

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

«Вебзастосунок для моніторингу відмовостійкості сервісів за принципами Chaos Engineering»

Виконав студент групи ЗПЗ-22-1
ГОРШКОВ Сергій Вікторович

Керівник роботи:
ТАЛАВЕР Олег Володимирович

Рецензент:
ГРОМСЬКИЙ Олександр Олександрович

Житомир – 2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення

Тетяна Вакалюк

«28» лютого 2026 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ГОРШКОВ Сергій Вікторович**

Керівник роботи: **ТАЛАВЕР Олег Володимирович**

Тема роботи: **«Вебзастосунок для моніторингу відмовостійкості сервісів за принципами Chaos Engineering»**,

затверджена Наказом закладу вищої освіти від **«27» лютого 2026 р., №93/с**

Вихідні дані для роботи: **предметом дослідження є методи, алгоритми та програмні засоби розробки веборієнтованої системи chaos engineering для контрольованого внесення збоїв у HTTP-трафік, моніторингу запитів. Об'єктом дослідження є процес забезпечення та контролю відмовостійкості розподілених вебсервісів у інформаційних системах.**

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Талавер О.В.	01.03.2026р.	01.03.2026р
2	Талавер О.В.	03.04.2026р	03.04.2026р
3	Талавер О.В.	05.05.2026р	05.05.2026р

РЕФЕРАТ

Випускна кваліфікаційна робота складається з вступу, 3-х розділів, висновків, списку використаних джерел із 34 найменувань (3 стор.), 3 додатків (10 стор.), 12 таблиць та 21 ілюстрація по тексту. Повний обсяг роботи становить 84 сторінки, з них 74 сторінки основного тексту.

Мета роботи – розробка та реалізація веборієнтованої системи моніторингу відмовостійкості програмних сервісів із вбудованим механізмом проведення Chaos Engineering-експериментів.

У першому розділі охарактеризовано предметну область chaos engineering та моніторингу надійності розподілених систем, виконано огляд і порівняльний аналіз існуючих підходів до імітації мережевих збоїв і спостереження за HTTP-трафіком, сформульовано задачу дослідження та обґрунтовано вибір технологій реалізації (Python, FastAPI, SQLAlchemy, SQLite, Jinja2, Bootstrap 5, Plotly.js).

У другому розділі визначено функціональні та нефункціональні вимоги до підсистеми хаос-проксі та моніторингу, обрано архітектурний підхід із розділенням панелі керування, окремих точок входу для сервісів і цільових бекенд-адрес виходу. Спроектовано структуру програмного комплексу, модель даних (сервіси, експерименти, метрики, інциденти, користувачі) і механізми взаємодії між компонентами: прийом HTTP-запитів на порті входу, застосування хаос-ефектів, пересилання на бекенд, запис метрик у базу даних та відображення в вебінтерфейсі.

У третьому розділі описано реалізацію серверної та клієнтської частин системи: HTTP-проксі на базі FastAPI з асинхронною обробкою запитів (httpx), паралельний запуск слухачів uvicorn для кожного сервісу, модулі хаос-ін'єкції, збору метрик, алертів і аналітики; вебінтерфейс на Jinja2, Bootstrap 5 і JavaScript; візуалізацію звітів за допомогою Plotly.js. Наведено конфігурацію середовища

					ІПЗ.КР.Б-121-26-ПЗ			
Змн.	Арк.	№ докум.	Підпис	Дата	Вебзастосунок для моніторингу відмовостійкості сервісів за принципами Chaos Engineering Пояснювальна записка	Лім.	Арк.	Аркушів
Розроб.		Гориков С.В.					3	84
Керівник		Талавер О.В.						
Рецензент		Громський О.О.				Житомирська політехніка група ЗІПЗ 22-1		
Зав. Каф.		Вакалюк Т.А.						

розгортання, механізми авторизації користувачів (ролі адміністратора та користувача), приклади користувацького інтерфейсу та результати автоматизованого тестування (pytest).

В результаті виконання роботи розроблено програмний комплекс, що забезпечує повний цикл chaos engineering на рівні HTTP-проксі: від реєстрації сервісів із заданими точками входу та виходу, запуску експериментів різних типів (затримка, імітація втрат з'єднання, таймаут, випадкові помилки, навантаження, комбінований хаос-експеримент та готові сценарії) і спостереження за потоком запитів без активного хаос, до формування аналітичних звітів, експорту даних у CSV та фіксації інцидентів у журналі активності. Результати роботи можуть бути застосовані в DevOps-практиках підприємств для підвищення надійності мікросервісної архітектури та перевірки стійкості клієнт-застосунків до типових мережесих відмов.

КЛЮЧОВІ СЛОВА: CHAOS ENGINEERING, HTTP-ПРОКСІ, МОНІТОРИНГ ВІДМОВОСТІЙКОСТІ, FASTAPI, SQLITE, ХАОС-ЕКСПЕРИМЕНТ, МЕТРИКИ, АНАЛІТИКА ВПЛИВУ ХАОСУ, PYTHON.

		Горшков С.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Талавер О.В.				4
Змн.	Арк.	№ докум.	Підпис	Дата		

ABSTRACT

The final qualification work consists of an introduction, 3 chapters, conclusions, a list of sources used with 34 names (3 pages), 3 appendices (10 pages), 12 tables and 21 illustrations in the text. The full volume of the work is 84 pages, of which 74 pages are the main text.

The purpose of the work is to develop and implement a web-oriented system for monitoring the fault tolerance of software services with a built-in mechanism for conducting Chaos Engineering experiments.

The first section describes the subject area of chaos engineering and monitoring the reliability of distributed systems, provides a review and comparative analysis of existing approaches to simulating network failures and monitoring HTTP traffic, formulates the research problem and justifies the choice of implementation technologies (Python, FastAPI, SQLAlchemy, SQLite, Jinja2, Bootstrap 5, Plotly.js).

The second section defines functional and non-functional requirements for the chaos proxy and monitoring subsystem, and an architectural approach is chosen with the separation of the control panel, separate entry points for services and target backend exit addresses. The structure of the software complex, the data model (services, experiments, metrics, incidents, users) and the interaction mechanisms between the components are designed: receiving HTTP requests at the entry port, applying chaos effects, forwarding to the backend, writing metrics to the database and displaying in the web interface. The third section describes the implementation of the server and client parts of the system: HTTP proxy based on FastAPI with asynchronous request processing (httpx), parallel launch of uvicorn listeners for each service, chaos injection, metrics collection, alerts and analytics modules; web interface on Jinja2, Bootstrap 5 and JavaScript; report visualization using Plotly.js. The configuration of the deployment environment, user authorization mechanisms (administrator and user roles), user interface examples and automated testing results (pytest) are presented.

As a result of the work, a software package was developed that provides a full cycle of chaos engineering at the HTTP proxy level: from registering services with specified entry and exit points, launching experiments of various types (delay,

		Горшков С.В.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Талавер О.В.				5
Змн.	Арк.	№ докум.	Підпис	Дата		

simulated connection losses, timeout, random errors, load, combined chaos experiment and ready-made scenarios) and monitoring the flow of requests without active chaos, to generating analytical reports, exporting data to CSV and recording incidents in the activity log. The results of the work can be applied in DevOps practices of enterprises to increase the reliability of microservice architecture and test the resilience of client applications to typical network failures.

KEYWORDS: CHAOS ENGINEERING, HTTP PROXY, RESILIENCE MONITORING, FASTAPI, SQLITE, CHAOS EXPERIMENT, METRICS, CHAOS IMPACT ANALYSIS, PYTHON.

		Горшков С.В.			ІПЗ.КР.Б-121-26-ІПЗ	Арк.
		Талавер О.В.				6
Змн.	Арк.	№ докум.	Підпис	Дата		

ЗМІСТ

ВСТУП.....	9
РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ВИБІР ІНСТРУМЕНТІВ	12
1.1. Постановка задачі	12
1.2. Аналіз аналогів програмного продукту.....	13
1.3. Архітектура системи моніторингу відмовостійкості	199
1.4. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення.....	22
Висновки до розділу 1	26
РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ СИСТЕМИ	27
2.1. Варіанти використання та структура системи	27
2.2. Структура бази даних.....	34
2.3. Проєктування алгоритмів роботи системи	40
2.4. Реалізація функціоналу системи	49
Висновки до розділу 2.....	53
РОЗДІЛ 3. ІНТЕРФЕЙС, ТЕСТУВАННЯ ТА РОЗГОРТАННЯ	55
3.1. Порядок встановлення та налаштування системи.....	55
3.2. Структура інтерфейсу системи	58
3.3. Тестування програмного комплексу.....	64
Висновки до розділу 3.....	68
ВИСНОВКИ.....	70
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	72
ДОДАТКИ.....	75
Додаток А	76
Додаток Б.....	78
Додаток В	82

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API (Application Programming Interface) - інтерфейс програмування застосунків
CP (Chaos Proxy) - хаос проксі
CE (Chaos Engineering) - хаос-інженерія
CORS (Cross-Origin Resource Sharing) - спільне використання ресурсів між різними джерелами
CPU (Central Processing Unit) - центральний процесор
CRUD (Create, Read, Update, Delete) - базові операції з даними
HTTP (HyperText Transfer Protocol) - протокол передачі гіпертексту
JSON (JavaScript Object Notation) - формат обміну даними
JWT (JSON Web Token) - вебтокен JSON для автентифікації
ORM (Object-Relational Mapping) - об'єктно-реляційне відображення
REST (Representational State Transfer) - архітектурний стиль вебсервісів
SLA (Service Level Agreement) - угода про рівень обслуговування
SLO (Service Level Objective) - цільовий показник рівня обслуговування
SQL (Structured Query Language) - мова структурованих запитів
URL (Uniform Resource Locator) - уніфікований локатор ресурсів
CNCF (Cloud Native Computing Foundation) - фонд хмарних нативних обчислень
CRD (Custom Resource Definition) - користувацьке визначення ресурсу

		Горшков С.В.			ІПЗ.КР.Б-121-26-ІПЗ	Арк.
		Талавер О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		8

ВСТУП

Актуальність теми у сучасній розробці програмного забезпечення все частіше використовуються мікросервісні архітектури. Такий підхід дозволяє окремо розгортати та масштабувати компоненти системи, проте ускладнює контроль її стабільної роботи. Навіть незначна відмова одного сервісу може вплинути на роботу інших компонентів та призвести до погіршення якості обслуговування користувачів.

Парадигма Chaos Engineering, яка зародилася в Netflix у 2010-х роках, пропонує системний підхід до підвищення надійності: у контрольованих умовах в систему навмисно вводяться збої різного характеру, а її реакція ретельно вимірюється та аналізується. Таким чином вдається виявити приховані вразливості та перевірити дієвість механізмів відновлення задовго до того, як вони проявляться в робочому середовищі. Методологічні основи дисципліни систематично викладено в працях Basiri та Rosenthal, де підкреслюється необхідність формування гіпотез про стабільний стан системи та вимірювання реакції системи на контрольовані збої [9, 20]. Відповідно до звіту DevOps Research and Assessment 2023, організації, що впроваджують практики Chaos Engineering, демонструють вищий рівень доступності сервісів порівняно з тими, що не використовують подібних підходів [1]. Підхід Site Reliability Engineering також підтверджує доцільність проактивного тестування відмовостійкості до виникнення інцидентів у продуктивному середовищі [29].

Водночас більшість існуючих інструментів Chaos Engineering - таких як Chaos Monkey, Gremlin або LitmusChaos - орієнтовані на корпоративний сегмент і потребують складного розгортання в хмарній або контейнеризованій інфраструктурі. Для малих та середніх команд розробників існує очевидний дефіцит легких у розгортанні рішень, що поєднують функції моніторингу та хаос-тестування в єдиному інтерфейсі. Розробка такого інструменту є актуальним завданням, що поєднує теоретичне дослідження методів забезпечення відмовостійкості та практичну реалізацію програмного продукту.

		Горшков С.В.			ІПЗ.КР.Б-121-26-ІЗ	Арк.
		Талавер О.В.				9
Змн.	Арк.	№ докум.	Підпис	Дата		

Метою роботи є розробка алгоритмів та реалізація програмного комплексу веборієнтованої системи моніторингу відмовостійкості HTTP-сервісів із вбудованим механізмом Chaos Engineering-експериментів.

Завдання роботи. Для досягнення поставленої мети необхідно вирішити такі завдання:

- провести аналіз існуючих підходів до забезпечення відмовостійкості програмних систем та методів Chaos Engineering;
- виконати огляд та порівняльний аналіз наявних аналогів програмного продукту;
- визначити архітектуру системи та обґрунтувати вибір інструментальних засобів;
- спроектувати структуру бази даних та алгоритми роботи основних модулів системи;
- реалізувати програмний комплекс, що включає серверну частину на FastAPI, HTTP хаос-проксі, багатосторінковий вебінтерфейс (Jinja2 та Bootstrap) та REST API (інтеграція через Swagger /docs);
- провести функціональне тестування розробленої системи та перевірити коректність реалізованих алгоритмів.

Об'єктом дослідження є процес тестування відмовостійкості HTTP-сервісів через хаос-проксі в умовах мікросервісної архітектури.

Предметом дослідження є методи та алгоритми автоматизованого введення збоїв у програмні системи та аналізу їх стійкості до відмов.

Методи дослідження у роботі використано методи системного аналізу при дослідженні предметної області, методи об'єктно-орієнтованого проектування при розробці архітектури системи, а також методи статистичного аналізу при розробці аналітичного модуля оцінки впливу хаос-експериментів на показники якості сервісів. Вимоги до якості програмного забезпечення узгоджено з моделлю ISO/IEC 25010 [33].

Практичне значення отриманих результатів є розроблений програмний комплекс може бути використаний у практичній діяльності команд розробки та

		Горшков С.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Талавер О.В.				10
Змн.	Арк.	№ докум.	Підпис	Дата		

DevOps-інженерів для систематичного тестування стійкості HTTP-сервісів без необхідності використання складної хмарної інфраструктури. Система дозволяє автоматизувати процес збору метрик доступності, наочно відображати вплив введених збоїв та приймати обґрунтовані рішення щодо поліпшення архітектури продукту.

		Горшков С.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Талавер О.В.				11
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ВИБІР ІНСТРУМЕНТІВ

1.1. Постановка задачі

Сучасна розробка програмного забезпечення характеризується переходом від монолітних систем до розподілених мікросервісних архітектур. Такий підхід забезпечує кращу масштабованість, незалежне розгортання компонентів та технологічну різноманітність, але водночас суттєво ускладнює забезпечення надійності системи в цілому. За даними дослідження CNCF Annual Survey 2023, понад 84% організацій використовують мікросервісну архітектуру або мають плани щодо її впровадження [2]. Архітектурні принципи мікросервісних систем детально розглянуто у Fowler та Richardson [11, 23].

Ключовою проблемою при експлуатації мікросервісних систем є непередбачувана поведінка в умовах часткових відмов. Відповідно до принципу CAP-теореми, розподілена система не може одночасно гарантувати консистентність, доступність та стійкість до розподілення. На практиці це означає, що певні класи збоїв - мережеві затримки, помилки залежних сервісів, вичерпання ресурсів - неминучі, і система повинна вміти коректно поводитися в таких умовах [3]. Патерни стійкості до відмов у промисловій експлуатації описано у Nygard [10].

Chaos Engineering є дисципліною, що дозволяє систематично виявляти слабкі місця в поведінці системи шляхом контрольованого введення збоїв. Основні принципи Chaos Engineering, сформульовані командою Netflix, включають формування гіпотези про стабільний стан системи і введення реалістичних збоїв, які можуть відбутися в продуктивному середовищі. Проведення експериментів у виробничому або максимально наближеному до нього середовищі та їх автоматизацію.

Предметна область задачі, що вирішується у цій роботі, охоплює кілька взаємопов'язаних напрямків:

По-перше, проксі-моніторинг HTTP-трафіку. Клієнт-застосунок надсилає запити на порт входу зареєстрованого сервісу, хаос-проксі пересилає їх на URL

		Горшков С.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Талавер О.В.				12
Змн.	Арк.	№ докум.	Підпис	Дата		

виходу (бекенд), вимірює затримку і фіксує HTTP-статус. Метрики накопичуються з реального потоку запитів, що точніше відображає поведінку застосунку, ніж ізольовані health-check. Без активних хаос-експериментів система працює в режимі спостереження (прозоре пересилання з записом метрик).

По-друге, керування хаос-експериментами. Панель керування дає можливість окремо для кожного сервісу керувати запуском та типом хаосу, як по сценарію так і за допомогою ручного налаштування. Кожен сервіс має окремий порт входу та виходу і незалежний набір експериментів.

По-третє, збір і аналіз метрик. Вивід даних на графік з налаштуваннями за який період та частоту оновлення змін з можливістю порівняти з застосуванням хаосу та без застосування. А також можливістю експортувати в CSV.

Функціонал майбутнього застосунку базується на структурованій моделі предметної області хаос інженерії, де центральне місце займають сутності HTTP-сервіс, хаос-експеримент та метрика спостереження. Система має підтримувати повний життєвий цикл моніторингових сервісів таких як реєстрацію з заданням порту входу та URL виходу, перегляд їхнього стану, активацію й деактивацію, а також видалення - що дозволяє гнучко керувати набором цілей для тестування відмовостійкості.

Для моделювання збоїв передбачається запуск і зупинка хаос-експериментів різних типів, зокрема готових сценаріїв із налаштованими параметрами конфігурації. Паралельно з цим система має забезпечувати режим спостереження, пересилання HTTP-запитів через проксі з фіксацією метрик без впровадження хаосу, що дає змогу отримати базову лінію продуктивності перед експериментами.

Доступ до функцій реалізується через авторизацію користувачів на основі JWT із розмежуванням ролей admin і user. Для контролю та аналізу передбачено журнал активності, перегляд активних експериментів і історії їхнього виконання, візуалізацію метрик у вигляді графіків, порівняльний аналіз впливу хаосу, контроль SLO та формування тижневого звіту. Результати спостережень мають

		Горшков С.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Талавер О.В.				13
Змн.	Арк.	№ докум.	Підпис	Дата		

бути доступні для експорту метрик і історії хаос-експериментів у форматі CSV. Взаємодія з системою здійснюється через REST API та вебінтерфейс, що забезпечує покриття як програмного інтегрування, так і роботи оператора безпосередньо в браузері.

Застосунок має відповідати вимогам продуктивності та зручності розгортання в навчальному й дослідницькому середовищі. Система має підтримувати одночасний моніторинг кількох сервісів без суттєвої деградації продуктивності проксі та бекенду.

Застосунок має бути крос-платформним і працювати на Windows, Linux та macOS, а також розгортатися без зовнішніх залежностей у вигляді окремих серверів баз даних, що спрощує локальне встановлення та демонстрацію результатів роботи.

1.2. Аналіз аналогів програмного продукту

Проектування ефективного вебзастосунку для моніторингу відмовостійкості сервісів за принципами Chaos Engineering потребує детального дослідження існуючих програмних рішень. На сьогодні ринок інструментів у цій галузі представлений широким спектром продуктів: від інфраструктурних платформ для симуляції збоїв у хмарних та контейнерних середовищах до сервісів зовнішнього моніторингу доступності. Узагальнений огляд сучасних підходів до хаос-інженерії наведено в [20].

Для проведення всебічного оцінювання функціональних можливостей існуючих систем у контексті перевірки відмовостійкості HTTP-сервісів було сформовано перелік ключових технічних і функціональних критеріїв, які дозволять об'єктивно порівняти наявні аналоги та визначити їхню відповідність вимогам майбутнього застосунку.

Цей аналіз фокусується на можливості активного введення збоїв, а не лише пасивного спостереження за доступністю, а також на наявності HTTP-моніторингу - збору метрик часу та успішності запитів на основі реального трафіку. Важливими елементами порівняння є можливість роботи без Kubernetes і складного DevOps-стеку, наявність вбудованого вебінтерфейсу та REST API

		Горишков С.В.			ІПЗ.КР.Б-121-26-ІПЗ	Арк.
		Талавер О.В.				14
Змн.	Арк.	№ докум.	Підпис	Дата		

для керування експериментами, а також аналітика впливу хаосу - порівняння показників у режимах з активним хаосом та без хаосу. Окремо враховуються відкритість вихідного коду, легкість розгортання та необхідність інтеграції з зовнішніми системами спостереження.

Першопрохідець у галузі хаос-інженерії це Chaos Monkey (Netflix). Первинно розроблений командою Netflix у 2010 році, інструмент автоматично та випадковим чином завершує роботу екземплярів сервісів у продуктивному середовищі AWS. Це змушує команди розробляти системи, стійкі до відмов окремих вузлів. Пізніше рішення було розширено до набору Simian Army, що включає інструменти для симуляції збоїв у різних аспектах інфраструктури. Еволюцію від ранніх експериментів Netflix до сучасних підходів хаос-інженерії описано в [27].

Сильними сторонами Chaos Monkey є великий досвід використання в масштабах Netflix, глибока інтеграція з AWS та широка спільнота практиків. Інструмент довів ефективність підходу випадкових збоїв у продакшен для підвищення відмовостійкості розподілених систем.

Проте архітектура рішення має суттєві обмеження у контексті сучасних вимог до моніторингу HTTP-сервісів. Chaos Monkey орієнтований на інфраструктурний рівень - завершення роботи віртуальних машин або контейнерів, а не на моделювання мережових збоїв на шляху HTTP-запиту. Інструмент тісно прив'язаний до AWS та Spinnaker, вимагає складного початкового налаштування, практично не має власного інтерфейсу для керування експериментами та аналітики результатів. Відсутність вбудованого HTTP-моніторингу та порівняльної аналітики впливу хаосу змушує команди використовувати окремі системи спостереження [4].

Gremlin комерційна SaaS-платформа для Chaos Engineering корпоративного рівня. Підтримує широкий спектр атак, надає зручний вебінтерфейс та API для керування експериментами. Має функціонал сценаріїв, де атаки можна комбінувати та виконувати в певній послідовності.

Перевагами Gremlin є найповніший серед розглянутих аналогів набір типів атак, зручний інтерфейс оператора, підтримка Kubernetes та інших контейнерних

		Горшков С.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Талавер О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		15

оркестраторів, а також вбудовані механізми безпеки. Платформа добре підходить для корпоративних команд із розвинутою інфраструктурою та бюджетом на ліцензування.

