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)
« » _____2025р.
Захищений з оцінкою
_____
(оцінка)
Члени комісії:
_____ _____
(особистий підпис) (розшифровка підпису)
_____ _____
(особистий підпис) (розшифровка підпису)

Виконавець
ст. ____Самойленко С.Д._____
залікова книжка № ___ – _____
гр. ____ІА-32____
_____
(особистий підпис виконавця)
« »_____2025р.

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___ Група_____Семестр_____

ЗАВДАННЯ

на курсову роботу студента

Самойленка Станіслава Дмитровича

(прізвище, ім’я, по батькові)

1.Тема роботи: Web-browser

2. Строк здачі студентом закінченої роботи_____

3. Вихідні дані до роботи:

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.

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

Проектування системи (огляд існуючих рішень, опис проекту, вимоги до застосунків системи (функціональні/нефункціональні), сценарії використання системи, концептуальна модель системи, вибір бази даних, мови програмування та середовища розробки, проєктування розгортання системи), реалізація компонентів системи (структура бази даних, вибір і обґрунтування патернів реалізації, інструкція користувача).

Додатки:

_____

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

_____

_____

_____

_____

6. Дата видачі завдання_____

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.	Проєктування та 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	17.12.2025	
6.			
7.			
8.			
9.			

Студент _____ **Станіслав САМОЙЛЕНКО**
(підпис) (Ім'я ПРІЗВИЩЕ)

Керівник_____ Олександр АМОНС
(підпис) (Ім'я ПРІЗВИЩЕ)

«____»_____ 20___ р.

ЗМІСТ

ВСТУП

Сучасні веб-технології стали основою для розвитку глобальних інформаційних систем і цифрової комунікації, що стали невід'ємною частиною щоденного життя. Однією з ключових складових цих технологій є веб-браузери, які виконують функцію клієнтського програмного забезпечення для запиту, обробки та інтерактивного відображення веб-ресурсів, таких як сторінки, зображення та файли. Взаємодія браузера з серверами базується на протоколі HTTP (HyperText Transfer Protocol), який забезпечує обмін даними у всесвітній мережі. Розробка веб-браузера є комплексним завданням, яке вимагає глибоких знань в області мережевих протоколів, стандартів рендерингу (HTML, CSS) та принципів роботи веб-додатків. У даній курсовій роботі розглядається розробка клієнтського програмного забезпечення, здатного коректно інтерпретувати веб-контент та надавати користувачу зручний інтерфейс для взаємодії з ним.

Основною метою цієї роботи є створення програмної моделі веб-браузера, який здатний не тільки приймати URL-адресу від користувача, але й коректно відображати структуру HTML-документа, надавати засоби для перегляду підключених CSS та JavaScript файлів, а також інспектувати завантажені ресурси. Важливим аспектом є ефективна обробка відповідей сервера, що забезпечується використанням відповідних архітектурних рішень та алгоритмів. Одним з найбільш важливих завдань є розпізнавання кодів стану HTTP, таких як перенаправлення (3xx), помилки клієнта (404) або помилки сервера (502, 503), та правильне реагування на них, що формує кінцевий досвід користувача.

Ще однією ключовою частиною є розбір (парсинг) отриманого HTML-документа та побудова його об'єктної моделі (DOM). Аналіз та обхід цієї структури є фундаментальною складовою функціонування браузера, оскільки це дозволяє не лише коректно візуалізувати сторінку, але й виявляти та завантажувати всі пов'язані ресурси. Ефективна обробка вузлів документа вимагає гнучких підходів, які дозволяють додавати нові операції до елементів структури, не змінюючи їхні базові

класи. Це є критичним для коректного рендерингу, збору інформації про ресурси чи виконання інших операцій над деревом документа.

Особливу увагу слід приділити розробці системи завантаження ресурсів та обробки відповідей, яка повинна бути гнучкою і розширюваною. Це вимагає побудови послідовної логіки обробки HTTP-відповідей, де різні обробники можуть бути динамічно скомбіновані для реагування на свій діапазон кодів (наприклад, успіх, перенаправлення, помилка). Водночас, необхідно визначити стандартизований алгоритм для загального процесу завантаження сторінки, залишаючи при цьому можливість для варіативності на окремих етапах. Також потрібні гнучкі механізми для створення спеціалізованих об'єктів, що відповідають за обробку різних типів ресурсів (HTML, CSS, зображення).

У результаті розробки моделі веб-браузера в рамках цієї курсової роботи, буде реалізовано ефективне клієнтське програмне забезпечення, яке здатне коректно взаємодіяти з веб-серверами, надавати користувачам доступ до структурованої веб-інформації та демонструвати практичне застосування надійних архітектурних рішень. Це дозволить створити гнучку архітектуру, готову до подальшого вдосконалення та імплементації більш складного функціоналу.

1 ПРОЄКТУВАННЯ СИСТЕМИ

1.1. Огляд існуючих рішень

Сучасний ринок програмного забезпечення для перегляду веб-сторінок представлений великою кількістю рішень, що задовольняють різноманітні потреби користувачів. Серед них можна виділити як домінуючі браузери загального призначення, так і спеціалізовані рішення, орієнтовані на приватність або конкретні екосистеми. Огляд цих рішень дозволяє краще зрозуміти сучасні тенденції у сфері розробки клієнтського програмного забезпечення та визначити переваги та недоліки різних підходів.

Google Chrome

Google Chrome — це, безперечно, один з найбільш популярних і широко використовуваних веб-браузерів у світі. Він розроблений компанією Google і базується на відкритому проекті Chromium. Chrome відомий своєю високою швидкістю обробки JavaScript завдяки рушію V8 та швидкому рендерингу сторінок за допомогою рушія Blink. Він має величезну екосистему розширень, що дозволяє значно розширювати його функціональність, а також глибоку інтеграцію з сервісами Google.

Однак, незважаючи на свою швидкість, Chrome має суттєві недоліки, зокрема, дуже високе споживання оперативної пам'яті та ресурсів процесора. Це може бути проблемою для систем з обмеженими апаратними можливостями. Крім того, існують постійні занепокоєння щодо конфіденційності даних користувачів, оскільки браузер активно збирає телеметрію для Google.

Mozilla Firefox

Firefox — це ще один популярний веб-браузер, який розробляється Mozilla Foundation і є проектом з повністю відкритим вихідним кодом. Його головною перевагою вважається сильний акцент на приватності та безпеці користувачів. Firefox використовує власний рушій рендерингу Gecko, що сприяє різноманітності веб-технологій. Він пропонує потужні вбудовані інструменти блокування трекерів, гнучкі налаштування та активну спільноту розробників.

Однією з переваг Firefox є його відносно помірне використання ресурсів порівняно з Chrome. Він також підтримує велику бібліотеку доповнень. Однак, у деяких тестах продуктивності він може дещо поступатися конкурентам, хоча для більшості користувачів ця різниця не є критичною.

Apple Safari

Safari — це веб-браузер, розроблений Apple, який є стандартним для всіх пристроїв цієї компанії, таких як macOS та iOS. Його ключова перевага — виняткова оптимізація під апаратне забезпечення Apple, що забезпечує найвищу енергоефективність та плавність роботи в цій екосистемі. Safari використовує рушій WebKit і також приділяє значну увагу приватності, зокрема завдяки функції "Intelligent Tracking Prevention" (Інтелектуальне запобігання відстеженню).

Проте Safari має суттєве обмеження: він доступний виключно на пристроях Apple. Це унеможливлює його використання на Windows чи Android. Крім того, його екосистема розширень значно менша, ніж у Chrome чи Firefox, а впровадження деяких новітніх веб-стандартів іноді відбувається повільніше.

Кожне з існуючих рішень має свої переваги та обмеження, що робить їх підходящими для різних сценаріїв використання. Для максимальної сумісності та доступу до найбільшої кількості розширень часто обирають Chrome. Для користувачів, які ставлять на перше місце приватність та відкритість, кращим вибором є Firefox. Для користувачів екосистеми Apple найкращим рішенням з точки зору продуктивності є Safari.

1.2. Загальний опис проєкту

Проєкт має на меті розробку програмної моделі веб-браузера, який здатний обробляти URL-адреси, введені користувачем, та формувати коректні запити до серверів відповідно до стандартів протоколу HTTP. Браузер повинен мати здатність отримувати відповіді, парсити (розбирати) та відображати структуру HTML-документів, а також надавати можливість інспектування підключених ресурсів, таких як CSS та JavaScript файли. Окрім цього, важливим елементом є реалізація коректної обробки кодів стану HTTP, наприклад, виконання переходів при перенаправленнях (redirects) або відображення сторінок помилок (404, 502/503).

Проєкт буде реалізовано як клієнтське програмне забезпечення, яке може працювати на операційних системах з підтримкою мережевих протоколів, таких як Linux, Windows чи macOS. Він забезпечить функціонування перегляду веб-сайтів шляхом надсилання HTTP-запитів та надання користувачу доступу до необхідних ресурсів у відповідь. Це також дозволить ефективно аналізувати структуру веб-сторінок через інспектування їхніх компонентів.

В рамках проєкту модель браузера буде реалізована з урахуванням важливих аспектів коректної інтерпретації стандартів, модульності та надійності. Одним з основних завдань є коректна реалізація обробки відповідей згідно з вимогами

протоколу HTTP, що включає інтерпретацію статусних кодів, заголовків та тіла відповіді. Особливу увагу буде приділено обробці помилок і забезпеченню правильного зворотного зв'язку для користувача, що дозволяє уникнути непередбачуваних ситуацій та забезпечити високий рівень стабільності роботи клієнта.

Важливим компонентом є можливість інспектування завантажених ресурсів, яка дозволить відслідковувати список підключених файлів (CSS, JS, зображень) та, можливо, переглядати їх вміст. Це є важливим для розуміння структури сторінки та аналізу її компонентів. Ця функціональність буде використовуватися для демонстрації процесу завантаження сторінки, що дозволяє наочно показати, з яких частин вона складається та як вони взаємопов'язані.

Обробка навігації та завантаження ресурсів є ще одним важливим аспектом проєкту. Це вимагає гнучкої архітектури, здатної застосовувати різні стратегії обробки. Наприклад, логіка завантаження сторінки може бути визначена у вигляді загального алгоритму, а для обробки різних типів відповідей (успіх, помилка, перенаправлення) може бути використаний гнучкий механізм послідовної обробки. Вибір архітектурних рішень буде залежати від вимог до гнучкості та розширюваності системи.

Загалом, цей проєкт спрямований на створення гнучкої та розширюваної моделі веб-браузера, яка здатна не тільки коректно інтерпретувати відповіді серверів, але й забезпечувати глибоке розуміння структури веб-документів та механізмів їх завантаження, що є важливим аспектом для вивчення теорії розробки програмного забезпечення.

1.3. Вимоги до застосунків системи

1.3.1. Функціональні вимоги до системи

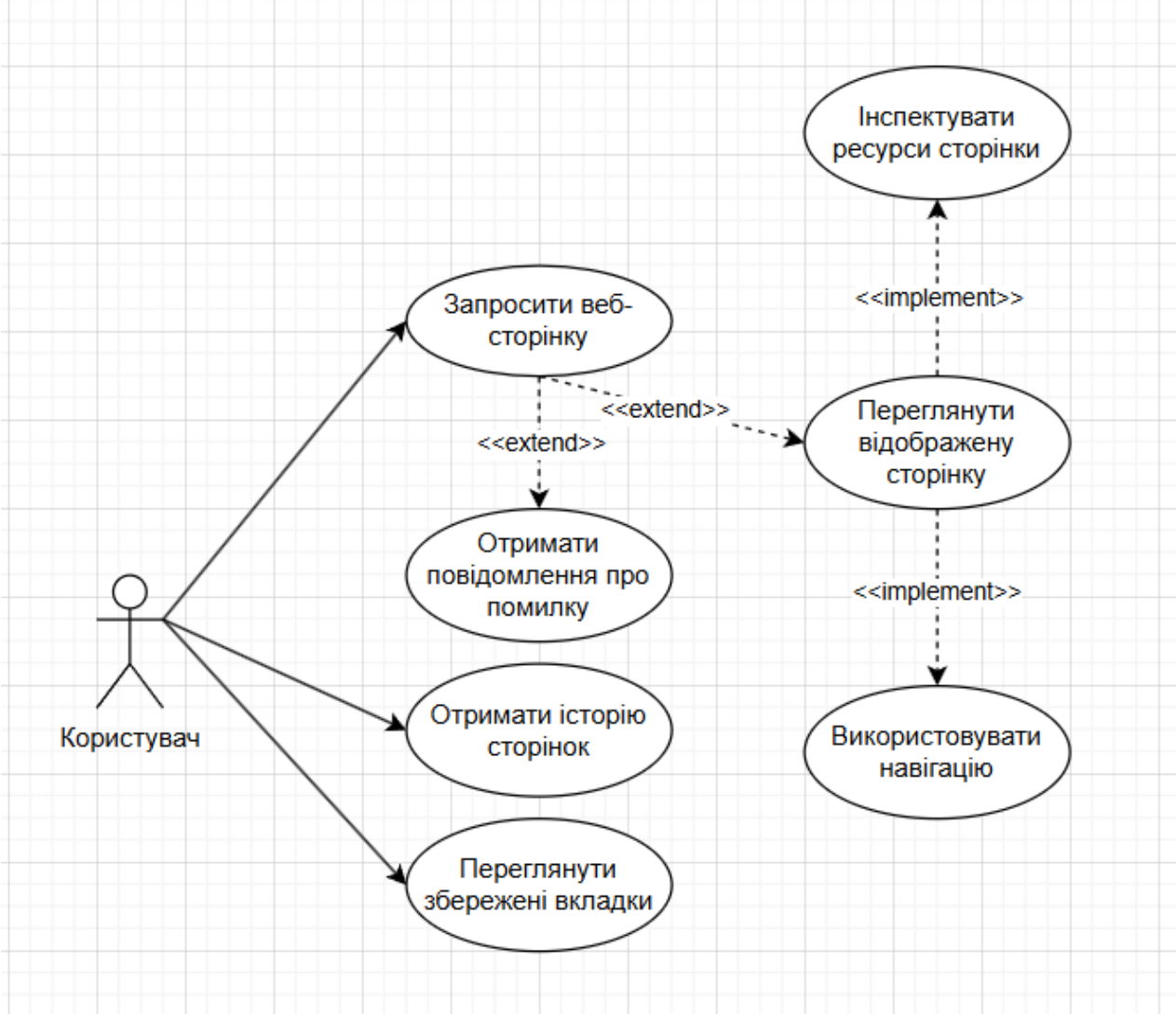

Рис. 1.3.1.1 – Діаграма використання для веб-браузера

Функціональні вимоги до системи:

а) Надсилання HTTP-запитів: Система повинна надавати користувачу адресний рядок для введення URL-адреси. Після підтвердження введення, система має формувати та надсилати коректний HTTP-запит (переважно GET) до вказаного сервера для отримання веб-сторінки.

