|  |  |  |
| --- | --- | --- |
|  |  |  |
| 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 №5**

по дисциплине «Архитектура клиент-серверных приложений»

**Тема практической работы: Введение в многослойные клиент-серверные архитектуры**

**Студент группы** ИКБО-10-19 Магин Константин Андреевич

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

(подпись студента)

**Руководитель практической работы** преподаватель Степанов В. С.

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

(подпись руководителя)

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

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

Москва 2021

**Цель:**

Ознакомить с многослойными клиент-серверными архитектурами, посмотреть разницу между ними, выявить плюсы и минусы.

**Задание:**

Поскольку для трёхуровневой архитектуры необходимо физическое разделение подсистем, то предлагается разработать трёхслойное приложение: БД, сервер, приложение. В качестве приложения можно использовать: запросы в postman/insomnia/testmace, простой сайт, десктопное приложение, мобильное приложение. В качестве БД можно использовать: SQLite, PostgreSQL.

Нечетные варианты реализуют синхронное взаимодействие через API. Чётные варианты реализуют несохранное асинхронное взаимодействие через WebSockets.

**Ход работы:**

В ходе выполнения практической работы были созданы файлы для клиента и сервера, реализована логика работы программы, реализовано сохранение сообщений в бд.

**Исходный код программы:**

WebSocketConfig.java

package com.web.pr4.websocket;  
  
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");  
 registry.addEndpoint("/webs").withSockJS();  
 }  
}

Sender.java

package com.web.pr5.websocket;  
  
import org.springframework.beans.factory.annotation.Autowired;  
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 Sender {  
  
 private dbMessageQuery dbMessageQuery;  
  
 @Autowired  
 public Sender(dbMessageQuery dbMessageQuery) {  
 this.dbMessageQuery = dbMessageQuery;  
 }  
  
 @MessageMapping("/webs")  
 @SendTo("/topic/messages")  
 public OutputMessage send(Message message) throws Exception {  
 String time = new SimpleDateFormat("HH:mm").format(new Date());  
 System.*out*.println(time);  
 dbMessage dbMessage = new dbMessage();  
 dbMessage.setFrom(message.getFrom());  
 dbMessage.setText(message.getText());  
 dbMessage.setDate(time);  
 dbMessageQuery.save(dbMessage);  
 return new OutputMessage(message.getFrom(),message.getText(), time);  
 }  
}

OutputMessage.java

package com.web.pr4.websocket;  
  
import lombok.Getter;  
import lombok.Setter;  
  
@Getter  
@Setter  
public class OutputMessage {  
 private String from;  
 private String text;  
 private String time;  
  
 public OutputMessage(String from, String text, String time){  
 this.from = from;  
 this.text = text;  
 this.time = time;  
 }  
}

Message.java

package com.web.pr4.websocket;  
  
import lombok.Getter;  
import lombok.Setter;  
  
@Getter  
@Setter  
public class Message {  
 private String from;  
 private String text;  
  
 public Message(String from, String text){  
 this.from = from;  
 this.text = text;  
 }  
}

dbMessage.java

package com.web.pr5.websocket;  
  
import lombok.Getter;  
import lombok.Setter;  
  
import javax.persistence.\*;  
  
@Getter  
@Setter  
@Entity  
@Table(name = "chat\_messages")  
public class dbMessage {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*AUTO*)  
 private Long id;  
  
 @Column(name = "from\_user")  
 private String from;  
  
 @Column(name = "text\_of\_message")  
 private String text;  
  
 @Column(name = "date\_of\_message")  
 private String date;  
}

dbMessageQuery.java

package com.web.pr5.websocket;  
  
import org.springframework.data.jpa.repository.JpaRepository;  
  
