télécom saint-étienne
école d'ingénieurs nouvelles technologies

# Web Services: Class Proiect

Maxime Makhloufi
Guillaume Terpend

12/15/16

École associée INSTITUT Mines-Télécom
UNIVERSITÉ JEAN MONNET SAINT-ÉTIENNE

# INDEX

## 1- Mock-up structure for the system’s component

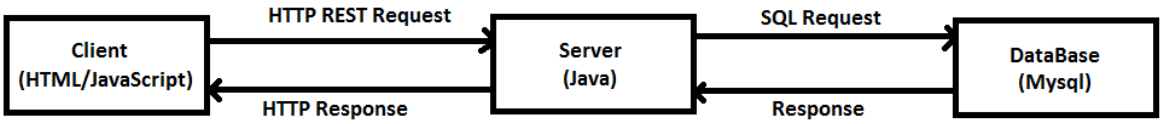


Figure 1: Mock-up structure of the system's components

## 2- Data Base

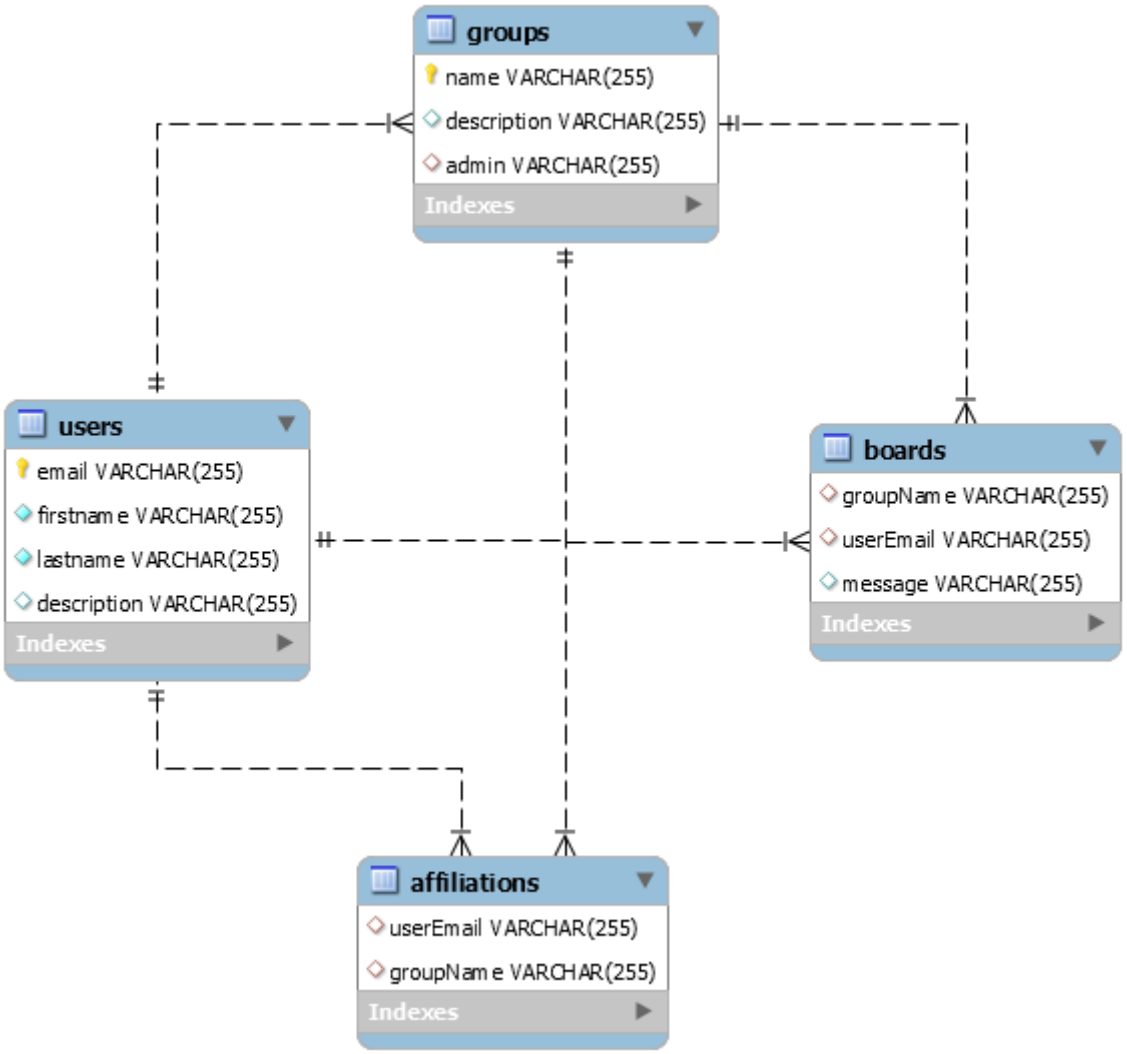


Figure 2: Data Base diagram

- **Table users:** The email is the unique ID (primary key)
- **Table groups:** The name is the unique ID (primary key)
- **Table boards:** No unique ID
- **Table affiliations:** No unique ID - This table is used to know which users are in which groups.

