MeteoCal

Project development for the 2014 Software Engineering 2 Course

Department of
Computer Science And Engineering

Authors:

Andrea Bignoli

Leonardo Cella

Supervisor:

Prof. Elisabetta Di Nitto

Preliminary Notes

In our presentation we will reference other documents produced during the development of the project. Everything that we are going to reference, including the source code, can be found at:

https://code.google.com/p/meteocal-bignoli-cella/

Requirement analysis and specification

Problem Description

The first step in the development of this project was an accurate and precise analysis of the initial requirements. This allowed us not only to formalize the initial description of the problem, but also to solve any ambiguity due to the natural language formulation.

> *"The X company wants to offer a new weather based online calendar for helping people scheduling their personal events avoiding bad weather conditions in case of outdoor activities. Users, once registered, should be able to create, delete and update events. An event should contain information about when and where the event will take place, whether the event will be indoor or outdoor. During event creation, any number of registered users can be invited. Only the organizer will be able to update or delete the event. Invited users can only accept or decline the invitation. Whenever an event is saved, the system should enrich the event with weather forecast information (if available). Moreover, it should notify all event participants one day before the event in case of bad weather conditions for outdoor events. Notifications are received by the users when they log into the system."*

Ambiguity Resolution

In our initial analysis we decided to solve any ambiguity by providing the most complete and flexible functionality we considered useful in a real world environment.

For instance we decided to allow the creation of events of any duration, without fixing an upper bond in this phase of the project, since we considered this an unnecessary limit for the usage of our application. Our service will provide a series of weather forecasts covering the whole duration of each event. Obviously the weather forecasts checks regarding adverse conditions will consider the whole duration of the event.

We also assumed that events of a user may overlap, since this was the possibility that granted more flexibility of operation to end user.

Ambiguity Resolution

An other important decision that we made is that the creator of an event should be allowed to choose, for each event which weather conditions should be considered adverse.

Even if this is not mentioned in the initial problem description, we considered this functionality as really important in a real world application.

The system will then produce notifications about weather forecasts depending on the set of adverse weather conditions defined by the creator of the event.

Functional Requirements

With the previous considerations we formulated the following functionalities to provide, devided by actor involved:

- Guest
  - Sign up
- User
  - Login
  - Account Management (view settings, edid settings, calendar privacy...)
  - Calendar Management (view, reach event, manage...)
  - Event Creation
  - Event Management (view, participate, cancel participation...)
  - Invitation Management (view, reach related event, reply...)
  - Notification Management (view, reach related event...)
  - Search for events and users on the system
  - View event and user pages according to visibility rules

Functional Requirements

- Event creator (on the created event)
  - Invite users
  - Edit event details, scheduling
  - Cancel event
  - Receive a suggested reschedule change in case of bad weather conditions

The system will provide and update weather forecasts for all the planned events whenever available and automatically notify the creator in case a better scheduling for an event is found. The notifications are generated according to the problem description.

The description presented here provides a basic overview of the functionalities our system offers. More details can be found in the Requirement Analysis and Specification Document.

Non – Functional Requirements

User visible aspects of the system, not directly related to the functional behaviour.

User Interface: a practical and intuitive website;

Client: To use our application is not required any specific knowledge, and is neither necessary an installation;

Errors Handling: all the errors observed in the testing phase were conveniently managed;

Documentation: RASD, DD, User Guide, Installation Guide.

Non – Functional Requirements

Brief but complete description of the mode of operation, purely descriptive.

- It starts with the registration of a new account;
- Through the login with a registered account, we are redirected to the page that contains our own calendar;
- In the upper part of the page there is an instruments bar, this groups the main offered functionalities and some alert messages;
- In closing, the calendar shows the events related to the connected user. To see them deeper, one has just to click on the desired event and a new page with all the details will be opened.

Architectural Considerations

The website is realized with the J2EE technology, it is sustained on a MySql database for the information management.

From the graphic point of view, we have used the components offered by the PrimeFaces library, these modules are then arranged with style sheets in order to get a more accurate arrangement.

In closing in the matter of the weather forecasts service, we rely upon an OpernWeatherMap.

The user who is interested in using our service, needs only an internet connection and an up-to-date browser.

Model Coherency – Alloy Analysis

We made use of Allow to verify the coherency of the model we formulated to answer the requirements we stated. The analysis is the most accurate possible in the representation of our system.

In particular, all the main relationship regarding users, events, notifications, invitations and weather forecasts have been considered and included in the model.

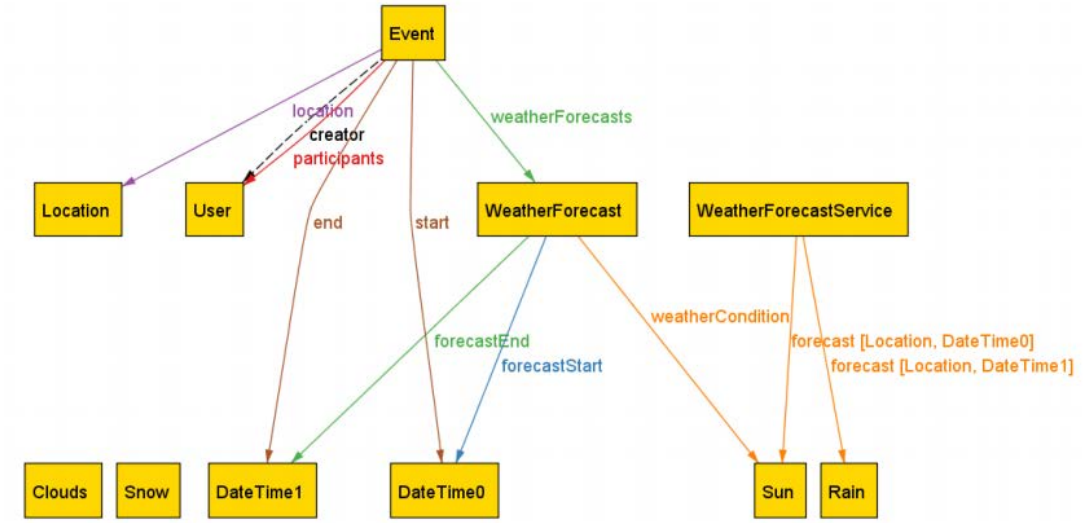
Model Coherency – Alloy Analysis

We were able to check different aspects regarding the coherency of the service offered. For example, the following code ensures that no notification regarding a private event can be sent to users that haven't been invited to the event.

```
// Since user that haven't been invited to a private event cannot partecipate to the
// event they won't receive any notification from it
assert noNotificationsAboutPrivateEventsForNonInvited {
        no n: Notification | n.recipient not in n.aboutEvent.invited and n.recipient !=
        n.aboutEvent.creator and n.aboutEvent in PrivateEvent
}
```

Model Representation – Alloy Analysis

The alloy analysis also allowed us to produce diagrams showing different relationship between the elements of our application.



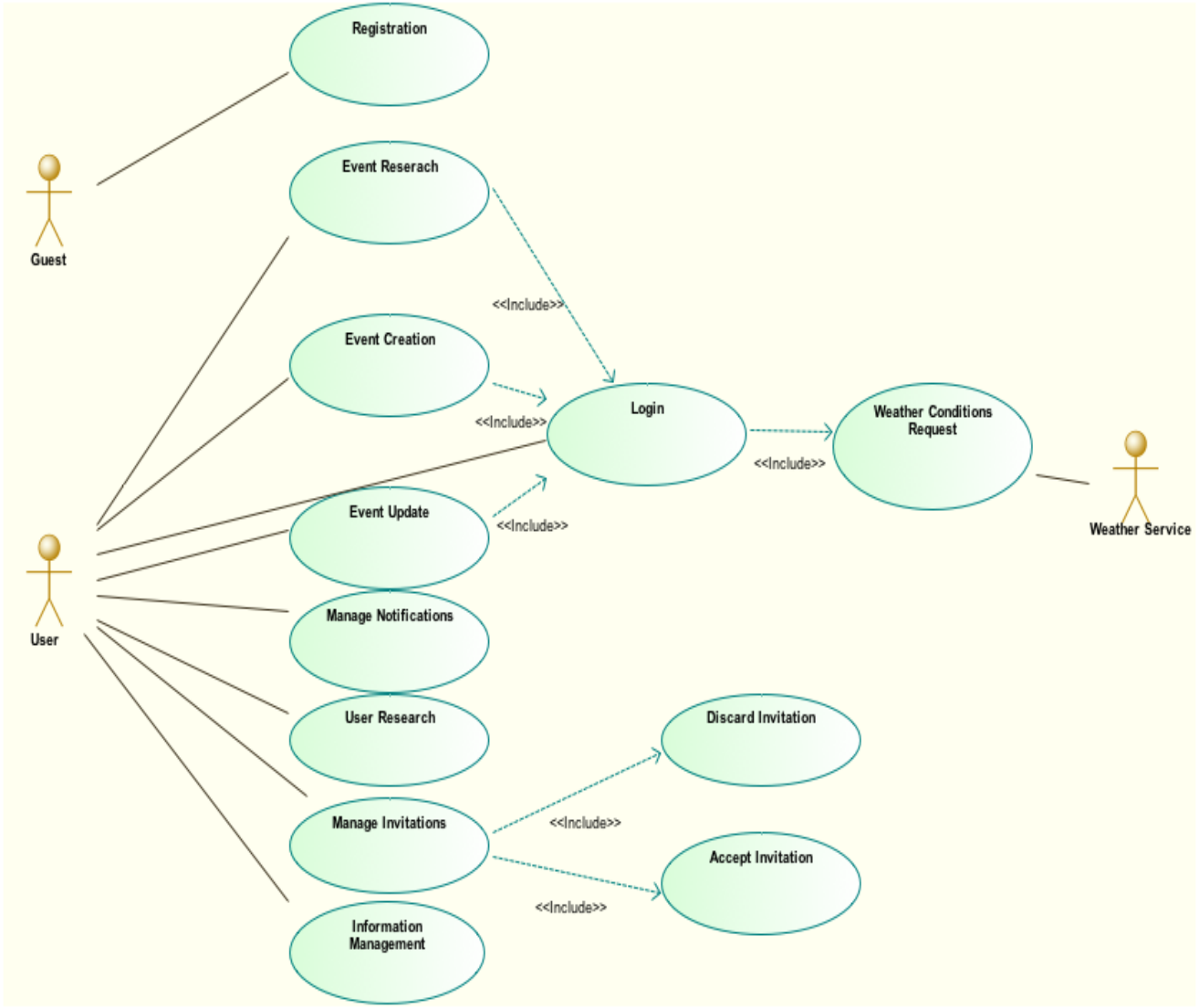
1. The forecast sequence associated to an event covers its duration.
2. The forecasts provided are the ones produced by the external weather forecast service for the given location and time (associated to weather conditions not represented here for clarity).

Use Case Analysis

A detailed version of the use cases is reachable in the RASD, here we want to take just a glance focusing on the main aspects.

We start by showing an overview diagram. Later in the consecutive pages, some use cases that were saw fit are analyzed in detail.

Use Case Analysis – Overview Diagram



Use Case Analysis – Registration

Actor: Guest;

Input conditions: a Guest chooses to sign up;

Output conditions: The Guest is granted access to a User account;

Events flow:

- The Guest is brought to a page that contains the registration form;
- Filling in the form's fields with an username and a password;
- The system tries to update its database with these new data;
- If so Guest can login the system with his own account. Otherwise in case of any error has arised, he has to repeat this procedure.

Use Case Analysis – Event Creation

Actor: User;

Input conditions: The User is already logged in the system;

Output conditions: His calendar contains the just created event;

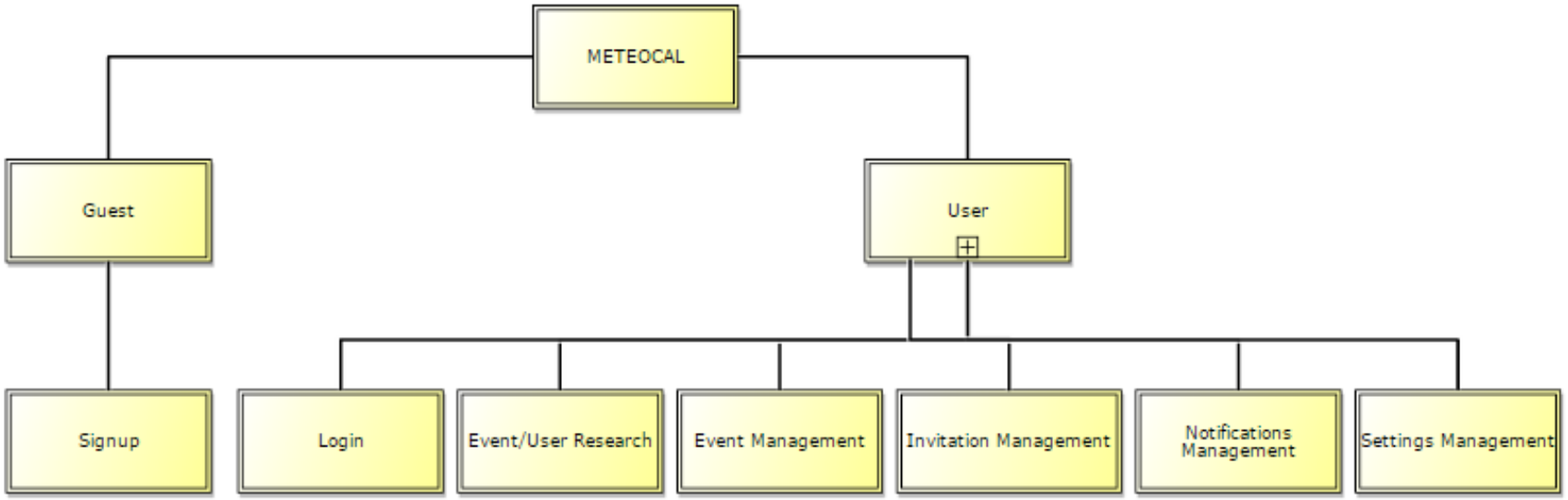
Events flow:

- The User starts to create a new event thanks to the topbar;
- The system provides a new form, and the User has to fill the fields with the information about the new event;
- Once the User confirms the operation, the system does the required checks and operations for the event;
- If so the User is brought to the created event page, otherwise if occurs any error the User can repeat this operation.

Design

Architectural Description – Sub Systems

To better define the domains involved in our system, we decide to report here a top-down approach that will describe at a hight-level the subsystems our application is based on, and their relation with the actors interacting with the system.



Analysis of the previous scheme.