public interface dbMessageQuery extends JpaRepository<dbMessage, Long> {  
}

index.html

<html>  
 <head>  
 <title>Chat WebSocket</title>  
 <script src="webjars/jquery/jquery.min.js"></script>  
 <script src="webjars/sockjs-client/sockjs.min.js"></script>  
 <script src="webjars/stomp-websocket/stomp.min.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('connectionDiv').style.visibility  
 = connected ? 'hidden' : 'visible';  
 ***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) {  
 var response = ***document***.getElementById('response');  
 var p = ***document***.createElement('p');  
 p.style.wordWrap = 'break-word';  
 p.appendChild(***document***.createTextNode(messageOutput.from + ": "   
 + messageOutput.text + " (" + messageOutput.time + ")"));  
 response.appendChild(p);  
 }  
 </script>  
 </head>  
 <body onload="disconnect()">  
 <div>  
 <div id="connectionDiv">  
 <div>  
 <input type="text" id="from" placeholder="Your nickname"/>  
 </div>  
 </div>  
 <br>  
 <div>  
 <button id="connect" onclick="connect();">Connect</button>  
 <button id="disconnect" disabled="disabled" onclick="disconnect();">  
 Disconnect  
 </button>  
 </div>  
 <br>  
 <div id="conversationDiv">  
 <input type="text" id="text" placeholder="Insert message..."/>  
 <button id="sendMessage" onclick="sendMessage();">Send</button>  
 <p id="response"></p>  
 </div>  
 </div>  
  
 </body>  
</html>

**Вывод программы:**

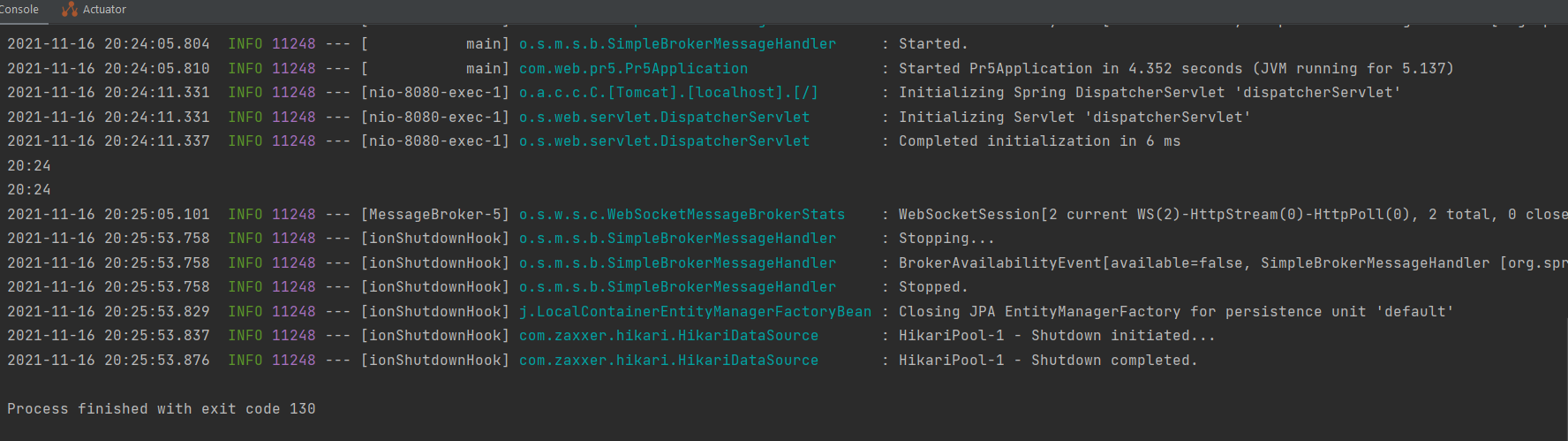


Рисунок 1 – запуск сервера и его работа.



Рисунок 2 – Работа клиента

Работа базы данных:

mysql> select \* from chat\_messages

-> ;

+----+-----------------+-----------+-----------------+

| id | date\_of\_message | from\_user | text\_of\_message |

+----+-----------------+-----------+-----------------+

| 8 | 20:24 | Jim | Hello Dave! |

| 9 | 20:24 | Dave | Hello Jim! |

+----+-----------------+-----------+-----------------+

2 rows in set (0.00 sec)

**Вывод:**

В результате выполнения практической работы были получены знания и навыки разработки приложения на основе 3х уровневой архитектуры с использованием технологии Websocket, Spring JPA и Spring Framework.