Architecture Design

KSU Student Portal

Version 1.0

Submitted in partial fulfillment of the requirements of the degree of MSE

Javier Ramos Rodríguez
CIS 895 – MSE Project
Kansas State University
Table of Contents

1. **Requirements** ........................................................................................................... 4

   1.1. **Use Case View** .................................................................................................. 4
       1.1.1. Unregistered User ......................................................................................... 5
       1.1.2. Registered User ........................................................................................... 6
           1.1.2.1. Manage Events .................................................................................... 8
           1.1.2.2. Manage Blog ..................................................................................... 9
       1.1.3. Administrator ............................................................................................. 10

2. **Dynamic View** .......................................................................................................... 11

   2.1. **Use Case Specification** .................................................................................... 11
       2.1.1. User Module ............................................................................................... 11
           2.1.1.1. Use Case: Register ............................................................................... 11
               2.1.1.1.1. Sequence Diagram ...................................................................... 14
               2.1.1.1.2. Collaboration Diagram ................................................................. 15
               2.1.1.1.3. Activity Diagram ........................................................................ 16
           2.1.1.2. Use Case: Sign In .................................................................................. 17
               2.1.1.2.1. Scenario 1 .................................................................................... 17
                   2.1.1.2.1.1. Sequence Diagram ................................................................. 18
                   2.1.1.2.1.2. Collaboration Diagram ........................................................... 19
               2.1.1.2.2. Scenario 2 .................................................................................... 20
                   2.1.1.2.2.1. Sequence Diagram ................................................................. 20
                   2.1.1.2.2.2. Collaboration Diagram ........................................................... 20
           2.1.2. Administration Module .............................................................................. 21
               2.1.2.1. Use Case: Get Users .......................................................................... 21
                   2.1.2.1.1. Sequence Diagram .................................................................... 22
                   2.1.2.1.2. Collaboration Diagram ............................................................... 23
               2.1.2.2. Use Case: Remove User .................................................................... 24
                   2.1.2.2.1. Sequence Diagram .................................................................... 25
                   2.1.2.2.2. Collaboration Diagram ............................................................... 26
2.1.3.  Unregistered User Module ________________________________ 27
  2.1.3.1.  Use Case: Browse Users ________________________________ 27
    2.1.3.1.1.  Sequence Diagram ________________________________ 28
    2.1.3.1.2.  Collaboration Diagram _____________________________ 29
  2.1.3.2.  Use Case: View User Profile ____________________________ 30
    2.1.3.2.1.  Sequence Diagram ________________________________ 31
    2.1.3.2.2.  Collaboration Diagram _____________________________ 32
  2.1.3.3.  Use Case: Search Users ________________________________ 33
    2.1.3.3.1.  Sequence Diagram ________________________________ 34
    2.1.3.3.2.  Collaboration Diagram _____________________________ 35
  2.1.3.4.  Use Case: Browse Public Events _________________________ 36
    2.1.3.4.1.  Sequence Diagram ________________________________ 37
    2.1.3.4.2.  Collaboration Diagram _____________________________ 38
  2.1.3.5.  Use Case: View Public Event____________________________ 39
    2.1.3.5.1.  Sequence Diagram ________________________________ 40
    2.1.3.5.2.  Collaboration Diagram _____________________________ 41
  2.1.3.6.  Use Case: Search Public Items __________________________ 42
  2.1.4.  Registered User Module ____________________________ 43
    2.1.4.1.  Use Case: Edit Profile______________________________ 43
      2.1.4.1.1.  Sequence Diagram ________________________________ 44
      2.1.4.1.2.  Collaboration Diagram _____________________________ 45
    2.1.4.2.  Use Case: Create Event ______________________________ 46
      2.1.4.2.1.  Sequence Diagram ________________________________ 47
      2.1.4.2.2.  Collaboration Diagram _____________________________ 48
    2.1.4.3.  Use Case: Browse Events ______________________________ 49
      2.1.4.3.1.  Sequence Diagram ________________________________ 50
      2.1.4.3.2.  Collaboration Diagram _____________________________ 51
    2.1.4.4.  Use Case: View Event______________________________ 52
    2.1.4.5.  Use Case: Delete Event ______________________________ 53
      2.1.4.5.1.  Sequence Diagram ________________________________ 54
      2.1.4.5.2.  Collaboration Diagram _____________________________ 55
3.  Static View ______________________________________________ 56
  3.1.  Architecture Diagram ________________________________ 56
  3.2.  Class Diagram __________________________________________ 57
    3.2.1.  User Module __________________________________________ 57
    3.2.2.  Unregistered User Module ______________________________ 58
    3.2.3.  Registered User Module ________________________________ 59
    3.2.4.  Model Classes _________________________________________ 60
1. Requirements

1.1. Use Case View

- User
  - Register
  - Sign In

- Unregistered User
  - Unregistered User Package

- Registered Users
  - Registered User Package
In the figure the relation between the different actors is shown. We have two actions that a user can do if he has not logged in yet. Which are sign in as a guess user or registered user if he/she is already a registered user, or register in the application. Once signed in, depending if the user is a registered or unregistered will have different functionalities that are described next.

