IMPLEMENTATION ARCHITECTURE

1. REDIS-SERVER CONTAINER

It exposes Redis-Server on port 6379 in order to guarantee the application persistence functionality, exploiting three relevant caches built on Hash data structure:

- User Cache (Map of <ID, User>): all users currently logged into the application;
- Message Cache (Map of <ID, Message>): all messages written by all the user of the application;
- Audit Cache (Map of <ID, Audit>): all the events registered in the application, which means insertion, deletion or update of a record into User or Message cache.

2. CHAT-MESSAGE CONTAINER

It exposes the application on port 10091 with some endpoints, which are useful for:

- Users, to connect and join the chat;
- Developers/Testers, to debug and troubleshoot the caches with manual HTTP calls (e.g. CURL) or using the Swagger GUI.

The real-time chat service is provided using WebSocket and, in particular, using STOMP on both client and server side.

3. OVERALL WORKFLOW

3.1 USER JOINING

When a client tries to join the chat, the WebSocket connection starts and the client sends to the server:

- a topic-subscription request, which will be directly managed by STOMP Broker which subscribes that client to the requested topic;
- a message to the “chat/newUser” endpoint, which will be processed (validated and persisted) and then forwarded to the chat topic thanks to the STOMP Broker and MessageBroker;

At this point, all the clients subscribed to the desired topic receive the notification that a new user has joined the chat. When a user leaves the chat, a WebSocketListener receives the disconnection event and sends a “user-leaving” message on the chat topic.

3.2 MESSAGE SENDING

When a user sends a message, the “chat/sendMessage” endpoint receives the request, processes it (validating and persisting it) and then, thanks to the STOMP Broker and Message Broker, forwards it to the chat topic. Again, all the users subscribed to that topic will receive the new message.