**SWE 574 Software Development As A Team, Fall 2015**

**Instructor: Suzan Üsküdarlı**

**Project BUCOMP**

**Design Specifications Document**

**09.11.2015**

**Revision 1.1**

**By Group 1**

# Revision History

|  |  |  |
| --- | --- | --- |
| **Revision** | **Date** | **Explanation** |
| 1.0 | 02.11.2015 | Initial design |
| 1.1 | 09.11.2015 | All sections are updated. |

# Introduction

The purpose of this software project is to develop the BUCOMP, **B**oğaziçi **U**niversity **CO**mmunity **M**anagement **P**latform, in Java, Linux/Windows and MySQL environment. BUCOMP is a web/mobile application which allows its users to manage communities and communications among community members by providing various functionalities.

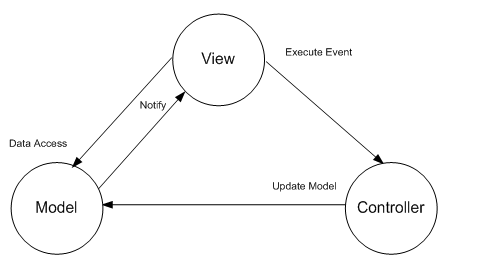
The design is based on BUCOMP Requirements Specification Document, final revision. The notation used in this document to describe the design of the application is mainly UML and conforms to organizational specifications given in [1]. The software architecture, overall high-level structure, components in terms of packages and classes and design details of all application functions and the user interface are given in later sections of this document.

# System Design

Since BUCOMP will be a web/mobile application and it will serve clients all over the world, its system design was planned and created to meet the associated needs of such an application. This section will explain overall design decisions of BUCOMP.

## Software Architecture

MVC is used in BUCOMP project as the software architectural pattern. The model is MySQL; View is an interface that is created with HTML, CSS and JQuery classes. Controller is the Java based web service and the communication between controller and the view is done by Ajax methods. See the figure below for MVC pattern.



## Software System Components

The BUCOMP solution will comprise of following system components.

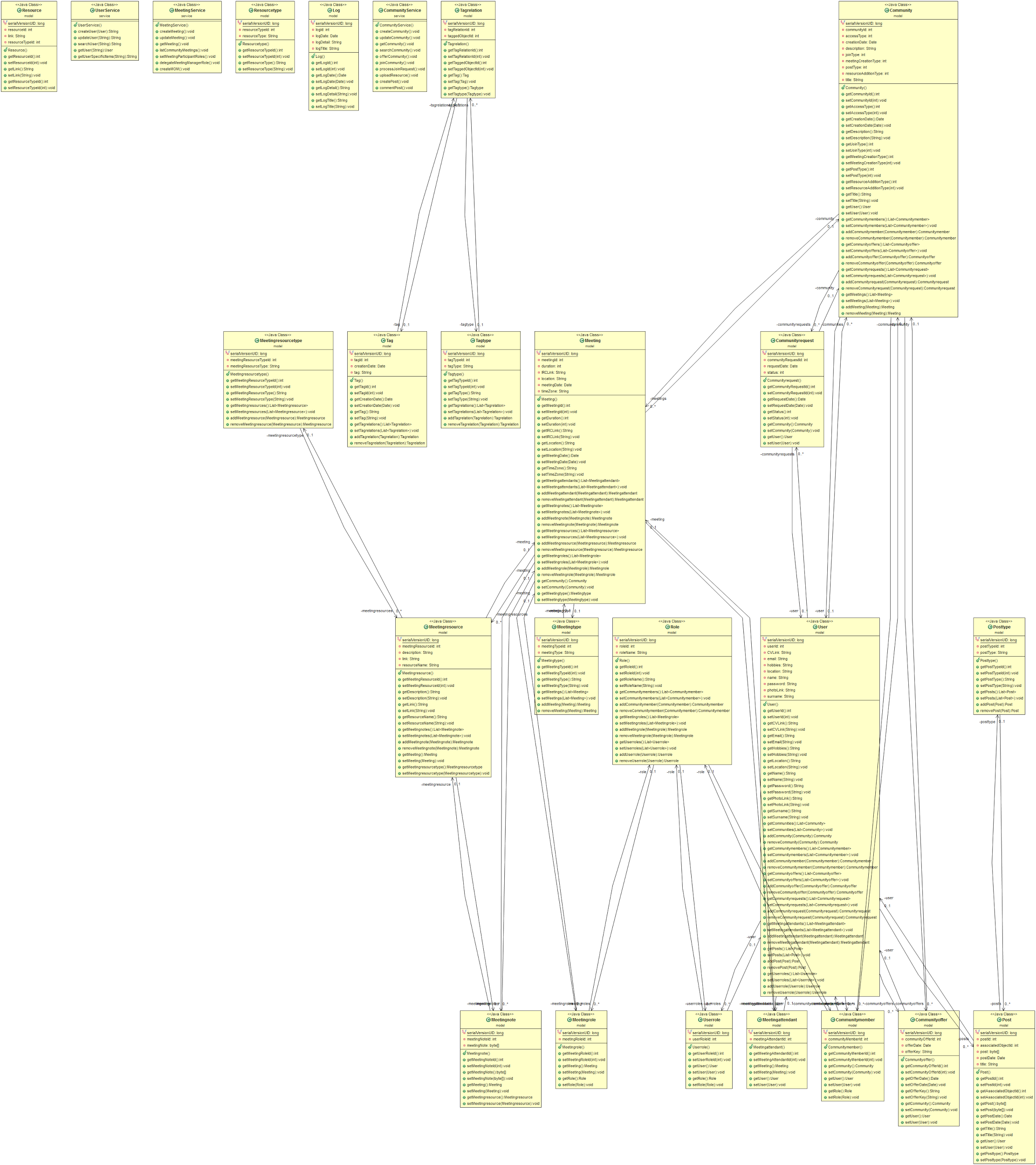
1. Database management system
2. Backend application
3. Web application
4. Mobile application

Backend application constitute the application’s backbone as it will provide access to database, and both mobile and web application will consume its RESTful web services. This part of the application is mainly responsible for data, thus a UML class diagram to show all the classes and data attribute is provided.

The classes in this diagram can be divided in 2 groups. First group is entity group, which corresponds database tables and largely for data representation. It conforms the “data transfer object” paradigm.

The second part is related to service classes, mainly provide web service methods which carry business logic, to manipulate data inside entity classes.

Following is the UML class diagram for BuComp back-end application.



