Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем та програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

Магістр
(назва освітнього ступеня)

на тему: Модернізація системи менеджменту та моніторингу проєктів на серверах з використанням технологій Python, Django, Kubernetes та LLM

Виконав(ла): студент(ка) 6 курсу, групи СПм-61
спеціальності 121 «Інженерія програмного Забезпечення»
(шифр і назва спеціальності)

_____ Лань О. П.
(підпис) (прізвище та ініціали)

Керівник _____ Мудрик І. Я.
(підпис) (прізвище та ініціали)

Нормоконтроль _____ _____
(підпис) (прізвище та ініціали)

Завідувач кафедри _____ _____
(підпис) (прізвище та ініціали)

Рецензент _____ _____
(підпис) (прізвище та ініціали)

Тернопіль 2025

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет _____
(повна назва факультету)

Кафедра _____
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри

_____ _____
(підпис) (прізвище та ініціали)

« » 20___ р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня _____
(назва освітнього ступеня)

за спеціальністю _____
(шифр і назва спеціальності)

студенту _____
(прізвище, ім'я, по батькові)

1. Тема роботи _____

Керівник роботи _____
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «___» _____ 20___ року № _____.

2. Термін подання студентом завершеної роботи _____

3. Вихідні дані до роботи _____

4. Зміст роботи (перелік питань, які потрібно розробити)

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

6. Консультанти розділів роботи

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

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка

Студент _____ _____
(підпис) (прізвище та ініціали)

Керівник роботи _____ _____
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Кваліфікаційна робота магістра. Тернопільський національний технічний університет імені Івана Пулюя, кафедра програмної інженерії, спеціальність 121 «Інженерія програмного забезпечення». ТНТУ, 2025, Сторінок 70, рисунків 31, презентація.

Тема: Модернізація системи менеджменту та моніторингу проєктів на серверах з використанням технологій Python, Django, Kubernetes та LLM.

Метою роботи є підвищення ефективності процесів адміністрування серверної інфраструктури шляхом створення автоматизованої системи менеджменту та моніторингу проєктів. Основний акцент зроблено на забезпеченні надійності розгортання, мінімізації часу реакції на інциденти та спрощенні взаємодії з середовищами завдяки інтеграції штучного інтелекту.

Об’єктом дослідження є процеси керування життєвим циклом, розгортання та моніторингу програмних застосунків у розподілених серверних середовищах. До цього кола питань входять методи автоматизації оркестрації контейнерів, алгоритми оптимізації розподілу обчислювальних ресурсів, а також механізми забезпечення відмовостійкості та діагностики інцидентів у високонавантажених інформаційних системах.

У роботі виконано комплексний аналіз предметної області та існуючих інструментів DevOps, спроєктовано архітектуру системи на базі фреймворку Django та мікросервісних принципів. Реалізовано програмні модулі для оркестрації контейнерів через API Kubernetes, розроблено підсистему збору даних у реальному часі, а також впроваджено компонент інтелектуального аналізу логів на основі великих мовних моделей (LLM). Створено інтуїтивно зрозумілий веб-інтерфейс для візуалізації метрик та керування проєктами, проведено тестування функціоналу на відповідність вимогам безпеки та продуктивності.

Ключові слова: система менеджменту проєктів, моніторинг серверів, Python, Django, веб-додаток, управління даними, безпека, продуктивність.

ABSTRACT

Master's Qualification Thesis. Ivan Puluj Ternopil National Technical University, Department of Software Engineering, Specialty 121 "Software Engineering". TNTU, 2025. 70 pages, 31figures, presentation.

Topic: Server-based systems using Python, Django, Kubernetes, and LLM technologies.

The objective of this thesis is to streamline server infrastructure administration by developing an automated platform for project management and supervision. Key emphasis is placed on guaranteeing reliable deployments, reducing incident resolution times, and facilitating smoother interaction with containerized ecosystems by leveraging modern web technologies and artificial intelligence.

The subject matter of this research covers the entire lifecycle management, deployment, and monitoring of software applications hosted in distributed server environments. This includes exploring methods for automated container orchestration, strategies for optimizing computational resource distribution, and techniques for ensuring fault tolerance and incident diagnostics within high-traffic systems.

The study comprises a thorough review of the domain and current DevOps instrumentation, followed by the architectural design of a system utilizing the Django framework and microservice patterns. The implementation phase included developing software modules for container orchestration via the Kubernetes API, creating a subsystem for real-time telemetry collection, and integrating an intelligent log analysis component powered by Large Language Models (LLM). A user-friendly web interface was built for visual metrics tracking and project control, with the system undergoing rigorous testing to meet security and performance standards.

Keywords: project management system, server monitoring, Python, Django, web application, data management, security, performance.

ЗМІСТ

ВСТУП

У сучасних реаліях масова міграція до розподілених мікросервісних архітектур та контейнеризації диктує необхідність кардинального перегляду стратегій адміністрування серверних потужностей. Класичні інструменти спостереження часто виявляються недостатньо ефективними в умовах динамічної зміни навантаження, що формує запит на інтелектуальні рішення для автоматизованої діагностики та відновлення сервісів. Головним завданням дослідження є проєктування програмного комплексу для адміністрування та нагляду за проєктами, реалізованого на базі екосистеми Python (Django), середовища Kubernetes та методів генеративного штучного інтелекту (LLM). Зазначена комбінація технологій дає змогу побудувати централізовану платформу, що об’єднує стабільний веб-інтерфейс, механізми контейнерної оркестрації та алгоритми глибокого аналізу логів для формування експертних рекомендацій щодо усунення технічних збоїв.

Впровадження розробленої системи вирішує проблему фрагментації інструментарію деплою та збору метрик, забезпечуючи наскрізну прозорість процесів DevOps. У дослідженні вирішено задачі проєктування архітектури, реалізації взаємодії з API Kubernetes та розробки алгоритмів обробки даних для нейромереж. Вагомий внесок зроблено у розробку механізмів кіберзахисту, регламентацію користувацьких повноважень та оптимізацію продуктивності системи за допомогою асинхронної обробки подій.

Практичне значення одержаних результатів визначається створенням програмного комплексу, що дозволяє інженерам автоматизувати розгортання середовищ та отримувати семантично значущі звіти про причини системних збоїв. Інтеграція методологій Agile та DevOps, посилена можливостями штучного інтелекту, уможливлює зміну парадигми адміністрування з ситуативного реагування на попередження інцидентів. Це сприяє зниженню когнітивного навантаження на персонал, мінімізації операційних ризиків та підвищенню якості програмних продуктів, що експлуатуються на серверних потужностях

1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

Аналіз вимог до модернізації системи передбачає визначення критичних метрик продуктивності та масштабування, які необхідно забезпечити шляхом переходу на архітектуру мікросервісів з використанням Django та оркестрації в Kubernetes. Окрему увагу приділено формулюванню сценаріїв інтеграції LLM, що дозволить автоматизувати обробку інцидентів та впровадити інтелектуальний предиктивний моніторинг серверних ресурсів у реальному часі.

