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 «Технології розроблення програмного забезпечення»

Тема: «Основи проектування розгортання»

Виконала:
студентка групи ІА-32
Глущенко Анастасія
Сергіївна

Перевірив:
Мягкий Михайло
Юрійович

Київ 2025

Вступ

Метою роботи 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

Було обрано тему:
Installer generator (iterator, builder, factory method, bridge, interpreter, client-server)
Генератор інсталяційних пакетів повинен мати якийсь спосіб налаштування файлів, що входять в установку, установки вікон з інтерактивними можливостями (галочка – створити ярлик на робочому столі; ввести в текстове поле деякі дані, наприклад, ліцензійний ключ і т.д.). Генератор повинен вивести один файл .exe або .msi

Теоретичні відомості

UML (Unified Modeling Language / Уніфікована мова моделювання) — це стандартизована графічна мова, яка використовується в інженерії програмного забезпечення для візуалізації, специфікації, конструювання та документування програмних систем. UML надає набір стандартних діаграм, які допомагають розробникам, аналітикам та замовникам однозначно розуміти та описувати структуру й поведінку системи.

Діаграми UML поділяються на дві основні категорії:

1. **Структурні діаграми (Structural Diagrams):** Описують *статику* системи. Вони показують компоненти системи та зв'язки між ними. До них належать діаграми класів, компонентів, розгортання, об'єктів та ін.
2. **Поведінкові діаграми (Behavioral Diagrams):** Описують *динаміку* системи. Вони показують, як система поводиться з часом та як її елементи взаємодіють. До них належать діаграми послідовностей, діяльності, станів, прецедентів та ін.

Розглянемо детальніше діаграми, зазначені у вашому завданні.

2. Діаграма розгортання (Deployment Diagram)

Діаграма розгортання належить до **структурних** діаграм UML.

Призначення

Діаграма розгортання моделює фізичну архітектуру системи. Вона показує, як програмні компоненти (артефакти) розподіляються по фізичних пристроях (вузлах) і як ці пристрої з'єднані між собою. Це єдина діаграма UML, що фокусується на апаратному забезпеченні.

Основні елементи

- **Вузол (Node):**
    - Це обчислювальний ресурс, зазвичай апаратний пристрій, на якому виконується програмне забезпечення.

    - Приклади: сервер додатків, сервер бази даних, клієнтський комп'ютер, мобільний пристрій, вбудований контролер.
    - Вузли можуть бути вкладеними (наприклад, сервер віртуалізації, що містить віртуальні машини).
- **Артефакт (Artifact):**
    - Це конкретний програмний продукт, результат процесу розробки, який розгортається на вузлі.
    - Приклади: виконуваний .jar або .war файл, бібліотека .dll, скрипт, схема бази даних, файл конфігурації.
- **Зв'язок (Communication Path):**
    - Лінія, що з'єднує два вузли. Вона показує, що між ними існує канал зв'язку.
    - Часто над зв'язком вказують протокол комунікації (наприклад, HTTP/S, TCP/IP, JDBC, RMI).
- **Залежність розгортання (Deployment Dependency):**
    - Пунктирна стрілка від артефакту до вузла (або всередині вузла), що показує, який артефакт розгортається на якому вузлі.

Коли використовується?

- Для планування фізичної інфраструктури проєкту.
- Для візуалізації, де саме буде виконуватись кожна частина програми.
- Для аналізу мережевих вимог та вузьких місць продуктивності.

3. Діаграма компонентів (Component Diagram)

Діаграма компонентів також належить до **структурних** діаграм UML.

Призначення

Діаграма компонентів описує логічну архітектуру системи, розбиваючи її на модульні частини — **компоненти**. Вона показує організацію компонентів та залежності між ними. Ця діаграма допомагає проектувати системи з низькою зв'язаністю (low coupling) та високою згуртованістю (high cohesion).

Основні елементи

- **Компонент (Component):**
  - Логічно відокремлена, модульна частина системи, яка інкапсулює свою поведінку та дані й може бути замінена незалежно від інших.
  - Приклади: Сервіс оплати, Модуль аутентифікації, Веб-інтерфейс, API-шлюз.
- **Інтерфейс (Interface):**
  - Набір операцій, які компонент надає або вимагає.
  - **Наданий інтерфейс (Provided Interface):** Позначається колом (lollipop notation). Це сервіси, які компонент пропонує іншим.
  - **Необхідний інтерфейс (Required Interface):** Позначається півколом (socket notation). Це сервіси, які потрібні компоненту від інших для своєї роботи.
- **Порт (Port):**
  - Точка взаємодії між компонентом та його оточенням (іншими компонентами). Інтерфейси часто "прикріплені" до портів.
- **Залежність (Dependency):**
  - Пунктирна стрілка, що показує, що один компонент (клієнт) залежить від іншого (постачальника).

Коли використовується?

- На етапі архітектурного проектування системи.
- Для визначення меж модулів та їхніх точок взаємодії.
- Для планування повторного використання компонентів.