[Full View of the Class Diagram](https://drive.google.com/file/d/0B5W-_hH437BiNDVxMzgzdDc5NTQ/view?usp=sharing)

## Environment

BUCOMP application development & runtime environment tools are listed in below:

|  |  |  |
| --- | --- | --- |
| Name | Type | Version |
| OS | Windows or Linux |  |
| Java | Programming Language | JDK 1.8 |
| JQUERY | Javascript Library | 1.11.x |
| Eclipse | IDE | Mars 4.5 |
| Maven | Build & Dependency Management | 3.3.3 |
| IntelliJ | IDE | 2015 |
| MySQL Server | Database Server | 5.7.9 |
| MySQL Workbench | Database Design & Modeling | 6.3.5 |
| Postman | Build, test and documentation tool for APIs. |  |

# Detailed Design

BUCOMP has 3 general software components consisting of frontend, backend and database management system. This section will include details of each components supported with mockups created for UI, signatures of services served by backend application, DAO and authentication & authorization decisions.

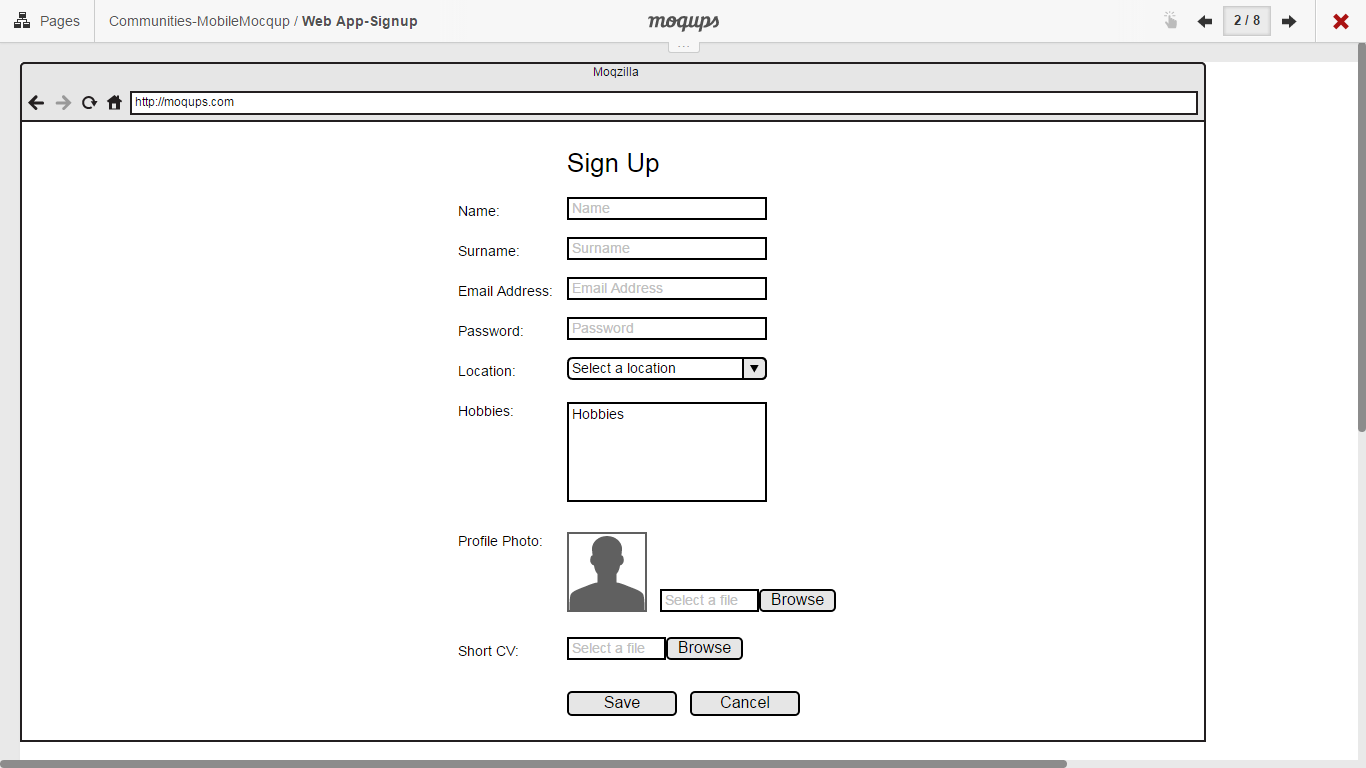
## Frontend Design

The BUCOMP system is divided into three components for web application. The view component will use;

* HTML pages
* CSS scripts for designing HTML pages
* Ajax call for controller methods
* JSON data that will be provided by Ajax calls

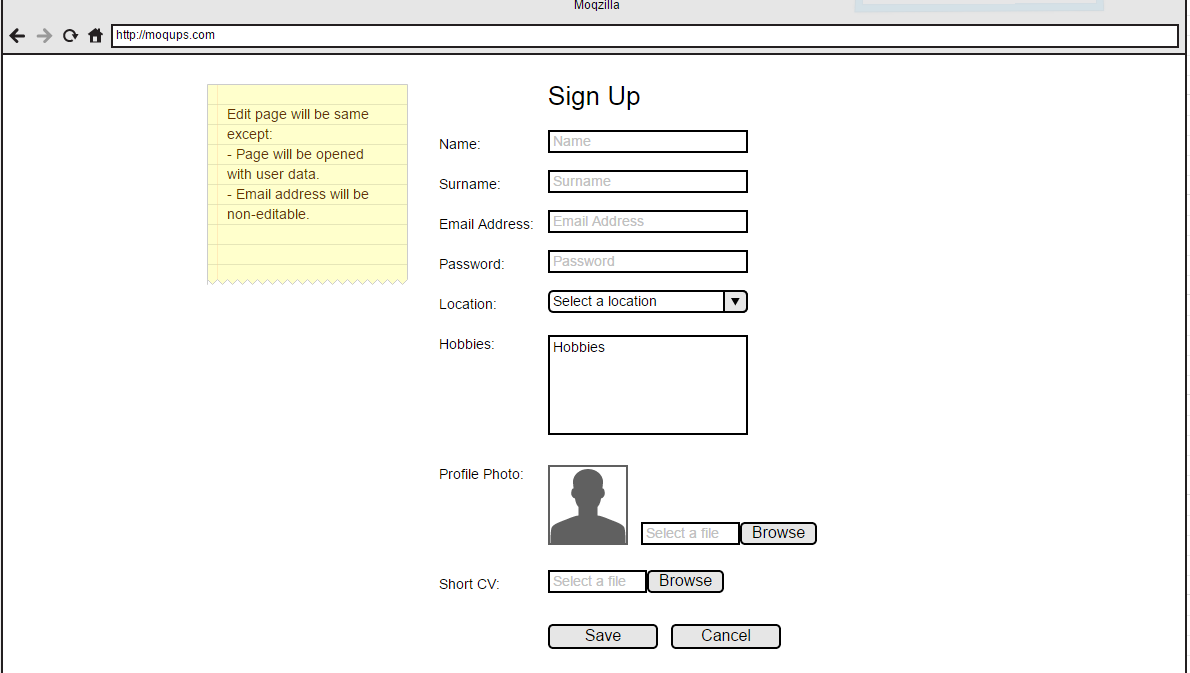
## Create User

Visitors should be able to register to the system by creating their profile on the system via “register.html” page which is shown in mockup below.



