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

Лабораторна робота №2
З дисципліни «Технології розроблення програмного забезпечення»
Тема: «Діаграма варіантів використання. Сценарії варіантів використання. Діаграми UML. Діаграми класів. Концептуальна модель системи»

Варіант 11

Виконала:
студентка групи ІА-31
Горлач Д. 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.

Зміст

Теоретичні відомості....3
Хід роботи:....4
Діаграма варіантів....4
Діаграма класів....8
Проєктування БД....9
Вихідні коди класів системи....9
Висновки:....11
Питання до лабораторної роботи....11

Завдання:

• Ознайомитись з короткими теоретичними відомостями.
• Проаналізувати тему та спроєктувати діаграму варіантів використання відповідно до обраної теми лабораторного циклу.
• Спроєктувати діаграму класів предметної області.
• Вибрати 3 варіанти використання та написати за ними сценарії використання.
• На основі спроєктованої діаграми класів предметної області розробити основні класи та структуру бази даних системи. Класи даних повинні реалізувати шаблон Repository для взаємодії з базою даних.
• Нарисувати діаграму класів для реалізованої частини системи.
• Підготувати звіт щодо виконання лабораторної роботи. Поданий звіт повинен містити: діаграму варіантів використання відповідно, діаграму класів системи, вихідні коди класів системи, а також зображення структури бази даних.

Хід роботи:

Теоретичні відомості

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.

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.

Діаграма варіантів використання (Use-Cases Diagram) – це UML діаграма за допомогою якої у графічному вигляді можна зобразити вимоги до системи, що розробляється. Діаграма варіантів використання – це вихідна концептуальна модель проєктованої системи, вона не описує внутрішню побудову системи.

Діаграми варіантів використання є відправною точкою при збиранні вимог до програмного продукту та його реалізації. Дана модель будується на аналітичному етапі побудови програмного продукту (збір та аналіз вимог) і дозволяє бізнес-аналітикам отримати більш повне уявлення про необхідне програмне забезпечення та документувати його.

Діаграма варіантів використання складається з низки елементів. Основними елементами є: варіанти використання або прецеденти (use case), актор або дійова особа (actor) та відносини між акторами та варіантами використання (relationship).

Сценарії використання – це текстові уявлення тих процесів, які відбуваються при взаємодії користувачів системи та самої системи. Вони є чітко формалізованими, покроковими інструкціями, що описують той чи інший процес у термінах кроків досягнення мети. Сценарії використання однозначно визначають кінцевий результат.

Діаграми класів використовуються при моделюванні програмних систем

найчастіше. Вони є однією із форм статичного опису системи з погляду її проєктування, показуючи її структуру [3]. Діаграма класів не відображає динамічної поведінки об'єктів зображених на ній класів. На діаграмах класів показуються класи, інтерфейси та відносини між ними.

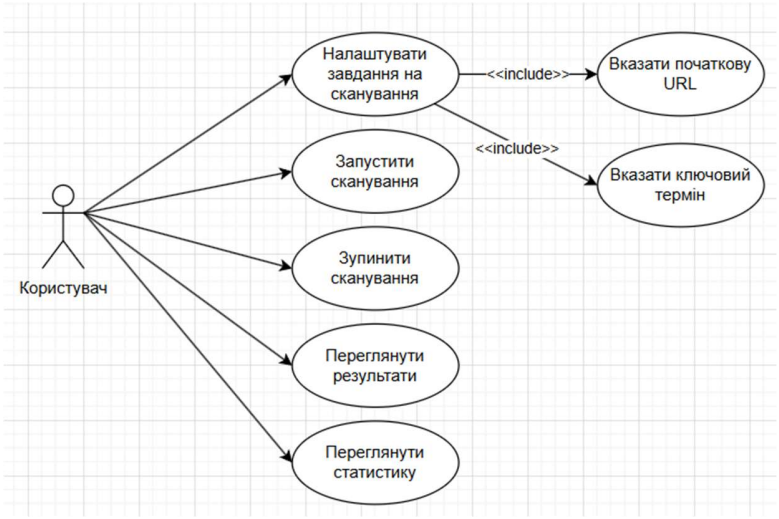
Клас – це основний будівельний блок програмної системи. Це поняття є і в мовах програмування, тобто між класами UML та програмними класами є відповідність, що є основою для автоматичної генерації програмних кодів або для виконання реінжинірингу. Кожен клас має назву, атрибути та операції. Клас на діаграмі показується як прямокутник, розділений на 3 області. У верхній міститься назва класу, у середній – опис атрибутів (властивостей), у нижній – назви операцій – послуг, що надаються об'єктами цього класу.