## 3- Server

The server will be in charge to receive the HTTP request from the client and to send SQL request to the data base.

We use a glassfish server and the JAX-RS API in order to create our web services.

We use four different packages:

- package com.makhloufiTerpend.model;
- package com.makhloufiTerpend.RESTfulService;
- package com.rest.DB;
- package com.rest.util;

### a. com.makhloufiTerpend.model

This package contains two classes: user.java and group.java. These are the POJO.

### b. com.rest.DB

This package contains two classes: DBClass.java and DBRequest.java.
DBClass.java is the class uses to speak with the database (use in TD5).
DBRequest.java contains the method use to send the SQL Request:

- **public static int createUser(User user)**
  Sends an SQL request to the database in order to create a user.
  Takes a user in parameters. If the request has been done successfully, the method will return 200, 400 if not.

- **public static String getUser(String email)**
  Sends an SQL request to the database in order to get a user.
  Takes the user's email in parameters. The method returns a String (JSON Object) that contains the information of the user if succeeded, null if not.

- **public static String getUser()**
  Sends a SQL request to the database in order to get all the users.
  The method returns a String (JSON Array) that contains the information of all the users in the data base if succeeded, null if not.

- **public static int updateUser(User user)**
  Sends an SQL request to the database in order to update a user.
  Takes a user in parameters. If the request has been done successfully, the method will return 200, 400 if not.

- **public static int deleteUser(User user)**
  Sends an SQL request to the database in order to delete a user.
  Takes a user in parameters. If the request has been done successfully, the method will return 200, 400 if not.

- **public static int createGroup(Group group)**
  Sends a SQL request to the database in order to create a group.
  Takes a group in parameters. If the request has been done successfully, the method will return 200, 400 if not.

- **public static String getGroup(String name)**
  Sends an SQL request to the database in order to get a group.
  Takes the group's name in parameters. The method returns a String (JSON Object) that contains the information of the group's information if succeeded, null if not.

- **public static String getGroup()**
  Sends an SQL request to the database in order to get all the groups.
  The method returns a String (JSON Array) that contains the information of all the groups in the data base if succeeded, null if not.

- **public static int updateGroup(Group group)**
  Sends an SQL request to the database in order to update a group.
  Takes a group in parameters. If the request has been done successfully, the method will return 200, 400 if not.

- **public static int deleteGroup(Group group)**
  Sends an SQL request to the database in order to delete a group.
  Takes a group in parameters. If the request has been done successfully, the method will return 200, 400 if not.

- **public static String getUserInGroup(Group group, User user)**
  Sends an SQL request to the database in order to check if a user is in group.
  Takes a group and a user in parameters. The method returns a String (JSON Object) that contains the information of the user if succeeded, null if not.

- **public static String getUserInGroup(Group group)**
  Sends an SQL request to the database in order to get all users in a group.
  Takes a group and a user in parameters. The method returns a String (JSON Array) that contains the information of all the users of the group if succeeded, null if not.

- **public static int joinGroup(User user, Group group)**
  Sends an SQL request to the database in order to join a group.
  Takes a user and a group in parameters. If the request has been done successfully, the method will return 200 (via the method joinBoard – see below), 400 if not.

- **public static int leaveGroup(User user, Group group)**
  Sends an SQL request to the database in order to leave a group.
  Takes a user and a group in parameters. If the request has been done successfully, the method will return 200 (via the method leaveBoard – see below), 400 if not.

- **public static int joinBoard(User user, Group group)**
  Sends an SQL request to the database in order to join a board. This method is called when we used the method joinGroup.
  Takes a user and a group in parameters. If the request has been done successfully, the method will return 200, 400 if not.

- **public static int leaveBoard(User user, Group group)**
  Sends an SQL request to the database in order to leave a board. This method is called when we used the method leaveGroup.
  Takes a user and a group in parameters. If the request has been done successfully, the method will return 200, 400 if not.

- **public static int createMessage(User user, Group group, String message)**
  Sends a SQL request to the database in order to write a message in the board of a group.
  Takes a user, a group, and a String in parameters. If the request has been done successfully, the method will return 200, 400 if not.

- **public static String getMessage(Group group)**
  Sends an SQL request to the database in order to get the board of a group.
  Takes a group in parameters. The method returns a String (JSON Array) that contains the messages in the board if succeeded, null if not.

### c. com.rest.util

This package contains the class ToJSON.java.
ToJSON.java is the class uses to parse the JSON (use in TDs).