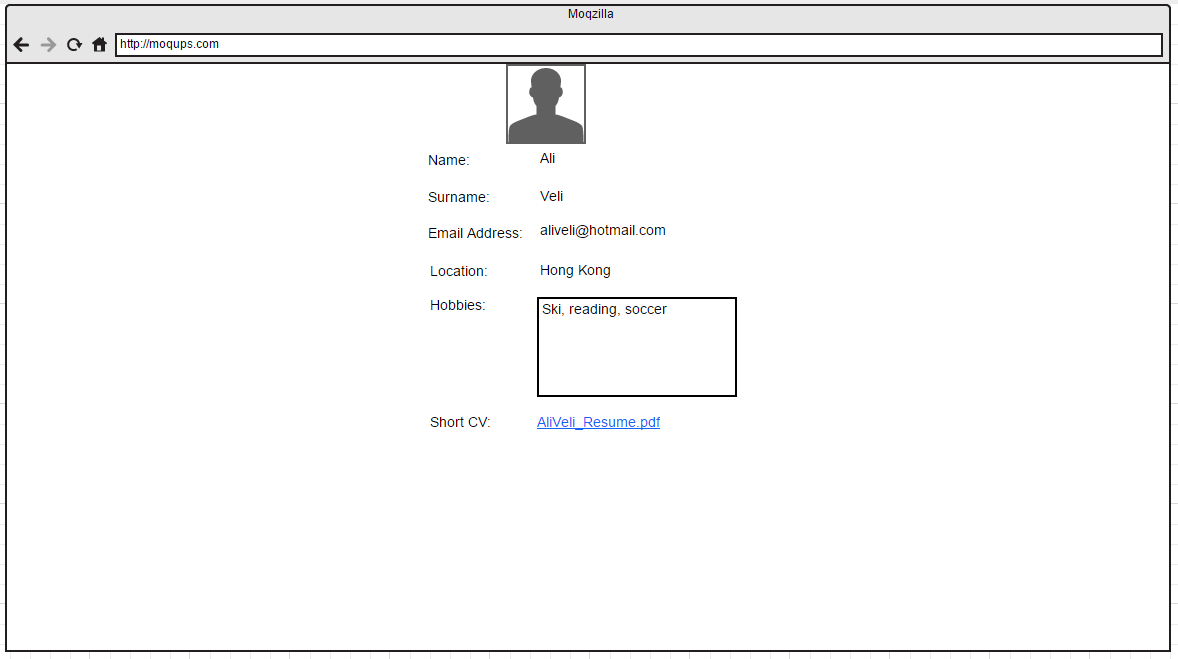
## Update User Information

Registered users should be able to update their profile information. on the system via “profile.html” page which is shown in mockup below.



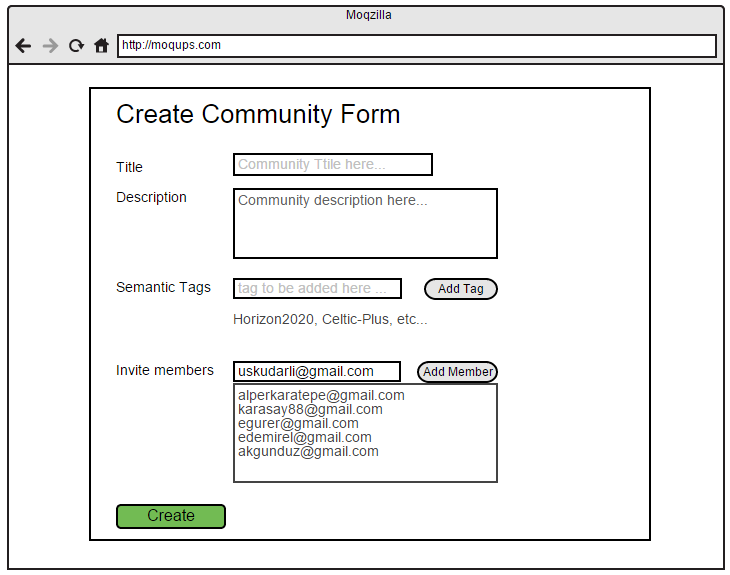
## Show/update User Details

## All registered users should be listed and user details should be displayed when clicked on the name via “UserProfile.html” page which is shown in mockup below.



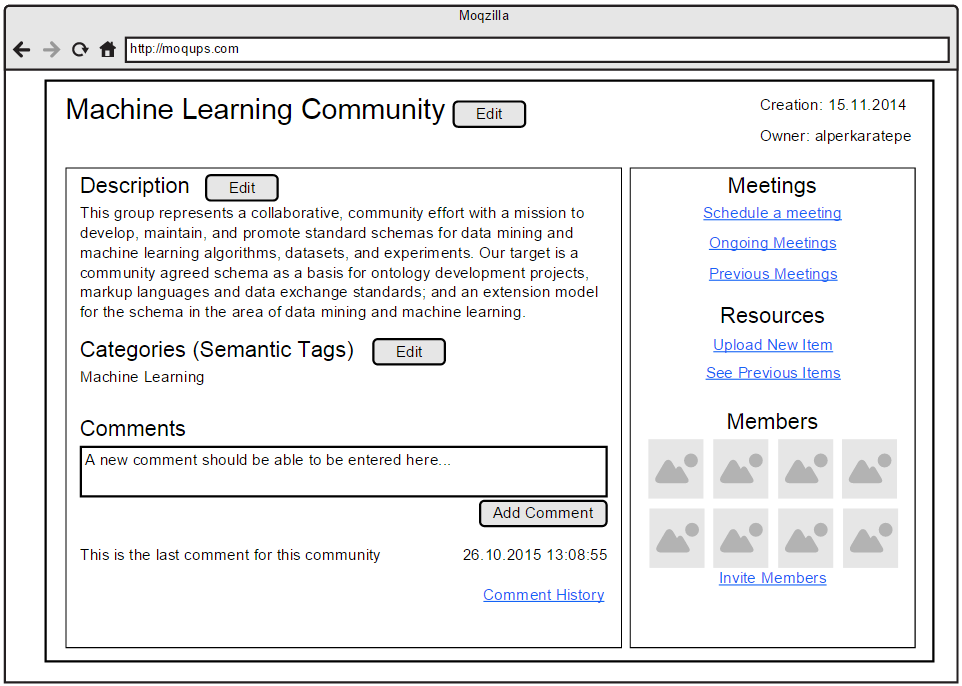
## Create Community

## Registered users should be able to create communities via “CreateCommunity.html” page which is shown in mockup below. The same page will be used for updating related community information.



