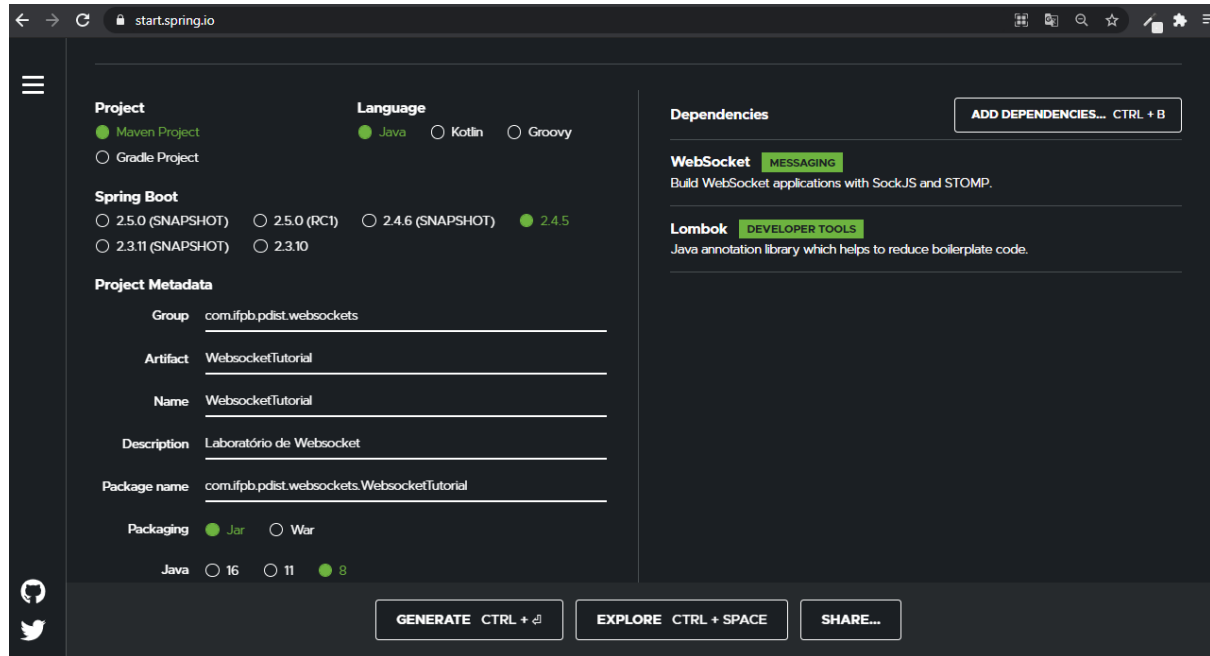


Neste pequeno laboratório nós criaremos um pequeno sistema de Chat para demonstrar um pouco do poder do Websocket.

Passo 1:

Acesso o site <https://start.spring.io/> e crie o seu projeto com as seguintes informações:



The screenshot shows the Spring Start web application interface. The 'Project' section has 'Maven Project' selected. The 'Language' section has 'Java' selected. The 'Spring Boot' section has '2.4.5' selected. The 'Project Metadata' section contains the following fields: Group (com.ifpb.pdist.websockets), Artifact (WebsocketTutorial), Name (WebsocketTutorial), Description (Laboratório de Websocket), and Package name (com.ifpb.pdist.websockets.WebsocketTutorial). The 'Packaging' section has 'Jar' selected. The 'Dependencies' section shows 'WebSocket' (MESSAGING) and 'Lombok' (DEVELOPER TOOLS) as selected dependencies. At the bottom, there are buttons for 'GENERATE CTRL + G', 'EXPLORE CTRL + SPACE', and 'SHARE...'. The URL in the browser is start.spring.io.

Ou use esse [link](#) para facilitar.

Passo 2:

Vamos copiar para dentro da pasta src/main/resources a pasta static que será disponibilizada.

Obs: Fica a seu critério verificar o conteúdo da pasta, nosso foco nesse tutorial é o Spring com Websocket.