Суттєвими недоліками є висока вартість корпоративних ліцензій та відсутність безкоштовного тарифного плану для невеликих команд і навчальних проєктів. Gremlin функціонує як хмарний сервіс, що створює залежність від зовнішнього постачальника. Хоча платформа підтримує атаки на рівні мережі, вона не реалізує модель HTTP-проксі з окремими точками входу та виходу для довільного трафіку. Вбудованого моніторингу доступності та аналітики кореляції немає - для цього потрібна інтеграція з Datadog, Grafana або аналогічними системами [5].

LitmusChaos - проєкт з відкритим вихідним кодом для Chaos Engineering у Kubernetes-середовищах, що є частиною CNCF. Підтримує понад 50 типів хаос-сценаріїв для контейнерів, вузлів та мережі. Має власний оператор Kubernetes та CRD для декларативного опису експериментів.

Основними перевагами LitmusChaos є відкрите програмне забезпечення, що забезпечує прозорість реалізації, глибока інтеграція з Kubernetes, широкий набір готових сценаріїв та підтримка GitOps-підходу до керування експериментами. Для команд, що вже працюють у контейнерній екосистемі, LitmusChaos є зрілим і активно розвивається.

Архітектура LitmusChaos демонструє повну залежність від Kubernetes: без кластера інструмент не може бути розгорнутий і використаний. Початкове налаштування оператора, CRD та ChaosHub вимагає значних DevOps-компетенцій. LitmusChaos не забезпечує вбудованого HTTP-моніторингу на рівні проксі між клієнтом і бекендом, а також метрики доступності та порівняння впливу хаосу потребують окремого стеку. Таким чином, рішення не підходить для середовищ без контейнерної оркестрації та для сценаріїв, де потрібно тестувати відмовостійкість взаємодії між клієнтом і сервісом поза межами кластера [6].

Uptime Robot, Freshping, StatusCake - представники класу інструментів зовнішнього моніторингу доступності. Вони виконують HTTP-перевірки із

		Горшков С.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Талавер О.В.				16
Змн.	Арк.	№ докум.	Підпис	Дата		

заданим інтервалом, відстежують доступність та час відгуку, надсилають сповіщення при збоях.

Перевагами цих сервісів є простота налаштування, зрозумілі дашборди, глобальні точки перевірки та безкоштовні базові тарифи. Вони ефективно вирішують задачу пасивного контролю доступності, оператор дізнається про збій після його виникнення.

Проте uptime-сервіси не реалізують Chaos Engineering: вони не вводять збої активно, а лише фіксують факт недоступності або перевищення порогу часу відповіді. Моніторинг здійснюється через періодичні зовнішні пінг-запити, а не через перехоплення реального продуктивного HTTP-трафіку між клієнтом і сервісом. Відсутня аналітика кореляції між штучно введеними збоями та показниками якості, немає механізму порівняння метрик. Для задачі навчального та дослідницького моделювання відмовостійкості такі інструменти покривають лише частину потреб - спостереження, але не хаос-тестування [7].

Окрім зазначених критеріїв, до порівняння включено ознаки відкритості вихідного коду, можливості роботи без Kubernetes та легкості розгортання. Це дозволяє наочніше показати, що кожен із розглянутих сервісів сфокусований на власній вузькій ніші: інфраструктурний хаос, корпоративна SaaS-платформа, Kubernetes-native CE або зовнішній uptime-моніторинг. Зведені результати дослідження функціональних можливостей обраних аналогів подано в порівняльній таблиці 1.1.

Таблиця 1.1

Порівняльний аналіз аналогів програмного продукту

Критерій	Chaos Monkey	Gremlin	LitmusChaos	Uptime Robot	Розроблювана система
HTTP-моніторинг	Ні	Ні	Ні	Так	Так
Chaos Engineering	Так	Так	Так	Ні	Так
Відкритий код	Так	Ні	Так	Ні	Так
Без Kubernetes	Ні	Часткова	Ні	Так	Так

продовж. табл. 1.1

Аналітика впливу	Ні	Часткова	Ні	Ні	Так
Легкість деплою	Низька	Висока	Низька	Висока	Висока
REST API	Обмежений	Так	Так	Так	Так

Проведений аналіз існуючих програмних рішень дозволяє зробити висновок, що сучасний ринок інструментів для хаос-інженерії та моніторингу відмовостійкості є достатньо розвиненим, проте суттєво фрагментованим. Більшість розглянутих систем фокусуються на вирішенні вузькоспеціалізованих завдань. Chaos Monkey забезпечує інфраструктурні збої в AWS, але не працює з HTTP-проксі та не надає аналітики. Gremlin пропонує найширший набір атак і зручний UI, але потребує хмарної підписки та зовнішнього моніторингу. LitmusChaos є потужним open-source рішенням для Kubernetes, однак недоступний без оркестратора. Uptime-сервіси дають простий моніторинг доступності, але не підтримують активне введення збоїв.

Спільною архітектурною та функціональною проблемою проаналізованих аналогів є відсутність консолідованої екосистеми, яка б органічно поєднувала хаос-ін'єкцію на шляху реального HTTP-трафіку, збір метрик доступності з цього трафіку та порівняльну аналітику впливу хаосу. Для повноцінного тестування відмовостійкості між клієнтом і сервісом взаємодії користувачі змушені комбінувати кілька розрізнених інструментів: хаос-платформу для інфраструктури, окремий uptime-моніторинг, систему візуалізації та власні скрипти для кореляції результатів. Це підвищує поріг входу, ускладнює локальне розгортання та робить процес менш придатним для навчальних і дослідницьких задач.

З огляду на виявлені недоліки існуючих аналогів, розробка власного спеціалізованого вебзастосунку є актуальним та обґрунтованим завданням, яке дозволить об'єднати розрізнений інструментарій в єдину систему.

Головною перевагою проєкту стане HTTP-проксі з окремими точками входу та виходу, що дає змогу моделювати мережеві збої без зміни коду сервісу і без Kubernetes. Застосунок забезпечить вбудовану панель керування для

		Горшков С.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Талавер О.В.				18
Змн.	Арк.	№ докум.	Підпис	Дата		

перегляду журналу запитів, метрик часу та активних експериментів, REST API для програмного керування, рольовий доступ та аналітику впливу хаосу - автоматичне порівняння показників у режимах з активним хаосом і без нього. Легке локальне розгортання робить систему доступною для команд, що тестують відмовостійкість без складного DevOps-стеку.

Об'єднання цих функціональних модулів у межах єдиного інтерфейсу дозволить мінімізувати адміністративне навантаження на оператора та забезпечить цілісний цикл без залучення сторонніх SaaS-сервісів.

1.3. Архітектура системи моніторингу відмовостійкості

При проєктуванні архітектури системи розглядалося кілька підходів до організації програмного комплексу. Для масштабу поставленої задачі доцільним є монолітне розгортання з модульним розмежуванням відповідальностей: центральна частина забезпечує керування та спостереження, а СР для кожного підключеного сервісу працює як окремий логічний вузол перехоплення трафіку без необхідності розподіленої інфраструктури та окремих кластерів. Система орієнтована на локальне використання: один оператор, обмежена кількість моніторингових сервісів, один файл SQLite. У таких умовах розподілення на незалежні мікросервіси не дає суттєвої вигоди в продуктивності, але значно ускладнює розгортання. Метою роботи є низький поріг входження. Моноліт дозволяє запустити всю панель керування однією командою, без Docker Compose і зовнішніх залежностей, що відповідає заявленій практичній цінності для малих команд. Підхід до локального розгортання та супроводу без окремих серверів узгоджується з практиками адміністрування хмарних і мережевих систем, де наголошують на простоті конфігурації та передбачуваності поведінки сервісу [21].

Система побудована за клієнт–серверною схемою з розділенням ролей. Панель керування призначена для оператора: реєстрація сервісів, запуск і зупинка хаос-експериментів, перегляд журналу, звітів і налаштувань доступу. Клієнт-застосунок (зовнішній споживач API) взаємодіє не з бекендом напряду, а з точкою входу СР відповідного сервісу. Таким чином усі робочі HTTP-запити

		Горшков С.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Талавер О.В.				19
Змн.	Арк.	№ докум.	Підпис	Дата		

проходять через єдиний контрольований канал, де можливі як прозоре перенаправлення, так і штучне відтворення мережових збоїв. Операторський доступ до панелі керування відокремлений від продуктивного трафіку клієнтів і захищений авторизацією. Канал СР для клієнт-запитів залишається відкритим для передачі HTTP без додаткової автентифікації на рівні проксі.

Багаторівнева організація серверної частини передбачає три логічні шари. Рівень представлення забезпечує відображення інформації оператору та прийом керуючих дій через вебінтерфейс і програмний інтерфейс обміну даними. Рівень бізнес-логіки реалізує перенаправлення HTTP-потоків, застосування хаос-ефектів, обробку результатів запитів, формування сповіщень і правила доступу. Рівень даних відповідає за збереження конфігурації сервісів, параметрів експериментів, журналу спостережень, інцидентів і облікових записів користувачів. Загальна архітектура системи зображена на рисунку 1.1.

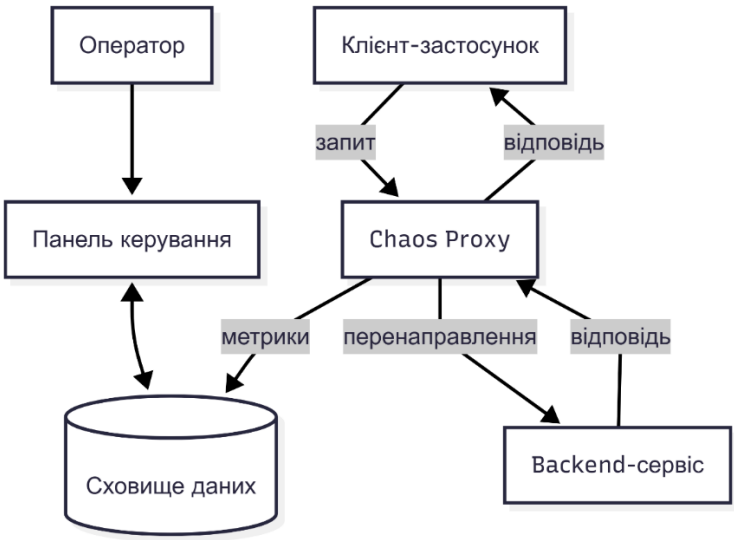


Рис. 1.1. Загальна архітектура системи СР

Програмний комплекс включає ядро панелі керування, яке ініціалізує середовище роботи, надає оператору інтерфейс і програмний інтерфейс для налаштування системи, а при старті активує модулі СР для усіх зареєстрованих сервісів. Модуль СР виконує роль HTTP-проксі між точкою входу (адреса, на яку звертається клієнт) і точкою виходу (адреса цільового бекенд-сервісу). Для кожного вхідного запиту СР завантажує конфігурацію відповідного сервісу, визначає наявність активних хаос-експериментів і або пересилає запит на бекенд

без змін, або попередньо застосовує обрані ефекти нестабільності. Відповідь бекендом повертається клієнту через СР, паралельно фіксується результат спостереження.

Модуль керування точками входу забезпечує, що кожен зареєстрований сервіс має власну точку входу СР і може оброблятися незалежно від інших. Це дозволяє одночасно спостерігати кілька бекенд-сервісів і проводити різні хаос-експерименти без взаємного впливу на рівні перехоплення трафіку.

Модуль хаос-ін'єкції відповідає за імітацію мережевих і сервісних збоїв на рівні НТТР: затримка обробки, часткова або повна втрата з'єднання, скорочення часу очікування відповіді, повна недоступність цілі, випадкові помилки, короткочасне підвищення навантаження, а також комбіновані сценарії, що поєднують кілька ефектів у межах одного експерименту. Якщо активних експериментів немає, модуль не змінює запит - СР працює в режимі спостереження як прозорий проксі.

Модуль збору метрик спостереження після кожного проходження запиту через СР формує запис у журналі: час, тривалість обробки, код відповіді, ознаку успіху чи помилки, а також інформацію про те, чи діяв хаос під час цього запиту. Критерій успіху визначається порівнянням фактичної відповіді бекенду із очікуваним результатом, заданим для сервісу. При помилках ініціюється реєстрація інциденту, накопичені спостереження передаються для аналізу порогових значень.

Модуль сповіщень аналізує останні записи спостереження для кожного сервісу і формує попередження при перевищенні заданих меж: високий рівень помилок, надмірна середня затримка, серія послідовних невдач. Механізм гістерезису обмежує повторення однакових сповіщень за короткий інтервал.

Модуль автентифікації та авторизації забезпечує розмежування доступу до панелі керування та керуючих операцій за ролями (адміністратор і користувач). Шар даних зберігає конфігурацію користувачів, сервісів, експериментів, метрик, інцидентів і журнал сповіщень. При ініціалізації системи формується початкова конфігурація та облікові записи за замовчуванням.

		Горшков С.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Талавер О.В.				21
Змн.	Арк.	№ докум.	Підпис	Дата		

Вебінтерфейс оператора охоплює розділи: активність (журнал запитів), сервіси, експерименти та готові сценарії, історія, аналітичні звіти, керування користувачами. Програмний інтерфейс обміну даними підтримує повний цикл керування сервісами, експериментами, переглядом журналу, агрегованими метриками, аналітикою впливу хаос та експортом результатів спостереження.

Потік даних у режимі спостереження (без активного хаос). Запит від клієнта надсилається на точку входу СР та перенаправляє до точки виходу (бекенду), а відповідь надсилається клієнту. Кожен такий цикл супроводжується записом у сховище даних і відображенням у панелі керування. У режимі хаос інженерії між прийомом запиту на вході СР і перенаправленням на вихід додається етап застосування обраних ефектів нестабільності. Метрики формуються лише від реального HTTP-трафіку, що проходить через СР. Запити, які обходять точку входу, у журнал системи не потрапляють.

1.4. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення

Для серверної частини обрано Python 3.10. Мова підтримує асинхронне програмування, має готові бібліотеки для HTTP-серверів і роботи з базами даних і добре підходить для прототипування вебзастосунків із невеликою командою розробки. Остання стабільна версія інтерпретатора демонструє помітне прискорення виконання коду завдяки оптимізації спеціалізованого компілятора CPython та впровадженню нових інструкцій віртуальної машини, що критично важливо для високонавантажених мережових застосунків і хаос-проксі, які обробляють потоки запитів у реальному часі. Розвинута екосистема Python надає готові, добре протестовані рішення для асинхронної обробки подій через модуль `asyncio`, високопродуктивні HTTP-клієнти типу `httpx` та швидкі ASGI-сервери `uvicorn`, що формують цілісний стек для неблокуючої архітектури без необхідності залучення зовнішніх черг повідомлень або складних оркестраторів. Наявність потужних ORM-бібліотек, зокрема `SQLAlchemy`, спрощує взаємодію з реляційними сховищами даних, абстрагуючи розробника від написання сирого SQL-коду та мінімізуючи ризик ін'єкцій. Крім того, мова користується

		Горшков С.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Талавер О.В.				22
Змн.	Арк.	№ докум.	Підпис	Дата		

підтримкою найпоширеніших сучасних вебфреймворків, таких як FastAPI, Flask та Django, що забезпечує гнучкість у виборі архітектурних патернів залежно від масштабу проєкту. Активна глобальна спільнота розробників та обширна, структурована документація значно знижують поріг входження, прискорюють діагностику помилок та полегшують інтеграцію сторонніх бібліотек, що є вирішальним фактором для академічних та комерційних проєктів з обмеженими ресурсами розробки.

Як вебфреймворк обрано FastAPI [12]. Він працює з `async/await` і добре поєднується з `asuncіo`, тому обробники запитів панелі керування та проксі не блокують один одного. FastAPI автоматично формує документацію API (Swagger UI на `/docs`), що зручно для перевірки ендпоінтів під час розробки та тестування. Вхідні дані валідуються через Pydantic [14]: помилки в параметрах запиту відсікаються на рівні API, без додаткових перевірок у кожному маршруті. За результатами незалежних бенчмарків TechEmpower, FastAPI демонструє продуктивність, порівнянну з рішеннями на Node.js та Go, що підтверджує його придатність для високонавантажених мікросервісних архітектур [8]. Документація FastAPI підтверджує підтримку OpenAPI та інтеграцію з Pydantic [12, 14]. Така комбінація швидкодії, автоматизації рутинних завдань документації та відповідності сучасним стандартам контракування API робить FastAPI ідеальним вибором для побудови керуючого інтерфейсу системи моніторингу.

Для роботи з шаром даних використано бібліотеку SQLAlchemy версії 2.0, яка є широко використовуваною для об'єктно-реляційного відображення в екосистемі Python та підтримує як синхронний, так і асинхронний режими виконання запитів. Вона надає декларативний синтаксис для визначення моделей даних, прозору генерацію SQL-запитів через власний компілятор, підтримку складних транзакційних сценаріїв із рівнями ізоляції та уніфікований інтерфейс для підключення до широкого спектру систем управління базами даних. У контексті даного застосунку як основне сховище обрано SQLite, що є вбудованою реляційною базою даних, яка не вимагає окремого серверного

		Горшков С.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Талавер О.В.				23
Змн.	Арк.	№ докум.	Підпис	Дата		

процесу, складного налаштування мережових з'єднань або постійного адміністративного супроводу. Усі дані системи зберігаються в єдиному крос-платформному файлі, що кардинально спрощує процес розгортання, резервного копіювання та міграції між середовищами розробки, тестування та продакшену. Попри репутацію "легкої" бази даних, SQLite успішно задовольняє обсяги даних та навантаження, характерні для цільового випадку використання системи моніторингу відмовостійкості, де пріоритетом є швидкість ініціалізації та мінімальні інфраструктурні витрати, а не масове паралельне читання/запис із сотень незалежних клієнтів. У реалізації системи використовується синхронний ORM-шар SQLAlchemy, що достатньо для обсягу записів метрик у SQLite при проксі-моделі. ORM SQLAlchemy [13] і вбудована SQLite [25] не потребують окремого серверного процесу БД.

Бібліотека Pydantic версії 2.5 інтегрована в архітектуру системи як основний інструмент валідації вхідних даних API та серіалізації відповідей, що дозволяє перенести перевірку типів, форматів та бізнес-правил на рівень інтерфейсу програмування. Її використання усуває необхідність написання великої кількості ручних умовних операторів у бізнес-логіці, знижує ризик виникнення помилок виконання та забезпечує чіткий контракт між клієнтом та сервером. Pydantic 2.5, побудована на компільованому бекенді pydantic-core мовою Rust, демонструє значне прискорення процесу валідації порівняно з попередніми версіями, що критично важливо для API, який обробляє потоки конфігураційних запитів та метрик у реальному часі. Автоматична генерація схем JSON на основі анотацій типів Python не лише забезпечує сумісність із фронтенд-додатками, а й безпосередньо інтегрується з OpenAPI-документацією FastAPI.

Авторизацію панель керування і REST API реалізовано через JWT (бібліотека python-jose, алгоритм HS256) з ролями admin/user. Формат токена відповідає RFC 7519 [19].

Для виконання HTTP-запитів до бекенду (URL виходу) у модулі хаос-проксі обрано бібліотеку httpx версії 0.25, яка надає асинхронний інтерфейс

		Горшков С.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Талавер О.В.				24
Змн.	Арк.	№ докум.	Підпис	Дата		

async/await і сумісна з циклом подій asyncio FastAPI/uvicorn[15]. Це дозволяє неблокуюче пересилання вхідного трафіку клієнта на цільовий сервіс під час застосування хаос-ефектів без окремих потоків на кожен запит. httpx підтримує налаштування таймаутів, роботу з HTTP/1.1 та опціонально HTTP/2, що важливо при нестабільній мережі або імітації відмов. У поєднанні з asyncio це забезпечує ефективну обробку реального трафіку через проксі, а не періодичного фонового опитування URL.

Клієнтська частина реалізована як багатосторінковий вебінтерфейс на серверних шаблонах Jinja2 (окремий HTML-маршрут на розділ). Дані підвантажуються через Fetch API. Для стилізації використано CSS-фреймворк Bootstrap 5.3, обраний як найпоширеніший інструмент для адаптивних вебінтерфейсів з мінімальними витратами на кастомізацію. Він надає багатий набір готових UI-компонентів, включаючи гнучкі сітки на основі Flexbox та CSS Grid, картки, модальні вікна, форми валідації та системи сповіщень, що значно прискорює розробку фронтенду та забезпечує єдиний візуальний стиль. Завдяки вбудованій системі медіа-запитів інтерфейс автоматично адаптується до різних розмірів екранів, забезпечуючи коректне відображення та зручну навігацію як на десктопних моніторах, так і на мобільних пристроях. Для візуалізації зібраних метрик інтегровано бібліотеку Plotly.js, яка спеціалізується на побудові інтерактивних наукових та аналітичних графіків без необхідності серверного рендерингу. Вона підтримує широкий спектр типів діаграм, але в контексті системи моніторингу найбільш затребуваною є функція відображення часових рядів із можливістю динамічного зуму, панорамування, фільтрації діапазонів даних та експорту зображень у векторних та растрових форматах. Інтерактивність графіків дозволяє операторам детально досліджувати кореляції між моментами запуску хаос-експериментів та змінами в показниках затримки або доступності сервісів, для наочного порівняння періодів з хаосом і без. Таким чином, клієнтська частина поєднує серверні шаблони Jinja2 [18], адаптивну верстку Bootstrap 5 [17] та інтерактивну візуалізацію метрик засобами Plotly.js [16].

		Горшков С.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Талавер О.В.				25
Змн.	Арк.	№ докум.	Підпис	Дата		

Вимоги до апаратного та програмного забезпечення

Характеристика	Мінімальні вимоги	Рекомендовані вимоги
ОС	Windows 10 / Ubuntu 20.04 / macOS 11	Windows 11 / Ubuntu 22.04 / macOS 13
Процесор	x86-64, 1 ГГц, 1 ядро	x86-64, 2 ГГц, 2+ ядра
Оперативна пам'ять	512 МБ	2 ГБ
Дисковий простір	200 МБ	1 ГБ (для зберігання метрик)
Python	3.10	3.11+
Браузер	Chrome 100+ / Firefox 100+	Chrome 120+ / Firefox 120+
Мережа	Доступ до Інтернет або внутрішньої мережі	Стабільне мережеве з'єднання

Висновки до розділу 1

У першому розділі виконано теоретичне дослідження предметної галузі, сформульовано задачу та вимоги до системи, проведено аналіз аналогів, обґрунтовано архітектуру та обрано засоби реалізації.