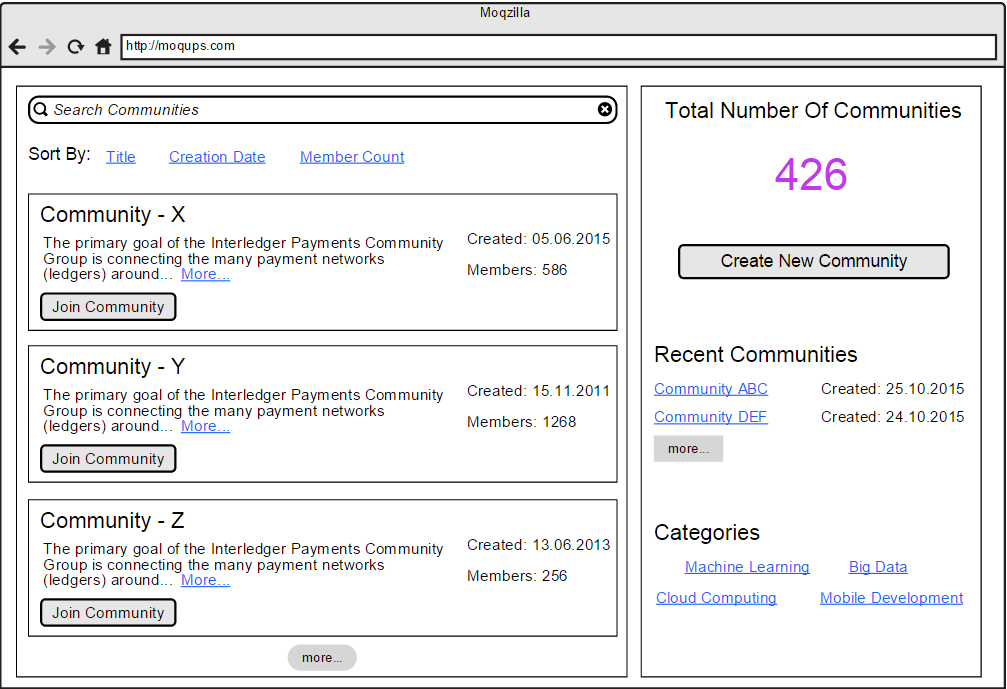
## Show Community Details

## When clicked on the Community name, community details should be able to be viewed in a separate page via “CommunityDetails.html” page which is shown in mockup below.



## Search And List Community

## All users should be able to search communities and list the result of any search via “CommunitySearch.html” page which is shown in mockup below.



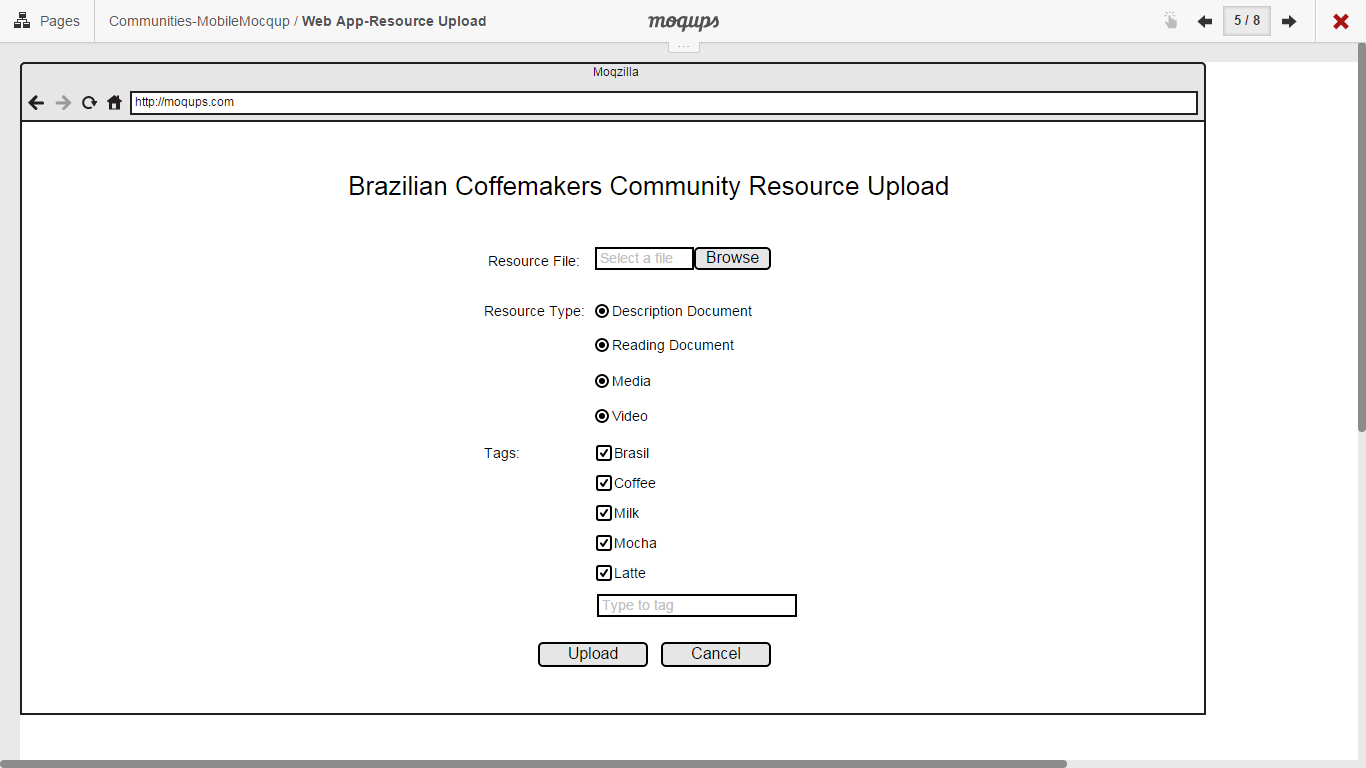
## List and Proceed Community Join Requests

## Users should be able to send request to join any community.Community admins should be able to see the requests coming from users for joining the community via “CommunityRequests.html” page which is shown in mockup below.



