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

Лабораторна робота № 2

з дисципліни «Технології розроблення програмного забезпечення»

Тема: «Основи проектування»

Виконала:
студентка групи ІА-32
Глущенко Анастасія
Сергіївна

Перевірив:
Мягкий Михайло
Юрійович

Київ 2025

Зміст

Вступ

Метою роботи є обрати зручну систему побудови UML-діаграм та навчитися будувати діаграми варіантів використання для системи що проєктується, розробляти сценарії варіантів використання та будувати діаграми класів предметної області

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

Вступ до мови UML

Мова UML є загальноцільовою мовою візуального моделювання, яка розроблена для специфікації, візуалізації, проєктування та документування компонентів програмного забезпечення, бізнес-процесів та інших систем. Мова UML є досить строгим та потужним засобом моделювання, який може бути ефективно використаний для побудови концептуальних, логічних та графічних моделей складних систем різного цільового призначення. Ця мова увібрала в себе найкращі якості та досвід методів програмної інженерії, які з успіхом використовувалися протягом останніх років при моделюванні великих та складних систем.

З погляду методології ООАП (об'єктно-орієнтованого аналізу та проєктування) досить повна модель складної системи є певною кількістю взаємопов'язаних уявлень (views), кожне з яких відображає аспект поведінки або структури системи. У цьому найзагальнішими уявленнями складної системи прийнято вважати статичне і динамічне, які у своє чергу можуть поділятися інші більш приватні.

Принцип ієрархічної побудови моделей складних систем передбачає розгляд процес побудови моделей на різних рівнях абстрагування або деталізації в рамках фіксованих уявлень.

Рівень представлення (layer) – спосіб організації та розгляду моделі на одному рівні абстракції, що представляє горизонтальний зріз архітектури моделі, тоді як розбиття представляє її вертикальний зріз. При цьому вихідна або початкова модель складної системи має найбільш загальне уявлення та відноситься до концептуального рівня. Така модель, що отримала назву концептуальної, будується на початковому етапі проєктування і може не містити багатьох деталей та аспектів системи, що моделюється.

Наступні моделі конкретизують концептуальну модель, доповнюючи її уявленнями логічного та фізичного рівня.

Загалом процес ООАП можна розглядати як послідовний перехід від розробки найбільш загальних моделей та уявлень концептуального рівня до більш приватних і детальних уявлень логічного та фізичного рівня. У цьому кожному етапі ООАП дані моделі послідовно доповнюються дедалі більше деталей, що дозволяє їм адекватно відбивати різні аспекти конкретної реалізації складної системи.

В рамках мови UML уявлення про модель складної системи фіксуються у вигляді спеціальних графічних конструкцій, що отримали назву діаграм. Діаграма (diagram) – графічне уявлення сукупності елементів моделі у формі зв'язкового графа, вершинам і ребрам (дугам) якого приписується певна семантика. Нотація канонічних діаграм є основним засобом розробки моделей мовою UML.

У нотації мови UML визначено такі види діаграм:

• варіантів використання (use case diagram);
• класів (class diagram);
• кооперації (collaboration diagram);
• послідовності (sequence diagram);
• станів (statechart diagram);
• діяльності (activity diagram);
• компонентів (component diagram);
• розгортання (deployment diagram).

Перелічені діаграми є невід'ємною частиною графічної нотації мови UML. Понад те, процес ООАП нерозривно пов'язаний з процесом побудови цих діаграм. При цьому сукупність побудованих таким чином діаграм є самодостатньою в тому сенсі, що в них міститься вся інформація, яка потрібна для реалізації проєкту складної системи. Кожна з цих діаграм деталізує та конкретизує різні уявлення про модель

складної системи у термінах мови UML. При цьому діаграма варіантів використання являє собою найбільш загальну концептуальну модель складної системи, яка є вихідною для побудови інших діаграм. Діаграма класів, за своєю суттю, логічна модель, що відбиває статичні аспекти структурної побудови складної системи.

Діаграми кооперації та послідовностей є різновидами логічної моделі, які відображають динамічні аспекти функціонування складної системи. Діаграми станів та діяльності призначені для моделювання поведінки системи. І, нарешті, діаграми компонентів і розгортання служать уявлення фізичних компонентів складної системи і тому представляють її фізичну модель.

