Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет навчально-науковий центр заочної форми навчання
(повна назва)

Кафедра програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Дослідження методів програмної оптимізації завантаження сайтів на формування досвіду користувачів
(тема)

Виконала:
здобувач 2 року навчання
групи ІПЗзм-23-1

Лілія НУРАЛІЄВА
(Власне ім'я, ПРІЗВИЩЕ)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)
Тип програми освітньо-наукова

Керівник доц. Олексій НАЗАРОВ
(посада, Власне ім'я, ПРІЗВИЩЕ)

Допускається до захисту
Зав. кафедри _____ Кирило СМЕЛЯКОВ
(підпис) (Власне ім'я, ПРІЗВИЩЕ)

2025 р.

Харківський національний університет радіоелектроніки

Факультет навчально-науковий центр заочної форми навчання
Кафедра програмної інженерії
Рівень вищої освіти другий (магістерський)
Спеціальність 121 – Інженерія програмного забезпечення
Тип програми освітньо-наукова програма
Освітня програма Інженерія програмного забезпечення
(шифр і назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«___» _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачці Нуралієвій Лілії Магсудівні
(прізвище, ім'я, по батькові)

1. Тема роботи «Дослідження методів програмної оптимізації завантаження сайтів на формування досвіду користувачів»
Затверджена наказом по університету від 10.04.2025 № 55 Стз
2. Термін подання здобувачем роботи до екзаменаційної комісії 02.06.2025
3. Вихідні дані до роботи статистичні дані щодо швидкості завантаження сторінок, аналітика метрик Web Vitals з використанням інструментів Lighthouse та Google PageSpeed Insights, приклади реальних сайтів з різним рівнем оптимізації продуктивності
4. Перелік питань, що потрібно опрацювати в роботі
аналіз ключових показників продуктивності вебсайтів (Web Vitals) та їхнього впливу на користувацький досвід, методи зменшення часу завантаження сторінок, проектування двох моделей вебсайту для проведення порівняльних експериментальних досліджень та аналіз отриманих результатів

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
	Отримання завдання	квітень 2025 р.	*виконано*
	Аналіз предметної галузі і постановка задачі	квітень 2025 р.	*виконано*
	Назва за розділами теоретичного і практичного дослідження	квітень 2025 р.	*виконано*
	Назва за розділами теоретичного і практичного дослідження	квітень 2025 р.	*виконано*
	Підготовка до апробації результатів дослідження. Публікація матеріалів	травень 2025 р.	*виконано*
	Назва за розділами теоретичного і практичного дослідження	травень 2025 р.	*виконано*
	Підготовка пояснювальної записки	травень 2025 р.	*виконано*
	Підготовка презентації та доповіді	травень 2025 р.	*виконано*
	Перевірка на плагіат	червень 2025 р.	*виконано*
	Нормоконтроль	червень 2025 р.	*виконано*
	Рецензування	червень 2025 р.	*виконано*
	Попередній захист	червень 2025 р.	*виконано*
	Занесення диплома в електронний архів	червень 2025 р.	*виконано*
	Допуск до захисту у зав. кафедри	червень 2025 р.	*виконано*

Дата видачі завдання 07.04.2025

Здобувачка _____ Лілія НУРАЛІЄВА
(підпис)

Керівник роботи _____ доц. Олексій НАЗАРОВ
(підпис) (посада, Власне ім’я, ПРІЗВИЩЕ)

РЕФЕРАТ / ABSTRACT

Пояснювальна записка містить: 64 с., 14 рис., 31 джерело.

ДОСЛІДЖЕННЯ МЕТОДІВ ПРОГРАМНОЇ ОПТИМІЗАЦІЇ, ЗАВАНТАЖЕННЯ САЙТІВ, ФОРМУВАННЯ ДОСВІДУ КОРИСТУВАЧІВ.

Об’єктом дослідження є методи програмної оптимізації завантаження сайтів та їх вплив на формування досвіду користувачів.

Метою роботи є аналіз і розробка програмних рішень для оптимізації швидкості завантаження вебсайтів з урахуванням їхнього впливу на досвід користувачів.

Методами дослідження є аналіз сучасних технологій оптимізації вебсайтів, оцінка їх ефективності на основі ключових показників продуктивності (KPIs) та розробка рішень для покращення швидкості завантаження.

У результаті кваліфікаційної роботи було проведено аналіз ключових методів оптимізації завантаження сайтів. Ефективність запропонованих рішень оцінювалася за такими показниками, як час завантаження сторінки, затримка першого відображення контенту та час відмальовування найбільшого елементу на сторінці.

RESEARCH ON SOFTWARE OPTIMIZATION METHODS FOR WEBSITE LOADING AND THEIR IMPACT ON USER EXPERIENCE.

The subject of the study is the methods of software optimization for website loading and their impact on user experience.

The objective of the work is to analyze and develop software solutions to optimize website loading speed, taking into account their impact on user experience.

The methods of the study include analyzing modern website optimization technologies, evaluating their effectiveness based on key performance indicators (KPIs), and developing solutions to improve loading speed.

As the result of the qualification work, an analysis of key website loading optimization methods was conducted. The effectiveness of the proposed solutions was evaluated using metrics such as page load time, first contentful paint delay, and the largest contentful paint delay.

Завідувачу кафедри
ПІ_____
(скорочена назва кафедри)
проф. Кирилу СМЕЛЯКОВУ_
(вчене звання, сласне ім'я, прізвище)

ЗАЯВА
щодо самостійності виконання кваліфікаційної роботи та можливості її публікації (та/або публікації анотації кваліфікаційної роботи) в електронному архіві відкритого доступу ElAr KhNURE

Я, Нуралієва Лілія Магсудівна, здобувачка гр. ІПЗзм-23-1, здобувач вищої освіти на другому (магістерському) рівні кафедри «Програмна інженерія», заявляю: мій курсовий проєкт на тему «Дослідження методів програмної оптимізації завантаження сайтів на формування досвіду користувачів», що буде представлений для публічного захисту, виконаний самостійно, не містить елементи плагіату і може бути опублікований в електронному архіві з відкритим доступом ElArKhNURE. Всі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлена з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», відповідно до якого виявлення плагіату є підставою для відмови в допуску роботи до захисту та застосування дисциплінарних заходів.

Дата 20.05.2025 Підпис

ЗМІСТ

Перелік скорочень 8
Вступ 9
1 Аналіз предметної галузі 10
1.1 Аналіз предметної галузі дослідження 10
1.2 Постановка задачі 11
2 Опис прийнятих проєктних рішень 16
2.1 Ключові показники продуктивності (KPIs) вебресурсів 16
2.2 Аналіз та вибір методів програмної реалізації 23
3 Опис програмної реалізації 310
4 Опис експериментальних досліджень 38
Висновки 40
Перелік джерел посилання 42
Перелік джерел посилання за науковими напрямами керівника та науковців кафедри програмної інженерії 45
Додаток А Звіт результатів перевірки на унікальність тексту в базі ХНУРЕ 46
Додаток Б Слайди презентації 48
Додаток В Апробація результатів роботи 57
Додаток Г Експертний висновок результатів перевірки кваліфікаційної роботи на відповідність оформлення вимогам ДСТУ 3008: 2015 64

ПЕРЕЛІК СКОРОЧЕНЬ

KPIs – Key Performance Indicators

FCP – First Contentful Paint

LCP – Largest Contentful Paint

INP – Interaction to Next Paint

CLS – Cumulative Layout Shift

TTFB – Time to First Byte

HTTP – Hypertext Transfer Protocol

SSR – Server-Side Rendering

SSG – Static Site Generation

UI – User Interface

UX – User Experience

REST – Representational State Transfer

ВСТУП

У сучасному світі цифрових технологій вебсайти стали його невід'ємною частиною. Швидкість завантаження вебресурсів відіграє критично важливу роль у формуванні досвіду користувачів та відповідно відображається у рейтингах пошукових систем.

Оптимізація швидкості завантаження вебсайтів включає в себе як технічні, так і користувацькі аспекти. Це не лише покращення продуктивності, скорочення часу відгуку мережі, зменшення розміру файлів і оптимізація коду, а й створення позитивного досвіду для відвідувачів, що вимагає впровадження інтуїтивно зрозумілих і швидких інтерфейсів. Тому ефективна оптимізація завантаження сайтів є складовою частиною успіху вебпроєкту.

Актуальність цієї теми обумовлена стрімким розвитком вимог користувачів до якості вебресурсів і конкуренцією між вебсайтами. Час завантаження став важливим критерієм для будь-якого вебсайту, і його вдосконалення може значно підвищити рівень задоволення і показників користуванням сайтом. У зв'язку з цим, оптимізація завантаження сайтів є актуальним і необхідним завданням для підтримки конкурентоспроможності та досягнення результатів на цифровому ринку.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз предметної галузі дослідження

Швидкість завантаження вебсайтів є одним із ключових факторів, що визначають якість взаємодії користувачів із цифровими продуктами. Затримки під час завантаження сторінок негативно впливають на рівень задоволеності користувачів, підвищують ймовірність їхнього відтоку та знижують ефективність бізнес-процесів.

Дослідження показують, що навіть незначне збільшення часу завантаження може призвести до меншої взаємодії, зниження кількості повторних відвідувань і зменшення довіри до бренду. Користувачі очікують швидкий доступ до інформації, і кожна секунда затримки суттєво знижує їхнє задоволення від роботи із вебсайтом.

