НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
Кафедра інформаційних систем та технологій

Тема_____Web-browser_____

Курсова робота
З дисципліни «Технології розроблення програмного забезпечення»

Керівник
доц. Амонс О.А.
«Допущений до захисту»
_____
(Особистий підпис керівника)
« » _____2024р.
Захищений з оцінкою
_____
(оцінка)

Виконавець
ст. Ястремський Б.В.
залікова книжка № ___ – _____
гр. ___ІА-34_____
_____
(особистий підпис виконавця)
« »_____2024р.

Члени комісії:

_____ _____
(особистий підпис) (розшифровка підпису)

_____ _____
(особистий підпис) (розшифровка підпису)

Київ – 2025

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
(назва навчального закладу)

Кафедра ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

Дисципліна «Технології розроблення програмного забезпечення»

Курс_3_ Група__ІА-34__Семестр____5____

ЗАВДАННЯ

на курсову роботу студента

Ястремського Богдана Валерійовича

(прізвище, ім’я, по батькові)

1.Тема роботи:Web-browser

2. Строк здачі студентом закінченої роботи: 08.12.25

3. Вихідні дані до роботи: Веб-браузер повинен мати можливість зробити наступне: мати адресний рядок для введення адреси сайту, переміщатися по сторінці, і відображати структур 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.

4. Зміст розрахунково – пояснювальної записки (перелік питань, що підлягають розробці)

Проєктування системи: Огляд існуючих рішень, Загальний опис проєкту, Вимоги до застосунків системи, Сценарії використання системи, Концептуальна модель системи, Вибір мови програмування та середовища розробки, Проєктування розгортання системи.

Реалізація компонентів системи: Архітектура системи, Специфікація системи, Вибір та обґрунтування патернів реалізації.

Додатки:

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

Діаграма варіантів використання та описані сценарії варіантів використання (use-case diagram), Діаграма класів системи (class diagram), Діаграма розгортання системи (deployment diagram), Діаграма станів для процесів, що протікають у системі (state diagram), Діаграма послідовностей для процесів у системі (sequence diagram), Діаграма архітектури взаємодії

6. Дата видачі завдання: 16.09.2025

КАЛЕНДАРНИЙ ПЛАН

№, п/п	Назва етапів виконання курсової роботи	Строк виконання етапів роботи	Підписи або примітки
1.	Підбір та вивчення літератури	30.09.2025	
2.	Проектування на написання розділу 1	31.10.2025	
3.	Розробка та написання розділу 2	20.11.2025	
4.	Подання курсової роботи на перевірку	25.11.2025	
5.	Захист курсової роботи	08.12.2025	
6.			
7.			
8.			
9.			
10.			
11.			
12.			
13.			
14.			
15.			
16.			
17.			
18.			

Студент _____ _Богдан ЯСТРЕМСЬКИЙ_
(підпис) (Ім’я ПРІЗВИЩЕ)

Керівник_____ _Олександр АМОНС__
(підпис) (Ім’я ПРІЗВИЩЕ)

«____»_____ 20___ р.

ЗМІСТ

ВСТУП

Веб-браузери дають нам доступ до нескінченної кількості онлайн-ресурсів: від простих вебсайтів до складних систем електронної комерції, потокових сервісів та інструментів для спільної роботи. Вони виконують роль головного інтерфейсу між користувачем і сайтами, доступної через глобальну мережу Інтернет.

Сучасні технології, що використовуються в браузерах, розвиваються великими темпами, і нові, потужні можливості постійно впроваджуються — це і підвищення швидкості обробки, і посилення безпеки, і, також, поліпшення взаємодії з користувачем (UX) та оптимізації процесу перегляду.

Звісно, на ринку вже є Google Chrome, Mozilla Firefox чи Microsoft Edge. Їхній функціонал є дуже широким і покриває потреби більшості користувачів. Проте, цей широкий, універсальний функціонал не завжди може повністю покрити специфічні можливості або вимоги.

Саме тому створення власного веб-браузера, навіть для вирішення дуже конкретних завдань або навчальних цілей, дає можливість глибоко дослідити й отримати розуміння архітектури веб-браузерів.

Однією з найважливіших і найбільш технічно складних функцій будь-якого веб-браузера є його здатність коректно завантажувати та відображати веб-сторінки. Це включає складний цикл взаємодії з серверами: від ініціації запиту та встановлення з'єднання до обробки отриманих даних.

Браузер повинен вміти не просто відправляти запити, а й грамотно обробляти різноманітні відповіді, що надходять від сервера. Це стосується як успішного виконання (наприклад, код 200 OK), так і різних типів перенаправлень (3xx), і особливо критично — коректної обробки кодів HTTP-помилок (4xx, 5xx), щоб користувач завжди бачив зрозуміле повідомлення.

1 ПРОЄКТУВАННЯ СИСТЕМИ

1.1. Огляд існуючих рішень

Веббраузер – це програмне забезпечення, яке виступає посередником між користувачем та вебсервером, забезпечує доступ до ресурсів в мережі, відображає їх вміст – текст, відео-, фото- та аудіоматеріали. Історія розвитку браузерів розпочалася у 1990 році, коли Тім Бернерс-Лі створив перший веббраузер під назвою WorldWideWeb (пізніше перейменований на Nexus). З розвитком технологій з'являлися нові браузери, які сьогодні класифікуються за різними критеріями.

Сьогодні на ринку [1] домінуючу позицію займає Google Chrome, який з 2012 року стабільно утримує перше місце за часткою користувачів у світі та в Україні. Його популярність зумовлена високою швидкістю завантаження сторінок, розширеними можливостями завдяки великій бібліотеці розширень та інтеграцією з екосистемою Google.

Також відомими є [1] Microsoft Edge, Apple Safari, Mozilla Firefox та Opera. Кожен з цих браузерів має свої унікальні переваги [2].

Microsoft Edge, побудований на базі Chromium, пропонує гарну швидкість та інтеграцію з операційною системою Windows, а також акцентує увагу на безпеці та енергоефективності, що робить його привабливим для користувачів Windows.

Apple Safari відомий своєю швидкістю та ефективною роботою, але його ключова проблема полягає у тісній інтеграції та оптимізації лише для операційних систем macOS та iOS, що обмежує його крос-платформне використання.

Mozilla Firefox є відкритим (open-source) рішенням, яке робить акцент на конфіденційності та безпеці користувачів, пропонуючи більше можливостей для налаштування та контролю над даними.

Opera вирізняється вбудованими функціями, такими як безкоштовний VPN, блокувальник реклами та режим економії трафіку, що робить його прекрасним вибором для користувачів, які цінують конфіденційність та можливість кастомізації.

Окрім основних гравців, існують і альтернативні браузери, які фокусуються на інших задачах. Наприклад, Brave пропонує власний блокувальник реклами та модель винагороди користувачів за перегляд реклами токенами. Tor Browser забезпечує підвищену анонімність та конфіденційність за рахунок маршрутизації трафіку через мережу Tor.

Таким чином, сучасний ринок веббраузерів містить багато рішень, кожне з яких прагне задовольнити потреби користувачів за рахунок певного набору функцій: від пріоритету швидкості (Chrome, Safari) до підвищеної конфіденційності (Firefox, Tor, Brave) та інтегрованих інструментів (Opera, Edge).

1.2. Загальний опис проєкту

Оскільки створення веб-браузеру без сторонніх бібліотек, які дають можливість виводити веб-сторінку на екран вимагає колосальних зусиль в плані команди, архітектури, знань всіх аспектів використовуваних мов програмування, і багато часу, то мій браузер буде доволі обмеженим в плані функціоналу до тих речей, які у всіх вищезгаданих браузерах є спільними.

По-перше, введення адреси сайту. Без поля вводу, браузер – не браузер, оскільки він не знатиме, що йому відображати. Тому воно обов’язково має бути. Проте, цей компонент також буде з меншою функціональністю. Воно буде приймати лише адресу сайту, тобто назву він не обробить. Обробка назви – зовсім інші алгоритми, і у вимоги не входить.

По-друге, переміщення. Можна буде переміщатися по сторінці, переходити за посиланнями.

По-третє, відображення структури html-документа, переглядання підключених javascript та css файлів, перегляд всіх підключених ресурсів (зображень). Це і є пряме призначення браузера, тому цей аспект також варто врахувати.

По-четверте, коректна обробка HTTP-кодів. Браузер має чітко розуміти, коли і який код дає ту чи іншу помилку. Для обробки помилок 404, 502 та 503 я створю

тестовий сервер, де браузер буде повертати відповідний код помилки і відображати необхідну сторінку.

Для дуже складних сайтів, які рідко містять звичайні html, css та js файли, я буду використовувати вбудований компонент WebEngine у фреймворк JavaFX, оскільки він є дуже потужним інструментом для рендеру вебсторінок з усім необхідним контентом.

А ось для сторінок тестового веб-серверу, я використовуватиму власний парсер, який буде самостійно оброболювати задану йому сторінку.

1.3. Вимоги до застосунків системи

Для більшої наочності я проілюструю всі вище наведені вимоги у вигляді UML-діаграми варіантів використання та наведу детальний її опис.

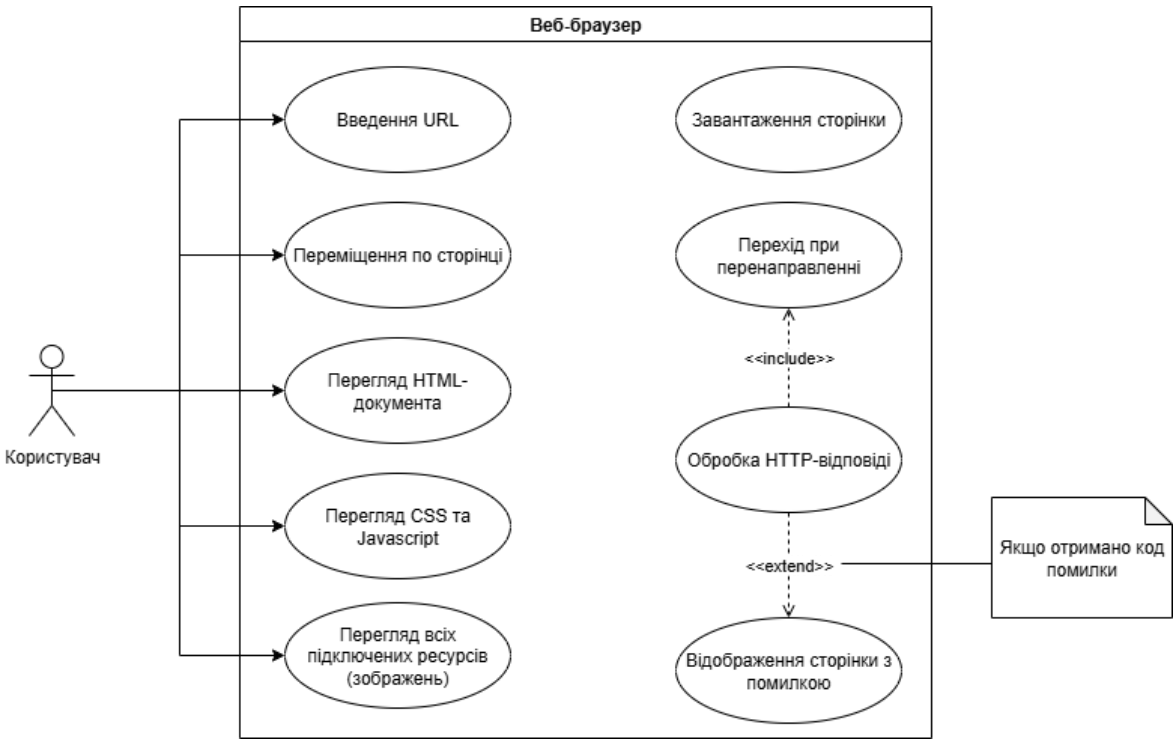

Рис. 1 – Діаграма варіантів використання системи

Актори: **Користувач** (головний, взаємодіє з браузером), **Веб-браузер** (головна система, частиною якої є **Веб-сервер**).

