Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

ЛАБОРАТОРНА РОБОТА №3

з дисципліни «Теорія розробки програмного забезпечення»

Тема: «Основи проектування розгортання»

Виконав:
Студент групи ІА-34
Цап Андрій

Перевірив:
Мягкий Михайло Юрійович

Київ – 2025

Зміст

Вступ

Мета: Навчитися проєктувати діаграми розгортання та компонентів для системи що проєктується, а також розробляти діаграми взаємодії, а саме діаграми послідовностей, на основі сценаріїв зроблених в попередній лабораторній роботі.

У процесі проєктування систем важливо не лише визначити функціональні можливості та структуру даних, але й описати архітектурні рішення, спосіб взаємодії модулів та їхнє фізичне розташування у середовищі виконання. UML-діаграми розгортання та компонентів дозволяють формалізувати логічну й фізичну структуру системи, забезпечити зрозуміле документування та спростити подальший етап реалізації.

У даній лабораторній роботі продовжується проєктування системи **“Project Management Software”**. На основі сценаріїв, створених у попередній роботі, будуються діаграми послідовностей, визначається архітектура системи, розробляється діаграма розгортання та компонентів, а також виконується розширення функціоналу програмної частини системи.

Теоретичні відомості

Діаграма розгортання (Deployment Diagram) - діаграми розгортання представляють фізичне розташування системи, показуючи, на якому фізичному обладнанні запускається та чи інша складова програмного забезпечення.

Головними елементами діаграми є вузли, пов'язані інформаційними шляхами. Вузол (node) – це те, що може містити програмне забезпечення. Вузли бувають двох типів. Пристрій (device) – це фізичне обладнання: комп'ютер або пристрій, пов'язаний із системою. Середовище виконання (execution environment) – це програмне забезпечення, яке саме може включати інше програмне забезпечення, наприклад операційну систему або процес-контейнер (наприклад, вебсервер).

Діаграми розгортань розрізняють двох видів: описові та екземплярні. На діаграмах описової форми вказуються вузли, артефакти і зв'язки між вузлами без вказівки конкретного обладнання або програмного забезпечення, необхідного для розгортання.

Діаграма компонентів - діаграма компонентів UML є представленням проєктованої системи, розбитої на окремі модулі.

Залежно від способу поділу на модулі розрізняють три види діаграм компонентів:

- логічні;
- фізичні;
- виконувані.

Компоненти можуть поділятися за фізичними одиницями – окремі вузли розподіленої системи – набір комп'ютерів і серверів; на кожному з вузлів можуть бути встановлені різні виконувані компоненти. Такий вид діаграм компонентів застарів і зазвичай замість нього використовують діаграму розгортань.

Діаграми послідовностей - діаграма послідовностей (Sequence Diagram) – це один із типів діаграм у моделюванні UML (Unified Modeling Language), який використовується для моделювання взаємодії між об'єктами системи у певній послідовності часу. Вона відображає, як об'єкти обмінюються повідомленнями, показуючи порядок і логіку виконання операцій.