Висока швидкість завантаження також впливає на позицію сайту в пошукових системах, зокрема Google, який використовує цей показник як один із факторів ранжування. Таким чином, забезпечення оптимальної швидкості є важливим як для формування позитивного користувацького досвіду, так і для підтримки конкурентоспроможності.

Рекомендація утримувати затримки на рівні менше ніж 2 секунди була широко цитована і залишалася «золотим стандартом» вебдизайну аж до 1990-х років [1]. Ідея полягала в тому, що швидкий відгук системи не давав користувачам відчуття затримки чи уповільнення в роботі. Якщо вебсайт довго завантажується, це може викликати розчарування, і користувач може подумати, що система не працює належним чином. Швидкий відгук допомагає забезпечити зручний і ефективний користувацький досвід, що важливо для підтримування інтересу користувачів. Таким чином можна сказати, що час відгуку має вирішальне значення для збереження уваги користувачів і для забезпечення їхньої задоволеності користуванням ресурсом. Чим довше затримка, тим більша ймовірність того, що користувачі відмовляться від подальшої взаємодії і перейдуть до конкурентів. Користувачі очікують, що сторінка завантажуватиметься до 2-х секунд, а вже за три секунди до 40%

користувачів можуть залишити ваш сайт. Більше того, 85% мобільних користувачів очікують, що сайти на їхніх пристроях завантажуватимуться так само швидко або ще швидше, ніж на десктопах [2].

Проблеми зручності використання вебсайтів часто виникають через низку факторів, які можуть значно погіршити взаємодію користувача з ресурсом. Коли користувачі не можуть швидко зрозуміти, як працює сайт, чи не можуть знайти потрібну інформацію, це знижує їхнє задоволення від використання. Такі труднощі можуть спричинити втрату інтересу та відмову від подальшого використання сайту. Складний для розуміння контент, непослідовні формати, труднощі з навігацією, дезорієнтація, відсутність взаємодії та надійності, неефективні пошукові можливості та нечітко визначені функції допомоги є проблемами зручності використання, які часто виявляються на комерційних вебсайтах [3].

1.2 Постановка задачі

У сучасному цифровому світі продуктивність вебсайтів відіграє ключову роль у забезпеченні якісного користувацького досвіду. Швидкість завантаження сторінки, інтерактивність елементів і візуальна стабільність – це основні аспекти, які визначають задоволеність користувачів та їхню взаємодію з вебресурсами. Затримки в завантаженні або некоректна робота елементів сторінки можуть призвести до втрати аудиторії ресурсу.

Web Vitals – це ініціатива від Google, яка надає єдині рекомендації щодо ключових показників якості, необхідних для забезпечення належного користувацького досвіду в інтернеті [4]. Web Vitals спрямована на те, щоб допомогти розробникам і власникам вебсайтів зрозуміти, які аспекти продуктивності вебсторінок є найбільш важливими для користувачів. В рамках цієї ініціативи Google визначає набір основних показників, які охоплюють ключові аспекти зручності користування сайтом.

Швидкість завантаження сторінок впливає не лише на досвід користувачів, а й на показники SEO. Google активно враховує продуктивність

сайту під час ранжування, що робить оптимізацію важливою складовою розвитку будь-якого онлайн-ресурсу. Завдяки використанню інструментів, таких як Google PageSpeed Insights, Lighthouse і Search Console, можна отримати дані про ці метрики сторінок і побачити практичні рекомендації для їх покращення.

Задача оптимізації завантаження вебресурсів полягає у зменшенні часу, необхідного для завантаження вебсторінок, що є критичним фактором для забезпечення позитивного користувацького досвіду.

Моніторинг і аналіз продуктивності вебресурсів є важливою частиною оптимізації, що дозволяє відстежувати ефективність завантаження сторінок і вчасно виявляти можливі проблеми. Для цього використовуються такі інструменти, як Google PageSpeed Insights, Lighthouse і WebPageTest, які надають детальні звіти про час завантаження, ефективність роботи сторінок і надають рекомендації щодо покращення цих метрик.

Чим швидше завантажується сторінка, тим менше користувачів покидають сайт через повільну роботу. Повільне завантаження може призвести до того, що відвідувачі не дочекаються повного завантаження і покинуть сторінку до того, як вона стане доступною для взаємодії. Користувачі очікують, що вебсторінки завантажуватимуться не більше пари секунд, і якщо цей час перевищується, ймовірність виходу з сайту збільшується.

До задач оптимізації входять скорочення часу завантаження сторінок шляхом мінімізації розміру файлів, зменшення кількості HTTP-запитів та використання ефективних методів кешування. Крім того, важливим аспектом є оптимізація зображень та інших медіафайлів, щоб забезпечити їх швидке завантаження без втрати якості. Використання асинхронного завантаження ресурсів і застосування технік lazy loading також допомагає скоротити час до першого відображення сторінки, дозволяючи користувачам взаємодіяти з основними елементами навіть до завершення повного завантаження.

Додатково, стратегія оптимізації може включати в себе розробку інтелектуальних алгоритмів для управління ресурсами, такими як пам'ять та обчислювальна потужність [5].

Фокус на оптимізації фронтенду значно ефективніший для покращення загальної продуктивності вебсайту, ніж на бекенді. Стейв Соудерс у своїй книзі зазначає, що: “Якщо скоротити час відповіді бекенду вдвічі, то загальний час відповіді для кінцевого користувача зменшився б лише на 5-10%. Якщо ж, робити те ж саме з фронтендом, то загальний час відповіді скоротиться на 40-45%.” [6]. Це зумовлено тим, що саме фронтенд обробляє безпосередню взаємодію з користувачем, тоді як час, витрачений на обробку запитів на бекенді, часто є менш критичним у загальному процесі завантаження сторінки. Тому зусилля з оптимізації слід спрямовувати на зменшення ваги сторінок, використання ефективних форматів даних, оптимізацію скриптів і зображень, а також застосування сучасних підходів до рендерингу.

Продуктивність завантаження вебсайту є дуже важливим фактором для забезпечення належного користувацького досвіду. Час завантаження сторінки впливає на те, чи залишиться користувач на сайті або ж покине його, що безпосередньо позначається на рівні взаємодії юзерів з контентом.

Швидкі сторінки не тільки поліпшують досвід користувача, але й мають позитивний вплив на рейтинг сайту в пошукових системах. І оскільки Google® використовує швидкість завантаження вебсайту як фактор для визначення його рейтингу, настав час для веб-розробників серйозно взятися за оптимізацію своїх сайтів для забезпечення швидкого відгуку [7]. Правильна оптимізація може допомогти значно зменшити час завантаження та підвищити ефективність роботи сайту, що, в свою чергу, сприяє збільшенню кількості відвідувачів і покращенню їхнього досвіду.

Оптимізація вебресурсів включає кілька важливих аспектів, серед яких зменшення розміру медіафайлів, таких як зображення та відео, через використання сучасних форматів стиснення, наприклад, WebP для зображень, що дозволяє зменшити їхній розмір без значних втрат якості. Крім того,

важливо мінімізувати розмір CSS та JavaScript файлів, застосовуючи методи мініфікації та об'єднання, що дозволяє зменшити кількість запитів до серверу та покращити швидкість завантаження сторінок, забезпечуючи більш ефективне використання ресурсів. Навіть зменшення використання тегів сильно впливає на зменшення ресурсу. Виключення необов'язкових тегів зберігає вашу HTML-структуру семантично правильною, одночасно зменшуючи розмір файлу та роблячи ваш код значно чистішим. У типовому документі це може означати економію на рівні 5-20% [7].

Кешування є іншою важливою стратегією для покращення швидкості завантаження вебресурсів, і воно включає впровадження механізмів кешування як на рівні браузера, так і на сервері. Це дозволяє уникнути повторного завантаження однакових ресурсів при кожному відвідуванні сторінки, що значно зменшує час завантаження та навантаження на сервер. Крім того, використання Content Delivery Networks (CDN) дає можливість зберігати статичні ресурси на декількох серверах, розташованих по всьому світу, і доставляти їх користувачам з найближчого серверу. Це знижує час завантаження завдяки географічній близькості серверів та дозволяє швидко обробляти запити незалежно від місцезнаходження користувача. Коли браузер може використовувати локальну копію, це економить час на встановлення з'єднання та час на завантаження [7].

Асинхронне завантаження є важливим елементом для оптимізації часу завантаження вебсторінок. Воно передбачає завантаження JavaScript та CSS файлів поступово та не одразу – таким чином не блокується рендеринг сторінки, що дозволяє користувачам бачити зміст сторінки, навіть якщо деякі скрипти чи стилі ще не завантажені. До того ж, використання технік lazy loading для зображень та інших ресурсів, які не є критичними для початкового рендерингу сторінки, дозволяє завантажувати ці ресурси лише тоді, коли вони попадають в поле зору користувача. Це зменшує час завантаження сторінки та знижує навантаження на мережу. Основна перевага lazy loading полягає в тому, що він зменшує кількість HTTP-запитів при початковому завантаженні

сторінки, оскільки запити надаються тільки для зображень, що знаходяться в межах видимої області [8].

