Хмельницький національний університет
Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

Українця Андрія Дмитровича
Прізвище, ім'я, по батькові студента(ки)

на здобуття ступеня вищої освіти Бакалавра

Програмна система тестування та моніторингу
Назва теми

якості інтернет послуг

Галузь знань 12 «Інформаційні технології»

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

Шифр КвРІПЗ.2101091.01.17.ПЗ

Виконав студент IV курсу, група ІПЗ-21-1 _____ Андрій УКРАЇНЕЦЬ
Підпис — Ім'я, ПРІЗВИЩЕ

Керівник канд. техн. наук, доцент _____ Юрій ФОРКУН
Науковий ступінь, звання — Підпис — Ім'я, ПРІЗВИЩЕ

Нормоконтролер ст. викладач _____ Ганна БЕДРАТЮК
Підпис — Ім'я, ПРІЗВИЩЕ

До захисту допускаю:
Завідувач кафедри інженерії програмного забезпечення _____ Леонід БЕДРАТЮК
Підпис — Ім'я, ПРІЗВИЩЕ

9 червня 2025 р.

Хмельницький 2025

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій
Кафедра Інженерії програмного забезпечення
Рівень вищої освіти Перший (бакалаврський)
Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення»
Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри Л. П. Бедратюк
20.01 2025 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

Українцю Андрію Дмитровичу
Прізвище, ім'я, по батькові студента

1. Тема кваліфікаційної роботи Програмна система тестування та моніторингу якості інтернет послуг

Керівник кваліфікаційної роботи Форкун Юрій Вікторович, канд. техн. наук, доцент
Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 07.02.2025 р. № 6
2. Строк подання студентом роботи на кафедру 01.06.2025 р.
3. Вихідні дані до роботи Матеріали переддипломної практики
4. Зміст пояснювальної записки (перелік питань, які потрібно розробити) Дослідження предметної області та постановка задачі, проєктування мобільного застосунку, програмна реалізація та тестування
5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень) Презентаційні матеріали (слайди, 18 шт.), діаграма варіантів використання, діаграма архітектури вебзастосунку за шаблоном MVC, спрощена ER-діаграма, ER-діаграма, загальна діаграма архітектури вебзастосунку, діаграма послідовностей для процесу автентифікації користувача діаграма послідовностей для процесу додавання історії подорожі, діаграма компонентів вебзастосунку

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Бедратюк Г.І, ст. викладач		
Антиплагіат	Форкун Ю.В., к.т.н.,	12.05.2025	5.06.2025

7. Дата видачі завдання «2» 01 2025р.

КАЛЕНДАРНИЙ ПЛАН

Назва етапів (розділів) кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1 Ознайомлення з тематикою дипломного проєктування, визначення та узгодження індивідуальних тем кваліфікаційних робіт (КвР)	01.12 – 31.12.2024	
2 Збір матеріалу за темою КвР; дослідження предметної області, в якій планується використання програмного забезпечення (ПЗ), визначення задач та вимог, розробка технічного завдання	01.01 – 20.02.2025	
3 Проєктування програмного забезпечення	21.02 – 20.03.2025	
4 Програмна реалізація з використанням відповідних засобів розробки. Тестування ПЗ	21.03 – 30.04.2025	
5 Написання вступу, загальних висновків, оформлення переліку джерел посилання та додатків. Оформлення пояснювальної записки КвР згідно вимог	01.05 – 25.05.2025	
6 Попередній захист КвР	Травень 2025	Згідно графіка
7 Перевірка КвР на плагіат, нормоконтроль, отримання відгуків, рецензій та інших супровідних документів. Брошурування (зшиття) пояснювальної записки.	26.05 – 30.05.2025	
8 Здача КвР на кафедру; підготовка КвР для розміщення у репозитарії ХНУ; підготовка до захисту та захист КвР	з 01.06.2025	

Студент _____ Українець А.Д.
Підпис Ініціали, прізвище

Керівник роботи _____ Форкун Ю.В.
Підпис Ініціали, прізвище

АНОТАЦІЯ

Тема кваліфікаційної роботи: «Програмна система тестування та моніторингу якості інтернет послуг».

Автор роботи: Українець Андрій Дмитрович.

Керівник роботи: Форкун Юрій Вікторович.

Пояснювальна записка: 60 с., 11 рис., 6 табл., 3 дод., 40 джерела.

Графічна частина: 18 слайдів.

Метою роботи є розробка системи яка допоможе користувачам дізнатись про якість інтернет послуг, тестувати їх, проводити аналіз помилок на обладанні та перевіряти кабель на помилки.

У роботі представлено розробку веборієнтованої програмної системи для тестування та моніторингу якості інтернет-послуг. Система дозволяє виконувати вимірювання ключових параметрів з'єднання, зокрема швидкості завантаження та вивантаження даних, затримки, втрати пакетів тощо. Архітектура рішення побудована з використанням сучасних технологій: фронтенд реалізовано на базі Vue.js, серверну частину — з використанням Java Spring Boot, а як проксі-сервер застосовано Envoy. Забезпечено збереження результатів, перегляд історії вимірювань і базову аналітику. Програмна система призначена для кінцевих користувачів, провайдерів і системних адміністраторів, які зацікавлені в контролі якості наданих інтернет-послуг.

У кваліфікаційній роботі проведено аналіз предметної області, досліджено аналогічні сервіси, визначено функціональні та нефункціональні вимоги, спроєктовано архітектуру системи, обґрунтовано вибір стеку реалізовано основні модулі мобільного застосунку та проведено тестування.

02.06.2025
Дата

Підпис

ВІДОМІСТЬ ДОКУМЕНТІВ

№ рядка	Формат	Позначення документа	Найменування документа	К-сть аркушів	№ екз.	Примітка
			Текстові документи			
1	А4	КвРІПЗ.2101082.01.10.ПЗ	Пояснювальна записка	69		
2	А4		Завдання на кваліфікаційну роботу	1		
3	А4		Анотація	1		
			Графічні документи			
4	А4	КвРІПЗ.2101091.01.17.Е8	Діаграми патерну об'єкт	1		
			запиту та репозиторій			
5	А4	КвРІПЗ.2101091.01.17.Е8	UML-діаграма варіантів використання	1		

Змн.	Арк.	№ докум.	Підпис	Дата	КвРІПЗ. 2101091.01.17.ПЗ		
Виконав		Українець А.Д.		06.06	Програмна система тестування та моніторингу якості інтернет послуг Пояснювальна записка	Літ.	Арк. 1 / Аркушів 1
Керівник		Фарнуш Ю.В.		06.06			
Н. Контр.		Бедратюк Г.І.		06.06		ХНУ, ІПЗ-21-1	
Зав. Каф.		Бедратюк Л.П.		06.06			

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ 8
ВСТУП 9
1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ 12
1.1 Змістовний аналіз предметної області, її структурних та функціональних особливостей 12
1.2 Аналіз наявного програмно-технічного забезпечення предметної області. 14
1.3 Визначення функціональних та нефункціональних вимог до програмного забезпечення 18
1.4 Висновки дослідження предметної області та постановки задачі 21
2 ПРОЄКТУВАННЯ МОБІЛЬНОГО ЗАСТОСУНКУ 22
2.1 Архітектурне проектування 22
2.1.1 Аналіз клієнт-серверної архітектури сучасних веб-сервісів 23
2.1.2 Аналіз підходів до проектування серверної архітектури 26
2.2 Детальне проектування 30
2.2.1 Аналіз та вибір типу бази даних 33
2.2.2 Проектування модульної архітектури 35
2.2.3 Проектування модульної архітектури 38
2.3 Проектування інтерфейсу користувача 40
3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ 45
3.1 Реалізація веб-застосунку 45
3.2 Вимоги до технічних та програмних засобів 51
3.3 Тестування системи 52
3.3.1 Тестування веб-застосунку 54
3.3.3 Аналіз результатів тестування веб-застосунку 57

3.4 Висновки програмної реалізації та тестування 62
ВИСНОВКИ 64
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ 67
ДОДАТОК А Технічне завдання 74
ДОДАТОК Б Діаграми діяльності 81
ДОДАТОК В Код програми 81
ДОДАТОК Г Презентаційні слайди 99

ПЕРЕЛІК СКОРОЧЕНЬ

API – Application Programming Interface (інтерфейс прикладного програмування)

DB – Database (база даних)

HTTP/HTTPS – HyperText Transfer Protocol / Secure (протокол передавання гіпертексту / захищений)

ELK Stack – Elasticsearch, Logstash, Kibana (стек засобів для збору та візуалізації логів)

UI – User Interface (інтерфейс користувача)

IP – Internet Protocol (інтернет-протокол)

DNS – Domain Name System (система доменних імен)

ICMP – Internet Control Message Protocol (протокол керування повідомленнями Інтернету)

QoS – Quality of Service (якість обслуговування)

CRUD – Create, Read, Update, Delete (операції створення, читання, оновлення та видалення)

JSON – JavaScript Object Notation (формат обміну даними)

CI/CD – Continuous Integration / Continuous Deployment (безперервна інтеграція / розгортання)

CPU – Central Processing Unit (центральний процесор)

RAM – Random Access Memory (оперативна пам’ять)

CLI – Command Line Interface (інтерфейс командного рядка)

UI/UX – User Interface / User Experience (інтерфейс користувача / досвід користувача)

ВСТУП

У сучасному світі інтернет-послуги є однією з ключових складових успішної роботи будь-якої компанії, особливо тієї, яка залежить від стабільного та швидкого інтернет-з'єднання. Для забезпечення високої якості послуг необхідно постійно моніторити основні параметри мережі, такі як швидкість передачі даних, затримка, втрати пакетів та стабільність з'єднання. Однак існуючі рішення для тестування якості інтернет-з'єднання часто є загальнодоступними та не завжди відповідають специфічним потребам компаній. Крім того, вони не забезпечують достатнього рівня контролю та конфіденційності, що є критично важливим для внутрішнього використання. Саме тому розробка спеціалізованої програмної системи, яка буде доступна виключно для працівників компанії, стає актуальною та необхідною.

Актуальність даної теми полягає в необхідності створення інструменту, який дозволить компанії ефективно контролювати якість інтернет-послуг, оперативно виявляти проблеми та приймати обґрунтовані рішення щодо покращення роботи мережі. Існуючі загальнодоступні рішення, такі як Speedtest або інші онлайн-сервіси, не завжди забезпечують необхідний рівень деталізації, конфіденційності та інтеграції з внутрішніми системами компанії. Крім того, вони не дозволяють зберігати історію тестів для подальшого аналізу, що ускладнює процес виявлення тривалих проблем або тенденцій у роботі мережі. Тому розробка власної програмної системи, яка буде адаптована під конкретні потреби компанії, є важливим кроком у напрямку покращення якості інтернет-послуг.

Метою даної роботи є розробка програмної системи для тестування та моніторингу якості інтернет-послуг, яка буде доступна виключно для працівників компанії. Система повинна автоматично вимірювати основні параметри мережі, такі як швидкість передачі даних, затримка, втрати пакетів та стабільність з'єднання, а також надавати зручний інтерфейс для перегляду

результатів та аналізу історії тестів. Важливим аспектом є забезпечення конфіденційності даних, оскільки система буде використовуватися виключно внутрішніми працівниками компанії. Для досягнення поставленої мети необхідно вирішити такі завдання.

Провести аналіз існуючих методів та інструментів для тестування якості інтернет-послуг, визначити їх переваги та недоліки. Визначити основні параметри якості інтернет-з'єднання, такі як швидкість передачі даних (download/upload), затримка (ping), втрати пакетів (packet loss) та стабільність з'єднання (jitter), та методи їх вимірювання. Розробити архітектуру програмної системи, яка забезпечить доступ лише для працівників компанії. Це включає створення модулів для збору даних, аналізу та візуалізації результатів, а також реалізацію механізмів автентифікації та авторизації. Реалізувати систему, використовуючи сучасні технології та інструменти, такі як Python для обробки даних, Django або Flask для створення бекенду, React або Vue.js для розробки інтерфейсу користувача, а також бази даних для зберігання історії тестів. Забезпечити захист даних та конфіденційність інформації, що обробляється системою. Це включає використання шифрування даних, захист від несанкціонованого доступу та реалізацію механізмів резервного копіювання. Протестувати систему на різних типах з'єднань (DSL, оптоволокно, мобільний інтернет) та оцінити її ефективність. Порівняти результати з існуючими інструментами, такими як Speedtest, щоб переконатися у точності та надійності системи. Результатом роботи стане програмна система, яка дозволить працівникам компанії ефективно моніторити якість інтернет-послуг, оперативно виявляти проблеми та приймати рішення щодо покращення роботи мережі. Система буде захищена від несанкціонованого доступу, що забезпечить конфіденційність даних та відповідність внутрішнім вимогам компанії. Розробка та впровадження подібної системи мають значний потенціал для покращення якості інтернет-послуг та забезпечення стабільної роботи мережі в компаії.

Крім того, система може бути інтегрована з іншими внутрішніми інструментами компанії, такими як системи моніторингу мережевого обладнання або платформи для управління інцидентами. Це дозволить автоматизувати процес виявлення та усунення проблем, що значно підвищить ефективність роботи IT-відділу компанії. Також система може бути розширена шляхом додавання нових функцій, таких як аналіз тенденцій у роботі мережі, генерація звітів або інтеграція з хмарними сервісами для зберігання даних.

Важливим аспектом є також економічна ефективність розробки та впровадження системи. Використання сучасних технологій та відкритих рішень дозволить мінімізувати витрати на розробку, а автоматизація процесів моніторингу дозволить скоротити витрати на обслуговування мережі. Крім того, система може бути адаптована під потреби інших підрозділів компанії, що робить її універсальним інструментом для покращення якості інтернет-послуг.

У висновку можна сказати, що розробка програмної системи для тестування та моніторингу якості інтернет-послуг, яка буде доступна виключно для працівників компанії, є важливим кроком у напрямку покращення якості послуг та забезпечення стабільної роботи мережі. Ця система дозволить компанії ефективно контролювати основні параметри мережі, оперативно виявляти проблеми та приймати обґрунтовані рішення щодо покращення роботи мережі, що в кінцевому підсумку позитивно вплине на продуктивність та конкурентоспроможність компанії.

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Змістовний аналіз предметної області, її структурних та функціональних особливостей

Програмна система тестування та моніторингу якості інтернет-послуг – це комплексне рішення, що дозволяє здійснювати аналіз роботи мережевої інфраструктури, зокрема швидкості з'єднання, рівня затримок, стабільності з'єднання та інших параметрів. Основна мета таких систем – забезпечення високої якості обслуговування кінцевих користувачів та своєчасне виявлення проблем у мережі.

Сучасний інтернет-сервіс потребує постійного контролю, оскільки навантаження на мережу зростає, а користувачі очікують стабільного з'єднання. Оператори зв’язку та інтернет-провайдери використовують різні інструменти для моніторингу стану мережі, аналізу продуктивності комутаторів, маршрутизаторів та агрегаторів трафіку. Програмна система, що пропонується в межах даного дослідження, має забезпечити автоматизований процес збору даних, їх обробку та надання аналітики у зручному форматі.

Система включає такі основні компоненти:

– база даних абонентів, де зберігається інформація про користувачів мережі, їхні тарифні плани, IP-адреси та історію використання послуг;

– доступ до бази даних мережевого обладнання, що містить інформацію про комутатори, маршрутизатори, агрегатори трафіку, їх конфігурацію та поточний стан;

– модуль тестування якості послуг, який проводить вимірювання швидкості з’єднання, пінгу, втрат пакетів та загальної продуктивності мережі;

– система сповіщень, що інформує адміністраторів про критичні події, зниження якості зв’язку або виявлені проблеми.

Одним із ключових завдань є розробка зручного веб-інтерфейсу для взаємодії з системою. Користувачі (адміністратори, технічні спеціалісти) повинні мати змогу переглядати звіти, аналізувати динаміку змін у якості інтернет-з'єднання та отримувати рекомендації щодо оптимізації мережі.

Для опису та формалізації основних процесів програмної системи буде використано методологію функціонального моделювання IDEF-0. На контекстній діаграмі можна виділити вхідні дані – це тести продуктивності, запити користувачів та інформація з комутаторів та агрегаторів. Вихідні дані включають звіти про якість інтернет-послуг, сповіщення про відхилення від норми та рекомендації щодо усунення проблем.

На діаграмі декомпозиції першого рівня будуть деталізовані основні функціональні можливості: збір даних із мережевого обладнання, проведення тестування швидкості, обробка результатів та генерація аналітичних звітів.

Крім того, система може включати можливості інтеграції зі сторонніми сервісами моніторингу та API для збору розширених даних про мережеву інфраструктуру. Для захисту інформації передбачається реалізація рівнів доступу та шифрування даних.

Також важливим аспектом роботи програмної системи є можливість автоматизованого аналізу історичних даних для виявлення закономірностей та трендів у зміні якості інтернет-послуг. Використання методів машинного навчання та статистичного аналізу дозволить прогнозувати потенційні проблеми в мережі та запобігати їхньому виникненню.

Одним із ключових викликів у процесі моніторингу є обробка великого обсягу даних у реальному часі. Для цього система повинна бути оптимізована для роботи з розподіленими базами даних, що забезпечить швидке отримання інформації та зменшення навантаження на центральний сервер. Оптимізація запитів до бази даних та використання індексування дозволять мінімізувати затримки при обробці запитів користувачів.

Ще одним важливим завданням є забезпечення високого рівня безпеки збереження даних. Оскільки система працює з конфіденційною інформацією,

такою як IP-адреси абонентів, параметри мережевого обладнання та статистика використання трафіку, необхідно реалізувати механізми шифрування даних, аутентифікації користувачів та розмежування прав доступу. Це дозволить запобігти несанкціонованому доступу до критично важливої інформації.

Крім того, у процесі розробки системи необхідно врахувати можливість її масштабування. Із зростанням кількості користувачів та мережевого обладнання система повинна зберігати свою продуктивність. Для цього можливе використання розподілених обчислень, балансування навантаження між серверами та адаптивного керування ресурсами.

Важливим є також інтуїтивно зрозумілий веб-інтерфейс, що дозволить користувачам швидко орієнтуватися в системі. Дашборди повинні відображати ключові показники якості інтернет-послуг, графіки змін параметрів мережі, історію тестувань та сповіщення про критичні події. Використання сучасних технологій фронтенд-розробки, таких як React чи Vue.js, дозволить створити швидкий та зручний інтерфейс.

Додатковою функціональністю, яка може бути реалізована у системі, є інтеграція з мобільними додатками. Це дасть можливість адміністраторам отримувати сповіщення про критичні проблеми безпосередньо на смартфон, а користувачам – запускати тести якості інтернет-з’єднання зі своїх мобільних пристроїв.

Таким чином, розробка такої програмної системи є актуальним завданням, що сприятиме покращенню якості інтернет-послуг, підвищенню ефективності роботи провайдерів та забезпеченню кращого користувацького досвіду.

1.2 Аналіз наявного програмно-технічного забезпечення предметної області

На сьогодні існує безліч програмних рішень для моніторингу мережевих з'єднань та перевірки якості інтернет-послуг, проте більшість з них є

закордонними розробками та не завжди адаптовані під потреби вітчизняних провайдерів. Тому важливо розглянути існуючі аналоги, визначити їхні переваги та недоліки, а також обґрунтувати доцільність розробки нової системи.

Серед популярних сервісів можна виділити такі:

PRTG Network Monitor – потужний інструмент для аналізу мережевого трафіку та стану обладнання, який підтримує інтеграцію з різними комутаторами та маршрутизаторами. Відрізняється широкими можливостями збору статистики.

Рисунок 1.1 – PRTG Network Monitor

З мінусів це складний для нових користувачів інтерфейс, не наявна локалізація та висока ціна за користування.

Nagios XI – система моніторингу, що дозволяє контролювати мережеві ресурси та виявляти проблеми.

Рисунок 1.2 – Nagios XI

Проте потребує складного налаштування та не має інтуїтивно зрозумілого веб-інтерфейсу;

Zabbix – відкрите рішення для моніторингу серверів і мереж, яке підтримує різні методи збору даних та візуалізацію показників у режимі реального часу. Попри свою гнучкість, має високиq поріг входження для нових користувачів.

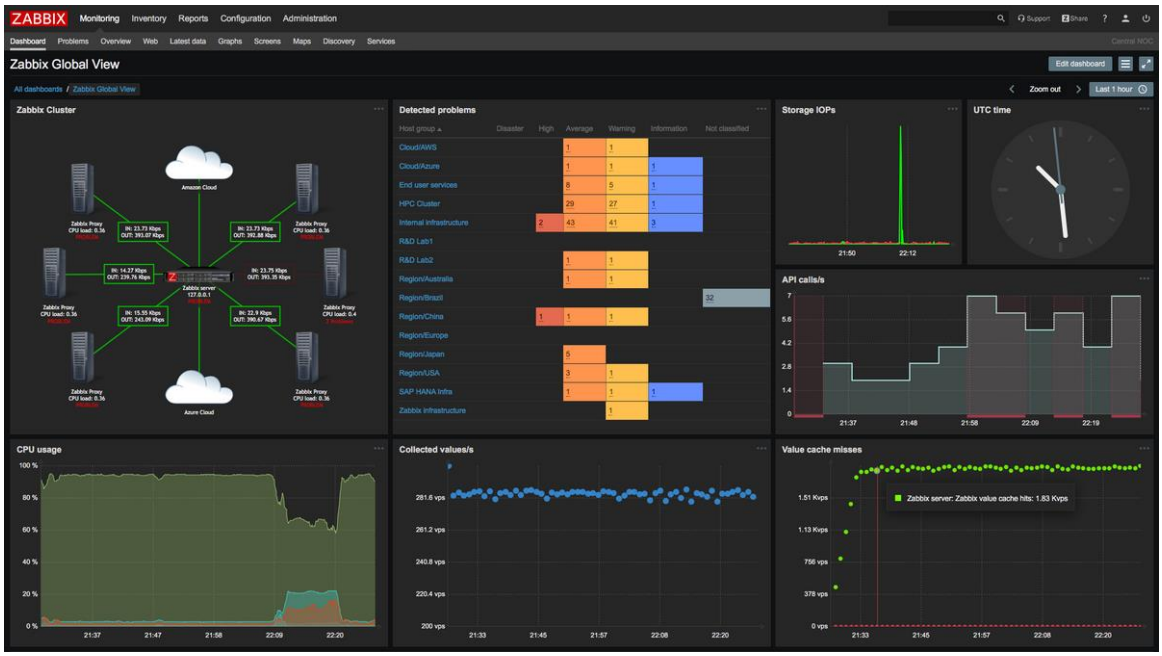


Рисунок 1.3 – Zabbix

Незважаючи на наявність цих систем, вони часто не містять зручного інтерфейсу для швидкого аналізу абонентських проблем. Враховуючи це, розроблювана програмна система пропонує унікальний функціонал для провайдерів, що дозволяє швидко виявляти проблеми на рівні окремих абонентів.

Основні можливості системи включають:

— аналіз статусу підключення: відображення, чи є лінк на обладнання та чи проходить трафік;

— визначення помилок на порту комутатора та оцінка довжини кабелю між комутатором та кінцевим обладнанням;

— відстеження MAC та IP-адреси пристрою;

— перегляд інформації про модель комутатора, аптайм пристрою та його місцезнаходження;

— моніторинг балансу абонента та його тарифного плану.

