Хмельницький національний університет
Факультет інформаційних технологій
Кафедра комп'ютерної інженерії та інформаційних систем

КВАЛІФІКАЦІЙНА РОБОТА

Програмно-технічний засіб моделювання та аналізу поведінки комп'ютерних систем у мережевому середовищі
Назва теми

Рівень вищої освіти *перший (бакалаврський)*

Галузь знань *12 «Інформаційні технології»*
Шифр, назва

Спеціальність *123 «Комп'ютерна інженерія»*
Шифр, назва

Освітня програма *«Комп'ютерна інженерія та програмування»*
Назва

Шифр КвРКІ 2301112.23.01.16 ПЗ

Виконав здобувач ІІІ курсу, гр. КІ2с-23-1 _____ Володимир КОТЮК
Підпис Ініціали, прізвище

Керівник _____ Володимир ГРИГА
Науковий ступінь, учене звання Підпис Ініціали, прізвище

Нормоконтролер _____ Сергій ЛИСЕНКО
Науковий ступінь, учене звання Підпис Ініціали, прізвище

До захисту допускаю:
завідувач кафедри КІІС _____ Ольга ПАВЛОВА
«04» червня 2026 р. Підпис Ініціали, прізвище
дата

Хмельницький 2026

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ІНФОРМАЦІЙНИХ СИСТЕМ

Рівень вищої освіти ПЕРШИЙ (БАКАЛАВРСЬКИЙ)

Галузь знань 12 ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ

Спеціальність 123 КОМП'ЮТЕРНА ІНЖЕНЕРІЯ

Освітня програма «КОМП'ЮТЕРНА ІНЖЕНЕРІЯ ТА ПРОГРАМУВАННЯ»

ЗАТВЕРДЖУЮ
Завідувачка кафедри КІІС
Ольга ПАВЛОВА
" 10 " 01 2026 р.

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

Котюку Володимиру Олександровичу
Прізвище, ім'я, по батькові студента

1. Тема проекту (роботи) Програмно-технічний засіб моделювання та аналізу поведінки комп'ютерних систем у мережевому середовищі
Керівник проекту (роботи) Грига Володимир Михайлович, к.т.н., доцент.
Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 20.01.2026 р. № 7

2. Термін подання здобувачем роботи на кафедру 01.06.2026 р.

3. Вихідні дані до роботи Завдання на кваліфікаційну роботу

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)
Аналіз предметної області функціонування комп'ютерних систем у мережевому середовищі та постановка задачі щодо розроблення програмно-технічного засобу
Проєктування архітектури програмно-технічного засобу моделювання, збору метрик, збереження даних і аналізу результатів моделювання
Програмна реалізація серверної та клієнтської частин, реалізація сценаріїв моделювання, аналізу результатів, ML-аналізу та тестування роботи системи

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)
Структурна схема програмно-технічного засобу
Блок-схема роботи візуального конструктора топології
Блок-схема алгоритму моделювання поведінки системи

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання «_10_»_____01_____2026 р.

КАЛЕНДАРНИЙ ПЛАН

№з/п	Назва етапів (розділів) дипломного проекту (роботи)	Термін виконання етапів проекту (роботи)	Примітка
1	Вибір напряму дослідження та узгодження тематики кваліфікаційної роботи з керівником	10.01.2026	виконано
2	Ознайомлення з предметною областю; формулювання мети та задач дослідження; визначення об'єкта та предмета дослідження	01.02.2026	виконано
3	Робота над розділом 1 – дослідження предметної області та постановка задачі	01.03.2026	виконано
4	Робота над розділом 2 – проектування програмно-технічного засобу моделювання та аналізу поведінки комп'ютерних систем у мережевому середовищі	01.04.2026	виконано
5	Робота над розділом 3 – програмна реалізація, тестування та аналіз роботи програмно-технічного засобу	29.04.2026	виконано
6	Оформлення пояснювальної записки згідно вимог	25.05.2026	виконано
7	Попередній захист ВКР	26.05.2026	виконано
8	Захист ВКР на засіданні ЕК	Червень 2026 року	

Здобувач _____ Володимир КОТЮК
Підпис Ім'я, ПРІЗВИЩЕ

Керівник кваліфікаційної роботи _____ Володимир ГРИГА
Підпис Ім'я, ПРІЗВИЩЕ

№ рядка	формат	Позначення	Найменування	Кіл. листів	№ екз	Примітка
			Текстові документи			
1		КвРКІ 2301112.23.01.16 ПЗ	Пояснювальна записка	59		
			Графічні матеріали			
2		КвРКІ 2301112.23.01.16 Е1	Структурна схема програмно-технічного засобу	1		
3		КвРКІ 2301112.23.01.16 Е2	Блок-схема роботи візуального конструктора топології	1		
4		КвРКІ 2301112.23.01.16 Е3	Блок-схема алгоритму моделювання поведінки системи	1		

Зм	Арк	№ докум	Підпис	Дата	КвРКІ 2301112.23.01.16 ВП		
Розробив		Котюк		01.06.26	Програмно-технічний засіб моделювання та аналізу поведінки комп'ютерних систем у мережевому середовищі. Відомість проекту	Літера	Аркуш	Аркушів
						У	1	1
Перевір.		Грига		01.06.26		ХНУ, КІ2с-23-1		
Н. контр.		Лисенко		01.06.26				
Затв.		Павлова		01.06				

АНОТАЦІЯ

Тема кваліфікаційної роботи: «Програмно-технічний засіб моделювання та аналізу поведінки комп'ютерних систем у мережевому середовищі».

Автор роботи: Володимир КОТЮК.

Керівник роботи: Володимир ГРИГА.

Пояснювальна записка: 59 с., 13 рис., 9 табл., 4 дод., 49 джерел.

Графічна частина: 3 креслення.

АНАЛІЗ, БАЗА ДАНИХ, ВЕБЗАСТОСУНОК, ВУЗОЛ, КАНАЛ ЗВ'ЯЗКУ, КОМП'ЮТЕРНА СИСТЕМА, МАШИННЕ НАВЧАННЯ, МЕРЕЖЕВЕ СЕРЕДОВИЩЕ, МОДЕЛЮВАННЯ, ТОПОЛОГІЯ.

Кваліфікаційна робота бакалавра присвячена розробці та дослідженню програмно-технічного засобу моделювання та аналізу поведінки комп'ютерних систем у мережевому середовищі. Актуальність теми зумовлена необхідністю попереднього оцінювання стабільності комп'ютерних систем за різних сценаріїв навантаження, зміни параметрів вузлів і каналів зв'язку.

Метою роботи є проєктування, реалізація та тестування веборієнтованого засобу для створення моделі комп'ютерної системи, налаштування її топології, запуску моделювання та аналізу отриманих результатів. Для досягнення поставленої мети було виконано аналіз існуючих підходів до моделювання і навантажувального тестування, розроблено архітектуру програмного засобу, спроєктовано структуру моделі системи, реалізовано серверну і клієнтську частини, конструктор топології, модуль аналізу результатів.

Підпис здобувача

30.05.2026

Дата

ЗМІСТ

ВСТУП

Актуальність дослідження. Комп’ютерні системи у мережевому середовищі часто працюють як набір взаємопов’язаних вузлів: клієнтських груп, маршрутизаторів, серверів застосунків, кеш-серверів, баз даних і каналів зв’язку між ними. Для користувача нестабільність такої системи не завжди означає повну відмову. Частіше система залишається доступною, але відповідає повільно, частина запитів завершується помилкою, а окремі вузли або канали поступово переходять у критичний стан.

У роботі розглядається поведінка комп’ютерної системи за різних сценаріїв навантаження. Зростання затримки, втрата пакетів, перевантаження ресурсів або збільшення частки помилок можуть спричинити перехід системи в нестабільний режим.

Метою кваліфікаційної роботи є розроблення програмно-технічного засобу для створення моделі комп’ютерної системи у мережевому середовищі, налаштування її вузлів, каналів зв’язку, сценаріїв навантаження та подій, запуску моделювання, аналізу отриманих метрик і виявлення нестабільних режимів роботи з використанням алгоритмів машинного навчання.

Результатом роботи є вебзастосунок, до складу якого входять серверна частина, клієнтський інтерфейс, база даних, візуальний конструктор топології, модуль моделювання та засоби аналізу результатів. Розроблений засіб призначений для створення керованої моделі комп’ютерної системи, запуску сценаріїв навантаження та оцінювання впливу параметрів вузлів і каналів зв’язку на стабільність роботи системи.

Об’єктом дослідження є процес функціонування комп’ютерних систем у мережевому середовищі.

Предметом дослідження є методи моделювання навантаження, аналізу метрик вузлів і каналів зв’язку та виявлення нестабільних режимів роботи комп’ютерної системи.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ ПІДХОДІВ ДО МОДЕЛЮВАННЯ ТА АНАЛІЗУ ПОВЕДІНКИ КОМП'ЮТЕРНИХ СИСТЕМ У МЕРЕЖЕВОМУ СЕРЕДОВИЩІ

1.1 Особливості функціонування комп'ютерних систем у мережевому середовищі

Комп’ютерна система у мережевому середовищі може розглядатися як сукупність обчислювальних вузлів, мережевого обладнання, каналів передавання даних і програмних сервісів, які разом виконують обробку запитів. У загальному випадку така система не обмежується одним сервером. Навіть простий вебзастосунок може мати клієнтську частину, маршрутизатор, сервер застосунку, кеш, базу даних та зовнішні залежності. Основні принципи побудови комп’ютерних мереж і взаємодії вузлів описуються у класичних працях з мережевих технологій [1, 2].

Вузлом у межах цієї роботи вважається логічний або фізичний компонент комп’ютерної системи, який бере участь в обробці або передаванні запитів. Це може бути група клієнтів, маршрутизатор, комутатор, балансувальник, сервер застосунку, кеш-сервер, база даних або інший сервіс. Для кожного вузла важливими є його роль у системі, обсяг доступних ресурсів, продуктивність обробки запитів та обмеження за кількістю одночасних з’єднань.

Канал зв’язку описує взаємодію між двома вузлами. Він має пропускну здатність, базову затримку, коливання затримки, втрату пакетів, розмір черги та протокол взаємодії. На практиці запит проходить через кілька каналів, тому навіть один слабкий канал може вплинути на загальний час відповіді. Передавання даних у таких системах часто спирається на стек TCP/IP, де TCP відповідає за встановлення з’єднання, доставку сегментів і повторне передавання втрачених даних [3].

Для прикладних систем важливим є також протокол HTTP [4], який використовується для організації взаємодії за моделлю запит-відповідь [5, 6].

Однак у цій роботі HTTP розглядається не як об’єкт фактичної перевірки зовнішнього сервісу, а як один із можливих протоколів взаємодії між змодельованими компонентами. Це дозволяє описувати типові для вебсистем параметри: RPS, час відповіді, помилки, повторні запити та перевищення часу очікування. Структурну схему комп'ютерної системи у мережевому середовищі наведено на рисунку 1.1.

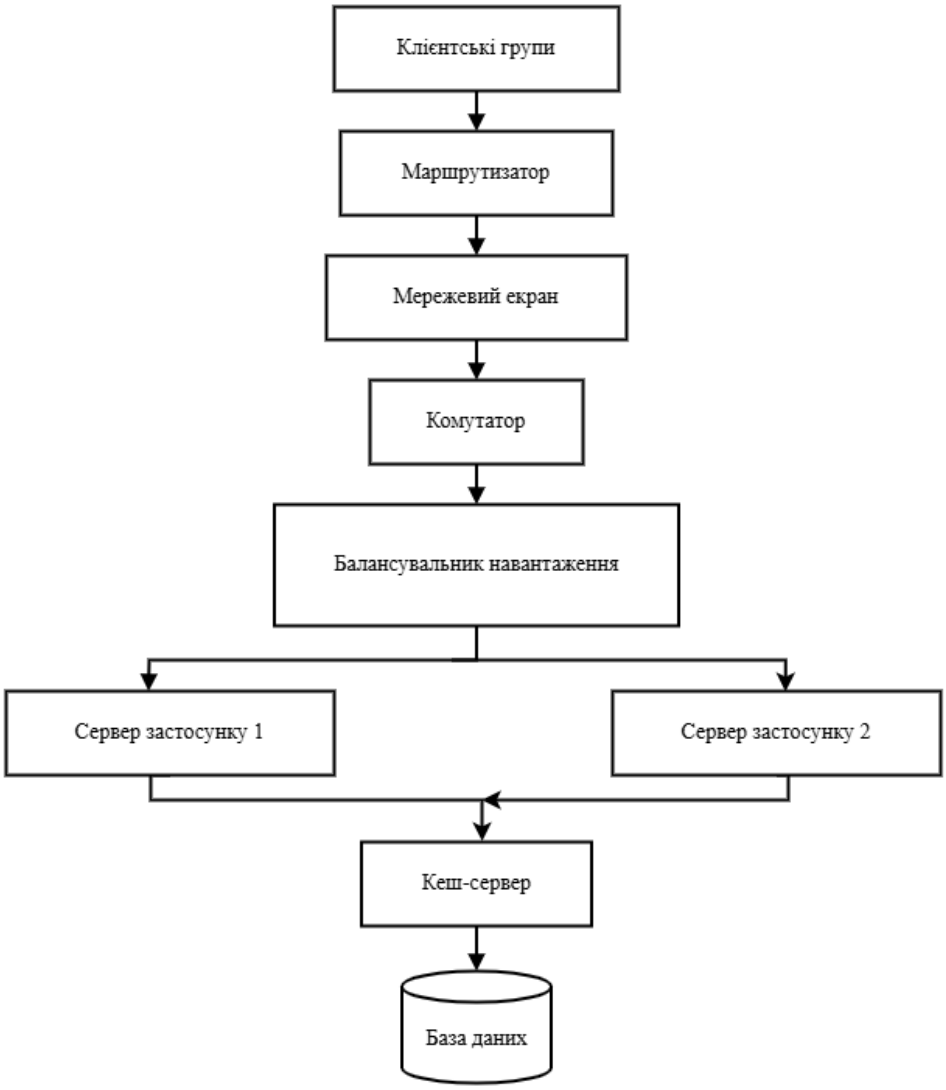


Рисунок 1.1 – Структурна схема комп'ютерної системи у мережевому середовищі

На рисунку 1.1 показано основні компоненти комп'ютерної системи у мережевому середовищі та зв'язки між ними. Якщо один із них стає

перевантаженим або канал між вузлами має погані характеристики, змінюється поведінка всієї системи.

Функціонування системи залежить від розподілу навантаження. Клієнтська група генерує запити, маршрутизатор або балансувальник передає їх далі, сервер застосунку виконує прикладну логіку, кеш зменшує кількість звернень до бази даних, а база даних повертає потрібні дані. Якщо один із цих елементів працює повільніше за інші, він стає вузьким місцем. Тому під час аналізу важливо враховувати не тільки середні значення метрик, а й стан кожного вузла та кожного каналу.

Стабільний режим означає, що система обробляє задане навантаження без перевищення критичних порогів. Нестабільний режим виникає тоді, коли хоча б один компонент починає працювати за межами допустимих значень: CPU або RAM наближаються до максимуму, збільшується затримка каналу, зростає втрата пакетів, з’являються помилки або різко падає пропускна здатність. Для виявлення таких переходів доцільно зберігати метрики в часі, а не обмежуватися одним підсумковим числом.

1.2 Аналіз мережевих навантажень та факторів деградації роботи систем

Стабільність комп’ютерної системи залежить від ресурсу вузлів, параметрів каналів зв’язку та профілю навантаження. Методи оцінювання продуктивності комп’ютерних систем зазвичай враховують навантаження, час відповіді, пропускну здатність і рівень використання ресурсів [7, 8]. Для цієї роботи важливо не тільки зафіксувати погіршення метрик, а й визначити можливу причину: критичний вузол, проблемний канал або невідповідний профіль навантаження.

Перевантаження CPU виникає тоді, коли кількість запитів або складність їх обробки перевищує обчислювальну здатність вузла. У результаті зростає час обробки, збільшуються черги, а частина запитів може завершуватися помилками.

Подібна ситуація виникає при нестачі RAM. Якщо активні з’єднання та буфери займають більшу частину пам’яті, система починає працювати нестабільно, а окремий вузол може перейти в критичний стан.

Важливим фактором є кількість активних з’єднань. Навіть якщо CPU і RAM ще не досягли критичного рівня, перевищення max connections може призвести до відмови частини запитів. Для серверів застосунків і баз даних це особливо помітно під час пікового або ступінчастого навантаження, коли кількість клієнтів зростає швидше, ніж система встигає обробляти попередні запити.

На рівні каналів зв’язку впливають пропускна здатність, затримка, втрата пакетів і коливання затримки. Обмеження пропускної здатності призводить до зростання використання каналу. Якщо трафік наближається до межі каналу, формуються черги, збільшується затримка, а частина пакетів може втрачатися. Втрата пакетів збільшує кількість повторних передавань і може викликати повторні запити на прикладному рівні. У підсумку система отримує додаткове навантаження саме в момент, коли вона вже працює гірше.

Окремо розглядається частка помилок. Вона показує частку запитів, які не були успішно оброблені. Зростання частки помилок може бути наслідком перевантаження вузла, недоступності залежності, високої затримки, втрат пакетів або перевищення часу очікування. У моделі цей показник не трактується окремо від інших метрик, а використовується разом із CPU, RAM, затримкою, втратою пакетів і пропускною здатністю. Для коректного визначення моменту деградації необхідно розглядати ці показники у взаємозв’язку. Наприклад, зростання затримки може бути пов’язане не лише з проблемами каналу зв’язку, а й із перевантаженням серверного вузла або збільшенням кількості активних з’єднань. Тому під час моделювання доцільно відстежувати зміну метрик у часі та визначати, які саме параметри першими наближаються до попереджувальних або критичних меж. Блок-схему переходу комп'ютерної системи до нестабільного режиму наведено на рисунку 1.2.

Рисунок 1.2 – Блок-схема переходу комп'ютерної системи до нестабільного режиму

На рисунку 1.2 показано послідовність переходу системи від стабільного режиму до нестабільного через зростання навантаження та перевищення порогових значень. Спочатку система переходить у режим попередження, а після

перевищення критичних порогів - у нестабільний режим. Основні фактори деградації комп'ютерної системи наведено в таблиці 1.1.

Таблиця 1.1 – Основні фактори деградації комп'ютерної системи

Фактор	Прояв у метриках	Можливий наслідок
Перевантаження CPU	зростання CPU до 90-100%	збільшення часу відповіді, помилки обробки запитів
Нестача RAM	RAM наближається до критичного порогу	відмова вузла або різке падіння продуктивності
Велика кількість з'єднань	активні з'єднання наближаються до максимальної кількості	відхилення нових запитів, черги, перевищення часу очікування
Висока затримка	збільшення затримки каналу	повільні відповіді та погіршення користувацького досвіду
Втрата пакетів	втрата частини пакетів	повторні запити, помилки, перевищення часу очікування
Високий RPS	швидке зростання кількості запитів	перевантаження серверного вузла або каналу
Обмеження пропускної здатності	використання каналу наближається до 100%	черги в каналі, падіння пропускної здатності
Відмова вузла	статус вузла переходить у failed	перенаправлення навантаження або недоступність частини системи

Дані таблиці 1.1 використовуються при формуванні критеріїв нестабільності. У програмному засобі ці фактори відображені через попереджувальні та критичні пороги для ресурсів вузлів, каналів та системних метрик.

1.3 Аналіз існуючих засобів моделювання, моніторингу та навантажувального тестування мережевих систем

Для визначення вимог до розроблюваного програмно-технічного засобу було розглянуто декілька груп існуючих рішень, які частково виконують подібні функції. До них належать системи моніторингу, засоби навантажувального тестування, емулятори мережевих умов та інструменти трасування.

Розглянуті засоби закривають лише окремі частини поставленої задачі. Одні з них призначені для збирання метрик, інші для створення навантаження, трасування запитів або моделювання мережевих умов. У цій роботі ці дії об’єднано в один робочий сценарій: створення моделі системи, налаштування топології, запуск моделювання, аналіз метрик і формування підсумкового звіту.

Аналіз існуючих рішень дав змогу визначити, які функції потрібно передбачити у власному програмно-технічному засобі. До них належать робота з вузлами і каналами зв’язку, задання навантаження, подій сценарію та критеріїв нестабільності, збереження результатів, аналіз метрик і формування PDF-звіту.