б) Обробка та відображення HTML: Система повинна коректно обробляти успішні HTTP-відповіді (статус 200 OK). Це включає парсинг (розбір) отриманого HTML-документа, побудову його структури та візуальне відображення ("рендеринг") для користувача (use case "Переглянути відображену сторінку").

в) Обробка помилок та перенаправлень: Система повинна коректно інтерпретувати коди стану HTTP у відповідях.

- При отриманні кодів перенаправлення (3xx), система повинна автоматично виконувати перехід за новою адресою, вказаною сервером.
- При отриманні помилок (404, 502, 503 тощо), система повинна відображати користувачу відповідну сторінку або повідомлення про помилку (use case "Отримати повідомлення про помилку").

г) Інспектування ресурсів сторінки: Система повинна дозволяти користувачу переглядати список всіх підключених до HTML-документа ресурсів. Це включає можливість ідентифікувати та переглянути список підключених файлів CSS, JavaScript та зображень.

д) Забезпечення навігації: Система повинна відстежувати історію переміщень користувача в рамках поточної сесії. Користувач повинен мати можливість використовувати елементи керування "Назад" та "Вперед" для переміщення між переглянутими сторінками.

е) Доступ до історії переглядів: Система повинна вести постійний журнал (історію) відвіданих користувачем URL-адрес. Користувач повинен мати доступ до цього журналу для перегляду та швидкого переходу до раніше відвіданих сторінок.

є) Керування збереженими вкладками (закладками): Система повинна надавати користувачу функціонал для збереження URL-адрес (створення "закладок"). Користувач повинен мати можливість переглянути список збережених вкладок та перейти за обраною адресою.

1.3.2. Нефункціональні вимоги до системи

Нефункціональні вимоги до системи:

а) Продуктивність (Швидкодія): Система повинна забезпечувати прийнятний час відповіді при завантаженні та відображенні веб-сторінок. Час від отримання HTTP-відповіді до відображення базової структури HTML не повинен викликати помітних затримок для користувача (наприклад, для сторінок середнього розміру).

б) Ефективність використання ресурсів: Браузер повинен ощадливо використовувати системні ресурси, зокрема оперативну пам'ять та потужності центрального процесора. При обробці кількох запитів або "важких" сторінок споживання ресурсів не повинно призводити до збоїв у роботі операційної системи.

в) Надійність та стабільність: Система повинна бути стійкою до збоїв. Вона повинна коректно обробляти несподівані або пошкоджені відповіді від сервера, а також помилки парсингу HTML, не завершуючи роботу аварійно. Користувачу має відображатися чітке повідомлення про помилку (як це вказано у функціональних вимогах).

г) Зручність використання (Usability): Інтерфейс користувача повинен бути простим та інтуїтивно зрозумілим. Ключові елементи, такі як адресний рядок та навігаційні кнопки, мають бути легкодоступними. Процес інспектування ресурсів має бути зрозумілим і надавати чітку інформацію.

д) Гнучкість та розширюваність (Maintainability): Архітектура системи повинна бути модульною. Це необхідно для демонстрації патернів проектування. Система повинна дозволяти легке додавання нових типів обробників (наприклад, для нових HTTP-кодів) або нових операцій над HTML-вузлами без необхідності кардинальної зміни існуючого коду.

е) Сумісність (Compliance): Система повинна дотримуватися фундаментальних стандартів. Вона має коректно інтерпретувати базові специфікації протоколу HTTP (обробка запитів/відповідей, заголовків, кодів стану) та базові теги HTML для коректного розбору та відображення структури документа.

є) Портативність (Portability): Програмне забезпечення повинно бути розроблене таким чином, щоб його можна було запустити на різних операційних системах (наприклад, Windows, macOS, Linux) за умови наявності необхідного універсального середовища виконання, якщо таке використовується для розробки.

1.4. Сценарії використання системи

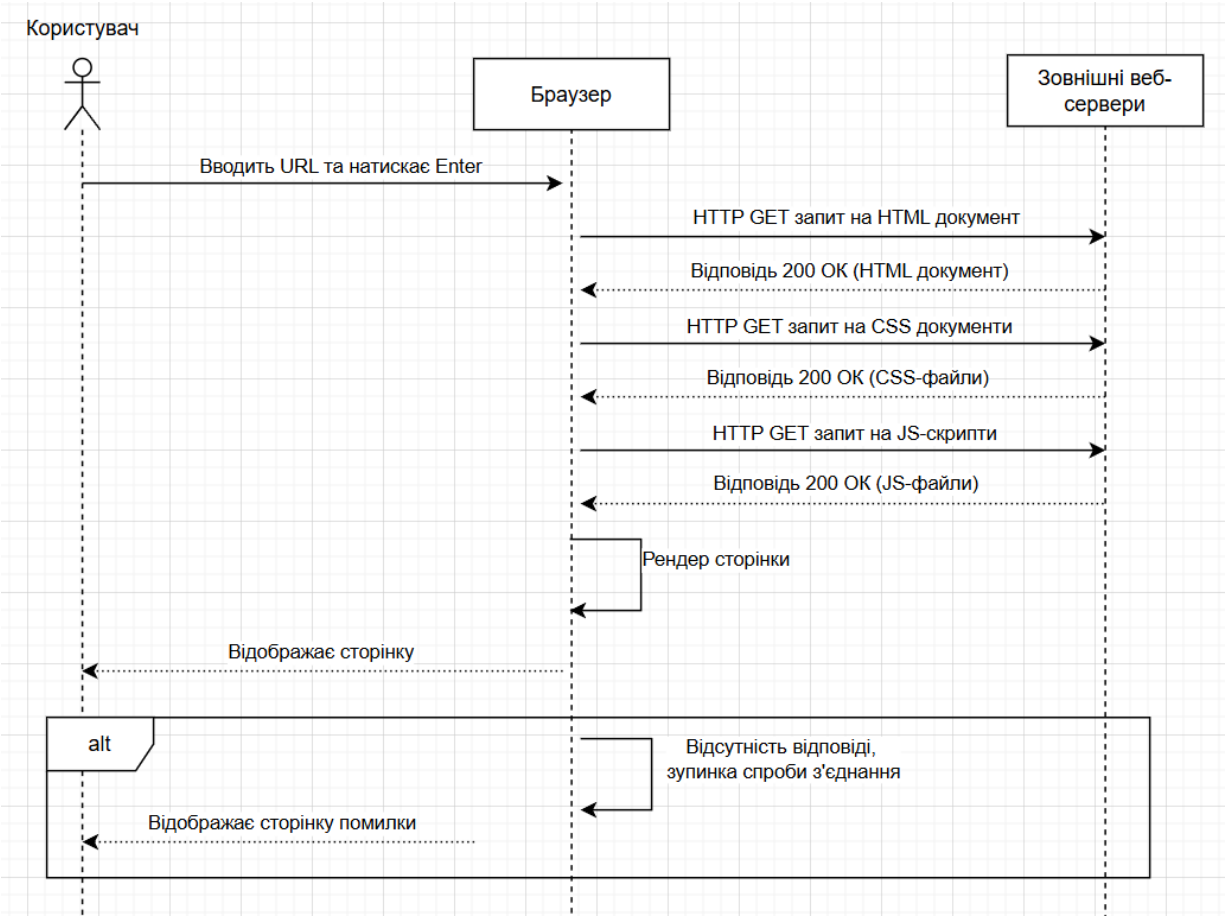

Рис. 1.4.1 – Діаграма послідовності

Прецедент: Завантаження та відображення веб-сторінки

Передумови:

1. Браузер запущений і готовий приймати команди від користувача.
2. Користувач має активне підключення до мережі.
3. Користувач ввів коректний URL в адресний рядок.

Постумови:

1. Запит успішно оброблено зовнішнім веб-сервером.
2. HTML-документ та всі пов'язані з ним ресурси (CSS, JS, зображення) отримані браузером.
3. Веб-сторінка коректно відображена (відрендерена) для користувача.

Взаємодіючі сторони:

1. **Користувач** — ініціює запит через адресний рядок.
2. **Браузер** — формує та відправляє HTTP-запити, отримує відповіді, парсить HTML/CSS та рендерить сторінку.
3. **Зовнішній веб-сервер** — отримує HTTP-запити та повертає ресурси (HTML, CSS, JS).

Короткий опис: Користувач вводить URL, браузер надсилає запит на отримання HTML-документа. Після його отримання, браузер парсить документ, знаходить посилання на додаткові ресурси (CSS, JavaScript) і послідовно завантажує їх, після чого рендерить фінальну сторінку.

Основний потік подій:

1. Користувач вводить URL в адресний рядок та натискає "Enter".
2. Браузер надсилає HTTP GET запит на вказаний URL для отримання HTML-документа.
3. Веб-сервер повертає відповідь 200 OK із тілом, що містить HTML-документ.
4. Браузер розбирає (парсить) HTML і знаходить посилання на зовнішні CSS та JS файли.
5. Браузер надсилає HTTP GET запити на отримання кожного CSS та JS-скрипта.
6. Веб-сервер повертає відповіді 200 OK із CSS/JS-файлами.

7. Браузер застосовує стилі (CSS) та виконує скрипти (JS) для побудови фінального вигляду сторінки.
8. Браузер відображає готову сторінку користувачу.

Прецедент: Обробка помилок завантаження сторінки

Передумови:

1. Браузер намагався надіслати HTTP-запит на отримання ресурсу.
2. Веб-сервер повернув код помилки (напр., 404 Not Found, 502 Bad Gateway, 503 Service Unavailable) *АБО* сервер не відповів (timeout).

Постумови:

1. Процес завантаження сторінки припинено.
2. Користувачу відображено спеціальну внутрішню сторінку браузера, що інформує про тип помилки.

Взаємодіючі сторони:

1. **Браузер** — ідентифікує код помилки у відповіді (або відсутність відповіді) та генерує сторінку помилки.
2. **Користувач** — отримує повідомлення про помилку.

Короткий опис: Під час запиту ресурсу браузер отримує від сервера код помилки (наприклад, 404), або не може встановити з'єднання. Браузер припиняє спроби завантаження та показує користувачу відповідну сторінку, що пояснює проблему.

Основний потік подій:

1. Браузер надсилає HTTP GET запит на ресурс.
2. Веб-сервер повертає відповідь зі статусом помилки (напр., 404 Not Found).
3. Браузер розпізнає код стану як помилку.

4. Браузер зупиняє подальшу обробку (напр., не намагається парсити тіло відповіді як HTML).
5. Браузер формує та відображає користувачу внутрішню сторінку (напр., "Сторінку не знайдено (404)").

Прецедент: Обробка перенаправлень (Redirect)

Передумови:

1. Браузер надіслав запит на отримання ресурсу за URL A.
2. Веб-сервер відповів кодом стану 3xx (напр., 301 Moved Permanently) та надав новий URL B у заголовку Location.

Постумови:

1. Браузер автоматично ініціював новий запит до URL B.
2. Вміст сторінки з URL B відображено користувачу.
3. Адресний рядок браузера оновлено на URL B.

Взаємодіючі сторони:

1. **Браузер** — обробляє відповідь 3xx, зчитує заголовок Location та ініціює новий запит.
2. **Веб-сервер** — ініціює перенаправлення.

Короткий опис: Браузер запитує сторінку, але сервер повідомляє, що вона переміщена. Браузер автоматично "слідує" за посиланням на нову адресу (прозоро для користувача) і завантажує сторінку звідти.

Основний потік подій:

1. Браузер надсилає HTTP GET запит на URL A.
2. Сервер повертає відповідь зі статусом 3xx.
3. Браузер розпізнає код перенаправлення.
4. Браузер зчитує новий URL (URL B) із заголовка Location у відповіді.
5. Браузер автоматично надсилає новий HTTP GET запит на URL B.

6. (Подальший потік продовжується як у "Прецеденті: Завантаження та відображення веб-сторінки" для URL B).

Прецедент: Інспектування ресурсів завантаженої сторінки

Передумови:

1. Веб-сторінка була успішно завантажена та відображена (згідно з першим прецедентом).
2. Користувач ініціював дію "Переглянути ресурси".

Постумови:

1. Користувачу надано список всіх CSS, JS та файлів зображень, які були завантажені для поточної сторінки.

Взаємодіючі сторони:

1. **Користувач** — ініціює запит на інспекцію.
2. **Браузер** — надає дані, зібрані під час завантаження сторінки.

Короткий опис: Користувач активує функцію інспектування. Браузер показує список всіх ресурсів, які були завантажені разом з HTML-документом.

Основний потік подій:

1. Користувач переглядає завантажену сторінку.
2. Користувач викликає функцію інспектування ресурсів.
3. Браузер звертається до своєї внутрішньої моделі завантаженої сторінки.
4. Браузер отримує зібраний раніше список URL-адрес всіх CSS-файлів.
5. Браузер отримує зібраний раніше список URL-адрес всіх JS-файлів та зображень.
6. Браузер відображає ці списки користувачу.