Варіанти використання:

- Введення URL - Користувач вводить URL у адресний рядок.
- Завантаження сторінки - Браузер робить запит на сервер і отримує HTML документ.
- Переміщення по сторінці - Користувач переміщається по вже завантаженій сторінці.
- Перегляд HTML документа - Користувач переглядає вихідний HTML документ.
- Перегляд CSS та JavaScript - Користувач переглядає підключені CSS і JavaScript файли.
- Перегляд зображень та інших ресурсів - Користувач переглядає зображення та інші ресурси, що завантажуються.
- Обробка HTTP відповіді - Браузер обробляє HTTP відповіді (наприклад, 200, 404, 502, 503).
- Перехід при перенаправленні - Браузер обробляє статуси 301/302 і здійснює автоматичний перехід.
- Відображення сторінки з помилкою - Браузер показує сторінку помилки (наприклад, 404, 502 чи 503). Ці помилки будуть тестуватися на локальному сервері.

Зв’язки між акторами і варіантами використання:

1. Користувач - Введення URL (напрямлена асоціація) Користувач ініціює дію введення URL в адресний рядок.
2. Користувач - Переміщення по сторінці (напрямлена асоціація). Користувач може переміщатися по сторінці після її завантаження.
3. Користувач - Перегляд HTML документа (напрямлена асоціація). Користувач може захотіти побачити HTML код завантаженої сторінки.

4. Користувач - Перегляд CSS та JavaScript (напрямлена асоціація). Користувач може перевірити або проаналізувати стилі та скрипти на сторінці.
5. Користувач - Перегляд зображень та інших ресурсів (напрямлена асоціація). Користувач може переглядати ресурси на сторінці, такі як зображення, відео тощо.
6. Веб-сервер - Завантаження сторінки (напрямлена асоціація) Веб-сервер надає браузеру HTML документ у відповідь на запит.
7. Веб-сервер - Перехід при перенаправленні (напрямлена асоціація). Веб-сервер відправляє код перенаправлення (301, 302), і браузер виконує новий запит.
8. Веб-сервер - Обробка HTTP відповіді (напрямлена асоціація) Веб-сервер відповідає браузеру HTTP статусами (наприклад, 200, 404, 502), а браузер обробляє ці відповіді.
9. Веб-сервер - Відображення сторінки з помилкою (напрямлена асоціація). Веб-сервер може відправити код помилки (404, 502), і браузер відображає відповідну сторінку помилки.
10. Браузер - Обробка HTTP відповіді (напрямлена асоціація) Браузер обробляє відповіді, отримані від сервера, і здійснює відповідні дії.
11. Обробка HTTP відповіді - Перехід при перенаправленні (include). Обробка перенаправлення є частиною загальної обробки HTTP відповіді.
12. Обробка HTTP відповіді - Відображення сторінки з помилкою (extend). Якщо отримано код помилки, обробка цього коду розширює стандартну поведінку обробки відповіді і показує сторінку помилки.

1.4. Сценарії використання системи

За наведеною вище діаграмою варіантів використання наведу 2 сценарії використання системи з відповідними діаграмами і подальшим описом. Обидві діаграми послідовності будуть розміщені в додатках (рис. 2 та рис. 3 відповідно)

1) Сценарій «Користувач – Введення URL»:

Передумови: Користувач відкрив веб-браузер, має доступ до Інтернету та бажає перейти на конкретний сайт.

Постумови: Браузер спробує підключитись до введеного URL, передавши запит на сервер, і почне завантажувати веб-сторінку.

Сторони взаємодії: Користувач, Веб-браузер.

Короткий опис: Користувач вводить адресу веб-сайту в адресний рядок браузера, після чого браузер ініціює HTTP-запит до серверу.

Основний перебіг подій:

- Користувач відкриває браузер.
- Користувач вводить URL у адресний рядок.
- Браузер обробляє введену адресу і здійснює HTTP-запит до веб-сервера для завантаження сторінки.

Винятки: Користувач вводить неправильний або неповний URL або неіснуючу адресу сайту. Або посилання з тестового серверу.

Примітки: у разі відсутності протоколу, браузер автоматично його додає.

2) Сценарій «Браузер – Обробка HTTP-відповіді»:

Передумови: Браузер отримав відповідь від сервера у вигляді HTTP статусу та вмісту (HTML, CSS, зображення тощо).

Постумови: Браузер інтерпретує HTTP відповідь та відповідно відображає контент на екран. Якщо є помилка (наприклад, 404), браузер відображає відповідну сторінку помилки.

Сторони взаємодії: Браузер, Веб-сервер.

Короткий опис: Після отримання відповіді від сервера, браузер аналізує HTTP статус, інтерпретує його та відображає контент. Якщо відповідь

містить помилку (наприклад, 404), браузер відображає сторінку з помилкою.

Основний перебіг подій:

- Браузер отримує HTTP-відповідь від сервера і перевіряє код статусу відповіді.
- Якщо відповідь успішна, браузер відображає сторінку з ресурсами.
- Якщо ні, то відображається сторінка з помилкою.

Винятки: Відповідь з помилкою або пошкодженим контентом.

Примітки: відсутні.

1.5. Концептуальна модель системи

В якості концептуальної моделі, я наведу діаграму класів з детальним описом кожного класу та розміщу її в додатках (рис. 4).

Browser (Браузер)

Опис: Керує взаємодією з користувачем, організовує завантаження сторінок, взаємодію з іншими класами.

Атрибути:

- addressBar: AddressBar (адресний рядок).
- currentPage: WebPage (поточна веб-сторінка).

Методи:

- loadPage(url: String): завантажує веб-сторінку за вказаним URL.
- handleError(errorCode: Integer): обробляє помилки (404, 502 і т.д.).
- setAddressBar(addressBar: AddressBar): встановлює адресний рядок

AddressBar (Адресний рядок)

Опис: Клас для управління введенням URL і запитом на завантаження веб-сторінки.

Атрибути:

- url: String: рядок, що містить введену адресу.
- protocol: String: протокол адреси.

Методи:

- getInput(): String: отримує введену URL-адресу.
- validateURL(url: String): Boolean перевіряє правильність введеного URL.
- setUrl(url: String): встановлює URL-адресу
- getProtocol(): String: отримує назву протоколу

WebPage (Веб-сторінка)

Опис: Клас, що містить інформацію про HTML-сторінку, підключені ресурси та їхній вміст.

Атрибути:

- htmlResources: List<HTMLFile> (HTML файли).
- cssResources: List<CSSFile> (CSS файли, підключені до сторінки).
- jsResources: List<JSFile> (JavaScript файли).
- imageResources: List<IImage> (зображення).
- rawHTML: String (сирий рядок HTML)

Методи:

- parseHTML(): парсить HTML-код і витягує відповідні елементи.
- loadResources(): завантажує всі підключені ресурси (CSS, JavaScript, зображення).
- setRawHTML(rawHTML: String): встановлює HTML

HTMLFile

Опис: Клас, що містить інформацію про html-файл:

Атрибути:

- fileName: String назва файлу
- filePath: String шлях до файлу

- content: String вміст файлу

Методи:

- getContent: отримати вміст файлу

CSSFile

Опис: Клас, що містить інформацію про css-файл:

Атрибути:

- fileName: String назва файлу
- filePath: String шлях до файлу
- content: String вміст файлу

Методи:

- loadCSS: завантажити вміст файлу

JSFile

Опис: Клас, що містить інформацію про js-файл:

Атрибути:

- fileName: String назва файлу
- filePath: String шлях до файлу
- content: String вміст файлу

Методи:

- loadJS: завантажити вміст файлу

ImageFile

Опис: Клас, що містить інформацію про файл-картинку:

Атрибути:

- fileName: String: назва файлу
- filePath: String: шлях до файлу
- content: String: вміст файлу
- loaded: Boolean: чи завантажена картинка

Методи:

- display(): відобразити файл
- loadImage(): завантажити файл
- getFileName(): String: отримати назву файлу
- getFilePath(): String: отримати шлях до файлу
- getContent(): String: отримати вміст файлу
- isLoaded(): boolean: завантажити вміст файлу

WebServer

Опис: Клас, що містить інформацію про вебсервер:

Атрибути:

- host: String зберігає ім’я хоста або адресу сервера.
- resources: List<String> список ресурсів, доступних на сервері
- pages: List<WebPage> список веб-сторінок, які сервер може відправляти у відповідь на запит.

Методи:

- processRequest(request: HTTPRequest): HTTPResponse обробляє HTTP запит і генерує HTTP відповідь
- addResource(resource: String): додати ресурс
- addPage(page: WebPage): додати сторінку

HTTPRequest:

Опис: Клас, що відповідає за відправлення запиту до сервера і отримання відповіді.

Атрибути:

- url: String адреса ресурсу.
- method: String — метод HTTP (наприклад, GET, POST).

Методи:

- sendRequest(): HTTPResponse відправляє запит на сервер і отримує HTTPResponse.
- parseResponse(response: HTTPResponse) обробляє отриману відповідь.

HTTPResponse:

Опис: Клас, що представляє відповідь сервера на HTTP запит.