В першому підрозділі розглянуто Хаос інженерію і роль HTTP-проксі в спостереженні за потоком запитів між клієнтом і бекендом.

В другому підрозділі сформульовано постановку задачі, функціональні й нефункціональні вимоги та порівняно аналоги (Chaos Monkey, Gremlin, LitmusChaos, uptime-сервіси). Встановлено, що вони не поєднують хаос на шляху HTTP-трафіку, збір метрик з нього та аналітику впливу хаос у легковаговому рішенні без Kubernetes.

В третьому підрозділі обґрунтовано клієнт–серверну монолітну архітектуру з СР, точками входу/виходу, панеллю керування та двома режимами: спостереження і хаос інженерія.

В четвертому підрозділі визначено стек реалізації (Python, FastAPI, SQLAlchemy, SQLite, httpx, Jinja2, Bootstrap, Plotly.js, JWT) і спосіб керування — вебінтерфейс та REST API.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ СИСТЕМИ

2.1. Варіанти використання та структура системи

Перед проєктуванням структури системи розглянемо трьох акторів. Адміністратор (роль admin) – це користувач, який має повні права конфігурації: реєстрація, редагування, активація/деактивація та видалення сервісів, керування обліковими записами, запуск і зупинка хаос-експериментів, перегляд активності, звітів, інцидентів, експорт даних. Доступ - через вебінтерфейс панелі керування та REST API після авторизації.

Користувач (роль user) - оператор хаос-тестування, який має обмежені права по відношенню до адміністратора: перегляд зареєстрованих сервісів (вхід/вихід, статус). Має змогу запускати і зупиняти хаос-експерименти та готові сценарії, переглядати журнал активності, звіти, експортувати дані. Проте не може створювати, редагувати або видаляти сервіси та керувати іншими користувачами. Розмежування прав доступу операторів відповідає типовій моделі role-based access control у веборієнтованих системах керування [24].

Клієнт-застосунок - зовнішня програма (браузер, Postman, демо-клієнт, власний код), яка надсилає HTTP-запити на порт входу СР. Саме ці запити проходять шлях від клієнта на порт входу спрямовуються на СР та перенаправляються на вихід де є частина бекенду і формують записи в таблиці метрик.

Записи метрик в базі даних створюються тільки якщо через проксі проходить реальний HTTP-трафік та виводяться на панелі керування. Дані які проходять повз проксі не відображаються та не фіксуються.

Для акторів Адміністратор і Користувач деталізовано тринадцять варіантів використання, що покривають повний цикл роботи системи. Актор Клієнт-застосунок реалізує окремий сценарій генерації метрик:

		Горшков С.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Талавер О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		27

UC-01: Вхід та вихід - користувач автентифікується на сторінці логін. Система перевіряє облікові дані, видає JWT і встановлює cookie сесії. Вихід очищає cookie. Без успішної авторизації доступ до HTML-сторінок панелі керування і захищених API-ендпоінтів заблоковано.

UC-02: Реєстрація сервісу (лише роль Admin) - адміністратор вказує назву, URL виходу та порт входу (або система призначає його автоматично). Система нормалізує адресу виходу, перевіряє унікальність порту, зберігає запис у БД і запускає окремий екземпляр CP (uvicorn у daemon-поточі) на обраному порту входу.

UC-03: Перегляд списку сервісів (Admin, User) - користувач отримує таблицю сервісів із URL входу, URL виходу, ознакою чи активовано та метаданими. Актуальність даних визначається останніми записами в журналі активності.

UC-04: Редагування сервісу (лише Admin) - зміна назви, опису, URL виходу, порту входу, успішний код відповіді, таймауту. При зміні порту входу система перереєструє проксі-слухач.

UC-05: Активування / деактивування сервісу (лише Admin) - перемикання стану активності без видалення конфігурації. Для деактивованого сервісу хаос-проксі повертає HTTP 503, метрики нових запитів не накопичуються.

UC-06: Видалення сервісу (лише роль Admin) - повне вилучення сервісу з каскадним очищенням пов'язаних метрик, хаос-експериментів, інцидентів, зупинка проксі-порту.

UC-07: Запуск хаос-експеримент (Admin, User) - вибір типу збою (latency, packet_loss, connection_loss, timeout, outage, error_rate, high_request_rate, composite) або готового сценарію, валідація параметрів та активація запису ChaosExperiment. Ефекти застосовуються до наступних запитів, що проходять через порт входу цього сервісу. За наявності duration_minutes експеримент автоматично завершується у фоновому потоці.

		Горшков С.В.			ІПЗ.КР.Б-121-26-ІЗ	Арк.
		Талавер О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		28

UC-08: Зупинити хаос-експеримент (Admin, User) - дострокове завершення: `active = false`, фіксація `ended_at`, формування `results_summary` для порівняльного аналізу.

UC-09: Переглянути активність (Admin, User) - журнал HTTP-запитів через проксі: час, сервіс, затримка, код статусу, успіх, ознака `chaos_active`; фільтрація за сервісом (GET /api/activity).

UC-10: Переглянути звіти та аналітику (Admin, User) - сторінка «Звіти»: інтерактивний графік динаміки метрик (Plotly.js), `chaos-impact` (порівняння «з хаос / без хаосу»), SLO-compliance, тижневий звіт.

UC-11: Експортувати метрики та історію хаосу (Admin, User) - вивантаження CSV (метрики сервісу, історія експериментів) для зовнішнього аналізу та архівування.

UC-12: Керувати користувачами (лише Admin) - створення, деактивація, видалення облікових записів, зміна паролів та призначення ролей `admin / user`.

UC-13: Надіслати HTTP через хаос-проксі (Клієнт-застосунок) - клієнт надсилає запит на порт входу. Проксі застосовує активні хаос-ефекти, пересилає запит на URL виходу, повертає відповідь і ініціює запис метрики. Цей сценарій є джерелом даних для UC-09 та UC-10.

На рисунку 2.1 наведено діаграму варіантів використання, яка відображає взаємодію Адміністратора, Користувача та клієнт-застосунку із функціональними можливостями системи в межах системного контексту. Сценарії керування сервісами, хаос-експериментами, перегляду активності, звітів, експорту даних та адміністрування обліковими записами (UC-02–UC-12) пов’язані зі сценарієм авторизації (UC-01) зв’язком `include`, оскільки доступ до панелі керування і REST API можливий лише після успішного входу оператора. Зв’язок `extend` між сценаріями перегляду активності (UC-09) і формування звітів (UC-10) та сценарієм надсилання HTTP через хаос-проксі (UC-13) показує, що журнал запитів і аналітика метрик наповнюються лише за наявності реального

		Горшков С.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Талавер О.В.				29
Змн.	Арк.	№ докум.	Підпис	Дата		

трафіку на портах входу. Побудована модель слугує орієнтиром для проєктування REST API та вебінтерфейсу, формування тестових сценаріїв і розподілу відповідальності між модулями chaos_proxu, port_manager, proxy_metrics, chaos_injector, alerting та auth.

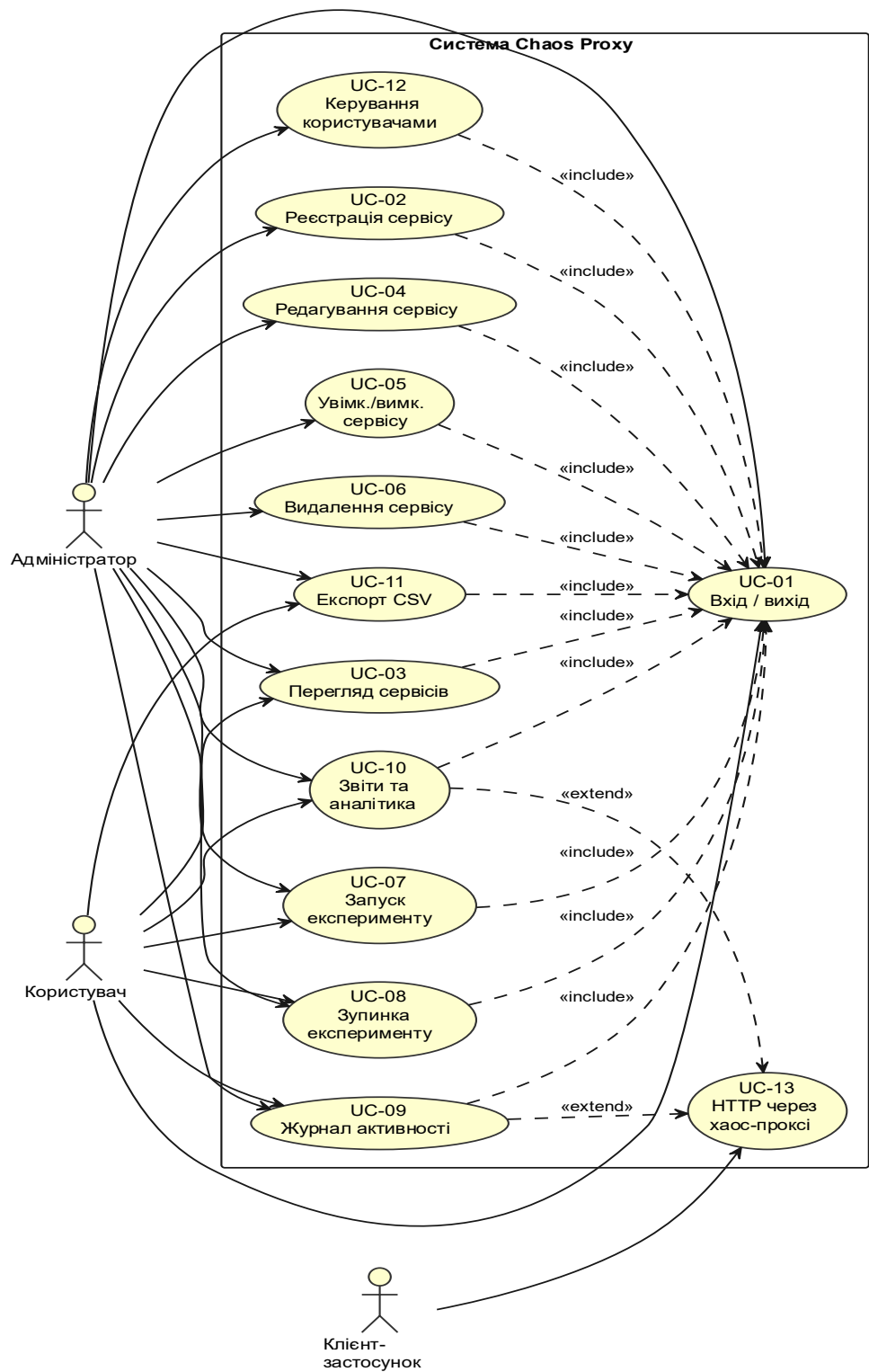


Рис. 2.1. Діаграма варіантів використання системи

Структура класів системи. Для реалізації системи визначено дві групи: моделі даних та класи/модулі бізнес-логіки, що реалізують проксі, хаос, метрики та сповіщення. Діаграму класів наведено на рисунку 2.2.

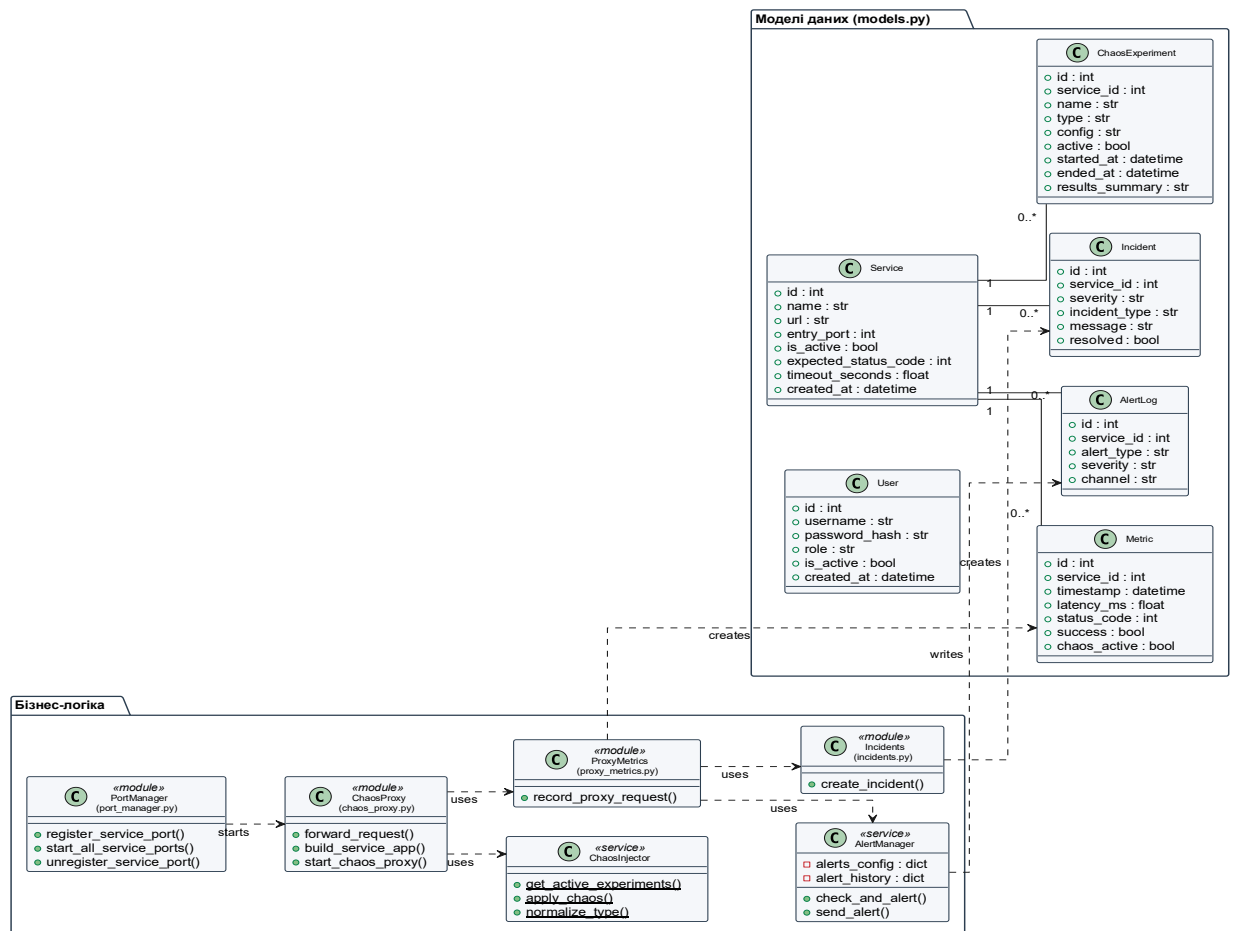


Рис. 2.2. Діаграма класів системи

Клас User є базовою сутністю обліку операторів панелі керування. Він зберігає ідентифікатор, унікальне ім'я користувача (username), криптографічний хеш пароля, роль доступу (admin або user), ознаку активності облікового запису та дату створення. У моделі даних не визначено власних методів - це ORM-сутність SQLAlchemy; операції реєстрації, автентифікації та перевірки прав реалізовані окремими модулями серверної частини. Клас User не пов'язаний безпосередньо з HTTP-трафіком клієнтів, він забезпечує розмежування доступу до керуючих функцій системи (налаштування сервісів, запуск експериментів, перегляд звітів).

Клас Service описує зареєстрований бекенд-сервіс як пару точка входу - точка виходу. Атрибути включають унікальну назву, URL виходу (адреса

цільового бекенду), порт входу CP, опис, ознаку активності, очікуваний HTTP-код успішної відповіді, параметри перевірки (заголовки, метод) та часові мітки. Сутність Service є центральною в предметній області: саме до неї прив'язуються експерименти, метрики спостереження, інциденти та сповіщення. Один Service відповідає одному логічному каналу перехоплення HTTP-трафіку через CP.

Клас ChaosExperiment фіксує параметри хаос-експерименту для конкретного сервісу. Він містить посилання на service_id, назву, тип ефекту, конфігурацію у форматі JSON, ознаку активності, час початку та завершення, а також за потреби підсумок результатів. Експеримент не виконує хаос самостійно - він описує умови, за яких модуль хаос-ін'єкції застосовує ефекти до запитів, що проходять через CP відповідного сервісу.

Клас Metric представляє результат спостереження за одним HTTP-запитом через CP. Запис містить час, тривалість обробки, HTTP-код відповіді, ознаку успіху чи помилки, текст помилки та прапорець chaos_active, який показує, чи діяли активні експерименти під час цього запиту. Metric є основою журналу «Активність» та аналітичних звітів; метрики формуються лише від реального трафіку через точку входу, а не від прямих звернень до бекенду.

Клас Incident описує зафіксовану подію відмови або аномалії, пов'язану з сервісом. Атрибути включають тип інциденту, рівень критичності, повідомлення, код статусу та затримку, а також ознаку вирішення та часові мітки створення й закриття. Інциденти створюються при невдалих запитах через CP або за критичних сповіщень. Механізм дедуплікації запобігає множенню однотипних записів за короткий інтервал.

Клас AlertLog є журналом згенерованих системою сповіщень. Кожен запис пов'язаний із сервісом, містить тип алерту, рівень критичності, текст повідомлення, канал доставки та час створення. AlertLog доповнює метрики та інциденти, фіксуючи моменти перевищення порогових значень (рівень помилок, затримки, серія невдач).

Модуль CP реалізує логіку HTTP-проксі між точкою входу та URL виходу. Він приймає вхідний запит, завантажує конфігурацію сервісу та активні

		Горшков С.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Талавер О.В.				32
Змн.	Арк.	№ докум.	Підпис	Дата		

експерименти, делегує застосування хаос-ефектів модулю хаос-ін'єкції, пересилає запит на бекенд і повертає відповідь клієнту. Окремо формується екземпляр проксі-додатку для кожного сервісу. При старті системи активуються СР для всіх зареєстрованих сервісів. Без активних експериментів модуль працює в режимі прозорого перенаправлення з фіксацією метрик.

Модуль керування точками входу (port_manager) відповідає за призначення унікальної точки входу кожному сервісу та запуск відповідного екземпляра СР у фоновому потоці. Це забезпечує паралельну та незалежну обробку кількох сервісів: кожен має власний канал перехоплення трафіку та власний набір активних експериментів.

Модуль збору метрик (proxy_metrics) виконує постобробку кожного запиту, що пройшов через СР: створює запис Metric, за необхідності ініціює Incident та передає останні спостереження для аналізу порогів у системі сповіщень. Таким чином модуль замикає ланцюг запит - оцінка результату - збереження - можливе попередження оператора.

Клас ChaosInjector інкапсулює логіку імітації мережевих і сервісних збоїв на рівні HTTP. Він визначає активні експерименти для сервісу, розгортає сценарії на набір простих ефектів і застосовує їх до функції пересилання запиту на бекенд. Якщо експериментів немає, запит виконується без модифікацій - реалізується режим моніторингу.

Модуль Incidents забезпечує створення записів Incident з урахуванням дедуплікації: повторні однотипні відмови за короткий період не дублюються в журналі, що підвищує змістовність даних для оператора.

Клас AlertManager реалізує підсистему сповіщень на основі аналізу останніх метрик сервісу. Він порівнює накопичені спостереження з порогоми (рівень помилок, середня затримка, послідовні невдачі), формує попередження, записує їх у AlertLog і виводить у консоль. Для критичних подій може ініціювати Incident. Механізм гістерезису обмежує повторення однакових сповіщень.

Зв'язки між сутностями. Один Service пов'язаний з багатьма ChaosExperiment, Metric, Incident та AlertLog (логічний зв'язок через service_id).

		Горшков С.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Талавер О.В.				33
Змн.	Арк.	№ докум.	Підпис	Дата		

CP використовує ChaosInjector для обробки запитів і модуль збору метрик для фіксації результатів, модуль керування точками входу запускає окремі екземпляри CP для кожного Service. User не має прямого зв'язку з метриками продуктивного трафіку - він керує конфігурацією та експериментами через панель керування. Таке розділення забезпечує цілісність даних, усуває надмірність і дозволяє системі одночасно виконувати функції хаос інженерії та моніторингу HTTP-потoku.

2.2. Структура бази даних

Для збереження даних і забезпечення стабільної роботи розроблюваної системи хаос інженерії та моніторингу HTTP-сервісів обрано вбудовану реляційну СУБД SQLite. Цей вибір обумовлений відсутністю окремого серверного процесу, простотою розгортання (усі дані зберігаються в одному файлі chaos_monitor.db) та достатністю можливостей для локального й навчального використання платформи. Як прошарок доступу до даних застосовано ORM SQLAlchemy у декларативному стилі: моделі Python відображаються на таблиці, а SQL-запити формуються на рівні абстракції, що спрощує підтримку схеми та роботу з сутностями предметної області.

Відповідно до об'єктно-орієнтованої моделі, описаної у попередньому параграфі, дані нормалізовано до логічно цілісної фізичної структури. Центральне місце в схемі посідає таблиця services, яка описує конфігурацію моніторингового сервісу як пари точка входу - точка виходу. До неї логічно прив'язані таблиці chaos_experiments, metrics, incidents та alert_logs через поле service_id (індексоване посилання без оголошених зовнішніх ключів на рівні SQLite). Окремий контур безпеки формує таблиця users, призначена для авторизації операторів панелі керування, вона не пов'язана з іншими таблицями зовнішніми ключами, оскільки облікові записи не беруть участі в обробці продуктивного HTTP-трафіку клієнтів. Загальну фізичну схему бази даних наведено на рис. 2.3.