1.5. Концептуальна модель системи

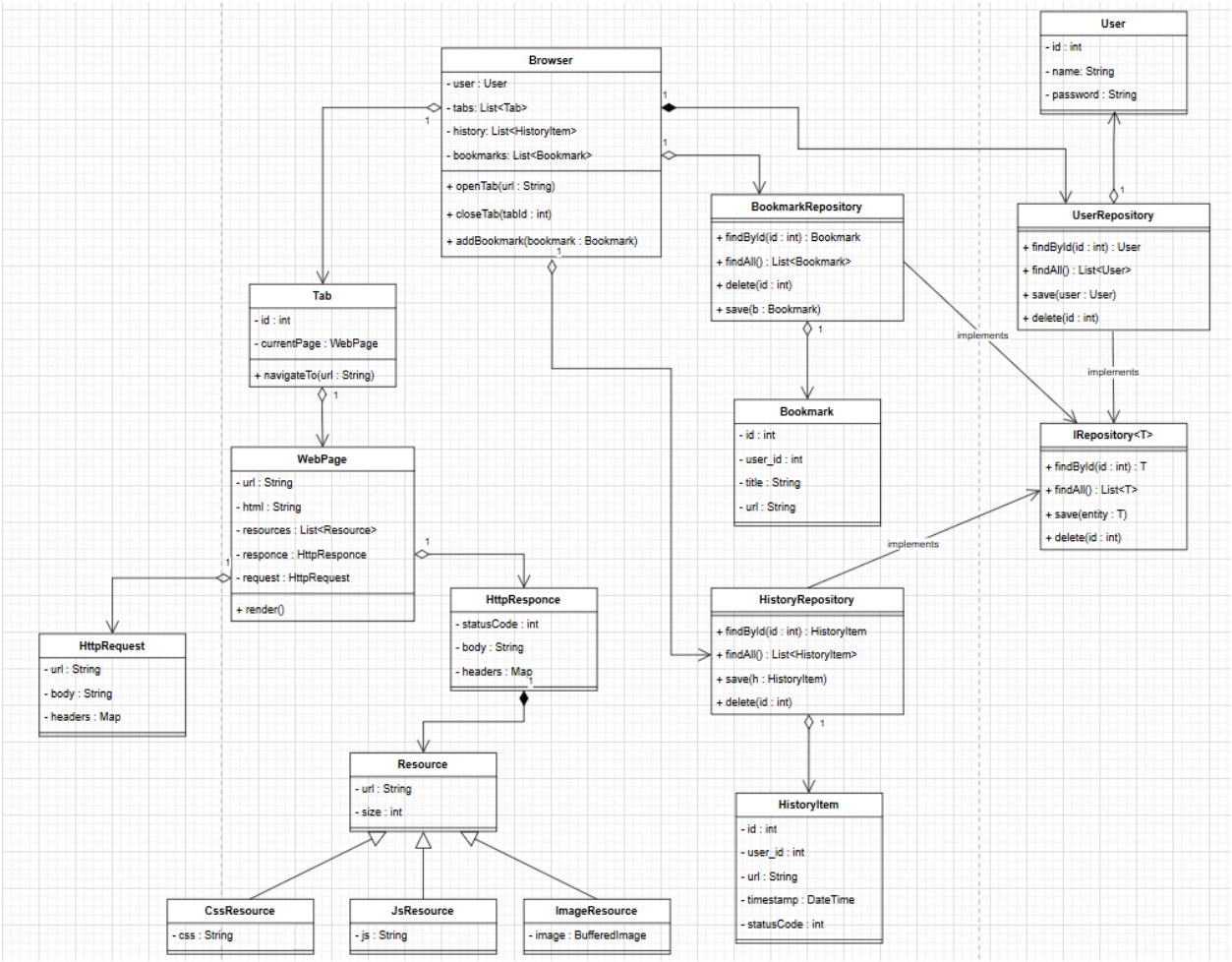

Рис. 1.5.1 – Діаграма класів для веб-браузера

На основі діаграми класів центральним елементом системи є клас Browser, який виконує роль координатора сесії користувача. Він агрегує об'єкт User, керує списком відкритих вкладок (об'єктів Tab) та надає доступ до сховищ даних, таких як BookmarkRepository та HistoryRepository. Клас Browser надає основні методи для взаємодії з користувачем, такі як openTab(url: String), closeTab(tabId: int) та addBookmark(bookmark: Bookmark).

Обробка навігації та завантаження веб-сторінок відбувається за участю кількох ключових класів. Клас Tab є основним обробником, який керує станом однієї вкладки та містить посилання на поточну сторінку (currentPage: WebPage). Метод MapsTo(url: String) ініціює процес завантаження. Він створює об’єкт WebPage, який представляє

завантажений ресурс. WebPage містить сам URL, отриманий HTML-вміст, а також об'єкти HttpRequest (дані запиту) та HttpResponse (дані відповіді, включаючи statusCode та body).

Клас WebPage також агрегує список підключених ресурсів, представлених базовим класом Resource. Ця структура використовує наслідування для обробки різних типів контенту: класи CssResource, JsResource та ImageResource розширюють Resource, додаючи специфічні поля для зберігання CSS-коду, JavaScript-коду або об'єктів зображень. Це дозволяє системі поліморфно обробляти та інспектувати всі завантажені компоненти сторінки.

Рівень управління даними (персистенції) реалізовано за допомогою патерну "Repository". Уніфікований інтерфейс IRepository<T> визначає базові CRUD-операції (findById, findAll, save, delete). Цей інтерфейс реалізується конкретними класами: UserRepository, BookmarkRepository та HistoryRepository. Ці репозиторії інкапсулюють логіку доступу до бази даних і керують збереженням, оновленням та вибіркою відповідних сутностей: User (дані користувача), Bookmark (збережені посилання) та HistoryItem (записи про відвідані сторінки з timestamp та statusCode).

Загалом діаграма класів демонструє чіткі зв’язки між компонентами, що забезпечують модульність, розширюваність і чітке розділення відповідальності (Separation of Concerns). Центральна роль Browser як фасаду для користувацьких дій, інкапсуляція логіки вкладки у Tab, представлення сторінки у WebPage та винесення логіки доступу до даних у репозиторії дозволяють легко підтримувати систему та тестувати окремі компоненти незалежно.

1.6. Вибір бази даних

Для зберігання та обробки даних користувачів у рамках серверної частини цього проєкту було обрано систему управління базами даних **MySQL**. Це рішення обґрунтоване кількома важливими факторами, серед яких доведена надійність,

висока продуктивність при одночасних підключеннях та широкі можливості для масштабування.

Причини вибору MySQL:

а) Висока продуктивність: MySQL є однією з найпопулярніших у світі реляційних баз даних, оптимізованих для веб-навантажень. Вона дозволяє ефективно працювати з великою кількістю одночасних підключень, які виникатимуть при синхронізації даних (історії, закладок) від багатьох клієнтів-браузерів. Завдяки використанню рушія InnoDB, механізмам індексації та оптимізації запитів, MySQL забезпечує високу швидкість обробки транзакцій.

б) Масштабованість: MySQL має чудову підтримку як вертикальної, так і горизонтальної масштабованості. Вона підтримує різноманітні конфігурації реплікації (Master-Slave, Master-Master), що дозволяє легко розширювати інфраструктуру читання даних. Для більш складних завдань існують рішення, такі як MySQL Cluster, що є важливим аспектом для майбутнього зростання проєкту, коли кількість користувачів та обсяг їхніх даних буде збільшуватися.

в) Реляційна модель: MySQL є класичною реляційною СУБД, що дозволяє зберігати дані у вигляді чітко структурованих таблиць. Це дозволяє зручно моделювати дані для зберігання інформації про акаунти користувачів, їхні закладки, записи історії та зв'язки між ними. Використання стандартизованої мови SQL дає змогу створювати надійні запити для синхронізації даних.

г) Підтримка транзакцій: Рушій зберігання InnoDB (стандартний для MySQL) повністю підтримує транзакції **ACID** (Atomicity, Consistency, Isolation, Durability). Це гарантує цілісність і надійність даних, навіть якщо процес синхронізації з боку клієнта-браузера буде перервано. Це критично важливо для забезпечення того, що дані користувача не будуть пошкоджені або частково збережені.

д) Підтримка поширених типів даних: MySQL підтримує всі стандартні типи даних SQL, необхідні для проєкту, а також, починаючи з версії 5.7, нативно підтримує тип даних **JSON**. Це може бути корисним для гнучкого зберігання неструктурованих

даних, наприклад, налаштувань браузера користувача або метаданих сторінок, без необхідності зміни схеми бази даних.

е) Архітектура рушіїв зберігання (Storage Engines): Унікальною особливістю MySQL є підтримка змінних рушіїв зберігання. Хоча InnoDB є найкращим вибором для транзакційного навантаження, можливість вибору (наприклад, MyISAM для операцій, що вимагають переважно швидкого читання) надає додаткову гнучкість при оптимізації роботи серверної частини.

є) Відкритий код і спільнота: MySQL є однією з найстаріших і найпопулярніших систем з відкритим вихідним кодом (GPL), що дозволяє використовувати її без ліцензійних відрахувань. Крім того, MySQL має величезну та активну світову спільноту, що забезпечує безперервний розвиток, підтримку та наявність численних форків (як-от MariaDB та Percona Server).

ж) Безпека: MySQL надає широкі та перевірені часом можливості для забезпечення безпеки даних. Це включає надійну аутентифікацію користувачів, гнучку систему привілеїв (на рівні баз, таблиць, стовпців), а також вбудовану підтримку SSL/TLS для шифрування з'єднань. Це є абсолютно необхідним для захисту даних користувачів, що передаються від браузера до сервера.

1.7. Вибір мови програмування та середовища розробки

Для реалізації цього проєкту було обрано мову програмування **Java**, а також інтегроване середовище розробки (IDE) **IntelliJ IDEA**. Це рішення обґрунтоване низкою факторів, що забезпечують ефективну розробку, високу надійність клієнтського додатку та зручність під час роботи над проєктом.

Причини вибору мови програмування Java:

а) Надійність та стабільність: Java є однією з найбільш надійних та перевірених часом мов програмування, що широко використовується для розробки складних настільних (desktop) та корпоративних додатків. Завдяки строгому

контролю типів, механізму обробки винятків та багаторічному розвитку, Java забезпечує високу стабільність, що є критичним аспектом для створення веб-браузера, який повинен надійно працювати протягом тривалих сесій.

б) Вбудована підтримка багатопоточності: Однією з ключових переваг Java є її потужна модель багатопоточності. Це є фундаментально важливим для реалізації браузера, оскільки дозволяє ефективно розділяти задачі:

- Основний потік для обробки інтерфейсу користувача (UI) залишається відгукливим.
- Фонові потоки (thread pools) використовуються для виконання мережевих операцій (завантаження HTML, CSS, зображень), парсингу документів та інших "важких" завдань, не блокуючи при цьому користувача.

в) Продуктивність та управління пам'яттю: Java демонструє високу продуктивність завдяки JIT-компіляції (Just-In-Time) коду у нативний. Також, автоматичне управління пам'яттю (Garbage Collection) значно спрощує розробку, автоматично звільняючи ресурси, що особливо важливо для програми, яка активно завантажує та обробляє великі обсяги даних (веб-сторінки, ресурси).

г) Платформонезалежність ("Write Once, Run Anywhere"): Завдяки використанню віртуальної машини Java (JVM), програма, написана на Java, може виконуватися на будь-якій операційній системі (Windows, macOS, Linux) без зміни коду. Це ідеально підходить для розробки настільного додатка, яким є браузер, дозволяючи охопити різні платформи.

д) Багата стандартна бібліотека (Ecosystem): Java має великі стандартні бібліотеки для роботи з мережею (сокети, HTTP-з'єднання), введенням-виведенням (I/O) та графічними інтерфейсами (Swing, JavaFX). Крім того, існує величезна екосистема сторонніх бібліотек, наприклад, для парсингу HTML (як Jsoup), що суттєво прискорює розробку.

Причини вибору середовища розробки IntelliJ IDEA:

а) Потужний інструмент для розробки на Java: IntelliJ IDEA вважається одним із найрозумніших та найефективніших IDE для Java. Вона надає глибокий аналіз коду, що є неоціненним при реалізації складних архітектурних рішень та патернів проектування (Proxy, Visitor, Factory Method тощо), які є вимогою курсової роботи.

б) Автоматизація процесу розробки: IntelliJ IDEA пропонує найкращі у своєму класі можливості для рефакторингу коду, інтелектуальне автодоповнення (code completion) та інтеграцію з системами контролю версій (наприклад, Git). Це дозволяє розробнику фокусуватися на логіці, а не на рутинних задачах, що значно прискорює розробку.

в) Налагодження та профілювання: IDE надає виняткові інструменти для налагодження (debugger). Це особливо важливо для відстеження помилок у багатопотокових додатках, дозволяючи аналізувати стан UI-потоку та фонових мережевих потоків одночасно. Інструменти профілювання допомагають знаходити "вузькі місця" у продуктивності, наприклад, при парсингу великих HTML-документів.

г) Підтримка сучасних інструментів: IntelliJ IDEA підтримує інтеграцію з системами збірки, такими як Maven та Gradle, що є стандартом для управління залежностями у Java-проєктах. Вбудовані інструменти для роботи з базами даних дозволяють ефективно взаємодіяти як з локальними БД (SQLite для історії), так і з серверними (MySQL для синхронізації).

д) Зручність користувацького інтерфейсу: Інтерфейс IntelliJ IDEA є гнучким, інтуїтивно зрозумілим та мінімалістичним, що дозволяє розробнику не відволікатися та швидко навігувати по проєкту, шукати файли та аналізувати структуру коду.

е) Широка спільнота та підтримка: IntelliJ IDEA (та її безкоштовна версія Community Edition) має величезну спільноту розробників та чудову документацію. Це дає можливість швидко знаходити рішення для будь-яких проблем, що виникають під час розробки.