Prometheus [9] і Grafana [10] часто використовують для збирання часових рядів, побудови панелей і контролю метрик у працюючих системах. Вони добре підходять для довготривалого спостереження за інфраструктурою. Однак для поставленої задачі їх недостатньо, оскільки користувачеві потрібно не тільки переглядати метрики, а й самостійно створювати модель топології, змінювати параметри вузлів та каналів і запускати керовані сценарії навантаження.

JMeter [11], k6 [12] і Locust [13] призначені для навантажувального тестування. За їх допомогою можна створювати потік запитів і перевіряти, як

сервіс поводиться під навантаженням. Проте ці засоби переважно працюють із реальним об’єктом тестування. Вони не зберігають повну модель системи у вигляді вузлів і каналів зв’язку та не надають готової структури для аналізу критичного вузла або проблемного каналу саме у змодельованій топології.

Засоби ns-3 [14], Mininet [15] і tc netem [16] орієнтовані на нижчий рівень моделювання мережі. Вони дозволяють точніше працювати з пакетами, затримками, чергами та віртуальними мережевими топологіями. Для цієї роботи такий підхід є надмірним, оскільки мета полягає не в детальному пакетному моделюванні, а у створенні прикладної моделі комп’ютерної системи з вузлами, каналами, ресурсами й аналітичним результатом.

OpenTelemetry [17], Jaeger [18] та засоби класу ELK [19] використовуються для трасування, журналювання та спостережуваності. Вони допомагають аналізувати реальні потоки даних і шукати причини проблем у розподілених системах. Водночас вони не призначені для візуального конструювання навчальної або проєктної моделі системи з подальшим запуском контрольованих сценаріїв навантаження. Порівняння існуючих програмно-технічних засобів наведено в таблиці 1.2.

Таблиця 1.2 – Порівняння існуючих програмно-технічних засобів аналізу та моделювання мережевих систем

Клас засобів	Приклади	Призначення	Обмеження
Системи моніторингу	Prometheus, Grafana	збирання метрик, панелі, сповіщення	не створюють модель топології та не запускають кероване моделювання

Кінець таблиці 1.2

Навантажувальні засоби	JMeter, k6, Locust	генерація трафіку, перевірка продуктивності сервісів	не описують вузли й канали моделі
Засоби моделювання мережі	ns-3, Mininet, tc netem	емуляція або моделювання мережевих умов	складніші для швидкої роботи з прикладною моделлю системи
Засоби трасування та журналювання	OpenTelemetry, Jaeger, ELK	аналіз запитів, трас, журналів	не призначені для побудови сценарної моделі та формування підсумку за запуском

Отже, наявні інструменти можна використати для окремих етапів роботи з мережевими системами, але вони не дають цілісного сценарію, потрібного в цій кваліфікаційній роботі. Розроблюваний засіб орієнтований не на промисловий моніторинг і не на детальне пакетне моделювання, а на створення керованої моделі комп'ютерної системи з подальшим аналізом результатів моделювання.

У такому підході основна увага приділяється не точному відтворенню всіх мережевих процесів на рівні пакетів, а оцінюванню впливу параметрів вузлів, каналів зв'язку та навантаження на загальну стабільність системи.

Для реалізації такого веборієнтованого засобу доцільно використовувати окрему серверну частину, клієнтський інтерфейс і базу даних. Серверний API може бути реалізований засобами FastAPI [20], клієнтська частина - за допомогою React [21] і Vite [22], а дані можуть зберігатися у реляційній базі з доступом через ORM [23]. Для розгортання вебзастосунків можуть використовуватися Docker Compose [24], Render [25] і Vercel [26]. У цій роботі ці технології розглядаються як практичний набір для створення власного програмно-технічного засобу.

1.4 Аналіз методів машинного навчання для виявлення аномалій та нестабільних режимів роботи

Аналіз результатів моделювання можна виконувати пороговими правилами або методами машинного навчання. Порогові правила прості: якщо CPU перевищує критичне значення, вузол вважається критичним; якщо затримка або втрата пакетів перевищують поріг, канал вважається проблемним. Такий підхід зручний для пояснення результатів, але він не завжди показує загальну картину поведінки системи. Наприклад, окремі метрики можуть не перевищувати поріг, але разом формувати нестабільний режим.

Для групування схожих станів системи можна використати KMeans. Метод розділяє точки метрик на групи за схожістю ознак. У результаті можна виділити нормальний режим, режим підвищеного навантаження, мережеву деградацію або критичну нестабільність. Ідея методу KMeans була описана Дж. МакКвіном [27], а сучасна реалізація доступна у бібліотеці scikit-learn [28].

Для пошуку нетипових інтервалів доцільно використовувати Isolation Forest. Метод добре підходить для виявлення аномальних точок без попередньої розмітки даних. Його суть полягає в тому, що аномальні спостереження ізолюються швидше, ніж типові точки [29]. Реалізація Isolation Forest у scikit-learn дозволяє застосувати метод до коротких наборів метрик після моделювання [30].

PCA використовується як допоміжний метод зменшення розмірності. Він дозволяє подати багатовимірні метрики у вигляді двох координат для візуалізації або додаткового аналізу. Метод головних компонент описаний у роботах з багатовимірного аналізу даних [31], а відповідний інструмент також доступний у scikit-learn [32].

У роботі не використовується складне контрольоване навчання. Для нього потрібний розмічений набір даних, де кожна точка має правильний клас: нормальний стан, деградація, відмова тощо. Для сценаріїв короткого

моделювання такої історичної розмітки немає. Тому достатнім є поєднання порогових правил і неконтрольованих методів, які працюють із самими метриками запуску. Загальні можливості scikit-learn для таких задач описані в документації бібліотеки [33]. Порівняння методів аналізу телеметричних даних наведено в таблиці 1.3.

Таблиця 1.3 – Порівняння методів аналізу телеметричних даних

Метод	Призначення	Переваги	Обмеження
Порогові правила	швидке визначення попереджувального і критичного стану	прості для пояснення, не потребують навчання	не показують групи режимів роботи
KMeans	групування часових інтервалів за схожими метриками	дозволяє виділити режими роботи системи	кількість кластерів потрібно задавати або підбирати
Isolation Forest	пошук аномальних інтервалів	працює без розмітки, підходить для коротких запусків	аномалія не завжди означає реальну відмову
PCA	зменшення розмірності та допоміжний аналіз ознак	полегшує подання багатовимірних даних	не виконує класифікацію самостійно
Контрольоване навчання	класифікація за готовими прикладами	може давати точні результати за наявності даних	потребує якісної розмітки і великої історії спостережень

Порівняння показує, що кожен із методів має власну сферу застосування. Порогові правила зручні для пояснення попереджувальних і критичних станів, KMeans може використовуватися для виділення режимів роботи, Isolation Forest для пошуку нетипових часових інтервалів, а PCA для допоміжного подання багатовимірних ознак. Тому в роботі використано комбінований підхід, у якому порогові правила відповідають за технічне пояснення стану вузлів і каналів, а методи машинного навчання доповнюють аналіз результатів моделювання.

1.5 Постановка задачі

За результатами аналізу предметної області було сформульовано задачу розроблення веборієнтованого програмно-технічного засобу для моделювання та аналізу поведінки комп'ютерних систем у мережевому середовищі. Засіб має охоплювати повний цикл роботи з моделлю системи: створення конфігурації, налаштування топології, запуск моделювання, збереження метрик, аналіз результатів і формування рекомендацій.

Основною сутністю в системі є модель комп'ютерної системи. Вона повинна містити загальну інформацію про запуск, структуру топології, набір вузлів, канали зв'язку між ними, профіль навантаження, події сценарію, критерії нестабільності та результати моделювання. Такий підхід дає змогу розглядати систему не як набір окремих числових параметрів, а як пов'язану структуру, у якій стан одного вузла або каналу може впливати на загальний результат.

Режим моделювання обрано як основний, оскільки він дозволяє задавати параметри, які складно безпечно або стабільно відтворити в реальному середовищі. До таких параметрів належать кількість клієнтів, RPS, затримка, втрата пакетів, коливання затримки, обмеження пропускної здатності, час очікування відповіді, події відмови вузлів або зміни характеристик каналів. Це потрібно не для точного пакетного моделювання всієї мережі, а для керованого відтворення типових умов деградації комп'ютерної системи.

Для роботи з моделлю передбачено візуальний конструктор топології. У ньому користувач додає вузли різних типів, з'єднує їх каналами, змінює положення елементів на полотні та редагує параметри окремих компонентів. Для вузлів задаються CPU, RAM, кількість з'єднань, час обробки запиту та пороги переходу в попереджувальний або критичний стан. Для каналів зв'язку задаються пропускна здатність, затримка, втрата пакетів, коливання затримки, розмір черги та протокол взаємодії.

У системі також передбачено підтримку сценаріїв навантаження. Система повинна дозволяти задавати постійне навантаження, лінійне зростання, ступінчасте навантаження та користувацький профіль. Для ступінчастого навантаження мають використовуватися інтервали, у яких задаються часові межі та значення RPS. Це дає змогу відтворити ситуації, коли система спочатку працює стабільно, а потім поступово або різко переходить до перевантаження.

Крім профілю навантаження, у моделі повинні задаватися події сценарію. Події використовуються для перевірки поведінки системи при зміні умов під час моделювання. Наприклад, може бути задана відмова вузла, відновлення вузла, зменшення пропускної здатності каналу, збільшення затримки, зростання втрати пакетів або різкий приріст трафіку. Це дозволяє перевіряти не тільки рівномірне навантаження, а й аварійні або нестабільні стани.

Перед запуском моделювання система повинна перевіряти готовність моделі. Необхідно контролювати наявність основних компонентів, коректність каналів зв'язку, заповнення параметрів вузлів, вибір профілю навантаження та встановлення критеріїв нестабільності. Якщо модель неповна, користувач повинен отримати повідомлення про те, що саме потрібно виправити перед запуском.

Під час моделювання необхідно розраховувати системні, вузлові та канальні метрики. До системних метрик належать RPS, пропускна здатність, середня затримка, процентилі затримки, частка помилок, кількість успішних і помилкових запитів, кількість перевищень часу очікування та повторних запитів.

Для вузлів потрібно визначати використання CPU, RAM, кількість активних з’єднань, кількість оброблених і помилкових запитів та стан вузла. Для каналів потрібно розраховувати використання каналу, затримку, втрату пакетів, кількість відкинутих пакетів і стан каналу.

Важливою вимогою є збереження знімка топології під час запуску моделювання. До нього повинні входити координати вузлів, канали зв’язку, параметри компонентів і стан полотна. Це потрібно для того, щоб сторінка результатів показувала саме ту конфігурацію системи, яка існувала на момент запуску. Якщо після цього користувач змінює модель, попередні результати не повинні втрачати зв’язок зі старою топологією.

Результат моделювання повинен подаватися не тільки як набір таблиць або графіків. Користувач має отримати короткий підсумок: стан системи, рівень ризику, критичний вузол, проблемний канал, час деградації та основну причину переходу до нестабільного режиму. Завдяки цьому користувач може швидко визначити, який елемент системи став вузьким місцем.

Окремо потрібно реалізувати аналіз результатів із використанням алгоритмів машинного навчання. Порогові правила застосовуються для визначення попереджувальних і критичних станів, а методи машинного навчання використовуються для виявлення характерних режимів роботи та аномальних інтервалів. Для цього передбачено використання KMeans, Isolation Forest та PCA. Результати ML-аналізу мають доповнювати основний висновок, а не замінювати технічне пояснення метрик.

Нефункціональні вимоги пов’язані з простотою запуску і подальшого розгортання. Система повинна підтримувати локальну роботу без складної інфраструктури, використовувати SQLite для локального середовища та мати можливість переходу на PostgreSQL через змінну середовища. Клієнтська частина повинна працювати у браузері, а серверна частина повинна надавати API для роботи з моделями, результатами аналізу і PDF-звітами.

Межі розроблюваного засобу також визначено окремо. Він не розглядається як заміна Prometheus, Grafana, JMeter, k6, ns-3 або Mininet, оскільки ці інструменти мають інше основне призначення. У цій роботі система орієнтована на простіший прикладний сценарій: створення моделі комп'ютерної системи, задання контрольованого навантаження, отримання метрик, аналіз результатів і формування підсумку за результатами моделювання.

Вхідними даними для програмного засобу є параметри моделі системи: топологія, вузли, канали зв'язку, профіль навантаження, події сценарію, критерії нестабільності, тривалість моделювання та крок розрахунку. Вихідними даними є метрики моделювання, стан вузлів і каналів, рівень ризику, результати ML-аналізу, рекомендації, графіки, таблиці та PDF-звіт.

Критеріями успішності роботи програмного засобу є можливість створити модель системи, виконати сценарій моделювання, отримати різні результати для стабільного і критичного сценаріїв, визначити причину нестабільності та сформувати рекомендації. Результати повинні показувати різницю між стабільною і перевантаженою системою без додаткових ручних обчислень.

1.6 Висновки до першого розділу

У першому розділі розглянуто предметну область, у якій працює розроблюваний засіб: комп'ютерні системи, що функціонують у мережевому середовищі та залежать від стану вузлів, каналів зв'язку, ресурсів компонентів і характеру навантаження. Окремо виділено параметри вузлів, параметри каналів і системні метрики, які надалі використовуються під час моделювання та аналізу результатів.

Під час аналізу факторів деградації встановлено, що нестабільність не завжди проявляється як повна відмова системи. Частіше спочатку збільшується час відповіді, зростає кількість активних з'єднань, підвищується використання CPU або RAM, погіршуються параметри каналу, збільшується частка помилок

або виникають перевищення часу очікування. Через це для оцінювання стану системи недостатньо одного підсумкового показника. Потрібно аналізувати кілька пов’язаних метрик у часі.

Огляд існуючих засобів показав, що системи моніторингу, навантажувального тестування, моделювання мережі та трасування вирішують окремі частини задачі. Prometheus і Grafana зручні для збирання та перегляду метрик, JMeter, k6 і Locust використовуються для створення навантаження, ns-3, Mininet і tc netem дають змогу працювати з мережевими умовами, а OpenTelemetry, Jaeger і ELK застосовуються для трасування та журналювання. Водночас ці засоби не забезпечують у межах одного інтерфейсу повний цикл роботи, який потрібний у цій роботі: створення моделі системи, налаштування топології, запуск моделювання, аналіз метрик і формування рекомендацій.

Для аналітичної частини розглянуто порогові правила та методи машинного навчання. Порогові правила доцільні для пояснення попереджувальних і критичних станів, оскільки їх легко пов’язати з конкретними метриками вузлів і каналів. KMeans може використовуватися для виділення режимів роботи, Isolation Forest для пошуку аномальних інтервалів, а PCA для допоміжного аналізу багатовимірних ознак. Такий підхід дозволяє поєднати інженерну оцінку стану системи з автоматизованим аналізом результатів моделювання.

На основі проведеного аналізу сформульовано задачу розроблення веборієнтованого програмно-технічного засобу для створення моделі комп’ютерної системи, налаштування вузлів і каналів зв’язку, задання сценаріїв навантаження та подій, запуску моделювання і подальшого аналізу результатів. Для наступного етапу визначено основні вхідні параметри, очікувані вихідні дані, межі системи та критерії успішності. Окремо обґрунтовано потребу в збереженні знімка топології, щоб результати моделювання залишалися пов’язаними з тією конфігурацією системи, для якої вони були отримані.

2 ПРОЄКТУВАННЯ ПРОГРАМНО-ТЕХНІЧНОГО ЗАСОБУ МОДЕЛЮВАННЯ ТА АНАЛІЗУ ПОВЕДІНКИ КОМП'ЮТЕРНИХ СИСТЕМ

2.1 Загальна архітектура програмно-технічного засобу

Програмно-технічний засіб спроєктовано як вебзастосунок, що складається з клієнтського рівня, серверного рівня, бази даних і сервісних модулів [34, 35]. Клієнтський рівень забезпечує роботу інтерфейсу користувача, конструктора топології, сторінки результатів, таблиць, графіків і запуск формування PDF-звіту. Серверний рівень надає REST API для роботи з моделями, топологією, метриками, аналізом і звітами.

До сервісних модулів належать модуль керування моделями, модуль моделювання, ML-модуль і модуль PDF-звіту. Вони відповідають за збереження параметрів системи, розрахунок поведінки в часі, аналіз метрик і формування документа з результатами запуску. Структурну схему програмно-технічного засобу наведено на рисунку 2.1.

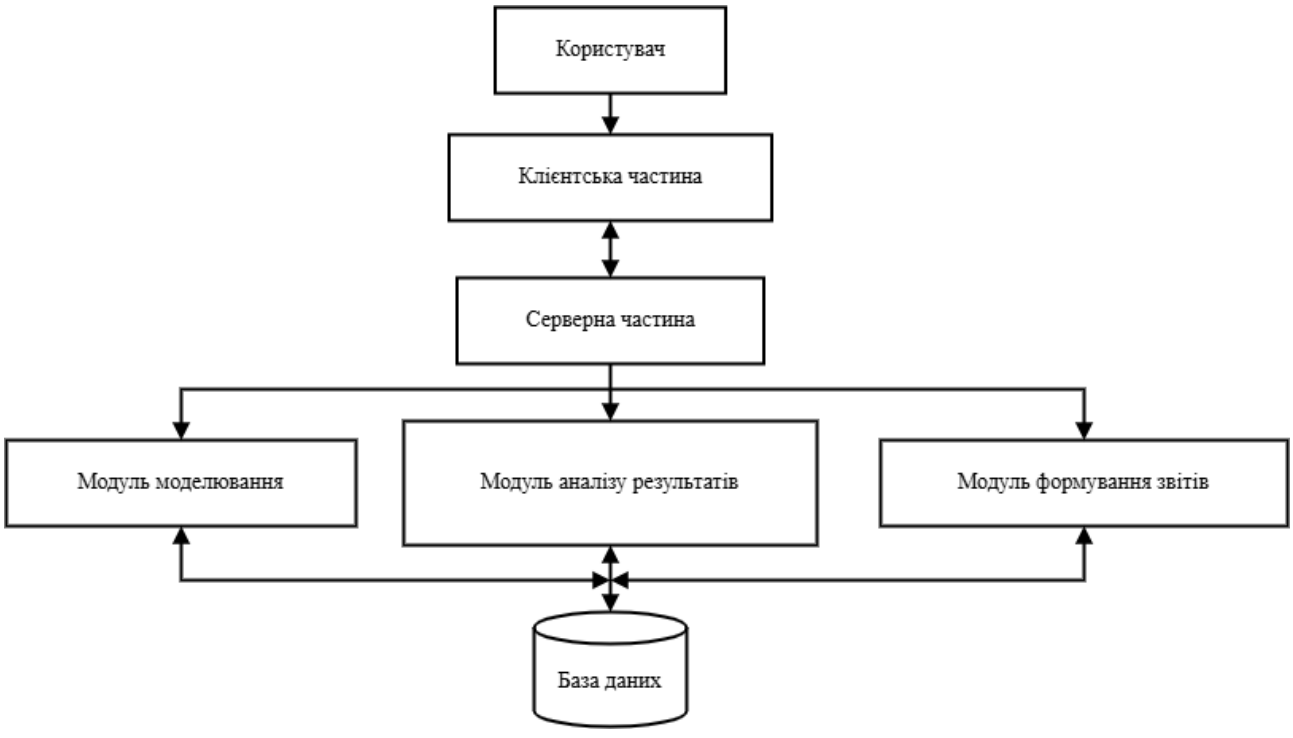


Рисунок 2.1 – Структурна схема програмно-технічного засобу

На рисунку 2.1 показано основні компоненти програмно-технічного засобу та зв'язки між ними. Через API виконуються операції з моделями, запуск моделювання, отримання результатів, ML-аналіз і формування PDF-звіту. Дані зберігаються в базі даних, що забезпечує зв’язок між моделлю, метриками та результатами аналізу.

Такий поділ спрощує супровід системи. Клієнтська частина не виконує складні розрахунки, а тільки передає параметри і відображає результати. Серверна частина відповідає за перевірку даних, запуск моделювання, збереження метрик і роботу з базою даних. Це дає змогу змінювати алгоритм моделювання або аналізу без повного переписування інтерфейсу.

