Івано-Франківський національний технічний університет нафти і газу

Інститут інформаційних технологій
Кафедра інформаційно-вимірювальних технологій

Козарук Микола Русланович
(прізвище, ім'я, по батькові)

УДК 004.77
(індекс)

МАГІСТЕРСЬКА РОБОТА

Дослідження швидкості передачі даних в мережі Інтернет з урахуванням мережевих затримок
(назва роботи)

Метрологія та вимірювальна техніка
(назва освітньої програми)

175 – «Інформаційно-вимірювальні технології»
(шифр і назва спеціальності)

М.Р. Козарук
(підпис, ініціали та прізвище здобувача освітнього ступеня)

Науковий керівник Чуйко Мирослава Михайлівна, к.т.н, доц.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Допущено до захисту

Завідувач кафедри

О. Є. Середюк
(посада) (підпис) (дата) (ініціали та прізвище)

Рецензент

_____
(посада) (підпис) (дата) (ініціали та прізвище)

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

Івано-Франківськ, 2024 р.

Івано-Франківський національний технічний університет нафти і газу
(повне найменування закладу вищої освіти)

Інститут *інформаційних технологій*

Кафедра *інформаційно-вимірювальних технологій*

Освітній рівень *другий (магістерський)*

Спеціальність *175- Інформаційно-вимірювальні технології*
(шифр і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри ІВТ
О.Є. Середюк
«____» _____2024 року

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТОВІ

Козаруку Миколі Руслановичу
(прізвище, ім'я, по батькові)

1. Тема роботи *Дослідження швидкості передачі даних в мережі Інтернет з урахуванням мережевих затримок.*

керівник роботи *Чуйко Мирослава Михайлівна, к.т.н, доц.*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від "03" 12.2024 року № 787/7

2. Строк подання студентом роботи 20.12.2024р.

3. Вихідні дані до роботи: *максимальна швидкість передачі даних, від 0 до 15000 Мбіт/с, мінімальний час затримки 0.3 мс*,

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Аналітичний огляд теоретичних основ швидкості передачі даних та мережевих затримок

2. Методологія та розробка програмного забезпечення вимірювання мережевих затримок

3. Метрологічний аналіз програмного забезпечення

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

1. Схема роботи програмного забезпечення
2. Схема роботи TCP/IP
3. Структурна Схема вимірювання Jitter
4. Схема вимірювання Ping
5. Схема вимірювання скорості завантаження/передачі даних
6. Схема роботи Websocket в програмному забезпечені
7. Демонстрація практичної реалізації, зовнішній вигляд сайту
8. Метрологічні аспекти програмного забезпечення

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
нормоконтроль	Лютак З.П., професор каф. ІВТ		

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської роботи	Термін виконання етапів роботи	Примітка
1.	*Аналітичний огляд теоретичних основ швидкості передачі даних та мережевих затримок*	*06.12.2024*	
2.	*Методологія та розробка програмного забезпечення вимірювання мережевих затримок*	*11.12.2024*	
3.	*Метрологічний аналіз програмного забезпечення*	*16.12.2024*	
4.	*Оформлення пояснювальної записки магістерської роботи*	*18.12.2024*	
5	*Оформлення ілюстративного матеріалу магістерської роботи*	*20.12.2024*	

Студент _____ *Козарук М.Р.*
(підпис) (прізвище та ініціали)

Керівник роботи _____ *Чуйко М.М.*
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Магістерська робота на тему «Дослідження швидкості передачі даних в мережі Інтернет з урахуванням мережевих затримок»: 66 с., 15 рис., 20 джерел, 8 аркушів графічного матеріалу.

Об’єкт дослідження – процеси передачі даних в мережі Інтернет з урахуванням мережевих затримок.

Предмет дослідження – вплив мережевих затримок на швидкість передачі даних, методи їх оцінки та мінімізації.

Мета роботи – визначити фактори, що впливають на швидкість передачі даних з урахуванням затримок, та розробити програмний комплекс для вимірювання ключових параметрів (швидкість завантаження/вивантаження, пінг, джиттер), а також сформувати рекомендації з оптимізації мережевої продуктивності.

Наукова новизна полягає у комплексному підході до вимірювання мережевих параметрів із використанням Node.js, WebSocket та збереженням результатів у базі даних для подальшого аналізу.

Практичне значення: розроблене програмне забезпечення може застосовуватись провайдерами, розробниками мережевого обладнання та кінцевими користувачами для оцінки якості з’єднання та пошуку «вузьких місць» мережі. Надано рекомендації з підвищення точності вимірювань відповідно до міжнародних стандартів (RFC, ITU-T).

КЛЮЧОВІ СЛОВА: ІНТЕРНЕТ, МЕРЕЖЕВІ ЗАТРИМКИ, ШВИДКІСТЬ ПЕРЕДАЧІ ДАНИХ, ПІНГ, ДЖИТТЕР, ПРОПУСКНА ЗДАТНІСТЬ, NODE.JS, WEBSOCKET, МЕТРОЛОГІЧНИЙ АНАЛІЗ.

ABSTRACT

Master’s thesis on the topic "Research of data transmission speed in the Internet network taking into account network delays": 66 pages, 15 pictures, 20 sources, 8 sheets of graphical material.

Object of the research – the processes of data transmission in the Internet network, taking into account network delays.

Subject of the research – the influence of network delays on data transmission speed, methods of their assessment and minimization.

Purpose of the work – to determine the factors affecting data transmission speed with regard to delays, and to develop a software suite for measuring key parameters (download/upload speed, ping, jitter), as well as to formulate recommendations for optimizing network performance.

Scientific novelty lies in the comprehensive approach to measuring network parameters using Node.js, WebSocket, and storing the results in a database for further analysis.

Practical significance: the developed software can be applied by providers, network equipment developers, and end users to evaluate connection quality and identify “bottlenecks” in the network. Recommendations are provided for increasing measurement accuracy in accordance with international standards (RFC, ITU-T).

KEYWORDS: INTERNET, NETWORK DELAYS, DATA TRANSMISSION SPEED, PING, JITTER, BANDWIDTH, NODE.JS, WEBSOCKET, METROLOGICAL ANALYSIS.

ЗМІСТ

ВСТУП

Актуальність теми. У сучасному світі інтернет став невід'ємною частиною повсякденного життя мільярдів людей по всьому світу [1]. Він об'єднує людей, організації та держави, забезпечуючи швидкий, зручний доступ до інформації, послуг і комунікації. З розвитком цифрових технологій, зростанням обсягів передаваних даних, швидкість передачі інформації в мережі інтернет - набуває критичного значення [2].

З кожним днем в геометричній прогресії збільшується кількість користувачів, які потребують високошвидкісного доступу до мережі для виконання різноманітних завдань: від перегляду відео високої чіткості та участі в онлайн-іграх до проведення відеоконференцій і дистанційної роботи, й взагалі ще багато де, що зображено на рис. 1. Водночас, розвиток Інтернету речей (IoT), впровадження технологій 5G та перехід до хмарних сервісів вимагають ще більшої пропускної здатності мереж та мінімізації затримок [3].

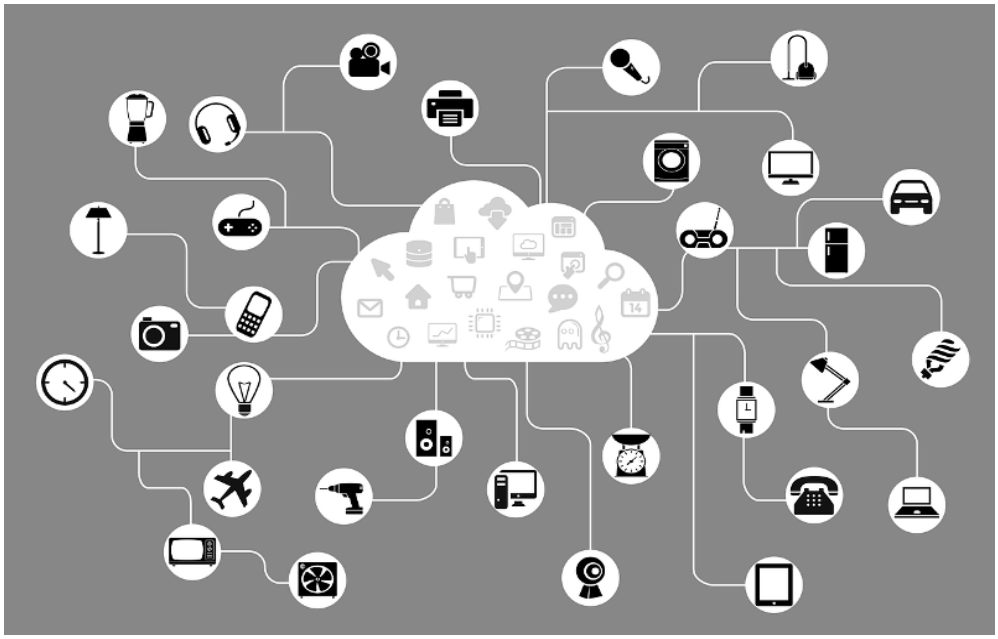


Рисунок 1 – Вплив IoT (інтернет речей) на наше життя

Мережеві затримки, або латентність, є одним із ключових факторів, що впливають на якість передачі даних та загальний користувацький досвід [4]. Вони можуть виникати з різних причин: від фізичних обмежень обладнання, віддаленості серверів до перевантаження мережі та неефективних маршрутів передачі даних. Розуміння і аналіз цих затримок є необхідними для оптимізації роботи мережі, забезпечення максимально можливої швидкості передачі інформації.

Дослідження швидкості передачі даних з урахуванням мережевих затримок має велике практичне значення [5]. Воно дозволяє виявити "вузькі місця" в мережі, оптимізувати маршрутизацію, покращити протоколи передачі та, врешті-решт, підвищити ефективність використання мережевих ресурсів. Це особливо важливо для провайдерів інтернет-послуг, розробників мережевого обладнання, програмного забезпечення, а особливо які прагнуть забезпечити своїм клієнтам найкращий сервіс.

Крім того, сучасні тенденції, такі як віддалена робота і навчання, які стали особливо актуальними внаслідок глобальних подій останніх років, ще більше підкреслюють важливість стабільного та швидкого інтернет-з'єднання [6]. Затримки в передачі даних можуть негативно вплинути на продуктивність праці, якість освіти та навіть на соціальну взаємодію між людьми.

Не менш важливим є питання кібербезпеки [7]. Швидкість, затримки в мережі - можуть впливати на ефективність систем захисту від кібератак. Наприклад, своєчасне виявлення, реагування на загрози часто залежать від швидкості передачі даних між різними компонентами системи безпеки.

З огляду на все вищезазначене, стає очевидним, що дослідження в цій галузі не лише мають теоретичне значення, але й можуть мати вагомий практичний внесок у підвищення якості життя суспільства. Оптимізація швидкості передачі даних з урахуванням мережевих затримок сприятиме ефективнішому використанню існуючої інфраструктури, зменшенню витрат на модернізацію мереж, покращенню загального користувацького досвіду [8].

Таким чином, обрана тема є не лише актуальною, але й необхідною для подальшого розвитку інформаційного суспільства [9]. Вона відкриває можливості для інновацій, підвищення конкурентоспроможності підприємств та покращення якості надання послуг у різних галузях економіки. Дослідження в цьому напрямку сприятимуть створенню більш надійних та швидких мереж, що відповідають потребам сучасного світу.

Отже, вивчення швидкості передачі даних в мережі інтернет з урахуванням мережевих затримок є своєчасним та відповідає актуальним викликам, з якими стикається інформаційне суспільство сьогодні [10]. Це сприятиме не лише науковому прогресу, але й практичному впровадженню новітніх технологій для покращення життя людей.

Мета дослідження. Метою даного дослідження є виявлення та аналіз основних факторів, що впливають на швидкість передачі даних в мережі інтернет з урахуванням мережевих затримок, а також розробка практичних рекомендацій для оптимізації цих процесів. Це дозволить підвищити ефективність використання мережевих ресурсів і покращити якість послуг, що надаються користувачам.

Завдання дослідження. Для досягнення поставленої мети необхідно виконати наступні завдання:

1. **Дослідити основи передачі даних в мережі Інтернет**:
    - Ознайомитися з принципами роботи мережі Інтернет.
    - Вивчити основні протоколи та технології, що використовуються для передачі даних.
2. **Аналіз мережевих затримок**:
    - Визначити, що таке мережеві затримки і як вони впливають на передачу даних.
    - Розглянути причини виникнення затримок, включаючи фізичні та програмні фактори.

3. **Виявити фактори, що впливають на швидкість передачі даних**:
    - Проаналізувати, як пропускна здатність, відстань між вузлами мережі та інші параметри впливають на швидкість.
    - Розглянути вплив зовнішніх факторів, таких як завантаженість мережі та якість з'єднання.
4. **Розробити методику вимірювання швидкості передачі даних з урахуванням затримок**:
    - Створити план дослідження для збору необхідних даних.
    - Вибрати інструменти та методи для проведення вимірювань.
5. **Провести експериментальні дослідження**:
    - Здійснити серію вимірювань швидкості передачі даних у різних умовах.
    - Зібрати та систематизувати отримані дані для подальшого аналізу.
6. **Аналіз та інтерпретація результатів**:
    - Обробити зібрані дані та виявити закономірності.
    - Оцінити вплив різних факторів на швидкість передачі даних.
7. **Розробка рекомендацій щодо оптимізації швидкості передачі даних**:
    - На основі отриманих результатів сформулювати практичні поради.
    - Запропонувати способи зменшення мережевих затримок та підвищення швидкості передачі.
8. **Підготовка висновків та пропозицій для подальших досліджень**:
    - Підсумувати основні знахідки дослідження.
    - Визначити напрямки для майбутніх досліджень у цій галузі.

Очікувані результати. В результаті виконання цих завдань очікується отримати:

- Комплексне розуміння впливу мережевих затримок на швидкість передачі даних в Інтернеті [11].
- Практичні рекомендації, які можуть бути застосовані для покращення роботи мережі.

- Підвищення ефективності використання мережевих ресурсів та покращення якості користувацького досвіду [12].
- Базу для подальших досліджень і розвитку в області оптимізації мережевих технологій.

Виконання цих завдань є критично важливим з огляду на стрімкий розвиток інформаційних технологій та зростання потреби у швидкому і надійному доступі до мережі інтернет. Отримані результати сприятимуть підвищенню якості надання інтернет-послуг, що є актуальним для провайдерів, кінцевих користувачів.

Покращення швидкості передачі даних та зменшення мережевих затримок позитивно вплине на різні сфери життя [13]:

- **Економіка**: Підвищення ефективності бізнес-процесів, зменшення витрат на інфраструктуру.
- **Освіта та наука**: Забезпечення якісного доступу до онлайн-ресурсів та дистанційного навчання.
- **Медицина**: Підтримка телемедицини та швидкого обміну медичними даними.
- **Соціальна сфера**: Поліпшення можливостей для спілкування та взаємодії між людьми.

Таким чином, реалізація мети та завдань даного дослідження має наукове, практичне значення. Це сприятиме розвитку інформаційного суспільства та підвищенню якості життя людей шляхом забезпечення швидкого, надійного доступу до мережі інтернет.

Об'єктом даного дослідження є процеси передачі даних в мережі інтернет, що супроводжуються мережевими затримками та впливають на швидкість, якість комунікації [14]. Це включає: технічну інфраструктуру мережі, протоколи зв'язку, маршрутизацію, інші компоненти, які забезпечують обмін інформацією між користувачами та серверами по всьому світу.

Предметом дослідження є вплив мережевих затримок на швидкість передачі даних в інтернеті, способи їх мінімізації [15]. Це передбачає аналіз факторів, що спричиняють затримки, вивчення їхнього впливу на користувацький досвід і розробку методів оптимізації передачі даних. Особлива увага приділяється ідентифікації "вузьких місць" у мережі, впровадженню практичних рекомендацій для підвищення ефективності мережевих з'єднань.

Наукова новизна дослідження полягає в розробці комплексного підходу до оцінки швидкості передачі даних в мережі інтернет з урахуванням мережевих затримок, що дозволяє одночасно виявляти, аналізувати і мінімізувати їхній вплив на якість мережевих з'єднань [16].

Унікальність роботи полягає у поєднанні теоретичного аналізу з практичними експериментами, реалізованими за допомогою створеної програмної інфраструктури.

Основні аспекти новизни:

A. **Розробка інноваційної програмної платформи для тестування мережі**:
   - ❖ Створений сервер, який реалізує багатофункціональне середовище для вимірювання ключових показників, таких як швидкість завантаження, вивантаження, пінг та джиттер.
   - ❖ Використання WebSocket-з'єднання для точного визначення пінгу, аналізу коливань затримки (джиттеру), що дозволяє в реальному часі виявляти мережеві аномалії.

B. **Інтеграція новітніх підходів до збору та обробки даних**:
   - ❖ База даних SQLite використовується для збереження результатів тестування, що забезпечує зручний доступ до історії вимірювання даних для подальшого аналізу.
   - ❖ Реалізовані механізми обробки великих обсягів даних без компресії, що гарантує достовірність отриманих результатів при зміні умов виконання.

C. **Комбінація реальних вимірювань та теоретичних розрахунків**:

- ❖ У роботі поєднані сучасні методи оцінки продуктивності мережі з аналізом факторів, які впливають на затримки, таких як фізична відстань, пропускна здатність мережі та завантаженість серверів.
- ❖ Використані підходи забезпечують інтегральну оцінку продуктивності інтернет-з'єднань у різних умовах експлуатації.

D. **Практична спрямованість дослідження**:

- ❖ Розроблені рекомендації щодо оптимізації роботи мережі можуть бути застосовані в реальних сценаріях для покращення якості послуг провайдерів [17].
- ❖ Програмне забезпечення, створене в рамках дослідження, може бути використане не лише для наукових цілей, а й у промислових застосуваннях.

Таким чином, **дана робота робить внесок в розвиток методології аналізу** та покращення швидкості передачі даних в інтернеті. Отримані результати можуть стати **основою для подальших досліджень у сфері оптимізації мережевих технологій**, зокрема в умовах зростання обсягів даних та збільшення вимог до якості зв’язку.

Практичне значення роботи полягає у створенні інструментів для вимірювання та аналізу ключових параметрів мережі, таких як швидкість завантаження і вивантаження даних, пінг та джиттер. Результати дослідження можуть бути використані інтернет-провайдерами для виявлення та усунення "вузьких місць" у мережі, покращення якості надання послуг і оптимізації роботи інфраструктури. Розроблена система тестування є універсальним рішенням, що може застосовуватись для моніторингу продуктивності мережі у реальних умовах.

1 АНАЛІТИЧНИЙ ОГЛЯД ТЕОРЕТИЧНИХ ОСНОВ ШВИДКОСТІ ПЕРЕДАЧІ ДАНИХ ТА МЕРЕЖЕВИХ ЗАТРИМОК

1.1 Поняття швидкості передачі даних в мережі інтернет

Швидкість передачі даних в мережі інтернет є одним із ключових показників, який визначає ефективність роботи мережевих з’єднань.

Вона відображає кількість інформації, що може бути передана за одиницю часу, зазвичай вимірюється в біт/с (bps) або його кратних одиницях, таких як кілобіт/с (kbps), мегабіт/с (Mbps) чи гігабіт/с (Gbps).

Швидкість передачі даних зазвичай поділяється на два основні типи:

- **Швидкість завантаження (download speed)** – це обсяг даних, який може бути отриманий з мережі за певний час.
- **Швидкість вивантаження (upload speed)** – це обсяг даних, який може бути переданий з пристрою користувача в мережу.

Ці показники мають вирішальне значення для різних сценаріїв використання, наприклад передача файлів в локальній мережі, з’єднання повинне бути як на рис 1.1. Ще один приклад, висока швидкість завантаження важлива для перегляду потокового відео чи завантаження великих файлів, тоді як швидкість вивантаження критична для відеоконференцій, онлайн-ігор або передачі даних на хмарні сервери.

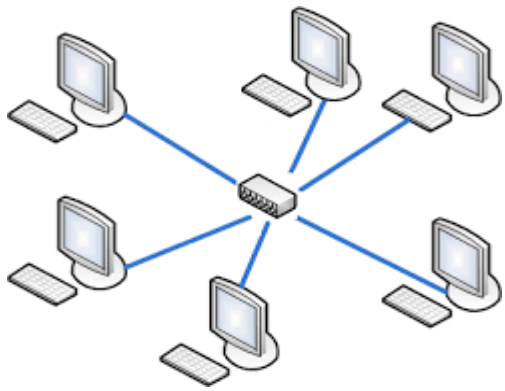


Рисунок 1.1 – Приклад з’єднання через локальну мережу

Основні фактори, що впливають на швидкість передачі даних:

1. **Пропускна здатність мережі.** Цей параметр визначає максимальний обсяг інформації, який може пройти через канал зв’язку. Він залежить від характеристик мережевого обладнання (маршрутизаторів, кабелів, модемів тощо), технології доступу (оптоволокно, DSL, мобільний зв’язок 4G/5G тощо).
2. **Фізичні обмеження середовища передачі.** Наприклад, у провідних мережах сигнал може втрачати інтенсивність із збільшенням відстані, що впливає на швидкість.
3. **Мережеві затримки.** Це час, необхідний для передачі даних між точками мережі, який залежить від кількості проміжних вузлів, якості маршрутизації та завантаженості мережі. Затримки можуть призводити до сповільнення навіть у мережах із високою пропускною здатністю.
4. **Протоколи зв’язку.** Протоколи передачі даних, такі як TCP (Transmission Control Protocol) або UDP (User Datagram Protocol), регулюють порядок і спосіб доставки даних. Наприклад, TCP забезпечує надійність передачі, але може знижувати швидкість через механізми контролю.
5. **Використання ресурсів користувачем.** Якщо кілька пристроїв підключені до однієї мережі, швидкість передачі даних між ними розподіляється, що може вплинути на продуктивність.

Практичні аспекти швидкості передачі даних:

На практиці швидкість передачі даних часто відрізняється від заявленої провайдером, оскільки реальні умови використання включають такі чинники, як завантаженість мережі, якість з'єднання та особливості роботи серверів.

У моєму програмному коді було реалізовано модулі для тестування швидкості завантаження і вивантаження даних, що дозволяє оцінювати реальні показники мережі та виявляти "вузькі місця". Наприклад, у розділі «/download» сервер передає великі обсяги даних для симуляції завантаження, а у «/upload» приймає дані, вимірюючи їхню швидкість передачі.

Ці тести дозволяють імітувати реальні умови роботи мережі та оцінювати швидкість передачі даних у різних сценаріях, таких як робота під високим навантаженням чи при зміні затримок. Важливість таких вимірювань полягає у виявленні факторів, які обмежують продуктивність мережі, та розробці рекомендацій для їх усунення.