Діаграма варіантів використання (Use-Cases Diagram)

Діаграма варіантів використання — це графічне зображення, яке описує взаємодію між користувачами (акторами) і системою, фокусуючись на функціональності, яку система надає користувачам. Кожен варіант використання (use case) описує одну задачу або функцію, яку система повинна виконати. Ці діаграми використовуються для візуалізації вимог та взаємодії з користувачами на етапі розробки системи.

Діаграми класів

Діаграма класів є основною діаграмою об'єктно-орієнтованого моделювання і описує структуру системи через класи об'єктів, їхні атрибути, методи та зв'язки між класами. Вона дозволяє побудувати модель даних і відобразити, як об'єкти взаємодіють між собою в межах системи.

Логічна структура бази даних

Логічна структура бази даних — це спосіб організації даних у базі для забезпечення їх ефективного зберігання та доступу. Вона включає в себе визначення таблиць, їхніх зв'язків, обмежень та індексів. Логічна структура

бази даних відображає, як інформація буде зберігатися і взаємодіяти всередині бази даних, не враховуючи фізичні аспекти зберігання.

Хід роботи

1. Діаграма варіантів використання

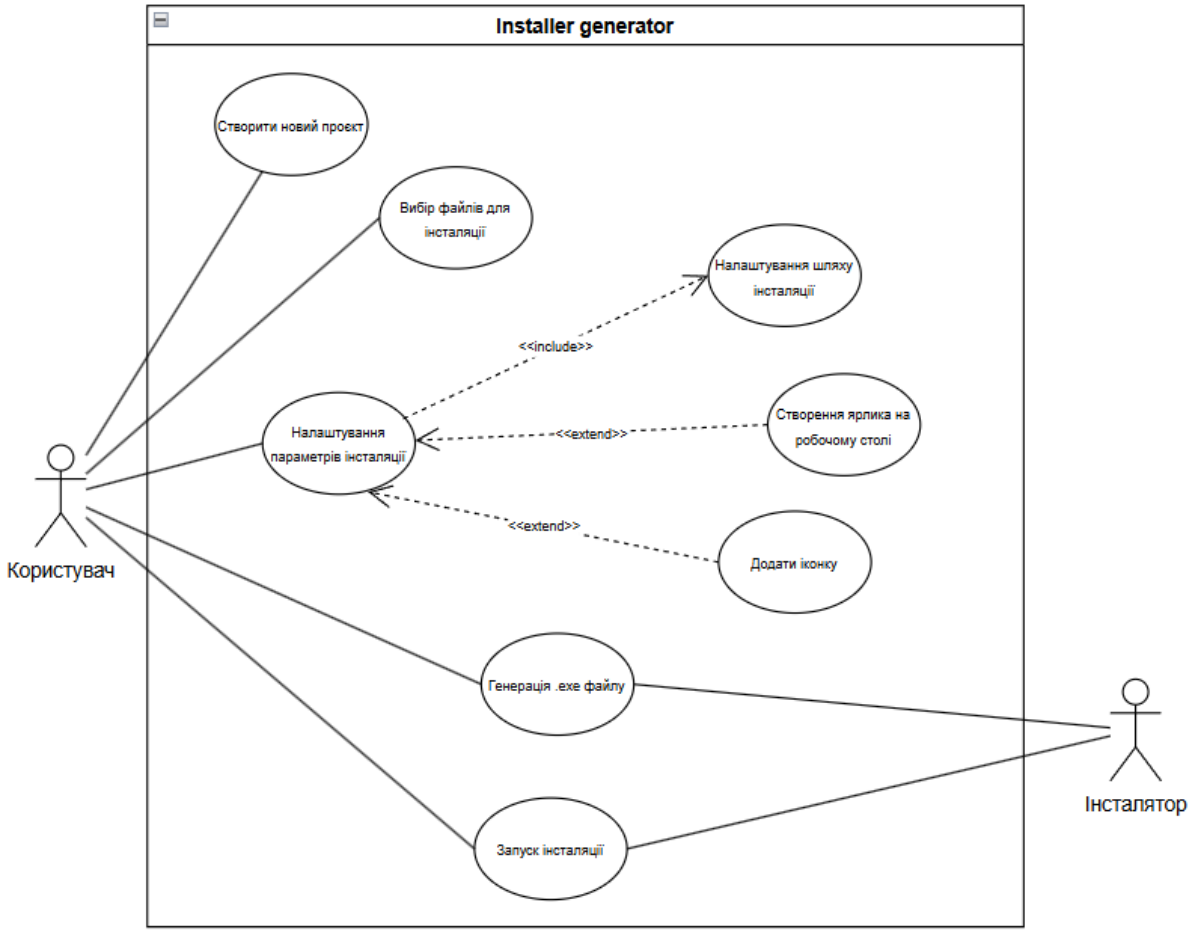

Рисунок 1 - Діаграма варіантів використання

Актори:

