МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра інформаційних технологій

«До захисту допущено»
В.о. завідувача кафедри

_____ Світлана ВАЩЕНКО

____ _____ 2024 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня магістр

зі спеціальності 122 «Комп'ютерні науки»,

освітньо-наукової програми «Інформаційні технології проектування»

на тему: **«Інформаційна технологія оптимізації швидкодії вебсайту»**

Здобувача групи _ІТ.м-21н_ _Задесенця Дениса Сергійовича__
(шифр групи) (прізвище, ім'я, по батькові)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

_____ _Денис ЗАДЕСЕНЕЦЬ
(підпис) (Ім'я та ПРІЗВИЩЕ здобувача)

Керівник ___к.т.н., доцент Анна НЕНЯ_____ _____
(посада, науковий ступінь, вчене звання, Ім'я та ПРІЗВИЩЕ) (підпис)

Суми – 2024

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра інформаційних технологій
Спеціальність 122 «Комп'ютерні науки»
Освітньо-наукова програма «Інформаційні технології проектування»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри ІТ

Світлана ВАЩЕНКО
«_____» _____ 2024 р.

ЗАВДАННЯ

на кваліфікаційну роботу магістра студентові

Задесенець Денис Сергійович
(прізвище, ім'я, по батькові)

1 Тема кваліфікаційної роботи *Інформаційна технологія оптимізації швидкодії вебсайту*

затверджена наказом по університету від «01» лютого 2024 р. № 0096-VI

2 Термін здачі студентом кваліфікаційної роботи «10» травня 2024 р.

3 Вхідні дані до кваліфікаційної роботи *технічне завдання на розробку рішення для оптимізації швидкодії завантаження вебсайту*

4 Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) *Аналіз предметної області; Постановка задачі та аналіз методів дослідження; Моделювання та проектування, Розробка плагіну та тестування*, презентація.

5 Перелік графічного матеріалу (з точним зазначенням обов'язкових слайдів презентації) *Об'єкт, предмет, мета та гіпотеза; Актуальність і передумови досліджень; Вибір методів реалізації; контекстна діаграма IDEF0; декомпозиція IDEF0; UML діаграма; Процес розробки: Програмна реалізація інформаційної технології; Тестування, Висновки; Апробація результатів роботи*.

6. Консультанти випускної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

Дата видачі завдання _____.

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної області	30.11.2023-22.02.2024	
2	Визначення вимог	23.01.2024-16.02.2024	
3	Аналіз і вибір методів реалізації	17.02.2024-15.03.2024	
4	Створення сайту	16.03.2024-10.04.2024	
5	Реалізація алгоритму оптимізації швидкодії завантаження вебсайту	11.04.2024-25.04.2024	
6	Тестування та виправлення помилок реалізації	26.04.2024-30.04.2024	
7	Завершення проекту	01.05.2024-10.05.2024	
8	Презентація	22.05.2024	

Магістрант _____ Задесенець Д.С.

Керівник роботи _____ к.т.н., доц. Неня А.В.

РЕФЕРАТ

Тема роботи: «Інформаційна технологія оптимізації швидкодії завантаження вебсайту»

Пояснювальна записка складається зі вступу, 4 розділів, висновків, списку використаних джерел із 41 найменування, додатків. Загальний обсяг роботи − 111 сторінок, у тому числі 83 сторінок основного тексту, 5 сторінки списку використаних джерел, 19 сторінок додатків.

Кваліфікаційну роботу магістра присвячено створенню інформаційної технології оптимізації швидкодії завантаження вебсайту.

В першому розділі роботи приведено результати огляду досліджень і публікацій з представленої теми, розглянуто методи оцінки продуктивності працівників, визначено переваги та недоліки даних методів.

Другий розділ присвячено формулюванню мети та задачі створюваного проекту, представлені результати аналізу алгоритмів оптимізації швидкодії завантаження вебсайтів які використовуються в наш час, в процесі якого були визначені їх переваги та недоліки.

Третій розділ представляє результати проектування інформаційної технології оптимізації швидкодії вебсайтів, а саме подані результати структурно-функціонального моделювання, моделюванню варіантів використання та проектуванню інформаційної технології.

У четвертому розділі роботи описано процес створення технології, демонструються варіанти її використання.

Результатом кваліфікаційної роботи магістра є інформаційна технологія оптимізації швидкодії завантаження вебсайту.

Ключові слова: веб-кешування, Critical Rendering Path, Lazy Loading, HTTP/2 Server Push, оптимізація, вебсайт, швидкість завантаження, BLOB.

ЗМІСТ

ВСТУП

Актуальність. У сучасному цифровому просторі вебсайти продовжують бути незамінними інструментами для компаній та їх клієнтів. Вирішальними факторами успіху таких вебсайтів є їх швидкість, ефективність та якість користувацького інтерфейсу.

Правильно спроектована структура та дизайн вебсайту надає багато переваг організації та їх клієнтам:

- прискорює час завантаження сторінок для користувачів;
- надає швидкий та зручний пошук інформації;
- дозволяє відслідковувати діяльність підприємства;
- збільшує обсяг користувачів.

Тому задача оптимізації вебсайтів стоїть дуже гостро перед розробниками, оскільки від їх рішень залежить швидкість обслуговування, продуктивність роботи та загальне задоволення користувачів.

Дослідження кваліфікаційної роботи магістрі зосереджено на аналізі та створенні методів для підвищення ефективності вебсайтів, що сприятиме покращенню їхньої продуктивності та користувацького досвіду. Оптимізація включає різні аспекти, від архітектурних розробок до оптимізації запитів до бази даних та оптимізації сторінок завантаження.

Об’єкт дослідження. Процес зберігання, завантаження та керування даними на клієнтській частині вебсайту.

Предмет дослідження. Технології керування локальним станом даних вебсайту.

Наукова новизна полягає у розробленні інформаційної технології на основі комбінації алгоритмів кешування та алгоритмів завантаження даних у формат BLOB.

Практична значимість. Впровадження інформаційної технології на основі вдосконаленого методу кешування дозволяє зменшити час завантаження сторінок

вебсайту, покращити швидкість роботи вебсайту та забезпечити простоту застосування розробниками.

Мета. Розроблення інформаційної технології оптимізації швидкодії вебсайту на комбінації алгоритмів кешування та алгоритмів завантаження даних у формат BLOB, що дозволить зменшити час завантаження сторінок вебсайту, покращити швидкість роботи вебсайту та забезпечити простоту застосування розробниками.

Результати дослідження та розробки можуть бути використані для вдосконалення вебсайтів, забезпечення швидкого завантаження сторінок, ефективної роботи з базою даних та оптимального використання ресурсів.

Для досягнення поставленої мети вирішуються такі задачі:

1. Аналіз сучасних методів оптимізації завантаження контенту;
2. Розробка алгоритму оптимізації завантаження вебдодатку;
3. Моделювання інформаційної системи оптимізації завантаження вебдодатку;
4. Імплементація алгоритму у вебсайт
5. Тестування отриманого рішення;
6. Порівняння отриманих результатів з іншими підходами до вирішення цієї задачі.

Результат роботи. Впроваджена інформаційна технологія в процес розроблення вебсайту та його функціонування забезпечує високий стандарт продуктивності та користувальницького досвіду. Результати досліджень та розробок можуть бути використані як практичний внесок у галузь оптимізації вебсайтів та служити основою для подальших досліджень у цій області.

Отримані результати мають вагоме значення на покращення користувацького досвіду, що впливає на популярність веб-ресурсів, бізнесу. Розробка та покращення методів оптимізації завантаження вебсайтів сприятиме економії ресурсів серверів, інтернет-трафіку, зменшенню вимог до обладнання користувачів.

Апробація результатів дослідження. Результати дослідження апробовані на наукових конференціях:

- «Інформатика Математика Автоматика», м. Суми: СумДУ, 22-26 квітня 2024 р.
- «Сучасні світові тенденції розвитку науки та інформаційних технологій», м. Одеса, 30–31 травня 2024 р.

Результати дослідження подані до публікації в збірнику праць VII Міжнародній науково-практичній конференції «Сучасні світові тенденції розвитку науки та інформаційних технологій» (рукопис статті прийнятий до друку редакцією від 09.05.2024 р.)

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз літературних джерел за темою кваліфікаційної роботи.

Веброзробка – складний, тривалий і багатоетапний процес. Сьогодні існує занадто багато різних підходів до розробки вебсайтів. Багато початківців, а також досвідчених інженерів заглиблюються в сучасні та популярні технології, але забувають про основи роботи Інтернету. У результаті розробники вибирають неправильну основу для своїх вебсайтів, що може сильно вплинути на якість кінцевого продукту [5].

Вебсайт являє собою набір взаємопов’язаних вебсторінок, до яких отримання доступу відбувається через мережу Інтернет за допомогою гіпертекстових посилань. Це електронна система зберігання даних, що містить різноманітний контент: текст, зображення, аудіо та відео.

Вебсайти служать для різних цілей та потреб користувачів у цифровому світі. Ось детальніше про призначення основних типів:

1. Інформаційні ресурси – вебсайти, що зосереджені на наданні інформації своїм відвідувачам.
2. Електронна комерція – це інтернет-магазини, платформи для продажу та аукціони, де покупці та продавці можуть взаємодіяти в цифровому просторі.
3. Комунікація та соціальні мережі – вебсайти, що сприяють соціальній взаємодії та зв’язкам між людьми через мережу Інтернет.
4. Освітні платформи – націлені на надання освітніх матеріалів, курсів та ресурсів для навчання.
5. Розваги та медіа – вебсайти, призначені для надання доступу чи відтворенні мультимедійного контенту – музики, ігор, фільмів та інші.

6. Портфоліо та особисті сторінки – індивідуальні вебсайти, створені для демонстрації робіт, проєктів, навичок або особистих досягнень [4].

Розробка структури вебсайту є складною задачею, що вимагає ретельного планування. Вибір структури впливає на зручність навігації та загальний досвід користувача. Ось докладніший огляд декількох поширених моделей організації контенту у вебсайтах:

1. Стандартна структура

Цей базовий підхід передбачає наявність головної сторінки як центрального елемента, з якого розгалужуються посилання на інші сторінки вебдодатку (рис. 1.1). Кожна з цих сторінок має посилання на головну сторінку або меню, що дозволяє мати легку навігацію. Така структура забезпечує простоту та інтуїтивність у використанні.

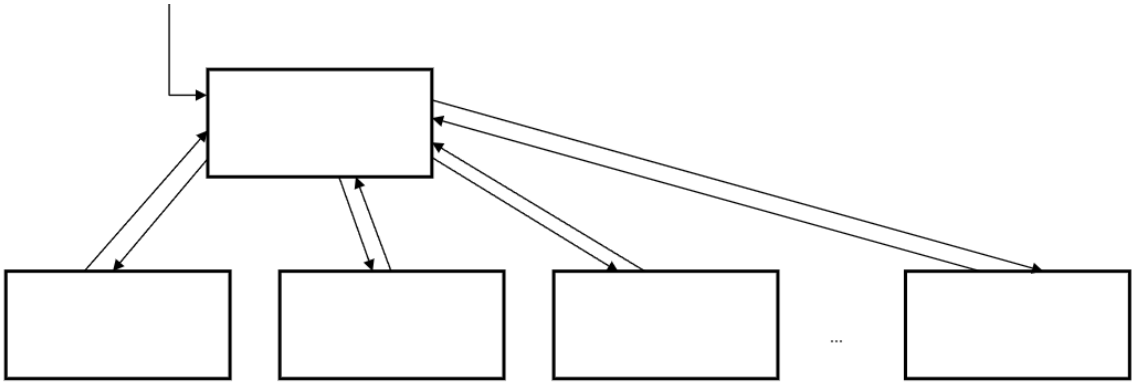

Рисунок 1.1 – Стандартна структура вебсайту

Джерело:[38]

2. Каскадна структура

В основі цієї моделі лежить ідея послідовного перегляду сторінок, де користувач може рухатися вперед або назад (рис. 1.2). Цей підхід часто застосовується для інструкцій, книг або презентацій, де важливо зберігати послідовність інформації.

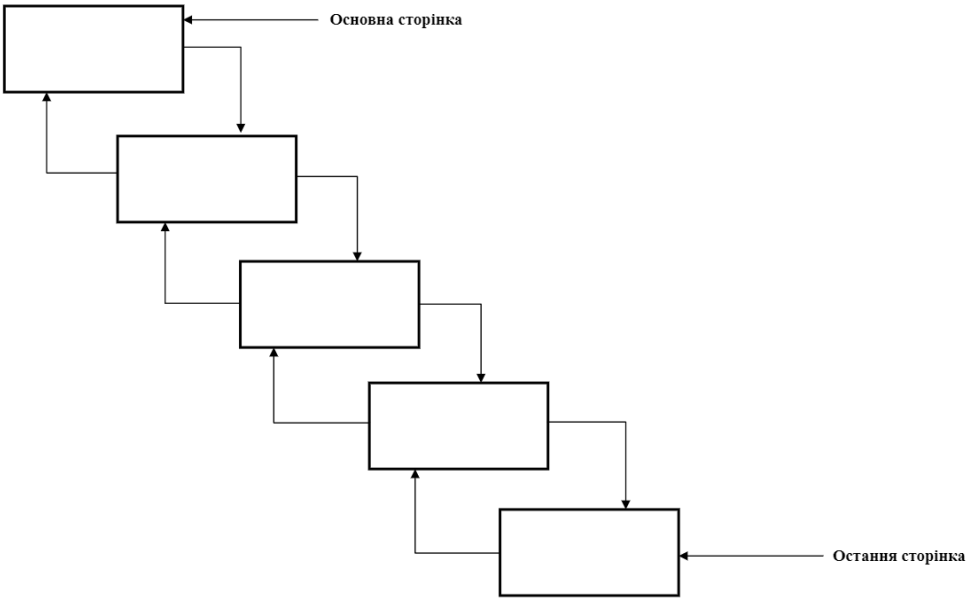

Рисунок 1.2 – Каскадна структура вебсайту

Джерело:[38]

3. Структура «Хмарочос»

У цій моделі доступ до певних рівнів інформації відбувається через ієрархічний підхід, де кожен наступний рівень доступний тільки після відвідування попереднього (рис. 1.3). Це схоже на переміщення між поверхами хмарочосу, де кожен поверх веде до наступного.

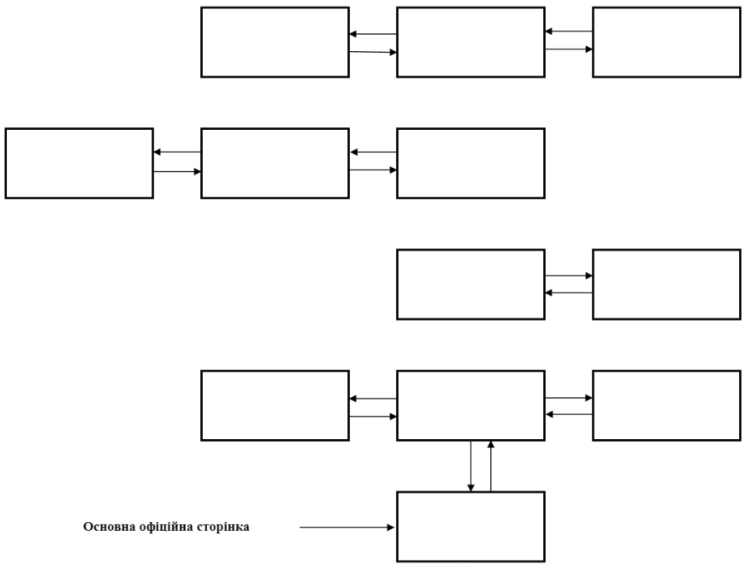

Рисунок 1.3 – Структура сайту типу «Хмарочос»

Джерело:[38]

4. Структура «Павутинна»

Цей варіант створює складну систему зв’язків, де кожна сторінка може містити посилання на багато інших сторінок сайту (рис. 1.4). Така структура сприяє високій взаємоз’єднаності контенту, але може бути складною та заплутаною.

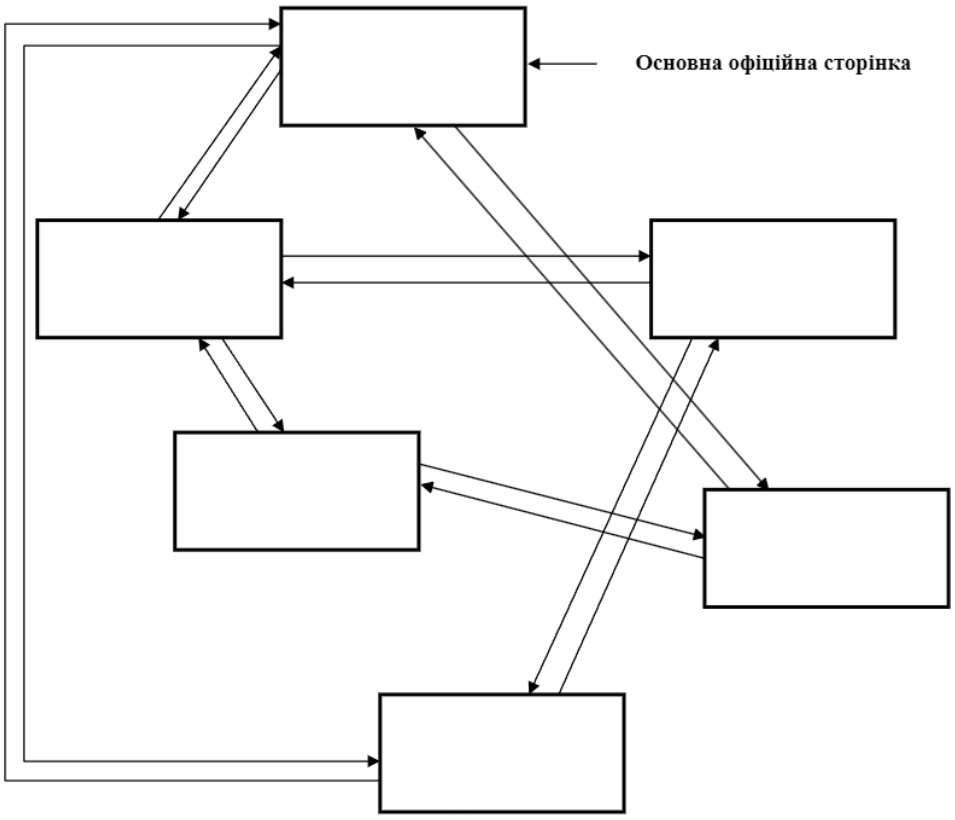

Рисунок 1.4 – Структура сайту типу «Павутина»

Джерело:[38]

Кожен тип структури має свої унікальні переваги та може бути оптимальним в залежності від цілей вебдодатку та потреб його користувачів. Важливо обирати структуру, яка найкраще відповідає змісту додатку та сприяє зручній та ефективній навігації [3].

Одне з головних рішень, які мають прийняти веб-розробники, це як реалізувати логіку та рендер графіки у своєму додатку. Це може бути важко, оскільки існує кілька способів реалізації вебсайту. Щоб краще зрозуміти принцип функціонування

вебсайту, заснований на обраній архітектурі, одну з яких слід вибрати під час прийняття рішення на етапі розроблення, потрібно чітко розуміти кожен підхід і узгоджену термінологію, яку можна застосовувати.

Саме тому актуальним є завдання впровадження методів і підходів для підвищення продуктивності вебсайтів на етапі проєктування.

Існує багато варіантів того, як оптимізувати вебсайт але щоб їх згрупувати та систематизувати, можна використовувати модель OSI (Open Systems Interconnection) [7]. Рівні моделі OSI зображені на рисунку 1.5.

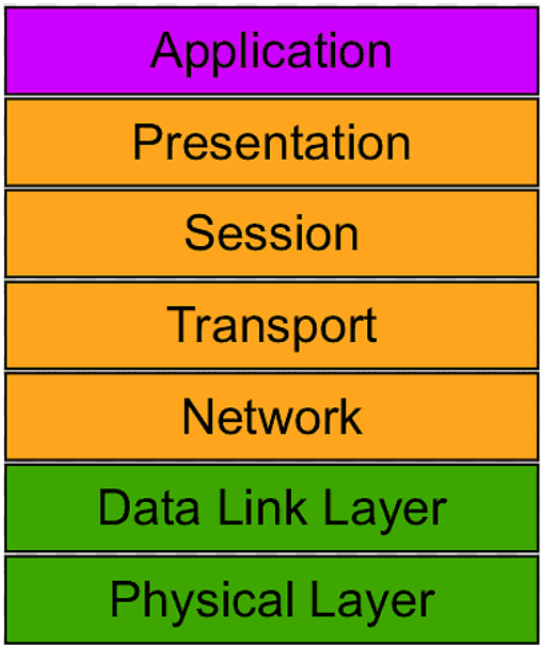

Рисунок 1.5 – Рівні моделі OSI

Джерело: [25]

Модель OSI, розроблена для стандартизації комунікацій у мережах, включає сім рівнів. Кожен рівень має власні завдання та обов'язки, які сприяють ефективній передачі даних.

1. Рівень застосунків є верхнім шаром в архітектурі OSI. Цей рівень слугує мостом для користувачів та їхніх додатків до мережі, дозволяючи взаємодію з мережевими сервісами. Протоколи, як-от HTTP, SMTP, FTP, діють на цьому етапі, спрощуючи обмін інформацією.

2. Рівень подання займається уніфікацією формату даних для різних систем. Цей шар вирішує задачі, пов’язані з перетворенням форматів даних, шифруванням, компресією, забезпечуючи безперешкодну взаємодію між системами.

3. Рівень сесії відповідає за встановлення, управління та завершення сесій зв’язку між програмами на різних кінцевих точках. Цей шар забезпечує координацію діалогу та синхронізацію передачі даних.

4. Транспортний рівень гарантує надійну передачу даних між вузлами. Шар контролює розмір, порядок пакетів даних та їх відновлення при помилках, використовуючи протоколи TCP та UDP.

5. Мережевий рівень відповідає за маршрутизацію пакетів через складні мережеві структури. Цей шар забезпечує адресацію, вибір маршруту для даних за допомогою протоколу IP.

6. Рівень каналу даних забезпечує передачу даних між безпосередньо з’єднаними вузлами. Він управляє доступом до медіа, виявляє та виправляє помилки на рівні фізичного каналу, використовуючи Ethernet, Wi-Fi, PPP.

7. Фізичний рівень є основою для передачі бітів через різноманітні фізичні медіа, такі як кабелі або бездротові з’єднання. Цей шар визначає електричні, механічні, процедурні характеристики з’єднань.

Оптимізація продуктивності вебсайту знаходиться на рівні застосунків, де розробники мають найбільші можливості для впровадження удосконалень. На цьому шарі можна застосувати оптимізацію алгоритмів обробки даних, ефективне використання ресурсів, кешування, компресію даних, паралельну обробку та інші техніки. Це дозволяє зменшити час відгуку додатків, знизити навантаження на мережу та покращити загальну продуктивність, забезпечуючи користувачам кращий досвід використання [4].

1.2 Аналіз методів та технологій оптимізації швидкодії вебсайту

Розглянемо чотири основних типи рендерингу інтерфейсу користувача вебсайтів: сервер-сайд, статичний, клієнт-сайд, гідратаційний та WebSockets.

Рендеринг на стороні сервера (SSR) – це коли додаток подає клієнту простий HTML, уникаючи відправлення великих об'ємів JavaScript клієнту. [27] SSR можна розділити на два типи: SSR з гідратацією та SSR без гідратації. SSR – це стара техніка, яка використовується старими фреймворками, такими як WordPress, Ruby on Rails та ASP.NET (рис. 1.6). Основна мета SSR – дати користувачеві статичний HTML з попередніми даними. На відміну від CSR, SSR не потрібно робити ще один виклик API до сервера, оскільки сервер генерує HTML-шаблон і завантажує в нього будь-які дані [1].

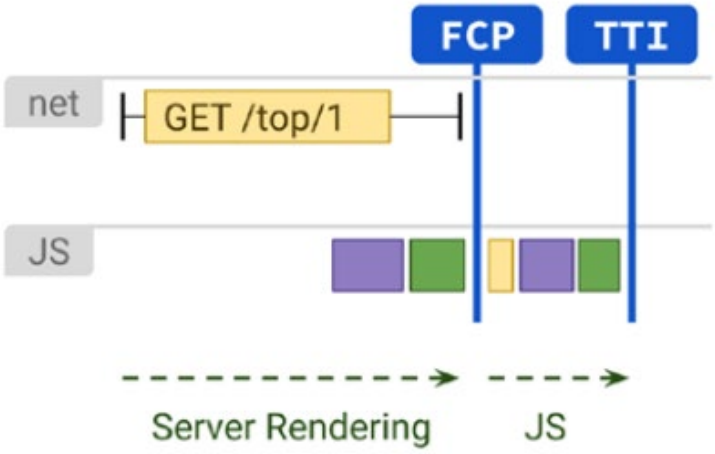

Рисунок 1.6 – Схема SSR рендерингу

Джерело: [26]

Статичний рендеринг є найпростішим з усіх методів рендерингу вебсайтів. Статична вебсторінка рендериться під час збірки додатку та забезпечує швидке перше відображення контенту та час до інтерактивності, за умови, що кількість JS на стороні клієнта обмежена. На відміну від візуалізації на сервері, статичний рендеринг може досягти послідовно швидкого часу до першого байту (TTFB), оскільки HTML для

сторінки не потрібно генерувати «на льоту» (runtime). Зазвичай, статичний рендеринг передбачає створення окремого HTML-файлу для кожного URL заздалегідь. Оскільки HTML-відповіді генеруються заздалегідь, статичні візуалізації можуть бути розгорнуті на кількох CDN для використання переваг кешування (рис. 1.7) [8].

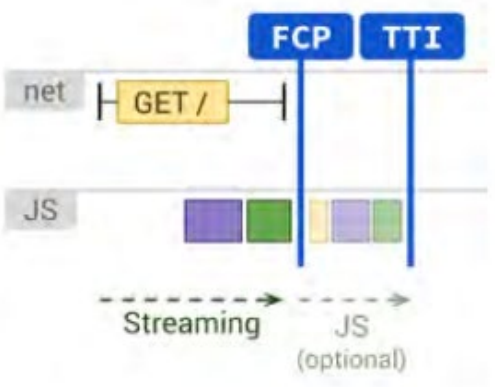

Рисунок 1.7 – Схема статичного рендерингу

Джерело: [26]

Клієнт-сайд рендеринг (CSR) означає відтворення сторінок безпосередньо в браузері за допомогою JavaScript (рис. 1.8). Вся логіка, вибірка даних, створення шаблонів і маршрутизація обробляються на клієнті, а не на сервері [4].

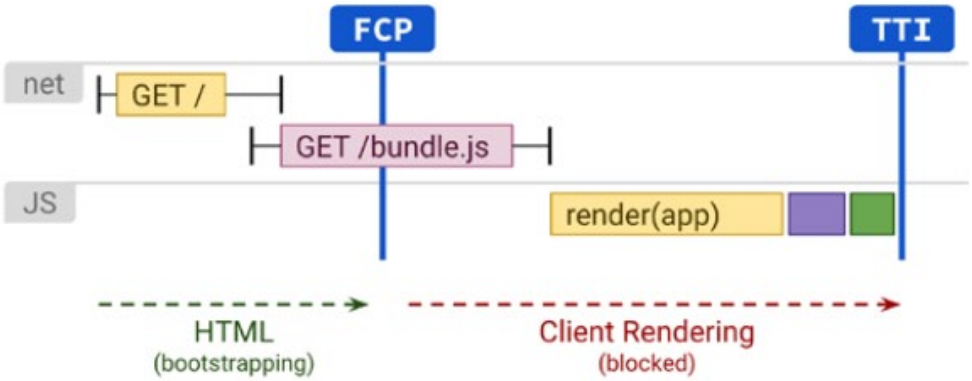

Рисунок 1.8 – Схема CSR

Джерело: [26]

Гідратація, яку часто називають універсальним рендерингом або просто SSR (серверний рендеринг), прагне усунути розрив між клієнтським та серверним відображенням шляхом їх комбінування (рис. 1.9). Запити навігації, такі як повне завантаження або перезавантаження сторінки, обробляються сервером який генерує вебсайт у HTML, після чого JavaScript та дані, що використовуються для рендерингу, вбудовуються в результативний документ та відображаються вже на стороні клієнта [32].

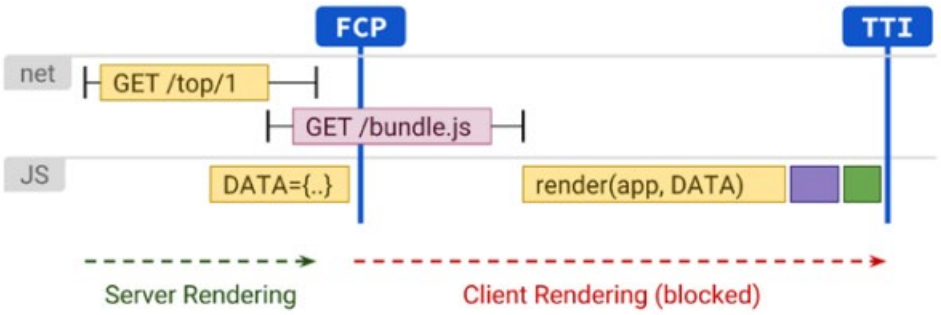

Рисунок 1.9 – Схема рендерингу Гідратації

Джерело: [26]

Ви можете встановити з’єднання в реальному часі між Front-end і Back-end через WebSockets. WebSockets – це механізм двостороннього зв’язку, який спирається на події.

У загальній архітектурі WebSocket додаток Front-end підключається до WebSocket API, шини подій або бази даних. Більшість архітектур використовують його як заміну REST, особливо в таких випадках використання, як чат-додатки; опитування вашого Back-end кожні кілька секунд стає дуже неефективним рішенням. WebSockets дозволяє отримувати оновлення з іншого кінця без необхідності створювати новий запит через двостороннє з’єднання (рис. 1.10) [31].

