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

Технології розроблення програмного забезпечення

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

Виконав студент групи ІА-23:

Каширов Д. О.

Перевірив:

Мягкий Михайло Юрійович

Київ 2024.

Тема роботи:

Навчитися розробляти діаграму прецедентів, діаграму класів системи, вихідні коди класів системи, а також зображення структури бази даних.

Варіант:

..6 Web-browser (proxy, chain of responsibility, factory method, template method, visitor, p2p) Веб-браузер повинен мати можливість зробити наступне: мати адресний рядок для введення адреси сайту, переміщатися і відображати структуру html документа, переглядати підключений javascript та css файли, перегляд всіх підключених ресурсів (зображень), коректна обробка відповідей з сервера (коди відповідей HTTP) - переходи при перенаправленнях, відображення сторінок 404 і 502/503.

Завдання:

1. Ознайомитися з короткими теоретичними відомостями.

2. Проаналізуйте тему та намалюйте схему прецеденту, що відповідає обраній темі лабораторії.

3. Намалюйте діаграму класів для реалізованої частини системи.

4. Виберіть 3 прецеденти і напишіть на їх основі прецеденти.

5. Розробити основні класи і структуру системи баз даних.

6. Класи даних повинні реалізувати шаблон Репозиторію для взаємодії з базою даних.

7. Підготувати звіт про хід виконання лабораторних робіт. Звіт, що подається повинен містити:

діаграму прецедентів, діаграму класів системи, вихідні коди класів системи, а також

зображення структури бази даних.

Теоретичні відомості

• **Діаграма варіантів використання (Use Case Diagram)** – це тип діаграми UML, що описує функціональність системи з точки зору її користувачів (акторів) і взаємодії між ними та системою. Вона показує, які дії (варіанти використання) можуть виконуватися користувачами, але не вдається у внутрішні механізми їх реалізації.

• **Сценарії варіантів використання (Use Case Scenarios)** – це текстовий опис варіантів використання, де детально викладається, як система повинна реагувати на дії користувачів у кожній конкретній ситуації. Включає в себе основний потік подій та альтернативні шляхи розвитку сценарію.

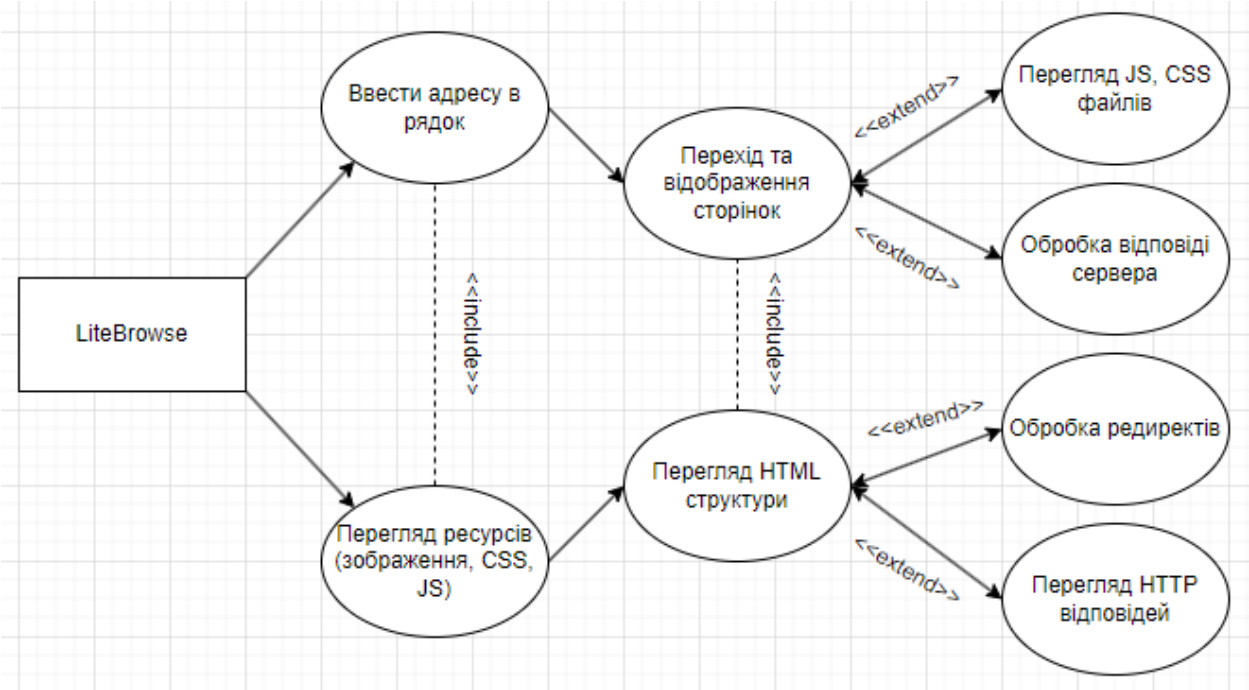
• **Діаграма класів (Class Diagram)** – це структура, яка моделює класи системи, їх властивості, методи, а також зв'язки між ними. Класи представляють основні об'єкти системи, які мають атрибути та операції, а також відображають взаємодію між різними компонентами.