1.1 Аналіз предметної області

Дослідження проблематики модернізації систем менеджменту та моніторингу проєктів на серверних потужностях вимагає комплексного розгляду актуальних векторів еволюції розробки програмних систем, хмарних обчислень та методів штучного інтелекту. Сучасний ландшафт інформаційних технологій характеризується стрімким переходом від монолітних архітектур до розподілених мікросервісних систем, що докорінно змінює підходи до розгортання, супроводу та діагностики програмних продуктів [1]. Традиційні методи адміністрування серверів, що базувалися на ручному налаштуванні конфігурацій та статичному моніторингу, втрачають ефективність в умовах динамічного масштабування та високої частоти оновлення коду. Це зумовлює виникнення потреби у високоавтоматизованих системах керування, здатних забезпечувати безперервність бізнес-процесів з мінімальним втручанням людини.

Ключовим елементом предметної області є технології контейнеризації та оркестрації, серед яких домінуючу позицію займає платформа Kubernetes. Вона виступає як абстракція над фізичною інфраструктурою, надаючи уніфікований інтерфейс для керування обчислювальними ресурсами. Однак впровадження Kubernetes суттєво підвищує поріг входження для інженерних команд та ускладнює процеси налагодження. Стандартні засоби візуалізації стану кластера часто надають надлишкову кількість низькорівневої технічної інформації, яка

може бути складною для інтерпретації менеджерами проєктів або розробниками, що не спеціалізуються на DevOps. Це створює попит на розробку надбудов та панелей керування, які б спрощували взаємодію з оркестратором та надавали агреговані дані про стан проєктів.

У контексті мов програмування та фреймворків для розробки керуючих систем, Python та Django займають стратегічно важливу нішу. Python є стандартом де-факто у сфері автоматизації системного адміністрування та науці про дані, що дозволяє використовувати єдину кодову базу як для бекенду веб-інтерфейсу, а також для сценаріїв автоматизації та модулів обробки даних. Фреймворк Django забезпечує швидку розробку надійних веб-застосунків з вбудованими механізмами автентифікації та адміністрування, що є критично важливим для корпоративних систем менеджменту.

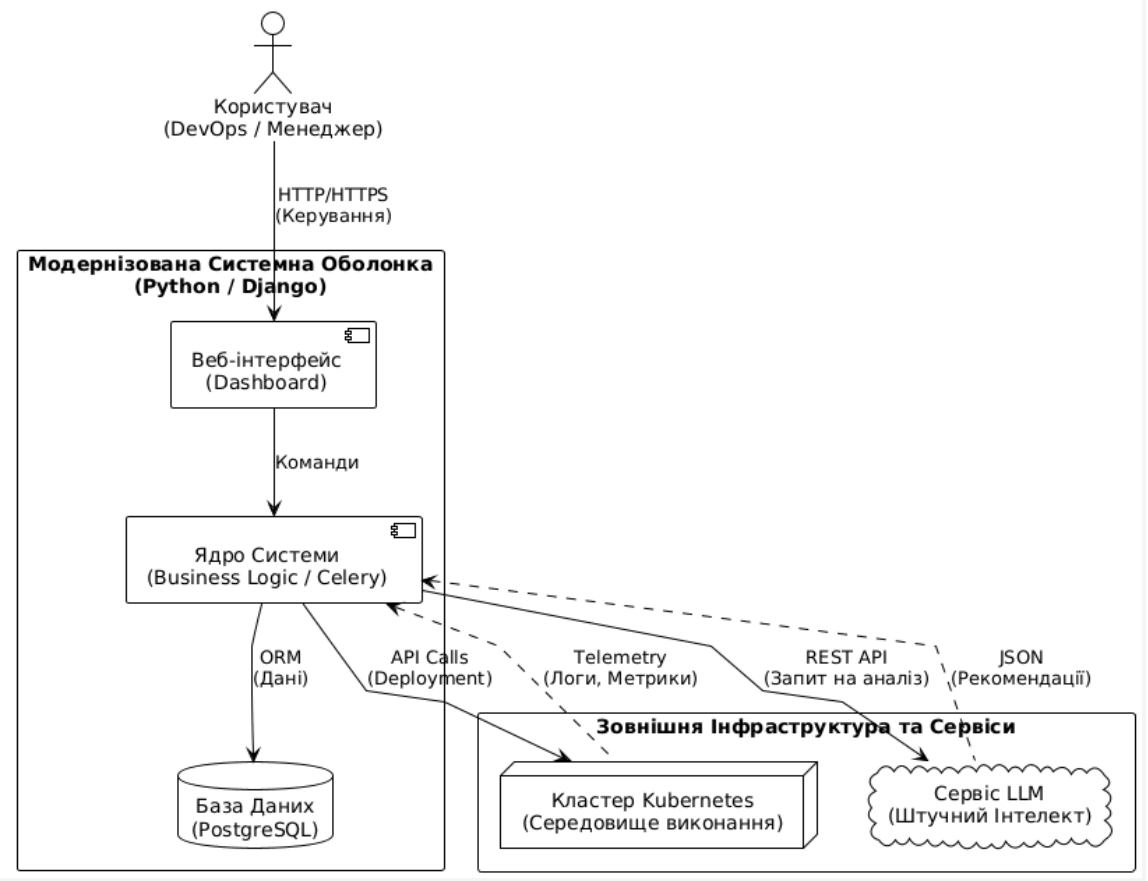


Рис. 1.1 – Концептуальна схема взаємодії компонентів у сучасних системах інтелектуального моніторингу

Окремим вектором аналізу є еволюція систем моніторингу. Зростання обсягів телеметричних даних, логів та метрик призводить до інформаційного шуму, в якому операторам складно виокремити кореневі причини збоїв. Традиційні системи моніторингу, що базуються на порогових значеннях, часто генерують велику кількість хибних сповіщень або, навпаки, пропускають аномалії, що не підпадають під статичні правила. Це актуалізує застосування підходів AIOps (Artificial Intelligence for IT Operations), що базуються на інтеграції алгоритмів машинного навчання в процеси обробки телеметрії [2].

Інтенсивна еволюція технологій великих мовних моделей (LLM) створює якісно нові перспективи у предметній області. На відміну від класичних методів машинного навчання, генеративні нейромережі здатні розуміти контекст неструктурованих текстових даних, таких як логи помилок, документація та повідомлення у чатах підтримки. Інтеграція LLM у системи моніторингу дозволяє реалізувати функціонал інтелектуального асистента, здатного не лише констатувати факт збою, але й аналізувати стек викликів, співставляти його зі змінами у коді та генерувати рекомендації щодо усунення проблеми [3]. Це змінює парадигму взаємодії користувача з системою моніторингу від пасивного споглядання графіків до активного діалогу з системою.

Таким чином, предметна область характеризується конвергенцією класичних методів керування проєктами, хмарних технологій оркестрації та новітніх розробок у галузі комп’ютерного інтелекту. До переліку критичних викликів, що потребують вирішення, є складність керування розподіленою інфраструктурою, інформаційне перевантаження систем моніторингу та дефіцит інструментів, орієнтованих на надання експертних рекомендацій для оперативного розв’язання інцидентів. Розробка системи, що поєднує можливості Django для керування, Kubernetes для виконання та LLM для аналізу, є відповіддю на актуальні запити індустрії щодо підвищення автономності та надійності програмних комплексів.

1.2 Постановка завдання та цілей

Постановка науково-прикладного завдання випливає з необхідності подолання розриву між зростаючою складністю інфраструктурних рішень на базі Kubernetes та потребою у спрощенні процедур їх адміністрування для кінцевих користувачів. В умовах експлуатації високонавантажених серверних систем виникає протиріччя між обсягами телеметричних даних, що генеруються розподіленими додатками, та здатністю операторів оперативно обробляти цю інформацію для прийняття управлінських рішень. Існуючі засоби моніторингу переважно забезпечують кількісну оцінку стану ресурсів, залишаючи якісну інтерпретацію причин збоїв на розсуд людини, що в критичних ситуаціях призводить до збільшення часу простою сервісів [4].

