Міністерство освіти і науки України

Національний технічний університет України

“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет інформатики та обчислювальної техніки

Лабораторна робота №2

«Основи проектування»

З дисципліни «**Технології розроблення програмного забезпечення**»

Виконав:

Студент групи ІА-31

Дук М. Д.

Перевірив:

Мягкий М. Ю.

Київ 2025

Мета: 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.

Технічне завдання:

Архіватор (strategy, adapter, factory method, facade, visitor)

Архіватор повинен являти собою візуальний додаток з можливістю створення і редагування архівів різного типу (.tar.gz, .zip, тощо) – додавання/ видалення файлів / папок, редагування метаданих (по можливості), перевірка checksum архівів, тестування архівів на наявність пошкоджень, розбиття архівів на частини.

Теоретичні відомості

ООАП і рівні моделей

ООАП розглядає модель складної системи як набір взаємопов’язаних уявлень (статичних і динамічних), що деталізуються від концептуального до логічного та фізичного рівнів. Концептуальна модель задає загальне уявлення на початку проєктування, далі вона конкретизується логічними і фізичними аспектами в межах ієрархічної побудови моделей.

Діаграми UML

UML фіксує уявлення про систему в діаграмах як графічних моделях з формальною семантикою вершин і ребер. Базові види діаграм: варіантів використання, класів, кооперації, послідовності, станів, діяльності, компонентів і розгортання.

Варіанти використання

Діаграма варіантів використання є вихідною концептуальною моделлю, що графічно відображає вимоги без опису внутрішньої реалізації системи. Вона слугує для окреслення меж функціональності, формулювання поведінкових вимог і підготовки до аналізу, проєктування, розробки та тестування.

Актори та прецеденти

Актор — це зовнішній суб’єкт (людина, пристрій, програма або інша система), що взаємодіє із системою для досягнення власних цілей. Варіант використання описує послугу системи як послідовність дій у діалозі з актором і позначається еліпсом із короткою назвою.

Відносини в use case

Між елементами застосовують асоціації (ненаправлені й направлені), узагальнення для спадкування властивостей, а також залежності типу include та extend. Include виносить спільну поведінку до окремого варіанту й завжди виконується, тоді як extend додає умовне розширення через точки розширення.

Сценарії використання

Сценарії — текстові покрокові описи процесів взаємодії користувачів і системи, що усувають неоднозначності діаграм. Рекомендована структура: передумови, постумови, сторони, короткий опис, основний перебіг подій, винятки та примітки.

Діаграми класів

Діаграма класів статично відображає структуру системи: класи з назвою, атрибутами та операціями, без показу динаміки. Для атрибутів і операцій визначають видимість: public (+), package (~), protected (#) і private (−) як елементи інтерфейсу та інкапсуляції.

Зв’язки між класами

Асоціації моделюють зв’язки і множинність між класами, можуть бути бінарними або клас-асоціаціями з власними атрибутами. Агрегація відображає відношення частина-ціле зі слабким зв’язком, тоді як композиція задає цілісність, де частини не існують поза цілим.

Логічна модель БД

Логічна модель БД — це структура таблиць, уявлень та індексів для програмування, на відміну від фізичної, що описує файли й розміщення даних на носіях. Відображення класів у таблиці можливе у схемах один клас — одна таблиця, одна таблиця — кілька класів або один клас — кілька таблиць.

Нормалізація

Нормалізація зменшує логічну надмірність і суперечливість даних; 1НФ вимагає атомарних значень, 2НФ — повної залежності неключових атрибутів від ключа, 3НФ — відсутності транзитивних залежностей, а НФ Бойса-Кодда посилює вимоги до детермінантів залежностей.

Проєктування БД

Проєктування виконують у засобах на кшталт Schema View або MS Access: створення таблиць, задання типів, призначення первинного ключа та побудова зв’язків між таблицями.

Аналіз вимог

Система буде являти собою графічний додаток для керування архівами різних форматів (наприклад, .tar.gz, .zip). Основний функціонал включає створення, редагування, перевірку цілісності та розподіл архівів. Для забезпечення гнучкості, масштабованості та простоти підтримки кодова база буде використовувати патерни проектування, зокрема:

- **Strategy:** Для реалізації алгоритмів стиснення/розтиснення для різних форматів архівів.
- **Adapter:** Для інтеграції зі зовнішніми бібліотеками роботи з архівами.
- **Factory Method:** Для інкапсуляції логіки створення об'єктів архівів різних типів.
- **Facade:** Для надання спрощеного єдиного інтерфейсу до складної підсистеми роботи з архівами.
- **Visitor:** Для реалізації операцій, що не залежать від основної логіки об'єктів (наприклад, розрахунок контрольних сум для елементів архіву).

