Programação Guiada 1 - WebSockets Java

Fábio de Azevedo Gomes - Cartão 00287696
Octavio Arruda - Cartão 00246622

October 2020

1 Implementação

Disponível no repositório utilizado.

1.1 Variáveis da classe `ChatServer`

```
@ServerEndpoint("/chat") // Specify displayed path
public class ChatServer
{
    /**
     * List containing all open sessions with
     * the websocket server
     */
    private static List<Session> sessions = new ArrayList
        <Session>();

    /**
     * Application logger
    */
    private static final Logger LOGGER = Logger.getLogger
        (ChatServer.class.getName());
```

Na definição da classe utilizamos a diretiva `@ServerEndpoint` para expor aos clientes o *path* pelo qual podem conectar. A variável `sessions` mantém controle das referências para as instâncias de conexões abertas, enquanto que a variável `LOGGER` é utilizada para o *debug* da aplicação no terminal, apresentando informações sempre que uma nova sessão é aberta ou uma sessão existente é fechada.

1.2 Método `openSession`

```
    /**
     * Receives connections from peers using the
```

```
     * websocket protocol
     * Saves the Session oject into the session set
     * for other methods to access
     * @param new_session The new session being created
     */
    @OnOpen
    public void openSession(Session new_session)
    {
        // Add session into list
        if (sessions.add(new_session))
        {
            // Log the session establishment
            LOGGER.info("New session established: " +
                new_session.getId());
        }
        else
        {
            // Session was already being kept track of
            LOGGER.warning("Session was already being
                kept track of: " + new_session.getId());
        }
    }
```

Este método é executado sempre que uma nova sessão é aberta com o servidor - visível pela diretiva `@OnOpen`, e salva uma referência a esta instância de sessão na lista `sessions` da classe, imprimindo suas informações no terminal. Caso esta sessão já esteja na lista ela não é adicionada, e uma mensagem de *warning* é exibida pelo *logger*.

1.3 Método `closeSession`

```
    /**
     * Called when a websocket session is being closed
     * with the server
     * Removes the Session object from the set, so no
     * more communication can be done with it
     * @param closing_session Session being closed
     */
    @OnClose
    public void closeSession(Session closing_session)
    {
        // Remove session from list
        if (sessions.remove(closing_session))
        {
            // Log the session removal
```

```
            LOGGER.info("Session closed: " +
                closing_session.getId());
        }
        else
        {
            // Session is not being kept track of
            LOGGER.warning("Session is not being kept
                track of: " + closing_session.getId());
        }
    }
```

Método encarregado de gerenciar as sessões que são fechadas com o servidor, de acordo com a diretiva `@OnClose`. Ao receber uma sessão fechante, este método remove-a da lista de sessões, e imprime informações sobre esta sessão no terminal. Caso esta sessão não esteja na lista, ela não é removida (obviamente) e uma mensagem de *warning* é enviada ao terminal.

1.4 Método `transmitMessage`

```
    /**
     * Receives messages from the client and
     * sends them to all connected sessions
     * @param message Message being broadcast
     */
    @OnMessage
    public void transmitMessage(String message) throws
        IOException
    {
        // Iterate through list
        for(Session session : sessions)
        {
            // Send message to each session
            session.getBasicRemote().sendText(message);
        }
    }
```

Responsável por receber as mensagens enviadas pelo cliente, como implicado pela diretiva `@OnMessage`. Itera pelas sessões atualmente abertas e envia a mensagem para elas.