Рис. 1.2 – Графічна інтерпретація дисбалансу між складністю систем та можливостями ручного адміністрування

Актуальною проблемою є відсутність уніфікованих програмних комплексів, які б органічно поєднували інструментарій розгортання проєктів із засобами

інтелектуальної діагностики, реалізованими з використанням передових технологій машинного навчання [5].

Основна ціль дослідження полягає в оптимізації процедур менеджменту та моніторингу проєктів на серверних потужностях шляхом розробки модернізованої програмної системи, що базується на синергетичному використанні фреймворку Django, платформи оркестрації Kubernetes а також технологій генеративного ШІ. Реалізація поставленого завдання вимагає створення інструментарію, здатного автоматизувати рутинні операції з керування життєвим циклом контейнеризованих застосунків та забезпечити семантичний аналіз системних подій для предиктивного виявлення інцидентів.

Об’єктом дослідження виступають процеси керування розгортанням, масштабуванням та моніторингом програмного забезпечення у кластерних середовищах. Предметом дослідження є методологічні підходи та інструментарій поєднання сучасних веб-фреймворків Python, механізмів оркестрації контейнерів та алгоритмів генеративного машинного навчання, спрямованих на проєктування адаптивних систем адміністрування.

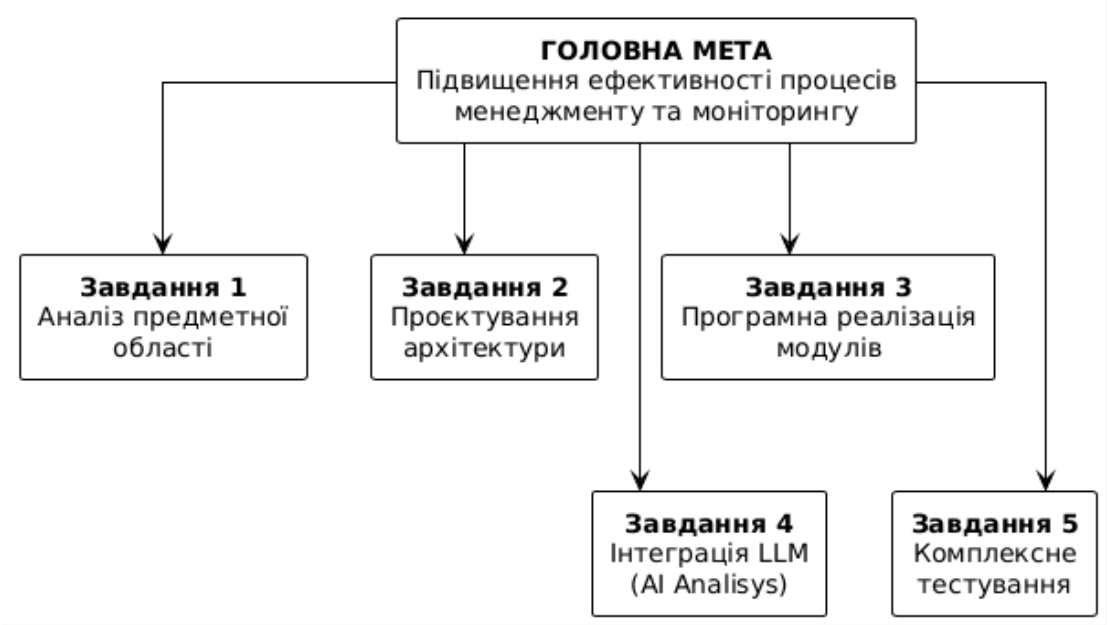


Рис. 1.3 – Декомпозиція мети дослідження на комплекс взаємопов’язаних завдань

Для досягнення сформульованої мети вимагає реалізації сукупності узгоджених кроків. Вихідним пунктом дослідження є проведення детального аналізу предметної області та існуючих підходів до організації DevOps-процесів, що дозволить виявити функціональні обмеження поточних рішень. Наступним етапом є проєктування архітектури програмної системи, яка забезпечує взаємодію між веб-інтерфейсом керування та API кластера Kubernetes, а також визначає механізми обробки даних моніторингу. Критично важливим завданням є розробка програмних модулів на мові Python з використанням Django для реалізації бізнес-логіки керування проєктами та інтерфейсу користувача. Окремим завданням виступає інтеграція великої мовної моделі в контур системи моніторингу, що передбачає розробку алгоритмів попередньої обробки логів та налаштування запитів до моделі для отримання рекомендацій щодо усунення помилок. Завершальним завданням є проведення комплексного тестування розробленої системи для перевірки її відповідності вимогам надійності, відмовостійкості та точності роботи інтелектуальних компонентів. Вирішення цих завдань дозволить створити цілісний продукт, що відповідає сучасним вимогам до систем керування серверною інфраструктурою.

1.3 Пошук акторів та варіантів використання

Процес ідентифікації акторів та визначення варіантів використання є фундаментальним етапом об’єктно-орієнтованого аналізу, який дозволяє формалізувати функціональні вимоги до програмної системи та окреслити межі її взаємодії із зовнішнім середовищем [6]. У контексті розроблюваної системи керування та моніторингу на базі технологій Python, Django, Kubernetes та LLM, під акторами розуміються не лише безпосередні користувачі, але й зовнішні програмні сутності, що ініціюють або беруть участь у сценаріях роботи системи. Аналіз предметної області та бізнес-процесів розробки програмного забезпечення дозволяє виділити три основні групи користувачів, які мають розмежовані права доступу та специфічні сценарії взаємодії з інтерфейсом.

Першим та ключовим актором виступає Адміністратор системи, роль якого передбачає повний контроль над конфігурацією програмного комплексу та інфраструктурними ресурсами. Функціональний профіль адміністратора включає налаштування підключення до кластерів Kubernetes, керування правами доступу інших користувачів, визначення квот на використання обчислювальних ресурсів та моніторинг загального стану здоров'я системи. Саме цей актор відповідає за інтеграцію з зовнішніми сервісами, включаючи налаштування API-ключів для доступу до великих мовних моделей. Варіанти використання для даної ролі зосереджені на забезпеченні стабільності платформи та безпеки даних, що вимагає наявності інструментів для перегляду аудиту дій та конфігурування глобальних параметрів моніторингу.

Другим значущим актором є Розробник, діяльність якого спрямована на безпосереднє керування життєвим циклом конкретних проєктів та застосунків. До сфери відповідальності розробника входить ініціація процесів розгортання програмного забезпечення, оновлення версій контейнерів, перегляд логів та діагностика помилок. Специфікою даної ролі в контексті модернізованої системи є активна взаємодія з підсистемою штучного інтелекту. Варіанти використання для розробника включають відправку запитів на аналіз лог-файлів за допомогою LLM, отримання автоматизованих рекомендацій щодо оптимізації коду або конфігурації, а також виконання команд масштабування подів у межах виділених лімітів. Цей актор потребує деталізованої інформації про роботу своїх сервісів, проте не має доступу до глобальних налаштувань кластера.