Оптимізація запитів до серверів включає зменшення кількості HTTP-запитів, що відправляються під час завантаження сторінки. Цього можна досягти за допомогою комбінування ресурсів, наприклад об'єднання CSS і JavaScript файлів, або використання спільних ресурсів для декількох сторінок. Крім того, використання API для отримання лише необхідних даних замість завантаження великих обсягів інформації дозволяє значно скоротити час завантаження сторінки, адже замість отримання всіх даних із бази, клієнт отримує тільки актуальну інформацію, що забезпечує більш ефективне використання ресурсів. Навантаження від створення нового запиту на з'єднанні з високою затримкою є досить великим, тому чим менше запитів надсилається на сервер, тим швидше завантажується сторінка [9]. Особливо це стає критичним при роботі з великими обсягами даних або у випадках, коли одночасно працюють багато користувачів.

У таких умовах важливу роль відіграє ефективна організація доступу до даних. Система управління базами даних може розподіляти дані на різних вузлах мережі, що забезпечує паралельний доступ до даних і прискорює роботу з ними [10]. Це дозволяє не лише знизити навантаження на окремі сервери, а й підвищити масштабованість та стабільність вебдодатку в умовах зростаючого трафіку.

Ще існує поняття зовнішньої оптимізації. Зовнішня оптимізація включає лінкбілдінг (створення зовнішніх посилань на сайт), активність у соціальних мережах, відгуки, рейтинги для залучення уваги та авторитетності до сайту [11].

2 ОПИС ПРИЙНЯТИХ ПРОЄКТНИХ РІШЕНЬ

2.1 Ключові показники продуктивності (KPIs) вебресурсів

Ключові показники продуктивності (KPIs) вебресурсів – це метрики, які використовуються для оцінки ефективності роботи вебсайту або вебдодатку. Вони допомагають визначити, наскільки добре вебресурс виконує свої завдання та задовольняє потреби користувачів. Основні KPIs для вебресурсів включають:

Час до першого відображення (FCP) – це час, який проходить від моменту переходу на сторінку до того моменту, коли браузер починає відображати щось на екрані [12]. Оптимізація цього процесу є важливою для покращення користувацького досвіду, адже чим швидше відображаються перші елементи, тим менше користувачів чекають на повне завантаження сторінки (див.рис.2.1).

Визначення та відокремлення критичних ресурсів від решти, а також завантаження лише тих файлів, які необхідні спочатку, може покращити час до першого відображення вебсторінки [13].

Рисунок 2.1 – Метрики вимірювання часу до першого відображення (за даними [14])

Час відображення найбільшого контенту (LCP) – це один із ключових показників продуктивності вебсторінки, що вимірює час, необхідний для завантаження найбільшого видимого елемента на сторінці. Цей показник є критично важливим для оцінки швидкості завантаження, оскільки саме великі елементи контенту найчастіше формують перше враження користувача про вебресурс (див.рис.2.2).

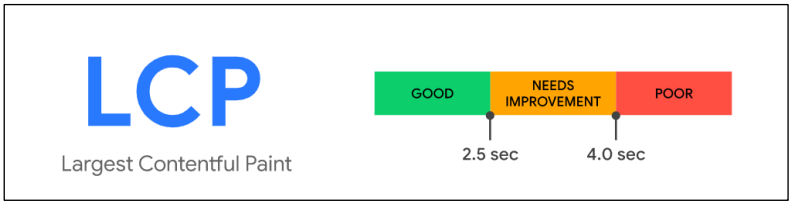

Рисунок 2.2 – Метрики вимірювання часу до відображення найбільшого контенту (за даними [15])

LCP зосереджується на реальному користувацькому досвіді, вимірюючи час до відображення найбільшого елемента в області перегляду браузера, яким може бути текстовий блок (великий заголовок чи параграф), зображення (фотографія, банер або фонова картинка), відео-фрейм, елемент <svg> або будь-який інший блоковий елемент із візуальним наповненням. Щоб забезпечити гарний користувацький досвід, сайти повинні досягати часу відображення найбільшого контенту за 2,5 секунди або менше [15].

Взаємодія з першим інтерактивним елементом (INP) – метрика, яка оцінює загальну інтерактивність вебсторінки, вимірюючи час, що проходить між взаємодією користувача (будь-яким інтерактивом) і моментом, коли сторінка реагує на цю дію.

Дані про використання Chrome показують, що 90% часу користувач проводить на сторінці після її завантаження (див.рис.2.3). Таким чином, важливо ретельно вимірювати швидкість відгуку протягом всього життєвого циклу сторінки. Саме це оцінює метрика INP [16].

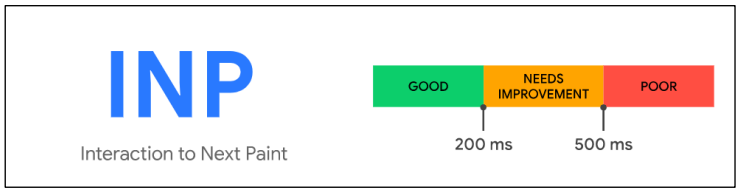

Рисунок 2.3 – Метрики вимірювання часу до взаємодії з першим інтерактивним відображенням (за даними [16])

200 мілісекунд вважається хорошим значенням INP, тоді як значення понад 500 мілісекунд вказує на погану продуктивність.

Кумулятивне зміщення макета (CLS) – це метрика, що вимірює візуальну стабільність вебсторінки (див.рис.2.4). Вона визначає, наскільки елементи на сторінці зміщуються під час її завантаження. Цей показник є важливим, оскільки раптові зміщення контенту можуть погіршити користувацький досвід, особливо коли користувачі намагаються взаємодіяти з елементами сторінки, які ще не були повністю завантажені або не є стабільними. CLS є важливою метрикою, орієнтованою на користувача, для вимірювання візуальної стабільності, оскільки допомагає кількісно оцінити, як часто користувачі стикаються з неочікуваними зміщеннями макета – низький рівень зміщення допомагає забезпечити приємне враження від веб-сторінки [17].

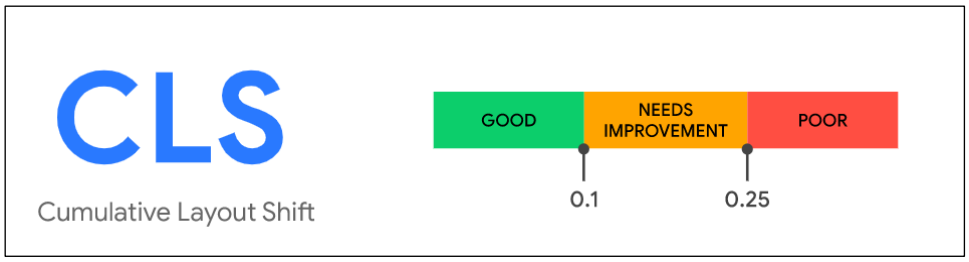

Рисунок 2.4 – Метрики вимірювання часу кумулятивного зміщення макета (за даними [17])

Показник CLS в 0.1 секунду вважається гарним рівнем для забезпечення стабільності макета сторінки і покращення користувацького досвіду.

Час до завантаження першого байта (TTFB) – це метрика, яка вимірює час між запитом ресурсу і моментом початку надходження першого байта відповіді [13] (див.рис.2.5).

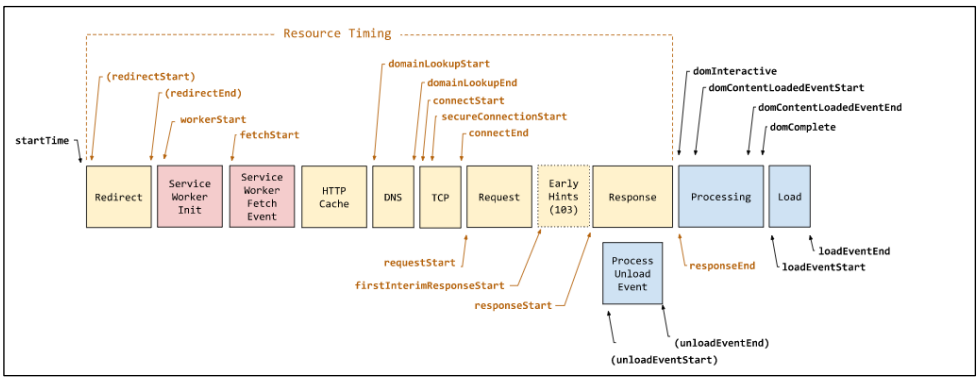

Рисунок 2.5 – Діаграма фаз мережевих запитів та пов'язаних з ними тимчасових рамок (за даними [18])

Гарні значення TTFB становлять 0,8 секунди або менше, а погані значення – більше 1,8 секунди (див.рис.2.6).

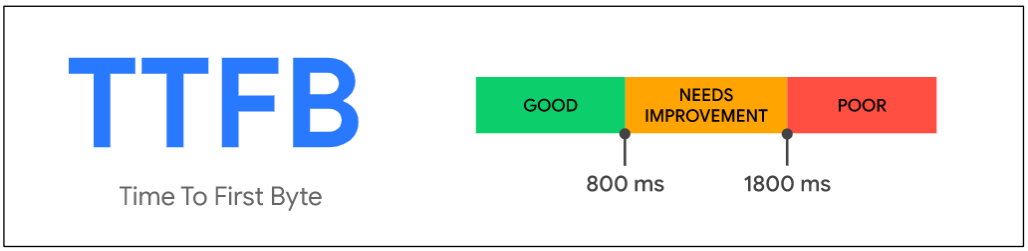

Рисунок 2.6 – Метрики вимірювання часу до завантаження першого байта (за даними [18])

Наприклад, якщо аналізувати веб-показники онлайн-магазину COMFY, який є одним із найбільших e-commerce магазинів в Україні, то можна побачити, що завдяки швидкій загрузці та оптимізації сайту наявні активні онлайн-продажі.