2 Demonstração de execução

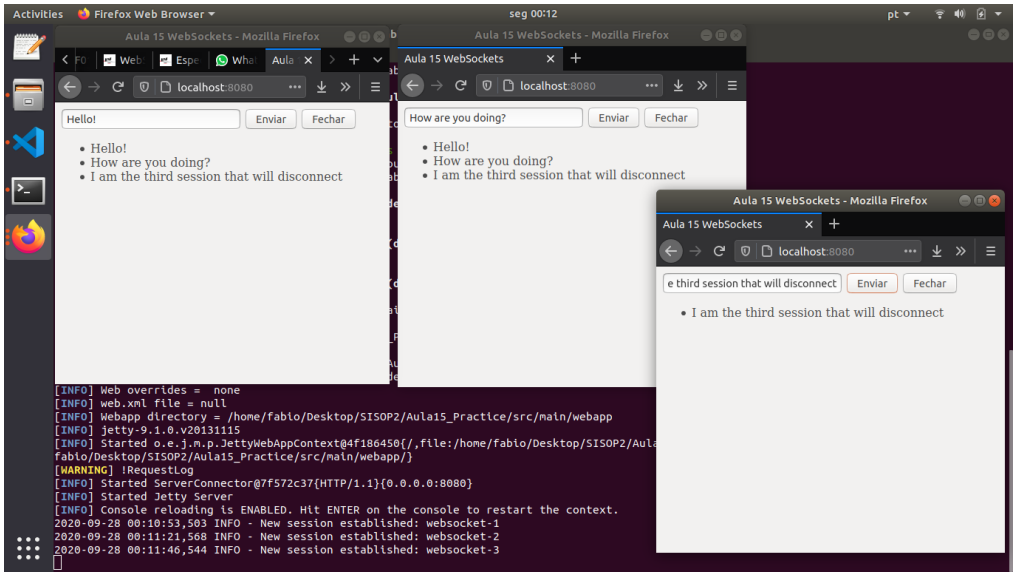

Figure 1: 3 clientes conectados, trocando mensagens

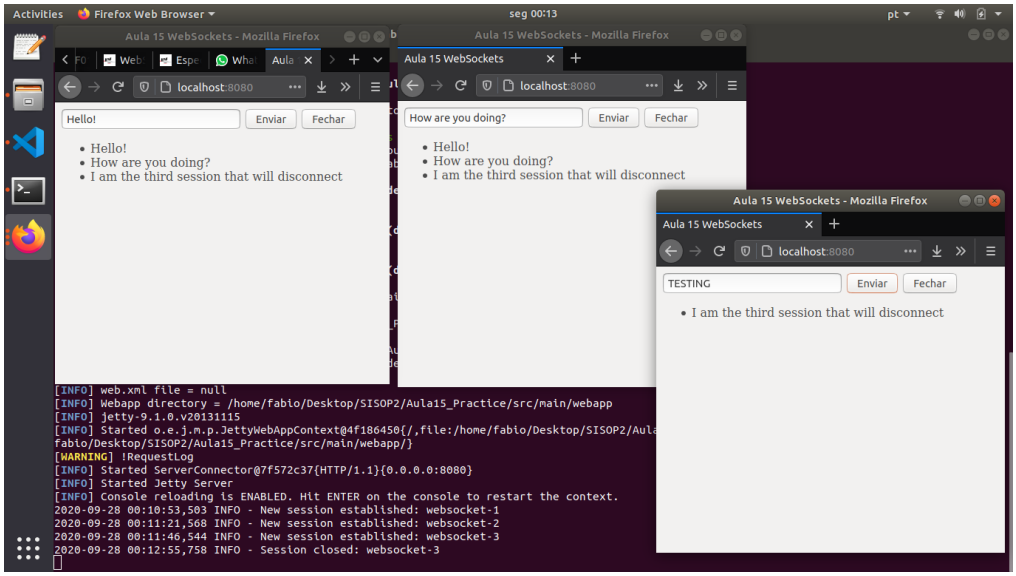

Figure 2: Terceiro cliente clicou em "Fechar"

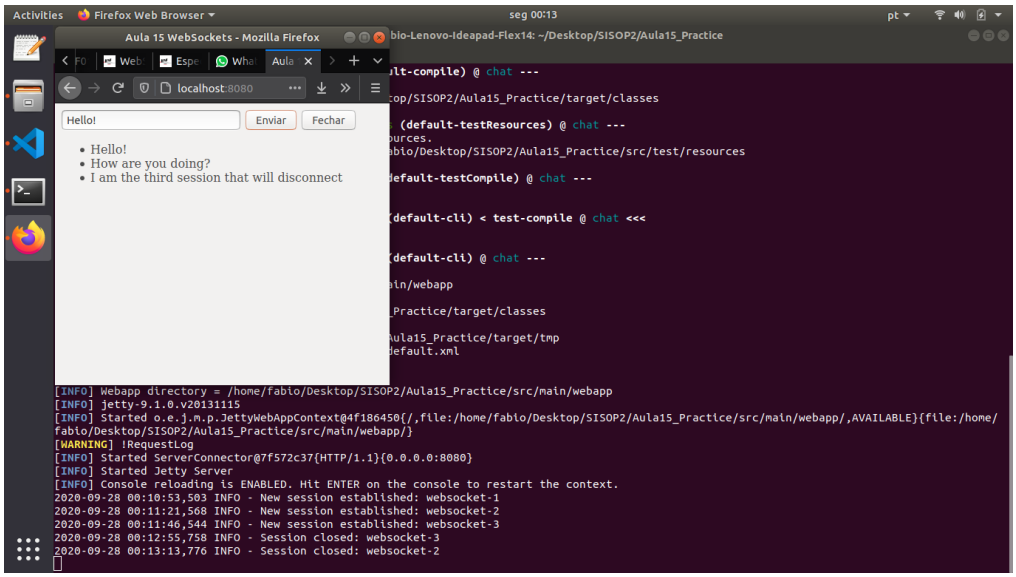

Figure 3: Segundo e terceiro cliente fechados, sobrando apenas o primeiro cliente

3 Observações

3.1 Caminho para o compilador `javac`

Um dos integrantes não pode executar o comando indicado na especificação - `mvn jetty:run` - pois estava obtendo erros na procura pelo compilador, e então foi preciso adicionar ao arquivo `pom.xml`, dentro de `<configuration>`, as linhas

```
<fork>true</fork>
<executable>usr/bin/javac</executable>
```

para que fosse possível realiza a compilação e execução do projeto. Isto provavelmente ocorreu devido à falta ou configuração incorreta da variável de ambiente `$JAVA_HOME`.

3.2 Diretivas `@`

As diretivas `@onClose`, `@onOpen` e `@onMessage` não estavam sendo reconhecidas pelo compilador, e foi preciso entrar as mesmas com **O** maiúsculo para que a aplicação operasse corretamente, e então as diretivas utilizadas foram:

- `@OnClose`
- `@OnOpen`
- `@OnMessage`