1.2 Мережеві затримки: визначення та класифікація

Мережеві затримки (або латентність) є одним із ключових факторів, що впливають на якість роботи мережі інтернет. Це час, який потрібен для передачі даних від джерела до пункту призначення та назад, вимірюваний у мілісекундах (мс).

Хоча затримки не впливають безпосередньо на пропускну здатність каналу, вони значно впливають на швидкість реакції мережі та загальний користувацький досвід.

Визначення мережевих затримок. Затримка як раніше було сказано — це сумарний час, витрачений на передачу пакета даних через різні компоненти мережі. Її величина залежить від фізичних характеристик каналу зв’язку, завантаженості мережі, типу маршрутизаторів, протоколів передачі даних і топології мережі.

Типовий шлях даних через мережу включає кілька етапів:

1. Відправлення даних з пристрою користувача
2. Обробка маршрутизаторами та іншими мережевими пристроями.
3. Передача через міжмережеві з'єднання.
4. Отримання даних кінцевим сервером та відповідь у зворотному напрямку.

Класифікація мережевих затримок. Мережеві затримки поділяються на кілька основних типів залежно від їх природи та причин:

1. **Затримка передачі (Transmission Delay):**
   - Це час, необхідний для передачі пакета даних через канал зв’язку.

- Хоча не буде використовуватись в моєму коді, але визначається формулою T_trans=L/R, де L — розмір пакета (у бітах), R — пропускна здатність каналу (у біт/с).
- Наприклад, для передачі 1 Мб даних через канал із пропускною здатністю 100 Мбіт/с затримка становитиме 0,08 с.

2. **Затримка обробки (Processing Delay):**
   - Час, витрачений на обробку пакета маршрутизатором або сервером, включаючи перевірку заголовка, визначення маршруту та виконання необхідних операцій.
   - Ця затримка залежить від продуктивності обладнання та програмного забезпечення.
3. **Затримка поширення (Propagation Delay):**
   - Час, який потрібен сигналу для проходження відстані між двома вузлами в мережі.
   - Також не буде використовуватись, але визначається як: T_prop = d / v, де d — відстань між вузлами, v — швидкість поширення сигналу (приблизно дорівнює швидкості світла в середовищі).
   - У волоконно-оптичних кабелях ця затримка може становити близько 5 мс на 1000 км.
4. **Черга очікування (Queuing Delay):**
   - Виникає, коли пакети очікують у черзі на обробку маршрутизатором або перед передачею через перевантажений канал.
   - Залежить від завантаженості мережі та алгоритмів управління чергами.
5. **Джиттер (Jitter):**
   - Коливання затримки між послідовними пакетами даних.
   - Особливо критичний для мультимедійних додатків, оскільки викликає переривання або зниження якості звуку та відео.

Практичні аспекти мережевих затримок. Мережеві затримки мають вирішальне значення для оцінки продуктивності мережі. У розробленому програмному коді реалізовані механізми для вимірювання затримок за допомогою WebSocket-з'єднання.

Зокрема:

- **Пінг:** вимірюється час між відправкою запиту та отриманням відповіді від сервера.
- **Джиттер:** обчислюється як середнє відхилення між часом передачі пакетів.

Ці вимірювання дозволяють отримати точну картину стану мережі. Наприклад, якщо пінг стабільний, але спостерігається високий джиттер, це може свідчити про перевантаженість маршрутизаторів. Зібрані дані зберігаються у базі даних SQLite, що в майбутньому дозволяє аналізувати історичні зміни затримок і виявляти залежності.

Значення для користувачів та провайдерів. Для кінцевих користувачів затримки можуть проявлятися у вигляді уповільнення завантаження веб-сторінок, переривань у потоковому відео чи "лагів" у онлайн-іграх. Провайдери, у свою чергу, можуть використовувати дані про затримки для оптимізації своєї інфраструктури, переналаштування маршрутів чи модернізації обладнання.

Розуміння природи та типів мережевих затримок є критичним для підвищення якості послуг, що надаються через інтернет – сприяє зменшенню негативного впливу латентності на роботу мереж.

1.3 Фактори, що впливають на швидкість передачі даних та мережеві затримки

Швидкість передачі даних та мережеві затримки залежать від великої кількості факторів, кожен з яких має різний рівень впливу залежно від типу мережі, використовуваних технологій та умов експлуатації.

Аналіз цих факторів дозволяє краще зрозуміти роботу мережі інтернет та виявити можливості для її оптимізації.

Одним із основних факторів є **пропускна здатність каналу** зв’язку. Вона визначає максимальну кількість інформації, яка може бути передана за одиницю часу. Пропускна здатність **залежить від типу мережевої технології** (оптоволоконні з'єднання, DSL, мобільний зв'язок 4G/5G тощо) та стану фізичних середовищ передачі. Наприклад, оптоволоконні мережі мають значно вищу пропускну здатність порівняно з традиційними мідними кабелями, що робить їх більш придатними для передачі великих обсягів даних.

Відстань між вузлами мережі є ще одним важливим фактором, який впливає на затримки. Дані мають пройти певний шлях від джерела до отримувача та чим більшою є ця відстань – тим більший час потрібен для передачі сигналу. У сучасних умовах, коли користувачі можуть обмінюватися інформацією з іншого континенту за лічені секунди, кожна милісекунда має значення. У випадках, коли сервера знаходяться далеко, використовуються методи кешування та оптимізації маршрутів, але ці підходи не завжди дозволяють повністю усунути затримки.

Стан обладнання також має суттєвий вплив на швидкість передачі даних. Маршрутизатори, комутатори та інші мережеві пристрої повинні обробляти великі обсяги інформації, їхня пропускна здатність часто стає критичною в умовах високих навантажень. Якщо обладнання застаріле або не відповідає сучасним стандартам – це призводить до збільшення затримок і зниження швидкості передачі даних.

Протоколи передачі даних, такі як **TCP або UDP**, визначають правила комунікації між пристроями в мережі. TCP забезпечує надійність передачі, але для цього використовує підтвердження доставки кожного пакета, що може збільшувати затримки. UDP, навпаки, не гарантує доставку, але дозволяє досягти мінімальної затримки, що важливо для додатків реального часу, таких як онлайн-

ігри чи відеоконференції. Вибір протоколу впливає на продуктивність мережі залежно від її цілей і завдань.

Навколишнє середовище також може спричиняти затримки. У бездротових мережах, таких як Wi-Fi – перешкоди від інших пристроїв, товсті стіни або великі відстані між маршрутизатором і користувачем можуть знижувати якість сигналу. Це часто призводить до повторної передачі даних, що збільшує загальний час передачі.

Перевантаження мережі є ще однією причиною підвищених затримок. У моменти пікового навантаження, коли багато користувачів одночасно використовують мережу, маршрутизатори можуть не встигати обробляти всі запити. Це створює черги даних, що очікують на передачу – відповідно, збільшує час доставки. Для вирішення цієї проблеми провайдери часто використовують механізми управління трафіком, такі як пріоритизація пакетів.

У **моєму дослідженні реалізовано методи тестування, які дозволяють оцінити вплив цих факторів** на реальну швидкість передачі даних і затримки. Наприклад, у серверному коді передбачено окремі ендпоінти для завантаження та вивантаження даних, які дозволяють вимірювати продуктивність мережі у різних умовах. Крім того, за допомогою WebSocket-з’єднань забезпечується точний аналіз затримок, включаючи джиттер, який є важливим показником для мультимедійних додатків.

Результати дослідження показали, що затримки часто виникають через комбінацію перелічених факторів, їхній вплив може значно варіюватися залежно від конкретного сценарію. Таким чином, розуміння цих факторів дозволяє не лише покращити роботу існуючих мереж, але й допомагає провайдерам та користувачам вибирати найбільш ефективні способи передачі даних.

1.4 Протоколи передачі даних та їх вплив на мережеві затримки

Протоколи передачі даних є ключовими компонентами мережі інтернет, які визначають правила обміну інформацією між пристроями. Їх правильна робота впливає на ефективність передачі даних та рівень мережевих затримок. Протоколи не тільки забезпечують надійність та безпеку, але й впливають на продуктивність з'єднань, особливо в умовах високих навантажень.

Основою роботи більшості мереж є модель OSI (Open Systems Interconnection), яка описує взаємодію між пристроями на різних рівнях. Протоколи передачі даних, такі як TCP (Transmission Control Protocol) та UDP (User Datagram Protocol), працюють на транспортному рівні цієї моделі та відіграють ключову роль у визначенні характеристик затримок.

TCP: Надійність із ціною затримок. TCP є найпоширенішим протоколом передачі даних, який гарантує доставку інформації. Він забезпечує надійність передачі через механізм підтвердження отримання (ACK) та повторну відправку втрачених пакетів. Хоча це забезпечує стабільність передачі, механізми підтвердження додають затримки, особливо на високозавантажених мережах або при значній відстані між вузлами.

Наприклад, TCP використовує алгоритм управління потоком, який уповільнює передачу даних у разі перевантаження мережі. Це допомагає уникнути втрати пакетів, але збільшує загальний час передачі. Крім того, кожна втрата пакета вимагає повторної передачі, що додатково впливає на латентність.

UDP: Швидкість без гарантій. UDP є альтернативним протоколом, що працює без механізму підтвердження доставки. Він дозволяє передавати дані з мінімальними затримками, оскільки не потребує додаткового оброблення підтверджень. Це робить його незамінним для реального часу, наприклад, в онлайн-іграх, відеоконференціях та потоковому передаванні мультимедіа.

Проте відсутність гарантій доставки означає, що втрата пакетів може призвести до спотворення звуку чи відео. Для компенсації цього розробники

додатків створюють власні механізми корекції, що може частково відновлювати втрачені дані.

HTTP/2 та HTTP/3: Еволюція веб-протоколів. HTTP/2, як наступник HTTP/1.1, впровадив мультиплексування, яке дозволяє передавати кілька потоків даних через одне з'єднання. Це значно зменшило затримки при завантаженні веб-сторінок. HTTP/3, який працює на основі протоколу QUIC (Quick UDP Internet Connections), ще більше скоротив затримки завдяки використанню UDP та уникненню проблем з повторним встановленням з'єднання при зміні мережі.

QUIC, на відміну від TCP, інтегрує в себе шифрування та підтвердження на рівні транспортного протоколу, що дозволяє уникнути додаткових затримок через окремі механізми безпеки, як у випадку з TLS у TCP.

Вплив протоколів на мережеві затримки. Протоколи передачі даних можуть по-різному впливати на затримки залежно від умов використання.
Приклад впливу:

- У стабільних мережах із високою пропускною здатністю TCP забезпечує надійну передачу даних, хоча й додає невеликі затримки.
- У мережах з високою завантаженістю чи великою кількістю втрат UDP надає перевагу швидкості за рахунок можливої втрати пакетів.
- HTTP/3 та QUIC мінімізують затримки завдяки більш сучасним механізмам роботи з потоками даних.

Практичне застосування в дослідженні:

У межах дослідження для вимірювання мережевих затримок використовувались обидва основні транспортні протоколи (TCP та UDP). Наприклад, у реалізованому програмному забезпеченні WebSocket-з'єднання використовує TCP для гарантії точності вимірювання пінгу та джиттера. Це дозволило забезпечити високу точність тестів, необхідну для аналізу продуктивності мережі.

Для оцінки швидкості завантаження та вивантаження даних використовувались ендпоінти, що працюють за HTTP/1.1, але з можливістю

адаптації під HTTP/2. У перспективі інтеграція QUIC дозволить ще більше зменшити затримки, особливо в умовах нестабільного з'єднання.

Отже протоколи передачі даних відіграють вирішальну роль у визначенні швидкості роботи мережі та рівня затримок. Їх вибір залежить від задач, які стоять перед користувачем чи організацією. Надійність TCP, швидкість UDP та новітні можливості QUIC демонструють, що правильна конфігурація протоколів може суттєво покращити продуктивність мережі, навіть у складних умовах.

1.5 Огляд сучасних методів та інструментів вимірювання мережевих параметрів

Вимірювання мережевих параметрів, таких як швидкість передачі даних, пінг, джиттер і втрата пакетів – є ключовим етапом для оцінки продуктивності мережі Інтернет. Сучасні методи вимірювання ґрунтуються на використанні програмних та апаратних інструментів, що дозволяють проводити як локальні, так й віддалені тести. Ці інструменти використовуються провайдерами, розробниками програмного забезпечення, кінцевими користувачами для аналізу стану мережі та пошуку проблем.

Методи вимірювання. Одним із основних методів оцінки швидкості передачі даних є тестування пропускної здатності мережі. Це передбачає передавання великих обсягів даних між клієнтом і сервером протягом певного часу з метою визначення середньої швидкості завантаження (download) та вивантаження (upload). Такий підхід широко використовується у популярних сервісах, таких як Speedtest та Fast, що дають змогу швидко оцінити продуктивність з'єднання [18-19].

Іншим методом є використання протоколів ICMP (Internet Control Message Protocol) для тестування пінгу. ICMP дозволяє відправляти контрольні пакети даних і вимірювати час, необхідний для отримання відповіді від сервера. Цей

метод є ефективним для визначення затримок, але має обмеження у разі, якщо сервери або маршрутизатори блокують ICMP-запити.

Для вимірювання джиттера, який є критичним для додатків реального часу – використовується метод аналізу часу між послідовними пакетами. Джиттер обчислюється як середнє відхилення між часом доставки пакетів, що дозволяє виявляти нестабільність мережі.

Сучасні інструменти. Одним із найпоширеніших інструментів для вимірювання мережевих параметрів є Speedtest by Ookla**,** його вигляд можна побачити на рис 1.2. Цей сервіс дозволяє визначити швидкість завантаження, вивантаження та й пінг, використовуючи розподілену мережу серверів. Ookla використовує протокол TCP для передачі даних, що забезпечує надійність вимірювань [18].

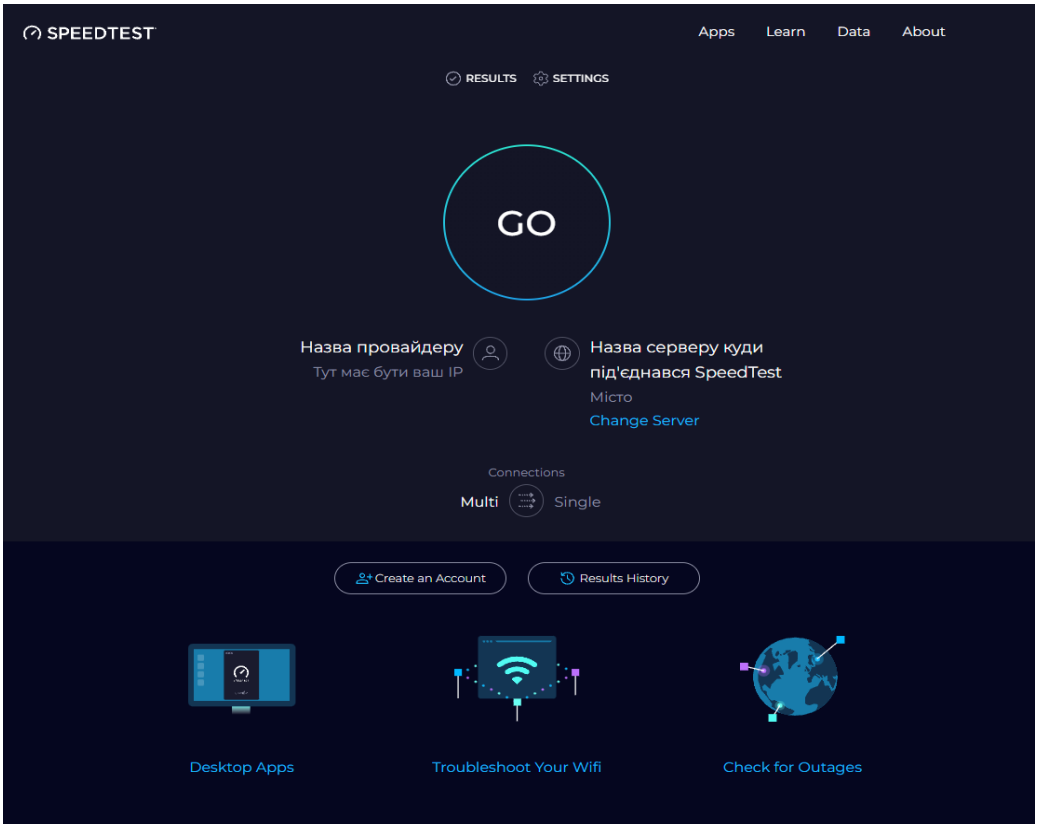


Рисунок 1.2 – Зовнішній вигляд Speedtest by Ookla

Pingplotter є потужним інструментом для аналізу затримок і втрати пакетів. Він використовує комбіновані запити ICMP, TCP та UDP, що дає змогу отримати повну картину стану мережі на різних етапах маршрутизації.

Для комплексного тестування мереж часто застосовується **iPerf**. Це програмний інструмент з відкритим кодом, який дозволяє оцінювати пропускну здатність мережі з використанням різних протоколів (TCP, UDP). iPerf може працювати у режимі клієнта та сервера, що дозволяє гнучко налаштовувати умови тестування [12].

Власний підхід до вимірювань. У рамках мого дослідження було розроблено серверне програмне забезпечення, яке дозволяє виконувати вимірювання основних мережевих параметрів у реальному часі. Для цього використовуються окремі ендпоінти:

- /download для тестування швидкості завантаження даних, де клієнт отримує великий обсяг інформації для оцінки пропускної здатності.
- /upload для тестування швидкості вивантаження, де дані передаються на сервер.
- WebSocket – з'єднання для аналізу пінгу та джиттера, яке забезпечує високу точність завдяки вимірюванню часу між відправленням і отриманням пакетів.

Ключовою перевагою розробленого інструменту є можливість адаптації під різні сценарії тестування, включаючи вимірювання під високим навантаженням чи в умовах погіршення якості зв'язку.

1.6 Огляд міжнародних стандартів та рекомендацій щодо оцінки мережевої продуктивності

При вимірюванні швидкості передачі даних та оцінці затримок в Інтернеті важливо розуміти, що результати не існують у вакуумі. Зазвичай провайдери, дослідники та розробники програмних рішень орієнтуються на загальноприйняті стандарти та рекомендації, які допомагають уніфікувати процедури тестування та робити висновки більш об'єктивними.

Одним із найвідоміших джерел рекомендацій є так звані **RFC-документи (Request for Comments)** від організації IETF (Internet Engineering Task Force). Ці документи розробляються експертами з усього світу та часто слугують базисом для створення інструментів, методик вимірювання мережевих показників. Наприклад, існують RFC, які описують методи: оцінки затримок, втрат пакетів, пропускної здатності, а також протоколи які призначені для тестування продуктивності окремих компонентів мережі.

Ось список RFC документів які я аналізував та настроював під них програмний код:

- **RFC 7679**: *"Framework for Performance Metric Development"* Цей документ описує загальний підхід до розробки метрик продуктивності мережі, включно з вимірюванням затримок та інших параметрів. Він допомагає уніфікувати підхід до оцінки роботи мережевих систем [5].
- **RFC 2680**: *"A One-way Delay Metric for IPPM"* Цей RFC стосується визначення та вимірювання односторонньої затримки (One-Way Delay, OWD) для Інтернет-протоколів. Він описує методологію та вимоги до точності, а також формат представлення результатів вимірювань [4].
- **RFC 5357**: *"A Two-Way Active Measurement Protocol (TWAMP)"* TWAMP — це протокол, що дозволяє проводити активні вимірювання двосторонньої затримки, втрат пакетів та інших показників. Документ описує, як налаштувати та виконувати тести, а також як інтерпретувати отримані результати [11].
- **RFC 5481**: *"Packet Delay Variation Applicability Statement"* У цьому документі описано показник варіації затримки (джиттер) і його застосування. RFC 5481 розкриває різні підходи до оцінки нестабільності часу доставки пакетів і пропонує методи обчислення джиттера [15].
- **RFC 6071**: *"SEcure Neighbor Discovery (SEND) problem statement"* (хоч цей документ більше про безпеку в мережах, він опосередковано може

впливати на затримки при встановленні безпечних з’єднань, бо описує проблеми, які можуть викликати додаткові затримки.) [20]

Крім того, міжнародні організації, як-от ITU (Міжнародний союз електрозв’язку) та ISO/IEC, пропонують свої стандарти, керівні документи. Вони включають рекомендації з оцінки якості обслуговування (Quality of Service, QoS), часу відгуку сервісів, а також стабільності підключення. Такі документи зазвичай розроблені для того щоб: різні оператори зв’язку або виробники обладнання могли однаково інтерпретувати результати тестів і покращувати мережеву інфраструктуру відповідно до уніфікованих критеріїв.

На національному рівні існують стандарти та нормативні акти, які можуть вимагати від провайдерів дотримання певних показників швидкості та стабільності мережі. Такі вимоги стимулюють операторів інвестувати в інфраструктуру, покращувати маршрутизацію та модернізувати обладнання, щоб відповідати очікуванням користувачів, регуляторів.

Для дослідників і розробників програмних інструментів, знання про існуючі стандарти та рекомендації надає кілька переваг. По-перше, це дозволяє створювати програмне забезпечення, результати якого можна порівнювати з даними інших досліджень у світі. По-друге, орієнтуючись на міжнародні норми, можна розробити систему, яка буде зрозумілою і корисною не лише локально, а й у глобальному масштабі. По-третє, використання загальноприйнятих вимог та методик робить результати більш надійними для всіх зацікавлених сторін — від інженерів до кінцевих користувачів.

Таким чином, огляд міжнародних стандартів та рекомендацій допомагає вписати результати конкретного дослідження у ширший контекст і забезпечує теоретичну базу для подальшого вдосконалення обраних методів вимірювання та інструментів аналізу мережевої продуктивності.

2 МЕТОДОЛОГІЯ ТА РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ВИМІРЮВАННЯ МЕРЕЖЕВИХ ЗАТРИМОК

2.1 Методологія вимірювання швидкості передачі даних з урахуванням затримок

Для точного дослідження швидкості передачі даних та впливу мережевих затримок необхідний системний підхід, що враховує всі етапи передачі інформації: від клієнтського пристрою до кінцевого сервера.

У цій роботі обрана методологія ґрунтується на симуляції реальних умов користування мережею, що дає змогу отримати об’єктивні результати. Вона поєднує практичні експерименти з аналізом факторів, які впливають на продуктивність мережі, щоб було більш зрозуміло на Аркуші 1 МР.МТТм - 29.00.00.001 – зображена функціональна схема яка і буде використовуватись в проекті.

Етапи вимірювання:

Дослідження поділяється на кілька ключових етапів:

1. **Підготовка середовища тестування.**

   Було створено серверну інфраструктуру для проведення вимірювань. Сервер виконує роль точки обробки даних, надаючи клієнтам інтерфейси для завантаження, вивантаження даних і проведення вимірювань затримок. Це дозволяє уникнути залежності від сторонніх сервісів і повністю контролювати процес тестування.

