|  |  |  |
| --- | --- | --- |
|  |  |  |
| МИНОБРНАУКИ РОССИИ | | |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |

Институт Информационных технологий

Кафедра Инструментального и прикладного программного обеспечения

**ПРАКТИЧЕСКАЯ РАБОТА №4**

по дисциплине «Архитектура клиент-серверных приложений»

**Тема практической работы:** Разработка клиент-серверного приложения с использованием технологии Websocket и Spring Framework

**Студент группы** ИКБО-01-19 Кузин Д.О.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись студента)

**Руководитель практической работы** ассистент Китанин С.С.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись руководителя)

Работа представлена «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_ 2021 г.

Допущен к работе «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2021 г.

Москва 2021

# Цель

Знакомство студентов с технологией Java Spring Framework и принципами работы WebSocket.

# Задание на практическую работу

Используя информацию из данной практической работы, необходимо реализовать клиент-серверное приложение с использованием Websocket. Суть приложения заключается в следующем. При обращении клиентской части по адресу /webs необходимо выполнять обработку Websocket. В случае, получения в вебсокете данных, необходимо ответить их же содержимым.

# Ход работы

В ходе работы были создано клиент-серверное приложение с Spring Framework, работающее с WebSocket. При обращении по адресу /webs доступен ввод сообщения, которое отображается у всех клиентов. Содержимое файлов с исходным кодов представлено на листингах 1-6.

Листинг 1 – Файл MVCConfig.java

package com.example.pr4.config;  
  
import org.springframework.context.annotation.Configuration;  
import org.springframework.web.servlet.config.annotation.ViewControllerRegistry;  
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;  
  
@Configuration  
public class MVCConfig implements WebMvcConfigurer {  
 public void addViewControllers(ViewControllerRegistry registry){  
 registry.addViewController("/webs").setViewName("webs/index.html");  
 }  
}

Листинг 2 – Файл WebSocketConfig.java

package com.example.pr4.config;  
  
import org.springframework.context.annotation.Configuration;  
import org.springframework.messaging.simp.config.MessageBrokerRegistry;  
import org.springframework.web.socket.config.annotation.EnableWebSocketMessageBroker;  
import org.springframework.web.socket.config.annotation.StompEndpointRegistry;  
import org.springframework.web.socket.config.annotation.WebSocketMessageBrokerConfigurer;  
  
@Configuration  
@EnableWebSocketMessageBroker  
public class WebSocketConfig implements WebSocketMessageBrokerConfigurer {  
  
 @Override  
 public void configureMessageBroker(MessageBrokerRegistry config) {  
 config.enableSimpleBroker("/topic");  
 config.setApplicationDestinationPrefixes("/app");  
 }  
  
 @Override  
 public void registerStompEndpoints(StompEndpointRegistry registry) {  
 registry.addEndpoint("/chat");  
 registry.addEndpoint("/chat").withSockJS();  
 }  
}

Листинг 3 – Файл MessageController.java

package com.example.pr4.controller;  
  
import com.example.pr4.model.Message;  
import com.example.pr4.model.OutputMessage;  
import org.springframework.messaging.handler.annotation.MessageMapping;  
import org.springframework.messaging.handler.annotation.SendTo;  
import org.springframework.stereotype.Controller;  
  
import java.text.SimpleDateFormat;  
import java.util.Date;  
  
@Controller  
public class MessageController {  
  
 @MessageMapping("/chat")  
 @SendTo("/topic/messages")  
 public OutputMessage send(Message message){  
 String time = new SimpleDateFormat("HH:mm").format(new Date());  
 return new OutputMessage(message.getText(), time);  
 }  
  