1.8. Проєктування розгортання системи

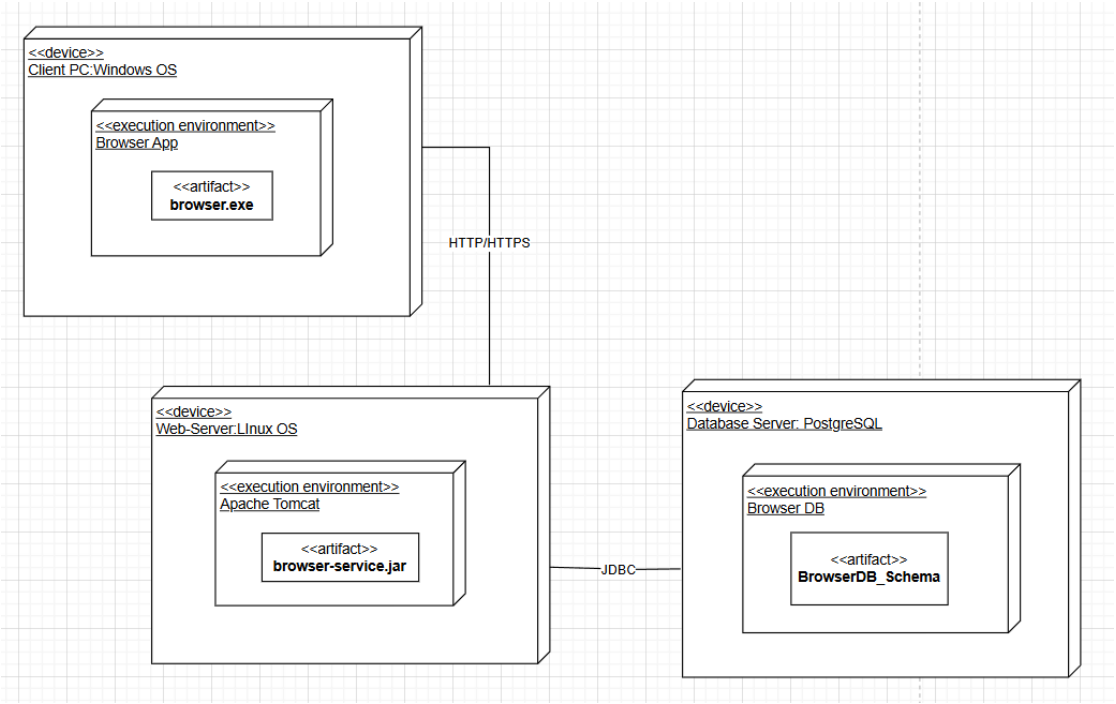

Рис. 1.8.1 – Діаграма розгортання веб-браузера

Проєктування розгортання системи базується на дворівневій архітектурі автономного настільного застосунку (Standalone Desktop Application). Така модель виключає необхідність використання проміжного сервера застосунків, зосереджуючи всю бізнес-логіку та збереження даних безпосередньо на пристрої користувача.

Основним вузлом розгортання є Client PC (персональний комп'ютер користувача), на якому встановлено середовище виконання Java (JRE/JDK). Головним артефактом системи є виконуваний файл Explorer-1.0.jar. Цей додаток виконує дві ключові функції:

1. Роль веб-клієнта: Додаток генерує та відправляє HTTP/HTTPS запити до Third-Party Servers (сторонніх веб-серверів) для отримання веб-ресурсів (HTML-документів, стилів CSS, скриптів JS) та їх візуалізації користувачеві.
2. Роль менеджера даних: Додаток самостійно керує збереженням історії переглядів та налаштувань.

Критично важливим компонентом архітектури є підсистема збереження даних, реалізована на основі вбудованої реляційної СУБД SQLite. Вся інформація (історія відвідувань, метадані сторінок) зберігається у локальному файлі history.db, що розташовується в кореневій директорії програми.

Взаємодія між додатком (Explorer-1.0.jar) та базою даних відбувається через драйвер JDBC:SQLite. Таке рішення забезпечує ряд архітектурних переваг:

- Висока продуктивність: Відсутність мережевих затримок при зверненні до бази даних, оскільки запити виконуються локально.
- Автономність: Браузер зберігає повну функціональність (перегляд історії) навіть за відсутності інтернет-з'єднання.
- Простота розгортання: Відсутність необхідності встановлювати та адмініструвати окремий сервер баз даних (як PostgreSQL або MySQL), оскільки рушій SQLite вбудовано в бібліотеки додатку.

Таким чином, запропонована схема розгортання представляє собою монолітний клієнтський додаток, що взаємодіє із зовнішнім світом виключно для завантаження контенту, а всі персональні дані користувача надійно інкапсулює у локальному файловому сховищі.

2 РЕАЛІЗАЦІЯ КОМПОНЕНТІВ СИСТЕМИ

2.1. Структура бази даних

Рис. 2.1.1 – Структура бази даних

Опис структури бази даних

Система використовує вбудовану реляційну базу даних SQLite, яка зберігається у локальному файлі history.db в кореневій директорії програми. Такий підхід забезпечує портативність додатку, оскільки не вимагає встановлення та налаштування окремого сервера баз даних.

Таблиця History

Таблиця призначена для зберігання інформації про відвідані веб-ресурси. Вона містить записи про URL-адреси, заголовки сторінок та час їх відвідування.

Поля таблиці:

- **id (INTEGER):** Унікальний ідентифікатор запису (первинний ключ — Primary Key). Поле має властивість AutoIncrement, що забезпечує автоматичне створення унікального номера для кожного нового запису в історії.
- **url (TEXT):** Повна адреса відвіданої веб-сторінки. Це поле має обмеження NotNull, що гарантує наявність посилання в кожному записі (запис без URL неможливий).
- **title (TEXT):** Заголовок сторінки (вміст тегу <title>). Зберігається у текстовому форматі та може бути порожнім (NULL), якщо заголовок не вдалося отримати.
- **visit_time (TEXT):** Час відвідування сторінки. На відміну від складних систем, де використовується тип TIMESTAMP, тут дата та час зберігаються у текстовому форматі (наприклад, ISO 8601 або інший рядковий формат дати).

Загальний опис

Запропонована структура реалізує просту модель зберігання даних. Відсутність зв'язків з іншими таблицями (такими як Users або WebPages) свідчить про те, що база даних призначена для локального використання (наприклад, історія браузера на одному пристрої) або для ситуацій, де не потрібна авторизація та кешування контенту. Тип даних TEXT використовується для більшості полів, що забезпечує гнучкість при збереженні рядків довільної довжини.

2.2. Архітектура системи

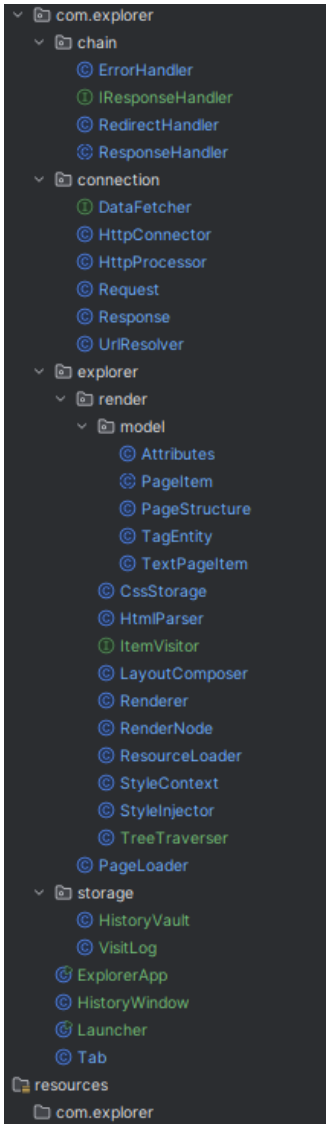

Рис. 2.2.1 – Структура клієнтської сторони

Основні компоненти архітектури програмного забезпечення

Архітектура додатку побудована за модульним принципом, де кожен компонент відповідає за окремий етап обробки даних: від завантаження веб-сторінки до її візуалізації та збереження в історії.

а) Модуль ядра браузера та рендерингу (Відповідає за перетворення HTML/CSS коду у графічний інтерфейс користувача):

1. BrowserApp: Головний клас та точка входу в застосунок. Ініціалізує графічний інтерфейс JavaFX, налаштовує основне вікно та запускає життєвий цикл програми.
2. BrowserPageLoader: Клас-оркестратор, який керує процесом завантаження. Він отримує URL, викликає мережевий модуль для завантаження контенту, передає його парсерам і ініціює процес рендерингу.
3. HtmlParser: Відповідає за лексичний та синтаксичний аналіз отриманого HTML-коду. Результатом його роботи є побудова DOM-дерева (Document Object Model) у пам'яті програми.
4. CssParser: Аналізує знайдені стилі (як вбудовані, так і зовнішні) та формує набір правил стилізації для елементів.
5. StyleCalculator: Обчислює фінальні візуальні параметри для кожного вузла DOM-дерева, застосовуючи правила CSS до відповідних тегів.
6. RenderTreeBuilder: Відповідає за побудову "дерева відображення" (Render Tree). Він поєднує логічну структуру DOM з обчисленими графічними стилями, готуючи дані для малювання.
7. FxRenderer: Компонент, що відповідає за безпосереднє малювання елементів на екрані засобами бібліотеки JavaFX (Canvas або Scene Graph), візуалізуючи кнопки, текст, блоки та зображення.

б) Модуль мережевої взаємодії (Забезпечує виконання HTTP-запитів до веб-серверів):

1. HttpClient: Клієнтський клас для створення з'єднання з веб-сервером, надсилання GET-запитів та отримання «сирого» тексту сторінки (HTML).
2. UrlResolver: Утилітний клас для обробки URL-адрес. Він допомагає перетворювати відносні посилання (наприклад, /style.css) в абсолютні, коректно обробляє протоколи (http/https).
3. ErrorHandler: Перехоплює мережеві помилки (наприклад, 404 Not Found або відсутність інтернету) та формує зрозумілі для користувача повідомлення або сторінки-заглушки.

в) Модуль роботи з даними та історією (Забезпечує взаємодію з локальною базою даних SQLite):

1. HistoryService: Сервісний клас, що інкапсулює бізнес-логіку роботи з історією. Він відповідає за додавання нових записів при відвідуванні сторінки та отримання списку відвіданих сайтів для відображення в інтерфейсі.
2. HistoryRepository: Реалізація патерну Repository (DAO). Цей клас містить SQL-запити та відповідає за безпосереднє з'єднання з файлом бази даних history.db (INSERT, SELECT).
3. History (Entity): Клас-сутність, що відображає структуру таблиці бази даних у Java-коді. Містить поля id, url, title та visit_time.
4. SQLiteConnectionFactory: (або аналогічний клас конфігурації) Відповідає за встановлення з'єднання з локальним драйвером JDBC для SQLite

2.2.1. Вибір та обґрунтування патернів реалізації

Обробка HTTP відповідей

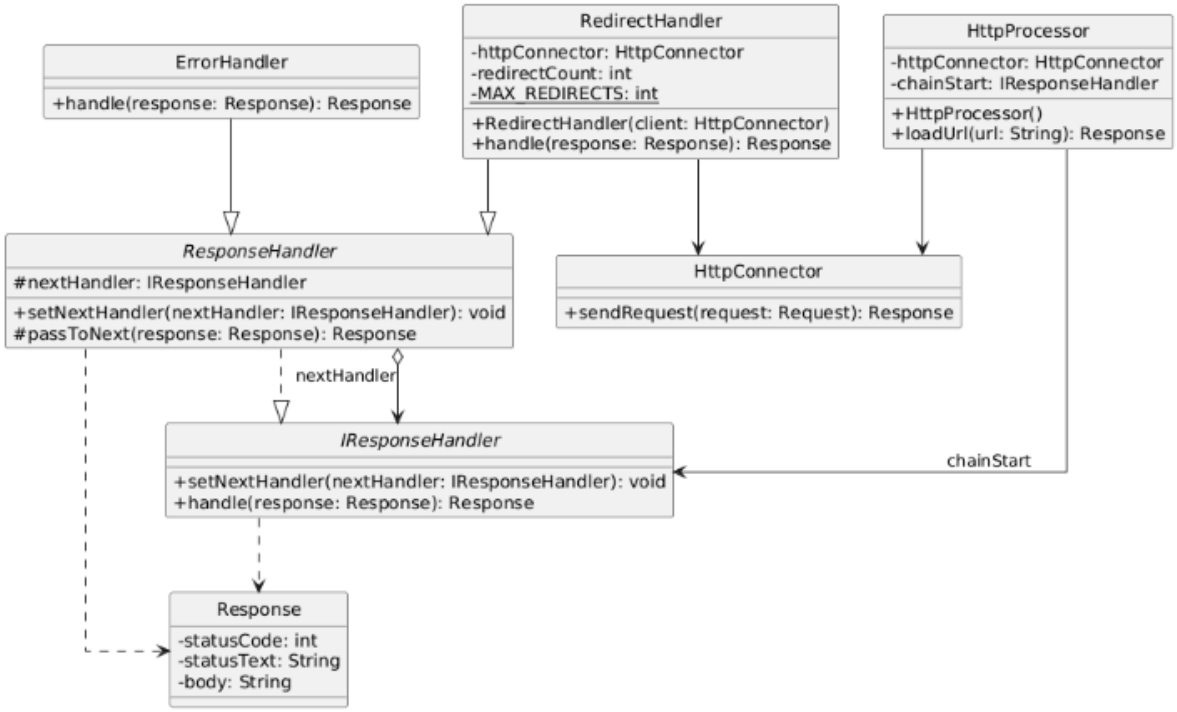

Рис. 2.2.2.1 – Діаграма класів патерну Chain of Responsibility

Проблема: Необхідність обробки різноманітних типів HTTP-відповідей (помилок, перенаправлень, успішних статусів) без створення громіздких умовних конструкцій та жорстких залежностей у клієнтському коді.

Рішення: Використання патерну Ланцюжок обов'язків (Рис. 2.2.2.1) дозволяє передавати отриману відповідь послідовно через набір незалежних обробників (RedirectHandler, ErrorHandler). Це забезпечує послаблення зв’язності компонентів, оскільки об’єкт-відправник (HttpProcessor) не знає, який саме обробник виконає

операцію, а додавання нових типів обробки здійснюється шляхом розширення ланцюжка без зміни базової логіки[7].