2. Вимоги до системи

2.1. Функціональні вимоги

- **FR-001: Базове керування архівами**
    - FR-001.1: Створення нового архіву вказаного типу.
    - FR-001.2: Відкриття та перегляд вмісту існуючого архіву.
    - FR-001.3: Додавання файлів і папок до архіву.
    - FR-001.4: Видалення файлів і папок з архіву.
- **FR-002: Управління метаданими та історією**
    - FR-002.1: Зберігання метаданих архіву (назва, дата модифікації, розмір, тощо) у базі даних.
    - FR-002.2: Можливість редагування частини метаданих, що підтримуються форматом (наприклад, коментарі).
    - FR-002.3: Ведення журналу історії операцій для кожного архіву.
- **FR-003: Перевірка цілісності та відновлення**
    - FR-003.1: Розрахунок та збереження контрольних сум (checksum) для архівів.
    - FR-003.2: Тестування архіву на наявність пошкоджень шляхом порівняння контрольних сум.
- **FR-004: Робота з великими архівами**
    - FR-004.1: Розбиття архіву на частини заданого розміру.
    - FR-004.2: Об'єднання частин у цілісний архів.

2.2. Нефункціональні вимоги

- **NF-001: Продуктивність:** Забезпечення високої швидкості операцій стиснення та розпаковування навіть для файлів великого об'єму (десятки ГБ).
- **NF-002: Надійність та безпека:** Обов'язкова перевірка цілісності даних для запобігання втраті інформації. Захист від пошкодження архівів під час обробки.
- **NF-003: Масштабованість та розширюваність:** Архітектура повинна дозволяти легке додавання підтримки нових форматів архівів.
- **NF-004: Зручність використання (Usability):** Наявність інтуїтивно зрозумілого графічного інтерфейсу користувача (GUI).

3. Сценарії використання (Use Cases)

UC-01: Створення нового архіву

- **Актор:** Користувач, Файлова система, БД.
- **Передумови:** Користувач має файли для архівування.
- **Основний потік подій:**
    1. Користувач запускає операцію "Створити архів".
    2. Система запитує файли/папки для додавання з файлової системи, тип архіву (.zip, .tar.gz, тощо), ім'я та розташування.
    3. Користувач вказує необхідні параметри та підтверджує створення.
    4. Система створює архів із заданими файлами.
    5. Система розраховує контрольну суму (checksum) для створеного архіву.
    6. Система зберігає метадані та контрольну суму архіву в базі даних.
- **Альтернативні потоки:**
    - **Помилка на етапах 4-6:** Система відображає повідомлення про помилку, описуючи її причину. Створення архіву або збереження даних скасовується.

UC-02: Додавання файлів до існуючого архіву

- **Актор:** Користувач, Файлова система, БД.
- **Передумови:** Обраний архів існує, не пошкоджений та доступний для запису.
- **Основний потік подій:**
    1. Користувач відкриває архів через GUI.
    2. Користувач ініціює операцію "Додати файли" та обирає цільові файли.
    3. Система додає обрані файли до архіву.
    4. Система оновлює контрольну суму архіву.
    5. Система оновлює метадані архіву в базі даних.
    6. Інтерфейс відображає оновлений вміст архіву.
- **Альтернативні потоки:**

- **Файл вже існує в архіві (крок 3):** Система запитує у користувача підтвердження на перезапис файлу.

