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 «Теорія розробки програмного забезпечення»

Тема: «Взаємодія компонентів системи»

Виконав:
Студент групи ІА-34
Цап Андрій

Перевірив:
Мягкий Михайло Юрійович

Київ – 2025

Зміст

Вступ

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

У даній лабораторній роботі було досліджено основні моделі взаємодії застосунків та методи організації обміну даними між компонентами. У межах реалізованої системи керування проєктами було обрано клієнт-серверну архітектуру як найбільш відповідну до структури платформи та характеру її функціональних можливостей.

Серверна частина представлена Spring Boot застосунком, який надає доступ до ресурсів системи через REST-інтерфейс. Клієнтська частина взаємодіє з сервером шляхом виконання HTTP-запитів і отримання даних у форматі JSON. Такий підхід дозволяє чітко відокремити логіку представлення даних від логіки їх обробки та забезпечує можливість подальшого масштабування проєкту.

Теоретичні відомості

Клієнт-серверна архітектура -клієнт-серверні додатки являють собою найпростіший варіант розподілених додатків, де виділяється два види додатків: клієнти (представляють додаток користувачеві) і сервери (використовується для зберігання і обробки даних). Розрізняють тонкі клієнти і товсті клієнти. Тонкий клієнт – клієнт, який повністю всі операції (або більшість, пов'язаних з логікою роботи програми) передає для обробки на сервер, а сам зберігає лише візуальне уявлення одержуваних від сервера відповідей. Грубо кажучи, тонкий клієнт – набір форм відображення і канал зв'язку з сервером. Прикладом тонкого клієнта є класичні Web-застосунки.

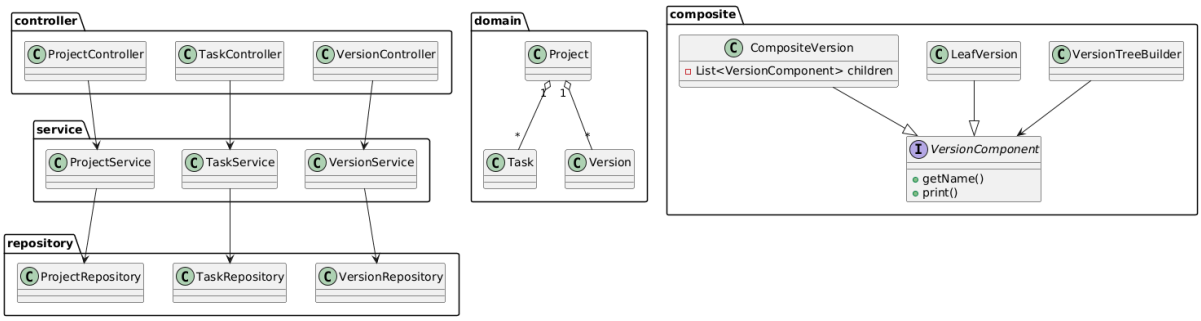
Peer-to-Peer архітектура -Peer-to-Peer (P2P) архітектура – це модель мережевої взаємодії, в якій кожен вузол (комп'ютер або пристрій) є одночасно клієнтом і сервером. У цій архітектурі всі вузли мають рівні права та можливості для обміну даними, ресурсами або виконання завдань. На відміну від клієнт-серверної моделі, де є 101 чітке розділення на клієнти й сервери, P2P-мережа дозволяє учасникам взаємодіяти безпосередньо, без необхідності в централізованому сервері.

Сервіс-орієнтована архітектура - сервіс-орієнтована архітектура (SOA, англ. service-oriented architecture) – модульний підхід до розробки програмного забезпечення, заснований на використанні розподілених, слабо пов’язаних (англ. Loose coupling) сервісів або служб, оснащених стандартизованими інтерфейсами для взаємодії за стандартизованими протоколами. Історично сервіс-орієнтована архітектура появилася як альтернатива монолітній архітектурі, в які вся система розроблялася та розгорталася як одне ціле.