1.1.1. Unregistered User

As we can see in the figure the Unregistered User has very limited functionalities. He can only see the public news/events or articles and browse the different users to see their profile but only the public information of the profile. He can also perform searches on public information.
In the figure we show the main functionalities of the Registered User. As we mention before, the registered user must sign in to access to this functionalities, additionally he can sign out at any moment. He/She will be able to manage (create and remove) the different information such articles, blog entries, events or Web links. He will be able to send/receive messages to/from other users or access the forum. Finally he can edit his/her profile to make the appropriate changes also to his/her blog.
The “Manage XXXXX” Use Cases need to be divided into more specific functionalities. Manage means create, delete, browse and view the different item. Since the functionality will be the same for all the “Manage XXXXX” use cases (only the data changes), we are going to specify only one, and the rest will be the same.
As we can see in the figure, a registered user can publish an event at anytime. He/She can also browse the different events. Selecting one event will allow the user to view all the information about it (View Event). The events that were created by the user will have the possibility to be deleted. When an event is created a profile will be associated to it.
The user will have the chance to edit his/her blog and add, remove or modify an entry. He/She can change the profile associated to the blog.
1.1.3. Administrator

There will be a special user that will have ability to remove other users from the system.

Administrator
(from Use Case View)

Get Users

<<extend>>

Remove User
## 2. Dynamic View

### 2.1. Use Case Specification

#### 2.1.1. User Module

##### 2.1.1.1. Use Case: Register

<table>
<thead>
<tr>
<th>Name</th>
<th>Register</th>
</tr>
</thead>
<tbody>
<tr>
<td>Objective</td>
<td>Register a new user in the application, so he/she can access to the main features.</td>
</tr>
<tr>
<td>Priority</td>
<td>High</td>
</tr>
<tr>
<td>Extends</td>
<td></td>
</tr>
<tr>
<td>Includes</td>
<td></td>
</tr>
<tr>
<td>Actors</td>
<td>User</td>
</tr>
<tr>
<td>Pre-conditions</td>
<td></td>
</tr>
<tr>
<td>Post-conditions</td>
<td>User will get registered in the application.</td>
</tr>
<tr>
<td>Main Scenario</td>
<td>1. The user selects “Register” from the user interface.</td>
</tr>
<tr>
<td></td>
<td>2. The system receives the request.</td>
</tr>
<tr>
<td></td>
<td>3. The system shows the user the register form.</td>
</tr>
<tr>
<td></td>
<td>4. User fills the form.</td>
</tr>
<tr>
<td></td>
<td>5. System validates the form.</td>
</tr>
<tr>
<td></td>
<td>6. The system receives the information and saves it.</td>
</tr>
<tr>
<td>Secondary Scenario</td>
<td></td>
</tr>
<tr>
<td>Exceptional Scenario</td>
<td>If there’s an error in the register form the system will tell the user to correct it.</td>
</tr>
<tr>
<td>Function</td>
<td>Register User</td>
</tr>
<tr>
<td>Non functional requirements</td>
<td></td>
</tr>
<tr>
<td>Execution</td>
<td></td>
</tr>
<tr>
<td>Frequency</td>
<td>Medium</td>
</tr>
<tr>
<td>Annotations</td>
<td></td>
</tr>
</tbody>
</table>

The scenario will be explained in detail using collaboration and sequence diagrams that show the dynamic behavior of the use case. At this point we don’t have information of the architecture that will be used, so we create an abstract model that can be implemented in any platform or architecture. We will divide the system are three layers: View, Model and Data.
The goal of this phase is to create a very robust and accurate specification but at the same time maintain an independence of the language or platform used. The next phase will have more details and will be platform dependent, for example if I decide to implement in the J2EE platform, the classes used in this model will have one or more corresponding Java classes. The goal is that the analysis, design and implementation are coherent with each other having each one different level of abstraction.
Three different types (stereotypes) of classes are used in the diagrams depending on the usage of the class.

1. **Entity Classes**: These types of classes are used to represent entities that contain data. Usually they will match to tables in the relational model or entity beans when using the J2EE platform.

2. **Control Classes**: These types of classes are used to represent classes that perform some kind of logic: business logic or controllers. In the view layer these classes will represent the controllers in the MVC pattern; in the model these classes will represent the business logic. For example, in the J2EE platform these classes probably will be session beans.

3. **Boundary Classes**: These classes in the view are used to represent the classes that the user interacts with. These classes will contain all the forms and all the other UI components. So, these classes are the boundary between users and the application. For example, in a J2EE web application these classes will be JSP pages. In the model, this stereotype is used to represent the boundary between the view and the model, so it will be used in the class that implements the facade design pattern.

To follow better the sequence diagrams, check at the same time the static class model on the bottom.
The scenario starts in the main window. The user chooses “Register” in the menu and the request in handled by the controller who shows the form in the RegisterUI. The user fills out the form and submits the information. The controller validates the fields and in case of an error asks the user to fill the form again. If the information is correct the controller requests the model to perform the operation. The request in handled by the model facade which asks as a boundary between the view and the business logic. The model facade asks the control class to perform the operation, this class creates the entity and performs the persistence, and then it returns the results.
2.1.1.2. Collaboration Diagram

1: register

2: register()

3: showRegForm()

4: fill form
8: correct form

5: submit()

6: validateFields()

7: showRegForm(String error)
15: showResults(String results)

9: registerUser()

10: registerUser()

13: showResults(String results)

12: persist(user)

14: showResults(String results)