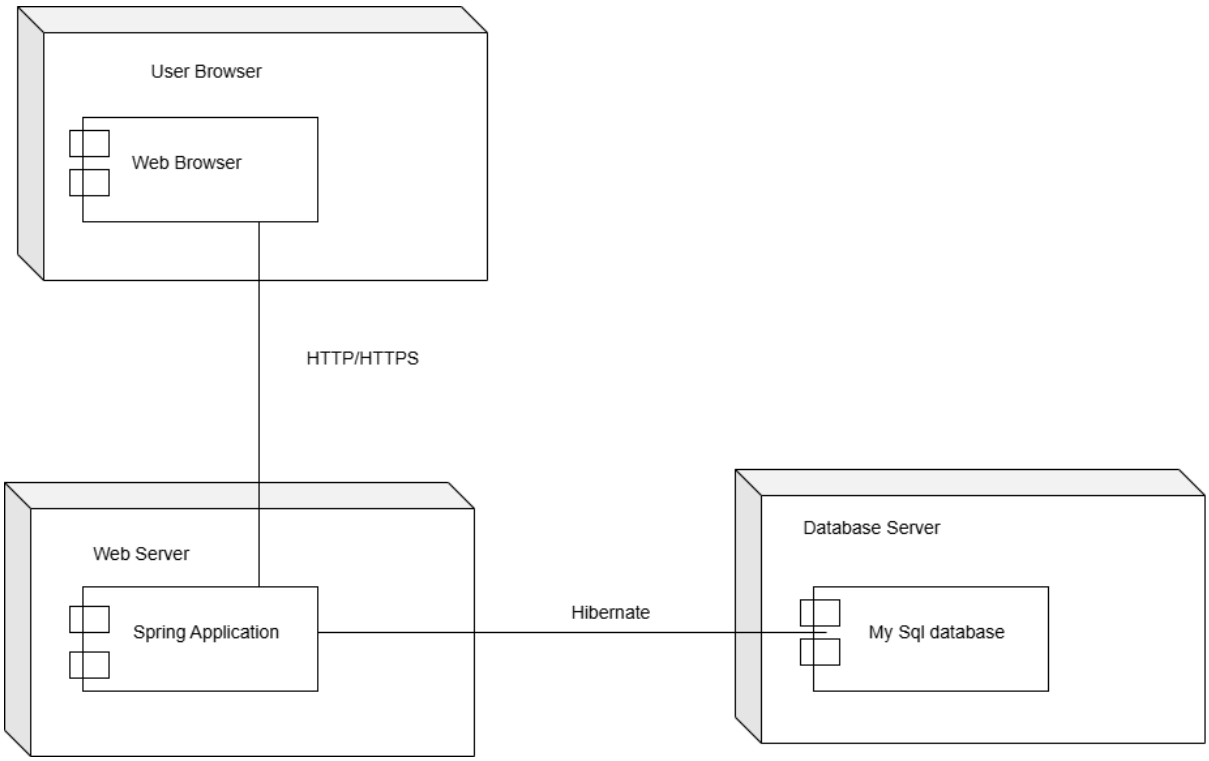
Актори (Actors): Зазвичай позначаються піктограмами або назвами. Це користувачі чи інші системи, які взаємодіють із системою. Актори можуть бути зовнішніми стосовно моделювання системи.

Об'єкти або класи: Розміщуються горизонтально на діаграмі. Вони позначаються прямокутниками з іменем об'єкта або класу під прямокутником. Кожен об'єкт має «життєвий цикл», який представлений вертикальною пунктирною лінією (лінія життя).

Повідомлення: Це лінії зі стрілками, які з'єднують об'єкти. Вони показують передачу повідомлень чи виклик методів. Стрілка може бути синхронною (звичайна стрілка) або асинхронною (лінія з відкритим трикутником) та з пунктирною лінією, що показує повернення результату.
Активності: Вказують періоди, протягом яких об'єкт виконує певну дію. На діаграмі це позначається прямокутником, накладеним на лінію життя.
Контрольні структури: Використовуються для відображення умов, циклів або альтернативних сценаріїв.

Хід роботи

Діаграма розгортання



1. User Browser

Компоненти: Web Browser
Призначення: Клієнтський вузол, через який Project Manager та Team Member взаємодіють із системою.

Функції:

- Відображення інтерфейсу системи управління проєктами.
- Надсилання HTTP/HTTPS-запитів до серверної частини (створення проєктів, завдань, спринтів).
- Отримання та показ результатів: списки проєктів, задач, вкладень.
- Інтерактивні елементи: форми, таблиці, кнопки, перегляд деталей.

2. Web Server

Компоненти: Spring Application
Призначення: Сервер застосунку, що обробляє бізнес-логіку.

Функції:

- Обробка HTTP-запитів від клієнтського браузера.
- Виконання бізнес-логіки (створення проектів, зміна статусу задач, додавання вкладень, планування спринтів).
- Взаємодія з базою даних через Hibernate/JPA.
- Формування відповідей для UI.