Атрибути:

- statusCode: Integer код статусу HTTP (наприклад, 200, 404, 301).
- headers: Map<String, String> заголовки відповіді (для зберігання метаданих).
- body: String тіло відповіді

Методи:

- getStatusCode(): Integer: повертає код статусу HTTP.
- getBody(): String: повертає вміст тіла відповіді.
- getHeaders(): Map<String, String>: повертає хедери
- setHeaders(headers: Map<String, String>): встановлює хедери
- setStatusCode(statusCode: Integer): встановлює код статусу HTTP
- setBody(body: String): встановлює вміст тіла відповіді

1.6. Вибір мови програмування та середовища розробки

Для реалізації цього проєкту, я планую використовувати Java як основну мову програмування. Вибір очевидний, оскільки Java вирізняється своєю кросплатформністю та надійністю. Це означає, що розроблений застосунок без проблем працюватиме на будь-якій операційній системі (Windows, macOS, Linux, тощо) без необхідності вносити суттєві зміни в його специфіку чи код.

Окрім того, Java має потужний фреймворк JavaFX [3], який є ідеальним рішенням для розробки інтерактивного графічного інтерфейсу (GUI). Він дозволяє

дуже легко і ефективно працювати з усіма необхідними елементами: кнопками, текстовими полями, відображенням даних та іншими інтерактивними компонентами.

Для швидкої та зручної розробки дизайну сцени (інтерфейсу) я буду використовувати застосунок SceneBuilder [3]. Цей інструмент надає можливість візуально наносити, розміщувати та маніпулювати всіма інтерактивними елементами на сцені. Результат роботи зберігається у зручному форматі FXML, який якраз і є стандартом для використання у фреймворку JavaFX.

В якості середовища розробки (IDE) я оберу IntelliJ IDEA. Вона відома своїми надзвичайно зручними та розумними інструментами для написання коду, включаючи автодоповнення та рефакторинг. IntelliJ IDEA також дозволяє з максимальною легкістю створювати та конфігурувати проєкт практично на будь-якій архітектурі, забезпечуючи комфортну та продуктивну роботу.

1.7. Проєктування розгортання системи

Саме тому, що я обрав мову Java в якості основної, то на діаграмі я позначив три комп’ютери з різними ОС, хоча, по суті, на діаграмі може бути і більше 3-х ПК.

Сервер, по суті, буде на моєму ПК, тому я задав девайс, як свою ОС – Windows. Але, оскільки застосунок десктопний, то сервером буде ОС, на якій користувач використовуватиме браузер.

Діаграма розгортання буде винесена в додатки (рис. 5).

2 РЕАЛІЗАЦІЯ КОМПОНЕНТІВ СИСТЕМИ

2.1. Архітектура системи

2.1.1. Специфікація системи

У розділі додатків (рис. 6) я додам діаграму компонентів системи. Це наш візуальний план, який я використав для моделювання, як усі складові системи взаємодіють між собою у часі. Вона чітко показує, як об'єкти системи обмінюються повідомленнями - кожен об'єкт виконує свою роль, дотримуючись чіткої послідовності та логіки виконання операцій.

На діаграмі показано, з адресної строки ми направляємо запит у браузер, а браузер повертає відповідь у вигляді вебсторінки.

Також я додав тестовий сервер test.com для перевірки помилок 404, 502 і 503 і відображення відповідних сторінок. Всі компоненти сервера взаємодіють з контролером, який повертає необхідну веб сторінку.

Контролер ініціалізує JavaFX застосунок (робота з інтерактивними елементами: кнопка, поле вводу), тестовий сервер (для перевірки помилок 404, 502 і 503 і виводу відповідних сторінок).

Також я додам у додатки діаграму станів (рис.7) для процесу «Завантаження веб-сторінки». Ця діаграма описує стани браузера під час завантаження веб-сторінки від моменту введення URL в адресний рядок до повного завантаження та відображення сторінки.

- Стан 1: (Очікування)

  Опис: Браузер чекає на введення URL користувачем у адресний рядок.

  Перехід до: (Завантаження), коли користувач вводить URL і натискає на кнопку пошуку.

- Стан 2: "Loading" (Завантаження)

  Опис: Браузер робить запит на сервер і чекає на відповідь.

  Перехід до:

(Помилка), якщо виникла помилка при завантаженні сторінки (наприклад, 404 або 502).

(Відображення), якщо сторінка успішно завантажена.

- Стан 3: (Відображення)

  Опис: Браузер отримав HTML-код сторінки, парсить його і завантажує ресурси (CSS, JS, зображення).

  Перехід до:

  (Очікування), коли сторінка повністю завантажена і готова до взаємодії.

  (Помилка), якщо виникли проблеми з ресурсами (наприклад, зображення не завантажено).

- Стан 4: (Помилка)

  Опис: Виникла помилка при завантаженні сторінки або ресурсів (наприклад, 404, 502, 503).

  Перехід до: (Очікування) після того, як користувач побачить повідомлення про помилку.

2.1.2. Вибір та обґрунтування патернів реалізації

Для забезпечення гнучкості та чистоти коду я вирішив використовувати два ключові патерни: Proxy (Заглушка) та Chain of Responsibility (Ланцюжок відповідальності). Діаграми класів для цих патернів будуть винесені в додатки (рис. 8 та рис. 9 відповідно)

Патерн Proxy [4], [6], [7], [9] (Заглушка) був обраний для ефективної роботи з відображенням зображень, особливо коли потрібна «лінива» (lazy) ініціалізація.

Я розпочав із визначення інтерфейсу IImage. У ньому зібрані всі необхідні методи для роботи із зображенням: відображення самої картинки, завантаження її вмісту, отримання імені, шляху та перевірка, чи вже завантажено вміст.

Далі я пов'язав цей інтерфейс із нашим вже існуючим класом ImageFile. Це клас, який реалізує IImage і реально займається обробкою, завантаженням та відображенням справжніх, «важких» зображень [8].

Після цього я створив ключовий клас — ImageProxy. Він також реалізує інтерфейс IImage, але його основне завдання — визначити заглушку (placeholder). Замість того, щоб одразу завантажувати велике зображення, ImageProxy покаже цю легку заглушку. Справжнє зображення буде завантажене лише тоді, коли клієнтський код дійсно викличе метод відображення.

По суті, я розділив відповідальність між реальним зображенням і його «представником». Це дає нам можливість гнучко керувати відображенням: замість того, щоб прописувати заглушки всередині основного класу, ми винесли цю логіку в окремий проксі-клас.

Перевага цього патерна в тому, що він не вимагає жодних змін у клієнтському коді. Клієнтський код завжди працює лише з об'єктом інтерфейсу IImage, який, залежно від ситуації, може бути або легкою заглушкою-проксі, або вже справжньою, завантаженою картинкою.

Патерн Chain of Responsibility [5], [6], [7], [10] ідеально підійшов для обробки різних типів HTTP-відповідей та помилок.

Я визначив інтерфейс HTTPResponseHandler, який містить два важливі методи: один для обробки самої відповіді, а інший для перевірки можливості обробки (чи належить ця відповідь до його компетенції).

Потім я створив абстрактний клас AbstractHTTPHandler, який реалізує ці методи, закладаючи спільну логіку для всіх наступних обробників. На основі цього абстрактного класу я створив окремі наслідники для кожного типу відповіді чи помилки, які нам потрібно контролювати:

- SuccessHandler.java - клас, що займається обробкою успішної відповіді 200 OK.
- NotFoundHandler.java - обробки помилки 404 (Не знайдено).
- BadGatewayHandler.java - клас для 502 (Поганий шлюз).
- ServiceUnavailableHandler.java - для випадку 503 (Сервіс недоступний).

Цей патерн значно полегшив логіку для нашого локального сервера: Тепер замість того, щоб мати хард-кодинг HTML-сторінок про помилку глибоко у функції в контролері, я визначаю їх у спеціалізованих класах-обробниках. Коли надходить відповідь, вона проходить по ланцюгу відповідальності, і той клас, який може її обробити (наприклад, 404), автоматично видає конкретну сторінку. Це робить код чистим, модульним і дуже легким для розширення.

2.2. Інструкція користувача

Спочатку впевніться, що у вас встановлене середовище виконання Java, бажано не нижче 17, (Java Runtime Environment), перевіривши це за допомогою командного рядка і команди java -version

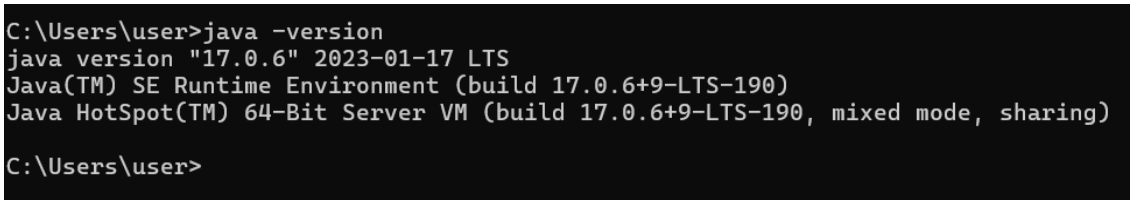

Рис. 10 – Перевірка версії Java у командному рядку

Потім завантажте репозиторій і запустіть .exe файл. Ви маєте побачити поле вводу, кнопку оновлення та адресний рядок.