- Користувач: Основний актор, який взаємодіє з системою для створення файлу .exe
- Інсталятор: Відповідає за фінальні етапи процесу: генерацію .exe файлу та запуск інсталяції.

Варіанти використання:

1. Створити новий проєкт: Користувач ініціює новий інсталяційний проект, визначаючи основні параметри.

2. Вибір файлів для інсталяції: Користувач вибирає файли та папки, які повинні бути включені в інсталяційний пакет.
3. Налаштування параметрів інсталяції: Користувач налаштовує основні параметри інсталяції.
4. Налаштування шляху інсталяції: Користувач визначає шлях для встановлення файлу на комп'ютері.
5. Створення ярлика на робочому столі: Користувач має можливість вибрати, чи створювати ярлик на робочому столі для зручного доступу до програми.
6. Додавання іконки: Користувач може вибрати, яку іконку додати до ярлика на робочому столі.
7. Генерація .exe файлу: Після завершення налаштувань користувач натискає кнопку для створення інсталяційного пакету у вигляді .exe файлу.
8. Запуск інсталяції: Після генерації інсталяційного файлу .exe, користувач запускає процес інсталяції.

Відносини між варіантами використання:

- Налаштування параметрів інсталяції <<include>> Налаштування шляху інсталяції
- Налаштування параметрів інсталяції <<extend>> Створення ярлика на робочому столі
- Налаштування параметрів інсталяції <<extend>> Додавання іконки

2. Три сценарії використання

Назва	Створення нового проєкту
Актори	Користувач
Передумови	Користувач створив новий інсталяційний проект
Постумови	Користувач створив новий інсталяційний проект і налаштував основні параметри
Порядок дій	1. Користувач натискає кнопку для створення нового проєкту. 2. Користувач вводить основні налаштування проєкту, такі як ім'я програми та версія. 3. Система приймає введені дані і створює новий проект, зберігаючи його у відповідному форматі.
Виняток №1	Користувач не ввів усі необхідні дані для проєкту. Система відображає повідомлення про помилку та пропонує заповнити всі поля.

Назва	Вибір файлів для інсталяції
Актори	Користувач
Передумови	Користувач створив новий проєкт і перейшов до етапу вибору файлів для інсталяції
Постумови	Користувач вибрав файли, які повинні бути включені до інсталяційного пакету
Порядок дій	1. Користувач натискає кнопку для вибору файлів. 2. Система відкриває вікно вибору файлів, де користувач може вибрати необхідні файли та папки. 3. Користувач вибирає файли, що повинні бути включені в інсталяцію, і підтверджує вибір. 4. Система зберігає вибрані файли в проєкті та готова до наступного етапу.
Виняток №1	Користувач намагається вибрати файл, який не підтримується для інсталяції (наприклад, формат файлу не підтримується). Система повідомляє користувача про помилку та пропонує вибрати інші файли.

Виняток №2	Користувач вибрав занадто велику кількість файлів, і система не може обробити вибір. Система відображає повідомлення про перевищення ліміту та просить вибрати меншу кількість файлів.

Назва	Генерація .exe файлу
Актори	Користувач, Інсталятор
Передумови	Користувач вибрав файли для інсталяції та налаштував параметри інсталяції
Постумови	Система згенерувала інсталяційний файл у форматі .exe, готовий для запуску
Порядок дій	1. Користувач натискає кнопку для генерації інсталяційного файлу. 2. Система перевіряє, чи всі необхідні файли та налаштування вибрані. 3. Якщо все правильно, система починає створення .exe файлу. 4. Система генерує інсталяційний файл, перевіряє його на наявність помилок і зберігає файл в обрану користувачем директорію.
Виняток №1	При генерації .exe файлу система виявляє помилку в процесі (наприклад, відсутність необхідних файлів або невірні налаштування).

