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 №9

Технології розроблення програмного забезпечення

Тема роботи: « РІЗНІ ВИДИ ВЗАЄМОДІЇ ДОДАТКІВ: CLIENT-SERVER, PEER-TO-PEER, SERVICE-ORIENTED ARCHITECTURE *»*

Виконав студент групи ІА-23:

Каширов Д. О.

Перевірив:

Мягкий Михайло Юрійович

Київ 2024

Мета роботи:

Ознайомлення з архітектурами взаємодії додатків:
Студенти повинні зрозуміти принципи трьох основних типів архітектур:

- **Client-Server (Клієнт-Сервер)**: взаємодія між клієнтським додатком і сервером для обробки запитів.
- **Peer-to-Peer (P2P)**: взаємодія рівноправних вузлів для обміну даними.
- **Service-Oriented Architecture (SOA)**: організація програмного забезпечення через сервіси, що забезпечують конкретні функціональні можливості.

Хід роботи:

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. Реалізувати частину функціоналу робочої програми у вигляді класів і їх взаємодій для досягнення конкретних функціональних можливостей.

3. Реалізувати взаємодію програми в одній з архітектур відповідно до обраної теми.

Ознайомлення з теоретичним матеріалом:

- Вивчіть основні принципи трьох архітектур:
    - **Клієнт-Сервер** (Client-Server): взаємодія між клієнтом і сервером, де сервер обробляє запити та надсилає відповіді.
    - **Рівноправний обмін (Peer-to-Peer):** вузли мережі мають однакові права та діляться ресурсами напряму.
    - **Сервісно-орієнтована архітектура (SOA):** взаємодія через сервіси з визначеними інтерфейсами.
- Порівняйте їх переваги, недоліки та приклади використання.

Вибір архітектури:

- Виберіть одну архітектуру для реалізації програми.
- Обґрунтуйте свій вибір.

Проєктування програми:

- Розробіть структуру програми, використовуючи **об'єктно-орієнтований підхід (ООП)**.
- Виділіть класи, їх методи та атрибути.
- Продумайте схему взаємодії між класами.

Реалізація функціоналу:

- Напишіть код програми у вибраній архітектурі. Приклади:
    - Для **Client-Server**: створіть сервер, що обробляє запити, і клієнт, який їх надсилає.
    - Для **P2P**: реалізуйте взаємодію між двома рівноправними вузлами (наприклад, передача файлу).
    - Для **SOA**: створіть сервіси з API (наприклад, REST API) для виконання певних задач.
- Використовуйте стандартні бібліотеки або фреймворки для обраної архітектури.

Реалізація взаємодії компонентів:

- Забезпечте передачу даних між компонентами відповідно до вибраної архітектури.

- Наприклад, у **Client-Server** використовуйте сокети для обміну даними, у **SOA** — HTTP-запити через REST API.

Тестування:

- Перевірте роботу програми в різних сценаріях.
- Наприклад, для Client-Server перевірте коректність обробки запитів, а для P2P — передачу даних між вузлами.

Короткі теоретичні відомості:

У сучасному програмуванні взаємодія додатків може бути організована через різні архітектурні підходи. Найпоширенішими з них є **Client-Server**, **Peer-to-Peer** та **Service-Oriented Architecture (SOA)**.

1. Client-Server архітектура

Ця архітектура передбачає взаємодію між двома сторонами:

- **Сервер**: основний вузол, який зберігає дані, обробляє запити клієнтів і надсилає відповіді.
- **Клієнт**: програмний компонент, який ініціює запити до сервера для отримання даних або виконання певних функцій.

Основні характеристики:

- Централізоване зберігання даних.
- Сервер обробляє кілька клієнтів одночасно.
- Застосовується у веб-додатках, банківських системах, іграх.

Приклад:

Запит на сервер для отримання сторінки через HTTP (веб-браузер — клієнт, веб-сервер — сервер).

Переваги:

- Простота адміністрування (централізоване управління).
- Безпека даних.

Недоліки:

- Залежність від сервера (його збій призводить до недоступності системи).
- Можливі проблеми масштабування.

2. Peer-to-Peer (P2P) архітектура

У цій архітектурі всі вузли є рівноправними, кожен з них може виконувати функції як клієнта, так і сервера. Взаємодія здійснюється без центрального сервера.

