Distributed Systems Programming

A.Y. 2020/21

Laboratory 3

In this laboratory activity, you are invited to practice with WebSockets, which represents 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 *ToDoManager* service developed in Laboratory 1;
- implementation of a WebSocket client application (in node.js) 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 *ToDoManager* 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 *ToDoManager* service;
- *DB Browser for SQLite* (https://sqlitebrowser.org/) for the management of the database for the *ToDoManager* service;
- a web browser (e.g., *Google Chrome*).

Context of the activity

The *ToDoManager* 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 task as the one they are currently working on. This task is defined as the *active task* of the user who selected it. The active task of a user must be a task that has been previously assigned to that same user. In order to provide this additional functionality, the *ToDoManager* service exposes a new REST API, only available for authenticated users:

- A user can select a task as their *active* task. In case the selected task has not been previously assigned to the user themselves, the operation fails. If the user had beforehand selected a different active task, the *active* status condition is removed for that previous task, because there must exist at most an active task for each user.

Both the *ToDoManager* 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 *ToDoManager* 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 tasks, 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 tasks to this client.
- Whenever a user logs in the *ToDoManager* 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 task that is currently active for the logged-in user, if any.
- Whenever a logged-in user selects a new task as their active task in the *ToDoManager* 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 *ToDoManager* 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 not logged in anymore.

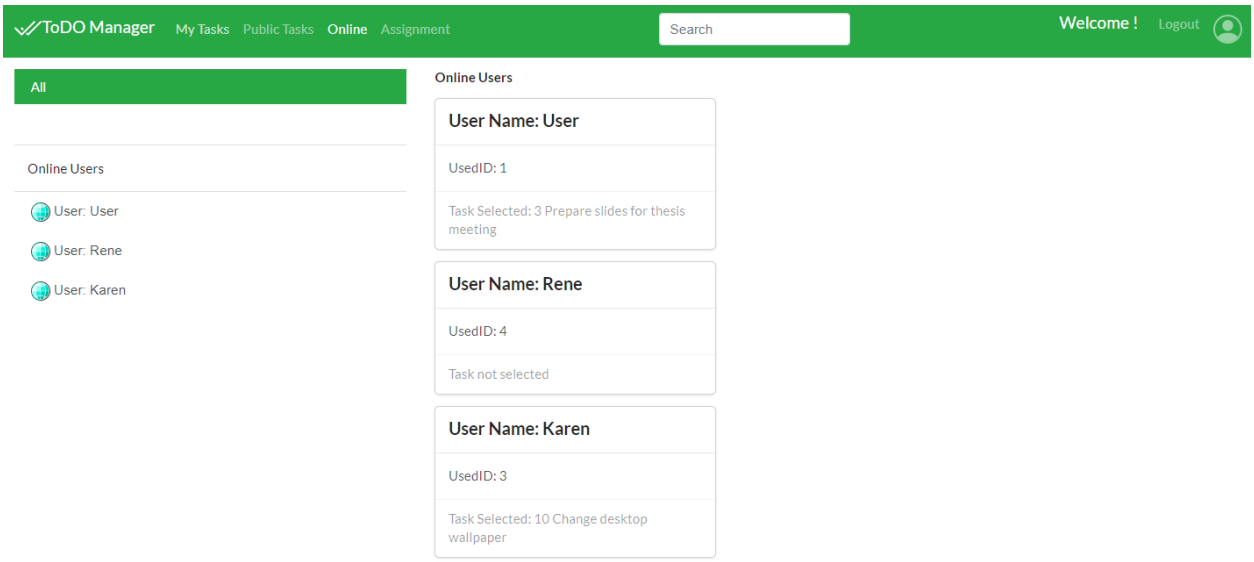
The messages exchanged in the WebSocket communication must be compliant with a JSON Schema that you are provided with.

The GUI of the React client has a page, called *Online*, where the information received throughout 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 that are currently logged-in the *ToDoManager* service;
- for each listed user, the user name and the user id;
- for each listed user, the task name and task id of the active task (if any).

Besides, all the pages (including *My Tasks*, *Public Tasks* and *Assignment*) display a list of the logged-in users in the left column. Each entry of this list shows the user id and 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 :



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 task;
- 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 *ToDoManager* service provided as solution of the first Laboratory activity. This implies that, if you want to use this client with your own solution, you must adapt it to your own REST API design.

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* introduce 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 *componentDidMount()* function of the *App.js* file. In the callbacks you will define for the Websocket communication, you can use the following line of code to update the state related to the logged-in users:
  this.setState({ usersList: this.state.usersList });
  When this function is executed, the content of both the *Online* page and the *Online Users* left column is automatically refreshed.
  In this line of code, *this.state.usersList* is an array, accessible in the *App.js* file, which should contain the most recent message the client has received in the WebSocket communication about each logged-in user. Each message is specified as described in the JSON Schema *ws_message_schema.json*, which you are provided with. Therefore:
  1) when the client is notified that a user has logged in the service, the related message must be introduced in the *this.state.usersList* array, if not present;
  2) when the client is notified that a user has selected a different task, the message related to that user in the *this.state.usersList* array must be updated;
  3) when the client is notified that a user has logged out of the service, the message related to that user in the *this.state.usersList* array must be removed.