2. **Реалізація методів вимірювання.**

   Швидкість передачі даних тестується через окремі ендпоінти:

   - **Швидкість завантаження (download):** клієнт завантажує великі обсяги даних із сервера, вимірюючи середню швидкість передачі.
   - **Швидкість вивантаження (upload):** клієнт передає дані на сервер, де вимірюється обсяг переданого за одиницю часу.

- **Мережеві затримки (ping):** використовується WebSocket-з’єднання, що дозволяє миттєво обмінюватися сигналами для оцінки часу передачі.
- **Коливання затримок (jitter):** аналізується стабільність передачі даних через послідовні вимірювання пінгу.

3. **Вибір параметрів тестування.**

Для забезпечення точності вимірювань застосовуються кілька підходів:

- Дані передаються в пакетах різного розміру (наприклад, 1 МБ, 10 МБ, 100 МБ), щоб оцінити вплив розміру пакета на швидкість.
- Тестування виконується у різний час доби для аналізу впливу завантаженості мережі.
- Усі результати зберігаються в базі даних SQLite для подальшого аналізу.

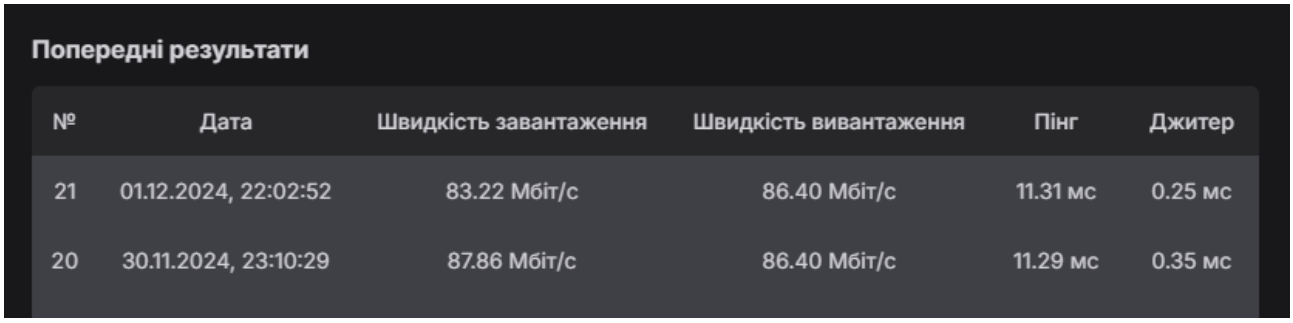
Інструменти для вимірювання:

Для реалізації вимірювань використовується програмний код, що забезпечує гнучкість та надійність процесу. Наприклад, ендпоінти **/download** і **/upload** налаштовані так, щоб працювати з великими обсягами даних без компресії, що дозволяє уникнути викривлення результатів. WebSocket-з’єднання забезпечує точний контроль часу передачі сигналів для пінгу та джиттера.

Особливістю обраної методології є використання **псевдовипадкових даних** для тестів. Це дозволяє забезпечити однакові умови для всіх користувачів і виключити вплив типу контенту на результати.

Обробка даних. Результати тестів зберігаються у вигляді структурованих записів що зображено на рис. 2.1 із такими параметрами:

- Дата й час тестування.
- Швидкість завантаження та вивантаження.
- Середній пінг.
- Середній джиттер.



Попередні результати

№	Дата	Швидкість завантаження	Швидкість вивантаження	Пінг	Джитер
21	01.12.2024, 22:02:52	83.22 Мбіт/с	86.40 Мбіт/с	11.31 мс	0.25 мс
20	30.11.2024, 23:10:29	87.86 Мбіт/с	86.40 Мбіт/с	11.29 мс	0.35 мс

Рисунок 2.1 – Попередні результати на моїй розробці (на сайті)

Такий підхід дозволяє відстежувати динаміку змін у мережевих параметрах, аналізувати залежність затримок від зовнішніх умов, таких як завантаженість мережі або відстань до сервера.

Після збору даних проводиться їх аналіз. Наприклад, якщо швидкість передачі даних зменшується в години пікового навантаження, це може вказувати на проблему з пропускною здатністю мережі. Аналогічно, високий джиттер свідчить про нестабільність, яка може бути викликана перевантаженням маршрутизаторів або проблемами з маршрутизацією.

Методологія, яка була розроблена в рамках цього дослідження, може бути використана не лише для наукових цілей, а й для практичного моніторингу мереж. Провайдери можуть застосовувати цей підхід для оцінки якості своїх послуг, а кінцеві користувачі — для діагностики проблем із підключенням.

Таким чином, запропонована методологія дозволяє комплексно оцінювати швидкість передачі даних і затримки в реальних умовах, що робить її цінним інструментом для оптимізації роботи сучасних мереж.

2.2 Показники мережевих затримок: latency, jitter, ping

Для оцінки якості мережевого з’єднання використовуються кілька ключових показників, які описують затримки при передачі даних. До них належать **latency**, **jitter** та **ping**. Ці параметри відображають, як швидко та

стабільно дані передаються між пристроями, вони є критично важливими для роботи додатків реального часу, таких як відеоконференції чи онлайн-ігри.

Latency — це час, потрібний для того, щоб пакет даних пройшов від відправника до отримувача й назад. Цей показник зазвичай вимірюється в мілісекундах (мс). Затримка складається з кількох компонентів:

- Час передачі сигналу через фізичне середовище (оптоволокно, кабель тощо).
- Час обробки на проміжних маршрутизаторах.
- Час очікування в чергах при перевантаженні мережі.

У програмному коді для вимірювання **latency** використовується WebSocket-з'єднання. Сервер отримує від клієнта запит із міткою часу, після чого відправляє відповідь із тією ж міткою, дозволяючи клієнту обчислити час передачі.

Формула для розрахунку latency:

$$L = T_{receive} - T_{send}, \quad (2.1)$$

де: L — затримка (латентність); T_{send} — час відправки запиту; $T_{receive}$ — час отримання відповіді.

У JavaScript це реалізовано так:

$$const\ latency = receiveTime - sendTime;$$

Ping — це один із методів вимірювання latency. Для цього відправляється спеціальний контрольний пакет даних (ICMP або через WebSocket), а потім вимірюється час його проходження до сервера і назад.

Ping часто використовується як базовий інструмент для перевірки доступності серверів. Низький ping (наприклад, < 50 мс) вказує на швидке з'єднання, тоді як високий (> 150 мс) може бути проблемою для інтерактивних додатків.

Формула для розрахунку ping:

$$P = T_{round\text{-}trip}, \quad (2.2)$$

де: $T_{round\text{-}trip}$— час кругового проходження пакета.

Реалізація у коді:

$$const\ ping = performance.now() - sendTime;$$

Ping вимірюється кілька разів, щоб отримати середнє значення, адже затримка може варіюватися.

Jitter — це міра варіативності затримки між послідовними пакетами даних. Якщо пакети приходять із різними затримками, це може викликати проблеми в потокових додатках, наприклад, втрату якості звуку чи відео.

Джиттер розраховується як середнє відхилення між затримками окремих пакетів:

$$J = \frac{1}{n}\sum_{i=1}^{n}|(T_{i+1} - T_i)|. \quad (2.3)$$

де: J — джиттер; T_i— час прибуття і-го пакета; n — кількість пакетів.

У коді це реалізовано через порівняння часу між послідовними сигналами:

$$const\ delta\ =\ currentTransit\ -\ previousTransit;$$

$$jitterValue += (Math.abs(delta) - jitterValue)/16;$$

У моєму програмному забезпеченні ці показники вимірюються у реальному часі. Сервер використовує WebSocket для обміну даними, що дозволяє точно оцінити затримки та джиттер. Зібрані результати зберігаються в базі даних SQLite для подальшого аналізу, зокрема, визначення пікових навантажень мережі та оцінки її стабільності.

Знання параметрів **latency**, **jitter** та **ping** дозволяє діагностувати проблеми з мережею. Наприклад, високий джиттер може свідчити про перевантаження маршрутизаторів, а підвищений пінг — про проблеми з маршрутизацією або фізичними обмеженнями з'єднання.

Запропонована методика вимірювання цих параметрів є універсальною й може бути використана як кінцевими користувачами для тестування якості з’єднання, так і провайдерами для моніторингу мережі та оптимізації її роботи.

2.3 Обґрунтування вибору Node.js для аналізу мережевих затримок

Для реалізації дослідження мережевих параметрів (пінг, джиттер, швидкість завантаження та вивантаження даних) було обрано платформу Node.js, завантажити її можна на офіційному сайті, який зображений на рис. 2.2.

Головною причиною такого вибору стала її подієво-орієнтована архітектура, що забезпечує асинхронну обробку запитів у режимі реального часу. Завдяки цьому можна одночасно взаємодіяти з великою кількістю клієнтів без істотних втрат продуктивності, що є критичним для точного вимірювання та аналізу мережевих затримок.

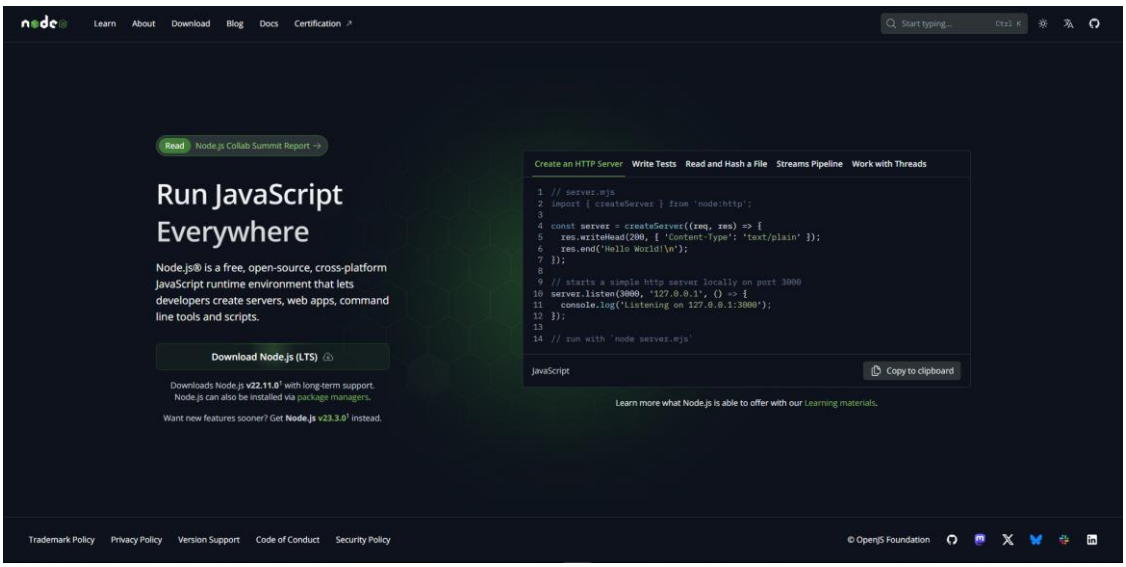


Рисунок 2.2 – Сайт Node.JS та лінк на завантаження

Node.js базується на високопродуктивному «движку» V8, що дозволяє ефективно обробляти велику кількість паралельних з’єднань та надає доступ до потужного стандартного модуля HTTP. Це спрощує реалізацію ендпоінтів, через

які клієнти можуть завантажувати або вивантажувати дані для тестування пропускної здатності мережі [20].

Наприклад, ендпоінт «/download» може надавати великий масив даних у потоковому режимі, мінімізуючи затримки, а «/upload» приймає інформацію без компресії для коректного вимірювання реального часу передачі даних.

Важливим аспектом для оцінки якості мережі є вимірювання часу відгуку у двосторонній комунікації. Node.js забезпечує інтеграцію з WebSocket через модуль «ws», що надає можливість реалізувати двосторонній, постійний канал зв'язку між клієнтом та сервером, різниця Websocket з'єднання та HTTP зображена на рис. 2.3.

Завдяки цьому типу підключення можна відправляти «ping» та отримувати «pong» у режимі реального часу, обчислюючи затримки з точністю до мілісекунд. Такий підхід дозволяє не лише фіксувати середні значення пінгу, а й аналізувати його коливання (джиттер).

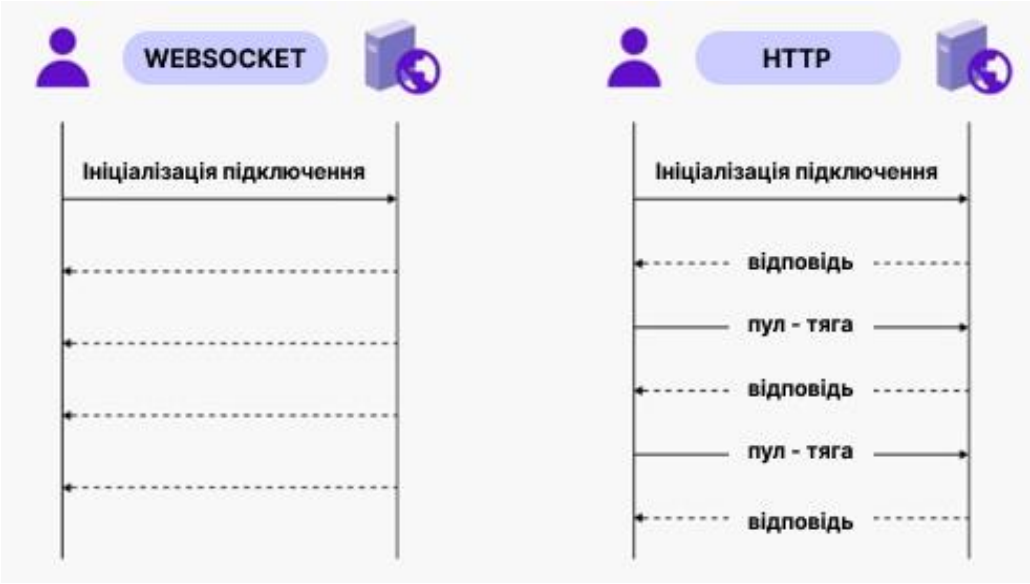


Рисунок 2.3 – Порівняння Websocket підключення з HTTP

Інтеграція з базами даних (наприклад, з SQLite) у Node.js реалізується швидко та прозоро, що дозволяє зберігати результати тестів на локальному сервері для подальшої аналітики. Збереження історії параметрів, таких як пінг чи

швидкість завантаження, відкриває можливості для побудови трендів та виявлення закономірностей змін якості зв’язку.

Крім того, широка екосистема пакетів npm робить Node.js гнучким інструментом для розширення функціоналу. Наприклад, модулі «cors» та «compression» дозволяють налаштовувати політики доступу та режими стиснення даних, оптимізуючи середовище для більш точних вимірювань мережевих характеристик.

Таким чином, вибір Node.js як технологічної платформи для аналізу мережевих затримок був зумовлений її асинхронною архітектурою, підтримкою сучасних протоколів, простою інтеграцією з базами даних та гнучким налаштуванням через широкий набір пакетів. Це забезпечує високу точність та продуктивність інструментів, створених у рамках даного дослідження.

2.4 Міжплатформенність застосунків на Node.js

Однією з ключових переваг Node.js є її здатність функціонувати на різних операційних системах (Windows, Linux, macOS тощо) без необхідності внесення суттєвих змін до коду. Ця міжплатформенність забезпечує зручність у розгортанні проекту для тестування мережевих затримок у різноманітних середовищах та на різних пристроях [20].

Використання JavaScript як основної мови розробки робить Node.js природним вибором для створення уніфікованого серверного рішення, яке легко взаємодіє з браузерними клієнтами, мобільними застосунками та десктопними системами. Завдяки цьому користувачі можуть проводити тестування як із персональних комп’ютерів, так і з мобільних пристроїв, отримуючи коректні результати незалежно від платформи, ще деякі характеристики зображені на рис. 2.4.

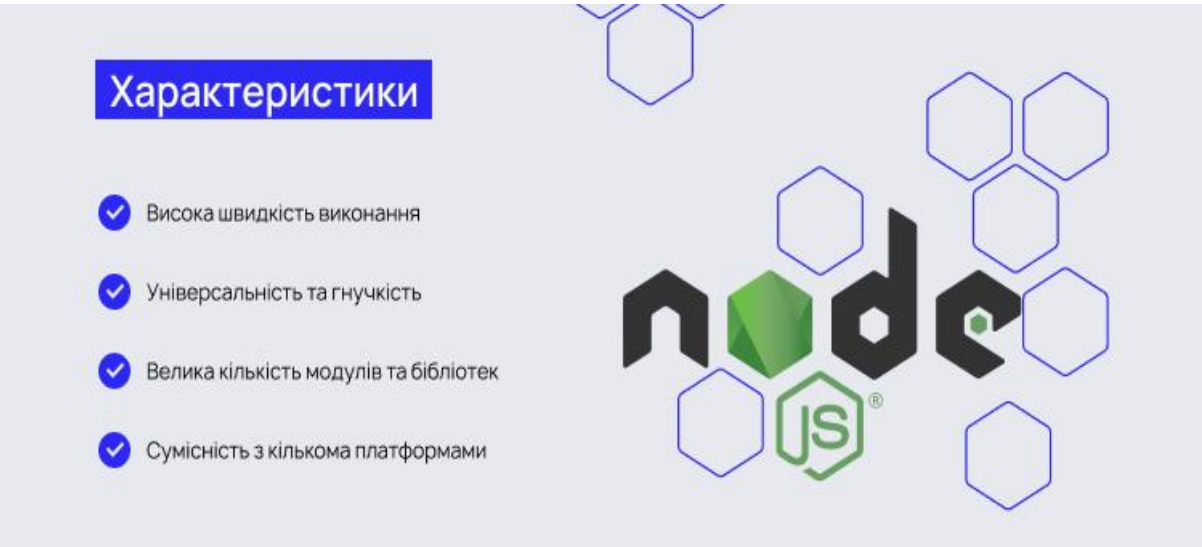


Рисунок 2.4 – Характеристики та ще деякі плюси використання Node.js

Додаткова гнучкість досягається завдяки можливості роботи Node.js у контейнеризованих середовищах, таких як Docker. Це уможливлює швидке та просте масштабування застосунку, його розгортання у хмарних інфраструктурах та створення стандартизованого середовища для тестів. Наявність npm-пакетів, сумісних із різними платформами, дозволяє зберегти єдину кодову базу, не залежно від операційної системи сервера.

Завдяки широкій підтримці стандартних протоколів (HTTP, WebSocket) та легкій адаптації до різних середовищ виконання - Node.js дає змогу забезпечити універсальну доступність інструментів для аналізу мережевих параметрів. Це підвищує точність, надійність та зручність користування розробленим рішенням, оскільки розгортання і використання не залежить від конкретної операційної системи чи типу пристрою.

Таким чином, можливість безпроблемного запуску, підтримки Node.js-застосунків на різних платформах робить цю технологію оптимальним вибором для забезпечення доступності, масштабованості проєктів, які спрямовані на тестування і моніторинг якості мережевих з’єднань.

2.5 Аналіз програмної архітектури та принципів підвищення точності вимірювань

При розробці програмного забезпечення, що вимірює ключові мережеві параметри (швидкість, пінг, джиттер) – важливо не лише забезпечити коректність і стабільність роботи коду, а й продумати архітектуру системи. Кінцевою метою є мінімізація похибок, які пов’язані із затримками при обробці даних, а також уніфікація підходу до вимірювань, щоб користувачі отримували максимально достовірні результати.

Використання Node.js та асинхронної архітектури. Серверна частина системи реалізована за допомогою Node.js. Цей вибір невипадковий: платформа Node.js базується на подієво-орієнтованій та асинхронній моделі обробки запитів [20]. Завдяки цьому сервер може ефективно опрацьовувати велику кількість одночасних з’єднань, майже не втрачаючи продуктивності. Це важливо з точки зору вимірювання: під час тестів користувач може запитувати дані великого обсягу (для перевірки швидкості завантаження) або надсилати значні пакети (для перевірки швидкості вивантаження). Асинхронна модель дає змогу уникнути «вузьких місць», які могли б призвести до штучного збільшення затримок.

Чітке розмежування ролей клієнта та сервера. Архітектура «клієнт-сервер» у даному випадку чітко розділяє обов’язки сторін. Сервер відповідає за надання тестових даних, прийом результатів та їх збереження до бази даних. Клієнтська частина — це інтерфейс, з яким взаємодіє користувач, ініціюючи тести та відображаючи результати. Такий поділ спрощує пошук та усунення проблем: якщо затримка виникла на стороні клієнта, то оптимізацію можна здійснювати саме там, не зачіпаючи серверну логіку, і навпаки.

Використання WebSocket для вимірювання пінгу та джиттера. Традиційні методи вимірювання пінгу (на основі ICMP) не завжди доступні або точні в умовах сучасного інтернету, де маршрутизатори інколи блокують або обробляють такі пакети інакше. Обравши WebSocket як канал двостороннього

зв’язку – розробник отримує можливість вимірювати час проходження повідомлень безпосередньо на прикладному рівні. Це робить вимірювання більш наближеними до реальних умов передачі даних веб-додатками. Крім того, WebSocket дозволяє гнучко контролювати інтервали запитів, збирати статистику й одразу її інтерпретувати.

Мінімізація впливу сторонніх факторів. Для забезпечення максимальної точності під час тестів було свідомо відмовлено від стиснення відповідей на стороні сервера (компресії). Компресія може штучно змінювати час передачі та отримані результати, оскільки реальний трафік користувача не завжди стискається однаково. Використовуючи опцію compression({ level: 0 }), ми досягаємо максимально наближеного до реальності часу передачі даних. Це важливо для сценаріїв, коли користувач хоче зрозуміти справжні можливості свого підключення до інтернету, без додаткових «покращень», які могли б спотворити картину.

Дотримання структурованого підходу до коду. Розробка коду за принципом «одна функція — одна задача» спрощує його тестування та верифікацію. Усі ключові операції, такі як завантаження, вивантаження, вимірювання пінгу, збереження результатів до бази даних, реалізовані окремими логічними блоками. Це дозволяє швидко виявити джерело похибок та оптимізувати окремі частини системи без ризику порушити інші компоненти.

Взаємодія з базою даних для аналізу результатів. Використання локальної бази даних (SQLite) допомагає зберігати результати вимірювань та згодом аналізувати їх у динаміці. З часом можна накопичити статистику про те, як змінюються швидкість чи затримки у різний час доби або за різного завантаження мережі. Такий аналіз дасть можливість знаходити закономірності, виявляти проблемні ділянки та надалі вдосконалювати методику вимірювань.

2.6 Постановка задачі та вимоги до програмного забезпечення

Метою розробки є створення програмного забезпечення, яке дозволить вимірювати ключові параметри мережі — затримки (latency), пінг (ping), джиттер (jitter) та швидкість передачі даних (upload/download). Основними завданнями є забезпечення точності вимірювань, стабільності роботи під час великого навантаження, а також простоти у використанні для кінцевого користувача.