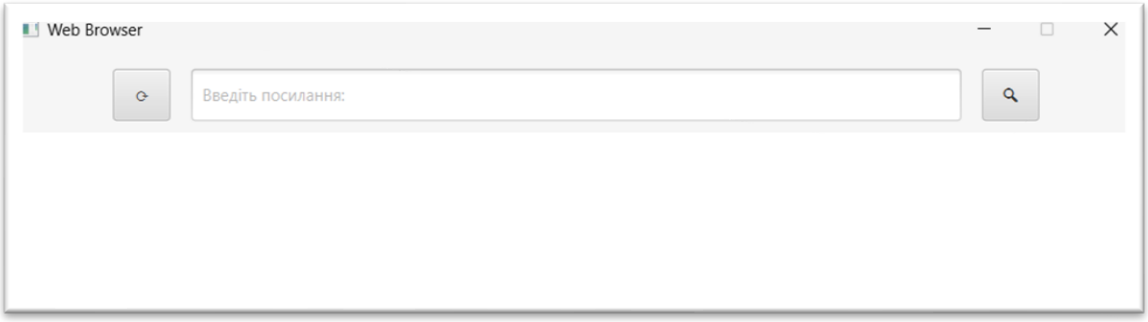

Рис. 11 – Застосунок працює

Як я вже описував вище, я створив окремий сервер для тестування помилок 404, 502 та 503. Оскільки сервер є створеним власноруч, то оброблювати з нього запити я буду за допомогою власного обробника, оскільки по суті, тут дуже простий html та css. Для тестування у вас є посилання на:

- test.com/index.html – код 200, звичайна сторінка

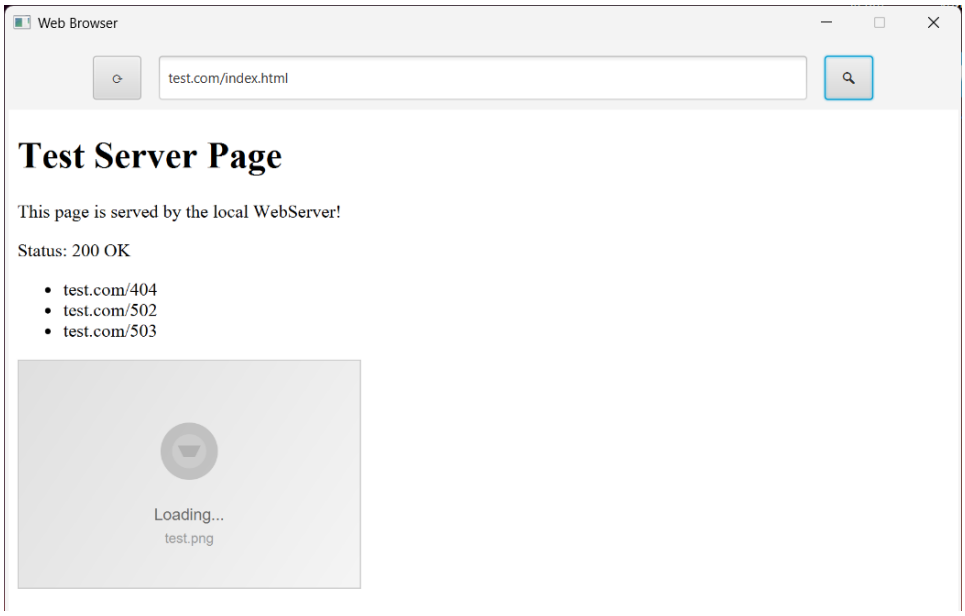

Рис. 12 – Тестова сторінка

- test.com/404– код 404

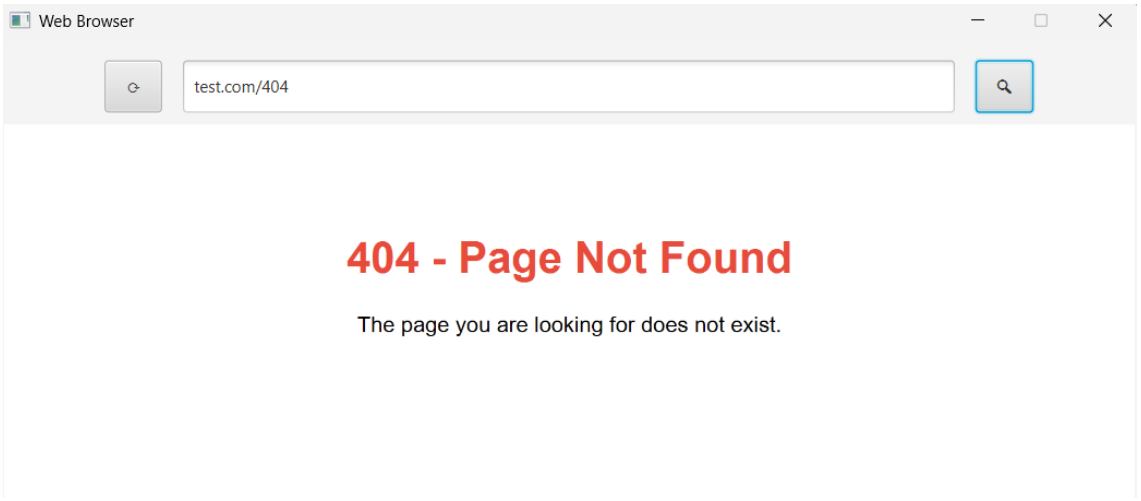

Рис. 13 – Сторінка 404

- test.com/502 – код 502

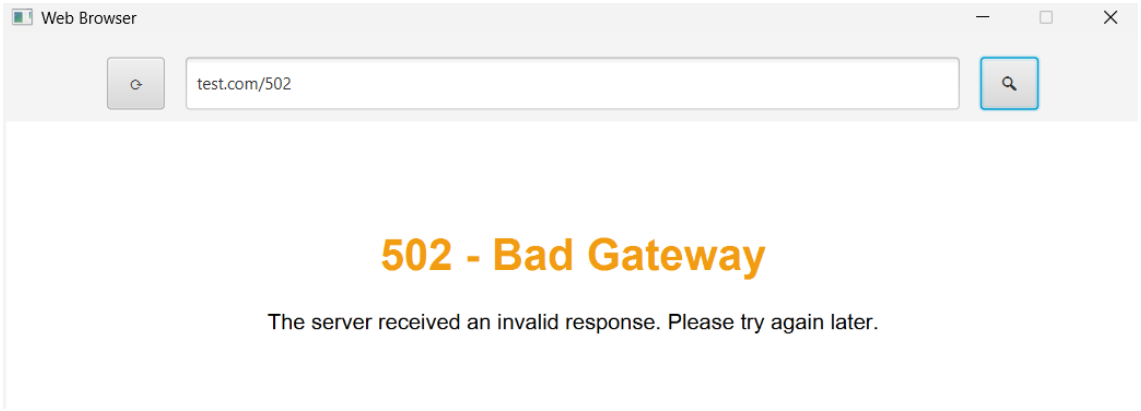

Рис. 14 – Сторінка 502

- test.com/503– код 503

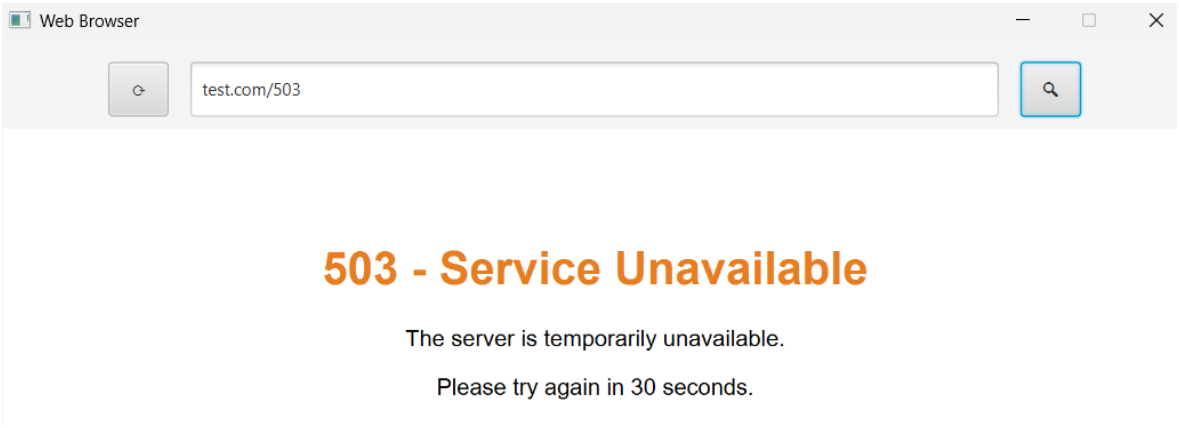

Рис. 15 – Сторінка 503

Для дуже складних сайтів, парсера, звісно, недостатньо, тому я застосував утиліту webEngine.loadPage(), бо зараз сучасні сайти містять дуже багато нових технологій, які виходять за межі стандартних файлів.

Наведу приклад для github.com:

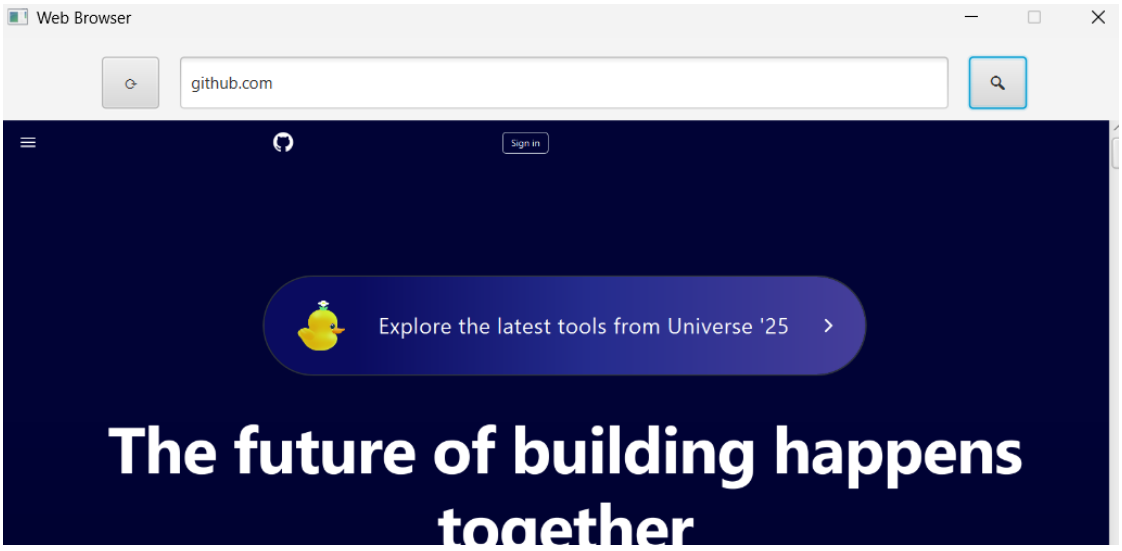

Рис. 16 – Відображення github.com

ВИСНОВКИ

У результаті проведеної роботи, ми прийшли до висновку, що не дивлячись на величезний вибір бразуерів у світі, кожен з яких містить величезну кількість функцій для комфорту користувача, створення власного веббраузера є не лише корисним, але й потужним кроком у освоєнні архітектури проєктування застосунків.

Першим етапом було проектування системи, в якому ми розробили діаграму варіантів використання, розписали сценарії використання системи, створили діаграму розгортання системи і визначилися з набором інструментів для розробки. Було обрано IntellijIdea в якості IDE, оскільки вона містить дуже багато корисних функцій, такі як побудова артефактів, автодоповнення назв методів, вміння знаходити помилки чи рядки, які будуть пропущені. JavaFX було обрано в якості фреймворку, оскільки вона містить в собі роботу з інтерактивними елементами, такі як кнопка, текстові поля і багато інших, які є невід'ємною частиною застосунку.

