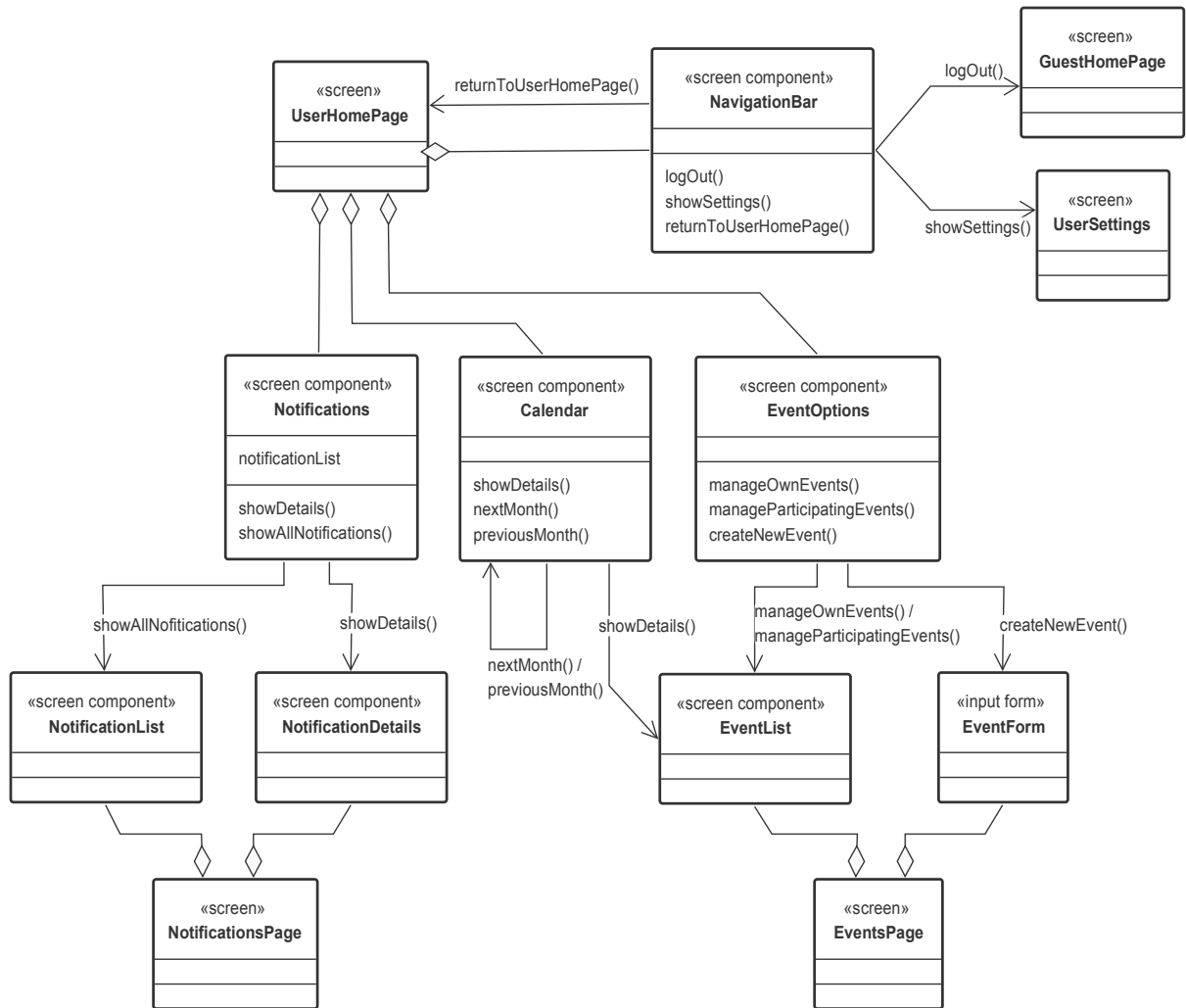


Fig 4.2

4.3 User Settings

The diagram in fig 4.3 represents user's settings page which consists of two main components. The *UserSettings* from which he can view his *username*, *password* and *email*, and the *ChangeSettings* component from which he can change the two latter credentials. In order to do so, he chooses the desired change option and in the *ChangeSettings* component a corresponding input form is loaded. After filling and submitting the form, a appropriate message is displayed in the *sessionMessage* field informing the user of a successful change or asking him to fill out the form again otherwise.