- There are two main subsystems, the first is related to the Guest the second involves the User.
- As it is easily observable, the guest can do only a simple operation of signing up. Thus, if he tries to make a different operation that requires the privileges of the User the system prevents this attempt.
- On the other hand, the User has at his disposal all the functionalities offered by the system for managing events, user informations and the relationships with other users. In the same way, the system prevents attempts of signups made by Userwith checks on the username during the registration.

We can than conclude that, this abstract separation in subsystems takes place in the real system in particular thanks to the usage of filters, security checks, and a strong splitting among functionalities.

As we will see in the next pages, our system has the following general modules:

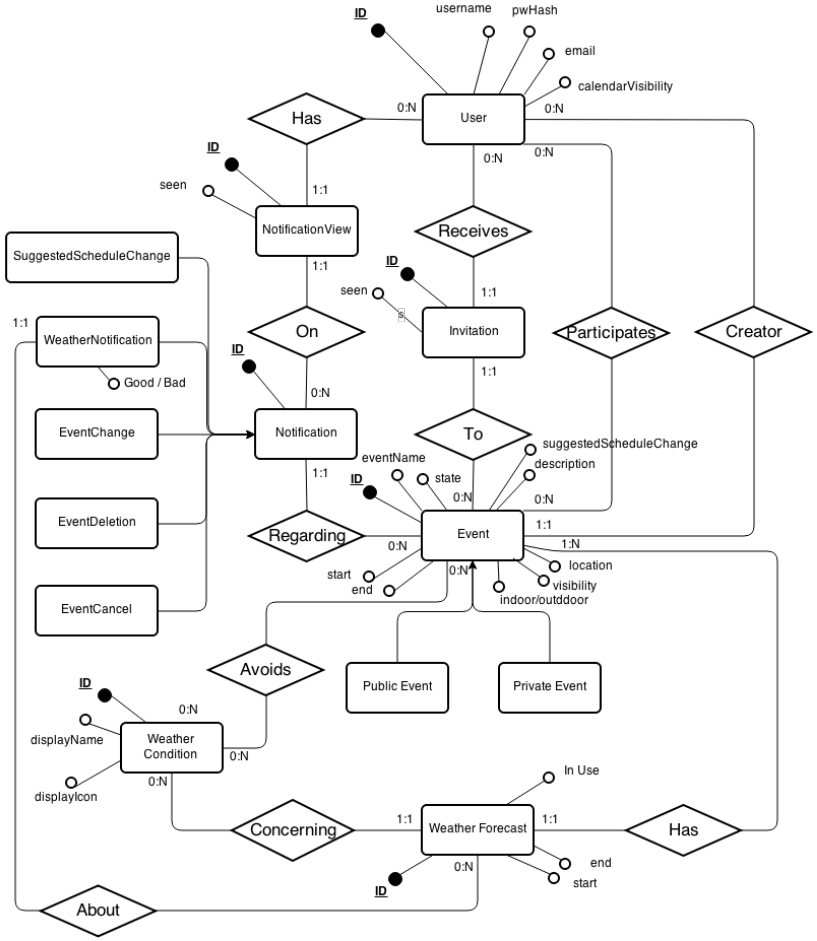
- View (xhtml + css)
- Web-Beans
- Facade-Interface
- FacadeImplementation-Beans
- DAO

Persistent Data Management

Our application will make use of a relational database for persistent data management. So, starting from the specifications described in the previous section, we started designing it by following the process reported in the following slides.

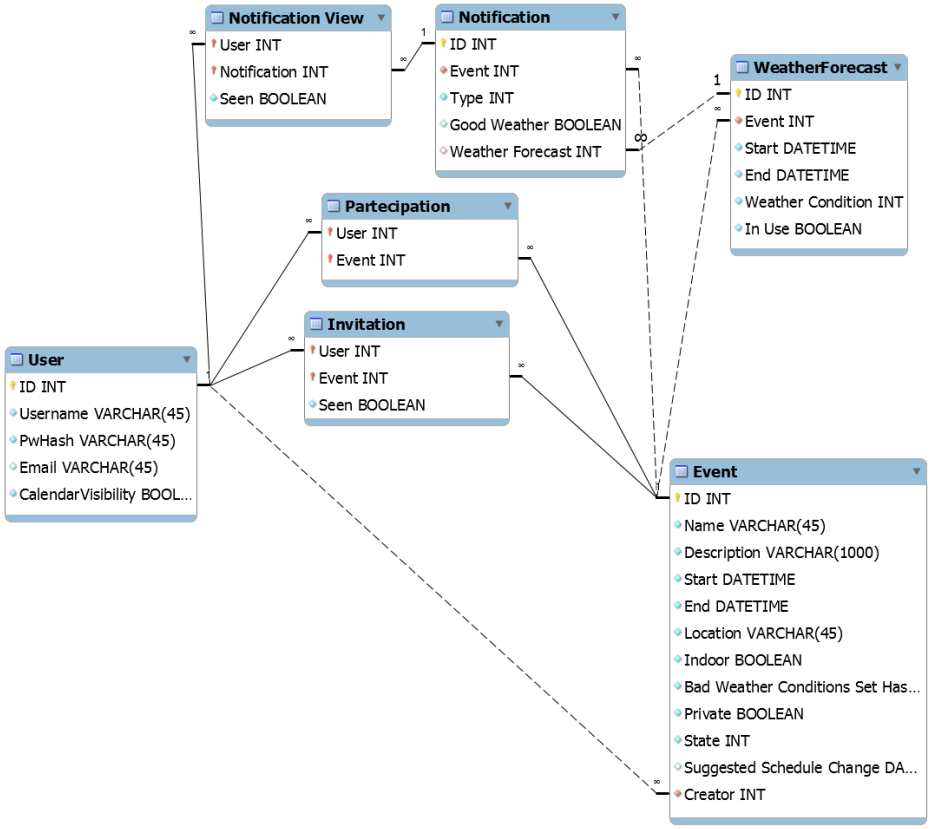
Persistent Data Management – Conceptual Design

Production of an ER diagram describing the conceptual design of the data structure:



Persistent Data Management – Logical Design

Translation of the ER diagram into an actual relational database description by expressing the relationships between entities through the usage of foreign keys and and join tables.



Persistent Data Management – Logical Design

In our implementation, the database structure is actually created using JPA. This allowed us to benefit of all the tools the API offer, while also define some flexible behaviours for field conversions between Java objects and database entries.

Persistent Data Management – Logical Design

An extract of the User table definition:

```
@Entity
@Table(name = TableDictionary.TABLE_USER)
public class User {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    @Column(unique = true)
    @NotNull(message = "Username cannot be empty")
    private String username;

    @NotNull(message = "Password cannot be empty")
    private String password;

    @Pattern(regexp = EMAIL_PATTERN,
            message = "invalid email")
    private String email;

    @OneToMany(mappedBy = "creator")
    private List<Event> createdEvents;

    @ManyToMany(mappedBy = "invited", fetch = FetchType.EAGER)
    private List<Event> invitedTo;

    @OneToMany(mappedBy = "user")
    private List<Invitation> invitations;
```

Persistent Data Management – Logical Design

JPA allows the creation of custom converters. For example, the following converter is used to efficiently convert the set of adverse conditions chosen for each event into an integer hash (representing the bitmask of the set).

```
/**
 *
 * @author Andrea Bignoli, inspired from code by Steven Gertiser
 */
@Converter(autoApply = true)
public class LocalDateTimePersistenceConverter implements
        AttributeConverter<LocalDateTime, java.sql.Timestamp> {

    @Override
    public java.sql.Timestamp convertToDatabaseColumn(LocalDateTime entityValue) {
        if (entityValue == null) {
            return null;
        }

        return Timestamp.valueOf(entityValue);
    }

    @Override
    public LocalDateTime convertToEntityAttribute(java.sql.Timestamp databaseValue) {
        if (databaseValue == null)
            return null;

        return databaseValue.toLocalDateTime();
    }
}
```

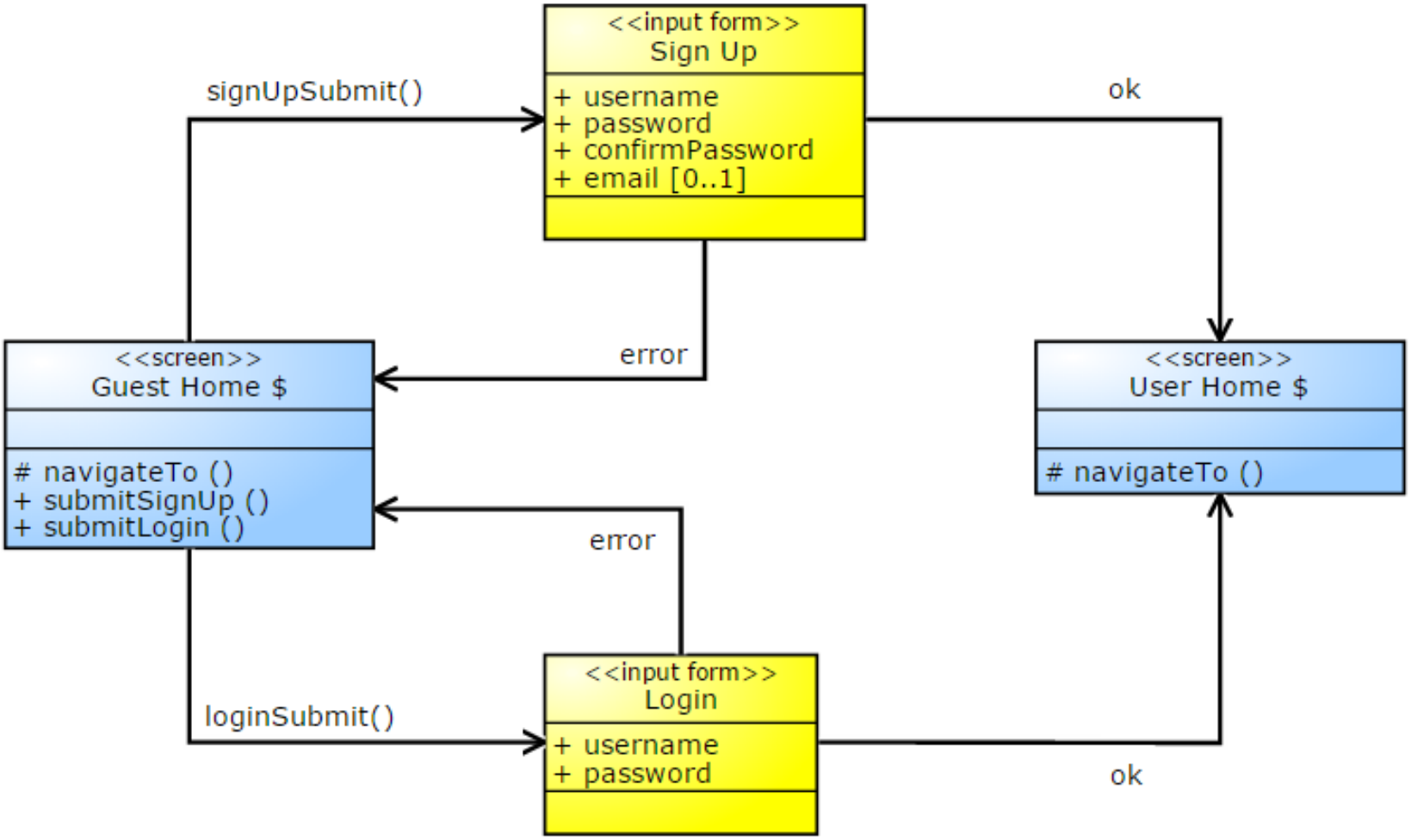
User Experience – Introduction

In this section we are going to describe the user interface level of our application, we will focus the attention in the main aspects.

For each diagram we will show the relation that holds among:

- The User Experience diagrams,
- The real pages, with the some code particularly relevant.

User Experience – Guest



Correspondence between the diagram and pages.

- The GuestHome takes place with the Index page: it contains the two forms, one for the registration and the other for the login. In case of error the user stays in the same page and an error message is shown.

- The UserHome is realized with the HomeCalendar page: it contains the topbar that groups the main funtionalities as will be described in the next diagrams, and the connected user's calendar.

```
<f:event listener="#{filterCalendarVisible.check}" type="preRenderView" />
```

User Experience – Code

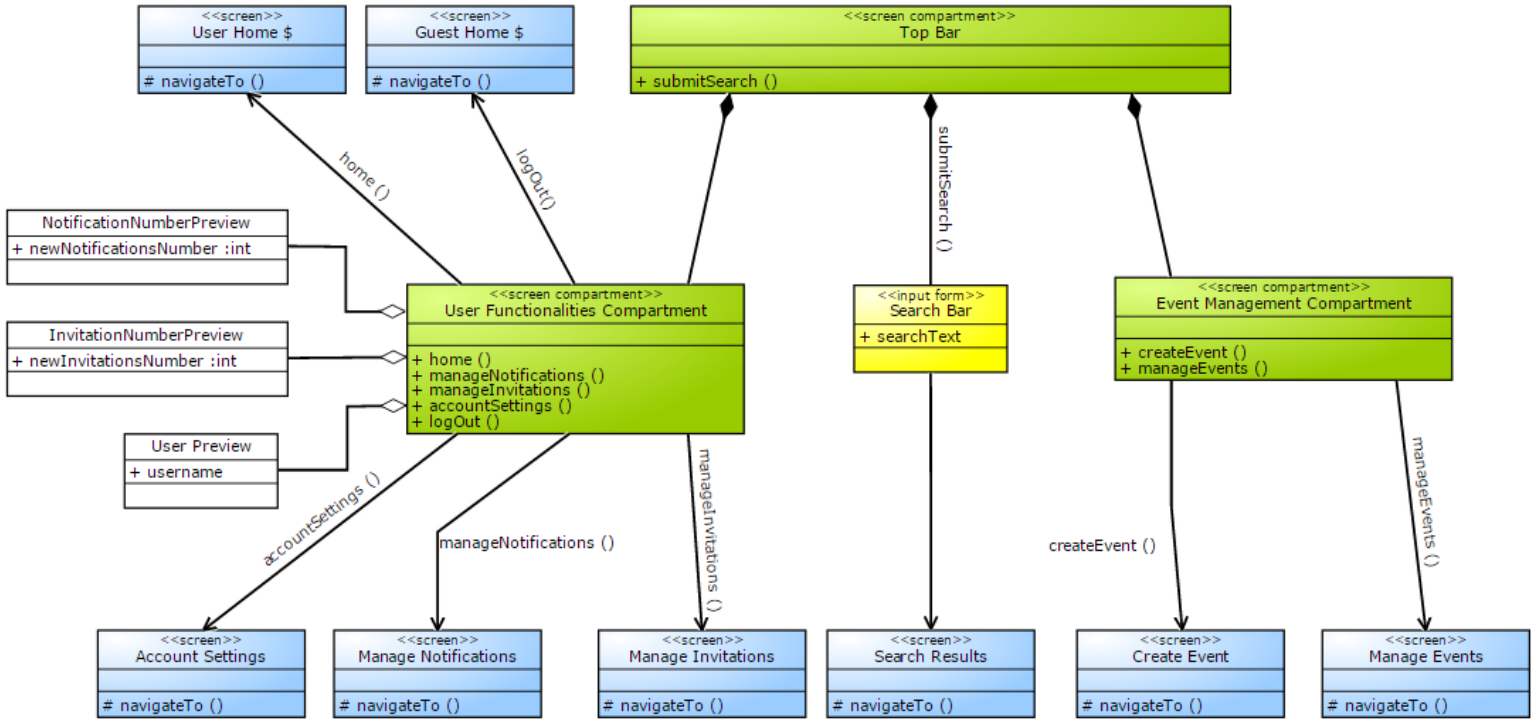
```
if (isNotLogged()) {
    FacesContext fc = FacesContext.getCurrentInstance();
    fc.getApplication().getNavigationHandler().handleNavigation(fc, null, errorOutcome);
}

if (userB.equals(loggedUser.getUsername())) {
    FacesContext fc = FacesContext.getCurrentInstance();
    fc.getApplication().getNavigationHandler().handleNavigation(fc, null, myCalendarOutcome);
    return;
}

try {
    visibility = um.getVisibilityOverUser(uf.findByUsername(userB).getId());
}
catch (NotFoundException ex) {
    FacesContext fc = FacesContext.getCurrentInstance();
    fc.getApplication().getNavigationHandler().handleNavigation(fc, null, errorOutcome);
    return;
}

if (visibility == UserUserVisibility.NOT_VISIBLE) {
    FacesContext fc = FacesContext.getCurrentInstance();
    fc.getApplication().getNavigationHandler().handleNavigation(fc, null, noVisibleOutcome);
}
else {//VISIBLE
    FacesContext fc = FacesContext.getCurrentInstance();
    fc.getApplication().getNavigationHandler().handleNavigation(fc, null, visibleOutcome);
}
```