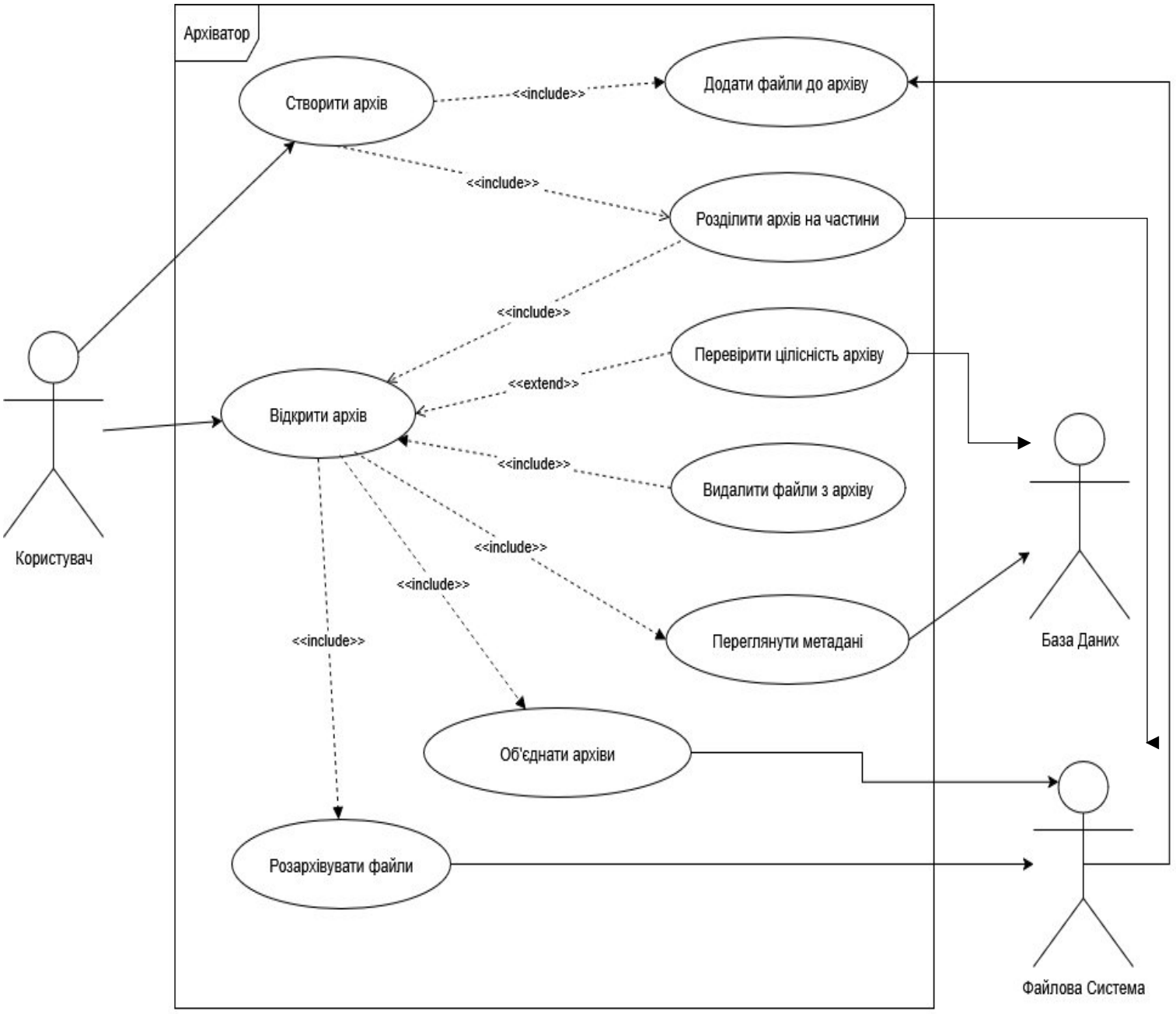
UC-03: Перевірка цілісності архіву

- **Актор:** Користувач, БД.
- **Передумови:** Обраний архів існує.
- **Основний потік подій:**
  1. Користувач ініціює для архіву операцію "Перевірити цілісність".
  2. Система розраховує поточну контрольну суму архіву.
  3. Система завантажує оригінальну контрольну суму з бази даних.
  4. Система порівнює отримані значення.
  5. Якщо суми збігаються, система сповіщає про успішну перевірку. В іншому випадку - попереджає про можливі пошкодження.
- **Альтернативні потоки:**
  - **Відсутність контрольних сум в БД (крок 3):** Система розраховує контрольну суму, зберігає її в БД та інформує користувача, що перевірка була проведена за новим зразком.