Постановка задачі. Проблема вимірювання мережевих параметрів полягає у поєднанні точності отриманих результатів із високою продуктивністю роботи системи. Інструмент має коректно працювати в умовах змінної пропускної здатності мережі, забезпечуючи реальні результати, незалежно від платформи чи операційної системи користувача.

Для цього потрібно реалізувати:

- Сервер, який обробляє запити клієнта, забезпечуючи передачу даних для тестування швидкості та вимірювання затримок.
- Клієнтську частину, яка проводить вимірювання на боці користувача й передає результати на сервер.
- Базу даних для зберігання історії результатів тестів.

Код повинен забезпечити:

- Вимірювання **ping** через протокол WebSocket.
- Аналіз **jitter** через обчислення варіації між затримками пакетів.
- Вимірювання швидкості завантаження та вивантаження даних за допомогою окремих ендпоінтів /download і /upload.
- Мінімізацію впливу стиснення чи кешування даних для збереження автентичності тестів.

Функціональні вимоги:

1. **Швидкість завантаження та вивантаження.** Ендпоінти /download та /upload повинні генерувати та обробляти великі обсяги даних, використовуючи потоки. Наприклад, у коді генерація даних для

завантаження реалізована через об'єкт Readable, який передає дані блоками:

Код JavaScript

```
const readable = new Readable({
  read() {
    if (bytesSent >= size) {
      this.push(null); // Завершення передачі
      return;
    }
    const buffer = Buffer.alloc(currentChunkSize,
'0123456789abcdef', 'utf-8');
    this.push(buffer);
    bytesSent += currentChunkSize;
  }
});
```

Цей підхід дозволяє ефективно працювати навіть із великими обсягами даних.

2. **Вимірювання ping та jitter.** Для забезпечення точності використовується WebSocket-з'єднання, яке дозволяє клієнту обмінюватися даними із сервером у реальному часі. У коді реалізована функція обчислення часу передачі сигналу:

Код JavaScript

```
const latency = performance.now() - sendTime;
```

Джиттер розраховується як середнє відхилення між часом доставки послідовних сигналів:

Код JavaScript

```
const delta = currentTransit - previousTransit;
jitterValue += (Math.abs(delta) - jitterValue) / 16;
```

3. **Збереження даних.**

   Результати тестів зберігаються в базі даних SQLite для подальшого аналізу:

Код JavaScript

```
db.run(
  `INSERT INTO results (date, download_speed, upload_speed, ping, jitter) VALUES (?, ?, ?, ?, ?)`,
  [date, downloadSpeed, uploadSpeed, ping, jitter]
);
```

4. **Інтерфейс.**

   Користувач отримує зручний інтерфейс, який дозволяє запускати тест, переглядати результати та візуалізувати дані.

   Нефункціональні вимоги:

   - Продуктивність. Сервер повинен обробляти сотні одночасних з'єднань без зниження точності вимірювань.
   - Міжплатформенність. ПЗ має однаково працювати на Windows, macOS, Linux, а також підтримувати мобільні платформи.
   - Безпека. Передача даних повинна виконуватися через захищені протоколи, якщо це можливо.

Розроблений інструмент забезпечить проведення точних тестів мережевих параметрів із можливістю їх аналізу та збереження. Інтеграція серверної частини з базою даних і клієнтським інтерфейсом створить зручний і надійний інструмент для моніторингу мережі.

2.7 Архітектура клієнт-серверної системи вимірювання

Архітектура системи вимірювання мережевих затримок побудована за моделлю клієнт-сервер, де сервер відповідає за обробку запитів і виконання вимірювань, а клієнтська частина здійснює ініціацію тестів, передачу даних та

отримання результатів. Така структура дозволяє розподілити обчислювальні ресурси між клієнтом і сервером, забезпечуючи точність і масштабованість.

2.7.1 Ключові компоненти архітектури

Серверна частина:

Сервер реалізовано на платформі Node.js. Він виконує кілька важливих функцій:

- Обробка запитів для тестів швидкості завантаження та вивантаження.
- Вимірювання пінгу та джиттера через WebSocket-з'єднання.
- Збереження результатів тестів у базі даних SQLite для подальшого аналізу.

Архітектура серверної частини представлена такими компонентами:

1. **HTTP-сервер:** використовується для обробки запитів /download та /upload, що дозволяє клієнту проводити тести швидкості.
2. **WebSocket-сервер:** забезпечує двостороннє спілкування з клієнтом для тестування затримок (ping) і нестабільності мережі (jitter).
3. **База даних:** SQLite використовується для збереження результатів тестів. Це дозволяє зберігати історію вимірювань, аналізувати дані в довгостроковій перспективі.

Код серверної частини для вимірювання пінгу:

Код JavaScript

```
wss.on('connection', (ws) => {
  ws.on('message', (message) => {
    const data = JSON.parse(message);
    if (data.type === 'ping') {
      ws.send(JSON.stringify({ type: 'pong', sendTime: data.sendTime }));
    }
  });
});
```

Сервер також генерує великі обсяги даних для тестування швидкості завантаження:

Код JavaScript

```
const readable = new Readable({
  read() {
    if (bytesSent >= size) {
      this.push(null); // Завершення передачі
      return;
    }
    const buffer = Buffer.alloc(currentChunkSize, '0123456789abcdef', 'utf-8');
    this.push(buffer);
    bytesSent += currentChunkSize;
  }
});
```

Клієнтська частина:

Клієнт відповідає за ініціацію тестів, передачу запитів серверу, обчислення параметрів і візуалізацію результатів. Основні функції клієнта включають:

- Відправлення запитів на сервер для тестів швидкості завантаження та вивантаження.
- Ініціація пінг-тестів через WebSocket-з'єднання.
- Відображення результатів тестів у зручному форматі.

Клієнтська частина реалізована на JavaScript і виконується в браузері користувача. Вона використовує методи **fetch** для взаємодії з сервером:

Код JavaScript

```
async function startDownloadTest() {
  const response = await fetch('/download?size=10000000');
  const reader = response.body.getReader();
  let totalBytes = 0;

  while (true) {
    const { done, value } = await reader.read();
    if (done) break;
    totalBytes += value.length;
  }
}
```

Для тесту «пінгу» клієнт відправляє мітку часу на сервер:

Код JavaScript

```
ws.send(JSON.stringify({ type: 'ping', sendTime: performance.now() }));
```

Сервер повертає отриману мітку часу, що дозволяє клієнту обчислити затримку:

Код JavaScript

```
const ping = performance.now() - sendTime;
```

2.7.2 Логіка взаємодії клієнта та сервера

1. **Тест швидкості завантаження:** клієнт надсилає запит на ендпоінт /download, сервер генерує великі обсяги даних і передає їх клієнту. Клієнт вимірює час, необхідний для отримання цих даних, і обчислює швидкість передачі.
2. **Тест швидкості вивантаження:** клієнт завантажує на сервер великі обсяги даних через ендпоінт /upload. Сервер приймає ці дані, а клієнт фіксує швидкість передачі.
3. **Вимірювання пінгу та джиттера:** клієнт надсилає запити через WebSocket-з'єднання, отримує відповіді й обчислює затримку передачі (ping) та її варіації (jitter).
4. **Збереження результатів:** результати кожного тесту передаються на сервер через ендпоінт /results для збереження в базі даних.

2.8 Реалізація програмного коду на Node.js

2.8.1 Розробка серверної частини

Серверна частина програмного забезпечення реалізована на платформі Node.js із використанням бібліотек **express**, **http**, та **ws**. Основною метою є обробка запитів для тестування швидкості передачі даних, пінгу, джиттера та збереження результатів у базі даних. Логіка серверу була розроблена з

урахуванням асинхронного характеру запитів, що дозволяє обробляти велику кількість одночасних підключень.

1. Налаштування серверу:

Сервер використовує Express для обробки HTTP-запитів і WebSocket для двосторонньої комунікації в режимі реального часу. Під час запуску створюється HTTP-сервер і WebSocket-сервер:

Код JavaScript

```
const app = express();
const server = http.createServer(app);
const wss = new WebSocket.Server({ server });
```

Express також обробляє статичні файли через папку **public**, де розміщений клієнтський інтерфейс.

2. Тест швидкості завантаження (download):

Ендпоінт /download забезпечує передачу великих обсягів даних для тестування швидкості завантаження. Ця реалізація дозволяє серверу генерувати дані динамічно, без завантаження великих файлів у пам’ять.

Дані передаються блоками через об’єкт Readable:

Код JavaScript

```
app.get('/download', (req, res) => {
  const size = parseInt(req.query.size) || 100 * 1024 * 1024; // Розмір файлу за замовчуванням
  const chunkSize = 1024 * 1024; // Блок 1 МБ
  let bytesSent = 0;

  const readable = new Readable({
    read() {
      if (bytesSent >= size) {
        this.push(null);
        return;
      }
      const remaining = size - bytesSent;
      const buffer = Buffer.alloc(Math.min(chunkSize, remaining), '0123456789abcdef', 'utf-8');
      this.push(buffer);
```

```
      bytesSent += buffer.length;
    }
  });

  res.set({
    'Content-Type': 'application/octet-stream',
    'Content-Length': size,
    'Cache-Control': 'no-cache, no-store, must-revalidate',
  });
  readable.pipe(res);
});
```

3. Тест швидкості відвантаження (upload):

Ендпоінт **/upload** отримує дані від клієнта й одразу завершує запит, оцінюючи час передачі:

Код JavaScript

```
app.post('/upload', express.raw({ type: '*/*', limit: '1gb' }), (req, res) => {
  res.set({
    'Cache-Control': 'no-cache, no-store, must-revalidate',
  });
  res.end();
});
```

Цей підхід мінімізує затримки обробки, забезпечуючи точні результати.

4. Тестування пінгу та джиттера:

WebSocket-сервер обробляє запити для оцінки пінгу та джиттера. Клієнт надсилає ping, а сервер відповідає pong із часом відправлення:

Код JavaScript

```
wss.on('connection', (ws) => {
  ws.isAlive = true;

  ws.on('message', (message) => {
    const data = JSON.parse(message);
    if (data.type === 'ping') {
      ws.send(JSON.stringify({ type: 'pong', sendTime: data.sendTime }));
    }
  });

  ws.on('pong', () => {
    ws.isAlive = true;
  });
});
```

Регулярна перевірка активності з’єднань запобігає утворенню "мертвих" сесій.

5. Збереження результатів у базу даних:

Сервер використовує SQLite для зберігання результатів. Записи включають дату, швидкість завантаження та вивантаження, пінг і джиттер:

Код JavaScript

```
db.run(
  `INSERT INTO results (date, download_speed, upload_speed, ping, jitter)
VALUES (?, ?, ?, ?, ?)`,
  [date, downloadSpeed, uploadSpeed, ping, jitter]
);
```

Це дозволяє створювати історію тестів для подальшого аналізу.

6. Отримання результатів:

Ендпоінт **/results** надає клієнту останні результати тестів у форматі JSON:

Код JavaScript

```
app.get('/results', (req, res) => {
  db.all('SELECT * FROM results ORDER BY id DESC LIMIT 10', [], (err,
rows) => {
    if (err) {
      res.status(500).send('Помилка отримання результатів');
    } else {
      res.json(rows);
    }
  });
});
```

Особливості реалізації:

1. **Асинхронна модель**

   Усі операції, включно з обробкою WebSocket-запитів і взаємодією з базою даних, реалізовані асинхронно. Це дозволяє серверу обробляти велику кількість одночасних підключень.

2. **Мінімізація впливу затримок**

   Відключення стиснення відповідей через compression({ level: 0 }) забезпечує автентичність тестів.

3. **Захист серверу**

   Використання перевірки активності WebSocket-з'єднань та обробки помилок через middleware підвищує стабільність роботи.

Розроблена серверна частина відповідає вимогам дослідження, забезпечуючи точність, стабільність і продуктивність.

2.8.2 Розробка клієнтської частини

Клієнтська частина реалізована з використанням HTML, JavaScript та CSS. Вона відповідає за взаємодію з користувачем, запуск тестів швидкості завантаження, відвантаження, пінгу та джиттера, а також за візуалізацію отриманих результатів.

1. Інтерфейс користувача (HTML):

Файл index.html містить структуру інтерфейсу:

- Кнопка старту для запуску тесту.
- Модальні вікна для кожного типу тесту (завантаження, відвантаження, пінг, джиттер).
- Таблиця результатів для збережених даних із бази даних.

Код побудований також з використанням TailwindCSS для адаптивного дизайну, що дозволяє налаштовувати елементи залежно від розміру екрану.

2. Динамічні функції (JavaScript):

Скрипт /public/script.js забезпечує функціональність клієнтської частини. Ключові компоненти:

- Ініціалізація та запуск тестів:

  Обробник кнопки старту викликає функцію startTest, яка запускає послідовність тестів, збирає дані та надсилає результати на сервер:

Код JavaScript

```
startButton.addEventListener('click', () => {
  if (isTesting) return;
  isTesting = true;
  startButton.disabled = true;
  startTest();
});
```

- Тестування пінгу та джиттера:

  Використовується WebSocket для вимірювання затримок та нестабільності мережі:

Код JavaScript

```
async function startDownloadTest() {
  const testDuration = DOWNLOAD_DURATION * 1000;
  let totalBytes = 0;

  const downloadUrl =
`/download?size=10000000&r=${Math.random()}`;
  const response = await fetch(downloadUrl, { cache: 'no-store'
});
  const reader = response.body.getReader();

  const startTime = performance.now();
  while ((performance.now() - startTime) < testDuration) {
    const { value, done } = await reader.read();
    if (done) break;
    totalBytes += value.length;
  }
  return totalBytes / testDuration;
}
```

- Відображення результатів:

Для відображення використовуються графіки, створені бібліотекою ApexCharts:

Код JavaScript

```
function createPingChart(pingTimes) {
  const options = {
    series: [{ name: "Ping", data: pingTimes }],
    chart: { type: "line" },
    xaxis: { categories: Array.from(pingTimes.keys()) },
  };
  new ApexCharts(document.querySelector("#modal-ping-content"),
options).render();
}
```

3. Інтеграція із сервером:

Клієнтська частина взаємодіє із сервером через HTTP-запити та WebSocket:

- Надсилання результатів на сервер:

Код JavaScript

```
async function sendResultsToServer(results) {
  await fetch('/results', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify(results),
  });
}
```

- Отримання та відображення попередніх результатів:

Код JavaScript

```
async function fetchAndDisplayResults() {
  const response = await fetch('/results');
  const results = await response.json();
  results.forEach(result => {
    const row = `<tr>
      <td>${result.id}</td>
      <td>${result.date}</td>
      <td>${result.download_speed} Мбіт/с</td>
      <td>${result.upload_speed} Мбіт/с</td>
      <td>${result.ping} мс</td>
      <td>${result.jitter} мс</td>
    </tr>`;
```

```
    resultsTableBody.innerHTML += row;
  });
}
```

4. Особливості реалізації:

- **Адаптивний дизайн:** інтерфейс виглядає коректно на мобільних і десктопних пристроях.
- **Зручність використання:** чіткі індикатори стану тестування та модальні вікна для результатів.
- **Візуалізація:** інтерактивні графіки пінгу та джиттера допомагають краще розуміти результати.

Клієнтська частина поєднує функціональність і естетику, забезпечуючи зручний інтерфейс для користувачів.

2.8.3 Структура та код

Структура проекта виглядає як зображено на рис. 2.5.

Рисунок 2.5 – Структура проекта

Почнемо з файла **package.json**, який знаходиться в основній папці проекта. Він потрібен щоб запустити команду “npm i” та встановити модулі які використовує проект.

package.json

```
{
  "name": "speedtest",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "dev": "nodemon server.js",
    "start": "node server.js"
  },
  "keywords": [],
  "author": "Mykola Kozaruk",
  "license": "ISC",
  "description": "Розробка програмного забезпечення для вимірювання
швидкості інтернету для магістрської роботи МТТм-23-1",
  "dependencies": {
    "axios": "^1.7.7",
    "body-parser": "^1.20.3",
    "compression": "^1.7.5",
    "cors": "^2.8.5",
    "express": "^4.21.1",
    "nodemon": "^3.1.7",
    "socket.io": "^4.8.1",
    "sqlite3": "^5.1.7",
    "ws": "^8.18.0"
  }
}
```

Наступним буде **server.js** – це основний файл серверної частини, який запускається при відтворені команди “**npm start**” або «**node server.js**», він знаходиться в «Додаток А».

В ньому досить багато коду, це пов’язано з тим що це самий важливий файл, він запускає не тільки серверну частину, а й передає дані на клієнтську частину.

Дальше йде клієнтська частина, яка знаходиться по шляху /public/. Наступним файлом йде **index.html,** в ньому знаходиться то, що клієнт (користувач) буде бачити, а точніше сама HTML5 структура цього проекту, який відображається, код цього файлу знаходиться в «Додаток Б»**.**

А дальше клієнтська частина «**script.js**», вона також важлива, без неї не будуть нормально відображатись графіки, та не буде підключення до серверної частини, код цього файлу находиться в «Додаток В».

Тепер прийдем до папки /public/css/ та файлу **mercury_dev.css –** «Додаток Г», це каскадна таблиця стилів, вона робить сам дизайн сайту приємнішим, а не просто чорно-білий аркуш.

2.9 Імітація вимірювання програмним забезпеченням

З самого початку – заходимо в консоль, та переходимо в папку де знаходиться проект, до прикладу у мене це D:\Work\app\, в терміналі це **“cd D:\Work\app”,** як зображено на рис. 2.6.

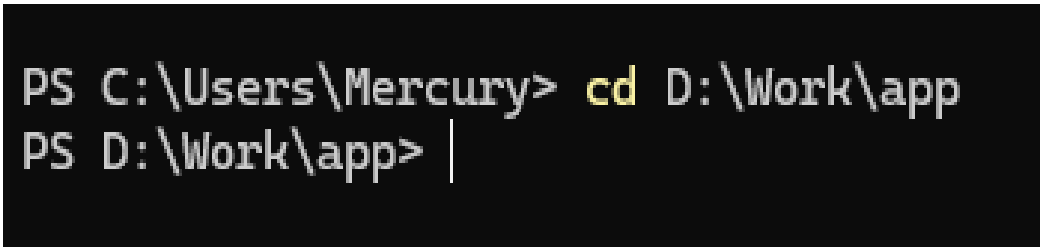


Рисунок 2.6 – Відкриття папки

Після чого встановлюємо модулі командою **«npm i»** щоб програмне забезпечення правильно працювало**,** як зображено на рис. 2.7.

```
PS D:\Work\app> npm i

up to date, audited 251 packages in 1s

29 packages are looking for funding
  run `npm fund` for details

2 moderate severity vulnerabilities

To address all issues, run:
  npm audit fix

Run `npm audit` for details.
PS D:\Work\app>
```

Рисунок 2.7 – Встановлення модулів

А дальше запускаємо програмне забезпечення через команду **«npm run start»**, як зображено на рис 2.8.

```
PS D:\Work\app> npm run start

> speedtest@1.0.0 start
> node server.js

Сервер запущений на порту 3000
```

Рисунок 2.8 – Запуск програмного забезпечення

Заходимо по локальному адресу + порт 3000. До прикладу http://localhost:3000, відкривається сайт, який виглядає як зображено на рис. 2.9, після чого нажимаємо кнопку «Почати тест», та запускається тест.

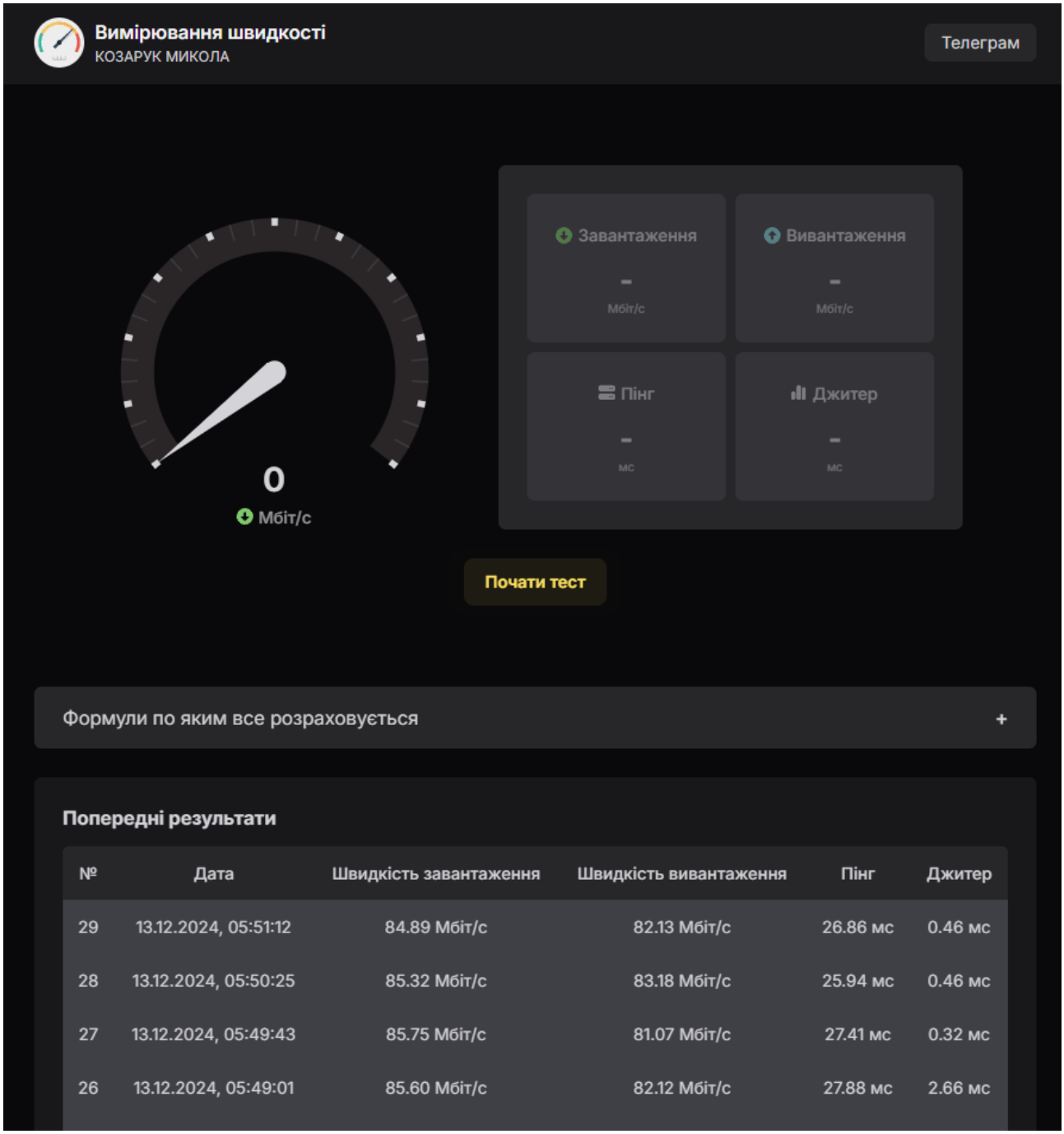


Рисунок 2.9 – Зовнішній вигляд сайту

Після завершення тесту можна натиснути на плитки які зображені на рис. 2.10, та навестись на графік.