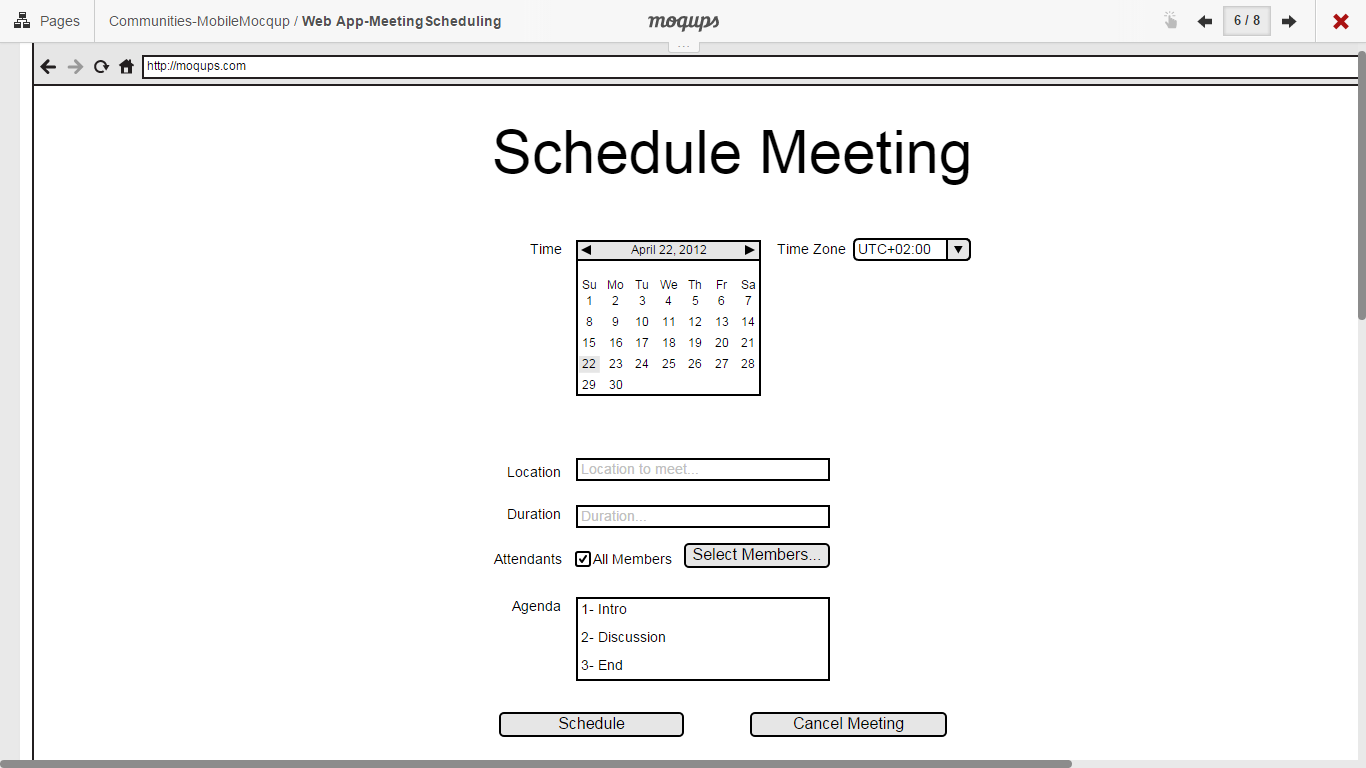
## Upload Resources to Community

## Community admins should be able to upload resources via “CommunityResources.html” page which is shown in mockup below.



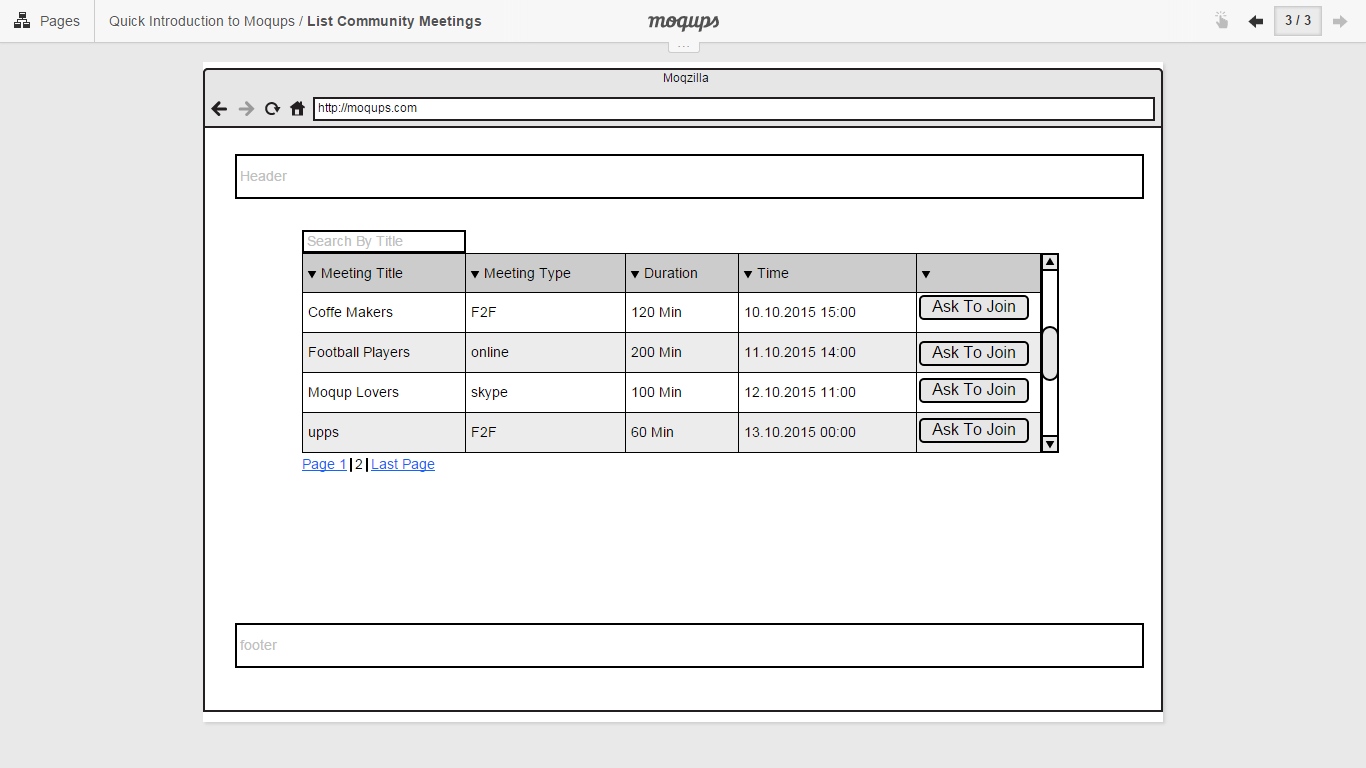
## Create Meeting

## Community members will be able to create meetings via “CreateMeeting.html” page which is shown in mockup below. Created Meeting will be updatable on the same page.