Таблиця 1.1 – Порівняння існуючих варіантів пз

Критерій	PRTG Network Monitor	Nagios XI	Zabbix	Розроблювана система
Моніторинг трафіку	Так	Так	Так	Так
Аналіз комутаторів	Частково	Так	Так	Так
Визначення MAC-адреси	Ні	Ні	Так	Так
Оцінка довжини кабелю	Ні	Ні	Ні	Так
Перевірка лінку	Так	Так	Так	Так
Інформація про тариф	Ні	Ні	Ні	Так
Інтуїтивно зрозумілий інтерфейс	Складний	Складний	Складний	Зручний для користувачів
Перевірка помилок на кабелі	Ні	Ні	Ні	Так

Проведений аналіз аналогічного програмного забезпечення показав, що розроблювана програмна система повинна мати:

– можливість моніторингу стану підключення абонента, включаючи перевірку наявності лінку, наявність трафіку та помилки на порту комутатора;

– зручний пошук абонентів за адресою, MAC-адресою, IP-адресою або кодом договору;

– сучасний, інтуїтивно зрозумілий дизайн інтерфейсу, що дозволяє швидко отримати необхідну інформацію

– функціонал детального аналізу підключення, включаючи визначення довжини кабелю між комутатором та кінцевим обладнанням;

– зручну форму авторизації для співробітників з рівнями доступу;

– можливість перегляду тарифного плану, балансу абонента та додаткової інформації про підключення.

Таким чином, розроблювана система надає більш спеціалізовані можливості для моніторингу якості інтернет-послуг на рівні окремих абонентів. Вона спрощує роботу технічної підтримки та адміністраторів мереж, дозволяючи ідентифікувати та вирішувати проблеми.

1.3 Визначення функціональних та нефункціональних вимог до програмного забезпечення

Для визначення вимог до програмного забезпечення використовується UML – мова моделювання, яка призначена для побудови діаграм, компоненти яких представляють об’єкти програмного забезпечення та документують і відображають вимоги до ПЗ. UML є загальноприйнятим стандартом для графічного опису програмного забезпечення, що допомагає наочно переглядати компонування системи та полегшує співпрацю з іншими її розробниками.

Однією з ключових UML-діаграм є діаграма варіантів використання, яка відображає основні функції системи, а також залежності та взаємозв’язки між варіантами використання і користувачами (акторами).

Таблиця 1.2 опис акторів програмної системи.

Актор	Короткий опис
Оператор служби підтримки	Має можливість перевіряти статус підключення абонента, діагностувати мережеві проблеми, отримувати інформацію про MAC-адресу, IP-адресу та активність пристрою.

Продовження таблиці 1.2

Технічний спеціаліст	Додатково до можливостей оператора, має змогу отримувати інформацію про порт комутатора, його помилки, перевіряти довжину кабелю та параметри лінку.
Адміністратор системи	Може налаштовувати доступи користувачів, додавати нові комутатори та мережеві вузли, переглядати та аналізувати лог-файли, змінювати конфігурацію системи.

Як видно із таблиці 1.2, у системі існує три основні ролі: оператор служби підтримки, технічний спеціаліст та адміністратор системи. Вони взаємодіють з різними рівнями доступу та мають різні функціональні можливості.

Таблиця 1.3 – основні варіанти використання програмної системи.

Актор	Найменування ВВ	Опис ВВ
Оператор служби підтримки	Перегляд статусу підключення абонента. Аналіз трафіку та помилок на порту	Оператор може перевірити наявність лінку, поточний IP-адрес, MAC-адресу, баланс абонента, статистику використання трафіку, визначити наявність помилок на порту комутатора.
Технічний спеціаліст	Діагностика проблем з підключенням. Отримання детальної інформації про порт комутатора	Може визначити причину відсутності зв’язку, перевіряти параметри підключення та довжину кабелю. Спеціаліст може переглядати параметри комутатора, його модель, порт та час останнього аптайму.
Адміністратор системи	Керування доступом користувачів. Налаштування мережевого обладнання	Може створювати, редагувати та видаляти облікові записи операторів та технічних спеціалістів. Адміністратор може оновлювати конфігурацію комутаторів, додавати нові пристрої.

Відповідно до описаних акторів та варіантів використання побудовано UML – діаграму.

Рисунок 1.4 –діаграма варіантів використання

Отже, проведений аналіз предметної області показав, що якість інтернет-послуг залежить від ефективного моніторингу та діагностики проблем на рівні обладнання та підключення абонента.

Також було проведено аналіз існуючих систем моніторингу та тестування інтернет-з'єднань. Визначено, що більшість таких систем надають лише базову інформацію про статус підключення без детальної діагностики на рівні комутатора та кінцевого обладнання.

На основі цього були сформульовані функціональні та нефункціональні вимоги до програмного забезпечення та виконано опис основних сценаріїв його використання.

1.4 Висновки дослідження предметної області та постановки задачі

У результаті аналізу предметної області встановлено, що стабільність та якість інтернет-з’єднання є критично важливими як для кінцевих користувачів, так і для бізнес-середовища. Параметри, які найчастіше потребують моніторингу, — це швидкість передавання даних, затримка та стабільність з'єднання. Саме ці показники формують уявлення користувача про якість отримуваних послуг.

Дослідження наявного програмного забезпечення показало, що більшість існуючих рішень є закритими, не передбачають гнучкої інтеграції з іншими системами, не зберігають історію вимірювань у доступному вигляді або потребують платної підписки. Крім того, вони не враховують потреби користувача в персоналізованій аналітиці та не дозволяють проводити більш глибоке тестування чи масштабоване розгортання.

На основі цього сформульовано задачу проєкту — створення веборієнтованої системи тестування та моніторингу якості інтернет-послуг, що поєднує простий інтерфейс, надійну серверну логіку та зберігання даних, а також підтримує сучасні вимоги до безпеки, масштабованості та зручності використання. Застосування таких технологій, як Vue.js, Spring Boot, Firebase та Envoy Proxy, дозволяє створити платформу з відкритою архітектурою, яку можна адаптувати під конкретні потреби користувача або організації.

Таким чином, постановка задачі відповідає актуальним викликам у сфері контролю якості інтернет-послуг і забезпечує чітке технічне підґрунтя для подальшої реалізації програмної системи.

2 ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Архітектурне проектування

Архітектура програмної системи тестування та моніторингу якості інтернет-послуг побудована відповідно до принципів модульності, масштабованості, гнучкості та розподіленості. Основною метою архітектурного проєктування є забезпечення ефективного розділення відповідальностей між компонентами системи, спрощення розширення функціональності та підвищення якості обслуговування.

Вибір архітектурного стилю

У розробці системи застосовано архітектурний стиль "клієнт-сервер" із подальшим розділенням на фронтенд, бекенд та інфраструктурний рівень. Для підвищення тестованості, модульності та керованості коду реалізовано чисту архітектуру (Clean Architecture) на стороні бекенду та принципи розділення стану і представлення у фронтенді.

Компонентна структура системи

Архітектура системи складається з таких основних компонентів:

Клієнтська частина (Frontend):

– реалізована на основі фреймворку Vue.js;

– відповідає за відображення інтерфейсу користувача, формування запитів до серверної частини, обробку результатів перевірок, графічне представлення метрик (швидкість, затримка, втрата пакетів тощо);

– компоненти реалізовані відповідно до підходів Stateful/Stateless для ефективного управління станом.

Серверна частина (Backend):

– реалізована з використанням Java Spring Boot;

– виконує функції обробки запитів клієнта, взаємодії з базою даних, запуску перевірок якості інтернет-послуг через відповідні утиліти (наприклад, ping, traceroute, iperf);

– архітектура побудована за принципами чистої архітектури — розділення на шари: domain, application, infrastructure, web.

API шлюз / проксі-сервер:

– Для маршрутизації запитів використовується Envoy Proxy;

– забезпечує балансування навантаження, маршрутизацію, базову авторизацію та агрегацію логів.

База даних:

– використовується реляційна система керування базами даних PostgreSQL;

– зберігає результати тестів, історію моніторингів, профілі користувачів, налаштування.

Система моніторингу та логування:

– для збору логів використовуються засоби на основі ELK Stack або інтеграція з Prometheus + Grafana (опціонально).

2.1.1 Аналіз клієнт-серверної архітектури сучасних веб-сервісів

Більшість сучасних веб-додатків та інтернет-сервісів базуються на клієнт-серверній архітектурі, яка забезпечує розподіл обчислювальних ресурсів між клієнтом та сервером. Основними компонентами такої архітектури є:

— клієнт: програмне забезпечення або пристрій, що надсилає запити до сервера та отримує відповідь. У контексті системи моніторингу інтернет-послуг клієнтська частина може бути представлена у вигляді веб-інтерфейсу для користувачів, які здійснюють тестування якості мережевого з'єднання;

— сервер: більш потужне програмно-апаратне забезпечення, що приймає та обробляє запити клієнтів, зберігає дані, виконує аналітику та генерує відповіді.

Архітектура клієнт-сервер передбачає, що клієнт відправляє запит на сервер, де він обробляється, і готовий результат повертається клієнту. Сервер може одночасно обслуговувати велику кількість клієнтів. Якщо запити

надходять одночасно, вони ставляться у чергу або обробляються паралельно залежно від можливостей серверного програмного забезпечення.

Протоколи взаємодії клієнта та сервера

Для ефективної комунікації між клієнтом і сервером використовуються стандартизовані протоколи передачі даних. Найпоширенішим з них є HTTP (HyperText Transfer Protocol) та його безпечна версія HTTPS. Він дозволяє передавати дані між клієнтом і сервером у форматах, що відповідають архітектурним підходам API.

З розвитком серверних API дедалі частіше використовують формати обміну даними, що відрізняються від традиційного гіпертексту (HTML). Найпоширеніші формати:

— JSON (JavaScript Object Notation): легкий для обробки формат, що широко використовується у веб-сервісах;

— XML (Extensible Markup Language): структурований формат, який застосовується для передачі даних у складних системах.

Одним із найпоширеніших архітектурних стилів для побудови API є REST (Representational State Transfer). Він базується на використанні стандартних HTTP-методів:

— GET: отримання даних;

— POST: створення нового ресурсу;

— PUT: оновлення ресурсу;

— DELETE: видалення ресурсу.

REST API дозволяє ефективно організовувати взаємодію клієнта та сервера, забезпечуючи стандартизовану структуру запитів і відповідей. Ключові вимоги RESTful API:

— використання правильних HTTP-методів відповідно до виконуваних операцій;

— відсутність збереження стану на сервері (stateless architecture) – кожен запит містить усю необхідну інформацію для його обробки.

REST-архітектура є одним із найбільш гнучких та масштабованих підходів до побудови веб-сервісів. Вона широко використовується у сучасних системах для забезпечення швидкої та безперебійної комунікації між клієнтами та серверами.

Одним із трендів у розробці веб-сервісів є мікросервісна архітектура, яка є альтернативою традиційній монолітній структурі. У мікросервісному підході система розбивається на окремі незалежні сервіси, кожен з яких виконує певну функцію. Вони взаємодіють між собою через API, що дозволяє досягти високої гнучкості та масштабованості.

Основні переваги мікросервісної архітектури:

— гнучкість у розробці та розгортанні – зміни в одному сервісі не впливають на роботу інших;

— масштабованість: можна окремо збільшувати ресурси для найнавантаженіших сервісів;

— надійність: вихід з ладу одного мікросервісу не призводить до збоїв у всій системі.

У межах даного проєкту мікросервісний підхід може бути корисним для розділення функціональних частин системи, наприклад:

— окремий сервіс для тестування швидкості інтернет-з'єднання;

— сервіс для збереження та аналізу результатів тестування;

— сервіс для управління користувачами та їх ролями.

Застосування клієнт-серверної архітектури у системі моніторингу якості інтернет-послуг

У межах даного проєкту клієнт-серверна архітектура використовується як основа взаємодії між компонентами системи. Клієнтська частина (веб-інтерфейс) забезпечує взаємодію користувача з функціоналом тестування та моніторингу. Серверна частина відповідає за отримання запитів, проведення тестів швидкості з'єднання, перевірку мережевих параметрів та формування звітності.

Для реалізації API було обрано REST-підхід, що дозволяє стандартизувати роботу з HTTP-запитами та покращити продуктивність і масштабованість системи.

Використання клієнт-серверної архітектури та RESTful API забезпечує:

— швидке реагування системи на запити користувачів;

— масштабованість – можливість легко розширювати функціонал та обробляти більшу кількість користувачів;

— надійність – чітке розмежування між клієнтською та серверною частинами мінімізує вплив збоїв у роботі окремих компонентів.

Таким чином, клієнт-серверна архітектура в поєднанні з RESTful API є оптимальним вибором для реалізації програмної системи моніторингу та тестування якості інтернет-послуг.

2.1.2 Аналіз підходів до проектування серверної архітектури

Проектування серверної архітектури є ключовим етапом розробки сучасних веб-додатків, оскільки від нього залежать продуктивність, масштабованість, безпека та зручність підтримки програмної системи. Існує кілька підходів до реалізації серверної архітектури, проте найбільш поширеними є два основні стилі: монолітний та мікросервісний. Кожен з них має свої переваги та недоліки, що визначають їх застосування у різних типах проєктів.

Монолітна архітектура є традиційним підходом до розробки програмного забезпечення, який передбачає створення єдиного цілісного додатку. Усі компоненти такого додатку знаходяться в одній кодовій базі, а система розгортається та виконується як один процес. Це означає, що всі модулі взаємодіють напряму, що забезпечує високу продуктивність та узгодженість даних.

Монолітний додаток зазвичай складається з декількох логічних рівнів:

– рівень бізнес-логіки: відповідає за реалізацію основних функцій системи, обробку даних та виконання бізнес-процесів. Це серце програми, де визначаються правила обробки інформації та її перетворення відповідно до вимог користувача;

– рівень доступу до сховища даних: реалізує операції читання, запису, оновлення та видалення даних у базі даних (CRUD-операції). Зазвичай цей рівень використовує ORM (Object-Relational Mapping) або інші механізми для взаємодії з базами даних;

– рівень представлення (UI): відповідає за інтерфейс користувача. У веб-додатках цей рівень зазвичай включає HTML-сторінки, JavaScript-код та CSS-стилі. Також можуть використовуватися фреймворки для побудови динамічного інтерфейсу;

– рівень інтеграції: надає можливість взаємодії з зовнішніми сервісами та системами. Хоча він не є обов'язковим, його використання дозволяє розширювати функціональність системи та інтегрувати її з іншими додатками.

Переваги монолітної архітектури

– простота реалізації: всі модулі знаходяться в одній кодовій базі, що значно спрощує початкову розробку;

– менша кількість наскрізних проблем: логування, безпека та управління доступом простіше реалізовувати в єдиній системі.

– легкість тестування: інтеграційні та наскрізні тести простіше реалізувати, оскільки всі компоненти знаходяться в одному середовищі;

– простота розгортання: моноліт можна швидко задеплоїти на сервер без необхідності координації роботи між різними сервісами.

Недоліки монолітної архітектури

– складність підтримки при збільшенні системи – зі зростанням коду стає важче впроваджувати зміни без ризику порушити роботу всієї системи;

– слабка гнучкість – оновлення будь-якої частини системи потребує розгортання всього додатку, що ускладнює швидке внесення змін;

– складність масштабування – якщо якась частина програми потребує збільшення ресурсів, доводиться масштабувати весь додаток, що є неефективним.

Мікросервісна архітектура

Мікросервісний підхід є сучасною альтернативою моноліту, яка передбачає поділ додатку на набір незалежних сервісів. Кожен сервіс є окремим процесом, що має власну базу даних (або використовує спільний кластер) та API для взаємодії з іншими сервісами.

Основні особливості мікросервісної архітектури

– слабке зв'язування – сервіси можуть взаємодіяти один з одним через API, що дозволяє змінювати або оновлювати окремі компоненти без ризику порушення роботи всієї системи;

– автономність сервісів – кожен сервіс має власну базу даних та логіку, що зменшує залежності між модулями;

– гнучкість у виборі технологій – різні сервіси можуть бути реалізовані на різних мовах програмування та використовувати різні бази даних;

– масштабованість – окремі сервіси можна масштабувати незалежно один від одного, що робить систему більш ефективною.

Переваги мікросервісної архітектури

— гнучке розширення функціоналу – додавання нових сервісів не потребує змін у всій систем;

— простота оновлення – кожен мікросервіс оновлюється незалежно, що зменшує ризики при розгортанні нових версій;

— підвищена відмовостійкість – якщо один сервіс виходить з ладу, решта системи продовжує працювати.

Недоліки мікросервісної архітектури

— складність розробки та тестування – необхідно враховувати взаємодію між сервісами, що ускладнює налагодження та тестування;

— проблеми з транзакціями – підтримка узгодженості даних між сервісами є складним завданням, яке вимагає використання розподілених транзакцій;

— збільшені витрати на інфраструктуру – кожен сервіс потребує окремого середовища виконання, що може збільшити використання ресурсів.

Вибір підходу для конкретного проєкту

Монолітна архітектура є хорошим вибором для невеликих проєктів, які не потребують високої масштабованості. Вона дозволяє швидко розпочати розробку та забезпечує ефективну взаємодію між компонентами.

Мікросервісна архітектура краще підходить для великих та складних систем, які потребують гнучкого масштабування та можливості оновлення окремих компонентів без переривання роботи всього додатку.

Для соціальних мереж та великих веб-платформ зазвичай використовується мікросервісний підхід, оскільки такі системи мають велику кількість функцій, які постійно розширюються. Проте на початкових етапах розробки нових продуктів часто використовується монолітний підхід для швидкого виведення продукту на ринок у форматі MVP (Minimum Viable Product). Такий підхід дозволяє швидко запустити базову версію додатку та поступово розширювати його функціональність. У міру зростання навантаження та складності система може бути поступово розділена на мікросервіси.

Таким чином, вибір між монолітом та мікросервісами залежить від масштабів проєкту, вимог до гнучкості, часу розробки та очікуваного навантаження на систему.

Однак, коли система зростає, виникає необхідність у більш гнучких і масштабованих рішеннях. У таких випадках доцільно розглядати перехід до мікросервісної архітектури. Вона дозволяє розробляти, тестувати, розгортати та підтримувати окремі компоненти незалежно один від одного, що значно спрощує масштабування та розширення функціональності.

З іншого боку, розгортання мікросервісів вимагає додаткових ресурсів та складної інфраструктури для управління взаємодією між сервісами, що

ускладнює адміністрування системи. Тому вибір між монолітною та мікросервісною архітектурою залежить від конкретних вимог проєкту, його складності, прогнозованого навантаження та довгострокових планів розвитку.

Якщо потрібно швидко запустити продукт із базовими функціями, монолітна архітектура може стати кращим варіантом, оскільки вона дозволяє швидко розробляти й тестувати систему. Проте, коли кількість користувачів зростає, а вимоги до продуктивності стають вищими, варто розглянути перехід до мікросервісів.

У випадку проєктування програмної системи для тестування та моніторингу якості інтернет-послуг вибір архітектури має ключове значення. З одного боку, монолітний підхід дозволить швидше реалізувати основний функціонал та налагодити роботу системи. З іншого боку, мікросервісна архітектура забезпечить гнучкість, можливість масштабування та ефективне управління окремими модулями. Оптимальним рішенням може бути використання гібридного підходу, де основні компоненти спочатку розробляються у монолітному вигляді, а згодом, у міру зростання навантаження та складності, поступово розділяються на окремі сервіси.

Загалом, під час розробки програмних систем критично важливо враховувати не лише поточні потреби, а й перспективи розвитку проєкту. Використання правильного підходу до архітектури серверної частини може значно вплинути на стабільність, продуктивність та зручність у подальшій підтримці й розширенні системи.

2.2 Детальне проектування

Під час проектування програмної системи було розглянуто два основні типи веб-служб: Process-Based (на основі процесів) та Event-Based (на основі подій).

У веб-службах на основі процесів кожен запит обробляється в окремому потоці або процесі, що потребує виділення серверних ресурсів для кожного потоку. Такі веб-сервери, як IIS або Apache, виділяють ресурси для обробки запитів навіть у стані очікування, що знижує ефективність роботи під високим навантаженням. Це обмеження робить їх менш придатними для систем із великою кількістю одночасних користувачів, наприклад, соціальних платформ або сервісів реального часу.

На противагу цьому, веб-служби на основі подій використовують асинхронну модель обробки запитів, де кожен запит ставиться у чергу подій (Event Queue), а спеціальний "loop-event-endless" цикл забезпечує постійний контроль за їх станом. Це дозволяє максимально ефективно використовувати ресурси сервера, не блокуючи потоки, і паралельно обробляти велику кількість запитів. До таких рішень належать Node.js та Nginx. Враховуючи вимоги, у даній системі було використано веб-сервер Node.js.

Для реалізації клієнтської частини (фронтенду) було обрано Vue.js — прогресивний JavaScript-фреймворк, який вирізняється своєю легкістю, простотою інтеграції та чудовою реактивністю. Vue.js дозволяє швидко створювати динамічні інтерфейси користувача, завдяки декларативному підходу до опису UI-компонентів та потужній екосистемі (Vue Router, Pinia/Vuex, Vite). Крім того, Vue.js має низький поріг входу, добре документований API та активну спільноту, що робить його ідеальним вибором для створення адаптивного і масштабованого інтерфейсу користувача.

Для реалізації серверної частини було використано мову програмування Java у поєднанні з фреймворком Spring Boot. Java є строго типізованою об'єктно-орієнтованою мовою програмування, що гарантує високу надійність, безпечність та масштабованість рішень. Spring Boot дозволяє швидко розгортати мікросервіси або монолітні застосунки з мінімальною кількістю конфігурацій, надаючи вбудовану підтримку REST API, роботу з базами даних (через Spring Data JPA), безпеку (Spring Security) та багато іншого. Завдяки розвинутій екосистемі, активному розвитку проєкту та широкій підтримці,

Spring Boot є одним із найкращих рішень для створення сучасних backend-систем.

Архітектура програмної системи реалізована згідно з принципами монолітної архітектури. Вона включає три взаємопов’язані рівні:

– Data Layer — шар доступу до даних, який відповідає за збереження, отримання та модифікацію даних у базі. Тут реалізуються сутності (Entity-класи), репозиторії та контексти даних, що взаємодіють з СУБД (наприклад, PostgreSQL або MySQL);

– Service Layer — шар бізнес-логіки. Тут реалізуються основні сервіси системи, що виконують операції CRUD (Create, Read, Update, Delete), обробляють правила бізнес-процесів та взаємодіють із шаром даних. У цьому шарі також використовуються шаблони проєктування, такі як Adapter, Command або Facade, для спрощення взаємодії між компонентами;

– Presentation Layer — клієнтська частина системи (реалізована на Vue.js), що спілкується із сервером через REST API, отримує оброблені дані та відображає їх користувачу. Presentation Layer забезпечує інтерактивність, зворотний зв’язок у реальному часі (наприклад, через WebSocket або SSE) та високу продуктивність.

Таким чином, усі шари системи мають чітко визначені функції:

– Data Layer – зберігає моделі предметної області та керує з’єднанням із базою даних;

– Service Layer – реалізує логіку додатку та координує доступ до даних;

– Presentation Layer – забезпечує інтерактивну роботу з кінцевим користувачем та виконує роль незалежного фронтенду, що може розгортатися окремо.

Цей підхід дозволяє досягти гнучкої масштабованості, кращого розділення відповідальностей та високої підтримуваності програмної системи, спрощуючи майбутнє тестування, супровід та розширення функціоналу2.4 Аналіз та вибір технологій і методів реалізації застосунку

2.2.1. Аналіз та вибір типу бази даних

Усі програмні системи, що забезпечують моніторинг мережевих ресурсів та якість інтернет-послуг, потребують збереження великого обсягу даних: інформації про підключення, логів подій, технічних метрик пристроїв та результатів діагностики. Надійне, швидке та гнучке зберігання такої інформації є критичним фактором для ефективної роботи системи. Тому на етапі проєктування важливим завданням є правильний вибір типу бази даних, що дозволить досягти балансу між продуктивністю, масштабованістю та підтримкою.

Сучасні системи керування базами даних умовно поділяються на три основні групи: реляційні БД, NoSQL-сховища та комбіновані підходи.

