Distributed Systems Programming

A.Y. 2020/21

Laboratory 5

In this laboratory activity, you are invited to practice with MQTT, an extremely lightweight publish/subscribe messaging protocol which today is used in a wide variety of applications thanks to an internal reliable message delivery mechanism and support for unreliable networks. For the communication with a web browser, MQTT messages can be transferred over the network and encapsulated into WebSocket frames (MQTT Over Websockets).

In this laboratory, you will also have the opportunity to practice with weak consistency client synchronization mechanisms in web applications.

The activity is comprehensive of the following tasks:

- modification of the *ToDoManager* service developed in Laboratory 3 with the introduction of a publish/subscribe mechanism managed by an MQTT broker for transferring information about task selections;
- adaptation of the React client developed in Laboratory 3 to the new features of the service, including the interaction with the MQTT broker.

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* and of the React application by implementing the required mechanism for a MQTT Over Websockets communication;
- *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*);
- Eclipse Mosquitto (https://mosquitto.org/) for instantiating a message broker that implements the MQTT protocol. Be aware that 2.x versions, released in December 2020, have a bug impacting the MQTT Over Websockets communication for some Operating Systems (e.g., Windows 10). Therefore, the recommended version for this activity is 1.6.12.

Context of the activity

In Laboratory 3, the ToDoManager service was extended with the functionality of task selection, so that each user could select a task, among the assigned tasks, as her *active* task (i.e., the task on which she is currently working on). A new constraint must be now introduced for this operation: a task can be the *active* task for **at most one** user at a time. In case a user tries to select a task which is *active* for another user, the operation fails. Note that this new constraint demands for a synchronization among clients. Here, eventual consistency is acceptable: it is accepted that in some time intervals a client may have a stale copy of task selections, and even that two users can try to select the same task at the same time. In this case, it is required that eventually, in the absence of other operations, all clients will agree about task selections. When a user tries to select a task there are two options (and you can choose which one you prefer): (1) the selection remains in a pending state until a confirmation or refusal of the selection comes from the server (2) the selection appears immediately as active to the user, but if it is later refused by the server, it is undone (optimistic approach). In both cases, the user must be informed about a failed selection with an alert.

Note that this modification of the service may require a modification of its REST API, because the task selection operation may fail in case of conflicts.

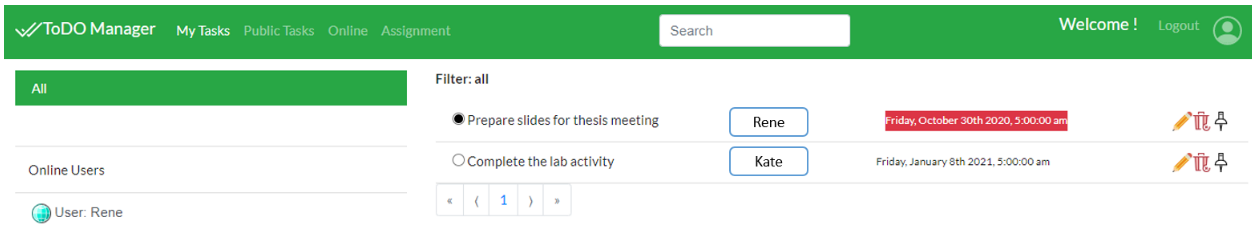
Additionally, the way in which the information about task selections is propagated from server to clients in the ToDoManager service must be modified: instead of using the websockets channels, an MQTT broker (i.e., Eclipse Mosquitto) will be used, using a publish-subscribe mechanism. All clients subscribe to topics associated with tasks (the topic for a task is named with the id of the task). Whenever task selections change, the server publishes the changes to the task topics. These changes are published through messages that convey the status of task selection (i.e., they identify the user for which a task is *active*), and their structure is defined in the JSON schema contained in the *mqtt_task_message_schema.json* file. More precisely, each message conveys the status of the task (*active* if a user has previously selected it, *inactive* otherwise) and, if the task is *active*, the information about the id and name of the user who selected it.

MQTT messages are published by the ToDoManager service in the following cases:

- After the ToDoManager service successfully establishes a connection with the broker, it publishes a message for each existing task. Each message must have the retained flag set to true, so that the broker will send the retained message for a given topic to a client which subscribes to that topic.