Другим етапом було реалізація компонентів системи. Я визначився з патернами проектування – Proxy та Chain of Responsibility. Використання цих патернів, очевидно збільшило ієрархію класів, але спростило підтримку проекту. За допомогою Proxy я визначив картинку-заглушку, яка буде завантажуватися до того, як буде завантажене реальне зображення. В середині робочого класу замість прописування умов із завантаження, я просто передав об'єкт інтерфейсу, який може приймати або заглушку, або реальне зображення. Патерн Chain of Responsibility дозволив дуже зручно розділити обробник HTTP-кодів-відповідей. Я створив окремі класи для кожного коду помилки, і визначив HTML-сторінку, яку треба повернути. Створивши ланцюжок і передавши його в обробник, я не переймаюся тим, як буде оброблена ця помилка – ланцюжок все зробить за мене, і мені не доведеться прописувати ці сторінки прямо в методі ініціалізації тестового сервера.

Підсумовуючи пророблену роботу, браузер виконує задані функції – має адресний рядок для вводу адреси сайту, може переміщатися по сторінці, оброблює переходи і помилки 404, 502 та 503 у тестовому сервері. Не дивлячись на те, що для більш складних сайтів довелося використати рішення фреймворку, мій браузер може

обробляти HTML та CSS, що відповідає завданню, проте цей застосунок можна покращити у майбутньому – додати авторизацію, оновити парсер, додати переміщення в історії, виводити власне історію відвідувань, режим інкогніто, або такі сучасні нововедення як режим ШІ від Chrome, який дає відповідь у чаті ШІ, базуючись на власній базі знань.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

[1] Названо лідерів серед браузерів у 2025 році – що обирають українці. URL: https://www.unian.ua/techno/communications/yakiy-brauzer-naypopulyarnishiy-v-ukrajini-edge-nazdoganyaye-chrome-12908850.html

[2] Типи браузерів: інструкції з пошуку найкращого. URL: https://surfshark.com/uk/blog/types-of-browsers?srsltid=AfmBOoqX1tzsSNR9fGcoN_l8M56mAPePIcrVReHmFXOIOo-QreFSzGKG

[3] JavaFX Tutorial: Getting started. URL: https://www.vojtechruzicka.com/javafx-getting-started/

[4] Замісник. URL: https://refactoring.guru/uk/design-patterns/proxy

[5] Ланцюжок обов'язків. URL: https://refactoring.guru/uk/design-patterns/chain-of-responsibility

[6] О. Швець, *Занурення в патерни проектування*, Refactoring.Guru, 2021

[7] Технології розроблення програмного забезпечення: Лабораторний практикум [Електронний ресурс] : навч. посіб. для здобувачів ступеня бакалавра за освітньою програмою «Інтегровані інформаційні системи» за спеціальністю 126 «Інформаційні системи та технології», / КПІ ім. Ігоря Сікорського; укладачі: О.А. Амонс., М.Ю. Мягкий – Електронні текстові дані (1 файл: 2 Мбайт). – Київ : КПІ ім. Ігоря Сікорського, 2025. – 113 с.

[8] What is lazy loading? URL: https://www.cloudflare.com/learning/performance/what-is-lazy-loading/

[9] Proxy Design Pattern. URL: https://www.geeksforgeeks.org/system-design/proxy-design-pattern/

[10] Chain of Responsibility Design Pattern. URL: https://www.geeksforgeeks.org/system-design/chain-responsibility-design-pattern/