Проектування архітектури

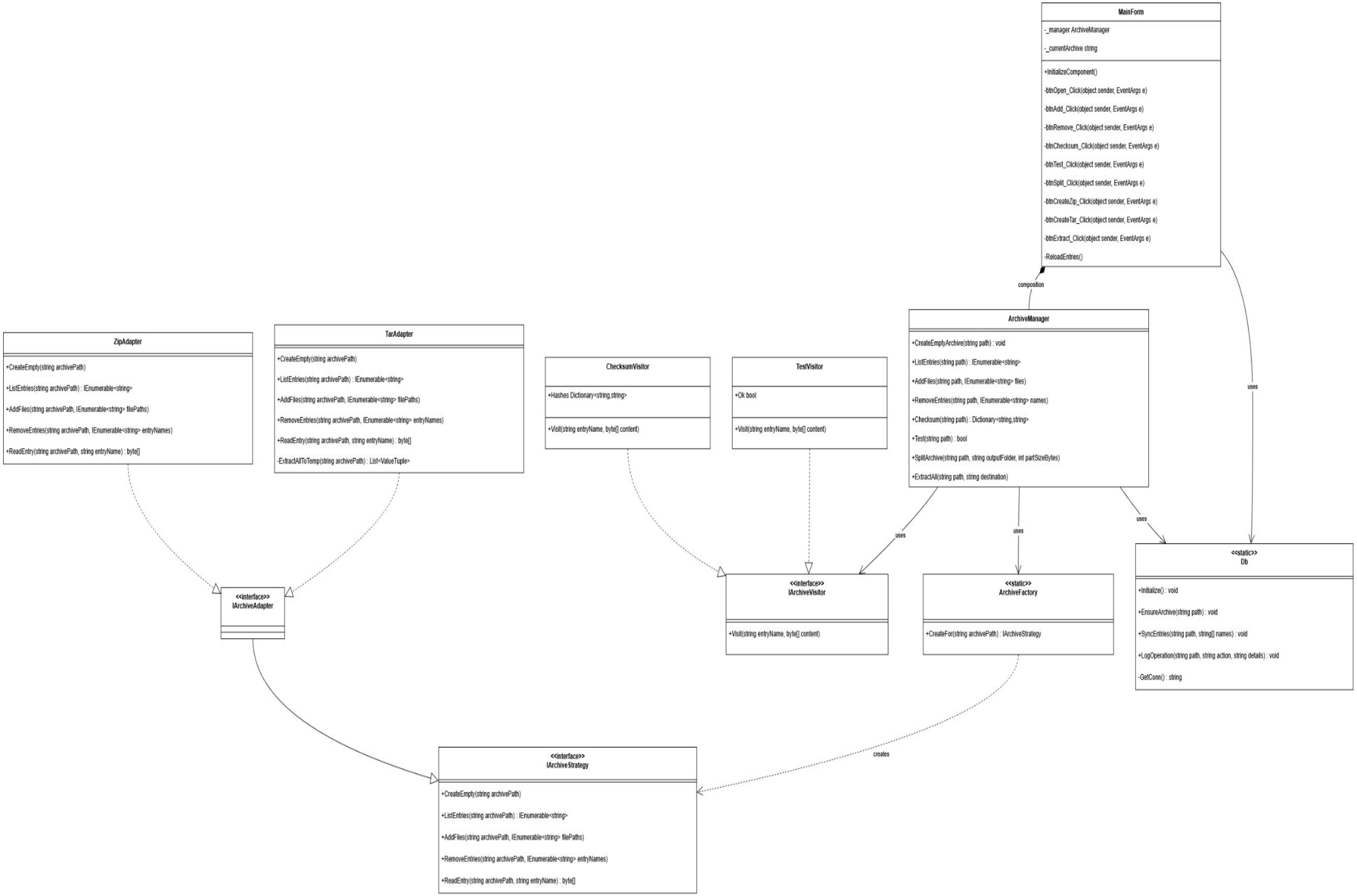
Для побудови UML діаграм було обрано редактор Draw.IO, через його простоту та інтуїтивність у використанні та через потужні властивості для моделювання.

Діаграма варіантів використання



Діаграма класів для реалізованої частини системи.

В діаграмі можливі зміни, оскільки система знаходиться на етапі проектування. Вона може бути розширена.