Passo 3:

Criaremos uma pasta chamada "web" em src/main/java/com.ifpb.pdist.WebsocketTutorial, dentro desta pasta criaremos uma classe chamada "WebSocketMessageConfig" com o seguinte conteúdo:

```

1 package com.ifpb.pdist.websockets.WebsocketTutorial.web;
2
3 import org.springframework.context.annotation.Configuration;
4 import org.springframework.messaging.simp.config.MessageBrokerRegistry;
5 import org.springframework.web.socket.config.annotation.EnableWebSocketMessageBroker;
6 import org.springframework.web.socket.config.annotation.StompEndpointRegistry;
7 import org.springframework.web.socket.config.annotation.WebSocketMessageBrokerConfigurer;
8
9 @Configuration
10 @EnableWebSocketMessageBroker
11 public class WebSocketMessageConfig implements WebSocketMessageBrokerConfigurer
12 {
13     public void registerStompEndpoints(final StompEndpointRegistry registry) {
14         registry.addEndpoint("/chat-example").withSockJS();
15     }
16
17     public void configureMessageBroker(final MessageBrokerRegistry registry) {
18         registry.setApplicationDestinationPrefixes("/app");
19         registry.enableSimpleBroker("/topic");
20     }
21 }
22

```

Passo 4:

Criaremos uma pasta chamada “model” em src/main/java/com.ifpb.pdist.WebsocketTutorial, dentro criaremos um Enum chamado “MessageType” com o seguinte conteúdo:

```

1 package com.ifpb.pdist.websockets.WebsocketTutorial.model;
2
3 public enum MessageType {
4     CHAT,
5     CONNECT,
6     DISCONNECT
7 }
8

```

Também criaremos a classe “ChatMessage” com o seguinte conteúdo:

```
public class ChatMessage {

    private MessageType type;

    private String content;

    private String sender;

    private String time;

    public MessageType getType() {
        return type;
    }

    public void setType(MessageType type) {
        this.type = type;
    }

    public String getContent() {
        return content;
    }

    public void setContent(String content) {
        this.content = content;
    }

    public String getSender() {
        return sender;
    }

    public void setSender(String sender) {
        this.sender = sender;
    }
}
```

```
    public String getTime() {
        return time;
    }

    public void setTime(String time) {
        this.time = time;
    }

    public ChatMessage() {}

    public ChatMessage(MessageType type, String sender) {
        this.type = type;
        this.sender = sender;
    }
}
```

Passo 5:

Criaremos uma pasta chamada “controller” em src/main/java/com.ifpb.pdist.WebsocketTutorial, dentro desta pasta criaremos um controller chamado “ChatController” com o seguinte conteúdo:

```
ChatController.java X
1 package com.ifpb.pdist.websockets.WebsocketTutorial.controller;
2
3 import org.springframework.messaging.handler.annotation.MessageMapping;
4
5 @Controller
6 public class ChatController {
7     @MessageMapping("/chat.send")
8     @SendTo("/topic/public")
9     public ChatMessage sendMessage(@Payload final ChatMessage chatMessage) {
10         return chatMessage;
11     }
12
13     @MessageMapping("/chat.newUser")
14     @SendTo("/topic/public")
15     public ChatMessage newUser(@Payload final ChatMessage chatMessage,
16                               SimpMessageHeaderAccessor headerAccessor) {
17         headerAccessor.getSessionAttributes().put("username", chatMessage.getSender());
18         return chatMessage;
19     }
20 }
21 }
```

Criaremos também uma classe chamada “WebSocketEventListener” com o seguinte conteúdo:

```
WebSocketEventListener.java X
1 package com.ifpb.pdist.websockets.WebsocketTutorial.controller;
2
3 import org.slf4j.Logger;
4
5 @Component
6 public class WebSocketEventListener {
7     private static final Logger LOGGER = LoggerFactory.getLogger(WebSocketEventListener.class);
8
9     @Autowired
10     private SimpMessageSendingOperations sendingOperations;
11
12     @EventListener
13     public void handleWebSocketConnectListener(final SessionConnectedEvent event) {
14         LOGGER.info("Bing bong. Nós tivemos uma nova conexão!!");
15     }
16
17     @EventListener
18     public void handleWebSocketDisconnectListener(final SessionDisconnectEvent event) {
19         final StompHeaderAccessor headerAccessor = StompHeaderAccessor.wrap(event.getMessage());
20
21         final String username = (String) headerAccessor.getSessionAttributes().get("username");
22
23         final ChatMessage chatMessage = new ChatMessage(MessageType.DISCONNECT, username);
24
25         sendingOperations.convertAndSend("/topic/public", chatMessage);
26     }
27 }
28 }
```

Passo 5:

Iremos até o `application.properties` e colocaremos a propriedade `server.port=8090`.

Passo 6:

Execute a sua aplicação.

Pronto! Você acaba de criar um chat, parabéns.