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 №3
Технологія розроблення програмного забезпечення

«*Діаграма розгортання. Діаграма компонентів. Діаграма взаємодій та послідовностей.*»

Варіант 11

Виконала
студентка групи ІА-24
Тильна Марія Сергіївна

Перевірив:
Мягкий Михайло Юрійович

Київ 2024р.

Зміст

Мета: навчитися створювати діаграми розгортання, компонентів та послідовностей у нотації UML для моделювання програмних систем.

Теоретичні відомості

Діаграма розгортання

Діаграма розгортання показує фізичне розташування програмних компонентів на апаратному забезпеченні системи. Вона відображає вузли (пристрої або сервери), де запускаються програмні модулі, та зв'язки між ними. Основною метою діаграми розгортання є візуалізація того, як система буде працювати в реальному середовищі. Вона демонструє, на яких пристроях або серверах розміщуються різні частини системи (клієнтська частина, сервер додатків, база даних) та як вони взаємодіють між собою. Це корисно для розуміння фізичної архітектури системи, її масштабованості та ефективності.

Діаграма компонентів

Діаграма компонентів використовується для відображення логічної структури програмного забезпечення та показує, як різні модулі або компоненти системи взаємодіють між собою. Компоненти — це окремі частини програмного забезпечення, які виконують конкретні функції та можуть взаємодіяти через інтерфейси. Ця діаграма допомагає зрозуміти структуру системи, які модулі вона містить, та як ці модулі обмінюються інформацією. Використовується для документування системи на етапі проектування та дає уявлення про логічну організацію коду.

Діаграма послідовностей

Діаграма послідовностей показує, як об'єкти системи взаємодіють між собою в хронологічному порядку для виконання певної операції. Вона фокусується на тому, які повідомлення передаються між компонентами та в якому порядку. Це корисний інструмент для моделювання динамічних аспектів системи, таких як процес реєстрації користувача або збереження даних. Діаграма допомагає зрозуміти, як відбувається взаємодія між клієнтом, сервером та базою даних у реальному часі, забезпечуючи чітке уявлення про порядок виконання операцій.

Завдання

1. Ознайомитися з короткими теоретичними відомостями.
2. Розробити діаграму розгортання для проектованої системи.
3. Розробити діаграму компонентів для проектованої системи.
4. Розробити діаграму послідовностей для проектованої системи.
5. Скласти звіт про виконану роботу

..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.

.