• **Концептуальна модель системи** – це абстрактне представлення об'єктів та зв'язків між ними, що відображає ключові аспекти системи з точки зору бізнесу або предметної області. Вона описує основні компоненти, їх взаємодію та структуру, але не деталізує технічну реалізацію.

Ці діаграми дозволяють аналізувати вимоги до системи та планувати її розробку.

Хід Роботи:

Діаграма прецедентів:



Пояснення:

1. Основний компонент – LiteBrowse (в центрі).
2. Дії включають Ввести адресу в рядок (для введення URL), Перехід та відображення сторінок HTML, Перегляд ресурсів (зображення, CSS, JavaScript) та Обробка відповідей сервера (наприклад, 404, 502/503 помилки).
3. Також є зв'язки для Перегляду HTML структури та Обробки редиректів.

Ця діаграма є базовим представленням функціоналу LiteBrowse, і ви можете деталізувати її залежно від вимог до кожної функції.

❖ **Предеценти на основі трьох прецедентів:**

❖ **Прецедент 1: Введення URL у адресний рядок**

1. Основний потік подій
2. Користувач відкриває програму "LiteBrowse".
3. Користувач вводить URL-адресу в адресний рядок.
4. Система перевіряє правильність формату введеної URL-адреси.
5. Якщо формат коректний, система передає URL для завантаження сторінки.

❖ **Альтернативний потік подій**

1. Якщо користувач вводить URL без префікса ("http://" або "https://"), система автоматично додає "http://" на початок.

❖ **Винятки**

1. Якщо введений формат URL некоректний (наприклад, відсутній домен чи доменний суфікс), система відображає повідомлення про помилку і просить користувача повторити введення.

❖ **Прецедент 2: Завантаження веб-сторінки**

1. Основний потік подій
2. Система приймає URL від користувача.
3. Система надсилає HTTP-запит до зазначеного серверу.
4. Система очікує на відповідь від сервера і отримує HTML-контент сторінки.
5. Система рендерить контент сторінки і відображає її користувачу.

❖ **Альтернативний потік подій**

1. Якщо завантаження сторінки триває занадто довго (наприклад, понад 5 секунд), система виводить індикатор завантаження для зручності користувача.

4. Винятки

1. Якщо сервер не відповідає протягом встановленого часу (timeout), система видає повідомлення про неможливість завантажити сторінку і пропонує користувачу спробувати пізніше.
2. Якщо сервер повертає помилку (наприклад, HTTP 404 або 503), система переходить до обробки помилок HTTP.

❖ **Прецедент 3: Обробка помилок HTTP**

1. Основний потік подій

2. Система отримує відповідь із помилкою HTTP від сервера.

3. Система визначає код помилки (наприклад, 404, 502, 503).

4. Відповідно до коду помилки, система відображає інформативне повідомлення про тип помилки.