2.2 Обґрунтування вибору технологій та середовища реалізації

Для реалізації програмно-технічного засобу було обрано стек технологій, який дозволяє розділити серверну логіку, клієнтський інтерфейс, збереження даних і модуль аналізу результатів. Такий поділ відповідає архітектурі системи, у якій користувач працює через браузер, а обробка моделі, запуск моделювання і ML-аналіз виконуються на серверній частині.

Серверну частину реалізовано мовою Python [36] із використанням FastAPI [20]. Python обрано через наявність бібліотек для обробки даних і машинного навчання. FastAPI використано для побудови REST API, типізованих схем запитів і відповідей, а також для зручної перевірки вхідних даних. Опис такого API може узгоджуватися зі специфікацією OpenAPI. Для опису структур даних застосовуються Pydantic-схеми [37], що спрощує перевірку параметрів моделі системи, вузлів, каналів, подій і критеріїв нестабільності. Для запуску серверної частини FastAPI може використовуватися ASGI-сервер Uvicorn [38].

Під час вибору засобів реалізації серверної частини також розглядалися Flask і Django. Flask є простим фреймворком і добре підходить для невеликих вебзастосунків, однак у цій роботі важливою була зручна побудова типізованого

API та перевірка структур запитів і відповідей. Django має більше вбудованих можливостей, але для розроблюваного засобу його функціональність була надмірною, оскільки основна логіка пов’язана не з типовим вебсайтом, а з обробкою моделі системи, запуском моделювання та поверненням результатів через API. Тому було обрано FastAPI.

Для роботи з базою даних використано SQLAlchemy [23]. Такий підхід дозволяє описувати сутності системи на рівні моделей і працювати з різними реляційними базами даних. Для локального запуску використовується SQLite [39], оскільки вона не потребує окремого сервера бази даних. Для розгортання передбачено PostgreSQL [40], який краще підходить для серверного середовища. Під час проєктування структури даних враховано необхідність зберігати модель системи, результати запусків, метрики й аналітичні висновки як пов’язані сутності, що відповідає загальним підходам до побудови data-intensive застосунків [41].

Для збереження даних окремо оцінювався варіант використання MongoDB, оскільки структура моделі системи містить вкладені об’єкти: вузли, канали, події та критерії нестабільності. Проте під час проєктування було вирішено використовувати реляційну модель, оскільки між моделлю, метриками, результатами аналізу та звітами існують чіткі зв’язки. Реляційна база даних спрощує контроль цілісності даних і дає змогу зручніше формувати вибірки для сторінки результатів та PDF-звіту.

Клієнтську частину реалізовано з використанням React [21], Vite [22] і TypeScript [42]. React обрано через зручність побудови інтерфейсів із багатьма станами, формами, таблицями й графіками. У клієнтській частині є сторінка моделей систем, покроковий конструктор топології та сторінка результатів моделювання. Vite використовується як інструмент збірки, оскільки забезпечує швидкий запуск проєкту під час розробки. TypeScript дає змогу зменшити кількість помилок при роботі з об’єктами моделі, вузлами, каналами та результатами моделювання.

На етапі вибору клієнтської технології розглядалися Vue.js і звичайна реалізація на HTML, CSS та JavaScript без окремого фреймворку. Використання чистого JavaScript могло б спростити структуру проєкту, але ускладнило б підтримку сторінок із великою кількістю станів, форм і взаємопов'язаних даних. Vue.js теж підходить для створення інтерфейсів, однак React було обрано через зручну організацію компонентів і відповідність структурі розроблюваного інтерфейсу.

Для відображення графіків використано Recharts [43]. Цієї бібліотеки достатньо для побудови графіків затримки, використання CPU/RAM, пропускної здатності, RPS і частки помилок.

Для побудови графіків порівнювалися Chart.js і D3.js. Chart.js є простим засобом для створення базових діаграм, але менш зручний для інтеграції з компонентною структурою React. D3.js надає широкі можливості для нестандартних візуалізацій, проте потребує більше часу на реалізацію. У цій роботі потрібно було відобразити зрозумілі часові графіки метрик, тому Recharts виявився достатнім варіантом.

Для ML-аналізу використано scikit-learn [33], оскільки бібліотека містить реалізації KMeans [28], Isolation Forest [30] і PCA [32], які застосовуються для групування режимів роботи та виявлення аномальних інтервалів. Альтернативами були TensorFlow і PyTorch, але ці бібліотеки більше орієнтовані на побудову та навчання складних нейронних мереж. У межах цієї роботи достатньо класичних методів аналізу табличних метрик, тому scikit-learn краще відповідає поставленій задачі.

Середовищем розробки обрано Visual Studio Code [44]. Це середовище підтримує роботу з Python, TypeScript, JavaScript, JSON, Markdown і файлами конфігурації проєкту. Для цієї роботи важливо, що в одному середовищі можна редагувати серверну частину, клієнтську частину, файли бази даних, конфігурації Render/Vercel і текст пояснювальної записки. Також Visual Studio

Code має вбудований термінал, підтримку Git і розширення для Python, TypeScript та роботи з Docker-файлами.

Для контролю версій використовується Git [45]. Репозиторій проєкту може бути розміщений на GitHub [46], що спрощує подальше розгортання клієнтської частини на Vercel [26] і серверної частини на Render [25]. Такий набір засобів достатній для реалізації, тестування і демонстрації розробленого програмно-технічного засобу. Використання зазначених технологій забезпечує можливість подальшого розвитку програмного засобу без суттєвої зміни його архітектури.

Поділ системи на клієнтську частину, серверний API, модуль моделювання, модуль аналізу та базу даних відповідає принципам розмежування відповідальності між компонентами програмної системи [34, 35]. Опис серверного API може узгоджуватися зі специфікацією OpenAPI [47]. Обраний підхід також узгоджується з принципами побудови надійних програмних систем, де важливими є спостережуваність, контроль стабільності, аналіз відмов і поступове вдосконалення системи [48]. Використання зазначених технологій забезпечує можливість подальшого розвитку програмного засобу без суттєвої зміни його архітектури.

2.3 Розроблення моделі комп’ютерної системи та структури її топології

Модель комп’ютерної системи повинна містити інформацію, достатню для повторного відтворення сценарію моделювання. До неї входять назва, опис, тривалість моделювання, крок розрахунку, профіль навантаження, параметри клієнтів, критерії нестабільності, набір вузлів, набір каналів, події сценарію та результати запуску.

Вузол описується типом, роллю, кількістю ядер CPU, умовною обчислювальною здатністю, витратами CPU на один запит, обсягом RAM, базовим використанням RAM, витратами RAM на один запит, максимальною кількістю з’єднань, часом обробки та попереджувальними і критичними

порогами. Канал зв’язку описується початковим і цільовим вузлом, пропускною здатністю, базовою затримкою, коливанням затримки, втратою пакетів, розміром черги і протоколом. Основні складові моделі комп’ютерної системи наведено у таблиці 2.1.

Таблиця 2.1 – Основні складові моделі комп’ютерної системи

Складова	Призначення	Основні параметри
Загальна інформація	ідентифікація моделі та режиму запуску	назва, опис, тривалість, крок моделювання
Вузол	опис компонента системи	тип, роль, CPU, RAM, max connections, processing time, пороги
Канал зв’язку	опис з’єднання між вузлами	початковий і кінцевий вузол, пропускна здатність, затримка, коливання затримки, втрата пакетів, протокол
Профіль навантаження	опис зміни RPS у часі	constant, linear growth, step load, custom profile
Подія сценарію	зміна параметрів під час моделювання	час, тип події, ціль, нове значення
Критерії нестабільності	правила переходу в попереджувальний або критичний стан	CPU, RAM, затримка, частка помилок, втрата пакетів, використання каналу
Результати моделювання	збережені метрики й аналітика	системні метрики, метрики вузлів, метрики каналів, ML-аналіз

Таблиця 2.1 показує, що модель поєднує статичну конфігурацію і динамічні дані, які з’являються після запуску моделювання. Завдяки цьому користувач може пов’язати отримані результати з конкретною конфігурацією системи.

Оскільки модель комп’ютерної системи містить не тільки числові параметри, а й зв’язки між компонентами, для її налаштування доцільно використовувати візуальне подання топології. У такому поданні система відображається як набір вузлів і каналів зв’язку між ними. Це дає змогу користувачу не лише вводити параметри у форму, а й бачити загальну структуру системи, напрям передавання навантаження та можливі місця виникнення вузьких місць.

Для цього у програмному засобі передбачено візуальний конструктор топології, який використовується на сторінці створення та редагування моделі. Його завдання полягає у тому, щоб поєднати налаштування параметрів із графічним поданням структури системи. Користувач може додавати вузли, з’єднувати їх каналами, змінювати положення елементів на полотні та редагувати параметри окремих компонентів.

Топологія може створюватися з порожньої конфігурації або на основі шаблону. У системі передбачено кілька варіантів початкової структури, зокрема базову систему, систему з балансувальником, систему з кешем і порожню модель. Після додавання вузлів користувач може змінювати їхнє положення на полотні, а канали зв’язку автоматично перебудовуються відповідно до нового розміщення елементів. Це дозволяє підтримувати зрозуміле візуальне подання топології та зберігати логічне розташування зв’язків між компонентами.

Такий підхід дає змогу відокремити процес редагування моделі від результатів її подальшого моделювання. Користувач працює з актуальною топологією на полотні, а під час запуску система фіксує її стан для подальшого аналізу. Це особливо важливо для порівняння різних запусків, оскільки одна й та сама модель може змінюватися користувачем після отримання результатів.

Позиції вузлів і параметри побудованої топології зберігаються у знімку топології. Завдяки цьому після запуску моделювання результати залишаються пов’язаними саме з тією конфігурацією, яка існувала на момент запуску. Якщо

користувач пізніше змінить модель, попередні результати не втрачають зв’язку зі старою структурою системи.

Сторінка конструктора поділяється на дві основні частини. Ліва панель містить параметри вибраного вузла або каналу, а також етапи налаштування моделі: загальну інформацію, вибір шаблону, навантаження, події сценарію, критерії нестабільності та перевірку готовності до запуску. Права частина містить полотно топології, на якому користувач додає вузли, з’єднує їх каналами, переміщує елементи, змінює масштаб і центрує схему. Блок-схему роботи візуального конструктора топології наведено на рисунку 2.2.

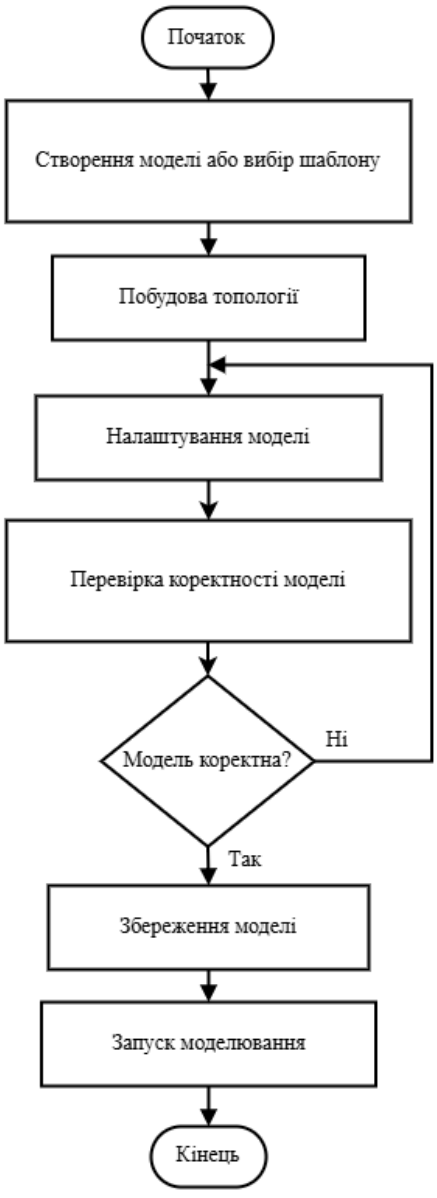


Рисунок 2.2 – Блок-схема роботи візуального конструктора топології

На рисунку 2.2 показано послідовність дій користувача під час створення та перевірки топології. Користувач створює нову модель або обирає один із підготовлених шаблонів, після чого переходить до побудови топології, налаштування параметрів вузлів і каналів зв'язку, задання навантаження, подій та критеріїв нестабільності. Перед запуском моделювання система перевіряє коректність моделі. Якщо перевірка виявляє помилки або неповні дані, користувач повертається до редагування параметрів.

Модель комп'ютерної системи у програмно-технічному засобі розглядається як сукупність взаємопов'язаних елементів, що описують її топологію, сценарій моделювання, критерії нестабільності та результати аналізу. Такий підхід дозволяє не обмежуватися лише збереженням окремих параметрів у базі даних, а подати модель як цілісну структуру, у якій вузли, канали зв'язку, навантаження та результати моделювання пов'язані між собою.

До складу топології входять вузли, канали зв'язку та знімок топології. Вузли описують окремі компоненти комп'ютерної системи, наприклад клієнтську групу, маршрутизатор, сервер застосунку, кеш або базу даних. Канали зв'язку задають взаємодію між вузлами та характеризуються пропускною здатністю, затримкою, втратою пакетів, коливанням затримки, розміром черги та протоколом взаємодії. Знімок топології фіксує стан візуальної моделі на момент запуску моделювання.

Сценарій моделювання визначає умови, за яких перевіряється поведінка комп'ютерної системи. До нього входять профіль навантаження, події сценарію, тривалість моделювання та крок розрахунку. Профіль навантаження задає характер зміни кількості запитів у часі, а події сценарію дозволяють змінювати параметри системи під час моделювання, наприклад імітувати відмову вузла, збільшення затримки каналу або зростання втрати пакетів.

Після завершення моделювання формуються результати, до складу яких входять системні метрики, метрики вузлів, метрики каналів, результати ML-аналізу, рекомендації та PDF-звіт. Системні метрики характеризують загальний

стан моделі, метрики вузлів показують використання ресурсів окремих компонентів, а метрики каналів відображають стан зв'язків між ними. ML-аналіз використовується для виявлення характерних режимів роботи та аномальних часових інтервалів. Структурну схему моделі комп'ютерної системи та її топології наведено на рисунку 2.3.

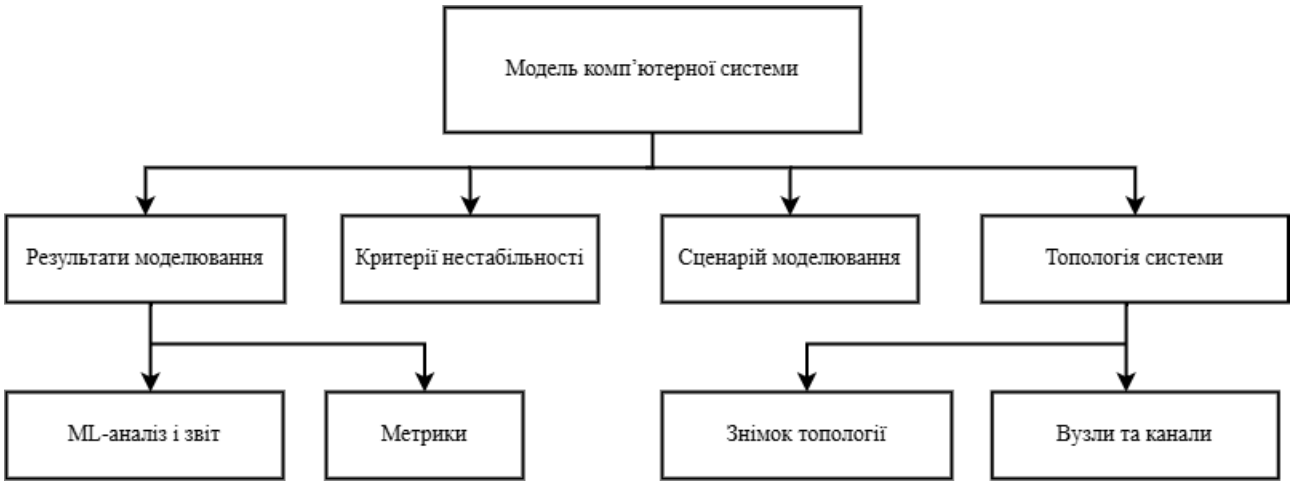


Рисунок 2.3 – Структурна схема моделі комп'ютерної системи та її топології

На рисунку 2.3 показано, що модель комп'ютерної системи складається з топології системи, сценарію моделювання, критеріїв нестабільності та результатів моделювання. Топологія системи включає вузли, канали зв'язку та знімок топології. Сценарій моделювання охоплює профіль навантаження, події сценарію, тривалість і крок моделювання. Результати моделювання містять системні метрики, метрики вузлів, метрики каналів, ML-аналіз, рекомендації та PDF-звіт.

2.4 Розроблення підсистеми моделювання мережевих сценаріїв

Сценарій навантаження визначає, як змінюється кількість запитів за секунду протягом моделювання. У системі передбачено чотири профілі: постійне навантаження, лінійне зростання, ступінчасте навантаження та користувацький профіль. Постійне навантаження використовує одне значення

RPS. Лінійне зростання переходить від Start RPS до Max RPS протягом заданої тривалості. Ступінчасте навантаження задається через інтервали навантаження. Користувацький профіль задається точками часу та відповідними значеннями RPS. Параметри сценарію навантаження наведено у таблиці 2.2.

Таблиця 2.2 – Параметри сценарію навантаження

Параметр	Опис	Вплив на моделювання
Start RPS	початкове значення запитів за секунду	використовується у лінійному та користувацькому профілях
Max RPS	максимальне або цільове значення RPS	визначає верхню межу навантаження
Clients	кількість клієнтів у сценарії	впливає на кількість активних з’єднань
Request profile	тип запиту за умовним розміром і складністю	впливає на трафік і навантаження на вузли
Час очікування відповіді	граничний час очікування відповіді	впливає на кількість перевищень часу очікування і частку помилок
Bandwidth limit	глобальне обмеження пропускної здатності	використовується для початкових параметрів каналів
Базова затримка	початкова затримка каналу	додається до сумарного часу відповіді
Втрата пакетів	відсоток втрати пакетів	впливає на помилки, повторні запити та затримку
Повторні запити	ознака використання повторних запитів	збільшує навантаження при помилках

Параметри з таблиці 2.2 використовуються під час розрахунку RPS, трафіку, затримки, помилок і кількості активних з’єднань.

Події сценарію задають зміни, які відбуваються під час моделювання. Кожна подія має час, тип, цільовий вузол або канал і нове значення. В інтерфейсі події відображаються як “Подія 1”, “Подія 2”, “Подія 3”. Для подій каналів ціллю є конкретний канал зв’язку, для подій вузлів - конкретний вузол, для події збільшення трафіку - вся система. Типи подій сценарію наведено в таблиці 2.3.

Таблиця 2.3 – Типи подій сценарію

Тип події	Ціль	Результат впливу
Відмова вузла	вузол	вузол переходить у стан failed, його навантаження не обробляється
Відновлення вузла	вузол	вузол повертається до участі в обробці запитів
Падіння пропускної здатності	канал	пропускна здатність каналу зменшується до нового значення
Стрибок затримки	канал	до затримки каналу додається нове значення
Стрибок втрат пакетів	канал	втрата пакетів у каналі збільшується
Зменшення CPU capacity	вузол	вузол обробляє менше запитів за тих самих умов
Різкий приріст трафіку	система	поточний RPS множиться на заданий коефіцієнт

Таблиця 2.3 показує, що події дозволяють перевірити поведінку системи не тільки при плавній зміні навантаження, а й при аварійних змінах конфігурації.

Після визначення профілю навантаження та подій сценарію виконується моделювання поведінки вузлів і каналів зв’язку. Алгоритм моделювання працює за часовими інтервалами. У кожному інтервалі система визначає поточне навантаження, розподіляє запити між вузлами і каналами, розраховує стан компонентів та формує системні метрики. Після завершення розрахунку

отримані результати зберігаються в базі даних і передаються на подальший аналіз.

Спочатку система зчитує конфігурацію моделі, до якої входять параметри вузлів, каналів зв’язку, профіль навантаження, події сценарію та критерії нестабільності. Після цього формуються часові інтервали моделювання відповідно до заданої тривалості та кроку розрахунку. Для кожного інтервалу визначається поточне значення RPS, враховуються події сценарію та розраховується навантаження, яке проходить через топологію.