ДОДАТКИ

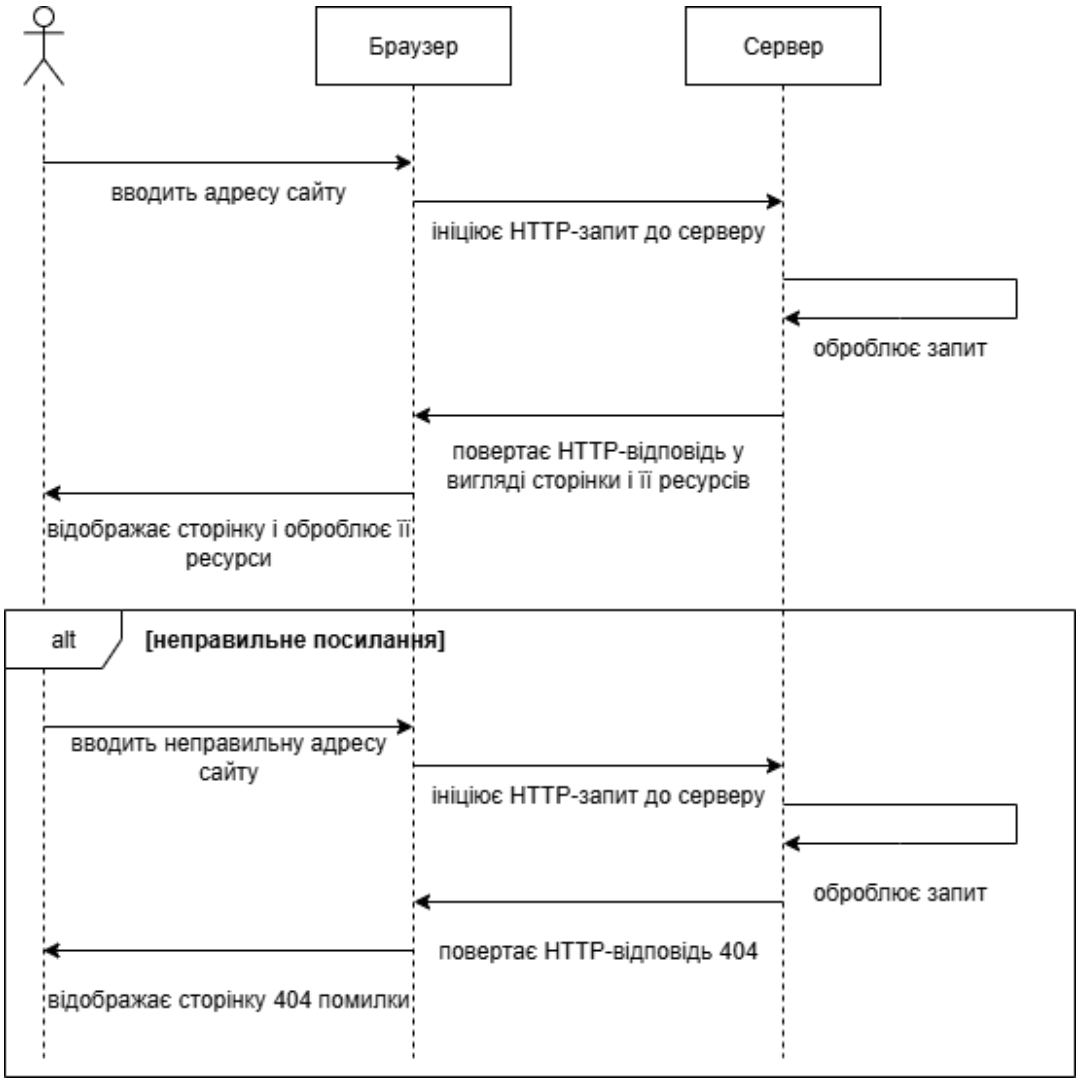

Рис. 2 – Діаграма послідовності для 1-го сценарію

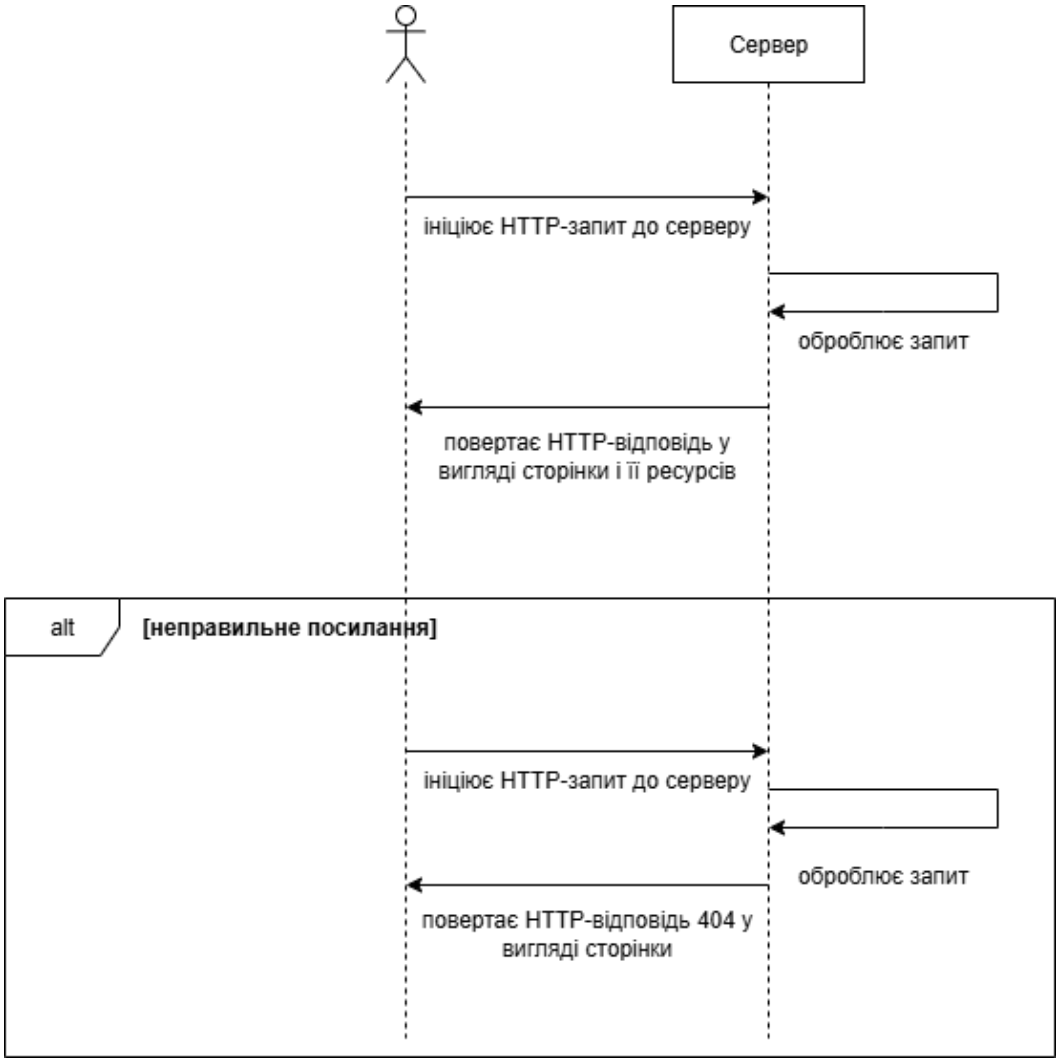

Рис. 3 – Діаграма послідовності для 2-го сценарію

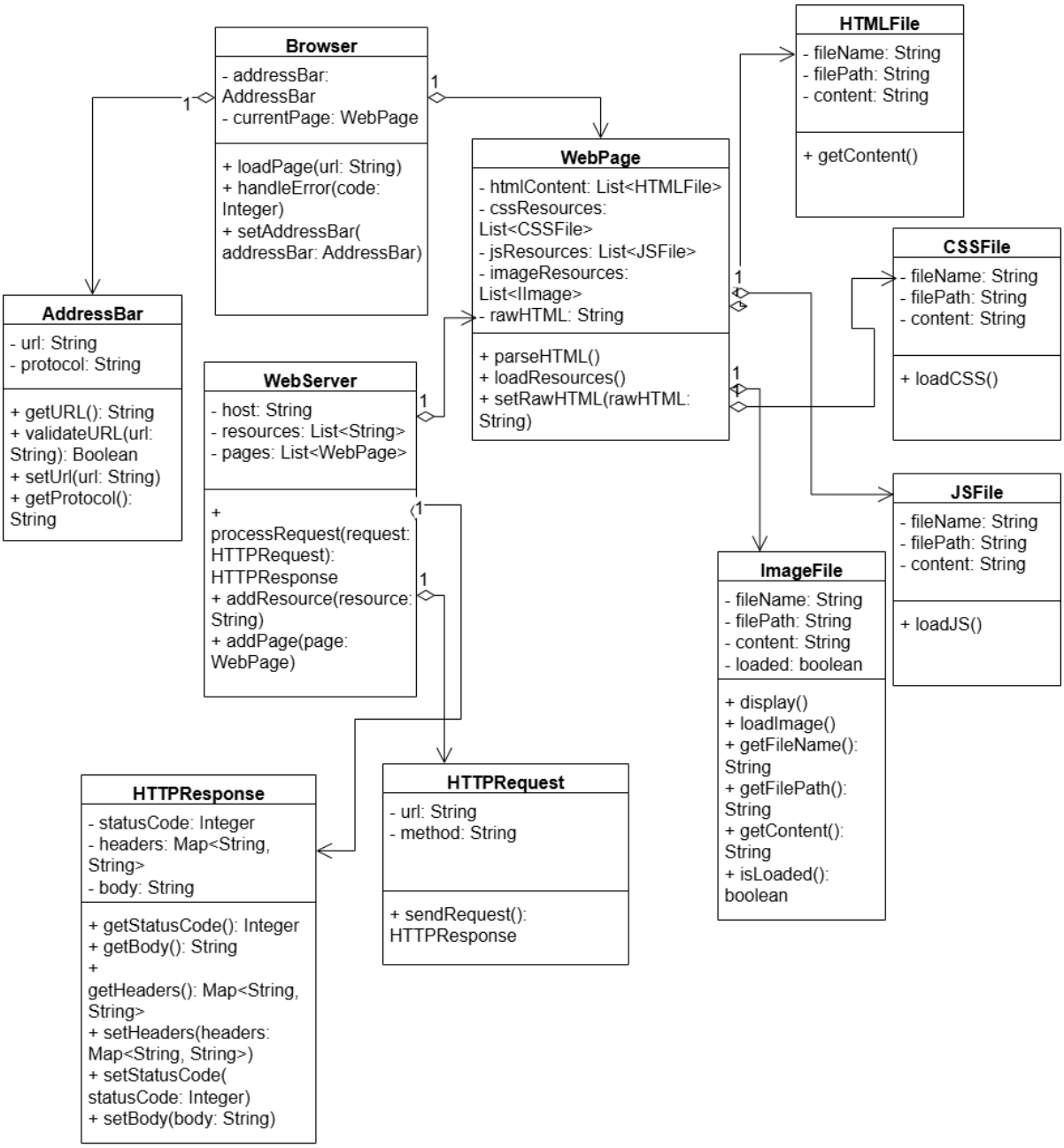

Рис. 4 – Діаграма класів системи

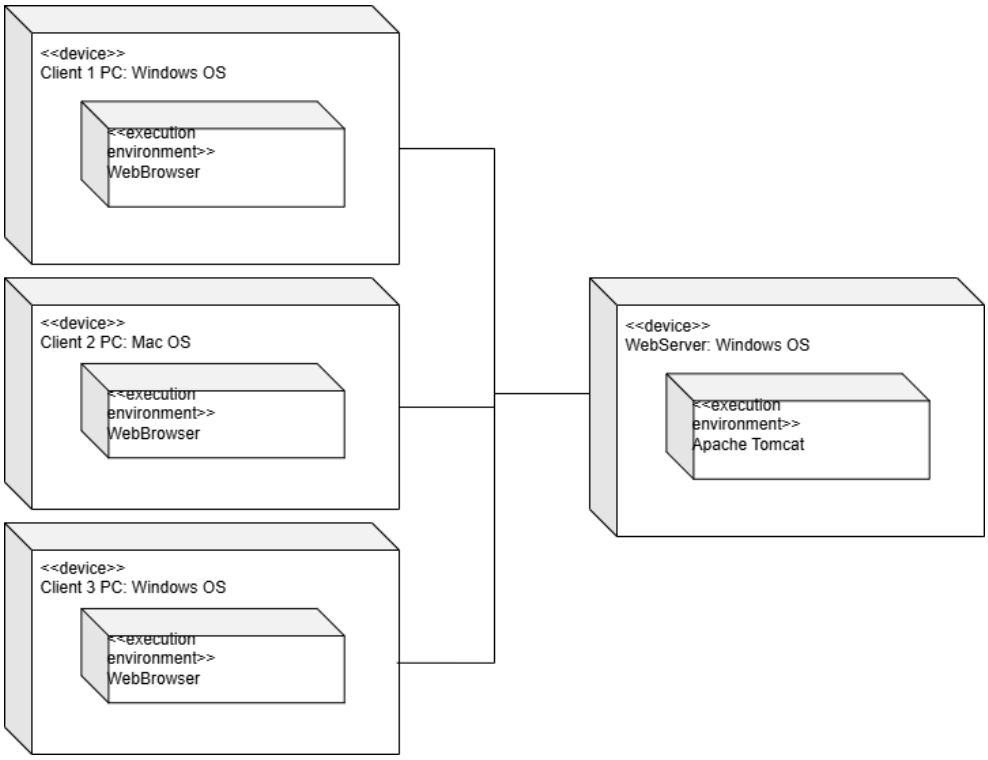

Рис. 5 – Діаграма розгортання системи