Основні характеристики:

- Децентралізована система.
- Рівноправна взаємодія між вузлами.
- Застосовується у файлообмінниках (BitTorrent), месенджерах (Skype).

Переваги:

- Відсутність центрального вузла.
- Висока надійність (система працює навіть при відключенні кількох вузлів).

Недоліки:

- Ускладнена безпека.
- Проблеми з пошуком і маршрутизацією даних.

3. Service-Oriented Architecture (SOA)

Сервісно-орієнтована архітектура полягає у взаємодії між незалежними сервісами, які можуть виконувати різні функції та взаємодіяти через стандартизовані протоколи (наприклад, HTTP, SOAP, REST).

Основні характеристики:

- Сервіси ізольовані та незалежні.
- Взаємодія через API.
- Застосовується у великих розподілених системах, мікросервісах.

Переваги:

- Гнучкість і масштабованість.
- Легке оновлення окремих компонентів.

Недоліки:

- Ускладнення системи через необхідність управління сервісами.
- Потреба в стандартизації інтерфейсів.

Порівняння архітектур:

Архітектура	**Централізація**	**Надійність**	**Складність реалізації**	**Застосування**
Client-Server	Централізована	Залежить від сервера	Середня	Веб-додатки, банківські системи
Peer-to-Peer (P2P)	Децентралізована	Висока	Середня	Файлообмінники, розподілені мережі
SOA	Змішана	Залежить від сервісів	Висока	Мікросервіси, великі системи

Висновок:

Вибір архітектури залежить від вимог проєкту. Наприклад:

- Для централізованих систем із багатьма користувачами підходить **Client-Server**.
- Для мереж з рівноправною взаємодією і без центрального вузла — **P2P**.
- Для великих масштабованих додатків — **SOA**.

Реалізація Коду та файлів:

Ця структура проекту вказує на добре організовану програму, яка поєднує кілька архітектурних і програмних підходів, таких як **Client-Server** і **Peer-to-Peer (P2P)**. Давайте детально розглянемо функціональність кожної папки та файлів:

1. node_modules

Містить сторонні бібліотеки, встановлені через npm. Ці бібліотеки використовуються в проекті для реалізації додаткових функцій, таких як робота з сервером, базою даних або P2P-мережами.

2. src

a. controllers

Містить логіку для обробки запитів і взаємодії між моделями та представленням.

- **indexController.js**
  Основний контролер, який управляє домашньою сторінкою або іншими базовими функціями програми.
- **userController.js**
  Логіка роботи з користувачами (авторизація, реєстрація, профілі, тощо).

b. middlewares

Містить проміжні шари для обробки запитів перед тим, як вони дійдуть до контролерів.

- **proxyMiddleware.js**
  Використовується для маршрутизації або перехоплення запитів (наприклад, для проксі-сервера, безпеки, логування або змінення даних).

c. models

Містить об'єкти, які представляють структуру даних.

- **Page.js**
  Модель для представлення сторінок веб-додатка (назва, вміст, мета-інформація).

- **User.js**
  Модель користувача, що зберігає інформацію про користувачів (ім'я, email, паролі).

d. observers

Містить класи для реалізації **паттернів проєктування** (Observer, Mediator, Visitor тощо).

- **ConcreteVisitor.js**
  Реалізація конкретного відвідувача для обходу елементів і виконання дій над ними.
- **Interpreter.js**
  Реалізація паттерну "Інтерпретатор" для роботи з певними командами чи мовами в програмі.
- **Logger.js**
  Логування подій (наприклад, інформація про дії користувачів або помилки).
- **Mediator.js**
  Координує взаємодію між різними об'єктами, уникаючи прямого зв'язку між ними.
- **Notifier.js**
  Реалізує функціонал для надсилання повідомлень (email, push-повідомлення).
- **Observer.js**
  Реалізація паттерну "Спостерігач" для відслідковування змін стану об'єктів.
- **Subject.js**
  Об'єкт, за змінами якого слідкують спостерігачі.
- **Visitor.js**
  Реалізація паттерну "Відвідувач" для розширення функціональності без зміни класів.

e. p2p

- **P2PNode.js**
  Клас для реалізації **Peer-to-Peer** взаємодії між вузлами. Наприклад, обмін файлами або повідомленнями між рівноправними учасниками мережі.

f. repositories

Слугують для роботи з даними (отримання, зберігання, оновлення).