Далі виконується розподіл запитів між вузлами і каналами зв’язку. Для вузлів розраховується використання CPU і RAM, кількість активних з’єднань, кількість оброблених і помилкових запитів. Для каналів визначається використання пропускної здатності, затримка, втрата пакетів і кількість відкинутих пакетів. На основі цих значень система перевіряє попереджувальні та критичні пороги, після чого визначає стан кожного вузла і каналу.

На завершальному етапі формуються системні метрики, які описують загальний стан моделі під час конкретного інтервалу моделювання. До них належать RPS, пропускна здатність, середня затримка, процентилі затримки, частка помилок, кількість успішних запитів, кількість помилок, перевищення часу очікування та повторні запити. Після обробки всіх інтервалів результати зберігаються в базі даних, а сформований набір метрик передається на ML-аналіз.

Запропонований алгоритм також забезпечує зв’язок між параметрами топології та результатами аналізу. Наприклад, зміна пропускної здатності каналу або відмова одного з вузлів одразу враховується під час розрахунку наступних часових інтервалів. Такий підхід робить процес моделювання більш наочним і дозволяє використовувати отримані дані для подальшого пояснення причин нестабільної роботи системи. Блок-схему алгоритму моделювання поведінки комп'ютерної системи наведено на рисунку 2.4.

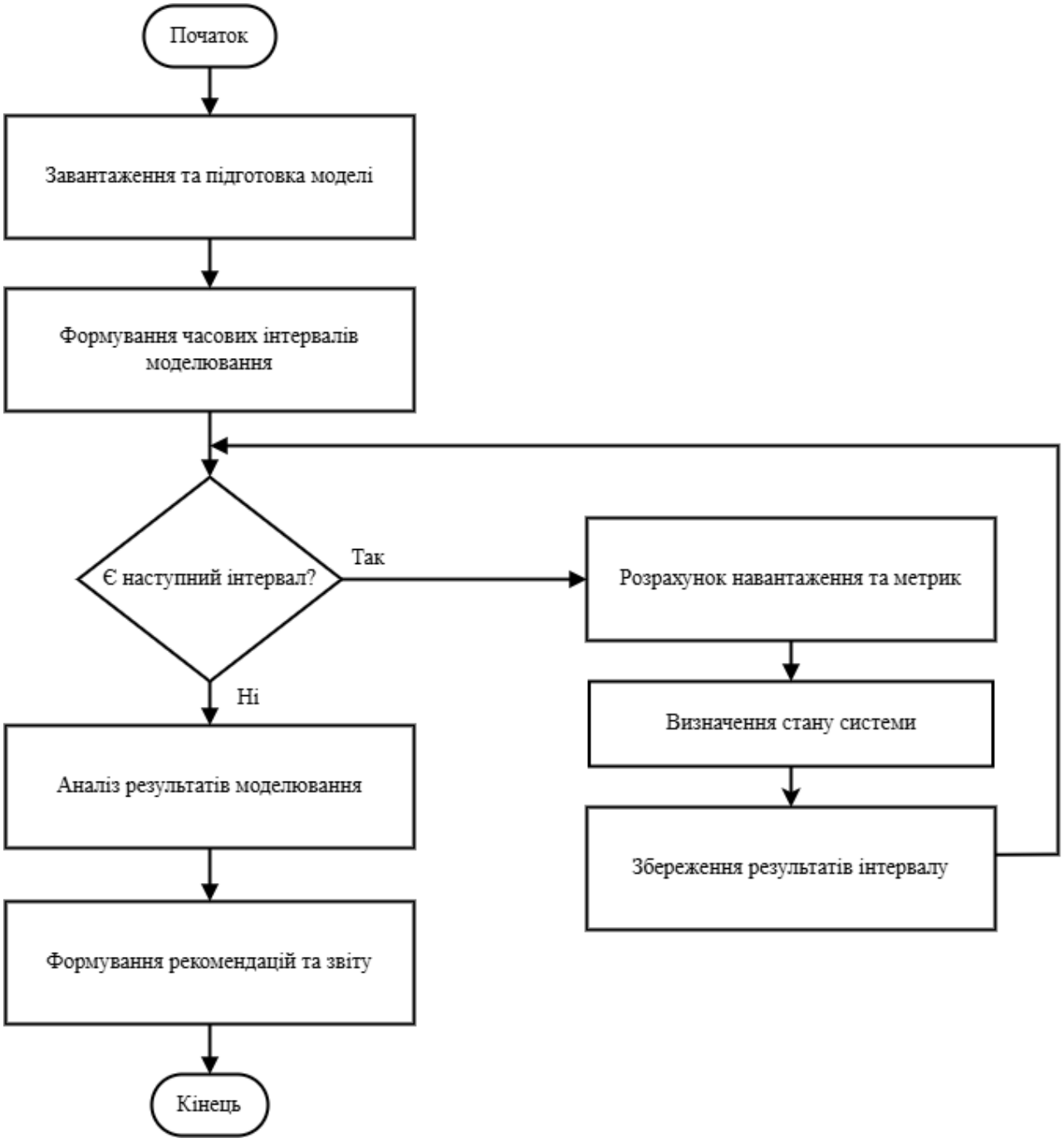


Рисунок 2.4 – Блок-схема алгоритму моделювання поведінки системи

На рисунку 2.4 показано основні етапи моделювання поведінки комп'ютерної системи та аналізу отриманих результатів. Оскільки моделювання виконується за інтервалами, система може визначати момент, коли вперше було перевищено попереджувальний або критичний поріг.

Такий порядок виконання моделювання дозволяє послідовно відстежувати зміну стану системи від початкового стабільного режиму до можливого

перевантаження або деградації. Важливо, що розрахунок виконується не лише для підсумкового результату, а для кожного окремого часового інтервалу. Завдяки цьому можна визначити не тільки факт переходу системи в нестабільний стан, а й момент, коли почали змінюватися основні метрики.

Для вузлів розраховуються використання CPU, використання RAM, кількість активних з'єднань, кількість оброблених запитів, кількість помилкових запитів, середній час обробки і стан. Для каналів розраховуються трафік, використання каналу, затримка, втрата пакетів, кількість відкинутих пакетів і стан. Для системи загалом формуються RPS, пропускна здатність, середня затримка, процентилі затримки, частка помилок, кількість успішних запитів, кількість помилок, кількість перевищень часу очікування, кількість повторних запитів, CPU, RAM, втрата пакетів і коливання затримки.

Окреме значення має збереження проміжних результатів моделювання. Якщо для кожного інтервалу зберігаються значення вузлових, канальних і системних метрик, користувач може порівнювати поведінку системи на різних етапах сценарію. Це дає змогу встановити, який саме компонент першим досяг попереджувального або критичного порогу, а також оцінити вплив подій сценарію на загальний стан моделі.

2.5 Розроблення підсистеми аналізу результатів моделювання та виявлення нестабільних режимів

Підсистема аналізу результатів моделювання призначена для перетворення набору розрахованих метрик у зрозумілий підсумок стану комп'ютерної системи. Після завершення моделювання користувач отримує не тільки таблиці та графіки, а й узагальнену оцінку: стан системи, рівень ризику, критичний вузол, проблемний канал, час деградації, основну причину нестабільності та рекомендації. Це потрібно тому, що за великої кількості метрик користувачу складно вручну визначити, який саме елемент системи вплинув на результат.

Вхідними даними для підсистеми аналізу є системні метрики, метрики вузлів і метрики каналів зв’язку. До системних метрик належать RPS, пропускна здатність, середня затримка, процентилі затримки, частка помилок, кількість успішних і помилкових запитів, кількість перевищень часу очікування та повторних запитів. Метрики вузлів відображають використання CPU і RAM, кількість активних з’єднань, кількість оброблених запитів, помилки та стан вузла. Метрики каналів описують використання пропускної здатності, затримку, втрату пакетів, кількість відкинутих пакетів і стан каналу.

Для первинної оцінки стану використовуються попереджувальні та критичні пороги. Якщо значення метрик залишаються в межах допустимих параметрів, система вважається стабільною. Якщо окремі метрики перевищують попереджувальні пороги, фіксуються ознаки деградації. Якщо перевищено критичні пороги або вузол чи канал переходить у критичний стан, система вважається нестабільною. Такий підхід дає змогу пов’язати підсумковий стан із конкретними параметрами моделі.

Оцінювання ризику виконується на основі сукупності ознак. Враховується не тільки одне найбільше значення, а й те, як поводяться різні групи метрик одночасно. Наприклад, висока затримка разом із втратою пакетів більше вказує на проблему каналу зв’язку, а зростання затримки разом із високим CPU або RAM може свідчити про перевантаження вузла. Якщо одночасно погіршуються декілька показників, рівень ризику підвищується.

Критичний вузол визначається за показниками використання ресурсів, кількістю активних з’єднань, помилками та станом вузла. Якщо один із вузлів має найвищі значення CPU, RAM або кількості з’єднань і при цьому впливає на загальну частку помилок, він позначається як потенційно критичний. Проблемний канал визначається за використанням пропускної здатності, затримкою, втратою пакетів і кількістю відкинутих пакетів. Такий поділ дозволяє окремо аналізувати ресурсні проблеми вузлів і мережеві проблеми каналів.

Окремо визначається час деградації. Для цього аналізуються часові інтервали моделювання і знаходиться перший момент, коли метрики переходять у попереджувальний або критичний стан. Це важливо для сценаріїв зі ступінчастим або лінійним навантаженням, оскільки користувач може побачити, при якому навантаженні система почала працювати нестабільно.

Порогові правила доповнюються методами машинного навчання. KMeans використовується для групування часових інтервалів за схожістю метрик. Це дозволяє виділити окремі режими роботи, наприклад стабільний режим, режим підвищеного навантаження або критичний режим. Isolation Forest застосовується для пошуку аномальних інтервалів, які суттєво відрізняються від інших точок моделювання. PCA використовується як допоміжний метод для зменшення розмірності та аналізу ознак.

Результати ML-аналізу не замінюють технічні пороги. Вони використовуються як додатковий інструмент для пошуку закономірностей у метриках. Основний висновок для користувача все одно формується на основі стану вузлів, каналів, системних метрик і перевищення заданих порогів. Це робить результат зрозумілішим, оскільки користувач бачить не тільки результат роботи алгоритму, а й технічне пояснення причини нестабільності. Вихідні дані аналізу результатів моделювання наведено у таблиці 2.4.

Таблиця 2.4 – Вихідні дані аналізу результатів моделювання

Дані	Призначення
Стан системи	коротко показує стабільність або нестабільність системи
Рівень ризику	визначає ступінь небезпеки сценарію: низький, середній, високий або критичний
Критичний вузол	показує вузол із найбільшим ризиком за CPU, RAM, з’єднаннями та помилками

Кінець таблиці 2.4

Критичний вузол	показує вузол із найбільшим ризиком за CPU, RAM, з’єднаннями та помилками
Проблемний канал	показує канал із найбільшим використанням, затримкою або втратою пакетів
Час деградації	фіксує перший момент переходу до попереджувального або критичного стану
Основна причина	пояснює, які метрики найбільше вплинули на результат
Рекомендації	пропонує змінити ресурси вузлів, параметри каналів або сценарій
Режими роботи	показує групи часових інтервалів за результатом KMeans
Аномальні інтервали	показує нетипові точки за результатом Isolation Forest

Таблиця 2.4 відображає набір даних, який використовується для формування сторінки результатів і PDF-звіту. Завдяки такому набору користувач отримує не тільки числові значення метрик, а й пояснення стану системи. Це спрощує порівняння різних конфігурацій, оскільки після кожного запуску можна оцінити, чи зменшився рівень ризику, чи змінився критичний вузол і чи покращилися параметри проблемного каналу.

2.6 Проєктування користувацького інтерфейсу та системи візуалізації результатів

Користувацький інтерфейс програмно-технічного засобу спроєктовано з урахуванням основного циклу роботи користувача. Спочатку користувач створює модель комп’ютерної системи, налаштовує її топологію, задає параметри вузлів і каналів зв’язку, вибирає профіль навантаження, додає події

сценарію та критерії нестабільності. Після цього виконується запуск моделювання, перегляд результатів, аналіз метрик і формування PDF-звіту.

У системі передбачено три основні сторінки. Сторінка моделей використовується для перегляду створених моделей, пошуку, фільтрації за рівнем ризику та переходу до редагування або перегляду результатів. Сторінка створення та редагування моделі призначена для налаштування параметрів системи і побудови топології. Сторінка результатів використовується для перегляду підсумку моделювання, графіків, таблиць, рекомендацій і результатів ML-аналізу.

Сторінка моделей має просту технічну структуру. Основним елементом є таблиця, у якій відображаються номер, назва моделі, рівень ризику, дата створення та доступні дії. Користувач може створити нову модель, знайти потрібну модель за назвою, перейти до результатів, відредагувати конфігурацію або видалити запис.

Сторінка створення моделі має покрокову структуру. Користувач послідовно заповнює загальну інформацію, вибирає шаблон, налаштовує вузли, канали, профіль навантаження, події сценарію, критерії нестабільності та перевіряє готовність моделі. Параметри моделі пов’язані з полотном топології, тому зміни вузлів і каналів одразу відображаються у структурі системи.

Сторінка результатів подає інформацію від загального до детального. У верхній частині відображається короткий підсумок моделювання: стан системи, рівень ризику, критичний вузол, проблемний канал, час деградації та основна причина нестабільності. Нижче розміщується знімок топології, основні метрики, графіки, хронологія, таблиці вузлів і каналів, результати ML-аналізу та рекомендації.

Візуалізація результатів будується навколо часових графіків. Вони дають змогу побачити, як змінювалися RPS, пропускна здатність, затримка, використання CPU і RAM, частка помилок, втрата пакетів та інші показники.

Якщо на графіку видно різке зростання затримки або частки помилок, користувач може зіставити це з подіями сценарію і станом вузлів або каналів.

Таблиці використовуються для детального перегляду даних: метрик вузлів, метрик каналів, аномальних інтервалів, системних показників і хронології зміни стану. Тому в інтерфейсі поєднуються обидва способи подання: графіки для загального розуміння динаміки і таблиці для точного аналізу.

Рекомендації пов’язані з результатами аналізу, а не подаються як загальні поради. Якщо система визначила перевантаження вузла, рекомендація стосується збільшення ресурсів, зменшення навантаження або зміни розподілу запитів. Якщо проблемним є канал зв’язку, рекомендація стосується пропускної здатності, затримки, втрати пакетів або структури топології.

Формування PDF-звіту є частиною системи візуалізації результатів. До нього входять загальна інформація про модель, підсумок стану системи, основні метрики, інформація про критичний вузол і проблемний канал, результати ML-аналізу, рекомендації та таблиці з даними. Такий звіт можна використовувати для порівняння запусків або як додатковий матеріал до пояснювальної записки.

Під час проєктування інтерфейсу важливо було не перевантажити користувача зайвими елементами. Основні дії повинні бути доступні без складної навігації: створення моделі, редагування топології, запуск моделювання, перегляд результатів і формування звіту. Інтерфейс має залишатися технічним і зрозумілим, оскільки головна увага приділяється коректному налаштуванню моделі та аналізу результатів.

2.7 Висновки до другого розділу

У другому розділі виконано проєктування програмно-технічного засобу моделювання та аналізу поведінки комп’ютерних систем у мережевому середовищі. Розроблено архітектуру вебзастосунку, у якій виділено клієнтський

рівень, серверний рівень, базу даних, модуль керування моделями, модуль моделювання, ML-модуль і модуль формування PDF-звіту.

Обґрунтовано вибір технологій реалізації. Для серверної частини обрано Python і FastAPI, для клієнтської частини React, Vite і TypeScript, для роботи з базою даних SQLAlchemy, SQLite і PostgreSQL. Для побудови графіків передбачено Recharts, для ML-аналізу scikit-learn, а як середовище розробки обрано Visual Studio Code.

У розділі визначено структуру моделі комп'ютерної системи. Модель містить загальну інформацію, вузли, канали зв'язку, профіль навантаження, події сценарію, критерії нестабільності та результати моделювання. Окремо спроєктовано візуальне подання топології та знімок топології, який фіксує конфігурацію системи на момент запуску.

Також розроблено структуру бази даних для збереження моделей, вузлів, каналів, подій, системних, вузлових і канальних метрик, результатів аналізу, аналітичних точок, PDF-звітів і шаблонів сценаріїв. Така структура дає змогу пов'язати параметри моделі з результатами моделювання і подальшим аналізом.

Окремо спроєктовано підсистему моделювання мережевих сценаріїв і підсистему аналізу результатів. Визначено профілі навантаження, події сценарію, параметри вузлів і каналів, а також алгоритм розрахунку метрик за часовими інтервалами. Порогові правила використовуються для пояснення попереджувальних і критичних станів, а методи KMeans, Isolation Forest і PCA доповнюють аналіз режимів роботи та аномальних інтервалів.

Користувацький інтерфейс спроєктовано відповідно до основного циклу роботи: створення моделі, налаштування топології, запуск моделювання, перегляд результатів і формування PDF-звіту. Передбачено сторінку моделей, сторінку створення та редагування моделі, сторінку результатів, часові графіки, таблиці метрик, рекомендації та експорт звіту. Отримані проєктні рішення використовуються у третьому розділі під час опису програмної реалізації.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА АНАЛІЗ РОБОТИ ПРОГРАМНО-ТЕХНІЧНОГО ЗАСОБУ

3.1 Реалізація серверної та клієнтської частин системи

Серверну частину реалізовано на основі FastAPI. Основний файл застосунку backend/app/main.py створює FastAPI-застосунок, налаштовує CORS, виконує ініціалізацію бази даних і підключає маршрутизатори для роботи з моделями систем, аналізом та звітами. API має префікс /api, що дозволяє відокремити серверні маршрути від клієнтської частини вебзастосунку.

Серверна частина містить маршрути для створення, редагування, запуску та видалення моделей системи. Через API також виконується отримання топології, системних метрик, метрик вузлів і каналів, результатів ML-аналізу та PDF-звіту. Структуру взаємодії клієнтської частини, серверного API, модулів моделювання, аналізу та бази даних наведено у додатку А.

Робота з базою даних виконується через SQLAlchemy. У конфігурації передбачено використання SQLite для локального запуску та підтримку PostgreSQL через змінну середовища DATABASE_URL. Такий підхід дозволяє запускати систему локально без додаткової інфраструктури, але залишає можливість розгортання у серверному середовищі. Для розгортання серверної частини використовується файл render.yaml, у якому задано команду встановлення залежностей, запуск міграцій Alembic і старт сервера Uvicorn.

Основні модулі серверної частини розміщені в каталозі backend/app. Файл models.py містить ORM-моделі бази даних, файл schemas.py містить схеми запитів і відповідей, а маршрути роботи з окремими частинами системи винесені в каталог routers. Зокрема, routers/experiments.py відповідає за роботу з моделями систем, routers/analysis.py використовується для отримання результатів ML-аналізу, а routers/reports.py забезпечує формування і завантаження PDF-звіту. Розрахункова логіка винесена у services/simulation_engine.py, ML-аналіз

реалізовано у services/ml_analyzer.py, а формування PDF-звіту виконується через services/report_generator.py.

Логіка запуску моделювання винесена у сервіс experiment_runner.py. Під час запуску сервіс очищає попередні результати моделі, встановлює статус running, викликає SimulationEngine, після завершення розрахунків запускає ML-аналіз і переводить модель у статус completed. Такий поділ дозволяє відокремити маршрут API від процесу виконання моделювання та подальшого аналізу результатів. Місце сервісу запуску моделювання у загальній архітектурі програмно-технічного засобу показано у додатку А.

Клієнтську частину реалізовано на основі React, Vite і TypeScript. Маршрути клієнтського застосунку визначені у файлі frontend/src/App.tsx. У вебзастосунку передбачено сторінку моделей систем, сторінку створення та редагування моделі, а також сторінку результатів моделювання. У коді ці сторінки реалізовані відповідно як ExperimentsPage, CreateExperimentPage і ExperimentDetailsPage.

Для взаємодії з API використовується модуль frontend/src/api. У ньому реалізовано функції отримання моделей, створення, редагування, запуску моделювання, отримання метрик, топології, результатів аналізу та формування PDF-звіту. Запити виконуються через fetch, а базова адреса API задається через VITE_API_BASE_URL або зберігається в локальних налаштуваннях. Це дає змогу використовувати один клієнтський інтерфейс як для локального запуску, так і для розгорнутої серверної частини [49].