Мікро-сервісна архітектура - сама назва дає зрозуміти, що мікро-сервісна архітектура є підходом до створення серверного додатку як набору малих служб. Це означає, що архітектура мікро-сервісів головним чином орієнтована на серверну частину, не дивлячись на те, що цей підхід так само використовується для зовнішнього інтерфейсу, де кожна служба виконується в своєму процесі і взаємодіє з іншими службами за такими протоколами, як HTTP/HTTPS, WebSockets чи AMQP. Кожен мікросервіс реалізує специфічні можливості в предметній області і свою бізнес-логіку в рамках конкретного обмеженого контексту, повинна розроблятись автономно і розвертатись незалежно.

Хід роботи

Діаграма класів спроектованої архітектури:



На UML-діаграмі зображено архітектуру багаторівневої системи управління проєктами, що складається з чітко розділених компонентів: controller, service, repository, domain. Така структура наслідує класичний підхід MVC/Service-Layer, забезпечує слабку зв’язаність елементів та полегшує розширення функціоналу.

У шарі controller розташовані три основні компоненти: ProjectController, TaskController та VersionController. Кожен з них відповідає за обробку HTTP-запитів, взаємодіє зі своїм відповідним сервісом та передає дані у вигляді моделей до шаблонів інтерфейсу. Контролери не містять бізнес-логіки, виконуючи роль тонких посередників.

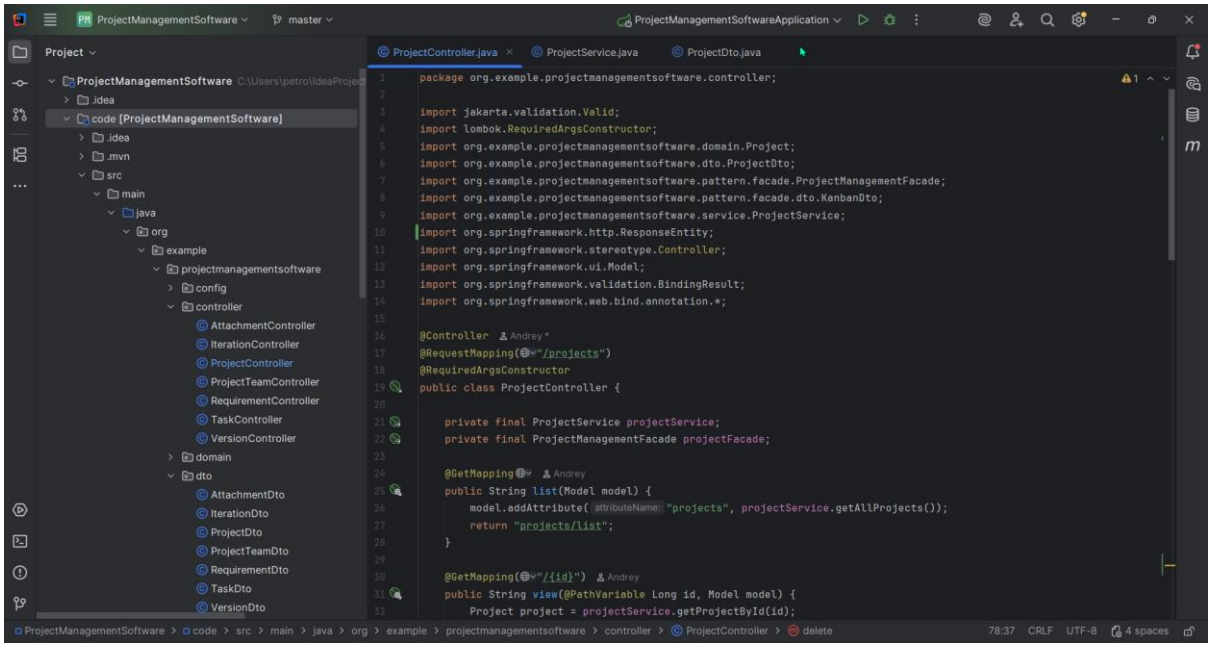
Сервісний шар представлений класами ProjectService, TaskService та VersionService. Саме тут зосереджена бізнес-логіка, перевірка даних, робота з файлами, обробка залежностей та підготовка даних для контролерів. Кожен сервіс звертається до свого відповідного репозиторію, забезпечуючи повне розмежування відповідальностей.