Реляційні бази даних

Реляційні СКБД є класичним і перевіреним рішенням. Усі дані організовано у вигляді таблиць із суворо визначеними схемами, що дає змогу чітко структурувати інформацію про мережеві пристрої, абонентів, записи журналів тощо. Кожен запис ідентифікується унікальним первинним ключем, що забезпечує точність і зв'язність даних.

Основними перевагами реляційних БД є:

– підтримка ACID-властивостей (атомарність, узгодженість, ізольованість, надійність);

– добре розвинений механізм індексування та запитів (SQL);

– можливість реалізації складних зв’язків між сутностями (наприклад, «абонент – комутатор», «пристрій – порт»);

– велика кількість стабільних і перевірених у промисловому середовищі рішень: PostgreSQL, MySQL, Oracle, MS SQL Server тощо.

Недоліком реляційних БД є зниження продуктивності при роботі з великою кількістю швидкозмінних або нестабільно структурованих даних (наприклад, телеметрія або тимчасові логи).

З огляду на те, що система тестування та моніторингу має чітко визначені сутності (абонент, пристрій, комутатор, лог-подія) — реляційна модель даних є основним вибором.

NoSQL-сховища

Для збереження високочастотних та неструктурованих даних, таких як телеметрія з мережевого обладнання, потоки логів чи кеші, доцільним є використання NoSQL-рішень у зв’язці з основною БД.

Сховища ключ–значення (Key-Value)

Ці бази даних ідеально підходять для кешування часто запитуваних даних (наприклад, поточний статус пристрою, останній результат пінгу або трасування). Вони забезпечують дуже високу швидкодію. Приклади: Redis, Memcached.

Документо-орієнтовані БД

У системах, де може виникати потреба зберігати гнучкі, варіативні за структурою об'єкти (наприклад, необов’язкові параметри пристроїв чи JSON-результати тестів), доцільно використовувати MongoDB або CouchDB. Проте у даному проєкті структура даних є стабільною, тому потреба у таких рішеннях обмежена.

Графові БД

У контексті мережевого моніторингу, графові БД можуть ефективно моделювати взаємозв’язки між пристроями (наприклад, топології мережі, зв’язки між вузлами та комутаторами). Проте для MVP-системи, орієнтованої на основні функції діагностики та контролю, впровадження графових БД є надмірним і ускладнює підтримку.

Колонкові БД

Бази даних колонкового типу (наприклад, Cassandra, ScyllaDB) можуть бути корисні при зберіганні великих обсягів телеметрії та логів. Проте їхня складність у реалізації фільтрації та обробці видалень робить їх менш зручними для MVP-реалізації.

З урахуванням вимог до системи моніторингу інтернет-послуг – стабільна структура даних, важливість цілісності, велика кількість запитів до системи – основним типом БД для реалізації обрано реляційну СКБД (наприклад, PostgreSQL або MySQL).

Додатково для покращення швидкодії й зменшення навантаження на основну базу використовуються сховища типу ключ-значення (наприклад, Redis) для кешування результатів тестів, тимчасових даних та метрик.

Такий підхід дозволяє забезпечити:

– надійне зберігання критичних даних;

– швидкий доступ до часто використовуваної інформації;

– гнучкість і масштабованість системи у майбутньому.

2.2.2 Проектування модульної архітектури

Після вибору технологічного стеку та системи керування базами даних було виконано проєктування архітектури програмної системи тестування та моніторингу якості інтернет-послуг. Система реалізована у вигляді монолітного багаторівневого вебзастосунку, структура якого поділена на модулі відповідно до логічних компонентів та функціональних обов’язків.

Центральним модулем системи є Application, який належить до презентаційного шару. У ньому зосереджено конфігурації, контролери для обробки HTTP-запитів, класи ініціалізації програми, а також логіку маршрутизації та з’єднання з клієнтською частиною, реалізованою на Vue.js. Саме цей модуль відповідає за запуск застосунку, обробку запитів REST API та координацію між іншими модулями.

Контрактний модуль (Contracts) виступає в ролі інтерфейсного шару між модулями. У ньому оголошено типи, структури даних (DTO) та загальні інтерфейси сервісів, що реалізуються в інших частинах системи. Завдяки цьому

досягається низький рівень зв’язування між компонентами, що відповідає принципам чистої архітектури.

Модуль Entities відповідає за зберігання та логічне структурування сутностей, які відповідають основним об'єктам предметної області: вимірювання швидкості, пінг, втрати пакетів, сеанси тестування, користувачі. Кожна сутність має власний репозиторій, що забезпечує абстрагований доступ до даних. У якості бази даних використано PostgreSQL, а для об’єктно-реляційного відображення застосовано JPA (Java Persistence API) через Spring Data. Внутрішні DTO дозволяють гнучко адаптувати запити під конкретні сценарії використання API без надлишкових даних.

Шар сервісів (Services) реалізує бізнес-логіку. Саме тут визначено поведінку системи при запуску вимірювання, валідацію параметрів, обробку результатів тестів та генерацію відповідей API. Усі сервіси зареєстровані через механізм Dependency Injection, що забезпечує гнучкість, тестованість і модульність.

Для роботи з даними реалізовано два підходи: шаблон Repository використовується для стандартних CRUD-операцій, тоді як шаблон Query Object — для реалізації складних запитів, зокрема фільтрації історії вимірювань за діапазоном дат, географічним регіоном або конкретним провайдером. Таке розділення дозволяє покрити широкий спектр функціональних вимог, не ускладнюючи основний кодовий простір.

Окремий модуль Utilities містить допоміжні компоненти: сервіси роботи з мережею, утиліти перевірки доступності ресурсів, обгортки над інструментами тестування (наприклад, запуск iperf або аналіз ping/traceroute). Також реалізовано механізм логування та надсилання сповіщень користувачу про завершення тесту або помилки в роботі системи.

Інфраструктурні аспекти (маршрутизація, TLS, балансування навантаження) обслуговуються через проксі-сервер Envoy, який замінює традиційний Nginx, надаючи розширені можливості для динамічного конфігурування та збору телеметрії.

Клієнтська частина розгортається окремо у вигляді SPA на основі Vue.js та взаємодіє з бекендом через REST API. Це дозволяє легко масштабувати інтерфейс, повторно використовувати компоненти та забезпечувати зручний користувацький досвід.

Загальна архітектура системи поєднує переваги строгої структуризації та гнучкості, забезпечуючи простоту розширення функціональності, підтримку багатокористувацької взаємодії, інтеграцію з зовнішніми сервісами моніторингу та адаптацію під різні сценарії використання (наприклад, для приватних мереж або операторських середовищ). Завдяки багаторівневому розподілу відповідальностей досягається надійність, тестованість і ефективність системи в умовах реального навантаження.

У майбутньому архітектура системи дозволяє інтегрувати нові механізми тестування, зокрема WebRTC-моніторинг або підтримку мобільних пристроїв. Це відкриває перспективи масштабування та комерційного розгортання сервісу.

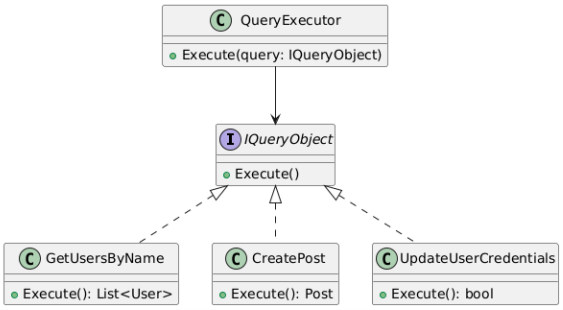


Рисунок 2.1 - Діаграма розширеного патерну «Об’єкт запиту»

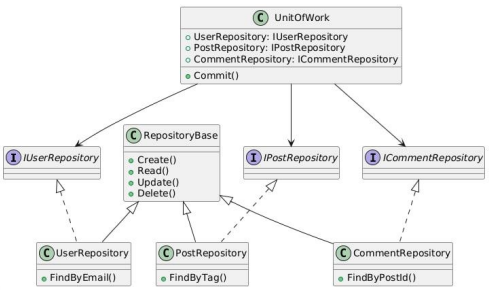


Рисунок 2.2 - Діаграма патерну «Репозиторій»

2.2.3. Проектування функціональної архітектури

Функціональна архітектура програмної системи відрізняється від модульної тим, що фокусується на проектуванні функціональних одиниць системи, орієнтованих на досягнення конкретних бізнес-цілей. Один функціональний модуль може охоплювати декілька логічних компонентів, що спільно виконують комплексну задачу.

Функціональна архітектура – це структурований опис функціональних можливостей системи, побудований з урахуванням технічних, бізнес-вимог, ієрархії функцій та їх взаємозалежностей у компонентах. Вона розробляється відповідно до функціональних вимог, що описуються сценаріями використання (use cases).

Для програмної системи тестування та моніторингу якості інтернет-послуг було виділено основні функціональні одиниці:

Автентифікація та авторизація

Система передбачає розмежування рівнів доступу користувачів (адміністратори, оператори моніторингу, тестувальники, зовнішні клієнти) з використанням ролей. Для забезпечення безпеки реалізовано механізм автентифікації з використанням JWT (JSON Web Token). Пара токенів (access token і refresh token) дозволяє забезпечити безперервний доступ до захищених ресурсів із можливістю безпечного оновлення токенів без повторної автентифікації.

Refresh-токени зберігаються в Redis із обмеженим часом життя та механізмом інвалідації. Таким чином досягається баланс між безпекою та зручністю використання.

Реєстрація та верифікація

Перед початком використання системи користувач проходить процедуру реєстрації. Щоб уникнути навантаження на базу даних, під час верифікації користувача його дані зберігаються тимчасово в Redis. Після підтвердження

електронної пошти за допомогою коду доступу, дані записуються до основної бази даних. Це дозволяє уникнути непотрібного зберігання неактивних акаунтів.

Створення та запуск тестових сесій

Користувач з відповідними правами може створювати тестові сесії, що містять параметри (тип тесту, IP-адресу, протокол, кількість повторів, періодичність, тощо). При цьому формується черга запитів, що зберігається у кеш-сховищі Redis для подальшої обробки агентами тестування. Діаграма створення тестової сесії представлена на рисунку Ж.1.

Збір результатів тестування та агрегація

Результати тестів надходять у систему асинхронно. Для уникнення надмірного навантаження на БД, спочатку дані збираються в Redis, а потім періодично агрегуються та зберігаються у базу даних групами (batch-запити). Цей підхід знижує кількість транзакцій та підвищує продуктивність.

Моделювання навантаження

Один із функціональних блоків відповідає за генерацію трафіку з визначеними параметрами (кількість з’єднань, розмір запитів тощо). Параметри формуються в інтерфейсі користувача та передаються агентам навантаження через API. Агентами виступають окремі мікросервіси, що масштабуються горизонтально.

Кешування популярних метрик

Показники, які часто запитуються (наприклад, середній час відповіді, втрати пакетів, доступність ресурсу), кешуються за стратегією Cache-Aside. Це дозволяє знизити навантаження на базу при візуалізації великої кількості статистики в режимі реального часу.

Стратегія асинхронного оновлення

Оновлення агрегованих даних відбувається за допомогою окремого Scheduler (планувальника завдань), реалізованого на основі Quartz. Це дозволяє системі виконувати накопичення і збереження метрик у базу даних через визначені інтервали (наприклад, кожні 5 хвилин), уникаючи втрат даних і не блокуючи основні потоки обробки.

Видалення та архівація старих даних

З метою оптимізації об’єму бази даних реалізовано функціональність архівації та видалення старих результатів тестів, що не використовуються у звітах або аналітиці. Видалення відбувається автоматично на основі політики зберігання, з можливістю налаштування інтервалу (наприклад, 6 місяців).

Робота з інцидентами

Функціональна архітектура також включає блок управління інцидентами – автоматичне створення події у разі виявлення відхилень від встановлених нормативів (наприклад, надмірно довгий ping або втрата доступу). Ці інциденти передаються через окремий канал повідомлень (наприклад, Kafka або Redis Pub/Sub) до системи оповіщення або панелі адміністратора.

2.3 Проектування інтерфейсу користувача

Проєктування інтерфейсу користувача системи тестування та моніторингу якості інтернет-послуг охоплює такі ключові етапи:

- розробка сценаріїв використання;
- побудова загальної структури інтерфейсу;
- створення прототипів екранів та модальних вікон;
- визначення стилістики;
- вибір та реалізація дизайн-концепції;
- оформлення екранів та додавання анімаційних елементів.

У межах цього проєкту сценарії використання були описані у першому розділі, тому основну увагу приділено наступним етапам – побудові структури інтерфейсу та візуальному оформленню системи.

Структура інтерфейсу

На основі сценаріїв використання розроблено логічну структуру інтерфейсу. Визначено основні сторінки (розділи) та модальні вікна, що

відповідають функціональним можливостям системи. Серед основних структурних елементів:

– головна сторінка – панель з результатами тестування та ключовими показниками;

– аналітика – розділ з візуалізацією даних у вигляді графіків;

– конфігурація системи – сторінка для налаштування параметрів тестування;

– історія тестів – архів виконаних перевірок;

– модальні вікна – для запуску нових тестів, перегляду деталей, авторизації та підтвердження дій;

– адміністративна панель – інтерфейс керування користувачами, ролями та скаргами.

Загальний макет інтерфейсу передбачає використання лівої навігаційної панелі та верхньої панелі керування. Ця структура забезпечує швидкий доступ до ключових функцій та логічну ієрархію взаємодії.

Прототипи екранів та модальних вікон

Кожен розділ інтерфейсу має окремий прототип, що описує розміщення елементів, таких як:

– таблиці результатів;

– фільтри для сортування тестів;

– панелі навігації;

– інтерактивні графіки;

– кнопки запуску, редагування, видалення тестів тощо.

Модальні вікна використовуються для запуску нового тестування, перегляду детальної інформації по окремому запису, редагування параметрів конфігурації, а також для дій з обліковим записом (авторизація, зміна паролю, підтвердження видалення тощо).

Стилістика та дизайн-концепція

Для стилістичного оформлення обрано концепцію Material Design, яка відповідає сучасним вимогам до UX/UI. Ця концепція дозволяє створити

зрозумілий, чистий і функціональний інтерфейс, що є інтуїтивно зрозумілим для користувача.

Використані такі стилістичні елементи:

– мінімалістичні кнопки з чітким виділенням станів (активний, неактивний, помилка);

– адаптивні кольорові схеми з урахуванням потреб користувачів із порушенням зору (наприклад, дальтонізм);

– стандартні компоненти Material Design – діалоги, меню, сповіщення, форми введення даних;

– динамічні анімації переходів, що не перевантажують користувача.

Основні розділи інтерфейсу

Головна сторінка

Відображає зведену таблицю результатів останніх тестів. Передбачено можливість сортування за часом виконання, статусом, типом тесту та іншими параметрами. У верхній частині сторінки розташовано поле пошуку, панель фільтрів та кнопки для запуску нового тесту. Результати представлені у вигляді карток або таблиць

Аналітика

Містить графічну інформацію про стабільність інтернет-з’єднання – швидкість, затримку, втрату пакетів тощо. Дані можна фільтрувати за періодом часу, типом тесту чи IP-адресою. Також доступна функція порівняння з попередніми результатами

Модальні вікна

Використовуються для взаємодії з окремими функціями, наприклад:

– запуск нового тесту з вибором типу, інтервалу, серверу;

– деталі окремого результату з логами, показниками, рекомендаціями;

– редагування налаштувань користувача;

– авторизація, підтвердження видалення або вихід з системи (рис. 3, 4).

Конфігурація тестів

Доступна авторизованим користувачам. Дозволяє вказувати:

– інтервали між тестами;

– IP-адреси для моніторингу;

– порогові значення якості інтернету;

– режими перевірки (ручний, автоматичний).

Історія тестів

Відображає усі виконані перевірки з можливістю пошуку, сортування та експорту у формат CSV/JSON. Також доступна функція повторного запуску тесту з тими ж параметрами.

– Налаштування облікового запису

Користувач може змінювати свої дані, переглядати статистику використання системи, видаляти або деактивовувати обліковий запис. У разі натискання кнопки видалення облікового запису – відкривається діалогове вікно з підтвердженням дії.

Адміністративна панель

Доступна тільки для модераторів і адміністраторів після авторизації з відповідною роллю. У цьому розділі реалізовано:

— перегляд скарг на користувачів або результати тестів;

— блокування користувачів у разі порушення правил;

— керування списком дозволених IP-адрес для моніторингу;

— управління ролями та привілеями.

На навігаційній панелі при вході адміністратора з’являється додатковий пункт «Адмін-панель».

Поведінка неавторизованого користувача

Для неавторизованих користувачів доступна обмежена функціональність. Вони можуть переглядати результати останніх тестів, але не можуть запускати нові перевірки, змінювати конфігурацію чи переглядати аналітику. При спробі скористатися цими функціями з’являється модальне вікно авторизації.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

3.1 Реалізація вебзастосунку

У розділі 2 зазначалося, що при розробці вебзастосунків з використанням фреймворку Vue.js основними будівельними блоками інтерфейсу є компоненти. Кожен компонент реалізує окрему частину функціональності, і вся вебсторінка складається з набору таких компонентів, що організовані в ієрархічну структуру.

У рамках вебверсії системи тестування та моніторингу якості інтернет-послуг компоненти відповідають за візуалізацію даних тестування, запуск нових перевірок, перегляд аналітики, керування користувачами тощо.

Типи компонентів у системі

У проєкті реалізовано такі компоненти:

- Sidebar: бічна панель з меню навігації;
- Navbar: — верхня панель з заголовком і кнопками;
- Footer: — нижній колонтитул з інформацією про систему;
- TestResultsTable:— таблиця з результатами перевірок;
- TestDetailsModal: — модальне вікно з деталями тесту;
- StatisticsChart: графіки з аналітичними показниками;
- RunTestForm: форма для запуску нового тесту;
- LoginForm: форма автентифікації користувача;
- SettingsPanel: панель налаштувань параметрів системи;
- RouterView: контейнер для виводу маршрутизованих сторінок;
- RouterLink: елементи посилань між сторінками;
- Breadcrumbs: індикатор поточного шляху в структурі сторінок;
- AuthGuard: захист маршрутів на основі автентифікації;
- UserRoleCheck: перевірка прав доступу;
- Notification: повідомлення про події (успіх, помилка, сповіщення);

– Modal, DialogConfirm, PopupAlert: для діалогів підтвердження, попереджень тощо.

Ці компоненти організовані у вигляді дерева: базові компоненти (наприклад, Layout) містять в собі складніші компоненти (наприклад, Sidebar, TestResultsTable, RouterView), що відповідають за відображення динамічного вмісту залежно від маршруту користувача.

Компоненти у Vue.js можуть бути двох типів:

Stateless-компоненти — не зберігають стан, отримують вхідні дані через props, не викликають змін інтерфейсу. Застосовуються для виведення статичного або умовно-статичного вмісту (заголовки, індикатори, прості кнопки).

Stateful-компоненти — містять внутрішній стан (data, computed, methods), реагують на зміну даних, автоматично оновлюються. Використовуються для інтерактивних елементів (форми, таблиці, графіки).

Таблиця 3.1

Параметр	Stateless компонент	Stateful компонент
Зміна стану	Не має внутрішнього стану	Має внутрішній стан data
Метод оновлення	Рендериться один раз	Реактивно оновлюється при зміні стану
Приклад використання	Заголовки, кнопки без логіки	Форми, фільтри, таблиці, графіки
Продуктивність	Вища продуктивність, простіші в тестуванні	Може бути менш продуктивним через оновлення

Завдяки реактивній системі Vue зміни в data, computed, watch автоматично призводять до оновлення DOM без потреби в ручному ререндері компонентів.

Для забезпечення масштабованості, модульності та легкості супроводу у вебзастосунку реалізовано архітектурний підхід Clean Architecture. Його суть полягає в поділі коду на шари за функціональністю з обмеженням залежностей між ними.

Шари Clean Architecture у системі:

- Presentation (Презентаційний шар): складається з компонентів Vue.js. Відповідає за інтерфейс користувача, взаємодію, події, візуалізацію. Приклад: TestResultsTable, RunTestForm, StatisticsChart.
- Application (Шар прикладної логіки): реалізує сценарії користування, контроль доступу, маршрутизацію запитів. Взаємодіє з Vuex-станом, сервісами API. Приклад: планування тесту після заповнення форми, обробка помилок, логування.
- Domain (Доменний шар): описує бізнес-логіку: типи тестів, параметри вимірювання якості інтернету, критерії оцінювання. Незалежний від UI або конкретного фреймворку. Приклад: моделі TestResult, QualityMetric, сервіси обробки результатів.
- Infrastructure (Інфраструктурний шар): реалізований на Spring Boot (Java). Забезпечує доступ до бази даних, виконання тестів (через Speedtest CLI, ping, traceroute), реалізацію REST API. Presentation шар взаємодіє з ним через HTTP-запити.

Переваги застосування Clean Architecture:

- можливість легко змінювати технології (заміна бібліотеки побудови графіків, зміна формату API);
- повторне використання логіки (наприклад, правила валідації тесту або розрахунок середніх значень);
- покращене тестування завдяки ізоляції логіки від візуального шару;
- гнучка структура для масштабування (додавання нових типів тестів, нових ролей тощо).

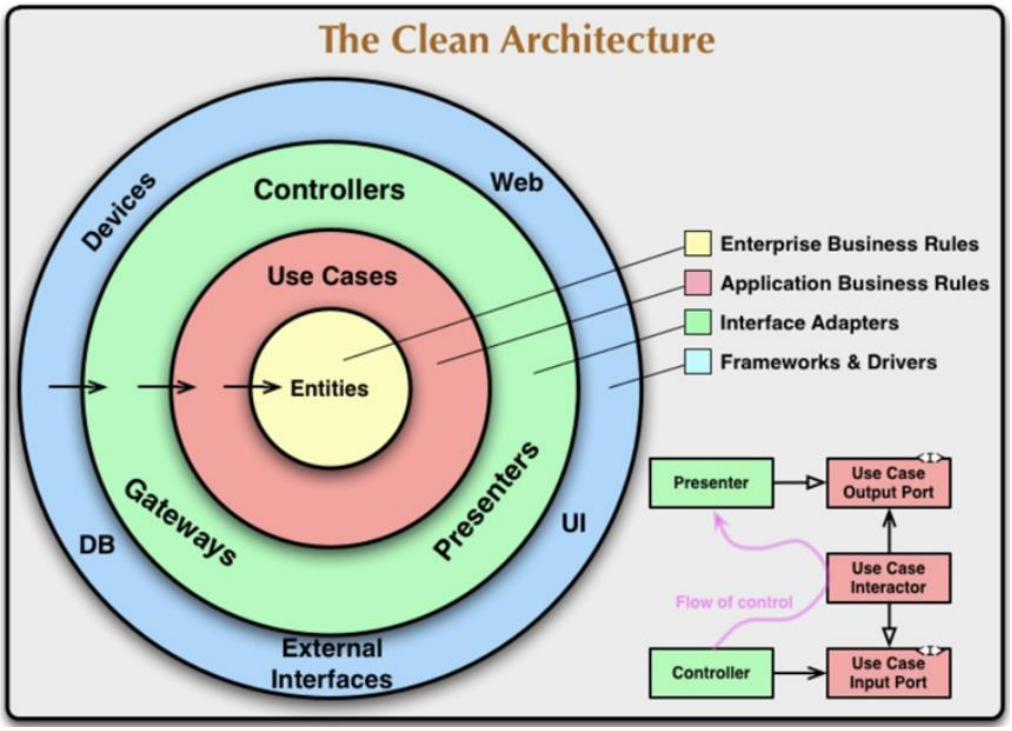


Рисунок 3.1 – Clean Architecture

Хоч і було обрано Serverless-архітектуру, вона не суперечить принципам Clean Architecture, а навпаки — може ефективно з ними поєднуватись, особливо у контексті структурування коду та ізоляції компонентів.