На рисунку 2.7 видно, що всі основні метрики Core Web Vitals для сайту COMFY відповідають рекомендованим значенням або навіть перевершують їх, що свідчить про гарну оптимізацію вебсайту.

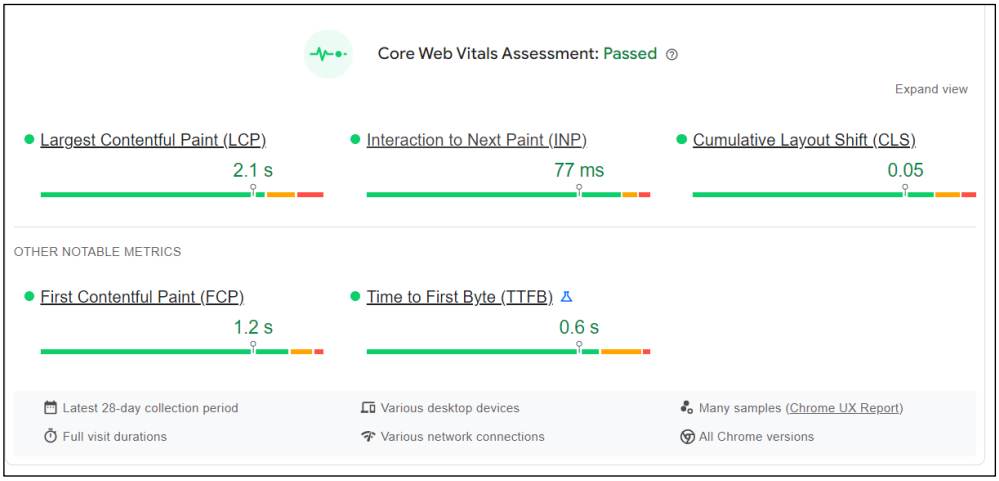

Рисунок 2.7 – Web Vitals оцінка веб-сайту Comfy (за даними [19])

LCP складає 2.1 секунди, що є хорошим показником, оскільки знаходиться у межах рекомендованого (<2.5 секунд). Це означає, що

найбільший вміст сторінки завантажується досить швидко, забезпечуючи позитивний досвід для користувачів.

INP – 77 мс є чудовим результатом (менше 200 мс). Це свідчить про високу чутливість сайту до взаємодії користувачів, що дозволяє забезпечити ефективний інтерактивний досвід.

CLS – значення 0.05 також є відмінним (менше 0.1). Це вказує на високу стабільність макета сторінки, без значних неочікуваних змін розташування елементів під час завантаження.

FCP – значення 1.2 секунди демонструє швидке завантаження першого видимого вмісту сторінки, що відповідає гарним практикам.

TTFB – значення 0.6 секунди свідчить про швидкий відгук сервера, що також позитивно впливає на загальну швидкість завантаження сайту.

Метрика перфомансу гірша за web vitals та трохи відрізняється (див.рис.2.8).

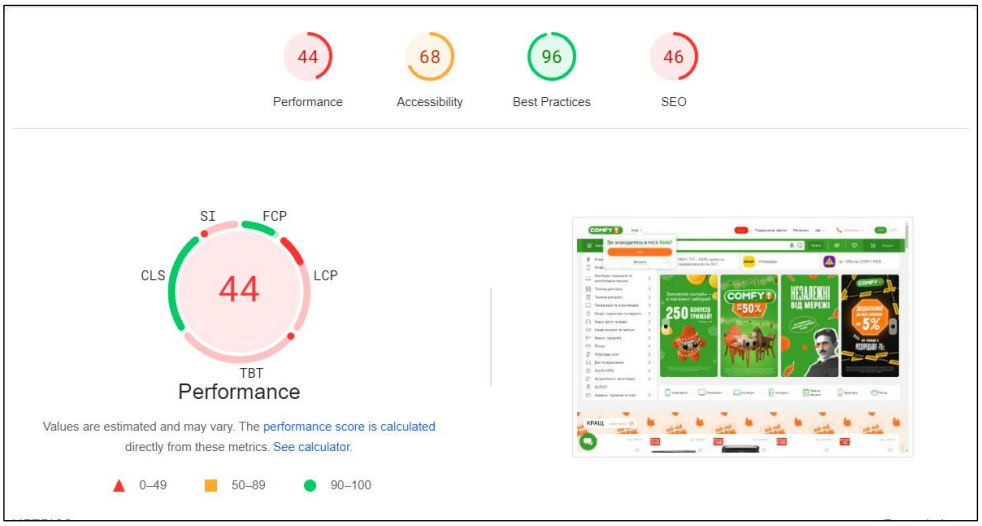

Рисунок 2.8 – Performance оцінка веб-сайту Comfy (за даними [19])

Перша метрика (для web vitals), що складає (2.1 секунди) отримана з реальних користувацьких даних. Дані збираються з реальних сесій користувачів на відміну від другої метрики, яка становить вже (2.9 секунди). Вона отримана через симуляцію в Lighthouse, де сайт тестується в особливих умовах, що включає емуляцію менш швидкого інтернет з’єднання, симуляцію менш продуктивного смартфона та відсутність кешування. Тобто, спеціально

створюються "гірші умови", щоб оцінити метрики продуктивності сайта у найгірших сценаріях. Таким чином, швидкість завантаження обмежується, що створює затримку та збільшує час на завантаження ресурсу.

Метрика LCP є проблемною як можна побачити з рисунка 4.3, оскільки перевищується оптимальний поріг у 2.5 секунди. На це впливає відображення слайдера з банерами, які займають майже всю сторінку. Тут LCP відображає час завантаження одного з найбільших елементів, таких як рекламний банер з зображеннями у каруселі (див.рис. 2.9).

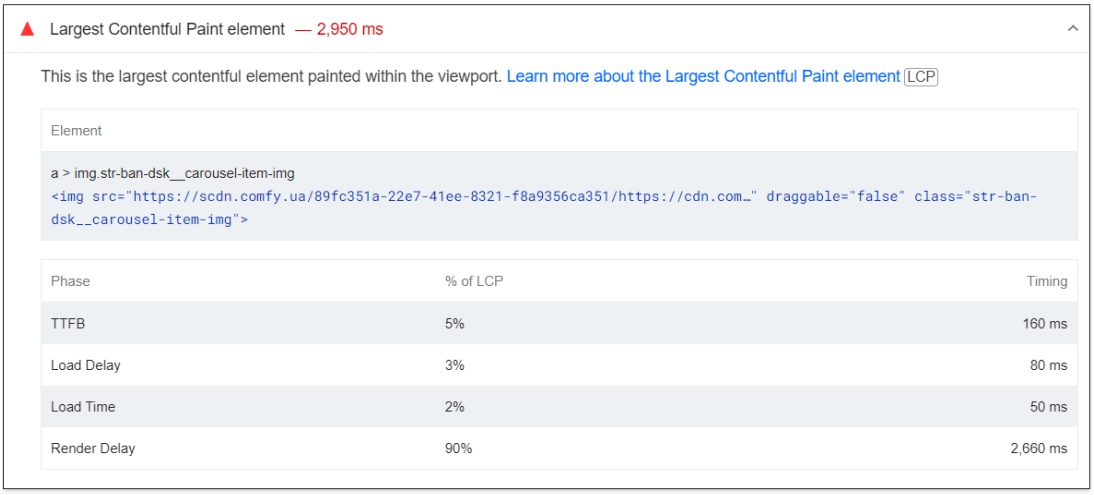

Рисунок 2.9 – Performance оцінка LCP (за даними [19])

Також є проблема з великою кількістю ресурсів, що блокують рендеринг (див. рис. 2.10). Загальний Total Blocking Time (TBT) становить 2130 ms [19], що значно впливає на час завантаження найбільшого елемента. Це пов'язано з великою кількістю JavaScript, CSS та DOM елементів, що уповільнюють завантаження сайту.

Рисунок 2.10 – Performance діагностика (за даними [19])

Оптимізація показників, зазначених у Lighthouse, включає кілька напрямків. Потрібно оптимізувати JavaScript-код за допомогою розділення коду на менші частини (code splitting), застосувати технологію tree-shaking для видалення непотрібного коду, який не використовується, оптимізувати та зменшити структуру DOM-дерева). Також необхідно запровадити lazy loading, мінімізацію та сучасні стандарти ES6+, а також видалити зайві сторонні бібліотеки; стиснути зображення у форматах WebP, AVIF та пріорітизувати завантаження ключових ресурсів; впровадити кешування.

Застосування цих підходів дозволить значно покращити продуктивність сайту, зменшити час завантаження основного контенту та підвищити задоволеність користувачів.

Також є некритичні показники, які можна покращити для оптимізації завантаження сайту. Дані показники не так багато важать, як попередні, але їх також бажано покращити для того, щоб веб-сторінка більш швидко завантажувалась (див.рис.2.11).