11: <<create>>

user: AppUser

user : ModelFacade

user : RegisterUIController

user : UserControl

user : MainUIController

user : MainUI

user : RegisterUI

user : User
2.1.1.1.3. Activity Diagram

- Fill Form
- Validate Form
- Create User
### 2.1.1.2. Use Case: Sign In

<table>
<thead>
<tr>
<th>Name</th>
<th>Sign In</th>
</tr>
</thead>
<tbody>
<tr>
<td>Objective</td>
<td>Register a new user in the application, so he/she can access to the main features.</td>
</tr>
<tr>
<td>Priority</td>
<td>High</td>
</tr>
<tr>
<td>Extends</td>
<td></td>
</tr>
<tr>
<td>Includes</td>
<td></td>
</tr>
<tr>
<td>Actors</td>
<td>User</td>
</tr>
<tr>
<td>Pre-conditions</td>
<td></td>
</tr>
<tr>
<td>Post-conditions</td>
<td>User will Sign in the application</td>
</tr>
</tbody>
</table>

#### Main Scenario

1. The user introduces his/her Login and Password.
2. The system receives the request.
3. The system checks if the information is correct according with the information stored in the system.
4. The system sends an error message if the log in was incorrect or forward the user to the main page.

#### Secondary Scenario

1. The user selects “Guess” to enter as an unregistered user.
2. The system receives the request and forwards him/her to the main page.

#### Exceptional Scenario

If there’s an error logging in the system will tell the user to correct it.

#### Function

Sign In User

#### Non functional requirements

**Execution**

**Frequency**

High

**Annotations**
2.1.1.2.1. Scenario 1

2.1.1.2.1.1. Sequence Diagram

[Sequence Diagram Image]

- User
- MainUI
- MainUIController
- ModelFacade
- UserControl
- AppUser
- WelcomeUI

- fill form
- showLoginForm()
- checkLogin()
- validateInput()
- [NO fields properly filled]
- [fields properly filled]
- checkLogin(String name, String passwd)
- checkLogin(String login, String Passwd)
- showResults(String results)
- checkResults(String results)
- login fail
- [results == false]
- [results == true]
- show()
2.1.1.2.1.2 Collaboration Diagram

1: showLoginForm()
2: fill form
3: checkLogin()
4: validateInput()
5: showLoginForm(String error)
6: correct form
7: checkLogin(String name, String passwd)
8: checkLogin(String login, String Passwd)
9: <<use>>
10: showResults(String results)
11: showResults(String results)
12: checkResults(String results)
13: login fail
14: show()
2.1.1.2.2 Scenario 2

2.1.1.2.2.2 Sequence Diagram

2.1.1.2.2.3 Collaboration Diagram
2.1.2. Administration Module

2.1.2.1. Use Case: Get Users

<table>
<thead>
<tr>
<th>Name</th>
<th>Get Users</th>
</tr>
</thead>
<tbody>
<tr>
<td>Objective</td>
<td>Get the different users registered in the application</td>
</tr>
<tr>
<td>Priority</td>
<td>Medium</td>
</tr>
<tr>
<td>Extends</td>
<td></td>
</tr>
<tr>
<td>Includes</td>
<td></td>
</tr>
<tr>
<td>Actors</td>
<td>Administrator</td>
</tr>
<tr>
<td>Pre-conditions</td>
<td>Sign into the system</td>
</tr>
<tr>
<td>Post-conditions</td>
<td>User will receive a list of different users</td>
</tr>
<tr>
<td>Main Scenario</td>
<td>1. The user selects “Get users” from the main menu.</td>
</tr>
<tr>
<td></td>
<td>2. The system receives the request and retrieves the list with all the users,</td>
</tr>
<tr>
<td></td>
<td>3. The system shows the user the list of registered users.</td>
</tr>
<tr>
<td>Secondary</td>
<td></td>
</tr>
<tr>
<td>Scenario</td>
<td></td>
</tr>
<tr>
<td>Exceptional</td>
<td></td>
</tr>
<tr>
<td>Scenario</td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Get registered users.</td>
</tr>
</tbody>
</table>

**Non functional requirements**

| Execution     |                                                                           |
|---------------|                                                                           |
| Frequency     | High                                                                      |
| Annotations   |                                                                           |
The *UserControl* class creates an object List that has generic list of abject then it queries the database and gets the list of users. In a loop, the *UserControl* class creates an *AppUser* object and stores it in the generic list that is returned to view for visualization.
2.1.2.1.2. Collaboration Diagram

[Diagram showing interaction between Administrator, WelcomeUIController, ModelFacade, UserListUI, UserControl, and AppUser.]
## 2.1.2.2. **Use Case: Remove User**

<table>
<thead>
<tr>
<th>Name</th>
<th>Remove User</th>
</tr>
</thead>
<tbody>
<tr>
<td>Objective</td>
<td>Delete one user from the database</td>
</tr>
<tr>
<td>Priority</td>
<td>Medium</td>
</tr>
<tr>
<td>Extends</td>
<td></td>
</tr>
<tr>
<td>Includes</td>
<td></td>
</tr>
<tr>
<td>Actors</td>
<td>Administrator</td>
</tr>
<tr>
<td>Pre-conditions</td>
<td>Sign into the system</td>
</tr>
<tr>
<td>Post-conditions</td>
<td>An User will be removed from the database</td>
</tr>
</tbody>
</table>

### Main Scenario

1. The Administrator selects one particular user from the user list.
2. The Administrator confirms the operation.
3. The system receives the request and looks for the specific user in the database.
4. The system removes the user.
5. The system shows the results of the operation

