|  |
| --- |
|  |
| 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 № 5** | |
| **по дисциплине** |  |
| **«Архитектура клиент-серверных приложений»**  **Тема: Введение в многослойные клиент-серверные архитектуры** | |
| Выполнил студент группы ИКБО-20-22 | Шумахер М.Е. |
| Принял преподаватель кафедры ИиППО | Волков М.Ю. |

|  |  |  |
| --- | --- | --- |
| Практические работы выполнены | «04» сентября 2024 г. |  |
| «Зачтено» | «04» сентября 2024 г. |  |

Москва 2024

**Цель работы:** знакомство с многослойными клиент-серверными архитектурами, посмотреть разницу между ними, выявить плюсы и минусы.

**Теоретическое введение:**

В предыдущих практических работах мы неявно познакомились с одной из архитектур распределенных систем: клиент-серверная архитектура.

‎Тонкие клиенты‎. Тонкий клиент спроектирован так, чтобы основная часть обработки данных происходила на сервере. Тонкий клиент как правило без жесткого диска: действуют как простой терминал к серверу и требует постоянной связи с сервером. ‎

‎‎Толстые клиенты‎. Толстый клиент выполняет основную часть обработки. У толстых клиентов нет необходимости в непрерывной связи с сервером, поскольку они в основном передают информацию на сервер.

Когда какой клиент использовать‎. Тонкие клиенты обеспечивают работу рабочего стола в средах, где конечный пользователь имеет четко определенное и регулярное количество задач, для которых используется система. Тонких клиентов можно найти в медицинских офисах, авиабилетах, школах, правительствах, производственных предприятиях и даже колл-центрах. Наряду с простотой установки, тонкие клиенты также предлагают более низкую общую стоимость владения по сравнению с толстыми клиентами.

Если вашим приложениям требуются мультимедийные компоненты или которые интенсивно используют пропускную способность, стоит присмотреться к разработке толстых клиентов. Одно из самых больших преимуществ толстых клиентов – некоторые операционные системы и программное обеспечение не могут работать на тонких клиентах. Толстые клиенты могут справиться с ними, поскольку у них есть свои собственные ресурсы.

Разница между «уровнем» и «слоем» в контексте распределённых систем

«Слой» и «уровень» относятся к функциональной части программного обеспечения, но «уровень» относится к ПО, которое работает в инфраструктуре отдельных частей приложения (на разных физических системах). Например, приложение "Контакты" на телефоне является трехслойным приложением, но одноуровневым приложением, потому что все три слоя работают на одном физическом устройстве – на телефоне. Так, «слои» не могут предложить тех преимуществ, которыми обладают «уровни».

Трёхуровневая архитектура распределённых систем. Трёхуровневая архитектура организует приложения в три логических и физических вычислительных уровня: уровень представления (пользовательский интерфейс), уровень приложений (обработка данных происходит здесь) и уровень данных (хранение и управление данными). Поскольку каждый уровень работает в своей собственной инфраструктуре, он может разрабатываться одновременно отдельной командой разработчиков и может обновляться или масштабироваться по мере необходимости, не влияя на другие уровни, не говоря уже о реплицировании.

В трехуровневом приложении вся связь проходит через уровень приложения. Уровень представления и уровень данных не могут напрямую взаимодействовать друг с другом.

**Постановка задачи:** 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.

**Программный код:**

Message.java

package main.prac5;  
import jakarta.persistence.\*;  
import lombok.\*;  
  
@Entity  
@NoArgsConstructor  
@Getter  
@Setter  
@Table(name = "message")  
public class Message {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.*AUTO*)  
  
 private int id;  
 private String text;  
}

MessageController.java

package main.prac5;  
  
  
import lombok.RequiredArgsConstructor;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.PostMapping;  
import org.springframework.web.bind.annotation.RequestBody;  
import org.springframework.web.bind.annotation.RestController;  
  
import java.util.List;  
  
@RestController  
@RequiredArgsConstructor  
public class MessageController {  
 private final MessageService messageService;  
  