Хід роботи

1. Діаграма розгортання

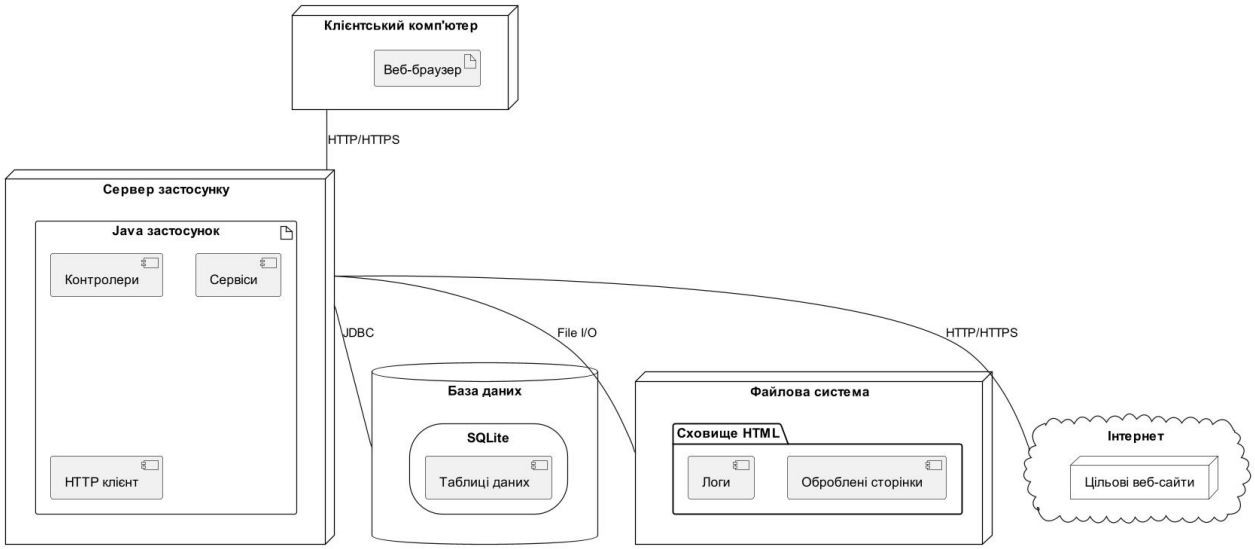

Рис.1 Діаграма розгортання

Ця діаграма відображає структуру розгортання додатку WebCrawler. Вона демонструє основні компоненти системи, їх зв’язки, та взаємодію між різними вузлами розгортання.

Основні елементи діаграми:

1. Клієнтський комп'ютер

- **Клієнтський комп'ютер** є кінцевим вузлом, з якого користувач взаємодіє із веб-сканером через веб-браузер.
- **Веб-браузер** — це інтерфейс для взаємодії користувача з веб-сканером.

2. Сервер застосунку

- **Сервер застосунку (appServer)** — це сервер, на якому розгортається головний застосунок веб-сканера.
- **Java застосунок** — представляє основний бекенд, що обробляє логіку.
- **Сервіси, контролери, HTTP клієнт**:
  - **Контролери** — забезпечують інтерфейс між користувацьким інтерфейсом і логікою.
  - **Сервіси** — містять бізнес-логіку.
  - **HTTP клієнт** — модуль, що обробляє HTTP-запити до цільових веб-сайтів.

3. База даних

- **SQLite** використовується для зберігання даних, таких як:
- **Таблиці даних** — представляють основні структури даних у БД.

4. Файлова система

- Включає **HTML сховище** та **логи**:
  - **HTML сховище** — для зберігання оброблених сторінок, які були завантажені і оброблені.
  - **Логи** — для зберігання інформації про роботу сканера, помилки та інші важливі дані.

5. Цільові веб-сайти

- **Цільові веб-сайти** — це зовнішні вузли, до яких під'єднується веб-сканер через HTTP/HTTPS для збору інформації.

Зв'язки між компонентами:

1. Клієнтський комп'ютер ↔ Сервер застосунку

- Зв'язок здійснюється через **HTTP/HTTPS**, що дозволяє користувачу взаємодіяти з веб-сканером.

2. Сервер застосунку ↔ База даних

- Зв'язок здійснюється за допомогою **JDBC**, щоб зберігати й отримувати дані.

3. Сервер застосунку ↔ Файлова система

- Здійснюється через **File I/O**, що дозволяє зберігати оброблені сторінки та логи.

