Міністерство освіти і науки України Національний технічний університет України “Київський політехнічний інститут імені Ігоря Сікорського” Факультет інформатики та обчислювальної техніки Кафедра інформаційних систем та технологій

ЛАБОРАТОРНА РОБОТА №9

з дисципліни "Технології розроблення програмного забезпечення"

Тема: «Веб-сервіс автоматизованої перевірки програм лабораторного практикуму на мові програмування Java »

Виконав
студент групи ІА–33:
Заранік М.Ю

Перевірив:
Мягкий М.Ю.

Київ 2025

Зміст

Вступ

Мета роботи

Вивчити види взаємодії додатків (Client-Server, Peer-to-Peer, Service-oriented Architecture), та реалізувати в проєктованій системі одну із архітектур.

Завдання

• Ознайомитись з короткими теоретичними відомостями.

• Реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.

• Реалізувати функціонал для роботи в розподіленому оточенні відповідно до обраної теми.

• Реалізувати взаємодію розподілених частин:

– *Для клієнт-серверних варіантів*: реалізація клієнтської і серверної частини додатків, а також загальної частини (middleware); зв'язок клієнтської і серверної частин за допомогою WCF, TcpClient, .NET-Remoting на розсуд виконавця.

– *Для однорангових мереж*: реалізація взаємодії клієнтських додатків за допомогою WCF Peer to peer channel.

– *Для SOA додатків*: реалізація сервісу, що надає послуги клієнтським застосуванням; викладання сервісу в хмару або підняття у вигляді Web Service на локальній машині; використання токенів для передачі даних про автентифікації, двостороннє шифрування.

• Підготувати звіт щодо виконання лабораторної роботи. Поданий звіт повинен містити: діаграму класів, яка представляє спроєктовану архітектуру. Навести фрагменти програмного коду, які є суттєвими для відображення реалізованої архітектури.

Теоретичні відомості

У процесі розроблення сучасних програмних систем ключову роль відіграє вибір архітектурної моделі, яка визначає загальну структуру застосунку, спосіб організації взаємодії між його компонентами та характер розподілу обчислювального навантаження. Архітектура програмного забезпечення виступає концептуальною основою побудови системи, задаючи правила інтеграції модулів, обмін даними та механізми масштабування. Найпоширенішими підходами, що застосовуються під час створення розподілених систем, є клієнт-серверна модель, однорангова архітектура (Peer-to-Peer), сервіс-орієнтована архітектура (SOA) та мікросервісний стиль. Кожен з цих підходів сформувався як відповідь на конкретні потреби розвитку програмного забезпечення та вирішує коло характерних задач.

Архітектурний стиль не обмежується лише технічними особливостями; він визначає логіку еволюції системи, механізми розширення, особливості розроблення та підтримки. Такі моделі дають можливість формалізувати поведінку системи на високому рівні, забезпечити передбачуваність взаємодій і зменшити складність при подальшому масштабуванні. Правильний вибір архітектури дозволяє ефективно розподіляти ресурси, підвищувати продуктивність, забезпечувати надійність та адаптивність застосунку до змін.

9.1.1 Клієнт-серверна архітектура

Клієнт-серверна модель є однією з базових у розподілених системах. У ній виділяються два основні типи компонентів: клієнт, який надає інтерфейс та виконує запити, і сервер, що містить основну бізнес-логіку, здійснює обробку даних та повертає результат. Такий поділ дозволяє централізувати управління системою та спростити оновлення, оскільки всі зміни виконуються на серверній стороні.

За обсягом логіки на стороні клієнта розрізняють тонкі та товсті клієнти.

Тонкий клієнт передає майже всі операції на сервер, що робить його легким і простим у розгортанні. У такому випадку сервер бере на себе основне навантаження. Товстий клієнт, навпаки, виконує частину бізнес-логіки локально, тим самим зменшуючи кількість запитів і навантаження на сервер.

Проміжним варіантом є SPA-застосунки, які після первинного завантаження працюють переважно автономно та взаємодіють із сервером лише для отримання чи відправлення необхідних даних. Такий підхід поєднує наочність веб-інтерфейсу з високою продуктивністю й інтерктивністю.