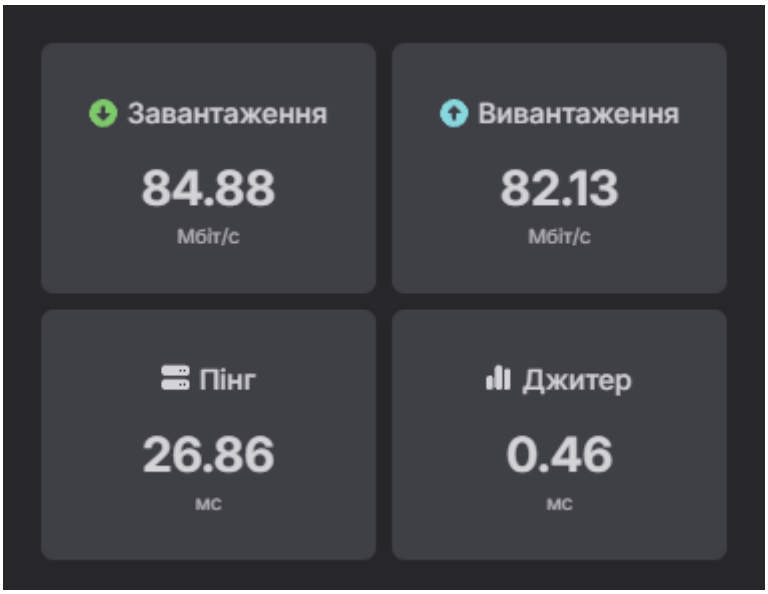


Рисунок 2.10 – Плитки на які можна натиснути після завершення тесту

Щоб дізнатись більш детальну інформацію достатньо навестись на графік, по типу який пакет/час та інше, це зображено на рис. 2.11.

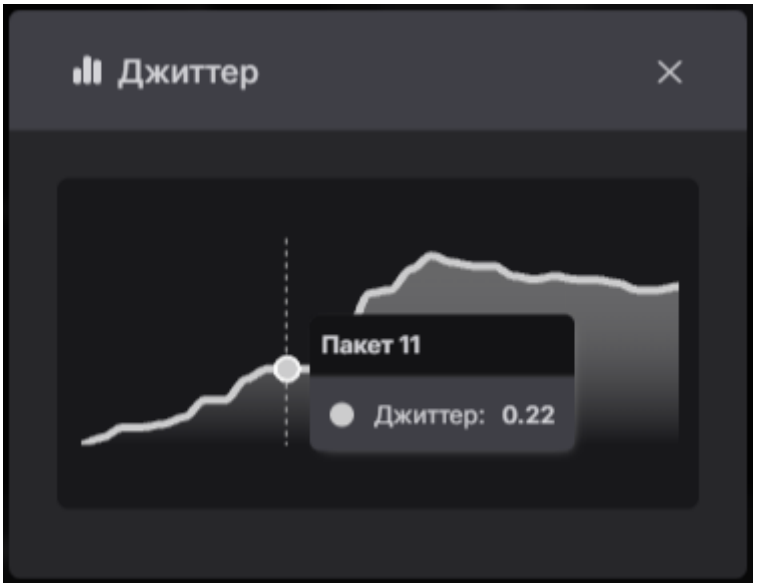
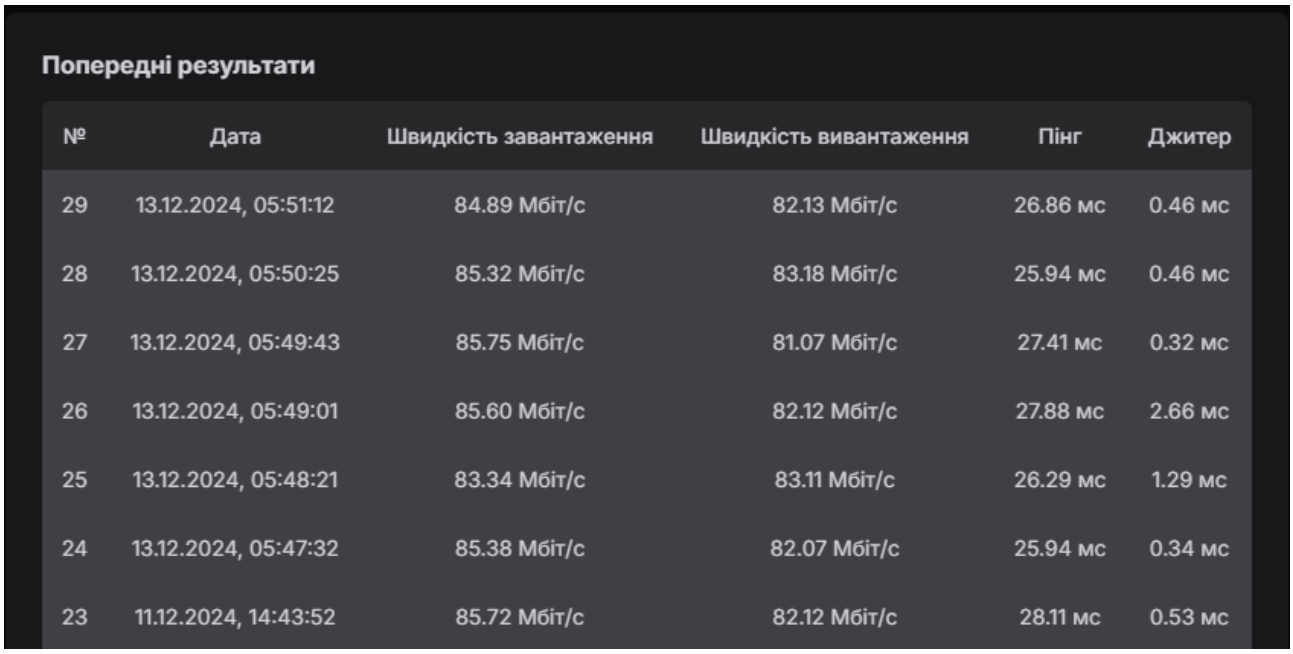


Рисунок 2.11 – Більш детальна інформація про джиттер.

Після завершення виконання, я повторив 7 раз вимірювання, для більш точної інформації, це виводиться та записується в базу даних, знизу на сайті виводяться попередні результати, а саме: номер вимірювання, час, скорість

завантаження та вивантаження, пінг, джиттер, до прикладу в мене результати получились такі як зображені на рис. 2.12.



Попередні результати

№	Дата	Швидкість завантаження	Швидкість вивантаження	Пінг	Джитер
29	13.12.2024, 05:51:12	84.89 Мбіт/с	82.13 Мбіт/с	26.86 мс	0.46 мс
28	13.12.2024, 05:50:25	85.32 Мбіт/с	83.18 Мбіт/с	25.94 мс	0.46 мс
27	13.12.2024, 05:49:43	85.75 Мбіт/с	81.07 Мбіт/с	27.41 мс	0.32 мс
26	13.12.2024, 05:49:01	85.60 Мбіт/с	82.12 Мбіт/с	27.88 мс	2.66 мс
25	13.12.2024, 05:48:21	83.34 Мбіт/с	83.11 Мбіт/с	26.29 мс	1.29 мс
24	13.12.2024, 05:47:32	85.38 Мбіт/с	82.07 Мбіт/с	25.94 мс	0.34 мс
23	11.12.2024, 14:43:52	85.72 Мбіт/с	82.12 Мбіт/с	28.11 мс	0.53 мс

Рисунок 2.12 – Результати виконання програми

Що відображають результати:

Наведені результати семи вимірювань демонструють відносно стабільну роботу мережі з незначними коливаннями показників у часі.

Швидкість завантаження та вивантаження даних тримається в діапазоні приблизно 83–86 Мбіт/с, що свідчить про сталу пропускну здатність каналу.

Пінг, який переважно коливається від 25 до 28 мілісекунд, вказує на помірну затримку, достатньо комфортну для більшості типів онлайн-діяльності (навігація, стрімінг, відеоконференції тощо).

Джиттер у межах від 0.3 до 2.7 мс загалом є невисоким, що означає стабільність затримок між послідовними пакетами й належну якість з’єднання для чутливих до затримок застосунків.

Отже, результати імітаційних вимірювань показують, що мережа працює стабільно, без різких провалів у швидкості чи раптових стрибків затримок. Такий

підсумок свідчить про надійність і прогнозованість характеристик з’єднання, а також про можливість успішного використання мережі в різних сценаріях — від звичайного веб-серфінгу до інтерактивних онлайн-сервісів.

3 МЕТРОЛОГІЧНИЙ АНАЛІЗ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Загальний метрологічний аналіз та оцінка похибок вимірювань

Метрологічний аналіз є важливим етапом у дослідженні мережевих параметрів, адже точність вимірювань безпосередньо впливає на достовірність отриманих результатів. У межах реалізованого програмного забезпечення похибки можуть виникати через кілька факторів: обмеження апаратного забезпечення, затримки в обробці даних на клієнтському та серверному боці, а також мережеві аномалії, що можуть спотворювати результати.

Основні джерела похибок:

- **Апаратні обмеження.** У тестах швидкості завантаження та вивантаження даних обчислення проводяться із залученням таймерів системи. Наприклад, у JavaScript використовується performance.now(), який забезпечує високу точність, але може допускати незначні похибки (на рівні мікросекунд).
- **Мережеві аномалії.** Перевантаження маршрутизаторів, затримки в чергах та втрати пакетів можуть спотворювати вимірювання. Наприклад, при тестуванні пінгу за допомогою WebSocket затримки у маршрутизаторі можуть вплинути на точність результату.
- **Оцифровка сигналів.** Під час вимірювань швидкості та джиттера дані передаються блоками. Це створює можливі розриви між передачею блоків, які можуть впливати на оцінку середнього значення затримок.

3.1.1 Формалізація похибок

Абсолютна похибка (Δ):

Абсолютна похибка визначає різницю між виміряним значенням (X_{meas}) та істинним значенням (X_{true}):

$$\Delta = X_{\text{meas}} - X_{\text{true}}, \tag{3.1}$$

Відносна похибка (δ):

Відносна похибка обчислюється як відношення абсолютної похибки до істинного значення:

$$\delta = \frac{\Delta}{X_{\text{true}}} \times 100\%, \tag{3.2}$$

Середньоквадратична похибка (RMSE):

Для оцінки точності вимірювань у серії тестів використовується середньоквадратична похибка:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(X_{i,\text{meas}} - X_{i,\text{true}}\right)^2}, \tag{3.3}$$

де: n — кількість вимірювань; $X_{i,\text{meas}}$ — виміряне значення iii-го тесту; $X_{i,\text{true}}$ — істинне значення.

3.1.2 Оцінка похибок у реалізованому коді

1. **Тест швидкості завантаження та вивантаження:** У коді для вимірювання швидкості використовується час передавання даних:

$$\text{Speed} = \frac{\text{Bytes Transferred} \times 8}{\text{Time Taken}}, \tag{3.4}$$

   Формула була реалізована через:

   $const\ speedMbps = (totalBytes * 8)/elapsedTime/(1024 * 1024);$

   Основне джерело похибок — невеликі затримки під час обчислення elapsedTime через затримки таймера.

2. **Тест пінгу:** Для пінгу враховуються час відправки та отримання WebSocket-запиту:

$$\text{Ping} = T_{\text{receive}} - T_{\text{send}}, \tag{3.5}$$

   Реалізація в коді:

   $const\ ping = performance.now() - sendTime;$

   Можливі похибки виникають через асинхронність виконання запиту.

3. **Тест джиттера:** Джиттер характеризує нестабільність затримок. Іншими словами, навіть якщо середній пінг невеликий, перепади між

часом доставки послідовних пакетів можуть створювати проблеми для чутливих до затримок додатків (наприклад, відеоконференцій). Для обчислення джиттера зазвичай беруть різницю затримок між послідовними пакетами. Наприклад, нехай T_i — час проходження і-го пакета, тоді:

$$\Delta i = | T_i - T_{i-1} |, \quad (3.6)$$

Джиттером можна вважати середнє цих відхилень для серії пакетів:

$$\text{Jitter} = 1/(\text{n} - 1) * \sum_{i=2}^{n} \Delta_i. \quad (3.7)$$

У коді реалізовано спрощений підхід: після отримання кожного нового значення пінгу ми обчислюємо різницю зі значенням попереднього, а потім коригуємо поточне значення джиттера за допомогою згладжування (експоненційне згладжування з використанням простої формули). Таким чином, ми отримуємо більш плавну криву змін джиттера, яка зменшує вплив поодиноких «стрибків».

Код реалізації:

$$jitterValue += (Math.abs(delta) - jitterValue)/16;$$

Похибки можуть виникати через нерівномірність таймінгів відправлення пакетів.

3.1.3 Методи мінімізації похибок

1. **Повторювані вимірювання:**

   У тестах передбачено кілька циклів вимірювань, наприклад, для пінгу використовується:

   $$const\ PING\_COUNT = 30;$$

   Це дозволяє обчислити середні значення та згладити аномалії.

2. **Збереження результатів:**

   Усі вимірювання зберігаються в базі даних SQLite, що дозволяє аналізувати їх у ретроспективі:

$$db.run(`INSERTINTOresults(\ldots)`, [\ldots]);$$

3. **Аналіз відхилень:**

   Обчислення середньоквадратичної похибки (RMSE) дозволяє визначити точність отриманих даних і скоригувати методи вимірювання.

Розроблений програмний інструмент дозволяє проводити точні вимірювання мережевих параметрів, але, як і будь-яка система, він піддається впливу похибок.

Впровадження методів повторюваних тестів, аналізу даних та коригування алгоритмів дозволяє мінімізувати похибки, забезпечуючи високу надійність отриманих результатів.

3.2 Детальний метрологічний аналіз вимірювань

При вимірюванні мережевих параметрів (пінгу, джиттера, швидкості завантаження та вивантаження даних) важливо не лише отримати числові значення, а й розуміти ступінь їхньої точності. Метрологічний аналіз допомагає визначити основні джерела похибок, оцінити їхній вплив і надати кількісні оцінки невизначеності вимірювань. Це дозволяє більш обґрунтовано інтерпретувати результати та робити висновки щодо якості з’єднання.

Основні джерела похибок:

1. **Похибки вимірювання часу**:

   Наш код використовує performance.now() у браузері та внутрішні таймери Node.js на сервері. Ці функції досить точні, проте вони залежать від продуктивності процесора, системного навантаження та особливостей операційної системи. Нехай похибка відліку часу характеризується стандартною невизначеністю utu_tut.

2. **Похибки, пов’язані з маршрутизацією та середовищем передачі**:

   Затримки в мережі можуть змінюватись через завантаженість каналу,

віддаленість сервера, стани проміжних вузлів. Ця нестабільність безпосередньо впливає на пінг і джиттер. Частково ми оцінюємо її статистично, за результатами кількох вимірювань.

3. **Похибки, пов'язані з обробкою даних**:
   При тестуванні швидкості завантаження та вивантаження ми розраховуємо швидкість, поділяючи обсяг переданих даних на час. Неточності можуть виникати через затримки у читанні потоків, можливої втрати пакетів (хоча вона й не врахована явно), або через вплив кешування, яке, втім, було свідомо мінімізоване.
4. **Похибки дискретизації**:
   Ми вимірюємо пінг, джиттер та швидкість через певні інтервали. Чим менше інтервал вимірювання, тим точніший результат, але тим сильніший вплив випадкових шумів. Для зменшення цих похибок ми використовуємо усереднення результатів кількох послідовних замірів.

Оцінка невизначеності за сучасними методиками:

Сучасний підхід до оцінки похибок вимірювань часто базується на рекомендаціях GUM, який передбачає виокремлення двох типів невизначеностей:

- **Тип А (статистична невизначеність)**: Оцінюється за допомогою статистичної обробки експериментальних даних. Наприклад, для пінгу ми маємо серію результатів $t_1, t_2, \dots t_n$. Середнє значення $\bar{t}$ дає нам приблизну оцінку середньої затримки, а стандартне відхилення s_t свідчить про розкид результатів. Чим менше s_t, тим стабільніший зв'язок.

$$\bar{t} = \frac{1}{n}\sum_{i=1}^{n} t_i, \tag{3.8}$$

$$s_t = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(t_i - \bar{t})^2}, \tag{3.9}$$

  Стандартна невизначеність типу А для пінгу буде приблизно $u_A(t) = \frac{s_t}{\sqrt{n}}$
- **Тип В (апріорна невизначеність)**: Враховує неточності засобів вимірювань та моделі. Наприклад, для performance.now() можна прийняти

апріорну невизначеність, визначену специфікацією браузера чи випробуваннями. Якщо, скажімо, точність таймера становить ±1 мс, то можна записати $u_B(t) \approx 0.5$ мс (при умові рівномірного розподілу похибки).

Для кожного параметра (пінг, джиттер, швидкість) ми можемо визначити повну невизначеність як комбіновану оцінку з урахуванням обох типів невизначеності:

$$u_c = \sqrt{u_A^2 + u_B^2}, \tag{3.10}$$

Приклад розрахунку для пінгу:

Припустимо, що під час тесту ми здійснили $n = 30$ вимірювань пінгу й отримали середнє значення $\bar{t} = 40$ мс та стандартне відхилення $s_t = 5$мс. Тоді статистична невизначеність (тип А) для середнього значення:

А) для середнього значення:

$$u_A(t) = \frac{5 \text{ мс}}{\sqrt{30}} \approx 0.91 \text{ мс}$$

Якщо апріорна похибка таймера становить ±1 мс, тоді $u_B(t) \approx 0.5$ мс.

Комбінована невизначеність:

$$u_c(t) = \sqrt{(0.91)^2 + (0.5)^2} \approx 1.03 \text{ мс}$$

Отже, ми можемо подати виміряний пінг як (40 ± 1.03) мс із довірчим інтервалом приблизно 68% (1 σ).

Розширена невизначеність:

Для більшої впевненості часто вводять коефіцієнт охоплення k (часто беруть $k = 2$ для приблизно 95% довіри). Тоді розширена невизначеність:

$$U = k \times uc(t) = 2 \times 1.03 \approx 2.06 \text{ мс}$$

Таким чином, пінг можна подати як (40 ± 2.06) мс з імовірністю 95%.

Аналогічний підхід для джиттера і швидкості

Для джиттера:

Обчислюємо середнє значення джиттера за результатами серії вимірювань $J_1, J_2, \ldots, J_n$:

$$\bar{J} = \frac{1}{n}\sum_{i=1}^{n} J_i, \tag{3.11}$$

$$s_J = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(J_i - \bar{J})^2}, \tag{3.12}$$

$$u_A(J) = \frac{s_J}{\sqrt{n}}, \tag{3.13}$$

Якщо відомі апріорні похибки (тип B) для джиттера, то комбінована невизначеність:

$$u_c(J) = \sqrt{u_A(J)^2 + u_B(J)^2}, \tag{3.14}$$

$$U(J) = k \times u_c(J), \tag{3.15}$$

Де k — коефіцієнт охоплення (наприклад, $k = 2$ для приблизно 95% довіри).

Для швидкості:

Нехай маємо серію вимірювань швидкості $S_1, S_2, \ldots, S_m$. Аналогічно спочатку обчислюємо середню та стандартне відхилення:

$$\bar{S} = \frac{1}{m}\sum_{j=1}^{m} S_j, \tag{3.16}$$

$$s_S = \sqrt{\frac{1}{m-1}\sum_{j=1}^{m}\left(S_j - \bar{S}\right)^2}, \tag{3.17}$$

$$u_A(S) = \frac{s_S}{\sqrt{m}}, \tag{3.18}$$

У разі наявності апріорної невизначеності типу B для швидкості:

$$u_c(S) = \sqrt{u_A(S)^2 + u_B(S)^2}, \tag{3.19}$$

$$U(S) = k \times u_c(S). \tag{3.20}$$

Таким чином, використовуючи наведені формули, можна послідовно оцінити метрологічні характеристики.

3.3 Рекомендації щодо підвищення точності

Нижче наведено кілька рекомендацій, які допоможуть підвищити точність вимірювань мережевих параметрів:

1. **Збільшення кількості повторних вимірювань**. Проведення серії тестів замість поодиноких вимірювань дозволить усереднити результати та

зменшити вплив випадкових факторів. Так можна отримати стабільніші статистичні оцінки середнього значення та стандартного відхилення.

2. **Використання різних часових інтервалів**. Варто повторювати тести у різний час доби та за різного ступеня завантаженості мережі. Це дозволить краще зрозуміти коливання параметрів і, відповідно, точніше інтерпретувати результати.
3. **Мінімізація сторонніх впливів**. Під час проведення вимірювань бажано закривати непотрібні програми та вкладки у браузері, відключати оновлення та фонові завантаження. Таким чином можна зменшити навантаження на мережу та уникнути перешкод, що спотворюють результати.
4. **Оптимізація налаштувань програмного забезпечення**. Використання асинхронної моделі обробки запитів, відмова від стиснення даних та контроль над кешуванням — усе це знижує ризик появи штучних затримок і дає змогу отримати більш точні та реалістичні дані.
5. **Урахування метрологічних характеристик обладнання та протоколів**. Варто враховувати точність внутрішніх таймерів та особливості протоколів. Якщо можливо, використання більш точних хардверних таймерів чи перевірених бібліотек для заміру часу зменшить випадкові похибки.

Застосування цих рекомендацій у комплексі допоможе суттєво підвищити достовірність результатів та зробити висновки щодо якості та стабільності мережевих з’єднань більш обґрунтованими.

ВИСНОВОК

У ході виконання магістерської роботи було комплексно досліджено вплив мережевих затримок на швидкість передачі даних в Інтернеті та розроблено програмне забезпечення для їх точного вимірювання. В основі лежить поєднання теоретичного аналізу з практичними експериментами, що дозволило не лише глибше зрозуміти природу затримок, але й знайти шляхи їх мінімізації.

Запропонована методика включає використання спеціально розробленого серверного додатка на базі Node.js та клієнтської частини, здатної імітувати реальні умови роботи мережі. Такий підхід дозволив отримати достовірні показники швидкості завантаження/вивантаження даних, пінгу та джиттера у різних сценаріях навантаження. Результати тестувань довели, що стабільність і швидкість мережевого з’єднання суттєво залежать від пропускної здатності, фізичних характеристик каналу, топології маршрутизації, типу протоколів і завантаженості мережі.