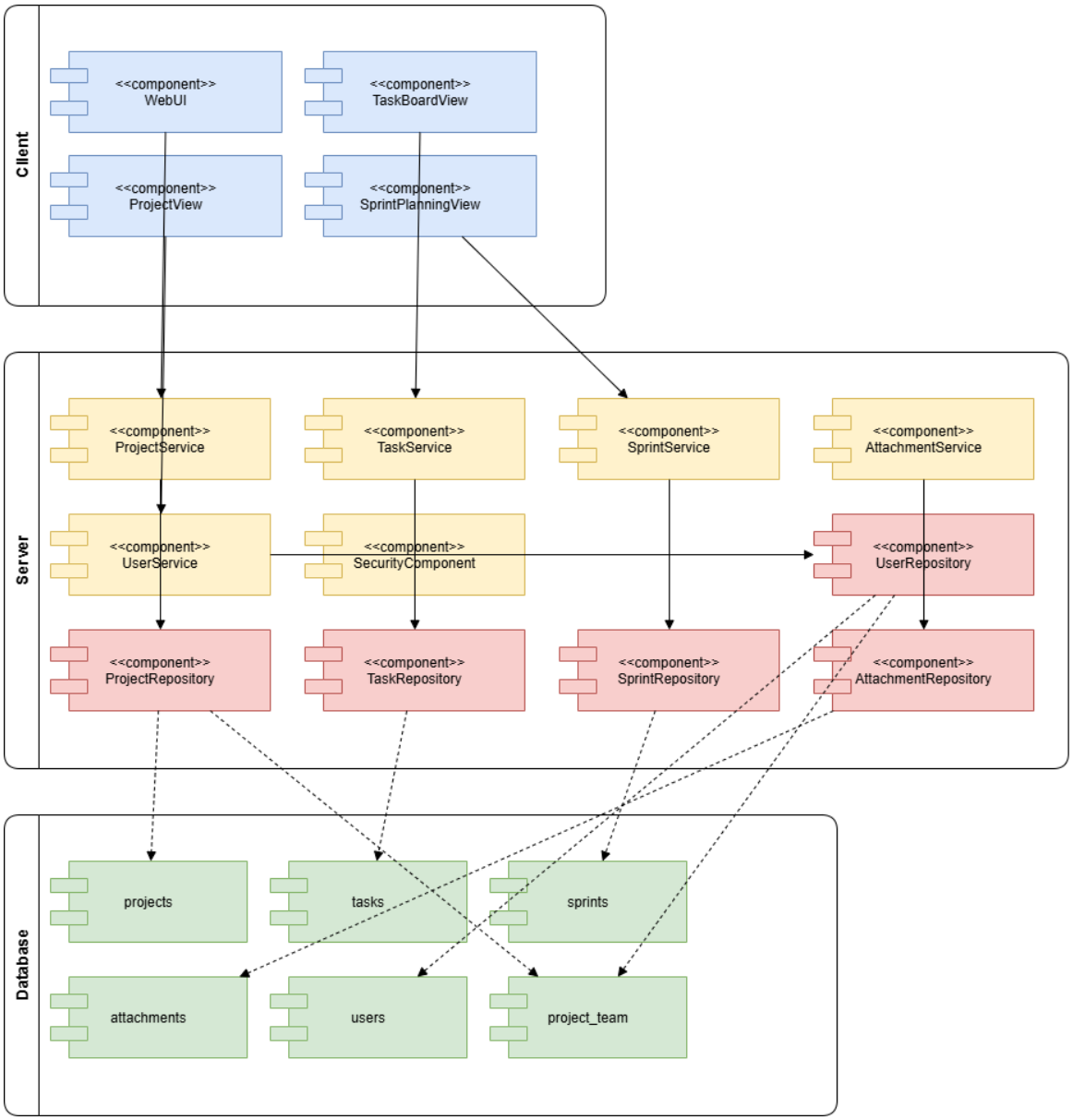
3. Database Server

Компоненти: MySQL Database
Призначення: Збереження даних системи.

Функції:

- Зберігання таблиць: users, projects, tasks, sprints, attachments, project_team.
- Забезпечення цілісності та зв’язків між об’єктами.
- Обробка SQL-запитів від Spring Application.