9.1.2 Peer-to-Peer архітектура

На відміну від централізованої моделі, Peer-to-Peer (P2P) архітектура передбачає рівноправність усіх вузлів мережі. Кожен учасник може одночасно виступати і як клієнт, і як сервер, надаючи й отримуючи ресурси без посередництва центрального вузла. Децентралізація значно підвищує стійкість системи до збоїв, оскільки її робота не залежить від одного сервера.

P2P системи широко застосовуються у задачах:

– обміну файлами;

– розподілених обчислень;

– криптовалют та блокчейн-мереж;

– інтернет-телефонії та конференцій.

Основними викликами P2P архітектури є синхронізація, безпека взаємодії та ефективний пошук ресурсів у розподіленому середовищі.

9.1.3 Сервіс-орієнтована архітектура (SOA)

Сервіс-орієнтована архітектура виникла як еволюція клієнт-серверного підходу та відповідає потребі створення масштабованих систем, у яких компоненти можуть розвиватися незалежно. У SOA система складається з набору автономних сервісів, кожен з яких виконує конкретну бізнес-функцію й надає свій функціонал через стандартизований інтерфейс. Як протоколи взаємодії часто використовуються SOAP або REST, що забезпечує незалежність сервісів від конкретних реалізацій.

SOA підсилюється застосуванням шини обміну даними (ESB), яка відповідає за маршрутизацію, трансформацію повідомлень і координацію взаємодії сервісів. Такий підхід спрощує інтеграцію різних частин системи, у тому числі застарілих модулів, які можуть бути «обгорнуті» сервісами.

9.1.4 Мікросервісна архітектура

Мікросервісний підхід є подальшим розвитком SOA і характеризується більш дрібною та чіткою декомпозицією системи. Кожен мікросервіс представляє собою автономну складову, що має:

– власну базу даних або ізольований контекст;

– незалежний життєвий цикл;

– можливість окремого розгортання;

– чітко визначений контракт взаємодії.

Мікросервіси взаємодіють через легкі протоколи, такі як HTTP або AMQP, а масштабування відбувається на рівні окремих компонентів, що дозволяє більш ефективно оптимізувати ресурси.

Перевагами мікросервісів є висока гнучкість, спрощене масштабування, ізоляція помилок та можливість паралельної роботи над різними частинами системи. Водночас така архітектура потребує розвинених інструментів моніторингу, логування, автоматизованого розгортання та забезпечення узгодженості даних.

Хід роботи

У ході роботи було створено повноцінний застосунок, що складається з фронтенд та бекенд-частини. Сервер реалізовано на базі Spring Boot, а клієнтську частину — у вигляді React-застосунку, який взаємодіє з API через HTTP-запити та механізм SSE.

Бекенд відповідає за основну бізнес-логіку й містить модулі роботи з файлами, задачами, користувачами та відправленими рішеннями. Було реалізовано можливість завантаження файлів у GridFS, імпорт репозиторіїв GitHub, перегляд історії поданих рішень і повний цикл керування задачами. Під час відправлення рішення виконуються перевірки лімітів, коректності даних, а статус перевірки передається клієнту в режимі реального часу через SSE. Для авторизованих користувачів доступні статистика виконаних завдань та дані про власний профіль.

Фронтенд реалізовано як React-додаток із розділенням логіки на сторінки та компоненти. Через форми користувач може створювати задачі, завантажувати файли, переглядати репозиторії GitHub, надсилати рішення та стежити за їх статусом. Для роботи з API використовується fetch, а для реального часу — EventSource, який отримує SSE-події від сервера. Інтерфейс забезпечує навігацію між списком задач, історією рішень, профілем користувача та панеллю адміністратора.

У результаті отримано інтегрований клієнт-серверний застосунок, де Spring Boot забезпечує обробку даних та бізнес-логіку, а React відповідає за зручний інтерфейс, взаємодію з користувачем і відображення стану системи у реальному часі.

У процесі реалізації було розроблено серверну та клієнтську частини застосунку, архітектури яких представлено на діаграмі класів рисунок 1.1 та рисунок 1.2 відповідно. На них відображені основні контролери та сервіси, що забезпечують роботу системи, а у випадку клієнтської частини основні сторінки.

