|  |
| --- |
|  |
| 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-20-21 | Хитров Н. С. |
| Принял преподаватель кафедры ИиППО | Волков М.Ю. |

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

Москва

2023

ОГЛАВЛЕНИЕ

[Цель работы 3](#_Toc148123595)

[Теоретическое введение 3](#_Toc148123596)

[Постановка задачи 5](#_Toc148123597)

[Программный код 6](#_Toc148123598)

[Результаты работы 12](#_Toc148123599)

[Вывод 13](#_Toc148123600)

# Цель работы

Знакомство студентов с одной из базовых технологии реализации конечной точки для передачи и приема данных по сети - webсокетом.

# 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 (рисунок 5.1) 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. (Вариант 25 - нечетный).

# Программный код

Листинг 1 – AviaTicketsController.java

@PostMapping("/getFlights")  
public JsonNode getFlights(@RequestBody FlightRequest flightRequest) throws IOException {  
  
 String origin = aviaService.findIATACode(flightRequest.getOrigin());  
 String destination = aviaService.findIATACode(flightRequest.getDestination());  
  
 String urlToApi = "https://api.travelpayouts.com/aviasales/v3/prices\_for\_dates?" +  
 "origin=" + origin + "&destination=" + destination +  
 "&departure\_at=" + flightRequest.getDeparture\_at() +  
 "&return\_at=" + flightRequest.getReturn\_at() +  
 "&sorting=price&direct=true&limit=10&" +  
 "token=15f19213084d9b861001e4d44ffe7d08";  
  
 RestTemplate restTemplate = new RestTemplate();  
 String result = restTemplate.getForObject(urlToApi, String.class);  
  
 ObjectMapper mapper = new ObjectMapper();  
 JsonNode jsonNode = mapper.readTree(result);  
  
 JsonNode dataNode = jsonNode.get("data");  
  
 aviaService.addIATACodeOfAirline(dataNode);  
 aviaService.convertCompanyCodeToName(dataNode);  
  
 flightRequest.setCreated\_at(LocalDate.*now*());  
 flightRequest.setPerson\_id(authenticatedPersonService.getAuthenticatedPerson().getId());  
  
 flightRequestService.saveRequest(flightRequest);  
  
 return dataNode;  
}

Листинг 2 – AviaService.java

public String findIATACode(String city) throws IOException {  
 String json = new String(Files.*readAllBytes*(Paths.*get*(  
 "src/main/resources/static/iatacodescities.json")));  
 JSONArray airports = new JSONArray(json); // Создаем массив JSON объектов из содержимого файла  
  
 for (int i = 0; i < airports.length(); i++) {  
 JSONObject airport = airports.getJSONObject(i);  
 if (airport != null && airport.optString("name").equals(city)) {  
 return airport.optString("code"); // Возвращаем IATA код для найденного города  
 }  
 }  
 return "City not found"; // Если город не найден, возвращаем null  
}

Листинг 3 – FlightSearchSystemApplication.java

@SpringBootApplication  
public class FlightSearchSystemApplication {  
  
 public static void main(String[] args) {  
 SpringApplication.*run*(FlightSearchSystemApplication.class, args);  
 }  
  
 @Bean  
 public WebMvcConfigurer corsConfigurer() {  
 return new WebMvcConfigurer() {  
 @Override  
 public void addCorsMappings(CorsRegistry registry) {  
 registry.addMapping("/\*\*").allowedOrigins("http://localhost:3000").allowedMethods("GET", "POST", "PUT", "DELETE");;  
 }  
 };  
 }  
  
}

Листинг 4 – index.html

<!DOCTYPE html>

<html lang="en">

  <head>

    <meta charset="utf-8" />

    <title>Ticket search system</title>

  </head>

  <body>

    <div id="root"></div>

  </body>

</html>

Листинг 5 – App.js

import { BrowserRouter, Route, Routes } from 'react-router-dom';

import './App.css';

import MainPage from './pages/MainPage/MainPage';

import RegPage from './pages/RegPage/RegPage';

import LogPage from './pages/LogPage/LogPage';

import UserPage from './pages/UserPage/UserPage';

function App() {

  return (

    <BrowserRouter className="App">

      <Routes>

        <Route path='/' element={<MainPage/>}/>

        <Route path='/main' element={<MainPage/>}/>

        <Route path='/register' element={<RegPage/>}/>

        <Route path='/login' element={<LogPage/>}/>

        <Route path='/account' element={<UserPage/>}/>

      </Routes>

    </BrowserRouter>

  );