Проведений метрологічний аналіз забезпечив розуміння точності отриманих даних, визначення основних джерел похибок і шляхів їх зменшення. Врахування рекомендацій, викладених у роботі, дозволить як провайдерам, так і кінцевим користувачам підвищити якість та надійність інтернет-з’єднання. Зокрема, використання програмного комплексу, розробленого в рамках дослідження, відкриває можливості для оперативного виявлення «вузьких місць», оптимізації роботи мережевого обладнання та раціонального використання мережевих ресурсів.

Таким чином, отримані результати мають суттєве практичне значення. Вони можуть бути впроваджені з метою підвищення ефективності та продуктивності мережі, покращення користувацького досвіду, забезпечення стабільної взаємодії в умовах стрімкого розвитку цифрового суспільства. Ця робота закладає основу для майбутніх досліджень та вдосконалення вимірювальних методик, спрямованих на досягнення більш високої якості

інтернет-послуг та підтримку інноваційних технологій, що стають критично важливими у сучасному світі.

ПЕРЕЛІК ПОСИЛАНЬ НА ДЖЕРЕЛА

1. **Internet World Stats. Usage and Population Statistics** [Електронний ресурс]. – Режим доступу: https://www.internetworldstats.com/ (дата звернення: 13.12.2024).
2. **Cisco Annual Internet Report (2018–2023) White Paper** [Електронний ресурс]. – Cisco Systems. – Режим доступу: https://www.cisco.com/ (дата звернення: 13.12.2024).
3. **3GPP TR 38.801. Study on new radio access technology: Radio access architecture and interfaces** [Електронний ресурс]. – 3GPP, 2017. – Режим доступу: https://www.3gpp.org/ (дата звернення: 13.12.2024).
4. **RFC 2680. A One-way Delay Metric for IPPM** [Електронний ресурс] / G. Almes, S. Kalidindi, M. Zekauskas. – IETF, Sep. 1999. – Режим доступу: https://www.rfc-editor.org/rfc/rfc2680 (дата звернення: 13.12.2024).
5. **RFC 7679. Framework for Performance Metric Development** [Електронний ресурс] / J. Morton, A. Clark, L. Ciavattone. – IETF, Dec. 2015. – Режим доступу: https://www.rfc-editor.org/rfc/rfc7679 (дата звернення: 13.12.2024).
6. **Google Cloud Performance Documentation** [Електронний ресурс]. – Режим доступу: https://cloud.google.com/docs/performance (дата звернення: 13.12.2024).
7. **ISO/IEC 27001:2013.** Information technology – Security techniques – Information security management systems – Requirements. – International Organization for Standardization, Geneva, 2013.
8. **ITU-T Recommendation Y.1541. Network performance objectives for IP-based services** [Електронний ресурс]. – ITU, 2011. – Режим доступу: https://www.itu.int/rec/T-REC-Y.1541 (дата звернення: 13.12.2024).
9. Cisco White Paper: Understanding Delay in Packet Voice Networks [Електронний ресурс]. – Режим доступу:

https://www.cisco.com/c/en/us/support/docs/voice/voice-quality/5125-delay-details.html (дата звернення: 13.12.2024).

10. **IETF QUIC Working Group Documentation** [Електронний ресурс]. – Режим доступу: https://quicwg.org/ (дата звернення: 13.12.2024).
11. **RFC 5357. A Two-Way Active Measurement Protocol (TWAMP)** [Електронний ресурс] / H. Chooqanian, M. Ramalho, J. Perser, R. Geib, R. Cole. – IETF, Oct. 2008. – Режим доступу: https://www.rfc-editor.org/rfc/rfc5357 (дата звернення: 13.12.2024).
12. **iPerf – The ultimate speed test tool for TCP, UDP and SCTP** [Електронний ресурс]. – Режим доступу: https://iperf.fr/ (дата звернення: 13.12.2024).
13. **Microsoft Azure Network Performance Monitoring** [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/en-us/azure/network-watcher/network-performance-monitor-overview (дата звернення: 13.12.2024).
14. **IEEE 802.3 Ethernet Working Group** [Електронний ресурс]. – Режим доступу: https://www.ieee802.org/3/ (дата звернення: 13.12.2024).
15. **RFC 5481. Packet Delay Variation Applicability Statement** [Електронний ресурс] / A. Morton, L. Ciavattone, G. Ramachandran, J. Babiarz. – IETF, March 2009. – Режим доступу: https://www.rfc-editor.org/rfc/rfc5481 (дата звернення: 13.12.2024).
16. **ETSI TR 102 643. Telecommunications and Internet converged Services and Protocols for Advanced Networking (TISPAN)** [Електронний ресурс]. – ETSI. – Режим доступу: https://www.etsi.org/ (дата звернення: 13.12.2024).
17. **MDN Web Docs. Using WebSockets** [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API (дата звернення: 13.12.2024).
18. **Speedtest by Ookla** [Електронний ресурс]. – Режим доступу: https://www.speedtest.net/ (дата звернення: 13.12.2024).

19. **FAST Speed Test by Netflix** [Електронний ресурс]. – Режим доступу: https://fast.com (дата звернення: 13.12.2024).
20. **RFC 6071. SEcure Neighbor Discovery (SEND) problem statement** [Електронний ресурс] / R. Bonica, et al. – IETF, Jan. 2011. – Режим доступу: https://www.rfc-editor.org/rfc/rfc6071 (дата звернення: 13.12.2024).

ДОДАТКИ

ДОДАТОК А

Код серверної частини, який находиться в основній деректорії.

server.js

```
// Увімкнення суворого режиму для підвищення якості коду
'use strict';

// Імпорт необхідних бібліотек
const express = require('express'); // Фреймворк для побудови серверу
const http = require('http'); // Модуль для створення HTTP-сервера
const WebSocket = require('ws'); // Бібліотека для роботи з WebSocket
const cors = require('cors'); // Дозволяє налаштувати CORS
const compression = require('compression'); // Зменшення розміру відповідей сервера
const path = require('path'); // Модуль для роботи з шляхами файлів
const { Readable } = require('stream'); // Для створення потоку даних

// Підключення SQLite для збереження даних
const sqlite3 = require('sqlite3').verbose();

const app = express(); // Створення додатку Express
const server = http.createServer(app); // Створення HTTP-сервера
const wss = new WebSocket.Server({ server }); // Ініціалізація WebSocket-сервера

// Створення або відкриття бази даних SQLite
const db = new sqlite3.Database('test_results.db');

// Створення таблиці, якщо вона не існує
db.serialize(() => {
  db.run(`
    CREATE TABLE IF NOT EXISTS results (
      id INTEGER PRIMARY KEY AUTOINCREMENT,
      date TEXT,
      download_speed REAL,
      upload_speed REAL,
      ping REAL,
      jitter REAL
    )
  `);
});

// Middleware для обробки JSON-запитів
app.use(express.json());

// Налаштування CORS для всіх запитів
app.use(cors());

// Вимкнення стиснення відповідей (для точних тестів)
app.use(compression({ level: 0 }));

// Віддача статичних файлів із папки `public`
app.use(express.static(path.join(__dirname, 'public')));
```

ПРОДОВЖЕННЯ ДОДАТКУ А

server.js

```
// Ендпоінт для завантаження даних (тест швидкості завантаження)
app.get('/download', (req, res) => {
  const size = parseInt(req.query.size) || 100 * 1024 * 1024; // Розмір файлу за замовчуванням: 100
МБ

  // Встановлення заголовків для відповіді
  res.set({
    'Content-Type': 'application/octet-stream',
    'Content-Length': size,
    'Cache-Control': 'no-cache, no-store, must-revalidate',
    'Pragma': 'no-cache',
    'Expires': '0',
    'Connection': 'keep-alive',
    'Content-Encoding': 'identity',
  });

  const chunkSize = 1024 * 1024; // Розмір блоку: 1 МБ
  let bytesSent = 0; // Відправлено байт

  // Створення потоку даних для завантаження
  const readable = new Readable({
    read() {
      if (bytesSent >= size) {
        this.push(null); // Кінець даних
        return;
      }

      const remaining = size - bytesSent; // Залишок даних
      const currentChunkSize = Math.min(chunkSize, remaining); // Поточний розмір блоку



      // Генерація псевдовипадкових даних
      const buffer = Buffer.alloc(currentChunkSize, '0123456789abcdef', 'utf-8');
      this.push(buffer);
      bytesSent += currentChunkSize;
    }
  });

  readable.pipe(res); // Передача даних через потік
});

// Ендпоінт для відвантаження даних (тест швидкості відвантаження)
app.post('/upload', express.raw({ type: '*/*', limit: '1gb' }), (req, res) => {
  // Обробка відповіді після завершення завантаження
  res.set({
    'Cache-Control': 'no-cache, no-store, must-revalidate',
    'Pragma': 'no-cache',
    'Expires': '0',
    'Connection': 'keep-alive',
  });
  res.end();
});
```

ПРОДОВЖЕННЯ ДОДАТКУ А

server.js

```
// Ендпоінт для отримання результатів тестів від клієнта
app.post('/results', (req, res) => {
  const { downloadSpeed, uploadSpeed, ping, jitter } = req.body; // Отримання параметрів тесту

  const date = new Date().toISOString(); // Поточна дата і час

  // Збереження даних у базу
  db.run(
    `INSERT INTO results (date, download_speed, upload_speed, ping, jitter) VALUES (?, ?, ?, ?, ?)`,
    [date, downloadSpeed, uploadSpeed, ping, jitter],
    function (err) {
      if (err) {
        console.error('Помилка запису в базу даних', err);
        res.status(500).send('Помилка збереження результатів');
      } else {
        res.status(200).send('Результати збережено');
      }
    }
  );
});

// Ендпоінт для отримання останніх результатів
app.get('/results', (req, res) => {
  db.all('SELECT * FROM results ORDER BY id DESC LIMIT 10', [], (err, rows) => {
    if (err) {
      console.error('Помилка отримання даних з бази', err);
      res.status(500).send('Помилка отримання результатів');
    } else {
      res.json(rows); // Надсилаємо дані клієнту
    }
  });
});

// WebSocket для тестування затримки та коливань (ping та jitter)
wss.on('connection', (ws) => {
  ws.isAlive = true; // Маркер активності з'єднання

  ws.on('message', (message) => {
    const data = JSON.parse(message); // Обробка повідомлення від клієнта
    if (data.type === 'ping') {
      ws.send(JSON.stringify({ type: 'pong', sendTime: data.sendTime })); // Відповідь із часом
відправлення
    }
  });

  ws.on('pong', () => {
    ws.isAlive = true; // Оновлення активності
  });
});
```

ПРОДОВЖЕННЯ ДОДАТКУ А

server.js

```
// Перевірка неактивних WebSocket-з'єднань
const interval = setInterval(() => {
  wss.clients.forEach((ws) => {
    if (ws.isAlive === false) return ws.terminate(); // Закриваємо неактивні з'єднання
    ws.isAlive = false;
    ws.ping(() => {}); // Перевірка активності
  });
}, 30000);

wss.on('close', () => {
  clearInterval(interval); // Зупиняємо перевірку при закритті сервера
});

// Middleware для обробки помилок
app.use((err, req, res, next) => {
  if (err && err.code === 'ECONNABORTED') {
    console.warn('Запит перервано клієнтом');
  } else if (err && err.message.includes('request aborted')) {
    console.warn('Запит перервано клієнтом');
  } else {
    console.error('Невідома помилка:', err);
    if (!res.headersSent) {
      res.status(500).send('Внутрішня помилка сервера');
    }
  }
});

// Запуск сервера на зазначеному порту
const PORT = process.env.PORT || 3000;

server.listen(PORT, () => {
  console.log(`Сервер запущено на порту ${PORT}`);
});
```

ДОДАТОК Б

Це публічна частина коду, яка знаходиться по шляху **/public/index.html**

index.html

```
<!DOCTYPE html>
<html lang="uk">
  <head>
    <title>Вимірювання швидкості інтернету - Козарук Микола</title>

    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <link rel="stylesheet" href="/css/mercury-dev.css" />
    <script src="https://cdn.tailwindcss.com"></script>
    <script>
      tailwind.config = {
        theme: {
          container: {
            center: true,
            screens: {
              sm: "600px",
              md: "728px",
              lg: "842px",
            },
          },
        },
      };
    </script>
    <link rel="preconnect" href="https://fonts.googleapis.com" />
    <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin />
    <link

href="https://fonts.googleapis.com/css2?family=Inter:ital,opsz,wght@0,14..32,100..900;1,14..32,100..900&display=swap"
      rel="stylesheet"
    />
  </head>
  <body>
    <header class="fixed w-full scroll-fx">
      <div class="container mx-auto flex h-full items-center">
        <a href="/" class="flex items-center">
          <svg
            class="logo"
            viewBox="0 0 100 100"
            xmlns="http://www.w3.org/2000/svg"
          >
            <path
              fill="#EDEDED"
              d="M50 0c-27.614 0-50 22.385-50 49.999 0 27.615 22.386 50.001 50 50.001s50-22.386 50-50.001c0-27.614-
22.386-49.999-50-49.999z"
            />
            <path
              fill="#D5D5D5"
              d="M78.559 27.014l1.427 1.428-37.697 33.551-4.283-4.283 40.553-30.696z"
            />
            <path
              fill="#BEBEBE"
              d="M61 76.999v6h-6v-6h-2v6h-6v-6h-2v6h-6v-6h-2v8h26v-8h-2z"
            />
```

ПРОДОВЖЕННЯ ДОДАТКУ Б

index.html

```
<path
              fill="#EFC75E"
              d="M47.031 8.094c-15.203 1.044-28.29 10.351-34.742 23.434l5.424 2.656c5.767-11.646 17.669-19.813
31.333-20.166.32-.008.636 0 .954 0v-6.014c-.983 0-1.971.021-2.969.09z"
            />
            <path
              fill="#E2574C"
              d="M85.985 49.988c0 5.471-1.437 10.873-3.686 15.61-.689 1.452-.172 3.182 1.235
3.976l.054.03c1.564.882 3.537.234 4.305-1.381 2.63-5.533 4.107-11.843 4.107-18.235 0-7.042-1.743-13.672-4.807-
19.5l-5.326 2.794c2.624 4.993 4.118 10.674 4.118 16.706z"
            />
            <path
              fill="#3DB39E"
              d="M16.407 69.613l.054-.029c1.408-.775 1.924-2.467 1.234-3.887-2.439-5.021-3.886-11.12-3.658-
17.104.209-5.493 1.686-10.681 4.13-15.293l-5.356-2.81c-2.57 4.867-4.211 10.29-4.674 16.029-.619 7.668.863 15.368
3.965 21.744.767 1.578 2.74 2.213 4.305 1.35z"
            />
            <path
              fill="#F4B459"
              d="M50 8.004v6.014c14.174.001 26.436 8.185 32.301 20.095l5.392-2.652c-6.847-13.899-21.151-23.454-
37.693-23.457z"
            />
            <path
              fill="#324D5B"
              d="M76.559 22.014l1.427 1.428-35.697 38.551-4.283-4.283 38.553-35.696z"
            />
            <path
              fill="#1E2D36"
              d="M50 44.999c-2.762 0-5 2.238-5 5s2.238 5 5 5 5-2.238 5-5-2.238-5-5-5z"
            />
          </svg>
          <div class="right">
            <h1 class="no-tailwind">Вимірювання швидкості</h1>
            <h2 class="uppercase no-tailwind">Козарук Микола</h2>
          </div>
        </a>
        <a href="https://t.me/Nlock" class="btn m-l">Телеграм</a>
      </div>
    </header>
    <main class="scroll-fx">
      <div class="container mx-auto">
        <div class="flex block-restrict-main justify-center">
          <div class="lb-block flex items-center justify-center relative">
            <canvas id="foo" width="280" height="250"></canvas>
            <div class="info-speed absolute flex flex-col items-center">
              <div id="speed" class="text-center">0</div>
              <div class="mb-speed flex items-center">
                <svg
                  class="svg-inline download"
                  xmlns="http://www.w3.org/2000/svg"
                  viewBox="0 0 512 512"
                >
```

ПРОДОВЖЕННЯ ДОДАТКУ Б

index.html

```
                  <path
                    fill="currentColor"
                    d="M256 0a256 256 0 1 0 0 512A256 256 0 1 0 256 0zM244.7 395.3l-112-112c-4.6-4.6-5.9-11.5-3.5-17.4s8.3-9.9 14.8-9.9l64 0 0-96c0-17.7 14.3-32 32-32l32 0c17.7 0 32 14.3 32 32l0 96 64 0c6.5 0 12.3 3.9 14.8 9.9s1.1 12.9-3.5 17.4l-112 112c-6.2 6.2-16.4 6.2-22.6 0z"
                    />
                </svg><span class="text-speed">Мбіт/с</span>
              </div>
            </div>
          </div>
          <div class="rb-block">
            <div class="cards grid grid-cols-2 bg-zinc-800 gap-3">
              <div
                class="card flex items-center flex-col"
                type="button"
                data-micromodal-trigger="modal-download"
              >
                <div class="card-header flex items-center">
                  <svg
                    class="svg-inline download"
                    xmlns="http://www.w3.org/2000/svg"
                    viewBox="0 0 512 512"
                  >
                    <path
                      fill="currentColor"
                      d="M256 0a256 256 0 1 0 0 512A256 256 0 1 0 256 0zM244.7 395.3l-112-112c-4.6-4.6-5.9-11.5-3.5-17.4s8.3-9.9 14.8-9.9l64 0 0-96c0-17.7 14.3-32 32-32l32 0c17.7 0 32 14.3 32 32l0 96 64 0c6.5 0 12.3 3.9 14.8 9.9s1.1 12.9-3.5 17.4l-112 112c-6.2 6.2-16.4 6.2-22.6 0z"
                    ></path>
                  </svg>
                  <h3>Завантаження</h3>
                </div>
                <div class="card-body flex flex-col items-center">
                  <div id="download-speed" class="text-center card-speedtest">-</div>
                  <div class="text-dop-card">Мбіт/с</div>
                </div>
              </div>

              <div
                class="card flex items-center flex-col"
                type="button"
                data-micromodal-trigger="modal-upload"
              >
                <div class="card-header flex items-center">
                  <svg
                    class="svg-inline upload"
                    xmlns="http://www.w3.org/2000/svg"
                    viewBox="0 0 512 512"
                  >
                    <path
                      fill="currentColor"
                      d="M256 512A256 256 0 1 0 256 0a256 256 0 1 0 0 512zm11.3-395.3l112 112c4.6 4.6 5.9 11.5 3.5 17.4s-8.3 9.9-14.8 9.9l-64 0 0 96c0 17.7-14.3 32-32 32l-32 0c-17.7 0-32-14.3-32-32l0-96-64 0c-6.5 0-12.3-3.9-14.8-9.9s-1.1-12.9 3.5-17.4l112-112c6.2-6.2 16.4-6.2 22.6 0z"
                      />
                  </svg>
```

ПРОДОВЖЕННЯ ДОДАТКУ Б

index.html

```
                <h3>Вивантаження</h3>
              </div>
              <div class="card-body flex flex-col items-center">
                <div id="upload-speed" class="text-center card-speedtest">-</div>
                <div class="text-dop-card">Мбіт/с</div>
              </div>
            </div>

            <div
              class="card flex items-center flex-col"
              type="button"
              data-micromodal-trigger="modal-ping"
            >
              <div class="card-header flex items-center">
                <svg
                  xmlns="http://www.w3.org/2000/svg"
                  class="svg-inline"
                  viewBox="0 0 512 512"
                >
                  <path
                    fill="currentColor"
                    d="M64 32C28.7 32 0 60.7 0 96l0 64c0 35.3 28.7 64 64 64l384 0c35.3 0 64-28.7 64-64l0-64c0-35.3-28.7-64-64-64L64 32zm280 72a24 24 0 1 1 0 48 24 24 0 1 1 0-48zm48 24a24 24 0 1 1 48 0 24 24 0 1 1 -48 0zM64 288c-35.3 0-64 28.7-64 64l0 64c0 35.3 28.7 64 64 64l384 0c35.3 0 64-28.7 64-64l0-64c0-35.3-28.7-64-64-64L64 288zm280 72a24 24 0 1 1 0 48 24 24 0 1 1 0-48zm56 24a24 24 0 1 1 48 0 24 24 0 1 1 -48 0z"
                  />
                </svg>
                <h3>Пінг</h3>
              </div>
              <div class="card-body flex flex-col items-center">
                <div id="ping-ms" class="text-center card-speedtest">-</div>
                <div class="text-dop-card">мс</div>
              </div>
            </div>

            <div
              class="card flex items-center flex-col"
              type="button"
              data-micromodal-trigger="modal-jitter"
            >
              <div class="card-header flex items-center">
                <svg
                  xmlns="http://www.w3.org/2000/svg"
                  class="svg-inline"
                  viewBox="0 0 448 512"
                >
                  <path
                    fill="currentColor"
                    d="M160 80c0-26.5 21.5-48 48-48l32 0c26.5 0 48 21.5 48 48l0 352c0 26.5-21.5 48-48 48l-32 0c-26.5 0-48-21.5-48-48l0-352zM0 272c0-26.5 21.5-48 48-48l32 0c26.5 0 48 21.5 48 48l0 160c0 26.5-21.5 48-48 48l-32 0c-26.5 0-48-21.5-48-48L0 272zM368 96l32 0c26.5 0 48 21.5 48 48l0 288c0 26.5-21.5 48-48 48l-32 0c-26.5 0-48-21.5-48-48l0-288c0-26.5 21.5-48 48-48z"
                  />
                </svg>
                <h3>Джитер</h3>
              </div>
```

ПРОДОВЖЕННЯ ДОДАТКУ Б

index.html

```
                    <div class="card-body flex flex-col items-center">
            <div id="jitter-info" class="text-center card-speedtest">-</div>
            <div class="text-dop-card">мс</div>
          </div>
        </div>
      </div>
    </div>
  </div>
  <div class="button-start-wrap flex items-center justify-center">
    <button id="start" class="btn variant-primary size-lg">
      Почати тест
    </button>
    <div class="butt-anim ping-main btn variant-primary size-lg absolute inline-flex"></div>
  </div>

  <div class="accordions">
    <button class="accordion w-full noselect">
      Формули по яким все розраховується
    </button>
    <div class="panel w-full">
      <div class="cont-acord">
        <section>
          <h2> -- Пінг (Ping):</h2>
          <p class="sh-p margin-b">
            Пінг вимірюється як середнє значення часу кругового
            проходження пакетів (RTT) між клієнтом і сервером.
          </p>
          <b>1. RTT (Round-Trip Time)_i:</b><br />
          <code class="withbg">RTT_i = receiveTime_i – SendTime_i</code
          ><br />
          Де:
          <ul class="dots">
            <li>
              <code class="bold">RTT_i</code> - час кругового проходження
              для i-го пакета.
            </li>
            <li>
              <code class="bold">sendTime_i</code> - час відправлення i-го
              пакета.
            </li>
            <li>
              <code class="bold">receiveTime_i</code> - час отримання
              відповіді на i-й пакет.
            </li>
          </ul>

          <br />
          <b>2. Середній пінг:</b><br />
          <code class="withbg">Середній пінг = 1 / N * Σ * RTT_i</code
          ><br />
          Де:
          <ul class="dots">
            <li><code class="bold">N</code> - кількість пакетів.</li>
            <li>
              <code class="bold">RTT_i</code> - час кругового проходження
              для i-го пакета.
            </li>
```