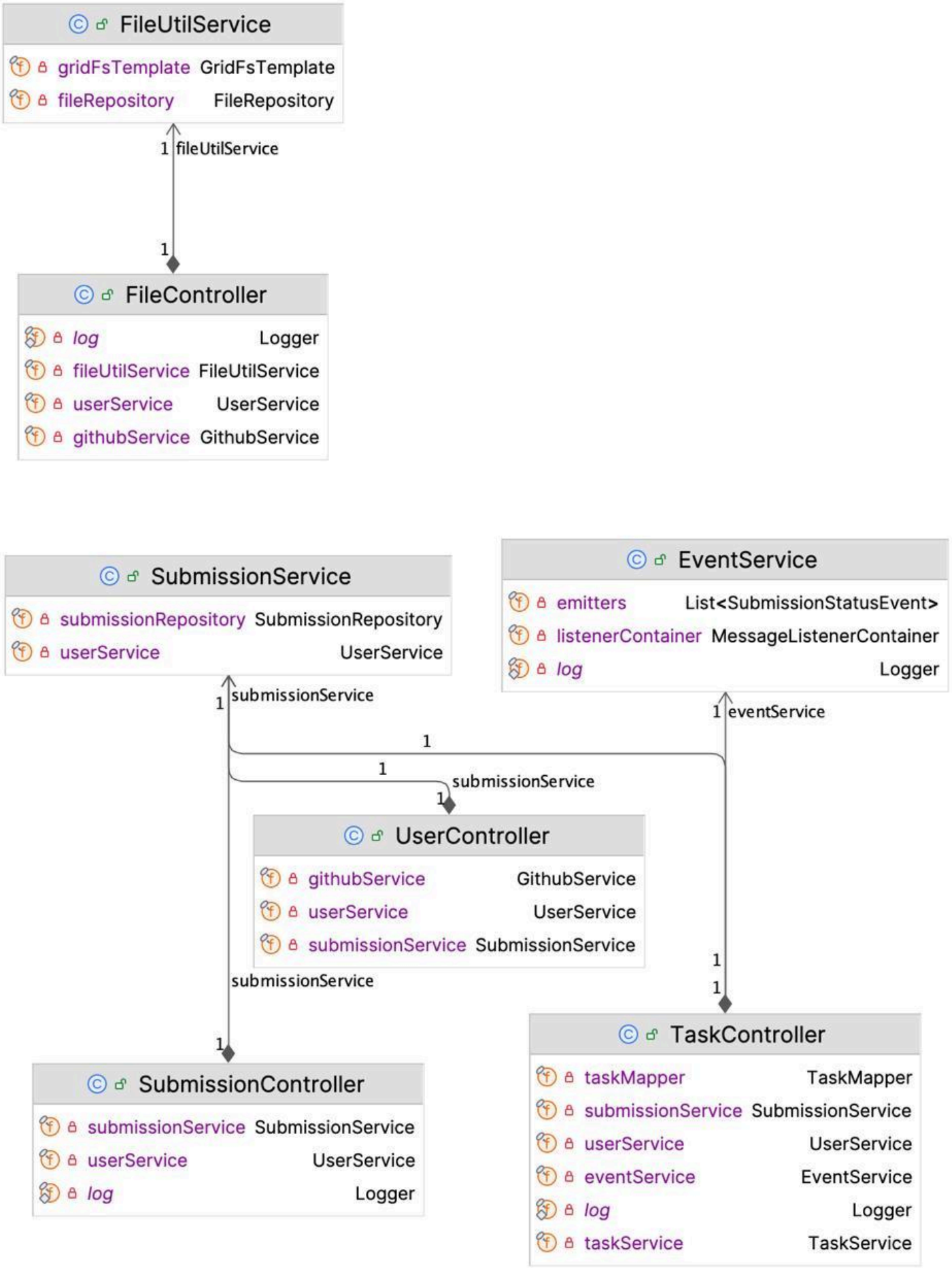

Рисунок 1.1. Діаграма класів серверної частини застосунку

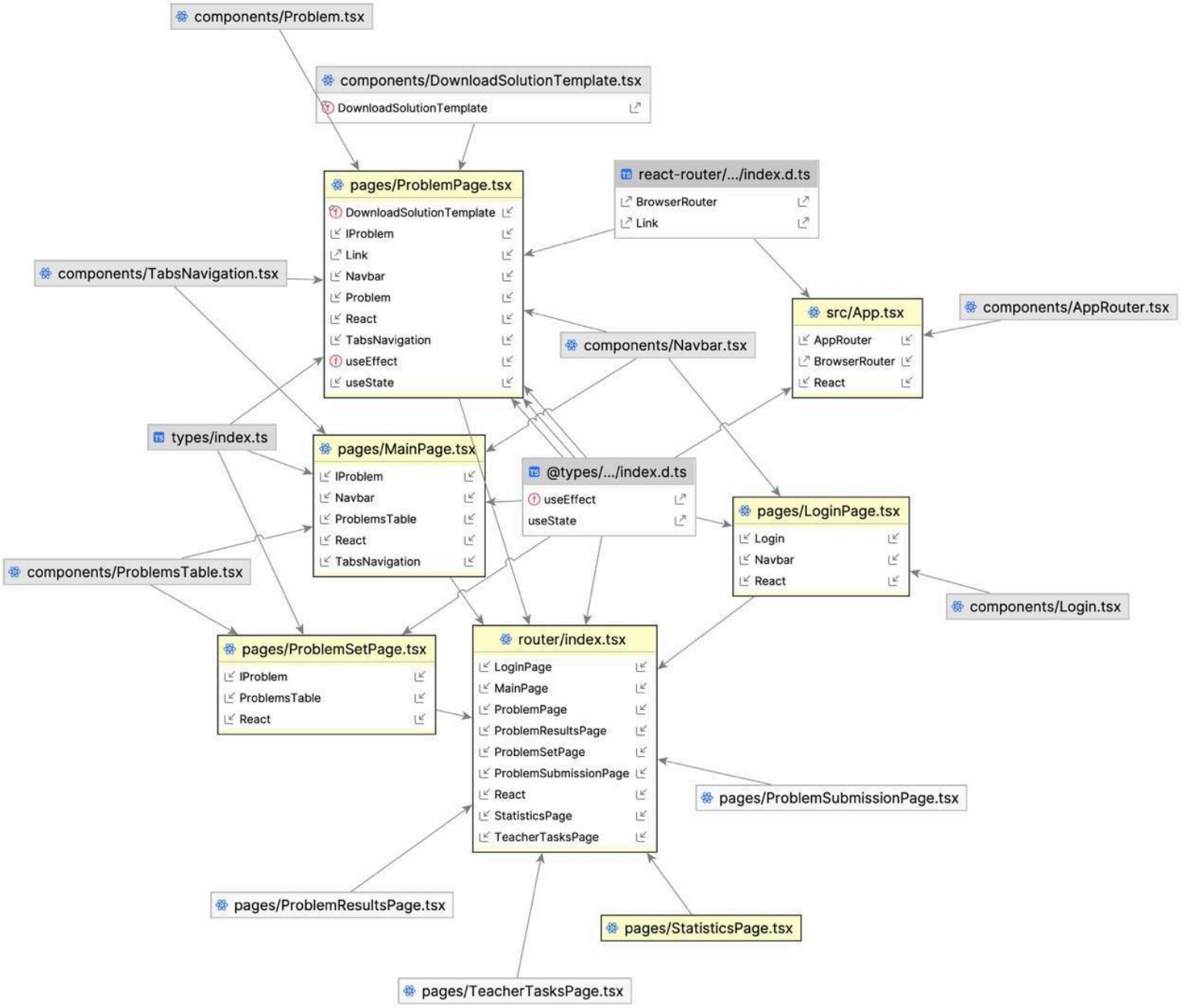

Рисунок 1.2. Діаграма класів клієнтської частини застосунку

Фрагменти програмного коду серверної частини

FileController

```
package com.example.demo.controller;
import java.io.IOException;

@Slf4j
@RestController
@RequiredArgsConstructor
@RequestMapping("files")
public class FileController {

   private final FileUtilService fileUtilService;
   private final UserService userService;
```

```
    private final GithubService githubService;

    @PostMapping("upload")
    @PreAuthorize("isAuthenticated()")
    public FileEntity uploadFile(@RequestParam("file") MultipartFile
file,
                                 @RequestParam FileType fileType)
throws IOException {
        String ownerId = userService.getCurrentUser().getId();
        return fileUtilService.uploadFile(
                file.getInputStream(),
                file.getOriginalFilename(),
                file.getContentType(),
                fileType,
                ownerId
        );
    }

    @GetMapping("download/{fileId}")
    public ResponseEntity<Resource>
downloadFile(@PathVariable("fileId") String fileId) {
        GridFsResource resourceFile =
fileUtilService.getFileById(fileId);
        String contentType = resourceFile.getContentType();

        HttpHeaders headers = new HttpHeaders();
        headers.setContentType(MediaType.parseMediaType(contentType));
        headers.setContentDisposition(
                ContentDisposition.attachment()
                        .filename(resourceFile.getFilename())
                        .build()
        );
        return
ResponseEntity.ok().headers(headers).body(resourceFile);
    }


    @GetMapping("github-save-zip/{repoName}")
    @PreAuthorize("isAuthenticated()")
    public FileEntity downloadGithubRepo(@PathVariable String
repoName) {
        log.debug("repoName: {}", repoName);
        return githubService.downloadAndSaveRepoToZip(repoName);
    }
}
```

SubmissionController

```
package com.example.demo.controller;

import com.example.demo.model.submission.SubmissionEntity;
import com.example.demo.service.submission.SubmissionService;