Репозиторний шар (ProjectRepository, TaskRepository, VersionRepository) забезпечує взаємодію з базою даних. Кожен репозиторій інкапсулює операції CRUD для відповідної сутності, зберігаючи систему розширюваною та незалежною від конкретної реалізації бази.

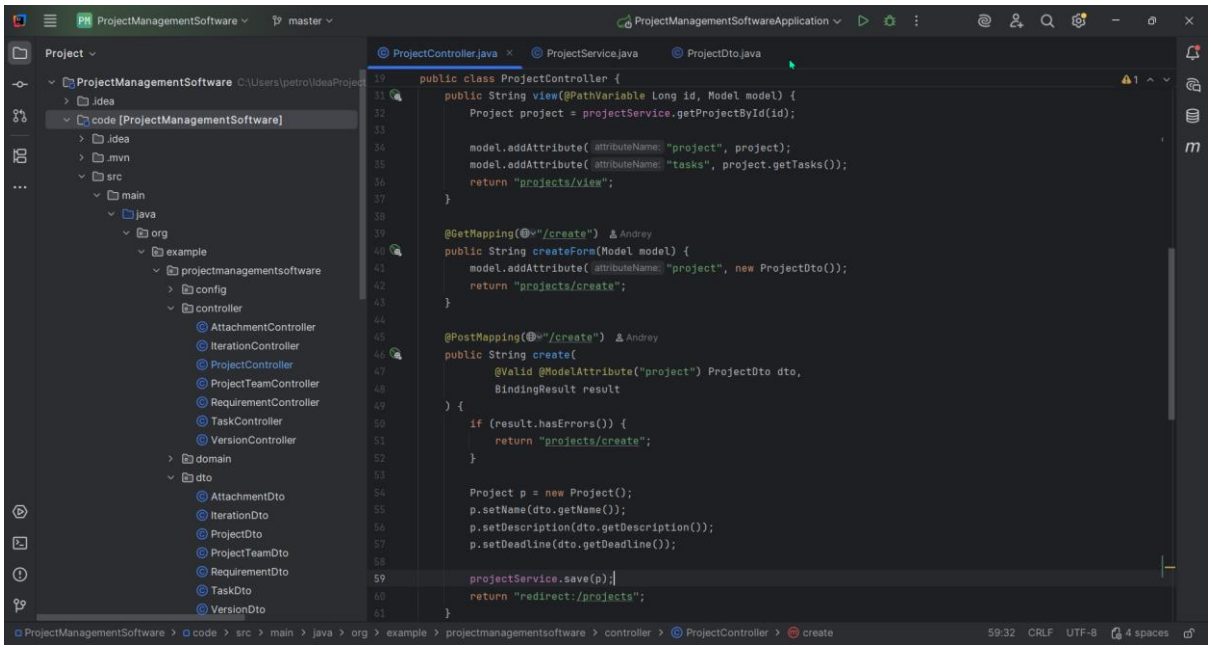
У шарі domain зображені основні сутності системи: Project, Task та Version. На діаграмі видно ключові зв’язки: проєкт може містити декілька задач та декілька версій (*зв’язок один-до-багатьох*). Таке моделювання відображає реальну структуру даних у системі та забезпечує коректну роботу всіх сервісів, які працюють із проєктною інформацією.

Уся система демонструє добре структуровану архітектуру: контролери відповідають за представлення, сервіси за логіку, репозиторії за дані, доменні моделі за структуру інформації, а модуль Composite забезпечує універсальне управління ієрархічними версіями. Така організація робить програму масштабованою, зрозумілою й легко підтримуваною.

