Quarkus - Using WebSockets

This guide explains how your Quarkus application can utilize web sockets to create interactive web applications. Because it's the *canonical* web socket application, we are going to create a simple chat application.
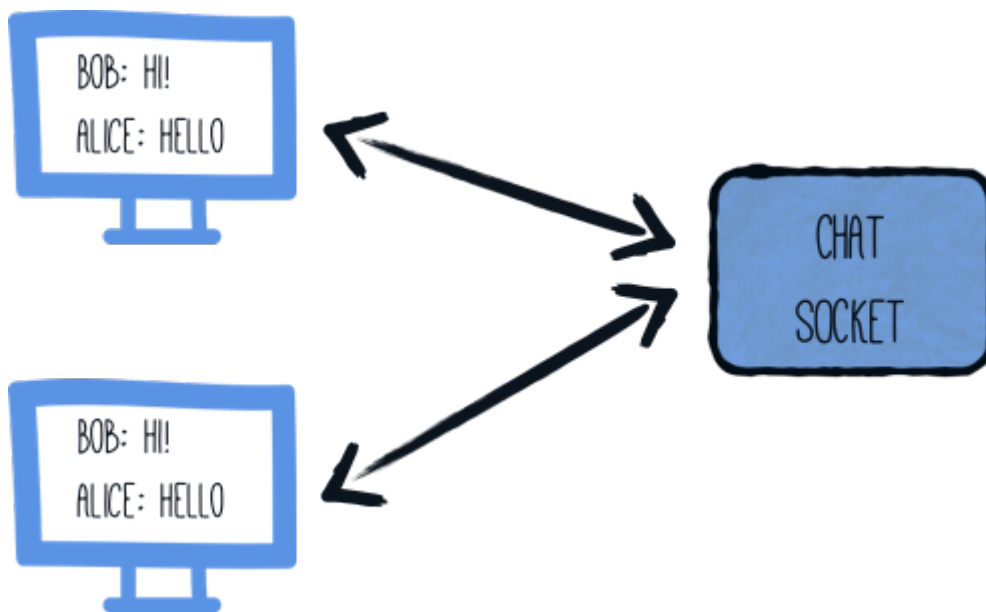
Prerequisites

To complete this guide, you need:

- less than 15 minutes
- an IDE
- JDK 1.8+ installed with `JAVA_HOME` configured appropriately
- Apache Maven 3.6.3

Architecture

In this guide, we create a straightforward chat application using web sockets to receive and send messages to the other connected users.

Solution

We recommend that you follow the instructions in the next sections and create the application step by step. However, you can skip right to the completed example.

Clone the Git repository: `git clone https://github.com/quarkusio/quarkus-quickstarts.git`, or download an archive.

The solution is located in the `websockets-quickstart` directory.

Creating the Maven project

First, we need a new project. Create a new project with the following command:

```
mvn io.quarkus:quarkus-maven-plugin:1.8.0.Final:create \
    -DprojectGroupId=org.acme \
    -DprojectArtifactId=websockets-quickstart \
    -Dextensions="undertow-websockets"
cd websockets-quickstart
```

This command generates the Maven project (without any classes) and import the `undertow-websockets` extension.

If you already have your Quarkus project configured, you can add the `undertow-websockets` extension to your project by running the following command in your project base directory:

```
./mvnw quarkus:add-extension -Dextensions="undertow-websockets"
```

This will add the following to your `pom.xml`:

```
<dependency>
    <groupId>io.quarkus</groupId>
    <artifactId>quarkus-undertow-websockets</artifactId>
</dependency>
```

Handling web sockets

Our application contains a single class that handles the web sockets. Create the `org.acme.websockets.ChatSocket` class in the `src/main/java` directory. Copy the following content into the created file:

```
package org.acme.websockets;

import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;

import javax.enterprise.context.ApplicationScoped;
import javax.websocket.OnClose;
import javax.websocket.OnError;
import javax.websocket.OnMessage;
import javax.websocket.OnOpen;
import javax.websocket.server.PathParam;
```

```java
import javax.websocket.server.ServerEndpoint;
import javax.websocket.Session;

@ServerEndpoint("/chat/{username}")           ①
@ApplicationScoped
public class ChatSocket {

    Map<String, Session> sessions = new ConcurrentHashMap<>(); ②

    @OnOpen
    public void onOpen(Session session, @PathParam("username")
String username) {
        sessions.put(username, session);
        broadcast("User " + username + " joined");
    }

    @OnClose
    public void onClose(Session session, @PathParam("username")
String username) {
        sessions.remove(username);
        broadcast("User " + username + " left");
    }

    @OnError
    public void onError(Session session, @PathParam("username")
String username, Throwable throwable) {
        sessions.remove(username);
        broadcast("User " + username + " left on error: " +
throwable);
    }

    @OnMessage
    public void onMessage(String message, @PathParam("username")
String username) {
        broadcast(">> " + username + ": " + message);
    }

    private void broadcast(String message) {
        sessions.values().forEach(s -> {
            s.getAsyncRemote().sendObject(message, result ->  {
                if (result.getException() != null) {
                    System.out.println("Unable to send message: " +
result.getException());
                }
            });
        });
    }

}
```

1. Configures the web socket URL

2. Stores the currently opened web sockets

A slick web frontend

All chat applications need a *nice* UI, well, this one may not be that nice, but does the work. Quarkus automatically serves static resources contained in the `META-INF/resources` directory. Create the `src/main/resources/META-INF/resources` directory and copy this index.html file in it.
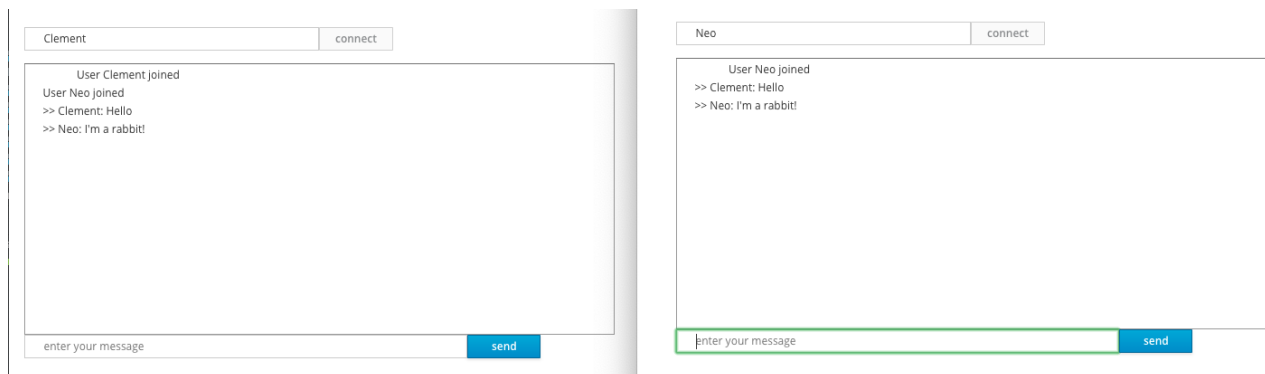
Run the application

Now, let's see our application in action. Run it with:

```
./mvnw compile quarkus:dev
```

Then open your 2 browser windows to http://localhost:8080/:

1. Enter a name in the top text area (use 2 different names).

2. Click on connect

3. Send and receive messages

As usual, the application can be packaged using `./mvnw clean package` and executed using the `-runner.jar` file. You can also build the native executable using `./mvnw package -Pnative`.

You can also test your web socket applications using the approach detailed here.