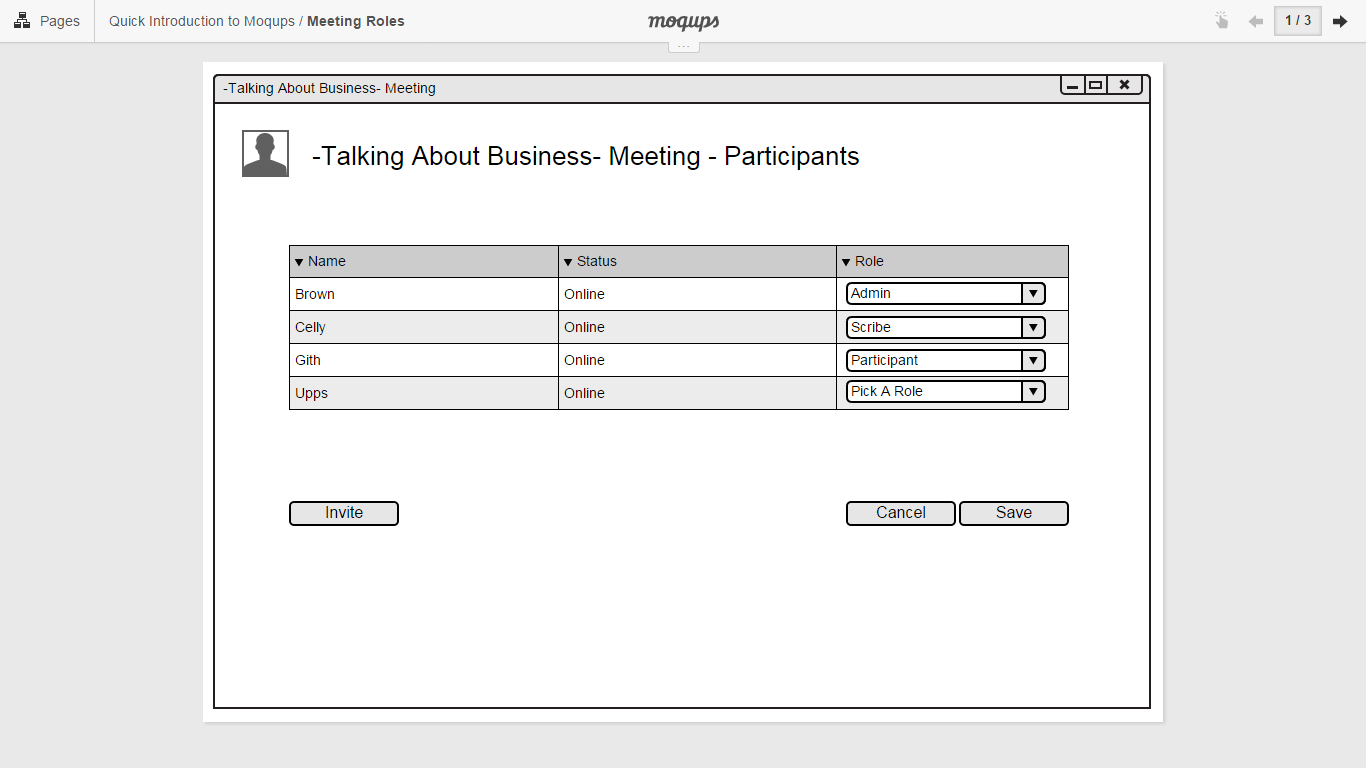
## List Community Meetings

## Community members should be able to see the list of meetings organized in that community via “CommunityMeeting.html” page which is shown in mockup below.



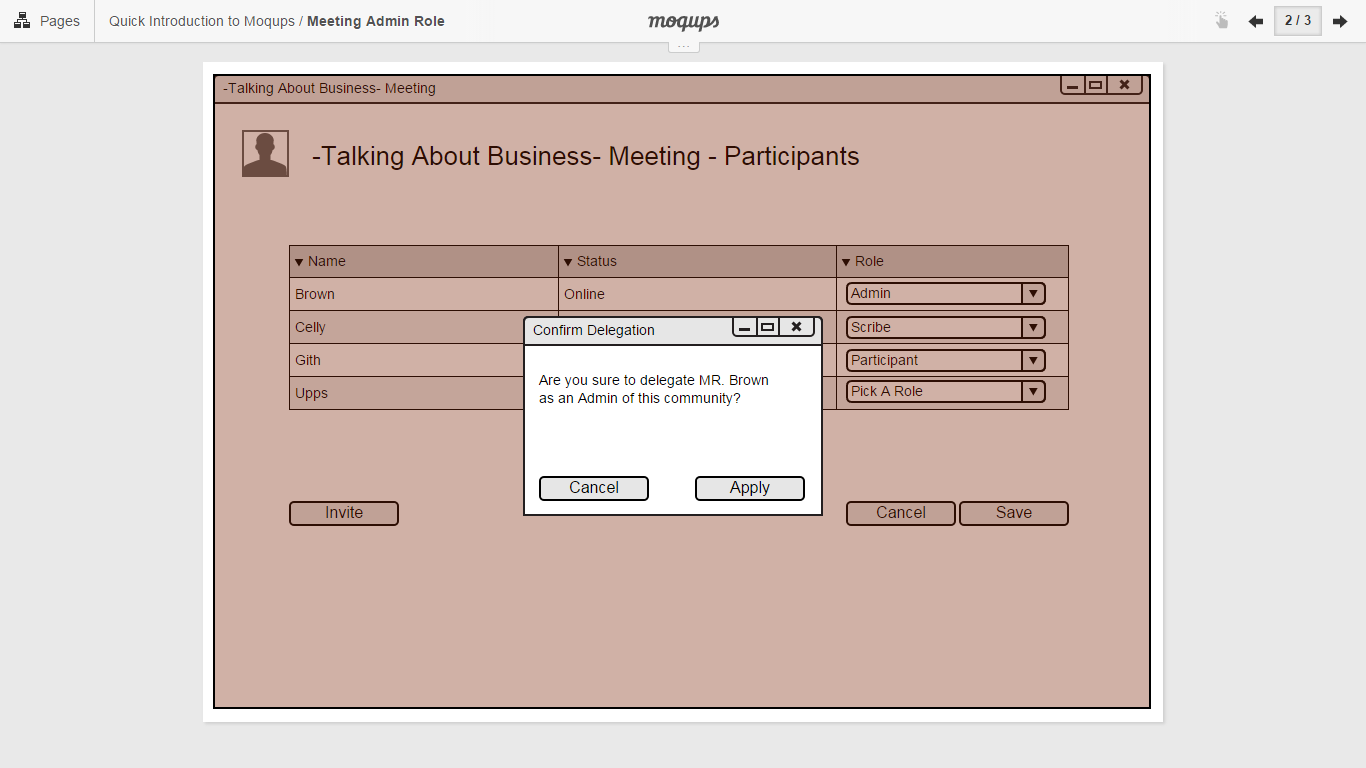
## Set Participants’ Role on Meeting

## The manager of the meeting should be able to determine the roles of the Meeting participants via “RoleMeeting.html” page which is shown in mockup below.