User Experience – Topbar



The Top Bar included in every user page of our platform:

Correspondence between the diagram and the page.

- This is the component that will be included in each page of the User Subsystem. As we have already said, it contains all the functionalities offered by the systems, it is like a bridge from the current page to the desired one.

- It doesn’t contain only links to other pages, but also a search textarea that allows the research of user and events.

This is realized following the definition of component, in fact we have only one page that defines this module, all the other includes this one.

User Experience – Code

```
<ui:composition  xmlns="http://www.w3.org/1999/xhtml"
                 xmlns:f="http://java.sun.com/jsf/core"
                 xmlns:h="http://xmlns.jcp.org/jsf/html"
                 xmlns:ui="http://java.sun.com/jsf/facelets"
                 xmlns:p="http://primefaces.org/ui">
    <h:head>
        <title></title>
    </h:head>
    <h:form>

        <p:menubar>
            <p:menuitem value="#{topbarBean.username}" url="/protected/personal/HomeCalendar.xhtml" icon="ui-icon-home"/>

            <f:facet name="options">
                <p:inputText style="margin-right:20px" placeholder="Search"  value="#{searchBean.searched}"/>
                <p:commandButton action="#{searchBean.search()}" type="submit" icon="ui-icon-search" ajax="false"/>
            </f:facet>

            <p:menuitem value="#{topbarBean.notificationNumber}" url="/protected/personal/ManageNotifications.xhtml" icon="ui-icon-flag"/>
            <p:menuitem value="#{topbarBean.invitationNumber}" url="/protected/personal/ManageInvitations.xhtml" icon="ui-icon-mail-closed"/>

            <p:submenu label="Events" >
                <p:menuitem value="Create" url="/protected/event/EventCreate.xhtml" icon="ui-icon-document" />
                <p:menuitem value="Manage" url="/protected/personal/ManageEvents.xhtml" icon="ui-icon-script" />
            </p:submenu>
            <p:submenu label="Account" >
                <p:menuitem value="Settings" url="/protected/personal/Settings.xhtml" icon="ui-icon-gear" />
                <p:menuitem value="Logout" action="#{loginBean.logout()}" ajax="false" icon="ui-icon-power"/>
            </p:submenu>

        </p:menubar>

    </h:form>


</ui:composition>
```

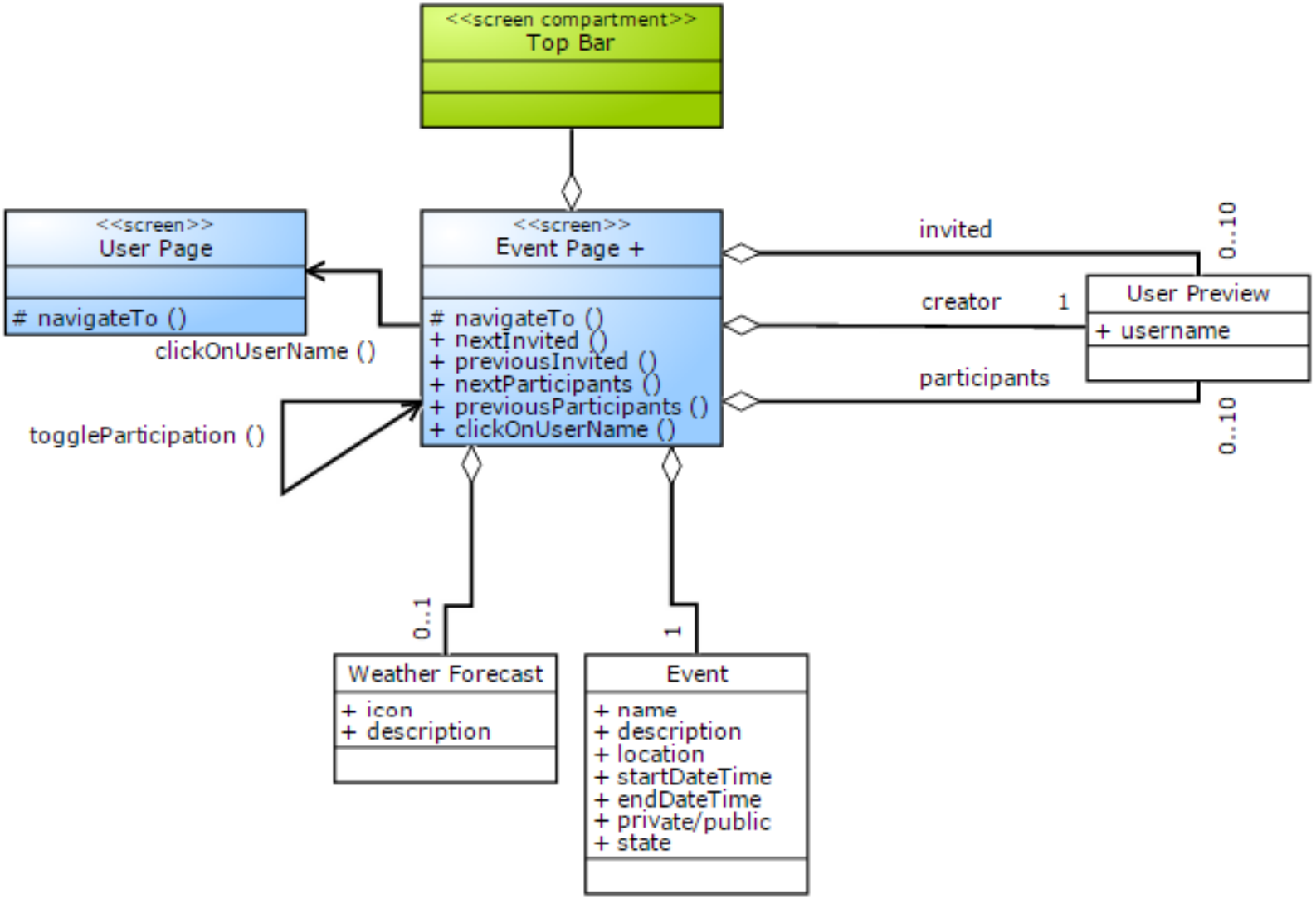
User Experience – Event Page

Now we will focus on one of the page with the main role in our system.

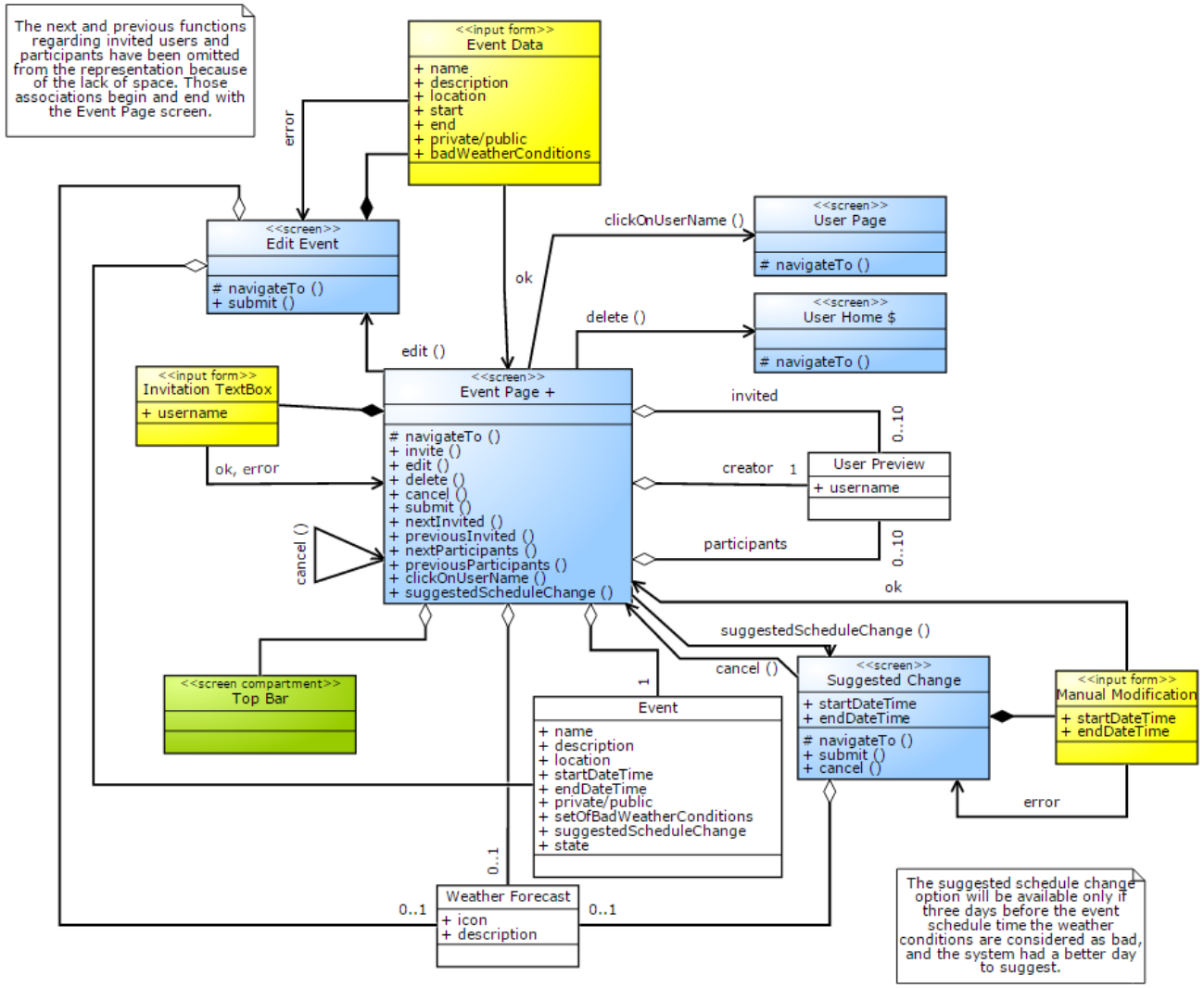
This page is not the page that contains all the details about the event, but is the page that has to evaluate the event relation over the connected user, and according to this results it dispatches the user to the right page.

We have made also complementary filters on the destination pages. So for example, if one tries to directly connect to the creator page even if he’s not the creator of the event, the systems prevents this operation.

User Experience – User Visibility



User Experience – Creator Visibility



User Experience – Filter's code

```
if (isNotLogged()) {
    SYSO_Testing.syso("I'm not logged");

    FacesContext fc = FacesContext.getCurrentInstance();
    fc.getApplication().getNavigationHandler().handleNavigation(fc, null, errorOutcome);

    return;
}
else {
    String username = loggedUser.getUsername();
    try {
        visibility = um.getVisibilityOverEvent(eventID);
        sessionUtility.setParameter(eventID);

        SYSO_Testing.syso("FilterEvent. Username " + username);
        SYSO_Testing.syso("FilterEvent. I'm logged, and I've to check the visibility");
        if (visibility == CREATOR) {
            FacesContext fc = FacesContext.getCurrentInstance();
            sessionUtility.setParameter(eventID);
            fc.getApplication().getNavigationHandler().handleNavigation(fc, null, creatorOutcome);
            return;
        }
        else {
            if (visibility == VIEWER) {
                FacesContext fc = FacesContext.getCurrentInstance();
                sessionUtility.setParameter(eventID);
                fc.getApplication().getNavigationHandler().handleNavigation(fc, null, viewerOutcome);
                return;
            }
            else {// NO VISIBILITY
                FacesContext fc = FacesContext.getCurrentInstance();
                fc.getApplication().getNavigationHandler().handleNavigation(fc, null, noVisibilityOutcome);
                return;
            }
        }
    }
    catch (NotFoundException ex) {
        error.setMessage("There is an incompatibility between you and the required event");
        FacesContext fc = FacesContext.getCurrentInstance();
        fc.getApplication().getNavigationHandler().handleNavigation(fc, null, errorOutcome);
        return;
    }
}
```

User Experience

In summary the events subsystem within the User domain is mapped with the following pages:

- EventPage
- EventPage As Creator
- EventPage As Viewer
- EventPage No Visibility

As was mentioned before, each page contains its filter except for the one that represents the case in which the user hasn’t visibility over an event, to allow perfectly safe resource management.

BCE – Introduction

Now we have decided to move back to the design representation, in particular here we will illustrate the BCE diagrams.