MainForm
-_manager ArchiveManager
-_currentArchive string
+InitializeComponent()
-btnOpen_Click(object sender, EventArgs e)
-btnAdd_Click(object sender, EventArgs e)
-btnRemove_Click(object sender, EventArgs e)
-btnChecksum_Click(object sender, EventArgs e)
-btnTest_Click(object sender, EventArgs e)
-btnSplit_Click(object sender, EventArgs e)
-btnCreateZip_Click(object sender, EventArgs e)
-btnCreateTar_Click(object sender, EventArgs e)
-btnExtract_Click(object sender, EventArgs e)
-ReloadEntries()
composition
uses
ArchiveManager
+CreateEmptyArchive(string path) : void
+ListEntries(string path) : IEnumerable<string>
+AddFiles(string path, IEnumerable<string> files)
+RemoveEntries(string path, IEnumerable<string> names)
+Checksum(string path) : Dictionary<string,string>
+Test(string path) : bool
+SplitArchive(string path, string outputFolder, int partSizeBytes)
+ExtractAll(string path, string destination)
uses
uses
uses
ZipAdapter
+CreateEmpty(string archivePath)
+ListEntries(string archivePath) : IEnumerable<string>
+AddFiles(string archivePath, IEnumerable<string> filePaths)
+RemoveEntries(string archivePath, IEnumerable<string> entryNames)
+ReadEntry(string archivePath, string entryName) : byte[]
TarAdapter
+CreateEmpty(string archivePath)
+ListEntries(string archivePath) : IEnumerable<string>
+AddFiles(string archivePath, IEnumerable<string> filePaths)
+RemoveEntries(string archivePath, IEnumerable<string> entryNames)
+ReadEntry(string archivePath, string entryName) : byte[]
-ExtractAllToTemp(string archivePath) : List<ValueTuple>
ChecksumVisitor
+Hashes Dictionary<string,string>
+Visit(string entryName, byte[] content)
TestVisitor
+Ok bool
+Visit(string entryName, byte[] content)
<<interface>>
IArchiveAdapter
<<interface>>
IArchiveVisitor
+Visit(string entryName, byte[] content)
<<static>>
ArchiveFactory
+CreateFor(string archivePath) : IArchiveStrategy
<<static>>
Db
+Initialize() : void
+EnsureArchive(string path) : void
+SyncEntries(string path, string[] names) : void
+LogOperation(string path, string action, string details) : void
-GetConn() : string
creates
<<interface>>
IArchiveStrategy
+CreateEmpty(string archivePath)
+ListEntries(string archivePath) : IEnumerable<string>
+AddFiles(string archivePath, IEnumerable<string> filePaths)
+RemoveEntries(string archivePath, IEnumerable<string> entryNames)
+ReadEntry(string archivePath, string entryName) : byte[]

Посилання для детального ознайомлення: [Діаграма класів]

Діаграма класів представляє структуру системи архівування з основними компонентами.

```
namespace ArchiverApp.Core
{
    2 usages  3 inheritors  PriMerro23  1 exposing API
    public interface IArchiveStrategy
    {
        1 usage  2 implementations  PriMerro23
        void CreateEmpty(string archivePath);
        6 usages  2 implementations  PriMerro23
        IEnumerable<string> ListEntries(string archivePath);
        1 usage  2 implementations  PriMerro23
        void AddFiles(string archivePath, IEnumerable<string> filePaths);
        1 usage  2 implementations  PriMerro23
        void RemoveEntries(string archivePath, IEnumerable<string> entryNames);
        3 usages  2 implementations  PriMerro23  More...
        byte[] ReadEntry(string archivePath, string entryName);
    }

    2 usages  2 inheritors  PriMerro23
    public interface IArchiveAdapter : IArchiveStrategy { }

    2 usages  2 inheritors  PriMerro23
    public interface IArchiveVisitor
    {
        2 implementations  PriMerro23
        void Visit(string entryName, byte[] content);
    }
}
```

Рис. 1 - IArchiveStrategy та IArchiveAdapter

```
namespace ArchiverApp.Core
{
    1 usage  PriMerro23
    public class ZipAdapter : IArchiveAdapter
    {
        0+1 usages  PriMerro23
        public void CreateEmpty(string archivePath){...}

        0+6 usages  PriMerro23
        public IEnumerable<string> ListEntries(string archivePath){...}

        0+1 usages  PriMerro23
        public void AddFiles(string archivePath, IEnumerable<string> filePaths){...}

        0+1 usages  PriMerro23
        public void RemoveEntries(string archivePath, IEnumerable<string> entryNames){...}

        0+3 usages  PriMerro23
        public byte[] ReadEntry(string archivePath, string entryName){...}
    }
}
```

Рис. 2 - ZipAdapter

```
namespace ArchiverApp.Core
{
    1 usage  PriMerro23
    public class TarAdapter : IArchiveAdapter
    {
        0+1 usages  PriMerro23
        public void CreateEmpty(string archivePath){...}

        0+6 usages  PriMerro23
        public IEnumerable<string> ListEntries(string archivePath){...}

        0+1 usages  PriMerro23
        public void AddFiles(string archivePath, IEnumerable<string> filePaths){...}

        0+1 usages  PriMerro23
        public void RemoveEntries(string archivePath, IEnumerable<string> entryNames){...}

        0+3 usages  PriMerro23
        public byte[] ReadEntry(string archivePath, string entryName){...}

        2 usages  PriMerro23
        private List<(string name, string path)> ExtractAllToTemp(string archivePath){...}
    }
}
```