Clean Architecture визначає, як має бути організований код у застосунку, незалежно від середовища його виконання, тоді як Serverless-архітектура визначає, де і як цей код буде запускатись. Наприклад, у розробленій системі можливо побудувати вебзастосунок відповідно до Clean Architecture, розділяючи бізнес-логіку, інтерфейс користувача та інфраструктурні залежності, а для розгортання та масштабування окремих частин системи використати серверлес-середовище, таке як AWS Lambda або Google Cloud Functions.

Це дозволяє зберегти чисту та підтримувану архітектуру коду, водночас отримуючи переваги serverless-платформ — зокрема автоматичне масштабування, оплату за використання та відсутність потреби в управлінні серверами.

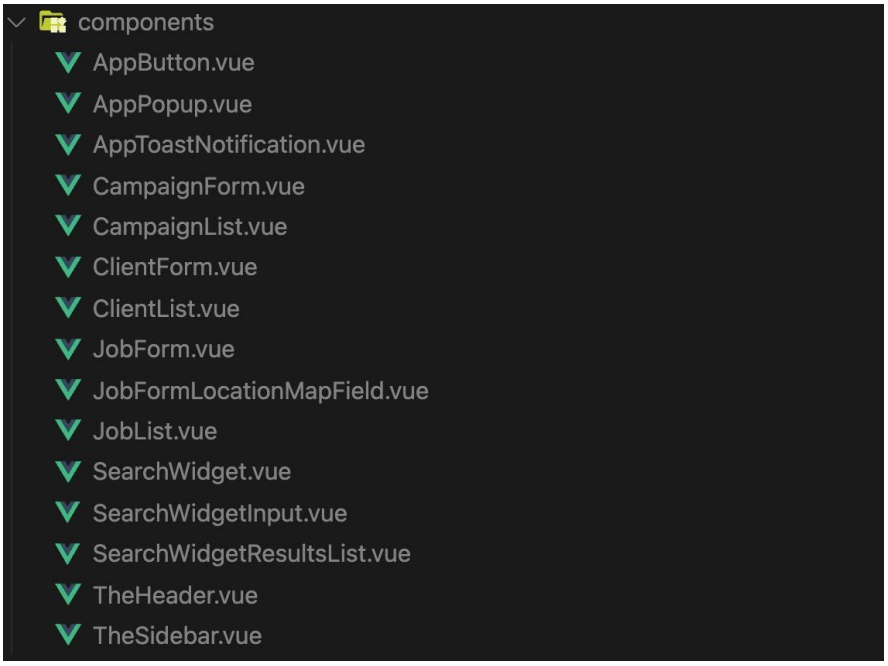


Рисунок 3.2 – Структура папок

Хоч система тестування та моніторингу якості інтернет-послуг орієнтована на обробку великої кількості даних і взаємодію з зовнішніми API (наприклад, для запуску speedtest), це не суперечить принципам Clean Architecture, а навпаки — такі системи особливо виграють від чіткої архітектурної організації. Clean Architecture визначає спосіб структурування коду з урахуванням розділення відповідальностей, що критично важливо у системах, де відбувається збір, збереження та аналітика даних про якість інтернет-з’єднання.

У реалізованій системі архітектура побудована за принципами Clean Architecture з чітким розмежуванням між інтерфейсом користувача, бізнес-логікою та інфраструктурною частиною. Це дозволяє змінювати або розширювати функціональність системи (наприклад, додати нові метрики чи інтерфейси перевірки) без ризику порушення існуючої логіки або дестабілізації програми.

Для розробки клієнтської частини було обрано фреймворк Vue.js. Основний програмний код зосереджений у папці src, яка організована за

модульним принципом. Кожна функціональна область (наприклад, модуль моніторингу) має власну структуру:

– components — багаторазові UI-компоненти (форми запуску тесту, діаграми швидкості тощо);

– utils — утиліти для форматування даних, валідації IP-адрес або вимірювань;

– services — логіка запитів до бекенду, обробка результатів, авторизація;

– features — основні функціональні блоки, такі як monitoring, results, user-settings, кожен з яких поділений на:

– data — джерела даних, DTO, запити;

– domain — бізнес-логіка модуля;

– presentation — інтерфейс користувача, управління станом (Pinia/Vuex).

У файлі main.ts здійснюється ініціалізація застосунку: підключення роутера, менеджера стану, глобальних стилів, налаштування аналітики та реєстрація сервісів. Саме тут реалізовано:

– перевірку доступності мережі перед запуском тестів;

– ініціалізацію конфігурацій користувача;

– підключення систем логування результатів.

Бекенд-система, реалізована на Spring Boot, дотримується структури, яка відповідає Clean Architecture:

– controller — обробка HTTP-запитів, REST API;

– service — бізнес-логіка перевірок, обробки результатів, авторизації;

– repository — взаємодія з базою даних (MongoDB або PostgreSQL);

– model/entity — опис структури збережених об’єктів (результати тестів, користувачі, конфігурації);

– config — параметри безпеки, CORS, DI, конфігурація API-доступу до сторонніх сервісів (наприклад, Speedtest CLI).

У контексті архітектури важливу роль відіграє патерн інверсії залежностей (Dependency Injection), реалізований у Spring Boot через анотації @Component, @Service, @Repository, @Autowired. Це дозволяє змінювати

реалізації компонентів без зміни їх споживачів — наприклад, легко замінити джерело даних (перейти з SQL на NoSQL або тимчасово підставити in-memory базу) або використати мок-репозиторій для юніт-тестування.

Приклад інтерфейсу репозиторію для збереження результатів перевірки якості інтернету:

```
public interface InternetQualityRepository {
    List<SpeedTestResult> fetchAllResultsByUser(String userId);
    SpeedTestResult fetchResultById(String userId, String resultId);
    SpeedTestResult saveResult(String userId, SpeedTestResult result);
    void deleteResult(String userId, String resultId);
}
```

І приклад реалізації через JPA:

```
@Service
public class DatabaseInternetQualityRepository implements
InternetQualityRepository {
    @Autowired
    private InternetQualityJpaRepository jpaRepository;

    @Override
    public List<SpeedTestResult> fetchAllResultsByUser(String userId) {
        return jpaRepository.findByUserId(userId);
    }

    @Override
    public SpeedTestResult fetchResultById(String userId, String resultId)
{
        return jpaRepository.findByUserIdAndId(userId, resultId)
            .orElseThrow(() -> new RuntimeException("Result not found"));
    }

    @Override
    public SpeedTestResult saveResult(String userId, SpeedTestResult
result) {
        result.setUserId(userId);
        return jpaRepository.save(result);
    }

    @Override
    public void deleteResult(String userId, String resultId) {
        jpaRepository.deleteByUserIdAndId(userId, resultId);
    }
}
```

Завдяки такому підходу система легко масштабується — наприклад, можна винести збереження великих логів або історії перевірок у окремий сервіс, не змінюючи основну бізнес-логіку. Clean Architecture також спрощує написання тестів, підтримку та розширення функціоналу, зокрема через

можливість впровадження нових типів перевірок (наприклад, тестування стабільності VPN або DNS-запитів).

У підсумку, завдяки структурованому підходу до архітектури та коду, вебзастосунок для моніторингу якості інтернет-послуг є масштабованим, підтримуваним і готовим до інтеграції з іншими сервісами — як внутрішніми, так і сторонніми.

3.2 Вимоги до технічних та програмних засобів

Розроблений вебсайт для перевірки якості інтернет-послуг призначений для використання на сучасних персональних комп'ютерах, ноутбуках, планшетах та мобільних пристроях, які мають сучасний браузер та стабільне інтернет-з'єднання. Для забезпечення стабільної та коректної роботи вебзастосунку необхідно дотримуватись наведених нижче мінімальних вимог до апаратного та програмного забезпечення.

Мінімальні апаратні вимоги:

- процесор: двоядерний з тактовою частотою не нижче 1.5 ГГц;
- оперативна пам'ять: не менше 2 ГБ;
- вільне місце на диску: щонайменше 200 МБ (для тимчасових файлів браузера);
- інтернет-з'єднання: стабільне підключення зі швидкістю не менше 1 Мбіт/с.

Програмне забезпечення:

- операційна система: Windows 10 або новіша, Linux (Ubuntu 20.04+), macOS 10.13+; також підтримуються сучасні версії Android та iOS;
- веб-браузер: останні стабільні версії Google Chrome, Mozilla Firefox, Microsoft Edge або Safari з підтримкою JavaScript та WebAssembly;
- додаткові компоненти: ввімкнений JavaScript, підтримка WebRTC та доступ до API браузера для перевірки параметрів мережі.

Додаткові вимоги:

– доступ до інтернету: безперервне інтернет-з’єднання є обов’язковим, оскільки усі функціональні можливості сайту (вимірювання затримки, швидкості завантаження/відвантаження тощо) потребують взаємодії з серверною частиною системи;

– дозволи браузера: дозвіл на використання WebRTC (для визначення IP-адреси), доступ до інформації про тип підключення (Network Information API), а також дозволи на виконання JavaScript-скриптів.

Система оптимізована для стабільної роботи у найпоширеніших середовищах користувачів, з урахуванням різних типів підключення та класів пристроїв. Завдяки використанню сучасних фреймворків та адаптивної верстки, вебзастосунок забезпечує коректне відображення й стабільну роботу як на стаціонарних комп’ютерах, так і на мобільних пристроях.

3.3 Тестування системи

Тестування є ключовим етапом життєвого циклу розробки вебзастосунків і забезпечує стабільну, безпечну та передбачувану роботу системи. Воно дозволяє виявити помилки як у логіці роботи клієнтської частини (інтерфейс), так і у взаємодії з сервером, базами даних чи сторонніми API. Залежно від цілей тестування застосовуються різні методи, зокрема:

– Unit-тести (модульне тестування): використовуються для перевірки окремих функцій, методів або класів без залежності від середовища чи зовнішніх ресурсів. Наприклад, у фронтенді можна тестувати функції обробки даних або валідації форм, а на бекенді — сервіси, обробники запитів, утиліти тощо;

– Integration-тести (інтеграційне тестування): охоплюють декілька компонентів або модулів, які працюють спільно. Наприклад, можна

протестувати, як фронтенд надсилає запити на бекенд і як сервер обробляє ці запити та відповідає;

– End-to-end (E2E) тести: емулюють дії кінцевого користувача у браузері. Такі тести перевіряють повноцінні сценарії взаємодії з системою: авторизацію, навігацію між сторінками, взаємодію з формами, надсилання даних на сервер тощо. Вони виконуються у реальному браузерному середовищі або в емуляторі (наприклад, за допомогою Puppeteer або Cypress);

– Візуальне тестування (Visual Testing): передбачає перевірку зовнішнього вигляду інтерфейсу. У випадку зміни дизайну чи стилів такі тести допомагають виявити небажані зміни у компонуванні, кольорах, шрифтах тощо (наприклад, з використанням бібліотек як Percy або Storybook з тестами);

– Тестування продуктивності (Performance Testing): оцінює, наскільки швидко та стабільно вебзастосунок реагує на запити користувача та навантаження (наприклад, через Lighthouse, Apache JMeter, k6 тощо);

Інструменти для тестування вебзастосунків:

– Jest: популярна бібліотека для unit- та integration-тестів JavaScript/TypeScript-коду. Широко використовується для перевірки логіки у Vue.js, React тощо. Дозволяє створювати ізольовані тести з моками;

– Vue Test Utils: офіційна бібліотека для тестування компонентів у Vue.js. Забезпечує зручний API для рендерингу компонентів, взаємодії з DOM, перевірки реакції на події, зміни стану тощо;

– Cypress: сучасний фреймворк для E2E тестування. Працює у реальному браузері, дозволяючи емулювати дії користувача, перевіряти навігацію, інтерфейс, обмін з сервером. Має потужний візуальний інтерфейс та підтримку скріншотів/відео;

– Postman / Insomnia: інструменти для ручного та автоматизованого тестування REST API. Дозволяють створювати сценарії перевірки, перевіряти правильність відповідей серверу, обробку помилок тощо;

– Mockito / WireMock (на бекенді): бібліотеки для мокування залежностей у Spring Boot. Дозволяють створювати підставні об’єкти (наприклад, сервіси, репозиторії) для ізольованого тестування окремих шарів системи;

– JUnit (на бекенді): стандартний фреймворк для unit-тестів у Java. Забезпечує створення повторюваних тестів для бізнес-логіки, контролерів, сервісів та DAO;

– Testcontainers: дозволяє запускати тимчасові контейнери (наприклад, баз даних чи сторонніх сервісів) для інтеграційних тестів у Java, що наближає тестове середовище до реального.

Завдяки використанню сучасних інструментів тестування на всіх рівнях (від юніт-тестів до інтеграційних та e2e-сценаріїв), вебзастосунок для перевірки якості інтернет-послуг забезпечує надійність, зручність користування та стабільність у роботі за різних умов.

3.3.1 Тестування вебзастосунку

Тестування вебзастосунку на ранніх етапах розробки найчастіше виконується за допомогою вбудованих емуляторів у браузерах або спеціалізованих інструментів для перевірки інтерфейсу та функціональності. Вони є незамінними для розробників і QA-фахівців, особливо під час перевірки адаптивності, логіки клієнтського коду та сценаріїв взаємодії.

До основних переваг використання емуляторів браузера належать:

– швидкість і зручність: завдяки інструментам розробника (DevTools), які надають популярні браузери (Google Chrome, Firefox, Edge), можна швидко перевірити зовнішній вигляд і поведінку вебзастосунку на різних типах пристроїв, розмірах екранів і орієнтаціях;

– можливість налагодження в режимі реального часу: DevTools дозволяють виводити консольні повідомлення, аналізувати мережеві запити

(XHR, fetch), переглядати DOM, профілювати продуктивність та виявляти помилки верстки або JavaScript-коду;

– тестування адаптивного дизайну: емулятори забезпечують перевірку, як вебінтерфейс відображається на смартфонах, планшетах і десктопах, що особливо важливо для системи, яка повинна бути доступною з різних типів пристроїв;

– інтеграція з автоматизованим тестуванням: браузерні середовища добре сумісні з інструментами для написання автоматизованих E2E-тестів, зокрема Cypress, Playwright, Selenium, що дозволяє запускати тестування у CI/CD-процесі.

Попри численні переваги, використання лише емуляторів не гарантує повну відповідність реальним умовам експлуатації. Наприклад:

– реальні пристрої можуть мати відмінності в обробці подій (особливо сенсорних), продуктивності, поведінці в умовах слабкого сигналу або нестабільного інтернету;

– нестандартні браузери або специфічні модифікації ОС можуть впливати на відображення та роботу JavaScript;

– деякі помилки (наприклад, пов’язані з кешуванням, поведінкою сервіс-воркерів чи безпекою) виявляються тільки на фізичних пристроях.

Оптимальним підходом є поєднання тестування у віртуальному середовищі (емулятор браузера, headless-браузери) з періодичним тестуванням на реальних пристроях і браузерах, що дозволяє забезпечити високу надійність та якість вебзастосунку в реальних умовах використання.

Для перевірки якості інтерфейсу користувача в системі моніторингу інтернет-послуг було реалізовано серію модульних, інтеграційних та UI-тестів. У таблиці 3.2 наведено перелік ключових тестів, які охоплюють елементи вебінтерфейсу, включаючи кастомні компоненти, інтерактивні віджети та сценарії взаємодії.

Таблиця 3.2 – Список тестів для інтерфейсу користувача

1	Форма входу	E2E / інтеграційний	Перевірка авторизації, валідації полів, обробки помилок сервера
2	Панель моніторингу	UI / snapshot	Відображення даних, реакція на зміну фільтрів, коректність оновлення
3	Кнопка запуску тесту	Unit / інтеграційний	Взаємодія з бекендом, реакція на натискання, зміна статусу
4	Картка результату тесту	Unit	Коректність обчислення часу відповіді, кольорові індикатори статусів
5	Навігація між сторінками	E2E	Симуляція переходу між вкладками, збереження стану сторінки
6	Адаптивність на мобільних	UI / емульований перегляд	Відображення компонентів на різних ширинах екрана

Аналогічно було розроблено тести і для бізнес-логіки застосунку. Це дозволяє ізольовано перевірити правильність функціонування ключових алгоритмів і обробки даних без прив’язки до інтерфейсу користувача. Тестування логіки охоплює взаємодію з API, обробку результатів, логіку обчислень, валідацію параметрів та реакцію на помилки з боку сервера чи мережі. Повний список тестів для логіки наведено у таблиці 3.3.

Таблиця 3.3 – Список тестів для бізнес-логіки

№	Назва	Тип тесту	Опис перевірки
1	Авторизація користувача	Unit/інтеграційний	Валідація введених даних, обробка відповідей API, збереження токена
2	Запуск тесту швидкості	Unit	Побудова запиту, перевірка параметрів, запуск таймера, обробка результату

Продовження таблиці 3.3

3	Обчислення статистики	Unit	Підрахунок середньої швидкості, затримки, виявлення втрат пакетів
4	Генерація звіту	Unit / snapshot	Побудова структури звіту, форматування даних, перевірка цілісності JSON-структури
5	Запит історії перевірок	Інтеграційний	Взаємодія з сервером, обробка пагінації, перетворення отриманих даних
6	Валідація параметрів тесту	Unit	Перевірка на допустимі значення, обробка граничних умов (порожні, некоректні поля)
7	Обробка помилок сервера	Unit/інтеграційний	Перевірка реакції на 4xx/5xx відповіді, вивід повідомлень, перезапуск запитів
8	Синхронізація з бекендом	Інтеграційний	Перевірка відправки результатів, відстеження статусу та підтвердження доставки
9	Управління сесією користувача	Unit	Перевірка терміну дії токена, автоматичне оновлення.

Ці тести дозволяють переконатися у стабільності ядра системи, виявити логічні помилки до розгортання, а також забезпечити належну роботу при зміні серверної логіки чи структури API.

3.3.2 Аналіз результатів тестування веб-застосунку

Для перевірки коректності реалізації функціоналу веб-застосунку програмної системи тестування та моніторингу якості інтернет-послуг було

проведено інтеграційне тестування ключових сценаріїв взаємодії користувача. Застосунок реалізовано відповідно до принципів Clean Architecture із чітким розмежуванням доменної логіки, інфраструктури та інтерфейсу користувача, що значно спрощує тестування і дозволяє забезпечити високу якість коду.

Тестування охоплювало такі основні функціональні сценарії:

– авторизація користувача та перевірка облікових даних через backend-сервіс Spring Boot;

– навігація між основними модулями застосунку: дашборд моніторингу, історія перевірок, налаштування профілю;

– запуск тесту швидкості інтернет-з'єднання та перевірка збереження результату;

– виведення звітів за попередні періоди;

– вихід користувача із системи та перевірка скидання сесії.

Опис логіки інтеграційного тестування:

– ініціалізація тестового середовища: запускається фронтенд Vue-застосунок з мок-сервером або тестовим інстансом бекенду (зазвичай із використанням Spring Boot Test або TestContainers);

– створення тестового користувача: через REST API реєструється новий користувач з унікальним email для уникнення конфліктів;

– авторизація: імітується вхід у систему з відповідним логіном і паролем;

– запуск моніторингу: ініціюється тест якості з'єднання, перевіряється коректність його виконання та отримання результату;

– перевірка історії: виконується запит до модуля історії перевірок, де перевіряється наявність щойно створеного результату;

– вихід: натискання кнопки «Вийти» має призвести до скидання сесії та повернення на екран авторизації;

Приклад інтеграційного тесту backend-сервісу з використанням Spring Boot:

```
@SpringBootTest
```

```
@AutoConfigureMockMvc
public class AuthControllerIntegrationTest {

    @Autowired
    private MockMvc mockMvc;

    @Test
    public void testUserRegistrationAndLoginFlow() throws Exception {
        String email = "test_" + System.currentTimeMillis() +
"@example.com";
        String password = "securePass123";

        // Реєстрація
        mockMvc.perform(post("/api/auth/register")
                .contentType(MediaType.APPLICATION_JSON)
                .content("""
                    {
                      "email": "%s",
                      "password": "%s",
                      "fullName": "Test User"
                    }
                """.formatted(email, password)))
                .andExpect(status().isOk());

        // Вхід
        mockMvc.perform(post("/api/auth/login")
                .contentType(MediaType.APPLICATION_JSON)
                .content("""
                    {
                      "email": "%s",
                      "password": "%s"
                    }
                """.formatted(email, password)))
                .andExpect(status().isOk())
```

```
                .andExpect(jsonPath("$.token").exists());
        }
    }
```

На стороні клієнта аналогічно тестувались основні маршрути Vue-застосунку за допомогою E2E-фреймворку Cypress. Було перевірено:

– рендеринг головної сторінки;

– коректність переходу між модулями (роутінг);

– реакцію на дії користувача (натискання, введення тексту, навігація);

– збереження та виведення результатів з API.

```
describe('Monitoring System E2E Test', () => {
  it('Login and perform speed test', () => {
    cy.visit('/login');
    cy.get('input[type="email"]').type('test@example.com');
    cy.get('input[type="password"]').type('securePass123');
    cy.get('button[type="submit"]').click();

    cy.url().should('include', '/dashboard');

    cy.get('#startTestButton').click();
    cy.contains('Результат перевірки').should('be.visible');

    cy.get('#logoutButton').click();
    cy.url().should('include', '/login');
  });
});
```

```
00:00 +0: Успішний логін користувача
[INFO] Initializing Firebase...
[INFO] Test user: login_1717320211452@example.com
[INFO] Registering new user...
[INFO] Registration complete.
[INFO] Entering email and password into login form...
[INFO] Submitting login form...
[INFO] Login successful. Navigated to HomeScreen.
00:10 +1: Navigate to settings and perform logout
[INFO] Opening ProfileScreen...
[INFO] Tapping settings button...
[INFO] Navigated to SettingsScreen.
[INFO] Tapping logout button...
[INFO] Navigated to LoginScreen after logout.
00:18 +2: All tests passed!
```

Рисунок 3.3 – Результат Firebase-проєкті

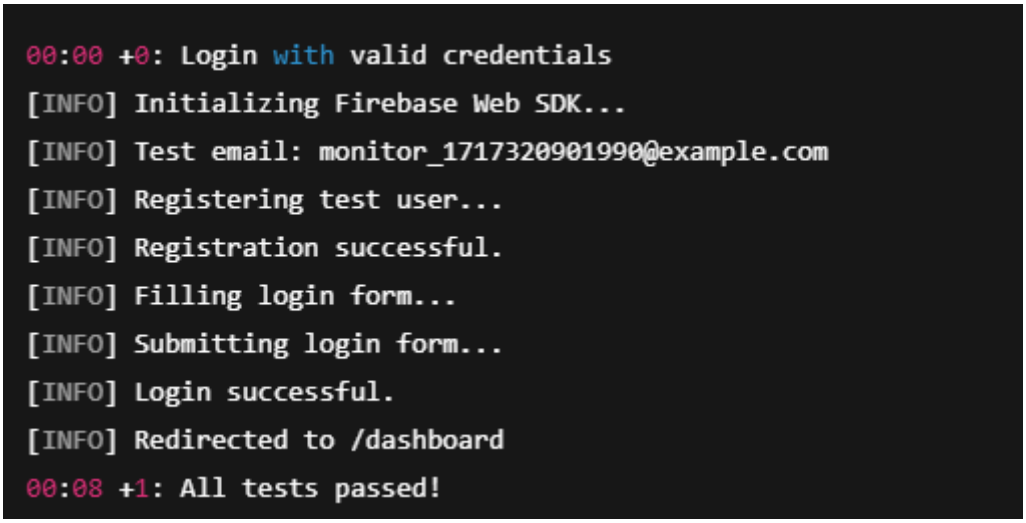


Рисунок 3.4 – Результат роботи базових дій користувача

Тестування показало, що:

– веб-застосунок коректно реалізує всі базові сценарії використання.