4. Сервер застосунку ↔ Інтернет

- **HTTP/HTTPS** використовується для сканування цільових веб-сайтів

2. Діаграма компонетів

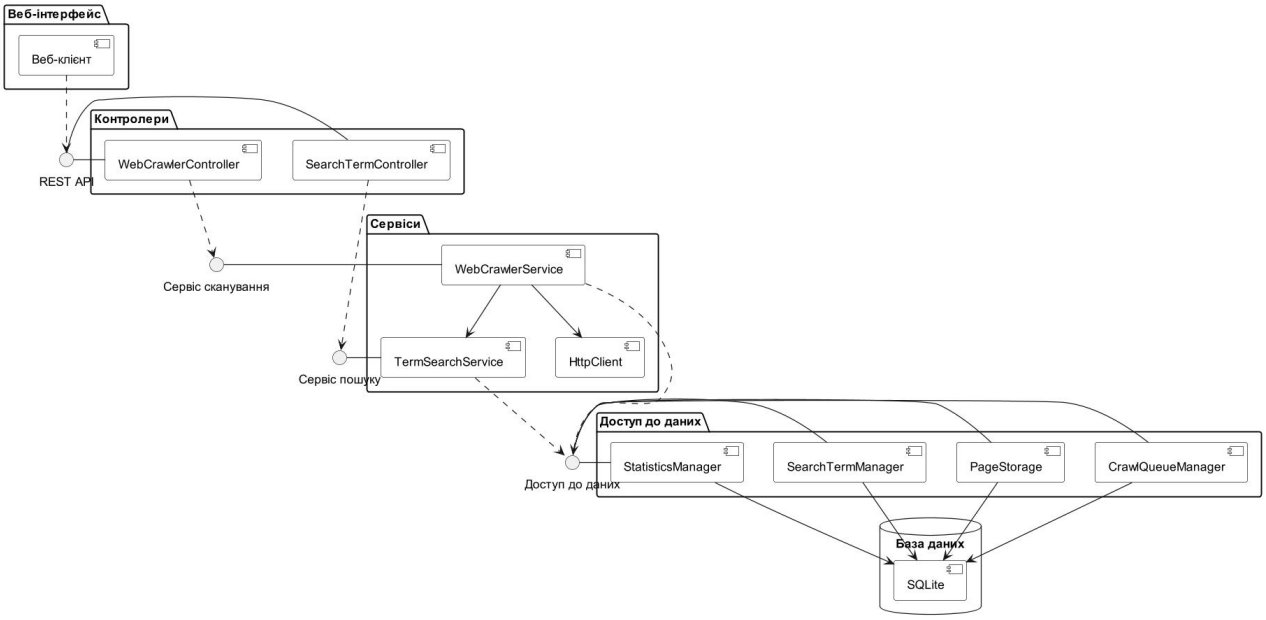

Рис.2 Діаграма компонентів

Інтерфейси

REST API

1. Використовується як механізм зв'язку між веб-клієнтом (користувацьким інтерфейсом) і сервером.
2. Дозволяє **Клієнту** взаємодіяти із серверними компонентами через HTTP/HTTPS.

Сервіс сканування (crawlService)

1. Інтерфейс, який об'єднує основні функціональності сканування веб-сторінок.

Сервіс пошуку (searchService)

1. Відповідає за пошук необхідних термінів у сторінках, які були оброблені.

Доступ до даних (dataAccess)

1. Інтерфейс для взаємодії з базою даних і доступу до даних (SQLite).
2. Включає логіку взаємодії з відповідними компонентами, такими як CrawlQueueManager, PageStorage, SearchTermManager, StatisticsManager.

Компоненти

1. Веб-інтерфейс

- **Веб-клієнт**
  Це інтерфейс, який взаємодіє з кінцевим користувачем (клієнтом) для ініціалізації запитів через REST API.

2. Контролери

- **WebCrawlerController**
  Контролер, що ініціює основний процес сканування веб-сторінок.
- **SearchTermController**
  Контролер, що відповідає за обробку термінів пошуку.

Зв'язки між ними:
WebCrawlerController і SearchTermController взаємодіють з REST API.

3. Сервіси

- **WebCrawlerService**
  Сервіс, що керує процесом сканування сторінок і обробляє дані з інтернету.
- **TermSearchService**
  Сервіс, що обробляє терміни пошуку.
- **HttpClient**
  Відповідає за HTTP-запити до цільових веб-сторінок.

Зв'язки між ними:
WebCrawlerService використовує HttpClient для завантаження сторінок.
TermSearchService відповідає за пошук термінів у збережених даних.

4. Доступ до даних

- **CrawlQueueManager**
  Управляє чергою URL, які плануються до сканування.
- **PageStorage**
  Зберігає збережені сторінки з процесу сканування.
- **SearchTermManager**
  Зберігає інформацію про знайдені терміни.
- **StatisticsManager**
  Збирає і зберігає статистику по веб-скануванню.

5. База даних

- **SQLite**
  Сховище даних для збереження інформації про URL, статистику, знайдені терміни, мета-дані, сканування сторінок тощо.