Для відображення графіків використовується Recharts. На сторінці результатів показуються графіки затримки, використання CPU і RAM, RPS, пропускної здатності та частки помилок. Таблиці вузлів, каналів, аномальних інтервалів і мережевих метрик реалізовані у вигляді звичайних таблиць з обмеженням показу перших записів і можливістю розгорнути повний список. Такий підхід дозволяє не перевантажувати сторінку результатів, але залишає доступ до детальних даних моделювання.

Для розгортання клієнтської частини передбачено файл vercel.json, який перенаправляє всі маршрути на index.html. Це потрібно для коректної роботи клієнтської маршрутизації у браузері, оскільки переходи між сторінками виконуються на рівні React-застосунку. Роботу користувача з візуальним конструктором топології показано у додатку Б.

Під час реалізації клієнтської частини основна складність була пов’язана не з окремими формами, а з узгодженням стану між лівою панеллю параметрів і полотном топології. Після перевірки було змінено частину підписів інтерфейсу, уточнено назви дій на сторінці моделей систем і залишено формування PDF саме на сторінці результатів, щоб звіт завжди був пов’язаний з конкретним запуском моделювання.

Початковою сторінкою вебзастосунку є сторінка моделей систем. Вона містить кнопку створення нової моделі, поле пошуку за назвою, фільтр за рівнем ризику і таблицю створених моделей. У таблиці показано номер, назву моделі, рівень ризику, дату створення та доступні дії. З цієї сторінки користувач може перейти до створення нової моделі, редагування існуючої конфігурації, перегляду результатів моделювання або видалення моделі.

Сторінка моделей систем виконує роль початкової точки роботи користувача з програмно-технічним засобом. Через неї користувач отримує доступ до всіх створених конфігурацій, може швидко оцінити їхній поточний стан і вибрати подальшу дію. Наявність пошуку та фільтрації спрощує роботу з кількома моделями, особливо якщо вони відрізняються параметрами навантаження, топологією або отриманим рівнем ризику.

Окреме значення має те, що сторінка не лише відображає список моделей, а й пов’язує між собою основні сценарії роботи із системою. З неї користувач переходить до створення нової конфігурації, редагування наявної топології, запуску моделювання та перегляду результатів. Така організація інтерфейсу робить роботу з вебзастосунком послідовною та зрозумілою, оскільки всі основні дії зосереджені в одному місці. Інтерфейс головної сторінки

вебзастосунку, на якій відображається список створених моделей комп’ютерних систем, наведено на рисунку 3.1.

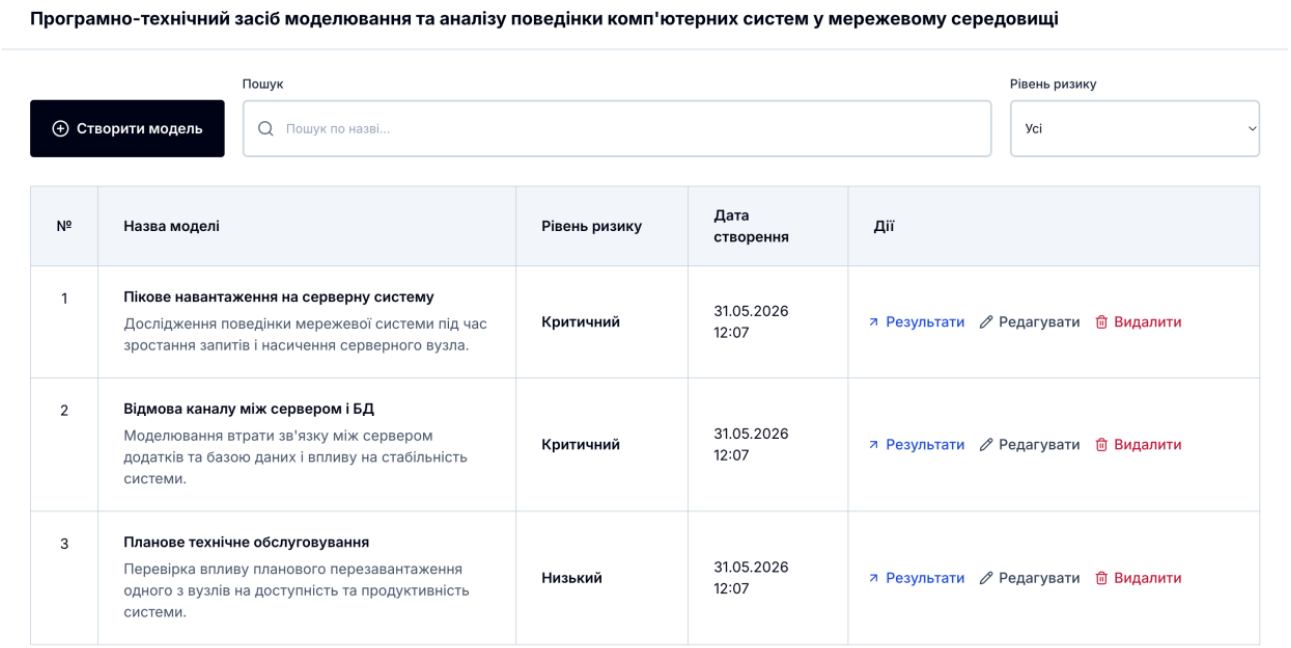


Рисунок 3.1 – Інтерфейс головної сторінки зі списком створених моделей

На рисунку 3.1 показано сторінку керування моделями систем. На ній користувач може переглядати створені моделі, виконувати пошук за назвою, фільтрувати записи за рівнем ризику, а також переходити до редагування конфігурації, перегляду результатів моделювання або видалення вибраної моделі.

Розміщення основних дій безпосередньо в рядках таблиці дає змогу швидко працювати з кожною моделлю окремо. Користувач одразу бачить, до якої конфігурації належить вибрана дія, тому може без зайвих переходів відкрити редагування, переглянути результати або видалити непотрібний запис. Такий підхід спрощує навігацію та робить інтерфейс головної сторінки більш зрозумілим.

3.2 Реалізація конструктора топології та налаштування параметрів моделі

Сторінка створення або редагування моделі реалізована як покроковий конструктор. У межах цього конструктора користувач спочатку заповнює загальну інформацію про модель і за потреби обирає шаблон, після чого налаштовує вузли, канали, профіль навантаження, події сценарію та критерії нестабільності. Завершальним етапом є перевірка готовності моделі перед збереженням або запуском моделювання. Початковий крок сторінки створення моделі системи наведено на рисунку 3.2.

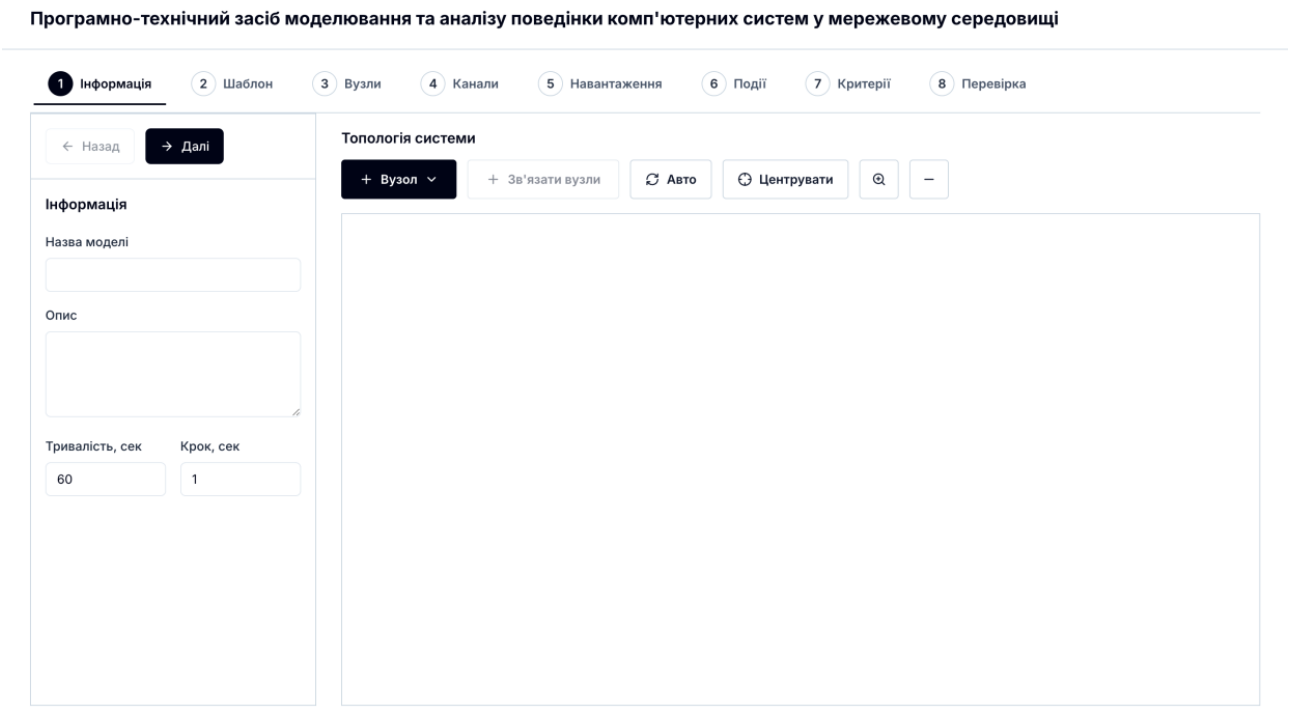


Рисунок. 3.2 – Початковий крок сторінки створення моделі системи

На рисунку 3.2 показано початковий стан сторінки створення моделі системи, з якого користувач розпочинає налаштування нової конфігурації. Оскільки сторінка побудована за принципом покрокового конструктора, її візуальне наповнення змінюється залежно від вибраного етапу. На початковому кроці відображаються елементи введення базової інформації про модель і навігація між наступними етапами налаштування.

Користувач може почати з шаблону або створити порожню модель. Доступні шаблони базової системи, системи з балансувальником і системи з кешем. На полотні можна додати вузол потрібного типу, з’єднати вузли каналом, вибрати вузол або канал для редагування, перемістити вузол, змінити масштаб, виконати автоматичне компонування або центрувати схему. Налаштування параметрів вузла у конструкторі топології наведено на рисунку 3.3.

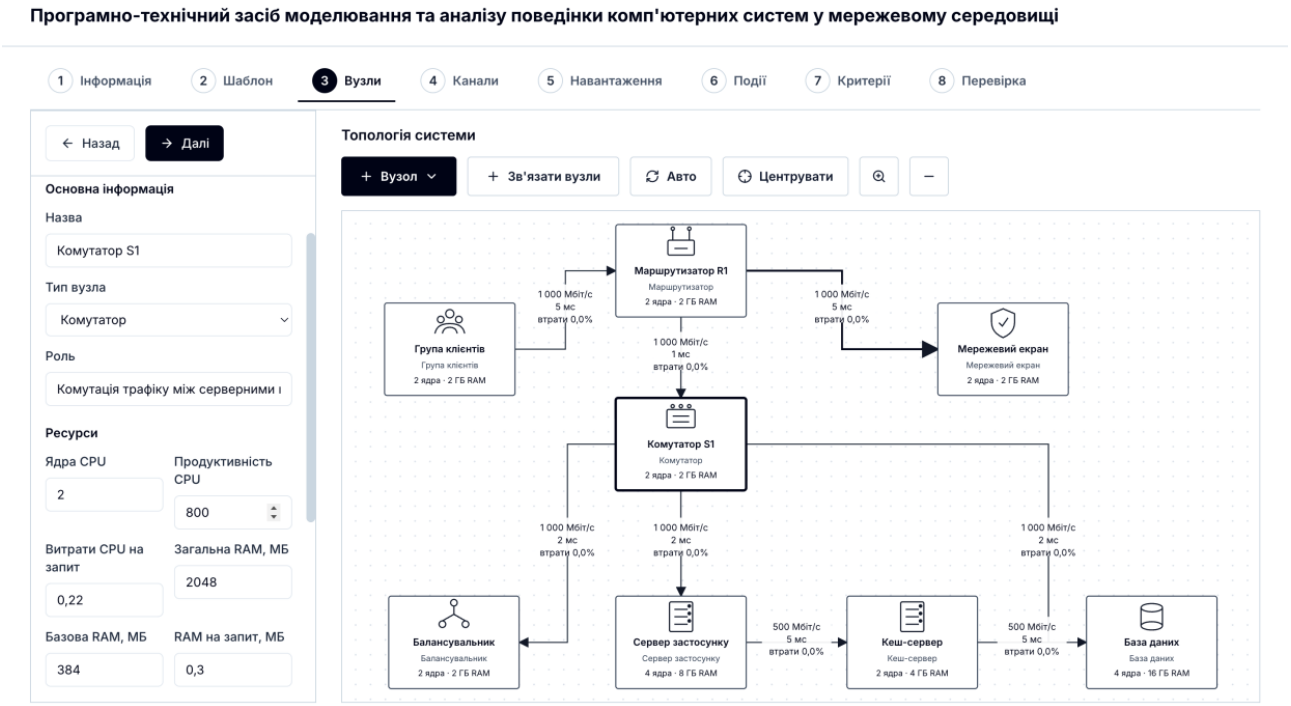


Рисунок. 3.3 – Налаштування параметрів вузла у конструкторі топології

На рисунку 3.3 показано, що для вузла редагуються назва, тип, роль, CPU, RAM, max connections, processing time і пороги нестабільності. Такі параметри надалі використовуються під час розрахунку CPU, RAM, активних з’єднань і стану вузла.

Для каналів редагуються початковий вузол, кінцевий вузол, пропускна здатність, базова затримка, коливання затримки, втрата пакетів, розмір черги і протокол. Після зміни положення вузлів маршрути каналів перераховуються, а дані про лінії та підписи каналів потрапляють до знімка топології.

Крок налаштування навантаження визначає, як саме система буде отримувати запити під час моделювання. Користувач задає кількість клієнтів, тип запиту, час очікування відповіді, використання повторних запитів і мережеві параметри за замовчуванням. Після цього обирається профіль навантаження. Для постійного профілю задається одне значення RPS, яке використовується протягом усього запуску. Для лінійного профілю вказуються початкове та цільове значення RPS, а система поступово збільшує навантаження від Start RPS до Max RPS протягом заданої тривалості. Для ступінчастого профілю користувач створює інтервали з початком, кінцем і значенням RPS. Це дозволяє змоделювати різкий перехід від нормального режиму до пікового навантаження. Для користувацького профілю задаються точки часу та відповідні значення RPS, після чого моделювання використовує ці точки як сценарій зміни інтенсивності запитів.

На цьому самому етапі задаються параметри, які впливають на обчислення мережевих показників: базова затримка, коливання затримки, втрата пакетів і глобальне обмеження пропускної здатності. Ці значення використовуються як початкові або типові параметри для каналів, якщо користувач не задав їх окремо. Профіль запиту впливає на умовний розмір і складність обробки запиту, а кількість клієнтів впливає на активні з’єднання та навантаження на вузли. Завдяки цьому користувач може перевірити не тільки стабільний режим, а й сценарії з поступовим ростом, піковим навантаженням або нерівномірним профілем трафіку.

Крок налаштування подій сценарію використовується для опису змін, які виникають під час моделювання. Користувач може додати одну або кілька подій, для кожної з яких задається час спрацювання, тип події, ціль і нове значення параметра. Подія може стосуватися конкретного вузла, конкретного каналу зв’язку або всієї системи. Для вузлів підтримуються відмова, відновлення і зменшення продуктивності CPU. Для каналів можна задати падіння пропускної здатності, стрибок затримки або збільшення втрати пакетів. Для всієї системи

доступне різке збільшення трафіку, яке множить поточний RPS на заданий коефіцієнт.

Під час створення події інтерфейс обмежує вибір цілі відповідно до типу події. Якщо подія стосується вузла, користувач обирає один із вузлів топології. Якщо подія стосується каналу, користувач обирає конкретний зв’язок між двома вузлами. Якщо подія впливає на всю систему, окрема ціль не потрібна. Після запуску моделювання події застосовуються у відповідних часових інтервалах і змінюють параметри моделі. Це дає змогу перевірити поведінку системи не тільки при зростанні навантаження, а й при аварійних ситуаціях: відмові сервера, погіршенні каналу або різкому збільшенні кількості запитів.

3.3 Реалізація сценаріїв моделювання, збору метрик та збереження результатів

Реалізація сценаріїв моделювання пов’язана з перетворенням параметрів моделі системи у послідовність розрахованих часових точок. Після того як користувач створив модель, налаштував топологію, вузли, канали зв’язку, профіль навантаження, події та критерії нестабільності, система повинна виконати запуск моделювання і сформувати набір результатів, які надалі використовуються для аналізу та відображення на сторінці результатів.

Це забезпечує зв’язок між заданою користувачем конфігурацією системи та фактичними результатами, які отримуються після виконання сценарію моделювання. Запуск моделювання виконується через серверну частину. Клієнтська частина передає запит на запуск для вибраної моделі системи, після чого сервер перевіряє наявність необхідної конфігурації, очищає попередні результати цього запуску і переводить модель у стан виконання. Такий підхід потрібний для того, щоб повторний запуск не змішував нові метрики зі старими даними. Якщо користувач змінив параметри вузлів, каналів або навантаження, нове моделювання повинно формувати окремий актуальний набір метрик.

Під час запуску система зчитує основні параметри моделі: тривалість моделювання, крок розрахунку, тип профілю навантаження, кількість клієнтів, початкове та максимальне значення RPS, параметри каналів, пороги нестабільності і список подій сценарію. На основі цих даних формується послідовність часових інтервалів. Для кожного інтервалу визначається поточне навантаження, яке залежить від обраного профілю. При постійному навантаженні значення RPS залишається незмінним, при лінійному зростанні воно поступово збільшується, при ступінчастому навантаженні змінюється відповідно до заданих інтервалів, а при користувацькому профілі визначається за заданими точками.

Окремо під час розрахунку враховуються події сценарію. Якщо на певному часовому інтервалі задано відмову вузла, зміну пропускної здатності каналу, стрибок затримки, збільшення втрати пакетів або різкий приріст трафіку, ці зміни застосовуються до відповідних компонентів моделі. Завдяки цьому система може відтворювати не тільки плавне зростання навантаження, а й ситуації, у яких деградація виникає через аварійну зміну параметрів.

Для кожного інтервалу моделювання розраховуються системні, вузлові та канальні метрики. Системні метрики описують загальний стан моделі: кількість запитів за секунду, пропускну здатність, середню затримку, процентилі затримки, частку помилок, кількість успішних і помилкових запитів, перевищення часу очікування та повторні запити. Вузлові метрики показують використання CPU і RAM, кількість активних з’єднань, кількість оброблених запитів, кількість помилок і стан окремого вузла. Канальні метрики відображають використання пропускної здатності, затримку, втрату пакетів, кількість відкинутих пакетів і стан каналу.

Після розрахунку метрик система перевіряє попереджувальні та критичні пороги. Якщо значення залишаються в допустимих межах, компонент вважається стабільним. Якщо окремі показники перевищують попереджувальні межі, фіксується деградація. Якщо перевищено критичні пороги або компонент

не може обробляти задане навантаження, вузол або канал переходить у критичний стан. Ці стани надалі використовуються для визначення загального рівня ризику моделі.

Збереження результатів виконується в базі даних. Загальні системні метрики записуються окремо від метрик вузлів і каналів, що дає змогу відображати як підсумкову картину, так і деталізацію за кожним компонентом топології. Такий поділ потрібний для сторінки результатів, де користувач може переглянути загальні графіки системи, таблиці стану вузлів і каналів, а також хронологію зміни показників у часі.

Важливою частиною реалізації є збереження знімка топології під час запуску моделювання. У знімку фіксуються координати вузлів, зв’язки між ними, параметри компонентів і стан полотна. Це потрібно для того, щоб результати моделювання залишалися пов’язаними саме з тією конфігурацією, за якою вони були отримані. Якщо користувач після запуску змінить модель, попередні результати все одно будуть показувати стару топологію, а не поточну змінену конфігурацію.