Вихідні коди



```
package org.example.projectmanagementsoftware.controller;

import jakarta.validation.Valid;
import lombok.RequiredArgsConstructor;
import org.example.projectmanagementsoftware.domain.Project;
import org.example.projectmanagementsoftware.dto.ProjectDto;
import org.example.projectmanagementsoftware.pattern.facade.ProjectManagementFacade;
import org.example.projectmanagementsoftware.pattern.facade.dto.KanbanDto;
import org.example.projectmanagementsoftware.service.ProjectService;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.*;

@Controller  Andrey *
@RequestMapping("/projects")
@RequiredArgsConstructor
public class ProjectController {

    private final ProjectService projectService;
    private final ProjectManagementFacade projectFacade;

    @GetMapping  Andrey
    public String list(Model model) {
        model.addAttribute( attributeName: "projects", projectService.getAllProjects());
        return "projects/list";
    }

    @GetMapping("/{id}")  Andrey
    public String view(@PathVariable Long id, Model model) {
        Project project = projectService.getProjectById(id);
```



```
public class ProjectController {
    public String view(@PathVariable Long id, Model model) {
        Project project = projectService.getProjectById(id);

        model.addAttribute( attributeName: "project", project);
        model.addAttribute( attributeName: "tasks", project.getTasks());
        return "projects/view";
    }

    @GetMapping("/create")  Andrey
    public String createForm(Model model) {
        model.addAttribute( attributeName: "project", new ProjectDto());
        return "projects/create";
    }

    @PostMapping("/create")  Andrey
    public String create(
            @Valid @ModelAttribute("project") ProjectDto dto,
            BindingResult result
    ) {
        if (result.hasErrors()) {
            return "projects/create";
        }

        Project p = new Project();
        p.setName(dto.getName());
        p.setDescription(dto.getDescription());
        p.setDeadline(dto.getDeadline());

        projectService.save(p);
        return "redirect:/projects";
    }
```

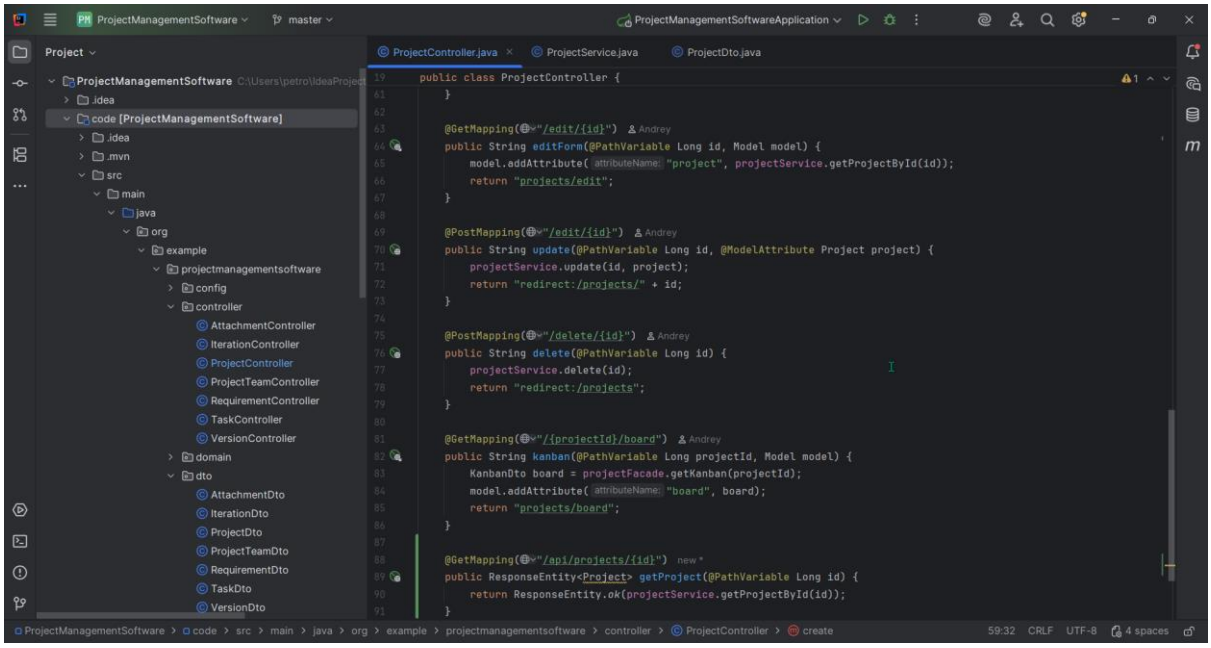
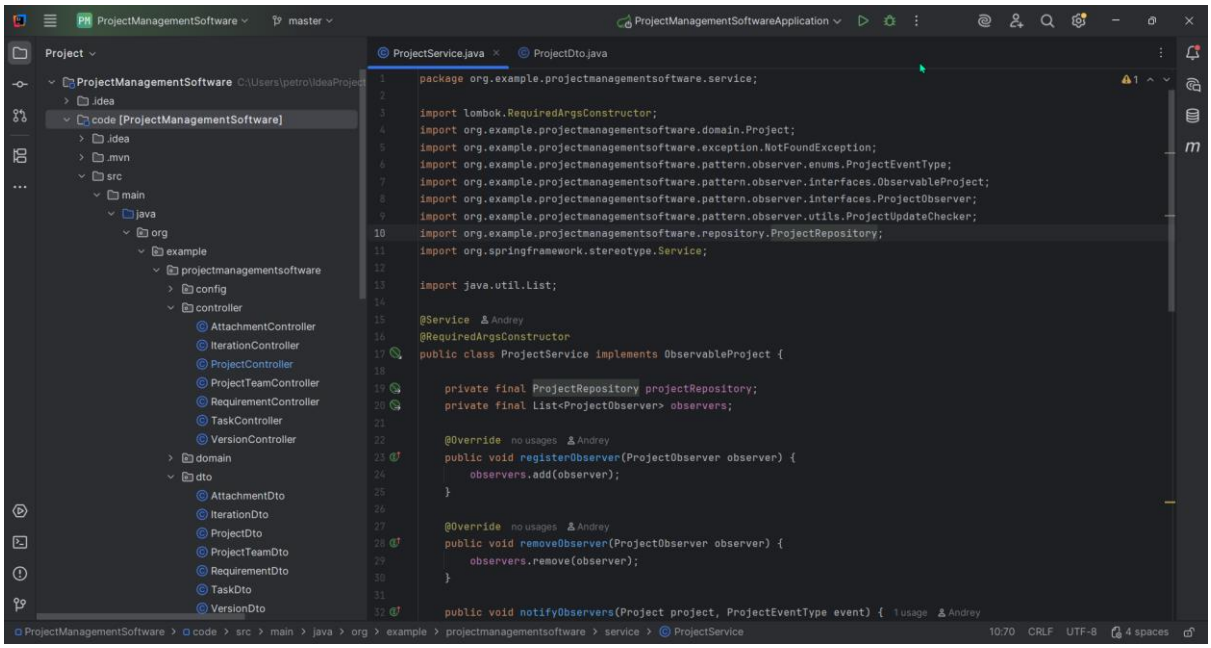