Взаємозв'язки

1. **Веб-клієнт взаємодіє з REST API**, який забезпечує доступ до серверної логіки через HTTP.
2. **Контролери** WebCrawlerController **та** SearchTermController відповідають за обробку HTTP-запитів і передають дані до відповідних сервісів.
3. **Сервіси (**WebCrawlerService**,** TermSearchService**)** працюють із HTTP-клієнтом та бізнес-логікою.
4. **Доступ до даних (**dataAccess**)** об'єднує інтерфейси для доступу до бази даних SQLite.

   CrawlQueueManager, PageStorage, SearchTermManager, StatisticsManager працюють зі сховищем SQLite.

5. **HttpClient** відповідає за завантаження веб-сторінок через HTTP-запити.

Функціональні блоки

Сканування URL:
Коли користувач ініціює сканування, WebCrawlerController передає запит до WebCrawlerService. Сервіс сканування завантажує сторінки через HttpClient та зберігає дані у PageStorage.

Пошук термінів:
SearchTermController передає пошукові запити до TermSearchService. Цей сервіс знаходить терміни у збережених сторінках.

Доступ до даних:
dataAccess забезпечує інтеграцію з **SQLite** для зберігання статистики, даних сканування, знайдених термінів, тощо.

3. Діаграма послідовностей

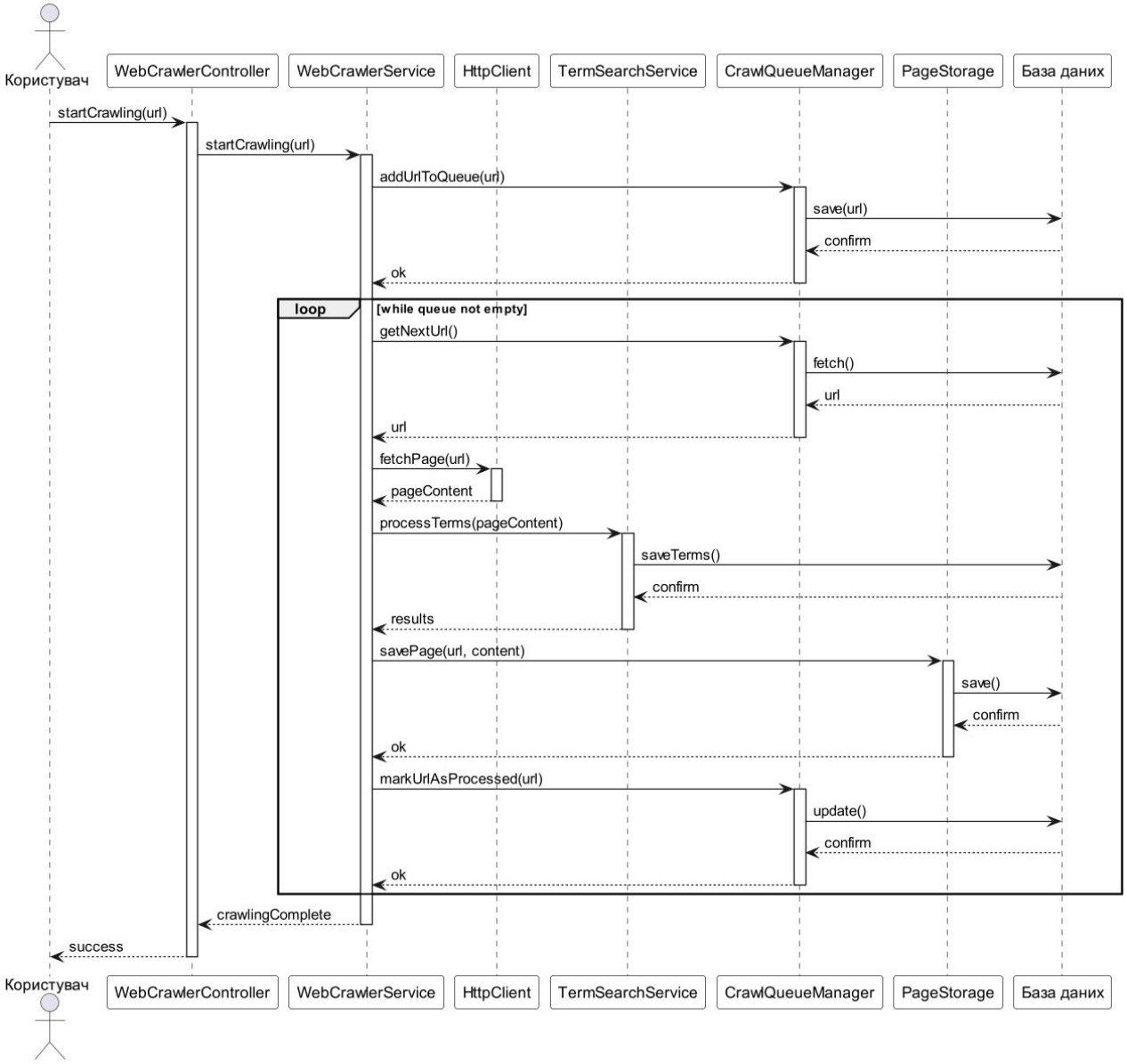

Рис.3 Діаграма послідовностей

Ця діаграма представляє послідовність дій для основного процесу сканування веб-сторінок, що ініціюється користувачем. Вона описує, як різні компоненти взаємодіють у послідовності дій, щоб виконати процес збору сторінок, обробки термінів, збереження даних, перевірку черги URL тощо.

Послідовність дій

1. **Користувач ініціює процес сканування**

1. user -> controller : startCrawling(url)
   Користувач відправляє HTTP-запит через інтерфейс до WebCrawlerController, щоб розпочати сканування.

Controller передає URL до сервісу

1. controller -> service : startCrawling(url)
   WebCrawlerController відправляє URL до основного сервісу WebCrawlerService.

Додавання URL до черги через CrawlQueueManager

1. service -> queueManager : addUrlToQueue(url)
   Сервіс передає URL до CrawlQueueManager для додавання до черги.
2. queueManager -> db : save(url)
   URL зберігається до бази даних SQLite.