Рис. 3 – TarAdapter

```
    public static class ArchiveFactory
    {
        7 usages  PriMerro23
        public static IArchiveStrategy CreateFor(string archivePath){...}
    }
}

namespace ArchiverApp.Core.Facade
{
    using System.Collections.Generic;
    using System.IO;
    using System.Linq;
    using ArchiverApp.Data;

    4 usages  PriMerro23
    public class ArchiveManager
    {
        2 usages  PriMerro23
        public static void CreateEmptyArchive(string path){...}

        1 usage  PriMerro23
        public IEnumerable<string> ListEntries(string path){...}

        1 usage  PriMerro23
        public void AddFiles(string path, IEnumerable<string> files){...}

        1 usage  PriMerro23
        public void RemoveEntries(string path, IEnumerable<string> names){...}

        1 usage  PriMerro23
        public Dictionary<string, string> Checksum(string path){...}

        1 usage  PriMerro23
        public bool Test(string path){...}

        1 usage  PriMerro23
        public void SplitArchive(string path, string outputFolder, int partSizeBytes){...}

        1 usage  PriMerro23
        public void ExtractAll(string path, string destination){...}
    }
```

Рис. 4 – ArchiveFactory та ArchiveManager

```
namespace ArchiverApp.Core.Visitors
{
    1 usage  PriMerro23
    public class ChecksumVisitor : IArchiveVisitor
    {
        2 usages
        public Dictionary<string, string> Hashes { get; } = new Dictionary<string, string>();
        1 usage  PriMerro23
        public void Visit(string entryName, byte[] content){...}
    }

    1 usage  PriMerro23
    public class TestVisitor : IArchiveVisitor
    {
        2 usages
        public bool Ok { get; private set; } = true;
        1 usage  PriMerro23
        public void Visit(string entryName, byte[] content){...}
    }
}
```

Рис. 5 – ChecksumVisitor та TestVisitor

```
namespace ArchiverApp.Data
{
    11 usages  PriMerro23
    public static class Db
    {
        1 usage  PriMerro23
        public static void Initialize(){...}

        1 usage  PriMerro23
        public static void EnsureArchive(string path){...}

        3 usages  PriMerro23
        public static void SyncEntries(string path, string[] names){...}

        6 usages  PriMerro23
        public static void LogOperation(string path, string action, string details){...}

        4 usages  PriMerro23
        private static string GetConn() => ConfigurationManager.ConnectionStrings["ArchiverDb"].ConnectionString;
    }
}
```