Третім актором є Менеджер проєкту, чиї інтереси лежать у площині спостереження за статусом виконання завдань та використанням ресурсів без втручання у технічні аспекти реалізації. Варіанти використання для менеджера обмежуються переглядом аналітичних дашбордів, отриманням звітів про доступність сервісів (uptime) та моніторингом споживання ресурсів для оцінки вартості інфраструктури. Ця роль вимагає спрощеного інтерфейсу візуалізації даних, який дозволяє швидко оцінити стан проєкту. Окрім людських ролей, доцільно виділити системного актора — Планувальник Kubernetes, який

автоматично взаємодіє з системою для передачі метрик та статусів подів, що ініціює внутрішні сценарії оновлення бази даних та генерації сповіщень.

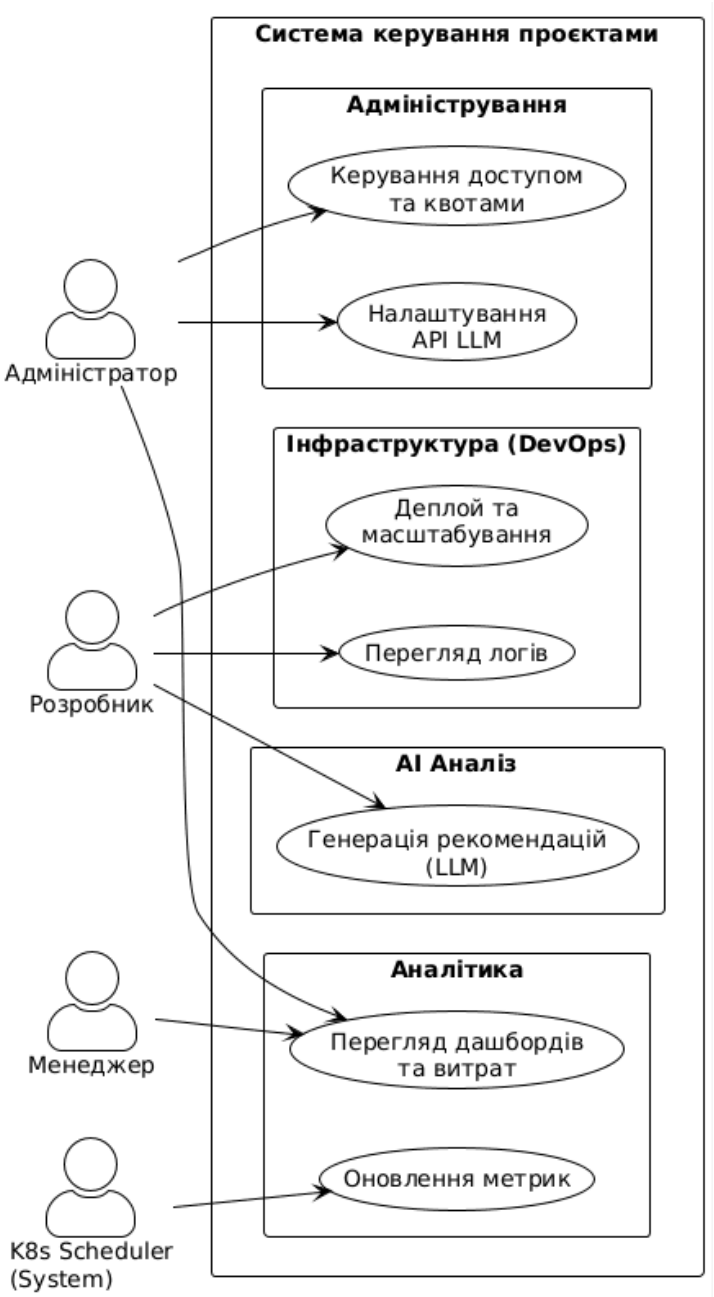


Рис. 1.4 – Діаграма варіантів використання системи з розподілом ролей акторів

Сукупність виявлених варіантів використання можна класифікувати за функціональними доменами. Домен автентифікації та авторизації охоплює сценарії реєстрації, входу в систему та розподілу ролей. Домен керування

інфраструктурою включає створення нових проєктів, налаштування змінних середовища, деплой образів контейнерів та керування репліками. Домен моніторингу та діагностики об’єднує сценарії збору метрик, візуалізації графіків навантаження CPU та RAM, агрегації логів у реальному часі.

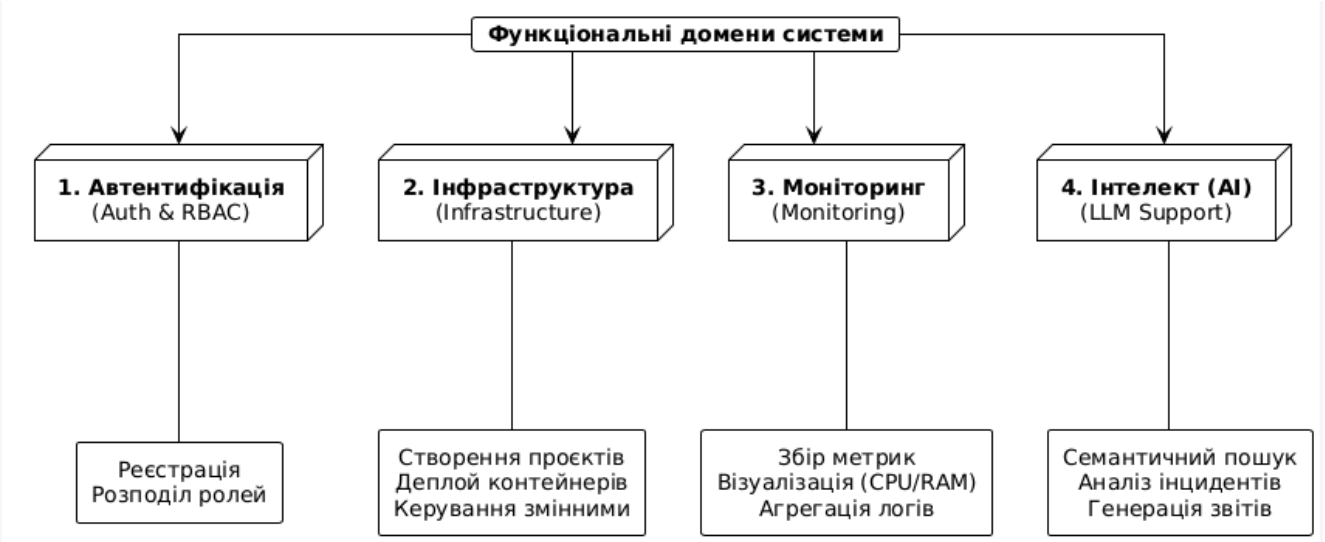


Рис. 1.5 – Класифікація функціональних можливостей системи за доменами

Найбільш інноваційним є домен інтелектуальної підтримки, який містить сценарії семантичного пошуку по базі знань інцидентів та автоматичної генерації звітів про причини збоїв на основі висновків LLM [7].

1.4 Опис ключових варіантів використання

Деталізація ключових варіантів використання є необхідною умовою для формалізації поведінкових аспектів проєктованої системи та визначення чітких алгоритмів взаємодії між користувачами та програмними компонентами. Опис сценаріїв базується на специфікаціях, сформульованих на етапі аналізу предметної області, та враховує архітектурні особливості інтеграції веб-фреймворку Django з середовищем оркестрації Kubernetes. Серед множини можливих прецедентів доцільно виокремити ті, які відіграють вирішальну роль у реалізації ключового функціоналу системи: розгортання сервісів, моніторинг ресурсів у режимі

поточного часу, а також автоматизована діагностика несправностей. Кожен із цих сценаріїв описує послідовність дій, що ініціюються актором, та відповідні реакції системи, включаючи звернення до локальних сховищ даних і зовнішніх API.

