|  |
| --- |
|  |
| МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ |
| Федеральное государственное бюджетное образовательное учреждение высшего образования  **"МИРЭА - Российский технологический университет"**  **РТУ МИРЭА** |
| Институт информационных технологий (ИТ) |
| Кафедра инструментального и прикладного программного обеспечения |

|  |  |
| --- | --- |
|  | |
| **ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ №4** | |
| **по дисциплине** | |
| **«Архитектура клиент-серверных приложений»**  **Тема: «**Разработка клиент-серверного приложения с использованием технологии Websocket и Spring Framework.» | |
| Выполнил студент группы **ИКБО-20-21** | Сидоров С.Д. |
| Принял преподаватель кафедры ИиППО | Волков М.Ю. |

|  |  |  |
| --- | --- | --- |
| Практические работы выполнены | « » 2023г. |  |
| «Зачтено» | « » 2023г. |  |

Москва 2023

**Теоретическое введение**

Websocket позволяет создать канал связи между клиентом и сервером. В частности, канал связи, который использует протокол WebSocket в качестве протокола связи. Протокол WebSocket совместим с протоколом HTTP. который также работает через TCP / IP . Однако он имеет исключительные улучшения, в части меньших накладных расходов, чем HTTP, и двунаправленную веб-связь. Таким образом, они в первую очередь предназначены для веб-приложений, которым требуется постоянное соединение с сервером. Связь WebSocket может использоваться **между любыми типами приложений** , но чаще всего WebSocket используется для облегчения связи между серверным приложением и приложением клиентом на основе браузера.

На рисунке 1 показано различие между «традиционным» способом связи с использованием HTTP и связи с использованием WebSockets.

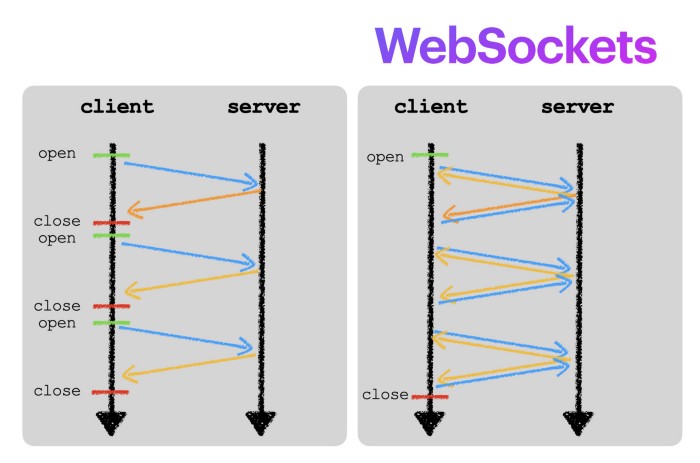


Рисунок 1 – Различие между «традиционным» (HTTP-опрос) способом связи с использованием HTTP и связи с использованием WebSockets

Традиционное веб-общение с использованием протокола HTTP работает следующим образом:

* во-первых, клиент (обычно веб-браузер) должен подключиться к серверу;
* затем клиент отправляет запрос ресурса (например, запрос веб-страницы);
* после этого сервер отвечает;
* закрывает канал связи.

С другой стороны, WebSocket работает следующим образом:

* держит соединение открытым;
* кроме того и клиент, и сервер могут делать запросы и отправлять ответы.

Протокол WebSocket призван заменить существующие обходные механизмы HTTP и предоставить эффективный протокол для одновременной двунаправленной связи с малой задержкой между браузерами и серверами по одному TCP-соединению.  HTTP изначально был разработан для передачи ресурсов типа "запрос-ответ" в распределенных гипермедийных системах, но не для одновременной двунаправленной связи. Для преодоления этих архитектурных ограничений используются несколько HTTP-механизмов (сгруппированных под неофициальным названием Comet ), которые часто бывают сложными и неэффективными.

**Постановка задачи**

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.