### Secondary Scenario

### Exceptional Scenario

If there’s an error while deleting the user, the administrator will be notified

### Function

Delete event

### Non functional requirements

### Execution

### Frequency

Medium

### Annotations

24
2.1.2.2.1. Sequence Diagram

- Administrator
- UserListUI
- UserListUIController
- ModelFacade
- UserControl
- AppUser

delete user

confirm

showConfirmation()

deleteUser()

deleteUser(login)

deleteUser(login)

showResults()

showResults()

<<remove>>
2.1.2.2.2. Collaboration Diagram

1: delete user
2: showConfirmation()
3: confirm
4: deleteUser()
5: deleteUser(login)
6: deleteUser(login)
7: <<remove>>
8: showResults()
9: showResults()
10: showResults deleteUser()
# 2.1.3. Unregistered User Module

## 2.1.3.1. Use Case: Browse Users

<table>
<thead>
<tr>
<th>Name</th>
<th>Browse Users</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Objective</strong></td>
<td>Browse the different user registered in the application</td>
</tr>
<tr>
<td><strong>Priority</strong></td>
<td>Medium</td>
</tr>
<tr>
<td><strong>Extends</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Includes</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Actors</strong></td>
<td>Unregistered User</td>
</tr>
<tr>
<td><strong>Pre-conditions</strong></td>
<td>Sign into the system</td>
</tr>
<tr>
<td><strong>Post-conditions</strong></td>
<td>User will receive a list of different users</td>
</tr>
<tr>
<td><strong>Main Scenario</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1. The user selects “browse users” from the main menu.</td>
</tr>
<tr>
<td></td>
<td>2. The system receives the request and retrieves the list with all the users.</td>
</tr>
<tr>
<td></td>
<td>3. The system shows the user the list of registered users.</td>
</tr>
<tr>
<td><strong>Secondary Scenario</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Exceptional Scenario</strong></td>
<td>There will be a possibility to browse user by name, country, major…</td>
</tr>
<tr>
<td><strong>Function</strong></td>
<td>Browse registered users.</td>
</tr>
</tbody>
</table>

### Non functional requirements

- **Execution**
  - **Frequency**: High
  - **Annotations**: 27
2.1.3.1.1. Sequence Diagram

The whole list of users will be shown but the Unregistered User will only be allow to click in the ones who have public visibility to see their profile.
2.1.3.1.2. Collaboration Diagram

1: Browse Users
2: getUsers()
10: showUserList(users)

: Unregistered User

: WelcomeUIController

: WelcomeUIController

: UserListUI

3: getUserList()
9: returnResults()

: UserControl

: ModelFacade

6: <<create>>
4: getUserList()
5: <<create>>
7: add(user)
8: returnResults(users)

user : AppUser

users : List
### 2.1.3.2. Use Case: View User Profile

<table>
<thead>
<tr>
<th>Name</th>
<th>View User Profile</th>
</tr>
</thead>
<tbody>
<tr>
<td>Objective</td>
<td>Browse the different user registered in the application</td>
</tr>
<tr>
<td>Priority</td>
<td>Medium</td>
</tr>
<tr>
<td>Extends</td>
<td>Browse users</td>
</tr>
<tr>
<td>Includes</td>
<td></td>
</tr>
<tr>
<td>Actors</td>
<td>Unregistered User</td>
</tr>
<tr>
<td>Pre-conditions</td>
<td>Sign into the system, browse users</td>
</tr>
<tr>
<td>Post-conditions</td>
<td>User views a registered user profile</td>
</tr>
<tr>
<td>Main Scenario</td>
<td>1. The user selects a user from the user list.</td>
</tr>
<tr>
<td></td>
<td>2. The system receives the request and finds the appropriate user profile.</td>
</tr>
<tr>
<td></td>
<td>3. The system also gets the blog entries.</td>
</tr>
<tr>
<td></td>
<td>4. The system returns the selected user profile.</td>
</tr>
<tr>
<td>Secondary Scenario</td>
<td></td>
</tr>
<tr>
<td>Exceptional Scenario</td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>View a specific user profile</td>
</tr>
</tbody>
</table>

**Non functional requirements**

<table>
<thead>
<tr>
<th>Execution</th>
<th>Frequency</th>
<th>Annotations</th>
</tr>
</thead>
</table>
2.1.3.2.2. Collaboration Diagram

1: view user profile
2: getUserProfile()
12: showProfile(user, blogs)

Unregistered User

UserListUI

UserListUIController

UserProfileUI

ModelFacade

UserControl

AppUser

blogs: List

blog: Blog

<<create>>

getUserProfile(login)

showResults(user, blogs)

getUserProfile(login)

showProfile(user, blogs)

setUserProfile(login)

showResults(user, blogs)

setUserProfile(login)

add(blog)

showProfile(user, blogs)
# Use Case: Search Users

<table>
<thead>
<tr>
<th>Name</th>
<th>Search Users</th>
</tr>
</thead>
<tbody>
<tr>
<td>Objective</td>
<td>Search for users registered in the application</td>
</tr>
<tr>
<td>Priority</td>
<td>Medium</td>
</tr>
<tr>
<td>Extends</td>
<td></td>
</tr>
<tr>
<td>Includes</td>
<td></td>
</tr>
<tr>
<td>Actors</td>
<td>Unregistered User</td>
</tr>
<tr>
<td>Pre-conditions</td>
<td>Sign into the system</td>
</tr>
<tr>
<td>Post-conditions</td>
<td>User will receive a list with the results of the search</td>
</tr>
</tbody>
</table>
| **Main Scenario** | 1. The user selects “search users” from the main menu.  
                        2. The systems show a form where the user introduces the search criteria.  
                        3. The user sends the search request.  
                        4. The system receives the request and retrieves the list of users corresponding to the search criteria.  
                        5. The system shows the user the results. |

| **Secondary Scenario** |                                                                 |
| **Exceptional Scenario** |                                                                 |
| Function                | Search registered users.                                                     |

