Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Технології розроблення програмного забезпечення

Лабораторна робота №2

«ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ. СЦЕНАРІЇ ВАРІАНТІВ ВИКОРИСТАННЯ. ДІАГРАМИ UML. ДІАГРАМИ КЛАСІВ. КОНЦЕПТУАЛЬНА МОДЕЛЬ СИСТЕМИ.»

Виконав:
студент групи ІА–22:
Клочков І. Ф.

Перевірив:
Мягкий Михайло Юрійович

Київ 2024

Зміст

Тема: Діаграма варіантів використання. Сценарії варіантів використання. Діаграми UML. Діаграми класів. Концептуальна модель системи.

Мета: Проаналізувати тему, намалювати схему прецедентів, діаграму класів, розробити основні класи і структуру бази.

Хід роботи

..18 Shell (total commander) (state, prototype, factory method, template method, interpreter, client-server)

Оболонка повинна вміти виконувати основні дії в системі - перегляд файлів папок в файлової системі, перемикання між дисками, копіювання, видалення, переміщення об'єктів, пошук.

1. Короткі теоретичні відомості

Діаграма варіантів використання

Мова UML – це універсальна мова для візуального моделювання, яка використовується для специфікації, візуалізації, проектування та документування компонентів програмного забезпечення, бізнес-процесів та інших систем. UML дозволяє будувати концептуальні, логічні та графічні моделі складних систем, застосовуючи найкращі методи програмної інженерії.

В рамках UML моделі складних систем представлені у вигляді графічних конструкцій, званих діаграмами. Серед основних видів діаграм UML:

- діаграма варіантів використання (use case diagram),
- діаграма класів (class diagram),
- діаграма послідовностей (sequence diagram),
- діаграма станів (statechart diagram),
- діаграма діяльності (activity diagram),
- діаграма компонентів (component diagram),
- діаграма розгортання (deployment diagram).

Діаграма варіантів використання є початковою концептуальною моделлю системи. Вона відображає загальні вимоги до функціональності системи і не деталізує її внутрішню структуру.

Основні завдання діаграми варіантів використання:

1. Визначення меж функціональності системи.
2. Формулювання загальних вимог до функціональної поведінки.
3. Розробка початкової концептуальної моделі системи.
4. Створення основи для подальшого аналізу, проектування та тестування.

Елементи діаграми:

- Актор (actor) – об'єкт або користувач, який взаємодіє із системою.
- Варіант використання (use case) – дії або послуги, які система надає актору.
- Зв’язки (relationships) – відношення між акторами та варіантами використання.

Діаграми UML. Діаграми класів. Концептуальна модель системи

Діаграми класів найчастіше використовуються при моделюванні програмних систем (ПС). Вони є формою статичного опису системи, показуючи її структуру, але не відображають динамічну поведінку об’єктів. На діаграмах класів зображуються класи, інтерфейси та відносини між ними.

Клас – це основний будівельний блок ПС. Клас має назву, атрибути та операції. На діаграмі клас показується у вигляді прямокутника, розділеного на три області:

- Верхня область – назва класу.
- Середня область – опис атрибутів (властивостей).
- Нижня область – назви операцій (послуг), що надаються об’єктами цього класу.

Атрибути класу визначають структуру даних, які зберігаються в об'єктах. Кожен атрибут має ім’я та тип, що визначає його значення в програмі.
Для атрибутів класу можна задати видимість:

- Відкритий (public) – доступний для будь-якого класу.
- Захищений (protected) – доступний лише для нащадків.
- Закритий (private) – недоступний ззовні і використовується тільки в класі.

Це дозволяє реалізувати інкапсуляцію даних, забезпечуючи захист від несанкціонованого доступу.

Клас містить визначення операцій, які об'єкти цього класу повинні виконувати. Кожна операція має сигнатуру з іменем, типом повернення та списком параметрів. Закриті операції є внутрішніми для об'єктів класу, в той час як відкриті формують інтерфейс класу.

На діаграмах класів зазвичай показуються асоціації та об'єднання (наслідування):

- Асоціація (Association) – відношення між об’єктами. Вона може мати назву та характеристику, таку як множинність, що показує, скільки об’єктів кожного класу може брати участь у зв’язку.
- Об'єднання (Generalization) – показує зв'язок між класом-родителем та класом-нащадком. Цей зв'язок використовується для виявлення спільних характеристик кількох класів, які об’єднуються у батьківський клас.