		Горшков С.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Талавер О.В.				34
Змн.	Арк.	№ докум.	Підпис	Дата		

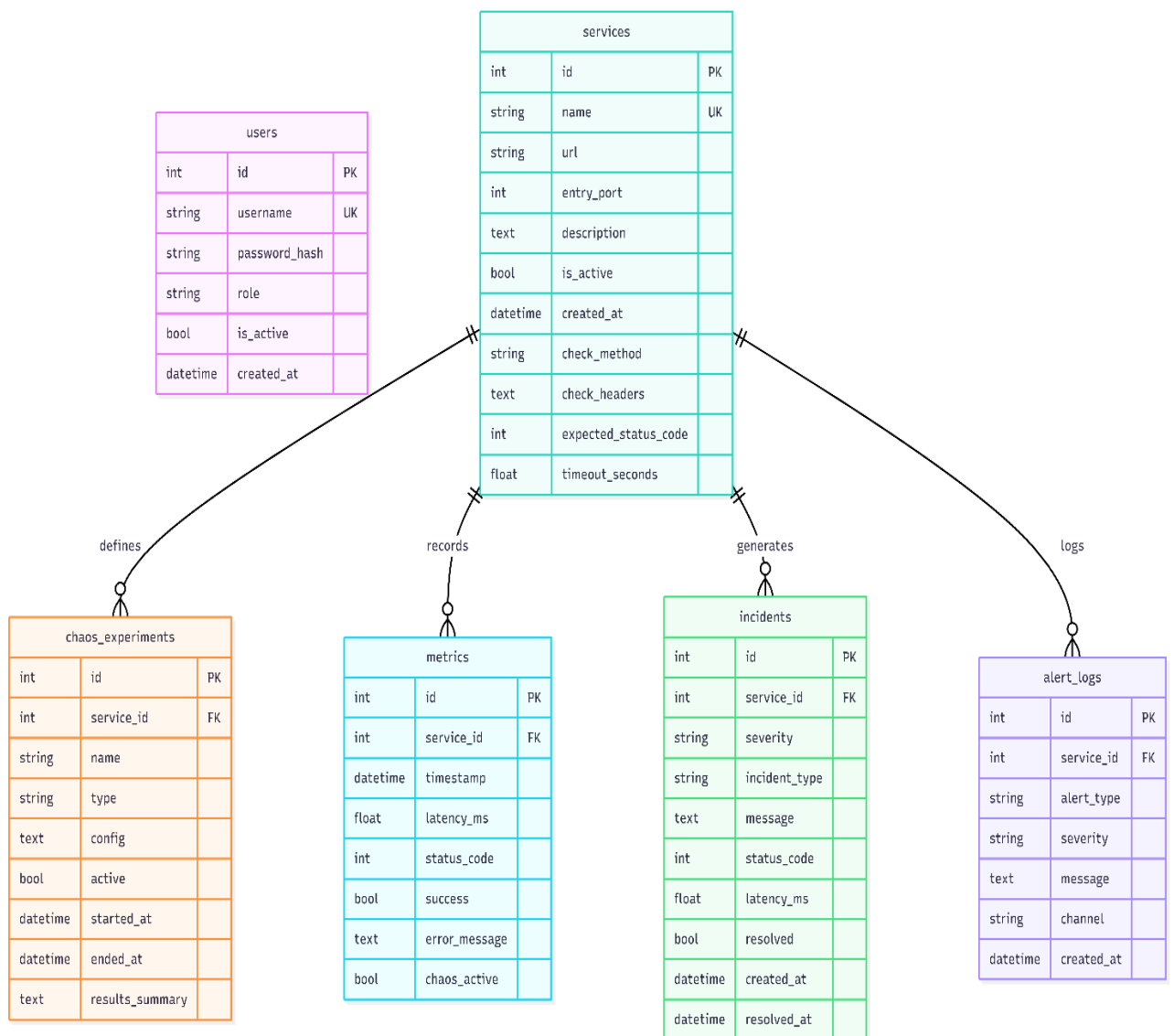


Рис. 2.3. Фізична схема бази даних

Базові параметри зареєстрованих бекенд-сервісів, критерії оцінки успішності відповіді та налаштування точки входу хаос-проксі зберігаються в таблиці **services**. Унікальність назви сервісу (**name**) запобігає дублюванню конфігурацій. Поле **url** задає адресу виходу (бекенд), а **entry_port** - порт входу, на якому приймається трафік через СР. Ознака **is_active** реалізує м'яке вимкнення: деактивований сервіс не пересилає запити на бекенд і повертає відповідь з кодом 503. Поля **expected_status_code**, **check_method**, **check_headers** та **timeout_seconds** фіксують параметри перевірки та очікуваної поведінки сервісу при обробці запитів через проксі. Детальний опис структури наведено в табл. 2.1.

Таблиця 2.1.

Опис структури таблиці services

Поле	Тип даних	Призначення
id	Integer (PK)	Унікальний ідентифікатор сервісу
name	String(255)	Назва сервісу (унікальне поле)
url	String(500)	URL виходу (адреса бекенд)
entry_port	Integer	Порт входу хаос-проксі
description	Text	Текстовий опис сервісу
is_active	Boolean	Ознака активності (м'яке вимкнення)
created_at	DateTime	Дата та час реєстрації
check_method	String(10)	Зарезервоване поле конфігурації
check_headers	Text (JSON)	Заголовки перевірки у форматі JSON
expected_status_code	Integer	Очікуваний HTTP-код успішної відповіді
timeout_seconds	Real	Таймаут перевірки, секунди

Параметри активних та завершених хаос-експериментів для кожного сервісу зберігаються в таблиці chaos_experiments. Поле type визначає вид імітованого збою, а config - JSON-конфігурацію конкретного сценарію. Для активного експерименту поле ended_at залишається порожнім, після зупинки може бути заповнено results_summary - стислий підсумок результатів. Структура таблиці наведена в табл. 2.2.

Таблиця 2.2.

Опис структури таблиці chaos_experiments

Поле	Тип даних	Призначення
id	Integer (PK)	Унікальний ідентифікатор експерименту
service_id	Integer (INDEX)	Посилання на сервіс
name	String(255)	Назва експерименту
type	String(50)	Тип хаос-ефекту
config	Text (JSON)	JSON-конфігурація параметрів

продовж. табл. 2.2

active	Boolean	Ознака активності експерименту
started_at	DateTime	Час початку
ended_at	DateTime	Час завершення
results_summary	Text (JSON)	Підсумок результатів після зупинки

Часові ряди результатів HTTP-запитів, що проходять через хаос-проксі, накопичуються в таблиці metrics. Записи формуються лише від реального трафіку клієнта через точку входу. Індеси на service_id та timestamp оптимізують вибірки для журналу активності та аналізу порогів сповіщень. Поле success відображає відповідність фактичної відповіді очікуваному коду сервісу. chaos_active дозволяє відокремити спостереження під час активного хаосу від «чистого» моніторингу. Структура таблиці наведена в табл. 2.3.

Таблиця 2.3.

Опис структури таблиці metrics

Поле	Тип даних	Призначення
id	Integer (PK)	Унікальний ідентифікатор запису
service_id	Integer (INDEX)	Посилання на сервіс
timestamp	DateTime (INDEX)	Час обробки запиту
latency_ms	Float	Затримка відповіді, мс
status_code	Integer	Фактичний HTTP-код
success	Boolean	Успішність відносно expected_status_code
error_message	Text	Текст помилки (за наявності)
chaos_active	Boolean	Ознака активного хаосу під час запиту

Контур авторизації панелі керування реалізовано таблицею users. Поле password_hash зберігає SHA-256-хеш пароля з попереднім змішуванням із SECRET_KEY із конфігурації застосунку (repper). Роль admin або user визначає рівень доступу до керуючих функцій системи. Таблиця users ізольована від таблиць моніторингового контуру, оскільки оператори не є учасниками продуктивного HTTP-потоків сервісів. Структура наведена в табл. 2.4.

Таблиця 2.4.

Опис структури таблиці users

Поле	Тип даних	Призначення
id	Integer (PK)	Унікальний ідентифікатор облікового запису
username	String(100)	Логін
password_hash	String(128)	SHA-256-хеш пароля
role	String(20)	Системна роль (admin або user)
is_active	Boolean	Ознака активного облікового запису
created_at	DateTime	Дата створення

Зафіксовані відмови та аномалії, пов'язані з конкретним сервісом, зберігаються в таблиці incidents. Запис переважно створюється при невдалому HTTP-запиті через CP (тип proxy_request_failed, текст помилки, код статусу, затримка). Додатково критичне сповіщення може ініціювати інцидент. Механізм дедуплікації не дублює однакові відкриті інциденти протягом п'яти хвилин. Поля resolved та resolved_at дозволяють відстежити життєвий цикл події - від фіксації до закриття, що відповідає загальним практикам керування інцидентами [30]. Структура наведена в табл. 2.5.

Таблиця 2.5.

Опис структури таблиці incidents

Поле	Тип даних	Призначення
id	Integer (PK)	Унікальний ідентифікатор інциденту
service_id	Integer (INDEX)	Посилання на сервіс
severity	String(20)	Рівень критичності
incident_type	String(50)	Тип події (напр., proxy_request_failed)
message	Text	Опис інциденту
status_code	Integer	HTTP-код (за наявності)
latency_ms	Float	Затримка (за наявності)
resolved	Boolean	Ознака вирішення
created_at	DateTime	Час створення
resolved_at	DateTime	Час закриття інциденту

Підсистема сповіщень фіксує результати аналізу останніх метрик сервісу в таблиці `alert_logs`. Записи з'являються, коли перевищено пороги з конфігурації (за замовчуванням: рівень помилок понад 10 %, середня затримка понад 1000 мс, три і більше послідовних збоїв). Повтор того самого типу сповіщення для одного сервісу блокується на 300 секунд завдяки `in-memory` журналу `alert_history`. У поточній реалізації канал доставки - лише `console`, кожне спрацьоване сповіщення відповідає одному запису в журналі. Структура наведена в табл. 2.6.

Таблиця 2.6.

Опис структури таблиці `alert_logs`

Поле	Тип даних	Призначення
<code>id</code>	Integer (PK)	Унікальний ідентифікатор запису
<code>service_id</code>	Integer (INDEX)	Посилання на сервіс
<code>alert_type</code>	String(50)	Тип сповіщення (<code>high_error_rate</code> , <code>high_latency</code> , <code>service_down</code>)
<code>severity</code>	String(20)	Рівень критичності
<code>message</code>	Text	Текст сповіщення
<code>channel</code>	String(20)	Канал доставки (<code>console</code>)
<code>created_at</code>	DateTime	Час формування запису

Ініціалізація бази даних виконується при старті застосунку функцією `init_db()`. Спочатку за декларативними моделями створюються відсутні таблиці. Для сумісності з раніше розгорнутими копіями застосунку виконуються програмні міграції: додавання нових колонок у `services` та `chaos_experiments` через `ALTER TABLE`, а також заповнення унікальних значень `entry_port` для існуючих записів. Якщо таблиця користувачів порожня, створюються облікові записи `admin` та `user` за замовчуванням. Доступ до даних у межах HTTP-запитів панелі керування організовано через генератор `get_db()`: на кожен запит FastAPI відкривається окрема ORM-сесія, яка гарантовано закривається після завершення обробки.

Висновок щодо цілісності схеми. Спроектована структура відповідає принципам нормалізації: конфігурація сервісів, експерименти, метрики,

інциденти та сповіщення рознесені по окремих таблицях, що усуває надмірне дублювання даних. Логічні зв'язки один сервіс - багато залежних записів реалізовані через `service_id` з індексацією для прискорення фільтрації та агрегації. На відміну від серверних СУБД, SQLite у даному проєкті не оголошує формальні зовнішні ключі з каскадним видаленням. Цілісність підтримується на рівні серверної логіки. Таким чином, база даних є надійним сховищем конфігурації сервісів, параметрів хаос-експериментів, результатів спостереження та подій, необхідних для роботи панелі керування та СР.

2.3. Проєктування алгоритмів роботи системи

Після опису архітектурних рішень і фізичної структури бази даних наступним етапом проєктування є формалізація динамічної поведінки системи. Якщо раніше було визначено, які дані зберігаються та як вони структуровані, то розробка алгоритмів описує логіку їх обробки, правила переходу між станами ключових сутностей і послідовність взаємодії між клієнтом, хаос-проксі, панеллю керування та бекенд. Це забезпечує передбачуваність виконання процесів моніторингу та хаос інженерії, коли один і той самий HTTP-запит може проходити як у прозорому режимі спостереження, так і під впливом активних експериментів.

Задля повноти опису застосовано інструментарій UML. Діаграми станів відображають життєвий цикл хаос-експерименту. Діаграми активностей - алгоритм пересилання запиту через СР. Діаграми послідовності - взаємодію клієнта, СР, бекенд і підсистеми збереження метрик, а також запуск експерименту з панелі керування. Діаграма компонентів узагальнює архітектурні модулі. Такий підхід формалізує центральний сценарій запит - хаос - бекенд - метрика і мінімізує ризик логічних розбіжностей між проєктом і реалізацією.

Центральний алгоритм. Основна задача пересилання HTTP-запиту через хаос-проксі. Метрика формується на кожен реальний запит, що проходить шлях від клієнта через порт входу із можливістю застосування хаос-ефектів на URL виходу до бекенду. СР визначає цільовий сервіс за портом входу, перевіряє його

		Горшков С.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Талавер О.В.				40
Змн.	Арк.	№ докум.	Підпис	Дата		

активність, завантажує конфігурацію та список активних експериментів, формує адресу бекенду з урахуванням шляху й параметрів запиту, збирає заголовки і делегує виконання HTTP-запиту модулю хаос-ін'єкції.

Якщо активних експериментів немає, запит виконується прозоро і система працює як звичайний проксі з моніторингом. Якщо експерименти є, до запиту застосовуються відповідні ефекти (затримка, імітація втрати пакетів або з'єднання, таймаут, повна відмова, випадкові помилки, або їх комбінація). Після отримання відповіді або фіксації помилки результат порівнюється з очікуваним HTTP-кодом сервісу, формується запис метрики; за невдачі може створюватися інцидент, а накопичені спостереження передаються підсистемі сповіщень. Клієнт отримує тіло та код відповіді бекенду (або JSON із описом помилки CP, якщо запит не дійшов до бекенду). Блок-схему алгоритму пересилання запиту наведено на рис. 2.4.

Алгоритм введення хаос-ефектів. Модуль хаос-ін'єкції реалізує обгортку над функцією HTTP-запиту на бекенд. Він отримує список активних експериментів у вигляді пар тип - конфігурація JSON і послідовно застосовує ефекти.

Спочатку тип `composite` розгортається на набір простих ефектів. До виконання запиту можуть застосовуватися затримка, імітація втрати пакетів або втрата з'єднання - у цих випадках запит до бекенду може не виконуватися взагалі. Для типу `timeout` зменшується ліміт очікування HTTP-клієнта. Для `outage` повертається помилка без звернення до бекенду. Після цього виконується один запит або серія паралельних запитів. Після отримання відповіді може спрацювати `error_rate` - випадкове відхилення з заданою ймовірністю. Обчислюється загальна затримка операції.

При порожньому списку експериментів усі кроки хаос пропускаються: виконується один прозорий HTTP-запит, що відповідає режиму моніторингу без введення збоїв.

		Горшков С.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Талавер О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		41

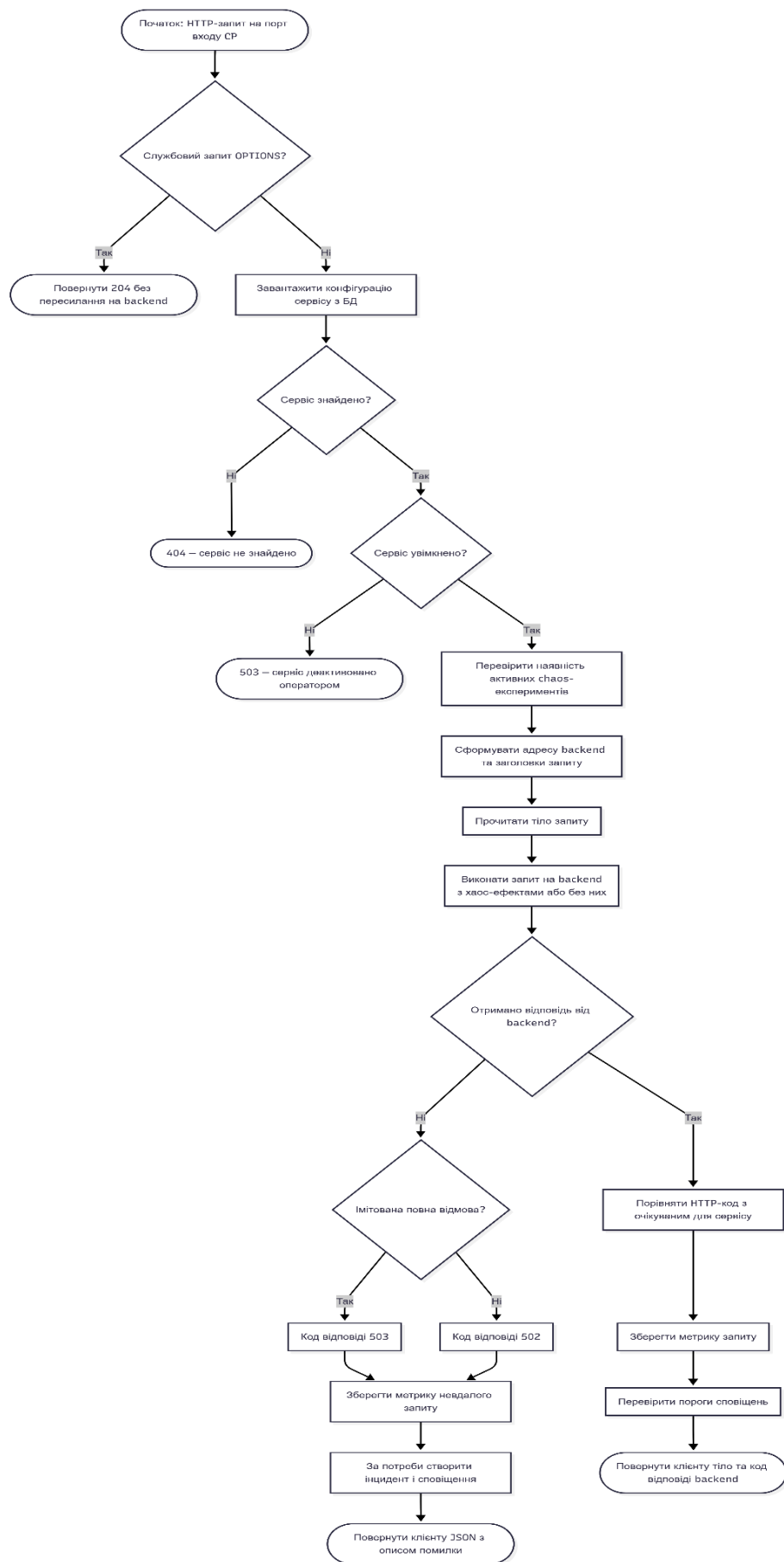


Рис. 2.4. Блок-схема алгоритму пересилання запиту через хаос-проксі

Життєвий цикл хаос-експерименту. Однією з ключових сутностей системи є хаос-експеримент, який має власний життєвий цикл. Модель «запуск - спостереження - завершення - аналіз результатів» узгоджується з рекомендаціями щодо навчального та промислового хаос інженерингу [28]. Його стани регламентовані діями оператора через панель керування та автоматизованим таймером завершення. Діаграму станів наведено на рис. 2.5.



Рис. 2.5. Діаграма станів хаос-експерименту

Активний - початковий стан після успішного створення експерименту оператором (вручну або з готового сценарію). У записі встановлюються `active = true`, `started_at`, поле `ended_at` залишається порожнім. У цьому стані СР застосовує відповідні хаос-ефекти до запитів даного сервісу. Метрики позначаються прапорцем `true`.

Завершений - фінальний стан. Перехід можливий вручну (оператор зупиняє експеримент через панель керування або автоматично - якщо при створенні задано тривалість, фоновий процес після закінчення інтервалу деактивує експеримент за тим самим принципом. Після переходу СР більше не застосовує ефекти цього експерименту. подальші запити проходять у режимі моніторингу (якщо інших активних експериментів немає).

Запуск хаос-експерименту з панелі керування. Як логічне продовження опису життєвого циклу, доцільно деталізувати алгоритм ініціалізації експерименту. Діаграму активностей цього процесу наведено на рис. 2.6.

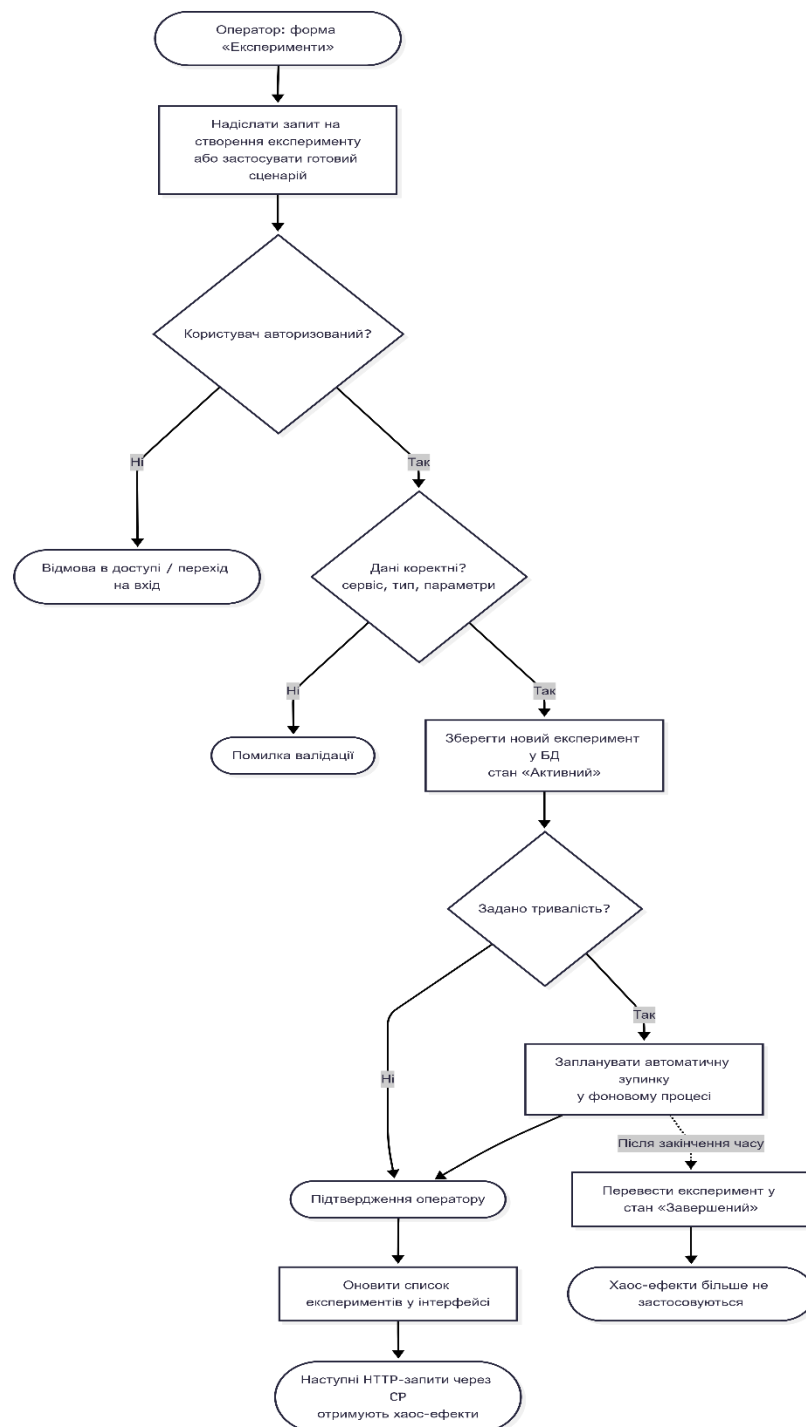


Рис. 2.6. Діаграма активностей запуску хаос-експерименту

Процес ініціюється оператором через вебінтерфейс: обирається сервіс, тип або готовий сценарій, параметри конфігурації та за потреби тривалість. Після відправки запиту на сервер панелі керування виконується перевірка

автентифікації - доступ до API мають лише авторизовані користувачі з дійсним JWT-токеном у cookie. Далі сервер валідує вхідні дані, створює запис у таблиці chaos_experiments у стані «Активний» і, якщо задано тривалість, планує автоматичну зупинку у фоновому потоці. Клієнт отримує підтвердження і наступні HTTP-запити через порт входу цього сервісу проходитимуть уже з урахуванням активного експерименту.