Рисунок 2.11 – Некритичні показники performance діагностики (за даними [19])

На рисунку в основному показано аспекти роботи з зображеннями, статичними ресурсами, компресією текста та загальним розміром мережевих запитів. Можна оптимізувати дані моменти, задавши чіткі розміри зображень для стабільності побудови макета сторінки, використавши формати зображень WebP, компресуючи текст для зменшення обсягу даних, а також налаштувати ефективне кешування статичних ресурсів для швидшого повторного завантаження. Це допоможе зменшити час завантаження сторінок, підвищити

зручність для користувачів і оптимізувати роботу сайту, що позитивно позначиться на завантаженні сайту та його рейтингу в пошукових системах.

2.2 Аналіз та вибір методів програмної реалізації

Завантаження вебсайту включає в себе декілька етапів: запит до сервера, обробку та передачу даних, рендеринг сторінки та відображення контенту в браузері користувача. Швидкість завантаження залежить від різних факторів, таких як продуктивність сервера, оптимізація коду, наявність кешування та ефективність роботи мережі.

Важливо вміти ефективно оптимізувати код, оскільки завантаження ресурсів може бути довготривалим процесом. Зображення та мультимедійні елементи, які покращують візуальне сприйняття сторінки користувачем, можуть призвести до уповільнення завантаження та створити додаткове навантаження на рендеринг сторінки.

За замовчуванням браузер завантажує всі зображення та відео відразу, тому непогано підвантажувати контент лише тоді коли це справді потрібно. Однією з найпоширеніших технік, що використовують для прискорення завантаження сторінки є “ліниве завантаження”. Це стратегія, яка визначає які ресурси є неблокуючими (некритичними) і підвантажує їх лише тоді, коли це необхідно; зазвичай це відбувається при якійсь взаємодії користувача з сайтом, таких як скролінг сторінки чи навігація [20].

Ще одна техніка, яку можна використати, це властивість preload інтерфейсу на мультимедійних елементах. Цей атрибут вказує браузеру на те, яким чином, на думку автора, буде забезпечено найкращий досвід користувача в плані завантаження ресурсу:

- none – вказує, що медіа не повинно бути попередньо завантажено;
- metadata – вказує, що повинні бути завантажені лише метадані файлу (наприклад, тривалість відео);

– auto – вказує, що весь медіафайл може бути завантажений, навіть якщо не очікується, що користувач його використовуватиме (за замовчуванням використовується самец ей підхід) [21].

Ще декілька атрибутів, які впливають на завантаження ресурсів це – prefetch, preload та preconnect. Вони сигналізують браузеру про потребу підготувати необхідні ресурси в якийсь конкретний час.

Preload – вказує браузеру заздалегідь завантажити ресурс, який знадобиться найближчим часом. Хоча в назві є термін "load", він не завантажує та не виконує скрипт, а лише ставить його завантаження та кешування з найвищим пріоритетом [22].

```
<link rel="preload" href="style.css" as="style" />
<link rel="preload" href="main.js" as="script" />
```

Preconnect – вказує браузеру, що користувач ймовірно потребуватиме ресурси з джерела цільового ресурсу, і тому слід ініціювати підключення до цього джерела заздалегідь.

```
<link rel="preconnect" href="https://example.com">
```

Prefetch – вказує браузеру, що користувач, ймовірно, потребуватиме цільовий ресурс для майбутньої навігації, і тому можна покращити досвід користувача, фоново завантажуючи та кешуючи цей ресурс; використовується для ресурсів навігації на тому ж сайті або для підресурсів, що використовуються сторінками того ж сайту [23].

```
<link rel="prefetch" href="next-page.html">
```

Google Fonts є популярним джерелом вебшрифтів, але їхнє стандартне підключення може негативно впливати на продуктивність сторінки. Оскільки браузер повинен зробити додаткові мережеві запити до серверів Google, це може збільшити First Contentful Paint (FCP) та Largest Contentful Paint (LCP).

Щоб оптимізувати завантаження Google Fonts, можна теж додати атрибут preconnect, що дозволить браузеру заздалегідь встановити з'єднання з серверами Google, зменшуючи час очікування:

```
<link rel="preconnect" href="https://fonts.googleapis.com">
```

Оптимізація зі сторони фронтенду зосереджена на покращенні швидкості завантаження та рендерингу вебсторінок у браузері користувача. Вона включає в себе компресію ресурсів та використання кешування для зменшення часу завантаження і покращення продуктивності сайту при наступних візитах сайту користувачами.

Одним із найбільш ефективних підходів є мінімізація та стиснення файлів. HTML, CSS та JavaScript файли можуть містити пробіли, коментарі та зайві символи, які не впливають на роботу самого коду, але збільшують його розмір. Для мінімізації та стиснення файлів на фронтенді існують такі популярні інструменти як Webpack, Gulp, Parcel тощо. Наприклад Webpack, який об’єднує модулі у зкомпільований файл та надає можливість розподілу коду, тобто підвантаження різних частин додатка за вимогою [24]. Parcel в цьому плані є теж не менш потужним інструментом, який допомагає сильно оптимізувати вебсайт. Наприклад коли кілька частин додатка залежать від однакових спільних модулів, вони автоматично дедуплікуються в окремий бандл автоматично, що дозволяє часто використовуваним залежностям завантажуватися паралельно з кодом додатка та кешуватися окремо браузером [25].

Велика кількість запитів до сервера теж негативно впливає на завантаження, адже кожен запит до бази даних або зовнішніх ресурсів додає затримку перед відображенням сторінки. Тому важливо оптимізувати виклик запитів з фронтенду, щоб не перенавантажувати сервер зайвими запитами. Відсутність кешування на сервері теж важлива, адже сервер змушений

обробляти кожен запит наново, що підвищує навантаження на систему та збільшує TTFB (Time to First Byte).

Не менш важливим аспектом є оптимізація коду. Використання великих бібліотек та неконтрольоване завантаження скриптів можуть блокувати та уповільнювати перший рендер сторінки, блокуючи відображення контенту. Це особливо критично для пристроїв, де продуктивність процесора не дуже висока. Зазвичай JavaScript-файли можуть блокувати рендеринг сторінки, якщо вони завантажуються синхронно. Особливо актуальним це є для великих вебдодатків, які використовують багато сторонніх бібліотек. Можна оптимізувати підвантаження скриптів за допомою атрибутів:

– defer – відкладає виконання скрипту до моменту, коли HTML-документ повністю завантажиться:

```
<script src="script.js" defer></script>
```

– async – дозволяє скрипту завантажуватися паралельно з HTML та виконується одразу після завантаження [26]:

```
<script src="script.js" async></script>
```

Також важливим кроком є перехід на сучасні формати зображень. WebP та AVIF забезпечують значно менший розмір файлів без втрати якості порівняно з PNG або JPEG. Вони підтримуються більшістю сучасних браузерів і можуть бути легко інтегровані через <picture>:

```
<picture>
  <source srcset="image.avif" type="image/avif">
  <source srcset="image.webp" type="image/webp">
  <img src="image.jpg" alt="Optimized Image">
</picture>
```

Елемент <picture> в HTML містить в собі декілька елементів <source> та один елемент <img>, щоб запропонувати альтернативні версії зображення для

різних сценаріїв відображення/пристроїв. Браузер розглядає кожен вкладений елемент <source> і вибирає найкраще відповідне зображення серед них. Якщо не знайдено відповідностей – або якщо браузер не підтримує елемент <picture> -вибирається URL атрибута src елемента <img>. Обране зображення потім відображається в просторі, зайнятому елементом <img> [27]. Таким чином, досягається краща оптимізація медіа-елементів.

Оптимізація фронтенду не завжди дає максимальний ефект, якщо серверна частина вебресурсу працює повільно. Оптимізація бекенду спрямована на зменшення часу відповіді сервера (TTFB), ефективне використання ресурсів та швидку обробку запитів.

HTTP кешування дозволяє зберігати відповіді на запити та повторно їх використовувати для подальших запитів, що надає ряд переваг. По-перше, оскільки немає необхідності відправляти запит на початковий сервер, чим ближче кеш до клієнта, тим швидше буде відповідь. Найпоширеніший приклад – це коли браузер самостійно зберігає кеш для своїх запитів. Крім того, використання кешу дозволяє знизити навантаження на початковий сервер, оскільки він не обробляє запит, не виконує додаткові операції, такі як парсинг, маршрутизація запиту, відновлення сесії через кукі, запити до бази даних або рендеринг шаблонів. Це значно зменшує навантаження на сервер та підвищує ефективність роботи системи. Коректна робота кешування є важливою для стабільності системи [28].

```
Cache-Control: max-age=31536000, public
```

CDN (Content Delivery Network) – розподіл контенту по серверах у різних частинах світу, щоб прискорити завантаження для користувачів у різних регіонах. Оскільки більшість CDN мають сервери в різних куточках світу, географічно вони можуть знаходитися ближче до користувачів, що зменшує затримку завантаження.

На відміну від поширеної думки, використання CDN може і суттєво знизити продуктивність. Підключаючись до стороннього вебсайту, браузер користувача змушений проходити більше етапів DNS-пошуку, узгодження контенту та інших процесів. Крім того, сучасні браузери не ділять кеш між різними джерелами для одного й того ж ресурсу з міркувань конфіденційності, тому користувач все одно повинен завантажувати один і той самий ресурс (наприклад, jQuery) кілька разів на різних вебсайтах [29].