We do it for representing how we have respected the Model-View-Controller pattern with the schemes, knowing that:

- Boundary: represents the view
- Control: represents the controller
- Entity: represents the model

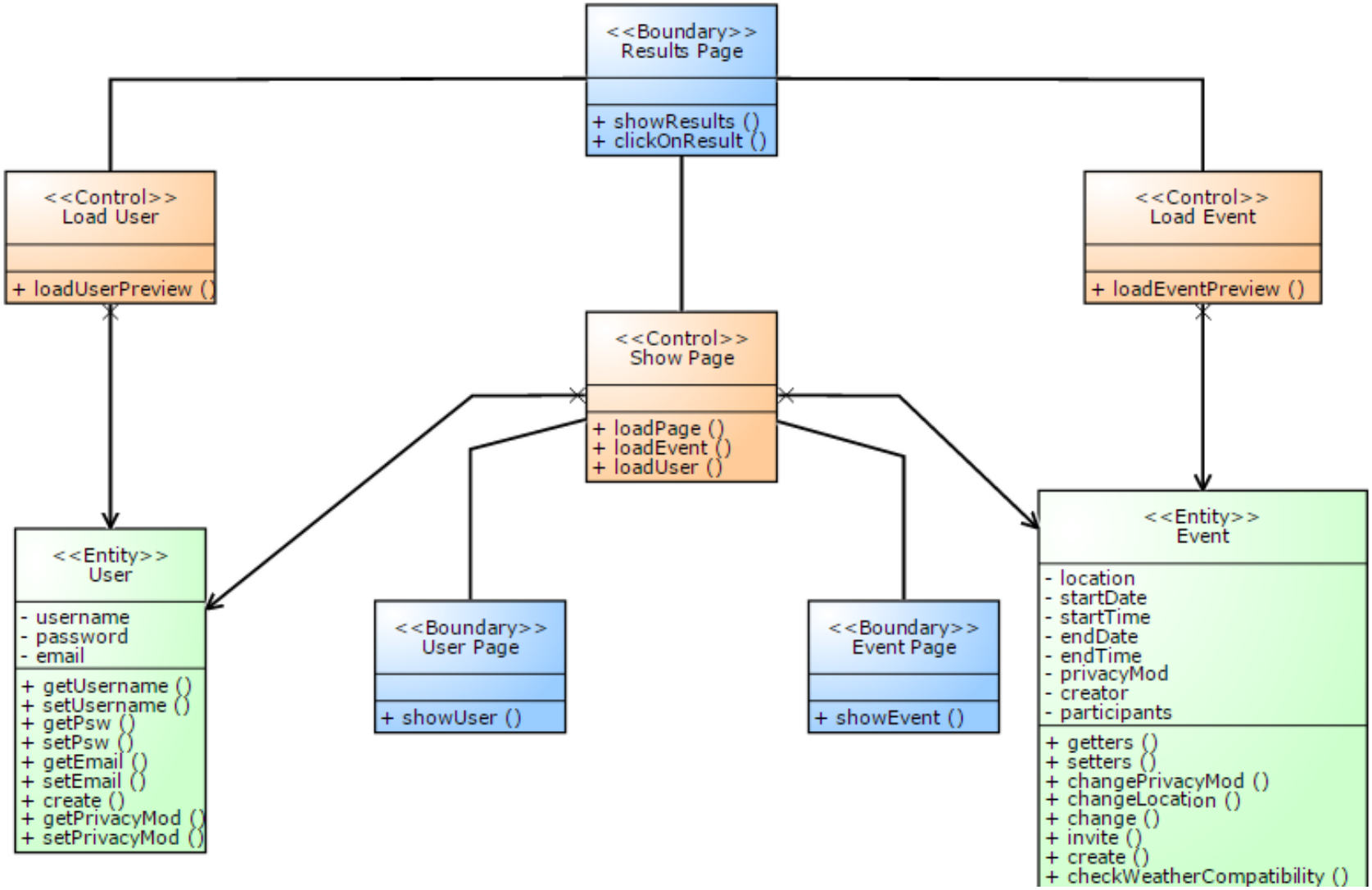
BCE

Our purpose is not to replace the same detailed analysis made in the DD. We want to show more relations among models, that could not be mapped before the developing, and then there is not in the other documents.

Now we show the relation between the Controls in the BCE diagrams that was done before the developing and the beans.

We will do it not focusing once more on the already discussed modules, but moving a little along the not described components even if they may appear less important.

BCE – Search Bar



BCE – Search Bar

This module was touched on before just one time, but it is interesting. Its purpose consists of making a list of results splitted in events and users, according to the word that the connected user is looking for.

The association betwen the diagram and the real system is shown in the next slide.

BCE – Search Bar Mapping

```
    public String search() {

        setEvents(ev.search(searched));
        setUsers(uf.search(searched));

        return resultsOutcome;
    }

    public List<User> getUsers() {
        return users;
    }

    public void setUsers(List<User> users) {
        if (this.users == null) {
            this.users = new ArrayList<User>();
        }
        for (User user : users) {
            this.users.add(user);
        }
    }

    public List<Event> getEvents() {
        return events;
    }

    public void setEvents(List<Event> events) {
        if (this.events == null) {
            this.events = new ArrayList<Event>();
        }
        for (Event event : events) {
            this.events.add(event);
        }
    }

}
```

This is the source code of the SearchBean.
We point out the following functions:

- search( ) : it loads the two sets of elements according to the *searched* word.
- *set/set user( )*
- *get/set event( )*

They translate the respectively functionalities:

- loadPage
- loadEvent
- loadUser

BCE – Search Bar Mapping

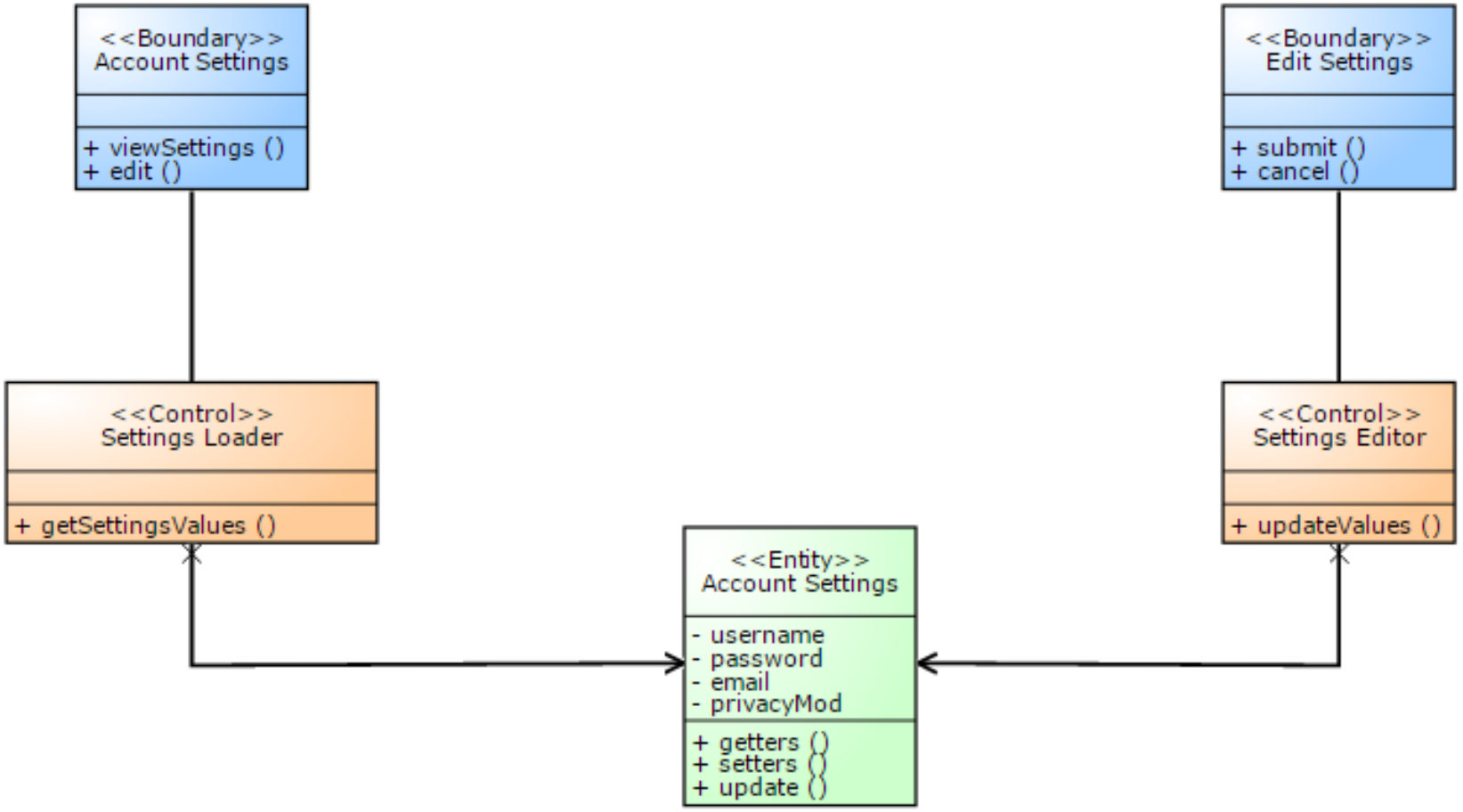
```
<div id="container">
    <div id="left">
        <p:panel id="users" header="List of compatible Users" style="margin-top:70px" class="panelForm">
            <p:dataList value="#{searchBean.users}" var="user">
                <p:link value="#{user.username}" outcome="/protected/user/UserCalendar.xhtml?username=#{user.username}">
                    <f:param name="username" value="#{user.username}"/>
                </p:link>
                <p:separator />
            </p:dataList>
        </p:panel>
    </div>
    <div id="right">
        <p:panel id="events" header="List of compatible Events" style="margin-top:70px" class="panelForm">
            <p:dataList value="#{searchBean.events}" var="event">
                <p:link value="#{event.name}" outcome="/protected/event/EventPage.xhtml?eventID=#{event.id}">
                    <f:param name="eventID" value="#{event.id}"/>
                </p:link>
                <p:separator />
            </p:dataList>
        </p:panel>
    </div>
</div>
```

This is the xhtml page that actualize the ResultsPage, in fact it has also the same name.

From this page is possible to move to:

- The Event Page;
- The User Page.

BCE – Settings



BCE – Settings

This module was never touched on before. Its purpose consists of loading the current setting values, and allows their editing.

As is represented in the scheme, in the real application we have:

- Two pages: one for the settings loading and the other for the modification.
- They are based on their respectively beans, which work on the same entity.

In the next pages we show this mapping.

BCE – Settings Mapping

```
<div id="main">
    <h:form id="form">
        <p:panelGrid columns="2">

            <h:outputText value="Username:" />
            <h:outputText value="#{settingsBean.username}" />

            <h:outputText value="Mail:" />
            <h:outputText value="#{settingsBean.email}" />

            <h:outputText value="Public calendar:" />
            <h:outputText value="#{settingsBean.privacyMod}" />

            <p:row>
                <p:column colspan="2">
                    <p:commandButton value="edit" action="/protected/personal/SettingsEdit?faces-redirect=true" type="submit"
                </p:column>
            </p:row>
        </p:panelGrid>

    </h:form>
</div>
```

This is the account settings page, and as was designed before it contains the functionality to edit the current settings.

And it relates to the settingBean for the data loading, that is equivalent to the SettingLoader.

BCE – Settings Mapping

```
<h:form>
    <p:panel id="privacyPanel" header="Changing Calendar Visibility Setting" style="margin-top:70px" class="panelForm">
        <h:panelGrid columns="2" style="margin-bottom:10px" cellpadding="5" id="settingEdit">
            <h:outputText value="Calendar Visibility: " />
            <p:selectOneButton value="#{settingsEditingBean.editedUser.calendarVisible}">
                <f:selectItem itemLabel="Visible" itemValue="#{!settingsEditingBean.editedUser.calendarVisible}" />
                <f:selectItem itemLabel="Hidden" itemValue="#{settingsEditingBean.editedUser.calendarVisible}" />
            </p:selectOneButton>

            <p:outputLabel value="Email" for="email" />
            <p:autoComplete id="email" value="#{settingsEditingBean.editedUser.email}" emptyMessage="not required">
            <f:validateRegex pattern="(^([a-zA-Z0-9_\-\.]+)@((\[[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.)|(([a-zA-Z0-9\-]+\.)+))([a-zA-Z]{2,4}|

            </p:autoComplete>

            <h:outputText value="Previous Password " />
            <p:password id="previousPassword" value="#{settingsEditingBean.previousPassword}" />

            <h:outputText value="New Password: " />
            <p:password id="newPassword" value="#{settingsEditingBean.editedUser.password}" feedback="true">
            <f:validateRegex pattern="(\w{4,})?" />
            </p:password>

            <h:outputText value="Password Confirmation: " />
            <p:password id="passwordConfirmation" value="#{settingsEditingBean.confirmationPassword}" feedback="true" />

            <p:commandButton value="Save" icon="ui-icon-check" action="#{settingsEditingBean.saveSettings()}" type="submit" ajax="false"/>
        </h:panelGrid>
    </p:panel>
</h:form>
```

This is the page that allow the insertion of the new values, it has a form for submitting them as the Edit Settings Boundary.

Code Structure and Highlights

Implementation Assignment

For the implementation phase of the project we decided to split the work among us as described here:

- Andrea Bignoli:
  - Business modules, database implementation, external weather forecast service interfaces, other back-end functionalities, some debugging and fixing activity in web tier
- Leonardo Cella:
  - Web tier, web pages and managed beans, resources and information supply according to privacy level (all security operations are managed by business tier)

Code Organization

We mirrored the previous description with the creation of:

- EJB Module:
  - Including all business operation related components
- WEB Module:
  - Including all web tier components and client pages

Data Access Object Pattern

To make full use of the tools that JPA offer to developers, we decided to integrate this pattern in our system. In particular this allow us to introduce a further level of abstraction in the access to the data contained in the database.

To actually implement this we created a basic DAO including operations common to each entity we manage (e.g. Find by primary key). On top of this we created individual DAOs for each entity (thanks to class inheritance) to define functionalities related only to the entity that specific DAO refers to.

Data Access Object Pattern

```java
public abstract class DAObase<T> {

    private final static String UNIT_NAME = "MeteoCalPU";

    @PersistenceContext(unitName = UNIT_NAME)
    private EntityManager em;

    private String tableName;
    private Class<T> databaseEntityClass;

    public DAObase(Class<T> databaseEntityClass, String tableName)
```

First lines of the basic Data Access Object.

```java
@Stateless
public class UserDAO extends DAObase<User> {

    public UserDAO() {
        super(User.class, TableDictionary.TABLE_USER);
    }

    public User findByUsername(String username) {
        Map<String, Object> parameters = new HashMap<String, Object>();
        parameters.put("username", username);
        String query = "select u from User u where u.username = :username";
        return super.findSingleResult(query, parameters);
    }
}
```

Specialization into User DAO by inheritance.