3. Діаграма класів предметної області

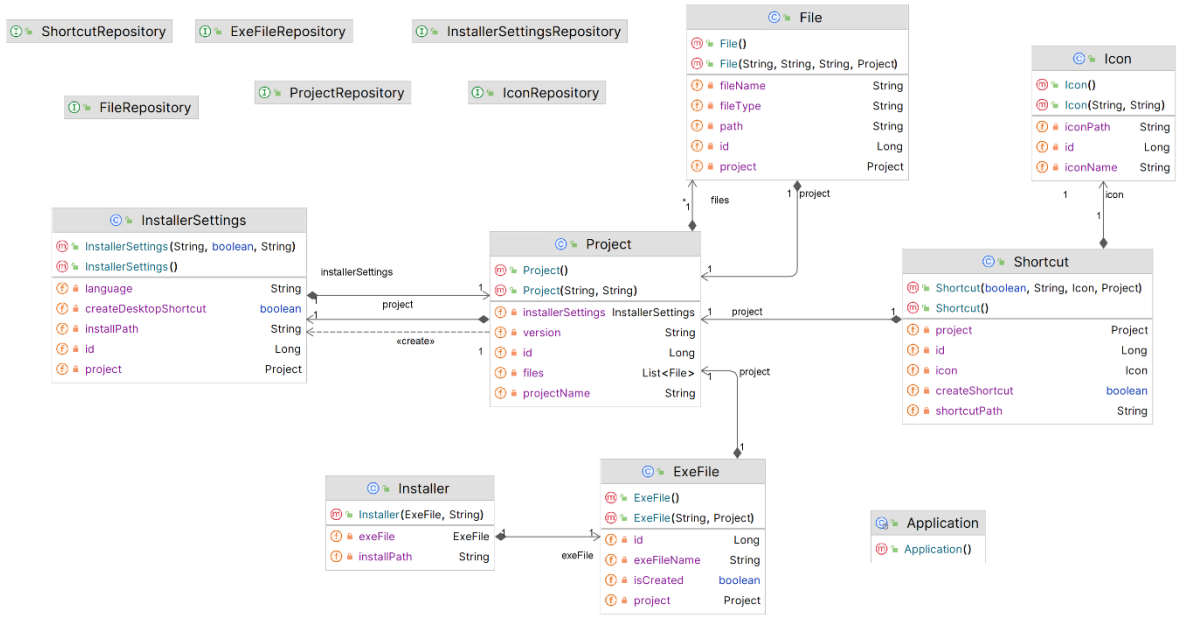

Рисунок 2 – Діаграма класів предметної області

4. Структура бази даних

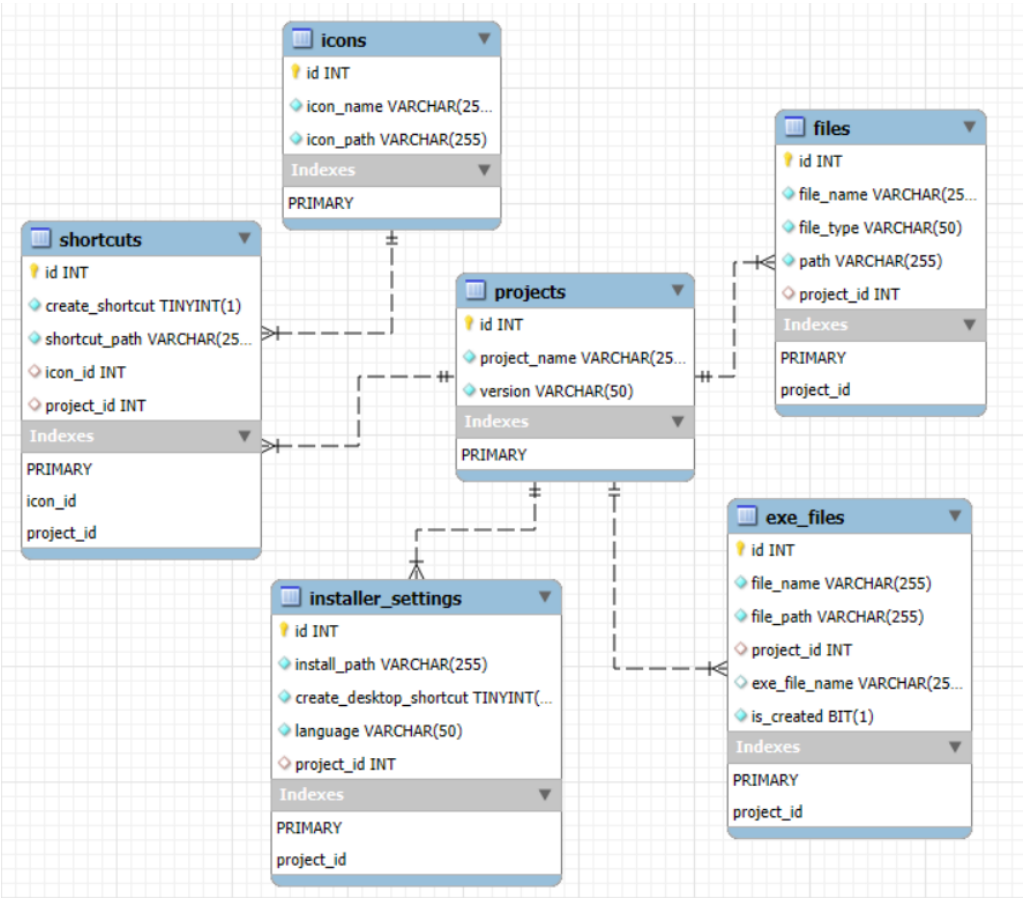

Рисунок 3 – Зображення структури бази даних

Опис структури бази даних:

База даних проєкту Installer Generator побудована для зберігання інформації про проєкти, файли, налаштування інсталяції, іконки, ярлики та згенеровані .exe файли. Її структура забезпечує узгодженість між сутностями та підтримує логіку генератора інсталяційних пакетів.

- projects — головна таблиця, яка містить базову інформацію про проєкти: назву та версію. Вона є центральною сутністю, з якою пов’язані інші таблиці.
- files — зберігає список файлів, що входять до складу конкретного проєкту. Кожен запис пов’язаний із таблицею projects через поле project_id.
- installer_settings — містить налаштування інсталяції: шлях встановлення, мову інтерфейсу та параметр створення ярлика на робочому столі. Пов’язана з конкретним проєктом у співвідношенні один до одного.
- icons — зберігає інформацію про іконки, які можуть використовуватися у процесі створення ярликів або інсталятора.
- shortcuts — описує ярлики, які створюються під час інсталяції. Кожен ярлик пов’язаний як із проєктом (project_id), так і з іконкою (icon_id).
- exe_files — зберігає інформацію про згенеровані виконувані файли (.exe), що є кінцевим результатом роботи генератора. Має посилання на відповідний проєкт.