**Non functional requirements**

<table>
<thead>
<tr>
<th>Execution</th>
<th>Frequency</th>
<th>Annotations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>High</td>
<td></td>
</tr>
</tbody>
</table>
2.1.3.3.1. Sequence Diagram

Unregistered User

WelcomeUI

WelcomeUIController

SearchUsersUI

ModelFacade

UserController

users : List

user : AppUser

search users

showSearchForm( )

fill form

searchUsers( )

searchUsers( criteria )

DB query

[repeat]

[while ResultSet.isEmpty == true]

searchUsers( criteria )

<<create>>

add(user )

showSearchResults( )

showResults( users )

showSearchResults( )

showResults( users )
## 2.1.3.4. Use Case: Browse Public Events

<table>
<thead>
<tr>
<th>Name</th>
<th>Browse Public Events</th>
</tr>
</thead>
<tbody>
<tr>
<td>Objective</td>
<td>Browse the different public events that are available in the application</td>
</tr>
<tr>
<td>Priority</td>
<td>Medium</td>
</tr>
<tr>
<td>Extends</td>
<td></td>
</tr>
<tr>
<td>Includes</td>
<td></td>
</tr>
<tr>
<td>Actors</td>
<td>Unregistered User</td>
</tr>
<tr>
<td>Pre-conditions</td>
<td>Sign into the system</td>
</tr>
<tr>
<td>Post-conditions</td>
<td>User will receive the list of events</td>
</tr>
<tr>
<td>Main Scenario</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1. The user selects “browse events” from the main menu.</td>
</tr>
<tr>
<td></td>
<td>2. The system receives the request and retrieves the list with all the items requested, in this case, only the public ones.</td>
</tr>
<tr>
<td></td>
<td>3. The system shows the user the list of events.</td>
</tr>
<tr>
<td>Secondary Scenario</td>
<td></td>
</tr>
<tr>
<td>Exceptional Scenario</td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Browse the public events</td>
</tr>
<tr>
<td>Non functional requirements</td>
<td></td>
</tr>
<tr>
<td>Execution</td>
<td></td>
</tr>
<tr>
<td>Frequency</td>
<td>Medium</td>
</tr>
<tr>
<td>Annotations</td>
<td></td>
</tr>
</tbody>
</table>
2.1.3.4.1. Sequence Diagram

User: Unregistered

: WelcomeUI
: WelcomeUIController
: ModelFacade
: EventControl
: events : List
: event : Event
: EventListUI

browse public events
getPublicEvents( )
getPublicEvents( )
[repeat]
[while ResultSet.isEmpty == true]
<<create>>
add( event)
[repeat]
<<create>>
[while ResultSet.isEmpty == true]
showResults( events)
showEventList( events)
2.1.3.4.2. Collaboration Diagram

1: browse public events
2: getPublicEvents()
3: getPublicEvents()
4: getPublicEvents()
5: <<create>>
6: <<create>>
7: add(event)
8: showResults(events)
9: showResults(events)
10: showEventList(events)

: Unregistered User
: WelcomeUI
: WelcomeUIController
: ModelFacade
: EventListUI

events : List
event : Event
## 2.1.3.5. **Use Case: View Public Event**

<table>
<thead>
<tr>
<th>Name</th>
<th>View Public Event</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Objective</strong></td>
<td>View the information of a specific event</td>
</tr>
<tr>
<td><strong>Priority</strong></td>
<td>Medium</td>
</tr>
<tr>
<td><strong>Extends</strong></td>
<td>Browse Public Events</td>
</tr>
<tr>
<td><strong>Includes</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Actors</strong></td>
<td>Unregistered User</td>
</tr>
<tr>
<td><strong>Pre-conditions</strong></td>
<td>Sign into the system, browse public information</td>
</tr>
<tr>
<td><strong>Post-conditions</strong></td>
<td>User will receive the information about one event.</td>
</tr>
</tbody>
</table>

### Main Scenario

1. The user may want to see a particular event by clicking in the header.
2. The system receives the request and finds the particular item.
3. The system returns the item information to the user.

### Secondary Scenario

### Exceptional Scenario

### Function

- View a public event

### Non functional requirements

<table>
<thead>
<tr>
<th>Execution</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Frequency</strong></td>
<td>Medium</td>
</tr>
<tr>
<td><strong>Annotations</strong></td>
<td>39</td>
</tr>
</tbody>
</table>
2.1.3.5.1. Sequence Diagram
2.1.3.5.2. Collaboration Diagram

1: view public event
2: viewPublicEvent()
3: viewPublicEvent( e_id)
4: viewPublicEvent( e_id)
5: <<create>>
6: showResults( event )
7: showEvent( event )