Фундаментальним сценарієм роботи системи є створення та розгортання нового програмного проєкту в кластері. Цей процес ініціюється актором із правами розробника або адміністратора через веб-інтерфейс. Користувач заповнює форму специфікації, вказуючи посилання на репозиторій образу контейнера, необхідні змінні середовища, обмеження ресурсів процесора та оперативної пам’яті, а також параметри мережевого доступу. Система виконує валідацію введених даних на відповідність синтаксичним правилам та наявним квотам користувача.

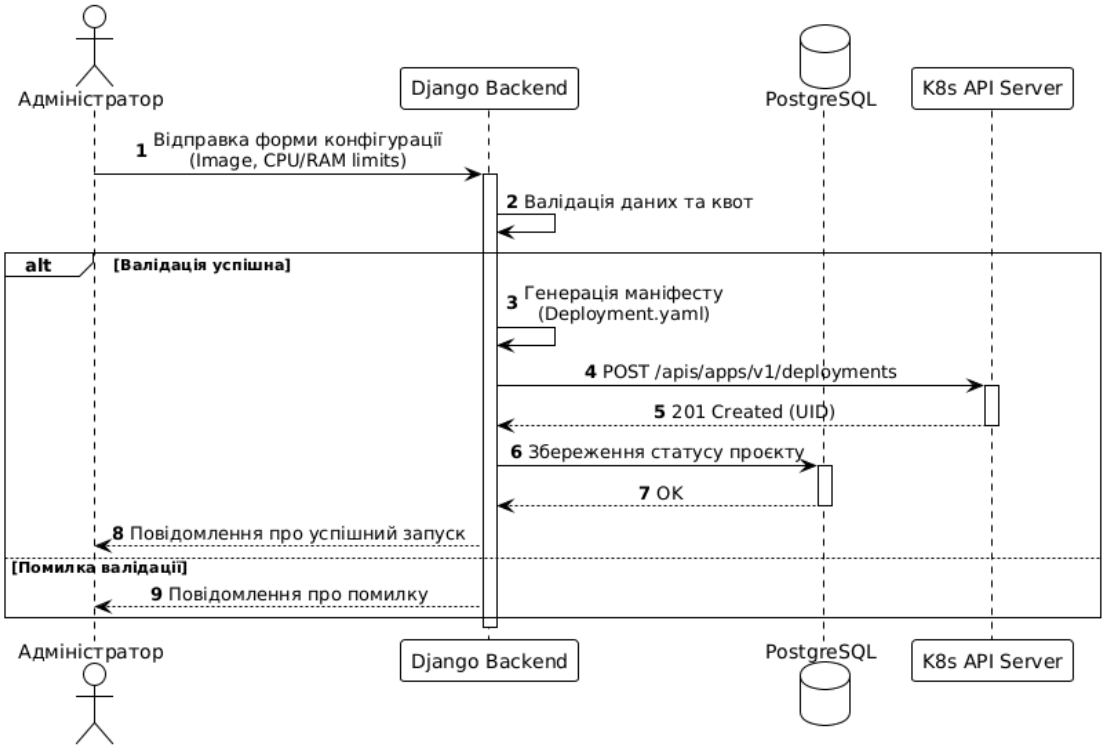


Рис. 1.6 – Діаграма послідовності створення нового розгортання у кластері

Другим ключовим варіантом використання є моніторинг стану системи та візуалізація метрик, який забезпечує прозорість функціонування інфраструктури. Сценарій передбачає безперервний збір телеметричних даних від агентів моніторингу (cAdvisor або Metrics Server), розташованих у кластері Kubernetes, та їх подальшу агрегацію. Користувач, переходячи на сторінку деталізації проєкту,

ініціює запит на отримання історичних та поточних даних про навантаження. Для запобігання блокуванню основного потоку веб-інтерфейсу, система використовує асинхронні виклики: модуль бекенду звертається до API кластера, отримує "сирі" масиви даних про використання процесорного часу (в millicores) та оперативної пам’яті (в мегабайтах), нормалізує їх та передає на фронтенд у форматі JSON [8].

На основі отриманих даних клієнтська частина застосунку генерує інтерактивні графіки, що дозволяють масштабувати часові інтервали для детального аналізу пікових навантажень. Важливим технічним аспектом цього сценарію є мінімізація затримки (latency) при оновленні даних, що досягається шляхом оптимізації запитів та використання механізмів кешування. Цей прецедент також включає механізм автоматичного оновлення статусу доступності сервісів (Health check), що реалізується через періодичні запити системи до налаштованих кінцевих точок здоров’я (Liveness та Readiness probes). У разі відсутності відповіді від сервісу, система автоматично змінює його статус в інтерфейсі та може ініціювати відправку сповіщення адміністратору.

Найбільш інноваційним сценарієм, що суттєво відрізняє розроблювану систему від традиційних адміністративних панелей, є інтелектуальний аналіз інцидентів із залученням великої мовної моделі (LLM). Цей варіант використання активується у двох випадках: автоматично при дететуванні критичної помилки (CrashLoopBackOff, OOMKilled) або за прямим запитом користувача через інтерфейс перегляду логів. Процес розпочинається з вибірки останніх записів журналювання (наприклад, останніх 50-100 рядків) з проблемного контейнера.