**Программный код**

Листинг 1 – WebSocketConfig.java

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("/webs").withSockJS();  
 }  
}

Листинг 2 – WebsController.java

import org.springframework.messaging.handler.annotation.MessageMapping;  
import org.springframework.messaging.handler.annotation.SendTo;  
import org.springframework.stereotype.Controller;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.ResponseBody;  
import org.springframework.web.servlet.ModelAndView;  
import java.text.SimpleDateFormat;  
import java.util.Date;  
import java.util.concurrent.atomic.AtomicInteger;  
  
@Controller  
@RequestMapping("/")  
public class WebsController {  
  
 private final AtomicInteger userId = new AtomicInteger(0);  
  
 @GetMapping("/")  
 @ResponseBody  
 public String index() {  
 return "Для подписки на топик перейдите по эндпоинту /webs";  
 }  
  
 @GetMapping("/webs")  
 public ModelAndView webs() {  
 ModelAndView modelAndView = new ModelAndView();  
 modelAndView.setViewName("webs.html");  
 return modelAndView;  
 }  
  
 @MessageMapping("/webs")  
 @SendTo("/topic/messages")  
 public OutputMessage send(Message message) {  
 String time = new SimpleDateFormat("HH:mm").format(new Date());  
 return new OutputMessage(message.getFrom(), message.getText(), time);  
 }  
}

Листинг 3 – Message.java

public class Message {  
  
 private String from;  
 private String text;  
  
 public String getFrom() {  
 return from;  
 }  
  
 public void setFrom(String from) {  
 this.from = from;  
 }  
  
 public String getText() {  
 return text;  
 }  
  
 public void setText(String text) {  
 this.text = text;  
 }  
}

Листинг 4 – OutputMessage.java

public class OutputMessage {  
  
 private final String from;  
 private final String text;  
 private final String time;  
  
 public OutputMessage(String from, String text, String time) {  
 this.from = from;  
 this.text = text;  
 this.time = time;  
 }  
  
 public String getFrom() {  
 return from;  
 }  
  
 public String getText() {  
 return text;  
 }  
  
 public String getTime() {  
 return time;  
 }  
}

Листинг 5 – Main.java

import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
  
@SpringBootApplication  
public class Pr4Application {  
  
 public static void main(String[] args) {  
 SpringApplication.*run*(Pr4Application.class, args);  
 }  
}

Листинг 6 – webs.html

<!DOCTYPE html>  
<html lang="ru">  
<head>  
 <title>Chat WebSocket</title>  
 <script src="https://cdnjs.cloudflare.com/ajax/libs/sockjs-client/1.4.0/sockjs.min.js"></script>  
 <script src="https://cdnjs.cloudflare.com/ajax/libs/stomp.js/2.3.3/stomp.min.js"></script>  
  
  
 <script>  
 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('/webs');  
 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 from = document.getElementById('from').value;  
 var text = document.getElementById('text').value;  
  
 stompClient.send("/app/webs", {},  
 JSON.stringify({'from': from, 'text': text}));  
 }  
  
 function showMessageOutput(messageOutput) {  
 let response = document.getElementById('response');  
 let p = document.createElement('p');  
 p.style.wordWrap = 'break-word';  
 p.appendChild(document.createTextNode(  
 messageOutput.from + ": "  
 + messageOutput.text + " ("  
 + messageOutput.time + ")")  
 );  
 response.appendChild(p);}

Продолжение листинга 6

</script>  
</head>  
<body onload="disconnect(); connect();">  
<div>  
 <div>  
 <input type="text" id="from" placeholder="Измените никнейм"/>  
 </div>  
 <br/>  
 <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>

**Вывод программы**

Чтобы начать работу приложения необходимо сначала запустить сервер (рисунок 2), а клиентам – открыть вебсокет и подписаться на топик, перейдя по эндпоинту /webs (рисунок 3), и произвести рассылку сообщений (рисунjr 4).