Interface Layers

To grant maximum maintainability and flexibility to our application, all the dependencies between the business tier and other components make use of interfaces.

For instance, every request coming from the Web Tier is actually directed to an interface defined for the facade that offers that service.

This weakens the component dependency constraints allowing greatly improved maintainability.

Interface Layers - Example

Consider for example the creation of an event, performed when the creator concludes the creation process on the web page. The EventCreationBean of the Web tier calls the create method of EventFacade, that manages event related operations:

```
public interface EventFacade {

    /**
     * Create a new event in the database based on data from the provided one.
     * The updated fields are:
     *
     * Name Description Country City Address Indoor flag Privacy flag Adverse
     * weather conditions set Start End
     *
     * The creator is the logged user, provided by the UserManager.
     *
     * The status is set to EventStatus.PLANNED.
     *
     * The creation includes a validation phase.
     *
     * @param e
     * @return the refernce to the saved event
     * @throws InvalidInputException If the validation phase fails.
     */
    public abstract Event create(Event e) throws BusinessException;
```

Interface Layers - Example

The execution is actually performed here:

```
@Stateless
public class EventFacadeImplementation implements EventFacade {

    public Event create(Event e) throws BusinessException {
        //....
```

This allows the maintainance of the create method. The only requirement is, in fact, to respect the JavaDoc description to respect the intended working.

Geographic Structure

During event creation the user can choose the location by selecting country and city from a list offered by the system.

The initial data can be found at this address:

http://www.unece.org/cefact/codesfortrade/codes_index.html

This data was converted in JSON format to be quickly loaded at startup by our application. The convertion was made using a Python script reported on the following slide.

Geographic Structure

```
import json

inputFile = open('CodeList.txt', 'r')
inputData = inputFile.readlines()
inputFile.close()

import json
class WorldEncoder(json.JSONEncoder):
    def default(self, obj):

        if isinstance(obj, World):
            return '{ "country" : ' + self.default(obj.countries) + '}'

        if isinstance(obj, list):
            result = '['
            count = 0

            for inner in obj:
                if(count > 0):
                    result += ','
                result += self.default(inner)
                count+=1

            result += ']'
            return result

        if isinstance(obj, Country):
            return '{ "name":"' + obj.name + '", "id" : "'
                + obj.c_id + '", "city" : ' + self.default(obj.cities) + '}'

        if isinstance(obj, City):
            return '{ "name":"' + obj.name + '", "countryId" : "' + obj.c_id + '"}'

        # Let the base class default method raise the TypeError
        return json.JSONEncoder.default(self, obj)
```

```
class World:
    def __init__(self):
        self.countries = []

class Country:
    def __init__(self, c_id, name):
        self.c_id = c_id
        self.name = name
        self.cities = []

class City:
    def __init__(self, c_id, name):
        self.c_id = c_id
        self.name = name

w = World()

for line in inputData:
    line = line[3:46]
    line = line.rstrip()

    prefix = line[:2]
    suffix = line[7:]

    if suffix[0] == '.':
        ## Country id
        if prefix in [c.c_id for c in w.countries]:
            print "Already existing"
        else:
            w.countries.append(Country(prefix,suffix[1:]))
    else:
        for saved_country in w.countries:
            if prefix == saved_country.c_id:
                saved_country.cities.append(City(prefix,suffix))

testOutput = open('world.json', 'w')
output = json.dumps(w, cls=WorldEncoder,
            encoding='latin1').replace('\\', '')[1:-1]
output.decode("utf-8")
testOutput.write(output)
testOutput.close()
```

External Weather Service Interface

The application make use of OpenWeatherMap. The service offers forecasts:

- For the next 5 days with a 3 hours precision
- For the next 16 days with a daily precision

Even though the first would be enough to provide a service that perfectly respects the requirements, combining the information of the two forecasts clearly allows a better result.

We decided to follow this approach then, even if this increased the work in development.

External Weather Service Interface

Our interface offers different functionalities to the rest of the application. The main ones are:

```
/**
 * If start before now but the end is not, the weather forecast sequence
 * will start with the closest available forecast to the current time.
 *
 * @param start
 * @param end
 * @param city
 * @param country
 * @return
 * @throws InvalidInputException
 */
public List<WeatherForecastBase> askForecast(LocalDateTime start, LocalDateTime end,
        String city, String country) throws InvalidInputException {...24 lines }
```

That returns a list of weather forecasts for the given location, covering the time interval delimited by start and end.

External Weather Service Interface

And:

```
/**
 * Returns null if no match is available.
 *
 * @param scheduledStart
 * @param scheduledEnd
 * @param forecastRange
 * @param adverseConditions
 * @return
 * @throws InvalidInputException
 */
public List<WeatherForecastBase> askClosestMatch(LocalDateTime scheduledStart, LocalDateTime scheduledEnd,
        String city, String country, EnumSet<WeatherCondition> adverseConditions) throws InvalidInputException
```

That given location and current scheduling for an event, returns the closest time frame of the same duration with forecasts not in conflict with the specified weather conditions to avoid.

This is used to efficiently produce suggestions for schedule changes in case of bad weather conditions.

External Weather Service Interface

An important thing to note is that the algorithms implemented here make the least assumptions possible about the structure of the data the external service provides.

For example, if OpenWeatherMap changes their forecasts range and accuracy, our application will interpret that data in the right way nevertheless. Obviously this doesn’t hold if they change the data structure actually returned, since we can’t do nothing about that possibility.

Other than that our forecast interface allows much more than what MeteoCal offers. It would be possible to provide weather forecasts on demand to the user (not only for events), without modifying anything outside of the web tier.

Automated Testing

Due to the strict time constraints the automated testing doesn't cover the whole application. We obviously can state though that, by the manual testing performed over the system, no anomaly has been observed. This process was also the base of our Test Cases document.

We focused mainly on Integration Testing using Arquillian. This type of tests covers for example, the whole operational cycle of our interfaces to the external service. This certify not only that it works, but also that it is coherent with the JavaDoc promises.

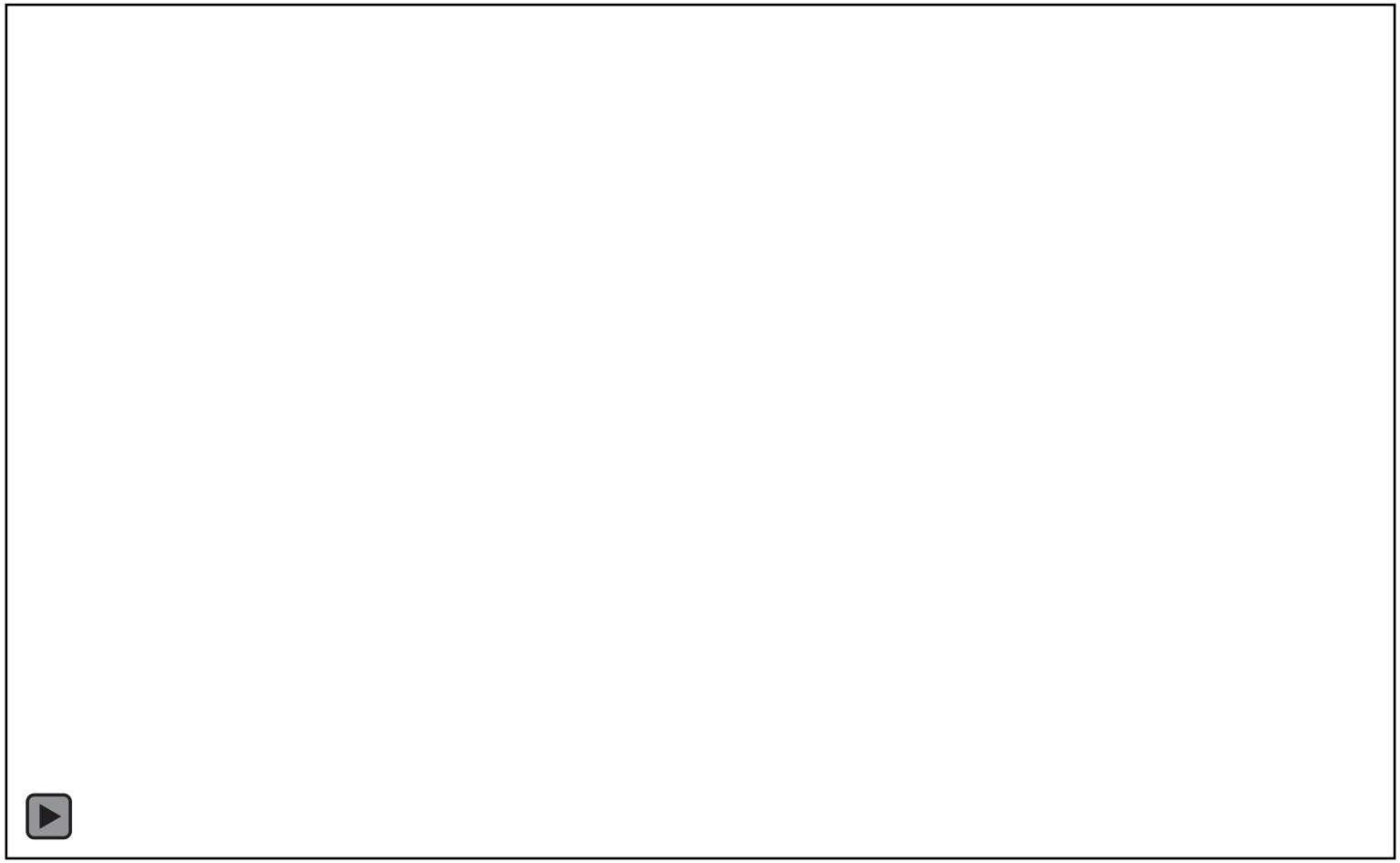
Demo

Demo

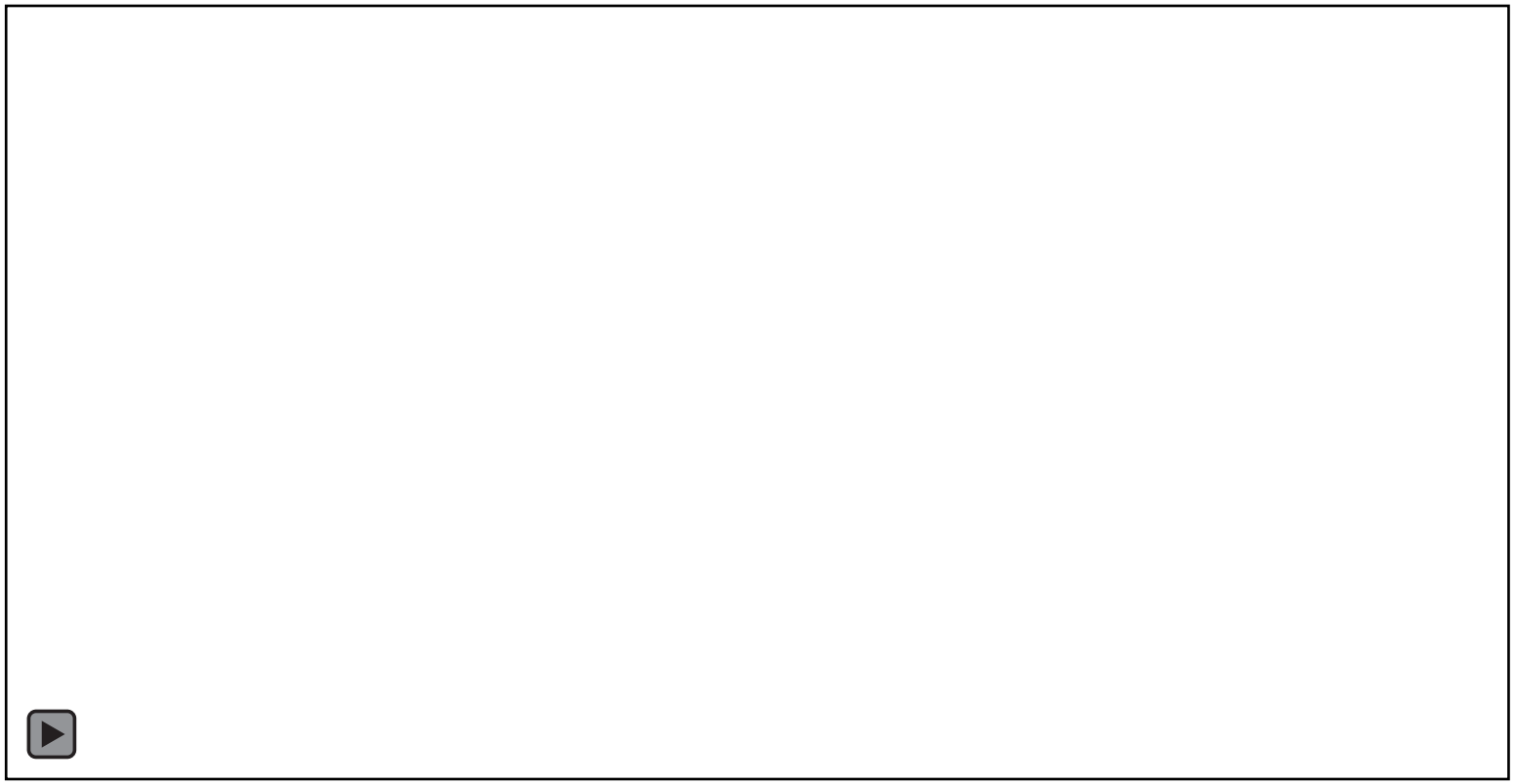
The following videos, will show some of the basic operations that can be performed in the system.

Naturally, due to the limited time available, the demo won’t cover all the functionalities that our platform offers. Any of the following can be tested by using the material provided in our repository by installing our system as explained in the Installation Manual.