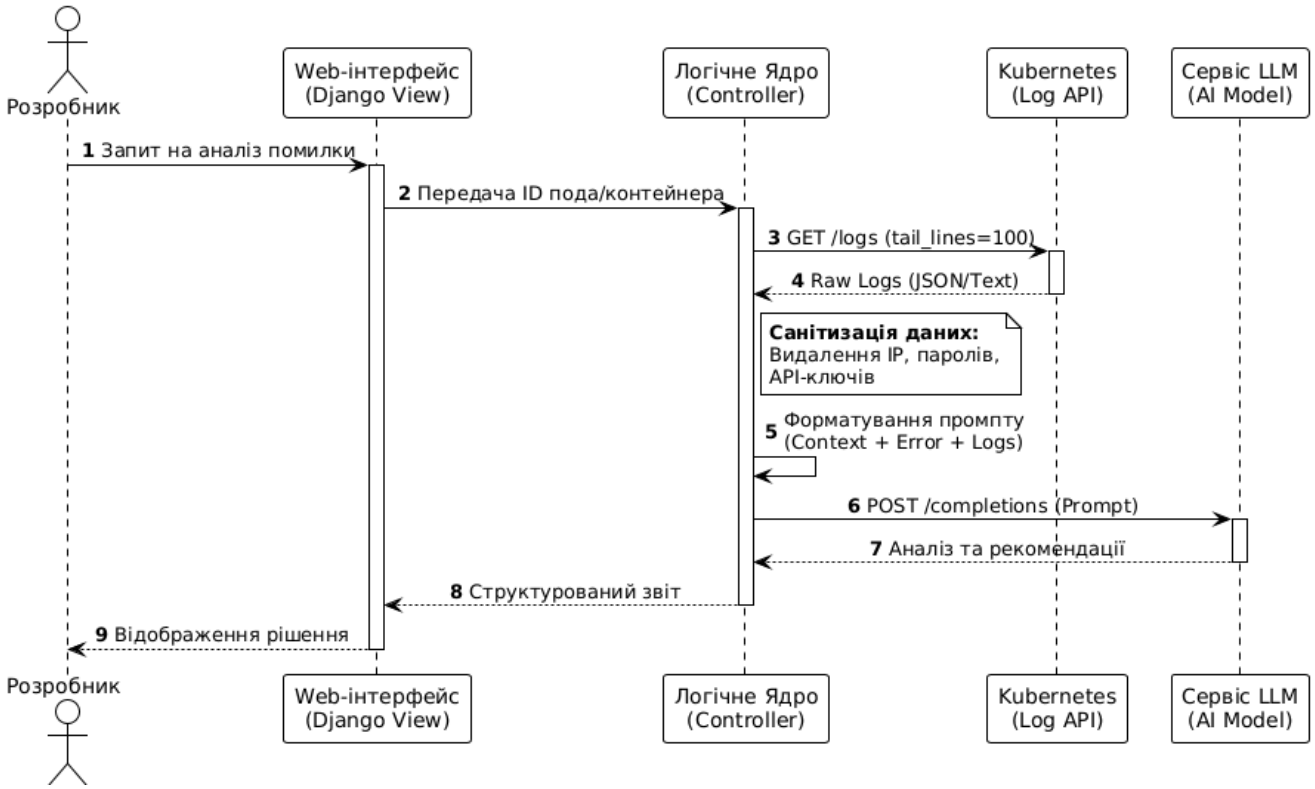


Рис. 1.7 – Діаграма послідовності інтелектуального аналізу інцидентів

Критично важливим етапом є попередня обробка тексту (санітизація): система за допомогою регулярних виразів видаляє конфіденційні дані, такі як IP-адреси, токени авторизації та паролі бази даних, щоб уникнути витоку приватної інформації при передачі даних зовнішньому провайдеру LLM [9]. Далі формується структурований контекстний запит (промпт), який включає опис коду помилки, очищені логи та інструкцію для моделі. Отримана відповідь, що містить пояснення причини збою, потенційні шляхи виправлення та, за можливості, виправлені фрагменти коду або конфігурації, інтерпретується системою та відображається користувачеві. Для підвищення економічної ефективності та швидкодії реалізовано механізм кешування відповідей: якщо система зустрічає ідентичну помилку (співпадає хеш логів), вона миттєво видає збережену рекомендацію без повторного звернення до платного API моделі. Таким чином, описані сценарії покривають повний життєвий цикл керування сервісом, трансформуючи рутинні операції адміністрування в автоматизовані інтелектуальні процеси.

2 ПРОЄКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОЇ СИСТЕМИ

Проєктування архітектури базується на мікросервісному підході, де кластер Kubernetes забезпечує оркестрацію контейнеризованих Django-сервісів для гарантування високої доступності та динамічного масштабованості системи. Етап розробки включає імплементацію бекенд-логіки мовою Python та глибоку інтеграцію моделей LLM, що дозволяє реалізувати функціонал інтелектуального аналізу логів і автоматизованого управління інцидентами без участі оператора.

2.1 Вибір процесу розробки

Визначення методології розробки програмного забезпечення є фундаментальним етапом проєктування, оскільки від цього залежить надійність створеного рішення, відповідність часових рамок та ефективність розподілу ресурсів. Специфіка досліджуваної системи, яка інтегрує веб-технології, засоби оркестрації контейнерів та компоненти штучного інтелекту, вимагає застосування гнучких підходів, здатних адаптуватися до динамічних змін у вимогах та технологічному стеку. Класичний послідовний підхід (Waterfall), який передбачає послідовне виконання етапів без можливості повернення на попередні стадії, в даному контексті визнана неефективною [10]. Це зумовлено високим ступенем невизначеності, пов’язаним з експериментальним характером інтеграції великих мовних моделей, поведінка яких потребує багаторазового тестування та налаштування.

Враховуючи необхідність поетапної реалізації функціоналу та частої перевірки гіпотез, як базовий процес розробки обрано ітеративно-інкрементальний підхід, реалізований у рамках методології Agile. Зокрема, доцільним є використання фреймворку Scrum, який структурує процес розробки на короткі часові інтервали — спринти [11]. Застосування цієї стратегії уможливлює розбиття складної архітектури системи на окремі незалежні модулі, такі як підсистема автентифікації, модуль взаємодії з Kubernetes API та інтерфейс

чат-бота, і розробляти їх паралельно або послідовно з регулярною інтеграцією. Кожна ітерація завершується отриманням потенційно готового до використання інкременту продукту, що дозволяє проводити проміжне тестування та оперативно виявляти архітектурні недоліки ще до завершення повного циклу розробки.

Рис. 2.1 – Схема ітеративно-інкрементального процесу розробки за методологією Scrum

Вибір гнучкої методології також обумовлений особливостями використовуваного технологічного стека. Фреймворк Django, що лежить в основі серверної частини, спроєктований з урахуванням принципів швидкої розробки (RAD), що органічно поєднується з філософією Agile. Використання мови Python дозволяє швидко створювати прототипи алгоритмів обробки даних, а платформа Kubernetes забезпечує можливість безперервної доставки та розгортання оновлень (CI/CD). Впровадження практик DevOps, які є невід'ємною частиною сучасних гнучких методологій, дозволяє автоматизувати процеси збірки контейнерів та їх деплою у тестове середовище, що пришвидшує цикл зворотного зв'язку [12].