Діаграма компонентів



На діаграмі зображено архітектуру системи *Project Management Software*, яку поділено на три основні частини: **Client**, **Server** та **Database**. Кожна частина містить свої компоненти та чітко визначені взаємодії між ними.

1. Client

Клієнтська частина системи представлена компонентами інтерфейсу користувача, що забезпечують відображення інформації та взаємодію користувача з функціоналом системи.

Компоненти:

- **WebUI** — базовий інтерфейс для загальної навігації та роботи користувача.
- **TaskBoardView** — компонент для перегляду та оновлення статусів завдань.
- **ProjectView** — інтерфейс для роботи зі списком проєктів, їх характеристиками та командою.
- **SprintPlanningView** — інтерфейс для планування та перегляду спринтів.

Призначення:

- відображення даних, отриманих із сервера;
- надсилання HTTP/HTTPS-запитів до Server-частини;

2. Server

Серверна частина відповідає за бізнес-логіку системи, обробку запитів від клієнта та роботу з базою даних.

Сервіси (Business Logic Layer):

- **ProjectService**
  Виконує операції створення, редагування, перегляду та видалення проєктів. Взаємодіє з ProjectRepository.
- **TaskService**
  Обробляє логіку завдань: призначення виконавця, зміну статусу, встановлення дедлайнів, прив’язку вкладень.
- **SprintService**
  Відповідає за створення спринтів, визначення тривалості та облік робочих годин.
- **AttachmentService**
  Реалізує додавання, перевірку та видалення вкладень, що прикріплюються до завдань.
- **SecurityComponent**
  Логіка перевірки доступу користувача, роботи з ролями та авторизацією.

Репозиторії (Data Access Layer):

- **ProjectRepository** — взаємодія з таблицею *projects*.
- **TaskRepository** — доступ до таблиці *tasks*.
- **SprintRepository** — операції з таблицею *sprints*.
- **UserRepository** — робота з таблицею *users*.
- **AttachmentRepository** — взаємодія з таблицею *attachments*.