Зв'язки між таблицями бази даних:

1. projects → files

Зв'язок один до багатьох (1:N) між таблицею projects та таблицею files. Один проєкт може містити багато файлів. Кожен файл прив’язується до конкретного проєкту через поле project_id в таблиці files.

2. projects → installer_settings

Зв'язок один до одного (1:1) між таблицею projects та таблицею installer_settings. Кожен проєкт має одне налаштування інсталяції, яке містить інформацію про шлях до інсталяції, мову інтерфейсу та налаштування ярлика на робочому столі. Поле project_id в таблиці installer_settings є зовнішнім ключем до таблиці projects.

3. projects → exe_files

Зв'язок один до багатьох (1:N) між таблицею projects та таблицею exe_files. Один проєкт може генерувати багато виконуваних файлів (.exe), кожен з яких пов'язаний із певним проєктом через поле project_id в таблиці exe_files.

4. projects → shortcuts

Зв'язок один до багатьох (1:N) між таблицею projects та таблицею shortcuts. Один проєкт може мати кілька ярликів. Кожен ярлик, який створюється для проєкту, прив'язується до проєкту через поле project_id в таблиці shortcuts.

5. shortcuts → icons

Зв'язок багато до одного (N:1) між таблицею shortcuts та таблицею icons. Кожен ярлик має одну іконку, яка визначається через зовнішній ключ icon_id в таблиці shortcuts.