Застосування стилів та завантаження ресурсів із тегів

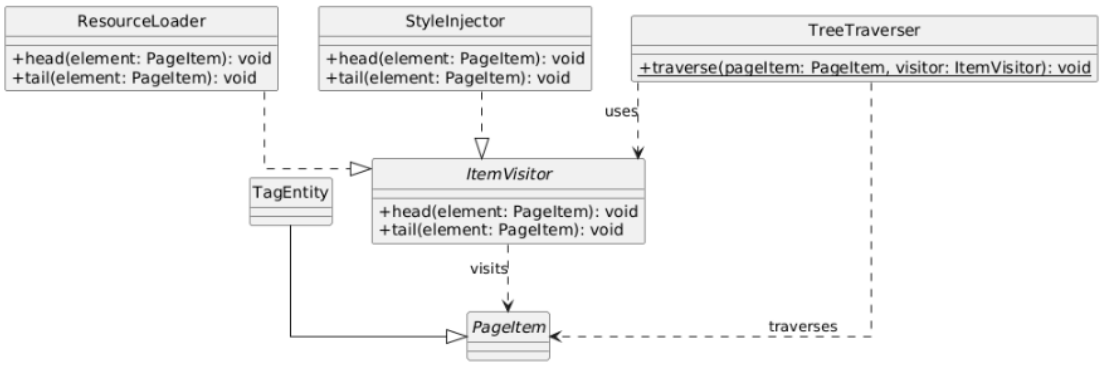

Рис. 2.2.2.2 – Діаграма класів патерну Visitor

Проблема: Класи DOM-елементів не повинні містити складної логіки обробки, такої як рендеринг чи мережеві запити, оскільки це ускладнює підтримку коду та порушує SOLID-принципи. Необхідно було знайти спосіб виконувати зовнішні операції над вузлами дерева без модифікації самих вузлів.

Рішення: Архітектуру побудовано з використанням патерну Visitor. Це дозволило винести всю бізнес-логіку обробки вузлів у зовнішні сервісні класи: ResourceFetcherVisitor відповідає за підтягування зовнішніх файлів, а CssApplierVisitor — за обчислення стилів. Використання допоміжного класу DomTraverser для навігації по дереву дозволяє легко розширювати функціонал браузера, додаючи нові операції без ризику зламати базову структуру DOM [8].

Процес завантаження веб-сторінки

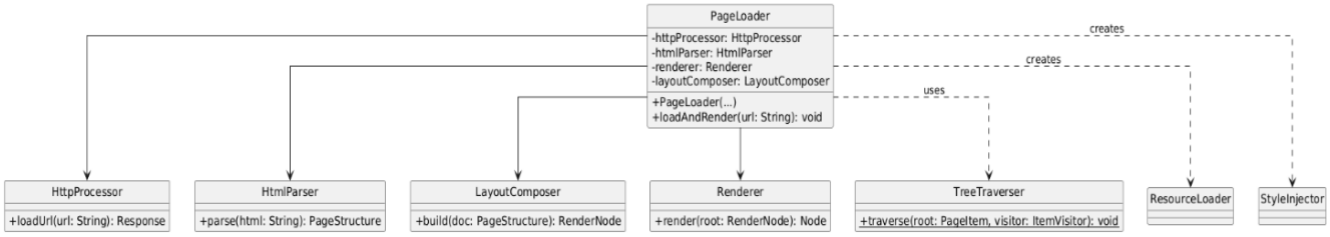

Рис. 2.2.2.3 – Діаграма класів патерну Facade

Проблема: Висока складність організації взаємодії між різнорідними компонентами системи (мережевим процесором HttpProcessor, парсером HtmlParser, механізмом візуалізації FxRenderer та резолвером посилань BaseUrlResolver) для виконання однієї бізнес-операції — завантаження веб-сторінки. Пряме звернення

клієнтського коду до кожного з цих модулів призводить до надмірної зв’язності (high coupling) та ускладнює підтримку коду.

Рішення: Використання патерну Фасад (Рис. 2.2.2.3) реалізовано через клас BrowserPageLoader, який надає спрощений інтерфейс до складної підсистеми браузерного рушія. Цей клас інкапсулює логіку ініціалізації та координації викликів внутрішніх компонентів, приховуючи від зовнішнього клієнта (наприклад, вкладки браузера) деталі реалізації процесів отримання даних, їх обробки та візуалізації[11].

Патерн Компонувальник (Composite)

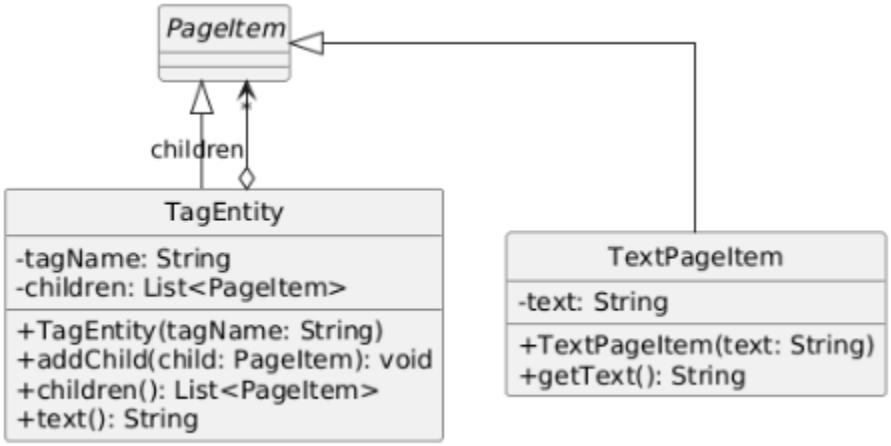

Рис. 2.2.2.4 – Діаграма класів патерну Composite

Проблема: Необхідність ефективного моделювання та обробки ієрархічної деревоподібної структури веб-документа (DOM), де елементи варіюються від простих (текст, зображення) до складних контейнерів (div, body), що містять вкладені компоненти. Критично важливим є забезпечення уніфікованого механізму керування цими об'єктами, щоб клієнтський код міг взаємодіяти як з окремими елементами, так і з їх групами однаково, уникаючи складної логіки перевірки типів при обході дерева.

Рішення: Використання патерну Компонувальник (Рис. 2.2.2.4) дозволяє згрупувати об'єкти у деревоподібні структури для представлення ієрархії "частина-ціле". Спільний абстрактний клас (Node) визначає єдиний інтерфейс для всіх компонентів. Складні вузли-контейнери (Element) зберігають посилання на дочірні елементи та делегують їм виконання операцій (наприклад, метод render() рекурсивно викликається для всіх нащадків), тоді як прості листові вузли (TextNode) виконують роботу безпосередньо. Такий підхід значно спрощує архітектуру рендерингу та дозволяє будувати сторінки будь-якої вкладеності[10].

2.3. Інструкція користувача

Даний посібник описує взаємодію користувача з клієнтською частиною веб-браузера та його сервсними модулями. Інструкція охоплює всі етапи роботи: від запуску клієнтського застосунку та автентифікації на REST-сервері до перегляду відрендерених HTML-сторінок та аналізу історії відвідувань.

2.3.1. Вимоги до програмного середовища

Для коректної роботи програмного продукту на комп'ютері користувача повинна бути встановлена віртуальна машина Java (JVM). Оскільки графічний інтерфейс базується на бібліотеці JavaFX, критично важливим є використання пакету JDK з включеним модулем JavaFX (наприклад, BellSoft Liberica JDK Full).

1. Завантажити пакет Liberica JDK 21 (Full version) для відповідної операційної системи (macOS/Windows/Linux) з офіційного сайту.
2. Виконати встановлення згідно зі стандартними інструкціями ОС.
3. Перевірити коректність встановлення, відкривши термінал та ввівши команду: java -version

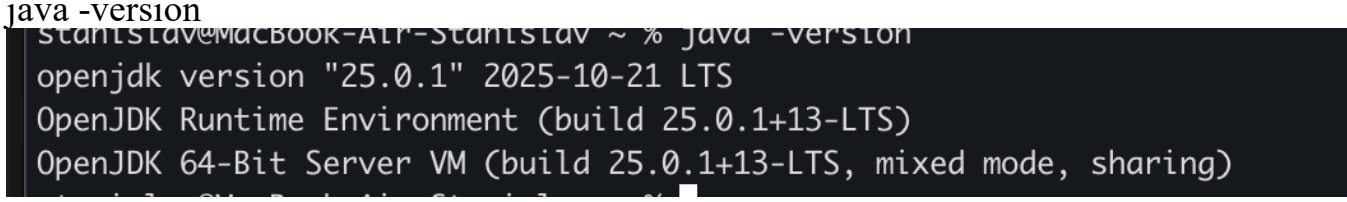

Рис. 2.3.1 – Перевірка версії Java у терміналі

2.3.2. Запуск програми

Процес запуску програми максимально спрощений для кінцевого користувача і складається з наступних кроків:

1. Підготовка файлів: Розархівувати наданий архів з програмою у зручну директорію на жорсткому диску. Переконатися, що у папці присутній виконуваний файл Explorer-1.0.jar.
2. Запуск через термінал (для macOS/Linux): Відкрити програму «Термінал», перейти у директорію з програмою та виконати команду запуску: java -jar Explorer-1.0.jar

(Якщо використовується повний шлях до JDK, команда може виглядати так: /path/to/java -jar Explorer-1.0.jar)

3. Автоматична ініціалізація бази даних**:** При першому запуску програма автоматично перевіряє наявність файлу бази даних. Якщо файл history.db відсутній, система створить його самостійно та згенерує необхідну структуру таблиці history засобами драйвера SQLite JDBC. Ніяких ручних SQL-скриптів виконувати не потрібно.
4. Початок роботи: Після успішного запуску відкриється головне вікно браузера. Користувач може ввести URL-адресу в навігаційний рядок та натиснути кнопку переходу.

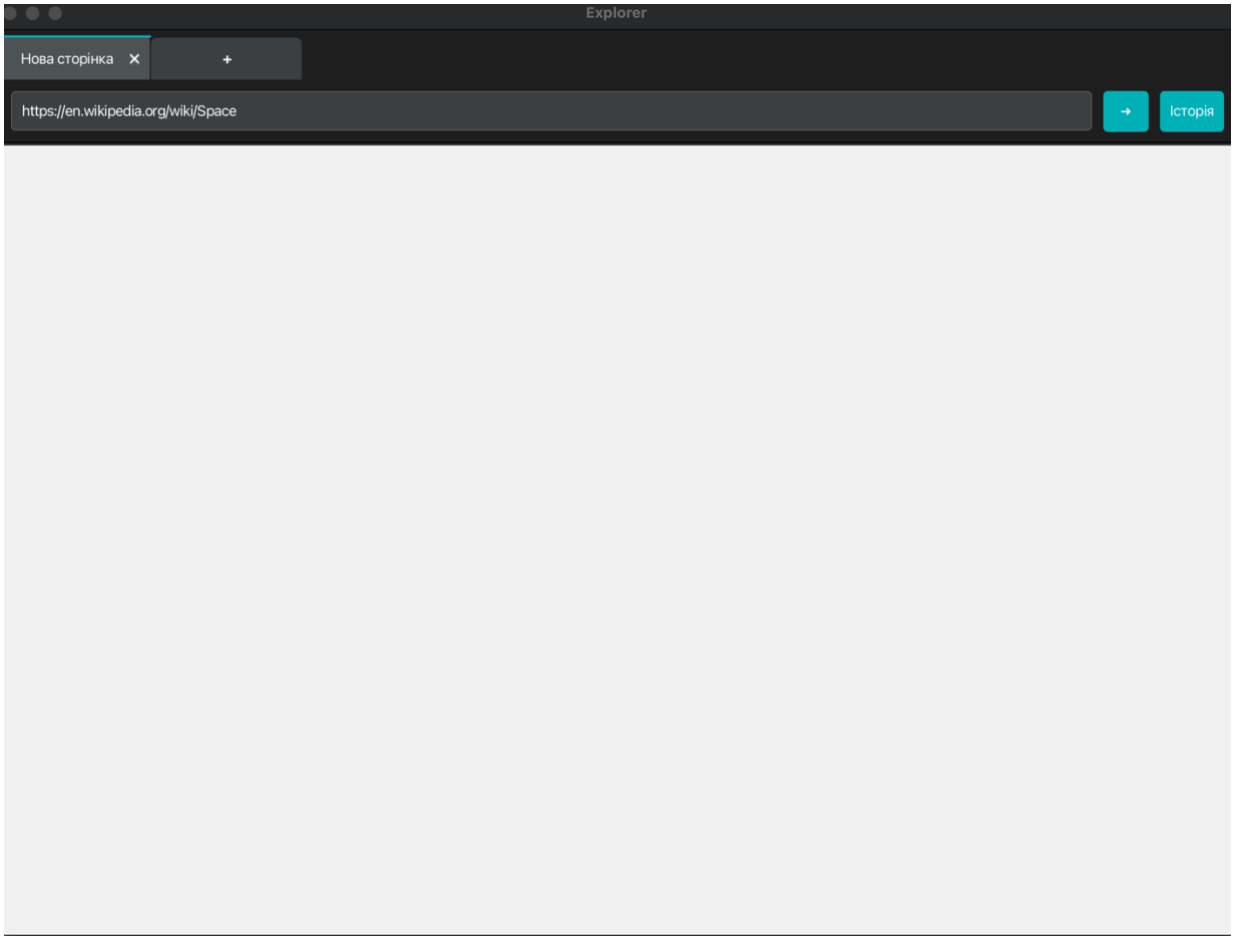

Рис. 2.3.2 – Головне вікно програми після запуску

ВИСНОВКИ

У процесі виконання курсової роботи було спроєктовано та програмно реалізовано власний веб-браузер з графічним інтерфейсом користувача. Розроблена система забезпечує повний цикл обробки веб-контенту: від ініціювання HTTP-запиту та отримання даних до синтаксичного аналізу HTML-коду і його візуалізації на екрані. Проєкт побудовано з дотриманням принципів об'єктно-орієнтованого програмування та використанням сучасних патернів проєктування, що гарантує гнучкість архітектури та зручність її супроводу.