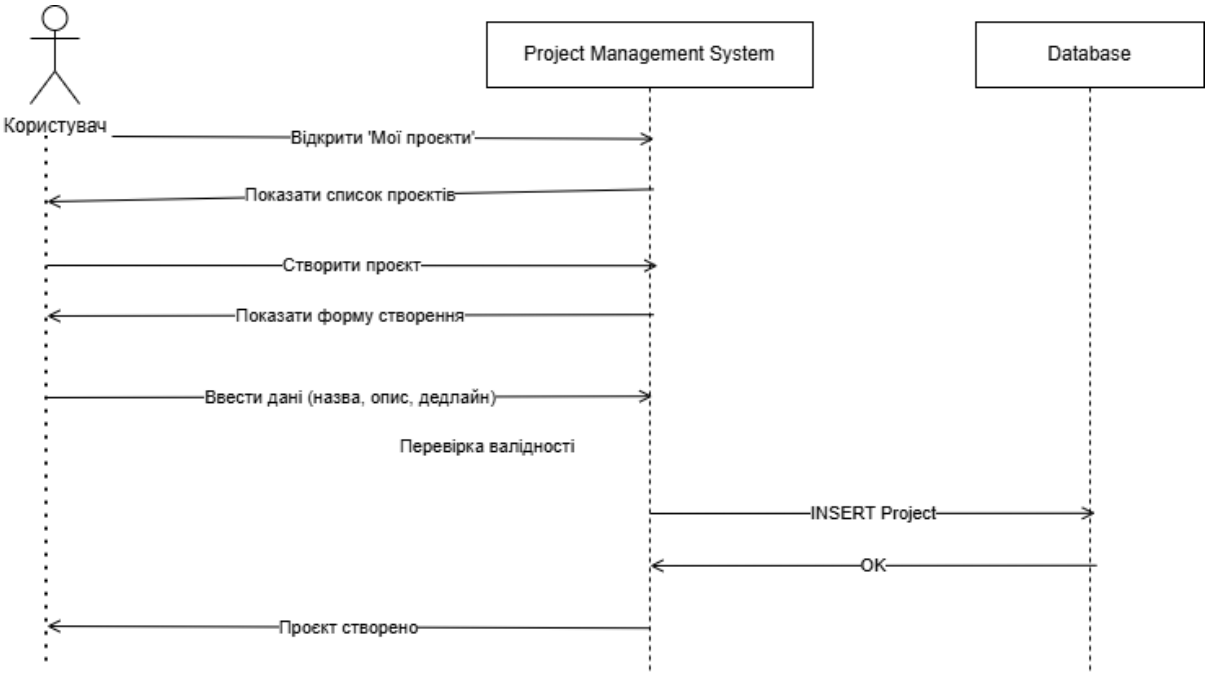
3. Database

База даних містить усі основні таблиці системи та забезпечує збереження інформації про користувачів, проєкти, спринти, завдання й вкладення.

Таблиці:

- **projects** — основна таблиця з даними проєктів: назва, опис, дедлайн.
- **tasks** — завдання зі статусом, дедлайном, пріоритетом та виконавцем.
- **sprints** — періоди роботи над частиною проєкту з прогнозованою кількістю годин.
- **users** — користувачі системи з ролями (MANAGER / MEMBER).
- **attachments** — прикріплені файли до завдань.
- **project_team** — зв’язувальна таблиця між проєктами та користувачами.