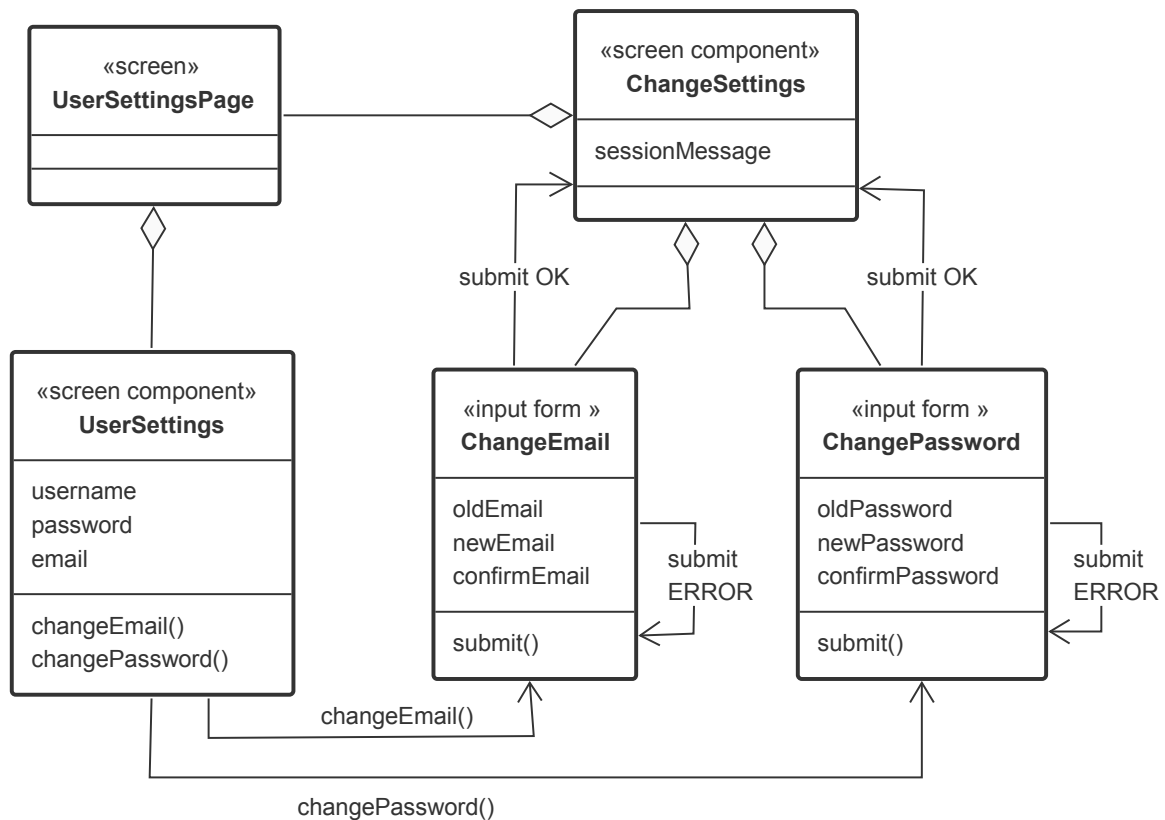


Fig 4.3

4.4. Manage Events Page

The diagram in fig 4.4 depicts the way events are displayed and managed. The *EventsPage* consists of two main components *EventsList* and *SingleEventView*. The *EventsList* component lists all user's events which can be filtered with respect to the fact if he's organizing or just participating in them. The *SingleEventView* can dynamically load different components depending which option user chooses. One option is to choose to create a new event which loads the *EventForm* component to be filled, submitted and validated much in the same way as in previous cases. Another option is to view a specific event's details making the *EventDetail* component load. Once loaded, depending on whether the user is the creator or only a participant, he can update, delete and invite users to his event, or choose to not participate in the other case.