ПРОДОВЖЕННЯ ДОДАТКУ Б

index.html

```
            </ul>
          </section>
          <br /><br />

          <section>
            <h2> -- Розрахунок джитеру (Jitter):</h2>
            <p class="sh-p margin-b">
              Джиттер вимірює варіабельність затримки пакетів і обчислюється
              за алгоритмом, описаним у RFC 3550, використовуючи
              експоненціальне згладжування.
            </p>
            <b>Формула для кожного вимірювання після першого:</b><br />
            <b>1. Обчислюємо різницю транзитних затримок:</b><br />
            <code class="withbg">δ_i = RTT_i - RTT_i-1</code
            ><br /><br />

            <b>2. Оновлюємо значення джитеру:</b><br />
            <code class="withbg">jitter = jitter_n + (|δ_i| - jitter_n / 16)</code
            ><br />
            Де:
            <ul class="dots">
              <li>
                <code class="bold">jitter</code> - значення джитеру.
              </li>
              <li>
                <code class="bold">jitter_n</code> - попереднє значення
                джитеру.
              </li>
              <li>
                <code class="bold">|δ_i|</code> - модуль різниці транзитних
                затримок (різниця між поточним і попереднім RTT).
              </li>
            </ul>
          </section>
          <br /><br />

          <section>
            <h2> -- Розрахунок швидкості завантаження даних:</h2>
            <p class="sh-p margin-b">
              Швидкість завантаження визначається як обсяг даних, завантажених за певний період часу.
            </p>

            <b>1. Швидкість у бітах на секунду (bps):</b><br />
            <code class="withbg">Швидкість зав.(bps) = Загальний обсяг зав. даних (у байтах) * 8 / Загальний
час (сек)</code
            ><br /><br />

            <b>2. Конвертація в мегабіти на секунду (Mbps):</b><br />
            <code class="withbg">Швидкість зав. (Mbps) = Швидкість зав. (bps) / 1024 * 1024</code
            ><br />

          </section>
          <br /><br />
          <section>
            <h2> -- Розрахунок швидкості вивантаження даних:</h2>
```

ПРОДОВЖЕННЯ ДОДАТКУ Б

index.html

```
                <p class="sh-p margin-b">
                  Швидкість вивантаження визначається аналогічно швидкості завантаження, але для даних,
відправлених від клієнта до сервера.
                </p>

                <b>1. Швидкість у бітах на секунду (bps):</b><br />
                <code class="withbg">Швидкість вив.(bps) = Загальний обсяг вив. даних (у байтах) * 8 / Загальний
час (сек)</code
                ><br /><br />

                <b>2. Конвертація в мегабіти на секунду (Mbps):</b><br />
                <code class="withbg">Швидкість вив. (Mbps) = Швидкість вив. (bps) / 1024 * 1024</code
                ><br />

              </section>
              <br /><br />
              <section>
                <h2 class="margin-b"> -- Додаткові пояснення:</h2>
                <ul class="dots">
                  <li>
                    <code class="bold withbg">Загальний обсяг завантажених/вивантажених даних</code> - сума всіх
байтів, отриманих або відправлених за час тесту.
                  </li>
                  <li>
                    <code class="bold withbg">Загальний час</code> - тривалість тесту, виміряна з моменту його
початку до завершення.
                  </li>
                  <li>
                    <code class="bold withbg">Швидкість завантаження</code> - швидкість, з якою дані завантажуються
з сервера на клієнт.
                  </li>
                  <li>
                    <code class="bold withbg">Швидкість вивантаження</code> - швидкість, з якою дані відправляються
з клієнта на сервер.
                  </li>
                  <li>
                    <code class="bold withbg">Пінг</code> - час, який потрібно пакету для проходження від клієнта
до сервера і назад.
                  </li>
                  <li>
                    <code class="bold withbg">Джитер</code> - варіабельність затримки пакетів.
                  </li>
                  <li>
                    <code class="bold withbg">x8</code> - перетворення байтів у біти (1 байт = 8 біт).
                  </li>
                  <li>
                    <code class="bold withbg">/ 1024 * 1024</code> - конвертація бітів у мегабіти (1 Мбіт = 1024 *
1024 біт).
                  </li>
                </ul>

              </section>

            </div>
          </div>
        </div>
```

ПРОДОВЖЕННЯ ДОДАТКУ Б

index.html

```
      <div class="results-from-db">
        <h2>Попередні результати</h2>
        <div class="table-wrap">
        <table class="table-auto w-full">
          <thead>
            <tr>
              <th>№</th>
              <th>Дата</th>
              <th>Швидкість завантаження</th>
              <th>Швидкість вивантаження</th>
              <th>Пінг</th>
              <th>Джитер</th>
            </tr>
          </thead>
          <tbody id="results">
            <tr>
              <td>2</td>
              <td>24.11.2024, 01:52:44</td>
              <td>100 Мбіт/с</td>
              <td>100 Мбіт/с</td>
              <td>100 мс</td>
              <td>100 мс</td>
            </tr>
            <tr>
              <td>1</td>
              <td>24.11.2024, 01:52:43</td>
              <td>100 Мбіт/с</td>
              <td>100 Мбіт/с</td>
              <td>100 мс</td>
              <td>100 мс</td>
            </tr>
          </tbody>
        </table>
      </div>
      </div>
    </div>
  </main>

  <footer class="scroll-fx">
    <div class="container mx-auto h-full items-center flex justify-center">
      <p>© 2024 Козарук Микола, МТТм-23-1</p>
    </div>
  </footer>

  <div
    class="modal micromodal-slide"
    id="modal-download"
    aria-hidden="true"
    aria-modal="true"
  >
    <div class="modal__overlay" tabindex="-1" data-micromodal-close>
      <div
        class="modal__container"
        role="dialog"
        aria-modal="true"
        aria-labelledby="modal-1-title"
      >
```

ПРОДОВЖЕННЯ ДОДАТКУ Б

index.html

```
<div class="modal__header noselect">
            <div class="modal-text-header flex items-center">
              <svg
                class="svg-inline"
                xmlns="http://www.w3.org/2000/svg"
                viewBox="0 0 512 512"
              >
                <path
                  fill="currentColor"
                  d="M256 0a256 256 0 1 0 0 512A256 256 0 1 0 256 0zM244.7 395.3l-112-112c-4.6-4.6-5.9-11.5-3.5-17.4s8.3-9.9 14.8-9.9l64 0 0-96c0-17.7 14.3-32 32-32l32 0c17.7 0 32 14.3 32 32l0 96 64 0c6.5 0 12.3 3.9 14.8 9.9s1.1 12.9-3.5 17.4l-112 112c-6.2 6.2-16.4 6.2-22.6 0z"
                ></path>
              </svg>
              <h3>Завантаження</h3>
            </div>

            <button
              class="modal__close"
              type="button"
              aria-label="Close modal"
              data-micromodal-close
            ></button>
          </div>

          <div class="modal__content" id="modal-download-content">
            <div class="warninfo">
              Запустіть тест щоб побачити результат :)
            </div>
          </div>
        </div>
      </div>
    </div>


    <div
      class="modal micromodal-slide"
      id="modal-upload"
      aria-hidden="true"
      aria-modal="true"
    >
      <div class="modal__overlay" tabindex="-1" data-micromodal-close>
        <div
          class="modal__container"
          role="dialog"
          aria-modal="true"
          aria-labelledby="modal-1-title"
        >
          <div class="modal__header noselect">
            <div class="modal-text-header flex items-center">
              <svg
                class="svg-inline"
                xmlns="http://www.w3.org/2000/svg"
                viewBox="0 0 512 512"
              >
```

ПРОДОВЖЕННЯ ДОДАТКУ Б

index.html

```
<path
                  fill="currentColor"
                  d="M256 512A256 256 0 1 0 256 0a256 256 0 1 0 0 512zm11.3-395.3l112 112c4.6 4.6 5.9 11.5 3.5
17.4s-8.3 9.9-14.8 9.9l-64 0 0 96c0 17.7-14.3 32-32 32l-32 0c-17.7 0-32-14.3-32-32l0-96-64 0c-6.5 0-12.3-3.9-14.8-
9.9s-1.1-12.9 3.5-17.4l112-112c6.2-6.2 16.4-6.2 22.6 0z"
                ></path>
              </svg>
              <h3>Вивантаження</h3>
            </div>

            <button
              class="modal__close"
              type="button"
              aria-label="Close modal"
              data-micromodal-close
            ></button>
          </div>

          <div class="modal__content" id="modal-upload-content">
            <div class="warninfo">
              Запустіть тест щоб побачити результат :)
            </div>
          </div>
        </div>
      </div>
    </div>

    <div
      class="modal micromodal-slide"
      id="modal-ping"
      aria-hidden="true"
      aria-modal="true"
    >
      <div class="modal__overlay" tabindex="-1" data-micromodal-close>
        <div
          class="modal__container"
          role="dialog"
          aria-modal="true"
          aria-labelledby="modal-1-title"
        >
          <div class="modal__header noselect">
            <div class="modal-text-header flex items-center">
              <svg
                xmlns="http://www.w3.org/2000/svg"
                class="svg-inline"
                viewBox="0 0 512 512"
              >
                <path
                  fill="currentColor"
                  d="M64 32C28.7 32 0 60.7 0 96l0 64c0 35.3 28.7 64 64 64l384 0c35.3 0 64-28.7 64-64l0-64c0-35.3-
28.7-64-64-64L64 32zm280 72a24 24 0 1 1 0 48 24 24 0 1 1 0-48zm48 24a24 24 0 1 1 48 0 24 24 0 1 1 -48 0zM64 288c-
35.3 0-64 28.7-64 64l0 64c0 35.3 28.7 64 64 64l384 0c35.3 0 64-28.7 64-64l0-64c0-35.3-28.7-64-64-64L64 288zm280
72a24 24 0 1 1 0 48 24 24 0 1 1 0-48zm56 24a24 24 0 1 1 48 0 24 24 0 1 1 -48 0z"
                ></path>
              </svg>
              <h3>Пінг</h3>
            </div>
```

ПРОДОВЖЕННЯ ДОДАТКУ Б

index.html

```
            <button
              class="modal__close"
              type="button"
              aria-label="Close modal"
              data-micromodal-close
            ></button>
          </div>

          <div class="modal__content" id="modal-ping-content">
            <div class="warninfo">
              Запустіть тест щоб побачити результат :)
            </div>
          </div>
        </div>
      </div>
    </div>

    <div
      class="modal micromodal-slide"
      id="modal-jitter"
      aria-hidden="true"
      aria-modal="true"
    >
      <div class="modal__overlay" tabindex="-1" data-micromodal-close>
        <div
          class="modal__container"
          role="dialog"
          aria-modal="true"
          aria-labelledby="modal-1-title"
        >
          <div class="modal__header noselect">
            <div class="modal-text-header flex items-center">
              <svg
                xmlns="http://www.w3.org/2000/svg"
                class="svg-inline"
                viewBox="0 0 448 512"
              >
                <path
                  fill="currentColor"
                  d="M160 80c0-26.5 21.5-48 48-48l32 0c26.5 0 48 21.5 48 48l0 352c0 26.5-21.5 48-48 48l-32 0c-26.5 0-48-21.5-48-48l0-352zM0 272c0-26.5 21.5-48 48-48l32 0c26.5 0 48 21.5 48 48l0 160c0 26.5-21.5 48-48 48l-32 0c-26.5 0-48-21.5-48-48L0 272zM368 96l32 0c26.5 0 48 21.5 48 48l0 288c0 26.5-21.5 48-48 48l-32 0c-26.5 0-48-21.5-48-48l0-288c0-26.5 21.5-48 48-48z"
                ></path>
              </svg>
              <h3>Джитер</h3>
            </div>

            <button
              class="modal__close"
              type="button"
              aria-label="Close modal"
              data-micromodal-close
            ></button>
          </div>
```

ПРОДОВЖЕННЯ ДОДАТКУ Б

index.html

```
<div class="modal__content" id="modal-jitter-content">
            <div class="warninfo">
              Запустіть тест щоб побачити результат :)
            </div>
          </div>
        </div>
      </div>
    </div>

    <script src="https://bernii.github.io/gauge.js/dist/gauge.min.js"></script>
    <script src="https://unpkg.com/micromodal/dist/micromodal.min.js"></script>
    <script src="https://cdn.jsdelivr.net/npm/apexcharts"></script>


    <script src="script.js" type="module"></script>
  </body>
</html>
```

ДОДАТОК В

Основний клієнтський скрипт script.js, він знаходиться по шляху /public/script.js

script.js

```
"use strict";

// Тривалість тестів (у секундах)
const DOWNLOAD_DURATION=15,UPLOAD_DURATION=15,PING_COUNT=30;

MicroModal.init();const
WS_URL=`${"https:"===window.location.protocol?"wss":"ws"}://${window.location.host}`
,startButton=document.getElementById("start"),downloadSpeedElem=document.getElementById("download
-
speed"),speedInfoElem=document.getElementById("speed"),uploadSpeedElem=document.getElementById("u
pload-speed"),pingElem=document.getElementById("ping-
ms"),jitterElem=document.getElementById("jitter-info"),mbSpeedElem=document.querySelector(".mb-
speed"),modalDownloadContent=document.getElementById("modal-download-
content"),modalUploadContent=document.getElementById("modal-upload-
content"),modalPingContent=document.getElementById("modal-ping-
content"),modalJitterContent=document.getElementById("modal-jitter-
content"),downloadCard=document.querySelector('[data-micromodal-trigger="modal-
download"]'),uploadCard=document.querySelector('[data-micromodal-trigger="modal-
upload"]'),pingCard=document.querySelector('[data-micromodal-trigger="modal-
ping"]'),jitterCard=document.querySelector('[data-micromodal-trigger="modal-
jitter"]'),resultsTableBody=document.getElementById("results");let isTesting=!1;const
downloadSVG=`
<svg class="svg-inline download" xmlns="http://www.w3.org/2000/svg" viewBox="0 0 512 512">
  <path fill="currentColor" d="M256 0a256 256 0 1 0 0 512A256 256 0 1 0 256 0zM244.7 395.3l-112-
112c-4.6-4.6-5.9-11.5-3.5-17.4s8.3-9.9 14.8-9.9l64 0 0-96c0-17.7 14.3-32 32-32l32 0c17.7 0 32
14.3 32 32l0 96 64 0c6.5 0 12.3 3.9 14.8 9.9s1.1 12.9-3.5 17.4l-112 112c-6.2 6.2-16.4 6.2-22.6
0z"></path>
</svg>`,uploadSVG=`
<svg class="svg-inline upload" xmlns="http://www.w3.org/2000/svg" viewBox="0 0 512 512">
  <path fill="currentColor" d="M256 512A256 256 0 1 0 256 0a256 256 0 1 0 0 512zm11.3-395.3l112
112c4.6 4.6 5.9 11.5 3.5 17.4s-8.3 9.9-14.8 9.9l-64 0 0 96c0 17.7-14.3 32-32 32l-32 0c-17.7 0-32-
14.3-32-32l0-96-64 0c-6.5 0-12.3-3.9-14.8-9.9s-1.1-12.9 3.5-17.4l112-112c6.2-6.2 16.4-6.2 22.6
0z"></path>
</svg>`;function resetUI(){downloadSpeedElem.textContent="-
",speedInfoElem.textContent="0",uploadSpeedElem.textContent="-",pingElem.textContent="-
",jitterElem.textContent="-
",gauge.set(0),resetModalContents(),[downloadCard,uploadCard,pingCard,jitterCard].forEach(e=>{e.c
lassList.remove("active-card"),e.classList.remove("finished")})}function
testStartedContents(){let e='<div class="warninfo">Тест запущено, зачекайте
результата</div>';modalDownloadContent.innerHTML=e,modalUploadContent.innerHTML=e,modalPingConten
t.innerHTML=e,modalJitterContent.innerHTML=e}function resetModalContents(){let e='<div
class="warninfo">Запустіть тест щоб побачити
результати</div>';modalDownloadContent.innerHTML=e,modalUploadContent.innerHTML=e,modalPingConten
t.innerHTML=e,modalJitterContent.innerHTML=e}function
setActiveCard(e,t=null){[downloadCard,uploadCard,pingCard,jitterCard].forEach(e=>e.classList.remo
ve("active-card")),e&&(e.classList.add("active-card"),t&&t.classList.add("active-card"))}function
setupGaugeForTest(e){gauge.set(0),"download"===e?(gauge.options.colorStart="#7ec96c",gauge.option
s.colorStop="#ade57b",mbSpeedElem.innerHTML=downloadSVG+'<span class="text-
speed">Мбіт/с</span>'):"upload"===e&&(gauge.options.colorStart="#88d7df",gauge.options.colorStop=
"#96acef",mbSpeedElem.innerHTML=uploadSVG+'<span class="text-speed">Мбіт/с</span>')}async
function
startTest(){resetUI(),testStartedContents();try{setActiveCard(pingCard,jitterCard);let{pingTimes:
e,jitterValues:t}=await startPingTest();createPingChart(e),createJitterChart(t);let
a=average(e),n=t[t.length-1]||0;setActiveCard(downloadCard),setupGaugeForTest("download");let
o=await startDownloadTest();createDownloadChart(o),gauge.set(0),speedInfoElem.textContent="0";let
r=o[o.length-1]?.speed||0;await
```

ПРОДОВЖЕННЯ ДОДАТКУ В

script.js

```
pause(2e3),setActiveCard(uploadCard),setupGaugeForTest("upload");let d=await
startUploadTest();createUploadChart(d);let l=d[d.length-
1]?.speed||0;setActiveCard(null),[downloadCard,uploadCard,pingCard,jitterCard].forEach(e=>e.class
List.add("finished")),gauge.set(0),speedInfoElem.textContent="0",await
sendResultsToServer({downloadSpeed:r,uploadSpeed:l,ping:a,jitter:n}),await
fetchAndDisplayResults()}catch(i){console.error("Ошибка во время
теста:",i)}finally{isTesting=!1,startButton.disabled=!1}}function pause(e){return new
Promise(t=>setTimeout(t,e))}async function startPingTest(){return new Promise((e,t)=>{let a=new
WebSocket(WS_URL),n=[],o=[],r=0,d=null,l;a.onopen=()=>{let
t=0,i=()=>{if(t>=30){clearInterval(l),a.close(),e({pingTimes:n,jitterValues:o});return}let
r=performance.now();a.send(JSON.stringify({type:"ping",sendTime:r}))};a.onmessage=e=>{let
a=JSON.parse(e.data);if("pong"===a.type){let l=performance.now(),i=a.sendTime,s=l-i;n.push(s);let
c=s;if(null!==d){let p=c-d;r+=(Math.abs(p)-
r)/16}d=c,o.push(r),updatePingDisplay(n,r),t++}},l=setInterval(i,200),i()},a.onerror=e=>{console.
error("WebSocket error:",e),clearInterval(l),a.close(),t(e)}})}function
updatePingDisplay(e,t){let
a=average(e);pingElem.textContent=a.toFixed(2),jitterElem.textContent=t.toFixed(2)}function
average(e){return e.reduce((e,t)=>e+t,0)/e.length}async function startDownloadTest(){let
e=0,t=new
AbortController,a=t.signal,n=[],o=Array.from({length:4},()=>downloadStream(a,t=>{e+=t})),r=perfor
mance.now(),d=setInterval(()=>{let t=(performance.now()-
r)/1e3,a=8*e/t,o=a/1048576;downloadSpeedElem.textContent=o.toFixed(2),speedInfoElem.textContent=o
.toFixed(2),gauge.set(o),n.push({time:t,speed:o})},500);await pause(15e3),t.abort(),await
Promise.allSettled(o),clearInterval(d);let l=(performance.now()-r)/1e3,i=8*e/l,s=i/1048576;return
downloadSpeedElem.textContent=s.toFixed(2),speedInfoElem.textContent=s.toFixed(2),gauge.set(s),n}
function downloadStream(e,t){return new Promise(async(a,n)=>{let
o=`/download?size=10000000&r=${Math.random()}`;try{for(;!e.aborted;){let r=await
fetch(o,{cache:"no-
store",signal:e}),d=r.body.getReader(),l=!1;for(;!l&&!e.aborted;){let{value:i,done:s}=await
d.read();i&&t(i.length),l=s}d.releaseLock()}a()}catch(c){e.aborted?a():(console.error("Ошибка
загрузки:",c),n(c))}})}async function startUploadTest(){let e=0,t=new
AbortController,a=t.signal,n=[],o=generateRandomData(2097152),r=Array.from({length:4},()=>uploadS
tream(o,a,t=>{e+=t})),d=performance.now(),l=setInterval(()=>{let t=(performance.now()-
d)/1e3,a=8*e/t,o=a/1048576;uploadSpeedElem.textContent=o.toFixed(2),speedInfoElem.textContent=o.t
oFixed(2),gauge.set(o),n.push({time:t,speed:o})},500);await pause(15e3),t.abort(),await
Promise.allSettled(r),clearInterval(l);let i=(performance.now()-d)/1e3,s=8*e/i,c=s/1048576;return
uploadSpeedElem.textContent=c.toFixed(2),speedInfoElem.textContent=c.toFixed(2),gauge.set(c),n}fu
nction uploadStream(e,t,a){return new Promise(async(n,o)=>{let
r=`/upload?r=${Math.random()}`;try{for(;!t.aborted;){let d=await
fetch(r,{method:"POST",body:e,signal:t,cache:"no-store",headers:{"Content-
Type":"application/octet-stream"}});if(!d.ok)throw Error("Network response was not
ok");a(e.size)}n()}catch(l){t.aborted?n():(console.error("Ошибка выгрузки:",l),o(l))}})}function
generateRandomData(e){let t=[],a=0;for(;a<e;){let n=Math.min(65536,e-a),o=new
Uint8Array(n);window.crypto.getRandomValues(o),t.push(o),a+=n}return new
Blob(t)}startButton.addEventListener("click",()=>{isTesting||(isTesting=!0,startButton.disabled=!
0,startTest())});var opts={angle:-
.21,lineWidth:.14,radiusScale:1,pointer:{length:.54,strokeWidth:.038,color:"#d4d4d8"},colorStart:
"#7ec96c",colorStop:"#ade57b",strokeColor:"#2a2627",generateGradient:!0,highDpiSupport:!0,renderT
icks:{divisions:10,divWidth:1.9,divLength:.23,divColor:"#d4d4d8",subDivisions:3,subLength:.5,subW
idth:.6,subColor:"#3a3a3f"}},target=document.getElementById("foo"),gauge=new
Gauge(target).setOptions(opts);gauge.maxValue=200,gauge.setMinValue(0),gauge.animationSpeed=107,
```