: Unregistered User
: EventListUI
: EventListUIController
: EventUI
: EventControl
: ModelFacade
:

event : Event
**2.1.3.6. Use Case: Search Public Items**

<table>
<thead>
<tr>
<th><strong>Name</strong></th>
<th><strong>Search Public Items</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Objective</strong></td>
<td>Search to look for public events that are available in the application</td>
</tr>
<tr>
<td><strong>Priority</strong></td>
<td>Medium</td>
</tr>
<tr>
<td><strong>Extends</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Includes</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Actors</strong></td>
<td>Unregistered User</td>
</tr>
<tr>
<td><strong>Pre-conditions</strong></td>
<td>Sign into the system</td>
</tr>
<tr>
<td><strong>Post-conditions</strong></td>
<td>User will receive the list of events</td>
</tr>
<tr>
<td><strong>Main Scenario</strong></td>
<td>1. The user selects “search” from the main menu.</td>
</tr>
<tr>
<td></td>
<td>2. The Systems shows the search form.</td>
</tr>
<tr>
<td></td>
<td>3. The user fills the forms and introduces the search criteria.</td>
</tr>
<tr>
<td></td>
<td>4. The system receives the request and retrieves the list with the items that much the search criteria from only the public ones.</td>
</tr>
<tr>
<td></td>
<td>5. The system shows the user the list of events.</td>
</tr>
<tr>
<td><strong>Secondary Scenario</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Exceptional Scenario</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Function</strong></td>
<td>Browse the public events</td>
</tr>
<tr>
<td><strong>Non functional requirements</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Execution</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Frequency</strong></td>
<td>Medium</td>
</tr>
<tr>
<td><strong>Annotations</strong></td>
<td></td>
</tr>
</tbody>
</table>
### 2.1.4. Registered User Module

#### 2.1.4.1. Use Case: Edit Profile

<table>
<thead>
<tr>
<th>Name</th>
<th>Edit Profile</th>
</tr>
</thead>
<tbody>
<tr>
<td>Objective</td>
<td>Allow user to modify his/her profile</td>
</tr>
<tr>
<td>Priority</td>
<td>Medium</td>
</tr>
<tr>
<td>Extends</td>
<td></td>
</tr>
<tr>
<td>Includes</td>
<td></td>
</tr>
<tr>
<td>Actors</td>
<td>Registered User</td>
</tr>
<tr>
<td>Pre-conditions</td>
<td>Sign In</td>
</tr>
<tr>
<td>Post-conditions</td>
<td>Information will be added or changed in the user’s profile</td>
</tr>
<tr>
<td><strong>Main Scenario</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1. The user selects “Edit profile” from the UI.</td>
</tr>
<tr>
<td></td>
<td>2. The system receives the request.</td>
</tr>
<tr>
<td></td>
<td>3. The system shows the profile modification form.</td>
</tr>
<tr>
<td></td>
<td>4. The user performs the appropriate changes.</td>
</tr>
<tr>
<td></td>
<td>5. The system receives the form and makes the changes in the database.</td>
</tr>
<tr>
<td><strong>Secondary Scenario</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Exceptional Scenario</strong></td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Edit user’s profile</td>
</tr>
<tr>
<td>Non functional requirements</td>
<td></td>
</tr>
<tr>
<td>Execution</td>
<td></td>
</tr>
<tr>
<td>Frequency</td>
<td>Low</td>
</tr>
<tr>
<td>Annotations</td>
<td></td>
</tr>
</tbody>
</table>
2.1.4.1.1. Sequence Diagram

Registered User
WelcomeUI
WelcomeUIController
ProfileUI
ProfileUIController
ModelFacade
UserController
AppUser
UserProfile

- Registered User
- WelcomeUI
- WelcomeUIController
- ProfileUI
- ProfileUIController
- ModelFacade
- UserController
- AppUser
- UserProfile

edit profile
editUserProfile()
showProfileForm()
fill form
correct form
checkFields()
showProfileForm()
[NO fields properly filled]
showProfileForm()
[fields properly filled]
updateProfile()
updateProfile()
<<update>>
showResults()
showResults()
showResults()

The parameters will be the form fields

44
2.1.4.1.2. Collaboration Diagram

1: edit profile
2: editUserProfile()
3: showProfileForm()
4: fill form
5: checkFields()
6: showProfileForm()
7: correct form
8: updateProfile()
9: updateProfile()
10: <<update>>
11: <<update>>
12: showResults()
13: showResults()
14: showResults()
### 2.1.4.2. Use Case: Create Event

<table>
<thead>
<tr>
<th>Name</th>
<th>Create Event</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Objective</strong></td>
<td>Provide a functionality that allows a registered user to create an event and publish it.</td>
</tr>
<tr>
<td><strong>Priority</strong></td>
<td>Medium</td>
</tr>
<tr>
<td><strong>Actors</strong></td>
<td>Registered User</td>
</tr>
<tr>
<td><strong>Pre-conditions</strong></td>
<td>Sign In</td>
</tr>
<tr>
<td><strong>Post-conditions</strong></td>
<td>An new event will be published in the system</td>
</tr>
</tbody>
</table>
| **Main Scenario** | 1. The user selects “Create Event” from the UI.  
2. The system receives the request.  
3. The system shows the corresponding form to the user.  
4. The user fills up the form and click on “Submit”.  
5. The system receives the information and stores it in the database. |

### Function
Create Event

### Non functional requirements

<table>
<thead>
<tr>
<th>Execution</th>
<th>Medium</th>
</tr>
</thead>
<tbody>
<tr>
<td>Annotations</td>
<td></td>
</tr>
</tbody>
</table>
2.1.4.2.1. Sequence Diagram