Рис. 5 – Db

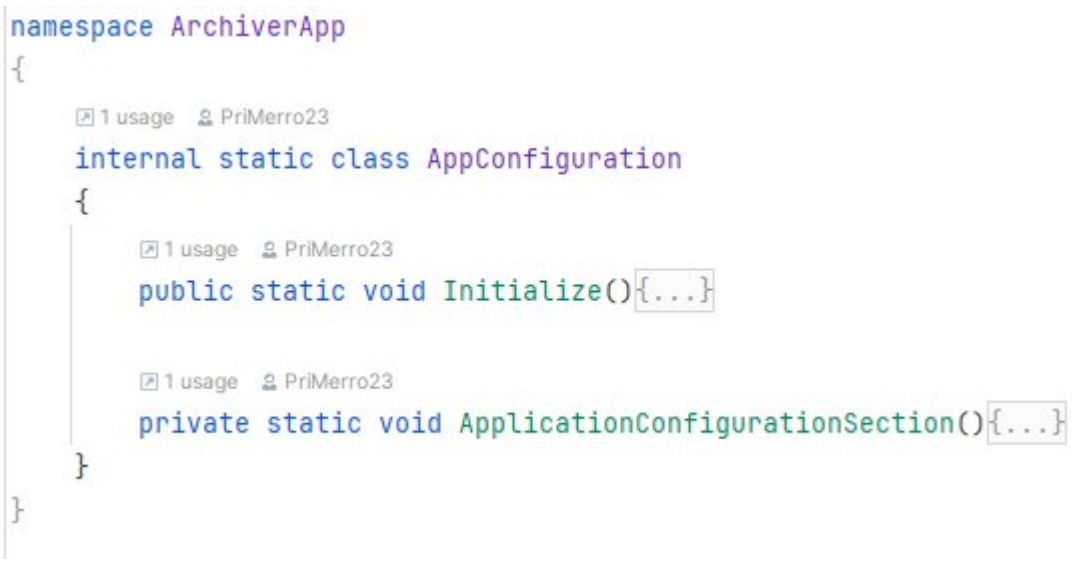
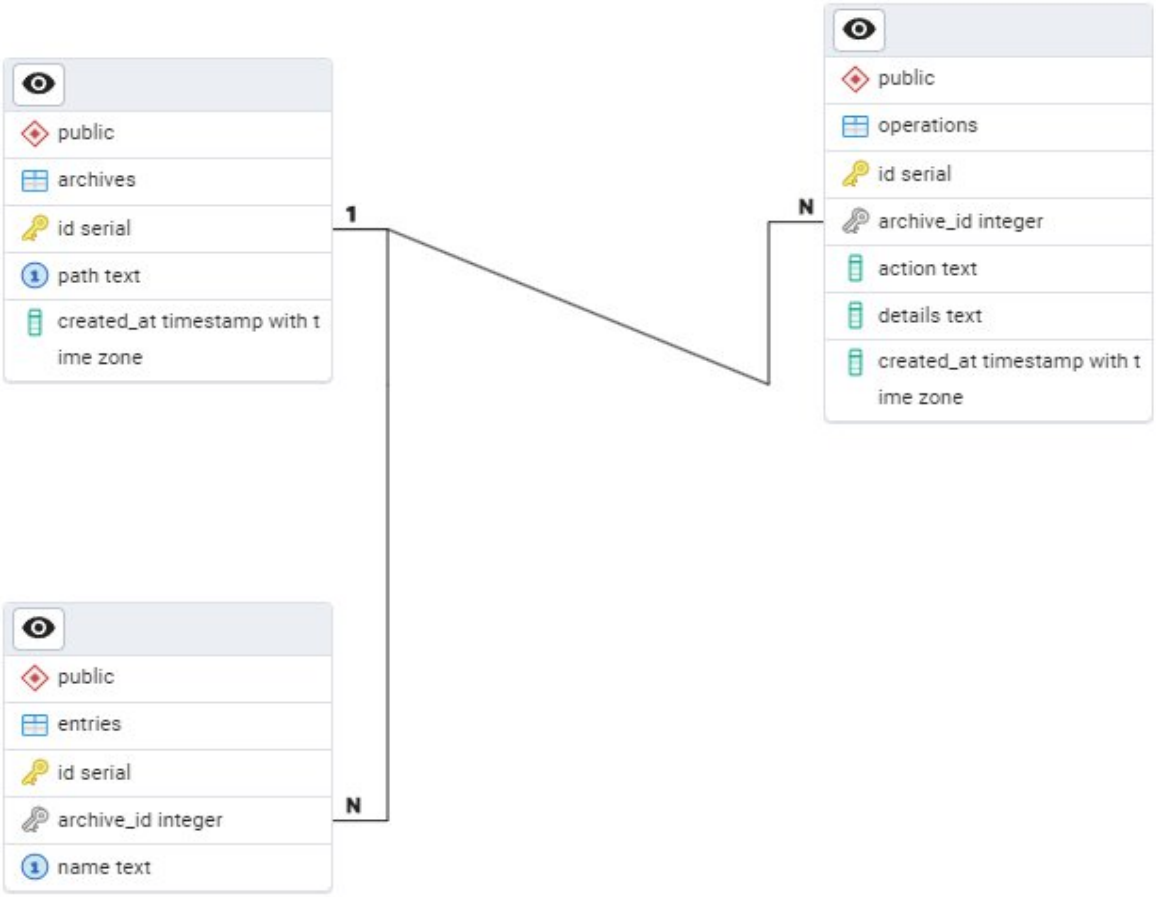


Рис. 6 – AppConfiguration

```
public partial class MainForm : Form
{
    private readonly ArchiveManager _manager = new ArchiveManager();
    private string? _currentArchive;

    1 usage  PriMerro23
    public MainForm(){...}

    1 usage  PriMerro23
    private void btnOpen_Click(object? sender, EventArgs e){...}

    1 usage  PriMerro23
    private void btnAdd_Click(object? sender, EventArgs e){...}

    1 usage  PriMerro23
    private void btnRemove_Click(object? sender, EventArgs e){...}

    1 usage  PriMerro23
    private void btnChecksum_Click(object? sender, EventArgs e){...}

    1 usage  PriMerro23
    private void btnTest_Click(object? sender, EventArgs e){...}

    1 usage  PriMerro23
    private void btnSplit_Click(object? sender, EventArgs e){...}

    1 usage  PriMerro23
    private void btnCreateZip_Click(object? sender, EventArgs e){...}

    1 usage  PriMerro23
    private void btnCreateTar_Click(object? sender, EventArgs e){...}

    1 usage  PriMerro23
    private void btnExtract_Click(object? sender, EventArgs e){...}

    5 usages  PriMerro23
    private void ReloadEntries(){...}
}
}
```

Рис. 7 – MainForm

Структура бази даних



	Name	Data type
🗑	id	integer
🗑	path	text
🗑	created_at	timestamp with tin

Рис. 1 - Таблиця Archives

Name	Data type
id	integer
archive_id	integer
name	text

Рис. 2 - Таблиця Entries

Name	Data type
id	integer
archive_id	integer
action	text
details	text
created_at	timestamp with tin

Рис. 3 - Таблиця Operations

У застосунку база даних необхідна для того щоб вести аудит дій зі створеними архівами та маніпуляцій з ними (додавання та видалення файлів, деталі час створення, тощо.)