Діаграма послідовності запиту від клієнта через СР. Для аналізу взаємодії між Клієнтом, СР, бекенд і базою даних побудовано діаграму послідовності. Як показовий сценарій обрано проходження HTTP-запиту без активного хаос, оскільки він демонструє базовий режим моніторингу та задіює всі основні компоненти обробки. Діаграму наведено на рис. 2.7.

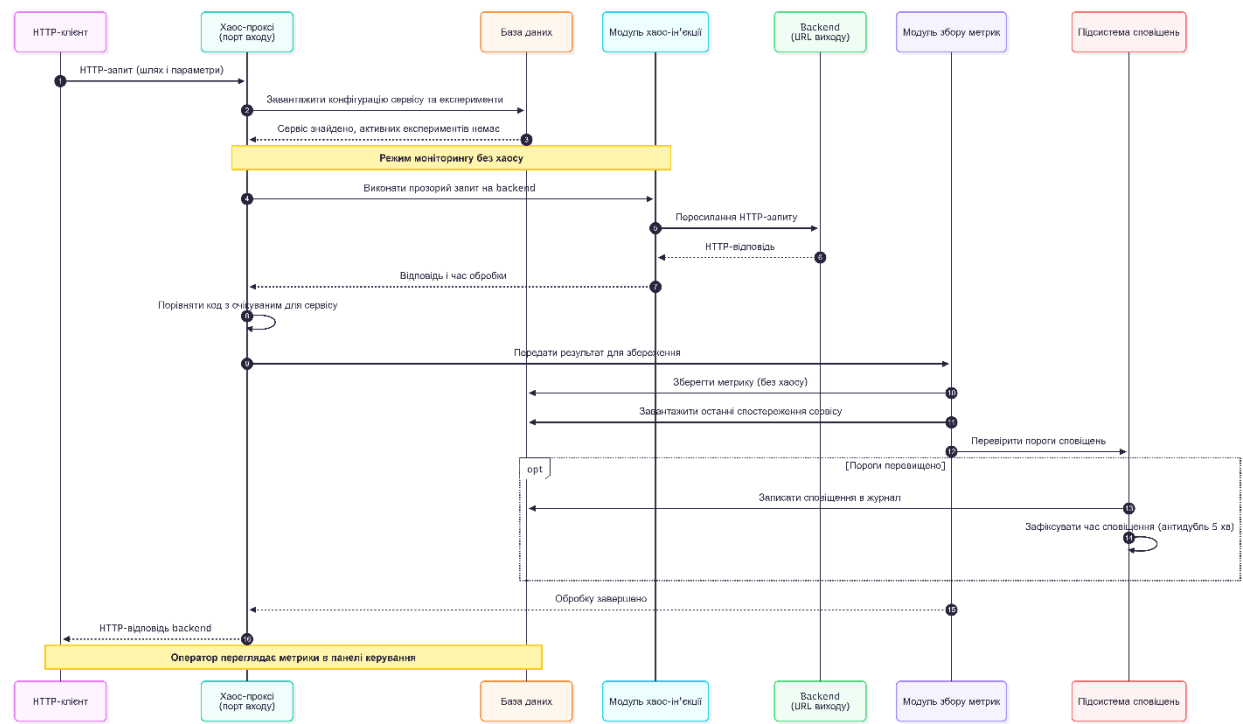


Рис. 2.7. Діаграма послідовності в режимі без хаосу

Процес починається з того, що клієнт надсилає запит на порт входу СР. СР завантажує конфігурацію сервісу з бази даних, визначає відсутність активних експериментів і виконує прозорий HTTP-запит на URL виходу. Бекенд повертає відповідь і СР порівнює код статусу з очікуваним значенням, формує запис метрики та передає його модулю збереження метрик. Той зберігає метрику у SQLite, за необхідності створює інцидент і ініціює перевірку порогів у менеджері

сповіщень. Відповідь повертається клієнту. Оператор може переглянути запис у журналі «Активність» на панелі керування через REST API - дані читаються з уже збережених метрик, без повторного опитування бекенду.

Діаграма послідовності, якщо запит надсилається під час активного хаос. Окремо варто розглянути сценарій, коли для сервісу існує активний експеримент. Діаграму послідовності наведено на рис. 2.8.

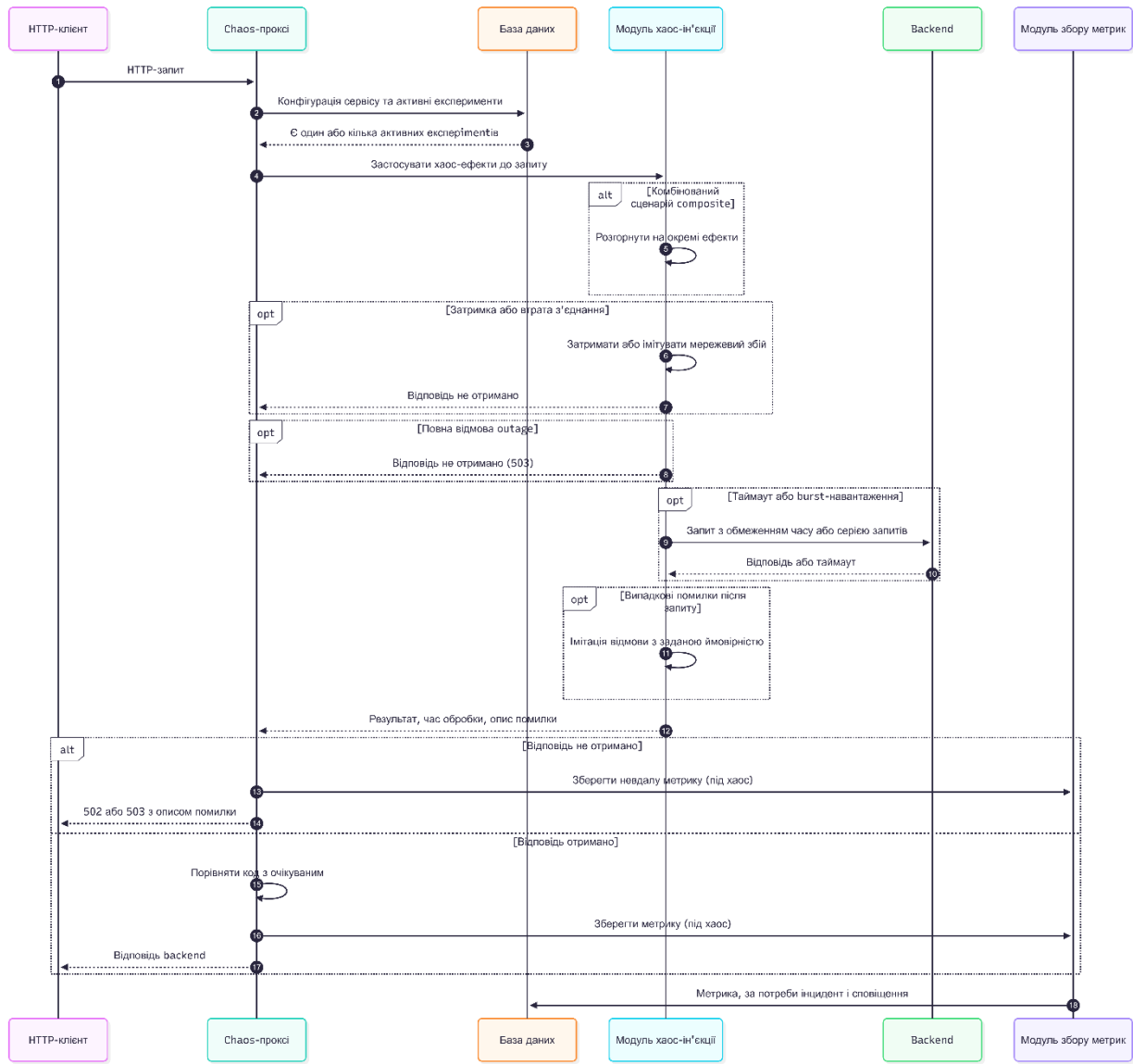


Рис. 2.8. Діаграма послідовності з застосуванням хаос-ефектів

Ланцюг обробки аналогічний базовому, але після завантаження конфігурації CP отримує список активних експериментів і передає його модулю хаос-ін'єкції. Той може затримати запит, імітувати мережевий збій, обмежити таймаут або повністю заблокувати звернення до бекенду. Якщо запит до бекенду

все ж виконується, результат може бути модифікований post-ефектами. Метрика зберігається з прапорцем активного хаосу, що дозволяє в подальшому порівняти поведінку системи з хаос і без хаосу у звітах аналітики.

Алгоритм запису метрик і сповіщень. Після кожного запиту через СР модуль збереження метрик виконує постобробку результату. Створюється запис у таблиці metrics з часовою міткою, затримкою, кодом відповіді, ознакою успіху, текстом помилки та прапорцем активності хаосу. Якщо запит визнано невдалим, створюється інцидент. Механізм дедуплікації не дублює однакові відкриті інциденти протягом п'яти хвилин.

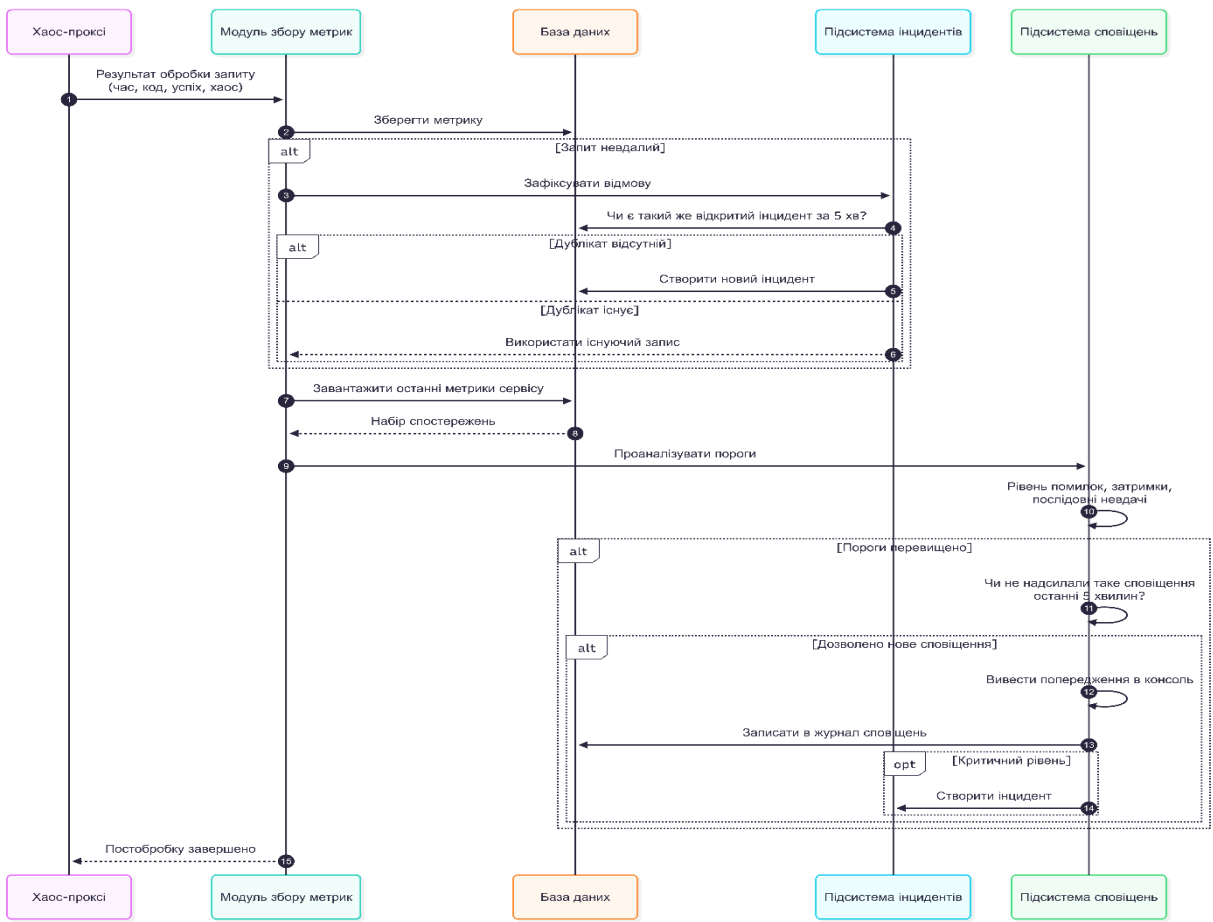


Рис. 2.9. Діаграма послідовності запису метрики, інциденту та сповіщення

Далі завантажуються останні метрики сервісу, і підсистема сповіщень аналізує їх на перевищення порогів: рівень помилок, середня затримка, серія послідовних невдач. Якщо поріг перевищено, формується сповіщення, записується рядок у alert_logs і виводиться повідомлення в консоль сервера.

Повтор того самого типу сповіщення для одного сервісу блокується на 300 секунд. Критичні сповіщення додатково ініціюють інцидент. Діаграму послідовності цього процесу наведено на рис. 2.9.

При старті застосунку виконується ініціалізація бази даних і, за увімкненої конфігурації, запуск CP для кожного зареєстрованого сервісу: для кожного вхідного порту у фоновому даємон-потоці піднімається окремий екземпляр HTTP-проксі. Таким чином кожен сервіс має незалежний канал перехоплення трафіку.

Панель керування надає REST API та вебінтерфейс оператору. Взаємодіє з SQLite через ORM SQLAlchemy. Окремо для кожного сервісу працюють екземпляри CP (окремі порти входу), які використовують модуль хаос-ін'єкції, пересилають запити на бекенд (URL виходу) і через модуль метрик записують результати в базу. Підсистема інцидентів і сповіщень опирається на ті самі метрики. Клієнт HTTP-трафіку (демо-застосунок або зовнішній клієнт) з'єднується лише з портом входу CP, а не з бекенду напряму - що забезпечує єдину точку спостереження та введення хаос. Узагальнену взаємодію програмних модулів наведено на рис. 2.10.

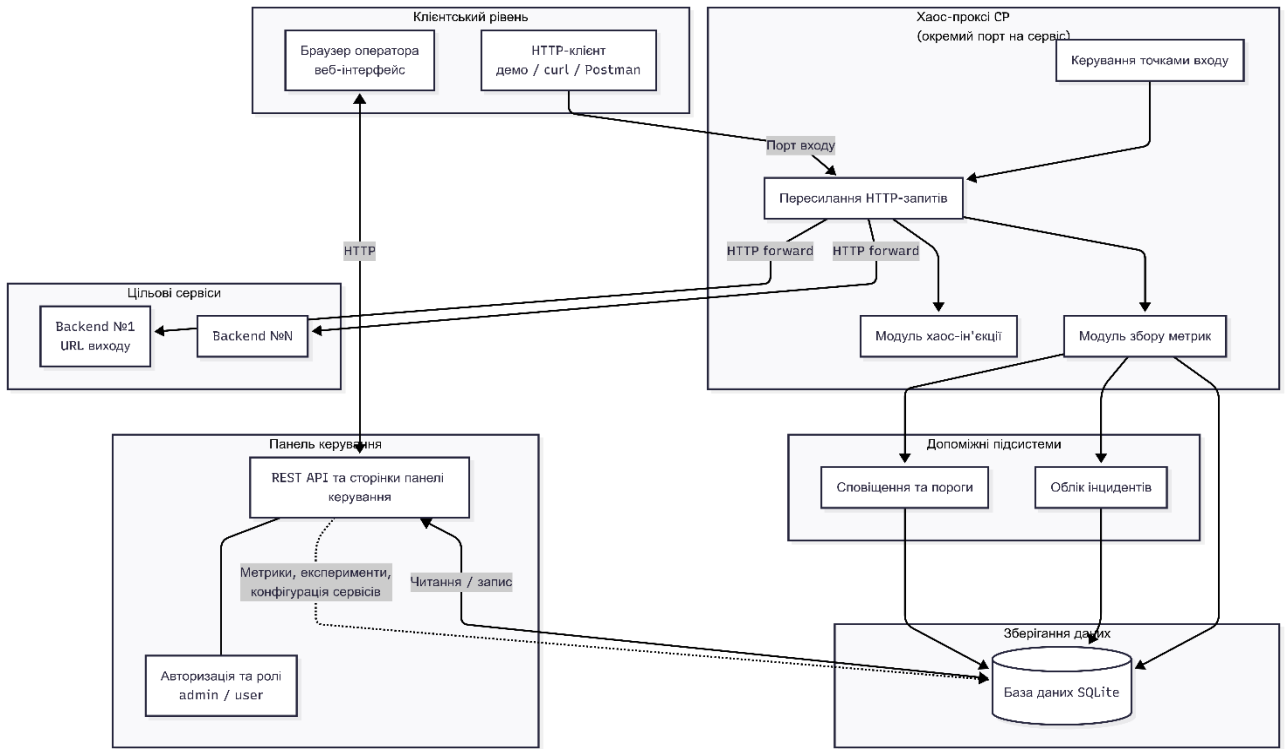


Рис. 2.10. Діаграма компонентів системи Chaos Proxy

Застосування UML дозволило формалізувати динамічну поведінку системи хаос інженерії та моніторингу. Деталізовано життєвий цикл хаос-експерименту з ручним та автоматичним завершенням, алгоритм пересилання HTTP-запиту через CP як центральний процес платформи, послідовність взаємодії клієнта, проксі, бекенд і бази даних у режимах з хаос і без нього, а також ланцюг метрика - інцидент - сповіщення. Діаграма компонентів відображає розділення панелі керування і CP з окремими точками входу. Сформований опис створює підґрунтя для програмної реалізації та забезпечує передбачуваність роботи системи при моніторингу та контрольованому введенні збоїв.

2.4. Реалізація функціоналу системи

Реалізацію основних архітектурних принципів доцільно розглянути на прикладі проходження HTTP-запиту через хаос-проксі. Саме цей сценарій задіює усі рівні системи такі як клієнтський HTTP-трафік, окремий екземпляр CP на порті входу, модуль хаос-ін'єкції, цільовий бекенд, збереження метрик у SQLite та подальше відображення результатів у панелі керування.

Систему реалізовано як модульний моноліт на Python який складається з панелі керування та незалежні екземпляри CP (окремі порти входу) спільно використовують одну базу даних. Подальший опис побудовано за ланцюгом обробки запиту — від прийому HTTP на точці входу, через застосування хаос-ефектів і пересилання на бекенд, до фіксації метрики, інциденту та сповіщення.

Логіку імітації збоїв інкапсулює клас ChaosInjector. Перед обробкою список активних експериментів нормалізується. Метод `apply_chaos` обгортає функцію HTTP-запиту на бекенд і застосовує ефекти у фіксованому порядку. Спочатку застосовується затримка, потім імітація втрати пакетів або з'єднання, далі обмеження таймауту або повна недоступність або виконання запиту. Після відповіді бекенд можлива випадкова відмова. Якщо активних експериментів немає, виконується один прозорий запит у режимі моніторингу.

Ключова частина алгоритму - послідовне застосування ефектів до виконання запиту на бекенд:

		Горшков С.В.			ІПЗ.КР.Б-121-26-ІЗ	Арк.
		Талавер О.В.				49
Змн.	Арк.	№ докум.	Підпис	Дата		

Спрощений фрагмент (відповідає chaos_injector.py):

```
for exp_type, config in expanded:
    if exp_type == "latency":
        await asyncio.sleep(config.get("delay_ms", 0) / 1000.0)
    elif exp_type == "packet_loss":
        if random.random() < config.get("loss_rate", 0.1):
            return None, latency_ms, False, "Packet loss"
# connection_loss
async with httpx.AsyncClient(timeout=timeout) as client:
    response = await make_request_func(client)
```

Таким чином логіка імітації збоїв зосереджена в одному класі, а СР лише передає контекст запиту та функцію пересилання.

Транспортний рівень СР реалізовано в модулі chaos_proxu.py. Функція forward_request завантажує конфігурацію сервісу, перевіряє його активність, формує URL бекенду і делегує виконання модулю хаос-ін'єкції. Після отримання результату СР порівнює HTTP-код з expected_status_code і передає дані модулю збору метрик (proxu_metrics.py), який створює запис у таблиці metrics, за потреби - інцидент і перевірку порогів сповіщень.

Ключове технічне рішення - asyncio та httpx для неблокуючого HTTP і окремі daemon-потоки uvicorn на кожен порт входу, що забезпечує паралельну обробку кількох сервісів. Пересилання запитів на URL виходу виконується асинхронним HTTP-клієнтом httpx [15] у поєднанні з uvicorn [31].