### d. com.makhloufiTerpend.RESTfulService

This package contains two classes: SocialMeetingsApplication.java and SocialMeetings.java. The first one is used to define the application's path. The second one contains the methods allowing to access resources:

- **public Response createUser**
  Role: add a new user
  Method: POST
  Endpoint: user/{email}
  Parameters: email (String), firstname (String), lastname (String), description (String)
  Result: 200 + succeed message if succeed, 400 + failed message if not

- **public Response getUser**
  Role: get the current user
  Method: GET
  Endpoint: user/{email}
  Parameters: email (String)
  Result: 200 + the user in JSON format, 400 + failed message if not

- **public Response getUser**
  Role: get all the users
  Method: GET
  Endpoint: user
  Parameters: /
  Result: 200 + all the users in JSON format, 400 + failed message if not

- **public Response updateUser**
  Role: update the current user
  Method: PUT
  Endpoint: user/{email}
  Parameters: email (String), firstname (String), lastname (String), description (String)
  Result: 200 + succeed message if succeed, 400 + failed message if not

- **public Response deleteUser**
  Role: delete the current user
  Method: DELETE
  Endpoint: user/{email}
  Parameters: email (String)
  Result: 200 + succeed message if succeed, 400 + failed message if not

- **public Response createGroup**
  Role: add a new group
  Method: POST
  Endpoint: group/{name}
  Parameters: name (String), description (String), admin (String)
  Result: 200 + succeed message if succeed, 400 + failed message if not

- **public Response getGroup**
  Role: get the current group
  Method: GET
  Endpoint: group/{name}
  Parameters: name (String)
  Result: 200 + the group in JSON format, 400 + failed message if not

- **public Response getGroup**
  Role: get all the groups
  Method: GET
  Endpoint: group
  Parameters: /
  Result: 200 + all the groups in JSON format, 400 + failed message if not

- **public Response updateGroup**
  Role: update the current group
  Method: PUT
  Endpoint: group/{name}
  Parameters: name (String), description (String), admin (String)
  Result: 200 + succeed message if succeed, 400 + failed message if not

- **public Response deleteGroup**
  Role: delete the current group
  Method: DELETE
  Endpoint: group/{name}
  Parameters: name (String), description (String), admin (String)
  Result: 200 + succeed message if succeed, 400 + failed message if not

- **public Response getUserInGroup**
  Role: get the current user if he is in the current group
  Method: GET
  Endpoint: group/{name}/user/{email}
  Parameters: name (String), email (String)
  Result: 200 + the user in JSON format, 400 + failed message if not

- **public Response getUserInGroup**
  Role: get all the users of the current groups
  Method: GET
  Endpoint: group/{name}/user
  Parameters: name (String)
  Result: 200 + all the users in JSON format, 400 + failed message if not

- **public Response joinGroup**
  Role: add the current user to the current group
  Method: POST
  Endpoint: "group/{name}/user/{email}"
  Parameters: name (String), email (String)
  Result: 200 + succeed message if succeed, 400 + failed message if not

- **public Response deleteUser**
  Role: remove the current from the current group
  Method: DELETE
  Endpoint: group/{name}/user/{email}
  Parameters: name (String), email (String)
  Result: 200 + succeed message if succeed, 400 + failed message if not

- **public Response getMessage**
  Role: get all the message from the current groups's board
  Method: GET
  Endpoint: "group/{name}/message"
  Parameters: name (String)
  Result: 200 + all messages and their publishers in JSON format, 400 + failed message if not

- **public Response publishMessage**
  Role: add a new message to the current group's board
  Method: POST
  Endpoint: "group/{name}/user/{email}/message"
  Parameters: name (String), email (String), message (String)
  Result: 200 + succeed message if succeed, 400 + failed message if not

## 4- Client

The client is composed of six HTML pages that work with JavaScript to display information.

The HTML pages are the following:

- **index.html**: default page
- **signup.html**: to allow a client to sign up
- **login.html**: to allow a client to log in
- **profile.html**: to display a user's profile
- **groups.html**: to create and see existing groups
- **groupprofile.html**: to display a group's profile

The scripts are the followings:

- **script.js:**
  Used by all the HTML pages.
  The script performs: - the cookies (creation, reading, removal),
  - the display of the menu,
  - the methods to get a user or a group
  - the method to check if we are logged in
  - the method to log out

- **index.js:**
  The script performs the display of the index.html page

- **signup.js:**
  The script performs: - the display of the signup.html page
  - the method to sign up

- **login.js:**
  The script performs: - the display of the login.html page
  - the method to log in

- **profile.js :**
  The script performs: - the display of the profile.html page
  - the methods to update/delete the account we are on our profile

- **groups.js :**
  The script performs: - the display of the groups.html page
  - the method to create a groups
  - the method get all the groups

- **groupprofile.js :**
  The script performs: - the display of the groupprofile.html page
  - the methods to update/delete the group if we are the admin
  - the methods to join/leave the group we are watching
  - the method to see/write in the group's board if we are a member
  - the method to get all the members of the current group