Логічна структура бази даних:
Існує дві моделі баз даних: фізична та логічна. Фізична модель зберігає дані у вигляді бінарних файлів, оптимізованих для зберігання та отримання інформації. Логічна модель відображає структуру таблиць, представлень, індексів та інших елементів для програмування і використання бази даних.

Процес проектування бази даних полягає в побудові зв’язків між програмними класами та таблицями. Основою для проектування таблиць є нормальні форми, що допомагають уникнути надмірності та аномалій оновлення.

Нормальні форми:

1. 1НФ – кожен атрибут відношення має одне значення.
2. 2НФ – всі неключові атрибути залежать від ключа.
3. 3НФ – немає транзитивних залежностей між неключовими атрибутами.

2. Схема прецедентів

Схема до обраної теми зображена на рисунку 1.

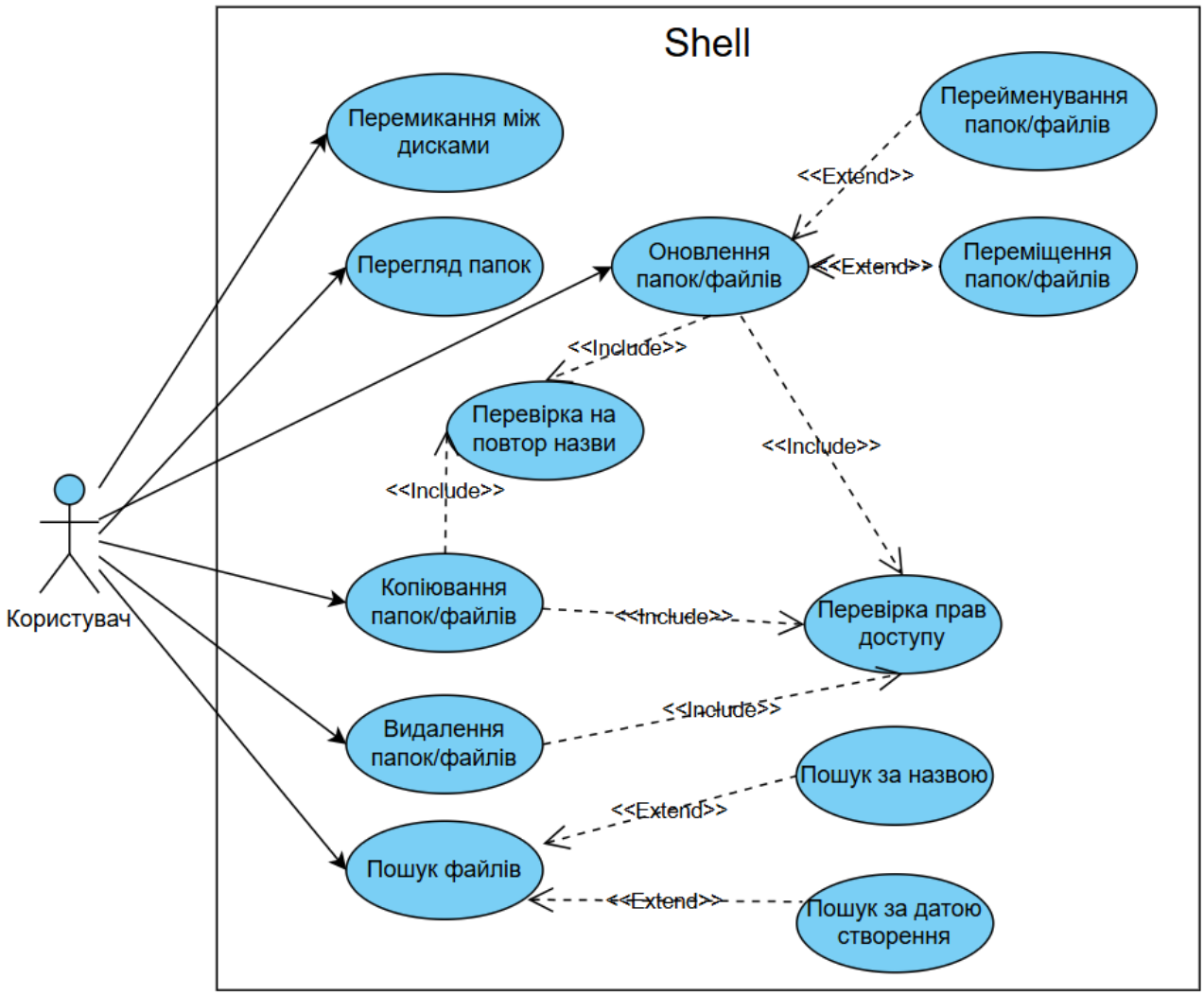

Рисунок 1 – діаграма варіантів використання

Ця діаграма є діаграмою варіантів використання (use case diagram) для файлової оболонки. Вона відображає взаємодію користувача з основними функціями програми.