Кешування у пам'яті полягає у збереженні результатів часто використовуваних запитів до бази даних у недовготривалій пам'яті, наприклад такому сховищі як Redis. Цей метод дозволяє значно скоротити час доступу до даних, оскільки операції з пам'яттю виконуються набагато швидше, ніж запити до бази даних. Наприклад, якщо певні запити до бази даних виконуються регулярно, збереження результатів цих запитів у кеші дозволяє повторно використовувати їх без необхідності звертатися до бази даних. Це знижує навантаження на сервери бази даних і значно покращує продуктивність, адже система може надавати відповіді користувачам набагато швидше.

Основні переваги кешування у пам'яті включають покращення швидкості відповіді на запити та зменшення навантаження на базу даних. Це особливо корисно для сайтів з високою відвідуваністю, де одні і ті самі запити часто повторюються.

Однак є й недоліки, бо пам'ять для кешу обмежена, і якщо кількість кешованих запитів завелика, це може призвести до швидкого заповнення пам'яті, тому потрібно звертати увагу на налаштовування очищення кешу або ж визначати термін зберігання даних у ньому. Якщо кешовані дані застарівають і не оновлюються вчасно, це може призвести до невідповідності даних, що негативно вплине на користувацький досвід.

Ще одним важливим аспектом є оптимізація роботи бази даних. Якщо запити до бази виконуються повільно, це може значно вплинути на

продуктивність всього сайту. Основні методи покращення швидкодії включають:

- індексацію полів у базі даних для швидкого пошуку інформації;
- зменшення кількості запитів через використання кешу або агрегації даних;
- розділення бази даних (sharding) для кращої масштабованості великих систем.

Реалізація пагінації теж суттєво оптимізує завантаження ресурсів. Якщо API повертає великі масиви даних (наприклад, список продуктів, публікацій, коментарів тощо), краще реалізувати пагінацію, щоб зменшити розмір відповіді і час обробки.

Окрім базових методів оптимізації, сучасна веброзробка пропонує ще кілька стратегій, що дозволяють значно покращити швидкість та зручність роботи вебресурсів.

Одним із таких підходів є SSR (Server-Side Rendering) та SSG (Static Site Generation). Вони дозволяють виконувати рендеринг сторінок на сервері або генерувати статичні HTML-файли, що зменшує навантаження на клієнтську частину додатка. Сучасні фреймворки, серед яких Next.js, Gatsby або Nuxt.js, активно використовують ці технології.

Ще однією перспективною технологією є Service Workers. Service Workers дозволяють кешувати ресурси, що робить вебдодатки доступними навіть без інтернет-з’єднання. Він працює в окремому потоці, незалежно від основного JavaScript-коду, де воркер і js код не мають безпосереднього доступу до стану один одного, але можуть обмінюватися повідомленнями. Воркери дозволяють виконувати ресурсомісткі завдання у фоновому режимі: оскільки вони працюють в окремому потоці, основний JavaScript-код, який відповідає за інтерфейс користувача, залишається респонсивним і не блокується [30].

Також варто враховувати зменшення використання сторонніх бібліотек та плагінів. Часто сайти містять занадто багато залежностей, що сповільнює

їх роботу. Використання легших альтернатив або власних оптимізованих рішень дозволяє суттєво покращити швидкодію.

3 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

У цьому розділі розглянуто реалізацію системи, що складається з клієнтської та серверної частин, розроблених для проведення експериментів з оптимізації завантаження вебсайту. Рішення побудовано з використанням сучасного інструментарію: NestJS на стороні сервера, React + Vite на стороні клієнта, а також PostgreSQL для збереження даних.

Для того, щоб провести дослідження треба створити 2 окремі проєкти для реалізації оптимізованої та неоптимізованої версій. Для розробки буде використовуватися PostgreSQL та Visual Studio Code. Фінальна візуальна версія обох вебзастосунків повинна мати однаковий інтерфейс для проведення експериментів, тому на вигляд версії вебсайту ідентичні. У серверній частині реалізовано два модулі – BooksModule та ReviewsModule, з контролерами, які відповідають за обробку запитів до відповідних колекцій даних (див. рис. 3.1).

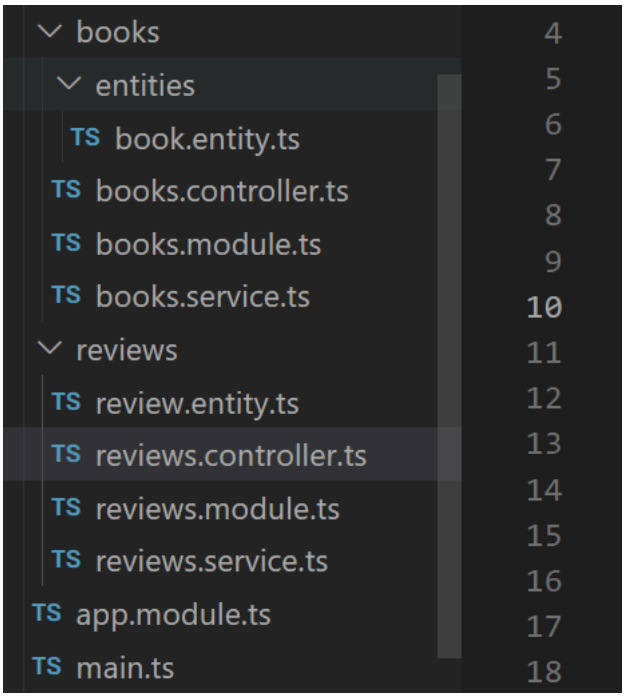

Рисунок 3.1 – Основні модулі для роботи з базою даних

Розглянемо файл app.module.ts (див. рис. 3.2), який конфігурує та запускає веб-додаток. Він містить глобальну конфігурацію через ConfigModule та налаштування підключення до бази даних PostgreSQL за допомогою TypeOrmModule.forRoot(). Логіка підключення до бази

інкапсульована в конфігурації TypeOrmModule, яка зчитує параметри з файлу оточення змінних (.env). Крім того, у головний модуль імпортуються функціональні модулі BooksModule, AuthorsModule та ReviewsModule, які інкапсулюють бізнес-логіку додатку та таким чином забезпечують розділення відповідальності.