Після завершення розрахунків модель переводиться у стан завершення, а сформовані метрики передаються на подальший аналіз. На цьому етапі система вже має достатній набір даних для побудови графіків, визначення стану вузлів і каналів, пошуку критичних компонентів, виконання ML-аналізу та формування PDF-звіту. Таким чином, підрозділ реалізації сценаріїв моделювання охоплює не тільки сам запуск, а й повний процес отримання даних, які використовуються в наступних частинах програмного засобу.

3.4 Реалізація алгоритмів аналізу режимів роботи та формування рекомендацій

ML-аналіз реалізовано у модулі backend/app/services/ml_analyzer.py. Після завершення моделювання з таблиці системних метрик формується набір ознак. До нього входять RPS, пропускна здатність, середня затримка, процентилі

затримки, частка помилок, кількість перевищень часу очікування, кількість повторів, CPU, RAM, кількість процесів, втрата пакетів і коливання затримки. Додатково для частини метрик формуються трендові та ковзні ознаки. Ознаки, що використовуються під час ML-аналізу, наведено в таблиці 3.1.

Таблиця 3.1 – Підсумки демонстраційних експериментів

Ознака	Джерело	Призначення
RPS	системні метрики	показує поточний рівень навантаження
Пропускна здатність	системні метрики	показує кількість успішно оброблених запитів
Середня затримка	системні метрики	описує середній час відповіді
Затримка P95/P99	системні метрики	показує поведінку повільних запитів
Частка помилок	системні метрики	показує частку помилкових запитів
Кількість перевищень часу очікування	системні метрики	фіксує перевищення часу очікування
Кількість повторів	системні метрики	показує додаткове навантаження від повторів
Використання CPU	системні метрики та метрики вузлів	описує насичення обчислювальних ресурсів
Використання RAM	системні метрики та метрики вузлів	описує використання пам’яті
Втрата пакетів	системні та канальні метрики	показує втрати в каналах
Jitter	системні метрики	показує нестабільність затримки

Використання каналу	метрики каналів	використовується для пошуку проблемного каналу

Дані таблиці 3.1 використовуються під час кластеризації режимів роботи, пошуку аномальних інтервалів і формування рекомендацій. Перед виконанням аналізу значення очищаються від порожніх або нескінченних значень, після чого для роботи з KMeans, Isolation Forest і PCA застосовується масштабування ознак. Якщо бібліотека машинного навчання не може виконати аналіз через занадто малий або некоректний набір даних, система переходить до резервної евристичної гілки.

Метод KMeans формує кластери часових точок, які відповідають різним режимам роботи системи. За середніми значеннями ознак ці кластери отримують текстові назви українською мовою: нормальна робота, підвищене навантаження, мережева деградація, деградація сервісу, критична нестабільність, відновлення або нестабільний перехід. У програмному коді цим режимам відповідають внутрішні англомовні мітки, але в інтерфейсі та пояснювальній записці вони подаються українською. Метод Isolation Forest використовується для визначення аномальних інтервалів, а PCA формує двовимірні координати для точок аналізу.

Після завершення аналізу система зберігає підсумковий результат у AnalysisResult, а дані окремих часових точок у MetricAnalysisPoint. Підсумковий результат містить кількість кластерів, кількість аномалій, оцінку ризику, рівень ризику, короткий висновок, рекомендації та фактори, які найбільше вплинули на результат аналізу.

Окремо у програмному засобі реалізовано формування PDF-звіту за результатами моделювання. Ця функція виконується на серверній стороні у модулі backend/app/services/report_generator.py. На сторінці результатів користувач натискає кнопку “Експортувати PDF”, після чого клієнтська частина викликає маршрут формування звіту і відкриває посилання для завантаження готового документа. Фрагмент PDF-звіту за результатами моделювання наведено на рисунку 3.4.

ЗВІТ АНАЛІЗУ ПОВЕДІНКИ КОМП'ЮТЕРНОЇ СИСТЕМИ

Сценарій: Планове технічне обслуговування

Дата запуску	Тривалість	Профіль навантаження
31.05.2026 15:16	4 с	Постійне навантаження

1. Резюме результату

Показник	Значення
Стан системи	Стабільна
Рівень ризику	Низький
Час деградації	Не зафіксовано
Критичний вузол	Database
Проблемний канал	Cache Server -> Database
Основна причина	Database: CPU 6%, RAM 16%; Cache Server -> Database: usage 1%, loss 0,00%

Причини критичного стану

Причина	Значення	Поріг / контекст
CPU критичного вузла	6%	critical 90%
RAM критичного вузла	16%	critical 95%
Usage каналу	1%	critical 95%
Частка помилок	0,0%	critical 10,0%

Рисунок. 3.4 – Фрагмент PDF-звіту за результатами моделювання

PDF-звіт містить загальну інформацію про запуск, підсумок стану системи, параметри моделі, стан вузлів і каналів, підсумкові метрики, висновок, рекомендації, параметри моделювання, аномальні інтервали та таблицю мережевих метрик.

PDF потрібний для збереження результатів моделювання та додавання їх до матеріалів кваліфікаційної роботи. На відміну від сторінки результатів, PDF є статичним документом і відображає стан моделі на момент формування звіту.

Для формування звіту також розглядався варіант експорту результатів у CSV або Excel. Такий формат зручний для подальшої ручної обробки даних, але менш придатний для швидкого перегляду результатів моделювання у завершеному вигляді. PDF-формат було обрано тому, що він дозволяє об’єднати підсумкові метрики, стан системи, критичні компоненти та рекомендації в одному документі, який можна зберегти окремо від вебзастосунку.

3.5 Тестування роботи системи та аналіз результатів моделювання

Тестування виконувалось на рівні основних користувацьких сценаріїв. Перевірялись створення моделі, налаштування топології, запуск моделювання, збереження результатів, ML-аналіз, відображення графіків і формування PDF-звіту. Для перевірки використано три сценарії.

Перший сценарій - стабільна система. У ньому задано помірне навантаження, нормальні ресурси вузлів, низьку втрату пакетів і нормальну затримку. Очікуваний результат: система стабільна, ризик низький, критичний вузол відсутній, проблемний канал відсутній.

Другий сценарій - пікове навантаження. У ньому задано високу кількість клієнтів, ступінчасте навантаження, перевантаження сервера застосунку і проблемний канал між сервером застосунку та базою даних. Очікуваний результат: система нестабільна, ризик критичний, критичний вузол - сервер застосунку, проблемний канал - сервер застосунку -> база даних, зростання CPU, RAM, частки помилок і затримки.

Третій сценарій - відмова вузла або зміна параметрів каналу. У ньому додається подія на певній секунді: відмова бази даних або збільшення затримки чи втрати пакетів на каналі. Очікуваний результат: зміна стану вузлів, поява попередження або критичного стану, аномальні інтервали та рекомендації. Результати тестування сценаріїв моделювання наведено у таблиці 3.2.

Таблиця 3.2 показує, що програмний засіб відрізняє стабільні сценарії від нестабільних і вказує на компонент, який найбільше вплинув на результат.

Порівняння стабільного і критичного сценаріїв дозволяє перевірити не лише коректність розрахунку метрик, а й правильність роботи аналітичної частини системи. У критичному сценарії особливу увагу приділено відображенню рівня ризику, проблемного компонента та зміни основних показників у часі. Фрагмент сторінки результатів моделювання для критичного сценарію наведено на рисунку 3.5.

Таблиця 3.2 – Результати тестування сценаріїв моделювання

Сценарій	Основні умови	Стан системи	Рівень ризику	Критичний вузол	Проблемний канал
Стабільна система	помірний RPS, достатні ресурси, низька затримка і втрата пакетів	стабільна	низький	не виявлено	не виявлено
Пікове навантаження	ступінчасте зростання RPS, перевантаження сервера застосунку	нестабільна	критичний	сервер застосунку	сервер застосунку - база даних
Подія сценарію	відмова вузла або погіршення параметрів каналу	нестабільна або з ознаками деградації	високий або критичний	база даних або сервер застосунку	канал із подією

На фрагменті інтерфейсу показано загальну інформацію сценарію моделювання. Для детальнішого аналізу результатів користувач може перейти до графіків. Основні графіки результатів моделювання наведено на рисунку 3.7.

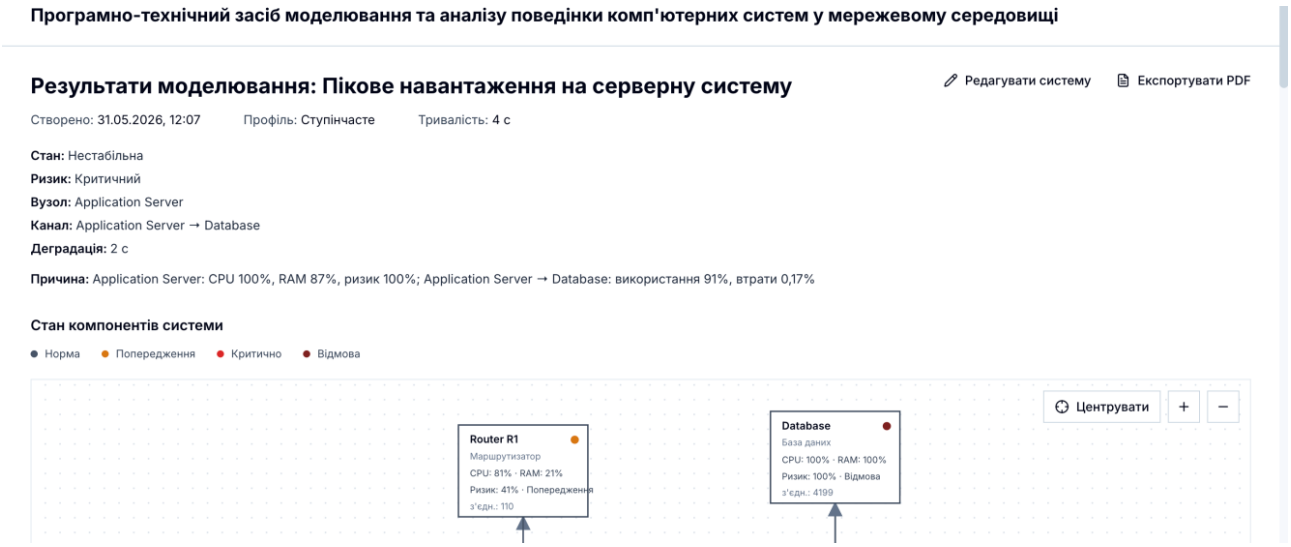


Рисунок. 3.5 – Фрагмент інтерфейсу сторінки результатів моделювання критичного сценарію.

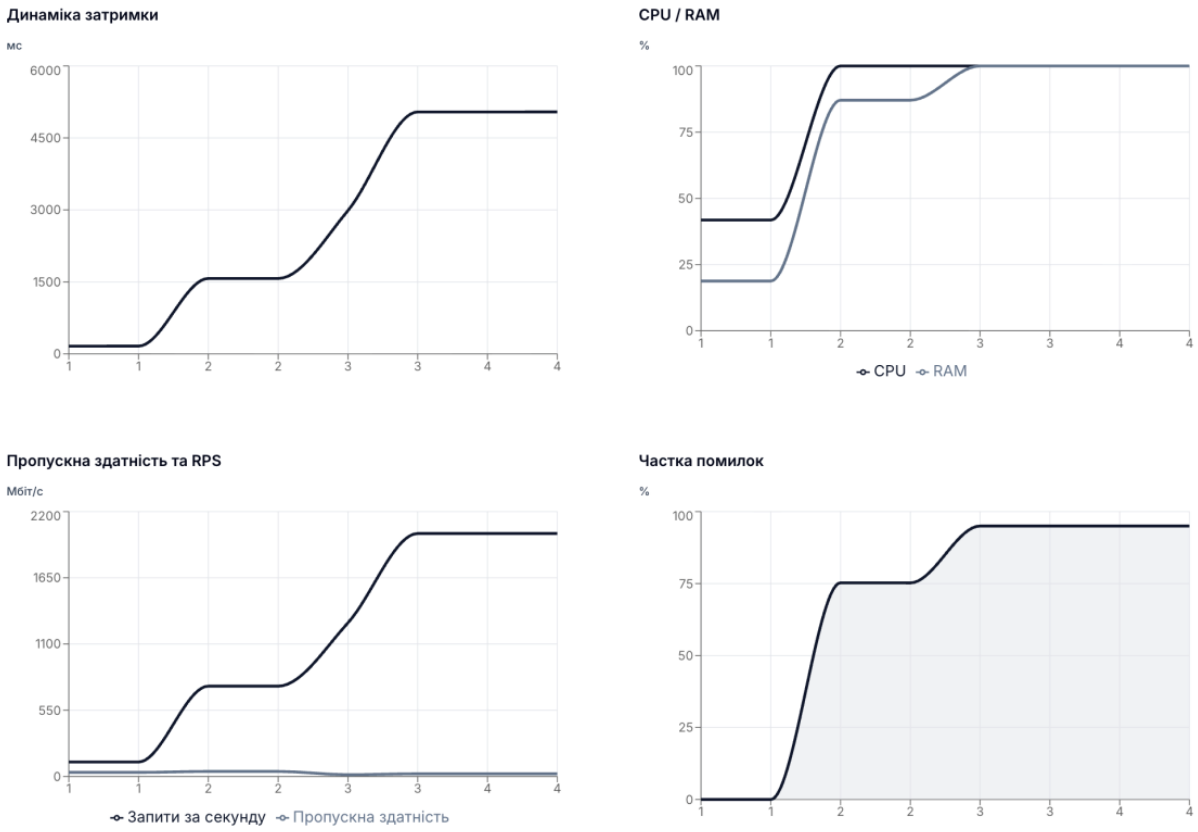


Рисунок. 3.6 – Основні графіки результатів моделювання

На рисунку 3.6 показано графіки затримки, використання CPU/RAM, RPS/пропускної здатності і частки помилок. За ними можна побачити момент деградації та зв’язати його з подією сценарію або зростанням навантаження.

Для критичного сценарію система додатково формує блок рекомендацій, у якому пояснюється, які параметри моделі можуть бути причиною нестабільності. На основі результатів аналізу користувач бачить, який вузол або канал потребує уваги, а також які зміни доцільно внести перед повторним запуском моделювання. Це дозволяє використовувати результати не тільки для фіксації проблеми, а й для подальшого коригування конфігурації системи. Фрагмент сторінки результатів із блоком рекомендацій наведено на рисунку 3.7.

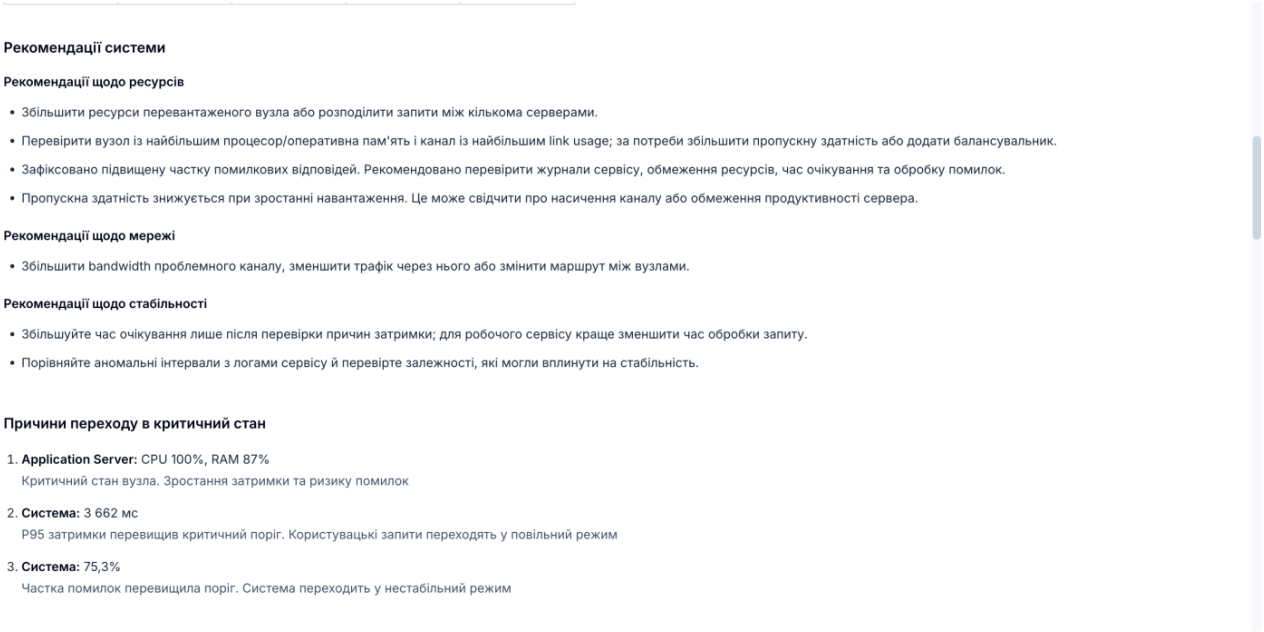


Рисунок 3.7 – Фрагмент сторінки результатів із рекомендаціями щодо покращення конфігурації системи

Під час аналізу результатів особливу увагу приділяли узгодженості між розрахованими метриками та підсумковими висновками системи. Для цього порівнювалися значення системних, вузлових і канальних показників, отримані на різних етапах виконання сценарію. Було встановлено, що зміна рівня

навантаження безпосередньо впливає на використання ресурсів вузлів, затримку обробки запитів і частку помилок. У разі перевищення встановлених порогів система коректно фіксувала ознаки деградації та відображала їх у підсумковому аналізі.

Окремо перевірялася коректність роботи механізму визначення критичного компонента. Для цього створювалися сценарії, у яких причиною погіршення роботи системи були різні елементи топології. В одних випадках нестабільність викликалася перевантаженням серверного вузла, в інших - обмеженням пропускної здатності каналу зв'язку або збільшенням затримки передачі даних. Результати перевірки показали, що система коректно визначає компонент, який найбільше вплинув на виникнення критичного стану, та використовує цю інформацію під час формування рекомендацій.

Також було перевірено роботу програмного засобу при багаторазовому запуску моделювання для однієї й тієї самої моделі. Це дозволило переконатися, що результати окремих запусків не змішуються між собою та зберігаються незалежно. Завдяки використанню знімка топології кожен результат залишається пов'язаним із конфігурацією системи, яка існувала на момент запуску моделювання. Це спрощує подальший аналіз і дає можливість порівнювати результати різних сценаріїв для однієї моделі системи.

Проведене тестування підтвердило працездатність основних функцій програмно-технічного засобу. Реалізовані механізми створення моделей, налаштування топології, запуску сценаріїв, розрахунку метрик, аналізу результатів та формування рекомендацій працюють узгоджено і забезпечують отримання інформативних результатів для подальшого дослідження поведінки комп'ютерних систем у мережевому середовищі.

Під час тестування було виявлено кілька практичних недоліків. Спочатку сторінка результатів могла залежати від поточного стану моделі, тому після редагування топології старі результати було складніше пов'язати з конфігурацією, за якою вони отримані. Для цього додано збереження знімка

топології під час запуску моделювання. Також після перевірки прибрано формування PDF зі сторінки списку моделей і залишено його на сторінці результатів.

Окремо перевірялась коректність українських підписів інтерфейсу, відображення каналів після переміщення вузлів і робота таблиць із великою кількістю метрик. Після цих перевірок уточнено частину назв полів у конструкторі, додано обмеження показу перших записів у таблицях і перевірено, що попередні результати залишаються пов’язаними зі старим знімком топології.

3.6 Висновки до третього розділу

У третьому розділі описано програмну реалізацію розробленого програмно-технічного засобу. Реалізація охоплює серверну частину, клієнтський інтерфейс, сторінку моделей систем, конструктор топології, запуск моделювання, збереження результатів, аналіз отриманих метрик, формування рекомендацій і створення PDF-звіту.

Серверна частина забезпечує роботу API, взаємодію з базою даних, запуск моделювання, розрахунок метрик, виконання ML-аналізу та формування звіту. Для збереження даних використовується реляційна структура, у якій окремо зберігаються моделі систем, вузли, канали зв’язку, події сценарію, системні метрики, метрики вузлів, метрики каналів, результати аналізу та сформовані звіти. Такий поділ дозволяє пов’язати результати моделювання з конкретною моделлю і не втрачати дані після повторного редагування конфігурації.