Registered User
: WelcomeUI
: WelcomeUIController
: CreateEventUI
: CreateEventUIController
: ModelFacade
: EventControl
: Event
: Item
: ItemProfile

createEvent
createEvent()
showCreateEventForm()
validateFields()
showCreateEventForm()
[NO fields properly filled]
correct form
showCreateEventForm()
[fields properly filled]
correct form
createEvent()
createEvent()
<<create>>
persist( event)
<<create>>
add( item )
showResults()
showResults()
showResults()
showResults()
2.1.4.2.2. Collaboration Diagram

1: create Event
   : Registered User

2: createEvent()
   : WelcomeUI

3: showCreateEventForm()
   : WelcomeUIController

4: validateFields()
   : CreateEventUI

5: showCreateEventForm()
6: correct form
   : CreateEventUIController

7: createEvent()
   : ModelFacade

8: createEvent()
9: <<create>>
   : EventControl

10: <<create>>
11: add(item)
   : EventControl

12: persist(event)
13: showResults()
14: showResults()
15: showResults()

item : ItemProfile
## 2.1.4.3. Use Case: Browse Events

<table>
<thead>
<tr>
<th>Name</th>
<th>Browse Events</th>
</tr>
</thead>
<tbody>
<tr>
<td>Objective</td>
<td>Browse the different events</td>
</tr>
<tr>
<td>Priority</td>
<td>Medium</td>
</tr>
<tr>
<td>Extends</td>
<td></td>
</tr>
<tr>
<td>Includes</td>
<td></td>
</tr>
<tr>
<td>Actors</td>
<td>Registered User</td>
</tr>
<tr>
<td>Pre-conditions</td>
<td>Sign into the system</td>
</tr>
<tr>
<td>Post-conditions</td>
<td>User will receive a list of the different events that he/she is interested.</td>
</tr>
</tbody>
</table>

### Main Scenario

1. The user selects “browse events” from the main menu.
2. The system receives the request and looks the user’s profile information.
3. The filtering module selects the events that the user is interested.
4. The system shows the user the list of the events adding a “delete” link to the events that the user published.

### Secondary Scenario

There will be a possibility to browse user by different fields. Or to browse the user created events.

### Exceptional Scenario

Function Browse events.

**Non functional requirements**

<table>
<thead>
<tr>
<th>Execution</th>
<th>Frequency</th>
<th>High</th>
</tr>
</thead>
</table>
In this case, first we find the user profile object to get the information that the system will use to filter the list of events so the user only gets the ones that is interested. After getting the profile object the model facade calls the EventControl class to get the events according to the profile.
2.1.4.3.2. Collaboration Diagram

Registered User : WelcomeUI : WelcomeUIController : ModelFacade : EventControl

1: browse events
2: browseEvents( )
3: getEvents(login )
12: showResults(events )
13: showEventList(events )
5: <<create>>
7: getEvents( profile)
11: showResults( events )
6: returnResults( profile)
8: <<create>>
10: add( event )
14: showEventList(events )
4: getUserProfile( login)
6: returnResults( profile)
3: getEvents(login )
7: getEvents( profile)

profile : UserProfile

: EventListUI

events : List

: EventControl

event : Event
## 2.1.4.4. Use Case: View Event

<table>
<thead>
<tr>
<th>Name</th>
<th>View Event</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Objective</strong></td>
<td>View the information of a specific event</td>
</tr>
<tr>
<td><strong>Priority</strong></td>
<td>Medium</td>
</tr>
<tr>
<td><strong>Extends</strong></td>
<td>Browse Events</td>
</tr>
<tr>
<td><strong>Includes</strong></td>
<td>Registered User</td>
</tr>
<tr>
<td><strong>Pre-conditions</strong></td>
<td>Sign into the system</td>
</tr>
<tr>
<td><strong>Post-conditions</strong></td>
<td>User will receive the information about one event.</td>
</tr>
<tr>
<td><strong>Main Scenario</strong></td>
<td>1. The user may want to see a particular event by clicking in the header. 2. The system receives the request and finds the particular item. 3. The system returns the item information to the user.</td>
</tr>
<tr>
<td><strong>Secondary Scenario</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Exceptional Scenario</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Function</strong></td>
<td>View a public event</td>
</tr>
<tr>
<td><strong>Non functional requirements</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Execution</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Frequency</strong></td>
<td>Medium</td>
</tr>
<tr>
<td><strong>Annotations</strong></td>
<td></td>
</tr>
</tbody>
</table>
### 2.1.4.5. Use Case: Delete Event

<table>
<thead>
<tr>
<th>Name</th>
<th>Delete Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>Objective</td>
<td>Delete one event from the database</td>
</tr>
<tr>
<td>Priority</td>
<td>Medium</td>
</tr>
<tr>
<td>Extends</td>
<td></td>
</tr>
<tr>
<td>Includes</td>
<td></td>
</tr>
<tr>
<td>Actors</td>
<td>Registered User</td>
</tr>
<tr>
<td>Pre-conditions</td>
<td>Sign into the system</td>
</tr>
<tr>
<td>Post-conditions</td>
<td>An Event will be removed from the database</td>
</tr>
</tbody>
</table>
| Main Scenario | 1. The user selects one particular that he/she has created.  
|              | 2. The system receives the request and looks for the specific item in the database.  
|              | 3. The system removes the event.                                              
|              | 4. The system shows the results of the operation                             |