import com.example.demo.service.user.UserService;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import java.util.List;

@Slf4j
@RestController
@RequiredArgsConstructor
@RequestMapping("submissions")
public class SubmissionController {

   private final SubmissionService submissionService;
   private final UserService userService;

   @GetMapping("history")
   @PreAuthorize("isAuthenticated()")
   public List<SubmissionEntity> getUserSubmissions() {
       String userId = userService.getCurrentUser().getId();
       return submissionService.getSubmissionsByUser(userId);
   }

}
```

TaskController

```
package com.example.demo.controller;
import
org.springframework.web.servlet.mvc.method.annotation.SseEmitter;

import java.util.List;
import java.util.concurrent.TimeUnit;

@Slf4j
@RestController
@RequestMapping("tasks")
@RequiredArgsConstructor
```

```
public class TaskController {

    private final TaskService taskService;
    private final TaskMapper taskMapper;
    private final SubmissionService submissionService;
    private final EventService eventService;
    private final UserService userService;

    @PostMapping("submit")
    @PreAuthorize("isAuthenticated()")
    public SubmissionEntity submitTask(@RequestBody @Valid
TaskSubmissionRequestDto submitDto) {
        TaskEntity task =
taskService.findTaskById(submitDto.taskId());
        String userId = userService.getCurrentUser().getId();
        Integer submissionCount =
submissionService.getNumberSubmissionsForTask(userId,
submitDto.taskId());

        if (submissionCount >= task.getSubmissionsNumberLimit()) {
            throw new SubmissionNotAllowedException("Number of
submissions exceeded");
        }

        return submissionService.createSubmission(submitDto);
    }

    @GetMapping("status")
    @PreAuthorize("isAuthenticated()")
    public SseEmitter taskStatus(@RequestParam String submissionId) {
        String userId = userService.getCurrentUser().getId();
        SseEmitter emitter = new
SseEmitter(TimeUnit.SECONDS.toMillis(60));
        eventService.createSubmissionStatusEvent(emitter, userId,
submissionId);
        return emitter;
    }

    @PostMapping("create")
    @ResponseStatus(HttpStatus.CREATED)