6. files → projects

Зв'язок багато до одного (N:1) між таблицею files та таблицею projects. Кожен файл належить одному проєкту, що позначено через поле project_id в таблиці files.

7. exe_files → projects

Зв'язок багато до одного (N:1) між таблицею exe_files та таблицею projects. Кожен виконуваний файл (.exe) пов’язаний з конкретним проєктом через поле project_id в таблиці exe_files.

8. installer_settings → projects

Зв'язок один до одного (1:1) між таблицею installer_settings та таблицею projects, де поле project_id в таблиці installer_settings є зовнішнім ключем до таблиці projects.

5. Архітектура проєкту в IDE IntelliJ

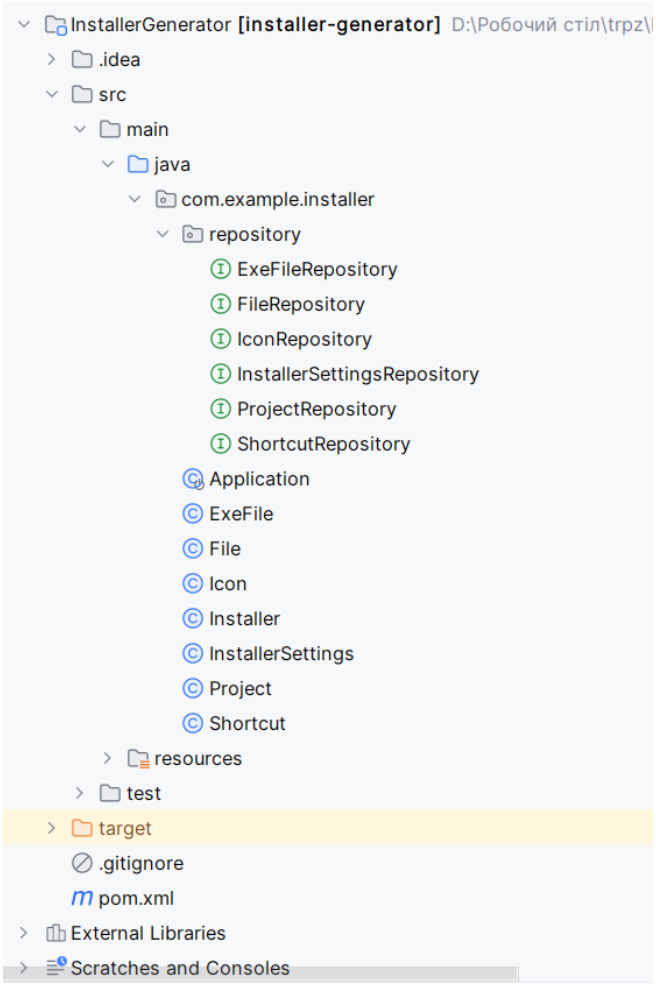

Рисунок 4 – Архітектура проєкту

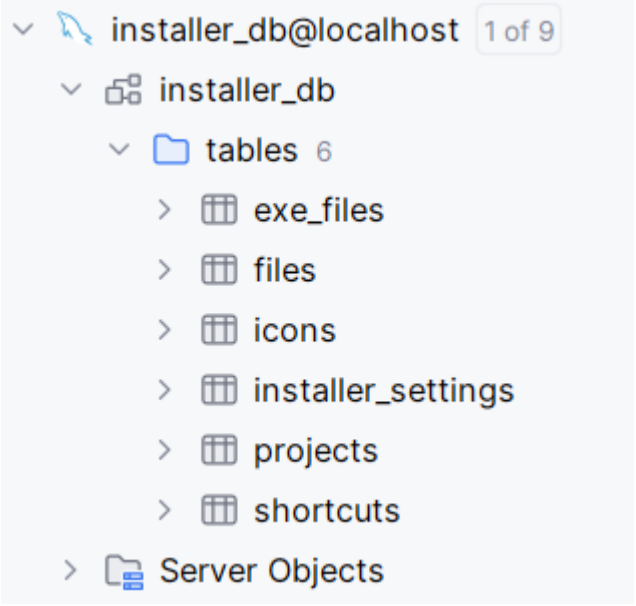

Рисунок 5 – База даних

6. package com.example.installer.repository – пакет з інтерфейсами репозиторіїв (патерн Repository)

```java
package com.example.installer.repository;

import com.example.installer.ExeFile;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface ExeFileRepository extends JpaRepository<ExeFile, Long> {

}
```

```java
package com.example.installer.repository;

import com.example.installer.File;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface FileRepository extends JpaRepository<File, Long> {

}
```

```java
package com.example.installer.repository;

import com.example.installer.Icon;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface IconRepository extends JpaRepository<Icon, Long> {

}
```

```java
package com.example.installer.repository;

import com.example.installer.InstallerSettings;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface InstallerSettingsRepository extends
JpaRepository<InstallerSettings, Long> {

}
```

```java
package com.example.installer.repository;

import com.example.installer.Project;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface ProjectRepository extends JpaRepository<Project, Long> {

}
```

```java
package com.example.installer.repository;

import com.example.installer.Shortcut;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
```

```java
@Repository
public interface ShortcutRepository extends JpaRepository<Shortcut, Long> {

}
```

Посилання на репозиторій GitHub:

https://github.com/glunana/InstallerGenerator.git

Висновки

Під час виконання лабораторної роботи було створено діаграму варіантів використання та описано три сценарії використання «Створення нового проєкту», «Вибір файлів для інсталяції» та «Генерація .exe файлу». Також було зображано діаграму класів предметної області та діаграму структури бази даних. У процесі розробки генератора інсталяційних пакетів було реалізовано основні компоненти системи, які відповідають вимогам до налаштувань файлів для інсталяції, інтерактивних можливостей (галочки, текстові поля) та генерації єдиного .exe файлу. Створено відповідні класи для моделей предметної області, налаштовано базу даних з використанням репозиторіїв для взаємодії з нею. Завдяки успішному тестуванню класів та перевірці роботи з базою даних, система готова до подальшого розвитку та інтеграції з іншими компонентами.

