Distributed Systems Programming

A.Y. 2022/23

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 *Film Manager* service developed in Laboratory 3 with the introduction of a publish/subscribe mechanism managed by an MQTT broker for transferring information about film 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 *Film Manager* 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 *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*);
- Eclipse Mosquitto (https://mosquitto.org/) for instantiating a message broker that implements the MQTT protocol.

Context of the activity

In Laboratory 3, the Film Manager service was extended with the functionality of film selection, so that each user could select a public film, among the films for which a review was issued to that user, as her *active* film (i.e., the film for which she is currently working on writing a review). A new constraint must be now introduced for this operation: a film can be the *active* film for **at most one** user at a time. In case a user tries to select a film that is *active* for another user, the operation fails.

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 film selections, and even that two users can try to select the same film at the same time. In this case, it is required that eventually, in the absence of other operations, all clients will agree about film selections. When a user tries to select a film 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.

This modification of the service may require a modification of its REST API, because the film selection operation may fail in case of conflicts.

Additionally, the information about film selections is propagated from server to clients in the Film Manager using an MQTT broker (i.e., Eclipse Mosquitto), based on a publish-subscribe mechanism. All clients subscribe to topics associated with public films (the topic for a film is named with the id of the film). Whenever film selections change, the server publishes the changes to the film topics. These changes are published through messages that convey the status of film selection (i.e., they identify the user for which a film is *active*), and their structure is defined in the JSON schema contained in the *mqtt_film_message_schema.json* file. More precisely, each message conveys the status of the film (*active* if a user has selected it, *inactive* if the film is currently not selected by any user, *deleted* if the film does no longer exist) and, if the film is *active*, the information about the id and name of the user who selected it.

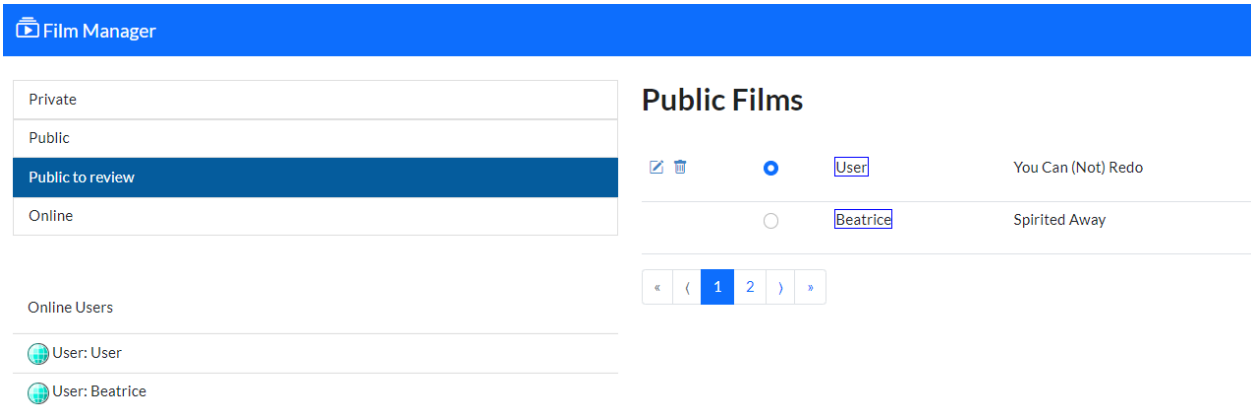
MQTT messages are published by the Film Manager service in the following cases:

- After the Film Manager service successfully establishes a connection with the broker, it publishes a message for each existing public film. 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 film becomes active for a different user, the Film Manager service publishes a new retained message, conveying the *active* status of that film together with the id and name of the user who selected it.
- When a film is not active anymore for any user (e.g., a user has switched its selection from this film to another one), the Film Manager service publishes a new retained message, conveying the *inactive* status of that film.
- When a public film is created, the Film Manager service publishes a new retained message, conveying the *inactive* status of that film.
- When a public film is deleted, the Film Manager service publishes a new retained message informing the subscribed clients about this event (i.e., with the film status set to *deleted*).

In the React client, the GUI of the *Public to Review* page is extended with a new element in each film row, as illustrated in the following picture. This element identifies the user who has selected that film as her *active* film, if any.



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

- The React client must subscribe to topics corresponding to the id of each public film for which a review request has been issued to the logged-in user, and which is

currently shown in the *Public to Review* page of the GUI. After this, the React client will receive a retained message for each film that satisfies these conditions, conveying the *active* status of that film 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 these films, it updates the status of the films in the *Public to Review* page of the GUI.
- Whenever the selection of a film performed by the logged-in user fails (e.g., the film 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 Film Manager service and React client with the support for MQTT communication).

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
- the mqtt_film_message_schema.json 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, which already contain the extended user interface necessary for this Lab. 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 *Film Manager* 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 which implements and supports the MQTT protocol. In order to enable MQTT Over Websockets, the URL to be specified for the connection to the MQTT broker must have the "ws" scheme, identifying the Websocket prtotocol (e.g., "ws://127.0.0.1:8080").
- In the React client, the MQTT communication management 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 films by the users of the Film Manager service:
  displayFilmSelection(topic, parsedMessage);
  When this function is executed, the content of the *Public to Review* page is automatically refreshed.
  In this line of code, *topic* is a string representing the film id (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_film_message_schema.json*, which you are provided with.

Optional work

In the activity of Laboratory 3, the communication between the Film Manager 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 Film Manager service publishes updates).