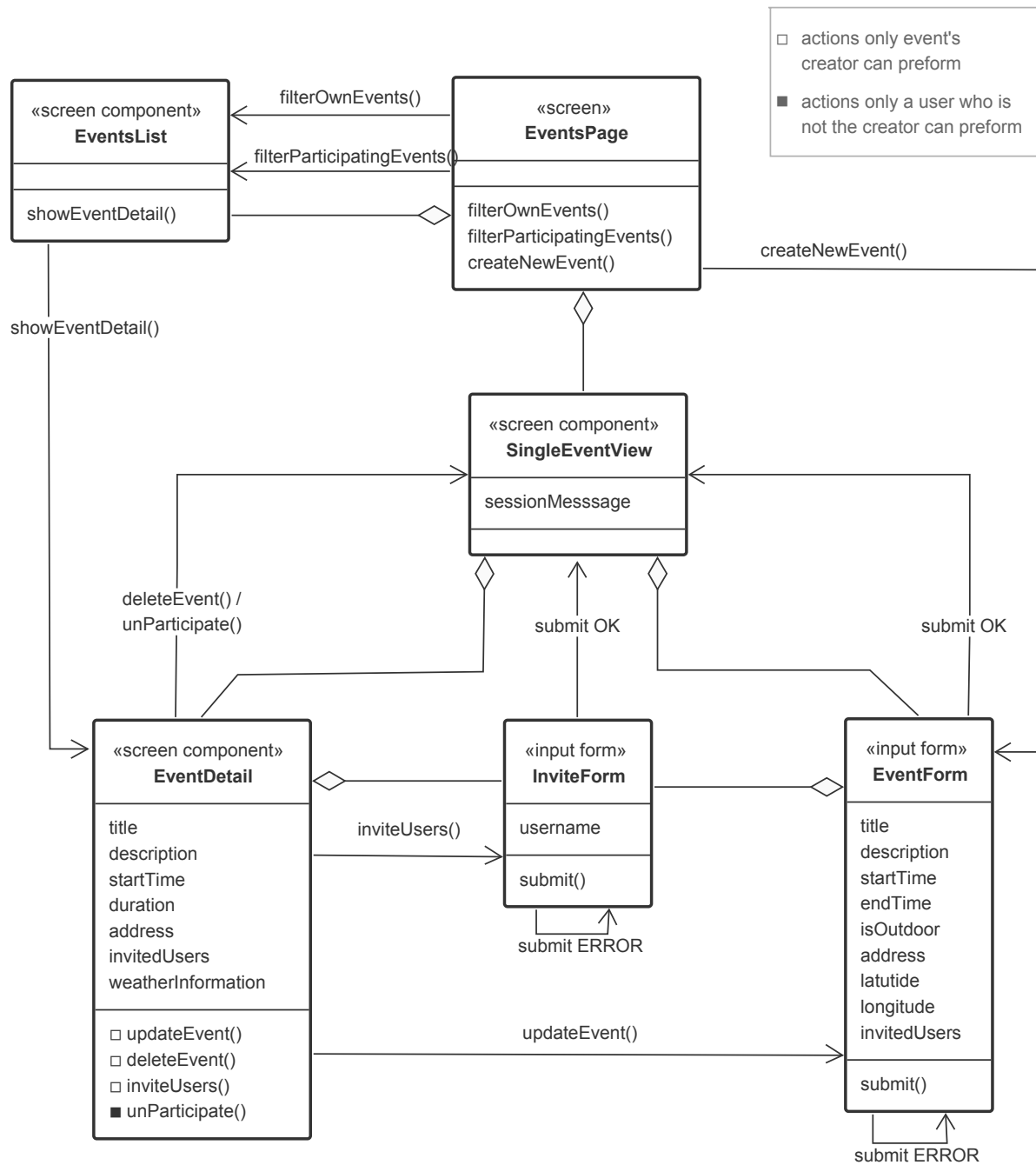


Fig 4.4

4.5 Notifications Page

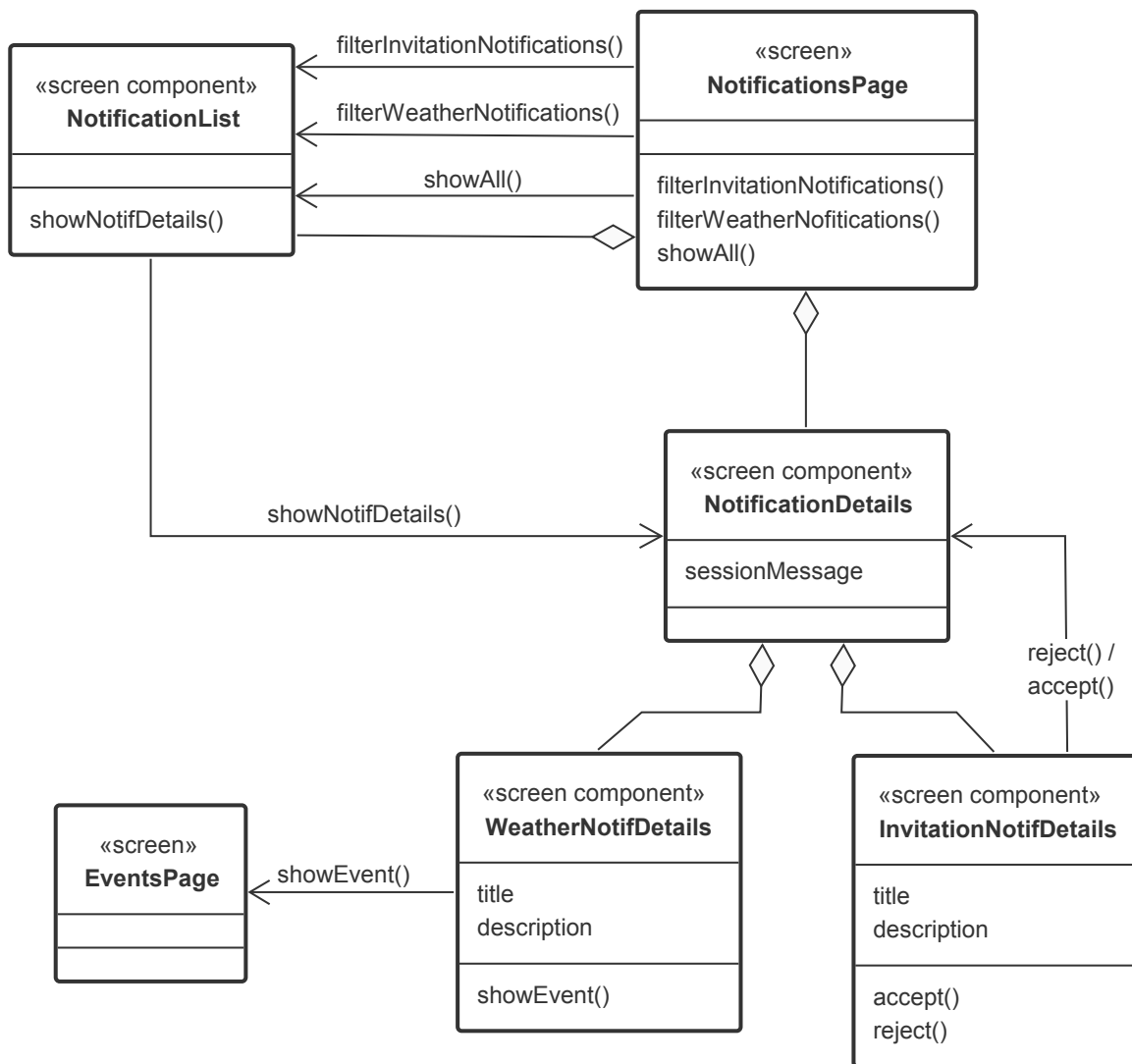


Fig 4.5

The diagram in the *fig 4.5* represents the Notifications Page. Its structure is similar to that of *EventsPage*. It consists of two main components, *NotificationList* and *NotificationDetails*. *NotificationList* lists all user's notifications which can be filtered regarding their type (weather or invitation). By selecting a specific notification its details are dynamically loaded in the *NotificationDetails* component. In the case of a weather notification, user can only view it, but in the case of notification being an invitation, user can choose to accept or reject it.

5. BCE diagrams

Another informative way to model the MeteoCal's system is using the Boundry-Control-Entity (BCE) Diagrams. This is done to further analyze the logic behind the application being developed. Quick explanation of the three main components is as follows:

Boundary represents <<screens>> or <<screen components>> explained in the previous chapter.

Control represents part of the system containing the logic of the application.

Entity represents system's data stored in a database.