- **ArticleRepository.js**
  Логіка для доступу до статей у базі даних або сховищі.
- **UserRepository.js**
  Логіка для доступу до даних користувачів.

g. routes

Містить маршрути для визначення шляхів та обробників запитів.

- **apiRoutes.js**
  API маршрути для роботи з даними (JSON-інтерфейс для клієнтів).
- **browserRoutes.js**
  Маршрути для роботи з інтерфейсом користувача через браузер.
- **index.js**
  Основний файл, що об'єднує всі маршрути.
- **p2pRoutes.js**
  Маршрути для взаємодії вузлів у P2P-мережі.
- **userRoutes.js**
  Маршрути, пов'язані з користувачами (реєстрація, вхід, профіль).

h. Інші файли

- **app.js**
  Головний файл програми. Налаштовує сервер, підключає маршрути, middleware, базу даних.
- **database.js**
  Логіка для підключення до бази даних.

3. index.html

Файл для основного інтерфейсу користувача (графічна оболонка). Зазвичай використовується в Client-Server архітектурі.

4. styles.css

Містить стилі для оформлення інтерфейсу.

5. package.json і package-lock.json

Містять інформацію про залежності та налаштування проєкту (наприклад, використовувані бібліотеки).

Призначення проекту

Цей проєкт демонструє взаємодію різних архітектур і паттернів. Можливі функції:

- Користувачі взаємодіють із сервером для отримання даних через API.
- Клієнти можуть обмінюватися інформацією напряму через P2P-мережу.
- Сервіси логування, сповіщень та координації використовують паттерни проєктування для розширення функціональності.

Вивід відповіді від сервера:

```
> litebrowse@1.0.0 start
> node src/app.js

Обробка сторінки: Example Page
Обробка користувача: Dmytro
Server is running on port 3000
[Logger] Подія: Сервер запущено, Деталі: {"port":3000}
[Notifier] Сповіщення: Сервер запущено
[Logger] Подія: Новий запит, Деталі: {"method":"GET","url":"/"}
[Notifier] Сповіщення: Новий запит
[Logger] Подія: Новий запит, Деталі: {"method":"GET","url":"/favicon.ico"}
[Notifier] Сповіщення: Новий запит
```

Рис 1. – Запуск сервера.

Сам проект:

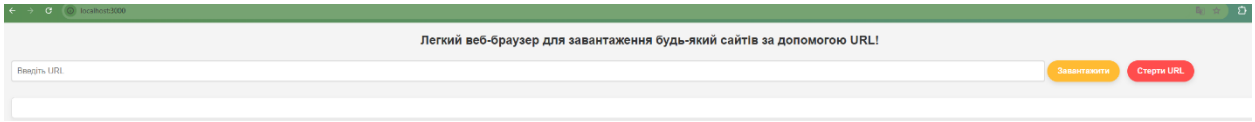

Рис 1. – Головна сторінка функціоналу користувача.