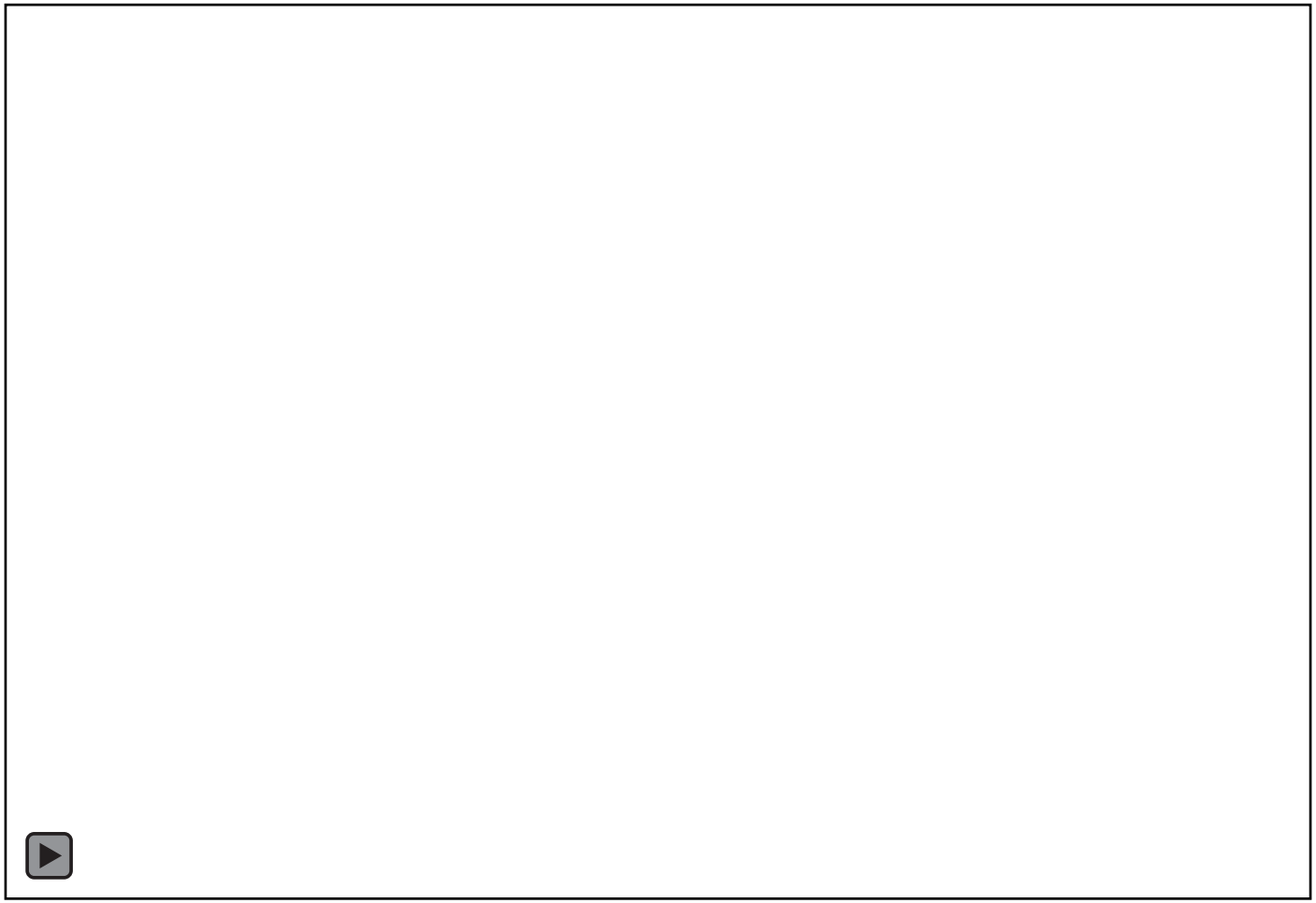
Demo - Registration



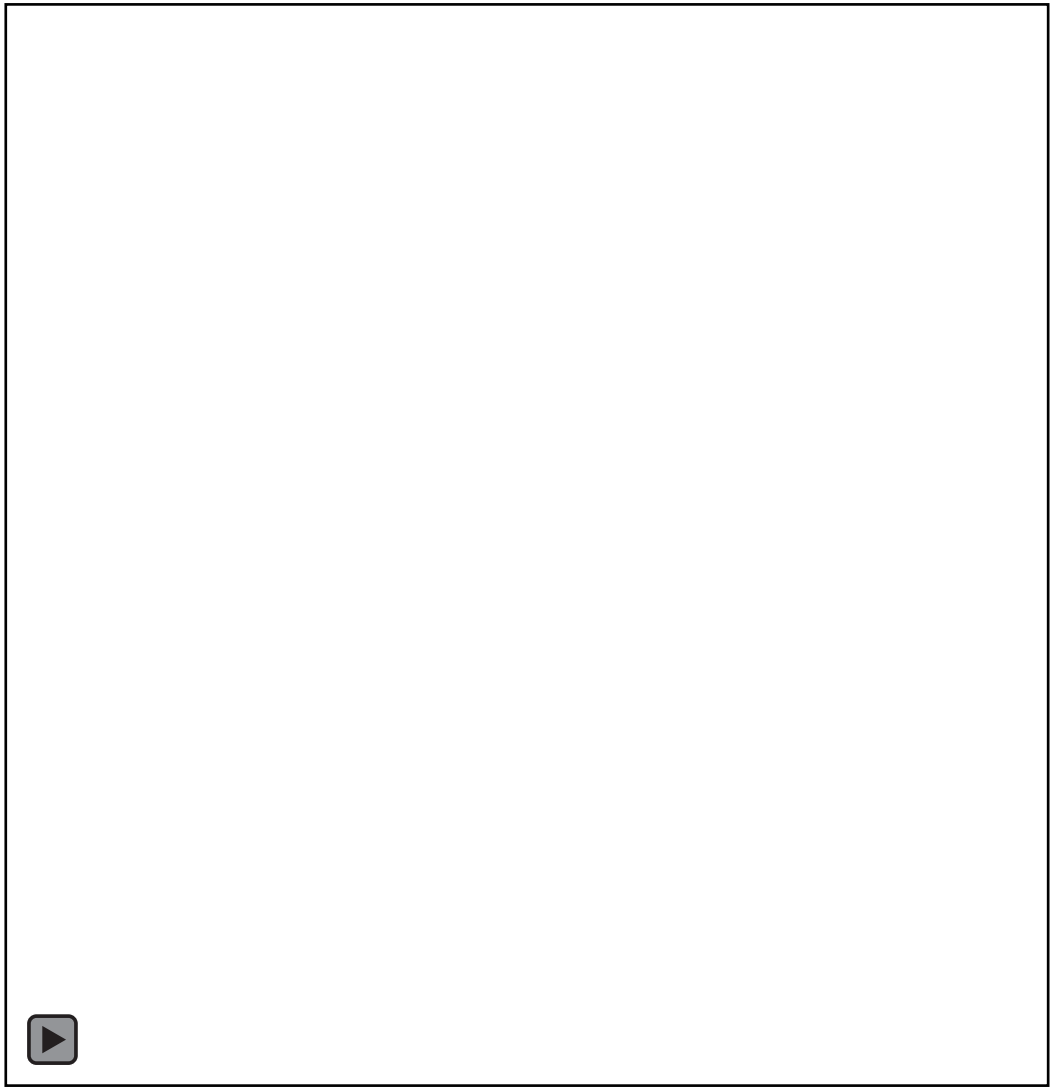
Demo – Registration with Error Handling



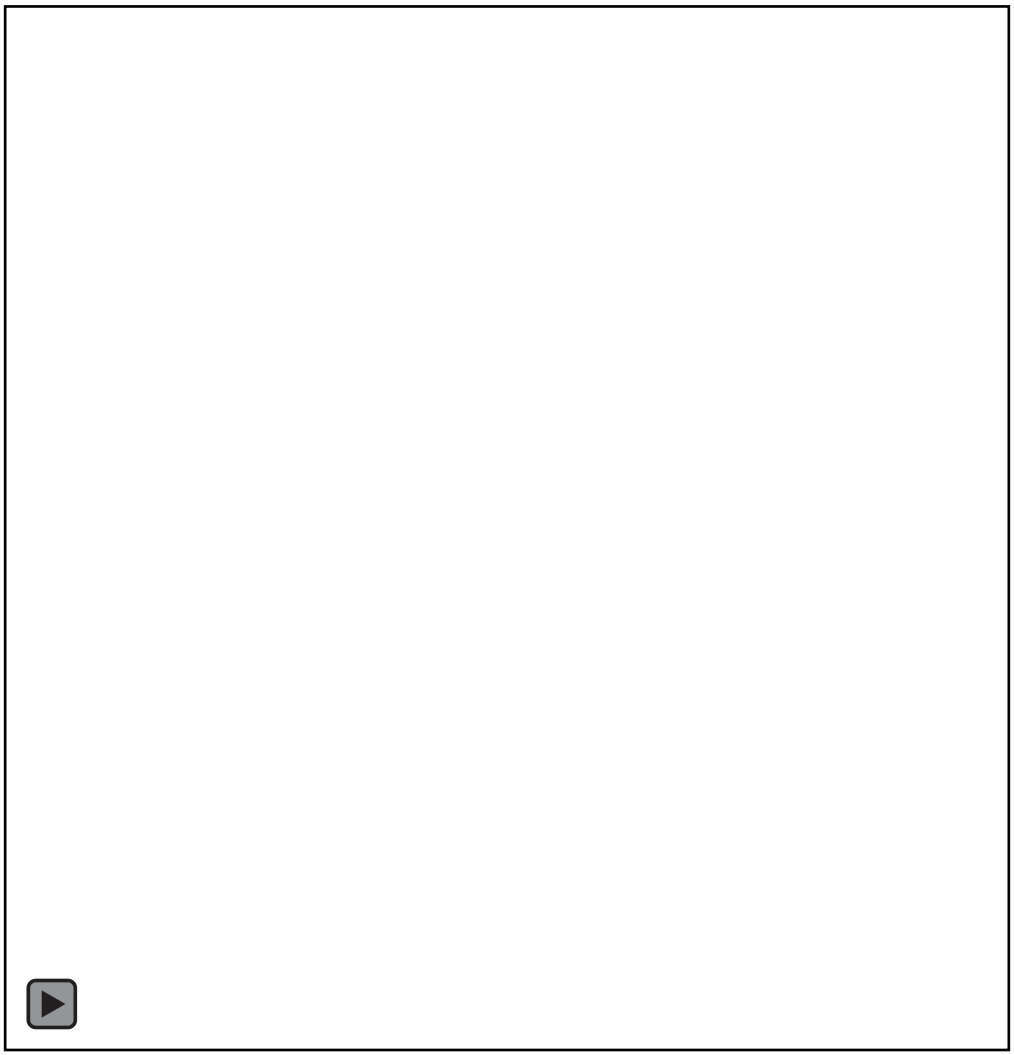
Demo - Login



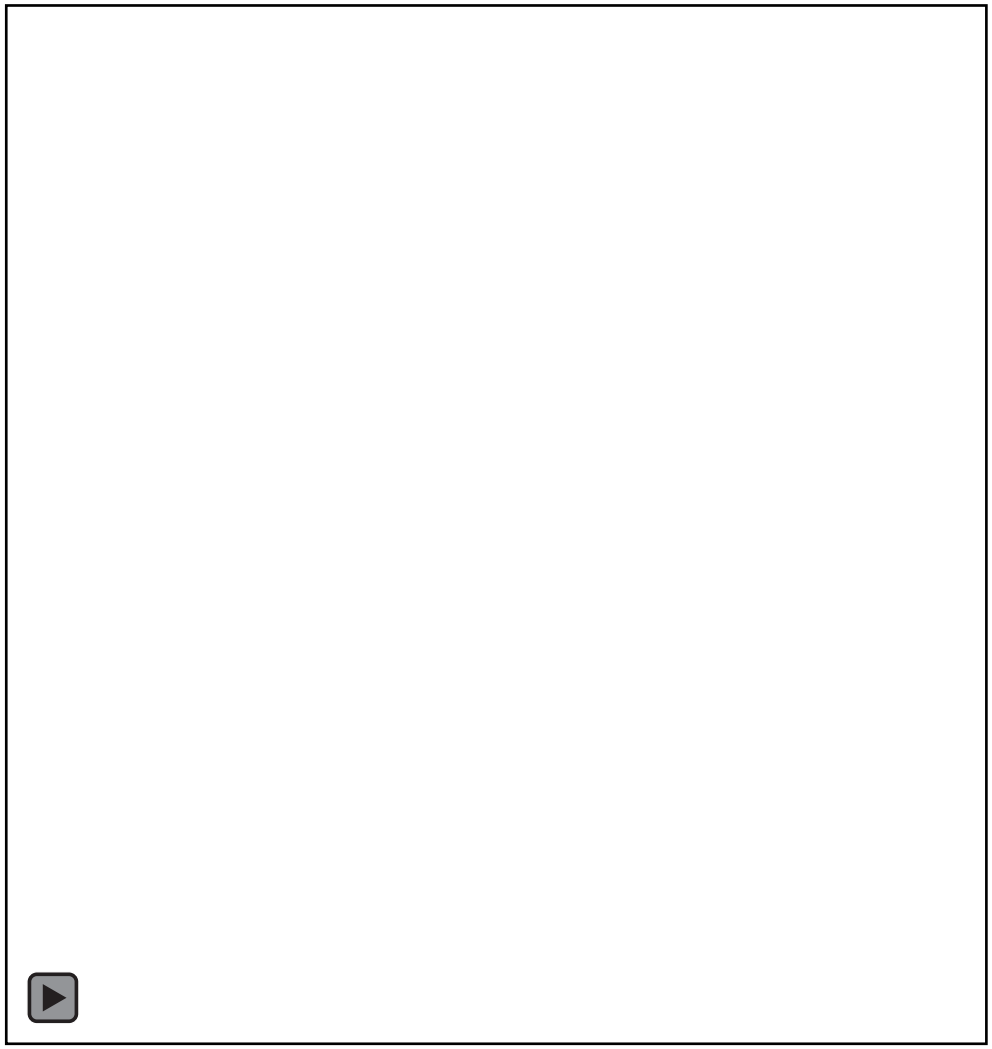
Demo – Create Event



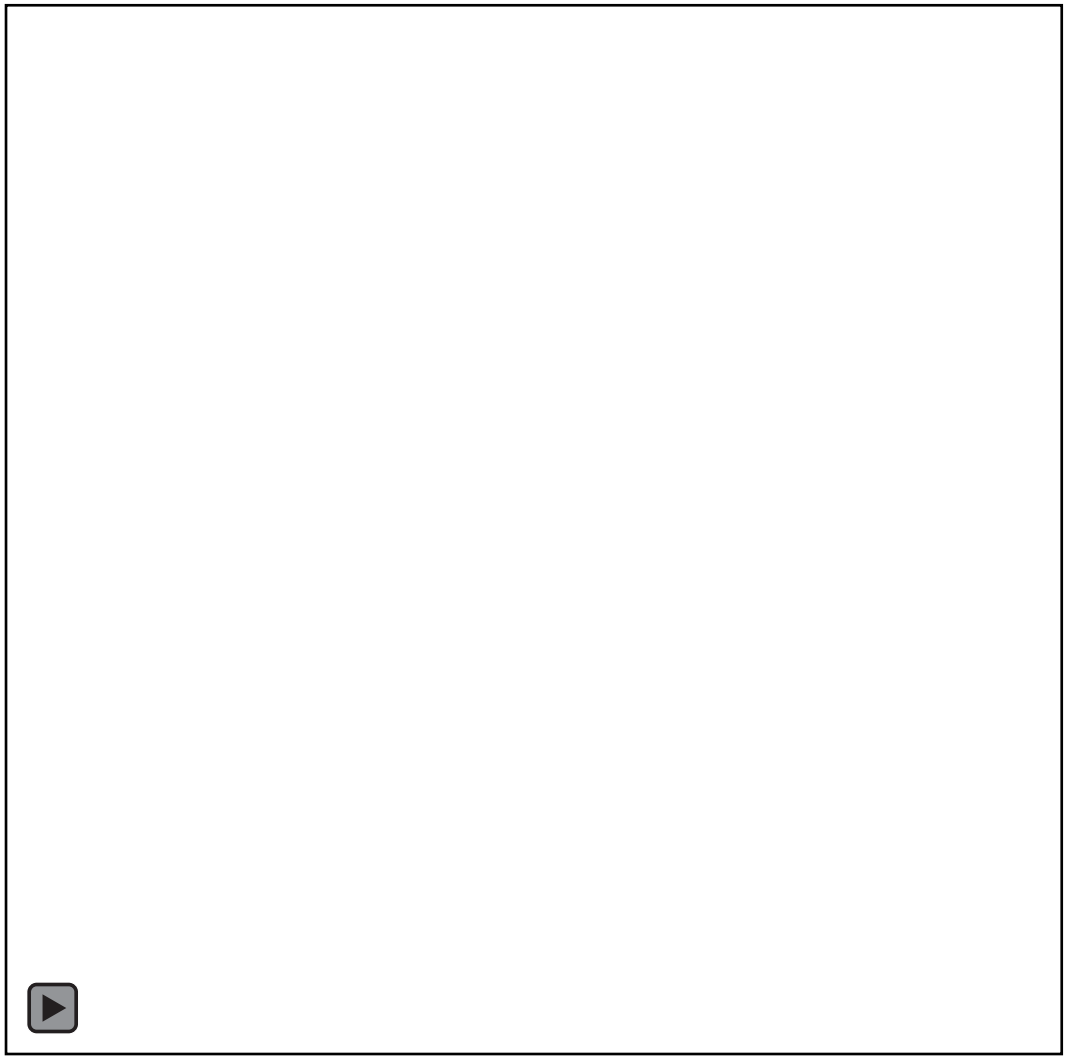
Demo – Invite More Users



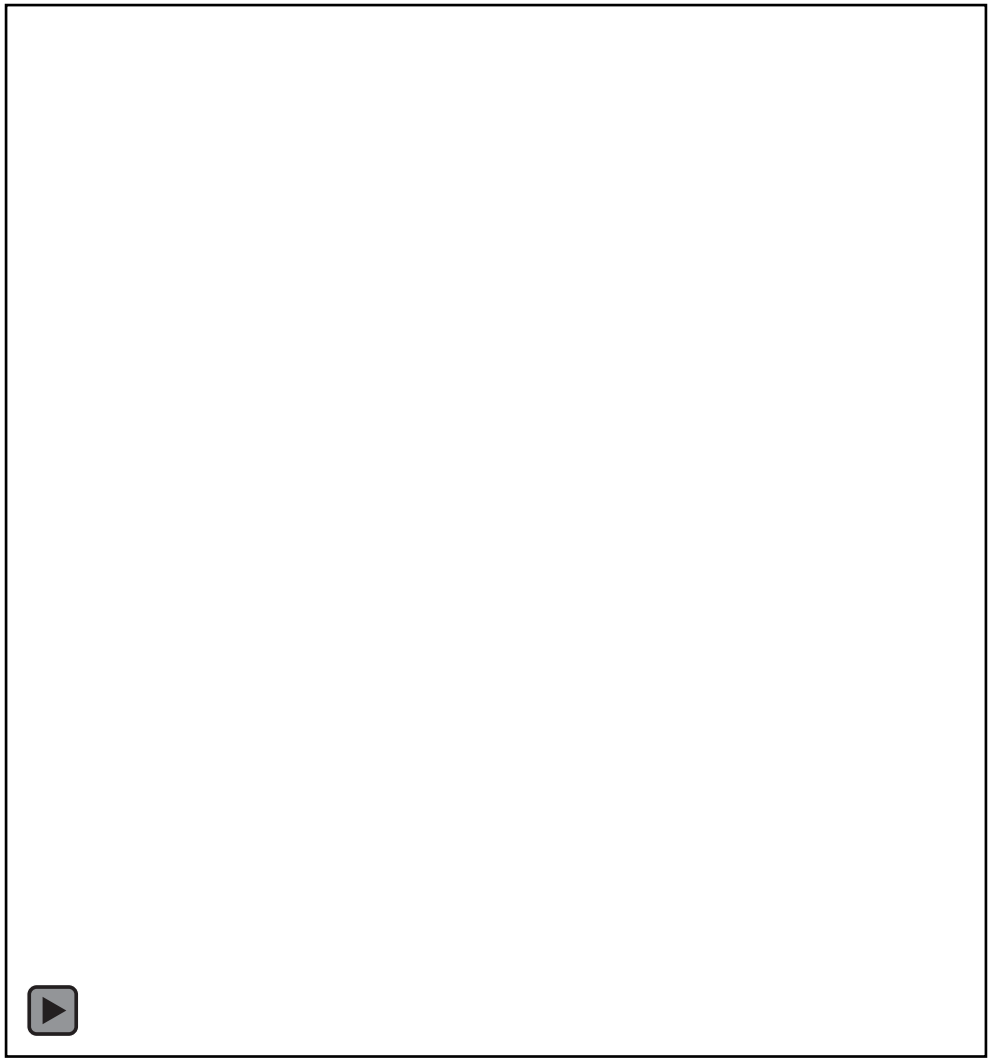