Реалізація REST API. REST API панелі керування реалізовано відповідно до архітектурного стилю REST [24] на базі фреймворку FastAPI [12]. Без авторизації доступна лише сторінка входу, решта маршрутів захищені залежностями require_user або require_admin. Операції створення, зміни та видалення сервісів і користувачів доступні лише адміністратору, а перегляд метрик, експериментів і звітів - користувачу та адміністратору. Валідація тіла запитів виконується моделями Pydantic [14]. Доступ до БД - через генератор сесії get_db(). Авторизація - JWT згідно RFC 7519 [19].

Ендпоінти не містять хаос-логіки - вони керують конфігурацією та читають накопичені метрики. Наприклад, /api/analytics/chaos-impact/{service_id} розділяє записи за прапорцем chaos_active, обчислює середню затримку, частку

		Горшков С.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Талавер О.В.				50
Змн.	Арк.	№ докум.	Підпис	Дата		

успішних запитів і показники деградації. Аналогічно побудовано SLO-compliance, тижневі звіти та експорт CSV.

Таблиця 2.7

Перелік основних REST API ендпоінтів

Метод	URL	Призначення	Доступ
POST	/api/auth/login	Авторизація, видача JWT	—
POST	/api/auth/logout	Вихід із системи	user, admin
GET	/api/auth/me	Дані поточного користувача	user, admin
PUT	/api/auth/me/password	Зміна власного пароля	user, admin
GET	/api/users	Список користувачів	admin
POST	/api/users	Створення користувача	admin
PUT	/api/users/{user_id}/password	Зміна пароля користувача	admin
PUT	/api/users/{user_id}/toggle	Активація / деактивація облікового запису	admin
DELETE	/api/users/{user_id}	Видалення користувача	admin
GET	/api/services	Список сервісів	user, admin
POST	/api/services	Реєстрація сервісу	admin
PUT	/api/services/{service_id}	Редагування сервісу	admin
PUT	/api/services/{service_id}/toggle	Увімкнення / вимкнення сервісу	admin
DELETE	/api/services/{service_id}	Видалення сервісу та пов'язаних даних	admin

продовж. табл. 2.7

GET	/api/experiments	Список експериментів (з фільтрами)	user, admin
POST	/api/experiments	Запуск експерименту	user, admin
POST	/api/experiments/{experiment_id}/stop	Зупинка експерименту	user, admin
GET	/api/chaos/scenarios	Готові сценарії хаосу	user, admin
POST	/api/chaos/scenarios/{scenario_name}/apply	Застосування готового сценарію	user, admin
GET	/api/activity	Журнал HTTP-запитів через СР	user, admin
GET	/api/metrics/{service_id}	Агреговані метрики для графіків	user, admin
GET	/api/health/{service_id}	Зведені показники за метриками проксі	user, admin
GET	/api/incidents	Журнал інцидентів	user, admin
PUT	/api/incidents/{incident_id}/resolve	Позначення інциденту вирішеним	user, admin
GET	/api/alerts	Журнал сповіщень	user, admin
GET	/api/analytics/chaos-impact/{service_id}	Порівняння «з хаос / без хаос»	user, admin
GET	/api/analytics/slo-compliance/{service_id}	Відповідність SLO	user, admin
GET	/api/reports/weekly/{service_id}	Тижневий звіт	user, admin
GET	/api/export/metrics/{service_id}	Експорт метрик у CSV	user, admin
GET	/api/export/chaos-history	Експорт історії експериментів у CSV	user, admin

Вебінтерфейс побудовано як набір серверних сторінок на Jinja2 [18] зі спільним каркасом base.html і стилізацією Bootstrap 5 [17]. Кожен функціональний розділ (активність, сервіси, експерименти, історія, звіти, користувачі) відповідає окремому маршруту FastAPI. Після завантаження HTML дані підвантажуються асинхронно через Fetch API, без перезавантаження всього застосунку. Спільний модуль авторизації додає JWT з cookie до запитів і при відповіді 401 перенаправляє на сторінку входу [19].

Логіка інтерфейсу організована за принципом сторінка - REST-ендпоінт. Журнал активності читає /api/activity, форми сервісів і експериментів - відповідні POST/PUT/DELETE маршрути, розділ звітів поєднує /api/metrics/{service_id} з блоками chaos-impact, SLO та тижневого звіту. Графіки будуються Plotly.js [16] на клієнті. Адміністративні дії доступні лише за наявності ролі admin, що відображається в навігаційній панелі разом із іменем оператора.

Запуск експерименту з панелі ілюструє другий канал системи. Оператор надсилає автентифікований POST-запит і сервер створює запис у chaos_experiments у стані «активний» і за потреби планує автозупинку. СР окремої команди не отримує та при наступному HTTP-запиті клієнта він читає активні експерименти з БД і застосовує ефекти. Таке розділення керування та обробки трафіку зменшує зв'язність компонентів і дозволяє масштабувати точки входу незалежно від панелі керування.

Висновки до розділу 2

У другому розділі виконано проєктування системи Chaos Proxy описано предметну модель, структуру даних, алгоритми роботи та динамічну поведінку основних процесів. Результати проєктування узгоджено з постановкою задачі з першого розділу та створюють основу для програмної реалізації і тестування в наступних розділах.

У першому підрозділі визначено варіанти використання системи та основні сутності предметної області. Ролі операторів поділено на адміністратора, користувача панелі керування та HTTP-клієнт, який генерує трафік через порт

		Горшков С.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Талавер О.В.				53
Змн.	Арк.	№ докум.	Підпис	Дата		

входу. Побудовано діаграми варіантів використання та класів, що відображають зв'язки між панеллю керування, хаос-проксі, модулями хаос-ін'єкції, збору метрик і сповіщень.

У другому підрозділі спроектовано реляційну модель бази даних SQLite із шести таблиць. Центральною сутністю є конфігурація сервісу - пара порт входу - адреса виходу. Окремо виділено облікові записи операторів, параметри хаос-експериментів, журнал HTTP-запитів, інциденти та записи сповіщень. Наведено фізичну схему бази даних і опис структури основних таблиць.

У третьому підрозділі формалізовано алгоритми функціонування системи. Центральним є пересилання HTTP-запиту через хаос-проксі з можливим застосуванням хаос-ефектів, збереженням метрик і формуванням сповіщень за порогоми. Окремо описано життєвий цикл хаос-експерименту, запуск експерименту з панелі керування та ланцюг «метрика - інцидент - сповіщення». Процеси подано у вигляді діаграм активностей, станів і послідовностей. Підтверджено, що моніторинг базується на кожному реальному запиті через точку входу, а не на фоновому опитуванні сервісів.

У четвертому підрозділі визначено склад програмного комплексу та інтерфейс взаємодії з ним: REST API панелі керування з авторизацією, логічне розділення панелі керування та окремих точок входу хаос-проксі для кожного сервісу, структура багатосторінкового вебінтерфейсу (активність, сервіси, експерименти, історія, звіти, користувачі). Сформований проєкт забезпечує два режими роботи — спостереження без активного хаосу та хаос-інженерію під час проведення експериментів.

		Горшков С.В.			ІПЗ.КР.Б-121-26-ІПЗ	Арк.
		Талавер О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		54

РОЗДІЛ 3. ІНТЕРФЕЙС, ТЕСТУВАННЯ ТА РОЗГОРТАННЯ

3.1. Порядок встановлення та налаштування системи

Системні вимоги та залежності програмного забезпечення є фундаментальним аспектом успішного розгортання та експлуатації будь-якої інформаційної системи, оскільки вони визначають сумісність компонентів, стабільність роботи та можливість подальшого масштабування. Перед встановленням розробленої системи хаос проксі та моніторингу HTTP-трафіку необхідно переконатися у наявності інтерпретатора Python версії не нижче 3.10, що зумовлено використанням сучасних можливостей мови (анотації типів, асинхронне програмування через `asyncio`, синтаксис `union`-типів `int | None`), які забезпечують коректну роботу FastAPI, `httpx` та SQLAlchemy 2.0. Система розрахована на роботу в операційних середовищах Windows, Linux та macOS без додаткових серверів баз даних або оркестраторів контейнерів. Усі зовнішні бібліотеки та залежності проєкту чітко визначені у файлі `requirements.txt` із зазначенням конкретних версій, що гарантує відтворюваність середовища розробки та унеможливорює конфлікти між компонентами. Встановлення залежностей здійснюється стандартним пакетним менеджером `pip`, який автоматично завантажує та інсталує необхідні модулі з репозиторію PyPI.

Порядок розгортання системи реалізовано у вигляді послідовності логічно пов'язаних кроків, кожен з яких виконує конкретну функцію підготовки середовища та ініціалізації компонентів. Послідовність підготовки середовища, ініціалізації компонентів і перевірки працездатності відповідає загальним принципам експлуатаційного супроводу програмних систем [21]. Перший етап - отримання вихідного коду (клонування репозиторію Git або розархівування) та перехід до кореневої директорії проєкту, що забезпечує коректні відносні шляхи до модулів (`app.py`, `chaos_proxy.py`, `config.py`), шаблонів, статичних ресурсів та конфігураційних файлів. Другий етап — створення та активація ізольованого віртуального середовища Python командою `python -m venv .venv`: усі залежності встановлюються локально в `.venv`, не впливаючи на глобальний інтерпретатор.

		Горшков С.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Талавер О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		55

Активация виконується платформо-специфічно: `.venv\Scripts\activate` (Windows) або `source .venv/bin/activate` (Linux/macOS). Третій етап - `pip install -r requirements.txt`. Четвертий етап - підготовка конфігурації: копіювання `.env.example` у `.env` (`copy .env.example .env` або `cp .env.example .env`) та зміна `SECRET_KEY` перед використанням у production, за потреби налаштовуються порти `dashboard`, хаос-проксі, пороги алертів і канали сповіщень. П'ятий етап - запуск системи скриптом `python run.py`, який піднімає ASGI-сервер `uvicorn` для панелі керування (за замовчуванням `http://127.0.0.1:8000`) з параметрами з `config.py`. Альтернатива для розробки: `uvicorn app:app --host 127.0.0.1 --port 8000 --reload`. Шостий етап - відкриття браузера за адресою `http://localhost:8000/login`, авторизація (за замовчуванням логін `admin` і пароль `admin`) та робота з вебінтерфейсом. Документація REST API - `http://localhost:8000/docs` (Swagger UI).

При першому запуску спрацьовує `init_db()` (`database.py`): створюється файл `chaos_monitor.db`, якщо він відсутній. SQLAlchemy виконує `create_all()` — шість таблиць (`users`, `services`, `chaos_experiments`, `metrics`, `incidents`, `alert_logs`); додаткові міграції колонок (`ALTER TABLE`) для сумісності зі старими версіями схеми. `backfill` унікальних `entry_port` для сервісів. `ensure_default_users()` створює облікові записи `admin` і `user` при першому запуску. Якщо файл БД уже існує, зміна паролів у `.env` не оновлює наявних користувачів. Паралельно на події `startup` викликається `start_chaos_proxy()`: для кожного зареєстрованого сервісу (за `CHAOS_PROXY_ENABLED=true`) у `daemon`-потоці запускається окремий `uvicorn` на порті входу (`:8002`, `:8003`, ...). SQLite оптимальна для розгортання, тестування та експлуатації при помірному навантаженні: один файл містить конфігурацію сервісів, метрики проксі-трафіку, хаос-експерименти, інциденти, журнал алертів і користувачів; резервне копіювання - копіювання `chaos_monitor.db`.

Налаштування параметрів централізовано у файлі `.env`, який зчитується класом `Settings` у модулі `config.py`, що відповідає принципу єдиного джерела налаштувань у DevOps-практиках [22]. Основні параметри наведено в табл. 3.1.

		Горшков С.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Талавер О.В.				56
Змн.	Арк.	№ докум.	Підпис	Дата		

Таблиця 3.1

Параметри конфігурації системи

Параметр	За замовчуванням	Призначення
DATABASE_URL	sqlite:///./chaos_monitor.db	Шлях до файлу бази даних SQLite
APP_HOST	127.0.0.1	Адреса прослуховування панелі керування
APP_PORT	8000	Порт панелі керування (dashboard)
CHAOS_PROXY_ENABLED	true	Увімкнення хаос-проксі при старті (false — CP не запускається)
CHAOS_PROXY_HOST	127.0.0.1	Адреса для формування URL точок входу CP
CHAOS_PROXY_PORT	8001	Базове значення для автопризначення портів входу сервісів
SECRET_KEY	change-me-in-production	Секрет для JWT та хешування паролів (pepper)
JWT_EXPIRE_HOURS	0	Термін дії JWT у годинах (0 — до виходу з системи)
ADMIN_USERNAME	admin	Логін облікового запису адміністратора за замовчуванням
ADMIN_PASSWORD	admin	Пароль адміністратора за замовчуванням
DEFAULT_USER_USERNAME	user	Логін користувача за замовчуванням
DEFAULT_USER_PASSWORD	user	Пароль користувача за замовчуванням
ALERT_LATENCY_THRESHOLD_MS	1000	Поріг середньої затримки для сповіщення, мс
ALERT_ERROR_RATE_THRESHOLD	0.1	Поріг частки помилок
ALERT_CONSECUTIVE_FAILURES	3	Кількість послідовних невдач для сповіщення «сервіс недоступний»

Пороги сповіщень підвантажуються в AlertManager у структуру alerts_config. Повторне сповіщення того ж типу для одного сервісу блокується на 300 секунд, що зменшує «шум» у журналі. Канал доставки в поточній версії - console (запис у alert_logs).

Для повної демонстрації передбачено демо-стенд необхідно запустити три процеси. Спочатку run.py, а потім demo/backend.py та demo/serve_client.py.

Схему розгортання системи в локальному середовищі наведено на рис. 3.1. Оператор працює з панеллю керування через браузер після входу в обліковий запис. HTTP-клієнт надсилає запити на порт входу хаос-проксі, проксі за потреби застосовує хаос-ефекти і пересилає запити на бекенд за адресою виходу. Результати спостереження зберігаються в файлі бази даних SQLite. Панель керування та окремі точки входу проксі запускаються з одного проєкту.

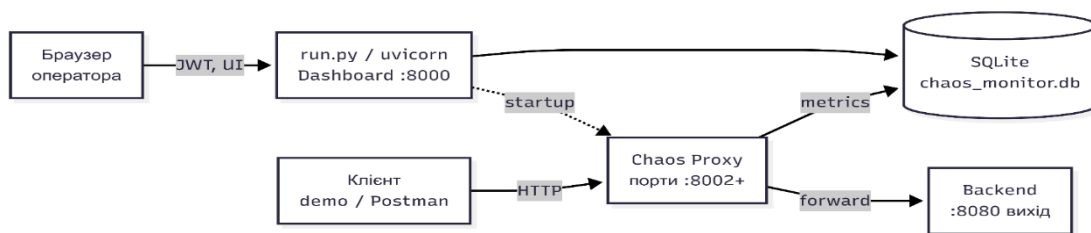


Рис. 3.1. Діаграма розгортання системи

3.2. Структура інтерфейсу системи

Клієнтську частину системи реалізовано як багатосторінковий вебзастосунок на серверних шаблонах Jinja2 [18], CSS-фреймворку Bootstrap 5.3 та vanilla JavaScript. На відміну від односторінкових архітектур, кожен функціональний розділ відповідає окремому маршруту FastAPI і власному HTML-шаблону. Спільний каркас винесено в base.html, що забезпечує єдину навігаційну панель, підключення скриптів авторизації та спільних утиліт. Після завантаження сторінки дані підвантажуються асинхронно через Fetch API - без повного перезавантаження всього застосунку, але в межах поточного розділу. Такий підхід знижує складність розгортання, мінімізує залежності та робить логіку інтерфейсу прозорою для супроводу.

Загальний дизайн виконано у світлій кольоровій гамі: нейтральний фон робочої області, темна навігаційна панель, акцентні елементи для кнопок і

		Горшков С.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Талавер О.В.				58
Змн.	Арк.	№ докум.	Підпис	Дата		

посилань. Картки мають легку тінь і заокруглені кути; інтерактивні компоненти Bootstrap забезпечують візуальний відгук при наведенні. Адаптивний макет побудовано на сітці Bootstrap і Flexbox, що гарантує коректне відображення на екранах від мобільних пристроїв до широкоформатних моніторів. Далі розглянуто структуру та основні елементи інтерфейсу за функціональними групами.

Взаємодія оператора з системою розпочинається з процедури входу. Неавторизований доступ до захищених маршрутів перенаправляється на сторінку автентифікації, наведену на рис. 3.2.

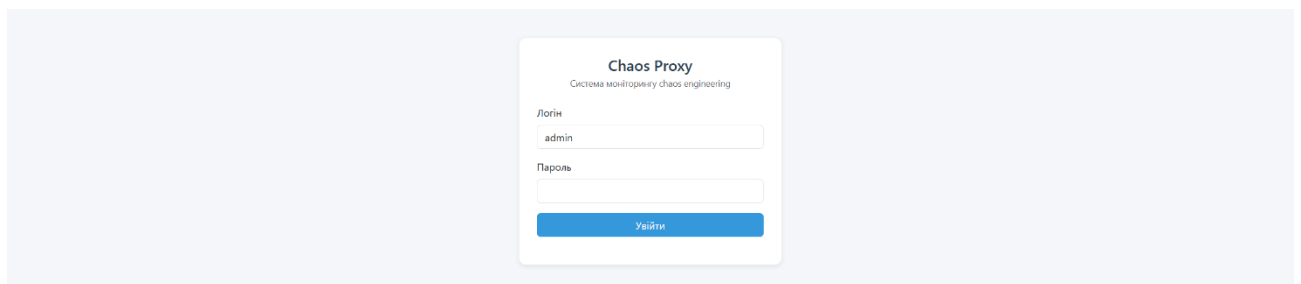


Рис. 3.2. Сторінка входу в систему

Сторінка входу має окремий макет без навігаційної панелі: поля «Логін» і «Пароль», кнопка «Увійти». Після успішної автентифікації JWT-токен зберігається в cookie, що дозволяє працювати з REST API та захищеними HTML-сторінками без повторного введення облікових даних. Реєстрація нових користувачів через публічний інтерфейс не передбачена - облікові записи створює адміністратор у відповідному розділі панелі керування.

Після входу оператор потрапляє на головний екран «Активність». У верхній навігаційній панелі відображаються ім'я користувача, роль (admin або user) та кнопка «Вихід». Для адміністратора додатково доступний пункт «Користувачі». Навігація охоплює розділи: Активність, Сервіси, Експерименти, Історія експериментів, Звіти; активний пункт підсвічується для зручної орієнтації в системі.

Головний операторський екран призначений для спостереження за HTTP-запитами, що проходять через хаос-проксі. Його інтерфейс наведено на рис. 3.3.

		Горшков С.В.			ІПЗ.КР.Б-121-26-ІЗ	Арк.
		Талавер О.В.				59
Змн.	Арк.	№ докум.	Підпис	Дата		

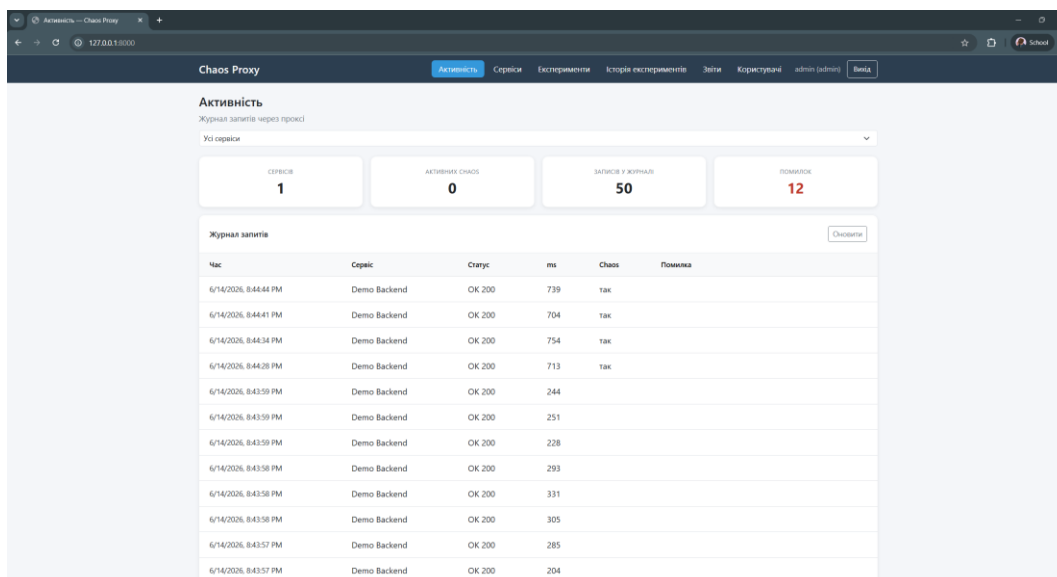


Рис. 3.3. Сторінка «Активність»

У шапці сторінки пояснено призначення журналу. Селектор сервісу дозволяє переглядати записи для всіх зареєстрованих сервісів або фільтрувати дані за одним обраним сервісом. Чотири інформаційні картки відображають зведені показники: кількість сервісів, кількість активних хаос-експериментів, загальну кількість записів у журналі та число невдалих запитів.

Основну частину екрана займає таблиця журналу з колонками: час запиту, назва сервісу, статус успішності (OK/FAIL) із HTTP-кодом, час затримки у мілісекундах, ознака активного хаосу та текст помилки за наявності. Передбачено ручне оновлення даних і автоматичне оновлення з інтервалом десять секунд. Інформаційний блок нагадує, що записи з'являються лише тоді, коли через порт входу проходить реальний HTTP-трафік: запуск хаос-експерименту без клієнтських запитів сам по собі не формує метрик.

Сторінка «Сервіси». Розділ призначений для перегляду та керування конфігурацією моніторингових каналів - пари «порт входу - URL виходу». Інтерфейс наведено на рис. 3.4.