Основні елементи діаграми:

1. **Актор (Користувач):** Представлений у вигляді чоловічка зліва. Користувач взаємодіє з програмою і викликає її функціональність.
2. **Варіанти використання (Use Cases):**
    - Овальні фігури позначають функції, доступні користувачу.
    - Користувач може виконувати наступні дії:

Взаємозв'язки:

- <<include>> позначає обов'язковий зв'язок, коли один варіант використання є частиною іншого. Наприклад:
    - Оновлення папок/файлів включає перевірку прав доступу.
    - Копіювання папок/файлів включає перевірку на повтор назви.
- <<extend>> позначає додаткову функцію, яка може бути виконана, але не є обов'язковою. Наприклад:
    - Пошук файлів може бути розширений пошуком за назвою файлів.
    - Оновлення папок/файлів може бути розширене перейменуванням папок/файлів.

3. Сценарії використання для 3 прецедентів

Оберемо 3 прецеденти і напишемо для них сценарії використання.

Прецедент 1: Переміщення папок/файлів

Передумови: Користувач має права доступу до вихідної та цільової папки.

Післяумови: Папка/файл переміщені до нової локації.

Актор: Користувач.

Опис: Користувач переміщує папку/файл до вказаного місця за допомогою команди.

Основний сценарій:

1. Користувач вводить команду для переміщення об’єкту.

2. Система перевіряє права доступу користувача до вихідної та цільової папки.
3. Система перевіряє, чи існує в цільовій папці об'єкт із таким самим ім'ям.
4. Якщо перевірки успішні, об'єкт переміщується.

Винятки:

- Якщо система не може перемістити об’єкт (через відсутність прав доступу, або повтору імені), система повідомляє про це користувача.

Примітки: Відсутні.

Прецедент 2: Видалення папок/файлів

Передумови: Користувач має права доступу на видалення.

Післяумови: Папка/файл видалені.

Актор: Користувач.

Опис: Користувач видаляє папку/файл у вказаному місці за допомогою команди.

Основний сценарій:

1. Користувач вводить команду для видалення об’єкту.
2. Система перевіряє права доступу користувача до об’єкту.
3. Якщо перевірка успішна, об'єкт видаляється.

Винятки:

- Якщо система не може видалити об’єкт (через відсутність прав доступу), система повідомляє про це користувача.

Примітки:

- Видалений файл переміщується до кошику.

Прецедент 3: Пошук файлів за назвою

Передумови: У файловій системі є файли заданої назви.

Післяумови: Список знайдених файлів, виведений у текстовому форматі.

Актор: Користувач.

Опис: Користувач знаходить файли заданої назви за допомогою команди.

Основний сценарій:

1. Користувач вводить команду для пошуку файлу.
2. Система шукає файл заданої назви.
3. Система виводить список знайдених файлів.

Винятки: Відсутні.

Примітки:

- Якщо нічого не знайдено, виводиться пустий список.

4. Діаграма класів

Діаграма класів зображена на рисунку 2.