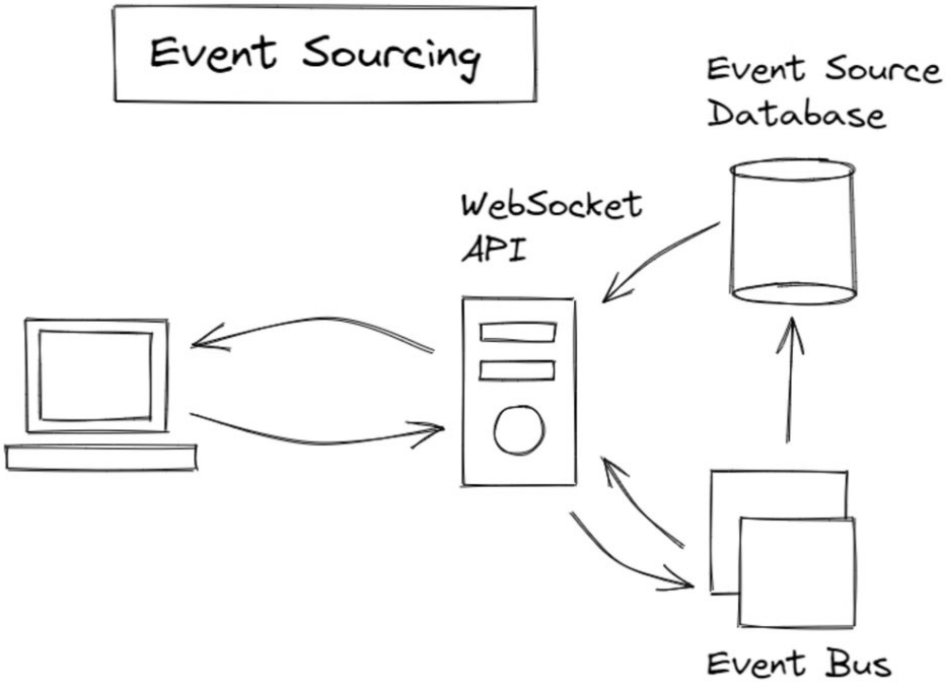

Рисунок 1.10 – Схема WebSocket

Джерело: [27]

WebSockets створюють крихітне з’єднання в порівнянні зі звичайними HTTP-запитами. Поєднання WebSockets з локальним кешем браузера створює додаток в реальному часі. Можна оновити стан програми на основі подій, отриманих від WebSocket. Однак існують деякі застереження щодо продуктивності, масштабованості та потенційних конфліктів даних [18].

Ефективність рендерингу вебсайтів відіграє важливу роль у забезпеченні швидкості завантаження, користувацького досвіду та оптимізації для пошукових систем. Різні методи рендерингу пропонують унікальні переваги та компроміси з точки зору продуктивності, доступності та динамічності контенту. Таблиця 1.1 надає порівняльний аналіз цих методів, висвітлюючи їхні ключові переваги та недоліки.

Таблиця 1.1. Переваги та недоліки різних методів рендерингу вебдодатку

Метод	Переваги	Недоліки
Статичний рендеринг	1. Швидкий рендеринг та час до початкової взаємодії 2. Дружній до оптимізації пошукових систем 3. Інкрементне статичне регенерування	1. Погана масштабованість 2. Відсутність динамічного вмісту 3. Деякі користувачі можуть бачити стару версію сторінок через проблеми кешування
Рендеринг на боці клієнта	1. Швидке завантаження та навігація 2. Користувачі мають досвід користування як SPA	1. Повільний початковий рендеринг 2. Погано оптимізований для пошукових систем 3. Великий об’єм пакету завантаження 4. JavaScript нерідко занадто великий
Гідратація	1. Швидкий початковий рендеринг 2. Дружній до оптимізації пошукових систем 3. Оптимізований JavaScript 4. Потоковий рендеринг	1. Практично, потрібно побудувати два застосунки – серверний та клієнтський 2. Спочатку отримуєте дані потім відбувається показ контенту

Продовження таблиці 1.1

WebSocket	1. Дозволяє двосторонній обмін даними в реальному часі 2. Зменшує навантаження на сервер, оскільки не потребує повторного з'єднання для кожного запиту 3. Ефективний у використанні пропускної спроможності мережі 4. Підходить для інтерактивних додатків, які вимагають швидкого обміну даними	1. Потребує налаштування сервера для підтримки протоколу WebSocket 2. Може бути складніше управляти та масштабувати порівняно з традиційним HTTP/HTTPS-з'єднанням 3. Потрібно вводити додаткових заходів безпеки, як TLS/SSL 4. Старі версії браузерів можуть не підтримувати WebSocket або по причині відсутності використання поліфілів

Джерело: розроблено автором

Для оптимізації продуктивності вебсайту важливо обрати метод рендерингу, який найкраще відповідає його потребам. Якщо пріоритетом є миттєве відображення контенту та висока видимість у пошукових системах, то статичний рендеринг або SSR можуть бути оптимальними виборами. Ці методи забезпечують швидке первісне завантаження сторінок, що сприяє позитивному враженню користувачів.

У ситуаціях, коли необхідно багато клієнтської взаємодії та динамічного контенту, рендеринг на стороні клієнта або використання гідратації може забезпечити більш плавний досвід користувача після первісного завантаження. Ці методи

вимагають оптимізації завантаження ресурсів, але мають велику гнучкість в інтерактивності додатків.

WebSocket повинен використовуватися для сценаріїв, які вимагають негайного обміну даними, наприклад для онлайн ігор, торгових платформ або чатів. Він знижує затримки в зв’язку, мінімізуючи навантаження на сервер та оптимізуючи пропускну здатність мережі.

При виборі методу рендерингу для оптимізації продуктивності вебсайту, важливо також розглянути майбутнє масштабування проєкту, вимоги до інфраструктури та ресурси, доступні для підтримки та розвитку вебсайту. Баланс між швидкодією, функціональністю та вартістю утримання є ключем до успішної реалізації оптимізованого вебсайту.

2 ПОСТАНОВКА ЗАДАЧІ ТА МЕТОДИ ДОСЛІДЖЕННЯ

2.1 Мета та задачі дослідження

Метою даного дослідження є розроблення інформаційної технології оптимізації завантаження вебсайтів на основі комбінації алгоритмів кешування та алгоритмів завантаження даних у формат BLOB з подальшим впровадженням її в розробку вебсайту із застосуванням ефективних методів оптимізації, спрямованих на покращення швидкодії вебсайтів.

Дослідження цього проєкту фокусується на проблематиці створення масштабних вебсайтів, що вимагають значних витрат часу, ресурсів та ефективної організації коду для оптимізації часу обробки запитів. Розробка таких сайтів є часозатратним через необхідність повної реалізації всіх функцій та виконання задач з використанням різноманітних бібліотек, класів та функцій, що вимагає залучення команди кваліфікованих спеціалістів. При успішній реалізації проєкту, існує ризик низької продуктивності сайту через обмеження мережевої пропускної спроможності та апаратного забезпечення. Крім того, неструктурований код може ускладнити розуміння та подальшу роботу над проєктом, особливо коли до команди приєднуються нові розробники, що вимагає додаткового часу на аналіз та ознайомлення з кодом.

Для досягнення поставленої мети вирішуються такі задачі:

1. Аналіз сучасних методів оптимізації завантаження контенту;
2. Розробка алгоритму оптимізації завантаження вебсайту;
3. Моделювання інформаційної системи оптимізації завантаження вебдодатку;
4. Імплементація алгоритму у вебсайт;
5. Тестування отриманого рішення;
6. Аналіз отриманого рішення переваги та недоліки;

7. Порівняння отриманих результатів з іншими підходами до вирішення цієї задачі.

2.2 Сервіси аналізу швидкодії вебсайту

Для ефективного аналізу та оптимізації завантаження вебсторінки, є можливість використовувати різні сервіси, що можуть оцінити основні показники сайту.

PageSpeed Insights (PSI) — це інструмент від Google, який аналізує зміст вебсторінки та надає пропозиції щодо того, як можна покращити її швидкість завантаження та загальний користувацький досвід, як на мобільних, так і на десктопних пристроях (рис. 2.1) [8].

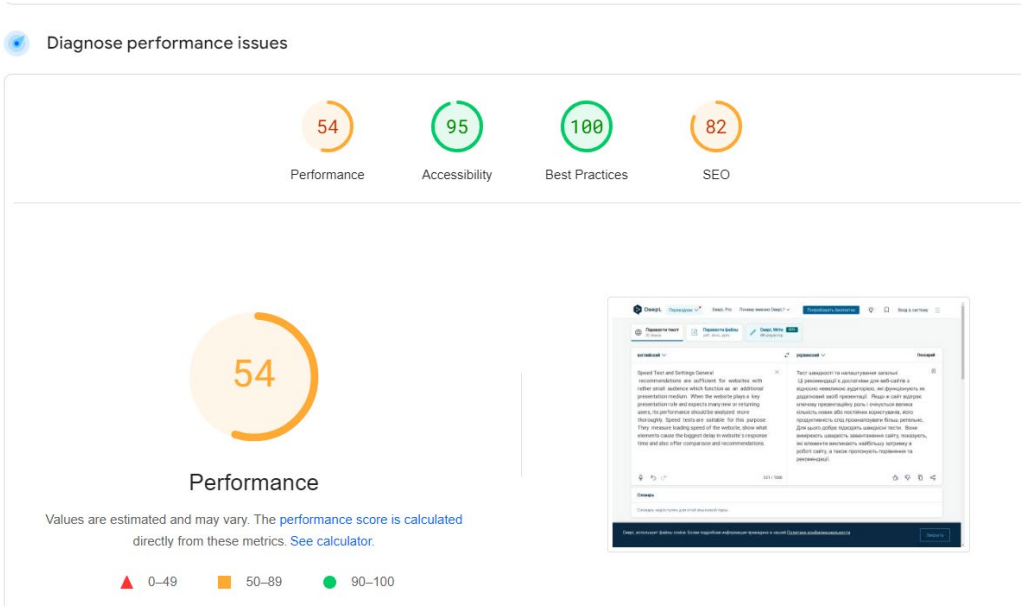

Рисунок 2.1 – Ілюстрація результатів аналізу швидкодії вебдодатку інструментом PageSpeed Insights

Джерело:[33]

GTmetrix є одним із провідних інструментів для аналізу швидкості та продуктивності вебсайтів. Він надає детальний огляд того, як сайт завантажується та виявляє конкретні проблеми, які можуть сповільнювати його та надає конкретні рекомендації для їх оптимізації, такі як мінімізація файлів, оптимізація зображень або виправлення блокування рендерингу ресурсів (рис. 2.2) [11].

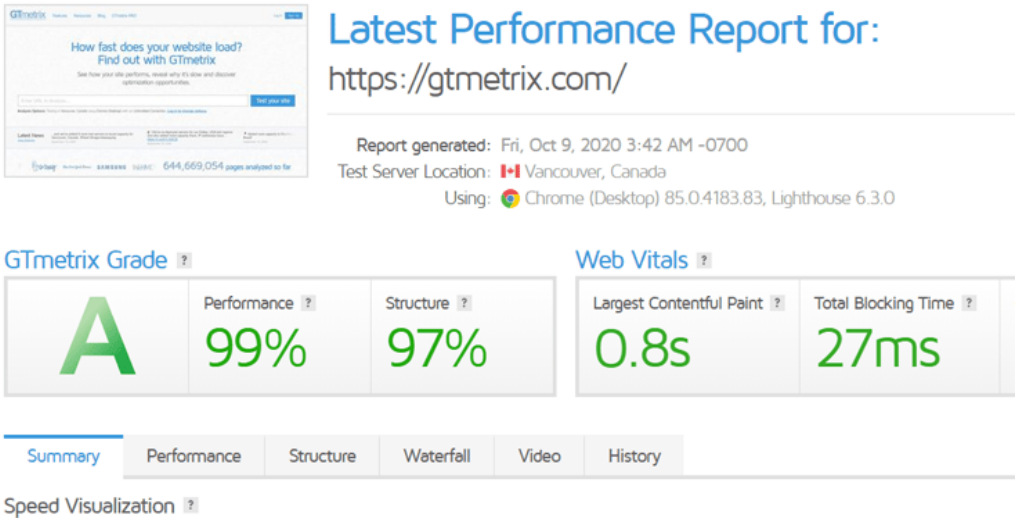

Рисунок 2.2 – Ілюстрація результатів аналізу швидкодії вебдодатку інструментом Gtmetrix

Джерело:[29]

WebPageTest дозволяє тестувати продуктивність сайту з різних місць по всьому світу, з використанням реальних браузерів (IE, Chrome, і т.д.) на реальних споживацьких швидкостях інтернету [13].

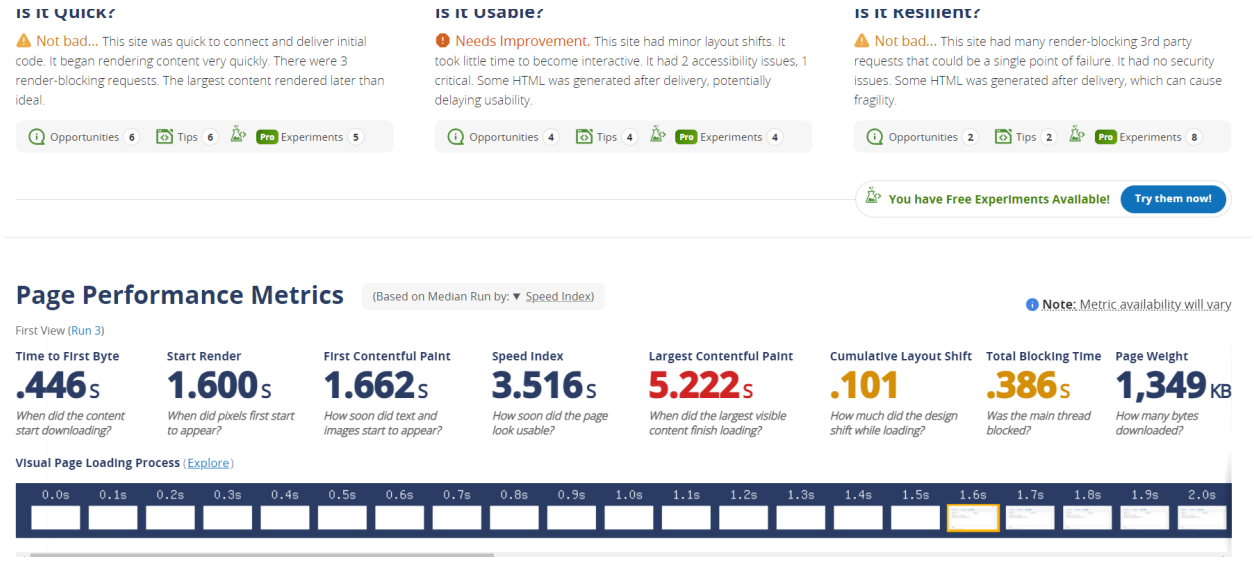

Рисунок 2.3 – Ілюстрація результатів аналізу швидкодії вебдодатку інструментом WebPageTest

Джерело:[34]

2.3 Методи оптимізації швидкодії вебсайту

Алгоритми завантаження сторінки сайту визначають як і в якому порядку ресурси вебсторінки завантажуються та відображаються у веб-браузері користувача. Оптимізація цих процесів може значно покращити час завантаження та взаємодію з сайтом. Розглянемо основні методи оптимізації:

1) Веб-кешування

«Веб-кешування» або «HTTP кешування» – це метод, що дозволяє тимчасово зберігати копії вебдокументів та медіа файлів для мінімізації затримок у відповідях сервера. Ця технологія працює шляхом зберігання версій документів, які були запитані, дозволяючи обслуговувати майбутні запити даними безпосередньо з кешу, що значно підвищує швидкість завантаження сторінок (рис. 2.4).

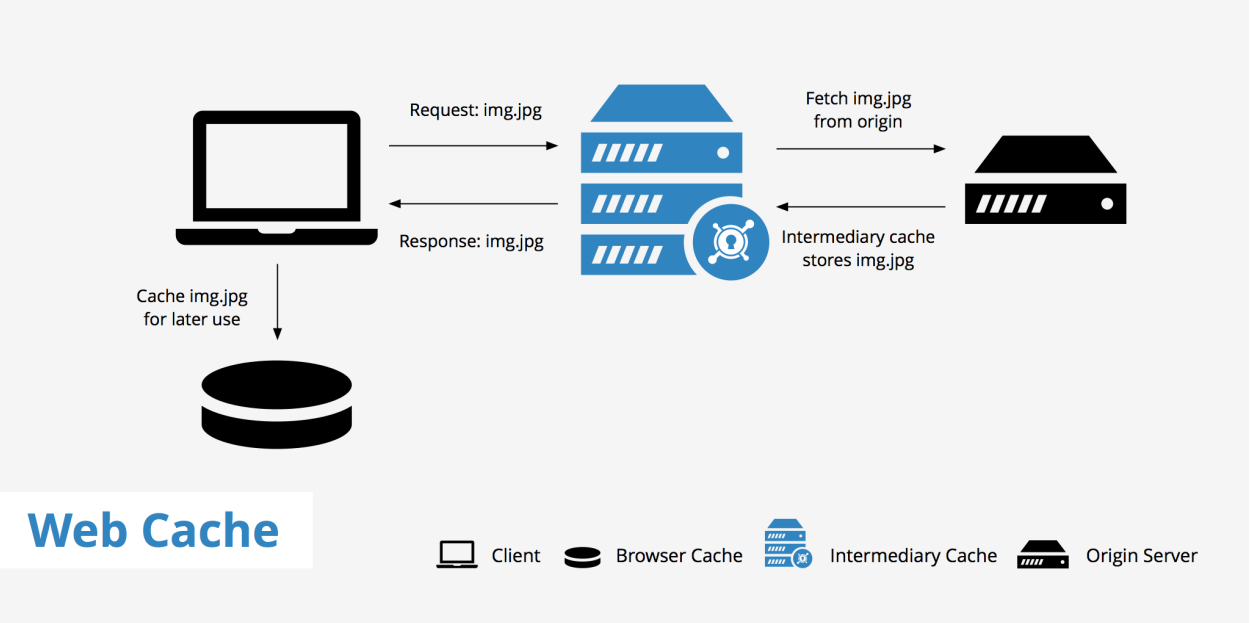

Рисунок 2.4 – Схема роботи веб-кешування

Джерело:[29]

Головна вигода від кешування полягає у прискоренні завантаження вебсайтів, що, у свою чергу, підвищує загальний рівень задоволеності користувачів. Це веде до того, що користувачі позитивніше сприймають роботу додатку [18].

Динамічний вміст вебсайтів – це такий вміст, який варіюється та адаптується згідно з конкретним користувачем чи обставинами, що змінюються з часом. Він може відрізнятися залежно від індивідуальних характеристик користувача або групи користувачів, як-от час відвідування сайту, географічне розташування, тип пристрою, вік, стать, а також історія перегляду вебдодатку. Цей тип контенту також відомий як персоналізований. З іншого боку, вміст, що змінюється з часом, може включати оновлення новин, блогів, переліків продукції в електронних магазинах та інше, що регулярно додається, оновлюється або вилучається [30].

Традиційно динамічний вміст не підлягав кешуванню через його змінну природу. Однак, завдяки прогресу у веб-технологіях, зараз існує можливість кешувати динамічний вміст, значно знижуючи затримки у передачі даних та зберігаючи інтерактивність з користувачем. Кешування динамічного вмісту полягає у збереженні вже обробленої і відрендереної вебсторінки. Проте існують дані, які не

підлягають кешуванню: інформація, що містить персональні дані користувача, стан особистих фінансових рахунків, а також дані про взаємодію з вебдодатком. Прикладами сторінок, що містять такий вміст, можуть бути особистий кабінет користувача або історія замовлень в інтернет-магазині.

2) Critical Rendering Path

Critical Rendering Path – це послідовність кроків, які браузер виконує для перетворення HTML, CSS та JavaScript в пікселі на екрані. Оптимізація цього процесу може значно покращити швидкість завантаження сторінки та час до першого відображення контенту (First Contentful Paint). Для ефективної роботи цього алгоритму завантаження потрібно визначити найважливіші ресурси, які в першу чергу повинен побачити користувач, а ресурси з нижчим пріоритетом завантажити в останню чергу (рис. 2.5).

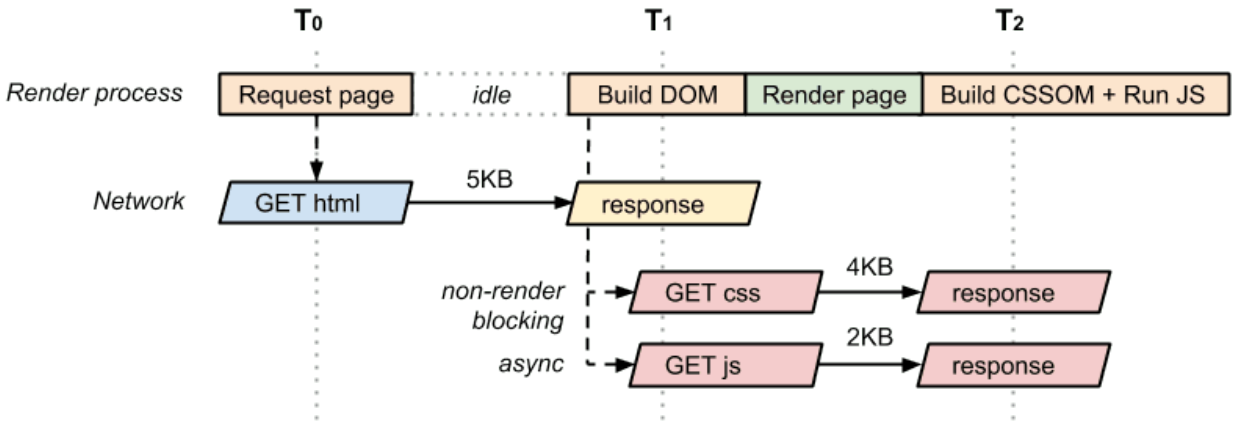

Рисунок 2.5 – Схема роботи Critical Rendering Path

Джерело: [5]

Для визначення порядку завантаження сторінки в даному алгоритмі використовується так зване «Дерево рендерингу». Дерево рендеринг – це комбінація вмісту та стилів, що складається з дерев DOM і CSSOM, об’єднаних разом. Під час формування дерева рендерингу, браузер аналізує кожен елемент, починаючи з кореня DOM, і визначає, до яких елементів застосовуються CSS-правила.

Важливою особливістю дерева рендерингу є те, що воно відтворює тільки елементи, які є видимими на сторінці. Наприклад, елементи зі стилем display: none; не будуть включені до дерева рендерингу, так само як і їхні дочірні елементи. Це означає, що елементи, які не мають візуального відображення на сторінці, наприклад заголовки сторінок (які зазвичай не відображаються візуально), також не будуть частиною дерева рендерингу [21].

3) Lazy Loading

Lazy Loading – це техніка, що дозволяє динамічно завантажувати та відображати вміст лише тоді, коли користувач прокручує сторінку. Цей метод покращує користувацький досвід, оскільки зображення та інший вміст завантажуються тільки тоді, коли вони потрібні. Однак, впровадження Lazy Loading у вебдодатках може бути викликом через відсутність вбудованої підтримки та необхідність враховувати різноманітні технічні аспекти, як-от обробку багатопоточності, HTTP-запити, управління пам’яттю та кешування. Незважаючи на свої переваги, такі як підвищення продуктивності та покращення загального досвіду користувачів, реалізація лінивого завантаження може ускладнити виконання деяких оптимізацій з боку оптимізуючих компіляторів, які не можуть передбачити повну структуру програми (рис. 2.6) [35].

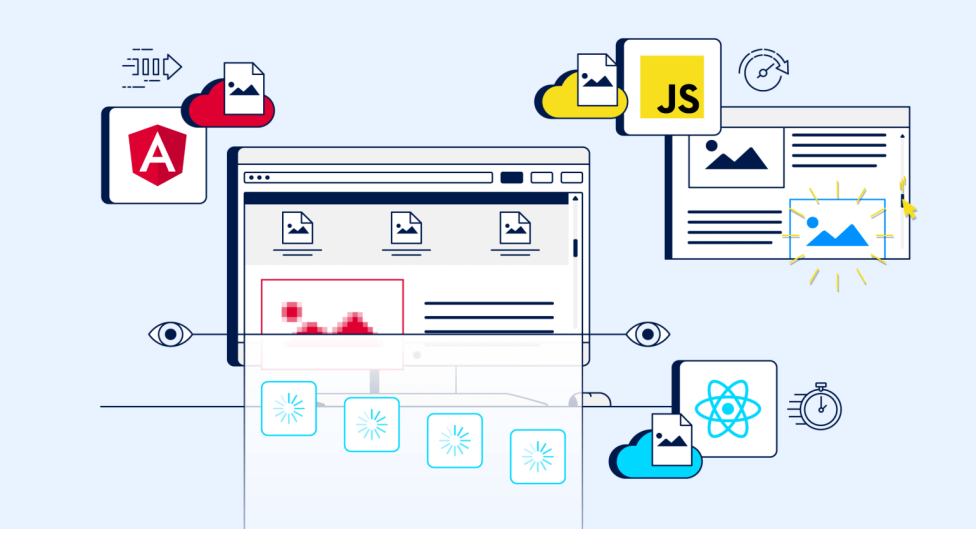

Рисунок 2.6 – Зображення роботи Lazy Loading

Джерело: [16]

Одним з способів реалізації даного алгоритму є поділ коду. Поділ коду дозволяє розбити JavaScript, CSS, і HTML код на менші фрагменти, звані чанками. Цей підхід дозволяє при початковому завантаженні вебдодатку відправляти лише ті частини коду, які безпосередньо необхідні для відображення стартової сторінки, наприклад, базову структуру розмітки, замість повного набору коду. Додаткові дані, необхідні для доповнення структури, можуть бути завантажені пізніше, використовуючи, наприклад, AJAX-запити. Це значно покращує швидкість завантаження та ефективність використання ресурсів, роблячи вебсайти швидшими та реактивнішими [7].

Також є можливість використовувати Intersection Observer API.

Intersection Observer API дозволяє асинхронно відстежувати, коли цільовий елемент перетинається з заданим предком або з вікном перегляду. Раніше визначення чи є елемент видимим на сторінці, або як взаємодіють між собою два елементи було складним і неефективним, часто призводячи до зниження продуктивності браузера та вебсайтів (рис. 2.7) [37, 40].

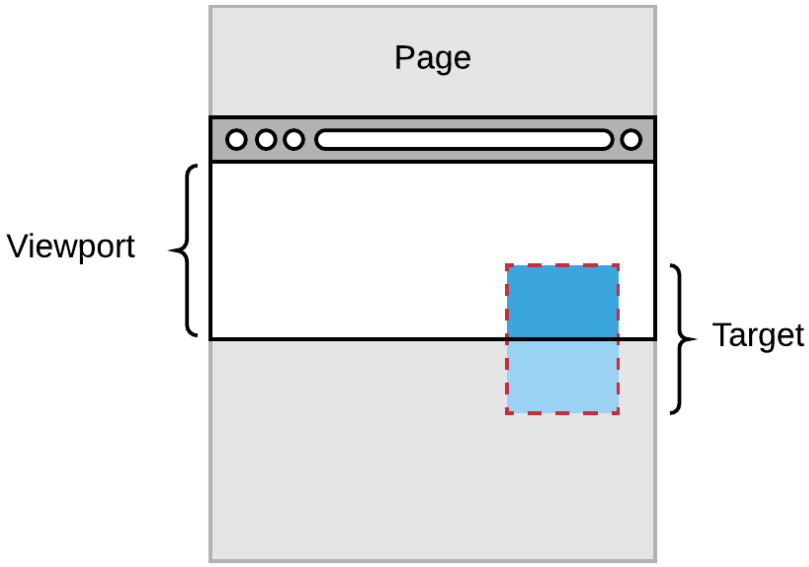

Рисунок 2.7 – Зображено як за допомогою Observer API відстежують наступний елемент на сторінці вебдодатку

Джерело: [12]

Однак, зі зростанням потреби у такій інформації в Інтернеті, поява цього API відкрила нові можливості для оптимізації вебдосвіду, включно з:

- Lazy Loading зображень та іншого контенту, що покращує швидкість завантаження сторінки під час прокрутки.
- Створенням сайтів з «нескінченною прокруткою», де новий контент завантажується автоматично, як тільки користувач доходить до певної точки, уникаючи потреби у вручну завантажених сторінках.
- Відстеженням видимості рекламних блоків для точного обчислення рекламних доходів.
- Активацією певних дій або анімацій лише тоді, коли користувач дійсно може їх побачити, збільшуючи ефективність і користувацький досвід [5].

4) HTTP/2 Server Push

Технологія Server Push дозволяє вебсайтіум «проштовхувати» ресурси до браузера відвідувача, не чекаючи на завантаження та аналіз HTML-документа браузером. Це може значно покращити швидкість завантаження сторінки та час її рендерингу. Використання цієї функції відбувається через додавання спеціального HTTP-заголовка «Link», який вказує на ресурси, що мають бути проштовхнуті. Для активації цієї можливості сайт може використовувати спеціальні плагіни для своєї системи управління контентом (CMS). Основними вимогами для ефективної роботи Server Push є використання сайтом HTTPS-протоколу та підтримка цієї технології браузером користувача [16].

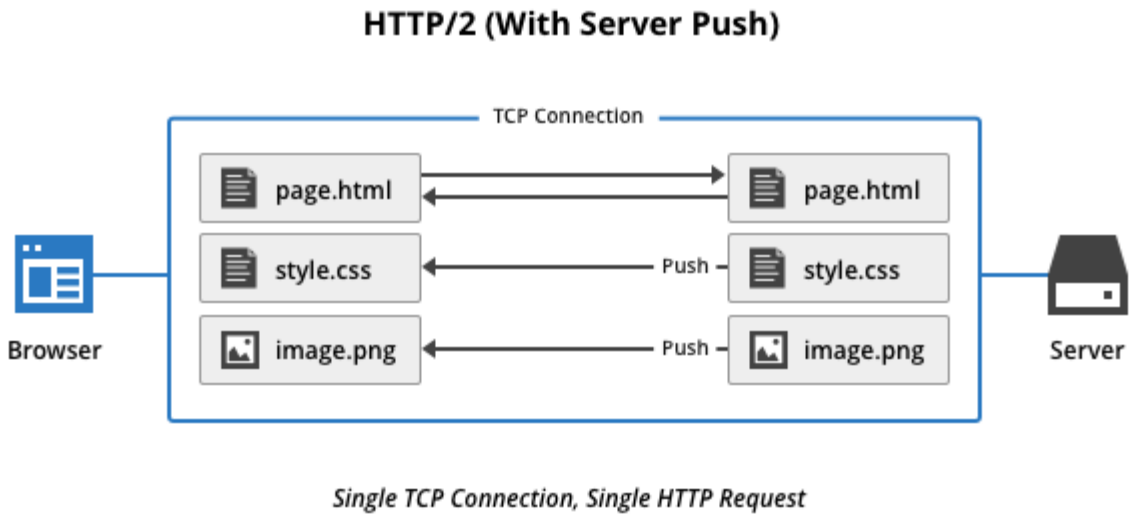

Рисунок 2.8 – Зображено які ресурси проштовхуються HTTP/2 Server Push *Джерело:[14]*

Для поліпшення продуктивності вебсайтів використовуються різноманітні методи оптимізації, які забезпечують швидше завантаження сторінок і кращий досвід для користувачів. В таблиці 2.1 розглянуто методи веб-оптимізації та зазначені їх основні переваги та обмеження.