3. Підтвердження від Бази даних: db --> queueManager : confirm.

Основний цикл сканування (поки URL у черзі)

1. service -> queueManager : getNextUrl()
   Сервіс отримує наступний URL для обробки з черги.
2. URL витягується з Бази даних: queueManager -> db : fetch().

Завантаження сторінки через HttpClient

1. service -> client : fetchPage(url)
   Сервіс використовує HttpClient для завантаження сторінки за URL.
2. Сторінка завантажується: client --> service : pageContent.

Обробка термінів на сторінці через TermSearchService

1. service -> searchService : processTerms(pageContent)
   Сервіс передає сторінку до TermSearchService.
2. searchService -> db : saveTerms()
   Знайдені терміни зберігаються до Бази даних.
3. db --> searchService : confirm.

Збереження сторінки до сховища через PageStorage

1. service -> storage : savePage(url, content)
   Сервіс передає сторінку до сховища.
2. storage -> db : save()
   Сторінка зберігається до Бази даних.
3. Підтвердження: db --> storage : confirm.

Позначення URL як обробленого в черзі

1. service -> queueManager : markUrlAsProcessed(url)
   Сервіс позначає URL у черзі як оброблений.
2. queueManager -> db : update()
   Оновлення статусу в Базі даних.
3. Підтвердження: db --> queueManager : confirm.

Сервіс завершує операцію після опрацювання всіх URL

1. service --> controller : crawlingComplete
   Після завершення процесу WebCrawlerService повідомляє, що сканування завершено.

Користувач отримує підтвердження

1. controller --> user : success
   Після завершення всього процесу контролер відповідає користувачу про успішне виконання.

Ключові моменти

- **Циклічний процес:** URL обробляються, поки черга не порожня.
- **HttpClient відповідає за завантаження сторінок**.
- **TermSearchService проводить пошук термінів на сторінках.**
- **CrawlQueueManager є головною логічною ланкою для обробки черги URL.**
- **Кожен компонент взаємодіє через Базу даних SQLite** для збереження інформації.

Висновок: У ході виконання цієї лабораторної роботи я ознайомилась з UML-діаграм, зокрема ознайомившись з діаграмами розгортання, компонентів та послідовностей.