- When a task becomes active for a different user, the ToDoManager service publishes a retained message, conveying the *active* status of that task together with the id and name of the user who selected it.
- When a task is not active anymore for any user (e.g., a user has switched its selection from this task to another one), the ToDoManager service publishes a retained message, conveying the *inactive* status of that task.
- When a task is created, the ToDoManager service publishes a retained message, conveying the *inactive* status of that task.
- When a task is deleted, the ToDoManager service publishes a retained message informing the subscribed clients about this event (i.e., with the task status set to *deleted*).

In the React client, the GUI of the *My Tasks* page is extended with a new element in each row for the tasks, as illustrated in the following picture. This element identifies the user who has selected that task as her *active* task, if any.


ToDO Manager
My Tasks
Public Tasks
Online
Assignment
Search
Welcome !
Logout
All
Online Users
User: Rene
Filter: all
Prepare slides for thesis meeting
Rene
Friday, October 30th 2020, 5:00:00 am
Complete the lab activity
Kate
Friday, January 8th 2021, 5:00:00 am
1

The React client receives the required information for displaying this kind of content throughout an MQTT Over Websockets communication with the same MQTT broker (i.e., Eclipse Mosquitto), with which the ToDoManager service itself has established a communication. In particular:

- When a user logs in the ToDoManager service, the React client must subscribe to topics corresponding to the id of each task which is assigned to the user (i.e., each task appearing in the *My Tasks* page of the GUI). After this, the React client will receive a retained message for each task assigned to the user, conveying the *active* status of that task together with the id and name of the user who selected it, if any.
- Whenever the React client receives an MQTT message related to one of the tasks, it updates the status of the task in the *My Tasks* page of the GUI.
- Whenever the selection of a task performed by the logged-in user fails (e.g., the task is *active* for another user), an alert message should be shown on the screen.

Note that the information about logged-in users is still transferred over Websockets, as it was in Laboratory 3.

How to experience this laboratory activity

You are invited to complete both the tasks required to fully carry out this laboratory activity (i.e., modification of the ToDoManager service and React client with the support for MQTT communication). You are also invited to modify the database provided for Laboratory 3 (i.e., databaseV4.db) in order to fulfill the constraints to be applied to the REST API exposed by the ToDoManager service for the task selection.

However, alongside this document, we provide you with:

- the *mosquitto.conf* file, containing the configuration of the Mosquitto MQTT broker required to enable MQTT Over Websockets. To launch Mosquitto with this configuration, you should use the following command:
  mosquito -v -c mosquitto.conf
- a JSON Schema, describing the structure of the messages exchanged in the MQTT communication;
- an extended version of the React client provided as solution for Laboratory 3. This version of the React client must be extended only with the support for the MQTT communication, and with the feature of dynamic update of the page where the information received by the MQTT broker 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 for Laboratory 3. 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 *mqtt* (https://www.npmjs.com/package/mqtt) module for the implementation of the support of the MQTT protocol. In order to enable MQTT Over Websockets, the URL to be specified for the connection to the MQTT broker must be on the "ws" protocol (e.g., "ws://127.0.0.1:8080").
- In the React client, the management of the MQTT communication can be introduced in the *App.js* file. In the callbacks you will define for the MQTT communication, you can use the following line of code to update the state related to the selection of the active tasks by the users of the ToDoManager service:
  this.displayTaskSelection(topic, parsedMessage);
  When this function is executed, the content of the *MyTasks* page is automatically refreshed.
  In this line of code, *topic* is a string representing the id of the task (i.e., it is the topic for which the MQTT message has been received), while *parsedMessage* is the JSON object retrieved after parsing the MQTT message. Each MQTT message is specified as described in the JSON Schema *mqtt_task_message_schema.json*, which you are provided with.

Optional work

In the activity of Laboratory 3, the communication between the ToDoManager service and the React client about the status of the logged-in users has been implemented by using Websockets. Now, as an optional work, you are invited to implement that communication in MQTT Over Websockets (e.g., you may define *online* topics for the users of the system, to which each client instance subscribes, and the ToDoManager service publishes updates).