    @PreAuthorize("hasAnyAuthority('ADMIN', 'TEACHER')")
    public String createTask(@RequestBody @Valid TaskCreateRequestDto
createDto) {
        if(createDto.testsPoints() + createDto.lintersPoints() !=
100){
            throw new ScoreExceededException("Score sum must be
100");
        }
```

```
        String ownerId = userService.getCurrentUser().getId();
        TaskEntity task = taskMapper.toEntity(createDto, ownerId);
        return taskService.save(task);
    }

    @DeleteMapping("delete")
    @PreAuthorize("hasAnyAuthority('ADMIN', 'TEACHER')")
    public void deleteTask(@RequestBody @Valid TaskDeletionRequestDto
deleteDto) {
        TaskEntity task =
taskService.findTaskById(deleteDto.taskId());
        String ownerId = userService.getCurrentUser().getId();

        if (!ownerId.equals(task.getOwnerId())) {
            throw new AccessDeniedException("You are not allowed to
delete this task");
        }

        taskService.removeTaskEntity(deleteDto.taskId());
    }

    @PutMapping("update")
    @PreAuthorize("hasAnyAuthority('ADMIN', 'TEACHER')")
    public TaskResponseDto updateTask(@RequestBody @Valid
TaskUpdateRequestDto updateDto) {
        TaskEntity task =
taskService.findTaskById(updateDto.taskId());
        String ownerId = userService.getCurrentUser().getId();

        if(updateDto.testsPoints() + updateDto.lintersPoints() !=
100){
            throw new ScoreExceededException("Score sum must be
100");
        }

        if (!ownerId.equals(task.getOwnerId())) {
            throw new AccessDeniedException("You are not allowed to
update this task");
        }

        TaskEntity updatedTask =
taskService.updateTask(taskMapper.toEntity(updateDto, ownerId));
        return taskMapper.toResponseDto(updatedTask);
    }

    @GetMapping("{id}")
    public TaskResponseDto findTask(@PathVariable String id) {
        TaskEntity task = taskService.findTaskById(id);
        return taskMapper.toResponseDto(task);
```

```
    }

    @GetMapping
    public List<TaskResponseDto> findAllTasks() {
        List<TaskEntity> tasks = taskService.findAll();
        return tasks.stream()
                .map(taskMapper::toResponseDto)
                .toList();
```