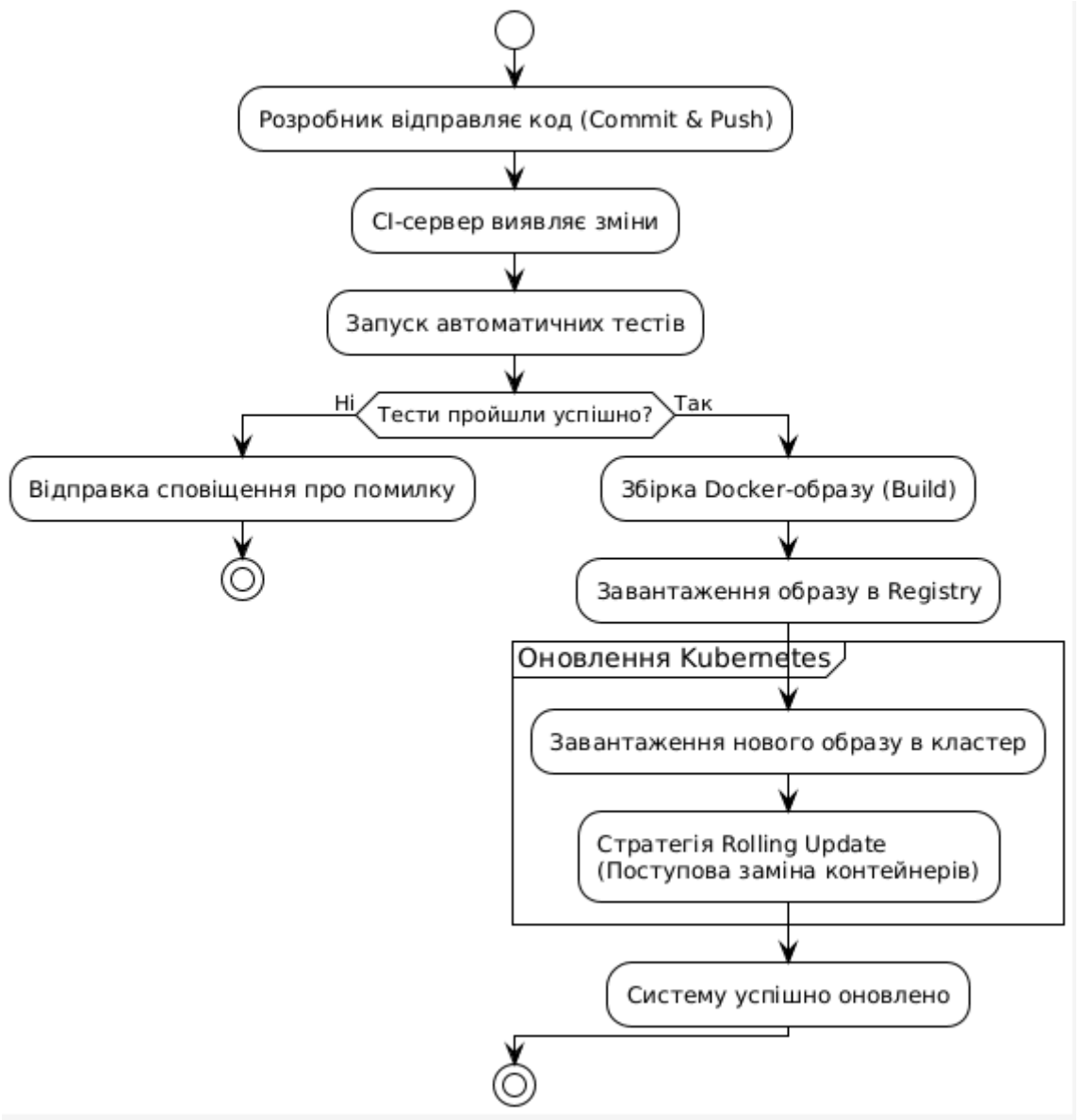


Рис. 2.2 – Реалізація циклу безперервної доставки (CI/CD) у гнучкій методології

Окрему увагу в обраному процесі розробки приділено керуванню ризиками, пов’язаними з використанням LLM. Оскільки інтеграція генеративного штучного інтелекту може призводити до непередбачуваних результатів, ітеративний підхід дозволяє виділити окремі спринти на дослідження (Spikes) для вибору оптимальної моделі, налаштування промпт-інжинірингу та валідації точності відповідей. Це мінімізує ризик створення системи, яка не відповідає вимогам надійності. Документування коду та архітектурних рішень здійснюється безперервно протягом усього процесу, що забезпечує підтримуваність системи у майбутньому. Таким чином, обрана стратегія розробки забезпечує баланс між гнучкістю, необхідною для наукового пошуку, та дисципліною, необхідною для створення стабільного інженерного рішення.

2.2 Проектування архітектури системи

Архітектура розроблюваної програмної системи базується на принципах модульності та слабкої зв’язаності компонентів, що є умовою для забезпечення масштабованості та надійності в умовах розподіленого серверного середовища. Концептуально система реалізується як багаторівнева клієнт-серверна структура, де чітко розмежовано рівні представлення даних, бізнес-логіки, зберігання інформації та виконання контейнеризованих навантажень. Центральним елементом архітектури виступає керуючий модуль, реалізований на базі фреймворку Django, який виконує роль оркестратора бізнес-процесів та забезпечує взаємодію між користувачем, кластером Kubernetes та сервісом штучного інтелекту.

Рівень бізнес-логіки спроєктовано з урахуванням асинхронної природи взаємодії із зовнішніми інтерфейсами. Оскільки операції з API Kubernetes та запити до великих мовних моделей можуть мати значну затримку, архітектура передбачає використання черги завдань. Для реалізації цього механізму застосовується зв’язка бібліотеки Celery та брокера повідомлень, наприклад Redis. Таке рішення дозволяє винести важкі обчислювальні процеси та мережеві запити

у фоновий режим, забезпечуючи високу чуйність інтерфейсу [13]. Коли користувач ініціює дію, наприклад розгортання нового сервісу, система створює відповідне завдання у черзі, яке згодом обробляється окремим воркером, не блокуючи основний потік виконання програми.

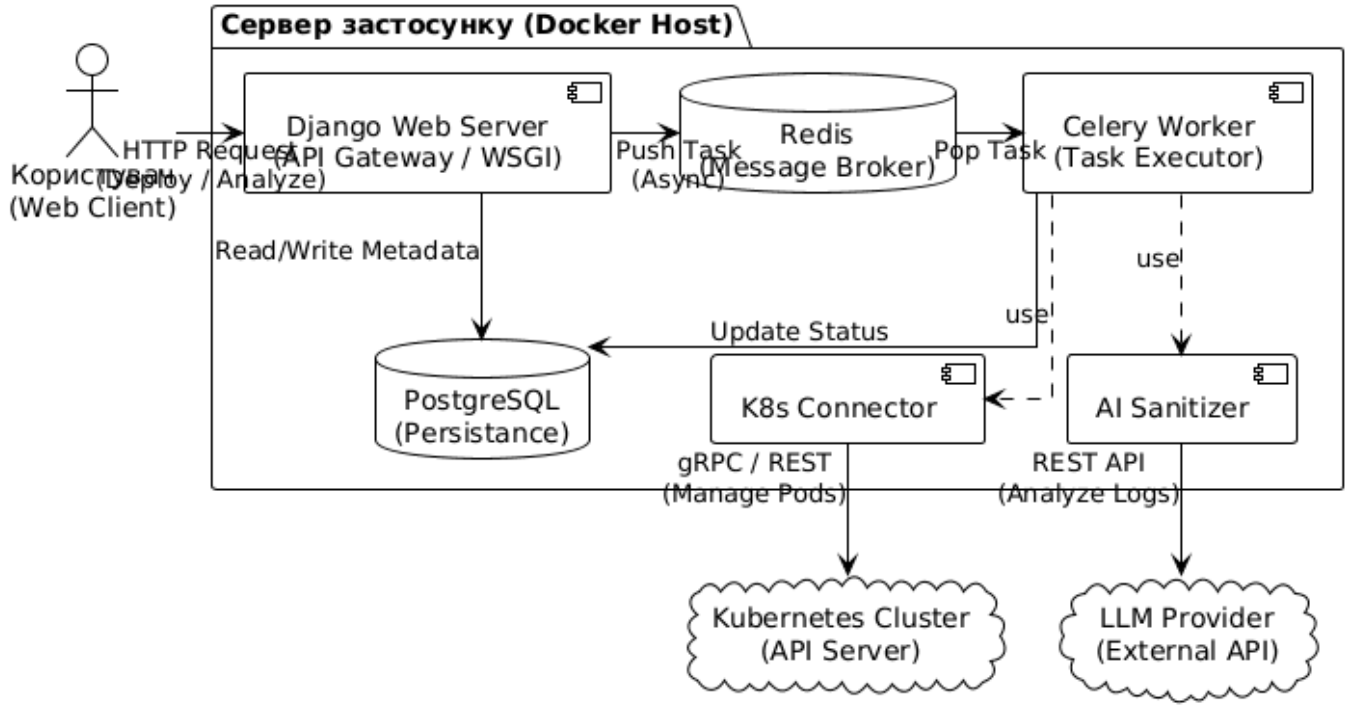


Рис. 2.3 – Компонентна діаграма архітектури системи з виділенням асинхронного контуру обробки завдань

Підсистема взаємодії з інфраструктурою базується на використанні клієнтських бібліотек Kubernetes, що дозволяє програмно керувати ресурсами кластера. Архітектура передбачає абстрагування низькорівневих базових ресурсів оркестратора (Pods, Deployments, Services) та інших у більш високорівневі сутності "Проєкт" або "Застосунок" у контексті бази даних системи. Це забезпечує синхронізацію стану між реляційною базою даних, де зберігаються налаштування та метадані користувачів, та фактичним станом компонентів інфраструктури. У якості основного сховища інформації обрано СУБД PostgreSQL, яка забезпечує цілісність транзакцій та надійне зберігання конфігурацій.