Клієнтська частина реалізує основний сценарій роботи користувача. Через сторінку моделей користувач створює або відкриває модель системи, у конструкторі топології налаштовує вузли, канали, навантаження, події та критерії нестабільності, а після запуску переходить до сторінки результатів. На сторінці результатів відображаються підсумковий стан системи, рівень ризику, критичний вузол, проблемний канал, графіки, таблиці метрик, результати ML-аналізу, рекомендації та можливість сформувати PDF-звіт.

Під час реалізації було враховано практичну проблему повторного запуску моделювання. Якщо користувач змінює модель після отримання результатів, старі результати не повинні прив’язуватися до нової топології. Для цього під час запуску зберігається знімок топології, який фіксує структуру системи на момент моделювання. Це дозволяє коректно переглядати результати навіть після подальшого редагування моделі.

Реалізація аналізу результатів показала, що для пояснення стану системи недостатньо лише числових метрик. Тому в системі поєднано порогові правила, які визначають попереджувальні та критичні стани, і методи машинного навчання, які допомагають виділяти режими роботи та аномальні інтервали. У результаті користувач отримує не тільки графіки, а й короткий висновок, рівень ризику, причину нестабільності та рекомендації щодо зміни параметрів вузлів, каналів або сценарію навантаження.

Проведене тестування підтвердило працездатність основного циклу системи. Програмний засіб дозволяє створити модель комп’ютерної системи, налаштувати топологію, виконати моделювання, зберегти метрики, визначити нестабільні режими, знайти критичні вузли та проблемні канали, сформувати рекомендації і підготувати PDF-звіт. Результати стабільних і критичних сценаріїв відрізняються за метриками, рівнем ризику та висновками, що підтверджує коректність реалізованої логіки.

Отже, у третьому розділі було показано, що розроблений програмно-технічний засіб реалізує основні функції, визначені у постановці задачі. Система забезпечує повний цикл роботи з моделлю комп’ютерної системи: від створення та налаштування топології до аналізу результатів моделювання і формування звіту. Отримана реалізація може використовуватися для навчального аналізу поведінки комп’ютерних систем, порівняння різних конфігурацій та демонстрації впливу навантаження, параметрів вузлів і каналів зв’язку на стабільність системи.

ВИСНОВКИ

У кваліфікаційній роботі було розроблено програмно-технічний засіб моделювання та аналізу поведінки комп'ютерних систем у мережевому середовищі. Мета роботи досягнута: реалізовано вебзастосунок, у якому користувач може створити модель системи, налаштувати вузли й канали зв'язку, задати сценарій навантаження, запустити моделювання та переглянути результати аналізу.

У першому розділі проаналізовано особливості функціонування комп'ютерних систем у мережевому середовищі. Визначено, що стабільність системи залежить від стану вузлів, параметрів каналів зв'язку, профілю навантаження і подій сценарію. Розглянуто основні фактори деградації, зокрема перевантаження CPU і RAM, зростання кількості з'єднань, збільшення затримки, втрату пакетів, обмеження пропускної здатності, помилки та відмову вузла. Також виконано аналіз існуючих програмно-технічних засобів і зроблено висновок, що для поставленої задачі потрібен окремий засіб, який поєднує створення моделі системи, побудову топології, запуск моделювання, ML-аналіз результатів і формування PDF-звіту.

У другому розділі спроєктовано архітектуру програмно-технічного засобу, структуру моделі комп'ютерної системи, логіку візуального конструктора топології, сценарії навантаження, події та критерії нестабільності. Описано алгоритм моделювання поведінки вузлів і каналів зв'язку, підсистему аналізу результатів, формування рекомендацій і структуру бази даних. Окремо передбачено збереження знімка топології, щоб результати залишалися пов'язаними з конфігурацією, яка існувала в момент запуску.

Для реалізації обрано Python, FastAPI, SQLAlchemy, SQLite/PostgreSQL, React, Vite, TypeScript, Recharts і scikit-learn. Такий набір засобів дозволив розділити серверну логіку, клієнтський інтерфейс, збереження даних, побудову

графіків і машинний аналіз результатів. Як середовище розробки використано Visual Studio Code, а для контролю версій - Git.

У третьому розділі описано фактичну реалізацію системи. Реалізовано серверну частину, клієнтський інтерфейс, сторінку моделей систем, покроковий конструктор топології, запуск моделювання, збереження системних метрик, метрик вузлів і каналів, сторінку результатів та формування PDF-звіту. Користувач може відредагувати модель після аналізу, змінити параметри системи і запустити нове моделювання.

Проведене тестування охопило стабільний сценарій, пікове навантаження та сценарій відмови вузла або погіршення параметрів каналу. У нестабільних сценаріях система визначала проблемний компонент, аномальні інтервали та формувала рекомендації щодо зміни конфігурації. Отриманий результат відповідає поставленим задачам.

Розроблений програмно-технічний засіб можна використовувати для навчального і проєктного аналізу поведінки комп’ютерних систем у мережевому середовищі. Система не замінює пакетні симулятори або промислові засоби моніторингу, але дозволяє створити керовану модель, перевірити її під різними сценаріями навантаження і отримати зрозумілий підсумок моделювання.

Подальший розвиток може включати розширення набору подій, підтримку історії кількох запусків однієї моделі, додаткові алгоритми аналізу та окремий модуль реального моніторингу як майбутнє розширення. Графічна частина роботи містить три обов’язкові креслення: архітектуру програмно-технічного засобу, схему роботи візуального конструктора топології та алгоритм моделювання поведінки системи, які наведено відповідно у додатках А, Б і В. У додатку Г наведено текст програми, що містить основні фрагменти програмного коду серверної та клієнтської частин розробленого вебзастосунку.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Tanenbaum A. S., Wetherall D. J. Computer Networks. 5th ed. Boston : Pearson, 2011. 960 p.
2. Kurose J. F., Ross K. W. Computer Networking: A Top-Down Approach. 8th ed. Boston : Pearson, 2021. 864 p.
3. Eddy W. RFC 9293: Transmission Control Protocol (TCP). RFC Editor, 2022. URL: https://www.rfc-editor.org/rfc/rfc9293 (дата звернення: 24.04.2026). DOI: 10.17487/RFC9293.
4. Fielding R., Nottingham M., Reschke J. RFC 9110: HTTP Semantics. RFC Editor, 2022. URL: https://www.rfc-editor.org/rfc/rfc9110 (дата звернення: 24.04.2026). DOI: 10.17487/RFC9110.
5. Fielding R. T. Architectural Styles and the Design of Network-based Software Architectures : doctoral dissertation. Irvine : University of California, 2000. URL: https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm (дата звернення: 24.04.2026).
6. MDN Web Docs. An overview of HTTP. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/Overview (дата звернення: 24.04.2026).
7. Harchol-Balter M. Performance Modeling and Design of Computer Systems. Cambridge : Cambridge University Press, 2013. 576 p.
8. Menasce D. A., Almeida V. A. F. Capacity Planning for Web Services: Metrics, Models, and Methods. Upper Saddle River : Prentice Hall PTR, 2001. 608 p.
9. Prometheus Authors. Prometheus Documentation. URL: https://prometheus.io/docs/introduction/overview/ (дата звернення: 24.04.2026).
10. Grafana Labs. Grafana Documentation. URL: https://grafana.com/docs/grafana/latest/ (дата звернення: 24.04.2026).
11. Apache Software Foundation. Apache JMeter User Manual. URL: https://jmeter.apache.org/usermanual/index.html (дата звернення: 24.04.2026).

12. Grafana Labs. k6 Documentation. URL: https://grafana.com/docs/k6/latest/ (дата звернення: 24.04.2026).

13. Locust. Locust Documentation. URL: https://docs.locust.io/ (дата звернення: 24.04.2026).

14. ns-3 Consortium. ns-3 Manual. URL: https://www.nsnam.org/docs/manual/html/ (дата звернення: 24.04.2026).

15. Mininet Team. Mininet Documentation. URL: http://mininet.org/walkthrough/ (дата звернення: 24.04.2026).

16. Linux Foundation. Linux Traffic Control. URL: https://docs.kernel.org/networking/tc-queue-filters.html (дата звернення: 24.04.2026).

17. OpenTelemetry Authors. OpenTelemetry Documentation. URL: https://opentelemetry.io/docs/ (дата звернення: 24.04.2026).

18. Jaeger Authors. Jaeger Documentation. URL: https://www.jaegertracing.io/docs/ (дата звернення: 24.04.2026).

19. Elastic. Elasticsearch Guide. URL: https://www.elastic.co/guide/en/elasticsearch/reference/current/index.html (дата звернення: 24.04.2026).

20. FastAPI. FastAPI Documentation. URL: https://fastapi.tiangolo.com/ (дата звернення: 24.04.2026).

21. React Team. React Documentation. URL: https://react.dev/ (дата звернення: 24.04.2026).

22. Vite Team. Vite Documentation. URL: https://vite.dev/guide/ (дата звернення: 24.04.2026).

23. SQLAlchemy Authors. SQLAlchemy Documentation. URL: https://docs.sqlalchemy.org/ (дата звернення: 24.04.2026).

24. Docker. Docker Compose Documentation. URL: https://docs.docker.com/compose/ (дата звернення: 24.04.2026).

25. Render. Render Documentation. URL: https://render.com/docs (дата звернення: 24.04.2026).

26. Vercel. Vercel Documentation. URL: https://vercel.com/docs (дата звернення: 24.04.2026).

27. MacQueen J. Some Methods for Classification and Analysis of Multivariate Observations. Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability. 1967. Vol. 1. P. 281–297.

28. scikit-learn developers. sklearn.cluster.KMeans. URL: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html (дата звернення: 24.04.2026).

29. Liu F. T., Ting K. M., Zhou Z.-H. Isolation Forest. Proceedings of the 2008 IEEE International Conference on Data Mining. 2008. P. 413–422. DOI: 10.1109/ICDM.2008.17.

30. scikit-learn developers. sklearn.ensemble.IsolationForest. URL: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html (дата звернення: 24.04.2026).

31. Jolliffe I. T. Principal Component Analysis. 2nd ed. New York : Springer, 2002. 487 p.

32. scikit-learn developers. sklearn.decomposition.PCA. URL: https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html (дата звернення: 24.04.2026).

33. scikit-learn developers. User Guide. URL: https://scikit-learn.org/stable/user_guide.html (дата звернення: 24.04.2026).

34. Bass L., Clements P., Kazman R. Software Architecture in Practice. 4th ed. Boston : Addison-Wesley, 2021. 464 p.

35. Fowler M. Patterns of Enterprise Application Architecture. Boston : Addison-Wesley, 2002. 560 p.

36. Python Software Foundation. Python Documentation. URL: https://docs.python.org/3/ (дата звернення: 24.04.2026).

37. Pydantic. Pydantic Documentation. URL: https://docs.pydantic.dev/ (дата звернення: 24.04.2026).

38. Encode. Uvicorn Documentation. URL: https://www.uvicorn.org/ (дата звернення: 24.04.2026).

39. SQLite Consortium. SQLite Documentation. URL: https://sqlite.org/docs.html (дата звернення: 24.04.2026).

40. PostgreSQL Global Development Group. PostgreSQL Documentation. URL: https://www.postgresql.org/docs/ (дата звернення: 24.04.2026).

41. Kleppmann M. Designing Data-Intensive Applications. Sebastopol : O'Reilly Media, 2017. 616 p.

42. TypeScript Team. TypeScript Documentation. URL: https://www.typescriptlang.org/docs/ (дата звернення: 24.04.2026).

43. Recharts Group. Recharts Documentation. URL: https://recharts.org/ (дата звернення: 24.04.2026).

44. Microsoft. Visual Studio Code Documentation. URL: https://code.visualstudio.com/docs (дата звернення: 24.04.2026).

45. Chacon S., Straub B. Pro Git. 2nd ed. New York : Apress, 2014. URL: https://git-scm.com/book/en/v2 (дата звернення: 24.04.2026).

46. GitHub Docs. Hello World. URL: https://docs.github.com/en/get-started/start-your-journey/hello-world (дата звернення: 24.04.2026).

47. OpenAPI Initiative. OpenAPI Specification v3.1.0. URL: https://spec.openapis.org/oas/v3.1.0.html (дата звернення: 24.04.2026).

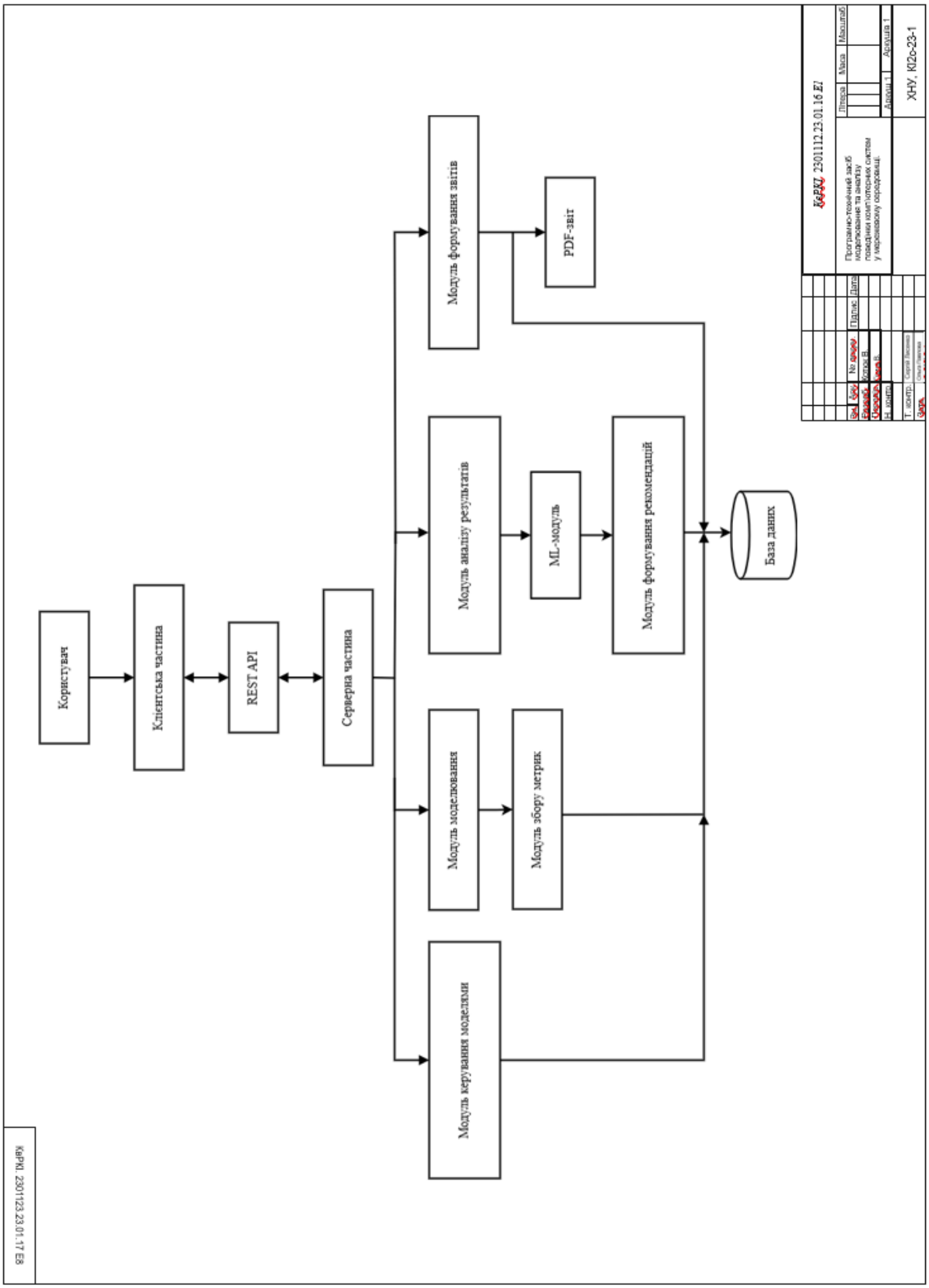
48. Beyer B., Jones C., Petoff J., Murphy N. R. Site Reliability Engineering: How Google Runs Production Systems. Sebastopol : O'Reilly Media, 2016. 552 p. URL: https://sre.google/sre-book/table-of-contents/ (дата звернення: 24.04.2026).

49. MDN Web Docs. Fetch API. URL: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API (дата звернення: 24.04.2026).

ДОДАТОК А

(обов’язковий)

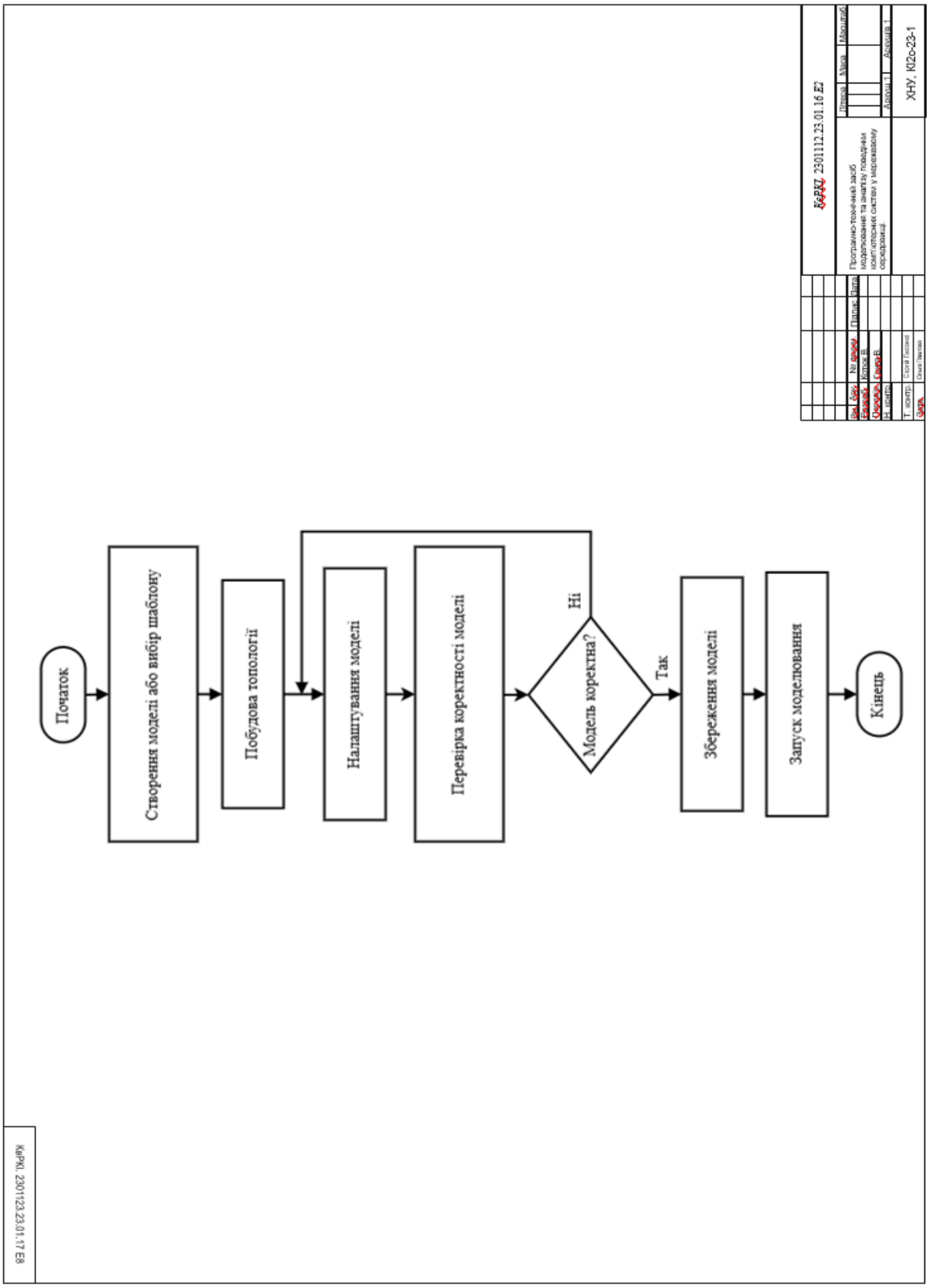
Копія креслення «Структурна схема програмно-технічного засобу»



ДОДАТОК Б

(обов’язковий)