Логічна модель бази даних є структурою таблиць, уявлень, індексів та інших логічних елементів бази даних, що дозволяють власне програмування та використання бази даних. Процес створення логічної моделі бази даних зветься проєктування бази даних (database design). Проєктування відбувається у зв'язку з опрацюванням архітектури програмної системи, оскільки база даних створюється зберігання
даних, одержуваних з програмних класів.

Хід роботи:

Аналіз теми:
11 Web crawler (proxy, chain of responsibility, memento, template method, composite, p2p) Веб-сканер повинен вміти розпізнавати структуру сторінок сайту, переходити за посиланнями, збирати необхідну інформацію про зазначений термін, видаляти не семантичні одиниці (рекламу, об'єкти javascript і т.д.), зберігати знайдені дані у вигляді структурованого набору html файлів вести статистику відвіданих сайтів і метадані.

Діаграма варіантів



Діаграма варіантів

Сценарій 1: Налаштування та запуск сканування

Передумови:

1. Застосунок "Веб-сканер" запущено на робочій станції користувача.
2. Користувач має початкову URL-адресу для сканування та ключовий термін.

Постумови:

1. Налаштування (URL, термін) збережені в системі.
2. Фоновий процес сканування (CrawlerService) ініційовано.
3. Початкова URL додана до черги на обхід у Storage.

Взаємодіючі сторони:

1. Користувач — вводить дані та ініціює процес.
2. Застосунок "Веб-сканер" (GUI) — отримує дані від користувача.
3. CrawlerService — отримує команду на запуск.

Короткий опис: Користувач вводить початкові параметри сканування та дає команду на запуск, після чого система починає фоновий процес обходу.

Основний потік подій:

1. Користувач вводить початкову URL та ключовий термін в інтерфейсі Застосунку "Веб-сканер".
2. Застосунок "Веб-сканер" зберігає ці налаштування.

3. Користувач натискає кнопку "Запустити сканування".
4. Застосунок "Веб-сканер" надсилає команду "Початок сканування" компоненту CrawlerService.
5. CrawlerService звертається до Storage (БД SQLite) та додає початкову URL до черги на обхід.

Сценарій 2: Обхід сайту та завантаження сторінок
Передумови:

1. Процес сканування запущено (CrawlerService активний).
2. У черзі в Storage є принаймні одна URL-адреса для обходу.

Постумови:

1. HTML-вміст сторінки успішно отримано від цільового сервера.
2. URL-адреса позначена як "відвідана" (або видалена з черги).

Взаємодіючі сторони:

1. CrawlerService — керує процесом.
2. Storage — надає URL-адреси з черги.
3. Downloader — виконує HTTP-запити.
4. External Server (Цільовий сервер) — повертає дані.

Короткий опис: Сервіс сканування бере URL з черги, передає її завантажувачу, який робить HTTP-запит і повертає отриманий HTML-вміст.
Основний потік подій:

1. CrawlerService бере наступну URL-адресу з черги у Storage.
2. CrawlerService передає цю URL компоненту Downloader.
3. Downloader формує та надсилає HTTP GET-запит до цільового External Server.
4. External Server обробляє запит і повертає Downloader-у відповідь (HTML, CSS, content).
5. Downloader отримує дані та повертає чистий HTML-вміст компоненту CrawlerService.

Сценарій 3: Парсинг, фільтрація та збереження результатів

Передумови:

1. CrawlerService отримав HTML-вміст сторінки від Downloader.

Постумови:

1. HTML-вміст проаналізовано, з нього вилучено нові посилання.
2. Нові, унікальні посилання додано до черги у Storage.
3. Вміст очищено від несемантичних елементів (<script>, <style>).
4. Якщо ключовий термін знайдено, очищений HTML-файл збережено у файловій системі.

Взаємодіючі сторони:

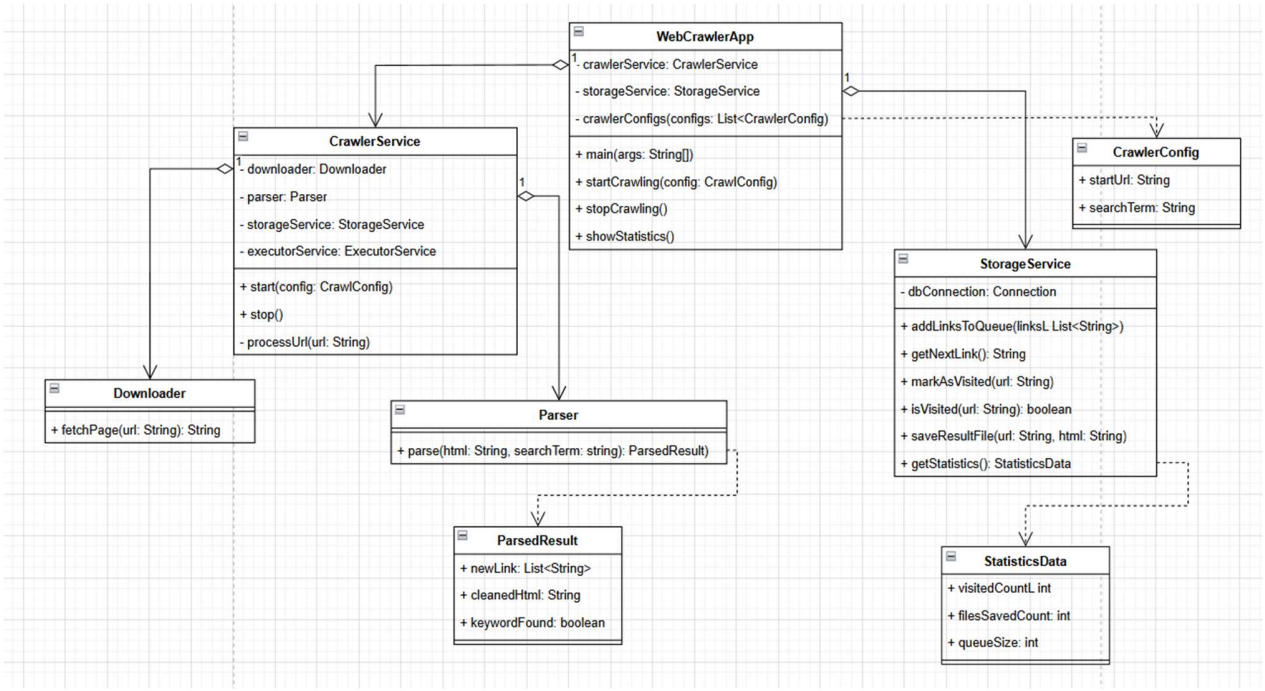
1. CrawlerService — координує обробку.
2. Parser — аналізує, фільтрує та шукає термін.
3. Storage — зберігає нові посилання (БД) та файли (файлова система).

Короткий опис: Отриманий HTML-вміст передається парсеру, який вилучає нові посилання, очищує контент від "шуму" та перевіряє наявність ключового терміну для подальшого збереження.

Основний потік подій:

1. CrawlerService надсилає отриманий HTML-документ компоненту Parser.
2. Parser аналізує HTML, вилучає всі гіперпосилання (<a> теги).
3. Parser очищує HTML від тегів <script>, <style> та ін.
4. Parser шукає вказаний ключовий термін в очищеному контенті.
5. Parser повертає CrawlerService "результат парсингу" (список нових посилань та очищений HTML, якщо термін знайдено).
6. CrawlerService передає нові посилання до Storage (БД SQLite) для додавання в чергу (з перевіркою на унікальність).
7. Якщо термін було знайдено, CrawlerService передає очищений HTML до Storage для збереження у файловій системі.