4. Діаграма послідовностей (Sequence Diagram)

Діаграма послідовностей належить до **поведінкових** діаграм (або, точніше, до підтипу "Діаграми взаємодії").

Призначення

Діаграма послідовностей моделює взаємодію об'єктів у часі. Вона детально показує, в якому **порядку** об'єкти обмінюються повідомленнями (наприклад, викликають методи один одного) для виконання конкретного сценарію (use case). Час на цій діаграмі йде згори вниз.

Основні елементи

- **Актор (Actor):**
    - Зовнішня сутність (користувач або інша система), яка ініціює потік подій. Позначається фігуркою людини.
- **Лінія життя (Lifeline):**
    - Вертикальна пунктирна лінія, що представляє одного учасника (об'єкт або клас) протягом певного часу. Зверху лінії життя знаходиться прямокутник з назвою учасника.
- **Фокус контролю (Activation Bar):**
    - Вузький прямокутник на лінії життя. Він показує період, протягом якого об'єкт активний (тобто виконує певну операцію або очікує відповіді).
- **Повідомлення (Message):**
    - Горизонтальна стрілка між лініями життя, що позначає комунікацію.
    - **Синхронне повідомлення (Synchronous):** Суцільна лінія із зафарбованою стрілкою. Відправник відправляє повідомлення і *чекає* на відповідь.
    - **Асинхронне повідомлення (Asynchronous):** Суцільна лінія з відкритою стрілкою. Відправник відправляє повідомлення і *не чекає* відповіді, продовжуючи свою роботу.
    - **Повідомлення-відповідь (Reply):** Пунктирна стрілка. Показує повернення результату від синхронного повідомлення.
- **Фрагменти взаємодії (Interaction Fragments):**

- Рамки, що дозволяють моделювати складну логіку:
  - **alt (Alternative):** Для моделювання блоків if-then-else.
  - **loop (Loop):** Для моделювання циклів.
  - **opt (Optional):** Для моделювання блоку, який виконується лише за певної умови.

Коли використовується?

- Для деталізації логіки одного прецеденту (Use Case).
- Для розуміння, як об'єкти співпрацюють для досягнення мети.
- Для виявлення вузьких місць або логічних помилок у взаємодії.