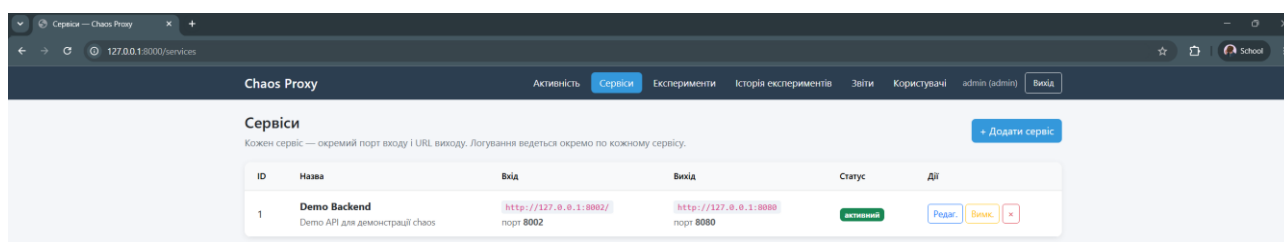


Рис. 3.4 – Сторінка «Сервіси»

Таблиця містить ідентифікатор, назву сервісу, адресу входу, адресу виходу та статус активності. Користувач із роллю user має право лише перегляду. Адміністратор може додавати нові сервіси через модальне вікно (назва, порт входу, параметри виходу, опис), редагувати наявні записи, тимчасово вимикати або вмикати сервіс, а також видаляти їх з системи. Деактивований сервіс не пересилає запити на бекенд і повертає відповідь з кодом 503.

Центральний розділ хаос інженерії дозволяє запускати та зупиняти експерименти для обраного сервісу. Сторінку наведено на рис. 3.5. Інтерфейс організовано у два стовпці. У лівій частині розміщено блок готових сценаріїв - пресетів типових збоїв (затримка мережі, проблеми з'єднання, високий рівень помилок, повна відмова сервісу, стрес-тест тощо), які можна застосувати до обраного сервісу одним натисканням. Нижче розташована форма власного експерименту: вибір сервісу, тип хаос-ефекту, динамічно змінювані поля конфігурації залежно від обраного типу, тривалість у хвилинах та кнопка запуску.

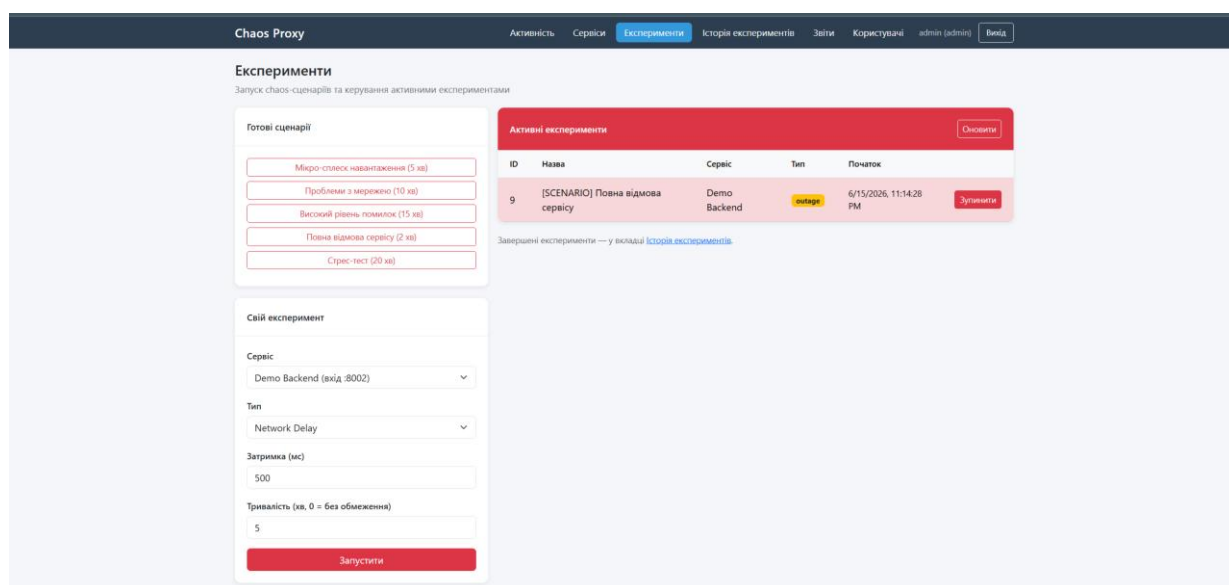


Рис. 3.5 – Сторінка «Експерименти»

У правій частині відображається таблиця активних експериментів з виділенням рядків, для кожного доступна дія «Зупинити». Список періодично оновлюється, що дозволяє оператору контролювати поточний стан хаосу без перезавантаження сторінки.

Архів завершених хаос-експериментів винесено в окремий розділ, наведений на рис. 3.6. Користувач може фільтрувати записи за сервісом, типом експерименту та статусом, переглядати час початку і завершення, а також вивантажувати історію хаосу у форматі CSV для подальшого аналізу або включення до пояснювальної записки.

ID	Назва	Сервіс	Тип	Статус	Початок	Кінець	Результат
9	[SCENARIO] Повна відмова сервісу	Demo Backend	outage	Завершено	6/15/2026, 11:14:28 PM	6/15/2026, 11:16:28 PM	—
8	network_delay_1781459136673	Demo Backend	network delay	Завершено	6/14/2026, 8:45:36 PM	6/14/2026, 8:48:00 PM	0% / 0 ms
7	network_delay_1781459058914	Demo Backend	network delay	Завершено	6/14/2026, 8:44:18 PM	6/14/2026, 8:44:49 PM	100% / 727.5 ms
6	random_failure_1781453008120	Demo Backend	random failure	Завершено	6/14/2026, 7:03:28 PM	6/14/2026, 7:06:53 PM	56.7% / 748.7 ms
5	[SCENARIO] Стрес-тест	Demo Backend	composite	Завершено	6/14/2026, 6:59:35 PM	6/14/2026, 7:06:53 PM	73.2% / 726.1 ms
4	[SCENARIO] Стрес-тест	Demo Backend	composite	Завершено	6/14/2026, 6:53:06 PM	6/14/2026, 6:59:31 PM	84.5% / 733.7 ms
3	[SCENARIO] Повна відмова сервісу	Demo Backend	outage	Завершено	6/14/2026, 6:48:32 PM	6/14/2026, 6:49:06 PM	0% / 1 ms
2	[SCENARIO] Високий рівень помилок	Demo Backend	error_rate	Завершено	6/14/2026, 6:48:31 PM	6/14/2026, 6:49:05 PM	5.9% / 66.9 ms
1	[SCENARIO] Проблема з мережею	Demo	composite	Завершено	6/14/2026, 6:48:25	6/14/2026, 6:53:09	88% / 151.9 ms

Рис. 3.6 – Сторінка «Історія Експериментів»

Розділ аналітики призначений для оцінки метрик, зібраних хаос-проксі під час реального HTTP-трафіку. Інтерфейс наведено на рис. 3.7. Після вибору сервісу оператор задає часовий діапазон: «Останні 12 годин» (оперативний огляд за замовчуванням) або «За весь період» (повна наявна історія в базі даних). Додатково доступний крок агрегації — від кількох секунд до доби- для групування сирих точок у зручні інтервали. На інтерактивному графіку Plotly.js відображається динаміка затримки та успішності запитів, заголовок графіка відображає обраний діапазон і крок.

Після вибору сервісу оператор задає часовий діапазон: «Останні 12 годин» (оперативний огляд за замовчуванням) або «За весь період» (повна наявна історія в базі даних). Додатково доступний крок агрегації — від кількох секунд до доби- для групування сирих точок у зручні інтервали. На інтерактивному графіку

Plotly.js відображається динаміка затримки та успішності запитів, заголовок графіка відображає обраний діапазон і крок.

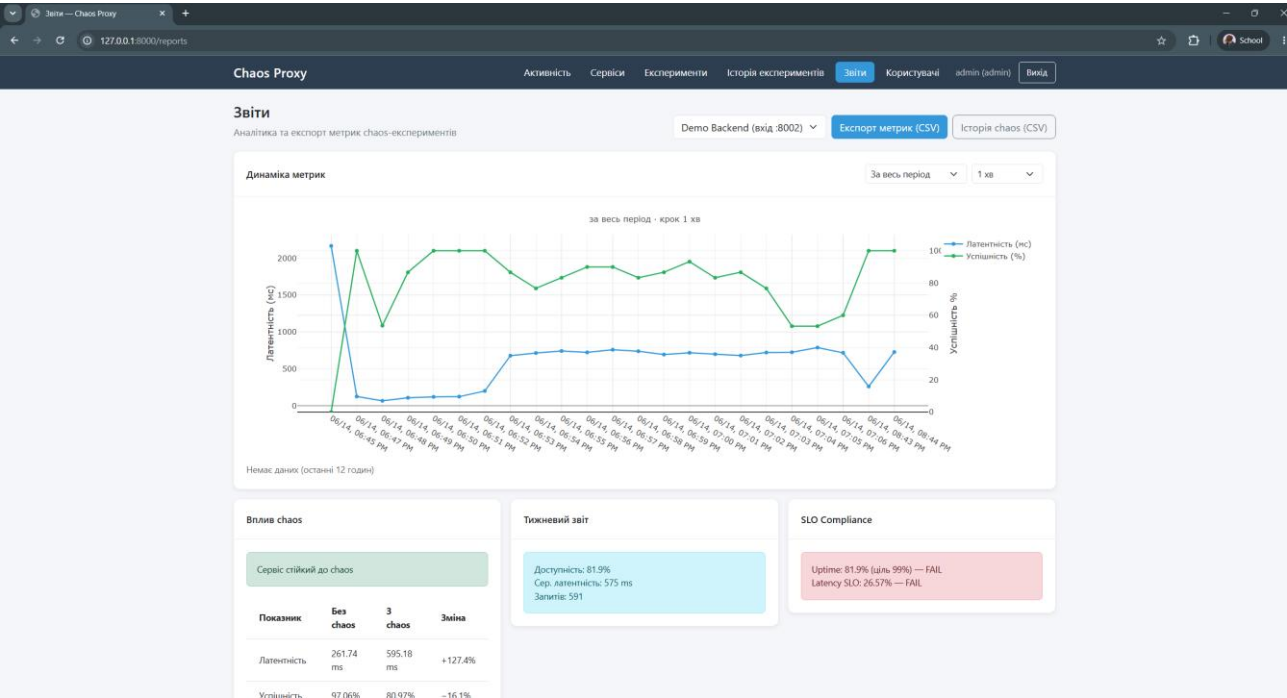


Рис. 3.7 – Сторінка «Звіти»

Під графіком розташовані блоки «Вплив хаосу» (порівняння метрик за періодами з активним та неактивним хаос), «Тижневий звіт» та «SLO Compliance». Кнопки експорту дозволяють вивантажити метрики обраного сервісу та історію хаос-експериментів у CSV.

Адміністрування облікових записів операторів панелі керування реалізовано окремим розділом, доступним лише користувачам із роллю admin. Інтерфейс наведено на рис. 3.8.

Користувачі

Новий користувач

Логін: Пароль: Роль:

Список користувачів

ID	Логін	Роль	Статус	Дії
1	admin	admin	активний	Змінити пароль захищений
2	user	user	активний	Змінити пароль On/Off X

Рис. 3.8 – Сторінка «Користувачі»

На сторінці розміщено форму створення нового користувача (логін, пароль, роль) та таблицю існуючих облікових записів. Адміністратор може змінити пароль (через модальне вікно), активувати або деактивувати користувача, а також видалити обліковий запис за винятком системного облікового запису admin.

3.3. Тестування програмного комплексу

Коректність і надійність системи перевірено кількома рівнями. Це узгоджено з тим, як працює хаос-проксі: метрики з'являються лише тоді, коли через порт входу проходять справжні HTTP-запити. Результати перевірок оформлено у вигляді тест-кейсів і зведених таблиць, що відповідає вимогам до тестової документації IEEE 829 [34].

Програмний інтерфейс панелі керування - автоматично згенеровану документацію REST API у переглядачі Swagger (рис. 3.9). На ній видно перелік операцій, методи запитів і параметри. Це підтверджує, що інтерфейс реалізовано, і полегшує подальші перевірки.

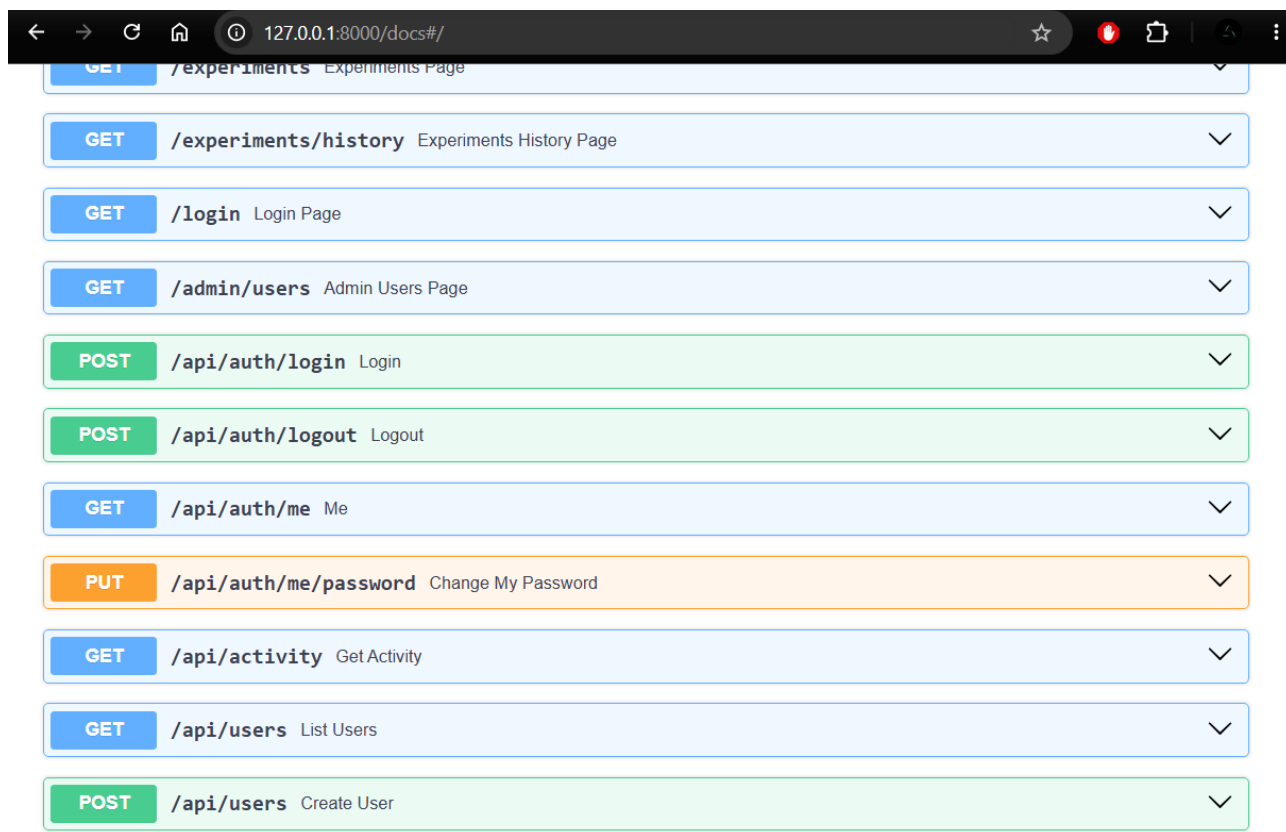


Рис. 3.9. Документація REST API (Swagger UI)

Далі виконано автоматизовані інтеграційні тести - п'ятнадцять перевірок REST API засобами pytest. Фреймворк pytest [26] дозволяє ізолювати тестове середовище, автоматично створювати тимчасову базу даних і швидко повторювати перевірки після змін у коді. Тести звертаються до застосунку без окремого запуску мережевого сервера, що прискорює перевірку. Перед кожним тестом створюється тимчасова база даних, після тесту — видаляється. Хаос-проксі під час цих тестів не запускається: пересилання трафіку та хаос-ефекти перевіряли окремо на демо-стенді. Перевіряли також роботу під обліковими записами адміністратора та звичайного користувача.

Усі п'ятнадцять тестів пройшли успішно (рис. 3.10). Перевірено вхід у систему, захист сторінок панелі, додавання й видалення сервісів, запуск хаос-експериментів, журнал активності, експорт даних, фільтрацію метрик за періодом, видалення сервісу разом із пов'язаними записами та розмежування прав доступу. Деталі — у табл. 3.2.

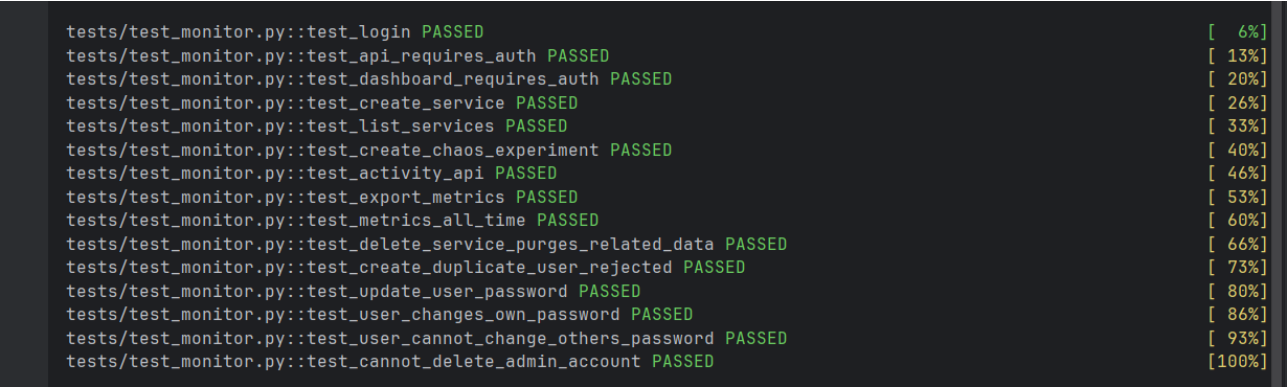


Рис. 3.10. Результат виконання автоматизованих тестів

Таблиця 3.2

Результати інтеграційного автоматичного тестування

ID	Назва тест-кейсу	Опис перевірки	Результат
ТС-01	test_login	POST /api/auth/login — видача access_token	PASS
ТС-02	test_api_requires_auth	GET /api/services без токена — 401	PASS
ТС-03	test_dashboard_requires_auth	GET / — редірект 302 на /login	PASS

продовж. табл. 3.2

TC-04	test_create_service	POST /api/services — створення сервісу з адресою входу та виходу	PASS
TC-05	test_list_services	GET /api/services — непорожній список	PASS
TC-06	test_create_chaos_experiment	POST /api/experiments — тип network_delay	PASS
TC-07	test_activity_api	GET /api/activity — структура журналу	PASS
TC-08	test_export_metrics	GET /api/export/metrics/{id} — експорт CSV	PASS

продовж. табл. 3.2

TC-09	test_metrics_all_time	?hours=12 vs ?all=true — фільтрація діапазону	PASS
TC-10	test_delete_service_purges_related_data	DELETE сервісу — очищення Metric, Experiment, Incident, AlertLog	PASS
TC-11	test_create_duplicate_user_rejected	Повторний POST /api/users — 400	PASS
TC-12	test_update_user_password	Admin змінює пароль user — новий логін OK	PASS
TC-13	test_user_changes_own_password	PUT /api/auth/me/password	PASS
TC-14	test_user_cannot_change_others_password	User не може змінити пароль admin — 403	PASS
TC-15	test_cannot_delete_admin_account	DELETE admin — 400	PASS

Функціональне тестування проведено на демо-стенді, близькому до реальної роботи. HTTP-клієнт надсилав запити на порт входу хаос-проксі, проксі за потреби застосовував хаос і пересилав запити на імітований бекенд. результати переглядали в панелі керування.

Проведено запуск панелі керування, демо-бекенд і демо-клієнт. Сервіс «Demo Backend» зареєстрували вручну в розділі «Сервіси»: вхід - порт 8002, вихід - порт 8080. Сім сценаріїв виконано успішно (табл. 3.3): базовий трафік без

		Горшков С.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Талавер О.В.				66
Змн.	Арк.	№ докум.	Підпис	Дата		

хаосу, затримка мережі, зупинка експерименту, імітація втрат у мережі, нестабільна поведінка бекенду, звіти та аналітика, перенаправлення неавторизованого користувача на сторінку входу.

Таблиця 3.3

Результати функціонального тестування

Сценарій	Вхідні умови	Очікуваний результат	Фактичний результат	Статус
FS-01. Базовий трафік	Демо клієнт - Ping / Серія ×10 через :8002, без хаос	Записи ОК у «Активність», низька затримка, хаос = ні	Журнал заповнюється, статус ОК	PASS
FS-02. Network Delay	Експеримент network_delay, delay 800 ms, клієнт Ping	Затримка зростає, хаос = так	ms помітно вищі, лічильник активних хаос = 1	PASS
FS-03. Зупинка хаосу	Зупинити експеримент після FS-02	Метрики повертаються до базової лінії	Після зупинки — знову ОК, хаос = ні	PASS
FS-04. Packet Loss / мережеві збої	Сценарій «Проблеми з мережею» або packet_loss	Частина запитів FAIL з текстом помилки	У журналі з'являються FAIL	PASS
FS-05. Нестабільний бекенд	Демо клієнт → «Нестабільний» (~35 % 503)	Змішані ОК/FAIL	Частка FAIL відповідає поведінці бекенду	PASS
FS-06. Звіти та аналітика	«Звіти», діапазон 12 год / за весь період	Графік Plotly, блок «Вплив хаос», SLO	Дані відображаються після накопичення метрик	PASS
FS-07. Авторизація	Доступ до / без login	Редірект на /login	302 → /login	PASS

Після змін у коді тести pytest проведено повторно. Повторне тестування після кожної ітерації розробки відповідає практиці безперервної перевірки інкрементів у Scrum [32]. На демо-стенді додатково спостерігали поведінку під

серіями запитів: затримка зростала під час активного хаос-експерименту і зменшувалась після його зупинки.