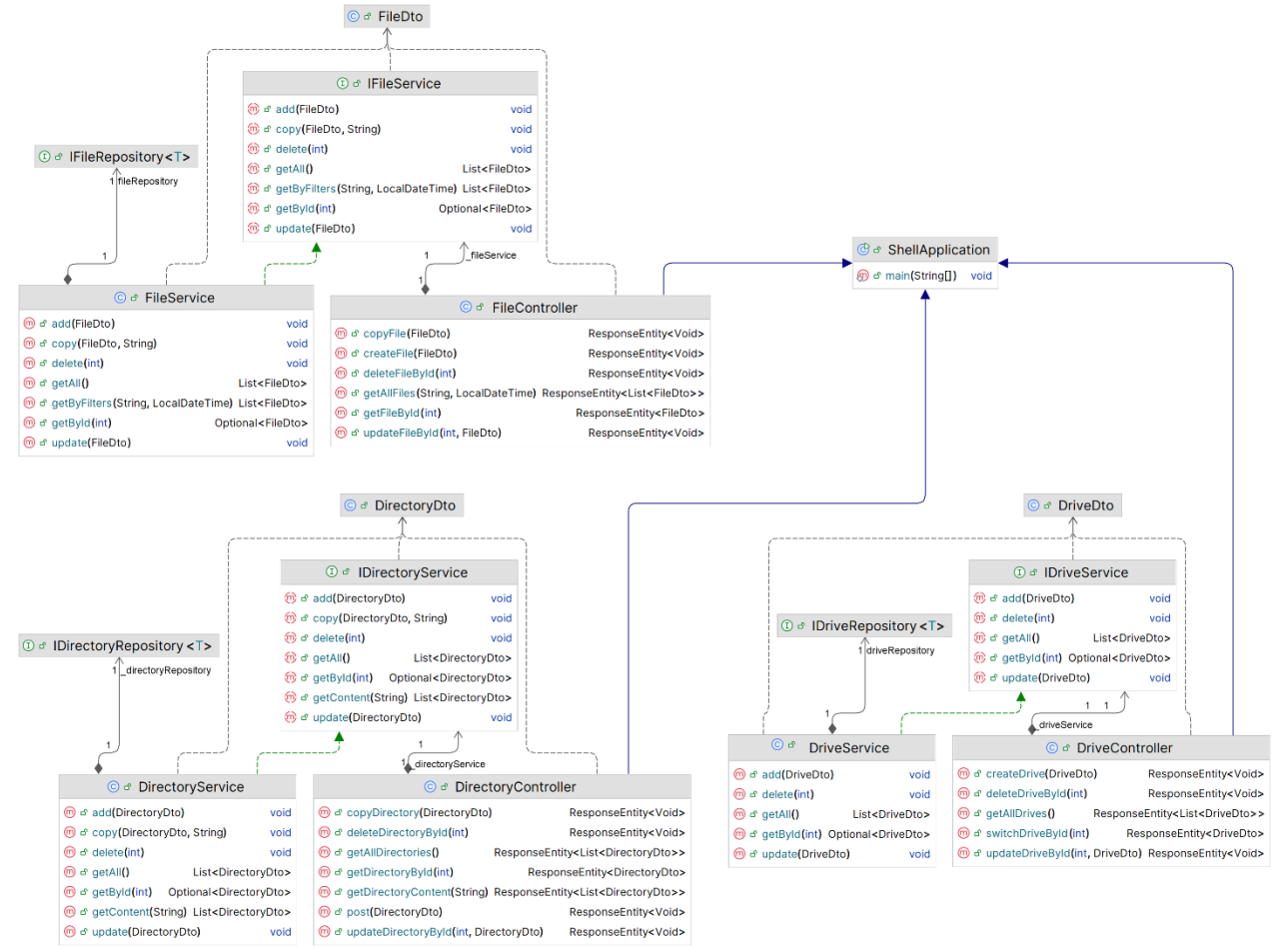

Рисунок 2 – діаграма класів

Дана діаграма класів відображає загальну структуру файлової оболонки, включаючи основні сервіси, контролери та репозиторії, що забезпечують її роботу.

- FileController, FileService, FileRepository: відповідають за керування файлами. FileService надає функціональність для створення, копіювання,

редагування, видалення та отримання файлів. FileRepository відповідає за взаємодію з базою даних, а FileController обробляє запити від користувача, передаючи їх на рівень сервісу.

- DirectoryController, DirectoryService, DirectoryRepository: відповідають за керування папками. DirectoryService надає функціональність для створення, копіювання, редагування, видалення та отримання папок. DirectoryRepository відповідає за взаємодію з базою даних, а DirectoryController обробляє запити від користувача, передаючи їх на рівень сервісу.
- DriveController, DriveService, DriveRepository: відповідають за керування дисками. DriveService надає основну функціональність і можливість перемикання між дисками. DriveRepository відповідає за взаємодію з базою даних, а DriveController обробляє запити від користувача, передаючи їх на рівень сервісу.
- ShellApplication: головний клас програми, що включає метод main() для запуску файлової оболонки. Він взаємодіє з іншими сервісами та забезпечує загальну координацію додатку.

Кожен контролер викликає відповідний сервіс для обробки запитів від користувача, а сервіси звертаються до репозиторіїв для взаємодії з базою даних або іншим сховищем.