Хід роботи

1. Діаграма розгортання системи

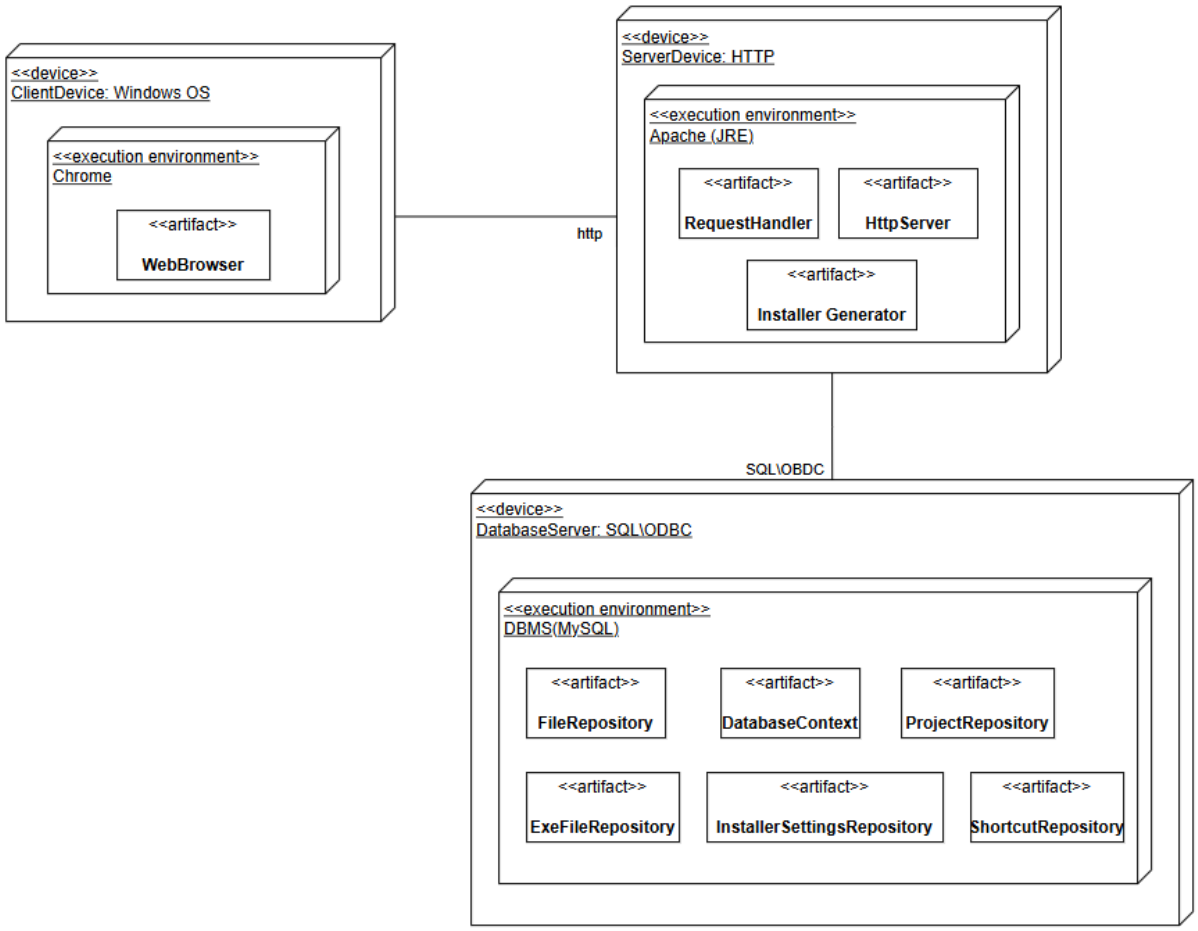

Рисунок 1 - Діаграма розгортання системи

1. **ClientDevice: Windows OS**
   - Це пристрій користувача (комп’ютер або ноутбук), на якому працює операційна система Windows.
   - Усередині пристрою виконується середовище **Chrome** (<<execution environment>>), яке забезпечує взаємодію користувача з веб-застосунком.
   - Артефакт **WebBrowser** відповідає за відображення інтерфейсу користувача та відправлення HTTP-запитів до серверу.
2. **ServerDevice: HTTP**
   - Це серверна машина, на якій працює серверна частина системи.

- Вона функціонує в середовищі **Apache (JRE)**, що надає можливість виконання Java-додатків.
- Основні артефакти:
    - **HttpServer** — компонент, який приймає HTTP-запити від клієнта і формує відповіді.
    - **RequestHandler** — обробляє запити користувача та забезпечує зв’язок між інтерфейсом і бізнес-логікою.
    - **Installer Generator** — основний модуль застосунку, який відповідає за створення проєктів, додавання файлів, налаштування інсталятора та управління даними.

3. **DatabaseServer: SQL/ODBC**
    - Це сервер бази даних, який забезпечує збереження інформації про проєкти, файли та налаштування інсталятора.
    - Середовище виконання — **DBMS (MySQL)** або аналогічна СУБД.
    - Основні артефакти:
        - **DatabaseContext** — шар взаємодії між сервером і базою даних.
        - **ProjectRepository**, **FileRepository**, **ExeFileRepository**, **InstallerSettingsRepository**, **ShortcutRepository** — модулі для виконання CRUD-операцій із відповідними таблицями бази даних.
4. **Зв’язки між пристроями:**
    - **ClientDevice ↔ ServerDevice** — обмін даними через протокол **HTTP**, що забезпечує відправлення запитів із браузера та отримання відповідей із сервера.
    - **ServerDevice ↔ DatabaseServer** — взаємодія через **SQL/ODBC**, яка дозволяє серверу виконувати операції читання, запису та оновлення даних у базі.