Таблиця 2.1 – Порівняння методів веб-оптимізації

Назва методу	Опис	Переваги	Обмеження
Веб-кешування	Зберігання локально вебдокументів для більш швидкого доступу	Зменшує навантаження сервера	Неефективний для динамічного вмісту

Продовження таблиці 2.1

Critical Rendering Path	Послідовність кроків для рендерингу	Покращує час відображення	Вимагає визначення пріоритетів
Lazy Loading	Завантаження контенту при прокрутці	Зменшує початкове навантаження на сторінку	Складна реалізація, оптимізація
HTTP/2 Server Push	Проштовхування ресурсів до браузера	Оптимізує загрузку ресурсів	Вимагає HTTPS, не підтримується всіма браузерами

Джерело: розроблено автором

Згідно розглянутих методів було вирішено оптимізувати швидкість завантаження вебдодатку за допомогою методу веб-кешування.

2.4 Вибір методу реалізації

Кеш – швидкісний шар пам’яті, спеціалізований на тимчасовому зберіганні вибіркового набору даних. Це технічне рішення дозволяє забезпечити швидшу відповідь на запити користувачів або систем, знижуючи час, необхідний для доступу до даних, збережених у первинних, більш повільних джерелах зберігання. Ця прискорена обробка запитів досягається за рахунок використання компонентів швидкого доступу, таких як оперативна пам’ять, що дозволяє ефективно повторно використовувати інформацію, яка раніше була отримана або обчислена.

Основна мета застосування кешування – це підвищення продуктивності системи шляхом мінімізації звернень до первинного джерела зберігання даних, що значно повільніше. Відмінно від традиційних баз даних, де інформація зберігається на довготривалу перспективу та в повному обсязі, кеш зберігає лише частковий набір даних і зазвичай робить це на тимчасовій основі (рис. 2.9) [24].

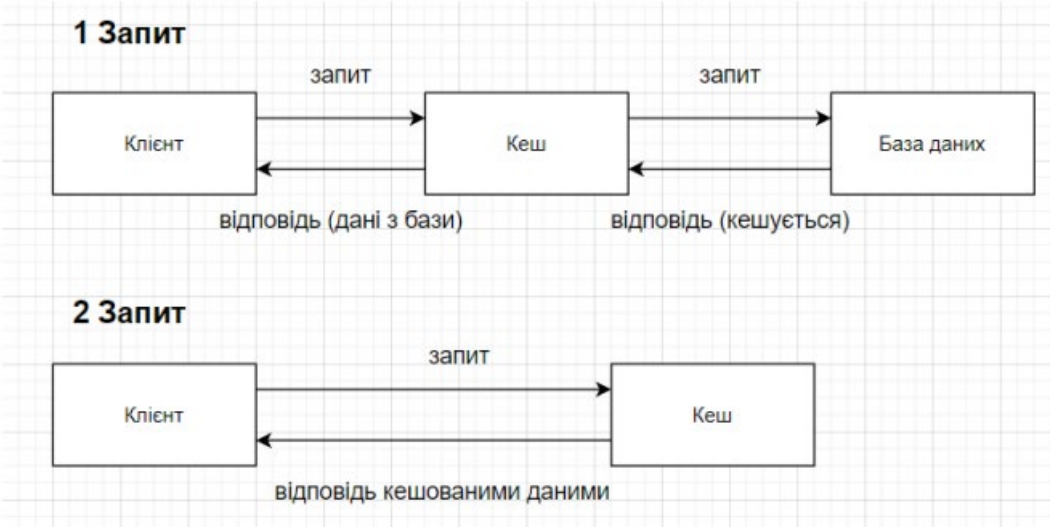

Рисунок 2.9 – Процес звернення до одних і тих же даних з використанням методу кешування

Джерело: розроблено автором

Такий підхід дозволяє оптимізувати використання ресурсів системи та забезпечити більш високу швидкість доступу до найбільш часто запитуваної інформації.

Використання кешування пропонує ряд важливих переваг для покращення роботи програмних додатків і зниження вартості їхньої експлуатації:

1. Покращення продуктивності додатків – використання кешу, який розміщений в оперативній пам’яті, забезпечує значно швидший доступ до даних порівняно з читанням інформації з дискових накопичувачів, як-от HDD або SSD. Це здатне скоротити час реакції додатка до долі мілісекунд, що суттєво підвищує загальну продуктивність та поліпшує користувацький досвід.

2. Зниження вартості бази даних – кешування може забезпечити високу кількість операцій вводу-виводу за секунду (IOPS), що дозволяє зменшити залежність від великої кількості баз даних або дорогих рішень для зберігання даних. Це, в свою чергу, може призвести до значного зниження загальних витрат на інфраструктуру, особливо у випадках коли вартість сервісу баз даних залежить від пропускної спроможності.

3. Передбачувана продуктивність – під час пікових навантажень, сучасні додатки часто стикаються з викликами, пов’язаними з піковими навантаженнями, наприклад під час великих рекламних кампаній або спеціальних подій. Такі піки можуть призводити до збільшення затримок або навіть відмов у обслуговуванні через перевантаження баз даних. Використання кешування допомагає уникнути цих проблем, забезпечуючи стабільну і передбачувану продуктивність незалежно від обсягу запитів, що сприяє збереженню високої доступності та швидкості обробки даних [22].

Ці переваги роблять кешування важливим елементом в архітектурі сучасних високопродуктивних систем, забезпечуючи їм переваги в швидкості, масштабованості та економічності.

Проте самого кешування буде недостатньо, тому для кращої швидкодії поєднаємо його у парі з BLOB. BLOB (Binary Large Object) – це тип даних, який використовується для зберігання великих об’єктів бінарних даних, таких як зображення, аудіо або відео файли, у базі даних. У веб-розробці BLOB використовується для зберігання та управління різноманітними динамічними або особистими даними, такими як профільні зображення користувачів, файли завантажень, контент сторінок тощо. Веб-розробники можуть використовувати BLOB для зберігання файлів безпосередньо в базі даних, що спрощує управління ресурсами та дозволяє отримати доступ до них за допомогою мов програмування або SQL-запитів. Це дозволяє створювати вебдодатки з розширеним функціоналом, таким як завантаження та зберігання файлів, робота з медіа контентом та іншими

ресурсами, безпосередньо у базі даних, що сприяє зручності та ефективності в розробці та управлінні додатками.

Запропонований авторами метод поєднує кешування і запис даних до BLOB (Binary Large Object), створюючи ефективний механізм для зберігання та отримання ресурсів вебдодатку. Використання кешування дозволить зберігати копії часто використовуваних ресурсів, таких як зображення, стилі або сценарії, у локальній пам’яті клієнта. Одночасно запис до BLOB дозволить зберігати динамічно генерований або особистий контент, такий як зображення користувачів чи індивідуалізовані налаштування. Це поєднання дозволить оптимізувати доступ до ресурсів, забезпечуючи швидкий доступ до статичного контенту через кешування і гнучкий зберігання та управління динамічними даними за допомогою BLOB. Такий підхід покращить продуктивність вебсайту, зменшить час завантаження сторінок та споживання трафіку даних, а також забезпечить більш ефективне управління ресурсами та особистим контентом користувачів. [6]

3 МОДЕЛЮВАННЯ Й ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ

3.1 Структурно-функціональне проєктування

На початковому етапі створення розроблення інформаційної технології важливою задачею є розробка контекстної діаграми IDEF0 для відображення основного процесу, покладеного в основу роботи алгоритму технології [19, 23]. Контекстна діаграма (рис.3.1) IDEF0 містить один основний функціональний блок, елементи вводу та виводу даних, компоненти управління, а також засоби, що використовуються, і була сконструйована за допомогою програмного забезпечення BPWin [39].

В процесі створення структури інформаційної технології були сформовані наступні дані:

1) вхідні дані: запит на отримання даних від серверу, отримання ресурсів;

2) вихідні дані: ініціалізовані дані, збережені дані до кешу, відображені дані з кешу;

3) контроль: config.json, об’єм кешу;

4) механізми реалізації: IndexDB API, Fetch API, сервер.

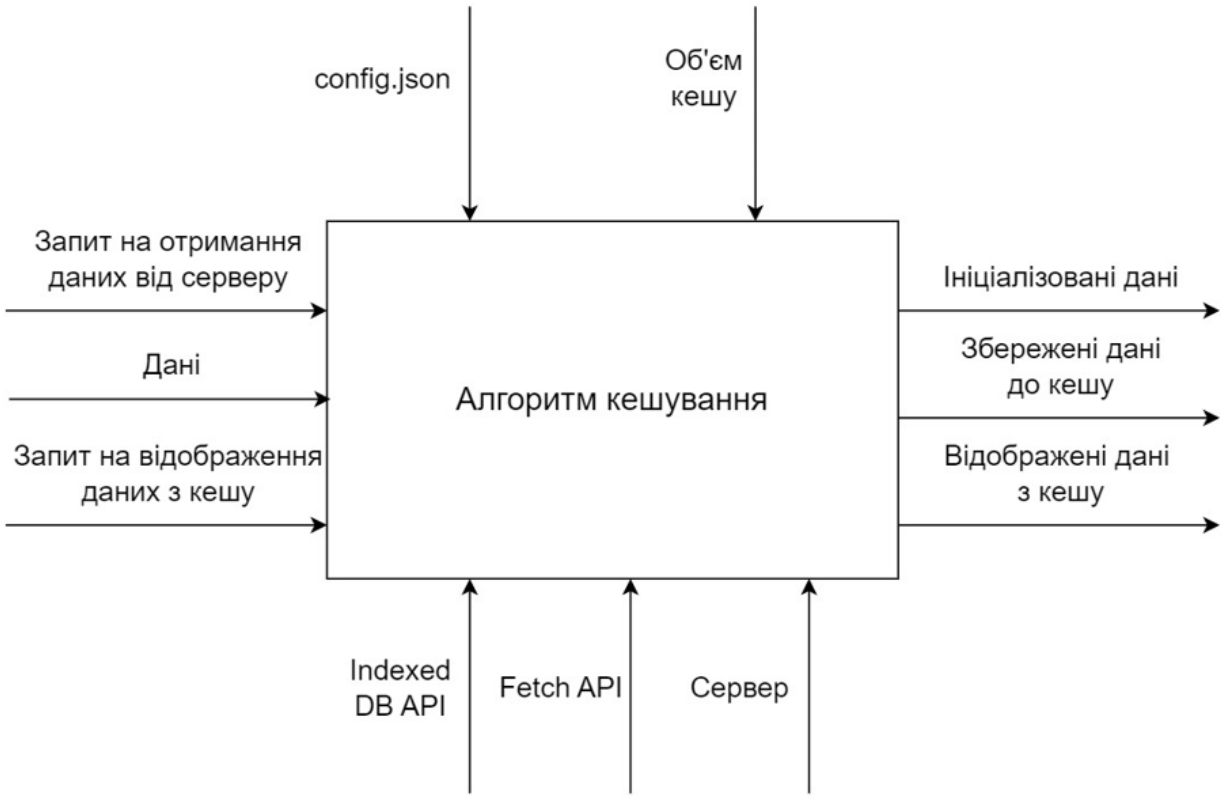

Рисунок 3.1 – Контекстна діаграма

Джерело: розроблено автором

Після загального опису системи наступний крок полягає в детальному аналізі її компонентів через функціональну декомпозицію. В контексті декомпонованої інформаційної технології, представленої на рисунку 3.2, аналіз виявляє три основні підпроцеси алгоритму технології оптимізації швидкодії вебсайтів:

1. Open Database.
2. Check and Load Images.
3. Cache Images.
4. Load Images from Cache.

Кожен з цих підпроцесів є важливою складовою у загальній структурі вебдодатку, дозволяючи створити організовану та орієнтовану на користувача інформаційну технологію, яка забезпечує ефективне управління та доступ до контенту.

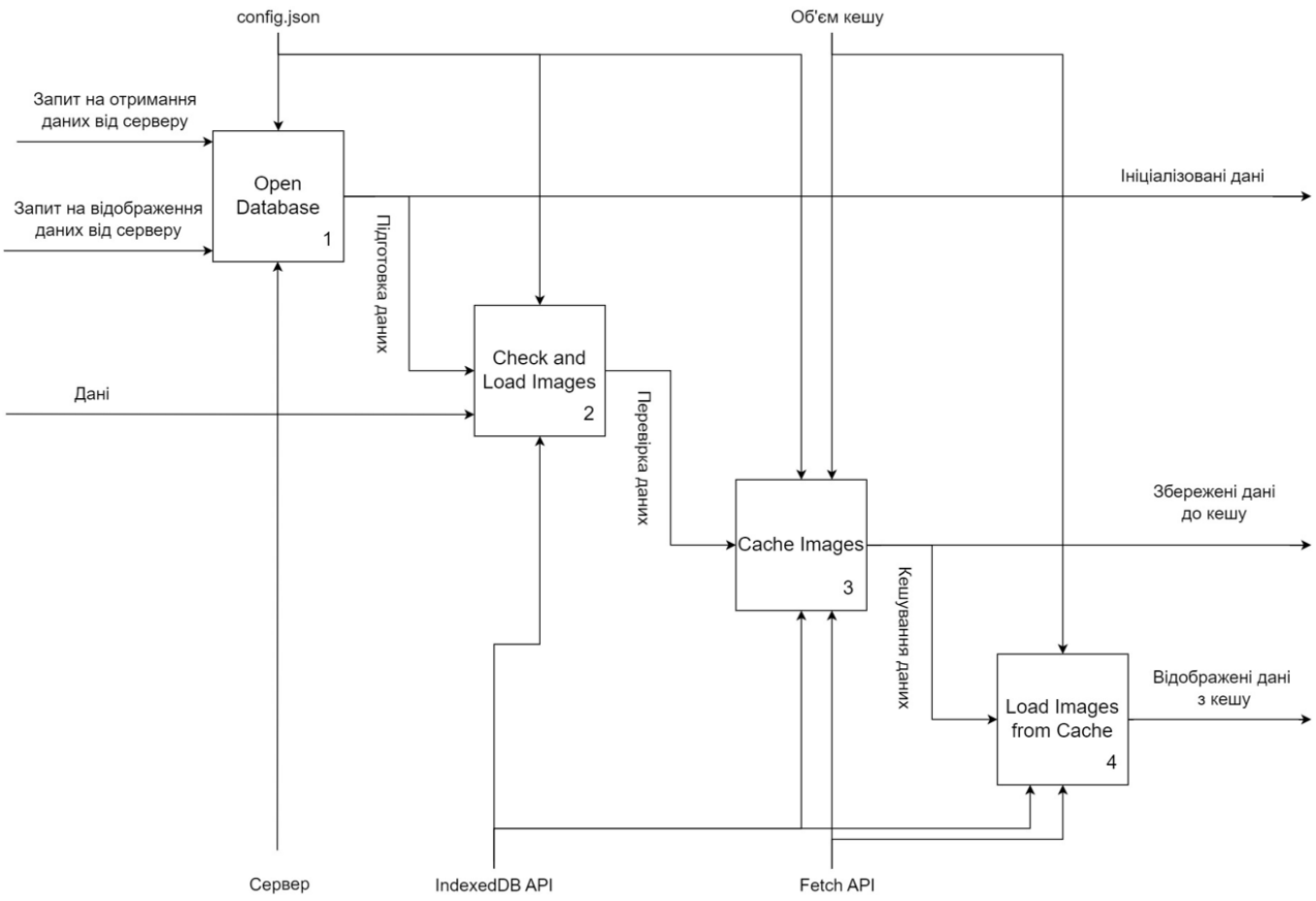

Рисунок 3.2 – Декомпозиція контекстної діаграми

Джерело: розроблено автором

3.2 Моделювання варіантів використання

Unified Modeling Language (UML) є уніфікованою мовою моделювання, яка використовується для створення графічних моделей об’єктів. Термін «unified» відображає універсальність мови, адже вона підходить для широкого спектру програмних систем, різноманітних сфер застосування, організацій різного розміру і професіоналів з різним рівнем навичок. UML дозволяє описувати системи за допомогою єдиному заданому синтаксисі. Це означає, що діаграми, створені на основі UML, будуть зрозумілі будь-якому фахівцю, який знайомий з цією мовою [41].

Як будь-яка інша мова, UML має власні правила оформлення моделей і синтаксис. За допомогою графічної нотації UML можна візуалізувати систему, об’єднати всі компоненти в єдину структуру, уточнювати і покращувати модель у процесі роботи. На загальному рівні графічна нотація UML містить 4 основні типи елементів:

1. фігури;
2. лінії;
3. значки;
4. написи.

Діаграма прецедентів на рис. 3.3 візуалізує взаємодію користувача з алгоритмом кешування.

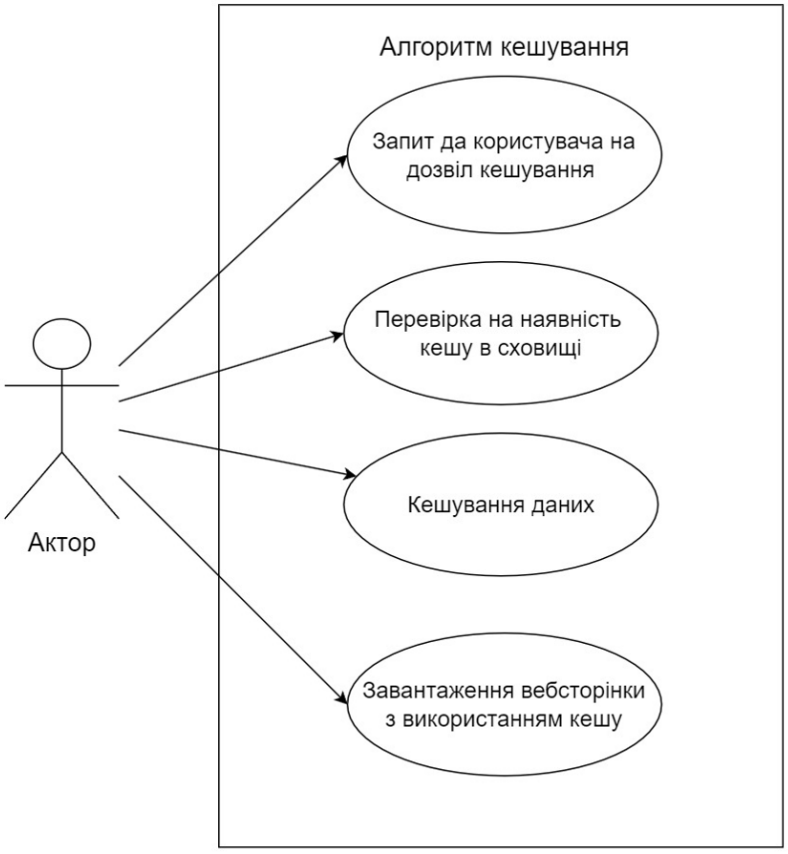

Рисунок 3.3 – Діаграма прецедентів

Джерело: розроблено автором

Актор на діаграмі представляє користувача, який ініціює процеси в алгоритмі.

Розглянемо кожен прецедент більш детально:

1) «Запит до користувача на дозвіл кешування» описує випадок, коли система звертається до користувача з проханням отримати дозвіл на збереження даних у кеші. Це гарантує згоду користувача на кешування інформації.

2) «Перевірка на наявність кешу в сховищі» – система, що перевіряє чи потрібні дані вже збережені у кеші, щоб уникнути зайвого запиту до форматування даних.

3) «Форматування даних та зображення в файли BLOB» вказує на процес перетворення даних до певного формату і їх збереження у вигляді BLOB (Binary Large Object), що є ефективним способом зберігання великих об’ємів даних у двійковий формат. [28]

4) «Кешування даних» відображає безпосереднє збереження даних у кеші після їх отримання та форматування, щоб прискорити майбутнє їх завантаження.

5) «Завантаження вебсторінки з використання кешу» завантаження сторінки з використанням вже збереженого кешу.

4 РОЗРОБКА ТА ВПРОВАДЖЕННЯ АЛГОРИТМУ АВТОМАТИЗОВАНОГО КЕШУВАННЯ

4.1 Концепція інформаційної технології

Розробка алгоритму кешування є актуальним завданням, особливо в сучасній інформаційній епосі, де швидкість доступу до інформації є критичним фактором для задоволення потреб користувачів. Кешування дозволяє значно зменшити час завантаження сторінок вебсайтів і відповідей на запити користувачів. Це особливо важливо для сайтів з великим обсягом відвідувань, де кожна секунда рахується для забезпечення задоволення користувачів. Для ефективної розробки нового алгоритму завантаження сторінки сайту, розроблено наступну структуру проекту (рис. 4.1):

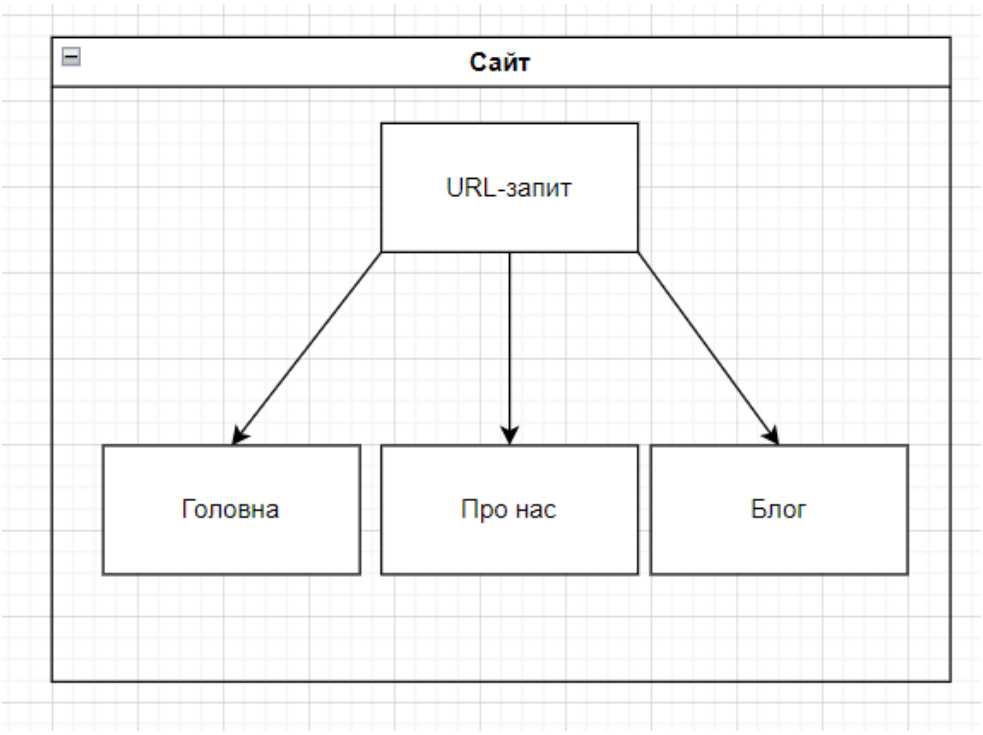

Рисунок 4.1 – Блок-схема проекту

Джерело: розроблено автором

Згідно даної структури процес кешування відбуватиметься для прискорення завантаження та тестування однієї з цих 3 сторінок. Якщо під час завантаження сторінку, кеш вже знаходиться в локальному сховищі, то алгоритм повинен завантажувати дані з локального сховища. [20]

З точки зору проекту можна виділити наступну структуру роботи алгоритму на веб сторінці для кешування даних (рис. 4.2):

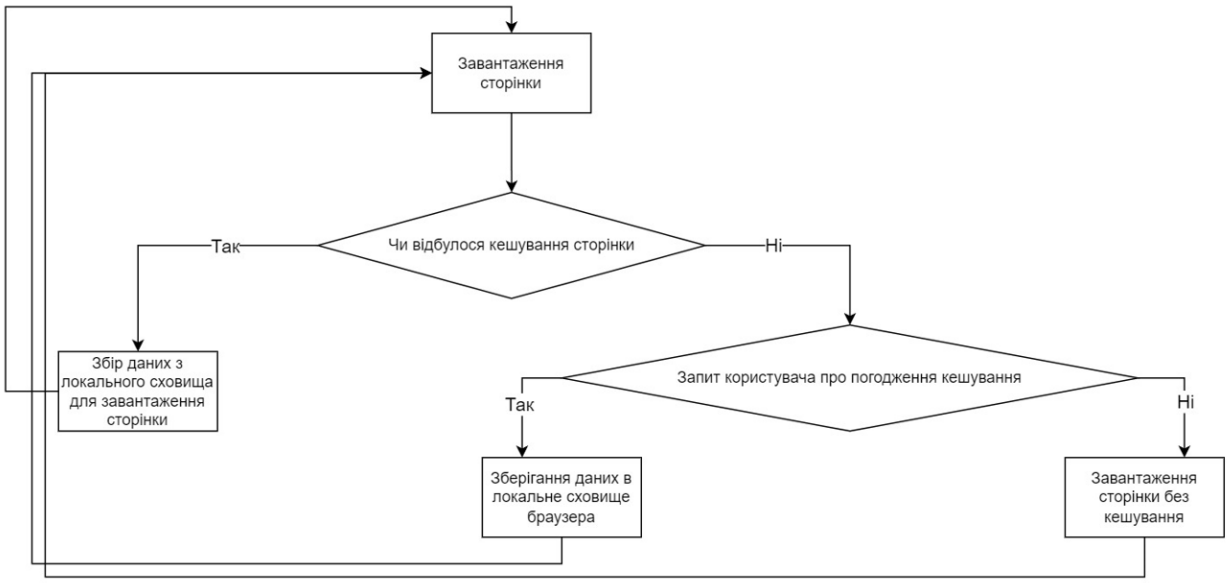

Рисунок 4.2 – Структура роботи алгоритму на вебсторінці

Джерело: розроблено автором

Для створення даної структури розробнику потрібно виконати наступні етапи реалізації:

1. Визначення вимог до вебсторінки
2. Розробка вебсторінки
3. Створення вебсерверу
4. Розробка кешування в локальному середовищі

4.2 Розроблення вебсайту для тестового впровадження інформаційної технології оптимізації швидкодії завантаження

Спочатку потрібно визначити структуру вебсайту та подальшого тестування алгоритмів. Всього основних сторінок буде 3: «Головна», «Про нас», та «Блог». Дані сторінки будуть наповнені великим об’ємом візуального контенту для подальшої перевірки алгоритмів завантаження. [18]

Далі потрібно розробити код для побудови веб сервера через Node.js [17] та встановити веб-адрес сайту та його дочірніх сторінок. Для виконання цих завдання створений наступний код index.js (рис. 4.8):

index.js > server > http.createServer() callback

```
const http = require('http');
const fs = require('fs');
const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/html; charset=utf-8');
  switch(req.url)
  {
    case '/':
      fs.createReadStream('./sites/main.html').pipe(res);
      break;
    case '/about':
      fs.createReadStream('./sites/about.html').pipe(res);
      break;
    case '/blog':
      fs.createReadStream('./sites/blogUnity.html').pipe(res);

}

});

server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});

```

Рисунок 4.8 – Код для створення вебсерверу

Джерело: розроблено автором

Цей код створює простий вебсервер за допомогою Node.js, який відповідає на HTTP запити і повертає HTML сторінки відповідно до запитаної URL адреси.

Спочатку завантажуються модулі **const http = require('http') та const fs = require('fs')**. Модуль http дозволяє Node.js передавати та приймати HTTP запити та відповіді. Fs модуль, який надає функціонал для роботи з файлами. Далі за допомогою наступних рядків: const hostname = '127.0.0.1'; і const port = 3000 визначають IP-адресу та порт, на яких сервер буде слухати запити.

Далі створюється http-сервер: const server = http.createServer((req, res) => {...}). Функція, передана у createServer, викликається кожного разу, коли сервер отримує новий запит. req (запит) і res (відповідь) - це об'єкти, що представляють вхідний запит від клієнта та вихідну відповідь від сервера, відповідно. Наступним кроком йде маршрутизація URL-запитів за допомогою конструкції switch/case:

- case '/':: Якщо URL є кореневим шляхом (/), сервер читає і відправляє файл main.html.
- case '/about':: Для шляху /about, сервер відправляє файл about.html.
- case '/blog':: Для шляху /blog, сервер відправляє файл blogUnity.html.

Далі запускається сам сервер: server.listen(port, hostname, () => { console.log(Server running at http://${hostname}:${port}/); });

Наступним кроком є розробка трьох веб сторінок сайту. Для початку потрібно розробити першу сторінку – **«Головну»**. Дана сторінка матиме значний візуальний контент та анімації для створення візуально приємного інтерфейсу (рис. 4.9).