Діаграма класів

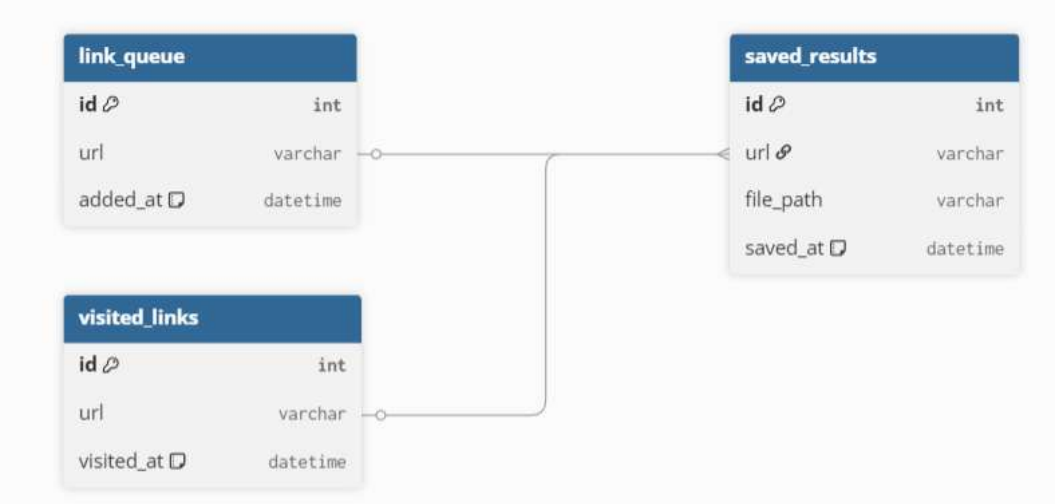


Діаграма класів

На основі діаграми класів, центральним елементом інтерфейсу користувача є клас WebCrawlerApp. Він виконує роль точки входу для користувача, ініціює запуск (startCrawling) та зупинку (stopCrawling) процесу, а також відображає зібрані дані (showStatistics). Вся основна логіка сканування делегується класу CrawlerService, який виступає координатором фонових процесів. WebCrawlerApp також отримує дані для відображення статистики безпосередньо від StorageService.
Обробка веб-сторінок відбувається за участю кількох ключових класів-виконавців. CrawlerService ініціює процес для кожної окремої URL-адреси. Він використовує клас Downloader для виконання HTTP-запитів та завантаження HTML-вмісту сторінки. Отриманий HTML передається класу Parser, який відповідає за його аналіз: вилучення нових гіперпосилань, очищення від несемантичних тегів (<script>, <style>) та пошук вказаного ключового терміну. Результати парсингу повертаються у вигляді об'єкта-контейнера ParsedResult.
Конфігурація системи здійснюється через об'єкт CrawlConfig, який WebCrawlerApp створює та передає у CrawlerService при старті. Цей об'єкт містить початкову URL та ключовий термін. Система зберігання даних реалізована через клас StorageService, який інкапсулює всю логіку взаємодії з постійною пам'яттю. Він керує підключенням до бази даних **SQLite** (dbConnection) для управління чергою URL-адрес (addLinksToQueue, getNextLink) та списком відвіданих сторінок

(isVisited, markAsVisited). Цей же сервіс відповідає за збереження очищених HTML-файлів (де знайдено термін) у файлову систему (saveResultFile).
Модуль статистики також є частиною StorageService. Він збирає дані про процес сканування (кількість відвіданих сторінок, кількість збережених файлів, розмір черги) та повертає їх у вигляді об'єкта StatisticsData на запит від WebCrawlerApp.
Рівень паралелізму забезпечується класом CrawlerService через використання ExecutorService (пулу потоків), який не показаний на діаграмі як поле, але мається на увазі в логіці роботи. Це дозволяє одночасно завантажувати та обробляти декілька сторінок, що є критичним для високої продуктивності сканера.
Мережевий рівень повністю інкапсульований у класі Downloader.
Загалом діаграма класів демонструє чіткі зв’язки між компонентами, що забезпечують високу модульність та чіткий розподіл обов’язків. Центральна координуюча роль CrawlerService та виділення окремих класів для специфічних завдань (Downloader, Parser, StorageService) дозволяють легко адаптувати систему до нових вимог. Завдяки такому розподілу, де логіка сканування, робота з мережею, парсинг та зберігання даних чітко розділені, застосунок залишається гнучким і зручним у підтримці.

Проєктування БД



Вихідні коди класів системи

```
public class LinkQueue {
    private int id;
```

```
    private String url;
    private String addedAt;

    public LinkQueue(int id, String url, String addedAt) {
        this.id = id;
        this.url = url;
        this.addedAt = addedAt;
    }

    public int getId() { return id; }
    public String getUrl() { return url; }
    public String getAddedAt() { return addedAt; }
}


public class VisitedLink {
    private int id;
    private String url;
    private String visitedAt;

    public VisitedLink(int id, String url, String visitedAt) {
        this.id = id;
        this.url = url;
        this.visitedAt = visitedAt;
    }

    public int getId() { return id; }
    public String getUrl() { return url; }
    public String getVisitedAt() { return visitedAt; }
}

public class SavedResult {
    private int id;
    private String url;
    private String filePath;
    private String savedAt;

    public SavedResult(int id, String url, String filePath, String savedAt) {
```

```
        this.id = id;
        this.url = url;
        this.filePath = filePath;
        this.savedAt = savedAt;
    }

    public int getId() { return id; }
    public String getUrl() { return url; }
    public String getFilePath() { return filePath; }
    public String getSavedAt() { return savedAt; }
}
```

Висновки: під час виконання лабораторної роботи, ми навчилися основам проектування, обрали зручну систему побудови UML-діаграм та навчилися будувати діаграми варіантів використання для системи що проєктується, розробляти сценарії варіантів використання та будувати діаграми класів предметної області.

Питання до лабораторної роботи

1. Що таке UML?