2. Діаграма компонентів

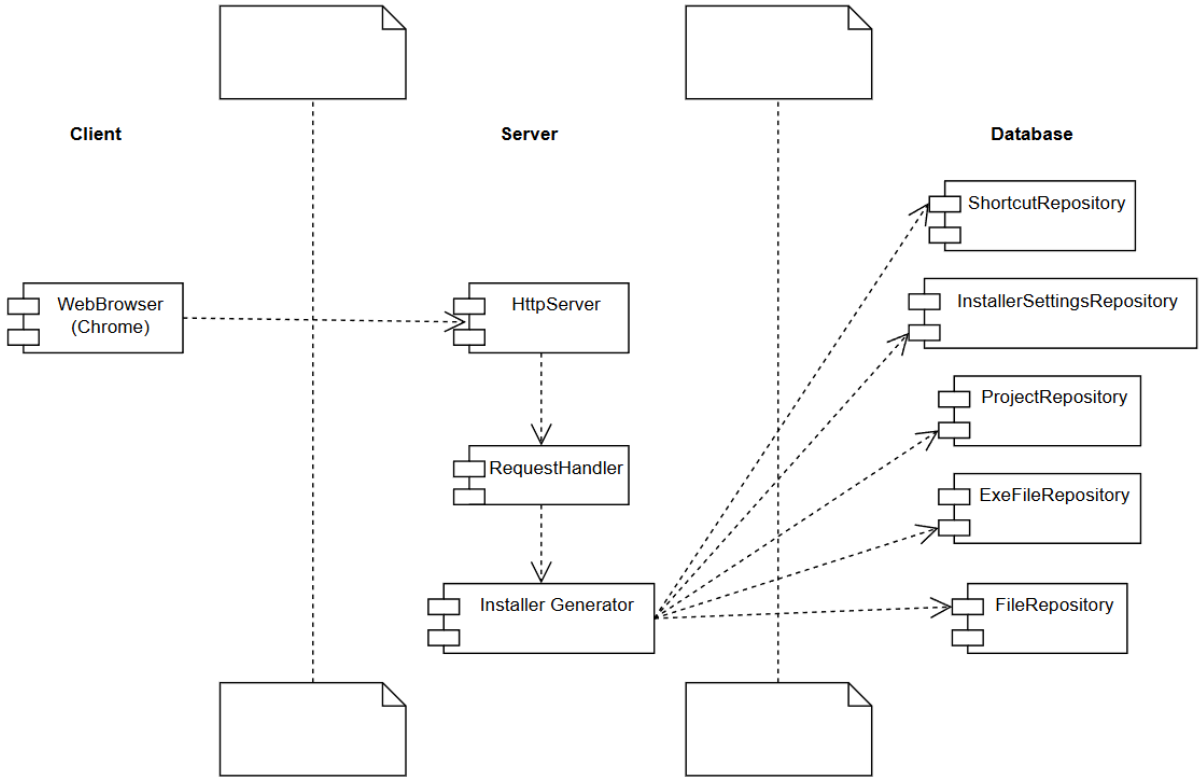

Рисунок 2 – Діаграма компонентів

1. **Клієнт: WebBrowser (Chrome)**
   - Це компонент користувацької частини системи, яка працює у веб-браузері.
   - Користувач взаємодіє з інтерфейсом через веб-браузер Chrome, відправляючи HTTP-запити на сервер.
   - Інтерфейс дозволяє користувачу вводити дані, створювати проєкти, додавати файли та налаштовувати інсталяційні параметри.
2. **Сервер: HttpServer**
   - Це сервер, що приймає HTTP-запити від клієнта і обробляє їх.
   - HttpServer виступає основним компонентом, що обробляє запити від браузера користувача і передає їх для подальшої обробки іншим компонентам.
3. **Обробник запитів: RequestHandler**

- Це компонент, що відповідає за обробку HTTP-запитів, що надходять від HttpServer.
- Він виконує бізнес-логіку, координує запити до Installer Generator для створення нових проєктів, додавання файлів і налаштування інсталятора.

4. **Генератор інсталятора: Installer Generator**
   - Це основний компонент серверної частини системи, що відповідає за створення інсталяційних файлів для проєктів.
   - Він виконує процес генерації файлів, включаючи додавання файлів до проєктів і налаштування параметрів інсталяції.
5. **База даних: SQL/ODBC**
   - Це сервер бази даних, що зберігає всю інформацію про проєкти, файли та налаштування інсталятора.
   - Взаємодія з базою даних здійснюється через SQL/ODBC.
6. **Репозиторії бази даних:**
   - ShortcutRepository: відповідає за збереження інформації про ярлики для інсталяційних файлів.
   - InstallerSettingsRepository: зберігає налаштування інсталятора для кожного проєкту.
   - ProjectRepository: містить інформацію про проєкти.
   - ExeFileRepository: зберігає файли типу .exe для генерації інсталятора.
   - FileRepository: зберігає інші типи файлів, пов'язані з проєктами.

3. Діаграми послідовностей

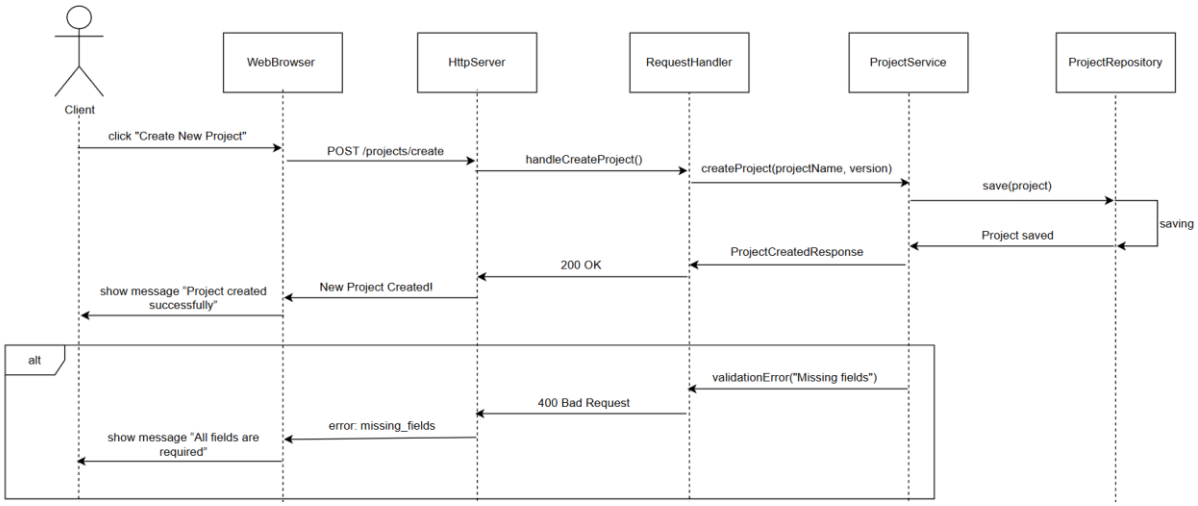

Рисунок 3 – Діаграма послідовностей для сценраію «Створення нового проєкту»