– після запуску перевірки швидкості, результати зберігаються на сервері та відображаються у списку історії.

– сеанси користувачів обробляються надійно: після виходу з системи доступ до даних обмежується.

– фронтенд правильно реагує на запити до API та виконує перенаправлення залежно від стану користувача.

Інтеграційне тестування підтвердило, що веб-застосунок є стабільним та надійним у виконанні основних операцій, зокрема авторизації, запуску перевірки якості інтернет-з'єднання, збереження історії та управління користувацькою сесією.

Для забезпечення комплексної перевірки якості програмної системи використовувались:

– юніт-тести для перевірки окремих функцій backend- і frontend-логіки;

– інтеграційні тести для перевірки взаємодії між модулями (наприклад, controller з service з repository у Spring Boot);

– end-to-end тести через Cypress для моделювання реальної поведінки користувача;

– навантажувальні тести, які виконувались на backend API з метою виявлення меж продуктивності під час масового виконання перевірок.

Таким чином, тестування підтвердило відповідність застосунку вимогам до якості, продуктивності та надійності, що є критично важливим для сервісу, який призначений для оцінки інтернет-з'єднання кінцевих користувачів.

3.4 Висновки програмної реалізації та тестування

У цьому розділі було реалізовано та протестовано веб-застосунок програмної системи для тестування та моніторингу якості інтернет-послуг, з урахуванням сучасних архітектурних підходів та технологічних рішень.

На основі принципів Clean Architecture було чітко відокремлено компоненти доменної логіки, інфраструктури та представлення. Це дало змогу досягти високої модульності системи, спростити супровід коду та забезпечити можливість масштабування в майбутньому.

Клієнтська частина (frontend) реалізована з використанням Vue.js, що забезпечило гнучкість та інтерактивність інтерфейсу. Серверна частина (backend) реалізована на Java Spring Boot, яка надала надійну платформу для реалізації REST API, взаємодії з базами даних та обробки користувацьких запитів. У якості проксі-сервісу замість Nginx застосовується Envoy Proxy, що дозволяє гнучко керувати маршрутизацією запитів і спрощує інтеграцію з іншими мікросервісами. Сформовано стійку архітектуру застосунку, що дозволяє незалежно тестувати та розвивати окремі компоненти. Забезпечено підтримку користувацької авторизації та управління сесіями. Реалізовано функції запуску перевірок швидкості інтернет-з'єднання, отримання звітів, відображення історії тестувань. Здійснено інтеграційне та E2E тестування, яке підтвердило стабільну роботу системи в межах основних сценаріїв користувача.

Під час тестування було перевірено: функціональність аутентифікації користувачів, правильність маршрутизації та переходів між інтерфейсними

модулями, коректну роботу запитів до backend API, відповідність очікуваної поведінки системи при введенні даних, запуску тестів та виході із системи.

Усі базові сценарії були успішно виконані, що свідчить про високий рівень готовності веб-застосунку до подальшого використання.

Таким чином, проведене програмне проєктування, реалізація і тестування підтверджують працездатність розробленої системи в умовах реального використання. Система є масштабованою, модульною, надійною і відповідає поставленим функціональним та нефункціональним вимогам.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було здійснено повномасштабну розробку веборієнтованої програмної системи тестування та моніторингу якості інтернет-послуг, що охоплює як фронтенд, так і бекенд частини, архітектурну організацію застосунку, проєктування моделі взаємодії з користувачем, налагодження мережевої інфраструктури, а також впровадження тестового середовища й проведення валідації реалізованої функціональності. Робота охопила всі ключові етапи створення програмного забезпечення — від збору й аналізу вимог до практичного впровадження функцій системи та оцінки її ефективності в умовах реального або наближеного до реального використання.

Проєктування архітектури системи від самого початку було зорієнтоване на досягнення модульності, гнучкості та здатності до масштабування. У якості фреймворку для фронтенду обрано Vue.js, що забезпечує високий рівень реактивності, продуктивну розробку UI-компонентів, а також зручне керування станом застосунку. Завдяки використанню компонентного підходу до структурування інтерфейсу, кожен елемент користувацького середовища можна незалежно вдосконалювати або замінювати без потреби в переробці загальної логіки. Це особливо важливо для програм, які можуть зазнавати частих змін у вигляді додавання нових функцій, оновлення дизайну або адаптації до інших платформ.

Для бекенд-реалізації застосовано Java Spring Boot — сучасний фреймворк, що дозволяє створювати масштабовані, безпечні й високопродуктивні RESTful-сервіси. У контексті даного проєкту серверна частина відповідає за обробку запитів на тестування швидкості інтернету, збереження й аналіз результатів, керування користувацькими сесіями, а також взаємодію з Firebase Authentication для забезпечення надійного та захищеного процесу автентифікації. Було також реалізовано механізм взаємодії з базою даних, що забезпечує збереження історичних даних вимірювань, що в

майбутньому може бути використано для візуалізації трендів або побудови аналітичних звітів.

Інтеграція з Firebase стала вагомим технологічним компонентом реалізованої системи. Firebase Authentication було використано як основний механізм керування користувачами, що гарантує безпеку зберігання облікових даних та надає додаткові функції, як-от автоматичне управління сесіями, валідацію даних, мультифакторну аутентифікацію та логування. Всі ці елементи істотно підвищують рівень довіри користувача до системи та зменшують ризик порушень безпеки, що особливо актуально у вебдодатках, де здійснюється передача приватної інформації або аналітичних даних.

З метою досягнення високої продуктивності й надійності взаємодії між мікросервісами та компонентами системи було реалізовано конфігурацію з використанням Envoy Proxy — високопродуктивного проксі-сервера з підтримкою сервіс-меш підходів. Його використання дозволяє виконувати інтелектуальну маршрутизацію запитів, балансування навантаження, повторні спроби виконання запитів при збої, а також збір аналітики по трафіку та помилках. Це підвищує загальну стійкість системи до навантаження та дозволяє підготувати її до подальшого масштабування — наприклад, при переході до мікросервісної архітектури.

Окрему роль у реалізації системи відіграло тестування. Було проведено розробку та запуск як модульних, так і інтеграційних тестів, що перевіряють коректність роботи ключових функцій, таких як автентифікація, запуск вимірювання швидкості з’єднання, навігація між сторінками, обробка помилок. У процесі тестування було змодельовано як позитивні, так і негативні сценарії поведінки, що дозволило виявити та усунути недоліки ще до впровадження системи у продуктивне середовище. Зокрема, було протестовано сценарії некоректного введення, повторної реєстрації, виведення даних у разі повільного або нестабільного з’єднання тощо. Автоматизовані тести, виконані за допомогою сучасних інструментів, таких як Postman для API, а також unit і e2e тестів для фронтенду, забезпечують надійність і повторюваність перевірок.

Результати тестування свідчать про стабільну роботу системи в заданих умовах експлуатації. Система успішно виконує функції вимірювання затримки (ping), швидкості завантаження (download) і вивантаження (upload), зберігає ці дані в структурованому форматі та відображає їх у візуальній формі для зручного аналізу користувачем. Також підтверджено правильність переходів між етапами тестування, коректне виведення повідомлень про помилки у разі втрати з’єднання або введення некоректних даних, а також збереження інформації в історії результатів.

У результаті проведеної роботи було створено повноцінний, функціональний і сучасний вебзастосунок, здатний надійно виконувати поставлені перед ним завдання та легко адаптуватися до нових викликів. Структура коду, архітектурна побудова, обрані фреймворки й бібліотеки відповідають актуальним стандартам і практикам розробки програмного забезпечення, що підвищує його життєздатність у довгостроковій перспективі. Реалізована система може бути інтегрована у більш масштабні платформи з моніторингу якості мережевих сервісів, а також використана як база для подальшого розширення функціональності — включаючи аналітичну панель, підтримку сповіщень, геолокаційну аналітику та адміністративний контроль.

Загалом, проєкт продемонстрував як технічну, так і організаційну доцільність розробки спеціалізованих вебзастосунків для моніторингу інтернет-послуг, а також підтвердив практичну ефективність обраних архітектурних та інженерних рішень. Отримані результати доводять, що застосування сучасного стеку технологій дозволяє реалізовувати складні інформаційні системи в межах обмеженого ресурсу часу та з дотриманням вимог до якості, продуктивності та зручності використання.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Глушаков В.В., Гуржій А.М., Дрозд О.О. Інженерія програмного забезпечення. Київ: ВНТУ, 2021. 276 с.

2. Бочкарьова І.М., Кадієвська О.О. Системи моніторингу в IT-інфраструктурах: навч. посібник. Харків: ХНУРЕ, 2020. – 164 с.

3. Sommerville I. Software Engineering (10th Edition). Pearson, 2016. – 816 p.

4. Bass L., Clements P., Kazman R. Software Architecture in Practice (3rd Edition). Addison-Wesley, 2012. – 624 p.

5. RFC 6419: Service Monitoring and Performance Metrics in IP Networks IETF, 2011. https://tools.ietf.org/html/rfc6419

6. Інтернет-послуги в Україні дослідження компанії Ookla**.** https://www.speedtest.net/global-index/ukraine

7. Огляд сервісу nPerf**.** https://www.nperf.com

8. Cloudflare Speed Test – офіційний сайт. https://speed.cloudflare.com

9. Шевчук В.Я., Дьяків О.Й. Моделювання та аналіз інформаційних систем. – Львів: Видавництво ЛНУ, 2022.

10. Грінченко С.І., Третяк А.М. Якість програмного забезпечення. – К.: КНЕУ, 2021.

11. ISO/IEC 25010:2011. Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) System and software quality models**.**

12. Козак О.М., Васильєв С.В. Програмні системи тестування інтернет-з’єднання // Сучасні інформаційні технології. – 2021. – №2(98). – С. 42–47.

13. Курган М.О. Побудова розподілених систем моніторингу з використанням open-source рішень // Вісник НТУУ "КПІ". – 2020. – №3. – С. 53–58.

14. Spring Boot Documentation. https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle

15. Envoy Proxy Documentation. https://www.envoyproxy.io/docs

16. Гамма Е., Хелм Р., Джонсон Р., Вліссідес Дж. Приклади проектування. Шаблони об’єктно-орієнтованого проектування. – К.: Діалектика, 2020.

17. Соммервілл І. Інженерія програмного забезпечення / Пер. з англ. 10-е вид. – К.: Вільямс, 2020.

18. Басс Л., Клементс П., Казман Р. Програмна архітектура: структура програмних систем. – Київ: Наука і техніка, 2021.

19. Кліманов В. В. Архітектура корпоративних інформаційних систем. – М.: БІНОМ, 2019.

20. Richards M., Ford N. Fundamentals of Software Architecture: An Engineering Approach. – O’Reilly Media, 2020.

21. REST API Tutorial https://restfulapi.net/

22. Microservices Architecture https://microservices.io/

23. Spring Boot Reference Documentation https://docs.spring.io/spring-boot/docs/current/reference/html/

24. JSON Official Site – https://www.json.org/json-en.html

25. Martin Fowler. *Monolith First* https://martinfowler.com/bliki/MonolithFirst.html

26. Robert C. Martin. "Clean Architecture: A Craftsman's Guide to Software Structure and Design"

27. Fowler M. "Patterns of Enterprise Application Architecture"
28. Spring Framework Documentation – https://docs.spring.io/spring-framework/docs/current/reference/html/
29. Spring Boot Reference Documentation – https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/
30. Spring Data JPA Reference Documentation – https://docs.spring.io/spring-data/jpa/docs/current/reference/html/
31. Envoy Proxy Official Documentation – https://www.envoyproxy.io/docs/envoy/latest/
32. Vue.js Official Documentation – https://vuejs.org/guide/introduction.html
33. Redis Documentation – https://redis.io/docs/
34. Quartz Scheduler Documentationт – https://www.quartz-scheduler.org/documentation/
35. JWT (JSON Web Token) Introduction & Specification https://jwt.io/introduction
36. MDN Web Docs. Browser compatibility & feature support
37. Google Developers. Web Fundamentals – Tools for testing
38. Jest Documentation. Getting Started
39. Cypress Documentation. End-to-End Testing Framework
40. Spring Framework. Testing in Spring Boot

ДОДАТОК А

(обов’язковий)

ТЕХНІЧНЕ ЗАВДАННЯ

Загальні відомості

Робота виконується в рамках проєкту розробки веб-застосунку, спрямованого на спрощення процесу перевірки якості інтернет з’єднання, а також для перевірки багатьох факторів на лінії мережі та збереженні усієї інформації та інструментів в одному додатку.

Застосунок допомагає працівникам телекомунікаційних мереж відстежувати стан мережі, знаходити проблеми на абонентських відводах, прив’язувати абонентів по оптичній мережі та слідкувати за роботою комутаторів.

Умовне позначення системи: Веб-застосунок «ITest» – застосунок для моніторингу та тестування якості інтернет послуг.

Обґрунтування розробки застосунку

Розробка здійснюється відповідно до «Завдання на кваліфікаційну роботу», затвердженого дипломним керівником та завідувачем кафедри інженерії програмного забезпечення.

Назва проєкту: «ITest» – застосунок для моніторингу та тестування якості інтернет послуг.

Призначення розробки

Веб-застосунок **«ITest»** призначений для надання працівникам телекомунікаційних мереж зручного та простого інструменту для організації роботи.

Застосунок дозволить користувачам перевіряти роботу мережі, переглядати інформацію про клієнтських відводах, вести історію сервісного обслуговування, отримувати нагадування при падінні мережі, та інші важливі події, а також зберігати всі дані про події на мережі.

Крім того, застосунок матиме функцію перевірки абонентського відводу та перевірки помилок на ньому.

Функціональне призначення мобільного застосунку «ITest»

Мобільний застосунок **«ITest»** надає користувачам такі можливості:

– Реєстрація по робочій пошті;

– перегляд стану мережі та інформацію на ній;

– ведення інформації по комутаторах на мережі;

– отримання та створення власних нагадувань про важливі події на мережі;

– зміна теми інтерфейсу;

Користувачами застосунку є працівники телекомунікаційних мереж, які користуються смартфонами або ПК на платформі Android, IOS, Windiws та MacOS.

Застосунок може використовуватися на будь-якому пристрої з представлених.

3 Вимоги до затосунку

3.1 Функціональні вимоги

– **реєстрація та авторизація користувачів**: система повинна забезпечувати реєстрацію нових користувачів і авторизацію існуючих. Під час реєстрації користувач має вводити електронну пошту, пароль і підтвердження пароля. Після реєстрації користувач має можливість входити до системи за допомогою введених облікових даних. Передбачено підтримку автентифікації через Google.

– **створення та збереження сценаріїв тестування**: користувач повинен мати змогу створювати власні сценарії для перевірки якості інтернет-послуг, зокрема вказуючи IP-адреси, тип перевірки (ping, HTTP, DNS тощо), частоту виконання, очікуваний результат. Сценарії зберігаються в особистому профілі користувача.

– **перегляд результатів тестування**: система повинна надавати можливість переглядати історію виконання тестів з детальною інформацією про затримку,

втрату пакетів, час відповіді сервера, наявність DNS-помилок тощо. Результати мають бути представлені у вигляді списків та графіків.
– **моніторинг у реальному часі**: користувач має можливість активувати моніторинг якості інтернет-з'єднання в реальному часі, з виведенням результатів на інформаційну панель.
– **налаштування сповіщень**: користувач повинен мати змогу налаштовувати сповіщення про критичні події (наприклад, відсутність відповіді, перевищення порогового часу затримки). Повідомлення можуть надсилатися електронною поштою або виводитися в інтерфейсі системи.
– **створення та керування профілями перевірок**: користувач може створювати кілька профілів тестування для різних мереж або сайтів, задаючи специфічні параметри для кожного з них.
– **адміністративна панель**: адміністратор має мати змогу переглядати загальну статистику системи, активність користувачів, керувати налаштуваннями серверної частини та додавати/видаляти доступні типи перевірок.
– **інтеграція з візуалізаційними інструментами**: система повинна підтримувати виведення логів і результатів тестів до зовнішніх інструментів, таких як Kibana або Grafana, через API чи файловий обмін.
– **експортування звітів**: користувач повинен мати можливість експортувати результати тестування у форматах CSV або PDF для подальшого аналізу.
– **мобільна підтримка**: система має забезпечувати адаптивний інтерфейс для доступу з мобільних пристроїв, включно з базовою функціональністю перегляду результатів, запуску перевірок і сповіщень.

3.2 Нефункціональні вимоги

Застосунок ITest повинен відповідати наступним вимогам до надійності:

– **надійність**: система повинна працювати стабільно протягом тривалого часу без збоїв. Результати тестування мають зберігатися навіть у разі втрати з’єднання або перезапуску серверу.
– **масштабованість**: архітектура системи має дозволяти горизонтальне масштабування, зокрема через контейнеризацію (Docker) та балансування

навантаження (Envoy Proxy), щоб підтримувати зростання кількості користувачів та перевірок.

– **безпека**: усі дані користувачів (включно з обліковими записами, конфігураціями та результатами тестів) повинні зберігатися у зашифрованому вигляді. Передбачено аутентифікацію з JWT-токенами, захист API через TLS, перевірку прав доступу.

– **конфіденційність**: доступ до даних про перевірки має бути лише у зареєстрованого користувача, який їх створив. Система повинна відповідати вимогам захисту персональних даних (GDPR-подібним, якщо застосовно).

– **продуктивність**: час відповіді інтерфейсу користувача на запит повинен бути не більше 1 секунди, а результати перевірки повинні оброблятися в межах 2–5 секунд (залежно від типу перевірки).

– **кросплатформеність**: інтерфейс системи має бути доступний у сучасних браузерах (Chrome, Firefox, Edge) та мати адаптивну верстку для мобільних пристроїв.

– **зручність користування**: система повинна мати інтуїтивно зрозумілий інтерфейс із чітким поділом на модулі (перевірки, профілі, звіти), графічне представлення даних та інтерфейс керування без надлишкових дій.

– **надійне логування та аудит**: система повинна вести журнал дій користувачів, помилок і подій моніторингу для спрощення діагностики та аналізу.

– **відмовостійкість**: у випадку відмови одного з мікросервісів або компонентів система повинна забезпечити автоматичне відновлення або резервне виконання задач.

– **можливість розгортання у хмарному середовищі**: система має бути сумісною з платформами AWS, GCP або Azure, з підтримкою CI/CD-процесів для оновлення.

3.3 Умови експлуатації

Умови експлуатації повинні відповідати санітарним і технічним вимогам використання мобільного пристрою або ПК, з урахуванням температурного режиму та рівня вологості, встановлених виробником гаджета. Застосунок підтримується на смартфонах та ПК.

3.4 Вимоги до середовища виконання

Програмна система тестування та моніторингу якості інтернет-послуг призначена для розгортання у хмарному середовищі або на локальному сервері з підтримкою вебінтерфейсу для користувачів. Система складається з клієнтської частини (інтерфейс адміністратора або користувача в браузері) та серверної частини, реалізованої з використанням фреймворку Spring Boot.

Для повноцінної роботи системи необхідно забезпечити доступ до мережі Інтернет, оскільки більшість функцій (авторизація, запуск перевірок, перегляд результатів, моніторинг у реальному часі) потребують з’єднання із сервером. Частковий функціонал, наприклад перегляд останніх локально збережених звітів, може бути доступний в автономному режимі, однак усі оновлення та нові перевірки потребують синхронізації з сервером.

Вимоги до середовища серверної частини (бекенду):

- Операційна система: Linux (Ubuntu 20.04 або новіша) або Windows Server 2019+
- Процесор: 4 ядра, 2.0 ГГц і вище
- Оперативна пам’ять: не менше 8 ГБ
- Вільне місце на диску: від 10 ГБ (в залежності від обсягу збережених логів і результатів перевірок)
- Середовище виконання: Java 17+, Docker (для контейнеризації), PostgreSQL 13+
- Додатково: Nginx або Envoy Proxy як реверс-проксі (з підтримкою HTTPS), доступ до SMTP-сервера для надсилання сповіщень

Вимоги до середовища клієнтської частини (вебінтерфейс):

- Браузери: останні версії Chrome, Firefox, Safari або Edge

– Операційна система: Windows 10+, macOS 11+, Linux (сучасні дистрибутиви), Android 10+, iOS 13+

– Екран: мінімальна роздільна здатність 1280×720, адаптивність забезпечена для мобільних пристроїв

– Підключення до Інтернету: стабільне з’єднання з пропускною здатністю не менше 1 Мбіт/с

Система оптимізована для сучасних браузерів та пристроїв, проте на старіших або повільних пристроях можлива знижена швидкодія при візуалізації великих обсягів статистики або запуску важких перевірок.

3.5 Вимоги до інформаційної та програмної сумісності

Програмна система тестування та моніторингу якості інтернет-послуг реалізована з використанням архітектури клієнт-сервер. Фронтенд-частина розроблена за допомогою фреймворку Vue.js, а бекенд — з використанням Java Spring Boot. Такий підхід забезпечує високу сумісність між клієнтською частиною, сервером і сторонніми сервісами.

– Клієнтська частина: сучасні браузери, що підтримують HTML5, CSS3, JavaScript (ES6+); адаптивний інтерфейс сумісний з десктопами, планшетами та мобільними пристроями.

– Бекенд: REST API, що забезпечує взаємодію з фронтендом, мобільним клієнтом та сторонніми системами (через HTTP/HTTPS).

– Проксі-сервер: система підтримує інтеграцію з Envoy Proxy як реверс-проксі для маршрутизації запитів та балансування навантаження.

– База даних: PostgreSQL використовується як основна СКБД для зберігання результатів перевірок, налаштувань користувачів, конфігурацій сповіщень тощо. Забезпечено сумісність із іншими СКБД SQL типу, у разі потреби міграції.

– Інтеграція з хмарними сервісами: підтримується підключення до зовнішніх систем аналітики, хмарного зберігання даних, систем сповіщення (наприклад, SMTP-сервери, Telegram Bot API).

– Протоколи та формати: система підтримує стандартні формати обміну даними, такі як JSON, CSV та XML, для інтеграції з іншими сервісами та вивантаження звітів.

Завдяки використанню технологій з відкритим кодом, дотриманню принципів Clean Architecture та модульності реалізації, система легко масштабуються та адаптується до змін у зовнішньому середовищі. Забезпечено сумісність між компонентами системи, незалежно від операційної системи серверів або пристроїв користувачів, що дозволяє зручно розгортати її у хмарних середовищах або на фізичній інфраструктурі замовника.

3.6 Спеціальні вимоги

Вимоги до інтерфейсу:

− Наявність світлої (білої) та темної (чорної) теми оформлення інтерфейсу, які користувач може змінювати вручну або відповідно до системних налаштувань пристрою.

− Інтерфейс повинен містити нижнє навігаційне меню для швидкого доступу до основних розділів системи (наприклад: головна, перевірки, статистика, налаштування).

− Кожен екран повинен містити чітке відображення своєї назви та рівня вкладеності, що дозволяє користувачу розуміти контекст поточної взаємодії з системою.

− Інтерфейс має бути адаптивним для коректного відображення на різних типах пристроїв (ПК, планшет, мобільний).

4 Вимоги до програмної документації

Після завершення розробки та під час передачі програмної системи замовнику надається повний комплект супровідної документації, що включає:

− Вихідний код усіх компонентів системи (фронтенд, бекенд, мобільний клієнт) із коментарями, що пояснюють логіку реалізації ключових функцій.

− Опис функціональних можливостей системи тестування та моніторингу якості інтернет-послуг (згідно зі специфікацією та реалізованими модулями).

− Технічне завдання (цей документ), що містить вимоги до функціоналу, архітектури, інтерфейсу, середовища виконання, безпеки та сумісності.
− Інструкція для користувача з описом типових сценаріїв використання, способів перегляду результатів перевірок, налаштування моніторингу та отримання сповіщень.