Шаблон репозиторію зображено на рисунку 3.

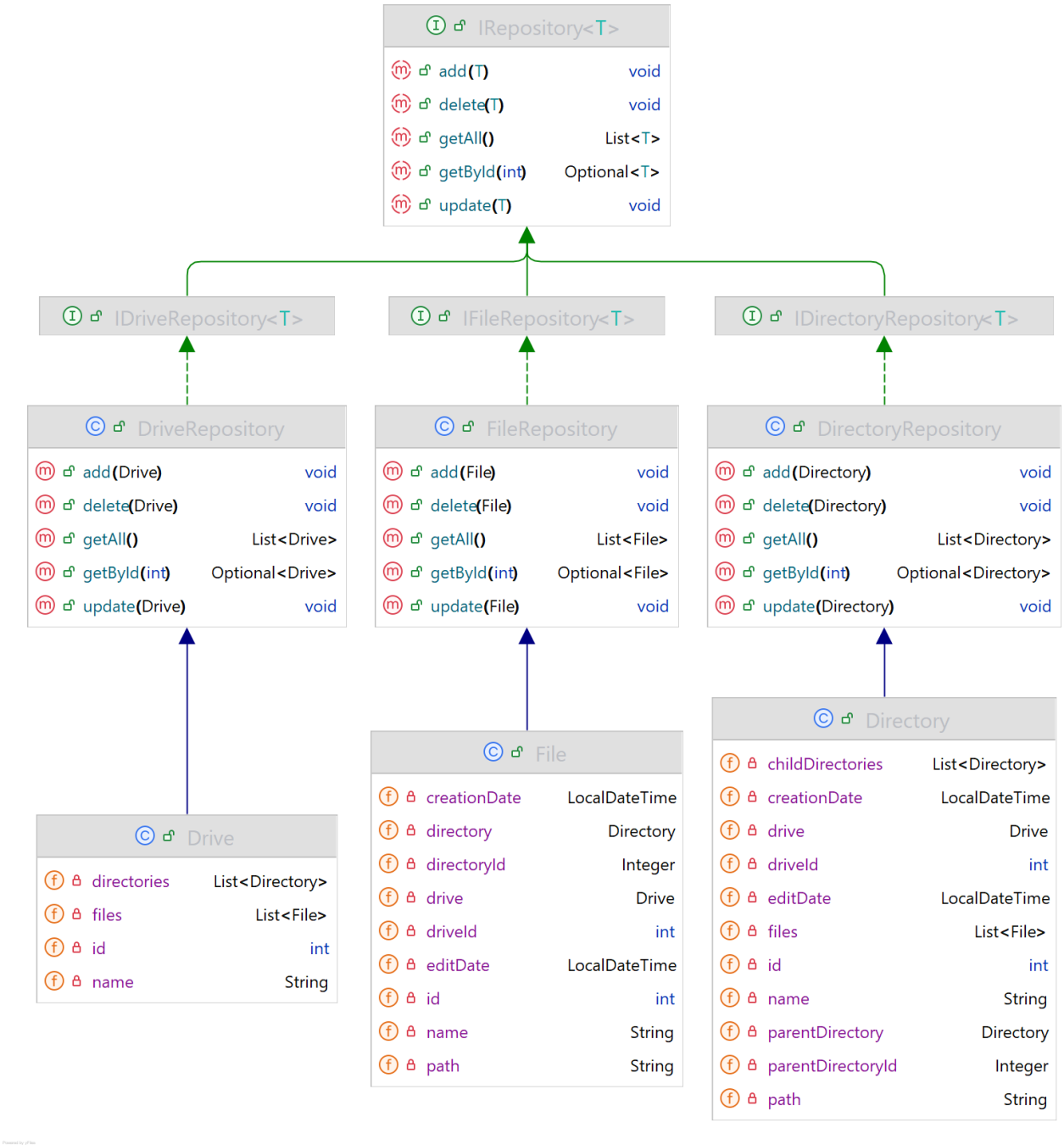

Рисунок 3 – шаблон репозиторію

На даній діаграмі зображений шаблон репозиторію, який використовується для взаємодії з базою даних.

Визначений загальний інтерфейс IRepository з спільними CRUD операціями від якого наслідуються конкретні інтерфейси IDriveRepository, IFileRepository, IDirectoryRepository.