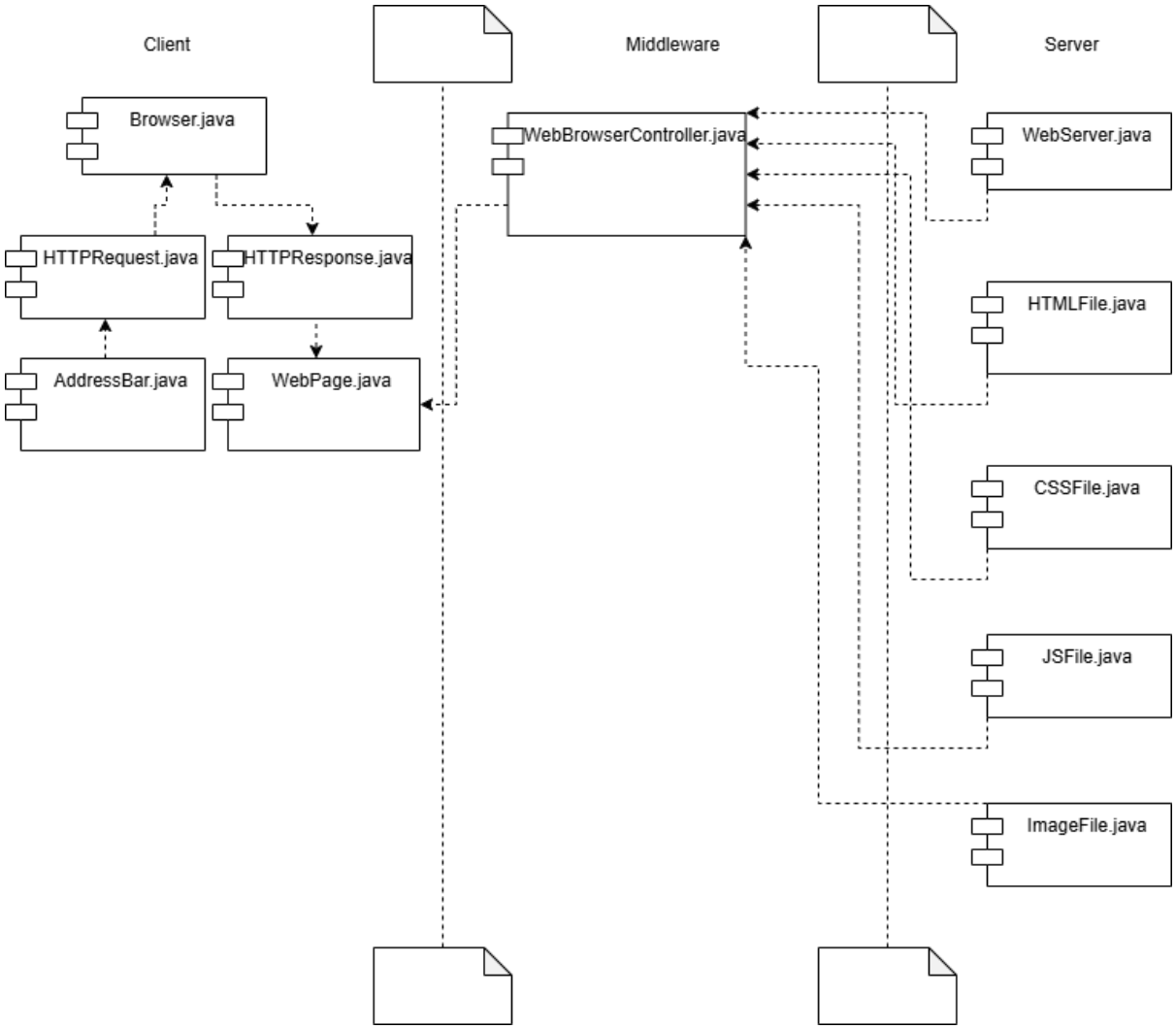

Рис. 6 – Діаграма компонентів системи

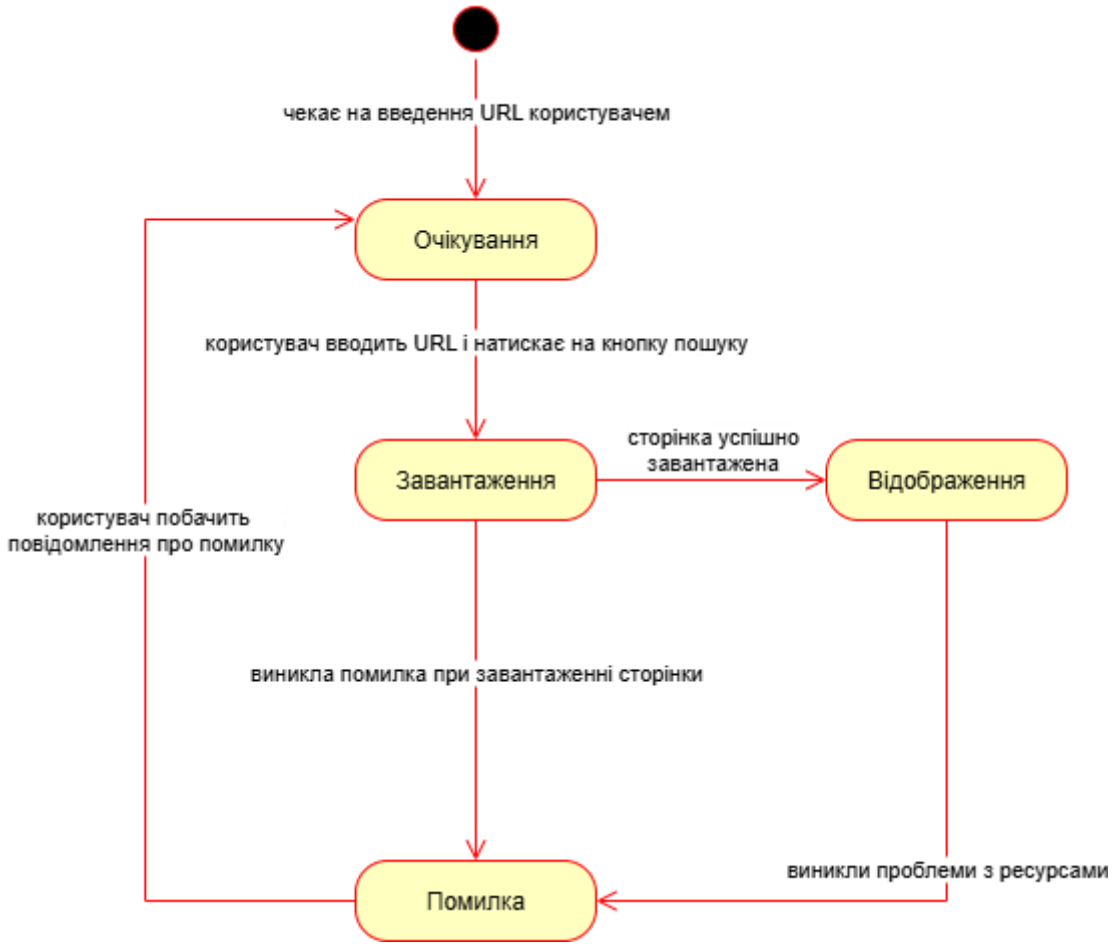

Рис. 7 – Діаграма станів для процесу «Завантаження веб-сторінки»

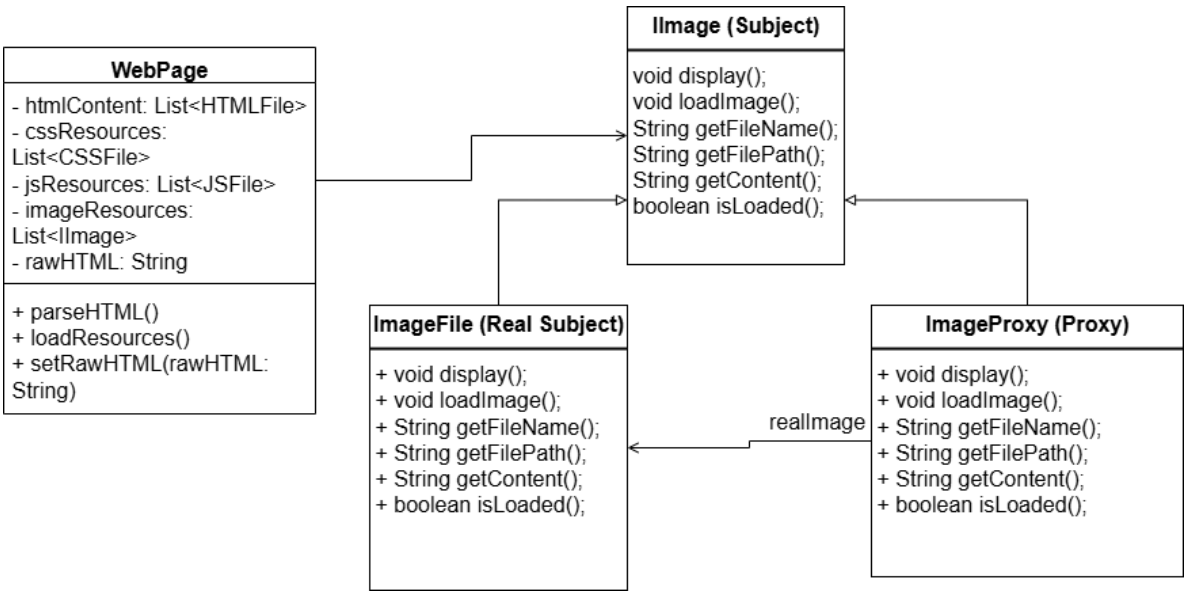

Рис. 8 – Схема шаблону «Proxy»

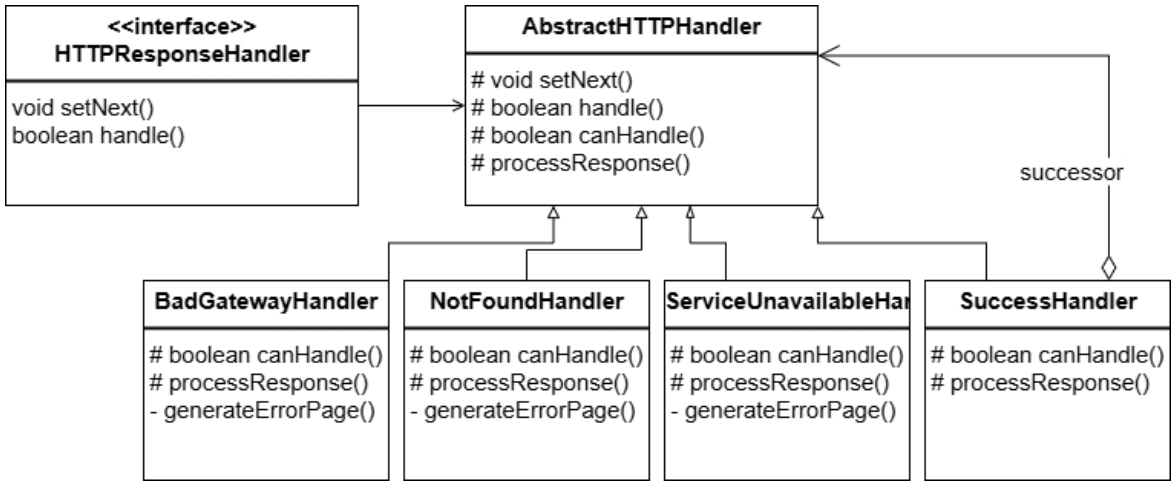

Рис. 9 – Схема шаблону «Chain of Responsibility»