Діаграма послідовностей

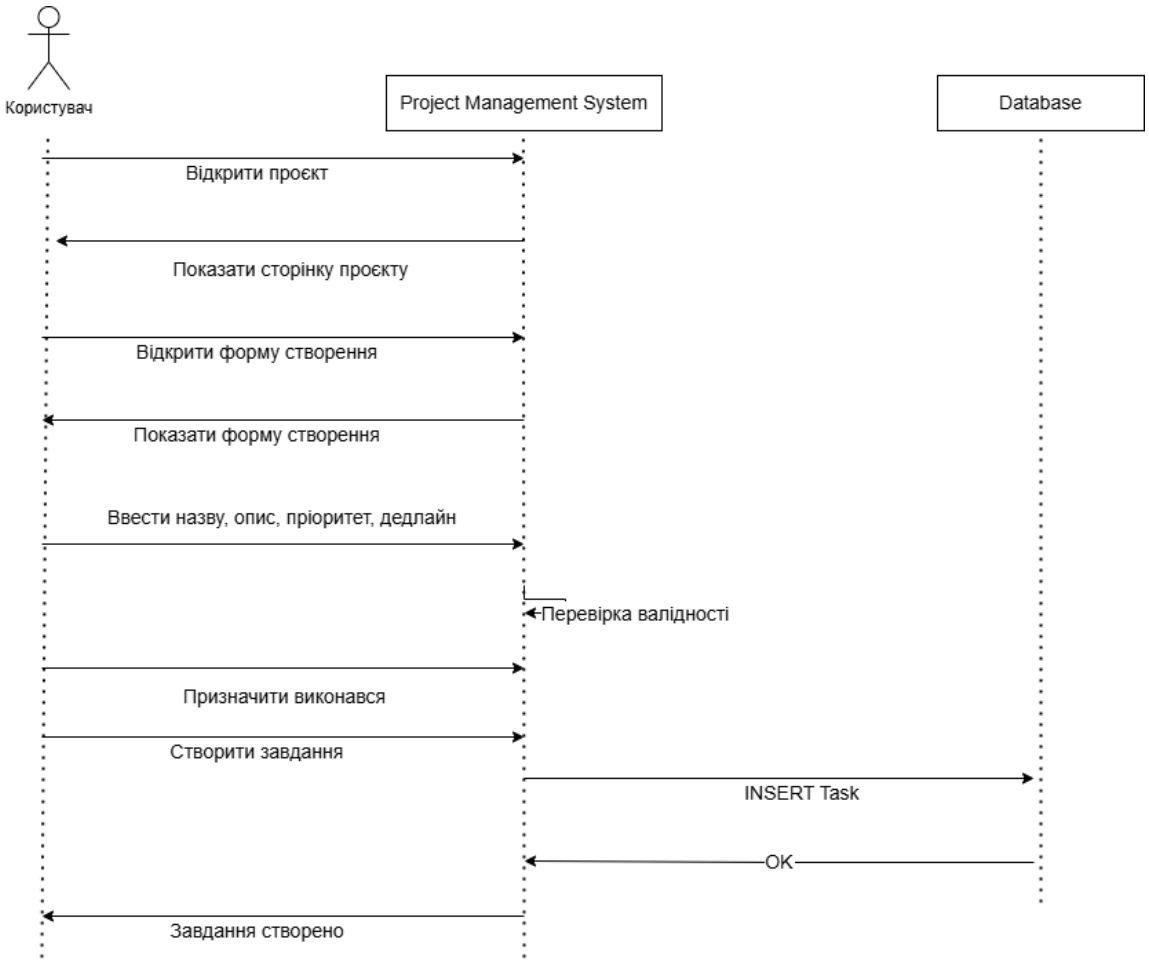


Діаграма демонструє взаємодію користувача-менеджера із системою керування проєктами під час створення нового проєкту та подальшого збереження його у базі даних.

Користувач відкриває розділ “Мої проєкти”, після чого система повертає список уже створених проєктів. Далі менеджер ініціює процес створення нового проєкту, і система відображає відповідну форму введення. Після введення користувачем назви, опису та дедлайну система виконує перевірку коректності полів.

У разі успішної валідації система формує новий об’єкт **Project** і надсилає запит на збереження даних у базу. База даних підтверджує успішне виконання операції, після чого система повідомляє користувача про створення нового проєкту та відображає його у списку проєктів.

У випадку помилки валідації система повертає користувача до етапу введення з відповідним повідомленням.



Ця діаграма описує процес створення нового завдання в існуючому проєкті та призначення виконавця з числа учасників команди.

Спочатку користувач відкриває сторінку потрібного проєкту, після чого система відображає його дані. Далі користувач переходить до створення нового завдання, і система надає форму для введення інформації. Після заповнення назви, опису, пріоритету та дедлайну система перевіряє коректність даних.

Після успішної валідації користувач обирає виконавця зі списку учасників проєкту. Система формує новий об’єкт **Task**, зберігає його у базі даних через відповідний запит та отримує підтвердження успішної операції. Після цього нове завдання з’являється у списку задач проєкту.

Якщо користувач намагається призначити виконавця, який не входить до команди, система блокує дію та повідомляє про помилку.

Вихідні коди системи

Class ProjectController:

ProjectController.java

```java
@Controller  new *
@RequestMapping("/projects")
@RequiredArgsConstructor
public class ProjectController {

    private final ProjectService projectService;

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

    @GetMapping("/create")  new *
    public String createProjectForm(Model model) {
        model.addAttribute( attributeName: "project", new Project ());
        return "projects/create";
    }

    @PostMapping("/create")  new *
    public String createProject(@ModelAttribute Project project) {
        projectService.createProject(project);
        return "redirect:/projects";
    }

    @GetMapping("/{id}")  new *
    public String projectPage(@PathVariable Long id, Model model) {
        model.addAttribute( attributeName: "project", projectService.getProjectById(id));
        return "projects/view";
    }
}
```

Class ProjectService:

ProjectService.java

```java
package org.example.projectmanagementsoftware.service;

import lombok.RequiredArgsConstructor;
import org.example.projectmanagementsoftware.domain.Project;
import org.example.projectmanagementsoftware.repository.ProjectRepository;
import org.springframework.stereotype.Service;

import java.util.List;

@Service  2 usages  new *
@RequiredArgsConstructor
public class ProjectService {

    private final ProjectRepository projectRepository;

    public List<Project> getAllProjects() {  1 usage  new *
        return projectRepository.findAll();
    }

    public Project createProject(Project project) {  1 usage  new *
        return projectRepository.save(project);
    }

    public Project getProjectById(Long id) {  1 usage  new *
        return projectRepository.findById(id)
                .orElseThrow(() -> new RuntimeException("Project not found"));
    }
}
```