}

export default App;

Листинг 6 – MainPage.jsx

import React from 'react'

import classes from './MainPage.module.css'

import Header from '../../components/Header/Header'

import SearchBox from '../../components/SearchBox/SearchBox'

import TicketList from '../../components/TicketsList/TicketList'

const MainPage = () => {

  return (

    <>

        <Header/>

        <main className={classes.main}>

            <h1>Подбор билетов для путешествий</h1>

            <p>Подбирайте и сравнивайте нужные билеты в один клик!</p>

            <SearchBox/>

            <TicketList/>

        </main>

    </>

  )

}

export default MainPage

Листинг 7 – package.json

{

  "name": "front",

  "version": "0.1.0",

  "private": true,

  "dependencies": {

    "@reduxjs/toolkit": "^1.9.5",

    "@testing-library/jest-dom": "^5.16.5",

    "@testing-library/react": "^13.4.0",

    "@testing-library/user-event": "^13.5.0",

    "axios": "^1.4.0",

    "moment": "^2.29.4",

    "react": "^18.2.0",

    "react-cookie": "^4.1.1",

    "react-dom": "^18.2.0",

    "react-icons": "^4.8.0",

    "react-redux": "^8.0.5",

    "react-router-dom": "^6.11.0",

    "react-scripts": "5.0.1",

    "redux": "^4.2.1",

    "universal-cookie": "^4.0.4",

    "web-vitals": "^2.1.4"

  },

  "scripts": {

    "start": "react-scripts start",

    "build": "react-scripts build",

    "test": "react-scripts test",

    "eject": "react-scripts eject"

  },

  "eslintConfig": {

    "extends": [

      "react-app",

      "react-app/jest"

    ]

  }

}

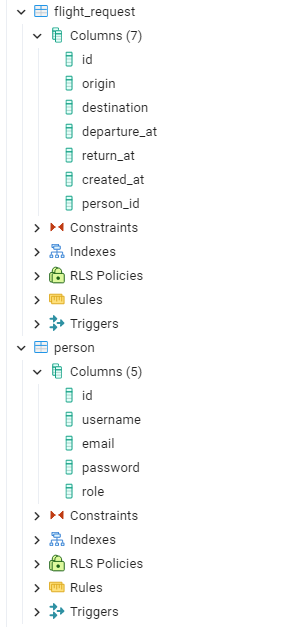


Рисунок 1 – Структура базы данных

# Результаты работы

Чтобы начать работу приложения необходимо сначала запустить сервер (рисунок 2), а пользователю перейти по ссылке http://localhost:3000.