5. Система пропонує можливі дії (наприклад, спробувати перезавантажити сторінку або перевірити URL).

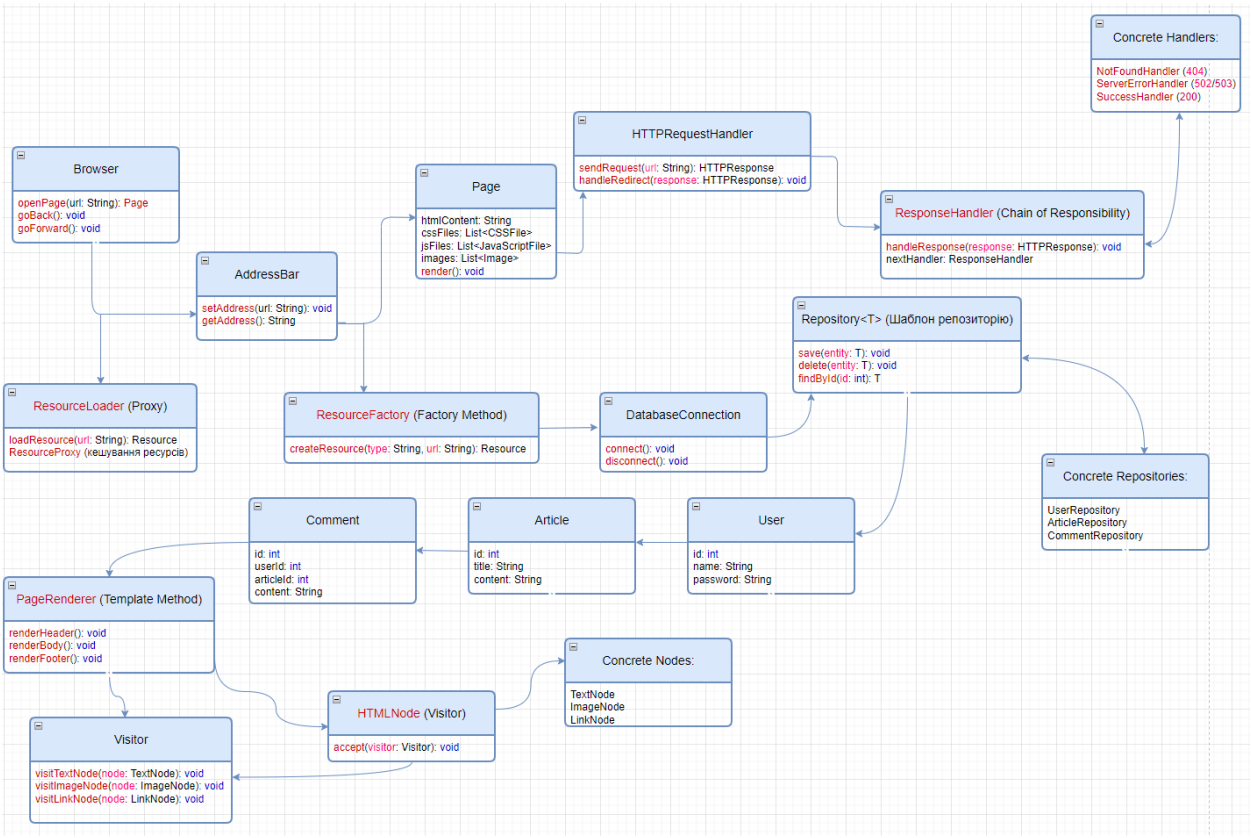
❖ **Альтернативний потік подій**

1. Для помилок типу 404 система пропонує користувачеві перевірити URL на наявність друкарських помилок або відвідати домашню сторінку.

❖ **Винятки**

1. Якщо відповідь не містить інформації про код помилки або її неможливо обробити, система видає загальне повідомлення про технічну помилку й пропонує користувачу звернутися до технічної підтримки.