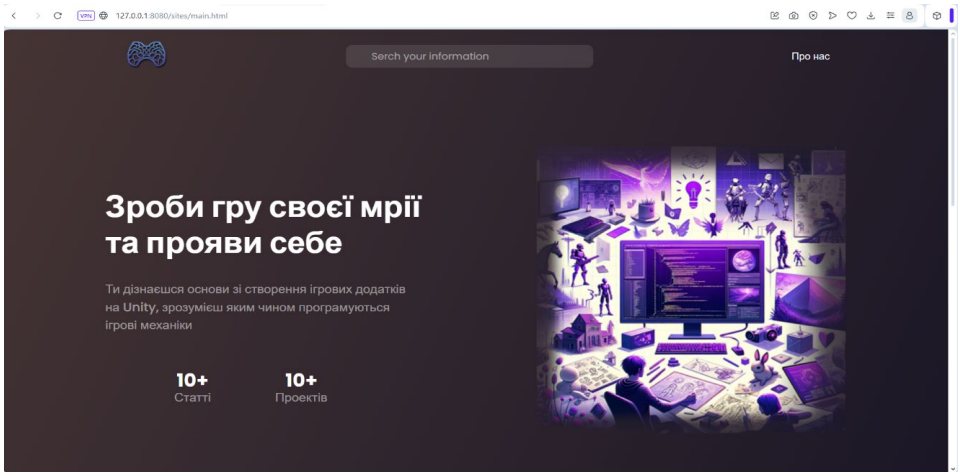

Рисунок 4.9 – Головна сторінка

Джерело: розроблено автором

Для побудови такої сторінку потрібно створити html-файл та підключити css стилі для розташування елементів сайту та встановлювати їм колір та шрифт. Для роботи стилів та скриптів, всередині тегу <head> потрібно підключити style.css (рис. 4.10).

```
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0"
    <title>Main site</title>
    <link rel="stylesheet" href="stiles/normalize.min.css">
    <link rel="stylesheet" href="stiles/style.css">
</head>
```

Рисунок 4.10 – Підключення стилів до сторінки

Джерело: розроблено автором

CSS (Cascading Style Sheets) — це мова стилів, яка використовується для задання візуального форматування вебсторінок, написаних на HTML або XML (включно з мовами, базованими на XML, як SVG або XHTML). CSS дозволяє веб-розробникам і дизайнерам контролювати стиль, розміщення, шрифти, кольори, та інші аспекти візуального представлення вебсторінки. Оформлення стилів відбуватиметься в загальному випадку через клас «.classname». Для простоти адаптування основного вмісту сайту створено тег з класом container (рис. 4.11).

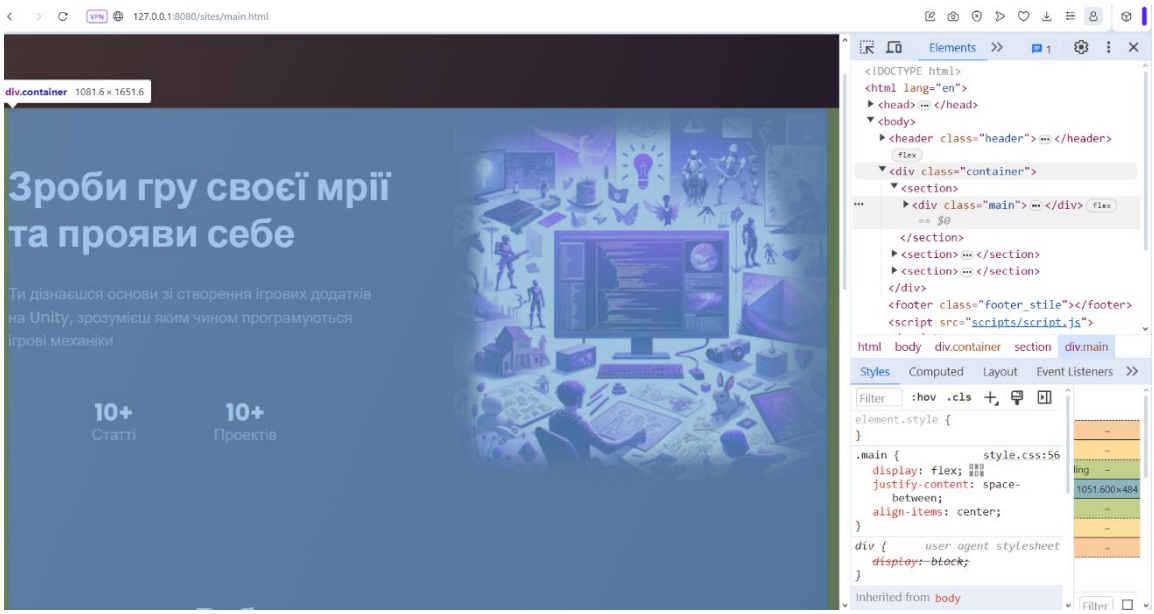

Рисунок 4.11 – Структура container

Джерело: розроблено автором

Клас container має наступні властивості (рис. 4.12):

- max-width: 1150px: властивість встановлює максимальну ширину елемента 1150 пікселів. Це допомагає забезпечити, що контент усередині контейнера не стане занадто широким на великих екранах, підтримуючи краще візуальне представлення та зручність читання.
- margin: 0 auto: відповідає за вирівнювання елемента по горизонталі. 0 вказує на відсутність вертикального відступу (margin)
- padding-left: 15px; і padding-right: 15px: задають відступи всередині елемента з лівого та правого боків відповідно.

```
.container{
    max-width: 1150px;
    margin: 0 auto;
    padding-left: 15px;
    padding-right: 15px;
}
```

Рисунок 4.12 – Властивості стилів класу «container»

Джерело: розроблено автором

Наступний блок з 3 візуальними елементами створюється за допомогою наступної розмітки html (рис 4.13):

```
<section>
<h2 class="title">Вибери статтю для навчання</h2>
<p class="desc">Для поновлення навичок є можливість розглянути різні статті</p>
<div class="card animate">
    <div class="card_item">
        <div class="card_cover">
            <img class="card_img" src="../sites/stiles/images/card_foto1.png" alt="">
        </div>
        <div class="card_info">
            <p class="card_main_inf">Введення в Unity</p>
            <p class="card_disc">@Unity</p>
        </div>
        <a href="blogUnity.html" class="card_link">Перейти</a>
    </div>
    <div class="card_item">
        <div class="card_cover">
            <img class="card_img" src="../sites/stiles/images/card_foto2.png" alt="">
        </div>
        <div class="card_info">
            <p class="card_main_inf">Базові функції в Unity</p>
            <p class="card_disc">@Unity</p>
        </div>
        <a href="blogUnity.html" class="card_link">Перейти</a>
```

Рисунок 4.13 – Блок контенту з вибором статті

Джерело: розроблено автором

Даний блок з трьох карток, які оформлюються тегом . Для кожної картки є своє фото, що додається тегом <img> та оформленим за допомогою стилів посиланням на окрему статтю. Загальний тег в свою чергу має важливі властивості для позиціонування (рис. 4.14):

- display: flex; встановлює контейнер на використання Flexbox, по одній осі.
- justify-content: space-between: визначає, як елементи всередині контейнера розподілятимуться вздовж головної осі (в цьому випадку горизонтальної, оскільки display: flex за замовчуванням встановлює горизонтальну ось як головну).

```
.card{
    display: flex;
    justify-content: space-between;
}
```

Рисунок 4.14 – Властивості класу card

Джерело: розроблено автором

Завдяки цим властивостям елементи розташовані горизонтально та рівномірно та відносно рівною відстанню між собою (рис. 4.15).

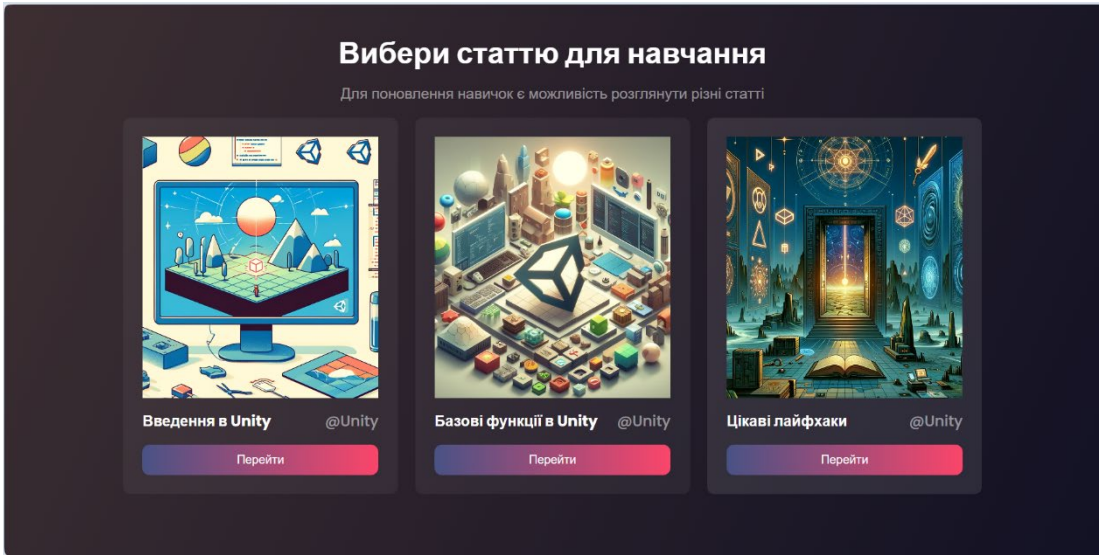

Рисунок 4.15 – Блок контенту, що відповідає за вибір статті

Джерело: розроблено автором

Всередині цих карток можна спостерігати кнопки при натисненні на яку, користувач перейде на сторінку блогу. Даний компонент реалізовується за допомогою тегу a href="blogUnity.html" class="card_link">Перейти. Тобто це просто клікабельне посилання але оформлене у вигляді стильної кнопки. Для встановлення такого кольору та форми на даний елемент застосовуються наступні стилі css (рис. 4.16):

```
.card_link{
    display: block;
    text-decoration: none;
    color: #fff;
    border-radius: 10px;
    background: linear-gradient(90deg, rgba(73,83,134,1) 0%, rgba(252,70,107,1) 100%);
    padding: 11px 20px;
    text-align: center;
}
```

Рисунок 4.16 – CSS властивості класу card_link

Джерело: розроблено автором

Для карток також застосовуються анімації: при наведенні колір плавно змінюється на сірий, при відведенні навпаки. Для таких анімацій також застосовуються стилі css. Для роботи анімації під час наведення тобто працювати при зміні положення курсору миші в напрямку анімованого об’єкту. Працює дана анімація за допомогою властивості **:hover** (рис. 4.17).

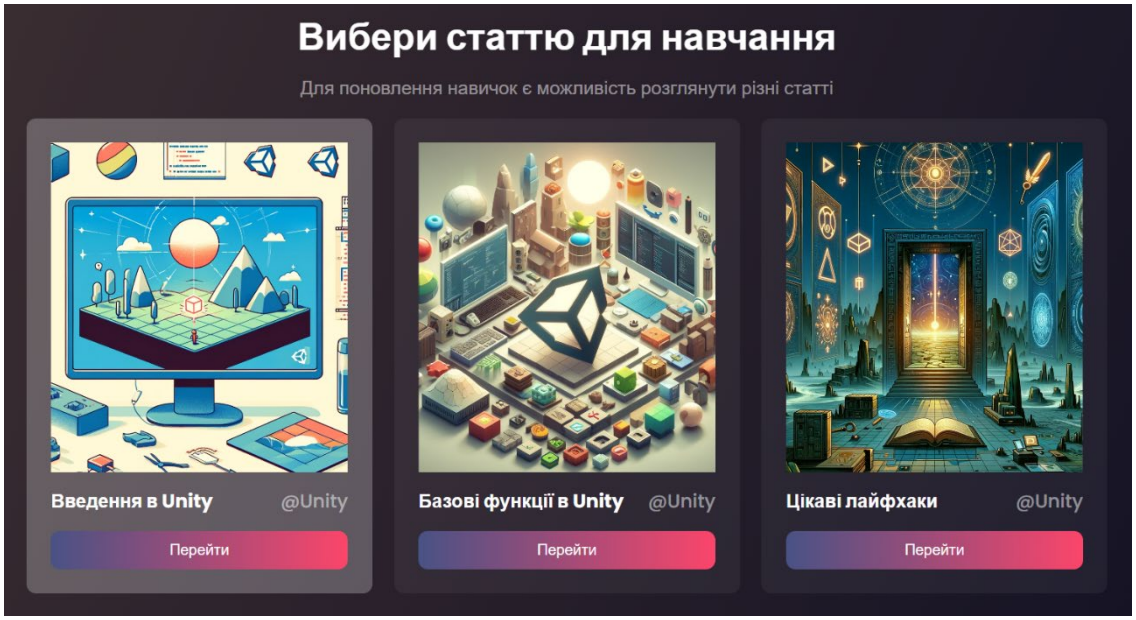

Рисунок 4.17 – Робота анімації при наведенні

Джерело: розроблено автором

Останнім контентним блоком є анімовані випадаючі списки, де можна додати корисні посилання для користувачів. Варто зазначити, що дані елементи вже анімовані з використанням як CSS так і JavaScript. [36] Скрипт потрібен для старту анімації відкривання та закривання контенту всередині випадаючого списку (рис. 4.18):

```
sites > scripts > JS script.js > ...
1   var coll = document.getElementsByClassName("collapsible");
2   var i;
3
4   for (i = 0; i < coll.length; i++) {
5     coll[i].addEventListener("click", function() {
6       this.classList.toggle("active");
7       var content = this.nextElementSibling;
8       if (content.style.maxHeight){
9         content.style.maxHeight = null;
10      } else {
11        content.style.maxHeight = content.scrollHeight + "px";
12      }
13    });
14  }
```

Рисунок 4.18 – Код для роботи анімації випадаючого списку

Джерело: розроблено автором

Звичайно, що для роботи скрипта потрібно додати відповідну розмітку обов'язково з тегом класу **collapsible**. Для оформлення використовуються звичайні тег та <button> (рис. 4.19):

```
<div class="link_list animate">
    <div class="link_item">
        <button type="button" class="collapsible">Корисні програми</button>
        <div class="content">
        <p>Lorem ipsum...</p>
        </div>
        <button type="button" class="collapsible">Статті та форуми</button>
        <div class="content">
        <p>Lorem ipsum...</p>
        </div>
    </div>
    <div class="link_item">
        <button type="button" class="collapsible">Корисні посилання та канали</button>
        <div class="content">
        <p>Lorem ipsum...</p>
        </div>
        <button type="button" class="collapsible">Документація</button>
        <div class="content">
        <p>Lorem ipsum...</p>
        </div>
    </div>
</div>
```

Рисунок 4.19 – Розмітка блоку «Корисні посилання»

Джерело: розроблено автором

Далі наступною сторінкою є «Блог», дана сторінка має на меті надати навчальну інформацію або якісь короткі відомості, тому дана сторінка має мінімальну кількість анімації. Загалом анімації відносяться тільки до меню, що розгортається (рис. 4.20).

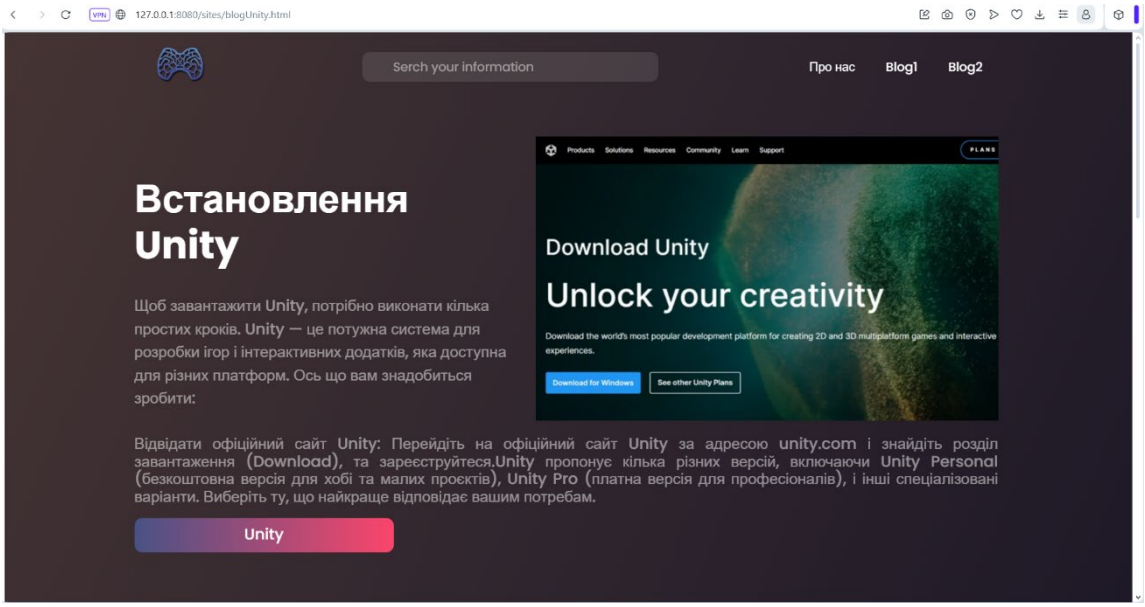

Рисунок 4.20 – Зовнішній вигляд сторінки «Блог»

Джерело: розроблено автором

HTML-розмітка має в свою чергу має схожу структуру але відрізняється від головної сторінки за контентом та відсутності анімації. Для спрощення роботи деякі стилі були запозичені з попередньої сторінки (рис. 4.21):

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Основні відомості</title>
    <link rel="stylesheet" href="stiles/normalize.min.css">
    <link rel="stylesheet" href="stiles/style.css">
    <link rel="stylesheet" href="stiles/blogStile.css">
    <script src="scripts/cacheScript.js"></script>
</head>
```

Рисунок 4.21 – Підключені стилі CSS до сторінки

Джерело: розроблено автором

Як і на попередній сторінці основний обгорнуто в клас container для спрощеного розміщення окремих блоків контенту. Для даної сторінки окремо передбачено анімоване меню навігації. Для розробки даного компоненту потрібно спочатку створити відповідну розмітку (рис. 4.22):

```
<div class="menu">
    <div class="blogtitle">
        <h1>Навігація</h1>
    </div>
    <div class="list_menu">
        <ul>
            <li><a href="#Installation">Встановлення Unity</a></li>
            <li><a href="#Registration">Реєстрація</a></li>
            <li><a href="#Download_Version">Завантаження версії</a></li>
            <li><a href="#views">Корисні посилання</a></li>
        </ul>
    </div>
</div>
```

Рисунок 4.22 – HTML-розмітка меню навігації

Джерело: розроблено автором

Далі за допомогою класу menu, потрібно надати певні властивості, щоб меню залишалося фіксованим та було доступним в будь-якому місті сторінки. Для цього за допомогою CSS потрібно додати наступні параметри (рис. 4.23):

```
.menu{
    padding: 10vh 50px;
    position: fixed;
    background: rgba(20, 20, 20, 0.5);
    height: 100vh;
    backdrop-filter: blur(16px);
    border-right: 7px solid #fff;
    left: -345px;
    transition: all 0.2s ease;
}
```

Рисунок 4.23 – Стилі для класу menu

Джерело: розроблено автором

Важливою властивістю потрібно відзначити position: fixed. Даний параметр визначає положення об’єкту на сторінці. При значенні fixed, елемент нікуди не зникає та залишається на тому самому місці, незалежно від місця сторінки яку переглядає користувач. Таким чином можна забезпечити меню постійною доступністю. За анімацію відкривання меню теж відповідає властивість **.menu:hover{left: 0; transition: all 0.2s ease;}** (рис. 4.24).

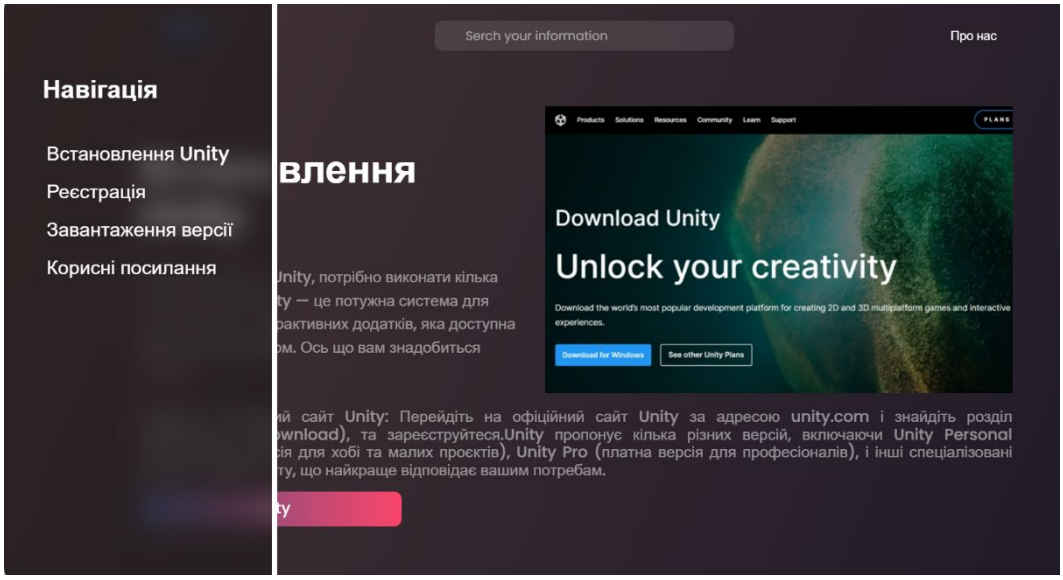

Рисунок 4.24 – Відкриття меню навігації

Джерело: розроблено автором

Загалом меню виконує функцію змісту сторінки сайту. Допомагає в навігації, шляхом перемиканням відразу на потрібну главу. Наступним кроком після реалізації «Блог» є сторінка «Про нас».

Сторінка «Про нас» має порівняно менший вміст, але відрізняється за принципами побудови анімації та компонентах (рис. 4.25).

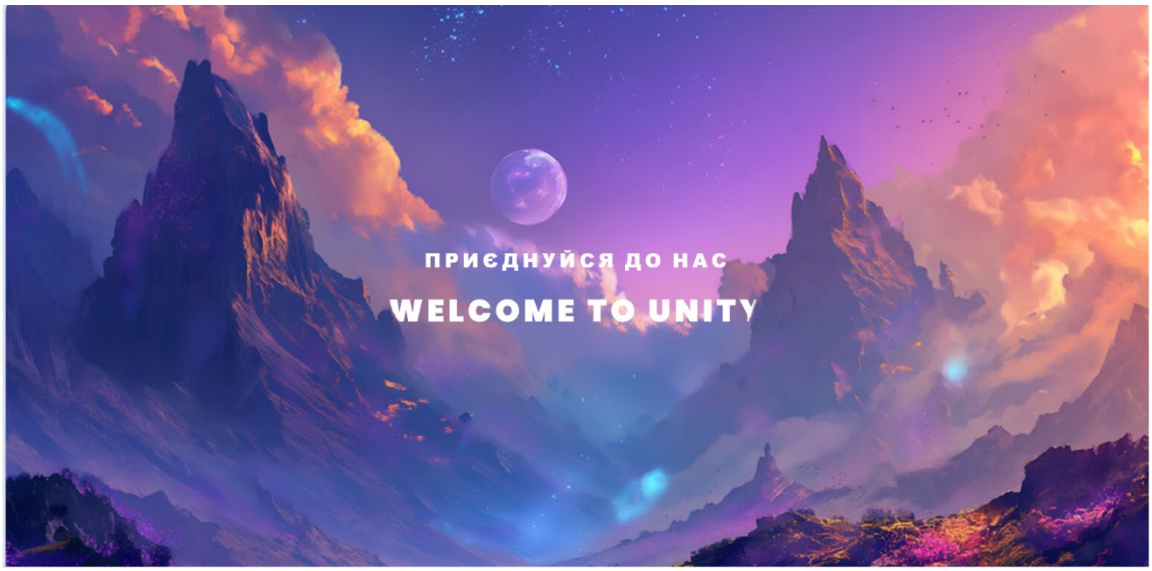

Рисунок 4.25 – Початок сторінки «Про нас»

Джерело: розроблено автором

Від початку сторінки анімація ефекту «Паралакс». Даний ефект надає можливість створити ефект об'єму при подальшому скролу користувачем сторінки. Цей ефект створюється за рахунок 3 шарів (переднього, середнього, заднього), які рухаються з різною швидкістю. Той шар, що ближче до користувача має відносно більшу швидкість порівняно з іншими шарами. Анімація працює за допомогою наступного коду anim.js (рис. 4.26):

```
sites > scripts > JS anim.js > ...
1   document.addEventListener("DOMContentLoaded", () => {
2
3       // Use Intersection Observer to determine if objects are within the viewport
4       const observer = new IntersectionObserver(entries => {
5         entries.forEach(entry => {
6           if (entry.isIntersecting) {
7             entry.target.classList.add('in-view');
8             return;
9           }
10          entry.target.classList.remove('in-view');
11        });
12      });
13
14      // Get all the elements with the .animate class applied
15      const allAnimatedElements = document.querySelectorAll('.animate');
16
17      // Add the observer to each of those elements
18      allAnimatedElements.forEach((element) => observer.observe(element));
19
20  });
```

Рисунок 4.26 – Скрипт anim.js

Джерело: розроблено автором

Під час скролу вниз користувач може побачити як різні шари зображення рухаються по різному, і створюється ефект «Паралакс» (рис. 4.27):

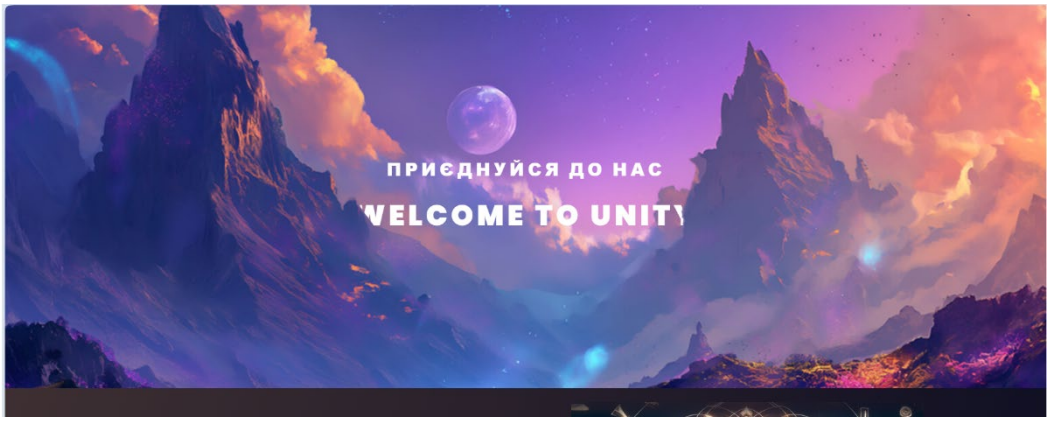

Рисунок 4.27 – Робота анімації «Паралакс»

Джерело: розроблено автором

Наступний вміст крім останнього компоненту має ідентичну структуру як і в попередніх сторінок це використання класу container (рис. 4.28).

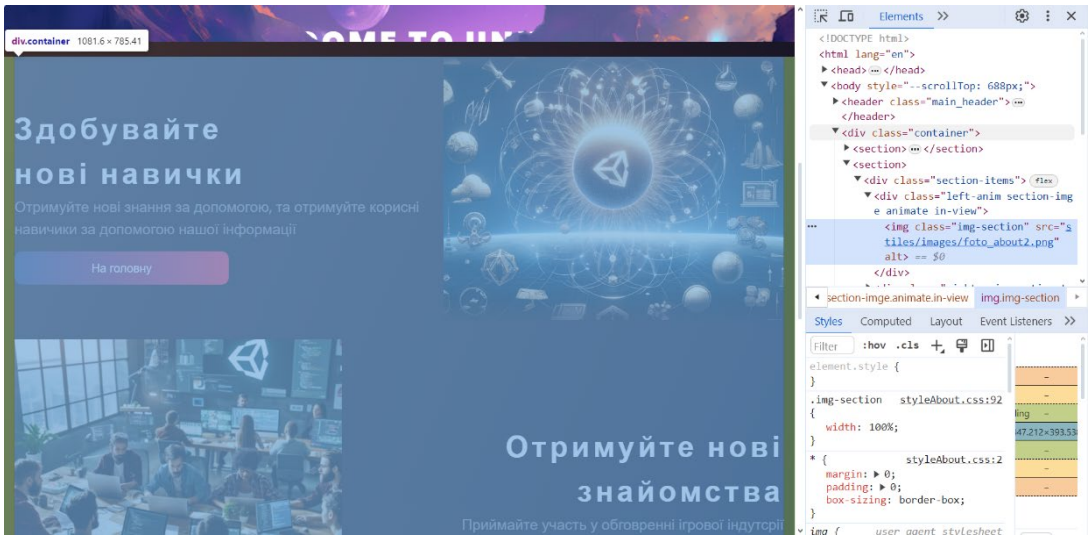

Рисунок 4.28 – Елементи, що належать класу container

Джерело: розроблено автором

Останнім компонентом є галерея відомих ігор зроблених на Unity. Даний компонент складається з декількох анімованих фото-карток. Крім цього для

створення такого компоненту використовувалися сторонні бібліотеки та стилі і в результаті отримано наступну розмітку (рис. 4.29):

```
<div class="swiper slider slider_main">
    <div class="swiper-wrapper slier__wrapper">
        <div class="swiper-slide slider__item">
            <div class="slider__img" data-swiper-parallax="20%" style="background-image: url(stiles/imagesGalary/journey.png);"></div>
        </div>
        <div class="swiper-slide slider__item">
            <div class="slider__img" data-swiper-parallax="30%" style="background-image: url(stiles/imagesGalary/2.png);"></div>
        </div>
        <div class="swiper-slide slider__item">
            <div class="slider__img" data-swiper-parallax="20%" style="background-image: url(stiles/imagesGalary/3.png);"></div>
        </div>
        <div class="swiper-slide slider__item">
            <div class="slider__img" data-swiper-parallax="30%" style="background-image: url(stiles/imagesGalary/4.png);"></div>
        </div>
        <div class="swiper-slide slider__item">
            <div class="slider__img" data-swiper-parallax="20%" style="background-image: url(stiles/imagesGalary/5.png);"></div>
        </div>
        <div class="swiper-slide slider__item">
            <div class="slider__img" data-swiper-parallax="30%" style="background-image: url(stiles/imagesGalary/6.png);"></div>
        </div>
        <div class="swiper-slide slider__item">
            <div class="slider__img" data-swiper-parallax="20%" style="background-image: url(stiles/imagesGalary/7.png);"></div>
        </div>
    </div>
```

Рисунок 4.29 – Розмітка компоненту «Галерея»

Джерело: розроблено автором

Також потрібно зазначити, що до сторінок створювалися додаткові анімація появлення, які реалізовуються за допомогою стилів css. Дані анімації створювалися за рахунок наступних властивостей (рис. 4.30):

```
@keyframes animText_see{
    from{
        color: transparent;
        transform: translateX(-500px);
    }
    to{
        transform: translateX(0px);
    }
}
@keyframes rightImage{
    from{
        opacity: 0;
        transform: translateX(+600px);
    }
    to{
        transform: translateX(0px);
    }
}
@keyframes cardAnim{
    from{
        transform: translateX(-1000px);
    }
    to{
        transform: translateX(0px);
    }
}
```

Рисунок 4.30 – CSS анімації для сайту

Джерело: розроблено автором

За допомогою @keyframes, можна створити анімації з допомогою яких можна плавно змінювати певні параметри вказуючи початковий стан та кінцевий.

4.3 Реалізація алгоритму

Після розробки сайту потрібно розробити алгоритми для їх завантаження. Перший алгоритм який буде розглянуто це цілеспрямоване кешування візуальних об’єктів. Принцип даного алгоритму ґрунтується на тому, що під час 1 завантаження сторінки, алгоритм створює кеш та зберігає його в пам’яті браузера. Після повторного перевантаження він завантажує дані вже з кешу, що повинно пришвидшувати процес

завантаження сторінки сайту. Для роботи алгоритму використовується скрипт cacheScript.js (рис. 4.31):

```
let db;
const request = window.indexedDB.open("imagesCacheDB", 1);

request.onerror = function(event) {
    console.error("Database error: ", event.target.error);
};

request.onupgradeneeded = function(event) {
    const db = event.target.result;
    db.createObjectStore("images", { keyPath: "url" });
};

request.onsuccess = function(event) {
    db = event.target.result;
    checkAndLoadImages();
};
```

Рисунок 4.31 – Початок коду алгоритму кешування

Джерело: розроблено автором

Цей код демонструє процес створення та використання IndexedDB в браузері для кешування зображень. IndexedDB — це низькорівнева API для клієнтського зберігання великих обсягів структурованих даних, включно з можливістю виконання запитів до цих даних.

Спочатку створюється змінна db для збереження посилання на базу даних. Далі викликається window.indexedDB.open(«imagesCacheDB», де «imagesCacheDB» — це назва бази даних. Цей метод відкриває з’єднання з базою даних, або створює нову базу даних, якщо вона в даний момент не існує.

Request.onerror = function(event) {…}: функція визначається для обробки помилок під час відкриття бази даних. Вона викликається, якщо IndexedDB не може успішно відкрити або оновити базу даних, і виводить повідомлення про помилку в консоль.

Db.createObjectStore(«images», { keyPath: «url» }); створює об’єктне сховище (таблицю) з назвою «images», де ключем є url. Це використовується для унікальної ідентифікації кожного запису в сховищі.

Наступним кроком є виведення вікна для ввімкнення користувачем алгоритму (рис. 4.32):

```
function checkAndLoadImages() {
    const transaction = db.transaction("images", "readonly");
    const store = transaction.objectStore("images");
    const countRequest = store.count();

    countRequest.onsuccess = function() {
        if (countRequest.result === 0) {
            // Кеш пустий, питаємо про кешування
            var cacheEnabled = confirm("Включити кешування? Натисніть OK для включення або Скасувати для вимкнення.");
            if (cacheEnabled) {
                cacheImages();
            }
        } else {
            // Кеш не пустий, завантажуємо зображення з кешу
            loadImagesFromCache();
        }
    };
}
```

Рисунок 4.32 – Виведення випливаючого вікна

Джерело: розроблено автором

Функція checkAndLoadImages перевіряє наявність зображень у кеші бази даних IndexedDB і вирішує, чи потрібно завантажувати зображення з кешу або кешувати нові зображення. Вона починається зі створення транзакції для доступу до об’єктного сховища images у режимі "readonly", що означає, що дані можуть тільки зчитуватися, але не модифікуватися.

У рамках цієї транзакції вона створює змінну store, яка відповідає за доступ до сховища зображень. Далі, функція виконує запит на підрахунок кількості записів у цьому сховищі за допомогою методу count(). Результат цього запиту обробляється у події onsuccess, яка визначає, чи є зображення в кеші. Якщо результат підрахунку дорівнює нулю, це означає, що кеш пустий. У цьому випадку користувачеві пропонується активувати кешування через діалогове вікно confirm, яке дозволяє вибрати, чи включити кешування зображень. Якщо користувач погоджується, викликається функція cacheImages(), яка має кешувати нові зображення.

З іншого боку, якщо в кеші вже є зображення (результат підрахунку більший за нуль), виконується функція loadImagesFromCache(), яка відповідає за завантаження зображень безпосередньо з кешу. Це покращує швидкість завантаження сторінки, оскільки зображення не потрібно завантажувати з інтернету.

Наступна функція – це функція кешування (рис. 4.33).