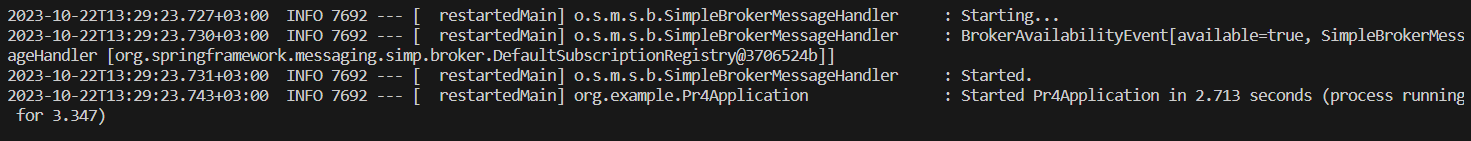


Рисунок 2 – Запуск сервера и структура проекта

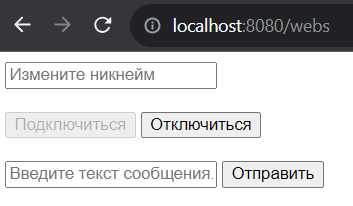


Рисунок 3 – Работа приложения

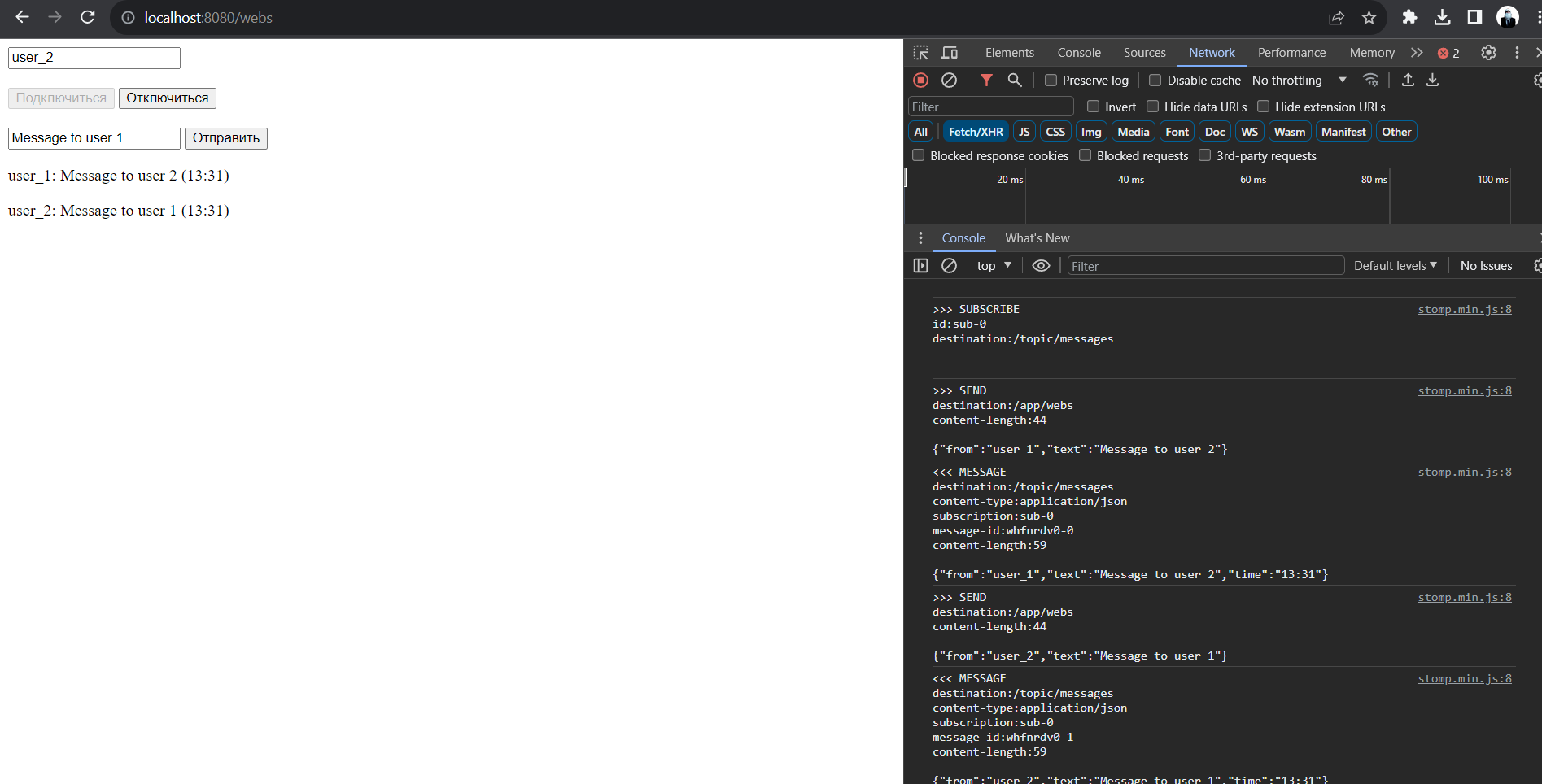


Рисунок 4 – Работа приложения

**Вывод**

В результате выполнения данной практической работы было проведено ознакомление с основным инструментом для организации полнодуплексного соединения – WebSocket и реализовано клиент-серверное приложение с его использованием.

**Список использованных источников**

1. Справочная информация Habr – URL: https://habr.com/ru/companies/ruvds/articles/414079/ (дата обращения: 16.10.2023) – текст: электронный.
2. Статья Baeldung – URL: https://www.baeldung.com/java-call-graphql-service (дата обращения: 17.10.2023) – текст: электронный.
3. Справочная информация Habr – URL: https://habr.com/ru/articles/513170/ (дата обращения: 20.10.2023) – текст: электронный.