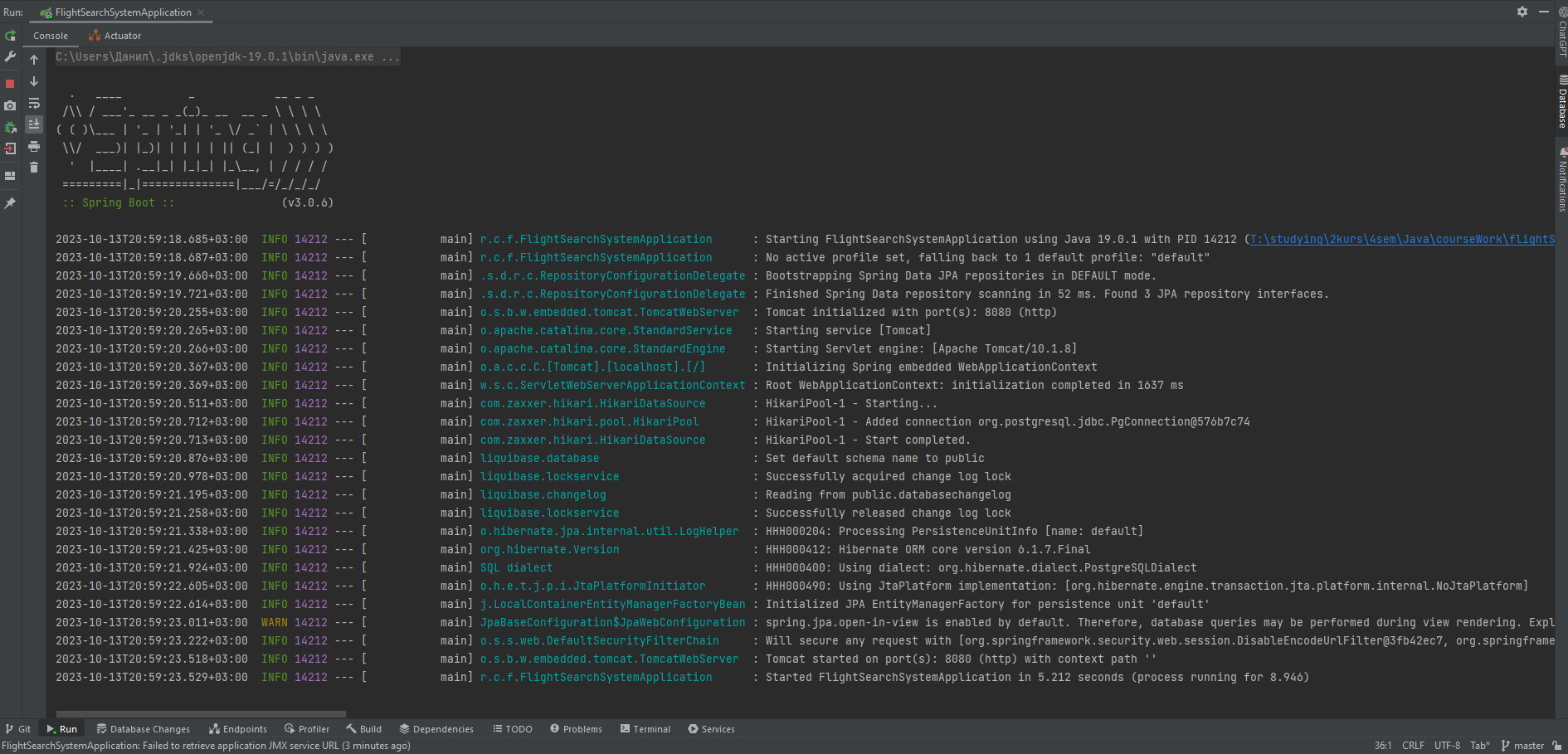


Рисунок 2 – Запуск сервера

Чтобы запустить клиентскую часть приложения, нужно запустить проект с помощью команды npm start (рисунок 3).

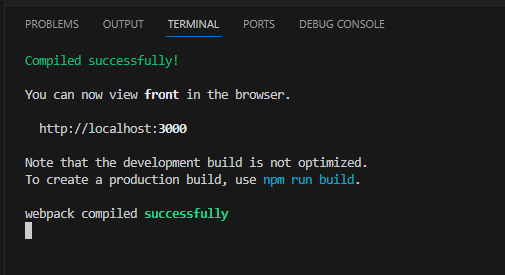


Рисунок 3 – Запуск клиентской части приложения

На данном рисунке (рисунок 4) представлена главная страница веб сервиса, полученного в результате разработки.