Діаграма класів:



Основні компоненти діаграми:

1. **Browser:** Основний клас браузера, який забезпечує навігацію між сторінками.
    - openPage(url: String): Page: відкриває сторінку за вказаною URL-адресою.
    - goBack(): void: повертає на попередню сторінку.
    - goForward(): void: переходить на наступну сторінку.
2. **AddressBar:** Компонент для введення та збереження адреси.
    - setAddress(url: String): void: встановлює URL-адресу.
    - getAddress(): String: повертає поточну адресу.
3. **Page:** Представляє веб-сторінку.
    - htmlContent: вміст HTML-сторінки.
    - cssFiles: список підключених CSS-файлів.
    - jsFiles: список підключених JavaScript-файлів.

- images: список зображень на сторінці.
- render(): void: метод для відображення сторінки.

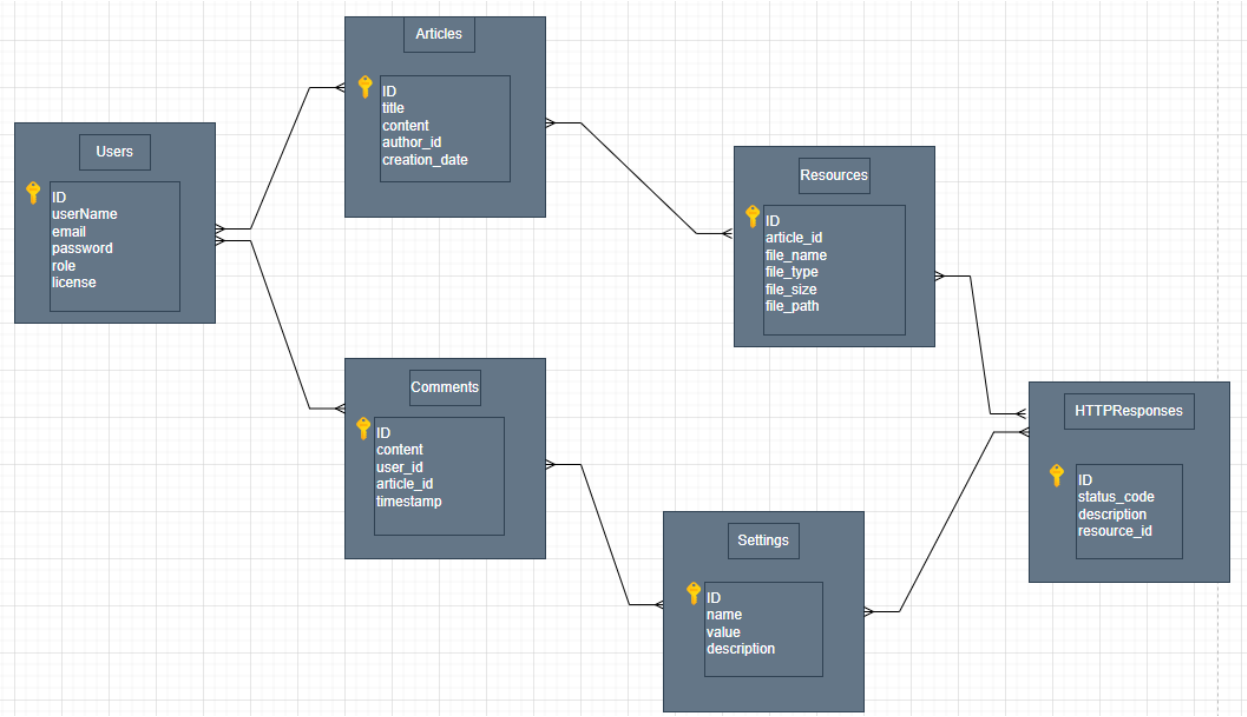
4. **HTTPRequestHandler:** Обробник HTTP-запитів.
   - sendRequest(url: String): HTTPResponse: відправляє запит на сервер.
   - handleRedirect(response: HTTPResponse): void: обробляє перенаправлення.
5. **ResponseHandler (Chain of Responsibility):** Ланцюг обробки відповідей від сервера.
   - handleResponse(response: HTTPResponse): void: обробляє відповідь від сервера.
   - nextHandler: наступний обробник у ланцюгу.
   - Concrete Handlers: конкретні обробники для різних статус-кодів HTTP:
     - NotFoundHandler (404): обробляє 404 Not Found.
     - ServerErrorHandler (502/503): обробляє серверні помилки 502 і 503.
     - SuccessHandler (200): обробляє успішні відповіді 200.
6. **ResourceLoader (Proxy):** Клас для завантаження ресурсів.
   - loadResource(url: String): Resource: завантажує ресурс за вказаною URL-адресою.
   - ResourceProxy: кешує ресурси, щоб уникнути повторного завантаження.
7. **ResourceFactory (Factory Method):** Фабричний метод для створення ресурсів.
   - createResource(type: String, url: String): Resource: створює ресурс залежно від його типу.
8. **DatabaseConnection:** Компонент для керування з'єднанням з базою даних.
   - connect(): void: встановлює з'єднання з базою даних.
   - disconnect(): void: завершує з'єднання з базою даних.