Фрагменти програмного коду клієнтської частини

LoginPage

```
const Login: React.FC = () => {
    const navigate = useNavigate();
    const token = useAuthStore(state => state.token);
    const setToken = useAuthStore(state => state.setToken);

    const [username, setUsername] = useState<string>("");
    const [password, setPassword] = useState<string>("");

    const handleSubmit = async (e: React.FormEvent) => {
        e.preventDefault();

        try {
            const response = await
fetch("http://localhost:8000/api/auth/login", {
                method: "POST",
                headers: {"Content-Type": "application/json"},
                body: JSON.stringify({username, password}),
            });

            if (!response.ok) throw new Error("Error");
            const data = await response.json();
            setToken(data.token);
            console.log(data.token);
            navigate('/')
        } catch (error) {
            console.error("Error:", error);
        }
    };


export default LoginPage;
```

ProblemPage

```
import React, { useEffect, useState } from 'react';
import Problem from "../components/Problem.tsx";
import Navbar from "../components/Navbar.tsx";
```

```
import type IProblem from "../types";
import TabsNavigation from "../components/TabsNavigation.tsx";
import { Link } from "react-router-dom";
import { DownloadSolutionTemplate } from
"../components/DownloadSolutionTemplate.tsx";

const ProblemPage: React.FC = () => {
    const [problem, setProblem] = useState<IProblem | null>(null);

    useEffect(() => {
        const fetchProblem = async () => {
            try {
                const response = await
fetch("http://localhost:8000/api/tasks/6901d40da193662393825c47");
                if (!response.ok) throw new Error("Failed to load
problem");
                const data = await response.json();
                setProblem(data);
                console.log(data);
            } catch (error) {
                console.error("Error loading problem:", error);
            }
        };
        fetchProblem();
    }, []);

    if (!problem) return <div>Loading...</div>;

    return (
        <>
            <Navbar/>

            <Link
                to='/problemset'
                className="decoration-none text-2xl ml-60 mt-2
font-bold text-black no-underline"
            >
                CSES Problem Set
            </Link>

            <TabsNavigation
                options={[
                    { value: 'tasks', path: '/problemset' },
                    { value: 'submit', path:
`/problemset/submit/${problem.id}` },
                    { value: 'result', path:
`/problemset/results/${problem.id}` },
                    { value: 'statistics', path:
`/problemset/statistics/${problem.id}` },
```

```
                ]}
            />

            <div className="max-w-3xl mx-auto p-6">
                <Problem
                    title={problem.title}
                    statement={problem.statement}
                    timeRestriction={problem.timeRestriction}
                    memoryRestriction={problem.memoryRestriction}

submissionsNumberLimit={problem.submissionsNumberLimit}
                />

                {problem.solutionTemplateFileId && (
                    <div className="mt-4">
                        <DownloadSolutionTemplate
solutionTemplateFileId={problem.solutionTemplateFileId}/>
                    </div>
                )}
            </div>
        </>
    );
};

export default ProblemPage;
```