5.1 Guest Home Page

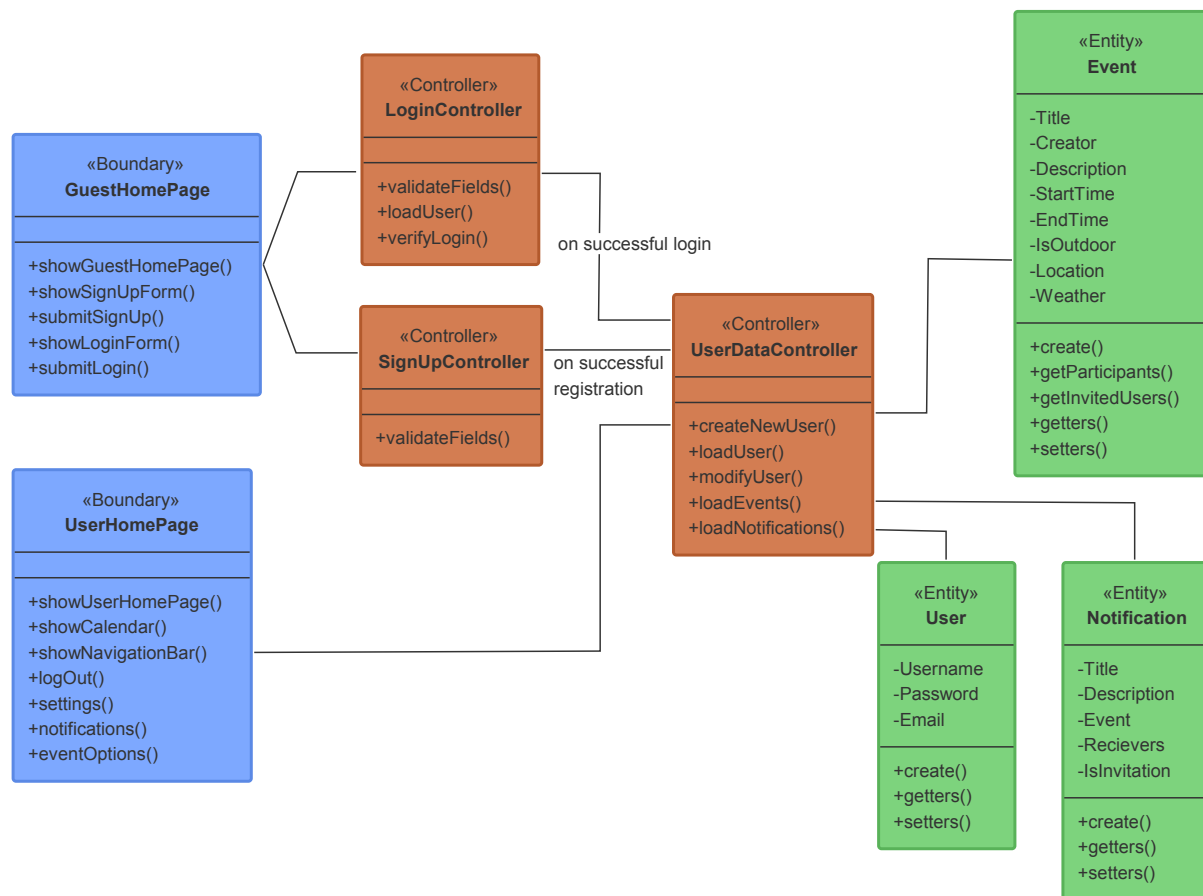


Fig 5.1