```
function cacheImages() {
    document.querySelectorAll('img').forEach(img => {
        const src = img.getAttribute('src');
        fetch(src).then(response => {
            if (response.ok) {
                return response.blob();
            }
            throw new Error('Network response was not ok.');
        })
        .then(blob => {
            const tx = db.transaction("images", "readwrite");
            const store = tx.objectStore("images");
            const item = {
                url: src,
                blob: blob
            };
            store.add(item);
        })
        .catch(error => console.error('Could not cache image:', src, error));
    });
}
```

Рисунок 4.33 – Функція кешування даних

Джерело: розроблено автором

Функція cacheImages виконує кешування зображень, знайдених на вебсторінці, у локальну базу даних IndexedDB. Процес кешування починається з вибору всіх елементів <img> на сторінці за допомогою document.querySelectorAll(‘img’). Для кожного зображення з атрибутом src, який містить URL-адресу зображення, виконується HTTP-запит методом fetch щоб отримати зображення з мережі. Якщо запит завершується успішно, відповідь сервера перетворюється у BLOB-об’єкт, який представляє двійкові дані зображення. У випадку невдачі запиту генерується виняток з відповідним повідомленням про помилку.

Після отримання BLOB-об’єкта відкривається транзакція з базою даних db в режимі «readwrite», що дозволяє змінювати дані у базі. Створюється новий запис у

сховищі images цієї бази даних, куди додається об’єкт з полями url та blob. Такий об’єкт містить URL-адресу зображення та саме зображення у форматі BLOB. Запис додається до бази даних методом store.add(item).

В разі виникнення помилки під час кешування зображення (наприклад, якщо зображення не може бути завантажене або збережене у базі даних), виконується блок catch, який реєструє помилку в консолі. Це допомагає в ідентифікації проблем, пов’язаних із мережевими запитами або збереженням даних.

Ця функція важлива для покращення продуктивності вебсторінок, оскільки зменшує залежність від мережевих запитів при повторному доступі до сторінок, швидше відображаючи зображення з локального кешу.

Наступна функція – це завантаження даних з кешу та їх форматування в готові зображення (рис. 4.34).

```
    function loadImagesFromCache() {
        const transaction = db.transaction("images", "readonly");
        const store = transaction.objectStore("images");
        document.querySelectorAll('img').forEach(img => {
            const src = img.getAttribute('src');
            const request = store.get(src);

            request.onsuccess = function() {
                if (request.result) {
                    const url = URL.createObjectURL(request.result.blob);
                    img.src = url;
                }
            };
        });
    }
});
```

Рисунок 4.34 – Функція loadImagesFromCache()

Джерело: розроблено автором

Функція loadImagesFromCache використовується для завантаження зображень з локального кешу, збереженого у базі даних IndexedDB. Функція ініціює транзакцію з базою даних db, використовуючи режим "readonly". Далі за допомогою методу

document.querySelectorAll(‘img’) знаходяться всі елементи <img> на сторінці. Для кожного зображення виконується функція, яка спочатку зчитує його атрибут src для визначення URL-адреси зображення. Для кожного URL зображення створюється запит до бази даних за допомогою методу get(src), де src є ключем для пошуку в об’єктному сховищі images. Коли запит завершується успішно, обробник request.onsuccess перевіряється чи існує результат запиту. Якщо зображення знайдено (request.result), для BLOB об’єкта зображення створюється URL за допомогою URL.createObjectURL(request.result.blob). Цей URL встановлюється як новий атрибут src для тега <img>, ефективно замінюючи посилання на зовнішнє зображення на локальне збережене зображення.

4.4 Тестування роботи вебсайту

Для перевірки роботи вебсайту спочатку користувач має створити локальний вебсервер та завантажити додаток, або якщо додаток на веб-хостингу перейти за відповідною URL-адресою.

В даному випадку вебсайт завантажуватиметься з локального серверу тому його потрібно запустити та перейти по відповідному URL-адресу: http://127.0.0.1:8080/sites/main.html.

Після завантаження сторінки користувач потрапляє на головну сторінку сайту (рис. 4.35).

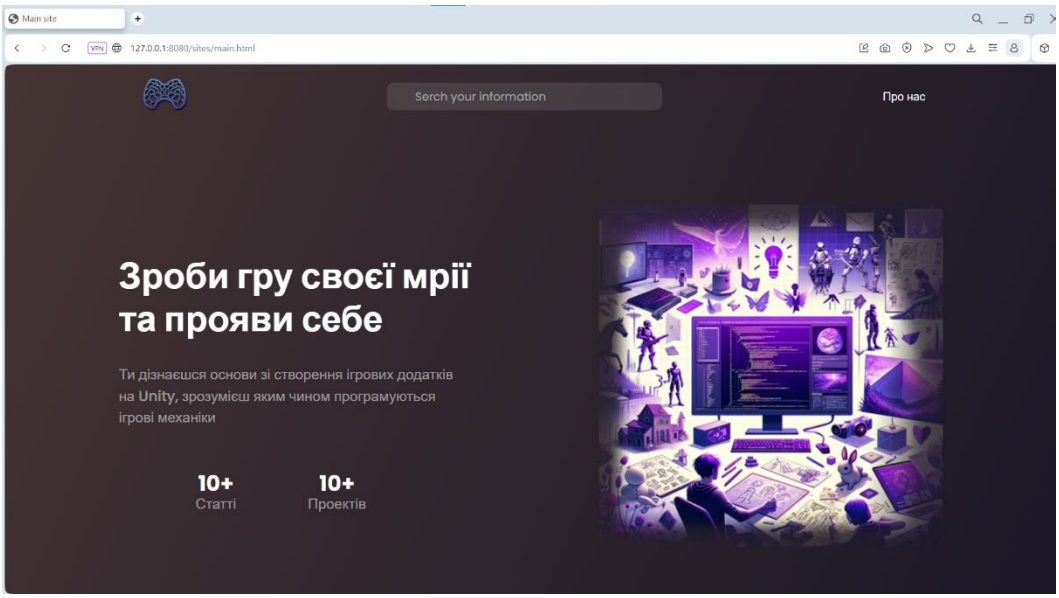

Рисунок 4.35 – Початок головної сторінки

Джерело: розроблено автором

Наступним анімованим блоком є вибір статті з 3 які пропонуються авторами сайту (рис. 4.36).

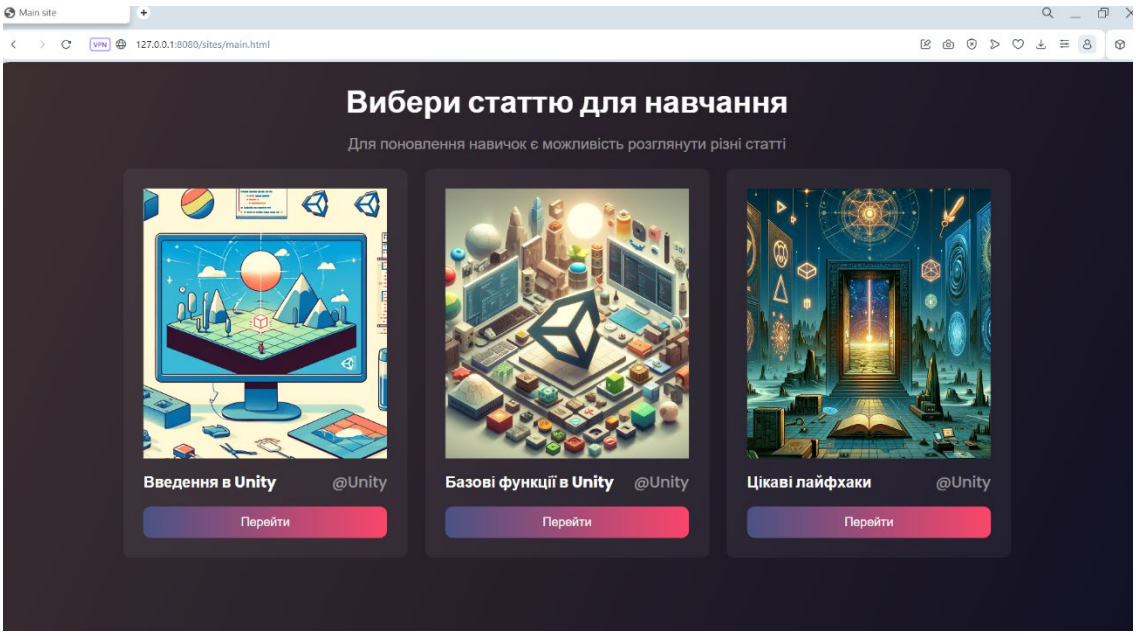

Рисунок 4.36 – Друга частина сторінки

Джерело: розроблено автором

Даний блок з’являється за допомогою анімації коли користувач тільки перейде в поле зору цього блоку. Даний блок складається з 3 анімованих карток при наведені курсору колір змінюється на сірий.

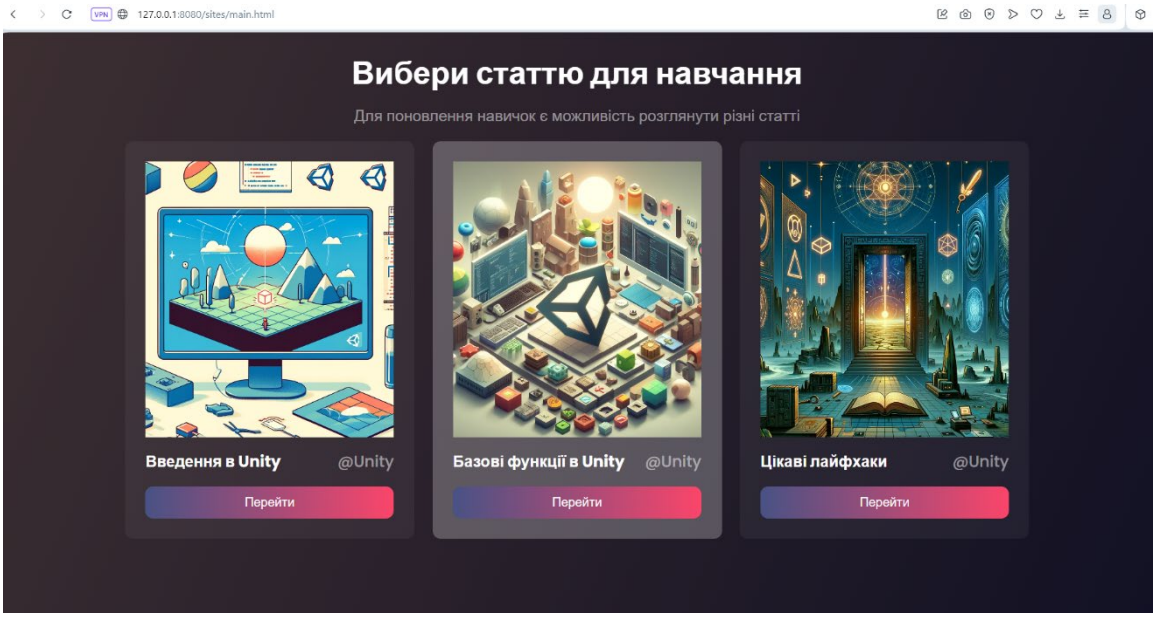

Рисунок 4.37 – Робота анімації

Джерело: розроблено автором

У вмісті карток є кнопки натискаючи на які можна потрапити на статтю. Наступним блоком є випадаючі списки які можуть наповнюватися корисним вмістом в майбутньому (рис. 4.38).

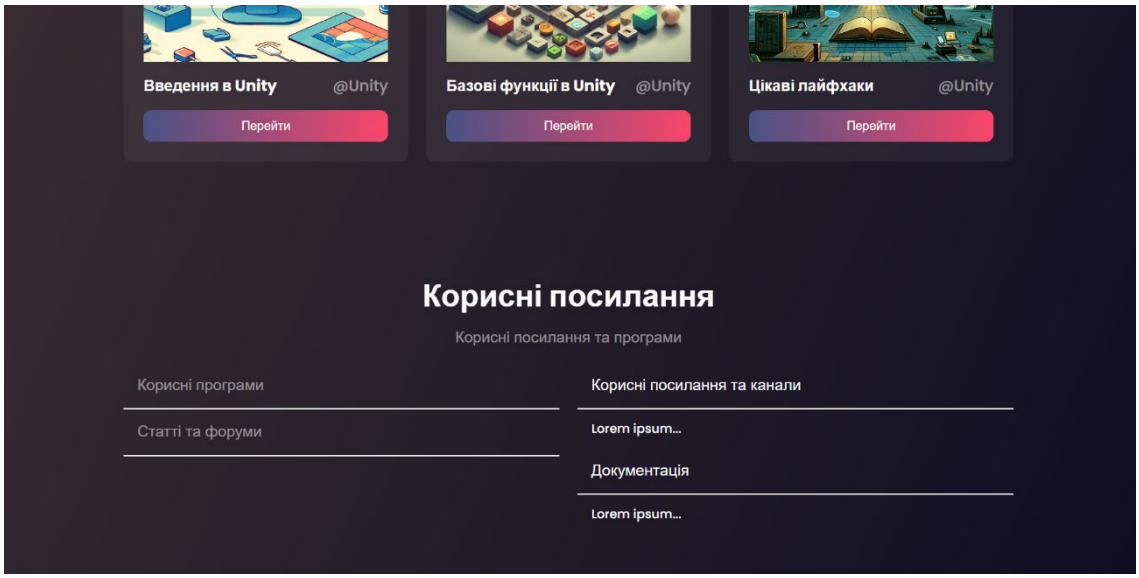

Рисунок 4.38 – Кінець головної сторінки

Джерело: розроблено автором

Наступна сторінка «Про нас». Щоб перейти на сторінку, користувач може піднятися на початок сторінки де спостерігатиме кнопку переходу «Про нас» (рис. 4.39).

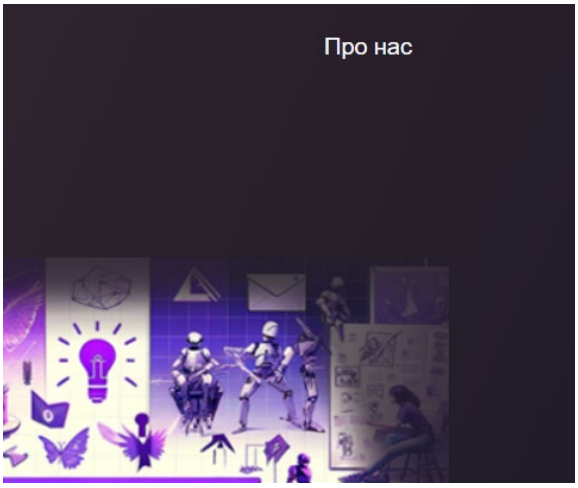

Рисунок 4.39 – Посилання на сторінку «Про нас»

Джерело: розроблено автором

Або користувач може ввести відповідний URL-адрес: http://127.0.0.1:8080/sites/about.html. Після переходу користувач спостерігає фото пейзажу та заголовок (рис. 4.40).

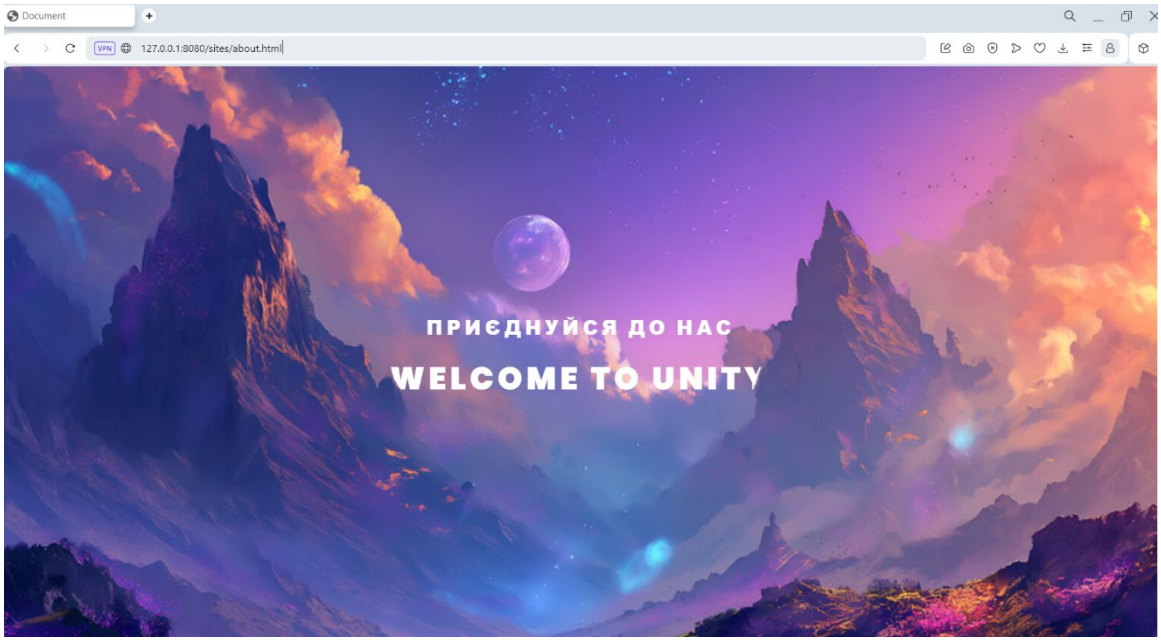

Рисунок 4.40 – Початок сторінки «Про нас»

Джерело: розроблено автором

Далі під час прокручування користувач може спостерігати так званий ефект «Паралакс», що надає ілюзії об’єму зображення.

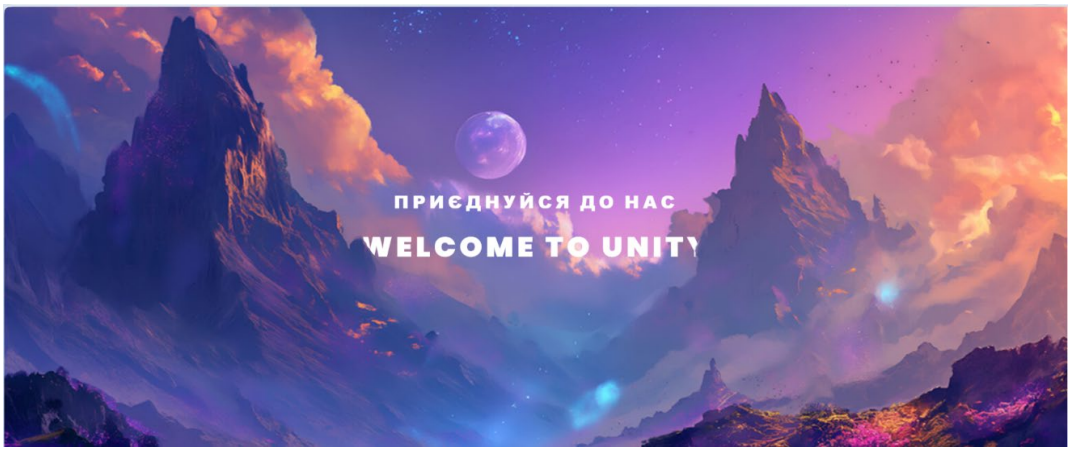

Рисунок 4.41 – Ефект «Паралакс»

Джерело: розроблено автором

Даний ефект відбувається в результаті різної швидкості переміщення 3 слоїв зображення, що додає їй об’ємності. Далі користувач буде спостерігати анімацію появлення контентних блоків які можна було спостерігати в попередній сторінці (рис. 4.42).

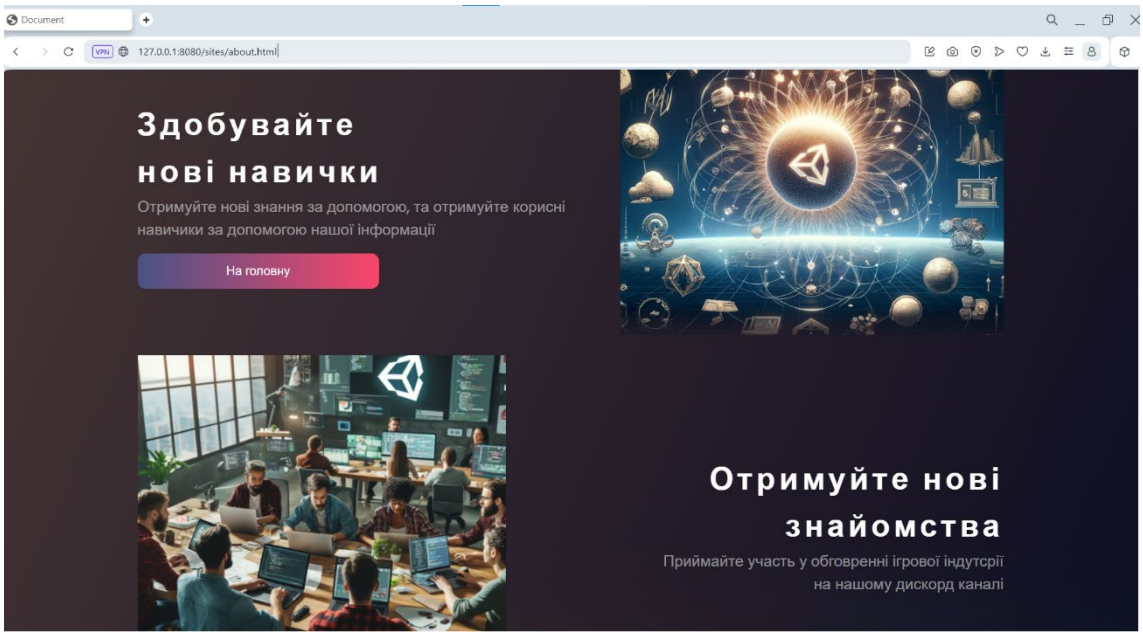

Рисунок 4.42 – Друга частина сторінки «Про нас»

Джерело: розроблено автором

Остання частина даної сторінки має складну анімацію скролу та прокручування фотографій. Дана частина представляє собою фото-галерею, яка складається з декількох фото про популярні ігри зроблені на Unity, що можна перегляду шляхом скролу (рис. 4.43).

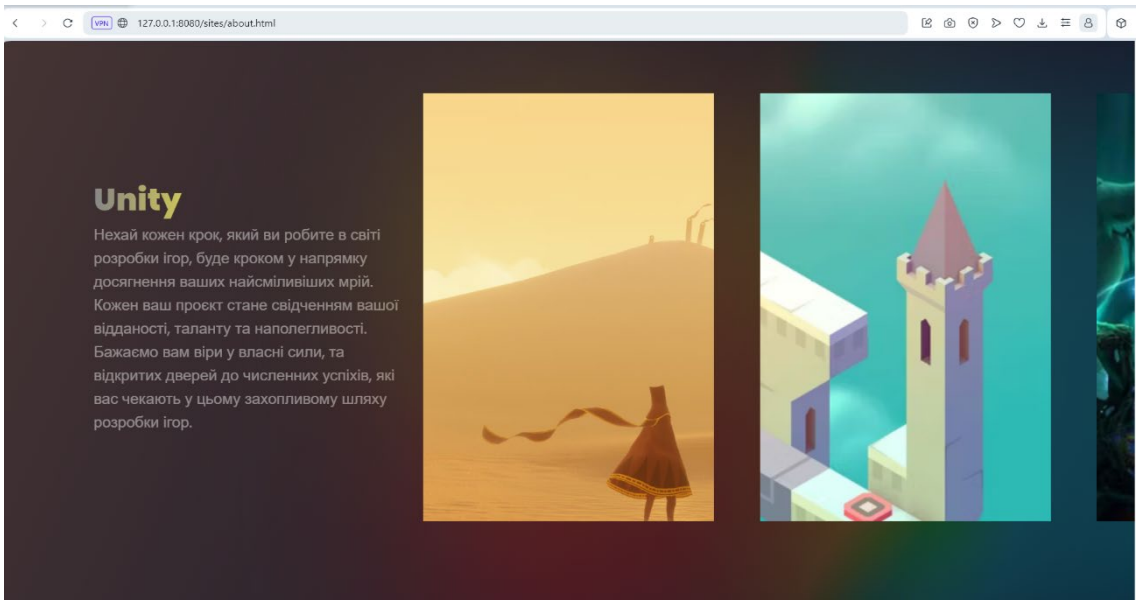

Рисунок 4.43 – Фото-галерея відео-ігор

Джерело: розроблено автором

В процесі перегляду текст плавно зникає та з’являються нові фото (рис. 4.44).

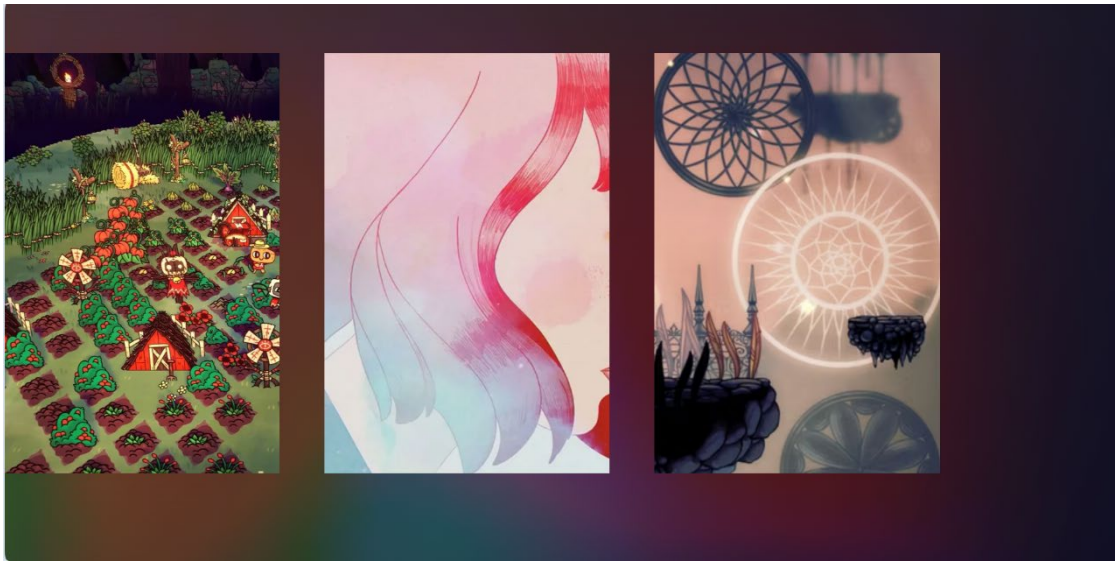

Рисунок 4.44 – Анімація перегляду фото-галереї

Джерело: розроблено автором

Щоб перейти на головну сторінку, користувач може повернутися в гору до фото пейзажу (рис. 4.45):

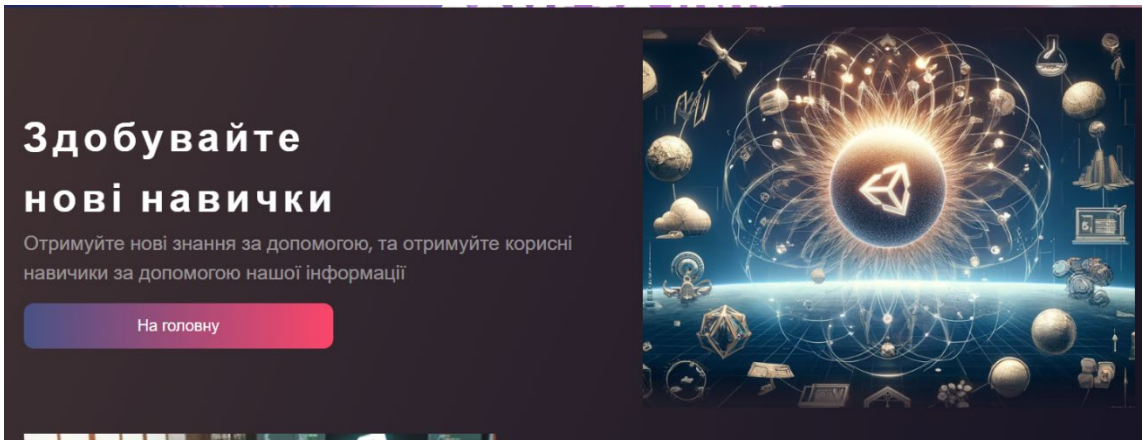

Рисунок 4.45 – Кнопка для повернення на головну сторінку

Джерело: розроблено автором

Наступною сторінкою є «Блог», це умовно сторінка з певною статтею в якій розповідаються принципи роботи програми, або пояснюються певні функції тощо. Для переходу як вже зазначалося вище потрібно вибрати одну з 3 статей, та натиснути на кнопку «Перейти».

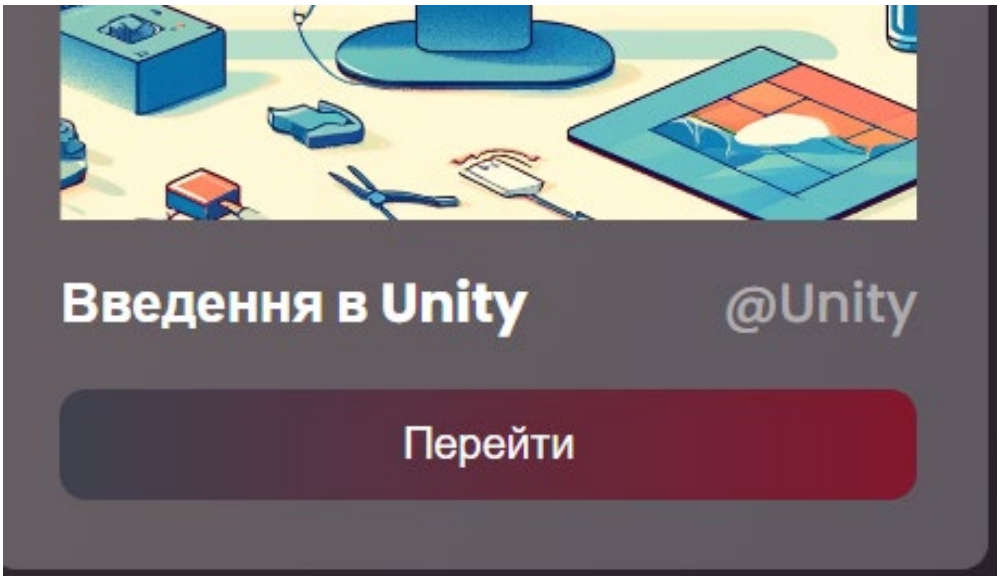

Рисунок 4.46 – Перехід на сторінку «Блог»

Джерело: розроблено автором

Користувач перейде на сторінку, що відповідає за надання документаційної інформації.

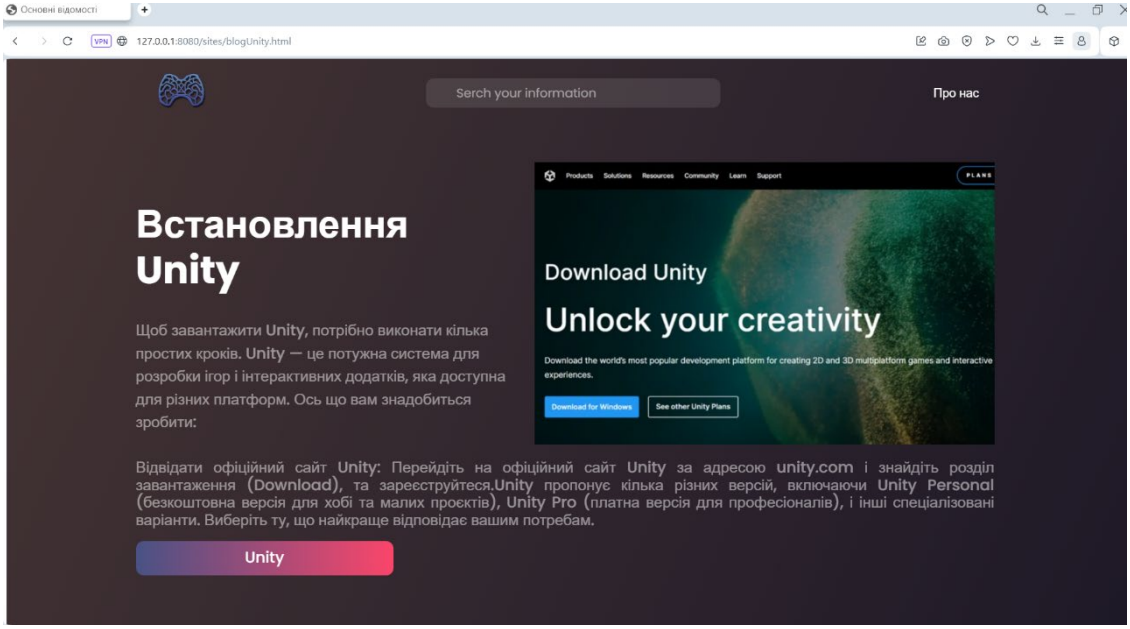

Рисунок 4.47 – Завантаження сторінки «Блог»

Джерело: розроблено автором

Для спрощення перегляду інформації, на цій сторінці відсутні анімації блоків крім навігаційного меню, в результаті користувач може шукати потрібну інформацію не чекаючи анімацію адже завдання сторінки надавати певні відомості, тлумачити термінологію та документацію, як це зроблено в документації Unity.

Блоки контенту всі супроводжуються текстом, фото та кнопкою для переходу на відповідну сторінку пов’язану з Unity (рис. 4.48).

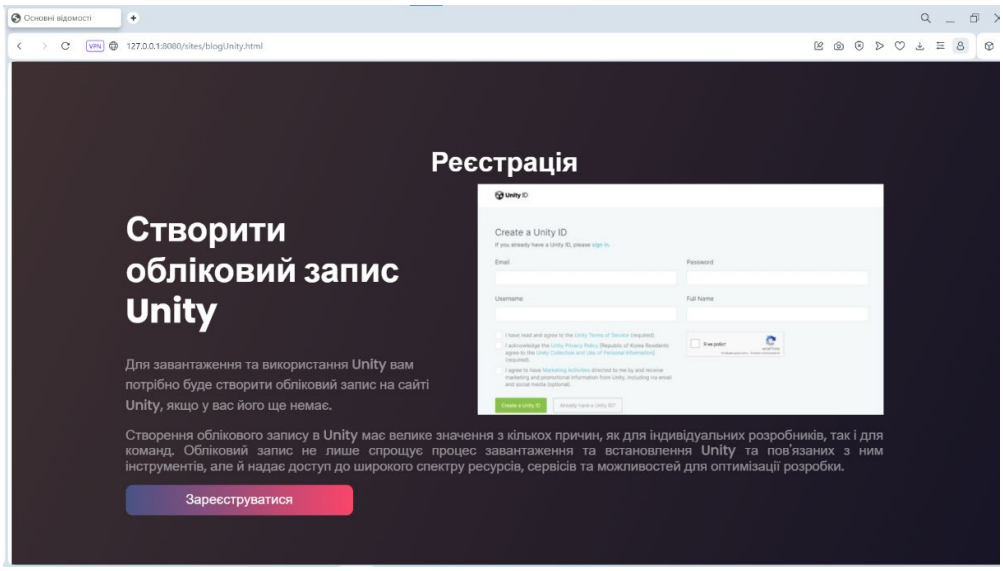

Рисунок 4.48 – Перегляд сторінки «Блог»

Джерело: розроблено автором

Для відкриття навігаційного меню користувачу потрібно перемістити курсор в лівий край сторінки, щоб відбулася анімація відкриття меню:

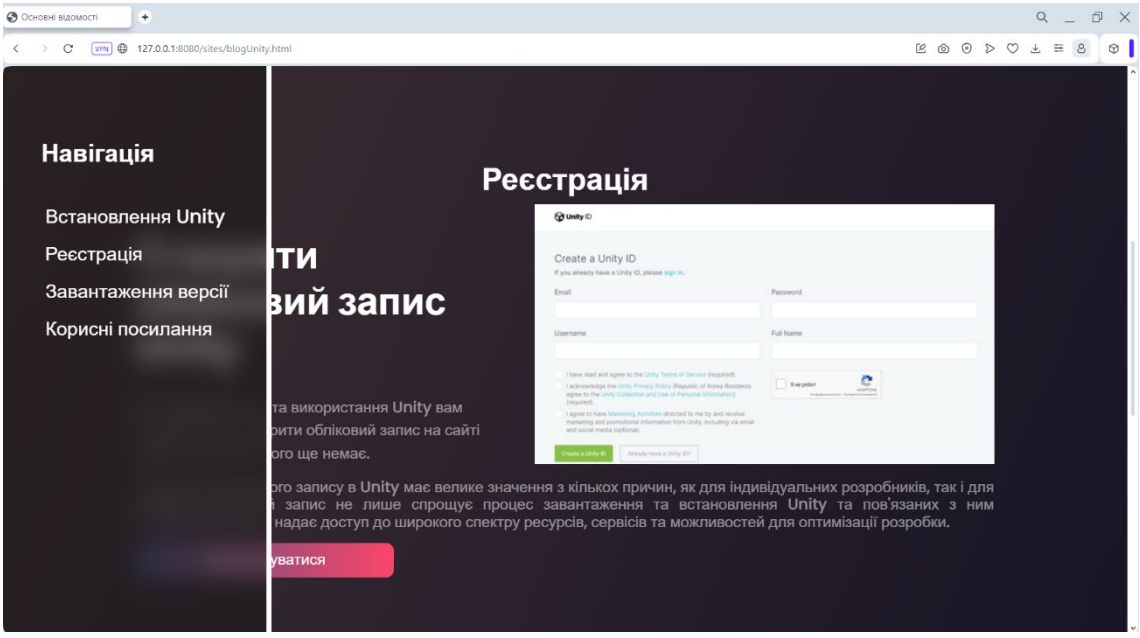

Рисунок 4.49 – Меню навігації

Джерело: розроблено автором

Дане меню використовуються для навігації по даній сторінці. Для того, щоб переміститися на певну главу, достатньо натиснути в меню на потрібний пункт (рис. 4.50):

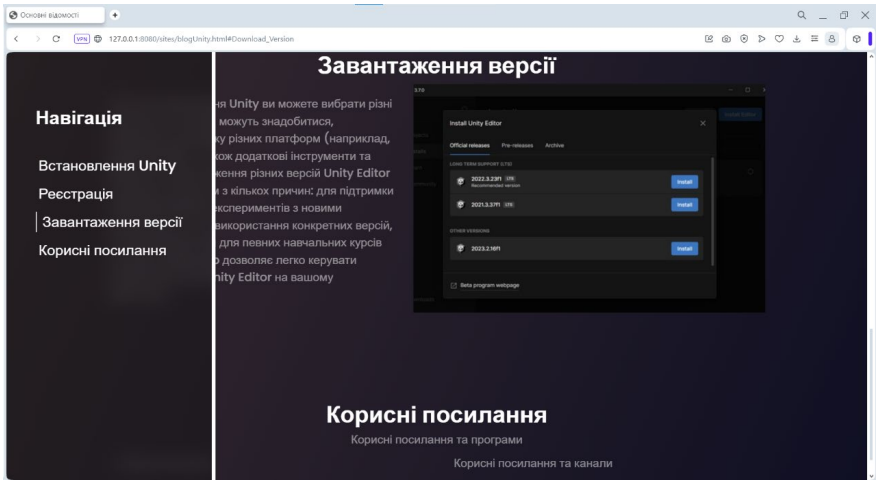

Рисунок 4.50 – Переміщення по сторінці через меню

Джерело: розроблено автором

Під час тестування вебсайту було виявлено, що він працює стабільно та функціонує повністю відповідно до очікувань. Усі основні функціональності доступні без будь-яких перешкод чи помилок. Вебсайт демонструє швидку відповідь на запити та ефективно взаємодіє з користувачем. Крім того, загальний дизайн та інтерфейс є зрозумілими та привабливими, що сприяє зручному користуванню. В результаті тестування можна підтвердити, що вебсайт готовий до використання та задовольняє всі вимоги та очікування користувачів.

4.5 Тестування впровадженої інформаційної технології

Наступним етапом потрібно провести аналіз алгоритмів завантаження сторінки та зафіксувати процеси завантаження. Для цього потрібно спочатку провести певні приготування. Налаштуємо алгоритм з котрим будемо порівнювати власний.

Для аналізу роботи даного алгоритму потрібно розробити популярний алгоритм «Lazy Loading». Алгоритм «lazy loading» (ліниве завантаження) є популярною технікою оптимізації вебсторінок і мобільних додатків, яка полягає в тому, щоб завантажувати контент лише тоді, коли він потрібен користувачеві, замість завантаження всього контенту одразу при відкритті сторінки. Для розробки цього алгоритму потрібно написати наступний код:

```
document.addEventListener('DOMContentLoaded', () => {
    const images = document.querySelectorAll('img');

    if ('IntersectionObserver' in window) {
        // Встановлення data-src та очищення src для усіх зображень
        images.forEach(img => {
            img.dataset.src = img.src;
            img.src = ''; // Очистити src або встановити запасне зображення
        });

        let lazyImageObserver = new IntersectionObserver((entries, observer) => {
            entries.forEach((entry) => {
                if (entry.isIntersecting) {
                    let lazyImage = entry.target;
                    lazyImage.src = lazyImage.dataset.src; // Відновити src з data-src
                    lazyImageObserver.unobserve(lazyImage); // Припинити спостереження
                }
            });
        });
```

Рисунок 4.51 – Початок скрипта «Lazy Loading»

Джерело: розроблено автором

Звичайно, можна зазначити, що для того щоб додати завантаження за цим алгоритмом потрібно просто до тегу додати параметр loading=»lazy», але складність в тому, що деякі браузери можуть цілеспрямовано встановлювати пріоритет на завантаження тегів з цим параметром. В даному скрипті розглядається вид алгоритму «примусовий lazy loading». Примусовим алгоритм є в результаті дій певний функцій які потрібно розглянути більш детально.

Спочатку код встановлює обробник події DOMContentLoaded, що гарантує, що скрипт запускається після повного завантаження початкової HTML сторінки. Це важливо для того, щоб усі зображення були доступні для маніпуляцій в DOM. Код використовує document.querySelectorAll(‘img’) для отримання усіх зображень на

сторінці та зберігає їх у змінну images як і в попередньому алгоритмі. Далі відбувається взаємодія з компонентом IntersectionObserver. IntersectionObserver дає можливість виявляти, коли об’єкт вступає в область видимості або виходить з неї, не використовуючи значні ресурси, які зазвичай потрібні при постійному відслідковуванні позиції об’єкту за допомогою подій прокрутки. Далі Усі зображення налаштовуються таким чином, що їх початковий src зберігається в data-src, а потім очищується src. Це запобігає небажаному завантаженню при ініціалізації сторінки.

```
        images.forEach((lazyImage) => {
            lazyImageObserver.observe(lazyImage); // Почати спостереження
        });
    } else {
        // Для браузерів, що не підтримують Intersection Observer
        images.forEach(img => {
            img.src = img.dataset.src;
        });
    }
});
```

Рисунок 4.52 – Кінець код алгоритму «Lazy Loading»

Джерело: розроблено автором

Якщо IntersectionObserver не підтримується браузером, встановлюється src для кожного зображення безпосередньо з data-src. Це гарантує, що зображення завантажуються звичайним способом, хоча і без оптимізації lazy loading.

Спочатку потрібно відрити інструмент розробника в браузері (рис. 4.53):

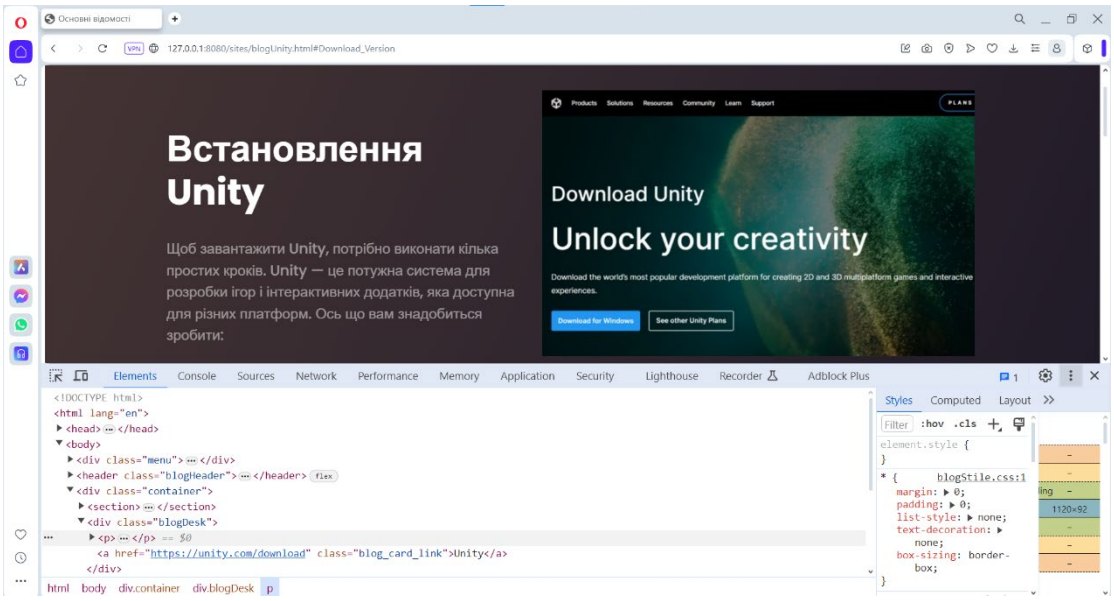

Рисунок 4.53 – Інструмент розробника

Джерело: розроблено автором

Далі потрібно перейти у вкладку «Application», щоб перевірити чи дана сторінка має збережені кешовані дані, якщо так то їх потрібно видалити за допомогою кнопки «Clear site data» (рис. 4.54).

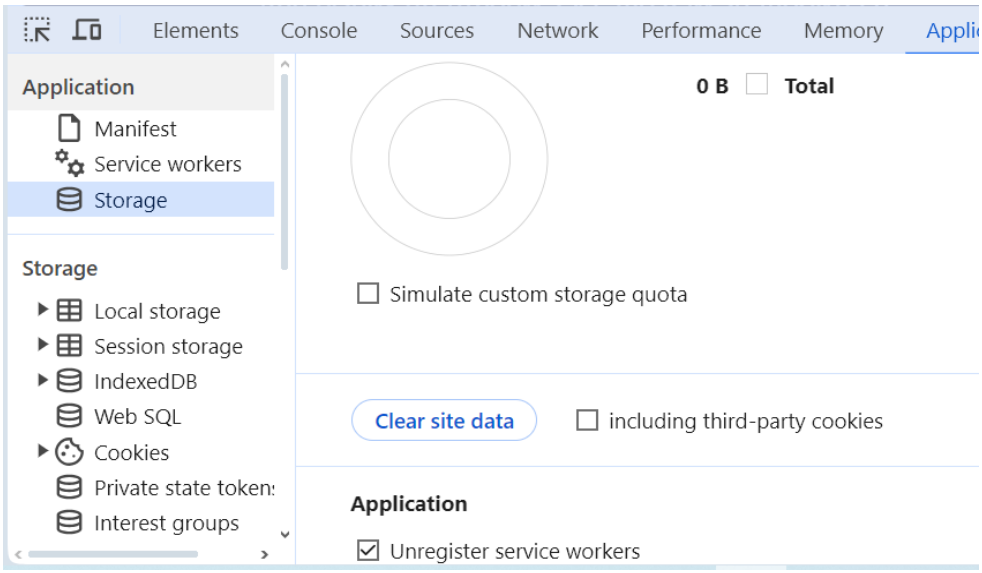

Рисунок 4.54 – Видалення кешу

Джерело: розроблено автором

Далі переходимо у вкладку «Network». Дана вкладка нам знадобиться для відслідковування процесу завантаження кожного файлу або зображення, та загальний час завантаження сторінки. В кінці потрібно встановити налаштування «Fast 3G» (рис. 4.55).

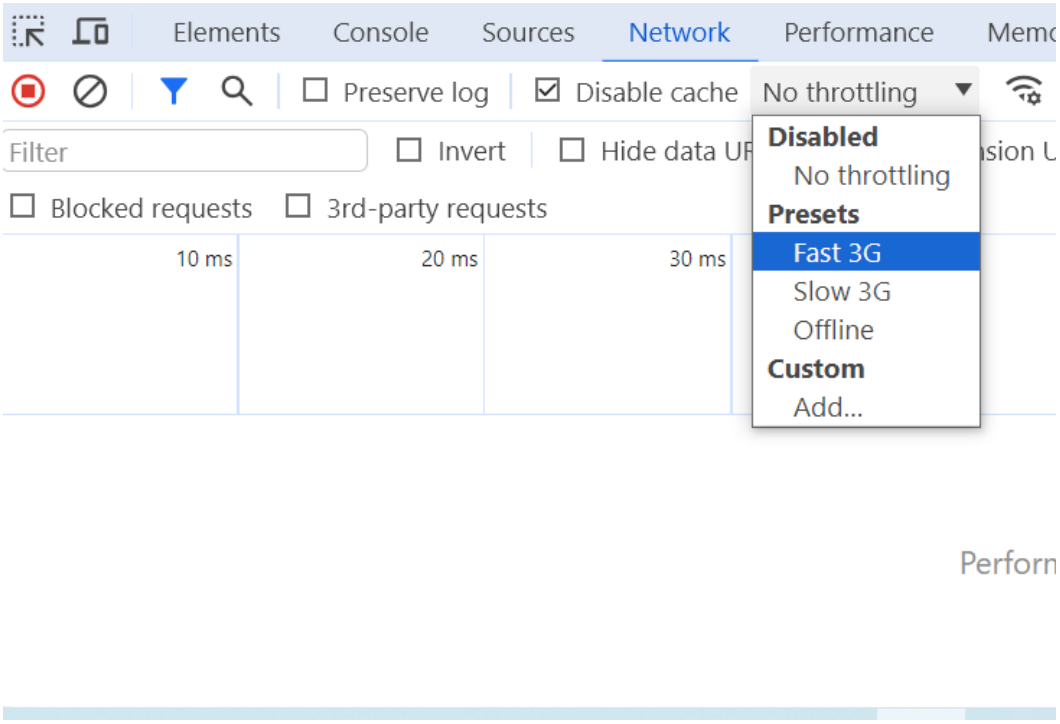

Рисунок 4.55 – Вкладка «Network»

Джерело: розроблено автором

Наступним кроком потрібно підключити сам скрипт алгоритму до сторінки вписавши 1 тег:

```
sites > blogUnity.html > html > head > script
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Основні відомості</title>
    <script src="scripts/cacheScript.js"></script>
    <link rel="stylesheet" href="stiles/normalize.min.css">
    <link rel="stylesheet" href="stiles/style.css">
    <link rel="stylesheet" href="stiles/blogStile.css">
</head>
<body>
    <div class="menu">
        <div class="blogtitle">
            <h1>Навігація</h1>
```

Рисунок 4.56 – додавання власного алгоритму на сторінку

Джерело: розроблено автором

Після перезавантаження сторінки користувач отримує спливаюче вікно в якому йому потрібно погодитись на кешування даних (рис. 4.57):

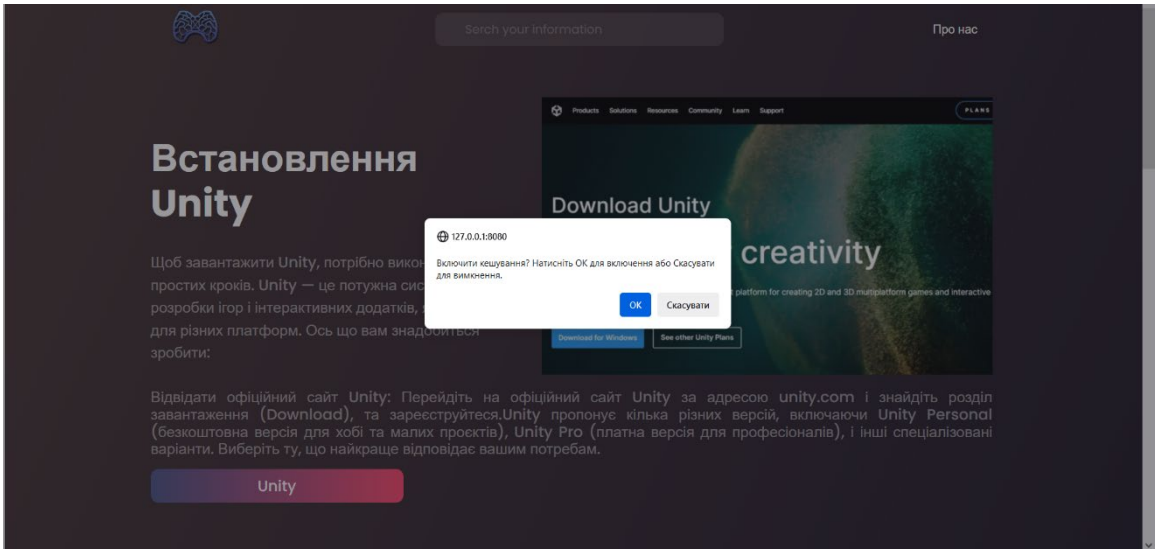

Рисунок 4.57 – Поява випливаючого вікна

Джерело: розроблено автором

Після перезавантаження знову повернемося в «Application», та переглянемо яку кількість місця зайняло кешування даних (рис. 4.59):

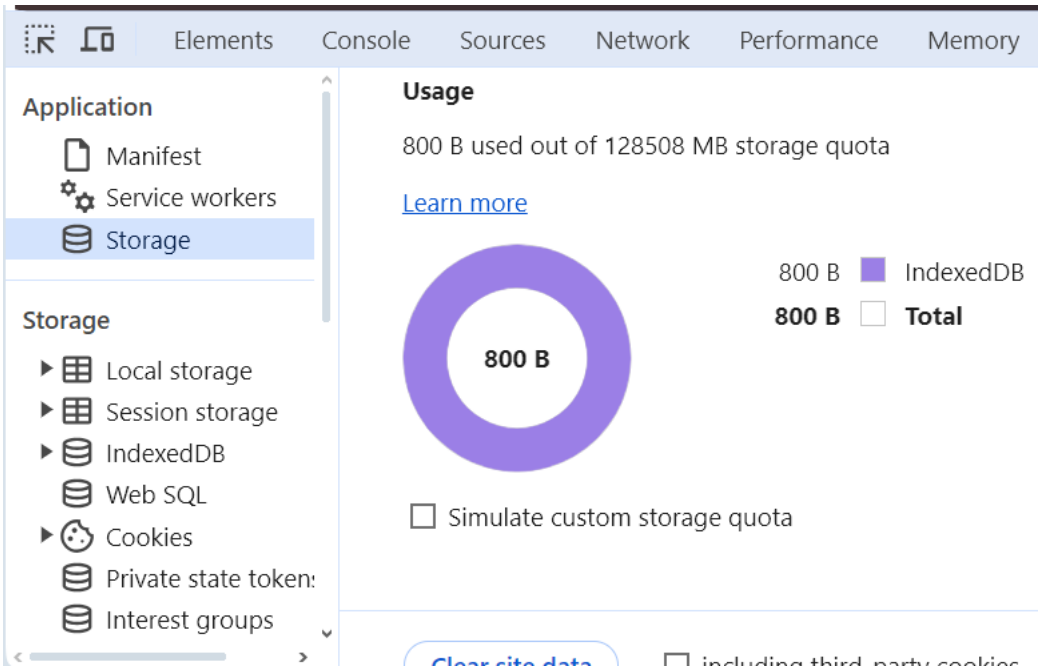

Рисунок 4.58 – Розгляд зарезервованої пам’яті

Джерело: розроблено автором

Після того як впевнилися, що алгоритм працює і кешування відбулося. Потрібно перейти у вкладку «Network» та перезавантажити стоірнку:

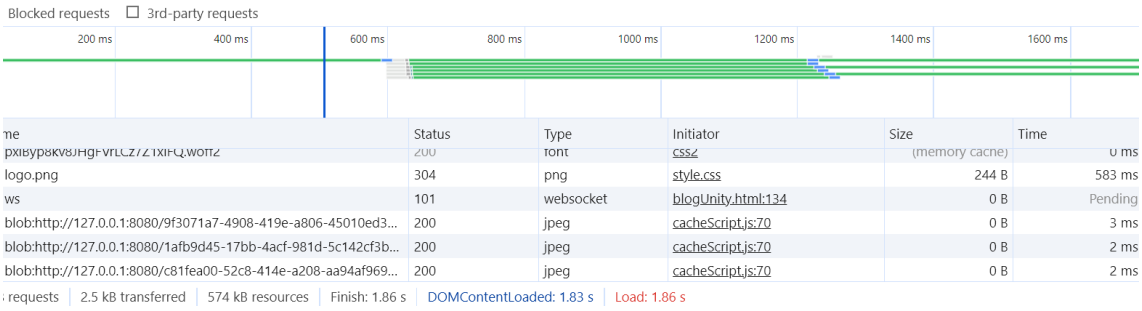

Рисунок 4.59 – Результати завантаження сторінки

Джерело: розроблено автором

Отже під час завантаження сторінки за допомогою власного алгоритму кешування тривалість завантаження сторінки становить 1.86 секунд.

Тепер потрібно порівняти з алгоритмом «Lazy Loading». Для цього потрібно відключити застосування кешу для завантаження потрібно натиснути «Disable cache».

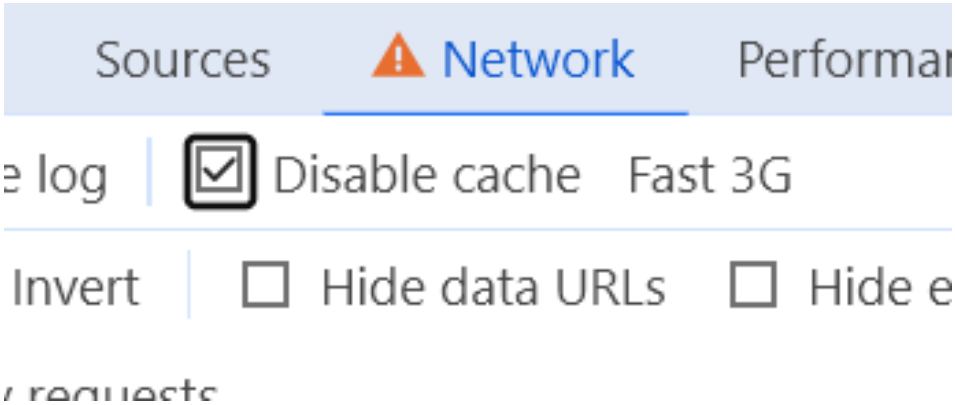

Рисунок 4.60 – Заборона на завантаження з кешу

Джерело: розроблено автором

Після перезавантаження сторінки, тривалість завантаження сторінки, а саме візуального контенту стало довшою - 4.59 секунд. Крім повільного завантаження сторінки, відбувається також дозавантаження додаткового візуального контенту – фото.

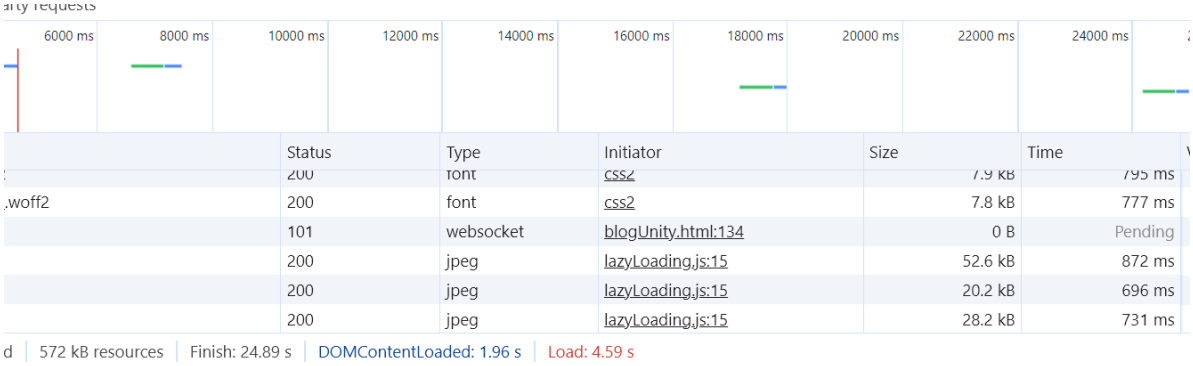

Рисунок 4.61 – Результати завантаження з алгоритмом «Lazy Loading»

Джерело: розроблено автором

Для більш точних результатів виконаємо тестування в 20 ітерацій. Результати зібрані до таблиці 4.1, де порівняно завантаження сторінки з алгоритмом кешування, Lazy Loading та без застосування алгоритмів. На основі отриманих результатів побудуємо графік порівняння швидкості завантаження вебсторінки (рис. 4.63).

Таблиця 4.1 – Порівняння швидкості завантаження вебсторінки

№	Власний алгоритм	Lazy Loading	Без оптимізації
1	1,89	4,6	6,91
2	1,99	4,59	7,04
3	1,96	4,64	6,96
4	1,87	4,6	7,1
5	1,95	4,63	6,98
6	1,97	4,62	6,96
7	1,87	4,6	6,97
8	1,92	4,58	6,99
9	1,97	4,61	7,06
10	1,94	4,65	7,04
11	1,87	4,63	6,99
12	1,96	4,61	7,01
13	1,98	4,64	7,03
14	1,89	4,58	7,03
15	1,95	4,63	7,06
16	1,97	4,55	6,94
17	1,97	4,62	6,93
18	1,92	4,64	6,95
19	1,91	4,62	6,93
20	1,93	4,63	6,94

Джерело: розроблено автором

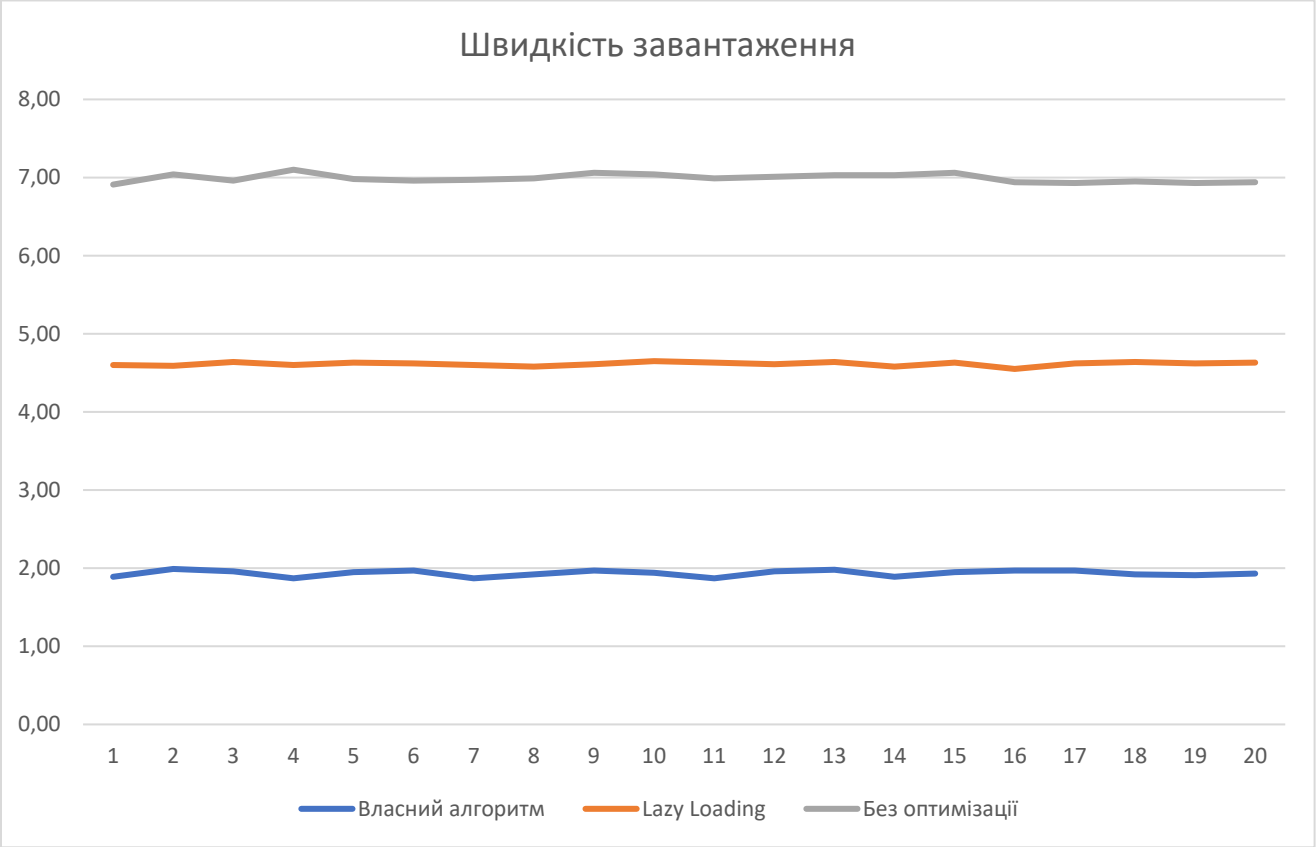

Рисунок 4.63 – Графік порівняння швидкості завантаження

Джерело: розроблено автором

Кешування картинок у форматі BLOB виявилося більше ефективнішим для завантаження вебсайту з кількох перспектив. По-перше, зберігання картинок у форматі BLOB безпосередньо в базі даних дозволяє їм бути викликаними разом зі сторінкою, що призводить до швидкого відображення контенту без очікування на додаткове завантаження. Це особливо важливо для користувачів з повільним Інтернет-з'єднанням або на мобільних пристроях, де швидкість завантаження має велике значення.

По-друге, кешування картинок у форматі BLOB дозволяє зменшити кількість HTTP-запитів, що потрібні для завантаження сторінки. Оскільки картинки вже знаходяться в базі даних, не потрібно виконувати окремі запити до сервера для кожної картинки. Це допомагає знизити навантаження на сервер та зменшити час завантаження сторінки.

Крім того, кешування картинок у форматі BLOB спрощує управління ресурсами сайту. Оскільки всі дані, включаючи картинки, зберігаються в базі даних, не потрібно вести окремий контроль за файлами на сервері. Це спрощує процес резервного копіювання та відновлення даних, а також полегшує розгортання та масштабування вебдодатку.

Отже, кешування картинок у форматі BLOB може виявитися важливим кроком у покращенні продуктивності та швидкості завантаження вебсайту, забезпечуючи більш зручний та ефективний спосіб управління контентом.

ВИСНОВКИ

В результаті виконання кваліфікаційної роботи магістра розроблена та впроваджена інформаційна технологія оптимізації швидкодії завантаження вебсайту.

З метою пошуку шляхів покращення продуктивності алгоритмів завантаження було проведено дослідження та розробку методів оптимізації вебсайтів з акцентом на процес зберігання, завантаження та керування даними на клієнтській частині вебсайту. Ці зусилля були спрямовані на підвищення продуктивності вебсайтів і покращення користувацького досвіду, які є критичними аспектами в сучасному цифровому просторі. Багато існуючих технологій використовують лише власні функції не поєднуючи їх з іншими для отримання ще більших результатів у оптимізації вебсайтів.

Визначивши мету та задачі проєкту наступним кроком було виконання проєктування, а саме структурно-функціонального моделювання, моделювання варіантів використання, проєктування моделі бази даних, та проєктування інформаційної технології. Результати проектування представлені відповідними діаграмами в пояснювальній записці.

Завершальним етапом кваліфікаційної роботи магістра є реалізація технології. Для демонстрації використання технології було створено вебсайт. Розроблена та впроваджена технологія дозволяє оптимізувати завантаження сторінок вебсайту. Цей алгоритм дозволяє скоротити час, необхідний для завантаження, а також зменшує навантаження на базу даних та полегшує керування контентом.

Результати тестового впровадження інформаційної технології в тестовий проект дозволяють зробити висновок, що поєднання декількох технологій призводить до зменшення кількість HTTP-запитів, що потрібні для швидкого відображення сторінки без очікування на додаткове завантаження

Наукова новизна полягає у розробленні інформаційної технології на основі комбінації алгоритмів кешування та алгоритмів завантаження даних у формат BLOB.

Впровадження інформаційної технології на основі вдосконаленого методу кешування дозволяє зменшити час завантаження сторінок вебсайту, покращити швидкість роботи вебсайту та забезпечити простоту застосування розробниками.

Основні теоретичні, методологічні та практичні результати проведеного дослідження, були подані у вигляді доповіді на наукових конференціях:

- «Інформатика Математика Автоматика», м. Суми: СумДУ, 22-26 квітня 2024 р.
- «Сучасні світові тенденції розвитку науки та інформаційних технологій», м. Одеса, 30–31 травня 2024 р.

Результати дослідження подані до публікації в збірнику праць VII Міжнародній науково-практичній конференції «Сучасні світові тенденції розвитку науки та інформаційних технологій» (рукопис статті прийнятий до друку редакцією від 09.05.2024 р.)

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. 2012 CSS fellows class [member activities]. IEEE Control Systems. 2012. Vol. 32, no. 3. P. 17–21.

2. Analyzing Critical Rendering Path Performance Engines – [Електронний ресурс] – Режим доступу: https://web.dev/articles/critical-rendering-path/analyzing-crp

3. Animating contents on scroll Using the intersection observer API – [Електронний ресурс] – Режим доступу: https://dev.to/estheridabor/animating-contents-on-scroll-using-the-intersection-observer-api-1k9e

4. Arni S. A., Mongkau D. C., Berelaku A. Analisis performa website menggunakan gtmetrix. Jurnal Minfo Polgan. 2023. Vol. 12, no. 1. P. 857–861.

5. Barker T. High performance responsive design: building faster sites across devices. O'Reilly Media, 2014. 176 p.

6. Bormann C., Hoffman P. Concise binary object representation (CBOR). RFC Editor, 2020. URL: https://doi.org/10.17487/rfc8949 (date of access: 06.05.2024).

7. Blokdyk G. Open Systems Interconnection A Complete Guide. Toronto : 5STARCooks, 2021. 23-27 p.

8. Books H. G. Don't panic! I'm a professional HTML email developer : customized 100 page lined notebook journal gift for a busy HTML email developer: far better than a throw away greeting card. Independently Published, 2020. 102 p.

9. Comparison between client-side and server-side rendering in the web development / T. Fadhilah Iskandar et al. IOP conference series: materials science and engineering. 2020.

10. Grigorik I. High performance browser networking: what every web developer should know about networking and web performance. O'Reilly Media, Incorporated, 2013, P. 400.

11. GTmetrix – [Електронний ресурс] – Режим доступу: https://gtmetrix.com/

12. Haeruddin N. Q., Faizal M. R., Baharuddin S. H. Analisis kinerja website parama pelindo menggunakan pingdom tools dan pagespeed insights. Jurnal informatika progres. 2023. Vol. 15, no. 1. P. 33–40.

13. Ham F., Alwiah Musdar I., Hasniati. Analisis performa website mind & soul menggunakan gtmetrix dan webpagetest. KHARISMA Tech. 2023. Vol. 19, no. 1. P. 1–12.

14. Holistic SEO Case Study: API – [Електронний ресурс] – Режим доступу: The Importance of a Holistic Approach and Clear Communication to SEO https://www.holisticseo.digital/seo-research-study/holistic-seo

15. How to Access Cached Pages in Google and Other Search Engines – [Електронний ресурс] – Режим доступу: https://indexsy.com/see-cached-pages/

16. Implementing Lazy Loading from Angular – [Електронний ресурс] – Режим доступу: https://dev.to/mana95/how-to-implement-lazy-loading-in-your-angular-application-4gi1

17. lnc m., Stewart R. Node. js: build web apis and applications with node. js. Independently Published, 2020, P. 65–70.

18. Let's build A bright future together. IEEE Intelligent Transportation Systems Magazine. 2016. Vol. 8, no. 2. P. 4.

19. Manenti, G., Ebrahimiarjestan, M., Yang, L., & Yu, M. (2019). Functional modelling and IDEF0 to enhance and support process tailoring in systems engineering. ISSE 2019 - 5th IEEE International Symposium on Systems Engineering, Proceedings.

20. Manzhikova S. T. Uml as a software project management tool. 2023. Vol. 23, no. 12. P. 65–70.

21. Meier G. Critical rendering path. Pagespeed optimierung. München, 2016. P. 93–100.

22. Mertz J., Nunes I. Understanding Application-Level Caching in Web Applications. ACM computing surveys. 2018. Vol. 50, no. 6. P. 8–9.

23. Mora, M., Adelakun, O., Galvan-Cruz, S., & Wang, F. (2021). Impacts of IDEF0-Based Models on the Usefulness, Learning, and Value Metrics of Scrum and XP Project Management Guides.

24. Nanda P., Singh S., Saini G. L. A Review of Web Caching Techniques and Caching Algorithms for Effective and Improved Caching. International Journal of Computer Applications. 2015. Vol. 128, no. 10. P. 41–45.

25. Osi Reference Layers – [Електронний ресурс] – Режим доступу: https://www.cleanpng.com/png-osi-model-conceptual-model-application-layer-data-3325733/

26. PageSpeed Insights – [Електронний ресурс] – Режим доступу: https://pagespeed.web.dev/

27. Queinnec C. Javascript. Technologies logicielles Architectures des systèmes. 2017, P. 19–20.

28. Santi Wahyuni F. PENERAPAN BLOB (BINARY LARGE OBJECT) ANALYSIS PADA SISTEM PENGENALAN RAMBU-RAMBU LALU LINTAS. Jurnal mnemonic. 2019. Vol. 1, no. 2. P. 43–44.

29. Shan T. C., Hua W. W. Data caching in web applications. Encyclopedia of Internet Technologies and Applications. 2008. P. 110–111.

30. Souders S. High performance web sites: essential knowledge for front-end engineers. O'Reilly Media, Incorporated, 2007.

31. Wang V., Salim F., Moskovits P. The WebSocket API. The Definitive Guide to HTML5 WebSocket. Berkeley, CA, 2013. P. 23–24.

32. Web performance acceleration by caching rendering results / Y. Nakano et al. 2015 17th asia-pacific network operations and management symposium (APNOMS), Busan, South Korea, P. 19–21 August 2015.

33. Web rendering patterns in a nutshell – [Електронний ресурс] – Режим доступу: https://dev.to/swastikyadav/the-story-of-web-rendering-patterns-36lp

34. WebPageTest – [Електронний ресурс] – Режим доступу: https://www.webpagetest.org/

35. Wickham M. Lazy Loading Images. Practical Android. Berkeley, CA, 2018. P. 47–84.

36. X-Check: improving effectiveness and efficiency of cross-browser issues detection for javascript-based web applications / G. Wu et al. IEEE Transactions on Services Computing. 2021. Vol. 14, no. 4. P. 887–890.

37. Генерація подій даних – [Електронний ресурс] – Режим доступу: https://markup-ua.com/pyat-shabloniv-zavantazhennya-danix-dlya-pidvishhennya-shvidkodi%D1%97-sajtiv/

38. Інформаційна структура сайту. Інструментальні засоби для веб-розробки – [Електронний ресурс] – Режим доступу: https://comscienceatschool.blogspot.com/p/blog-page_12.html

39. Нотація IDEF0 – [Електронний ресурс] – Режим доступу: https://elib.lntu.edu.ua/sites/default/files/elib_upload/Кондіус%202%20готовва/page9.html

40. Робота з Intersection Observer API – [Електронний ресурс] – Режим доступу: https://medium.com/@jstify.community/%D1%80%D0%BE%D0%B1%D0%BE%D1%82%D0%B0-%D0%B7-intersection-observer-api-cffae44d0133

41. Що таке UML-діаграми? – [Електронний ресурс] – Режим доступу: https://evergreens.com.ua/ua/articles/uml-diagrams.html

ДОДАТОК А

Планування робіт

Деталізація мети проєкту методом SMART. Продуктом дипломного проєкту є вебсайт, що реалізує технологію аналізу ефективності роботи IT-фахівців. Результати деталізації методом SMART розміщені у табл. А.1.

Таблиця А.1 − Деталізація мети методом SMART

Specific (конкретна)	Розробити інформаційну технологію оптимізації швидкодії вебсайту на основі кешування даних.
Measurable (вимірювана)	Результатом роботи є розроблений алгоритм, що оптимізує швидкодію завантаження вебсайту.
Achievable (досяжна)	Мета роботи розбита на задачі та побудований план реалізації. Для виконання проєкту необхідні знання технологій оптимізації швидкодії вебсайтів, а також знання з HTML, CSS, Node.js, MySQL та навичок в писанні документації. Враховуючи всі знання та вміння мета є такою, яку можливо досягти.
Relevant (реалістична)	Розроблена інформаційна технологія дозволить оптимізувати швидкодію завантаження вебсайту та позитивно покаже себе у порівнянні зі вже відомим алгоритмом.
Time-framed (обмежена у часі)	Ціль має бути досягнута в термін виконання дипломної роботи. Проєкт має виконуватись в терміни, визначені з календарним графіком.

Джерело: розроблено автором

Планування змісту структури робіт ІТ-проєкту (WBS). Структура декомпозиції робіт (WBS) у проєктному менеджменті є орієнтованою на доконане виконання проєкту декомпозицією проєкту на менші частки. Структура декомпозиції робіт є ключовою часткою робіт по проєкту, яка організовує командну роботу по проєкту у керовані частини.

WBS є ієрархічною декомпозицією проєкту у фази, кінцеві результати та пакети робіт. Вона є ієрархічною структурою, що показує подальший розподіл необхідних для виконання мети зусиль; наприклад, програма, проєкт чи договір. У проєкті чи договорі, розробка WBS відбувається, починаючи з кінцевих цілей та успішного розподілу її у керовані частини, що можуть бути оцінені за критеріями розміру, тривалості та відповідальностей (наприклад, системи, підсистеми, компоненти, задачі, підзадачі та пакети робіт) та включають усі необхідні для досягнення мети проєкту кроки.

Система декомпозиції робіт надає загальний каркас для природнього розвитку загального планування та контролю договору і є базисом для розподілу роботи на частини, що можуть бути визначеними, та з яких може бути зроблене Технічне Завдання і установлені звіти по технічним даним, графікам, вартостям, робочим годинам .

Структура декомпозиції робіт дозволяє зібрати докупи підлеглі витрати по задачах, матеріалах тощо на вищий рівень «батьківських» задач, матеріалів тощо. Для кожного елементу структури декомпозиції робіт генерується опис задачі, що має бути виконаною. Ця техніка іноді називається структурою декомпозиції системи використовується для визначення і налагодження сумарних рамок проєкту.

WBS організовується навколо ключових продуктів проєкту (чи запланованих результатів), а не необхідних робіт для випуску продукту (заплановані дії). Так як заплановані результати є бажаним завершенням проєкту, вони формують відносно стабільний набір категорій, у яких ціни запланованих для їх досягнення необхідних дій можуть бути зібрані докупи. Добре розроблена WBS робить легко досяжним призначення кожної діяльності проєкту до виключно однієї термінальної події у WBS.

Додатково до її функцій у обліку витрат WBS також допомагає співвіднести вимоги одного рівня системних специфікацій до іншого, наприклад, відповідність матриці вимог перехресних посилань до функціональних вимог на вищий чи нижчий рівні документації. WBS діаграма проєкту зображена на рисунку А.1.

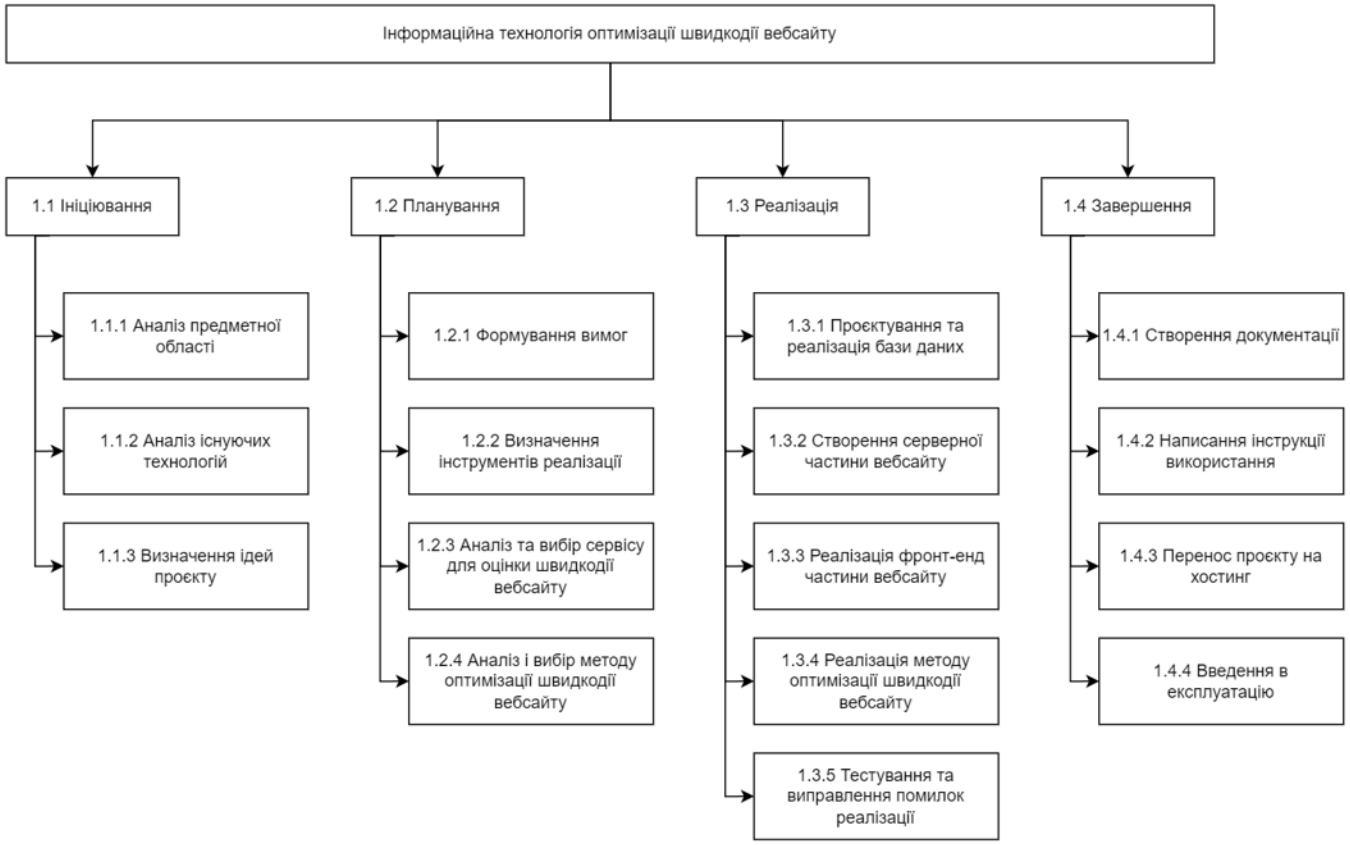

Рисунок А.1 - WBS структура проєкту

Джерело: розроблено автором

Організаційна структура проєкту (OBS). Організаційна розбивка проєкту (OBS), відома також як структура розбиття організації, це інструмент управління, що відображає складну ієрархію структури організації під час виконання проєкту. З урахуванням того, що організації можуть мати складні структури з численними рівнями управління та функціональними підрозділами, OBS допомагає систематизувати цей лабіринт, розкладаючи його на рівні та групуючи області зі схожими завданнями. Це сприяє кращому розумінню керівництвом того, як працює організація та хто відповідає за кожен аспект проєкту. Крім того, OBS сприяє

покращенню комунікації та співпраці між відділами та командами, що сприяє успішному завершенню проєкту. Застосування OBS також може бути корисним для керування витратами та контролю їх на різних рівнях організації, що допомагає у кращому розподілі ресурсів та зменшенні витрат без втрати якості проєкту. OBS-структуру проєкту представлено на рисунку А.2. Таблиця з ролями проєкту має такий вигляд:

Таблиця А.2 – Виконавці проєкту

Роль	Ім’я	Проєктна роль
Розробник та тестувальник	Задесенець Д.С.	Виконує дослідження та покращення методу оптимізації швидкодії вебсайту. Тестує систему на продуктивність та виявлення помилок.
Менеджер проєкту	Неня А.В.	Забезпечує підтримку в аналізі та реалізації. Відповідає за дотримання графіків та вимог щодо виконання завдань.

Джерело: розроблено автором

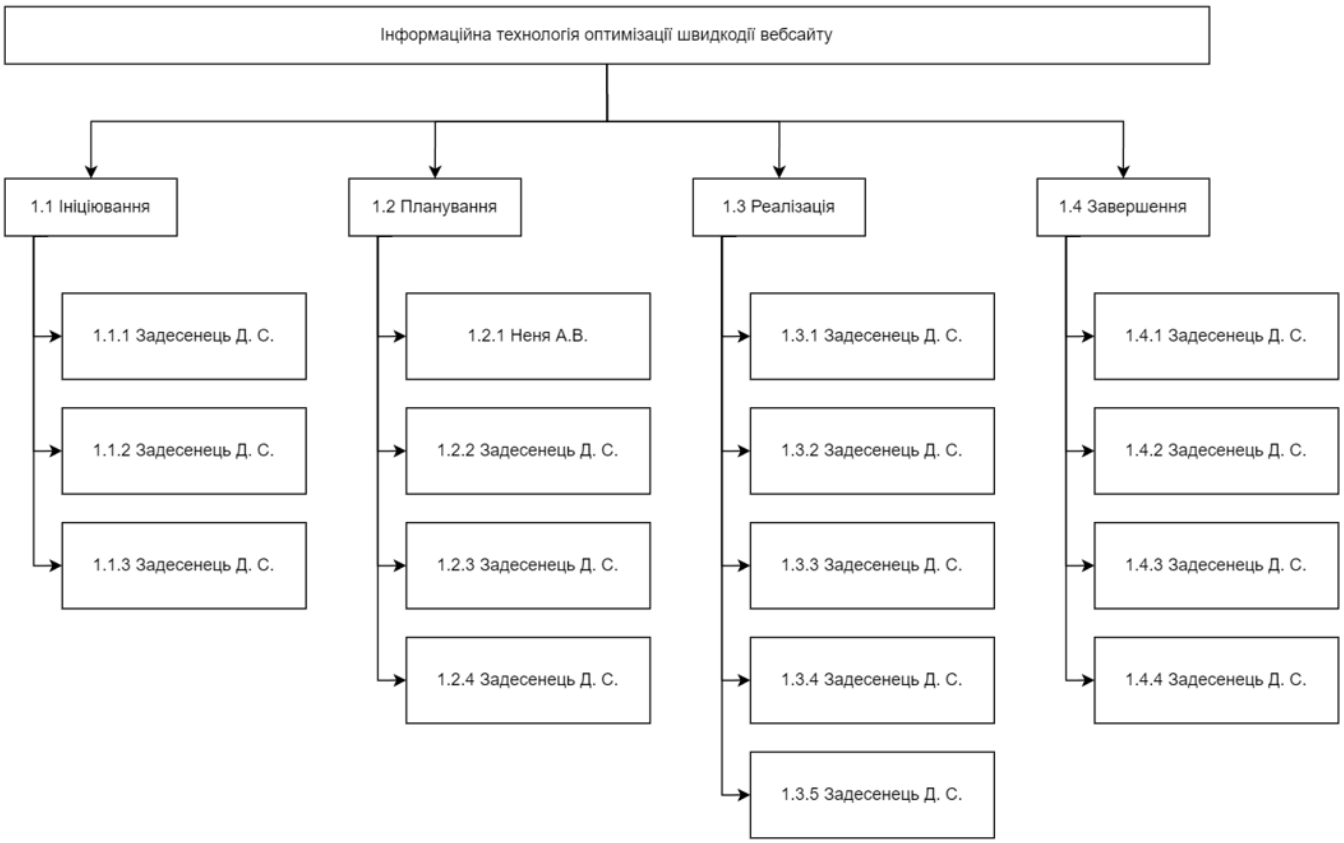

Рисунок А.2 – OBS-структура проєкту

Джерело: розроблено автором

Побудова календарного графіку виконання й діаграми Ганта ІТ - проєкту. Створення календарного графіка для ІТ-проєкту - це детальний план, який відображає послідовність подій, завдань та строків, необхідних для успішного завершення проєкту. Він може включати різні елементи, такі як строки виконання завдань, залежності між завданнями та критичні шляхи.

Цей графік є важливим інструментом управління проєктами, який допомагає планувати, відстежувати та контролювати прогрес. Він дозволяє керівництву та команді проєкту чітко уявити, коли мають бути виконані різні етапи проєкту та гарантувати вчасне виконання завдань.

У контексті ІТ-проєктів календарний графік допомагає встановити строки виконання програмних модулів, тестування, релізів та інших етапів проєкту. Крім того, він допомагає вирішувати проблеми зі звуженням строків та залежностями між

різними етапами, що сприяє ефективному управлінню ресурсами та вчасному завершенню проєкту. Результати побудови календарного плану проєкту зображені на рисунку А.3.

Task name	Start date	End date	Assigned	Status
Аналіз предметної області	30.11.2023	20.12.2023	Денис Задесенець	Done
Аналіз існуючих технологій	21.12.2023	05.01.2024	Денис Задесенець	Done
Визначення ідей проєкту	08.01.2024	22.01.2024	Денис Задесенець	Done
Формування вимог	23.01.2024	02.02.2024	Денис Задесенець	Done
Визначення інструментів реалізації	05.02.2024	16.02.2024	Денис Задесенець	Done
Аналіз та вибір сервісу для оцінки швидкодії	19.02.2024	04.03.2024	Денис Задесенець	Done
Аналіз та вибір методу оптимізації швидкодії вебсайту	05.03.2024	15.03.2024	Денис Задесенець	Done
Проєктування сайту	18.03.2024	26.03.2024	Денис Задесенець	Done
Створення серверної частини сайту	27.03.2024	01.04.2024	Денис Задесенець	Done
Реалізація фронт-енд частини вебсайту	02.04.2024	10.04.2024	Денис Задесенець	Done
Реалізація методу оптимізації швидкодії вебсайту	11.04.2024	25.04.2024	Денис Задесенець	Done
Тестування та виправлення помилок реалізації	26.04.2024	30.04.2024	Денис Задесенець	Done
Створення документації	01.05.2024	06.05.2024	Денис Задесенець	Done
Написання інструкції використання	07.05.2024	08.05.2024	Денис Задесенець	Done
Перенос проєкту на хостинг	09.05.2024	09.05.2024	Денис Задесенець	Done
Введення в експлуатацію	10.05.2024	10.05.2024	Денис Задесенець	Done

Рисунок А.3 – Календарний план проєкту

Джерело: розроблено автором

Графік Ганта є ефективним інструментом управління проєктами, який наглядно відображає розклад проєкту у формі стовпців, що представляють часові проміжки, та горизонтальних ліній, що показують різні завдання або етапи проєкту. Він допомагає керівникам проєкту та учасникам відстежувати виконання завдань, управляти ресурсами та контролювати прогрес проєкту.

Графік Ганта графічно відображає календарний графік проєкту з усіма відомостями про завдання, їх тривалість та дати початку та закінчення. Це дає можливість всім учасникам проєкту чітко бачити, що треба робити, коли це має бути виконано та хто відповідальний за кожне завдання. Крім того, керівник проєкту може відстежувати прогрес виконання завдань з часом, що допомагає вчасно виявляти можливі проблеми або затримки та приймати необхідні заходи для їх вирішення.

Діаграма Ганта також дозволяє візуалізувати критичний шлях проєкту - це найважливіші етапи або завдання, які мають найбільший вплив на загальний прогрес проєкту. Це допомагає зосередитися на успішному виконанні ключових завдань та етапів проєкту.

Загалом, графік Ганта забезпечує повне уявлення проєкту, дозволяє контролювати виконання завдань, управляти ресурсами та вирішувати будь-які проблеми, що можуть виникнути під час виконання проєкту. Результати побудови діаграми Ганта проєкту зображені на рисунку А.4.

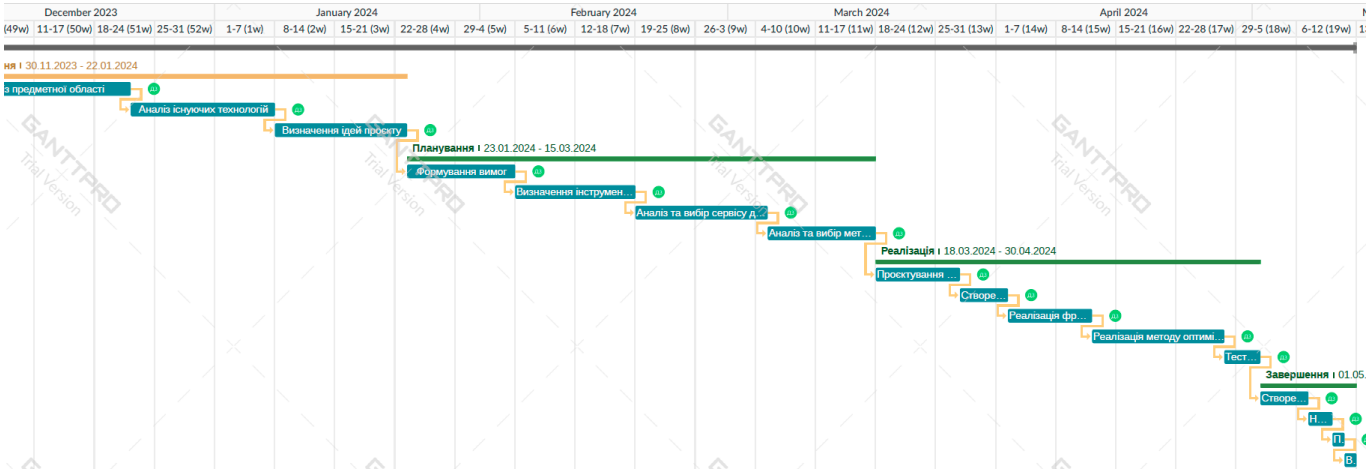

Рисунок А.4 – Діаграма Ганта

Джерело: розроблено автором

Управління ризиками. Управління ризиками – це процес реагування на непередбачені умови та ситуації, які можуть виникнути під час виконання проєкту та призвести до негативних наслідків. Цей процес включає моніторинг та контроль за ризиками з метою їх зниження або уникнення. Звісно, неможливо передбачити всі можливі ризики проєкту, але процес аналізу ризиків ґрунтується на особистому досвіді аналізатора. Якщо цей аналізатор має достатню кваліфікацію, то зазвичай більшість ризиків можна передбачити і відповідно підготуватися до них.

На основі проведеного аналізу було сформовано матрицю Risk Breakdown Structure:

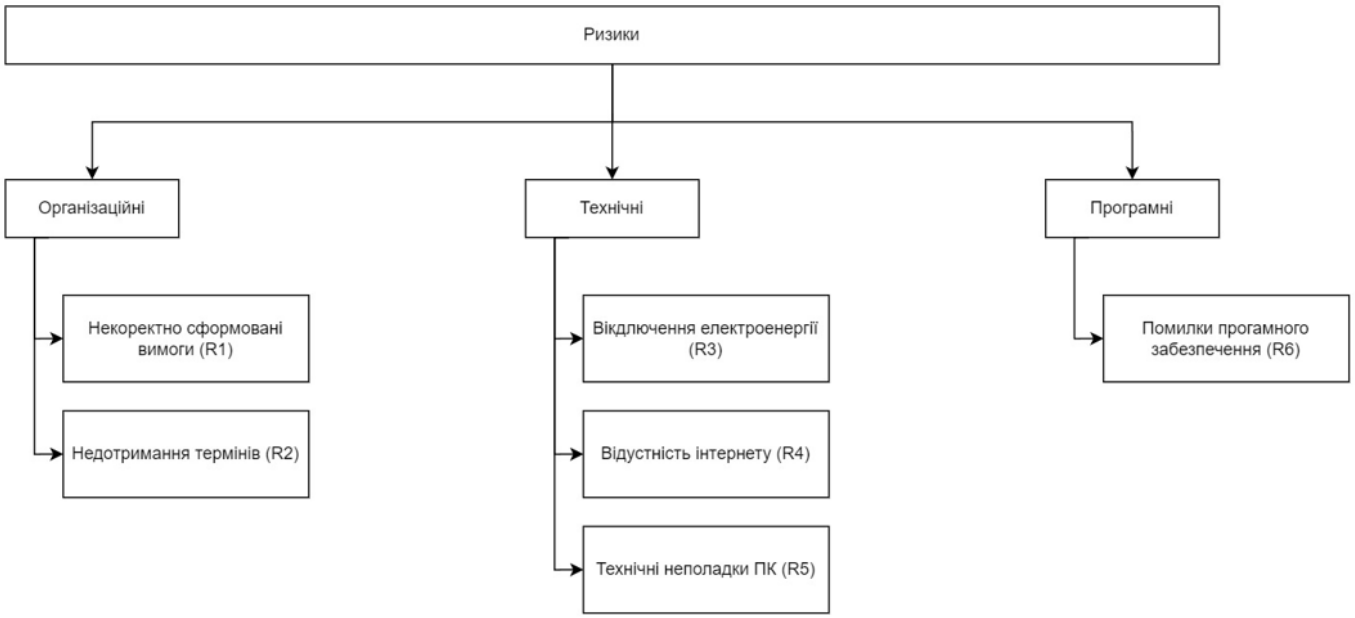

Рисунок А.5 – RBS-матриця

Джерело: розроблено автором

Далі було створено таблиці ймовірності виникнення (А3), відповідальності рівня ризику (А.4), відповідальності ступеня впливу (А.5), відповідальності ступеня витрат (А.6):

Таблиця А.3 – Матриця відповідальності ймовірності виникнення

Ймовірніть виникнення		R1	R2	R3	R4	R5	R6
1	Мінімальна						
2	Низька						
3	Середня						
4	Висока						
5	Максимальна						

Джерело: розроблено автором

Таблиця А.4 – Матриця відповідальності рівня ризику

Рівень ризику		R1	R2	R3	R4	R5	R6
1	Прийнятний						
2	Виправданий						
3	Недопустимий						

Джерело: розроблено автором

Таблиця А.5 – Матриця відповідальності ступеня впливу

Ступінь впливу		R1	R2	R3	R4	R5	R6
1	Мінімальна						
2	Незначний						
3	Допустимий						
4	Значний						
5	Максимальний						

Джерело: розроблено автором

Таблиця А.6 – Матриця відповідальності ступеня втрат

Ступінь втрат		R1	R2	R3	R4	R5	R6
1	Мінімальні						
2	Незначні						
3	Допустимі						
4	Значні						
5	Максимальні						

Джерело: розроблено автором

ДОДАТОК Б

Програмний код

about.html

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Document</title>
   <script src="scripts/cacheScript.js"></script>
   <link rel="stylesheet" href="libs/swiper/swiper-bundle.min.css">
   <link rel="stylesheet" href="stiles/styleAbout.css">
   <script src="scripts/about.js" defer></script>
</head>
<body>
   <header class="main_header">
      <div class="layers">
         <div class="layer_header">
            <div class="layer_caption">Приєднуйся до нас</div>
            <div class="layers_title">Welcome to Unity</div>
         </div>
         <div class="layer layers_base" style="background-image: url(stiles/images/laer_base.png);"></div>
         <div class="layer layers_midle" style="background-image: url(stiles/images/laer_midle.png);"></div>
         <div class="layer layers_front" style="background-image: url(stiles/images/laer_front.png);"></div>
      </div>
   </header>
   <div class="container">
      <section>
         <div class="section-items">
            <div class="left-anim section-text animate">
               <h2 class="layers_title">Здобувайте <br> нові навички</h2>
               <p class="section-discription">Отримуйте нові знання за допомогою, та отримуйте корисні навички за допомогою нашої інформації</p>
               <a href="main.html" class="card_link">На головну</a>
            </div>
            <div class="right-anim section-imge animate">
               <img class="img-section" src="stiles/images/foto_about.png" alt="">
            </div>
         </div>
      </section>
```

```
            <section>
              <div class="section-items">
                <div class="left-anim section-imge  animate">
                  <img class="img-section" src="stiles/images/foto_about2.png" alt="">
                </div>
                <div class="right-anim section-text animate" style="text-align: right;">
                  <h2 class="layers_title">Отримуйте нові знайомства</h2>
                  <p class="section-discription">Приймайте участь у обговренні ігрової індутсрії
<br> на нашому дискорд каналі</p>
                </div>
              </section>
            </div>
            <div class="description">
              <div class="logo">Unity</div>
              <p class="disc">Нехай кожен крок, який ви робите в світі розробки ігор, буде кроком у
напрямку досягнення ваших найсміливіших
                мрій. Кожен ваш проєкт стане свідченням вашої відданості, таланту та
наполегливості. Бажаємо вам віри у власні сили, та відкритих дверей до численних успіхів, які вас
чекають у цьому захопливому шляху розробки ігор.</p>
            </div>
            <div class="swiper slider slider_main">
              <div class="swiper-wrapper slier__wrapper">
                <div class="swiper-slide slider__item">
                  <div    class="slider__img"    data-swiper-parallax="20%"    style="background-image:
url(stiles/imagesGalary/journey.png);"></div>
                </div>
                <div class="swiper-slide slider__item">
                  <div    class="slider__img"    data-swiper-parallax="30%"    style="background-image:
url(stiles/imagesGalary/2.png);"></div>
                </div>
                <div class="swiper-slide slider__item">
                  <div    class="slider__img"    data-swiper-parallax="20%"    style="background-image:
url(stiles/imagesGalary/3.png);"></div>
                </div>
                <div class="swiper-slide slider__item">
                  <div    class="slider__img"    data-swiper-parallax="30%"    style="background-image:
url(stiles/imagesGalary/4.png);"></div>
                </div>
                <div class="swiper-slide slider__item">
                  <div    class="slider__img"    data-swiper-parallax="20%"    style="background-image:
url(stiles/imagesGalary/5.png);"></div>
                </div>
                <div class="swiper-slide slider__item">
                  <div    class="slider__img"    data-swiper-parallax="30%"    style="background-image:
url(stiles/imagesGalary/6.png);"></div>
                </div>
                <div class="swiper-slide slider__item">
                  <div    class="slider__img"    data-swiper-parallax="20%"    style="background-image:
url(stiles/imagesGalary/7.png);"></div>
                </div>
```

```
            </div>
        </div>
        <div class="swiper slider slider_bg">
            <div class="swiper-wrapper slier__wrapper">
                <div class="swiper-slide slider__item">
                    <div class="slider__img" data-swiper-parallax="20%" style="background-image:
url(stiles/imagesGalary/journey.png);"></div>
                </div>
                <div class="swiper-slide slider__item">
                    <div class="slider__img" data-swiper-parallax="30%" style="background-image:
url(stiles/imagesGalary/2.png);"></div>
                </div>
                <div class="swiper-slide slider__item">
                    <div class="slider__img" data-swiper-parallax="20%" style="background-image:
url(stiles/imagesGalary/3.png);"></div>
                </div>
                <div class="swiper-slide slider__item">
                    <div class="slider__img" data-swiper-parallax="30%" style="background-image:
url(stiles/imagesGalary/4.png);"></div>
                </div>
                <div class="swiper-slide slider__item">
                    <div class="slider__img" data-swiper-parallax="20%" style="background-image:
url(stiles/imagesGalary/5.png);"></div>
                </div>
                <div class="swiper-slide slider__item">
                    <div class="slider__img" data-swiper-parallax="30%" style="background-image:
url(stiles/imagesGalary/6.png);"></div>
                </div>
                <div class="swiper-slide slider__item">
                    <div class="slider__img" data-swiper-parallax="20%" style="background-image:
url(stiles/imagesGalary/7.png);"></div>
                </div>
            </div>
        </div>
        <script src="scripts/anim.js"></script>
        <script src="libs/swiper/swiper-bundle.min.js"></script>
        <script src="scripts/galaryScroll.js"></script>
    </body>
    </html>
```