Опис подій:

1. **Крок 1: Користувач натискає кнопку "Create New Project"**
   Користувач ініціює процес створення нового проекту, натискаючи кнопку "Create New Project" на веб-сторінці в браузері.
2. **Крок 2: Клієнт (WebBrowser) відправляє запит на сервер**
   Веб-браузер відправляє POST запит на сервер за адресою /projects/create, передаючи дані нового проекту (наприклад, projectName та version).
3. **Крок 3: Сервер (HttpServer) приймає запит**
   Сервер приймає запит від клієнта і передає його до обробки в RequestHandler, який займається обробкою запитів і надає відповідь.
4. **Крок 4: Обробка запиту в RequestHandler**
   RequestHandler приймає запит і передає його для обробки методу createProject в ProjectService.
5. **Крок 5: Створення нового проекту в ProjectService**
   Метод createProject в ProjectService отримує параметри projectName та version і створює новий об'єкт Project.

Після створення проекту, інформація зберігається в базі даних через ProjectRepository.

6. **Крок 6: Збереження проекту в базі даних через ProjectRepository**

   ProjectRepository здійснює операцію збереження проекту в базі даних.

7. **Крок 7: Успішний результат**

   Після успішного створення проекту в базі даних, сервер повертає код відповіді 200 OK з повідомленням про успішне створення проекту.

8. **Крок 8: Помилка через відсутні поля**

   Якщо деякі обов'язкові поля не були заповнені (наприклад, projectName або version), сервер поверне помилку 400 (Bad Request), і на клієнтській стороні відобразиться повідомлення про відсутність необхідних полів.

9. **Крок 9: Відповідь на клієнтській стороні**

   Якщо проект створено успішно, веб-сторінка відобразить повідомлення "Project created successfully".

   Якщо виявлена помилка (наприклад, незаповнені поля), на екрані відобразиться повідомлення "All fields are required".

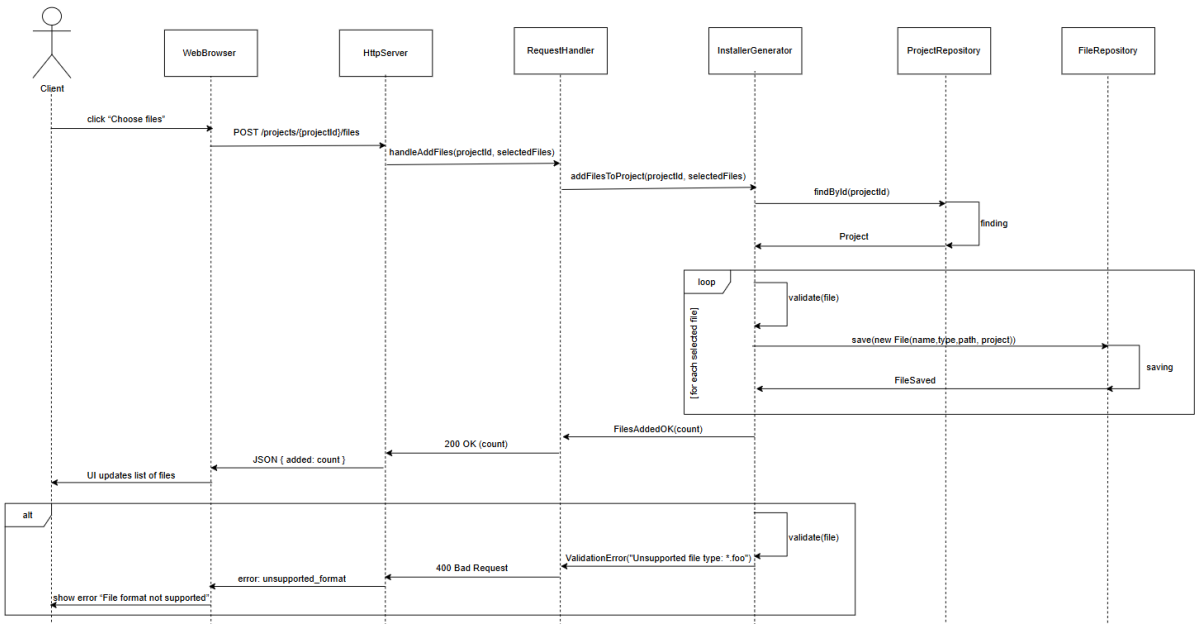

Рисунок 4 – Діаграма послідовностей для сценраію «Вибір файлів для інсталяції»

Опис подій:

1. **Крок 1: Користувач вибирає файли для інсталяції**

   Користувач натискає кнопку "Choose File" в інтерфейсі веб-браузера.

2. **Крок 2: Клієнт (WebBrowser)**:

   Веб-браузер відправляє HTTP POST запит на сервер з вибраними файлами.

