UNIVERSITAT ROVIRA I VIRGILI, TARRAGONA
Curs 2023/2024

Online chat application

Distributed Systems, Assignment 1 (Version 1)

February 28, 2024

1 Assignment description

In this first task you will work on different **communication patterns** present in distributed systems. For that purpose, you will implement an online chat application on **python**.

You will implement a client and a server. Different clients may share a server to connect the application. The server will have to be up and running so that clients can connect to the application.

The application will provide the following services:

- **Private chats** between two (different) clients.
- **Group chats** between multiple clients.

2 Implementation

2.1 Client

At launch, the client UI shall ask for the client's username. Then, it shows the following options:

1. Connect chat. Connects to a chat (either private of group chat) by specifying its id. The connection must open a dedicated UI with the chat. Once connected, the client listens to the messages in the chats and can write messages to it.

2. Subscribe to group chat. Starts listening to the messages of a group chat, by specifying its id. If the group chat in non-existent, the server must create it from scratch.

3. Discover chats. Asks the server for a list of chats active at the time of the request. Active chats are all group chats and private chats of currently connected clients.

4. Access insult channel. The insult channel is used to send an insult message to a (undefined) client. The receiver must be connected.

2.2 Server

The server will run in a separate process. It will serve as the backend of the chat application. You must run two components from the server: a name server that matches usernames with addresses, and a message broker.

2.2.1 Name Server

The server must provide **a chat namespace in Redis**. For each chat's id, the name server must provide its connection parameters. Chat ids are usernames, for private chats, and group names, for group chats.

- When directly connecting a chat, clients must ask the server for its address to establish the corresponding connection.

- At launch, clients will automatically register their IP address and port into the name server, associated with its username.

2.2.2 Message broker

Group chats, chat discoveries and spam messages should rely on a RabbitMQ broker. You are allowed to run RabbitMQ from a docker image. You should configure a simple persistency layer on your RabbitMQ server for fault-tolerance.

2.3 Private chats

Private chats operate with direct requests between clients. **In this section, you must use gRPC**. Private chats shall implement the following capabilities:

1. Establish a connection between two clients. You have to implement a simple protocol, so that a client can start a connection with another client with known IP address and port. The connection opens a dedicated UI for the conversation in each client.

2. Send and receive messages. All sent messages shall be displayed in both clients' user interfaces (UI).

Clients should coherently manage connection errors.

2.4 Group chats

Group chats use collective, indirect communication between clients. **In this section, you must use RabbitMQ Exchanges (pubsub)**. Group chats shall implement the following capabilities:

1. Subscribe to a group chat. Clients subscribe to a group chat. Once connected, all subscribed clients will receive all messages written to it from the moment of subscription.

2. Connect a group chat. While connected to a group chat, clients receive all the messages from the group, sorted by antiquity. Connecting to a group opens a dedicated UI for the conversation in the connecting client, and enables writing to the group.

3. Send messages to the group chat. Every connected client must be able to send messages to the chat. Messages shall be displayed in the UIs of all clients.

4. Implement both transient and persistent group chats: Demonstrate that persistent group chats enable asynchronous communication between users (not coinciding in the same time).

2.5 Chat discovery

The idea is to replace the Redis repository with an event channel to discover active chats. Compare both approaches (shared memory vs events) for chat discovery. **The chat discovery functionality must be built on a RabbitMQ**.

1. The client publishes a discovery event on the broker. The discovery event must reach all connected clients.
2. Connected clients respond to the event with their connection parameters.
3. The petitioner client shows the responses in its UI.

2.6 Insult Channel

To understand the use of Queues in RabbitMQ we will implement a group chat using a Queue instead of an Exchange. Explain the differences between both approaches.

You have to implement a group where clients can write and receive insults. Each insult will reach a client.

Insults are sent through RabbitMQ Queue. Each message must reach one, and only one, undefined client. Ideally, messages should be distributed equally across all connected clients.

3 Delivery

3.1 Project requirements

- **The system must be launched using only two scripts: `start-server.sh` (to launch the server, must be executed a single time) and `start-client.sh` (to launch a client, can be executed multiple times)**. Every other process and/or terminal must be launched automatically from inside the scripts.

- You are allowed to have configuration parameters (IPs and ports, for instance) hardcoded in a separate `config.yaml` file.
- UIs can be implemented directly in terminals with `ascii` code.
- Your system should permit the communication between computers with known IPs in a shared network.

3.2 Code delivery

You will publish your project in a public GitHub repository of your own. The repository must contain:

- All the code and auxiliary files necessary to run your system.
- A `README.md` with a brief description of the project and the steps to run it.

4 Documentation

You must provide a concise and coherent report for this task, comprising three sections:

1. Abstract. A brief summary (150 words) of your project. Tips for writing an abstract.
2. System design and discussion. Explain the major design decisions of your project and discuss on their appropriateness and limitations. Point out the strengths and/or additional features of the application and justify them.
3. Questions. Provide thoughtful responses to the following questions.
    3.1. **Q1** Are private chats persistent? If not, how could we give them persistency?
    3.2. **Q2** Are there stateful communication patterns in your system?
    3.3. **Q3** What kind of pattern do group chats rely on? In terms of functionality, compare transient and persistent communication in group chats using RabbitMQ.

3.4. **Q4** Redis can also implement Queues and pubsub patterns. Would you prefer to use Redis than RabbitMQ for events ? Could you have transient and persistent group chats? Compare both approaches.

5 Evaluation

- You will have to upload the documentation to the course's Moodle page, before its deadline. The documentation must include a link to the code's GitHub page.
- The task must be implemented and delivered in pairs, and the mark will be the same for both teammates. A group interview will be conducted to evaluate the task.
- You are free to implement the details not described in this document on your own criteria. Such decisions must be justified in the documentation.