5.2 User Settings

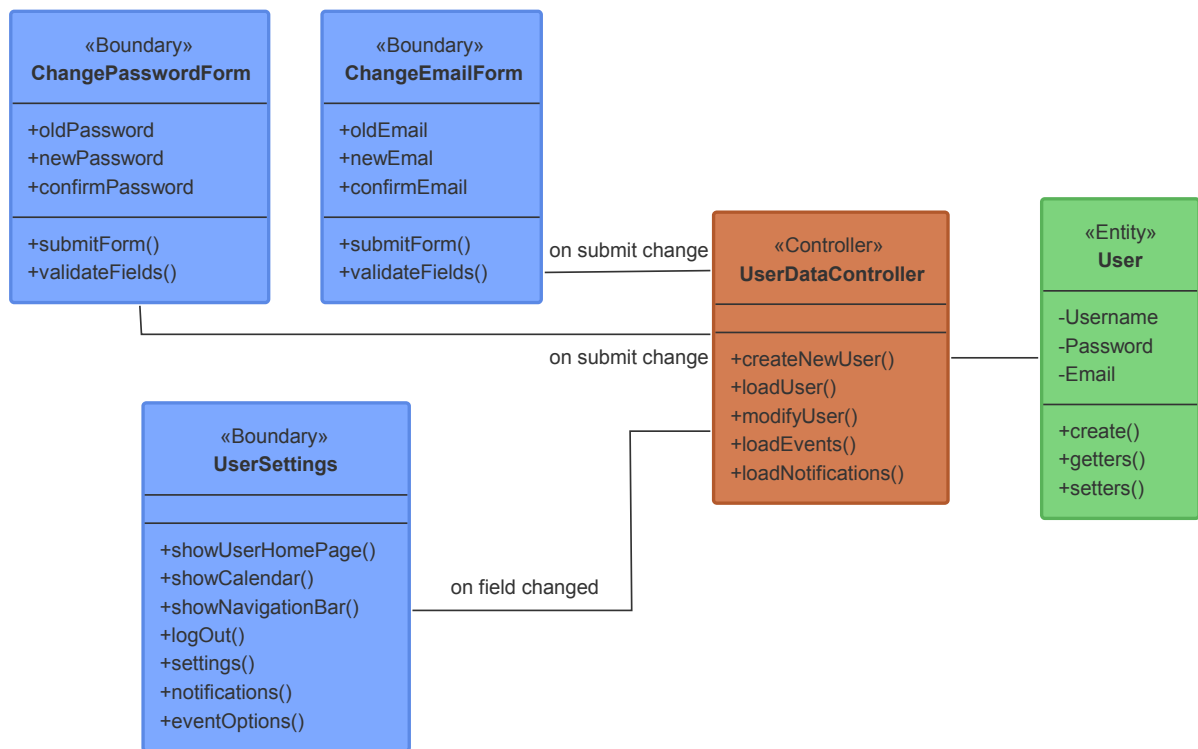


Fig 5.2