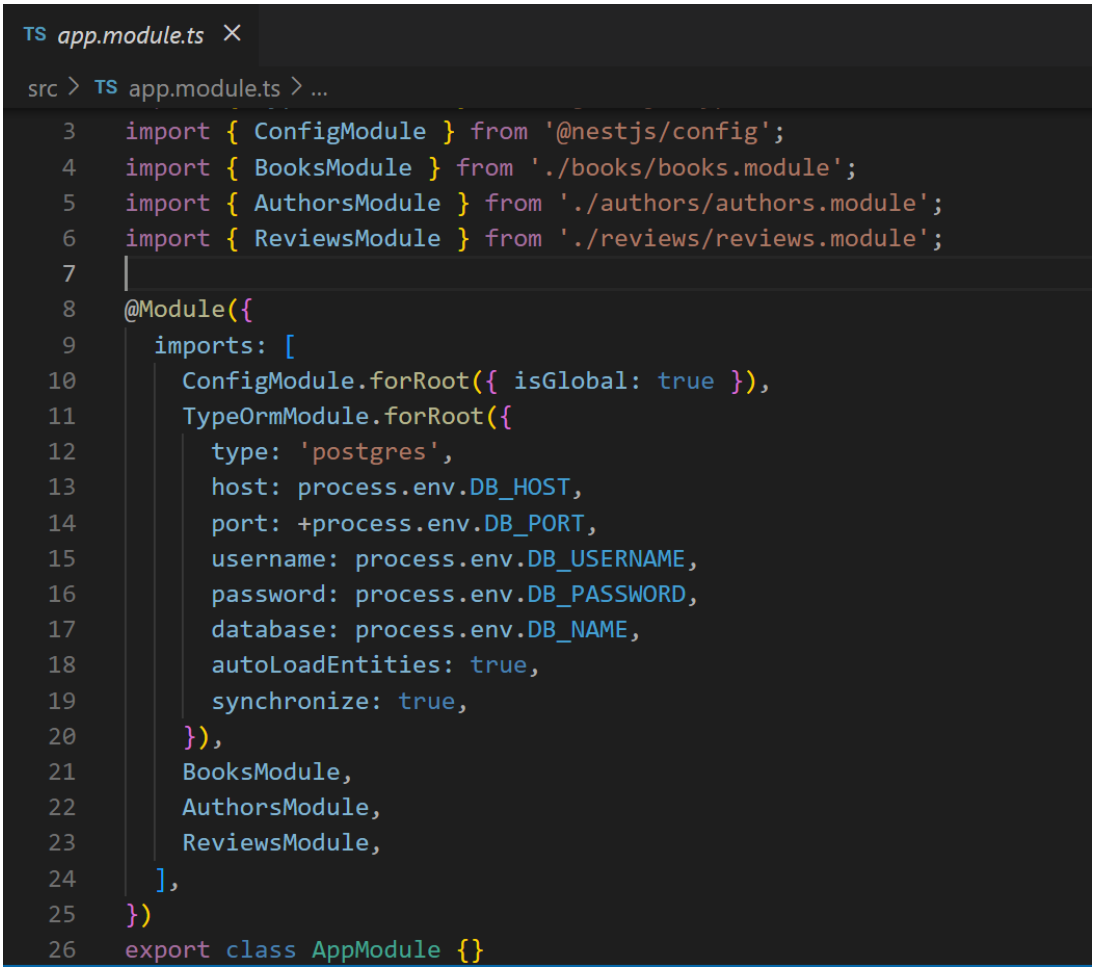

Рисунок 3.2 – Файл app.module.ts

Сутність Book у нашому проєкті моделює книгу з такими основними полями: унікальним ідентифікатором id, назвою title, опціональними описом, обкладинкою, видавництвом, роком видання, кількістю сторінок, ISBN та булевим прапором isFeatured, який вказує, чи є книга популярною.

```
import { Author } from 'src/authors/entities/author.entity';
import { Entity, Column, PrimaryGeneratedColumn, ManyToOne } from 'typeorm';

@Entity()
```

```
export class Book {
  @PrimaryGeneratedColumn()
  id: number;

  @Column()
  title: string;

  @Column({ nullable: true })
  description: string;

  @Column({ nullable: true })
  coverUrl: string;

  @Column({ nullable: true })
  publisher: string;

  @Column({ type: 'int', nullable: true })
  year: number;

  @Column({ type: 'int', nullable: true })
  pageCount: number;

  @Column({ nullable: true })
  isbn: string;

  @Column({ default: false })
  isFeatured: boolean;

  @Column({ nullable: true })
  genre: string;

  @Column({ type: 'int', nullable: true, default: 0 })
  views: number;

  @ManyToOne(() => Author, (author) => author.books, {
    eager: true,
    cascade: true,
  })
  author: Author;
}
```

В нашому сервісі для роботи з книгами реалізований метод findAll, який відповідає за отримання списку книг з бази даних. Метод підтримує два режими роботи – повне завантаження всіх книг та завантаження з пагінацією. Якщо метод викликається без параметру withLimit або зі значенням false, він виконує запит на отримання всіх книг одразу. Після отримання усіх записів повертається повний масив книг в полі data з полем featured = true.

```
export class BooksService {
  async findAll(page: number, limit: number) {
```

```
        const [data, total] = await
this.booksRepository.findAndCount({
          skip: (page - 1) * limit,
          take: limit,
          order: { id: 'ASC' },
          where: { isFeatured: true },
        });

        return {
          data,
          total,
          page,
          limit,
        };
      }
    }
```

Якщо метод викликається з параметром withLimit = true, виконується пагінований запит, який обмежує кількість отриманих книг на сторінку. У нашому випадку це 10 записів на сторінку (limit = 10), а початкова сторінка – перша (page = 1). Запит здійснюється методом findAndCount, який одночасно повертає масив книг для поточної сторінки та загальну кількість книг у базі total.

Фронтенд вебзастосунку реалізовано за допомогою React. Основний компонент App відповідає за побудову головної сторінки, де компонуються ключові секції інтерфейсу. Він включає такі основні підкомпоненти:

- Hero – верхній блок із вітальним банером;
- PopularBooks – секція з популярними книжками;
- LibraryVideo – відеоблок з автоматичним відтворенням відео про бібліотеку;
- Reviews – блок із відгуками користувачів про вебсайт;
- Footer – нижній колонтитул сторінки з копірайтом.

В оптимізованій версії зображення фону у компоненті Hero використано у стиснутому (compressed) форматі, що дозволяє суттєво зменшити обсяг завантажуваних даних і покращити швидкість рендерингу сторінки. У неоптимізованій версії фон подається у вихідній якості без стиснення, що збільшує час завантаження.

Компонент PopularBooks відповідає за відображення популярних книжок на головній сторінці. У неоптимізованій версії всі книги фетчаться одразу з сервера при завантаженні сторінки. Пагінація при цьому реалізована на фронтенді – відображаються перші 10 книжок із отриманого масиву, а решта просто відсікається методом slice.

Нижче наведено код компонента PopularBooks, де використовується axios для запиту всіх книжок із бекенду, а потім у рендері відображаються перші 10 книг у вигляді сітки:

```
export default function PopularBooks() {
  const [books, setBooks] = useState<IBook[]>([]);

  useEffect(() => {
    axios
      .get(`${backendDomain}/books`)
      .then((res) => {
        setBooks(res.data.data);
      })
      .catch((err) => console.error(err));
  }, []);

  return (
    <section className="books-section">
      <h2>Popular Books</h2>
      <div className="book-grid">
        {books.slice(0, 10).map((book) => (
          <div key={book.id} className="book-card">
            <img src={book.coverUrl} alt={book.title} />
            <div className="book-content">
              <h3>{book.title}</h3>
              <p className="author">{book.author.name}</p>
              <p className="year">{book.year}</p>
            </div>
          </div>
        ))}
      </div>
    </section>
  );
}
```

Такий підхід, хоч і простий у реалізації, створює надлишкове завантаження, особливо при великій кількості книжок, і може негативно впливати на продуктивність сторінки.

В оптимізованій версії додатку дані для компонентів PopularBooks та Reviews завантажуються поступово через API з підтримкою серверної

пагінації. Зокрема, у запиті до бекенду для книжок використовується параметр withLimit=true, який дозволяє отримати обмежену кількість записів (наприклад, 10 книжок) замість усіх одразу. Це значно зменшує обсяг переданих даних і пришвидшує завантаження сторінки.

В компоненті App спочатку здійснюється окремий запит за книгами, а після їх успішного завантаження – другий запит за відгуками користувачів. Це дозволяє уникнути одночасного завантаження великого обсягу даних і розвантажує мережу.

Компонент PopularBooks у цій версії приймає масив книжок через пропси і відображає їх без додаткової пагінації на фронтенді, оскільки вона вже реалізована на сервері. Для оптимізації завантаження зображень обкладинок використовується атрибут loading="lazy", що відтерміновує їх завантаження до моменту, коли вони з’являються у видимій частині вікна переглядача.

```
<img src={book.coverUrl} alt={book.title} loading="lazy" />
```

У базовій (неоптимізованій) версії компонент LibraryVideo відтворює відео автоматично при завантаженні сторінки з увімкненими параметрами autoPlay, muted, loop, playsInline та з попереднім завантаженням (preload="auto").

```
import { LIBRARY_VIDEO_PROPS } from "./constants";
export default function LibraryVideo() {
  return (
    <video
      autoPlay
      muted
      loop
      playsInline
      preload="auto"
      poster={LIBRARY_VIDEO_PROPS.poster}
    >
      <source src={LIBRARY_VIDEO_PROPS.videoSrc} type="video/mp4"
/>
      Your browser does not support the video tag.
    </video>
```

```
  );
}
```

Однак такий підхід призводить до завантаження та відтворення відео навіть тоді, коли користувач не бачить його в полі видимості, що збільшує трафік і навантаження на браузер.

В оптимізованій версії використовується IntersectionObserver, який відстежує, коли відео стає видимим на екрані (принаймні на 50%). Лише тоді відео починає завантажуватися та відтворюватися. Відео не завантажується завчасно (preload="none"), що зменшує витрати ресурсів і покращує продуктивність сторінки.

```
import { useEffect, useRef, useState } from "react";
import { LIBRARY_VIDEO_PROPS } from "./constants";
export default function LibraryVideo() {
  const videoRef = useRef<HTMLVideoElement | null>(null);
  const [isVisible, setIsVisible] = useState(false);

  useEffect(() => {
    const observer = new IntersectionObserver(
      ([entry]) => {
        if (entry.isIntersecting) {
          setIsVisible(true);
        }
      },
      { threshold: 0.5 }
    );

    if (videoRef.current) {
      observer.observe(videoRef.current);
    }

    return () => {
      if (videoRef.current) {
        observer.unobserve(videoRef.current);
      }
    };
  }, []);

  useEffect(() => {
    if (isVisible && videoRef.current) {
      videoRef.current.play().catch(() => {});
    }
  }, [isVisible]);

  return (
    <video
      ref={videoRef}
```

```
            muted
            loop
            playsInline
            preload="none"
            poster={LIBRARY_VIDEO_PROPS.poster}
          >
            {isVisible && (
           <source  src={LIBRARY_VIDEO_PROPS.videoSrc}  type="video/mp4"
/>
            )}
            Your browser does not support the video tag.
          </video>
        );
      }
```

Таким чином, оптимізована версія покращує продуктивність завантаження за рахунок структурованої архітектури, відкладеного завантаження відео за допомогою IntersectionObserver, lazy loading зображень, та попереднього обмеженого завантаження даних.

4 ОПИС ЕКСПЕРЕМЕНТАЛЬНИХ ДОСЛІДЖЕНЬ

Після завершення тестування та проведення досліджень можна дійти до висновків, що оптимізація продуктивності вебдодатків має суттєвий вплив на швидкість завантаження сторінок та загальний досвід користувачів. Завдяки впровадженим підходам, таким як стиснення медіафайлів, ліниве завантаження (lazy loading), пагінація тощо вдалося значно покращити ключові метрики продуктивності вебсайту. Крім того, поступове завантаження контенту та розділення запитів дозволили рівномірно розподілити навантаження на сервер і клієнт, що сприяє більш стабільній та швидшій роботі вебзастосунку.