Рис 1,2,3 - Код ProjectController



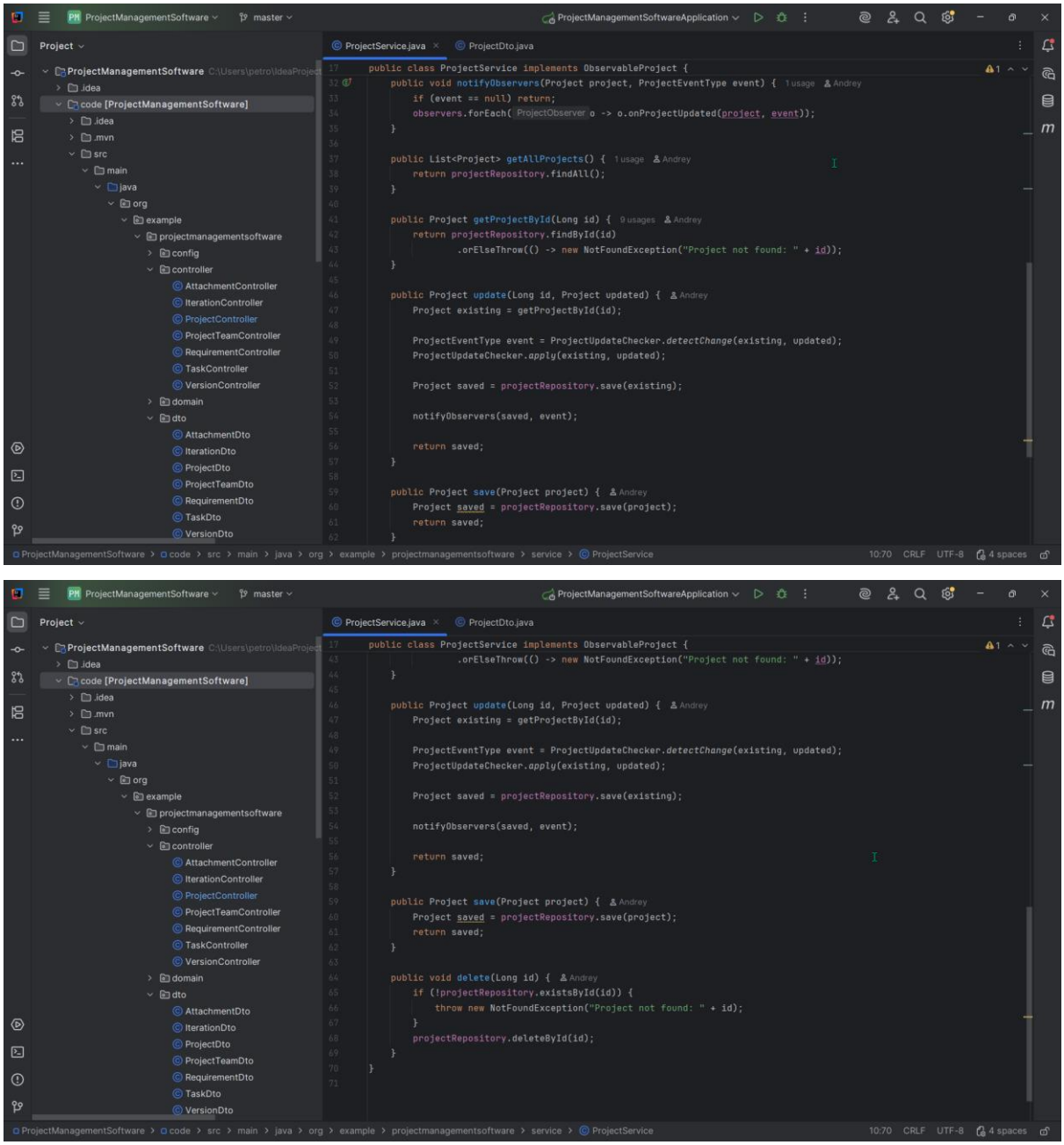


Рис 4,5,6 - Код ProjectService

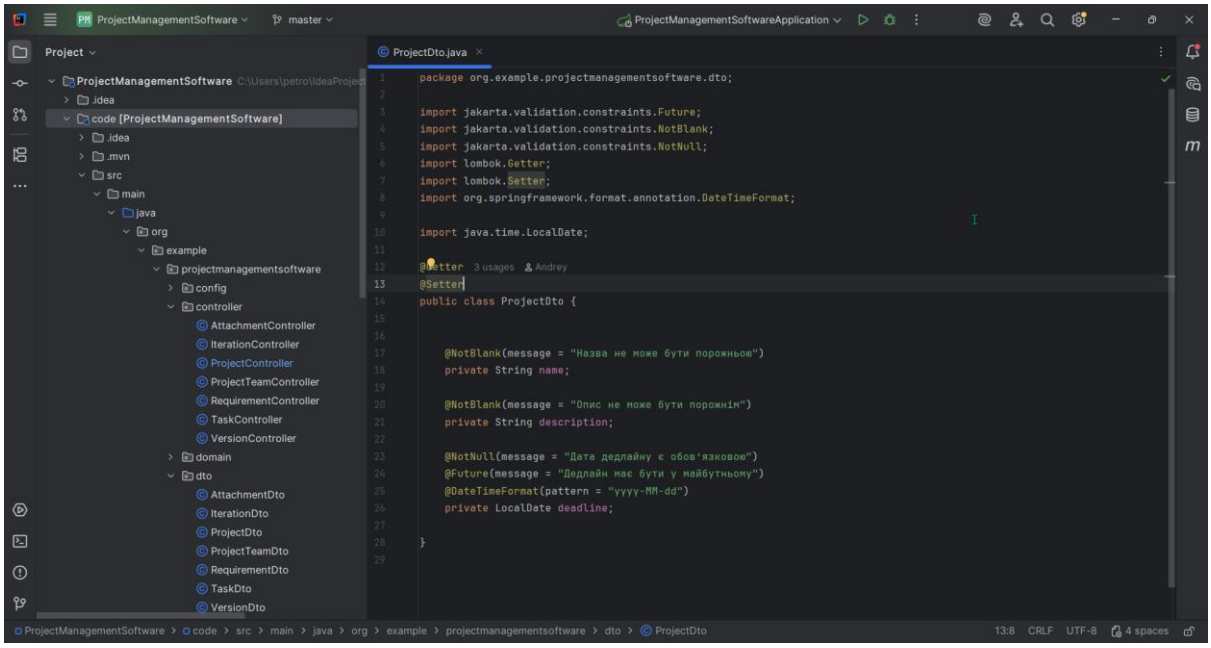


Рис 7 - Код ProjectDto

Висновки

У процесі виконання лабораторної роботи були розглянуті принципи взаємодії застосунків у розподілених системах та проаналізовані три основні архітектурні підходи: Client–Server, P2P та Service-Oriented Architecture. На основі структури розробленої системи було визначено, що клієнт-серверна модель є найбільш доцільною для впровадження.

Реалізація REST-взаємодії дозволила створити чіткий розподіл між клієнтською та серверною частинами, організувати обмін даними у стандартизованому форматі та забезпечити гнучкість у розширенні функціоналу. Додавання окремих ендпоїнтів та демонстрація їхньої роботи підтвердили відповідність системи вимогам розподіленого середовища.

Контрольні запитання

1. Що таке клієнт-серверна архітектура?

Модель, де клієнт надсилає запити, а сервер виконує логіку, працює з даними і відповідає.

2. Розкажіть про сервіс-орієнтовану архітектуру.

SOA — підхід, де система складається з незалежних сервісів. Кожен сервіс виконує окрему функцію, має чіткий інтерфейс і взаємодіє з іншими через стандартизовані протоколи.

3. Якими принципами керується SOA?

- слабке зв’язування
- повторне використання сервісів
- чітко визначені контракти
- незалежність реалізації
- доступність через стандартизовані інтерфейси
- можливість комбінування сервісів у більші процеси

4. Як між собою взаємодіють сервіси в SOA?