Основними теоретичними та практичними результатами роботи є:

а) Реалізація власного механізму рендерингу на базі JavaFX. Розроблено алгоритми побудови DOM-дерева та базового застосування стилів, що дозволило коректно відображати структуру веб-сторінок. Використання патерну Visitor дозволило відокремити операції обробки вузлів (такі як обчислення стилів або завантаження ресурсів) від класів самих елементів DOM, дотримуючись принципу єдиної відповідальності.

б) Організація локального зберігання даних (Persistence Layer). Замість громіздких клієнт-серверних рішень було впроваджено легковагову вбудовану СУБД SQLite. Це забезпечило портативність додатку (вся історія зберігається в одному файлі history.db) та високу швидкість роботи. Для взаємодії з базою даних застосовано патерн DAO (Data Access Object), який абстрагує бізнес-логіку від прямих SQL-запитів.

в) Побудова гнучкої архітектури завантаження сторінок. Впровадження патерну Template Method (Шаблонний метод) дозволило стандартизувати алгоритм завантаження сторінки (парсинг -> стилізація -> рендеринг), гарантуючи правильну послідовність етапів, при цьому залишаючи можливість перевизначення реалізації окремих кроків у майбутньому.

г) Забезпечення модульності системи. Архітектуру додатку чітко розділено на логічні шари: модуль мережевої взаємодії (HttpClient), модуль парсингу (HtmlParser, CssParser) та модуль візуалізації (FxRenderer). Така слабка зв'язність компонентів (Loose Coupling) значно спрощує тестування та модифікацію окремих частин браузера без впливу на систему в цілому.

Система реалізована мовою програмування Java з використанням середовища IntelliJ IDEA. Вибір Java дозволив ефективно використати сувору типізацію для побудови надійних структур даних та вбудовані засоби для роботи з мережею.

Підсумовуючи, можна стверджувати, що мета курсової роботи досягнута. Створений програмний продукт є функціональним веб-оглядачем, що демонструє

роботу ключових принципів побудови браузерних рушіїв. Перспективи розвитку проєкту включають реалізацію підтримки JavaScript, розширення можливостей CSS (підтримка Flexbox) та вдосконалення системи кешування контенту.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Веб-браузер - це. [Електронний Ресурс]. За посиланням: https://uk.wikipedia.org/wiki/Браузер
2. Що таке HTML? [Електронний Ресурс]. За посиланням: https://www.hostinger.com/tutorials/what-is-html
3. Що таке JavaScript? [Електронний Ресурс]. За посиланням: https://uk.wikipedia.org/wiki/JavaScript
4. Що таке CSS? [Електронний Ресурс]. За посиланням: https://css.in.ua/article/shcho-take-html_10
5. Протоколи HTTP і HTTPS: що це таке, як вони працюють і чим відрізняються? [Електронний Ресурс]. За посиланням: https://besthosting.ua/ua/http_and_https_protocols.php
6. Що таке Google Chrome? [Електронний Ресурс]. За посиланням: https://uk.wikipedia.org/wiki/Google_Chrome
7. Що таке Chain Of Responsibility? [Електронний Ресурс]. За посиланням: https://uk.wikipedia.org/wiki/Ланцюжок_відповідальностей
8. Що таке Visitor? [Електронний Ресурс]. За посиланням: https://refactoring.guru/uk/design-patterns/visitor
9. Що таке Template Method? [Електронний Ресурс]. За посиланням: https://uk.wikipedia.org/wiki/Фабричний_метод_(шаблон_проєктування)
10. Що таке Facade? [Електронний Ресурс]. За посиланням: https://refactoring.guru/uk/design-patterns/facade
11. Що таке Composite? [Електронний Ресурс]. За посиланням: https://refactoring.guru/uk/design-patterns/composite