Demo – Decline an Invitation



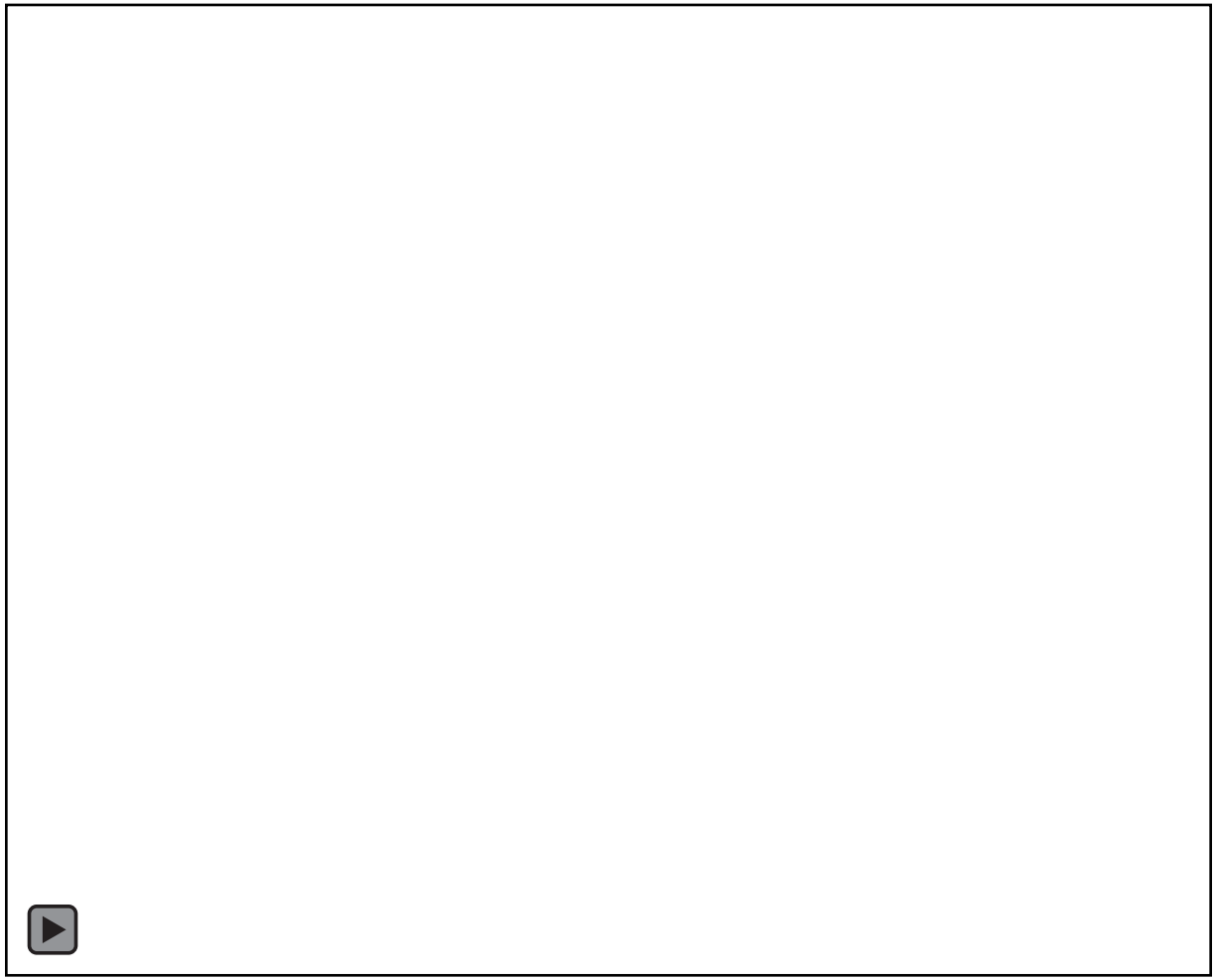
Demo – Join / Cancel Participation



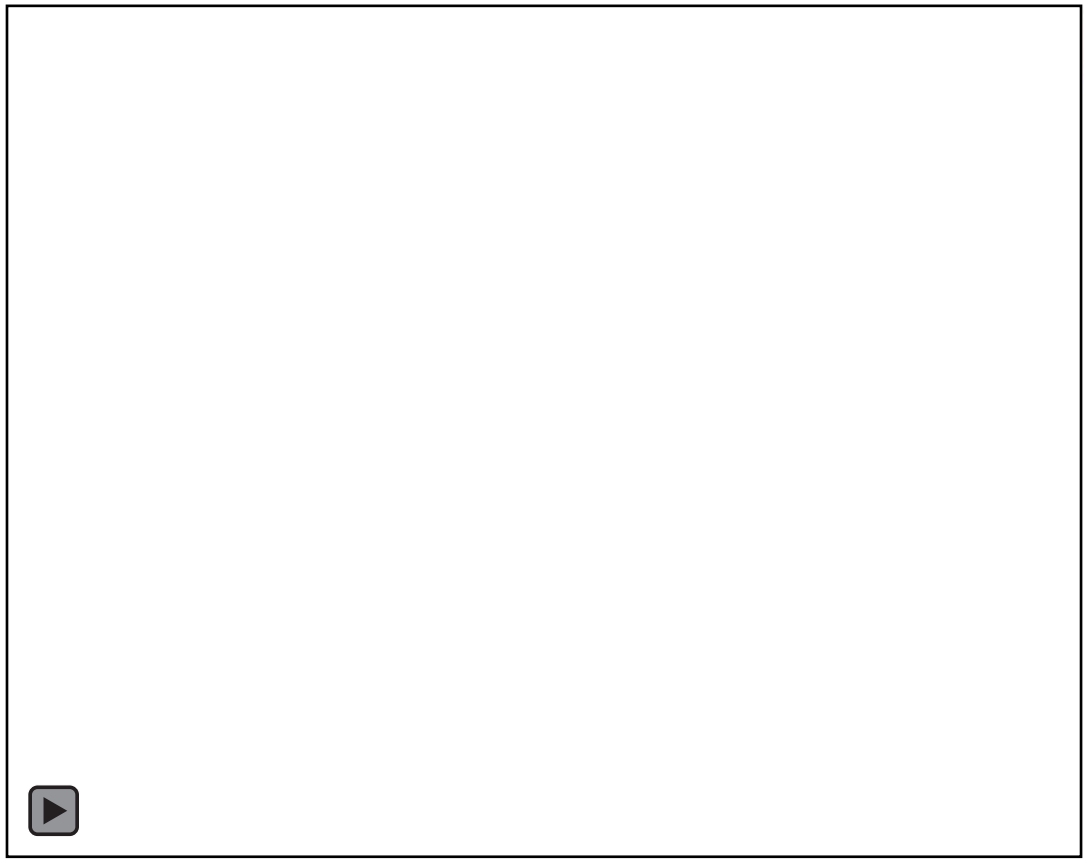
Demo – Search



Demo – Edit Settings

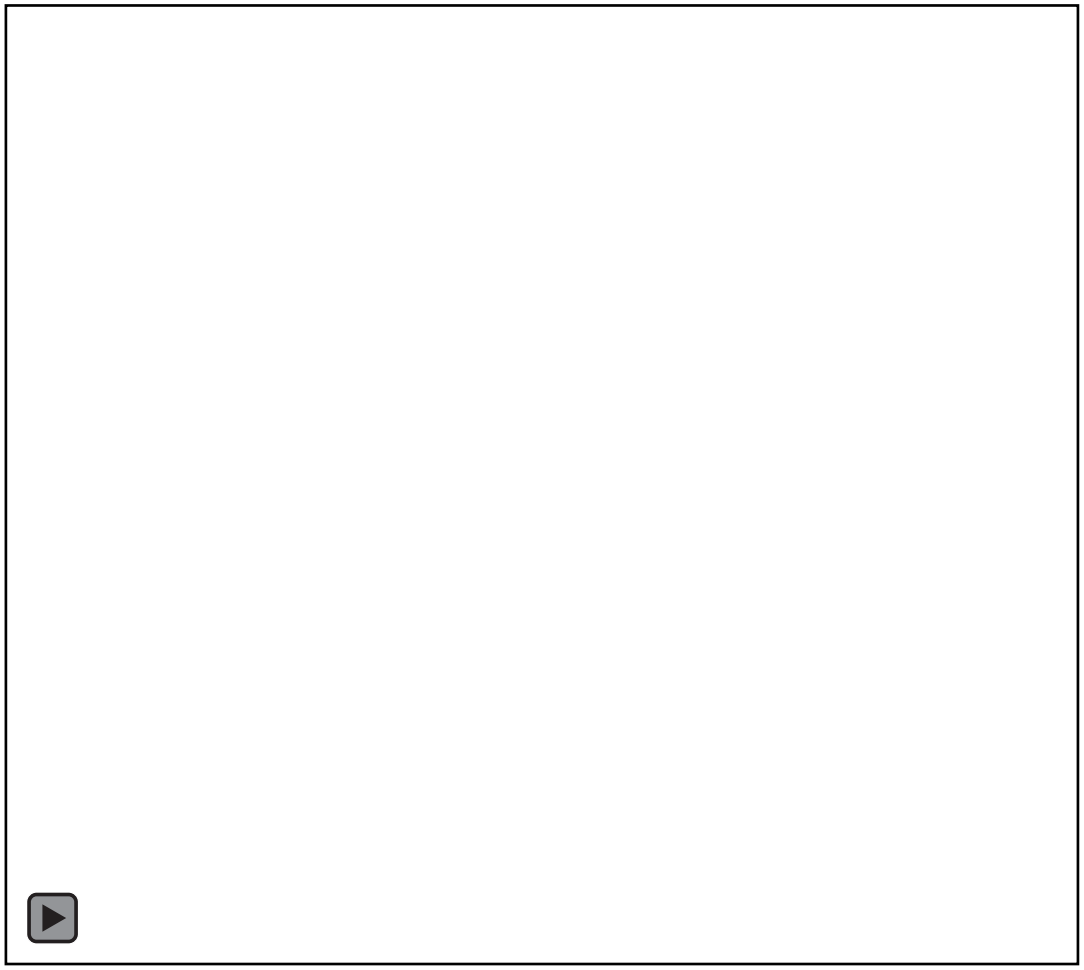


Demo – Public Calendar Visualization



One of the two events, the second, is private, so no information is shown, other than the fact that User1 is busy in that time frame