   Запит має такий формат: POST /projects/{id}/addFiles, передаючи файл(и) через multipart/form-data.

3. **Крок 3: Обробка запиту сервером (HttpServer)**

   Запит приймається HttpServer, який виступає посередником для обробки HTTP-запитів.

4. **Крок 4: Обробка запиту в RequestHandler**

   RequestHandler приймає запит від HttpServer та направляє його до InstallGenerator для подальшої обробки. Запит передається для обробки і додавання файлів до конкретного проекту.

5. **Крок 5: InstallGenerator**:

   InstallGenerator отримує дані з запиту та додає вибрані файли до поточного проекту.

   Файли перевіряються на коректність типу і, у разі потреби, зберігаються у вказаному місці.

6. **Крок 6: Додавання файлів до проекту**

   Для кожного файлу виконується збереження в базі даних через **FileRepository**, включаючи збереження шляху, типу файлу та інформації про сам файл.

7. **Крок 7: Збереження файлів**

   Файли додаються до відповідного проекту за допомогою методу addFile(), який додає файл до проекту в базі даних. Потім проект зберігається через ProjectRepository.

8. **Крок 8: Реакція серверу**

Після успішного додавання файлів, сервер повертає статус 200 OK, підтверджуючи успішне виконання запиту.

Клієнт отримує відповідь у форматі JSON з кількістю доданих файлів.

9. **Крок 9: Повідомлення про помилку (якщо файл невірний)**

   Якщо тип файлу неправильний (наприклад, не підтримуваний), сервер повертає помилку 400 (Bad Request) із повідомленням про неправильний тип файлу.

10. **Крок 10: Оновлення інтерфейсу користувача**

    У разі успішного завантаження файлів інтерфейс користувача оновлюється з новими файлами, які тепер відображаються в списку файлів проекту.

4. Код програми

HTML файли

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Create Project</title>
</head>
<body>
<h1>Create a New Project</h1>

<div th:if="${message}" style="color: green;">
    <p th:text="${message}"></p>
</div>

<form th:action="@{/createProject}" method="POST">
    <div>
        <label for="projectName">Project Name:</label>
        <input type="text" id="projectName" name="projectName"
placeholder="Enter Project Name" required />
    </div>
    <div>
        <label for="version">Version:</label>
        <input type="text" id="version" name="version" placeholder="Enter
Version (e.g., 1.0.0)" required />
    </div>
    <button type="submit">Create Project</button>
    <p><a th:href="@{/projects}">View projects list</a></p>
</form>
</body>
</html>
```

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head><title>Project details</title></head>
<body>
<p><a th:href="@{/projects}">← Back to list</a></p>

<h2>Project details</h2>
<ul>
  <li><b>ID:</b> <span th:text="${project.id}"></span></li>
  <li><b>Name:</b> <span th:text="${project.projectName}"></span></li>
  <li><b>Version:</b> <span th:text="${project.version}"></span></li>
</ul>

<h3>Installer settings</h3>
<div th:if="${project.installerSettings == null}">
  <i>Not set yet</i>
  <form th:action="@{|/projects/${project.id}/setInstallerSettings|}"
method="post">
    <label for="installPath">Install Path:</label>
    <input type="text" id="installPath" name="installPath"
placeholder="Enter Install Path" required />
    <br />
    <label for="createDesktopShortcut">Create Desktop Shortcut:</label>
    <input type="checkbox" id="createDesktopShortcut"
name="createDesktopShortcut" />
    <br />
    <label for="language">Language:</label>
    <select id="language" name="language">
      <option value="English">English</option>
      <option value="Spanish">Spanish</option>
      <option value="French">French</option>
    </select>
    <br />
    <button type="submit">Save Settings</button>
  </form>
</div>

<div th:if="${project.installerSettings != null}">
  <ul>
    <li th:text="'Path: ' + ${project.installerSettings.installPath}"></li>
    <li th:text="'Desktop shortcut: ' +
${project.installerSettings.createDesktopShortcut}"></li>
    <li th:text="'Language: ' +
${project.installerSettings.language}"></li>
  </ul>
</div>

<h3>Files</h3>
<div th:if="${#lists.isEmpty(project.files)}">
  <i>No files yet</i>
</div>
<ul th:if="${!#lists.isEmpty(project.files)}">
  <li th:each="f : ${project.files}"
      th:text="${f.fileName} + ' (' + ${f.fileType} + ') - ' +
${f.path}"></li>
</ul>

<h3>Files for Installation</h3>
<form th:action="@{|/projects/${project.id}/addFiles|}" method="post"
enctype="multipart/form-data">
  <label for="files">Select Files:</label>
```

```
  <input type="file" id="files" name="files" multiple />
  <button type="submit">Add Files</button>
</form>

</body>
</html>
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head><title>Projects</title></head>
<body>
<h1>Projects</h1>

<div th:if="${message}" style="color:green" th:text="${message}"></div>

<p><a th:href="@{/createProject}">+ Create new project</a></p>

<table border="1" cellpadding="6" cellspacing="0">
    <thead>
    <tr>
        <th>ID</th>
        <th>Name</th>
        <th>Version</th>
        <th>Actions</th>
    </tr>
    </thead>
    <tbody>
    <tr th:each="p : ${projects}">
        <td th:text="${p.id}"></td>
        <td th:text="${p.projectName}"></td>
        <td th:text="${p.version}"></td>
        <td>
            <a th:href="@{|/projects/${p.id}|}">Details</a>
            <form th:action="@{|/projects/${p.id}/delete|}" method="post"
style="display:inline">
                <button type="submit" onclick="return confirm('Delete this
project?')">Delete</button>
            </form>
        </td>
    </tr>
    </tbody>
</table>

</body>
</html>
```