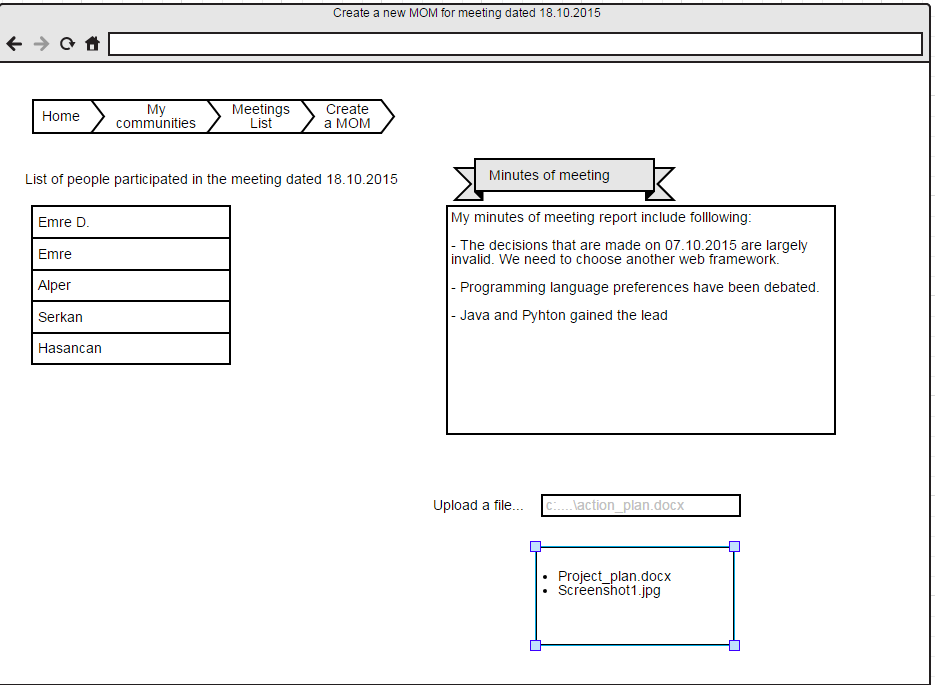
## Delegate Managers’ Role on Meeting

## Meeting manager will be able to delegate managerial role to other members of the community via “RoleMeeting.html” page which is shown in mockup below.



## Create MOM

## Note takers will be able to create and upload meeting decision documents via “CreateMOM.html” page which is shown in mockup below.



## Show Reports

## The system should be able to show required reports in a dashboard style page via “Reports.html” page.

## Create Post

## Community members should be able to upload media, document or write text for the posts section of the community via “postfiles.html” page.

## Comment Post

## Community members should be able to write comment for the posts via “postfiles.html” page.

## Comment Post

## The system should display the user-specific items on landing page after logged-in via “UserProfile.html” page.

## Backend Design

Backend side of the application will consist of a web service api and database access layer.

### Web Service API

Backend side of the application will provide following web service api for frontend side.

The API will be available at this location: http://<server-ip>:8080/api/\*

#### User Services

1. **getUsers**

URL : /api/users

HTTP Method : GET

Request : Empty

Response : APPLICATION\_JSON\_VALUE (Collection of User object), Empty if no record found.

HTTP Status : Successful Case : OK (200)

If no record found : NO\_CONTENT (204)

1. **getUserById**

URL : /api/users/{id}

HTTP Method : GET

Request : Empty

Response : APPLICATION\_JSON\_VALUE (User object), Empty if no record found.

HTTP Status : Successful Case : OK (200)

If no record found : NO\_CONTENT (204)

1. **createUser**

URL : /api/users

HTTP Method : POST

Request : APPLICATION\_JSON\_VALUE (User object)

Response : APPLICATION\_JSON\_VALUE (User object)

HTTP Status : Successful Case : CREATED (201)

Error Case : INTERNAL\_SERVER\_ERROR (500)

1. **updateUser**

URL : /api/users/{id}

HTTP Method : PUT

Request : APPLICATION\_JSON\_VALUE (User object)

Response : APPLICATION\_JSON\_VALUE (User object)

HTTP Status : Successful Case : OK (200)

Error Case : INTERNAL\_SERVER\_ERROR (500)

1. **deleteUser**

URL : /api/users/{id}

HTTP Method : DELETE

Request : Empty

Response : APPLICATION\_JSON\_VALUE (User object)

HTTP Status : Successful Case : OK (200)

Error Case : INTERNAL\_SERVER\_ERROR (500)

1. **getUserSpecificItems** ?

What are user specific items?

#### Community Services

1. **getCommunities**

URL : /api/communities

HTTP Method : GET

Request : Empty

Response : APPLICATION\_JSON\_VALUE (Collection of Community object)

HTTP Status : Successful Case : OK (200)

If no record found : NO\_CONTENT (204)

1. **getCommunityById**

URL : /api/communities/{id}

HTTP Method : GET

Request : Empty

Response : APPLICATION\_JSON\_VALUE (Community object)

HTTP Status : Successful Case : OK (200)

If no record found : NO\_CONTENT (204)

1. **createCommunity**

URL : /api/communities

HTTP Method : POST

Request : APPLICATION\_JSON\_VALUE (Community object)

Response : APPLICATION\_JSON\_VALUE (Community object)

HTTP Status : Successful Case : CREATED (201)

Error Case : INTERNAL\_SERVER\_ERROR (500)

1. **updateCommunity**

URL : /api/communities/{id}

HTTP Method : PUT

Request : APPLICATION\_JSON\_VALUE (Community object)

Response : APPLICATION\_JSON\_VALUE (Community object)

HTTP Status : Successful Case : OK (200)

Error Case : INTERNAL\_SERVER\_ERROR (500)

1. **deleteCommunity**

URL : /api/communities/{id}

HTTP Method : DELETE

Request : Empty

Response : APPLICATION\_JSON\_VALUE (Community object)

HTTP Status : Successful Case : OK (200)

Error Case : INTERNAL\_SERVER\_ERROR (500)

1. **searchCommunity**

URL : /api/communities?key=<value>

HTTP Method : GET

Request : Empty

Response : APPLICATION\_JSON\_VALUE (Collection of Community object)

HTTP Status : Successful Case : OK (200)

If no record found : NO\_CONTENT (204)

1. **offerCommunity**

URL : /api/communities?userId=<value>

HTTP Method : GET

Request : Empty

Response : APPLICATION\_JSON\_VALUE (Collection of Community object)

HTTP Status : Successful Case : OK (200)

If no record found : NO\_CONTENT (204)

1. **requestCommunityMembership**

URL : /api/communities/membership?userId=<value>&communityId=<value>

HTTP Method : POST

Request : Empty

Response : Empty

HTTP Status : Successful Case : OK (200)

Error Case : INTERNAL\_SERVER\_ERROR (500)

1. **processMembershipRequest**

URL : /api/communities/membership?userId=<value>&communityId=<value>&isApproved=<yes|no>

HTTP Method : PUT

Request : Empty

Response : APPLICATION\_JSON\_VALUE (Collection of Community object)

HTTP Status : Successful Case : OK (200)

Error Case : INTERNAL\_SERVER\_ERROR (500)

1. **leaveCommunity**

URL : /api/communities/membership?communityMemberId=<value>

HTTP Method : DELETE

Request : Empty

Response : Empty

HTTP Status : Successful Case : OK (200)

Error Case : INTERNAL\_SERVER\_ERROR (500)

1. **uploadResource**

URL : /api/communities/documents

HTTP Method : POST

form-data parameters :

* file : A file posted in a multipart request
* communityId : Community which the uploaded resource belongs to.
* userId : User who uploads the resource.