 @GetMapping("/findAll")  
 public List<Message> getMessage() {  
 return messageService.findAll();  
 }  
  
 @PostMapping("/save")  
 public boolean save(@RequestBody Message message) {  
 return messageService.save(message);  
 }  
}

MessageRepo.html

package main.prac5;  
  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.stereotype.Repository;  
  
@Repository  
public interface MessageRepo extends JpaRepository<Message, Long> {  
}

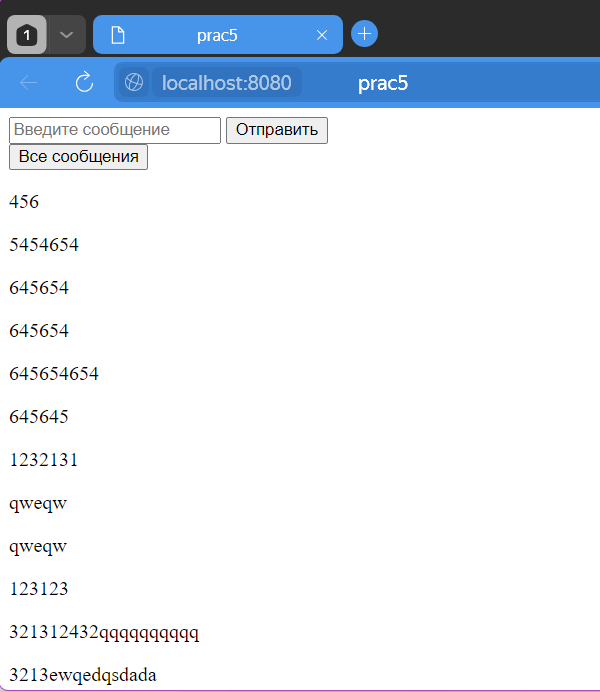
MessageService.html

package main.prac5;  
import lombok.RequiredArgsConstructor;  
import org.springframework.stereotype.Service;  
  
import java.util.List;  
  
@Service  
@RequiredArgsConstructor  
public class MessageService {  
 private final MessageRepo messageRepo;  
  
 public boolean save(Message message) {  
 messageRepo.save(message);  
 return true;  
 }  
  
 public List<Message> findAll() {  
 return messageRepo.findAll();  
 }  
}

index.html

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1.0">  
 <title>praс5</title>  
</head>  
<body>  
<div>  
 <input type="text" id="messageInput" placeholder="Введите сообщение">  
 <button onclick="sendMessage()">Отправить</button>  
</div>  
  
<div>  
 <button onclick="getAllMessages()">Все сообщения</button>  
</div>  
  
<div id="messageList"></div>  
  
<script>  
 function sendMessage() {  
 const message = *document*.getElementById("messageInput").value;  
 const xhr = new XMLHttpRequest();  
 xhr.open("POST", "http://localhost:8080/save", false);  
  
 xhr.setRequestHeader("Content-Type", "application/json");  
  
 xhr.send(*JSON*.stringify({ text: message }));  
  
 if (xhr.status != 200) {  
 alert(xhr.status);  
 } else {  
 *document*.getElementById("messageInput").value = '';  
 }  
 }  
  
 function getAllMessages() {  
 const xhr = new XMLHttpRequest();  
 xhr.open("GET", "http://localhost:8080/findAll", false);  
  
 xhr.send();  
  
 if (xhr.status === 200) {  
 const messages = *JSON*.parse(xhr.responseText);  
  
 const messageList = *document*.getElementById("messageList");  
 messageList.innerHTML = '';  
  
 messages.forEach(function(message) {  
 const p = *document*.createElement("p");  
 p.textContent = message.text;  
 messageList.appendChild(p);  
 });  
 } else {  
 alert("Failed to fetch messages!");  
 }  
 }  
  
 // document.addEventListener("DOMContentLoaded", function() {  
 // getAllMessages();  
 // });  
</script>  
</body>  
</html>

Вывод программы:



**Вывод**: в результате выполнения практической работы была освоена работа с многослойными клиент-серверными архитектурами, выявлена разница между ними, а также плюсы и минусы.