blogUnity.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Основні відомості</title>
    <script src="scripts/cacheScript.js"></script>
    <link rel="stylesheet" href="stiles/normalize.min.css">
    <link rel="stylesheet" href="stiles/style.css">
```

```
    <link rel="stylesheet" href="stiles/blogStile.css">
  </head>
  <body>
    <div class="menu">
      <div class="blogtitle">
        <h1>Навігація</h1>
      </div>
      <div class="list_menu">
        <ul>
          <li><a href="#Installation">Встановлення Unity</a></li>
          <li><a href="#Registration">Реєстрація</a></li>
          <li><a href="#Download_Version">Завантаження версії</a></li>
          <li><a href="#views">Корисні посилання</a></li>
        </ul>
      </div>
    </div>
    <header class="blogHeader">
      <a href="main.html" class="header_logo"></a>
      <div class="form">
        <input type="search" class="form_input" placeholder="Serch your information">
      </div>
      <nav class="header_nav">
        <ul class="header_list">
          <li class="header_item"> <a class="header_link" href="/sites/about.html"> Про нас
</a></li>
        </ul>
      </nav>
    </header>
    <div class="container">
      <section>
        <div class="main">
          <div class="main_text">
            <h1 class="main_title" id="Installation">Встановлення Unity</h1>
            <p class="main_desc">Щоб завантажити Unity, потрібно виконати кілька простих
кроків.
                Unity — це потужна система для розробки ігор і інтерактивних додатків, яка
доступна для різних платформ. Ось що вам знадобиться зробити:
            </p>
          </div>
          <div class="main_image">
            <img class="main_img animate" src="./stiles/images/unityDowanload.jpg" alt="">
          </div>
        </div>
      </section>
      <div class="blogDesk">
        <p>Відвідати офіційний сайт Unity: Перейдіть на офіційний сайт Unity
          за адресою unity.com і знайдіть розділ завантаження (Download), та
зареєструйтеся.Unity пропонує кілька різних версій, включаючи Unity Personal (безкоштовна версія
для хобі та малих проєктів), Unity Pro (платна версія для професіоналів), і інші спеціалізовані
варіанти. Виберіть ту, що найкраще відповідає вашим потребам.</p>
        <a href="https://unity.com/download" class="blog_card_link">Unity</a>
```

```
        </div>
        <section>
          <h2 class="title" id="Registration">Реєстрація</h2>
          <div class="main">
            <div class="main_text">
              <h1 class="main_title">Створити обліковий запис Unity</h1>
              <p class="main_desc">Для завантаження та використання Unity вам потрібно буде створити обліковий запис на сайті Unity, якщо у вас його ще немає.
              </p>
            </div>
            <div class="main_image">
              <img class="main_img animate" src="./stiles/images/unityRegistration.jpg" alt="">
            </div>
          </div>
        </section>
        <div class="blogDesk">
          <p>Створення облікового запису в Unity має велике значення з кількох причин, як для індивідуальних розробників, так і для команд. Обліковий запис не лише спрощує процес завантаження та встановлення Unity та пов'язаних з ним інструментів, але й надає доступ до широкого спектру ресурсів, сервісів та можливостей для оптимізації розробки.</p>
          <a href="https://id.unity.com/en/conversations/45c5319b-631d-486c-8497-39cbf113baa301bf?view=register" class="blog_card_link">Зареєструватися</a>
        </div>
        <section>
          <h2 class="title" id="Download_Version">Завантаження версії</h2>
          <div class="main">
            <div class="main_text">
              <p class="main_desc">Під час встановлення Unity ви можете вибрати різні компоненти, які вам можуть знадобитися, включаючи підтримку різних платформ (наприклад, Android, iOS), а також додаткові інструменти та бібліотеки. Завантаження різних версій Unity Editor може бути корисним з кількох причин: для підтримки старіших проектів, експериментів з новими функціями або для використання конкретних версій, які рекомендуються для певних навчальних курсів чи гайдів. Unity Hub дозволяє легко керувати різними версіями Unity Editor на вашому комп'ютері.
              </p>
            </div>
            <div class="main_image">
              <img class="main_img animate" src="./stiles/images/versionUnity.jpg" alt="">
            </div>
          </div>
        </section>
        <section>
          <h2 class="title" id="views">Корисні посилання</h2>
          <p class="desc">Корисні посилання та програми</p>
          <div class="link_list animate">
            <div class="link_item">
              <button type="button" class="collapsible">Корисні програми</button>
              <div class="content">
              <p>Lorem ipsum...</p>
              </div>
              <button type="button" class="collapsible">Статті та форуми</button>
```

```
                    <div class="content">
                    <p>Lorem ipsum...</p>
                    </div>
                </div>
                <div class="link_item">
                    <button type="button" class="collapsible">Корисні посилання та канали</button>
                    <div class="content">
                    <p>Lorem ipsum...</p>
                    </div>
                    <button type="button" class="collapsible">Документація</button>
                    <div class="content">
                    <p>Lorem ipsum...</p>
                    </div>
                </div>
            </div>
            </section>
    </div>
    <script src="scripts/script.js"></script>
</body>
</html>
```

About.js

```
window.addEventListener('scroll',e =>{
    document.body.style.cssText = `--scrollTop: ${this.scrollY}px`
})
```

main.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Main site</title>
    <script src="scripts/cacheScript.js"></script>
    <link rel="stylesheet" href="stiles/normalize.min.css">
    <link rel="stylesheet" href="stiles/style.css">
</head>
<body>
    <header class="header">
        <a href="main.html" class="header_logo"></a>
        <div class="form">
            <input type="search" class="form_input" placeholder="Serch your information">
        </div>
        <nav class="header_nav">
            <ul class="header_list">
                <li class="header_item"> <a class="header_link" href="/sites/about.html"> Про нас
</a></li>
            </ul>
        </nav>
```

```
        </header>
        <div class="container">
          <section>
            <div class="main">
              <div class="main_text">
                <h1 class="main_title animate">Зроби гру своєї мрії та прояви себе </h1>
                <p class="main_desc animate">Ти дізнаєшся основи зі створення ігрових додатків
на Unity,
                   зрозумієш яким чином програмуються ігрові механіки
                </p>
                <ul class="main_list animate">
                  <li class="main_item"><span class="main_span">10+</span>Статті</li>
                  <li class="main_item"><span class="main_span">10+</span>Проектів</li>
                </ul>
              </div>
              <div class="main_image">
                <img class="main_img animate" src="./stiles/images/foto1.png" alt="">
              </div>
            </div>
          </section>
          <section>
          <h2 class="title">Вибери статтю для навчання</h2>
          <p class="desc">Для поновлення навичок є можливість розглянути різні статті</p>
          <div class="card animate">
            <div class="card_item">
              <div class="card_cover">
                <img class="card_img" src="../sites/stiles/images/card_foto1.png" alt="">
              </div>
              <div class="card_info">
                <p class="card_main_inf">Введення в Unity</p>
                <p class="card_disc">@Unity</p>
              </div>
              <a href="blogUnity.html" class="card_link">Перейти</a>
            </div>
            <div class="card_item">
              <div class="card_cover">
                <img class="card_img" src="../sites/stiles/images/card_foto2.png" alt="">
              </div>
              <div class="card_info">
                <p class="card_main_inf">Базові функції в Unity</p>
                <p class="card_disc">@Unity</p>
              </div>
              <a href="blogUnity.html" class="card_link">Перейти</a>
            </div>
            <div class="card_item">
              <div class="card_cover">
                <img class="card_img" src="../sites/stiles/images/card_foto3.png" alt="">
              </div>
              <div class="card_info">
                <p class="card_main_inf">Цікаві лайфхаки</p>
                <p class="card_disc">@Unity</p>
```

```
            </div>
            <a href="blogUnity.html" class="card_link">Перейти</a>
          </div>
        </div>
        </section>
        <section>
        <h2 class="title">Корисні посилання</h2>
        <p class="desc">Корисні посилання та програми</p>
        <div class="link_list animate">
          <div class="link_item">
            <button type="button" class="collapsible">Корисні програми</button>
            <div class="content">
            <p>Lorem ipsum...</p>
            </div>
            <button type="button" class="collapsible">Статті та форуми</button>
            <div class="content">
            <p>Lorem ipsum...</p>
            </div>
          </div>
          <div class="link_item">
            <button type="button" class="collapsible">Корисні посилання та канали</button>
            <div class="content">
            <p>Lorem ipsum...</p>
            </div>
            <button type="button" class="collapsible">Документація</button>
            <div class="content">
            <p>Lorem ipsum...</p>
            </div>
          </div>
        </div>
        </section>
    </div>
<footer class="footer_stile"></footer>
<script src="scripts/script.js"></script>
<script src="scripts/anim.js"></script>
</body>
</html>
```

Anim.js

```
document.addEventListener("DOMContentLoaded", () => {

        // Use Intersection Observer to determine if objects are within the viewport
        const observer = new IntersectionObserver(entries => {
          entries.forEach(entry => {
                if (entry.isIntersecting) {
                  entry.target.classList.add('in-view');
                  return;
                }
                entry.target.classList.remove('in-view');
          });
```

```
        });

        // Get all the elements with the .animate class applied
        const allAnimatedElements = document.querySelectorAll('.animate');

        // Add the observer to each of those elements
        allAnimatedElements.forEach((element) => observer.observe(element));

});
```

GalaryScroll.js

```
const sliderMain = new Swiper('.slider_main',{
   freeMode: true,
   centeredSlides:true,
   mousewheel:true,
   parallax: true,
   breakpoints:{
      0:{
         slidesPerView:2.5,
         spaceBetween:20
      },
      680:{
         slidesPerView:3.5,
         spaceBetween:60
      }
   }
})
const sliderBg = new Swiper('.slider_bg',{
   centeredSlides:true,
   parallax: true,
   spaceBetween: 60,
   slidesPerView:3.5,
})
sliderMain.controller.control = sliderBg

document.querySelectorAll('.slider__item').forEach(item => {
        item.addEventListener('click', event => {
                item.classList.toggle('opened')
        })
})
let desc = document.querySelector('.description')
sliderMain.on('slideChange', e => {
        sliderMain.activeIndex > 0 ? desc.classList.add('hidden') : desc.classList.remove('hidden')
})
```

cacheScript.js

```
document.addEventListener('DOMContentLoaded', function() {
   let db;
   const request = window.indexedDB.open("imagesCacheDB", 1);
```

```
        request.onerror = function(event) {
            console.error("Database error: ", event.target.error);
        };

        request.onupgradeneeded = function(event) {
            const db = event.target.result;
            db.createObjectStore("images", { keyPath: "url" });
        };

        request.onsuccess = function(event) {
            db = event.target.result;
            checkAndLoadImages();
        };

        function checkAndLoadImages() {
            const transaction = db.transaction("images", "readonly");
            const store = transaction.objectStore("images");
            const countRequest = store.count();

            countRequest.onsuccess = function() {
                if (countRequest.result === 0) {
                    // Кеш пустий, питаємо про кешування
                    var cacheEnabled = confirm("Включити кешування? Натисніть ОК для включення або Скасувати для вимкнення.");
                    if (cacheEnabled) {
                        cacheImages();
                    }
                } else {
                    // Кеш не пустий, завантажуємо зображення з кешу
                    loadImagesFromCache();
                }
            };
        }

        function cacheImages() {
            document.querySelectorAll('img').forEach(img => {
                const src = img.getAttribute('src');
                fetch(src).then(response => {
                    if (response.ok) {
                        return response.blob();
                    }
                    throw new Error('Network response was not ok.');
                })
                .then(blob => {
                    const tx = db.transaction("images", "readwrite");
                    const store = tx.objectStore("images");
                    const item = {
                        url: src,
                        blob: blob
                    };
                    store.add(item);
```

```
            })
            .catch(error => console.error('Could not cache image:', src, error));
        });
    }

    function loadImagesFromCache() {
        const transaction = db.transaction("images", "readonly");
        const store = transaction.objectStore("images");
        document.querySelectorAll('img').forEach(img => {
            const src = img.getAttribute('src');
            const request = store.get(src);

            request.onsuccess = function() {
                if (request.result) {
                    const url = URL.createObjectURL(request.result.blob);
                    img.src = url;
                }
            };
        });
    }
});
```

index.js

```
const http = require('http');
const fs = require('fs');
const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/html; charset=utf-8');
  switch(req.url)
  {
    case '/':
       fs.createReadStream('./sites/main.html').pipe(res);
      break;
    case '/about':
       fs.createReadStream('./sites/about.html').pipe(res);
       break;
    case '/blog':
      fs.createReadStream('./sites/blogUnity.html').pipe(res);

  }

});

server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

LazyLoading.js

```
document.addEventListener('DOMContentLoaded', () => {
    const images = document.querySelectorAll('img');

    if ('IntersectionObserver' in window) {
        // Встановлення data-src та очищення src для усіх зображень
        images.forEach(img => {
            img.dataset.src = img.src;
            img.src = ''; // Очистити src або встановити запасне зображення
        });

        let lazyImageObserver = new IntersectionObserver((entries, observer) => {
            entries.forEach((entry) => {
                if (entry.isIntersecting) {
                    let lazyImage = entry.target;
                    lazyImage.src = lazyImage.dataset.src; // Відновити src з data-src
                    lazyImageObserver.unobserve(lazyImage); // Припинити спостереження
                }
            });
        });

        images.forEach((lazyImage) => {
            lazyImageObserver.observe(lazyImage); // Почати спостереження
        });
    } else {
        // Для браузерів, що не підтримують Intersection Observer
        images.forEach(img => {
            img.src = img.dataset.src;
        });
    }
});
```