5.3 Event and Notification Management

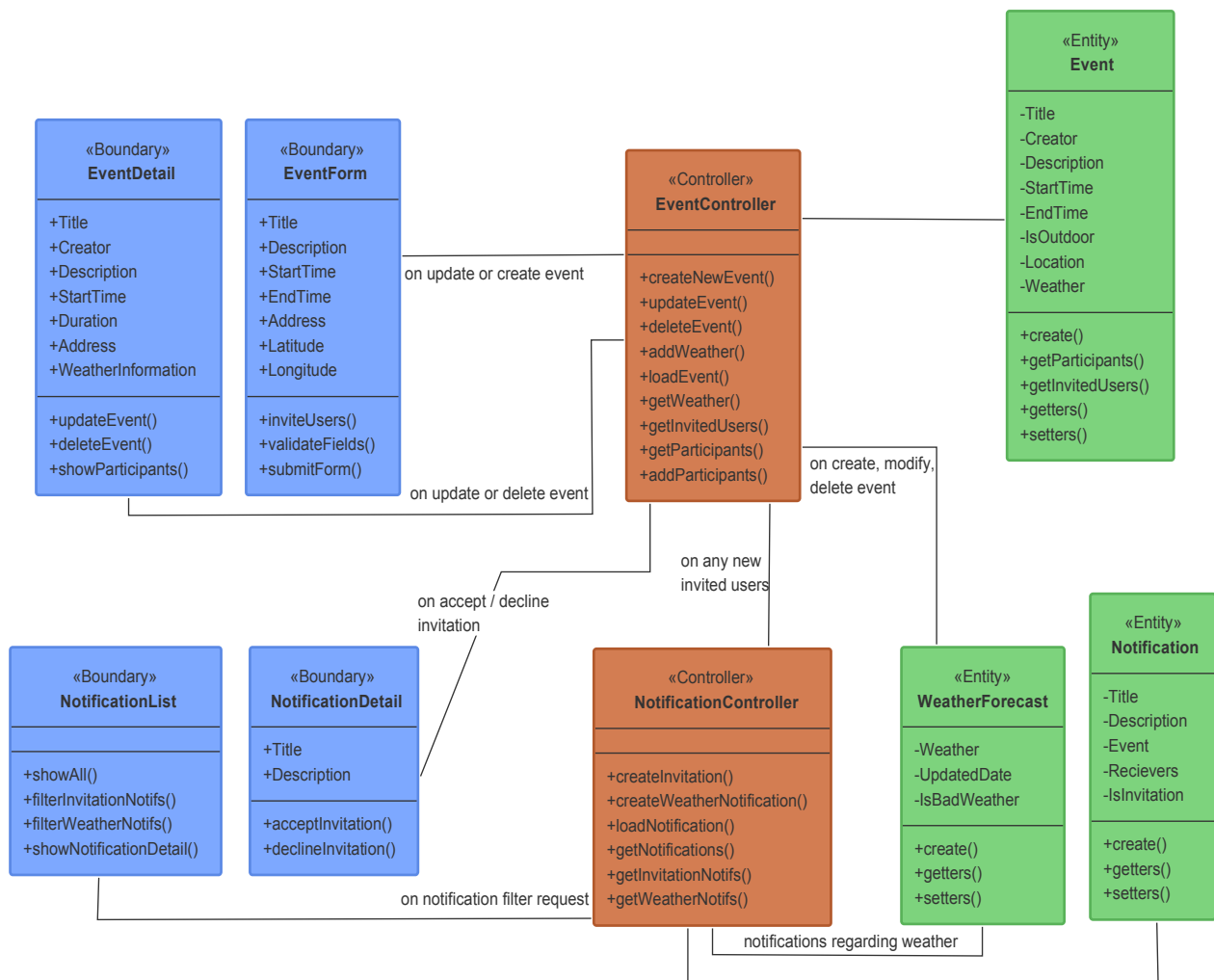


Fig 5.3