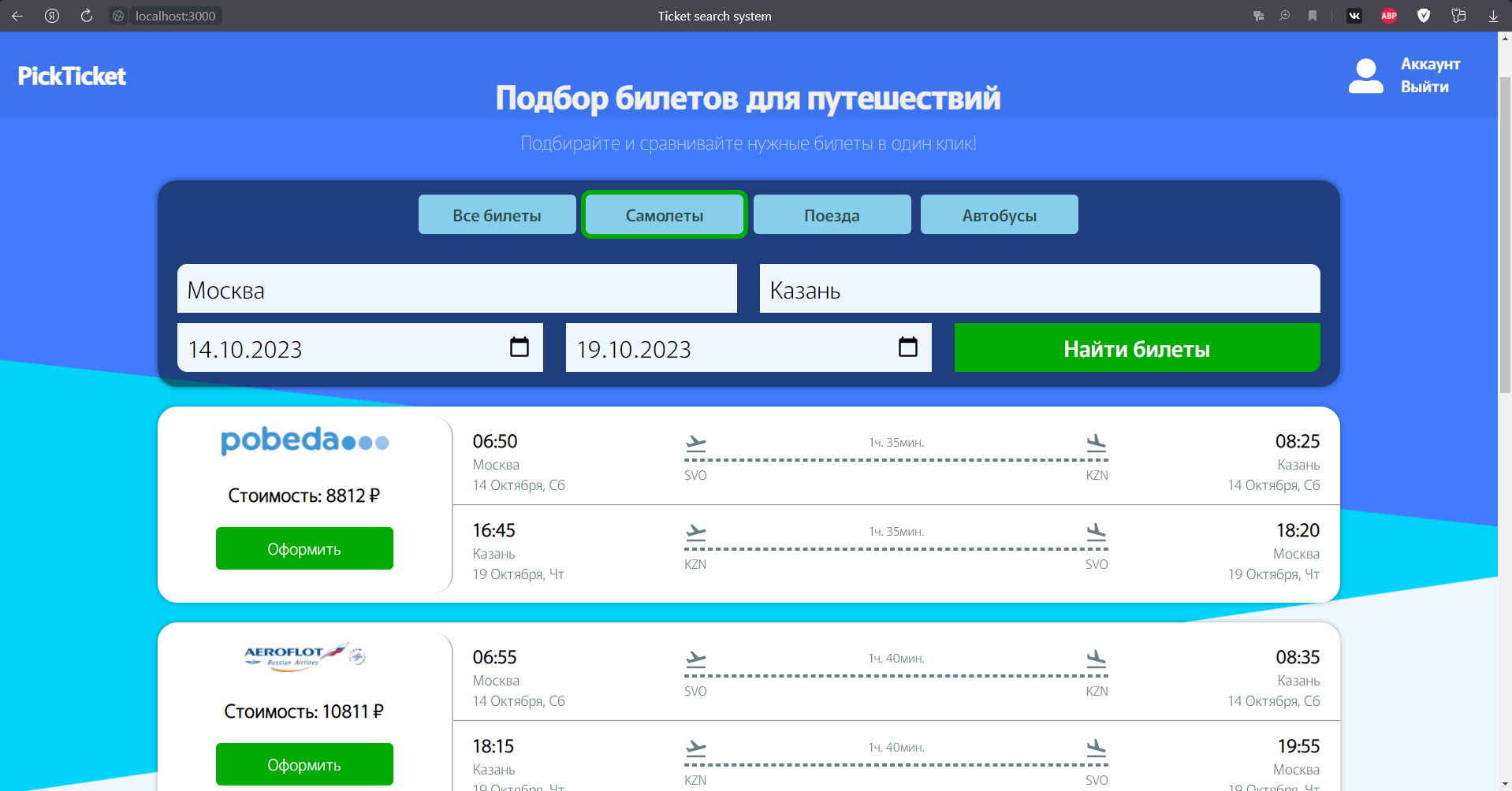


Рисунок 4 – Работа приложения

# Вывод

В результате выполнения данной практической работы было проведено ознакомление с многослойными клиент-серверными архитектурами, оценена разницу между ними, выявены плюсы и минусы.

**Список использованных источников**

1. Справочная информация JavaRush – URL: https://javarush.com/groups/posts/2488-obzor-rest-chastjh-3-sozdanie-restful-servisa-na-spring-boot (дата обращения: 13.10.2023) – текст: электронный.
2. Статья Baeldung – URL: https://www.baeldung.com/building-a-restful-web-service-with-spring-and-java-based-configuration (дата обращения: 13.10.2023) – текст: электронный.­­