Create HTML:

create.html

```html
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<link rel="stylesheet" th:href="@{/css/styles.css}">

<head>
    <title>Створити проєкт</title>
</head>

<body>

<h1>Створити проєкт</h1>

<form th:action="@{/projects/create}" th:object="${project}" method="post">

    <label>Назва:</label>
    <input type="text" th:field="*{name}"/><br/>

    <label>Опис:</label>
    <textarea th:field="*{description}"></textarea><br/>

    <label>Дедлайн:</label>
    <input type="date" th:field="*{deadline}"/><br/>

    <button type="submit">Створити</button>
</form>

</body>
</html>
```

List HTML:

list.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<link rel="stylesheet" th:href="@{/css/styles.css}">

<head>
    <title>Мої проєкти</title>
</head>

<body>
<h1>Мої проєкти</h1>

<a href="/projects/create">Створити новий проєкт</a>

<ul>
    <li th:each="p : ${projects}">
        <a th:href="@{|/projects/${p.id}|}" th:text="${p.name}"></a>
    </li>
</ul>

</body>
</html>
```

View HTML:

view.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<link rel="stylesheet" th:href="@{/css/styles.css}">

<head>
    <title th:text="${project.name}"></title>
</head>

<body>

<h1 th:text="${project.name}"></h1>

<p><b>Опис:</b> <span th:text="${project.description}"></span></p>
<p><b>Дедлайн:</b> <span th:text="${project.deadline}"></span></p>

<a th:href="@{|/tasks/create/${project.id}|}">Створити нове завдання</a>

</body>
</html>
```

Висновок

У ході виконання лабораторної роботи було виконано проєктування архітектури системи управління проєктами за допомогою UML-нотацій. Створено діаграму компонентів, яка визначає логічну структуру та залежності між модулями системи, а також діаграму розгортання, що відображає фізичне розміщення програмних елементів у середовищі виконання. На основі сценаріїв варіантів використання побудовано діаграми послідовностей, які демонструють порядок взаємодії між користувачем, інтерфейсом та внутрішніми сервісами системи.

Також було доопрацьовано програмну частину: реалізовано повний цикл роботи з даними — від введення на UI до збереження в базу даних та подальшого відображення. Створені візуальні форми та інтеграція з базою даних дозволили сформувати завершений функціональний модуль системи.

Контрольні запитання

1. Що собою становить діаграма розгортання?

Це UML-діаграма, яка показує, на якому обладнанні запускається система: сервери, компи, бази даних і як вони між собою з’єднані.

2. Які бувають види вузлів на діаграмі розгортання?

- апаратні вузли (сервер, ПК, телефон);
- віртуальні/програмні вузли (віртуальна машина, контейнер, середовище виконання).

3. Які бувають зв’язки на діаграмі розгортання?

Комунікаційні зв’язки — показують, як вузли обмінюються даними.

4. Які елементи присутні на діаграмі компонентів?

- компоненти (модулі системи),
- інтерфейси,
- залежності,
- артефакти.

5. Що становлять собою зв’язки на діаграмі компонентів?

Вони показують залежності між компонентами: хто використовує чий інтерфейс, хто від кого залежить.

6. Які бувають види діаграм взаємодії?

Основні:

- діаграма послідовностей,
- діаграма співробітництва (комунікацій),

7. Для чого призначена діаграма послідовностей?

Щоб показати порядок викликів між об’єктами: хто кому що відправляє і в якій послідовності.

8. Які ключові елементи можуть бути на діаграмі послідовностей?

- об’єкти / актори,
- життєві лінії,
- повідомлення (виклики методів),
- активації,
- блоки умов/циклів (alt, opt, loop).

9. Як діаграми послідовностей пов’язані з діаграмами варіантів використання?

Вони деталізують один конкретний use-case, показуючи покроково, як цей сценарій виконується в системі.

10. Як діаграми послідовностей пов’язані з діаграмами класів?

Вони демонструють, як об’єкти цих класів взаємодіють у реальному сценарії. Тобто класи дають структуру, а послідовність показує їхню поведінку.