Demo – Suggested Re-Schedule



Project Report FP and COCOMO II

Project Report – FP

- Internal Logic Files :

ILF	*Complexity*	*FP*
User	Average	10
Event	High	15
Geographic Structure	Low	7
Total:		32

- External Logic Files :

ELF	*Complexity*	*FP*
Weather Forecasts	High	10
Total:		10

Project Report – FP

- External Inputs :

EI	*Complexity*	*FP*
Login / Logout	Low	2 x 3
Create/Delete / Update	High	3 x 6
Participate / Cancel Participation	Low	2 x 3
Invite User	Low	3
Decline Invitation	Low	3
Search Event / User	Low	2 x 3
Insert/Update User	Low	2 x 3
Total:		48

Project Report – FP

- External Inquiries:

EQ	*Complexity*	*FP*
User Profile	Low	3
Events – Created and Participating To	Low	2 x 3
Other User Calendar	High	6
Invitations	Low	3
Notifications	Low	3
Event Details	Low	3
Total:		24

- External Outputs:

EO	*Complexity*	*FP*
Suggested Schedule Changes	High	7
Total:		7

Project Report – FP

In summary:

Function Type	*Value*
Internal Logic Files	32
External Logic Files	10
External Inputs	48
External Inquiries	24
External Outputs	7
Total:	121

With this estimation we get a LOC equal to:

121 * 46 = 5566

Our project is made of 7116 lines of code (without considering empty lines, lines with only parenthesis or comments and css files).

COCOMO II – Scale Drivers

Scale Driver	Factor	*Value*
Precedentedness	Low	4.96
Development Flexibility	High	2.03
Risk Resolution	Very High	1.41
Team Cohesion	Very High	2.19
Process Maturity	High	3.12
Total:		13.71

These parameters were evaluated one per time, according to their definition and not by taken the standard values.

COCOMO II – Cost Drivers

Scale Driver	Factor	*Value*
Required Software Reliability	Low	0.92
Data Base Size	Nominal	1.00
Product Complexity	High	1.17
Required Reusability	High	1.07
Documentation match to life-cycle needs	Nominal	1.00
Execution Time Constraint	Very Low	n/a
Main Storage Constraint	Very Low	n/a
Platform Volatility	Low	0.87
Analyst Capability	High	0.85
Programmer Capability	High	0.88
Application Experience	Low	1.10
Platform Experience	Nominal	1.00
Language and Tool Experience	Nominal	1.00
Personnel continuity:	Very Low	1.29
Usage of Software Tools	Nominal	1.00
Multisite development	Extra High	0.80
Required development schedule	High	1.00
Product:		0.85

COCOMO II – Effort Calculation

Using this formula:

$$Effort = A \times EAF \times KSLOC^{E}$$

We obtained:

$$Effort = 2.94 \times 0.85 \times 5.566^{1.0471} = 15.0808\ PM$$

COCOMO II – Duration Calculation

Using this formula:

$$Duration = 3.67 \times Effort^{F}$$

Where:

$$F = 0.28 \times 0.2 \times (E - B)$$

We obtained:

$$Duration = 3.67 \times 15.0808^{0.3074} = 8.4 \sim 8$$

COCOMO II – Human Resources

$$P = \frac{Effort}{Duration} = 1.79 \sim 2$$

This value is correct, but the estimated duration is far from the real value. In our Project Report document we performed a mathematical analysis showing that an error in our estimated values for FP, Cost Drivers and Scale Drivers can't be the cause of this discrepancy, or, at least, not the only one.

The total amount of hours spent on the project, not including acceptance testing, project reporting and presentation is 455 hours. This value doesn't explain the estimated duration either (considering the COCOMO II conversion factor of 152 hours per month).

COCOMO II – Wrong Estimation Analysis

The most likely explanation for this issue is probably the fact that we enforced the requirements of the project by always chosing the most flexible, complete and scalable option.

Moreover we also implemented different functionalities that weren't included in the initial description, but that we considered really important for our service.

Another possible explanation could be the fact, that perhaps this project would have been more suited for a three people team, according to the COCOMO II analysis. That, at least, would have explained the obtained results.

Acceptance Testing

Acceptance Testing

In this section we will describe the acceptance testing performed over the implementation of MeteoCal by Salvatore Agnese and Gabriele Giganti. Their work can be found at:

https://code.google.com/p/meteocal1415agnesegiganti/

The details of this process are reported in the Acceptance Testing document that can be found in our repository.

Acceptance Testing

To perform this phase phase of the project as efficiently and accurately as possible, we devided our work in a Verification and Validation phase. The first steps are reported here:

- Certify that functionalities promised in RASD satisfy the requirements described in the problem description
- Check that the implementation of the system adheres to the guidelines of the RASD

Acceptance Testing

In this first part of the project we focused on the question:

"*Does the software provide the functionalities required in the initial problem description?*"

The answer, obtained by the analysis of RASD and the User Manual alongside with the actual implementation, was positive.

Acceptance Testing – System Test Cases

After the first phase we continued our analysis making sure that every functionality provided is actually working as expect, even in presence of erroneous input.

Initially we decided to follow the Test Cases document provided by the authors, classifying each test with an outcome, a list of issues and notes. The latter should be taken more as suggestions, to improve the software.

Acceptance Testing – System Test Cases

The possible outcomes of our analysis are:

- • SUCCESS
  - If the main goals of the test have been accomplished and no highly relevant objection can be made to the management of any particular input condition.
- • PARTIAL SUCCESS
  - If the main goals of the test have been accomplished but there is one or more highly relevant objection can be made to the management of a particular input condition stated or not stated in the test case documentation, but related to it.
- • FAIL
  - If the main goals of the test have not been accomplished.
- • UNKNOWN
  - If the occurrence of some particular conditions made the test unavailable

System Test Cases

The major part of the System Test Cases received a successful outcome. Even in those cases some improvements may be made following the test notes.

In the next slides we are going to show some examples of this process.

System Test Cases - Login

Outcome	PARTIAL SUCCESS	
Issues	*Description*	*Value*
	The issue can be replicated by following these steps: • Login using valid data in the login page at http://localhost:8080/meteo/ • Login succeeds, the home page is shown. (URL doesn't change, the request has probably just been dispatched) • Refresh the page or go to http://localhost:8080/meteo/ • The login page previously mentioned is shown even though since no log out was performed, the current session is still active • Try to login using the previous data or even an other account. • The login fails. This is probably due to the fact that the session is still open. The only way for the end user to get back into the system to his pages is actually manually changing the URL to any of its personal pages, which is probably not intended.	2
	User authentication isn't used for managing page access. More information can be found in section 3.2.1.	3
Notes		

System Test Cases – Update Event

An other extract. As the previous one we will explain later the mentioned issue. In this case, updating an event causes the participants to lose their participation. Since this is probably intended, we classified the potential issue as a note.

Outcome	SUCCESS	
Issues	*Description*	*Value*
Notes	• [POTENTIAL IMPROVEMENT] Participants should be notified about the change by the system instead of removing the participation in their place. In fact it would be probably better if the user could remove his participation by himself after being notified about a change. Moreover the invitation doesn't show in any way that the previous participation has been removed due to a change. Anyway the presented solution is acceptable.	

Additional System Test Cases – Page Access

To complete the testing process, we decided to integrate the proposed test cases with an additional test.

Objective	User pages access should be restricted to logged users
Page	Any of the pages that should be accessed only by logged user (e.g. Home Page)
Input	A user tries to access the page by manually changing the URL in his browser
Expected Output	Pages should be correctly shown if the user is logged in the system. If no session is access, the pages shouldn't be shown without any system errors. Showing the login page instead is an acceptable solution.
Obtained Output	Pages are shown if the user is logged. If the user isn't logged unexpected and potentially behaviours have been discovered.

Additional System Test Cases – Page Access

Outcome	FAIL	
Issues	*Description*	*Value*
	Page access is not ruled by security constraints. Different pages that should require login are accessible (by changing manually the URL) when no session is active. This causes NullPointer exception when the page references properties relative to the current session (that does not exist). This happens for example at: http://localhost:8080/meteo/faces/user/homepage.xhtml Other pages that show non intended behaviours are: http://localhost:8080/meteo/faces/user/profilePage.xhtml http://localhost:8080/meteo/faces/user/createEvent.xhtml Being able to access the last one is actually really dangerous since it is possible, using this method, to create events without an active session! A consequence of this fact is that the newly created event has emailCreator equal to null. Since sessions expire due to limited duration, this may happen unexpectedly when a user is logged for some time. Moreover the occurrence of this situations can be dangerous to the execution of the application itself that can crash, making a fresh re-start necessary.	3
Notes		

Additional Notes – Participation Management

Participation to the events are considered though as no more than a positive answer to an invitation. This causes some behaviors that may be considered rather unpleasant for the end user. An example of this is shown here:

- The creator invites user X to event E.
- X accepts the invitation and event E is added to X's calendar.
- Creator of event E decides to remove the invitation sent to X. Consequence: the participation of X to E is completely removed, and E doesn't appear anymore on X calendar, while X doesn't receive any notice of this.

At the current state of the system, there's no way for a user to cancel his participation to an event.

Additional Notes – Page Navigation URLs

There is no way to get a link to the page of an event or of a user on the system, since the pages dedicated to showing these contents don't use the URLs to identify the resource requested by the system. While this doesn't represent an obstacle to the system in providing its services to the users there are also some negative consequences.

For example there's no way for a person to show an other a particular event since there is no way to link it, nor a search functionality.

Another important consequence is correlated to the security protection used in managing event privacy level and will be analyzed in the next section.

Additional Notes – Event Page Access

At the current state of the system the only way for a user to access the page of an event he's not invited to is clicking on that event on the calendar of a participant that has set his calendar as public. Even then, only invited users can actually participate to the event since there is no “Participate” button.

The only protection that a private event page has is actually the absence of a link to that page visible to user non invited to the event. This by itself, could be considered a rather unsafe approach to this resource management problem.

Additional Notes – Event Page Access

This approach in fact has a flaw in the current state of the system, even if its occurrence is rather hidden. The steps to observe this flaw are the following:

- Public event E is created
- Since the event is public, it will be visible on the calendar page of the creator or a participant to the event if the owner set his calendar as public. User U, not invited to the event, uses the search functionality to get on such page, where there is a link to E, since E is public and stops here
- Creator of E modifies E writing secret information in the description of the event and setting the event as private
- User U which still has available a link to E, since he got to a calendar containing it when E was public. User U can then use this link and access the page of E and read all sensible information contained in its description

Code Quality

The provided code has no documentation for the most part, since there is no JavaDoc and comments are almost totally absent.

On the bright side the structure of the code is very modular, and the operations contained in each module are usually not complex. So, regardless of the lack of documentation, the code should be maintainable with very little effort.

Conclusions

The set of operations performed in the system test cases fully fully covers the requirements extracted from the initial description.

As for the implementation, the majority of operations tested showed no highly relevant issues. There are anyway some that needs to be fixed since they can represent a danger for the execution of the system or problematic for the end user. The most important issue is probably the one regarding the fact that page access should be restricted to logged users, since it can lead to application failures.

To recap, the product answers fully the requests stated in the problem descriptions, but has some flaws that, for the most part, should be easily fixable. Moreover, the product offers functionalities that ease its usage. An example is the contact list, that allows suggestions for the users to invite to an event.

Acceptance Testing On Our Implementation

Acceptance Testing On Our Implementation

In this section we will answer some of the issues tracked in the Acceptance Testing document by Francesco Lattari and Alessandro Rimoldi, in which they analysed our production. Their repository can be found at:

https://code.google.com/p/meteocal-lattari-rimoldi/

Their evaluation of the System Test Cases defined by us confirms our results for the major part. In this section we will only go over the issues highlighted by them.

Acceptance Testing On Our Implementation - Login

About TC2 – Login:

"However if a user, once logged, manually inserts the address of the Index page, he can perform again the login, even if he's already logged. This must be avoided."

This isn't really an issue, and it doesn't go against our specification. It is possible anyways to change this really easily by performing a check on page request, to see whether the user is already logged. If this is the case the user can be redirected to Home Calendar. As mentioned, this isn't an issue, but can anyways be fixed in no more than 5 lines of code.

About TC15 – Join/Decline Invitation:

"If the user accepts the invitation, the invitation still be visible on his invitations list and the "accept" and "reject" buttons are still available. So a user can accept the invitation continuously even if the system won't do anything about that. The worst thing is that a user can first accept and then reject the invitation."

This isn't an issue. In fact, our specifications and our test cases clearly underline this behavior, coherent with the initial problem description. We assume that the testing group didn't read this, and made wrong assumptions on the expected outcome. Anyway to obtain the result requested by the testing team, we would have to modify just one line of code.

About TC18 – Visualize A Generic User Calendar:

"If the calendar privacy is set to "visible" and an event is set to private, the user can view that the request user is busy, but clicking on the event will redirect to the error page."

This is a minor issue. The most expectable behavior on click for the end user is probably seeing nothing to happen, or obtain a dialog that explicitly tells that the event is private. Either way this could be done really easily by changing less than 10 lines of code.