ДОДАТКИ

Додаток А

Посилання на репозиторій проєкту в GitHub

GitHub: https://github.com/brxhh/TRPZ/tree/main/final%20coursework

Додаток Б

Інтерфейс користувача

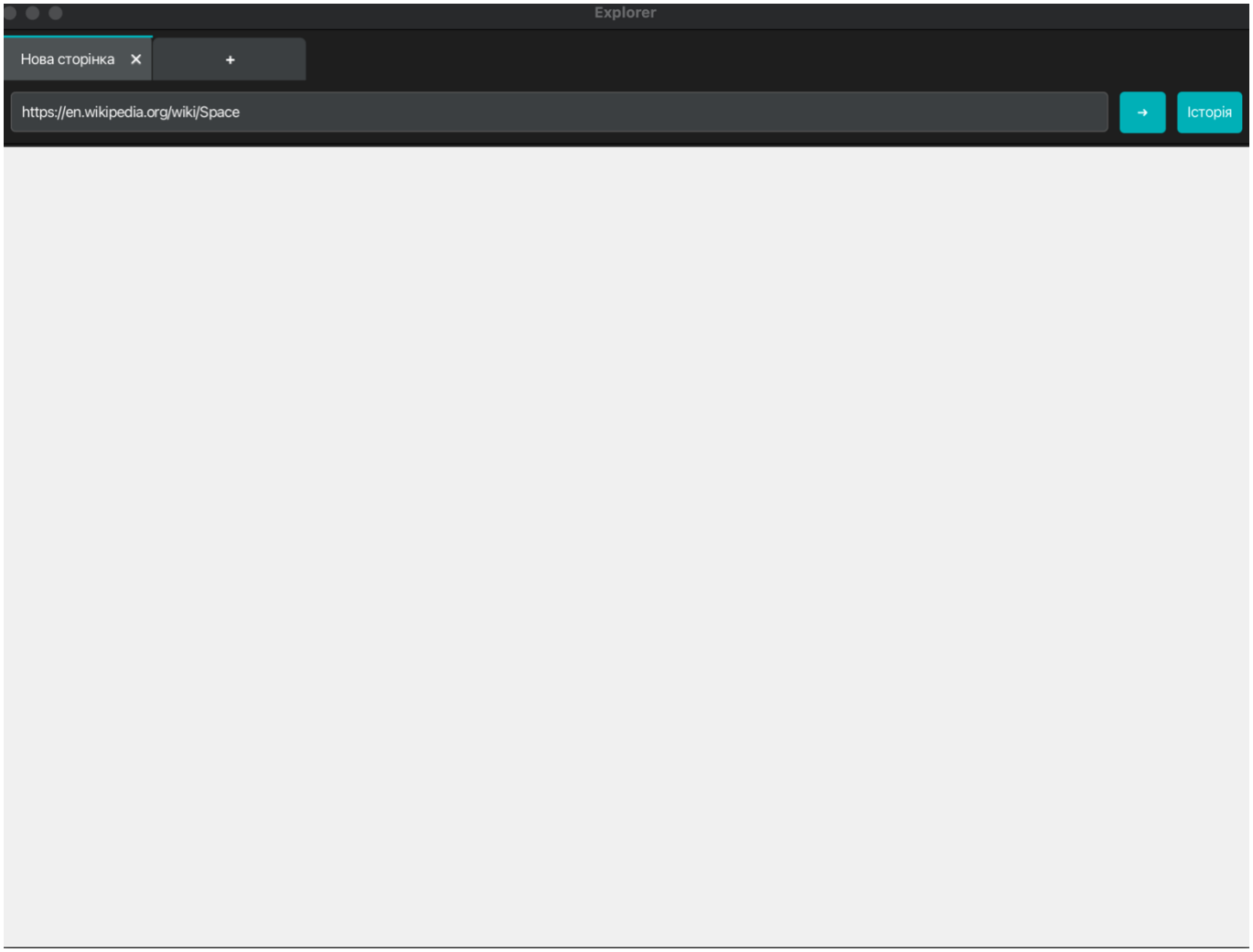

Рис. 3.1 – Головне вікно

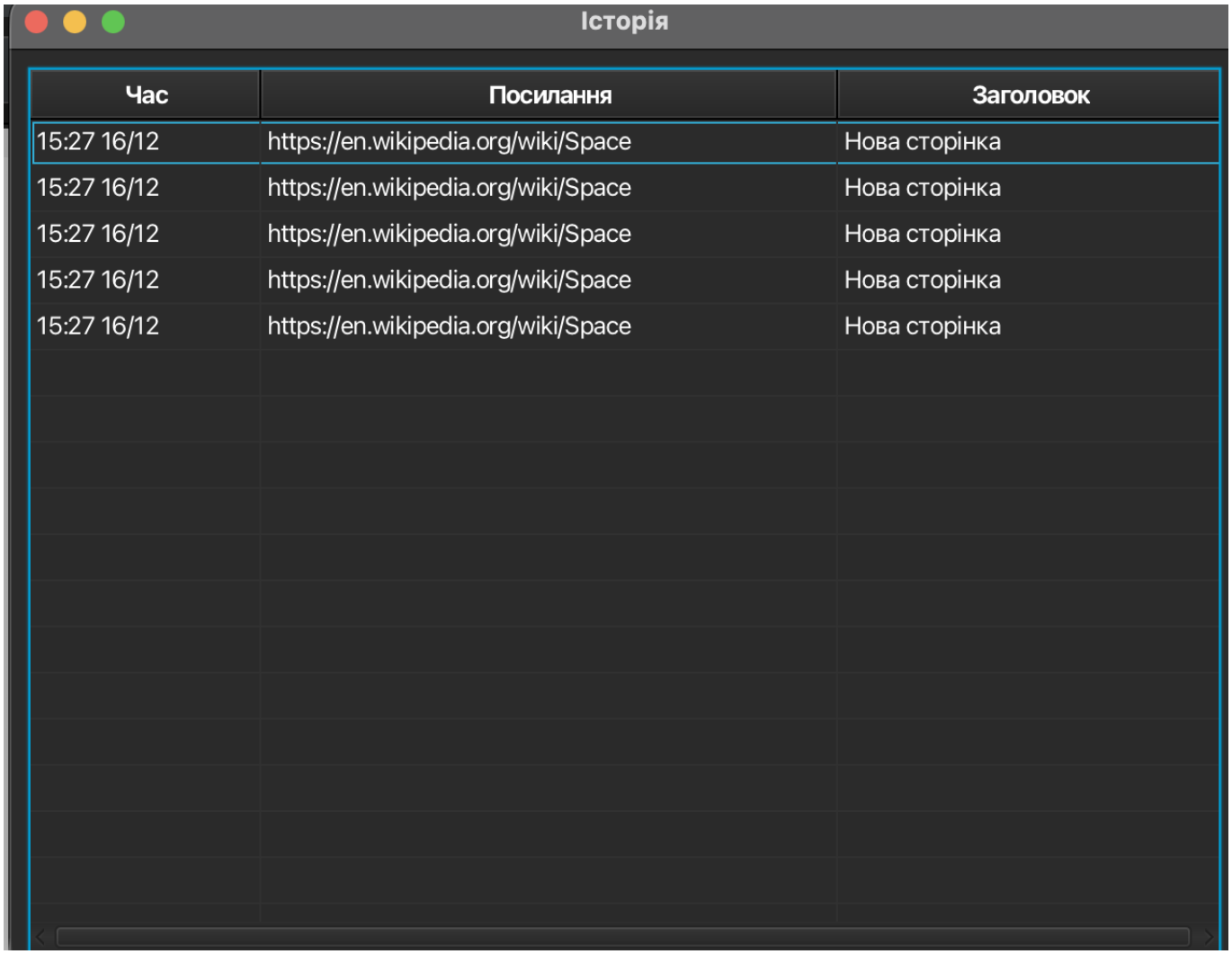

Рис. 3.2 – Вікно перегляду історії пошуку

Додаток В

Лістинг коду основних класів

```
package com.explorer.chain;

import com.explorer.connection.Response;

public class ErrorHandler extends ResponseHandler {

    @Override
    public Response handle(Response response) {
        if (response.isClientError() || response.isServerError()) {

            if (response.getBody() != null && !response.getBody().isEmpty()) {
                return response;
            }


            final int statusCode = response.getStatusCode();
            final String statusText = response.getStatusText();

            if (response.isNotFound()) {
                return Response.notFound("Resource not specified or found.");
            } else if (response.isBadRequest()) {
                return Response.badRequest(statusText);
            } else if (response.isServiceUnavailable() || response.isBadGateway()) {
                return Response.serviceUnavailable(statusText);
            } else if (response.isServerError()) {
                return Response.internalError(statusText);
            } else if (response.isClientError()) {
                String reason = String.format("%d %s: Access Denied or Malformed
Request.", statusCode, statusText);
                return Response.create(
                        statusCode,
                        statusText,
                        null,
                        String.format("<html><body><h1>%d %s</h1><p>The server
responded with an error.</p></body></html>", statusCode, statusText)
                );
            }
            return response;
        }

        return passToNext(response);
    }
}
```

```
package com.explorer.chain;


import com.explorer.connection.Response;

public interface IResponseHandler {
    void setNextHandler(IResponseHandler nextHandler);
    Response handle(Response response);
}
```

```
package com.explorer.chain;

import com.explorer.connection.HttpConnector;
import com.explorer.connection.Request;
import com.explorer.connection.Response;

import java.io.IOException;

public class RedirectHandler extends ResponseHandler {
    private final HttpConnector httpConnector;
    private int redirectCount = 0;
    private static final int MAX_REDIRECTS = 5;

    public RedirectHandler(HttpConnector client) {
        this.httpConnector = client;
    }

    @Override
    public Response handle(Response response) {
        if (response.isRedirect()) {

            if (redirectCount >= MAX_REDIRECTS) {
                return Response.internalError("Redirect loop detected. Exceeded " +
MAX_REDIRECTS + " redirects.");
            }

            String newLocation = response.getHeader("Location");

            if (newLocation != null) {
                System.out.println("Redirecting to: " + newLocation);
                redirectCount++;
                try {
                    Request newRequest = Request.createGet(newLocation);
                    Response newResponse = httpConnector.sendRequest(newRequest);

                    return handle(newResponse);
                } catch (IOException | IllegalArgumentException e) {
                    System.err.println("Redirect failed due to I/O or invalid URL: "
+ e.getMessage());
                    return Response.internalError("Error during redirect processing
to " + newLocation + ": " + e.getMessage());
                }
            } else {
                System.err.println("Redirect response received without Location
header.");
                return Response.internalError("Server sent a redirect code without a
Location header.");
            }
        }

        return passToNext(response);
    }
}
```

```
package com.explorer.chain;

import com.explorer.connection.Response;

abstract class ResponseHandler implements IResponseHandler {
    protected IResponseHandler nextHandler;
```

```
    @Override
    public void setNextHandler(IResponseHandler nextHandler) {
        this.nextHandler = nextHandler;
    }

    protected Response passToNext(Response response) {
        if (nextHandler != null) {
            return nextHandler.handle(response);
        }
        return response;
    }
}
```

```
package com.explorer.connection;

public interface DataFetcher {
    String loadResource(String url);
}
```

```
package com.explorer.connection;

import javax.net.ssl.SSLSocketFactory;
import java.io.*;
import java.net.Socket;
import java.nio.charset.Charset;
import java.nio.charset.StandardCharsets;
import java.nio.charset.UnsupportedCharsetException;
import java.util.HashMap;
import java.util.Map;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class HttpConnector {

    private final Pattern CHARSET_PATTERN = Pattern.compile("charset=([\\w\\-]+)",
Pattern.CASE_INSENSITIVE);

    public Response sendRequest(Request request) throws IOException {
        Socket socket = null;

        try {
            if (request.isSecure()) {
                SSLSocketFactory factory = (SSLSocketFactory)
SSLSocketFactory.getDefault();
                socket = factory.createSocket(request.getHost(), request.getPort());
                System.out.println("Established secure connection to " +
request.getHost());
            } else {
                socket = new Socket(request.getHost(), request.getPort());
            }

            OutputStream output = socket.getOutputStream();
            output.write(request.toRequestString().getBytes(StandardCharsets.UTF_8));
            output.flush();

            InputStream input = socket.getInputStream();
            BufferedReader reader = new BufferedReader(new InputStreamReader(input,
StandardCharsets.ISO_8859_1));
```

```
            return parseResponse(reader);

        } catch (IOException e) {
            System.err.println("Network error: Could not connect to " +
request.getHost() + " over port " + request.getPort() + ". Error: " +
e.getMessage());
            String protocol = request.isSecure() ? "HTTPS" : "HTTP";
            return Response.serviceUnavailable("Connection via " + protocol + " to "
+ request.getHost() + " failed.");
        } finally {
            if (socket != null) {
                try {
                    socket.close();
                } catch (IOException e) {
                }
            }
        }
    }

    private Response parseResponse(BufferedReader reader) throws IOException {
        String statusLine = reader.readLine();
        if (statusLine == null || statusLine.isEmpty()) {
            return Response.internalError("Empty or malformed response from
server.");
        }

        String[] statusParts = statusLine.split(" ", 3);
        if (statusParts.length < 3) {
            return Response.internalError("Malformed status line: " + statusLine);
        }

        int statusCode = Integer.parseInt(statusParts[1]);
        String statusText = statusParts[2];

        Map<String, String> headers = new HashMap<>();
        String headerLine;
        while (!(headerLine = reader.readLine()).isEmpty()) {
            String[] headerParts = headerLine.split(": ", 2);
            if (headerParts.length == 2) {
                headers.put(headerParts[0].trim(), headerParts[1].trim());
            }
        }

        String contentType = headers.get("Content-Type");
        Charset charset = determineCharset(contentType);

        System.out.println("Content-Type: " + contentType);
        System.out.println("Charset " + charset);

        StringBuilder bodyBuilder = new StringBuilder();
        String line;

        while ((line = reader.readLine()) != null) {
            bodyBuilder.append(line).append("\n");
        }

        byte[] isoBytes =
bodyBuilder.toString().getBytes(StandardCharsets.ISO_8859_1);
        String body = new String(isoBytes, charset);

        System.out.println("Body: " + body);
```

```
        return Response.create(statusCode, statusText, headers, body);
    }

    private Charset determineCharset(String contentTypeHeader) {
        if (contentTypeHeader != null) {
            Matcher matcher = CHARSET_PATTERN.matcher(contentTypeHeader);
            if (matcher.find()) {
                String charsetName = matcher.group(1);
                try {
                    return Charset.forName(charsetName);
                } catch (UnsupportedCharsetException e) {
                    System.err.println("Unsupported charset found: " + charsetName);
                }
            }
        }
        return StandardCharsets.UTF_8;
    }
}
```

```
package com.explorer.connection;

import com.explorer.chain.ErrorHandler;
import com.explorer.chain.RedirectHandler;
import com.explorer.chain.IResponseHandler;

import java.io.IOException;

public class HttpProcessor implements DataFetcher {

    private final HttpConnector httpConnector;
    private final IResponseHandler chainStart;

    public HttpProcessor() {
        this.httpConnector = new HttpConnector();

        RedirectHandler redirectHandler = new RedirectHandler(httpConnector);
        IResponseHandler errorHandler = new ErrorHandler();

        redirectHandler.setNextHandler(errorHandler);

        this.chainStart = redirectHandler;
    }

    public Response loadUrl(String url) throws IllegalArgumentException {
        try {
            Request request = Request.createGet(url);

            Response initialResponse = httpConnector.sendRequest(request);

            return chainStart.handle(initialResponse);

        } catch (IllegalArgumentException e) {
            System.err.println("Invalid URL provided: " + url + " - " +
e.getMessage());
            return Response.badRequest("The provided URL format is invalid.");

        } catch (IOException e) {
            System.err.println("Unexpected fatal load error: " + e.getMessage());
            return Response.internalError("An unexpected system error occurred during
```

```
page loading.");
        }
    }

    @Override
    public String loadResource(String url) {
        try {
            Request request = Request.createGet(url);
            Response response = httpConnector.sendRequest(request);

            if (response.isSuccessful()) {
                return response.getBody();
            } else {
                System.err.println("Failed to load resource " + url + ". Status: " +
response.getStatusCode());
                return "";
            }
        } catch (IllegalArgumentException e) {
            System.err.println("Invalid URL for resource: " + url);
            return "";
        } catch (java.io.IOException e) {
            System.err.println("Error loading resource " + url + ": " +
e.getMessage());
            return "";
        }
    }
}
```

```
package com.explorer.connection;

import javax.net.ssl.SSLSocketFactory;
import java.io.*;
import java.net.Socket;
import java.nio.charset.Charset;
import java.nio.charset.StandardCharsets;
import java.nio.charset.UnsupportedCharsetException;
import java.util.HashMap;
import java.util.Map;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class HttpConnector {

    private final Pattern CHARSET_PATTERN = Pattern.compile("charset=([\\w\\-]+)",
Pattern.CASE_INSENSITIVE);

    public Response sendRequest(Request request) throws IOException {
        Socket socket = null;

        try {
            if (request.isSecure()) {
                SSLSocketFactory factory = (SSLSocketFactory)
SSLSocketFactory.getDefault();
                socket = factory.createSocket(request.getHost(), request.getPort());
                System.out.println("Established secure connection to " +
request.getHost());
            } else {
                socket = new Socket(request.getHost(), request.getPort());
            }
```

```
            OutputStream output = socket.getOutputStream();
            output.write(request.toRequestString().getBytes(StandardCharsets.UTF_8));
            output.flush();

            InputStream input = socket.getInputStream();
            BufferedReader reader = new BufferedReader(new InputStreamReader(input,
StandardCharsets.ISO_8859_1));

            return parseResponse(reader);

        } catch (IOException e) {
            System.err.println("Network error: Could not connect to " +
request.getHost() + " over port " + request.getPort() + ". Error: " +
e.getMessage());
            String protocol = request.isSecure() ? "HTTPS" : "HTTP";
            return Response.serviceUnavailable("Connection via " + protocol + " to "
+ request.getHost() + " failed.");
        } finally {
            if (socket != null) {
                try {
                    socket.close();
                } catch (IOException e) {
                }
            }
        }
    }

    private Response parseResponse(BufferedReader reader) throws IOException {
        String statusLine = reader.readLine();
        if (statusLine == null || statusLine.isEmpty()) {
            return Response.internalError("Empty or malformed response from
server.");
        }

        String[] statusParts = statusLine.split(" ", 3);
        if (statusParts.length < 3) {
            return Response.internalError("Malformed status line: " + statusLine);
        }

        int statusCode = Integer.parseInt(statusParts[1]);
        String statusText = statusParts[2];

        Map<String, String> headers = new HashMap<>();
        String headerLine;
        while (!(headerLine = reader.readLine()).isEmpty()) {
            String[] headerParts = headerLine.split(": ", 2);
            if (headerParts.length == 2) {
                headers.put(headerParts[0].trim(), headerParts[1].trim());
            }
        }

        String contentType = headers.get("Content-Type");
        Charset charset = determineCharset(contentType);

        System.out.println("Content-Type: " + contentType);
        System.out.println("Charset " + charset);

        StringBuilder bodyBuilder = new StringBuilder();
        String line;

        while ((line = reader.readLine()) != null) {
            bodyBuilder.append(line).append("\n");
```

```
        }

        byte[] isoBytes =
bodyBuilder.toString().getBytes(StandardCharsets.ISO_8859_1);
        String body = new String(isoBytes, charset);

        System.out.println("Body: " + body);

        return Response.create(statusCode, statusText, headers, body);
    }

    private Charset determineCharset(String contentTypeHeader) {
        if (contentTypeHeader != null) {
            Matcher matcher = CHARSET_PATTERN.matcher(contentTypeHeader);
            if (matcher.find()) {
                String charsetName = matcher.group(1);
                try {
                    return Charset.forName(charsetName);
                } catch (UnsupportedCharsetException e) {
                    System.err.println("Unsupported charset found: " + charsetName);
                }
            }
        }
        return StandardCharsets.UTF_8;
    }
}
```

```
package com.explorer.connection;

import java.util.HashMap;
import java.util.Map;

public class Request {
    private final String method;
    private final String host;
    private final String path;
    private final Map<String, String> headers;
    private final boolean isSecure;

    private Request(String method, String host, String path, boolean isSecure) {
        this.method = method;
        this.host = host;
        this.path = path;
        this.headers = new HashMap<>();
        this.headers.put("Host", host);
        this.headers.put("Connection", "close");
        this.headers.put("User-Agent", "SimpleBrowserFX/1.0 (Java)");
        this.isSecure = isSecure;
    }

    public static Request createGet(String url) throws IllegalArgumentException {
        try {
            java.net.URL urlObj = new java.net.URL(url);
            String protocol = urlObj.getProtocol();
            boolean secure = protocol.equalsIgnoreCase("https");

            String host = urlObj.getHost();
            String path = urlObj.getPath().isEmpty() ? "/" : urlObj.getPath();
            if (urlObj.getQuery() != null) {
                path += "?" + urlObj.getQuery();
```

```
            }
            return new Request("GET", host, path, secure);
        } catch (java.net.MalformedURLException e) {
            throw new IllegalArgumentException("Invalid URL: " + url, e);
        }
    }

    public String getHost() { return host; }

    public int getPort() {
        return isSecure ? 443 : 80;
    }

    public boolean isSecure() { return isSecure; }

    public String toRequestString() {
        StringBuilder sb = new StringBuilder();
        sb.append(method).append(" ").append(path).append(" HTTP/1.1\r\n");

        for (Map.Entry<String, String> entry : headers.entrySet()) {
            sb.append(entry.getKey()).append(":
").append(entry.getValue()).append("\r\n");
        }

        sb.append("\r\n");
        return sb.toString();
    }
}
```

```
package com.explorer.connection;

import java.util.HashMap;
import java.util.Map;

public class Response {

    private final int statusCode;
    private final String statusText;
    private final Map<String, String> headers;
    private final String body;
    private final HttpStatus statusClass;

    private Response(int statusCode, String statusText, Map<String, String> headers,
String body) {
        this.statusCode = statusCode;
        this.statusText = statusText;
        this.headers = headers != null ? headers : new HashMap<>();
        this.body = body != null ? body : "";
        this.statusClass = HttpStatus.getByCode(statusCode);
    }

    public static Response create(int statusCode, String statusText, Map<String,
String> headers, String body) {
        return new Response(statusCode, statusText, headers, body);
    }

    public static Response notFound(String requestedPath) {
        String body = String.format(
                "<html><body><h1>404 Not Found</h1><p>The requested URL
<strong>%s</strong> was not found on this server.</p></body></html>",
```

```
                requestedPath
        );
        Map<String, String> headers = new HashMap<>();
        headers.put("Content-Type", "text/html; charset=utf-8");
        return new Response(404, "Not Found", headers, body);
    }

    public static Response serviceUnavailable(String reason) {
        String body = String.format(
                "<html><body><h1>503 Service Unavailable</h1><p>The server is
temporarily unable to service your request: <strong>%s</strong></p></body></html>",
                reason
        );
        Map<String, String> headers = new HashMap<>();
        headers.put("Content-Type", "text/html; charset=utf-8");
        return new Response(503, "Service Unavailable", headers, body);
    }

    public static Response internalError(String errorDetail) {
        String body = String.format(
                "<html><body><h1>500 Internal Server Error</h1><p>An unexpected error
occurred: <strong>%s</strong></p></body></html>",
                errorDetail
        );
        Map<String, String> headers = new HashMap<>();
        headers.put("Content-Type", "text/html; charset=utf-8");
        return new Response(500, "Internal Server Error", headers, body);
    }

    public static Response badRequest(String reason) {
        String body = String.format(
                "<html><body><h1>400 Bad Request</h1><p>The request could not be
understood by the server due to malformed syntax:
<strong>%s</strong></p></body></html>",
                reason
        );
        Map<String, String> headers = new HashMap<>();
        headers.put("Content-Type", "text/html; charset=utf-8");
        return new Response(400, "Bad Request", headers, body);
    }


    public int getStatusCode() { return statusCode; }
    public String getStatusText() { return this.statusText; }

    public String getHeader(String name) {
        return headers.entrySet().stream()
                .filter(entry -> entry.getKey().equalsIgnoreCase(name))
                .map(Map.Entry::getValue)
                .findFirst()
                .orElse(null);
    }

    public String getBody() { return body; }


    public boolean isSuccessful() { return statusClass == HttpStatus.SUCCESS; }
    public boolean isRedirect() { return statusClass == HttpStatus.REDIRECTION; }
    public boolean isClientError() { return statusClass == HttpStatus.CLIENT_ERROR; }
    public boolean isServerError() { return statusClass == HttpStatus.SERVER_ERROR; }

    public boolean isNotFound() { return statusCode == 404; }
```

```
    public boolean isServiceUnavailable() { return statusCode == 503; }
    public boolean isBadRequest() { return statusCode == 400; }
    public boolean isBadGateway() { return statusCode == 502; }

    @Override
    public String toString() {
        return "HTTP Status: " + statusCode + " " + statusText +
                " (" + statusClass + ")" + "\n" +
                "Headers: " + headers.size() + "\n" +
                "Body size: " + body.length();
    }


    public enum HttpStatus {
        SUCCESS(200, 299),
        REDIRECTION(300, 399),
        CLIENT_ERROR(400, 499),
        SERVER_ERROR(500, 599),
        INFORMATIONAL(100, 199),
        UNKNOWN(-1, -1);

        private final int min;
        private final int max;

        HttpStatus(int min, int max) {
            this.min = min;
            this.max = max;
        }

        public static HttpStatus getByCode(int code) {
            if (code >= SUCCESS.min && code <= SUCCESS.max) return SUCCESS;
            if (code >= REDIRECTION.min && code <= REDIRECTION.max) return
REDIRECTION;
            if (code >= CLIENT_ERROR.min && code <= CLIENT_ERROR.max) return
CLIENT_ERROR;
            if (code >= SERVER_ERROR.min && code <= SERVER_ERROR.max) return
SERVER_ERROR;
            if (code >= INFORMATIONAL.min && code <= INFORMATIONAL.max) return
INFORMATIONAL;
            return UNKNOWN;
        }
    }
}
```

```
package com.explorer.connection;

import java.net.URI;
import java.net.URISyntaxException;

public class UrlResolver {

    public static String resolve(String baseUrl, String relativeUrl) {
        if (relativeUrl == null || relativeUrl.isEmpty()) return "";

        if (relativeUrl.startsWith("//")) {
            if (baseUrl.startsWith("https:")) return "https:" + relativeUrl;
            return "http:" + relativeUrl;
        }

        try {
```

```
            URI base = new URI(baseUrl);
            return base.resolve(relativeUrl).toString();
        } catch (URISyntaxException | IllegalArgumentException e) {
            System.err.println("URL Resolve Error: " + e.getMessage());
            return relativeUrl; // Повертаємо як є, якщо не вийшло
        }
    }
}
```

```
package com.explorer.explorer;

import com.explorer.explorer.render.*;
import com.explorer.explorer.render.model.PageStructure;
import com.explorer.connection.HttpProcessor;
import com.explorer.connection.Response;
import javafx.application.Platform;
import javafx.beans.property.StringProperty;
import javafx.scene.control.Label;
import javafx.scene.layout.VBox;

import java.net.URL;
import java.util.HashMap;
import java.util.Map;

public class PageLoader {

    private final HttpProcessor httpProcessor;
    private final Renderer renderer;
    private final HtmlParser htmlParser;
    private final VBox viewPort;
    private final StringProperty titleProperty;
    private final LayoutComposer layoutComposer;

    private final Map<String, String> loadedScripts = new HashMap<>();
    private String currentBaseUrl;

    public PageLoader(VBox viewPort, StringProperty titleProperty) {
        this.httpProcessor = new HttpProcessor();
        this.htmlParser = new HtmlParser();
        this.renderer = new Renderer();
        this.layoutComposer = new LayoutComposer();
        this.viewPort = viewPort;
        this.titleProperty = titleProperty;
    }

    public void clearScripts() {
        loadedScripts.clear();
    }

    public Map<String, String> getLoadedScripts() {
        return loadedScripts;
    }

    public final void loadAndRender(String url) {
        System.out.println("Starting page load for: " + url);

        Response response = fetchHttpResponse(url);

        if (response.isClientError() || response.isServerError()) {
            displayError(response);
```

```
            return;
        }

        PageStructure domPageStructure = parseHtml(response.getBody());

        fetchResources(domPageStructure);

        applyStyles(domPageStructure);

        buildFxNodes(domPageStructure);

        System.out.println("Page rendering finished.");
    }

    protected Response fetchHttpResponse(String url) {
        try {
            this.currentBaseUrl = new URL(url).toExternalForm();
        } catch (java.net.MalformedURLException e) {
            System.err.println("Invalid URL format for base URL: " + url);
        }
        return httpProcessor.loadUrl(url);
    }

    protected PageStructure parseHtml(String htmlContent) {
        return htmlParser.parse(htmlContent);
    }

    protected void fetchResources(PageStructure domPageStructure) {
        Platform.runLater(() -> viewPort.getChildren().addFirst(new Label("Fetching
resources...")));

        ResourceLoader resourceVisitor = new ResourceLoader(httpProcessor,
currentBaseUrl);

        TreeTraverser.traverse(domPageStructure.getRoot(), resourceVisitor);

        loadedScripts.forEach((scriptName, script) -> {
            System.out.println("Script name" + scriptName);
            System.out.println("Script" + script);
        });

        this.loadedScripts.putAll(resourceVisitor.getLoadedScripts());
    }

    protected void applyStyles(PageStructure domPageStructure) {
        TreeTraverser.traverse(domPageStructure.getRoot(), new StyleInjector());
    }

    protected void buildFxNodes(PageStructure domPageStructure) {
        RenderNode renderRoot = layoutComposer.build(domPageStructure);

        javafx.scene.Node fxRoot = renderer.render(renderRoot);

        Platform.runLater(() -> {
            if (!domPageStructure.title().isEmpty()) {
                titleProperty.set(domPageStructure.title());
            } else {
                titleProperty.set(currentBaseUrl);
            }

            viewPort.getChildren().clear();
            viewPort.getChildren().add(fxRoot);
```

```
        });
    }

    protected void displayError(Response response) {
        Platform.runLater(() -> {
            viewPort.getChildren().clear();
            viewPort.getChildren().add(new Label("Error: " +
response.getStatusCode()));
        });
    }
}
```

```
package com.explorer;

import com.explorer.storage.HistoryVault;
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.TabPane;
import javafx.stage.Stage;

public class ExplorerApp extends Application {

    @Override
    public void start(Stage primaryStage) {
        primaryStage.setTitle("Explorer");

        HistoryVault.init();

        TabPane tabPane = new TabPane();
        tabPane.setStyle("-fx-background-color: #2b2b2b;");

        createNewTab(tabPane);

        javafx.scene.control.Tab addTab = new javafx.scene.control.Tab("+");
        addTab.setClosable(false);
        addTab.setStyle("-fx-font-weight: bold; -fx-font-size: 14px;");
        tabPane.getTabs().add(addTab);

        tabPane.getSelectionModel().selectedItemProperty().addListener((obs, oldVal,
newVal) -> {
            if (newVal == addTab) {
                createNewTab(tabPane);
            }
        });

        Scene scene = new Scene(tabPane, 1100, 800);

        applyDarkTheme(scene);

        primaryStage.setScene(scene);
        primaryStage.show();
    }

    private void createNewTab(TabPane tabPane) {
        Tab tabContent = new Tab();
        javafx.scene.control.Tab tab = new javafx.scene.control.Tab("Нова сторінка");
        tab.setContent(tabContent.getView());

        int size = tabPane.getTabs().size();
        int index = size > 0 ? size - 1 : 0;
```

```
        tabPane.getTabs().add(index, tab);
        tabPane.getSelectionModel().select(tab);

        tabContent.titleProperty().addListener((obs, oldTitle, newTitle) -> {
            if (newTitle != null && !newTitle.isEmpty()) {
                tab.setText(newTitle);
            }
        });
    }

    private void applyDarkTheme(Scene scene) {
        String css = """
            .root {
                -fx-base: #2b2b2b;
                -fx-background: #2b2b2b;
            }
            .tab-pane .tab-header-area .tab-header-background {
                -fx-background-color: #1e1e1e;
            }
            .tab-pane {
                -fx-tab-min-width: 120px;
                -fx-tab-max-width: 200px;
                -fx-tab-min-height: 35px;
            }
            .tab {
                -fx-background-color: #3c3f41;
                -fx-background-insets: 0 1 0 0;
                -fx-background-radius: 5 5 0 0;
                -fx-text-fill: #a9b7c6;
            }
            .tab:selected {
                -fx-background-color: #4e5254;
                -fx-text-fill: white;
                -fx-border-color: #00adb5;
                -fx-border-width: 2 0 0 0;
            }
            .tab-label {
                -fx-font-family: 'Segoe UI', sans-serif;
                -fx-font-size: 13px;
            }
            .button {
                -fx-background-color: #00adb5;
                -fx-text-fill: white;
                -fx-background-radius: 4;
                -fx-cursor: hand;
            }
            .button:hover {
                -fx-background-color: #00ced1;
            }
            .text-field {
                -fx-background-color: #3c3f41;
                -fx-text-fill: white;
                -fx-prompt-text-fill: #6d6d6d;
                -fx-background-radius: 4;
                -fx-border-color: #555;
                -fx-border-radius: 4;
            }
            .text-field:focused {
                -fx-border-color: #00adb5;
            }
            .scroll-pane > .viewport {
```

```
                -fx-background-color: #2b2b2b;
            }
        """;

        scene.getStylesheets().add("data:text/css," + css.replace("\n", ""));
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