ProjectController.java

```
package com.example.installer.controller;

import java.nio.file.Path;
import java.nio.file.Paths;
import java.nio.file.Files;
import java.util.Set;
import java.nio.file.StandardCopyOption;
import java.io.IOException;
import com.example.installer.Project;
import com.example.installer.File;
import com.example.installer.InstallerSettings;
import org.springframework.web.multipart.MultipartFile;
import com.example.installer.repository.ProjectRepository;
```

```java
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.servlet.mvc.support.RedirectAttributes;

import java.util.Optional;

@Controller
public class ProjectController {

    @Autowired
    private ProjectRepository projectRepository;

    @GetMapping("/createProject")
    public String showCreateProjectForm() {
        return "createProject";
    }

    @PostMapping("/createProject")
    public String submitProject(@RequestParam String projectName,
                                @RequestParam String version,
                                RedirectAttributes redirectAttributes) {
        Project p = new Project();
        p.setProjectName(projectName);
        p.setVersion(version);
        projectRepository.save(p);

        redirectAttributes.addFlashAttribute("message",
                "Проєкт '" + projectName + "' успішно створено!");
        return "redirect:/createProject";
    }

    @GetMapping("/projects")
    public String listProjects(Model model) {
        model.addAttribute("projects", projectRepository.findAll());
        return "projects";
    }

    @GetMapping("/projects/{id}")
    public String projectDetails(@PathVariable Long id, Model model,
                                 RedirectAttributes ra) {
        Optional<Project> opt = projectRepository.findById(id);
        if (opt.isEmpty()) {
            ra.addFlashAttribute("message", "Проєкт з id=" + id + " не
знайдено.");
            return "redirect:/projects";
        }

        Project project = opt.get();

        model.addAttribute("project", project);
        return "project-details";
    }


    @PostMapping("/projects/{id}/delete")
    public String deleteProject(@PathVariable Long id, RedirectAttributes
ra) {
        projectRepository.deleteById(id);
        ra.addFlashAttribute("message", "Проєкт видалено.");
```

```
        return "redirect:/projects";
    }

    @PostMapping("/projects/{id}/setInstallerSettings")
    public String setInstallerSettings(@PathVariable Long id,
                                       @RequestParam String installPath,
                                       @RequestParam(required = false,
defaultValue = "false") boolean createDesktopShortcut,
                                       @RequestParam String language,
                                       RedirectAttributes
redirectAttributes) {
        Optional<Project> projectOpt = projectRepository.findById(id);
        if (projectOpt.isEmpty()) {
            redirectAttributes.addFlashAttribute("message", "Project not
found");
            return "redirect:/projects";
        }

        Project project = projectOpt.get();

        if (project.getInstallerSettings() == null) {
            InstallerSettings newInstallerSettings = new
InstallerSettings(installPath, createDesktopShortcut, language);
            newInstallerSettings.setProject(project);
            project.setInstallerSettings(newInstallerSettings);
        } else {
            project.setInstallerSettings(installPath,
createDesktopShortcut, language);
        }

        projectRepository.save(project);
        redirectAttributes.addFlashAttribute("message", "Installer settings
updated for project " + project.getProjectName());
        return "redirect:/projects/" + id;
    }







    @PostMapping("/projects/{id}/addFiles")
    public String addFilesToProject(@PathVariable Long id,
@RequestParam("files") MultipartFile[] files,
                                    RedirectAttributes redirectAttributes)
{
        Optional<Project> projectOpt = projectRepository.findById(id);
        if (projectOpt.isEmpty()) {
            redirectAttributes.addFlashAttribute("message", "Project not
found");
            return "redirect:/projects";
        }

        Project project = projectOpt.get();
        for (MultipartFile file : files) {
            Set<String> allowedTypes = Set.of("zip", "tar", "jar");
            String fileExtension =
file.getOriginalFilename().substring(file.getOriginalFilename().lastIndexOf
('.') + 1);

            if (!allowedTypes.contains(fileExtension)) {
```

```
                redirectAttributes.addFlashAttribute("message", "Invalid
file type: " + file.getOriginalFilename());
                return "redirect:/projects/" + id;
            }

            String uploadDir = System.getProperty("user.dir") +
"/src/main/resources/static/uploads/";
            Path path = Paths.get(uploadDir + file.getOriginalFilename());

            try {
                Files.copy(file.getInputStream(), path,
StandardCopyOption.REPLACE_EXISTING);
            } catch (IOException e) {
                redirectAttributes.addFlashAttribute("message", "Error
saving file: " + e.getMessage());
                return "redirect:/projects/" + id;
            }

            File newFile = new File();
            newFile.setFileName(file.getOriginalFilename());
            newFile.setFileType(file.getContentType());
            newFile.setPath(path.toString());
            newFile.setProject(project);
            project.addFile(newFile);
        }

        projectRepository.save(project);

        redirectAttributes.addFlashAttribute("message", "Files added
successfully to project " + project.getProjectName());
        return "redirect:/projects/" + id;
    }

}
```