Порівняння результатів Lighthouse аналізу для неоптимізованої та оптимізованої версій вебдодатку підтверджує значний позитивний вплив від впровадження методів оптимізації. У неоптимізованій версії загальна оцінка продуктивності становила 82, тоді як після оптимізації вона зросла до 91, що демонструє покращення на приблизно 11%. Зокрема, показник Largest Contentful Paint (LCP) зменшився з 1.6 с до 1.3 с, що було досягнуто за рахунок стискання головного фонового зображення на сторінці, яке в попередній версії передавалося без оптимізації. Також важливу роль відіграла реалізація лінивого завантаження відео за допомогою Intersection Observer – у неоптимізованій версії відео починало завантаження одразу після завантаження сторінки, незалежно від його видимості. У вдосконаленій реалізації відео підвантажується лише тоді, коли потрапляє у видиму частину екрана, що зменшило початкове навантаження на мережу.

Total Blocking Time (TBT), який є критично важливим показником для взаємодії з користувачем, зменшився з 710 мс до 370 мс, що складає покращення на понад 47%. Це стало можливим завдяки поступовому, асинхронному завантаженню даних, де запити за книжками та відгуками надсилаються послідовно, а не одночасно. Speed Index, що відображає загальну швидкість відображення контенту, зменшився з 1.3 с до 1.0 с – покращення на 23%.

У результаті, з впровадженням цих методів оптимізації, загальна ефективність вебдодатку зросла: час завантаження сторінки в оптимізованій версії значно скоротився, навантаження на сервер зменшилось, а досвід користувачів покращився. Ці покращення не тільки підвищили швидкість взаємодії користувачів з додатком, але й допомогли знизити витрати на обробку запитів, що є важливим для стабільної та масштабованої роботи системи.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було проведено аналіз проблемної області дослідження, вивчено сучасні підходи до оптимізації завантаження вебресурсів та їх вплив на продуктивність вебзастосунків. Було розглянуто та реалізовано ряд ефективних методів оптимізації з боку фронтенду і бекенду, зокрема ліниве завантаження ресурсів, компресія медіафайлів, пагінація тощо. Також було розроблено та протестовано 2 версії застосунку для завантаження книжкового каталогу, в якому ці методи були застосовані на практиці. Проведені заміри продуктивності підтвердили доцільність впроваджених рішень і засвідчили суттєве покращення, призвівши до зниження навантаження на сервер та зменшення часу відповіді системи.

Узагальнюючи вищезазначене, можна сказати, що оптимізація продуктивності вебсайтів – це комплексний підхід, який безпосередньо впливає на успіх додатку в цифровому середовищі. Швидкість завантаження, зокрема метрики Largest Contentful Paint (LCP), Interaction to Next Paint (INP), Cumulative Layout Shift (CLS) та інші, визначають, наскільки зручним і швидким буде досвід користувача. Неправильне розпорядження JavaScript, CSS, сторонніми скриптами або наявність надлишкової логіки та кількості DOM-елементів можуть створювати значні перешкоди.

Метрики Web Vitals від Google допомагають розробникам фокусуватися на критичних аспектах, водночас підтримуючи SEO-оптимізацію. Такі інструменти як Lighthouse надають практичні рекомендації щодо оптимізації вебресурсу. Завдяки цьому можливо мінімізувати затримки, підвищити інтерактивність та забезпечити стабільну візуалізацію. Інвестування в оптимізацію продуктивності не тільки покращує досвід користувачів, а й сприяє досягненню бізнес-цілей, таких як зростання трафіку на онлайн-платформі. У сучасних умовах, коли користувачі очікують майже миттєвого завантаження, а пошукові системи висувають вимоги до високої продуктивності, ігнорування цих факторів може призвести до втрати конкурентних переваг.

Зважаючи на описані методи оптимізації, можна стверджувати, що досягнення високої продуктивності вебсайту є результатом комплексного підходу, що включає як покращення роботи на серверному рівні, так і оптимізацію клієнтської частини.

Правильне налаштування оптимізації продуктивності потребує врахування конкретних вимог і характеристик проєкту. Важливо зрозуміти, які ресурси є критичними для роботи сайту, а які можна оптимізувати або завантажити відкладено.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Nielsen, Jakob. “User Interface Directions for the Web.” Communications of the ACM 42, no. 1 (1999): 65-72.

2. Hogan - Lara Callender. Designing for Performance: Weighing Aesthetics and Speed. O'Reilly Media, Inc., 2014.

3. Lee, Youngjin, and K. A. Kozar. “Understanding of Website Usability: Specifying and Measuring Constructs and Their Relationships.” Decision Support Systems 52, no. 2 (2012): 450-463.

4. Web Vitals. URL: https://web.dev/articles/vitals?hl=en (дата звернення: 27.12.2024).

6. Souders, Steve. High Performance Websites: Essential Knowledge for Front-End Engineers. O'Reilly Media, September 2007.

7. Manhas, Dr. “A Study of Factors Affecting Websites Page Loading Speed for Efficient Web Performance.” International Journal of Computer Sciences and Engineering (2013).

8. Mjelde, Eivind, and Andreas L. Opdahl. “Load-Time Reduction Techniques for Device-Agnostic Web Sites.” Journal of Web Engineering (2017): 311-346.

9. Zakas, Nicholas C. “The Evolution of Web Development for Mobile Devices.” Queue 11, no. 2 (February 2013): 30–39.

12. First paint Glossary. URL: https://developer.mozilla.org/en-US/docs/Glossary/First_paint (дата звернення: 27.12.2024).

13. Natarajan, Harini, and Rashmi Rashmi. An Experimental Case Study. Spring 2017. 55 pages.

14. FCP - First Contentful Paint. URL: https://web.dev/articles/fcp?hl=en (дата звернення: 27.12.2024).

15. LCP - Largest Contentful Paint. URL: https://web.dev/articles/lcp?hl=en (дата звернення: 27.12.2024).

16. INP - Interaction to Next Paint. URL: https://web.dev/articles/inp?hl=en (дата звернення: 27.12.2024).

17. CLS - Cumulative Layout Shift. URL: https://web.dev/articles/cls?hl=en (дата звернення: 27.12.2024).

18. TTFB - Time to First Byte. URL: https://web.dev/articles/ttfb?hl=en (дата звернення: 27.12.2024).

19. PageSpeed Insights Report on Comfy. URL: https://pagespeed.web.dev/analysis/https-comfy-ua-ua/a5d26igm8w?hl=en-US&form_factor=desktop (дата звернення: 26.12.2024).

20. Lazy Loading. URL: https://developer.mozilla.org/en-US/docs/Web/Performance/Guides/Lazy_loading (дата звернення: 01.03.2025).

21. Preload. URL: https://developer.mozilla.org/en-US/docs/Web/API/HTMLMediaElement/preload (дата звернення: 01.03.2025).

22. Preload attributes. URL: https://developer.mozilla.org/en-US/docs/Web/HTML/Attributes/rel/preload (дата звернення: 01.03.2025).

23. Prefetch. URL: https://developer.mozilla.org/en-US/docs/Web/HTML/Attributes/rel/prefetch (дата звернення: 01.03.2025).

24. Webpack. URL: https://github.com/webpack/webpack (дата звернення: 01.03.2025).

25. Parcel. URL: https://parceljs.org/features/production/#compression (дата звернення: 01.03.2025).

26. Script attributes. URL: https://developer.mozilla.org/en-US/docs/Web/HTML/Element/script (дата звернення: 01.03.2025).

27. Picture. URL: https://developer.mozilla.org/en-US/docs/Web/HTML/Element/picture (дата звернення: 01.03.2025).

28. Caching. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/Caching (дата звернення: 01.03.2025).

29. CDN. URL: https://developer.mozilla.org/en-US/docs/Glossary/CDN (дата звернення: 01.03.2025).

30. Web workers. URL: https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps/Guides/Offline_and_background_operation (дата звернення: 01.03.2025).

31. GitHub - Lilly-nn/ 2025_Nuraliieva_L_M. *GitHub*. URL: https://github.com/Lilly-nn/2025_Nuraliieva_L_M (дата звернення: 30.05.2025).

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ ЗА НАУКОВИМИ НАПРЯМАМИ КЕРІВНИКА ТА НАУКОВЦІВ КАФЕДРИ ПРОГРАМНОЇ ІНЖЕНЕРІЇ

5. Дробицький Д. С. Дослідження та розробка стратегії оптимізації та підвищення продуктивності веб-додатків на базі NodeJS та React з використанням штучного інтелекту / Д. С. Дробицький ; наук. керівник доц. О. В. Вечур // Радіоелектроніка та молодь у XXI столітті: матеріали 28-го Міжнар. молодіж. форуму, 16–18 квітня 2024 р. – Харків : ХНУРЕ, 2024. – Т. 6. – С. 124–125.

10. Константинов О. Дослідження та оптимізація продуктивності баз даних у розподілених обчислювальних мережах // 25-а Міжнар. наук.-техн. конф. «Current Trends in the Development of Scientific Research in Today’s Conditions» (ICT-2024). – 2024. – 7 с.

11. Брухтій С. С. Тенденція розвитку методів рендерингу web-сайтів для пошукової оптимізації сайтів (SEO) // Радіоелектроніка та молодь у XXI столітті: матеріали 28-го Міжнар. молодіж. форуму. – Т. 6. – Харків : ХНУРЕ, 2024. – С. 918–919. DOI: https://doi.org/10.30837/IYF.IIS.2024.918.