9. **Repository<T> (Шаблон репозиторію):** Загальний клас для доступу до бази даних.
    - save(entity: T): void: зберігає об'єкт у базі даних.
    - delete(entity: T): void: видаляє об'єкт з бази даних.
    - findById(id: int): T: знаходить об'єкт за ідентифікатором.
    - Concrete Repositories:
        - UserRepository: репозиторій для користувачів.
        - ArticleRepository: репозиторій для статей.
        - CommentRepository: репозиторій для коментарів.
10. **User:** Клас користувача.
    - id: ідентифікатор користувача.
    - name: ім'я користувача.
    - password: пароль користувача.
11. **Article:** Клас для статті.
    - id: ідентифікатор статті.
    - title: заголовок статті.
    - content: вміст статті.
12. **Comment:** Клас для коментарів.
    - id: ідентифікатор коментаря.
    - userId: ідентифікатор користувача, який залишив коментар.
    - articleId: ідентифікатор статті, до якої належить коментар.
    - content: текст коментаря.
13. **PageRenderer (Template Method):** Клас для відображення сторінки за шаблоном.
    - renderHeader(): void: рендерить заголовок сторінки.
    - renderBody(): void: рендерить основний контент сторінки.
    - renderFooter(): void: рендерить нижню частину сторінки.

14. **HTMLNode (Visitor):** Абстрактний клас для елементів HTML-сторінки, які підтримують патерн Visitor.
    - accept(visitor: Visitor): void: приймає відвідувача.
    - Concrete Nodes:
        - TextNode: текстовий вузол.
        - ImageNode: вузол для зображень.
        - LinkNode: вузол для посилань.
15. **Visitor:** Відвідувач для HTML-вузлів.
    - visitTextNode(node: TextNode): void: відвідує текстовий вузол.
    - visitImageNode(node: ImageNode): void: відвідує вузол зображення.
    - visitLinkNode(node: LinkNode): void: відвідує вузол посилання.

Ця структура забезпечує функціональність веб-браузера, включаючи відкриття сторінок, завантаження ресурсів, обробку запитів, збереження даних у базі даних та використання патернів проектування, таких як Proxy, Chain of Responsibility, Factory Method, Template Method та Visitor.

Структура бази даних:



Висновок:

У даній роботі було розроблено структуру базових компонентів веб-браузера, які взаємодіють для забезпечення повноцінного функціоналу. В процесі проектування використано ключові патерни проектування, зокрема Proxy для оптимізації завантаження ресурсів, Chain of Responsibility для обробки HTTP-відповідей, Factory Method для створення ресурсів, Template Method для рендерингу сторінок та Visitor для роботи з HTML-вузлами. Створення структурованих класів для управління сторінками, запитами, ресурсами та з'єднанням з базою даних підвищує модульність і спрощує підтримку системи. Така архітектура дозволяє легко розширювати функціонал браузера та забезпечує надійну й ефективну роботу з даними.