ПРОДОВЖЕННЯ ДОДАТКУ В

script.js

```
gauge.set(0);const
createChartOptions=(e,t,a,n,o)=>({chart:{type:e,height:165,width:320,toolbar:{show:!1}},fill:{type:"gradient",colors:t,gradient:{shade:"dark",type:"vertical",shadeIntensity:.5,inverseColors:!0,opacityFrom:1,opacityTo:0,stops:[0,100]}},grid:{show:!1},series:[{name:a,data:n}],xaxis:{categories:o||[],labels:{show:!1},axisBorder:{show:!1},axisTicks:{show:!1},tooltip:{enabled:!1}},yaxis:{show:!1},tooltip:{style:{fontSize:"12px",fontFamily:"inherit"},theme:"dark"},dataLabels:{enabled:!1}}),renderChart=(e,t)=>{new ApexCharts(document.querySelector(e),t).render()};function
createPingChart(e){let t=e.map((e,t)=>`Пакет
${t+1}`),a=e.map(e=>parseFloat(e.toFixed(2))),n=createChartOptions("area",["#cccccc"],"Пінг",a,t)
;modalPingContent.innerHTML='<div id="chart_ping"
class="chart"></div>',renderChart("#chart_ping",n)}function createJitterChart(e){let
t=e.map((e,t)=>`Пакет
${t+1}`),a=e.map(e=>parseFloat(e.toFixed(2))),n=createChartOptions("area",["#cccccd"],"Джиттер",a
,t);modalJitterContent.innerHTML='<div id="chart_jitter"
class="chart"></div>',renderChart("#chart_jitter",n)}function createDownloadChart(e){let
t=e.map(e=>e.time.toFixed(2)+"
c"),a=e.map(e=>parseFloat(e.speed.toFixed(2))),n=createChartOptions("area",["#7ec96c"],"Швидкість
завантаження",a,t);modalDownloadContent.innerHTML='<div id="chart_download"
class="chart"></div>',renderChart("#chart_download",n)}function createUploadChart(e){let
t=e.map(e=>e.time.toFixed(2)+"
c"),a=e.map(e=>parseFloat(e.speed.toFixed(2))),n=createChartOptions("area",["#88d7df"],"Швидкість
вивантаження",a,t);modalUploadContent.innerHTML='<div id="chart_upload"
class="chart"></div>',renderChart("#chart_upload",n)}async function
sendResultsToServer(e){try{let t=await fetch("/results",{method:"POST",headers:{"Content-
Type":"application/json;charset=utf-8"},body:JSON.stringify(e)});if(!t.ok)throw Error("Failed to
send results to server")}catch(a){console.error("Error sending results to server:",a)}}async
function fetchAndDisplayResults(){try{let e=await fetch("/results");if(!e.ok)throw Error("Failed
to fetch results from server");let t=await
e.json();resultsTableBody.innerHTML="",t.forEach(e=>{let
t=document.createElement("tr"),a=document.createElement("td");a.textContent=e.id;let
n=document.createElement("td"),o=new Date(e.date);n.textContent=o.toLocaleString();let
r=document.createElement("td");r.textContent=e.download_speed.toFixed(2)+" Мбіт/с";let
d=document.createElement("td");d.textContent=e.upload_speed.toFixed(2)+" Мбіт/с";let
l=document.createElement("td");l.textContent=e.ping.toFixed(2)+" мс";let
i=document.createElement("td");i.textContent=e.jitter.toFixed(2)+"
мс",t.appendChild(a),t.appendChild(n),t.appendChild(r),t.appendChild(d),t.appendChild(l),t.append
Child(i),resultsTableBody.appendChild(t)})}catch(a){console.error("Error fetching and displaying
results:",a)}}fetchAndDisplayResults();var
acc=document.getElementsByClassName("accordion");Array.from(acc).forEach(e=>e.addEventListener("c
lick",function(){this.classList.toggle("active");var
e=this.nextElementSibling;e.style.maxHeight=e.style.maxHeight?null:e.scrollHeight+"px"}));
```

ДОДАТОК Г

Публічна частина, таблиця стилів яка знаходиться в проекті по шляху /public/css/**mercury_dev.css**

mercury_dev.css

```
.card-header,.mb-speed{font-weight:500}.btn,html{-webkit-tap-highlight-
color:transparent}.btn:before,.micromodal-slide.is-
open,article,aside,details,figcaption,figure,footer,header,hgroup,menu,nav,section{display:block}
.btn:before,.noevents{pointer-events:none}.btn,.btn:not(.is-link):hover{text-
decoration:none}.btn._bold span,header h1{font-
weight:600}.accordion,.btn,.modal__btn,[type=button]:not(:disabled),[type=reset]:not(:disabled),[
type=submit]:not(:disabled),button:not(:disabled){cursor:pointer}.micromodal-slide
.modal__container,.micromodal-slide .modal__overlay,.modal__btn{will-change:transform}.card-
header,ul.dots li{margin-bottom:1rem}:root{--color-first:#09090b;--color-secondary:#18181b;--
color-third:#27272a;--color-fourth:#3f3f46;--color-fifth:#52525b;--color-sixth:#71717a;--color-
seventh:#a1a1aa;--color-eighth:#d4d4d8;--color-ninth:#e4e4e7;--color-tenth:#f4f4f5;--color-
eleventh:#fafafa;--border-radius:6px;--border-radius-md:9px;--color-dw:#7ec96c;--color-
up:#88d7df;--btn-primary:#ffe261;--btn-primary-filled:#ffe261;--color-footer:#060606;--reader-
font-family:-apple-system,BlinkMacSystemFont,"Inter",Roboto,Helvetica Neue,Helvetica,sans-serif;-
-height-header:7rem;--height-footer:5.2rem;--selection-bg:#1c1a12;--selection-text:#ffe261;--
scrollbar-bg:#18181b;--scrollbar-thumb:#52525b;--scrollbar-thumb-
hover:#71717a}a,abbr,acronym,address,applet,article,aside,audio,b,big,blockquote,body,canvas,capt
ion,center,cite,code,dd,del,details,dfn,div,dl,dt,em,embed,fieldset,figcaption,figure,footer,form
,h1,h2,h3,h4,h5,h6,header,hgroup,html,i,iframe,img,ins,kbd,label,legend,li,mark,menu,nav,object,o
l,output,p,pre,q,ruby,s,samp,section,small,span,strike,strong,sub,summary,sup,table,tbody,td,tfoo
t,th,thead,time,tr,tt,u,ul,var,video{margin:0;padding:0;border:0;font-
size:100%;font:inherit;vertical-align:baseline}body{color:var(--color-eighth);font-
size:1.4rem;line-height:180%;font-family:var(--reader-font-family);background-color:var(--color-
first);background-repeat:no-repeat;margin:0;padding:0}html{font-size:62.5%;backface-
visibility:hidden;text-size-adjust:100%;-webkit-font-smoothing:subpixel-
antialiased}::selection{background-color:var(--selection-bg);color:var(--selection-text)}::-
webkit-scrollbar{background-color:var(--scrollbar-bg);width:14px;height:16px;z-index:999999}::-
webkit-scrollbar-track{background-color:transparent}::-webkit-scrollbar-thumb{background-
color:var(--scrollbar-thumb);border-radius:16px;border:solid 3px var(--scrollbar-
bg)}.card,.results-from-db table{background:var(--color-fourth)}::-webkit-scrollbar-
button{display:none}::-webkit-scrollbar-thumb:hover{background-color:var(--scrollbar-thumb-
hover)}button{outline:0}body,html,main{height:100%}ol,ul{list-
style:none}blockquote,q{quotes:none}blockquote:after,blockquote:before,q:after,q:before{content:"
";content:none}table{border-collapse:collapse;border-spacing:0}.scroll-fx{padding-left:calc(-100%
+ 100vw)}.card,.cards{padding:2.4rem;border-radius:var(--border-radius)}main.scroll-
fx{height:auto;padding-top:var(--height-header);min-height:calc(100% - var(--height-
footer))}header{top:0;left:0;background:var(--color-secondary);height:var(--height-header);z-
index:100}header h1{font-size:1.6rem;line-height:1.5;color:var(--color-tenth)}header h2{font-
size:1.2rem}.logo{height:4.2rem;margin-right:.9rem}footer{height:var(--height-
footer);background:var(--color-footer)}.m-l{margin-left:auto}.info-speed{bottom:0}#speed{font-
size:2.8rem;font-weight:700;line-height:1.4}.text-speed{margin-
left:.4rem;opacity:.7}.download{color:var(--color-dw)}.upload{color:var(--color-
up)}.card{transition:opacity .3s}.card-header svg{margin-right:.5rem}.accordion:after,.card-
body,.cont-acord h2,body .apexcharts-tooltip-title,code.bold{font-weight:700}.card-body{font-
size:2.4rem}.rb-block{margin-left:4.8rem}.block-restrict-main{margin-top:6.8rem;margin-
bottom:2.4rem}.card,.noselect{-webkit-touch-callout:none;-webkit-user-select:none;-khtml-user-
select:none;-moz-user-select:none;-ms-user-select:none;user-select:none}.text-dop-card{font-
size:1rem;opacity:.8;font-weight:400;line-height:1}.accordions{margin-top:6.8rem}
```

ПРОДОВЖЕННЯ ДОДАТКУ Г

mercury_dev.css

```
.results-from-db{padding:2.2rem 2.4rem;background-color:var(--color-secondary);margin-top:2.4rem;border-radius:var(--border-radius);margin-bottom:2.4rem}.accordion,.panel{background-color:var(--color-third)}.results-from-db h2{font-weight:700;font-size:1.6rem;margin-bottom:1.2rem}.results-from-db table{border-radius:var(--border-radius)}.results-from-db tr td,.results-from-db tr th{padding:1.2rem;text-align:center}.results-from-db tr th{background:var(--color-third);font-weight:500}.results-from-db tr th:first-child{border-radius:var(--border-radius) 0 0 0}.results-from-db tr th:last-child{border-radius:0 var(--border-radius) 0 0}.card{opacity:.5}.btn.is-filled:before,.card.active-card,.card.finished{opacity:1}.table-wrap{overflow-x:scroll;border-radius:var(--border-radius)}.button-start-wrap .butt-anim{z-index:-2;min-width:10.6rem}.button-start-wrap button#start{position:relative;z-index:5}.ping-main:not(:disabled){animation:1s cubic-bezier(0,0,.2,1) infinite ping-main}.btn.is-link:before,.btn.variant-input:before,.micromodal-slide,button#start:disabled~.butt-anim{display:none}.accordion{padding:1.4rem 2.4rem;text-align:left;outline:0;font-size:1.6rem;transition:.3s;border-radius:var(--border-radius)}.accordion.active,.accordion:hover{background-color:var(--color-fourth)}.accordion:after{content:"\002B";font-size:1.6rem;float:right;margin-left:.6rem}.active:after{content:"\2212"}.panel{max-height:0;overflow:hidden;transition:max-height .4s cubic-bezier(.6, 0, 0, 1);border-radius:0 0 var(--border-radius) var(--border-radius)}.accordion.active{border-radius:var(--border-radius) var(--border-radius) 0 0}.cont-acord{padding:1.6rem}.svg-inline{display:inline-block;height:1em;vertical-align:-.125em;overflow:visible}.btn span,.noscroll{overflow:hidden}.btn-group{display:flex;align-items:center}.btn-group .btn:not(:first-child):not(:last-child){border-radius:0}.btn-group .btn:not(:last-child):first-child{border-top-right-radius:0;border-bottom-right-radius:0}.btn-group .btn:not(:first-child):last-child{border-top-left-radius:0;border-bottom-left-radius:0}.btn-group .btn+.btn{margin-left:-1px}.btn{position:relative;z-index:3;display:inline-flex;align-items:center;justify-content:center;column-gap:8px;min-width:1px;height:32px;padding:0 14px;flex-shrink:0;border-radius:6px;border:none;background:0 0;color:rgba(0,0,0,.8);color:var(--btn-default-text);text-align:center;line-height:30px;user-select:none;transition:color .2s,border .2s,opacity .2s,transform .2s,filter .2s}.btn:before{content:"";position:absolute;top:0;bottom:0;left:0;right:0;z-index:-1;background-color:currentColor;border-radius:inherit;opacity:.08;transition:opacity .3s cubic-bezier(.4, 0, .6, 1)}.btn:not(.is-link):hover:before{opacity:.1}.btn:active,.card:active{transform:scale(.96)}.btn:active:before{opacity:.15}.btn:disabled{opacity:.6}.btn span{white-space:nowrap;text-overflow:ellipsis}.btn.is-outline{border:1px solid #dcdee2;border:solid 1px var(--btn-default-border)}.btn.is-outline:not(.is-rounded):before{border-radius:5px}.btn.is-active{color:var(--btn-default-text-color-active)}.btn.is-icon{width:32px}.btn.is-link{padding:0;height:auto;min-width:20px;background:0 0;border-radius:0;line-height:18px}.btn.is-full-width{width:100%}.btn.size-xs{font-size:12px}.btn.size-xs:not(.is-link){padding-left:8px;padding-right:8px;height:24px;line-height:22px}.btn.size-xs.is-icon{width:24px;font-size:12px}.btn.size-sm{height:26px;line-height:24px;font-size:13px;padding:0 10px;font-weight:400}.btn.size-sm.is-icon{width:26px;font-size:12px}.btn.size-sm.is-rounded{border-radius:26px}.btn.size-lg{height:40px;padding-left:18px;padding-right:18px;font-weight:600;line-height:38px;border-radius:8px}.btn.size-lg.is-icon{width:40px}.btn.is-icon{padding-left:0;padding-right:0}.btn.variant-light{background:#fff;background:var(--btn-light-bg);color:#212529;color:var(--btn-light-text)}.btn.variant-light:before{background:rgba(0,0,0,.024);background:var(--btn-light-bg-hover)}.btn.variant-light.is-outline{border:1px solid #dcdee2;border:solid 1px var(--btn-light-border)}
```

ПРОДОВЖЕННЯ ДОДАТКУ Г

mercury_dev.css

```
.btn.variant-secondary{color:#8a8a8e;color:var(--text-secondary)}.btn.hover-primary:hover,.btn.variant-primary{color:var(--btn-primary)}.btn.hover-danger:hover:before,.btn.hover-primary:hover:not(.is-filled):not(.is-plain):before,.btn.hover-success:hover:before,.btn.variant-danger:before,.btn.variant-primary:not(.is-filled):not(.is-plain):before,.btn.variant-success:before{background:currentColor;opacity:.08}.btn.hover-primary:hover.is-outline,.btn.variant-primary.is-outline{border-color:currentColor}.btn.hover-primary:hover.is-filled,.btn.variant-primary.is-filled{color:#fff;color:var(--white)}.btn.hover-primary:hover.is-filled.is-outline,.btn.variant-primary.is-filled.is-outline{border-color:var(--btn-primary-filled)}.btn.hover-primary:hover.is-filled:before,.btn.variant-primary.is-filled:before{color:var(--btn-primary-filled)}.btn.hover-primary:hover.is-link,.btn.variant-primary.is-link{color:var(--text-link)}.btn.hover-primary:hover.is-link:hover,.btn.variant-primary.is-link:hover{filter:brightness(.8)}.btn.hover-danger:hover,.btn.variant-danger{color:#f44336;color:var(--red)}.btn.hover-danger:hover.is-outline,.btn.variant-danger.is-outline{border-color:currentColor}.btn.hover-danger:hover.is-filled,.btn.variant-danger.is-filled{color:#fff;color:var(--white)}.btn.hover-danger:hover.is-filled:before,.btn.variant-danger.is-filled:before{color:#f44336;color:var(--red)}.btn.hover-danger:hover:hover:before,.btn.variant-danger:hover:before{opacity:.12}.btn.hover-success:hover,.btn.variant-success{color:#3cce7b;color:var(--green)}.btn.hover-success:hover.is-filled,.btn.variant-success.is-filled{color:#fff;color:var(--white)}.btn.hover-success:hover.is-filled:before,.btn.variant-success.is-filled:before{color:#3cce7b;color:var(--green)}.btn.hover-success:hover.is-outline,.btn.variant-success.is-outline{border-color:currentColor}.btn.variant-info,.btn.variant-info.is-filled:before{color:#007deb;color:var(--blue)}.btn.variant-info.is-filled,.btn.variant-warning.is-filled{color:#fff}.btn.variant-info.is-outline,.btn.variant-warning.is-outline{border-color:currentColor}.btn.variant-warning,.btn.variant-warning.is-filled:before{color:#f77519;color:var(--orange)}.btn.variant-black{color:#fff;color:var(--white)}.btn.variant-header,.btn__loader .loader{color:currentColor}.btn.variant-black:before{background:#000;opacity:.7}.btn.is-filled:active:before,.btn.variant-black:hover:before{opacity:.8}.btn.variant-input{background:#fff;background:var(--foreground)}.btn.variant-input.is-outline{border:1px solid #dcdee2;border:solid 1px var(--input-border)}.modal__container,.warninfo{border-radius:var(--border-radius)}.btn.variant-input.is-outline:hover{z-index:4;border-color:var(--input-border-focus)}.btn.is-plain:before,.btn.variant-header:before{opacity:0}.btn.is-rounded{border-radius:40px}.btn.is-shadow{box-shadow:0 1px 3px rgba(0,0,0,.18),0 0 1px rgba(0,0,0,.18)}.btn.is-loading span,.btn.is-loading svg:not(.loader){color:transparent}.btn.is-filled:hover:before{opacity:.9}.btn__loader{position:absolute;top:50%;left:50%;z-index:5;margin-top:-9px;margin-left:-9px}.card:not(.active-card):has(.finished):hover{opacity:.7}.warninfo{padding:1.8rem 2.4rem;background:var(--selection-bg);color:var(--selection-text);margin:1.2rem}.modal__overlay{position:fixed;top:0;left:0;right:0;bottom:0;background:rgb(0 0 0 / 80%);backdrop-filter:blur(4px);display:flex;justify-content:center;align-items:center}.modal__container{background-color:var(--color-third);max-width:500px;max-height:100vh;overflow-y:hidden;box-sizing:border-box}.modal__header{display:flex;justify-content:space-between;align-items:center;padding:1.8rem 3.2rem;background:var(--color-fourth);font-size:1.6rem;font-weight:500}.modal-text-header svg{margin-right:.8rem}.modal__title{margin-top:0;margin-bottom:0;font-weight:600;font-size:1.25rem;line-height:1.25;color:#00449e;box-sizing:border-box}.modal__close{background:0 0;border:0;margin-left:4.8rem}.chart .apexcharts-canvas,code.withbg{background:var(--color-secondary);border-radius:var(--border-radius)}.modal{z-index:101;position:relative}
```

ПРОДОВЖЕННЯ ДОДАТКУ Г

mercury_dev.css

```
.modal__header .modal__close:before{content:"\2715"}.modal__content{margin-top:2rem;margin-
bottom:2rem;line-height:1.5;color:rgba(0,0,0,.8)}.modal__btn{font-size:.875rem;padding:.5rem
1rem;background-color:#e6e6e6;color:rgba(0,0,0,.8);border-radius:.25rem;border-style:none;border-
width:0;-webkit-appearance:button;text-transform:none;overflow:visible;line-
height:1.15;margin:0;-moz-osx-font-smoothing:grayscale;-webkit-backface-
visibility:hidden;backface-visibility:hidden;-webkit-
transform:translateZ(0);transform:translateZ(0);transition:transform .25s ease-
out;transition:transform .25s ease-out,-webkit-transform .25s ease-
out}.modal__btn:focus,.modal__btn:hover{-webkit-
transform:scale(1.05);transform:scale(1.05)}.modal__btn-primary{background-
color:#00449e;color:#fff}@keyframes mmfadeIn{from{opacity:0}to{opacity:1}}@keyframes
mmfadeOut{from{opacity:1}to{opacity:0}}@keyframes
mmslideIn{from{transform:translateY(15%)}to{transform:translateY(0)}}@keyframes
mmslideOut{from{transform:translateY(0)}to{transform:translateY(-10%)}}.micromodal-slide[aria-
hidden=false] .modal__overlay{animation:.3s cubic-bezier(0,0,.2,1) mmfadeIn}.micromodal-
slide[aria-hidden=false] .modal__container{animation:.3s cubic-bezier(0,0,.2,1)
mmslideIn}.micromodal-slide[aria-hidden=true] .modal__overlay{animation:.3s cubic-
bezier(0,0,.2,1) mmfadeOut}.micromodal-slide[aria-hidden=true] .modal__container{animation:.3s
cubic-bezier(0,0,.2,1) mmslideOut}body .apexcharts-tooltip.apexcharts-theme-dark{background:var(-
-color-fourth);color:var(--color-eighth)}.chart .apexcharts-
canvas{margin:2.4rem}code.withbg{padding:.2rem .6rem}ul.dots{list-style:disc;margin-
left:1.8rem}.cont-acord h2{font-size:1.8rem;color:var(--color-eleventh)}.margin-b{margin-
bottom:2rem}.sh-p{color:var(--color-seventh)}.cont-acord section:last-child,ul.dots li:last-
child{margin-bottom:unset}.cont-acord section{padding-left:1.4rem;border-left:.2rem solid var(--
color-fifth);margin-bottom:1.6rem}@keyframes ping-
main{100%,75%{transform:scale(1.4);opacity:0}}@media screen and (max-width:727px){.rb-
block{margin-left:unset;margin-top:2rem}.block-restrict-main{flex-flow:column}.results-from-
db{margin:1em 0}.results-from-db tr td:not(:first-child){min-width:13rem}.container{padding-
right:2rem;padding-left:2rem}header h1{font-size:1.4rem}header h2{font-size:1.2rem}}
```

БІБЛІОГРАФІЧНА ДОВІДКА

Тема магістерської роботи: Дослідження швидкості передачі даних в мережі Інтернет з урахуванням мережевих затримок.

Перелік графічної частини (8 аркушів):

Аркуш 1. МР.МТТм - 29.00.00.001 – Схема роботи програмного забезпечення;

Аркуш 2. МР.МТТм - 29.00.00.002 – Схема роботи TCP/IP;

Аркуш 3. МР.МТТм - 29.00.00.003 – Схема вимірювання Jitter;

Аркуш 4. МР.МТТм - 29.00.00.004 – Схема вимірювання Ping;

Аркуш 5. МР.МТТм - 29.00.00.005 – Схема вимірювання скорості завантаження/передачі даних;

Аркуш 6. МР.МТТм - 29.00.00.006 – Схема роботи Websocket в програмному забезпечені;

Аркуш 7. МР.МТТм - 29.00.00.007 – Демонстрація практичної реалізації, зовнішній вигляд сайту;

Аркуш 8. МР.МТТм - 29.00.00.008 – Метрологічні аспекти програмного забезпечення.

20.12.2024 р. _____

(Підпис) (П.І.П.)