**Comments App – Documentation – Tudor Campan**

**How to run the application?**

-**Frontend**: The files are located in “src/main/resources/static”. You open the folder in terminal and type “python3 -m http.server”. It should open the applicaction on localhost:8000 (you type this on a chrome page, then you see the app). If not, you should kill the process running on 8000, then try again.

-**Comment Sender service**: start from IDE, runs on localhost:8080.

-**Comment Reciever service**: start from IDE, runs on 8087.

-**Queue**: You subscribe to AMQP which is a RabbitMQ service provider. You create a queue and click on it to see the credential. In both send and receive function(from the sender and receiver), you have to provide your credentials in connection factory. The queue is by default deployed.

You get the credentials from here:

A screenshot of a computer

Description automatically generated

And put them here:

ConnectionFactory connectionFactory = new ConnectionFactory();  
connectionFactory.setUsername("YOUR\_USERNAME");  
connectionFactory.setPassword("YOUR\_PASSWORD");  
connectionFactory.setPort(5672);  
connectionFactory.setUri("YOUR\_URI");

-**Database**: connect to MySql services and type your local credentials in the “application.properties” file.

spring.datasource.url=jdbc:mysql://localhost:3306/YOUR\_DB  
spring.datasource.username=YOUR\_USERNAME  
spring.datasource.password=YOUR\_PASSWORD  
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver  
spring.jpa.hibernate.ddl-auto=update  
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL57Dialect

**How do you use the application?**

It’s pretty straight forward: it displays some comments in a list, and when you want to write a comment, you type it in the text box and submit it. It will be then displayed among the others.

A close-up of a white background

Description automatically generated

**How I developed the application?**

Based on the requirements (both functional and non-functional), I decided to do a service which asynchronously receives a comment via websockets and sends it via RabbitMQ to the service which handles the database operations. The service receives the queue and stores the comment into the database.

Additionally, the front-end initially receives all the comments from the database via a GET request and displays them.

**Which Technologies/Design Patterns I used?**

For the front-end part, I used a **HTML** page (without a framework) in which I wrote **Javascript** for the logics.

For the Back-end services I used **Java+Spring**.

I used **RabbitMQ** for communicating between services.

I used **WebSockets** for asynchronous transmission.

For testing I used Mockito and **Junit**.

I used a **MVC** architecture, trying to respect the **SOLID** principles and to keep coupling low and cohesion high.

**What’s the flow of the program?**

The CommentSender program has a WebSocket connection configured. The frontend connects to the back-end via STOMP when the page is loaded and sends the comment(also via STOMP) to the application.

const ***stompClient*** = new StompJs.Client({  
 brokerURL: 'ws://localhost:8080/gs-guide-websocket'  
});

***stompClient***.publish({  
 destination: "/app/hello",  
 body: ***JSON***.stringify({'name': $("#name").val()})  
});

The CommentSender receives the comment via WebSockets and sends it on the RabbitMQ queue.

private static final QueueSender *queueSender* = new QueueSender();  
@MessageMapping("/hello")  
public void comment(Comment comment) throws Exception {  
 //System.out.println(HtmlUtils.htmlEscape(comment.getText()));  
 *queueSender*.send(comment.getText());  
}

It uses the **QueueSender** class, which has a send method. In the method, the service connects to RabbitMQ and our specific queue, and sends asynchronously the comment on the queue.

Then, the CommentReciever service is listening for the queue ( the listening is done in **Bootstrap** class and it can be activated and desactivated via a Boolean variable).

The listening is done via a **QueueConsumer** class which also connects to the queue and listens for comments.

I used **JPARepository** for the repo. The **CommentService,** using the repository, is storing comments in the Database/retrieving comments from the Database.

The **CommentController** is sending the initial list of comments using a GET request.

**What are the testing strategies used?**

1.**Unit and integration tests with Junit** for the methods in the **CommentReciever** service.

2.Integration test for receiving the coment via WebSockets.

3.**POSTMAN** testing for the GET Request and the WebSockets Connection.

4.**LOGS** to see if :

-the message is sent through websockets;

-the message is received through websockets;

-the message is received via RabbitMQ;

5.**QUEUE MONITORING** to test if the services are communicating.

**What are the possible future improvements?**

1.Do more integration tests with Junit to have 100% coverage.

2.Make the front-end prettier.

3.Do more cleaning on the code ( as a part of the code was from the Web).