Distributed Systems Programming

A.Y. 2024/25

Laboratory 4

In this laboratory activity, you are invited to practice with WebSockets, which represent a computer communications protocol providing full-duplex communication channels over a single TCP connection. WebSockets are often used when a continuous high-load interaction between a client and server (e.g., a web browser and a web server) is required.

The activity is comprehensive of the following tasks:

- implementation of a WebSocket server application (in node.js) and its integration within the *Film Manager* service developed in Laboratory 1;
- implementation of a WebSocket client application (in javascript) and its integration within the React client you are provided with this assignment.

The tools that are recommended for the development of the solution are:

- *Visual Studio Code* (https://code.visualstudio.com/) for the extension of the *Film Manager* service by implementing a WebSocket server, and for the extension of the React application by implementing a WebSocket client;
- *PostMan* (https://www.postman.com/) for testing the extended version of the *Film Manager* service;
- *DB Browser for SQLite* (https://sqlitebrowser.org/) for the management of the database for the *Film Manager* service;
- a web browser (e.g., *Google Chrome*).

Context of the activity

The *Film Manager* service is extended with a new functionality with respect to the service version developed for Laboratory 1, i.e., it offers the possibility for users to select a public film, among the ones for which they are reviewers, as the one they are currently working on. This film is defined as the *active film* of the user who selected it. In order to provide this additional functionality, the *Film Manager* service exposes a new REST API, only available for authenticated users:

- A user can select a public film as their *active* film, among the films for which they are reviewers. In case a review request for the selected film has not been previously issued to the user themselves, the operation fails. If the user had beforehand selected a different active film, the *active* status condition is removed for that previous film, because there must exist at most one active film for each user.

Both the *Film Manager* service and the React client are extended with the functionality to communicate by using channels based on WebSockets. These channels are used by the server (i.e., the *Film Manager* service) to inform all the clients (e.g., multiple instances of the React client) about the current status of the logged in users and the status of their active films, and by the clients to retrieve these updates. In particular, their communication is organized in the following way:

- When the Websocket connection is established with a new client, the server sends the information about all the currently logged in users and their selected films to this client.
- Whenever a user logs in the *Film Manager* service, the WebSocket server sends a message to all the clients with which a WebSocket channel is still open, in order to inform them that the user is currently logged in the service. In these messages, the server also specifies the film that is currently active for the logged-in user, if any.
- Whenever a logged-in user selects a new film as their active film in the *Film Manager* service, the WebSocket server sends a message to all the clients with which a WebSocket channel is still open, in order to inform them about this status update.
- Whenever a logged-in user logs out from the *Film Manager* service, either because the session has expired (after 5 minutes) or because a new login operation has been performed, the WebSocket server sends a message to all the clients with which a WebSocket channel is still open, in order to inform them that the user is not logged in anymore.

The messages exchanged in the WebSocket communication must be compliant with a JSON Schema that you are provided with.

The GUI of the React client has a page, called *Online*, where the information received through the Websockets communication is displayed as soon as it is received. This page, therefore, must display the following pieces of each information:

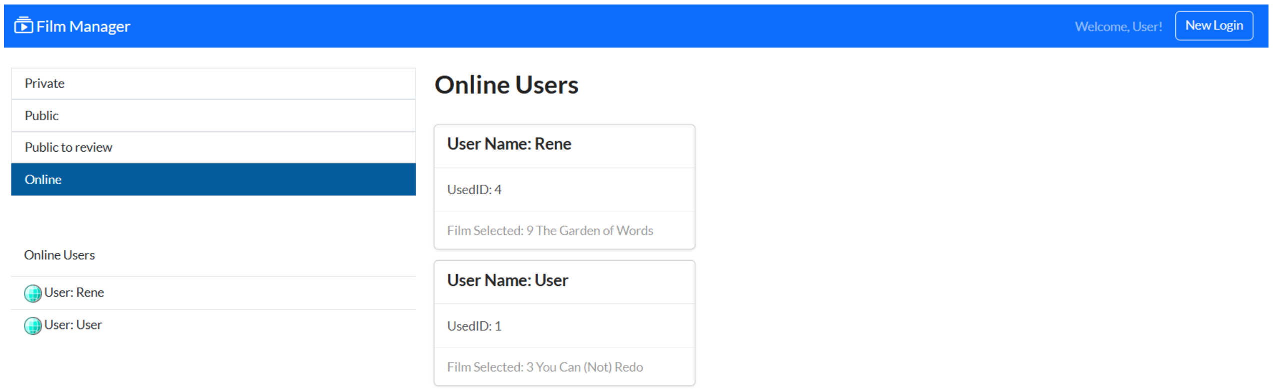
- the list of users who are currently logged-in the *Film Manager* service;
- for each listed user, the user name and the user id;