4 Стадії та етапи розробки

Таблиця А.1 – Стадії та етапи розробки проєкту «VetCare»

Стадія розробки	Етапи робіт	Зміст робіт
Технічне завдання (01.02.2025 – 10.02.2025)	Обґрунтування необхідності розробки застосунку	Аналіз ринку та потреб користувачів, визначення аудиторії, формування вимог до функціональності, UI/UX та безпеки, вибір технологічного стека (Flutter, Firebase), визначення стадій та етапів розробки.
Ескізний проєкт (10.02.2025 – 10.03.2025)	Розробка ескізного проєкту	Проєктування архітектури застосунку, структури даних у Firestore, схеми взаємодії з Firebase Authentication та Firebase Storage, створення UX-макетів головних екранів.
Технічний проєкт (11.03.2025 – 30.03.2025)	Розробка технічного проєкту	Деталізація архітектури, створення схем збереження та отримання даних, проєктування основних алгоритмів.
Робочий проєкт (1.04.2025 – 20.04.2025)	Розробка застосунку	Реалізація основних функцій (авторизація, управління профілями тварин, запис на прийом), інтеграція Firebase, Firestore, Firebase Storage, відлагодження та оптимізація.
Розробка документації (21.04.2025 –	Створення документації	Опис архітектури та структури коду, документація бази даних, опис тестових сценаріїв.

05.05.2025)		
Тестування системи (06.05.2025 – 20.05.2025)	Перевірка та виправлення помилок	Проведення функціонального, інтеграційного та E2E-тестування, корекція знайдених помилок та доопрацювання.

6 Порядок контролю та приймання:

Види випробувань:

– Юніт-тестування: перевірка окремих функцій та методів, зокрема сервісів збору даних, обчислення метрик якості з'єднання (швидкість, затримка, втрата пакетів) та логіки взаємодії з API.

– Модульне тестування (тестування компонентів): оцінка роботи окремих частин інтерфейсу (Vue-компоненти), їхньої реактивності, а також взаємодії з користувачем.

– Інтеграційне тестування: перевірка коректності зв'язку між фронтендом, бекендом (Spring Boot), базою даних та проксі-сервером Envoy у процесах реєстрації, запуску тестів, отримання результатів.

– Snapshot-тестування: перевірка зовнішнього вигляду веб-інтерфейсу після оновлень, виявлення неочікуваних змін в UI.

– Тестування "кінець у кінець" (end-to-end, E2E): перевірка повного циклу роботи користувача в системі — авторизація, запуск тесту якості інтернету, перегляд результатів, моніторинг у реальному часі.

– Випробування на відмовостійкість: тестування стабільності роботи системи за умов навантаження, втрати мережевого з’єднання, нестабільного інтернету або помилок у відповідях від серверів.

Вимоги до приймання:

– Відповідність вимогам: система повинна повністю відповідати функціональним та нефункціональним вимогам, що визначені у технічному завданні.

– Відмовостійкість та стабільність: програмна система повинна стабільно працювати у різних мережевих умовах, витримувати помірне навантаження та забезпечувати коректне функціонування на різних типах пристроїв.

– Функціональне тестування: перевірка реалізації ключових функцій — реєстрація/авторизація користувачів, запуск тестів якості інтернету, перегляд результатів, отримання повідомлень про збої або сповіщень моніторингу.

– Взаємодія з сервісами: перевірка інтеграції з зовнішніми сервісами, зокрема базою даних, проксі-сервером Envoy, модулями збирання метрик, REST API бекенду, а також (за потреби) сторонніми API для геолокації чи аналітики.

– Безпека та конфіденційність: перевірка захисту персональних даних користувача, безпечної аутентифікації, а також захисту від несанкціонованого доступу до результатів тестів та системної інформації.

– Відповідність документації: технічна документація повинна бути актуальною, структурованою, зрозумілою та відповідати реалізованому функціоналу системи.

Технологічний стек:

– Фронтенд: Vue.js 3 (Composition API, Vue Router, Pinia)

– Бекенд: Java Spring Boot (REST API, WebSocket для моніторингу)

– Proxy-сервер: Envoy Proxy (замість Nginx)

– База даних: PostgreSQL

– Тестування: JUnit, Mockito, Cypress (для UI), Postman/Newman (для API)

– Інфраструктура: Docker, Docker Compose

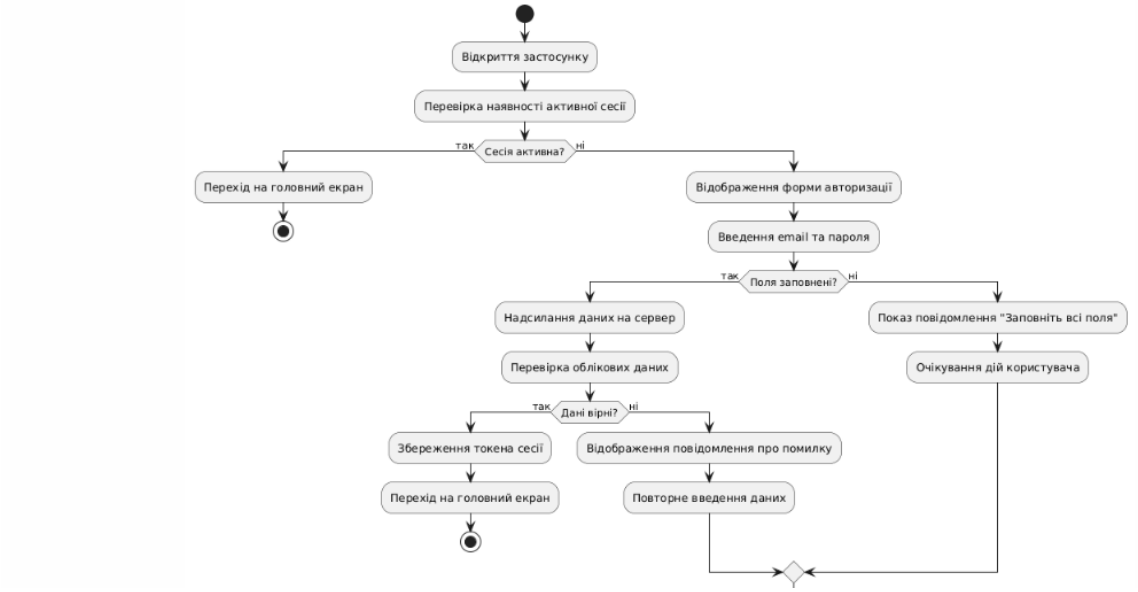
– Моніторинг та логування: Grafana, Prometheus, ELK Stack (опційно)

– Інтеграції: OpenTelemetry (для трасування запитів), Speedtest CLI / власні скрипти для тестування мережі

ДОДАТОК Б

(обов’язковий)

ДІАГРАМИ ДІЯЛЬНОСТІ



Додаток Б.1 – рисунок «Діаграма діяльності для процесу авторизації»

ДОДАТОК В

(обов’язковий)

КОД ПРОГРАМИ

```
plugins {
    id 'org.springframework.boot' version '3.0.0'
    id 'io.spring.dependency-management' version '1.1.0'
    id 'java'
}

group = 'com.example'
version = '0.0.1-SNAPSHOT'
sourceCompatibility = '17'

repositories {
    mavenCentral()
}

dependencies {
    implementation 'org.springframework.boot:spring-boot-starter-web'
    implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
    runtimeOnly 'com.h2database:h2'
    testImplementation 'org.springframework.boot:spring-boot-starter-test'
}
```

```
package com.example.internetquality;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class InternetQualityApplication {
    public static void main(String[] args) {
        SpringApplication.run(InternetQualityApplication.class, args);
    }
}
```

```
package com.example.internetquality.model;

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.Id;

@Entity
public class QualityTest {
    @Id
    @GeneratedValue
    private Long id;

    private String testName;
    private String status;  // Наприклад, "PENDING", "RUNNING", "COMPLETED"
    private double downloadSpeedMbps;
    private double uploadSpeedMbps;
    private int pingMs;

    // геттери і сеттери
    // конструктори
}
```

```
package com.example.internetquality.repository;

import com.example.internetquality.model.QualityTest;
import org.springframework.data.jpa.repository.JpaRepository;
```

```
public interface QualityTestRepository extends JpaRepository<QualityTest, Long>
{}
package com.example.internetquality.service;

import com.example.internetquality.model.QualityTest;
import com.example.internetquality.repository.QualityTestRepository;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.Optional;

@Service
public class QualityTestService {
    private final QualityTestRepository repository;

    public QualityTestService(QualityTestRepository repository) {
        this.repository = repository;
    }

    public QualityTest createTest(QualityTest test) {
        test.setStatus("PENDING");
        return repository.save(test);
    }

    public List<QualityTest> getAllTests() {
        return repository.findAll();
    }

    public Optional<QualityTest> getTestById(Long id) {
        return repository.findById(id);
    }

    // Тут можна додати логіку запуску тестів, оновлення статусу і результатів
}
package com.example.internetquality.controller;

import com.example.internetquality.model.QualityTest;
import com.example.internetquality.service.QualityTestService;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api/tests")
public class QualityTestController {
    private final QualityTestService service;

    public QualityTestController(QualityTestService service) {
        this.service = service;
    }

    @PostMapping
    public ResponseEntity<QualityTest> createTest(@RequestBody QualityTest test)
{
        QualityTest created = service.createTest(test);
        return ResponseEntity.ok(created);
    }

    @GetMapping
    public ResponseEntity<List<QualityTest>> getAllTests() {
        return ResponseEntity.ok(service.getAllTests());
    }

    @GetMapping("/{id}")
```

```
    public ResponseEntity<QualityTest> getTestById(@PathVariable Long id) {
        return service.getTestById(id)
                .map(ResponseEntity::ok)
                .orElse(ResponseEntity.notFound().build());
    }
}
npm init vue@latest
# Вибери Vue 3, без додаткових фреймворків для простоти
cd <project-name>
npm install
npm install axios
npm run dev
<template>
  <div id="app">
    <h1>Моніторинг якості інтернет-послуг</h1>

    <form @submit.prevent="createTest">
      <input v-model="newTestName" placeholder="Назва тесту" required />
      <button type="submit">Створити тест</button>
    </form>

    <h2>Список тестів</h2>
    <ul>
      <li v-for="test in tests" :key="test.id">
        {{ test.testName }} — Статус: {{ test.status }}
        <button @click="fetchTestDetails(test.id)">Деталі</button>
      </li>
    </ul>

    <div v-if="selectedTest">
      <h3>Деталі тесту: {{ selectedTest.testName }}</h3>
      <p>Статус: {{ selectedTest.status }}</p>
      <p>Швидкість завантаження: {{ selectedTest.downloadSpeedMbps }} Mbps</p>
      <p>Швидкість відвантаження: {{ selectedTest.uploadSpeedMbps }} Mbps</p>
      <p>Пінг: {{ selectedTest.pingMs }} ms</p>
    </div>
  </div>
</template>

<script setup>
import { ref, onMounted } from 'vue'
import axios from 'axios'

const tests = ref([])
const newTestName = ref('')
const selectedTest = ref(null)

const fetchTests = async () => {
  const response = await axios.get('http://localhost:8080/api/tests')
  tests.value = response.data
}

const createTest = async () => {
  if (!newTestName.value) return
  await axios.post('http://localhost:8080/api/tests', {
    testName: newTestName.value
  })
  newTestName.value = ''
  await fetchTests()
}

const fetchTestDetails = async (id) => {
  const response = await axios.get(`http://localhost:8080/api/tests/${id}`)
  selectedTest.value = response.data
}
```

```
onMounted(fetchTests)
</script>

<style>
body {
  font-family: Arial, sans-serif;
  margin: 20px;
}
</style>
implementation 'org.springframework.boot:spring-boot-starter-security'
implementation 'io.jsonwebtoken:jjwt-api:0.11.5'
runtimeOnly 'io.jsonwebtoken:jjwt-impl:0.11.5'
runtimeOnly 'io.jsonwebtoken:jjwt-jackson:0.11.5'
@Service
public class QualityTestService {
    // ...

    public QualityTest runTest(Long id) {
        QualityTest test = repository.findById(id).orElseThrow();
        test.setStatus("RUNNING");
        repository.save(test);

        // Імітація виконання тесту (можна робити асинхронно)
        test.setDownloadSpeedMbps(Math.random() * 100);
        test.setUploadSpeedMbps(Math.random() * 50);
        test.setPingMs((int)(Math.random() * 100));
        test.setStatus("COMPLETED");
        return repository.save(test);
    }
}
@PostMapping("/{id}/run")
public ResponseEntity<QualityTest> runTest(@PathVariable Long id) {
    QualityTest test = service.runTest(id);
    return ResponseEntity.ok(test);
}
npm install chart.js@3
npm install vue-chartjs@4
<template>
  <!-- ... -->
  <div v-if="selectedTest && selectedTest.status === 'COMPLETED'">
    <h3>Графік результатів тесту</h3>
    <line-chart :chart-data="chartData" />
  </div>
</template>

<script setup>
import { ref, watch } from 'vue'
import { Line } from 'vue-chartjs'
import {
  Chart,
  Title,
  Tooltip,
  Legend,
  LineElement,
  CategoryScale,
  LinearScale,
  PointElement,
} from 'chart.js'

Chart.register(Title, Tooltip, Legend, LineElement, CategoryScale, LinearScale,
PointElement)

const selectedTest = ref(null)

const chartData = ref({
  labels: ['Ping (ms)', 'Download (Mbps)', 'Upload (Mbps)'],
```

```
  datasets: [
    {
      label: 'Результати',
      backgroundColor: '#f87979',
      data: [],
    },
  ],
})

watch(selectedTest, () => {
  if (selectedTest.value && selectedTest.value.status === 'COMPLETED') {
    chartData.value.datasets[0].data = [
      selectedTest.value.pingMs,
      selectedTest.value.downloadSpeedMbps,
      selectedTest.value.uploadSpeedMbps,
    ]
  }
})

// компонент графіка
const lineChart = {
  extends: Line,
  props: ['chartData'],
  mounted() {
    this.renderChart(this.chartData, { responsive: true, maintainAspectRatio:
false })
  },
}

</script>
package com.example.internetquality.model;

import jakarta.persistence.Entity;
import jakarta.persistence.Id;

@Entity
public class AppUser {

    @Id
    private String username;
    private String password;  // Зберігати в зашифрованому вигляді (BCrypt)

    // геттери/сеттери
}
package com.example.internetquality.security;

import com.example.internetquality.model.AppUser;
import com.example.internetquality.repository.AppUserRepository;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;

import java.util.Collections;

@Service
public class CustomUserDetailsService implements UserDetailsService {
    private final AppUserRepository userRepository;

    public CustomUserDetailsService(AppUserRepository userRepository) {
        this.userRepository = userRepository;
    }

    @Override
    public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
```

```
        AppUser user = userRepository.findById(username)
                .orElseThrow(() -> new UsernameNotFoundException("User not
found"));
        return new org.springframework.security.core.userdetails.User(
                user.getUsername(),
                user.getPassword(),
                Collections.emptyList()
        );
    }
}
package com.example.internetquality.security;

import io.jsonwebtoken.Claims;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;
import org.springframework.stereotype.Component;

import java.util.Date;

@Component
public class JwtUtil {
    private final String SECRET_KEY = "your-secret-key"; // краще в
application.properties

    public String generateToken(String username) {
        return Jwts.builder()
                .setSubject(username)
                .setIssuedAt(new Date())
                .setExpiration(new Date(System.currentTimeMillis() + 1000 * 60 *
60 * 10)) // 10 годин
                .signWith(SignatureAlgorithm.HS256, SECRET_KEY)
                .compact();
    }

    public String extractUsername(String token) {
        return getClaims(token).getSubject();
    }

    public boolean validateToken(String token, String username) {
        return extractUsername(token).equals(username) &&
!isTokenExpired(token);
    }

    private Claims getClaims(String token) {
        return Jwts.parser()
                .setSigningKey(SECRET_KEY)
                .parseClaimsJws(token)
                .getBody();
    }

    private boolean isTokenExpired(String token) {
        return getClaims(token).getExpiration().before(new Date());
    }
}
package com.example.internetquality.security;

import jakarta.servlet.FilterChain;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import
org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetails;
import
org.springframework.security.web.authentication.WebAuthenticationDetailsSource;
```

```
import org.springframework.stereotype.Component;
import org.springframework.web.filter.OncePerRequestFilter;

import java.io.IOException;

@Component
public class JwtRequestFilter extends OncePerRequestFilter {

    private final CustomUserDetailsService userDetailsService;
    private final JwtUtil jwtUtil;

    public JwtRequestFilter(CustomUserDetailsService userDetailsService, JwtUtil
jwtUtil) {
        this.userDetailsService = userDetailsService;
        this.jwtUtil = jwtUtil;
    }

    @Override
    protected void doFilterInternal(HttpServletRequest request,
                                    HttpServletResponse response,
                                    FilterChain filterChain)
            throws ServletException, IOException {
        final String authorizationHeader = request.getHeader("Authorization");

        String username = null;
        String jwt = null;

        if (authorizationHeader != null &&
authorizationHeader.startsWith("Bearer ")) {
            jwt = authorizationHeader.substring(7);
            username = jwtUtil.extractUsername(jwt);
        }

        if (username != null &&
SecurityContextHolder.getContext().getAuthentication() == null) {
            UserDetails userDetails =
userDetailsService.loadUserByUsername(username);

            if (jwtUtil.validateToken(jwt, userDetails.getUsername())) {
                UsernamePasswordAuthenticationToken authToken =
                        new UsernamePasswordAuthenticationToken(userDetails,
null, userDetails.getAuthorities());

                authToken.setDetails(new
WebAuthenticationDetailsSource().buildDetails(request));

                SecurityContextHolder.getContext().setAuthentication(authToken);
            }
        }
        filterChain.doFilter(request, response);
    }
}
package com.example.internetquality.security;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationManager;
import
org.springframework.security.config.annotation.authentication.configuration.Auth
enticationConfiguration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.web.SecurityFilterChain;
import
org.springframework.security.web.authentication.UsernamePasswordAuthenticationFi
lter;
```

```
@Configuration
public class SecurityConfig {

    private final JwtRequestFilter jwtRequestFilter;

    public SecurityConfig(JwtRequestFilter jwtRequestFilter) {
        this.jwtRequestFilter = jwtRequestFilter;
    }

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http.csrf(csrf -> csrf.disable())
            .authorizeHttpRequests(auth -> auth
                .requestMatchers("/api/auth/**").permitAll()
                .anyRequest().authenticated()
            )
            .sessionManagement(sess ->
sess.sessionCreationPolicy(SessionCreationPolicy.STATELESS));

        http.addFilterBefore(jwtRequestFilter,
UsernamePasswordAuthenticationFilter.class);

        return http.build();
    }

    @Bean
    public AuthenticationManager
authenticationManager(AuthenticationConfiguration config) throws Exception {
        return config.getAuthenticationManager();
    }
}
package com.example.internetquality.controller;

import com.example.internetquality.model.AppUser;
import com.example.internetquality.repository.AppUserRepository;
import com.example.internetquality.security.JwtUtil;
import org.springframework.http.ResponseEntity;
import org.springframework.security.authentication.AuthenticationManager;
import
org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/api/auth")
public class AuthController {

    private final AuthenticationManager authenticationManager;
    private final JwtUtil jwtUtil;
    private final AppUserRepository userRepository;
    private final BCryptPasswordEncoder passwordEncoder = new
BCryptPasswordEncoder();

    public AuthController(AuthenticationManager authenticationManager,
                          JwtUtil jwtUtil,
                          AppUserRepository userRepository) {
        this.authenticationManager = authenticationManager;
        this.jwtUtil = jwtUtil;
        this.userRepository = userRepository;
    }

    @PostMapping("/register")
    public ResponseEntity<?> register(@RequestBody AppUser user) {
        user.setPassword(passwordEncoder.encode(user.getPassword()));
        userRepository.save(user);
```

```
        return ResponseEntity.ok("User registered");
    }

    @PostMapping("/login")
    public ResponseEntity<?> login(@RequestBody AppUser user) {
        authenticationManager.authenticate(
                new UsernamePasswordAuthenticationToken(user.getUsername(),
user.getPassword()));

        String token = jwtUtil.generateToken(user.getUsername());
        return ResponseEntity.ok(token);
    }
}
<template>
  <canvas ref="canvas"></canvas>
</template>

<script setup>
import { onMounted, ref, watch } from 'vue'
import {
  Chart,
  Title,
  Tooltip,
  Legend,
  LineElement,
  CategoryScale,
  LinearScale,
  PointElement,
} from 'chart.js'

Chart.register(Title, Tooltip, Legend, LineElement, CategoryScale, LinearScale,
PointElement)

const props = defineProps({
  chartData: Object
})

const canvas = ref(null)
let chart = null

onMounted(() => {
  chart = new Chart(canvas.value, {
    type: 'line',
    data: props.chartData,
    options: {
      responsive: true,
      maintainAspectRatio: false,
    }
  })
})

watch(() => props.chartData, (newData) => {
  if(chart) {
    chart.data = newData
    chart.update()
  }
})
</script>

<style scoped>
canvas {
  width: 100%;
  height: 300px;
}
</style>
<template>
```

```
  <div>
    <!-- Інший код -->
    <div v-if="selectedTest && selectedTest.status === 'COMPLETED'">
      <h3>Графік результатів тесту</h3>
      <LineChart :chartData="chartData" />
    </div>
  </div>
</template>

<script setup>
import { ref, watch } from 'vue'
import LineChart from './components/LineChart.vue'

const selectedTest = ref(null)  // Заповнюється після вибору тесту

const chartData = ref({
  labels: ['Ping (ms)', 'Download (Mbps)', 'Upload (Mbps)'],
  datasets: [
    {
      label: 'Результати тесту',
      backgroundColor: 'rgba(255,99,132,0.2)',
      borderColor: 'rgba(255,99,132,1)',
      data: [],
      fill: true,
      tension: 0.4
    }
  ]
})

watch(selectedTest, (newVal) => {
  if(newVal && newVal.status === 'COMPLETED') {
    chartData.value.datasets[0].data = [
      newVal.pingMs,
      newVal.downloadSpeedMbps,
      newVal.uploadSpeedMbps,
    ]
  }
})
</script>
plugins {
    id 'java'
    id 'org.springframework.boot' version '3.0.5'
    id 'io.spring.dependency-management' version '1.1.0'
}

group = 'com.example'
version = '0.0.1-SNAPSHOT'
sourceCompatibility = '17'

repositories {
    mavenCentral()
}

dependencies {
    implementation 'org.springframework.boot:spring-boot-starter-web'
    implementation 'org.springframework.boot:spring-boot-starter-security'
    implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
    implementation 'io.jsonwebtoken:jjwt-api:0.11.5'
    runtimeOnly 'io.jsonwebtoken:jjwt-impl:0.11.5'
    runtimeOnly 'io.jsonwebtoken:jjwt-jackson:0.11.5'
    runtimeOnly 'com.h2database:h2'
    implementation 'org.springframework.boot:spring-boot-starter-validation'
}

tasks.named('test') {
    useJUnitPlatform()
```

```
}
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=

spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.jpa.hibernate.ddl-auto=update

jwt.secret=VerySecretKey1234567890
jwt.expiration=36000000
package com.example.internetquality.model;

import jakarta.persistence.Entity;
import jakarta.persistence.Id;

@Entity
public class AppUser {
    @Id
    private String username;
    private String password;

    public AppUser() {}

    public AppUser(String username, String password) {
        this.username = username;
        this.password = password;
    }
    public String getUsername() { return username; }
    public void setUsername(String username) { this.username = username; }
    public String getPassword() { return password; }
    public void setPassword(String password) { this.password = password; }
}
package com.example.internetquality.security;

import io.jsonwebtoken.*;

import org.springframework.beans.factory.annotation.Value;

import org.springframework.stereotype.Component;

import java.util.Date;

@Component

public class JwtUtil {

@Value("${jwt.secret}")

private String secret;

@Value("${jwt.expiration}")

private Long expiration;

public String generateToken(String username) {

return Jwts.builder()

.setSubject(username)

.setIssuedAt(new Date())

.setExpiration(new Date(System.currentTimeMillis() + expiration))

.signWith(SignatureAlgorithm.HS256, secret)
```

```
.compact();
}
public String extractUsername(String token) {
return getClaims(token).getSubject();
}
public boolean validateToken(String token, String username) {
return (extractUsername(token).equals(username)) && !isTokenExpired(token);
}
private Claims getClaims(String token) {
return Jwts.parser()
.setSigningKey(secret)
.parseClaimsJws(token)
.getBody();
}
private boolean isTokenExpired(String token) {
return getClaims(token).getExpiration().before(new Date());
}
}
package com.example.internetquality.security;
import com.example.internetquality.model.AppUser;
import com.example.internetquality.repository.AppUserRepository;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;
import java.util.Collections;
@Service
public class CustomUserDetailsService implements UserDetailsService {
private final AppUserRepository userRepository;
public CustomUserDetailsService(AppUserRepository userRepository) {
this.userRepository = userRepository;
}
@Override
public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
AppUser user = userRepository.findById(username)
```

```
.orElseThrow(() -> new UsernameNotFoundException("User not found"));

return new org.springframework.security.core.userdetails.User(

user.getUsername(),

user.getPassword(),

Collections.emptyList()

);

}

}

package com.example.internetquality.security;

import jakarta.servlet.FilterChain;

import jakarta.servlet.ServletException;

import jakarta.servlet.http.HttpServletRequest;

import jakarta.servlet.http.HttpServletResponse;

import
org.springframework.security.authentication.UsernamePasswordAuthenticationToken;

import org.springframework.security.core.context.SecurityContextHolder;

import org.springframework.security.core.userdetails.UserDetails;

import
org.springframework.security.web.authentication.WebAuthenticationDetailsSource;

import org.springframework.stereotype.Component;

import org.springframework.web.filter.OncePerRequestFilter;

import java.io.IOException;

@Component

public class JwtRequestFilter extends OncePerRequestFilter {

private final CustomUserDetailsService userDetailsService;

private final JwtUtil jwtUtil;

public JwtRequestFilter(CustomUserDetailsService userDetailsService, JwtUtil
jwtUtil) {

this.userDetailsService = userDetailsService;

this.jwtUtil = jwtUtil;

}

@Override

protected void doFilterInternal(HttpServletRequest request,

HttpServletResponse response,

FilterChain filterChain)

throws ServletException, IOException {

final String authHeader = request.getHeader("Authorization");
```

```
String username = null;

String jwt = null;

if(authHeader != null && authHeader.startsWith("Bearer ")) {

jwt = authHeader.substring(7);

try {

username = jwtUtil.extractUsername(jwt);

} catch (Exception e) {

// invalid token

}

}

if(username != null && SecurityContextHolder.getContext().getAuthentication() ==
null) {

UserDetails userDetails = this.userDetailsService.loadUserByUsername(username);

if(jwtUtil.validateToken(jwt, userDetails.getUsername())) {

UsernamePasswordAuthenticationToken authToken =

new UsernamePasswordAuthenticationToken(userDetails, null,
userDetails.getAuthorities());

authToken.setDetails(new
WebAuthenticationDetailsSource().buildDetails(request));

SecurityContextHolder.getContext().setAuthentication(authToken);

}

}

filterChain.doFilter(request, response);

}

}

package com.example.internetquality.security;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.security.authentication.AuthenticationManager;

import
org.springframework.security.config.annotation.authentication.configuration.Auth
enticationConfiguration;

import org.springframework.security.config.annotation.web.builders.HttpSecurity;

import org.springframework.security.config.http.SessionCreationPolicy;

import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

import org.springframework.security.web.SecurityFilterChain;
```

```
import
org.springframework.security.web.authentication.UsernamePasswordAuthenticationFi
lter;

@Configuration

public class SecurityConfig {

private final JwtRequestFilter jwtRequestFilter;

public SecurityConfig(JwtRequestFilter jwtRequestFilter) {

this.jwtRequestFilter = jwtRequestFilter;

}

@Bean

public SecurityFilterChain securityFilterChain(HttpSecurity http) throws
Exception {

http.csrf(csrf -> csrf.disable())

.authorizeHttpRequests(auth -> auth

.requestMatchers("/api/auth/**").permitAll()

.anyRequest().authenticated()

)

.sessionManagement(sess ->
sess.sessionCreationPolicy(SessionCreationPolicy.STATELESS));

http.addFilterBefore(jwtRequestFilter,
UsernamePasswordAuthenticationFilter.class);

return http.build();

}

@Bean

public BCryptPasswordEncoder passwordEncoder() {

return new BCryptPasswordEncoder();

}

@Bean

public AuthenticationManager authenticationManager(AuthenticationConfiguration
config) throws Exception {

return config.getAuthenticationManager();

}

}

package com.example.internetquality.controller;

import com.example.internetquality.model.AppUser;

import com.example.internetquality.repository.AppUserRepository;

import com.example.internetquality.security.JwtUtil;

import org.springframework.http.ResponseEntity;
```

```
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.BadCredentialsException;
import
org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.web.bind.annotation.*;
@RestController
@RequestMapping("/api/auth")
public class AuthController {
private final AuthenticationManager authenticationManager;
private final JwtUtil jwtUtil;
private final AppUserRepository userRepository;
private final BCryptPasswordEncoder passwordEncoder;
public AuthController(AuthenticationManager authenticationManager,
JwtUtil jwtUtil,
AppUserRepository userRepository,
BCryptPasswordEncoder passwordEncoder) {
this.authenticationManager = authenticationManager;
this.jwtUtil = jwtUtil;
this.userRepository = userRepository;
this.passwordEncoder = passwordEncoder;
}
@PostMapping("/register")
public ResponseEntity<?> register(@RequestBody AppUser user) {
if(userRepository.existsById(user.getUsername())) {
return ResponseEntity.badRequest().body("User already exists");
}
user.setPassword(passwordEncoder.encode(user.getPassword()));
userRepository.save(user);
return ResponseEntity.ok("User registered successfully");
}
@PostMapping("/login")
public ResponseEntity<?> login(@RequestBody AppUser user) {
try {
authenticationManager.authenticate(
```

