Platform for player management in Grails: Web App and REST API

AIT ERRAMI Amine
GAUTHIER Thomas

2018-2019

Part 1: Web Application

1. Access control

a. Users

Users with no specific roles can access to their matches, their messages, and check their profile.

b. Admins

Admins have the same rights than the users but have more complete panels. Only them have the right to create message, users, or matchs.

c. Banned users

When an admin delete a user, he's kept in the database but flagged as banned. Those users have no rights at all. They are redirected to a page informing them they are banned when trying to connect.

d. Not logged-in

Users who didn't sign in have access to nothing but the login form.

2. Required functionalities

a. Users

As said above, admins have the right to create new users and to attribute them a profile picture. They can modify it later or ban the user. Created users have by default the role “ROLE_USER”, no admin can be add.
Basic users don't have access to the list of every users, but they can access to the page showing their informations.

b. Matches

All matches are stored in the database with the winner, the loser, and their scores. It's possible that there is no winner or loser, for example if a game is played alone.
Concerning update and create, after submitting the form and before storing a new match, the controller checks if the form is correctly filled. If not, a message, stored in a flash, is displayed to explain the error.

c. Messages

Admins can create news messages. Not-read messages are displayed differently to the users if they have never opened them. Basic users can only see the messages they sent or received, if they try to read other messages they are automatically redirected to the “display” page, where their related messages are listed.
When creating or updating a message, you can't have author = target nor a null content.

3. Details

a. Taglibs

We used two Taglibs :

- ApplicationPropertiesTagLib :
  - Used to get the path where the images are stored.
  - Called whenever we need to retrieve an image.

- CurrentUserInfoTagLib :
  - Used to get the username of the user connected and its profile image name.
  - Called mainly in the navbarto display the name and the picture of the user.

4. Bonus functionalities

a. Drag’n Drop

To add or update a user image, we implemented a drag'n drop system.
You can find the JavaScript functions in charge of this in */grails-app/assets/javascripts/*dragAndDropImageUpload.js.

b. Cron

We set up a cron which deletes read messages every day at 4:00AM, thanks to the Quartz plugin.
The object fulfilling this role is **MessageJob**, located in the */job/mbds/firstgrails* folder.

Part 2: REST API

1. Overview

a. Specifications and results

b. Manual

c. Specifications and documentation

The REST API documentation is located in the folder “doc” in the root of the repository. This documentation was generated by the tool “swagger”, the entry point of the documentation is the htlm file.

2. Required functionalities

All the required functionalities has been implemented. Find above a the description of those functionalities:

a. Implementation of REST API

Our API concerns the three main entities: User, Match, Message.
For each of those entities, our API allows the usage of the following methods: GET / POST / PUT / DELETE. The API provides more than 5 requests per entity.
HTTP error codes and custom messages. The documentation has been generated with “swagger”.

b. Test procedure

We used postman for testing our API. We created a postman collection and each element of this collection contains the requests, parameters, tests, header parameters.
The test script tests that we get the right response for the right request.

In order to perform a GET operation on entities, you should have a user or admin session. If you want to perform a POST operation on new entities, you should have an admin token.

Before launching the tests, be sure to launch the “login” request in the “authentication” folder which sets the token environment variable. That environment variable is automatically given as a value to the key “X-Auth-Token” in the header of all the requests.

3. Bonus functionalities

We have implemented the spring security bony functionality. In order to perform a GET operation on entities, you should have a user or admin session. If you want to perform a POST operation on new entities, you should have an admin token.