ProjectService.java

```
package com.example.installer.service;

import com.example.installer.Project;
import com.example.installer.repository.ProjectRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service
public class ProjectService {

    @Autowired
    private ProjectRepository projectRepository;

    // Логіка для створення нового проекту
    public Project createNewProject(String projectName, String version) {
        Project project = new Project();
        project.setProjectName(projectName);
        project.setVersion(version);

        return projectRepository.save(project);
    }
}
```

Висновки

Цей проект успішно реалізує архітектуру з трьох основних компонентів: клієнтський пристрій, сервер і база даних. На клієнтському пристрої використовується веб-браузер для взаємодії з сервером через HTTP-запити, що обробляються сервером за допомогою HttpServer та RequestHandler. Основний функціонал, зокрема генерація установчих файлів, реалізується в Installer Generator. Сервер взаємодіє з базою даних, зберігаючи інформацію про проекти, файли та налаштування інсталятора.

Реалізовано ключові функції, такі як створення проектів, додавання файлів для інсталяції, а також налаштування інсталятора. Користувач може заповнювати форму для проекту, додавати файли та налаштовувати параметри інсталяції, які зберігаються в базі даних. Веб-форми на клієнтському боці передають дані на сервер, де вони обробляються і зберігаються.

Діаграми розгортання, компонентів і послідовностей наочно демонструють взаємодію між компонентами системи, і всі необхідні функції реалізовані згідно з вимогами. Реалізований код ефективно підтримує процеси створення проектів та управління файлами, забезпечуючи інтеграцію з базою даних і відображенням даних на UI.

Контрольні запитання

1. Що собою становить діаграма розгортання?

Діаграма розгортання (Deployment Diagram) у UML показує фізичне розміщення апаратних вузлів (серверів, клієнтів) і компонентів системи на цих вузлах. Вона відображає, як програмне забезпечення реалізується на апаратних елементах.

2. Які бувають види вузлів на діаграмі розгортання?

Фізичні вузли (Node): реальні апаратні пристрої (сервер, комп'ютер, мобільний пристрій).

Вузли виконання (Execution Environment): середовище, у якому запускаються компоненти (операційна система, віртуальна машина, контейнер).

3. Які бувають зв'язки на діаграмі розгортання?

Ассоціація між вузлами (Communication Path): показує можливість обміну даними між вузлами.

Залежність (Dependency): вказує, що один вузол або компонент залежить від іншого.

4. Які елементи присутні на діаграмі компонентів?

• Компоненти (Component): самостійні частини ПЗ.

• Інтерфейси (Interface): порти, через які компоненти взаємодіють.

• Пакети (Package): групування компонентів.

• Зв'язки (Dependency, Association): взаємозв'язки між компонентами.

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

Зв'язки відображають залежності між компонентами: хто на кого покладається, хто надає чи використовує інтерфейс іншого компонента.

6. Які бувають види діаграм взаємодії?

Діаграма послідовностей (Sequence Diagram) – показує порядок повідомлень між об'єктами.

Діаграма комунікацій (Communication Diagram) – показує зв’язки між об’єктами та обмін повідомленнями.
Діаграма часу (Timing Diagram) – показує зміни станів об’єктів у часі.
Діаграма взаємодії (Interaction Overview Diagram) – поєднує елементи кількох діаграм взаємодії.
7. Для чого призначена діаграма послідовностей?
Вона описує порядок і часову послідовність взаємодії між об’єктами системи для реалізації певного сценарію чи варіанту використання.
8. Які ключові елементи можуть бути на діаграмі послідовностей?
• Об’єкти/Актори (Lifelines)
• Повідомлення (Messages) – виклики методів між об’єктами
• Активності (Activation bars) – час виконання дії об’єкта
• Події створення/знищення об’єктів
9. Як діаграми послідовностей пов’язані з діаграмами варіантів використання?
Кожна діаграма послідовностей реалізує сценарій певного варіанту використання, деталізуючи, як актори взаємодіють із системою крок за кроком.
10. Як діаграми послідовностей пов’язані з діаграмами класів?
Повідомлення на діаграмі послідовностей зазвичай відповідають методам класів, а об’єкти на діаграмі є екземплярами класів із діаграми класів.