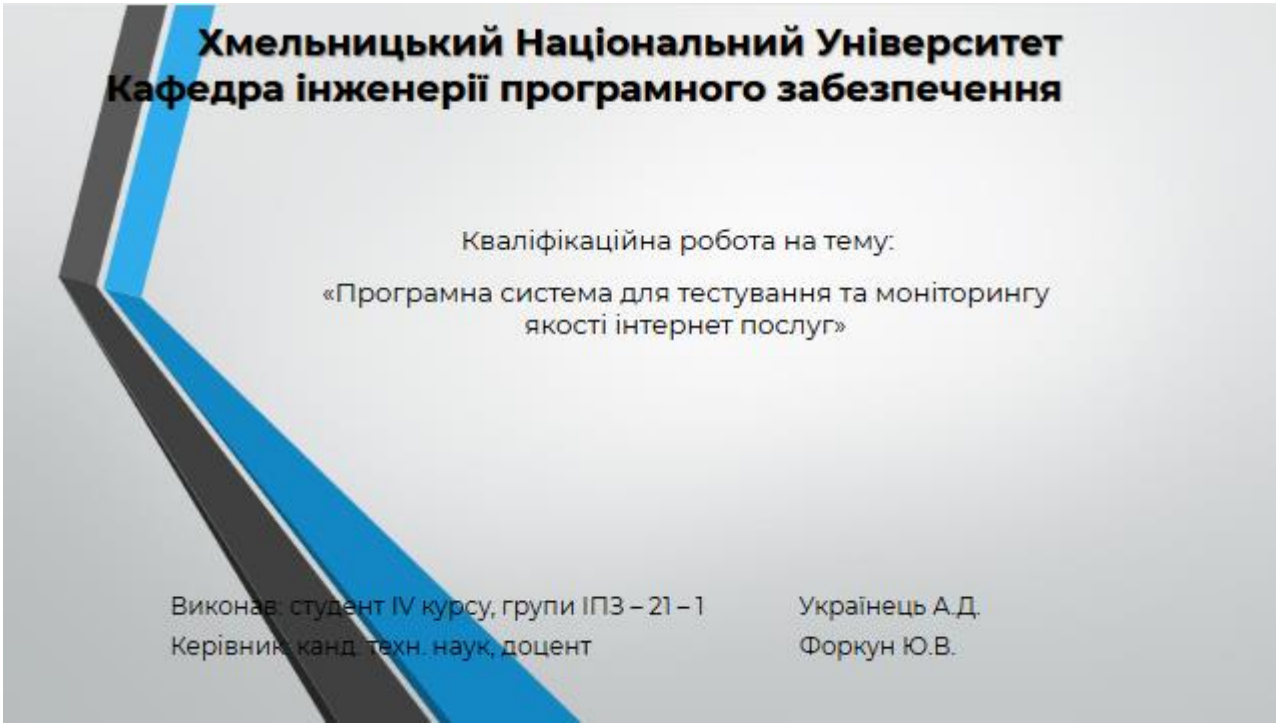
```
new UsernamePasswordAuthenticationToken(user.getUsername(),
user.getPassword()));

} catch (BadCredentialsException e) {

return ResponseEntity.status(401).body("Incorrect username or password");

}

final String token = jwtUtil.generateToken(user.getUsername());

return ResponseEntity.ok(token);

}

}

package com.example.internetquality.model;

import jakarta.persistence.Entity;

import jakarta.persistence.GeneratedValue;

import jakarta.persistence.Id;

@Entity

public class QualityTest {

@Id

@GeneratedValue

private Long id;

private String status; // NEW, RUNNING, COMPLETED

private double downloadSpeedMbps;

private double uploadSpeedMbps;

private int pingMs;

public QualityTest() {

this.status = "NEW";

}

// геттери/сеттери

// ...

public Long getId() { return id; }

public void setId(Long id) { this.id = id; }

public String getStatus() { return status; }

public void setStatus(String status) { this.status = status; }

public double getDownloadSpeedMbps() { return downloadSpeedMbps; }

public void setDownloadSpeedMbps(double downloadSpeedMbps) {
this.downloadSpeedMbps = downloadSpeedMbps; }

public double getUploadSpeedMbps() { return uploadSpeedMbps; }

public void setUploadSpeedMbps(double uploadSpeedMbps) { this.uploadSpeedMbps =
uploadSpeedMbps; }
```

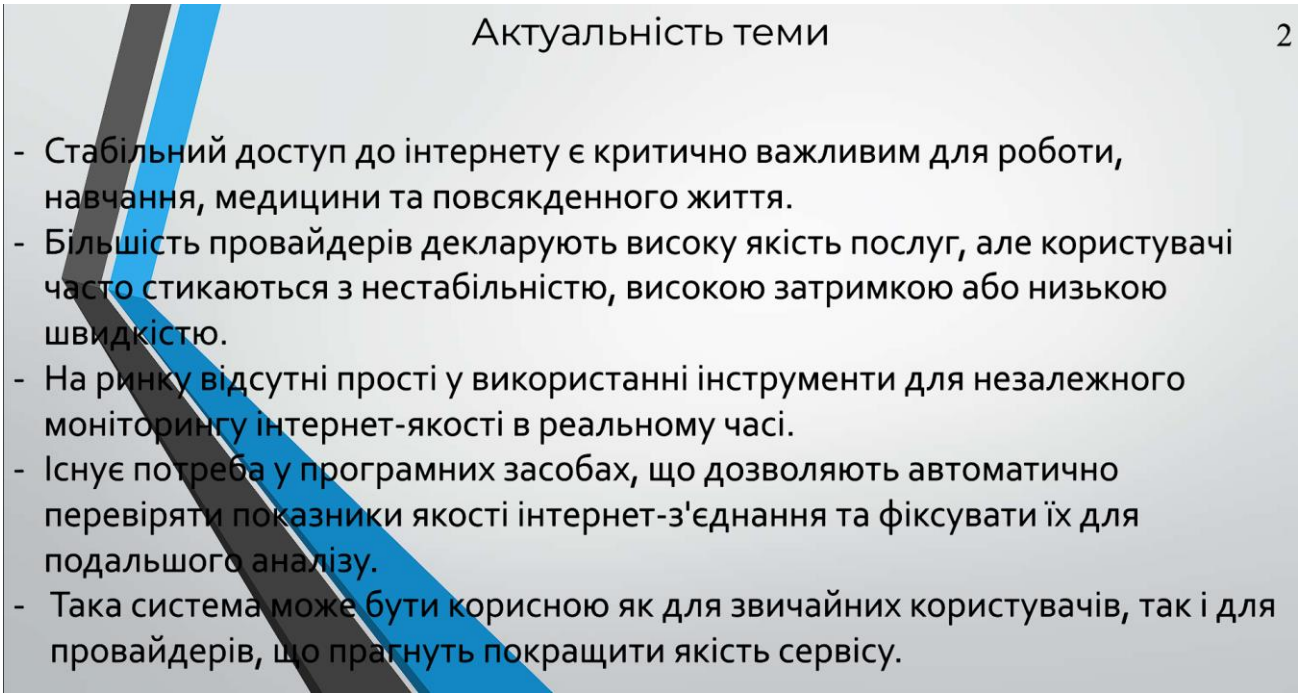
```
public int getPingMs() { return pingMs; }

public void setPingMs(int pingMs) { this.pingMs = pingMs; }

}

package com.example.internetquality.repository;

import com.example.internetquality.model.QualityTest;

import org.springframework.data.jpa.repository.JpaRepository;

public interface QualityTestRepository extends JpaRepository<QualityTest, Long>
{

}
```

ДОДАТОК Г

(обов’язковий)

ПРЕЗЕНТАЦІЙНІ СЛАЙДИ





Мета та завдання

Метою є створення ефективного інструменту для безперервного контролю якості інтернет-сервісів, який дозволить своєчасно реагувати на технічні проблеми, підвищити стабільність роботи системи та покращити якість користувацького досвіду. Система повинна мати модульну архітектуру, підтримувати масштабування, бути кросплатформною, зручна в налаштуванні та інтеграції з існуючими інфраструктурами компаній.

Етап	Завдання
Аналіз	Дослідити наявні аналоги систем і виявити потреби кінцевих користувачів
Вимоги	Визначити перелік функціональних та нефункціональних вимог до майбутнього застосунку
Архітектура	Обрати оптимальні технології та архітектурне рішення для реалізації системи
Проєктування	Сформувати структуру бази даних та описати логіку взаємодії між компонентами
Розробка	Розробити користувацький інтерфейс та реалізувати серверну частину з відповідною логікою
Функціонал	Здійснити реалізацію функціоналу для реєстрації, записів та збереження супровідної інформації
Тестування	Провести тестування працездатності системи й перевірку відповідності технічним вимогам
Оцінка	Оцінити результати реалізації та сформувати напрями подальшого вдосконалення.

Аналіз предметної області

- Якість інтернет-послуг визначається ключовими параметрами: швидкість завантаження та вивантаження, затримка (ping), стабільність з'єднання, а також наявність втрат пакетів.
- Існує низка стандартів для оцінки якості , проте більшість пересічних користувачів не мають доступу до професійного обладнання для вимірювання.
- Сучасні онлайн-сервіси (Speedtest, Fast.com) дозволяють оцінити швидкість, але не забезпечують автоматичний моніторинг, збереження історії результатів, або сповіщення про погіршення якості.
- Програмні системи моніторингу переважно орієнтовані на великі компанії або операторів зв'язку, що ускладнює їх використання у побутових умовах.
- Це створює передумови для розробки зручної, доступної та адаптивної системи, здатної проводити регулярні вимірювання та візуалізувати динаміку зміни показників.

Аналіз предметної області

Порівняння функцій вже існуючих застосунків

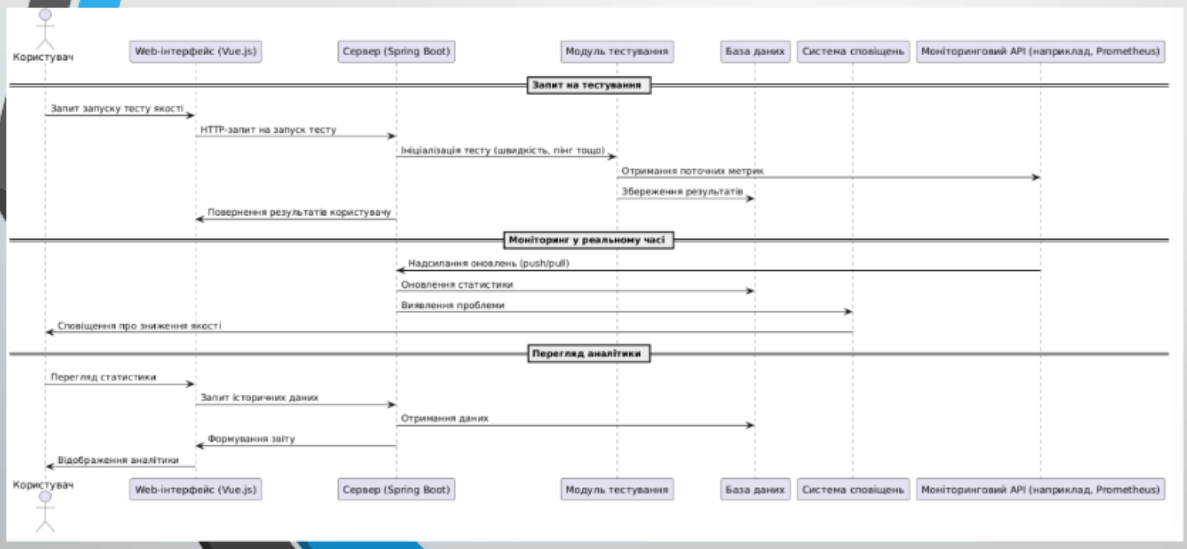
Функціонал	PRTG Network Monitor	Nagios XI	Zabbix
Моніторинг трафіку	+	+	+
Аналіз комутаторів	+/-	+	+
Визначення MAC-адреси	-	-	+
Оцінка довжини кабелю	-	-	-
Перевірка лінку	+	+	+
Інформація про тариф	-	-	-
Інтуїтивно зрозумілий інтерфейс	-	-	-
Перевірка помилок на кабелі	-	-	-

Аналіз предметної області

Функціональні та не функціональні вимоги

Функціональні вимоги	Нефункціональні вимоги
Збір даних із мережевого обладнання та бази абонентів	Висока продуктивність і обробка даних у реальному часі
Проведення автоматизованих тестів (швидкість, затримка, втрата пакетів)	Масштабованість для підтримки великої кількості користувачів
Формування звітів про якість інтернет-послуг	Надійність і відмовостійкість системи
Система сповіщень про критичні події та відхилення	Захищеність даних (аутентифікація, шифрування)
Веб-інтерфейс для перегляду аналітики	Зручність і інтуїтивність користувацького інтерфейсу
Інтеграція з зовнішніми сервісами через API	Підтримка історії змін і можливість аналізу трендів

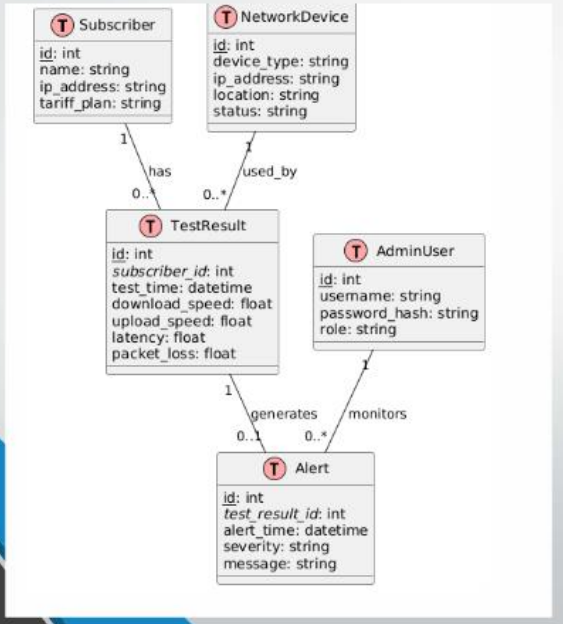
Проектування системи



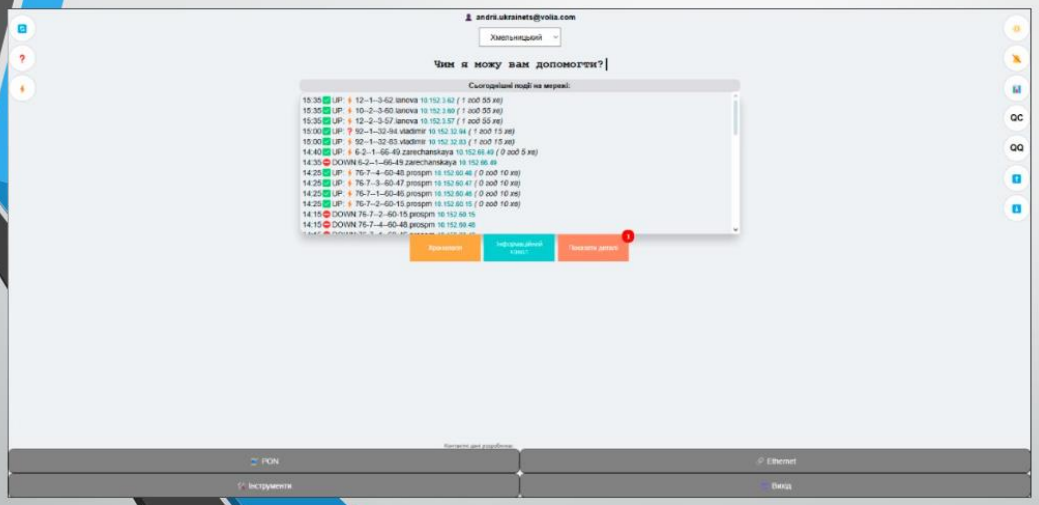
Діаграма послідовності

Проектування системи

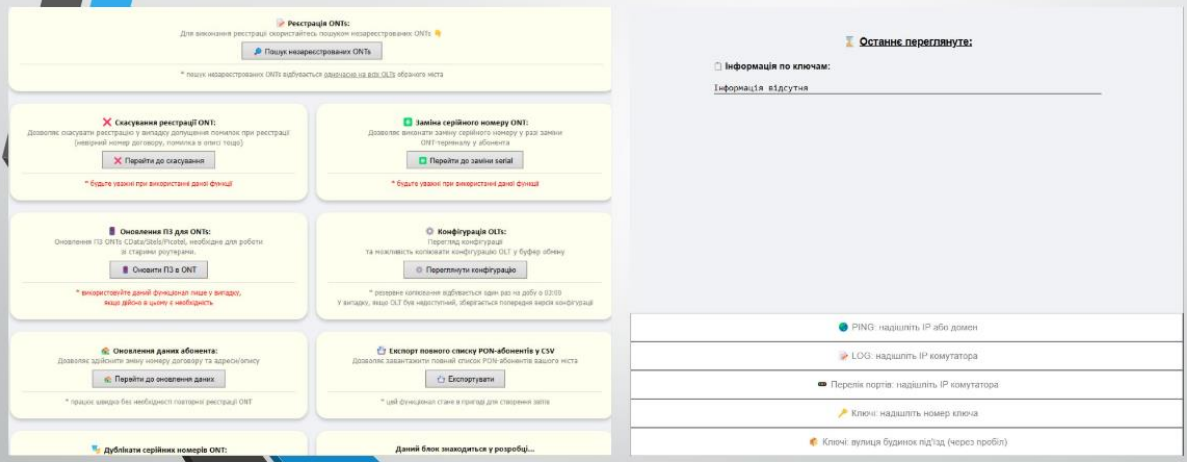
ER-діаграма



Проектування застосунку



Проектування застосунку



Проектування мобільного застосунку

Технології, які використовувались при розробці

Vue JS

Spring boot

Java

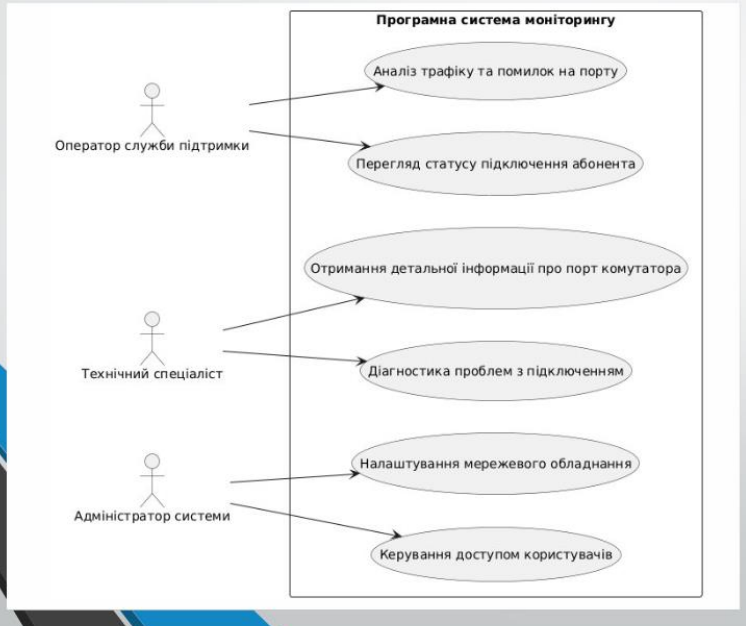
Vue.js був обраний для розробки клієнтської частини завдяки його:

- простоті у використанні та швидкому входженню в розробку;
- реактивному підходу, що дозволяє легко оновлювати інтерфейс без перезавантаження сторінки;
- гнучкості та широкій екосистемі компонентів для побудови зручного UI;
- хорошій підтримці української мови в інтерфейсі через бібліотеки локалізації.

