Diana Ramírez

Alhassane Bah

Web Services

Class Project

Resource Oriented Architecture

1. Objectives

- Understanding the Representational State Transfer (REST).
- Design of a resource-oriented architecture.

2. Functional Requirements

The main goal of this project is to design a social meetings web service, which provides a subset of the functionalities of a social networking portal that facilitates the group meetings between interested parties.

To accomplish with this task, we have considered the next System entities:

- User, that has the following attributes:
  - Id
  - email address (used to identify a particular user)
  - first name
  - last name
  - biography
  - A list of groups that the user is a member of.
  - A list of groups that the user is joined to.
  - Password

- Group, which has the following attributes:
  - Id
  - unique name
  - description
  - user who is the admin of the group
  - a list of users which are the group members
  - discussion board which holds a list of comments

- Comment
  - Id
  - user posting the comment
  - date of the comment
  - comment

The system will supports the following scenarios which are explained:

- User sign up
  - A user is able to sign up in the application by using their Facebook or Twitter account.

        - In this scenario our application will be a client that will register a new user by using their twitter or facebook accounts.
    - A user is able to sign up in the application by adding their email address and a password.
- User changes his/her full name and/or biography
    - A user can update the information of their profile
- User deletes his/her account
    - A user can delete his own account, once the account is deleted, the user will be redirected to the sign up page.
- User creates new group
    - A user can see all the groups created and it can create a group himself, the user will become the group administrator.
- User changes the description of the group he/she owns
    - A user can update the information about the groups where he is the administrator.
- User deletes groups he/she owns
    - A user can delete the groups managed by him. Once a group is deleted, the users that where members of it will no longer belong to the group as well.
- User views a list of groups with descriptions and membership count
    - Once a user signs in, a list of all the available groups with their name and description is displayed.
- User joins group
    - A user can join to any of the groups displayed.
- User views members of the group he/she owns or has joined
    - A user can choose one of the groups he's joined and the members of the group will be displayed.
- User comments on the dashboard of the group
    - A user is able to submit a comment in the group dashboard which holds a list of comments.
- User leaves the group
    - A user can decide to leave a group. The group won't be displayed anymore as one of the groups he is member of.
- User checks the profile of other users
    - Once a user is able to see the members of the groups he belongs to, he can as well choose a profile of a member and check it by selecting it.

The entities are exposed as addressable, interconnected resources with suitable representations, and the scenarios are supported through the uniform HTTP request interface.

3. Resource Oriented Architecture

The resources described for our architecture are:

- Users
- Groups
- Comments

These resources, support the following two representations of the user, group and comment resources:
– HTML (for human use)
– JSON (for programmatic clients)

The uniform REST verbs are mapped to the functionality described in the requirements by the methods defined in the User controller:

Route	**HTTP Verb**	**Description**	**Scenario**
/	Request Mapping	Is the main path to access to all the methods that will change the state of the resources. The user session is taken here.	Sign up / Sign in. Access to all the methods.
/user/home/{id}	Get	This method takes the user id as parameter and returns an user with all the information to display about his/her profile	Displaying the user profile. User checks the profile of other users.
/user/save	Post	This method takes a new user object as a parameter and saves it in the database.	Sign up
/user/delete/{id}	Get	This method takes a user id as a parameter and deletes it from the database.	User deletes his/her account
/user/new	Get	This method will let us create a new user by using a form. It just creates an user object where the data written in the form will be kept.	It will be helpful for registering a new user
/user/edit/{id}	Get	This method takes a user id as a parameter to display its information and gets the requested user to be able to update its information later.	User changes his/her full name and/or biography
/user/update	Post	The method saves the information of a user profile that was modified.	User changes his/her full name and/or biography
/group/new	Get	This method will let us create a new group by using a form, It just creates a group object where the data written in the form will be kept.	It will be helpful for registering a new group
/group/create	Post	This method will save in the database the new group created by using the form.	User creates new group

/group/edit/{id}	Get	This method takes a group id as a parameter to display its information and gets the requested group to be able to update its information later.	User changes the description of the group he/she owns
/group/update	Post	The method saves the information of a group that was modified.	User changes the description of the group he/she owns
/group/delete/{id}	Get	This method takes the group id as a parameter to delete a group.	User deletes groups he/she owns
/group/leave/{id}	Get	This method takes as a parameter the id of the group that the user wants to leave from to remove it from the list of users where it belongs.	User leaves the group
/group/join/{id}	Get	This method takes a group Id as a parameter and it add the given group to the list of groups to which the user is joined.	User joins group
/group/getComment/{id}	Get	This method takes a group Id as a parameter and it displays all the comments that correspond to a group. In other words it returns a list of comments related to a group.	User comments on the dashboard of the group
/group/createComment	Post	This method saves a new comment in a given group.	User comments on the dashboard of the group
/group/getMember	Post	This method allows us to get all the members of a group and display them by sending the group id as a parameter.	User views members of the group he/she owns or has joined
/group/home	Get	This method takes a user as parameter and returns the lists of the groups a user owns	User views a list of groups with descriptions and

		and the ones he/she is member of	membership count

Note: The GET methods that are called to get entities instances are also listed to return json format objects.

4. Implementation

For this project we made use of Spring Boot as a Java based technology framework for managing the whole application.

We propose a MVC structure where we propose the previously described elements as our entities, we define a controller to manage the changes that required to be done over each entity and the view is displayed by using html.
Following this structure, the source files are distributed in controller, dao, model and service packages.
In the controller we define all the REST methods, the model has the entities defined and the dao and service packages keep the classes that are in charge of the persistence duties.

We made use of the annotations provided by the framework to make use of the REST verbs
And define our methods over our resources.

The persistence is provided by using MongoDB. Our entities are mapped in order to store the data in an embedded Mongo database provided by Spring boot.

Maven was used to manage the different dependencies (libraries) needed by the application.

Finally, the project can be found on github in the following repository: https://github.com/alhassaneBah/WS_Repo.git

5. Testing

We have defined a test suite using JUnit (for Java-based project) to prove that the persistence elements are working in a right way. The tests done are over the CRUD functions of each entity defined: Users, Groups and Comment. The tests are all listed in the src/test/java file, in the com.ws package in a class called WsProjectApplicationTests.java where all the test can be run automatically.

We have also defined a HTTP-client-based system-level testing of the services using the JSON or XML representation. In these case we make some http requests to prove that the json objects are effectively being returned. We make a request for all the GET methods defined in the controller and print the resulting json objects returned by the respective methods. This test are done in the class that launches the application so that the request results can be printed in console.

6. Conclusion

By working on this project we have RESTful services based application. We have created a client to consume a service by adding the sign in with Twitter and Facebook, as well as we have defined a service, exposing methods to which other people or programmatic clients can have access. We have been able to define a Resource Oriented Architecture, where we use the HTTP methods to do read and write operations (CRUD).