Висновок: Виконання лабораторної роботи дозволило ознайомитися з основними методами об’єктно-орієнтованого моделювання системи за допомогою UML. Було створено діаграму варіантів використання, яка наочно відображає взаємодію користувачів із системою та ключові функції, що вона надає. За допомогою діаграми класів було показано основні класи системи, їх атрибути та методи, а також зв’язки між класами, що дозволяє зрозуміти структурну організацію програми та логіку взаємодії об’єктів.

Відповіді на контрольні питання:

1. Що таке UML?

UML (Unified Modeling Language) — це уніфікована мова моделювання, яка використовується для графічного представлення, документування та проектування програмних систем. Вона допомагає описати структуру, поведінку та взаємодію компонентів системи.

2. Що таке діаграма класів UML?

Діаграма класів UML — це статична діаграма, яка показує класи системи, їх атрибути, методи та зв’язки між класами. Вона використовується для

моделювання структури об’єктно-орієнтованих програм.

3. Які діаграми UML називають канонічними?

Канонічні (основні) діаграми UML — це ті, що найчастіше використовуються для опису системи:

• Діаграма класів

• Діаграма варіантів використання (Use Case)

• Діаграма послідовності (Sequence)

• Діаграма станів (State)

• Діаграма активностей (Activity)

4. Що таке діаграма варіантів використання?

Діаграма варіантів використання (Use Case Diagram) показує взаємодію користувачів (акторів) з системою через варіанти використання. Вона відображає, хто і як використовує систему, без деталізації внутрішньої реалізації.

5. Що таке варіант використання?

Варіант використання (Use Case) — це послідовність дій, які система виконує для досягнення певної мети користувача. Наприклад, “Реєстрація користувача”
або “Надіслати повідомлення”.

6. Які відношення можуть бути відображені на діаграмі використання?

На діаграмі варіантів використання можуть бути:

• Association (асоціація) — зв’язок між актором і варіантом використання.

• Include (включення) — один варіант використання завжди виконує інший.

• Extend (розширення) — додатковий варіант, який може виконуватися у певних умовах.

• Generalization (успадкування) — актор або варіант використання наслідує інший.

7. Що таке сценарій?

Сценарій — це конкретна реалізація варіанту використання, тобто послідовність кроків, які відбуваються під час виконання дії користувачем та системою.

8. Що таке діаграма класів?

Діаграма класів — це графічне представлення класів, їх атрибутів і методів, а також зв’язків між класами. Вона описує структуру системи.

9. Які зв’язки між класами ви знаєте?

Основні типи зв’язків:

• Асоціація (Association) — загальний зв’язок між класами.

• Агрегація (Aggregation) — «має»; клас складається з інших, але частини можуть існувати окремо.

• Композиція (Composition) — сильніша агрегація; частини не можуть існувати без цілого.

• Успадкування (Generalization) — один клас наслідує властивості іншого.

• Залежність (Dependency) — один клас використовує інший тимчасово.

10. Чим відрізняється композиція від агрегації?

• Агрегація: частини можуть існувати без цілого.

• Композиція: частини не можуть існувати без цілого; знищення цілого знищує

частини.

11. Чим відрізняється агрегація від композиції на діаграмах класів?

• Агрегація позначається порожнім ромбом на стороні цілого.

• Композиція позначається заповненим ромбом на стороні цілого.

12. Що являють собою нормальні форми баз даних?

Нормальні форми — це правила організації таблиць БД для уникнення надлишковості та аномалій при оновленні даних. Основні:

• 1НФ — всі атрибути атомарні.

• 2НФ — всі неключові атрибути залежать від усього первинного ключа.

• 3НФ — немає транзитивних залежностей між неключовими атрибутами.

13. Що таке фізична і логічна модель БД?

• Логічна модель — структура даних з таблицями, полями та зв’язками, незалежно від СУБД.

• Фізична модель — конкретна реалізація БД у СУБД, включає типи даних, індекси, обмеження, фізичне зберігання.

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

Кожна таблиця БД часто відповідає класу у програмі, а рядки таблиці — об’єктам класу. Поля таблиці відображаються як атрибути класу, а зв’язки між таблицями як зв’язки між класами.