Java (Spring Boot) став основою серверної частини, оскільки:

- це стабільна, безпечна та широко підтримувана мова для створення веб-сервісів;
- Spring Boot спрощує створення REST API, інтеграцію з базою даних і масштабування;
- має вбудовану підтримку безпеки, логування, обробки помилок;
- зручно підходить для роботи з багатопотоковими задачами, такими як обробка мережевих запитів чи тестів у фоновому режимі.

Проектування застосунку



Вимоги до технічних та програмних засобів

Категорія	Вимога
ОС пристрою	Windows 10+, Linux, macOS, IOS, Android 8
Процесор	64-бітний, мінімум 2 ядра
Оперативна пам’ять	Не менше 2 ГБ
Вільна пам’ять	Щонайменше 150 МБ
Інтернет-з’єднання	Стабільне Wi-Fi або мобільний інтернет (3G+)
Системне ПЗ	Android OS / iOS / Windows з любим браузером
Доступ до ресурсів	Мережа
Інтернет-залежність	Потрібне постійне з’єднання з інтернетом

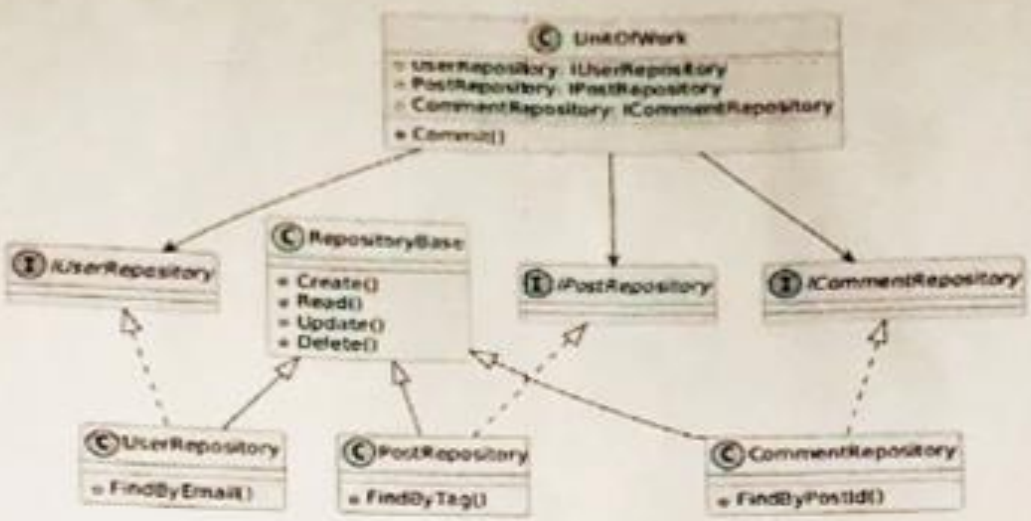
Тестування

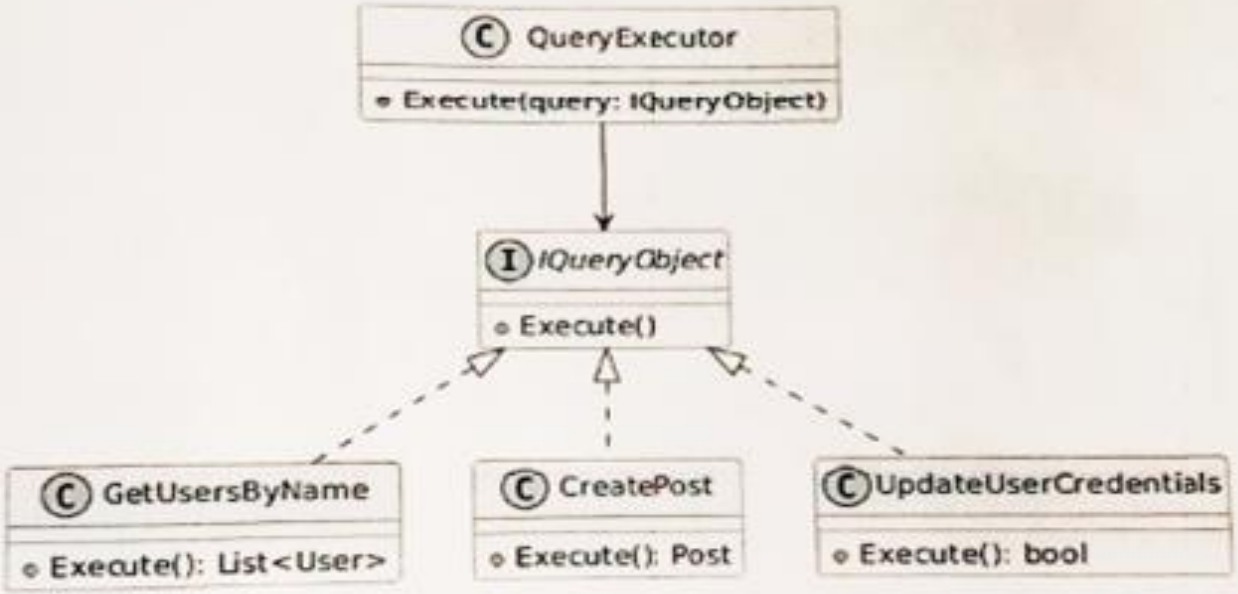

```
================== unit tests: core logic ==================
[PASS] com.qamonitor.tests.QualityCheckServiceTest.testLatencyCalculation
[PASS] com.qamonitor.tests.QualityCheckServiceTest.testPacketLossDetection
[PASS] com.qamonitor.tests.QualityCheckServiceTest.testJitterComputation
[PASS] com.qamonitor.tests.QualityCheckServiceTest.testThresholdValidation
[PASS] com.qamonitor.tests.AlertServiceTest.testTriggerAlertForHighLatency

================ integration tests: database ===============
[PASS] com.qamonitor.tests.DatabaseIntegrationTest.testMetricStorageToInfluxDB
[PASS] com.qamonitor.tests.DatabaseIntegrationTest.testAlertPersistenceInPostgres
[PASS] com.qamonitor.tests.DatabaseIntegrationTest.testUserPreferencesMapping
[PASS] com.qamonitor.tests.DatabaseIntegrationTest.testNotificationLogInsertion

==================== API tests ====================
[PASS] com.qamonitor.tests.ApiTest.testGetMetricsEndpoint
[PASS] com.qamonitor.tests.ApiTest.testPostNewThreshold
[PASS] com.qamonitor.tests.ApiTest.testUnauthorizedAccess
[PASS] com.qamonitor.tests.ApiTest.testMetricsQueryParams

=================== UI test results ===================
[PASS] com.qamonitor.tests.DashboardUITest.testDashboardLoad
[PASS] com.qamonitor.tests.DashboardUITest.testLiveGraphRendering
[PASS] com.qamonitor.tests.SettingsUITest.testThresholdInputValidation
[PASS] com.qamonitor.tests.SettingsUITest.testSaveUserSettings
[PASS] com.qamonitor.tests.NotificationUITest.testAlertVisibility
```

Висновки

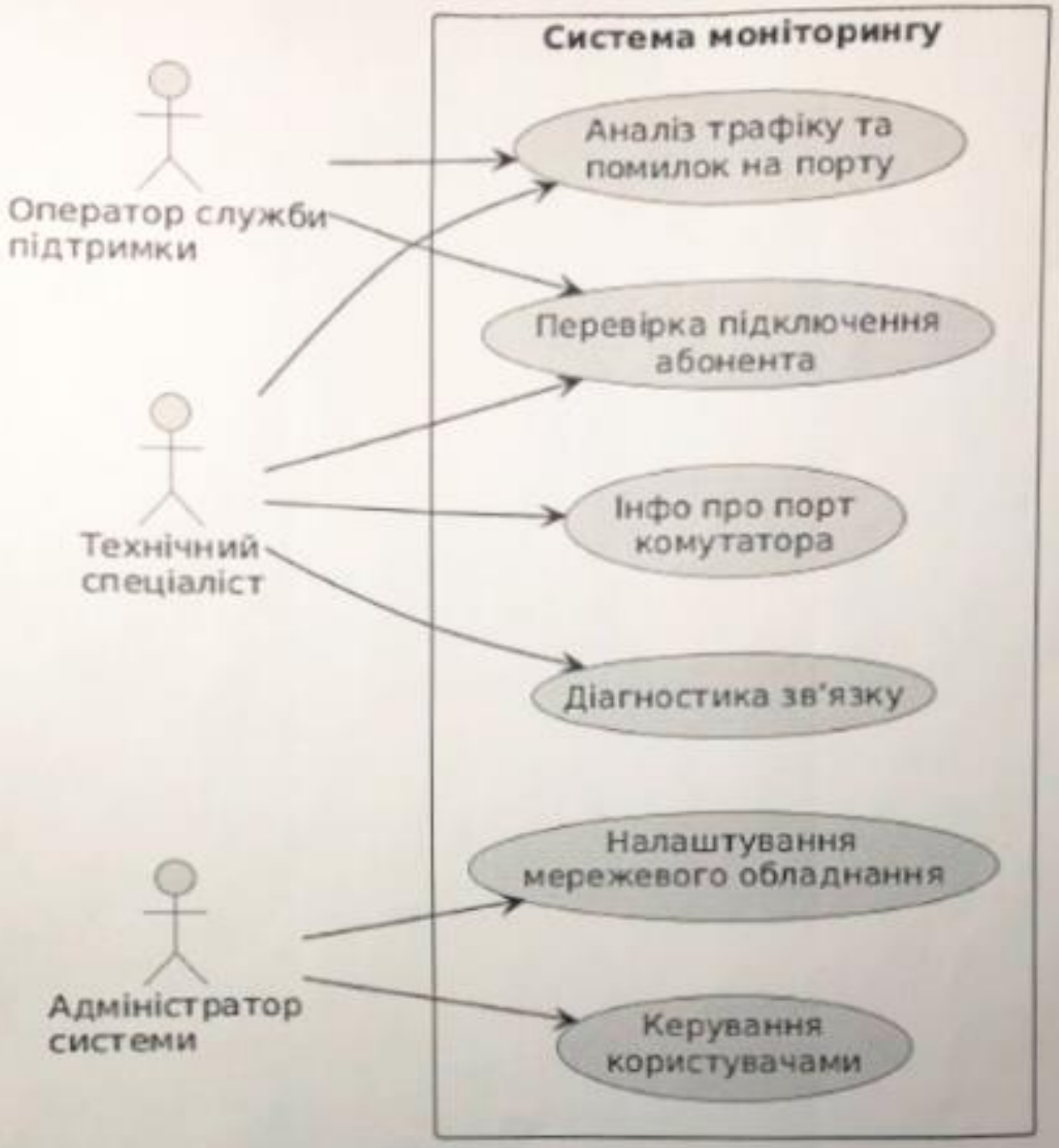
Поставлене завдання	Вимога
Провести аналіз предметної області та аналогів	Проаналізовано існуючі сервіси та інструменти моніторингу якості інтернету
Визначити вимоги до програмної системи	Сформульовано функціональні та нефункціональні вимоги до системи
Розробити архітектуру системи моніторингу	Побудовано архітектуру клієнт-серверної системи з фронтендом на Vue.js і бекендом на Spring Boot
Реалізувати функціонал тестування якості інтернету	Створено модулі вимірювання швидкості, затримки, втрат пакетів
Забезпечити візуалізацію результатів	Реалізовано зберігання даних у базі даних з можливістю подальшого аналізу
Додати підтримку автоматичного тестування	Реалізовано періодичне фонове тестування з фіксацією результатів у базі даних
Забезпечити зручність користувача	Створено простий вебінтерфейс з авторизацією та доступом до статистики
Забезпечити можливість масштабування системи	Система спроєктована з урахуванням масштабованості: використано мікросервісну архітектуру та Envoy Proxy

ГРАФІЧНІ МАТЕРІАЛИ





					КвРІПЗ.2101091.01.17.Е8	
					Програмна система тестування та моніторингу якості інтернет послуг	Літ. / Маса. / Масштаб
Змн.	Арк.	№ докум.	Підпис	Дата		
Розробив		Українець А.Д.		06.06		
Керівник		Форкун Ю.В.			Логічна модель бази даних	Аркуш 1 / Аркушів 2
Н. Контр.		Бедратюк Г.І.		06.06		
Зав. каф.		Бедратюк Л.П.		06.06		ХНУ, ІПЗ-21-1



					КвРІПЗ.2101082.01.10.Е8	
					Програмна система тестування та моніторингу якості інтернет послуг Логічна модель бази даних	Літ. / Маса / Масштаб
Змн.	Арк.	№ докум.	Підпис	Дата		
Розробив		Українець А.Дю				Аркуш 1 / Аркушів 2
Керівник		Форкун Ю.В.				
Н. контр.		Бедратюк Г.І.				ХНУ, ІПЗ-21-1
Зав. каф.		Бедратюк Л. П.				

СУПРОВІДНІ ДОКУМЕНТИ

Завідувачу кафедри інженерії програмного
забезпечення проф. Леоніду БЕДРАТЮКУ
здобувача вищої освіти
України Андрія Дмитровича
факультет ІТ, IV курс, група ІПЗ-21-1

ЗАЯВА

З правилами чинного Положення про систему забезпечення академічної доброчесності в Хмельницькому національному університеті, згідно з яким виявлення академічного плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту і застосування заходів дисциплінарної та академічної відповідальності, ознайомлений. Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на наявність академічного плагіату оповіщений та надаю свою згоду на обробку й збереження університетом моєї роботи в інституційному репозитарії Хмельницького національного університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-обчислювального комплексу StrikePlagiarism та/або програмно-технічного засобу AntiPlagiarism і використання роботи для виявлення академічного плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення текстових збігів у роботах.

Робота надається для перевірки в електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

05.06.2025
дата

підпис

Anti-Plagiarism (UA) v-15.281 Educational

The maximum coincidence with one document 1.0%

Dictionaries check: en_US, ru_RU, ua_UA. **Errors in the documents: 13%**

	Document		Sum coincidence on the DB	
	Symbols	Lexemes	Symbols	Lexemes
ID: 244348 Title: БКР_Програмна система моніторингу та тестування якості інтернет послуг Added in a DB: 2025-06-09 Authors: Українець Андрій Heads: Форкун Ю.В., кад. техн.наук, доцент Consultants: Opponents:	91478	793	3994 (4%)	63 (8%)

Plagiarism sources

ID	Description	Plagiarism presence in the document	
		Symbols	Lexemes

Протокол аналізу звіту подібності науковим керівником

Заявляю, що я ознайомився (-лась) з Повним звітом подібності, який був згенерований Системою виявлення і запобігання плагіату щодо роботи:

Автор: Українець

Співавтор:

Назва: БКР_Програмна система моніторингу та тестування якості інтернет послуг Українець

Науковий керівник:

Підрозділ: Кафедра інженерії програмного забезпечення

Коефіцієнт подібності 1:7.9%

Коефіцієнт подібності 2:3%

Мікропробіли: 0

Заміна букв: 3

Інтервали: 0

Білі знаки: 29

Дата створення звіту: 2025-06-07 14:04:13.0

Після аналізу Звіту подібності констатую наступне:

☑ **Запозичення, виявлені в роботі є законними і не є плагіатом. Рівень подібності не перевищує допустимої межі. Таким чином робота незалежна і приймається.**

☐ **Запозичення не є плагіатом, але перевищено граничне значення рівня подібностей. Таким чином робота повертається на доопрацювання.**

☐ **Виявлено запозичення і плагіат або навмисні текстові спотворення (маніпуляції), як передбачувані спроби укриття плагіату, які роблять роботу невідповідною вимогам законодавства (Ст. 32. ЗУ Про вищу освіту, пункт 3.1, Ст. 42. ЗУ Про освіту) та вимог НАЗЯВО (Критерій 5), а також кодексу етики і процедурам. Таким чином робота не приймається.**

Обгрунтування:

06.06.2025

Дата

експерт

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

освітнього ступеня «Бакалавр»

Дипломник Українець Андрій Дмитрович

Тема Програмна система тестування та моніторингу якості інтернет послуг

Спеціальність 121 – Інженерія програмного забезпечення

Обсяг кваліфікаційної роботи:

Кількість листів креслень 2 кількість сторінок записки 60

1.Короткий зміст пояснювальної записки та прийнятих рішень У кваліфікаційній роботі розглянуто створення веб-застосунку для перевірки якості надання інтернет послуг. Проведено аналіз предметної області, сформульовано функціональні та нефункціональні вимоги, обґрунтовано вибір архітектурних та технологічних рішень. Реалізовано функціональний мобільний застосунок із можливістю запису на прийом, ведення профілю тварини, збереження медичних документів і нагадувань про догляд. Проведено тестування, яке підтвердило коректну роботу програмного забезпечення.

2. Висновок про відповідність роботи поставленому завданню Кваліфікаційна робота виконана відповідно до поставленого завдання та з дотриманням всіх вимог.

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки та передових методів роботи У вступі обґрунтовано актуальність теми, сформульовано мету, завдання, об'єкт і предмет дослідження. У першому розділі проведено аналіз предметної області, розглянуто наявні сервіси, виявлено їхні переваги та недоліки, сформульовано функціональні та нефункціональні вимоги до застосунку. У другому розділі спроєктовано архітектуру системи, побудовано логічну модель даних, ER-діаграму та розроблено інтерфейс з урахуванням вимог UX/UI. У третьому розділі реалізовано функціонал застосунку за допомогою VueJS та SpringBoot: текст кабеля, перегляд комутатора, порта та помилок на ньому, інформація про тариф, баланс та трафіку. Проведено модульне та інтеграційне тестування. Результатом роботи став сучасний, зручний і масштабований веб-застосунок, що відповідає поставленим вимогам і має потенціал для подальшого розвитку.

4. Позитивні сторони роботи Тематика кваліфікаційної роботи є актуальною, оскільки є потреба у зручному цифровому рішенні для організацій та користувачів. Робота вирізняється всебічним аналізом аналогів, продуманим архітектурним проєктуванням, використанням сучасних технологій, а також орієнтацією на зручність і потреби кінцевого
користувача

5. Негативні сторони роботи дизайн застосунку не є прогресивним та не має анімацій або гарного оформлення.

6. Оцінка графічного оформлення та пояснювальної записки Графічне оформлення виконано відповідно до теми кваліфікаційної роботи та подано у вигляді діаграм і рисунків. Пояснювальна записка оформлена згідно вимог чинних стандартів.

7. Відгук про кваліфікаційну роботу в цілому Кваліфікаційна робота заслуговує позитивної оцінки. Матеріал пояснювальної записки структурований, послідовний, чіткий та простий, що дозволяє чітко зрозуміти викладений матеріал у рамках тематики проектування. Графічний матеріал дає можливість наочно побачити деталі проектування системи.

8. Інші зауваження

9. Оцінка кваліфікаційної роботи Кваліфікаційна робота виконана у повному обсязі, відповідає поставленій задачі та заслуговує на оцінку «добре».

РЕЦЕНЗЕНТ Князь Юрій, к.т.н, завідувач кафедри кібербезпеки, ХНУ

"10" 06 2025 р. _____
(підпис)

РІШЕННЯ ЕКСПЕРТНОЇ КОМІСІЇ
КАФЕДРИ ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Назва кваліфікаційної роботи Програмна система тестування та моніторингу якості інтернет послуг
Автор Українець Андрій Дмитрович
Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»
Рівень вищої освіти Перший (бакалаврський)
Спеціальність 121 «Інженерія програмного забезпечення»
Науковий керівник: Форкун Ю.В., канд. техн. наук, доцент
На основі аналізу кваліфікаційної роботи на дотримання вимог академічної доброчесності (у т.ч. відсутності ознак академічного плагіату) з урахуванням результатів перевірки роботи спеціалізованим програмним засобом(ами) комісія зробила такий висновок:

№	Висновок	Позначка про відповідність
1	Ознаки академічного плагіату	
1.1	Запозичення, виявлені в роботі, є законними і не є академічним плагіатом (далі – зазначаються підстави віднесення запозичень до правомірних, якщо потрібно). Робота приймається до захисту.	**відповідає**
1.2	Виявлені запозичення не є академічним плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи (далі – зазначаються детальні та аргументовані підстави віднесення запозичень до правомірних). Робота приймається до захисту, але має бути відкоригована.	
1.3	Виявлені запозичення не є академічним плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. Робота може бути допущена до захисту після того як буде відкоригована та доопрацьована і успішно пройде повторну перевірку на академічний плагіат.	
1.4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття текстових запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	
2	Інші види порушень академічної доброчесності	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) у тексті кваліфікаційної роботи системами перевірки на плагіат виявлено схожість з деякими документами в частині загальновживаних обов'язкових словосполучень у стандартних бланках (титулка, завдання, анотація, відомість документів), у структурі змісту, назвах розділів/підрозділів тощо, у назвах публікацій та у переліку джерел посилання;

2) в якості запозичень системою було зафіксовано деякі послідовності вихідного коду і посиланання на бібліотеки, які є стандартними мовними конструкціями програмування та не можуть розглядатися як об'єкт авторських прав і, відповідно, їх порушення;

3) усі запозичення є фрагментарними або мають належним чином оформленні посилання;

4) виявлені модифікації тексту не впливають на відсоток схожості.

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів ідентичності/ схожості, складає 3,8%, що, з урахуванням наведених обгрунтувань, відповідає характеру теми і свідчить на користь кваліфікаційної роботи.

Дата 5.06.25

Завідувач кафедри — Підпис — Леонід БЕДРАТЮК (Ім'я, ПРІЗВИЩЕ)

Гарант освітньої програми — Підпис — Леонід БЕДРАТЮК (Ім'я, ПРІЗВИЩЕ)

Керівник кваліфікаційної роботи — Підпис — Юрій ФОРКУН (Ім'я, ПРІЗВИЩЕ)