Конкретні класи репозиторії DriveRepository, FileRepository, DirectoryRepository імплементують ці інтерфейси і реалізують функціональність для роботи з дисками, файлами, папками.

Тобто кожна сутність (Drive, File, Directory) має відповідний репозиторій, який відповідає за її взаємодію з базою даних.

5. Структура бази даних

Структура бази даних зображена на рисунку 4.

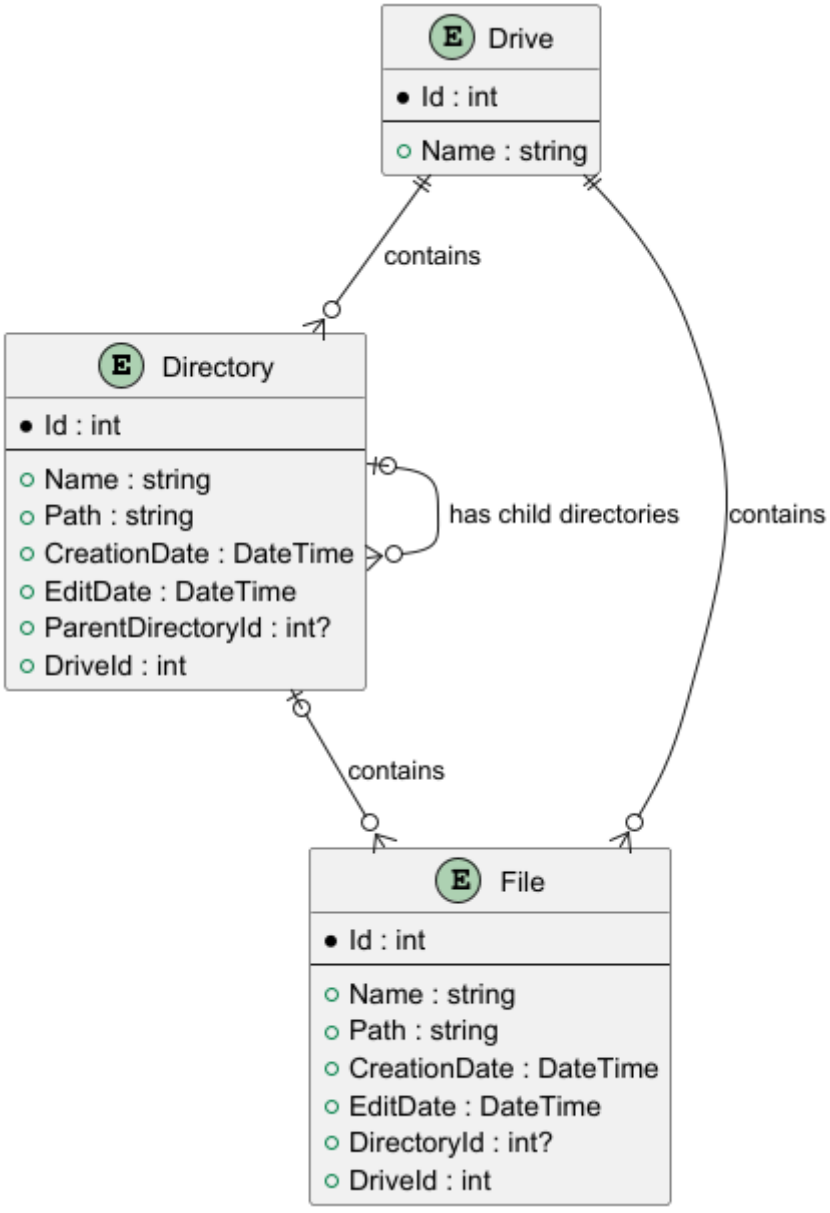

Рисунок 4 – структура бази даних

Дана діаграма відображає загальну модель бази даних файлової оболонки.

Основні сутності бази даних:

- File: описує файл у файловій системі, який має назву, шлях, дату створення та редагування, диск, та може бути у папці.
- Directory: описує папку у файловій системі, яка має назву, шлях, дату створення та редагування, диск, та може бути у папці, або мати дочірні папки.
- Drive: описує диск у файловій системі, який має назву та може містити у собі папки та файли.

Посилання на код: https://github.com/ivanchikk/trpz

Висновок: виконавши дану лабораторну роботу, я проаналізував тему, намалював схему прецедентів, діаграму класів, розробив основні класи і структуру бази.