#### Secondary Scenario

#### Exceptional Scenario

- If there’s an error while deleting the event the user will be notified

#### Function

- Delete event

#### Non functional requirements

#### Execution

#### Frequency

- Medium

#### Annotations
2.1.4.5.1. Sequence Diagram

User

: Registered User

: EventListUI
: EventListUIController

: ModelFacade
: EventControl
: Event

delete event

confirm
delteEvent( )
delteEvent( e_id)
delteEvent( e_id )
<<remove>>
showConfirmation( )
showResults( )
showResults( )
showResults( )
2.1.4.5.2. Collaboration Diagram

1: delete event
3: confirm

2: showConfirmation()

4: deleteEvent()

5: deleteEvent(e_id)

6: deleteEvent(e_id)

7: <<remove>>

8: showResults()

9: showResults(

10: showResults()
3. Static View

3.1. Architecture Diagram

The clients will use a Web browser to connect to the application server that will be divided in three tiers that can be implemented in different machines. The presentation tier is in charge of accepting requests from the client, ask the application for the appropriate answer and generate the view with the results for the client. We will use Java Server Faces (JSF) to implement the controller and generate the view. The application tier is in charge of implementing the business model. We will use Enterprise Java Beans (EJB) in this tier. Finally, we have the data tier where we will use an Oracle database to manage the information used by the application.
3.2. Class Diagram

3.2.1. User Module

- **User**
  - checkLogin()
  - registerUser()
  - persist()
  - getUserList()
  - getUserProfile()
  - getBlogEntries()
  - searchUsers()
  - deleteUser()
  - updateProfile()

- **View**
  - checkLogin()
  - showResults()
  - registerUser()
  - getUserList()
  - returnResults()
  - getUserProfile()
  - searchUsers()
  - getPublicEvents()
  - viewPublicEvent()
  - deleteUser()
  - updateProfile()
  - createEvent()
  - getEvents()
  - deleteEvent()

- **Model**
  - checkLogin()
  - showResults()
  - registerUser()
  - getUserList()
  - returnResults()
  - getUserProfile()
  - searchUsers()
  - getPublicEvents()
  - viewPublicEvent()
  - deleteUser()
  - updateProfile()
  - createEvent()
  - getEvents()
  - deleteEvent()

- **ModelFacade**
  - checkLogin()
  - showResults()
  - registerUser()
  - getUserList()
  - returnResults()
  - getUserProfile()
  - searchUsers()
  - getPublicEvents()
  - viewPublicEvent()
  - deleteUser()
  - updateProfile()
  - createEvent()
  - getEvents()
  - deleteEvent()

- **MainUI**
  - showLoginForm()
  - checkResults()
  - register()
  - showWelcome()

- **RegisterUI**
  - submit()
  - validateFields()
  - showResults()

- **RegisterUIController**
  - submit()
  - validateFields()
  - showResults()

- **MainUIController**
  - checkLogin()
  - validateInput()
  - showResults()
  - checkResults()
  - register()
  - showWelcome()

- **UserControl**
  - checkLogin()
  - registerUser()
  - persist()
  - getUserList()
  - getUserProfile()
  - getBlogEntries()
  - searchUsers()
  - deleteUser()
  - updateProfile()
3.2.2. Unregistered User Module

Unregistered User
(From Use Case View)

WelcomeUI
show()

UserListUI
showUserList()
showConfirmation()

UserProfileUI
showProfile()

SearchUsersUI
showSearchForm()
showSearchResults()

WelcomeUIController
getUsers()
returnResults()
getPublicItems()
showSearchForm()
searchUsers()
showResults()
editUserProfile()
createEvent()

UserListUIController
getUserProfile()
showResults()
deleteUser()

EventListUIController
viewPublicEvent()
showResults()
deleteEvent()

EventUI
showEvent()

eventList()
showResults()
showConfirmation()

EventUI
showSearchForm()
showSearchResults()

EventControl
getPublicEvents()
createEvent()
persist()
getEvents()
viewPublicEvent()
deleteEvent()

List
add()

Model
(Model Facade)

checkLogin()
showResults()
registerUser()
getUserList()
returnResults()
getUserProfile()
searchUsers()
getPublicEvents()
viewPublicEvent()
deleteUser()
updateProfile()
createEvent()
getEvents()
deleteEvent()

Blog

AppUser
(AppUser View)

checkLogin()
registerUser()
deleteUser()
getProfile()

UserControl
(UserControl View)

checkLogin()
registerUser()
deleteUser()
getProfile()
getBlogEntries()
searchUsers()
deleteUser()
updateProfile()
3.2.3. Registered User Module
3.2.4. Model Classes

- **<entity>** Blog
- **<entity>** AppUser
- **<entity>** UserProfile
- **<entity>** Filter
- **<control>** UserControl
  - checkLogin()
  - registerUser()
  - persist()
  - getUserList()
  - getUserProfile()
  - getBlogEntries()
  - searchUsers()
  - deleteUser()
  - updateProfile()
- **<boundary>** ModelFacade
  - checkLogin()
  - showResults()
  - registerUser()
  - getUserList()
  - returnResults()
  - getUserProfile()
  - searchUsers()
  - getPublicEvents()
  - NewPublicEvent()
  - deleteUser()
  - updateProfile
- **<control>** EventControl
  - getPublicEvents()
  - createEvent()
  - persist()
  - getEvents()
  - viewPublicEvent()
  - deleteEvent()
- **<entity>** Event
  - add()