Контрольні запитання

1. Що таке UML?

UML (Unified Modeling Language) - універсальна мова візуального моделювання для специфікації, проєктування, візуалізації та документування систем.

2. Що таке діаграма класів UML?

Діаграма класів - статичне представлення структури системи: класи, їх атрибути, операції та зв’язки.

3. Які діаграми UML називають канонічними?

Канонічні діаграми: варіантів використання, класів, кооперації, послідовності, станів, діяльності, компонентів, розгортання.

4. Що таке діаграма варіантів використання?

Діаграма варіантів використання відображає вимоги: актори, use case та їхні відношення.

5. Що таке варіант використання?

Варіант використання - послідовність дій, яку система виконує у взаємодії з актором для досягнення мети.

6. Які відношення можуть бути відображені на діаграмі використання?

Відношення: асоціація, узагальнення, включення (include), розширення (extend).

7. Що таке сценарій?

Сценарій використання - текстовий покроковий опис виконання use case (передумови, кроки, винятки, результат).

8. Що таке діаграма класів?

Діаграма класів - графічна модель класів із їх атрибутами, операціями та зв’язками (асоціації, узагальнення, агрегації, композиції).

9. Які зв’язки між класами ви знаєте?

Зв’язки між класами: асоціація, узагальнення (успадкування), агрегація, композиція.

10. Чим відрізняється композиція від агрегації?

Агрегація - слабкий зв’язок «частина-ціле», частини можуть існувати окремо;

композиція - сильний зв’язок, частини не існують без цілого.

11. Чим відрізняється зв'язки типу агрегації від зв'язків композиції на діаграмах класів?

На діаграмі: агрегація - порожній ромб біля цілого; композиція – зафарбований ромб біля цілого.

12. Що являють собою нормальні форми баз даних?

Нормальні форми - правила структурування таблиць для зменшення надмірності: 1НФ (атомарні значення), 2НФ (повна функціональна залежність від усього ключа), 3НФ (без транзитивних залежностей), НФ Бойса-Кодда (кожен детермінант - ключ).

13. Що таке фізична модель бази даних? Логічна?

Фізична модель - файлові/дискові структури зберігання даних; логічна модель - таблиці, зв’язки, індекси в СУБД.

14. Який взаємозв'язок між таблицями БД та програмними класами?

Таблиці відповідають сутностям/класам: найчастіше один клас – одна таблиця; можливі варіанти один клас - кілька таблиць або одна таблиця - кілька класів.