**1. Architecture Description**

**1.1. JEE Architecture Overview**

Before starting to explain our application’s architecture we want to focus on JEE Architecture.



JEE has a four tiered architecture divided as:

* **Client Tier**: it contains Application Clients and Web Browsers and it is the layer that interacts directly with the actors. As our project will be a web application the client will use a web browser to access pages;
* **Web Tier**: it contains the *Servlets* and Dynamic Web Pages that needs to be elaborated. This tier receives the requests from the client tier and forwards the pieces of data collected to the business tier waiting for processed data to be sent to the client tier, eventually formatted;
* **Business Tier**: it contains the Java Beans, that contain the business logic of the application, and Java Persistence Entities.
* **EIS Tier**: it contains the data source. In our case it is the database allowed to store all the relevant data and to retrieve them.

**3. User Experience**

In this part, we want to describe the experience given by our system to its user with an User Experience(UX) Diagram. To create that, we use a Class Diagram, using some stereotypes (*<<screen>>*, *<<screen compartment>>* and *<<input form>>* ) to make it more understandable. In this Diagram, *<<screen>>* represents pages, *<<screen compartment>>* represents parts of the page that can be shared with others. *<<input form>>*, eventually, represents some input fields that can be fulfilled by a user (this information will be submitted to the system clicking on a button).

We also decided to divide the UX Diagram in functionalities in order to better understand all the Diagram.

Each of the paragraphs below is titled with the name of the functionality represented by the UX Diagram drawn.

**3.1. Homes**

We can see below the Diagram that represents the different home pages showing how a generic user can interact with all the system. All the two home pages inherit from an Home screen that is also marked with $, so it is reachable from every page of the system. For an easier interpretation of the schema we decided to draw some components that are contained in all the screens, but we decided to draw them only in this UX Diagram, if not, it would have been very difficult to understand other diagrams.

Here is a list of these elements:

* **Log In Panel** and **Guest Menu**: They are in all guest’s pages;
* **Notification Panel,** **User Menu, Calendar** and **Log Out Panel**: They are in all user’s pages;
* **Search Panel:** It is in all pages of guest and user;

Moreover, from a *Guest Home* we can log in through the *Log In Panel* and, if there is no error, we will be redirected to our user’s home page. If there is an error we will be redirected to the *Guest Home* again. If we’ve already logged in we can perform a logout through the *Log Out Panel*.

*Menus* contain the principal functionalities exploited by each type of generic user.

The *Notification Panel* shows if there are new notifications(like delete of an event, change on detail of an event, or propose to change detail of an event by the system) or pending invitations to an event and allows to reach the associated pages.

The last functionality expressed by this UX Diagram is the possibility of signing up reachable from the *Guest Menu*. If we click on the right link the *Sign Up* screen is shown. This screen contains a form to be fulfilled with all the data of the new user. Eventually, we can submit the data to the system (that will have different clear behaviors in case of error or not).

UX DIAGRAM

**3.2. Event Managing**

This part of the UX Diagram allows to understand how the event managing functionalities are offered through the user experience.

UX DIAGRAM

From the *Calendar,* a screen compartment of the *User Home*, we can access our daily view(*Day View* screen), which represent a view of the event present in that day and then, if we want to, create, modify or delete it.

If we want to insert a new event which is not yet in the system we can do it through an appropriate screen (*Event Creation* screen).

If we want to update or delete an event, we can do through an appropriate screen (*Event Update* screen), simply modifying data about the event, or pressing delete button.

Every time that a user create, modify or delete an event the *Day View* page is reloaded with the result of the operation.

**3.4. User Search, Notification and Invitation Management**

This part of the UX Diagram allows to understand how the user search, notification and invitation management functionalities are offered through the user experience.

UX DIAGRAM

Search functionalities can be performed through the *Search Bar,* and can be used by a user or by a guest*.* From the input form associated to the compartment we can submit a username for the research and send them to the system that will answer with the *Search Results* page, showing a set of *User Preview*s. Starting from here we can also click on the user’s name and see its related calendar. If someone makes his calendar private, he does not compare in the *Search Results* page. Viewing other user calendar’s we can see all his public events.

We can also receive notification or invitation requests. Invitations are sent from a user, which is proposing you to participate to his event. Notification are messages sent from the system, like to propose a change of date in one of your event, to let you know that an event in your calendar received some modification from his creator, and so on. These event will be notified by the *Notification Panel*. Through the appropriate link we can access to the *Pending Invitation* screen that shows us the set of pending invitations and, for each of them, it allows to accept it or decline it. Same things for the notifications, where we can see all the new notification in the *Notification Page* screen where also here, for each of them, we can accept it, decline it, or put the notification as read.

**4. BCE Diagrams**

We decided to give a further design schema of MeteoCal using the Boundary-Control-Entity pattern, because it is very close to the Model-View-Controller pattern (in fact we can say that boundaries maps to the view, controls map to the controller and entities map to the model) and UML defines a standard to represent it.

It is important to know that boundaries are partially derived from the Use Cases Diagram provided in the RASD (so they don’t need further explanations) and they gather some screen in the UX Diagram. All the methods of the screens are written into the appropriate boundaries (maybe some names changed a bit only to make them more understandable in this context). There is a method *showXXX()* for every screen in the UX Diagram. Screen compartment doesn’t have a “show method” because we considered them as pieces of screens and not as standalone screens.

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.

We decided to separate the BCE diagram into sub-diagram, to make it clearer and understandable.

**4.1. Entity overview**

All entities of the BCE model will be presented in a very fragmented way, to prevent the diagrams to be too chaotic. For this reason we provide an entity overview, in the way that the reader can always refer to this diagram.

ENTITY OVERVIEW

**4.2. Sign up and Log in**

The two boundaries *GuestHome* and *UserHome* represent the interface with users and the basic functionalities that they can exploit in their home page.

We write down now a synthetic description of the controllers used:

* ***ProfileDataManager:*** this controller manages the verification of profile data in case of sign up or modification of profile information. It also creates a new user in case of signing up correctly.
* ***LogInManager:*** this controller manages the verification of log in fields. The logic is that firstly, it examines if the fields are filled correctly (we need *checkLogInFields()* because it might happen that a user fills the password field with only three characters, and we know that, for instance, our password has to be at least eight characters long. In this case there is no need to look for the user in the database, because we know that the log in will be incorrect in any case). Then the controller can load the correspondent user finding it by e-mail address and then check the password.

BCE DIAGRAM

**4.2. User Search, Notification and Invitation**

Searching functionalities are shared between the user and the guest. Notification and invitation can be received only by users.

The controllers are:

* ***UserDataLoader:*** the loader of personal data about the user. This controller is linked to the user, the notifications and the invitations. It is used to manage the notification and invitation panel of the user. In fact it loads personal user profiles, pending invitations and new notifications. It also allow to accept or to decline invitations and notifications.
* ***DataLoader:*** this loader loads information that are not dependent from the user. In fact it loads other users’ profile and search results (with “search results” we mean the results of a search performed to find users).

BCE DIAGRAM

**4.4.4. Help requests management**

The *UserDataLoader* is used to:

* Show User Profile;
* Show User Calendar;
* Show Calendar Day View.

The *EventManager*:

* Creates a new event;
* Update an existing event;
* Check if the event that user is going to create is fulfilled with valid values;
* Delete an existing event.

All these features are put in relation with the Event entity using the entity’s methods (see Sequence Diagram section).

BCE DIAGRAM