Код для шаблону «Proxy»:

```
package org.example.webbrowser;

public interface IImage {
    void display();

    void loadImage();

    String getFileName();

    String getFilePath();

    String getContent();

    boolean isLoaded();
}
```

```
package org.example.webbrowser;

import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.Base64;

public class ImageFile implements IImage {
    private String fileName;
    private String filePath;
    private String content; // Base64 encoded content
    private boolean loaded;

    public ImageFile(String fileName, String filePath) {
        this.fileName = fileName;
        this.filePath = filePath;
        this.content = "";
        this.loaded = false;
    }

    @Override
    public String getFileName() {
        return fileName;
```

```
}

public void setFileName(String fileName) {
   this.fileName = fileName;
}

@Override
public String getFilePath() {
   return filePath;
}

public void setFilePath(String filePath) {
   this.filePath = filePath;
}

@Override
public String getContent() {
   return content;
}

public void setContent(String content) {
   this.content = content;
}

@Override
public boolean isLoaded() {
   return loaded;
}

@Override
public void loadImage() {
   try {
      byte[] fileContent = Files.readAllBytes(Paths.get(filePath));
      this.content = Base64.getEncoder().encodeToString(fileContent);
      this.loaded = true;
   } catch (IOException e) {
      this.loaded = false;
   }
}
```

```
    @Override
    public void display() {
        if (!loaded) {
            loadImage();
        }
    }
}
```

```
package org.example.webbrowser;

import java.util.Base64;

public class ImageProxy implements IImage {
    private ImageFile realImage; // RealSubject
    private String fileName;
    private String filePath;
    private String placeholderContent; // Заглушка
    private boolean isRealImageLoaded;

    public ImageProxy(String fileName, String filePath) {
        this.fileName = fileName;
        this.filePath = filePath;
        this.isRealImageLoaded = false;
        this.placeholderContent = createPlaceholder();
    }

    @Override
    public String getFileName() {
        return fileName;
    }

    @Override
    public String getFilePath() {
        return filePath;
    }

    @Override
    public String getContent() {
        if (isRealImageLoaded && realImage != null) {
            return realImage.getContent();
        }
```

```
        return placeholderContent;
    }

    @Override
    public boolean isLoaded() {
        return isRealImageLoaded;
    }

    @Override
    public void display() {
        loadImage();
    }

    @Override
    public void loadImage() {
        if (!isRealImageLoaded) {

            realImage = new ImageFile(fileName, filePath);
            realImage.loadImage();

            isRealImageLoaded = true;
        }
    }

    public String createPlaceholder() {
        String svg = String.format("""
            <svg width='300' height='200' xmlns='http://www.w3.org/2000/svg'>
                <defs>
                    <linearGradient id='grad' x1='0%%' y1='0%%' x2='100%%' y2='100%%'>
                        <stop offset='0%%' style='stop-color:#e0e0e0;stop-opacity:1' />
                        <stop offset='100%%' style='stop-color:#f5f5f5;stop-opacity:1' />
                    </linearGradient>
                </defs>
                <rect width='300' height='200' fill='url(#grad)' stroke='#cccccc' stroke-
width='2'/>
                <circle cx='150' cy='80' r='25' fill='#999999' opacity='0.5'/>
                <circle cx='150' cy='80' r='15' fill='#cccccc'/>
                <polygon points='140,75 145,85 155,85 160,75' fill='#999999' opacity='0.5'/>
                <text x='150' y='140' text-anchor='middle' font-family='Arial' font-size='14'
fill='#666666'>
```

```
                Loading...
            </text>
            <text x='150' y='160' text-anchor='middle' font-family='Arial' font-size='12'
fill='#999999'>
                %s
            </text>
        </svg>
        """, fileName);

        return "" +
Base64.getEncoder().encodeToString(svg.getBytes());
    }
}


package org.example.webbrowser;

import java.util.ArrayList;
import java.util.List;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class WebPage {
    private List<HTMLFile> htmlResources;
    private List<CSSFile> cssResources;
    private List<JSFile> jsResources;
    private List<IImage> imageResources;
    private String rawHTML;

    public WebPage() {
        this.htmlResources = new ArrayList<>();
        this.cssResources = new ArrayList<>();
        this.jsResources = new ArrayList<>();
        this.imageResources = new ArrayList<>();
    }

    public List<HTMLFile> getHtmlResources() {
        return htmlResources;
    }
```

```
public List<CSSFile> getCssResources() {
   return cssResources;
}

public List<JSFile> getJsResources() {
   return jsResources;
}

public List<IImage> getImageResources() {
   return imageResources;
}

public void setRawHTML(String rawHTML) {
   this.rawHTML = rawHTML;
}

public String getRawHTML() {
   return rawHTML;
}

public void parseHTML() {
   if (rawHTML == null || rawHTML.isEmpty()) {
      return;
   }

   Pattern cssPattern = Pattern.compile("<link[^>]*href=[\'"]([^\'"]*\\.css)[\'"][^>]*>");
   Matcher cssMatcher = cssPattern.matcher(rawHTML);
   while (cssMatcher.find()) {
      String cssPath = cssMatcher.group(1);
      CSSFile cssFile = new CSSFile(extractFileName(cssPath), cssPath);
      cssResources.add(cssFile);
   }

   Pattern jsPattern = Pattern.compile("<script[^>]*src=[\'"]([^\'"]*\\.js)[\'"][^>]*>");
   Matcher jsMatcher = jsPattern.matcher(rawHTML);
   while (jsMatcher.find()) {
      String jsPath = jsMatcher.group(1);
      JSFile jsFile = new JSFile(extractFileName(jsPath), jsPath);
      jsResources.add(jsFile);
   }
```

```
        Pattern imgPattern = Pattern.compile("<img[^>]*src=[\"']([^\"']*)[\"'][^>]*>");
        Matcher imgMatcher = imgPattern.matcher(rawHTML);
        while (imgMatcher.find()) {
            String imgPath = imgMatcher.group(1);
            ImageFile imageFile = new ImageFile(extractFileName(imgPath), imgPath);
            imageResources.add(imageFile);
        }

        HTMLFile mainHTML = new HTMLFile("index.html", "index.html", rawHTML);
        htmlResources.add(mainHTML);
    }


    public void loadResources() {
        for (CSSFile css : cssResources) {
            css.loadCSS();
        }

        for (JSFile js : jsResources) {
            js.loadJS();
        }

        for (IImage img : imageResources) {
            img.loadImage();
        }
    }

    private String extractFileName(String path) {
        int lastSlash = path.lastIndexOf('/');
        return lastSlash >= 0 ? path.substring(lastSlash + 1) : path;
    }
}
```

Код для шаблону «Chain of Responsibility»:

```
package org.example.webbrowser;

/**
 * Chain of Responsibility Pattern: Handler Interface
```

```
 *
 * This interface defines the contract for HTTP response handlers.
 * Each handler in the chain can process a specific HTTP status code
 * or pass the request to the next handler in the chain.
 */
public interface HTTPResponseHandler {

    /**
     * Sets the next handler in the chain
     *
     * @param next The next handler to call if this handler cannot process the response
     */
    void setNext(HTTPResponseHandler next);

    /**
     * Handles the HTTP response
     * Each handler checks if it can process the given status code.
     * If yes, it handles the response and returns true.
     * If no, it passes the response to the next handler in the chain.
     *
     * @param response The HTTP response to handle
     * @return true if the response was handled, false otherwise
     */
    boolean handle(HTTPResponse response);
}


package org.example.webbrowser;

public abstract class AbstractHTTPHandler implements HTTPResponseHandler {

    protected HTTPResponseHandler nextHandler;

    @Override
    public void setNext(HTTPResponseHandler next) {
        this.nextHandler = next;
    }

    @Override
    public boolean handle(HTTPResponse response) {
```

```
        if (canHandle(response)) {
            processResponse(response);
            return true;
        } else if (nextHandler != null) {
            return nextHandler.handle(response);
        }
        return false;
    }

    /**
     * Determines if this handler can process the given response
     *
     * @param response The HTTP response
     * @return true if this handler can process the response
     */
    protected abstract boolean canHandle(HTTPResponse response);

    /**
     * Processes the HTTP response
     * This method is called only if canHandle() returns true
     *
     * @param response The HTTP response to process
     */
    protected abstract void processResponse(HTTPResponse response);
}
package org.example.webbrowser;

/**
 * Concrete Handler for HTTP 502 Bad Gateway responses
 */
public class BadGatewayHandler extends AbstractHTTPHandler {

    @Override
    protected boolean canHandle(HTTPResponse response) {
        return response.getStatusCode() == 502;
    }

    @Override
    protected void processResponse(HTTPResponse response) {
        // Enhance error response
```

```
        if (!response.getBody().contains("502")) {
            String errorPage = generateErrorPage();
            response.setBody(errorPage);
        }

        response.getHeaders().put("X-Handled-By", "BadGatewayHandler");
        response.getHeaders().put("X-Error-Type", "Server Error");
        response.getHeaders().put("X-Retry-Recommended", "true");
    }

    /**
     * Generates a user-friendly 502 error page
     *
     * @return HTML content for 502 error page
     */
    private String generateErrorPage() {
        return """
            <!DOCTYPE html>
            <html>
            <head>
                <title>502 - Bad Gateway</title>
                <style>
                    body { font-family: Arial; text-align: center; padding: 50px; }
                    h1 { color: #f39c12; }
                </style>
            </head>
            <body>
                <h1>502 - Bad Gateway</h1>
                <p>The server received an invalid response. Please try again later.</p>
            </body>
            </html>
            """;
    }
}
```

```
package org.example.webbrowser;

/**
 * Concrete Handler for HTTP 404 Not Found responses
 */
public class NotFoundHandler extends AbstractHTTPHandler {
```

```
@Override
protected boolean canHandle(HTTPResponse response) {
    return response.getStatusCode() == 404;
}

@Override
protected void processResponse(HTTPResponse response) {
    if (!response.getBody().contains("404")) {
        String errorPage = generateErrorPage();
        response.setBody(errorPage);
    }

    response.getHeaders().put("X-Handled-By", "NotFoundHandler");
    response.getHeaders().put("X-Error-Type", "Client Error");
}

/**
 * Generates 404 error page
 *
 * @return HTML content for 404 error page
 */
private String generateErrorPage() {
    return """
        <!DOCTYPE html>
        <html>
        <head>
            <title>404 - Not Found</title>
            <style>
                body { font-family: Arial; text-align: center; padding: 50px; }
                h1 { color: #e74c3c; }
            </style>
        </head>
        <body>
            <h1>404 - Page Not Found</h1>
            <p>The page you are looking for does not exist.</p>
        </body>
        </html>
        """;
```

```
    }
}
package org.example.webbrowser;

/**
 * Concrete Handler for HTTP 503 Service Unavailable responses
 */
public class ServiceUnavailableHandler extends AbstractHTTPHandler {

    @Override
    protected boolean canHandle(HTTPResponse response) {
        return response.getStatusCode() == 503;
    }

    @Override
    protected void processResponse(HTTPResponse response) {
        // Check for Retry-After header
        String retryAfter = response.getHeaders().get("Retry-After");

        // Enhance error response
        if (!response.getBody().contains("503")) {
            String errorPage = generateErrorPage(retryAfter);
            response.setBody(errorPage);
        }

        response.getHeaders().put("X-Handled-By", "ServiceUnavailableHandler");
        response.getHeaders().put("X-Error-Type", "Server Error");
        response.getHeaders().put("X-Retry-Recommended", "true");
    }

    /**
     * Generates a user-friendly 503 error page
     *
     * @param retryAfter Suggested retry time (can be null)
     * @return HTML content for 503 error page
     */
    private String generateErrorPage(String retryAfter) {
        String retryMessage = retryAfter != null
            ? "<p>Please try again in " + retryAfter + " seconds.</p>"
            : "<p>Please try again in a few moments.</p>";
```

```
        return """
            <!DOCTYPE html>
            <html>
            <head>
                <title>503 - Service Unavailable</title>
                <style>
                    body { font-family: Arial; text-align: center; padding: 50px; }
                    h1 { color: #e67e22; }
                </style>
            </head>
            <body>
                <h1>503 - Service Unavailable</h1>
                <p>The server is temporarily unavailable.</p>
                """ + retryMessage + """
            </body>
            </html>
            """;
    }
}
```

```
package org.example.webbrowser;
public class SuccessHandler extends AbstractHTTPHandler {

    @Override
    protected boolean canHandle(HTTPResponse response) {
        return response.getStatusCode() == 200;
    }

    @Override
    protected void processResponse(HTTPResponse response) {
        response.getHeaders().put("X-Handled-By", "SuccessHandler");
    }
}
```

```
package org.example.webbrowser;

public class HTTPHandlerChain {

    private HTTPResponseHandler firstHandler;
```

```
    public HTTPHandlerChain() {
        buildDefaultChain();
    }

    /**
     * Builds the default chain of handlers
     */
    private void buildDefaultChain() {
        // Create handlers
        HTTPResponseHandler notFoundHandler = new NotFoundHandler();
        HTTPResponseHandler badGatewayHandler = new BadGatewayHandler();
        HTTPResponseHandler serviceUnavailableHandler = new
ServiceUnavailableHandler();
        HTTPResponseHandler successHandler = new SuccessHandler();

        // Link handlers in chain
        successHandler.setNext(notFoundHandler);
        notFoundHandler.setNext(badGatewayHandler);
        badGatewayHandler.setNext(serviceUnavailableHandler);

        // Set first handler
        firstHandler = successHandler;
    }

    /**
     * Processes an HTTP response through the handler chain
     *
     * @param response The HTTP response to process
     * @return true if response was handled by any handler in the chain
     */
    public boolean process(HTTPResponse response) {
        if (firstHandler == null) {
            return false;
        }
        return firstHandler.handle(response);
    }
}
```