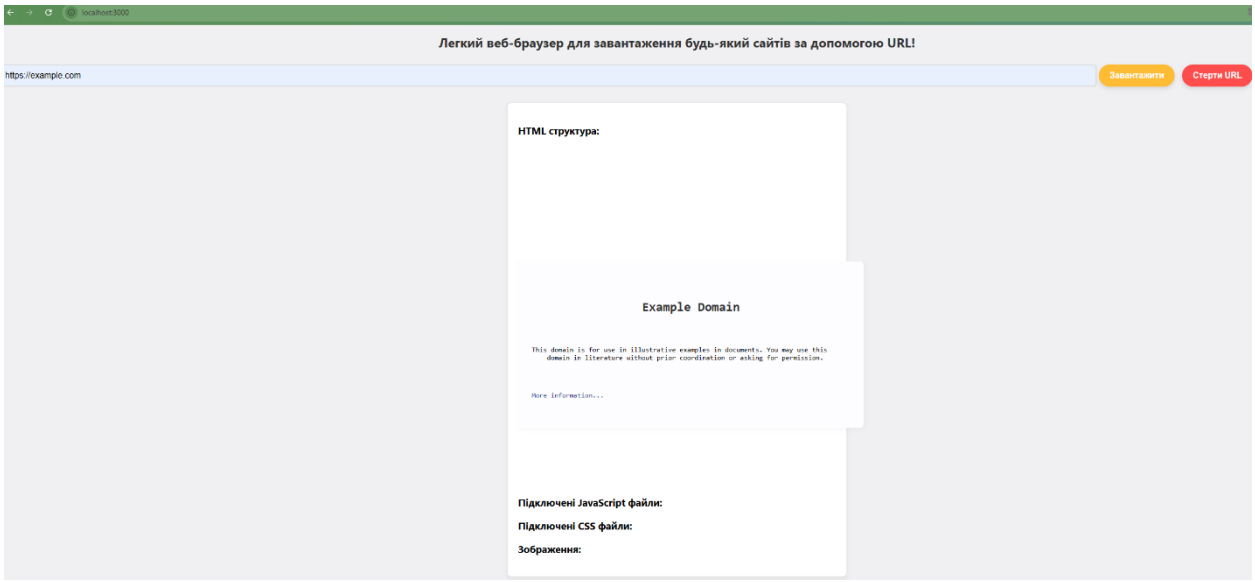

Рис 2. – Функціонал створення LiteBrowse.

Завантаження будь-яких сторінок у LiteBrowse.

```
Proxy Middleware: обробляється запит до https://example.com
Запит: GET /browser/load?url=https%3A%2F%2Fexample.com
[Logger] Подія: Запит успішно виконано, Деталі: {"url":"/browser/load?url=https%3A%2F%2Fexample.com","method":"GET"}
[Notifier] Сповіщення: Запит успішно виконано
```

Рис 3. – Сповіщення на сервері про запит на URL-адрес

Рис 4. - Завантаження інших сторінок, наприклад калькулятор.

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
  code: 'ETIMEDOUT',
  errno: -4039,
  config: {
    transitional: {
      silentJSONParsing: true,
      forcedJSONParsing: true,
      clarifyTimeoutError: false
    },
    adapter: [ 'xhr', 'http', 'fetch' ],
    transformRequest: [ [Function: transformRequest] ],
    transformResponse: [ [Function: transformResponse] ],
    timeout: 0,
    xsrfCookieName: 'XSRF-TOKEN',
    xsrfHeaderName: 'X-XSRF-TOKEN',
    maxContentLength: -1,
    maxBodyLength: -1,
    env: { FormData: [Function], Blob: [class Blob] },
    validateStatus: [Function: validateStatus],
```

Рис 5. - Відповідь в терміналі.

Висновки

У ході виконання курсової роботи було розроблено веб-браузер з використанням клієнт-серверної архітектури, що забезпечує ефективну взаємодію між клієнтом і сервером. У проєкті були впроваджені кілька ключових патернів проєктування, таких як Proxy, Chain of Responsibility, Factory Method, Template Method, Visitor, P2P, які дозволили покращити структурованість, розширюваність та зручність підтримки коду.

Було розроблено систему, яка реалізує функціонал перегляду HTML-документів, обробки ресурсів (CSS, JavaScript, зображення), роботи з відповідями сервера (перенаправлення, обробка кодів помилок), а також забезпечує можливість використання P2P-з'єднань.

Розробка проєкту включала:

1. Аналіз існуючих рішень і формування вимог.
2. Проєктування архітектури системи та ключових компонентів.
3. Реалізацію функціональних можливостей та створення бази даних на основі MySQL.
4. Створення сценаріїв використання системи та їх детальний опис.
5. Тестування проєкту для перевірки відповідності функціональних і нефункціональних вимог.

Під час розробки використовувались сучасні інструменти, такі як Visual Studio Code, HTML, CSS, JavaScript, Node.js, а також база даних MySQL. Це дозволило створити зручне середовище для роботи з веб-браузером, а також забезпечити стабільність і швидкодію системи.

Впровадження клієнт-серверної архітектури та використання патернів проєктування довели свою ефективність у побудові гнучкої та масштабованої системи. Даний проєкт може слугувати базою для подальших покращень, зокрема додавання нових функцій, оптимізації продуктивності та інтеграції з іншими системами.

Результати виконаної роботи підтвердили, що поставлені цілі були досягнуті, а завдання – виконані.