Index.ts

```
import React from "react";
import LoginPage from "../pages/LoginPage.tsx";
import MainPage from "../pages/MainPage.tsx";
import ProblemSetPage from "../pages/ProblemSetPage.tsx";
import ProblemPage from "../pages/ProblemPage.tsx";
import ProblemSubmissionPage from
"../pages/ProblemSubmissionPage.tsx";
import ProblemResultsPage from "../pages/ProblemResultsPage.tsx";
import StatisticsPage from "../pages/StatisticsPage.tsx";
import TeacherTasksPage from "../pages/TeacherTasksPage.tsx";

export interface IRoute {
    path: string;
    element: React.ReactNode;
}

export enum RouteNames {
    LOGIN = '/login',
    PROBLEM_SET = '/problemset',
    MAIN = '/',
```

```
    TEACHER_PANEL = '/teacher-panel',
}

export const publicRoutes: IRoute[] = [
    {path: RouteNames.LOGIN, element: <LoginPage/>},
    {path: RouteNames.PROBLEM_SET, element: <ProblemSetPage/>},
    {path: RouteNames.MAIN, element: <MainPage/>},
    {path: RouteNames.PROBLEM_SET + '/task/:id', element:
<ProblemPage/>},
    {path: RouteNames.PROBLEM_SET + '/submit/:id', element:
<ProblemSubmissionPage/>},
    {path: RouteNames.PROBLEM_SET + '/results/:id', element:
<ProblemResultsPage/>},
    {path: RouteNames.PROBLEM_SET + '/statistics/:id', element:
<StatisticsPage/>},
    {path: RouteNames.TEACHER_PANEL, element: <TeacherTasksPage/>},
];

export const privateRoutes: IRoute[] = []
```

ProblemResultsPage

```
import React, { useEffect, useState } from 'react';
import Navbar from "../components/Navbar.tsx";
import { Link, useLocation } from "react-router-dom";
import TabsNavigation from "../components/TabsNavigation.tsx";
import { Table, Badge, Modal, Button } from "react-bootstrap";

type SubmissionStatus =
    "SUBMITTED" |
    "COMPILING" |
    "COMPILATION_SUCCESS" |
    "COMPILATION_ERROR" |
    "WRONG_ANSWER" |
    "ACCEPTED" |
    "TIME_LIMIT_EXCEEDED" |
    "OUT_OF_MEMORY";

interface ISubmission {
    id: string;
    taskId: string;
    status: SubmissionStatus;
    createdAt: string;
    logs?: string;
    userId?: string;
```

```
}

const getBadgeVariant = (status: SubmissionStatus): string => {
    switch (status) {
        case "ACCEPTED": return "success";
        case "COMPILATION_SUCCESS": return "success"
        case "COMPILATION_ERROR": return "danger";
        case "WRONG_ANSWER": return "danger"
        case "TIME_LIMIT_EXCEEDED": return "danger"
        case "SUBMITTED": return "secondary";
        default: return "dark";
    }
}

const ProblemResultsPage: React.FC = () => {
    const location = useLocation();
    const { state } = location;
    const submissionIdToTrack: string | undefined =
state?.submissionId;
    const taskId = location.pathname.split("/")[3];


    useEffect(() => {
        if (!submissionIdToTrack) return;
        const eventSource = new
EventSource(`http://localhost:8000/api/tasks/status?submissionId=${sub
missionIdToTrack}`);
        eventSource.onopen = () => console.log("SSE Connection
opened.");
        eventSource.onmessage = (event) => {
            try {
                const updatedSubmission: ISubmission =
JSON.parse(event.data);
                setTrackedSubmission(updatedSubmission);
            } catch (err) {
                console.error("Error parsing SSE data:", err);
            }
        };

        eventSource.onerror = (err) => {
            console.error("SSE Error:", err);
            eventSource.close();
        };

        return () => {
            eventSource.close();
            console.log("SSE Connection cleanup.");
        };
    }, [submissionIdToTrack]);
```

```
   return (
       <>
           <Navbar/>
           <Link to='/problemset' className="decoration-none
text-2xl ml-60 mt-2 font-bold text-black no-underline">
               CSES Problem Set
           </Link>

           <TabsNavigation options={[
               { value: 'tasks', path: '/problemset' },
               { value: 'submit', path:
`/problemset/submit/${taskId}` },
               { value: 'result', path:
`/problemset/results/${taskId}` },
               { value: 'statistics', path:
`/problemset/statistics/${taskId}` },
           ]} />

 export default ProblemResultsPage;
```

2. Питання до лабораторної роботи

1. **Що таке клієнт-серверна архітектура?**
   Клієнт-серверна архітектура — це модель, у якій клієнт надсилає запити, а сервер їх обробляє і повертає результат. Клієнт відповідає за інтерфейс і ініціює дії, сервер — за логіку, обробку та зберігання даних.