- for each listed user, the film title and film id of the active film (if any).

The GUI also has a page, called *Films to Review*, listing the public films for which the logged-in user has been invited to write a review.

Besides, all the pages of the GUI display a list of the logged-in users in the left column. Each entry of this list shows the user name for a logged-in user.

The following picture illustrates the *Online* page, showing how the information related to the logged-in users are displayed:



Online Users

User Name: Rene

UsedID: 4

Film Selected: 9 The Garden of Words

User Name: User

UsedID: 1

Film Selected: 3 You Can (Not) Redo

How to experience this laboratory activity

You are invited to complete both the tasks required to fully carry out this laboratory activity (i.e., development of the WebSocket server and client). However, alongside this document, we provide you with:

- an updated database, including the information about the *active* status of each film;
- a JSON Schema, describing the structure of the messages exchanged in the WebSocket communication;
- a React client, which must be extended only with the functionality of WebSocket client, and with the feature of dynamic update of the page where the information received by the WebSocket server is displayed. Instead, all the graphical aspects are already implemented. Be aware that the provided React client is compatible with the implementation of the *Film Manager* service provided as solution of Laboratory 1, and with a possible design solution for the new API that must be defined for film selection. This implies that, if you want to use this client with your own solution, you must adapt it to your own REST API design.

- a slightly modified version of the *Film Manager*. Beware that there is no explicit logout operation performed by the client, thus this version proposes an alternative solution to make the server aware of the automatic logout that needs to be performed for any active session, after the expiration time. This logic has been implemented into the *UserController.js*

Note that also the client program is maintaining a local timer to clear the current session as soon as the login has expired.

Useful tips

Here are some recommendations provided with the aim to help you in completing the tasks required by this Lab session activity:

- You are invited to use the node.js *ws* (https://www.npmjs.com/package/ws) module for the implementation of the WebSocket server. A first reason is that, among all the node.js modules offering this functionality, *ws* is currently the most used one (for reference, see here: https://www.npmtrends.com/express-ws-vs-socket.io-vs-websocket-vs-ws). Besides, *ws* has useful features, such as the possibility to easily retrieve the list of all the connected clients. For the implementation of the WebSocket client, you should instead use the native WebSocket APIs, instead of other node.js modules.
- In the React client, the management of the Websocket communication must be introduced in the *Main()* function of the *App.js* file. In this file, *onlineList* is an array, which retains a local view of the current status of the logged-in users. Each array element contains the most recent Websocket message the client has received about one of the logged-in users. Each message is specified as described in the JSON Schema *ws_message_schema.json*, which you are provided with. Therefore:
    1) when the client opens the Websockets connection, the *onlineList* array is set to the contents of the received message;
    2) when the client is notified that a user has logged in the service, the related message must be added to the *onlineList* array, if not present;
    3) when the client is notified that a user has selected a different film, the message related to that user in the *onlineList* array must be updated;
    4) when the client is notified that a user has logged out of the service, the message related to that user in the *onlineList* array must be removed.
- The REACT client is implemented to interact with the REST APIs designed for the *Film Manager* service that has been published as solution of Lab1. In order to make

it compliant with your own implementation, you should change the code dealing with the REST APIs, i.e., you should mainly modify the API.js file of the REACT client.