Через стандартизовані протоколи: SOAP, REST, XML/JSON. Один сервіс надсилає запит іншому через його публічний інтерфейс.

5. Як розробники взнають про існуючі сервіси і як робити до них запити?

Через реєстр сервісів, документацію або опис контрактів. Виклик відбувається через URL або ендпоінт сервісу.

6. У чому полягають переваги та недоліки клієнт-серверної моделі?

Переваги:

- централізоване управління даними
- безпека
- масштабованість
- розділення обов’язків

Недоліки:

- залежність від сервера
- навантаження концентрується в одній точці
- може бути "вузьке місце" при великому трафіку

7. У чому полягають переваги та недоліки однорангової моделі взаємодії?

Переваги:

- немає центральної точки відмови
- краща масштабованість у плані ресурсів
- кожен вузол може бути і клієнтом, і сервером

Недоліки:

- складніше забезпечити безпеку
- важко контролювати мережу
- непередбачуваність доступності вузлів

8. Що таке мікро-сервісна архітектура?

Підхід, де додаток розбивається на дуже дрібні незалежні сервіси. Кожен мікросервіс реалізує вузьку функцію, має свою базу даних, розгортається і масштабується окремо.

9. Які протоколи використовуються для обміну даними в мікросервісній архітектурі?

- HTTP/REST
- gRPC
- WebSockets
- AMQP, Kafka

10. Чи можна назвати підхід сервіс-орієнтованою архітектурою, коли ми в проєкті між шаром веб-контролерів та шаром доступу до даних реалізуємо шар бізнес-логіки у вигляді сервісів?

Ні, це просто шар бізнес-логіки всередині моноліту. SOA — це архітектура між окремими сервісами, а не між шарами в одному проєкті.