Request : Empty

Response : APPLICATION\_JSON\_VALUE (Resource object)

HTTP Status : Successful Case : OK (200)

Error Case : INTERNAL\_SERVER\_ERROR (500)

1. **createPost**
2. **commentPost**

#### Meeting Services

1. **getCommunityMeetings**

URL : /api/meetings/{communityId}

HTTP Method : GET

Request : Empty

Response : APPLICATION\_JSON\_VALUE (Collection of Meeting object)

HTTP Status : Successful Case : OK (200)

If no record found : NO\_CONTENT (204)

1. **createMeeting**

URL : /api/meetings/{communityId}

HTTP Method : POST

Request : APPLICATION\_JSON\_VALUE (Meeting object)

Response : APPLICATION\_JSON\_VALUE (Meeting object)

HTTP Status : Successful Case : CREATED (201)

Error Case : INTERNAL\_SERVER\_ERROR (500)

1. **updateMeeting**

URL : /api/meetings/{id}

HTTP Method : PUT

Request : APPLICATION\_JSON\_VALUE (Meeting object)

Response : APPLICATION\_JSON\_VALUE (Meeting object)

HTTP Status : Successful Case : OK (200)

Error Case : INTERNAL\_SERVER\_ERROR (500)

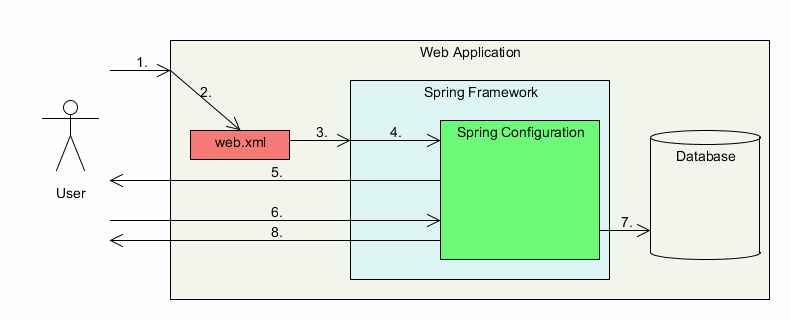
1. **setMeetingParticipantRoles**
2. **delegateMeetingManagerRole**
3. **createMOM**

#### Reporting Services

### Database Access Layer

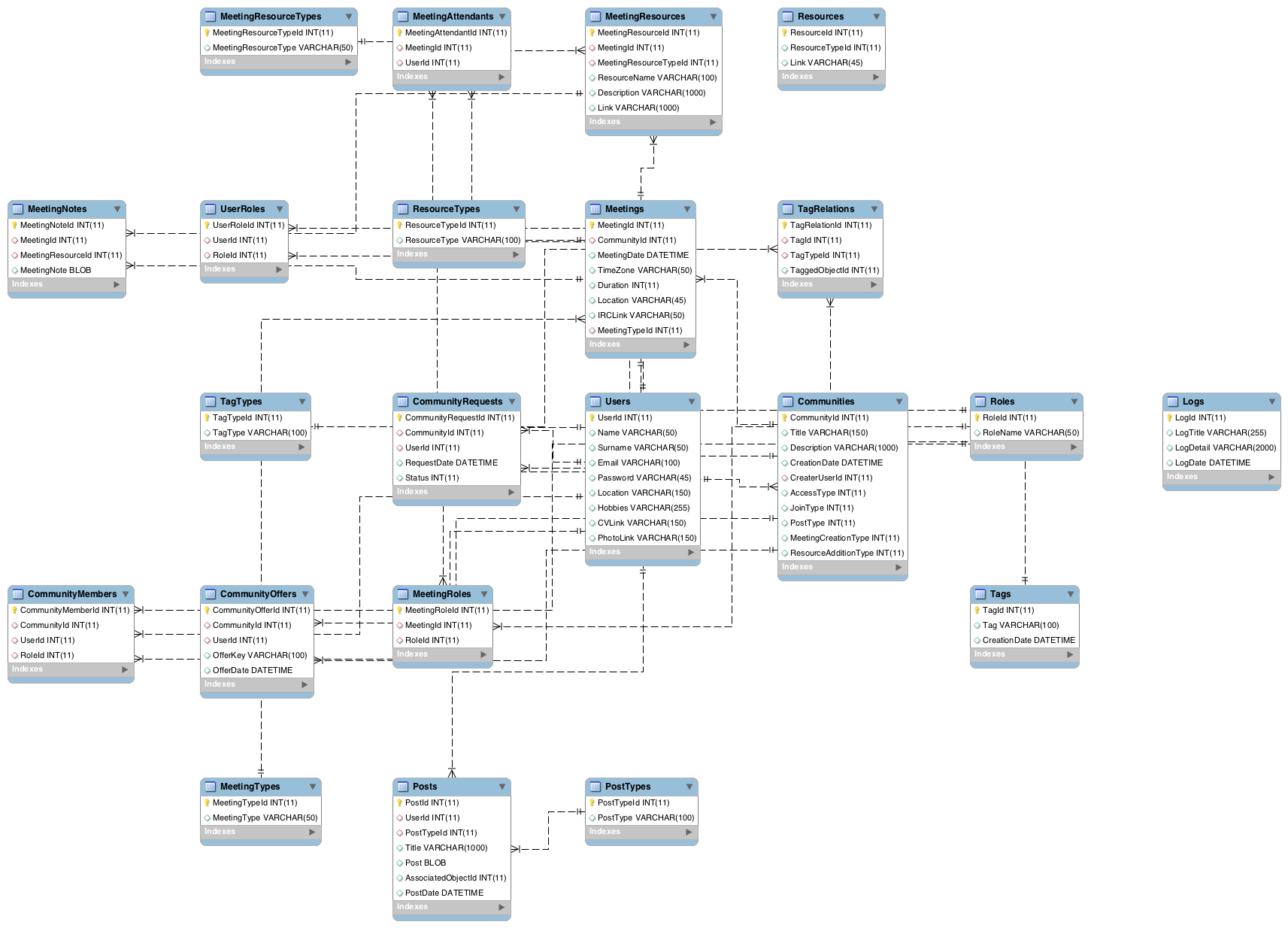
## Authentication & Authorization

Spring-security framework will be used for authentication and authorization requirements. The general interaction flow is shown in figure below:



1. User accesses a URL on a web application
2. The web application refers to web.xml
3. The web.xml matches the URL pattern
4. The control is redirected to *DispatcherServlet* in Spring framework
5. Spring framework finds that the all URLs are secured and hence displays login page to the user
6. The user enters his login name and password
7. Spring configuration file knows that it has authenticate against the database and hence it accesses the database
8. Spring performs the authentication and authorization against the database and if the credentials are successful then it redirects to the original accessed URL [2]

# Database Design



# References

[1] UML2.0 Specifications, [http://www.uml.org/#UML2.0.](http://www.uml.org/#UML2.0)

[2] Spring Security: Authentication and Authorization Using Database, http://www.studytrails.com/frameworks/spring/spring-security-using-db.jsp