UML (Unified Modeling Language) — це уніфікована мова моделювання, яка використовується для специфікації, візуалізації, конструювання та документування артефактів програмних систем. Це стандарт, що дозволяє розробникам і архітекторам описувати структуру та поведінку системи за допомогою графічних нотацій.

2. Що таке діаграма класів UML?

Це діаграма структури, яка описує систему шляхом показу її класів, їхніх атрибутів, методів (операцій) та зв'язків між цими класами. Вона відображає статичний вигляд системи.

3. Які діаграми UML називають канонічними?

Канонічними називають основні типи діаграм, що охоплюють різні аспекти системи. Згідно зі стандартом UML 2.0, це:

Діаграма варіантів використання (Use Case).

Діаграма класів (Class).

Діаграми взаємодії (Sequence, Communication).

Діаграма станів (State Machine).

Діаграма діяльності (Activity).
Діаграма компонентів (Component).
Діаграма розгортання (Deployment).

4. Що таке діаграма варіантів використання?

Це діаграма поведінки, яка описує функціональність системи через взаємодію зовнішніх користувачів (акторів) із системою для досягнення певних цілей (варіантів використання).

5. Що таке варіант використання (Use Case)?

Це опис послідовності дій (сценаріїв), які виконує система у відповідь на запит актора, що приносить певний відчутний результат для цього актора.

6. Які відношення можуть бути відображені на діаграмі використання?

Асоціація (Association): зв'язок між актором і варіантом використання.
Включення (<<include>>): один варіант використання обов'язково викликає інший.
Розширення (<<extend>>): додаткова функціональність, яка виконується лише за певних умов.
Узагальнення (Generalization): відношення "батько-нащадок" між акторами або варіантами використання.

7. Що таке сценарій?

Це конкретна послідовність кроків (подій), що описує взаємодію актора з системою в межах одного варіанта використання. Один Use Case може мати основний сценарій (успішний шлях) та альтернативні (обробка помилок або винятків).

Діаграми класів та об'єктно-орієнтоване моделювання

8. Що таке діаграма класів?

Це головний інструмент моделювання структури ПЗ, що визначає типи об'єктів у системі та їхні статичні стосунки.

9. Які зв’язки між класами ви знаєте?

Асоціація: загальний зв'язок ("знає про").
Агрегація: зв'язок "частина-ціле", де частина може існувати без цілого.
Композиція: сувора форма агрегації, де частина не може існувати без цілого.
Узагальнення (Inheritance): успадкування від батьківського класу.

Залежність (Dependency): зміна в одному класі може вплинути на інший.
Реалізація: зв'язок між інтерфейсом та класом, що його реалізує.

10. Чим відрізняється композиція від агрегації?

Агрегація (слабкий зв'язок): якщо "ціле" (клас-контейнер) буде знищено, "частини" (об'єкти всередині) продовжують існувати. Приклад: Університет та Студенти.
Композиція (сильний зв'язок): "ціле" повністю володіє своїми "частинами". При знищенні цілого знищуються і всі його частини. Приклад: Книга та її Сторінки.

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

Агрегація позначається лінією з порожнім (білим) ромбом на боці цілого класу.
Композиція позначається лінією з зафарбованим (чорним) ромбом на боці цілого класу.

Проєктування баз даних та зв'язок з кодом

12. Що являють собою нормальні форми баз даних?

Це набір правил (критеріїв), яким повинна відповідати структура бази даних для усунення надмірності (дублювання) даних та запобігання аномаліям оновлення. Основні форми:
1НФ: Атомарність значень (немає масивів/списків у комірках).
2НФ: Відповідність 1НФ + кожен неключовий атрибут залежить від усього первинного ключа.
3НФ: Відповідність 2НФ + відсутність транзитивних залежностей (неключові поля не залежать від інших неключових полів).
13. Що таке фізична модель бази даних? Логічна?
Логічна модель: описує структуру даних (таблиці, поля, типи, зв'язки) без прив'язки до конкретної СУБД. Вона фокусується на бізнес-логіці зберігання.
Фізична модель: це реалізація логічної моделі в конкретній СУБД (PostgreSQL, MySQL тощо). Вона включає специфічні типи даних, індекси, обмеження (constraints), тригери та налаштування зберігання.

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

У сучасному програмуванні цей зв'язок реалізується через ORM (Object-Relational Mapping):
Таблиця БД зазвичай відповідає Класу.

Рядок у таблиці відповідає Об'єкту (екземпляру класу).
Стовпець (поле) таблиці відповідає Атрибуту (властивості) класу.
Зв'язки між таблицями (Foreign Keys) відповідають Посиланням (асоціаціям) між об'єктами в коді.