```
package com.explorer;

import com.explorer.storage.HistoryVault;
import com.explorer.storage.VisitLog;
import javafx.beans.property.SimpleStringProperty;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableView;
import javafx.scene.layout.BorderPane;
import javafx.stage.Stage;

public class HistoryWindow {

    public void show() {
        Stage stage = new Stage();
        stage.setTitle("Історія");

        TableView<VisitLog> table = new TableView<>();

        TableColumn<VisitLog, String> timeCol = new TableColumn<>("Час");
        timeCol.setCellValueFactory(data -> new
SimpleStringProperty(data.getValue().getFormattedTime()));
        timeCol.setPrefWidth(120);

        TableColumn<VisitLog, String> urlCol = new TableColumn<>("Посилання");
        urlCol.setCellValueFactory(data -> new
SimpleStringProperty(data.getValue().getUrl()));
        urlCol.setPrefWidth(300);

        TableColumn<VisitLog, String> titleCol = new TableColumn<>("Заголовок");
        titleCol.setCellValueFactory(data -> new
SimpleStringProperty(data.getValue().getTitle()));
        titleCol.setPrefWidth(200);

        table.getColumns().addAll(timeCol, urlCol, titleCol);

        table.getItems().addAll(HistoryVault.fetchAll());

        BorderPane root = new BorderPane(table);
        root.setPadding(new Insets(10));
        root.setStyle("-fx-base: #2b2b2b; -fx-control-inner-background: #2b2b2b; -fx-
background-color: #2b2b2b; -fx-table-cell-border-color: transparent;");

        Scene scene = new Scene(root, 640, 480);
        stage.setScene(scene);
        stage.show();
```

```
    }
}
```

```
package com.explorer;

public class Launcher {
    public static void main(String[] args) {
        ExplorerApp.main(args);
    }
}
```

```
package com.explorer;

import com.explorer.explorer.PageLoader;
import com.explorer.storage.HistoryVault;
import javafx.beans.property.SimpleStringProperty;
import javafx.beans.property.StringProperty;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.control.Button;
import javafx.scene.control.ScrollPane;
import javafx.scene.control.TextField;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.HBox;
import javafx.scene.layout.Priority;
import javafx.scene.layout.VBox;

public class Tab {

    private final BorderPane root;
    private final TextField addressBar;
    private final PageLoader pageLoader;
    private final StringProperty titleProperty = new SimpleStringProperty("Нова
сторінка");

    public Tab() {
        VBox viewPort = new VBox(10);
        viewPort.setPadding(new Insets(15));
        viewPort.setStyle("-fx-background-color: #f0f0f0;");

        ScrollPane scrollPane = new ScrollPane(viewPort);
        scrollPane.setFitToWidth(true);
        scrollPane.setFitToHeight(true);
        scrollPane.setStyle("-fx-background-color: #2b2b2b; -fx-border-color:
transparent;");

        addressBar = new TextField("https://en.wikipedia.org/wiki/Space");
        addressBar.setPromptText("Введіть посилання");
        addressBar.setPrefHeight(35);

        Button goButton = new Button("➔");
        goButton.setPrefHeight(35);
        goButton.setPrefWidth(40);
        goButton.setStyle("-fx-font-size: 16px; -fx-padding: 0;");

        pageLoader = new PageLoader(viewPort, titleProperty);

        Button historyBtn = new Button("Історія");
        historyBtn.setPrefHeight(35);
```

```
        historyBtn.setOnAction(e -> new HistoryWindow().show());

        HBox topBar = new HBox(10, addressBar, goButton, historyBtn);
        HBox.setHgrow(addressBar, Priority.ALWAYS);
        topBar.setAlignment(Pos.CENTER_LEFT);
        topBar.setPadding(new Insets(10, 15, 10, 15));

        topBar.setStyle("-fx-background-color: #1e1e1e; -fx-border-color: #000; -fx-
border-width: 0 0 1 0;");

        goButton.setOnAction(e -> loadPage());
        addressBar.setOnAction(e -> loadPage());

        root = new BorderPane();
        root.setTop(topBar);
        root.setCenter(scrollPane);
    }

    public BorderPane getView() {
        return root;
    }

    public StringProperty titleProperty() {
        return titleProperty;
    }

    private void loadPage() {
        String url = addressBar.getText().trim();
        if (url.isEmpty()) return;

        if (!url.startsWith("http://") && !url.startsWith("https://")) {
            url = "https://" + url;
            addressBar.setText(url);
        }

        pageLoader.clearScripts();

        final String finalUrl = url;
        String currentTitle = titleProperty.get();
        HistoryVault.logVisit(finalUrl, currentTitle);
        new Thread(() -> pageLoader.loadAndRender(finalUrl)).start();
    }
}
```