Виявлені обмеження. Без HTTP-трафіку через порт входу журнал активності порожній — окремого фонового пінгу немає. При великому навантаженні файлова база SQLite може тимчасово блокуватися, для промислового використання доцільна серверна СУБД. Старі метрики автоматично не видаляються — файл бази з часом зростає. Кожен сервіс обслуговується окремим процесом проксі на своєму порту. При багатьох сервісах зростає навантаження на пам'ять і кількість портів. Автоматизовані тести не перевіряють живий хаос-проксі - це робиться на демо-стенді. Звернення до бекенду по HTTPS із внутрішніми сертифікатами потребує додаткового налаштування. Для навчального стенду ці обмеження не є критичними; вони визначають напрямки подальшого розвитку системи.

Висновки до розділу 3

У третьому розділі описано розгортання, користувацький інтерфейс та перевірку працездатності розробленої системи Chaos Proxy. Показано, як система запускається в локальному середовищі, як оператор працює з панеллю керування та як підтверджено коректність основних функцій на автоматизованих і ручних перевірках.

У першому підрозділі наведено системні вимоги, порядок встановлення та запуску. Описано використання віртуального середовища Python, файлу конфігурації .env і централізоване задання параметрів. При першому запуску створюється файл бази даних SQLite із шести таблиць, для кожного зареєстрованого сервісу піднімається окремий хаос-проксі на своєму порту входу. Наведено схему розгортання компонентів системи.

У другому підрозділі розглянуто клієнтську частину- багатосторінковий вебінтерфейс панелі керування. Описано сторінки входу, журналу активності, керування сервісами, хаос-експериментів, історії експериментів, звітів і (для адміністратора) облікових записів користувачів. Підтверджено, що інтерфейс

		Горшков С.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Талавер О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		68

охоплює повний цикл роботи оператора: від спостереження за HTTP-трафіком через проксі до запуску хаос-експериментів і аналізу їхнього впливу.

У третьому підрозділі виконано тестування програмного комплексу. Переглянуто документацію REST API у Swagger UI. П'ятнадцять автоматизованих інтеграційних тестів перевірили роботу API панелі керування, сім функціональних сценаріїв на демо-стенді. Успішно перевірено авторизацію, керування сервісами, хаос-експерименти, журнал активності, експорт і аналітику, а також розмежування прав адміністратора та звичайного користувача.

У четвертому підрозділі зафіксовано обмеження поточної версії: метрики формуються лише при реальному трафіку через порт входу, SQLite обмежує масштабування при високому навантаженні, автоматизовані тести не замінюють перевірку живого хаос-проксі на демо-стенді. Ці обмеження не заважають навчально-демонстраційному використанню, але визначають напрямки подальшого вдосконалення системи.

		Горшков С.В.			ІПЗ.КР.Б-121-26-ІПЗ	Арк.
		Талавер О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		69

ВИСНОВКИ

У межах випускної кваліфікаційної роботи досягнуто поставленої мети: спроектовано та реалізовано програмний комплекс Chaos Proxy - веборієнтовану систему тестування відмовостійкості HTTP-сервісів на принципах Хаос інженерії. Система забезпечує контрольоване внесення збоїв у реальний HTTP-трафік між клієнтом і бекендом, збір метрик на шляху запиту через хаос-проксі та аналітику впливу хаос-експериментів. Рішення орієнтоване на малі та середні команди, яким потрібен легковаговий інструмент без Kubernetes, хмарних агентів і складної інфраструктурної підготовки.

Проаналізовано предметну галузь Хаос інженерії та виконано порівняльний огляд існуючих рішень. Встановлено, що корпоративні платформи орієнтовані на великі середовища з оркестрацією та мають високий поріг входження. Це обґрунтувало доцільність розробки власного локального хаос-проксі: клієнт надсилає запити на порт входу, СР за потреби робить збої, пересилає запит на URL виходу (бекенд) і фіксує результати в SQLite. Так продемонстровано застосування методології хаос інженерії без прив'язки до конкретної платформи розгортання.

Спроектовано об'єктно-орієнтовану структуру системи та модель даних із шести таблиць. Центральною сутністю є конфігурація сервісу з парою порт входу - URL виходу. Формалізовано динамічну поведінку ключових процесів засобами UML. Отримані рішення втілено у програмному коді на базі Python 3.10+, FastAPI, SQLAlchemy, httpx, Jinja2, Bootstrap та vanilla JavaScript. Реалізовано панель керування з JWT-авторизацією та розмежуванням прав, модуль хаос-ін'єкції з підтримкою затримок, втрат з'єднання, таймаутів, випадкових помилок, burst-навантаження та composite-сценаріїв, окремі точки входу СР для кожного сервісу, запис метрик і підсистему сповіщень за порогоми.

Систему розгорнуто у локальному середовищі та підтверджено її працездатність багаторівневим тестуванням: п'ятнадцять автоматизованих інтеграційних тестів pytest і сім функціональних сценаріїв на демо-стенді.

Випробування підтвердили коректність авторизації, CRUD сервісів, хаос-

		Горшков С.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Талавер О.В.				70
Змн.	Арк.	№ докум.	Підпис	Дата		

експериментів, експорту даних, каскадного видалення та RBAC. На демо-стенді система стабільно обробляє безперервний потік запитів через проксі.

Розроблений комплекс дає командам доступний інструмент перевірки стійкості сервісів «на шляху запиту»: реєстрація сервісу, генерація трафіку на порт входу, запуск хаос, перегляд журналу та звітів, попередження при деградації метрик. Модульна архітектура (панель керування, СР, хаос-ін'єкція, збір метрик) створює підґрунтя для подальшого розширення - зокрема розкладу експериментів, інтеграції зовнішніх каналів сповіщень, оновлення журналу в реальному часі та переходу на серверну СУБД при зростанні навантаження.

		Горшков С.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Талавер О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		71

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. DORA. Accelerate: State of DevOps Report 2023 [Електронний ресурс]. Google LLC, 2023. 96 p. URL: <https://dora.dev/research/2023/dora-report/> (дата звернення: 10.06.2026).
2. CNCF. CNCF Annual Survey 2023 Results [Електронний ресурс]. Cloud Native Computing Foundation, 2024. URL: <https://www.cncf.io/reports/cncf-annual-survey-2023/> (дата звернення: 10.06.2026).
3. Brewer E. A. Towards Robust Distributed Systems // Proceedings of the 19th Annual ACM Symposium on Principles of Distributed Computing (PODC). ACM, 2000. P. 7.
4. Netflix Technology Blog. Chaos Engineering Upgraded [Електронний ресурс]. 2018. URL: <https://netflixtechblog.com/chaos-engineering-upgraded-878d341f15fa> (дата звернення: 10.06.2026).
5. Gremlin Inc. Chaos Engineering Platform Documentation [Електронний ресурс]. 2024. URL: <https://www.gremlin.com/docs/> (дата звернення: 10.06.2026).
6. LitmusChaos. LitmusChaos Documentation [Електронний ресурс]. CNCF, 2024. URL: <https://docs.litmuschaos.io/> (дата звернення: 10.06.2026).
7. Uptime Robot. How Uptime Monitoring Works [Електронний ресурс]. 2024. URL: <https://uptimerobot.com/blog/> (дата звернення: 11.06.2026).
8. TechEmpower. Framework Benchmarks Round 22 [Електронний ресурс]. 2023. URL: <https://www.techempower.com/benchmarks/> (дата звернення: 11.06.2026).
9. Basiri A., Behnam N., de Rooij R., Khatib L., Laskovski M., Qin J., Weber K. Chaos Engineering. IEEE Software. 2016. Vol. 33, No 3. P. 35–41. DOI: 10.1109/MS.2016.47.
10. Nygard M. T. Release It! Design and Deploy Production-Ready Software. 2nd ed. Raleigh: Pragmatic Bookshelf, 2018. 376 p.
11. Richardson C. Microservices Patterns: With examples in Java. Shelter Island: Manning Publications, 2018. 520 p.

		Горишков С.В.			ІПЗ.КР.Б-121-26-ІЗ	Арк.
		Талавер О.В.				72
Змн.	Арк.	№ докум.	Підпис	Дата		

12. Ramírez S. FastAPI Documentation [Електронний ресурс]. 2024. URL: <https://fastapi.tiangolo.com/> (дата звернення: 13.06.2026).
13. SQLAlchemy Development Team. SQLAlchemy Documentation [Електронний ресурс]. 2024. URL: <https://docs.sqlalchemy.org/> (дата звернення: 13.06.2026).
14. Pydantic Development Team. Pydantic Documentation [Електронний ресурс]. 2024. URL: <https://docs.pydantic.dev/> (дата звернення: 13.06.2026).
15. Encode OSS. HTTPX Documentation [Електронний ресурс]. 2024. URL: <https://www.python-httpx.org/> (дата звернення: 13.06.2026).
16. Plotly Technologies Inc. Plotly JavaScript Open Source Graphing Library [Електронний ресурс]. 2024. URL: <https://plotly.com/javascript/> (дата звернення: 14.06.2026).
17. Bootstrap Team. Bootstrap 5 Documentation [Електронний ресурс]. 2024. URL: <https://getbootstrap.com/docs/5.3/> (дата звернення: 14.06.2026).
18. Pallets Projects. Jinja2 Documentation [Електронний ресурс]. 2024. URL: <https://jinja.palletsprojects.com/> (дата звернення: 14.06.2026).
19. IETF. RFC 7519: JSON Web Token (JWT) [Електронний ресурс]. 2015. URL: <https://datatracker.ietf.org/doc/html/rfc7519> (дата звернення: 14.06.2026).
20. Rosenthal C., Jones N., Wilkins N. Chaos Engineering: System Resiliency in Practice. Sebastopol: O'Reilly Media, 2020. 278 p.
21. Limoncelli T. A., Hogan C. J., Chalup S. R. The Practice of Cloud System Administration. Upper Saddle River: Addison-Wesley, 2014. 512 p.
22. Kim G., Humble J., Debois P., Willis J. The DevOps Handbook. Portland: IT Revolution Press, 2016. 480 p.
23. Fowler M. Microservices [Електронний ресурс]. 2014. URL: <https://martinfowler.com/articles/microservices.html> (дата звернення: 14.06.2026).
24. Fielding R. T. Architectural Styles and the Design of Network-based Software Architectures: Dissertation. Irvine: University of California, 2000.
25. SQLite Consortium. About SQLite [Електронний ресурс]. URL: <https://www.sqlite.org/about.html> (дата звернення: 14.06.2026).

		Горшков С.В.			ІПЗ.КР.Б-121-26-ІЗ	Арк.
		Талавер О.В.				73
Змн.	Арк.	№ докум.	Підпис	Дата		

26. Pytest Development Team. pytest Documentation [Електронний ресурс]. 2024. URL: <https://docs.pytest.org/en/latest/> (дата звернення: 16.06.2026).

27. Stenberg J. Chaos Engineering: The History, Principles, and Practice [Електронний ресурс]. InfoQ, 2017. URL: <https://www.infoq.com/articles/chaos-engineering-introduction/> (дата звернення: 11.06.2026).

28. Hochstein L., Rosenthal C. Learning Chaos Engineering. Sebastopol: O'Reilly Media, 2019. 200 p.

29. Beyer B., Jones C., Petoff J., Murphy N. R. Site Reliability Engineering: How Google Runs Production Systems. Sebastopol: O'Reilly Media, 2016. 550 p. URL: <https://sre.google/sre-book/table-of-contents/> (дата звернення: 14.06.2026).

30. Atlassian. Incident Management Best Practices [Електронний ресурс]. 2023. URL: <https://www.atlassian.com/incident-management> (дата звернення: 13.06.2026).

31. Encode OSS. Uvicorn Documentation [Електронний ресурс]. 2024. URL: <https://www.uvicorn.org/> (дата звернення: 13.06.2026).

32. Schwaber K., Sutherland J. The Scrum Guide [Електронний ресурс]. 2020. URL: <https://scrumguides.org/> (дата звернення: 14.06.2026).

33. ISO/IEC 25010:2011. Systems and Software Engineering — Systems and Software Quality Requirements and Evaluation (SQuaRE). Geneva: ISO, 2011.

34. IEEE Std 829-2008. IEEE Standard for Software and System Test Documentation. New York: IEEE, 2008.

		Горшков С.В.			ІПЗ.КР.Б-121-26-ІПЗ	Арк.
		Талавер О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		74

ДОДАТКИ

		Горшков С.В.			ІПЗ.КР.Б-121-26-ІПЗ	Арк.
		Талавер О.В.				75
Змн.	Арк.	№ докум.	Підпис	Дата		

ЛІСТИНГ ВИХІДНОГО КОДУ

Лістинг models.py:

```

class Service(Base):
    __tablename__ = "services"

    id = Column(Integer, primary_key=True, index=True)
    name = Column(String(255), unique=True, nullable=False)
    url = Column(String(500), nullable=False)
    entry_port = Column(Integer, nullable=False, default=8001)
    description = Column(Text, nullable=True)
    is_active = Column(Boolean, default=True)
    created_at = Column(DateTime, default=datetime.utcnow)
    expected_status_code = Column(Integer, default=200)

class ChaosExperiment(Base):
    __tablename__ = "chaos_experiments"

    id = Column(Integer, primary_key=True, index=True)
    service_id = Column(Integer, nullable=False, index=True)
    name = Column(String(255), nullable=False)
    type = Column(String(50), nullable=False)
    config = Column(Text, nullable=False)
    active = Column(Boolean, default=True)
    started_at = Column(DateTime, default=datetime.utcnow)
    ended_at = Column(DateTime, nullable=True)
    results_summary = Column(Text, nullable=True)

class Metric(Base):
    __tablename__ = "metrics"

    id = Column(Integer, primary_key=True, index=True)
    service_id = Column(Integer, nullable=False, index=True)
    timestamp = Column(DateTime, default=datetime.utcnow, index=True)
    latency_ms = Column(Float, nullable=True)
    status_code = Column(Integer, nullable=True)

```

		Горшков С.В.			ІІЗ.КР.Б-121-26-ІІЗ	Арк.
		Талавер О.В.				76
Змн.	Арк.	№ докум.	Підпис	Дата		

```

success = Column(Boolean, nullable=False)
error_message = Column(Text, nullable=True)
chaos_active = Column(Boolean, default=False)

```

Лістинг database.py:

```

def get_db() -> Generator[Session, None, None]:
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()

def init_db():
    Base.metadata.create_all(bind=engine)
    _migrate_service_columns()
    _migrate_experiment_columns()
    _backfill_entry_ports()
    from auth import ensure_default_users
    db = SessionLocal()
    try:
        ensure_default_users(db)
    finally:
        db.close()

```

		Горшков С.В.			ІПЗ.КР.Б-121-26-ІПЗ	Арк.
		Талавер О.В.				77
Змн.	Арк.	№ докум.	Підпис	Дата		

Лістинг вихідного коду

Лістинг chaos_injector.py:

```

@staticmethod
async def apply_chaos(experiments, make_request_func):
    start_time = time.time()
    expanded = ChaosInjector._expand_experiments(experiments)

    for exp_type, config in expanded:
        if exp_type == "latency":
            await asyncio.sleep(config.get("delay_ms", 0) / 1000.0)
        elif exp_type == "packet_loss":
            if random.random() < config.get("loss_rate", 0.1):
                return None, latency_ms, False, "Packet loss"
        elif exp_type == "connection_loss":
            if random.random() < config.get("loss_rate", 0.5):
                return None, latency_ms, False, "Connection loss"

    for exp_type, config in expanded:
        if exp_type == "outage":
            return None, latency_ms, False, "Chaos outage (503)"

    async with httpx.AsyncClient(timeout=timeout) as client:
        response = await make_request_func(client)

    for exp_type, config in expanded:
        if exp_type == "error_rate":
            if random.random() < config.get("error_rate", 0.0):
                raise Exception("Random failure")

    latency_ms = (time.time() - start_time) * 1000
    return response, latency_ms, success, None

```

Лістинг chaos_proxy.py:

```

async def forward_request(service_id: int, request: Request, path: str =
    ""):
    service = db.query(Service).filter(Service.id == service_id).first()
    if not service.is_active:

```

		Горшков С.В.			ІПЗ.КР.Б-121-26-ІПЗ	Арк.
		Талавер О.В.				78
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        raise HTTPException(status_code=503, detail="Сервіс
деактивовано")

    experiments = ChaosInjector.get_active_experiments(db, service_id)
    chaos_active = len(experiments) > 0

    response, latency_ms, _, error_msg = await ChaosInjector.apply_chaos(
        experiments, make_request
    )

    if response is None:
        await record_proxy_request(
            service_id, latency_ms, status, False, error_msg,
chaos_active
        )
        return JSONResponse(status_code=status, content={"error":
error_msg})

    success = response.status_code == (service.expected_status_code or
200)
    await record_proxy_request(
        service_id, latency_ms, response.status_code, success, err,
chaos_active
    )
    return Response(
        content=response.content,
        status_code=response.status_code,
        headers=out_headers,
    )

```

Лістинг proxy_metrics.py:

```

async def record_proxy_request(
    service_id, latency_ms, status_code, success, error_message,
chaos_active
):
    metric = Metric(
        service_id=service_id,
        timestamp=datetime.utcnow(),
        latency_ms=latency_ms,
        status_code=status_code,

```

		Горшков С.В.			ІПЗ.КР.Б-121-26-ІІЗ	Арк.
		Талавер О.В.				79
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        success=success,
        error_message=error_message,
        chaos_active=chaos_active,
    )
    db.add(metric)
    db.commit()

    if not success:
        create_incident(
            db, service_id, "proxy_request_failed",
            message=error_message or f"HTTP {status_code}",
            severity="high" if chaos_active else "medium",
        )

    alert_manager.check_and_alert(db, service_id, recent_metrics,
service_name)

```

Лістинг port_manager.py:

```

from threading import Thread

import uvicorn

from config import settings

_port_service: dict[int, int] = {}
_started: set[int] = set()

def port_for_service(service_id: int) -> int:
    return settings.CHAOS_PROXY_PORT + service_id

def register_service_port(service_id: int, port: int):
    if port in _port_service and _port_service[port] != service_id:
        raise ValueError(f"Порт {port} вже зайнятий іншим сервісом")
    _port_service[port] = service_id
    if port in _started:
        return

    if not settings.CHAOS_PROXY_ENABLED:
        return

    from chaos_proxy import build_service_app

    app = build_service_app(service_id)

    def run():
        try:
            uvicorn.run(app, host=settings.CHAOS_PROXY_HOST, port=port,

```

		Горшков С.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Талавер О.В.				80
Змн.	Арк.	№ докум.	Підпис	Дата		

```

log_level="warning")
    except SystemExit:
        pass

    Thread(target=run, daemon=True).start()
    _started.add(port)

def unregister_service_port(port: int):
    _port_service.pop(port, None)

def start_all_service_ports():
    from database import SessionLocal
    from models import Service

    db = SessionLocal()
    try:
        for s in db.query(Service).all():
            port = s.entry_port or port_for_service(s.id)
            register_service_port(s.id, port)
    finally:
        db.close()

```

		Горшков С.В.			ІПЗ.КР.Б-121-26-ІПЗ	Арк.
		Талавер О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		81

Лістинг вихідного коду

Лістинг auth.py:

```
def hash_password(password: str) -> str:
    return
    hashlib.sha256(f"{settings.SECRET_KEY}:{password}".encode()).hexdigest()

def require_admin(user: User = Depends(get_current_user)) -> User:
    if user.role != "admin":
        raise HTTPException(status_code=403, detail="Доступ лише для адміністратора")
    return user
```

Лістинг app.py вхід у систему:

```
@app.post("/api/auth/login")

def login(data: LoginRequest, db: Session = Depends(get_db)):
    user = db.query(User).filter(User.username == data.username,
    User.is_active == True).first()

    if not user or not verify_password(data.password,
    user.password_hash):
        raise HTTPException(status_code=401, detail="Невірний логін або пароль")

    token = create_token(user.username, user.role)

    response = JSONResponse({"access_token": token, "role": user.role})

    response.set_cookie(key=COOKIE_NAME, value=token, httponly=True,
    samesite="lax")

    return response
```

Лістинг app.py створення сервісу:

```
@app.post("/api/services")

def create_service(service: ServiceCreate, db: Session = Depends(get_db),
    _=Depends(require_admin)):
    port = service.entry_port or _next_entry_port(db)
```

		Горшков С.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Талавер О.В.				82
Змн.	Арк.	№ докум.	Підпис	Дата		

```

db_service = Service(
    name=service.name,
    url=normalize_target_url(service.target_url),
    entry_port=port,
    description=service.description,
)

db.add(db_service)

db.commit()

register_service_port(db_service.id, port)

return _service_dict(db_service)

```

Лістинг app.py запуск хаос-експерименту:

```

@app.post("/api/experiments")

def create_experiment(exp: ChaosExperimentCreate, db: Session =
Depends(get_db), _=Depends(require_user)):

    db_exp = ChaosExperiment(
        service_id=exp.service_id,
        name=exp.name,
        type=exp.type,
        config=json.dumps(exp.config),
        active=True,
    )

    db.add(db_exp)

    db.commit()

    if duration:
        _schedule_auto_stop(db_exp.id, duration)

    return {"id": db_exp.id, "message": f"Експеримент '{exp.name}'
запущено"}

```

Лістинг app.py журнал активності:

		Горшков С.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Талавер О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		83

```

@app.get("/api/activity")

def get_activity(limit: int = 50, service_id: Optional[int] = None, db:
Session = Depends(get_db), _=Depends(require_user)):

    q = db.query(Metric)

    if service_id:

        q = q.filter(Metric.service_id == service_id)

    metrics = q.order_by(Metric.timestamp.desc()).limit(limit).all()

    return {"items": [...], "total": total, "error_count": error_count}

```

Лістинг app.py аналітика впливу хаосу:

```

@app.get("/api/analytics/chaos-impact/{service_id}")

def chaos_impact(service_id: int, db: Session = Depends(get_db),
_ = Depends(require_user)):

    with_chaos = db.query(Metric).filter(

        Metric.service_id == service_id, Metric.chaos_active == True

    ).all()

    without_chaos = db.query(Metric).filter(

        Metric.service_id == service_id, Metric.chaos_active == False

    ).all()

    return {

        "without_chaos": calc_stats(without_chaos),

        "with_chaos": calc_stats(with_chaos),

        "impact": {

            "latency_increase_percent": ...,

            "success_rate_decrease_percent": ...,

            "is_resilient": latency_degradation < 200 and
success_degradation < 30,

        },

    }

```

		Горшков С.В.			ІПЗ.КР.Б-121-26-ІПЗ	Арк.
		Талавер О.В.				84
Змн.	Арк.	№ докум.	Підпис	Дата		