2. **Розкажіть про сервіс-орієнтовану архітектуру.**
   Сервіс-орієнтована архітектура складається з незалежних сервісів, кожен з яких виконує окрему бізнес-функцію. Сервіси мають стандартизовані контракти, автономність та можуть повторно використовуватися в різних частинах системи.

3. **Якими принципами керується SOA?**
   SOA базується на слабкому зв’язуванні, чітких контрактах, повторному використанні сервісів, автономності компонентів, стандартизованих інтерфейсах і незалежній еволюції кожного сервісу.

4. **Як між собою взаємодіють сервіси в SOA?**
   Сервіси взаємодіють через повідомлення, дотримуючись контрактів. Комунікація може бути синхронною або асинхронною, але завжди відбувається через стандартизований протокол, що гарантує сумісність.

5. **Як розробники взнають про існуючі сервіси і як робити до них запити?**
   Розробники використовують каталоги або реєстри сервісів, що містять інформацію про кінцеві точки та формати повідомлень. Запити формуються на основі контрактів, описаних у WSDL, OpenAPI чи інших специфікаціях.

6. **У чому полягають переваги та недоліки клієнт-серверної моделі?**
   Переваги: централізоване зберігання даних, чіткий розподіл обов’язків, простіша безпека і контроль. Недоліки: залежність від сервера як єдиної точки відмови та потреба у масштабуванні зі збільшенням навантаження.

7. **У чому полягають переваги та недоліки однорангової моделі взаємодії?**
   Переваги: відсутність центрального сервера, стійкість до збоїв, рівноправність вузлів. Недоліки: складніша безпека, проблеми з консистентністю даних, труднощі з керуванням та координацією.

8. **Що таке мікросервісна архітектура?**
   Мікросервісна архітектура — це стиль, у якому система складається з окремих малих сервісів, що виконують незалежні бізнес-функції. Кожен сервіс може розгортатися, оновлюватися і масштабуватися автономно.

9. **Які протоколи використовуються для обміну даними в мікросервісній архітектурі?**
   Використовуються HTTP/REST, gRPC, AMQP (RabbitMQ), Kafka, іноді WebSockets. Вибір залежить від вимог до продуктивності, типу комунікації та надійності.

10. **Чи можна назвати підхід сервіс-орієнтованою архітектурою, коли ми між контролерами та DAO створюємо шар сервісів?**
    Ні, це не SOA. Це внутрішня структуризація моноліту. SOA передбачає автономні сервіси, що працюють у мережі, мають власні контракти та інфраструктуру, а не просто окремі класи в межах одного застосунку.

Висновок

У ході лабораторної роботи було реалізовано клієнт-серверний застосунок, у якому клієнтська частина створена на React, а серверна — на Sprinp Boot. Сервер відповідає за обробку запитів, виконання бізнес-логіки та роботу з даними, тоді як клієнт забезпечує взаємодію з користувачем і надсилає запити до backend через REST API.

Основна увага приділялася чіткому розмежуванню відповідальностей між клієнтом і сервером. React-інтерфейс формує запити до визначених ендпойнтів та відображає отримані результати, тоді як Spring Boot реалізує контролери, сервісний шар і роботу зі сховищем даних. Такий підхід дає змогу підтримувати чисту архітектуру: клієнтська частина не містить логіки обробки, а сервер працює незалежно від UI, що спрощує розширення і тестування.

Завдяки використанню клієнт-серверної моделі застосунок став гнучким, масштабованим і придатним для подальшого розвитку: інтерфейс можна оновлювати окремо від серверної логіки, а сервер — розширювати, не змінюючи роботу фронтенду.

Backend: https://github.com/makszaranik/cses-java

Frontend: https://github.com/makszaranik/trpz-frontend