}

Листинг 4 – Файл Message.java

package com.example.pr4.model;  
  
public class Message {  
 private String text;  
  
 public String getText() {  
 return text;  
 }  
  
 public void setText(String text) {  
 this.text = text;  
 }  
}

Листинг 5 – Файл OutputMessage.java

package com.example.pr4.model;  
  
public class OutputMessage {  
 private String text;  
 private String time;  
  
 public OutputMessage(String text, String time) {  
 this.text = text;  
 this.time = time;  
 }  
  
 public String getText() {  
 return text;  
 }  
  
 public void setText(String text) {  
 this.text = text;  
 }  
  
 public String getTime() {  
 return time;  
 }  
  
 public void setTime(String time) {  
 this.time = time;  
 }  
}

Листинг 6 – Файл index.html

<!DOCTYPE html>  
<html lang="ru" >  
<head>  
 <meta http-equiv="Content-Type" content="text/html; charset=utf-8">  
 <title>Chat WebSocket</title>  
 <script src="https://cdn.jsdelivr.net/npm/sockjs-client@1/dist/sockjs.min.js"></script>  
 <script src="webs/js/stomp.js"></script>  
 <script type="text/javascript">  
 var stompClient = null;  
  
 function *setConnected*(connected) {  
 document.getElementById('connect').disabled = connected;  
 document.getElementById('disconnect').disabled = !connected;  
 document.getElementById('conversationDiv').style.visibility  
 = connected ? 'visible' : 'hidden';  
 document.getElementById('response').innerHTML = '';  
 }  
  
 function *connect*() {  
 var socket = new SockJS('/chat');  
 stompClient = Stomp.over(socket);  
 stompClient.connect({}, function(frame) {  
 *setConnected*(true);  
 console.log('Connected: ' + frame);  
 stompClient.subscribe('/topic/messages', function(messageOutput) {  
 *showMessageOutput*(JSON.parse(messageOutput.body));  
 });  
 });  
 }  
  
 function *disconnect*() {  
 if(stompClient != null) {  
 stompClient.disconnect();  
 }  
 *setConnected*(false);  
 console.log("Disconnected");  
 }  
  
 function *sendMessage*() {  
 var text = document.getElementById('text').value;  
 stompClient.send("/app/chat", {},  
 JSON.stringify({'text':text}));  
 let field = document.getElementById('text')  
 field.value = ""  
 }  
  
 function *showMessageOutput*(messageOutput) {  
 var response = document.getElementById('response');  
 var p = document.createElement('p');  
 p.style.wordWrap = 'break-word';  
 p.appendChild(document.createTextNode(messageOutput.text + " (" + messageOutput.time + ")"));  
 response.appendChild(p);  
 }  
 </script>  
</head>  
<body onload="*disconnect*()">  
<div>  
 <div>  
 <button id="connect" onclick="*connect*();">Подключиться</button>  
 <button id="disconnect" disabled="disabled" onclick="*disconnect*();">  
 Отключиться  
 </button>  
 </div>  
 <br />  
 <div id="conversationDiv">  
 <input type="text" id="text" placeholder="Введите текст сообщения..."/>  
 <button id="sendMessage" onclick="*sendMessage*();">Отправить</button>  
 <p id="response"></p>  
 </div>  
</div>  
  
</body>  
</html>

Результаты работы программы представлены на рисунке 1.

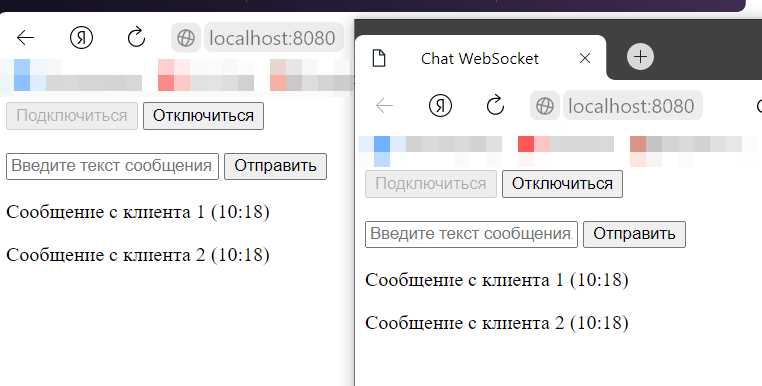


Рисунок – Результат работы программы со стороны первого клиента

# 

# Вывод

В результате работы были получены навыки по работе с Java Spring Framework и WebSocket на примере веб-приложения с отправкой сообщений.