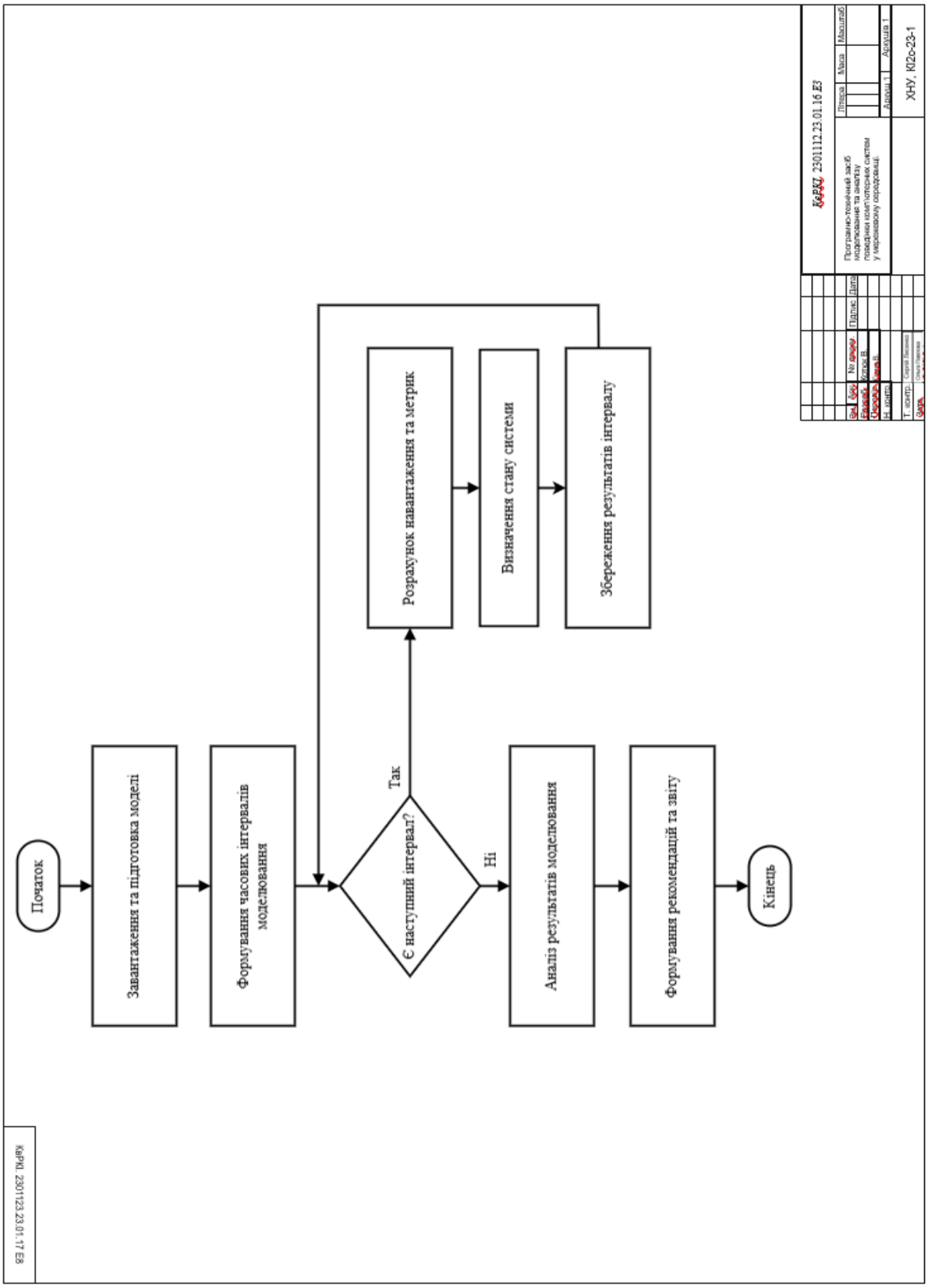
Копія креслення «Блок-схема роботи візуального конструктора топології»



ДОДАТОК В

(обов’язковий)

Копія креслення «Блок-схема алгоритму моделювання поведінки системи»



ДОДАТОК Г

Текст програми

```
from contextlib import asynccontextmanager

from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware

from app.api import analysis, experiments, reports
from app.config import settings
from app.database import SessionLocal, init_db
from app.schemas import HealthResponse
from app.seed import seed_all


@asynccontextmanager
async def lifespan(app: FastAPI):
    init_db()
    db = SessionLocal()
    try:
        if settings.AUTO_SEED_DEMO:
            seed_all(db)
    finally:
        db.close()
    yield


app = FastAPI(
    title="Програмно-технічний засіб для моделювання",
    description="Платформа моделювання мережевих вузлів, каналів зв'язку, навантаження, ML-аналізу нестабільності та формування PDF-звітів.",
    version="1.0.0",
    lifespan=lifespan,
)

app.add_middleware(
    CORSMiddleware,
    allow_origins=settings.cors_origins,
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

api_prefix = settings.API_PREFIX.rstrip("/")
app.include_router(experiments.router, prefix=api_prefix)
app.include_router(experiments.scenario_router, prefix=api_prefix)
app.include_router(analysis.router, prefix=api_prefix)
app.include_router(reports.router, prefix=api_prefix)
```

```
@app.get(f"{api_prefix}/health", response_model=HealthResponse, tags=["health"])
def health() -> HealthResponse:
    database = "sqlite" if settings.sqlalchemy_url.startswith("sqlite") else "postgresql"
    return HealthResponse(
        status="ok",
        environment=settings.ENVIRONMENT,
        database=database,
        api_prefix=api_prefix,
    )
@router.post("/{experiment_id}/run")
async def run_experiment(experiment_id: int, db: Session = Depends(get_db)) -> dict[str, str | int]:
    experiment = db.get(Experiment, experiment_id)
    if not experiment:
        raise HTTPException(status_code=404, detail="Experiment not found")
    if experiment.mode != "simulation":
        raise HTTPException(status_code=410, detail="Реальний моніторинг видалено з проєкту")
    if not start_experiment_job(experiment_id):
        raise HTTPException(status_code=409, detail="Experiment is already running")
    return {"status": "running", "experiment_id": experiment_id}


@router.get("/{experiment_id}/metrics")
def experiment_metrics(experiment_id: int, db: Session = Depends(get_db)) -> list[dict]:
    experiment = db.get(Experiment, experiment_id)
    if not experiment:
        raise HTTPException(status_code=404, detail="Experiment not found")
    metrics = (
        db.query(Metric).filter(Metric.experiment_id == experiment_id).order_by(Metric.elapsed_seconds.asc()).all()
    )
    return [
        {
            column.name: getattr(metric, column.name).isoformat()
            if column.name == "timestamp"
            else getattr(metric, column.name)
            for column in Metric.__table__.columns
        }
```

```
            for metric in metrics
        ]
    class SimulationEngine:
        async def run(self, db: Session, experiment: Experiment, stop_event: asyncio.Event) -> None:
            interval = max(0.2, float(experiment.interval_seconds or 1.0))
            duration = max(1, int(experiment.duration_seconds or 1))
            total_intervals = max(1, math.ceil(duration / interval))
            previous_latency = float(experiment.latency_ms or 30)

            random.seed(experiment.id * 17 + duration)

            for index in range(total_intervals):
                if stop_event.is_set():
                    break

                elapsed = min(duration, round((index + 1) * interval, 3))
                system_metric, node_metrics, link_metrics, previous_latency = self._simulate_interval(
                    db=db,
                    experiment=experiment,
                    elapsed_seconds=elapsed,
                    interval=interval,
                    previous_latency=previous_latency,
                )

                system_metric["timestamp"] = utc_now()
                db.add(Metric(experiment_id=experiment.id, **system_metric))
                db.add_all(NodeMetric(experiment_id=experiment.id, **payload) for payload in node_metrics)
                db.add_all(LinkMetric(experiment_id=experiment.id, **payload) for payload in link_metrics)
                db.commit()

        def _simulate_interval(self, db: Session, experiment: Experiment, elapsed_seconds: float, interval: float, previous_latency: float):
            nodes = db.query(NetworkNode).filter(NetworkNode.experiment_id == experiment.id).order_by(NetworkNode.id.asc()).all()
            links = db.query(NetworkLink).filter(NetworkLink.experiment_id == experiment.id).order_by(NetworkLink.id.asc()).all()
            events = db.query(ScenarioEvent).filter(ScenarioEvent.experiment_id ==
```

```
experiment.id).order_by(ScenarioEvent.
time_second.asc()).all()

            event_state            =
self._event_state(events,
elapsed_seconds)
            current_rps            =
self._current_rps(experiment,
elapsed_seconds)                    *
float(event_state.get("traffic_multipl
ier", 1.0))
            request_size_kb        =
self._request_size_kb(experiment)
            traffic_mbps           =
current_rps  *  request_size_kb  *  8  /
1024

            link_snapshots = [

self._link_snapshot(link,
traffic_mbps, current_rps, experiment,
event_state)
                for link in links
            ]
            node_loads             =
self._distribute_node_loads(nodes,
current_rps,
set(event_state["failed_node_ids"]))
            node_snapshots = [

self._node_snapshot(node,
node_loads.get(node.id,          0.0),
interval,                  current_rps,
set(event_state["failed_node_ids"]),
event_state)
                for node in nodes
            ]

            max_cpu                =
max(snapshot.cpu_usage for snapshot in
node_snapshots)
            max_ram                =
max(snapshot.ram_usage for snapshot in
node_snapshots)
            link_latency           =
sum(snapshot.latency_ms  for  snapshot
in link_snapshots)
            link_packet_loss       =
max([snapshot.packet_loss_observed for
snapshot in link_snapshots] or [0.0])

            latency_avg  =  max(1.0,
float(experiment.latency_ms  or  0)  +
link_latency)
            error_rate             =
self._error_rate(
                max_cpu=max_cpu,
                max_ram=max_ram,

max_link_usage=max([snapshot.link_usag
e_percent     for     snapshot     in
link_snapshots] or [0.0]),

packet_loss=link_packet_loss,

latency_avg=latency_avg,
                timeout_risk=0.0,
                failed_nodes=sum(1
for  snapshot  in  node_snapshots  if
snapshot.status == "failed"),

experiment=experiment,
            )
```

```
            planned_requests       =
max(1,     int(round(current_rps     *
interval)))
            error_count            =
int(round(planned_requests           *
error_rate / 100))
            success_count  =  max(0,
planned_requests - error_count)

            system_metric = {
                "elapsed_seconds":
elapsed_seconds,
                "rps":
round(planned_requests / max(interval,
0.001), 3),
                "latency_avg":
round(latency_avg, 3),
                "error_rate":
round(error_rate, 3),
                "success_count":
success_count,
                "error_count":
error_count,
                "cpu_usage":
round(max_cpu, 3),
                "ram_usage":
round(max_ram, 3),

"packet_loss_observed":
round(link_packet_loss, 3),
            }
            return    system_metric,
[], [], latency_avg
     def run_ml_analysis(db: Session,
experiment_id: int) -> AnalysisResult:
         metrics = (
db.query(Metric).filter(Metric.experim
ent_id                              ==
experiment_id).order_by(Metric.elapsed
_seconds.asc()).all()
         )
         experiment                    =
db.get(Experiment, experiment_id)
         if not experiment:
             raise
ValueError("Experiment not found")

         df                            =
_metrics_dataframe(metrics)
         feature_df                    =
_feature_dataframe(df)

         try:
             labels,   cluster_count,
pca_points,           anomaly_scores,
anomaly_flags                         =
_sklearn_analysis(feature_df)
         except Exception:
             labels,   cluster_count,
pca_points,           anomaly_scores,
anomaly_flags                         =
_heuristic_analysis(feature_df)

         point_risks = [

_point_risk_score(df.iloc[index].to_di
ct(), anomaly_scores[index]) for index
in range(len(df))
         ]
         anomaly_count   =   int(sum(1
for flag in anomaly_flags if flag))
```

```
        overall_risk                =
_overall_risk_score(df, anomaly_count,
point_risks, _throughput_drop(df))

        recommendations             =
build_recommendations(metrics,
overall_risk, anomaly_count)
        result = AnalysisResult(

experiment_id=experiment_id,
            algorithm="KMeans       +
IsolationForest + PCA",

cluster_count=cluster_count,

anomaly_count=anomaly_count,

risk_score=round(overall_risk, 2),

risk_level=risk_level(overall_risk),

summary=recommendations["summary"],

recommendations_json=recommendations,
        )
        db.add(result)
        db.commit()
        db.refresh(result)
        return result


    def
_sklearn_analysis(feature_df:
pd.DataFrame) -> tuple[list[int], int,
list[list[float]],        list[float],
list[bool]]:
        from sklearn.cluster import
KMeans
        from   sklearn.decomposition
import PCA
        from sklearn.ensemble import
IsolationForest
        from   sklearn.preprocessing
import StandardScaler

        row_count = len(feature_df)
        scaler = StandardScaler()
        x_scaled                    =
scaler.fit_transform(feature_df.to_num
py(dtype=float))

        cluster_count    =    min(4,
row_count)  if  row_count  >=  4  else
max(1, row_count)
        labels                      =
KMeans(n_clusters=cluster_count,
random_state=42,
n_init="auto").fit_predict(x_scaled).t
olist()
        pca_points                  =
PCA(n_components=2,
random_state=42).fit_transform(x_scale
d).tolist()  if  row_count  >=  2  else
[[0.0, 0.0]]

        if row_count >= 4:
            model                   =
IsolationForest(contamination=min(0.2,
max(0.05,     2     /     row_count)),
random_state=42)
            predictions             =
model.fit_predict(x_scaled)
```

```
            raw_scores      =      (-
model.score_samples(x_scaled)).tolist(
)
            anomaly_scores          =
normalize_0_1(raw_scores)
            anomaly_flags           =
[prediction  ==  -1  for  prediction  in
predictions]
        else:
            anomaly_scores   =   [0.0
for _ in range(row_count)]
            anomaly_flags  =  [False
for _ in range(row_count)]

        return               labels,
cluster_count,             pca_points,
anomaly_scores, anomaly_flags
```

Протокол аналізу звіту подібності експертом

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

Автор: Володимир КОТЮК

Співавтор:

Назва: Програмно-технічний засіб моделювання та аналізу поведінки комп'ютерних систем у мережевому середовищі

Експерт: Володимир ГРИГА

Підрозділ: Кафедра комп'ютерної інженерії та інформаційних систем

Коефіцієнт подібності 1:2.63%

Коефіцієнт подібності 2:0.26%

Мікропробіли: 3

Заміна букв: 0

Інтервали: 0.

Білі знаки: 0

Дата створення звіту: 2026-06-10 16:55:52.0

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:

2026-06-10 Доцент Андрій Нічепорук

Дата експерт

Tue Jun 09 21:07:18 EEST 2026, Медзатий Дмитро Миколайович, Хмельницький національний університет, ХНУ

Anti-Plagiarism (http://ap.km.ua) v-15.701

Максимальне співпадіння з одним документом 1.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 8%**

ID: 274479 Назва: БКР Програмно-технічний засіб моделювання та аналізу поведінки комп'ютерних систем у мережевому середовищі Додано в БД: 2026-06-09 Автора: Володимир КОТЮК Керівники: Володимир ГРИГА Консультанти: Опоненти:	Документ		Сумарний збіг по Базі Даних	
	Символи	Лексеми	Символи	Лексеми
	97090	820	3244 (3%)	47 (6%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

Дипломник: Котюк Володимир Олександрович

Тема: Програмно-технічний засіб моделювання та аналізу поведінки комп'ютерних систем у мережевому середовищі.

Спеціальність: 123 «Комп'ютерна інженерія»

Обсяг кваліфікаційної роботи:

Кількість листів креслень ___3___ Кількість сторінок записки ___59___

1. Короткий зміст роботи та прийнятих рішень: Метою кваліфікаційної роботи є розроблення програмно-технічного засобу моделювання та аналізу поведінки комп'ютерних систем у мережевому середовищі.

2. Висновок про відповідність роботи дипломному завданню: Робота повністю відповідає поставленому завданню та вимогам до кваліфікаційної роботи.

3. Характеристика виконання кожного розділу, ступінь використання останніх досягнень науки і техніки і передових методів роботи: В першому розділі кваліфікаційної роботи досліджено предметну область: проаналізовано особливості функціонування комп'ютерних систем у мережевому середовищі, фактори деградації їх роботи, існуючі засоби моделювання й моніторингу, а також методи машинного навчання для виявлення нестабільних режимів. В другому розділі виконано проєктування програмно-технічного засобу: розроблено загальну архітектуру системи, обґрунтовано вибір технологій, спроєктовано модель комп'ютерної системи, структуру топології, підсистеми моделювання, аналізу результатів та візуалізації даних. В третьому розділі здійснено програмну реалізацію та перевірку роботи засобу: реалізовано серверну й клієнтську частини, візуальний конструктор топології, механізми моделювання, збору й аналізу метрик, алгоритми виявлення нестабільних режимів і формування рекомендацій, а також проведено тестування системи.

4. Позитивні сторони роботи: висока практична цінність роботи.

5. Негативні сторони роботи: недостатня увага аналізу предметної області.

6. Оцінка графічного оформлення та пояснювальної записки роботи: Пояснювальна записка оформлена коректно, згідно діючих стандартів оформлення документації.

7. Відгук про роботу в цілому: Робота виконана на належному технічному рівні.

8. Інші зауваження: _____

9. Оцінка дипломної роботи: добре (B / 86)

Рецензент (прізвище, ім'я, по батькові, посада, місце роботи) _____

Сіверюх Микола Васильович, PhD, ст. викладач

к-ф. Кібербезпека

"10" червня 2026 р. _____(підпис)

Зав. кафедри КІІС
д-р. філософії Ользі ПАВЛОВІЙ

Володимир КОТЮК
ПІБ здобувача вищої освіти

ФІТ, 3 курсу, групи КІ2с-23-1

ЗАЯВА

З правилами чинного Положення про систему забезпечення академічної доброчесності у Хмельницькому національному університеті, згідно з яким виявлення академічного плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту і застосування заходів академічної відповідальності, ознайомлений (а). Про використання спеціалізованих програмних засобів (СПЗ) StrikePlagiarism та Anti-Plagiarism для перевірки кваліфікаційних робіт здобувачів вищої освіти на наявність академічного плагіату оповіщений (а). Надаю університету право на передачу моєї роботи для обробки та збереження в базах даних СПЗ і використання роботи для виявлення академічного плагіату в інших роботах, які перевіряються СПЗ.

Також надаю свою згоду на обробку й збереження університетом моєї роботи в Інституційному репозитарії Хмельницького національного університету.

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

1 травня 2026 року

РІШЕННЯ ЕКСПЕРТНОЇ КОМІСІЇ

КАФЕДРИ КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ІНФОРМАЦІЙНИХ СИСТЕМ
ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Назва кваліфікаційної роботи Програмно-технічний засіб моделювання та аналізу поведінки комп'ютерних систем у мережевому середовищі
Автор Володимир КОТЮК
Освітня програма Комп'ютерна інженерія та програмування
Рівень вищої освіти перший (бакалаврський)
Спеціальність 123 Комп'ютерна інженерія
Науковий керівник: к.т.н., доцент Володимир ГРИГА

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) усі запозичення фрагментарні, або мають належним чином оформленні посилання;

2) окремі виявлені збіги є загальновживаними фразами або виразами, про що свідчить посилання системи на збіг з джерелами на один фрагмент речення;

3) всі зафіксовані системою ознаки модифікації тексту відносяться до комбінування латинських символів зі україномовними скороченнями індексів в формулах, що не є модифікацією тексту.

4) значна частина знайденого плагіату відноситься до списку використаних джерел

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів/ ідентичності/схожості StrikePlagiarism, складає 2,63%; та системою Anti-Plagiarism складає 1%, що, з урахуванням наведених обґрунтувань, відповідає характеру наукового дослідження і свідчить на користь кваліфікаційної роботи.

01.06.2026

Завідувач кафедри _____ Підпис — Ольга ПАВЛОВА (Ім'я, ПРІЗВИЩЕ)

Гарант освітньої програми _____ Підпис — Андрій НІЧЕПОРУК (Ім'я, ПРІЗВИЩЕ)

Керівник кваліфікаційної роботи _____ Підпис — Володимир ГРИГА (Ім'я, ПРІЗВИЩЕ)