Distributed Applications WS2025/26 (Prof. Dr. Cathrin Möller)

Exercise 8 - 11./15.12.2025

Learning goals

Today's exercise demonstrates creating and using a **websocket** endpoint for asynchronous events. We will reuse the existing Spring Boot application and enhance the ProductDetail page with a product review feature.

We want to get hands on working with message based (websocket) communication using the **STOMP** protocol.

Websocket **messages** and events could be triggered by a frontend or a backend. They can also be subscribed from a frontend or another backend application (often used for minimum state changes to be synced between microservices). Sometimes messages received are modified before they are published to all **subscribers**.

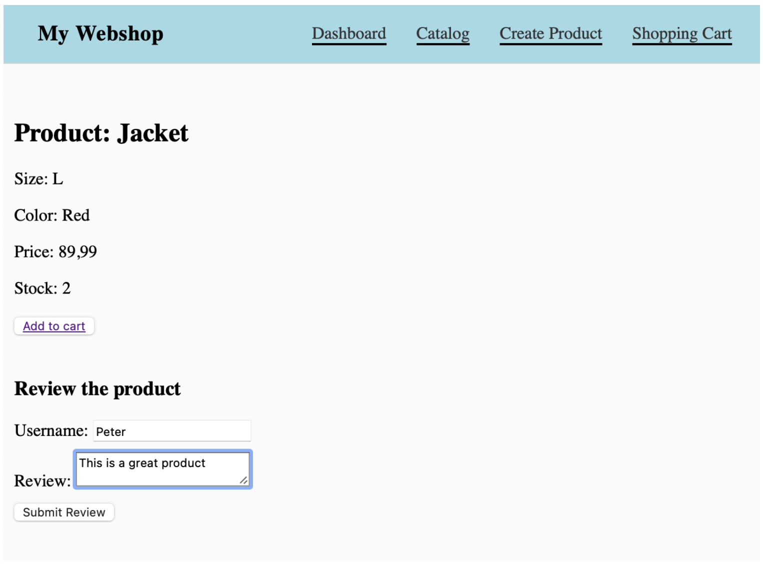
For websocket message subscription and publishing, as for RESTful API requests, correct data format and mapping is crucial. Configuration must match the called endpoints and JSON object attribute names should follow the correct schema and be typo free as well.

Practical Exercises

1) Create a Java class for a ***Review*** model with camel case named fields for a productId (=reference id), a productName (duplicate from product for display purposes), a userName, a reviewText and a date.

2) Add a <section> with a <form> at your ProductDetail page. We need an <input> to enter a username and a <textarea> to enter a review. Add hidden input fields for the productId and the productName. Add a submit button that triggers a POST request for the review.
Hint: Take care that the product id is received and also the product name can be populated. If it was only for the id, you could solve this using path params. But as more values need to be transported, hidden input fields can help as the information was fetched already to display the product detail page.

It might look similar (or prettier) to this:



Product: Jacket

Size: L

Color: Red

Price: 89,99

Stock: 2

Review the product

Username: Peter

Review: This is a great product

3) Create a ***ReviewController*** that listens to the POST request (you can use RequestParams) and creates an instance of ***Review***, including the current date in a format „yyyy-MM-dd HH:mm:ss". You can use the Java time packages for this:

```
import java.time.LocalDateTime;

import java.time.format.DateTimeFormatter;
```

Return the product-detail template view and add the Review to the model. Add a conditional in Thymeleaf to render the „Reviews" on the bottom of the ProductDetail page - only - if present in the model. This way you do not have to populate a default (empty) Review to all controller endpoints serving the same template file.

4) Now we refactor first the backend, not only listening to a POST request but to be able to receive a published Websocket / STOMP message as well.

This way we can stay on the product detail screen and can handle review messages in the background. This approach might also be needed if reviews can be made from different users simultaneously using different servers and we want to simulate synchronized live updates without page reloads.

Install the dependency „**spring-boot-starter-websocket**“.

Create a **WebSocketConfig.java** next to your Controller that configures the message broker and provides the stomp endpoints that external clients can subscribe to connecting to a ws:// URL. It is important that it gets the @Configuration annotation to connect it to the application context of Spring. Otherwise, calling the ws URL would result in a 404.

Use the code snippets from the lecture to override the method **configureMessageBroker** to enable a broker for /topic and adding an application destination prefix /shop.

Override **registerStompEndpoints** adding a stomp registry endpoint

/review-websocket. This defines the callable websocket endpoint to be available from ‚ws://localhost:8080/review-websocket'.

Modify your Review handling Controller (or add a new one) adding a method mapping for incoming messages published to /review and send them to the broker to /topic/reviews. The received **Review** can be modified to set the missing date attribute. You can use the same Java model class for the received and the broadcasted message - as long as we map all fields correctly.

5) Now you need to refactor the template part, adding Javascript inside a <script> in your product detail page. The goal is to send and display the latest reviews without a page reload (imagine reviews could be created by users using different instances of your backend simultaneously).

Check the lecture slides how to inject the scripts for a stomp client, configure it and activate it.

Prevent default form submission and send the review instead using stompClient.publish with destination /shop/review. Be aware you might need to create the JSON body manually if the HTML input fields are differently named than the attributes in your Java Object.

Reset the form after submission using .reset().

Subscribe to /topic/reviews and render all received reviews as child nodes of a placeholder div (using its id).

You can submit the form now more than once and the messages should appear asynchronously.

Troubleshooting: Inspect the network tab in your browser debugging tools to see what happens. Use console.log if parsing in the JS fails.

You can also debug the availability of your Websocket endpoint in the backend using DevTools: const ws = new WebSocket(‚ws://localhost:8080/review-websocket');