Інтеграція з компонентом штучного інтелекту реалізована через окремий архітектурний модуль, на який покладено задачі препроцесингу та анонімізації даних. Перед відправкою логів або описів помилок до LLM, дані проходять через фільтр, що видаляє конфіденційну інформацію, та форматувальник, що структурує контекст для моделі. Взаємодія здійснюється через RESTful API, де система виступає клієнтом по відношенню до сервісу LLM. Результати аналізу зберігаються у сховищі даних із застосуванням механізмів кешування для уникнення повторних запитів по ідентичних інцидентах. Така архітектурна модель забезпечує гнучкість, дозволяючи за необхідності замінювати компоненти, наприклад, змінювати провайдера LLM або мігрувати базу даних, без суттєвих змін у кодовій базі основного застосунку [14].

2.3 Побудова схем бази даних

Проєктування схеми бази даних є визначальним етапом розробки інформаційної системи, оскільки від правильності організації структур даних залежить швидкість обробки запитів, цілісність інформації та можливість подальшого масштабування програмного комплексу. Враховуючи використання фреймворку Django, який базується на патерні Active Record, за основу обрано реляційну архітектуру зберігання, втілену засобами системи керування базами даних PostgreSQL [15]. Логічна структура бази даних відображає ієрархію об’єктів предметної області та зв’язки між ними, забезпечуючи персистентне зберігання конфігурацій користувачів, параметрів інфраструктури та результатів роботи інтелектуальних агентів. Центральне місце у схемі займає сутність користувача, яка розширює стандартну модель автентифікації додатковими атрибутами, необхідними для розмежування доступу до ресурсів кластера та зберігання персональних квот.

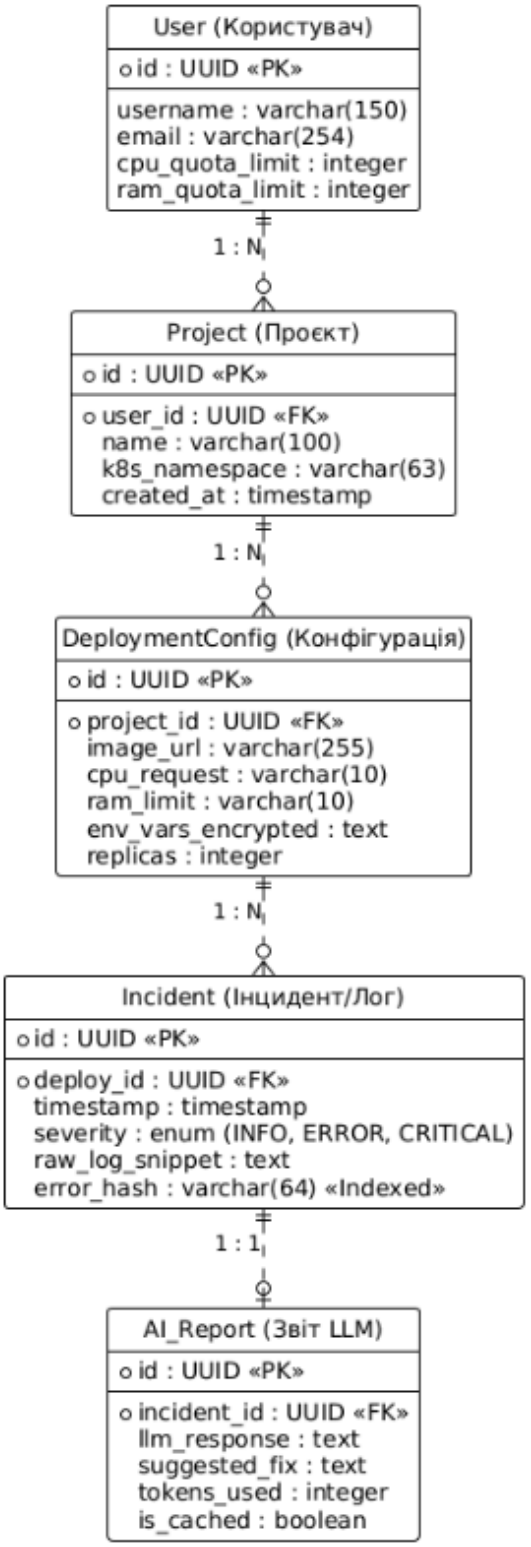


Рис. 2.4 – ER-діаграма концептуальної моделі даних програмного комплексу

Зв’язок між користувачем та проєктами реалізовано за типом «один до багатьох», що дозволяє одному адміністратору керувати декількома незалежними середовищами (наприклад, Dev, Stage, Prod). Для забезпечення високої швидкодії

веб-інтерфейсу спроєктовано сутності, що дублюють метадані активних подів та сервісів («тіньові» копії). Це архітектурне рішення мінімізує затримки, оскільки читання даних для відображення списків відбувається з локальної бази, а не через повільні мережеві запити до API кластера. Актуалізація цих даних покладена на асинхронні фонові процеси, які забезпечують періодичну синхронізацію локального стану з реальним станом інфраструктури [16].

Спеціалізований сегмент схеми бази даних присвячено підсистемі моніторингу та взаємодії зі штучним інтелектом. Розроблено структуру для реєстрації інцидентів, яка включає часові мітки, рівень критичності події та текстовий опис помилки, отриманий з логів. Ця таблиця має зв’язок із сутністю аналітичних звітів, де зберігаються відповіді великої мовної моделі. Зберігання пари «запит-відповідь» дозволяє реалізувати механізм кешування, запобігаючи повторним зверненням до платних API LLM при виникненні ідентичних помилок, а також формує базу знань для подальшого донавчання моделі. Забезпечення цілісності даних досягається використанням зовнішніх ключів з налаштованими правилами каскадного видалення, а для оптимізації вибірок передбачено індексацію полів, що найчастіше використовуються у фільтрації запитів. Розроблена схема є нормалізованою, що мінімізує надлишковість даних та ризики виникнення аномалій при оновленні інформації.

2.4 Побудова UML-діаграм класів

Побудова діаграми класів є ключовим етапом об’єктно-орієнтованого проєктування, що дозволяє візуалізувати статичну структуру програмної системи, визначити типи даних, методи обробки інформації та характер взаємозв’язків між окремими компонентами коду. Розробка UML-діаграми для системи керування серверною інфраструктурою базується на принципах інкапсуляції бізнес-логіки та розмежуванні відповідальності між модулями представлення, опрацювання інформації та комунікації із зовнішніми інтерфейсами [17]. У контексті використання фреймворку Django, архітектура класів розділяється на дві основні

групи: класи-моделі (Models), що відображають структуру бази даних та містять методи роботи з даними, та класи-сервіси (Services/Utils), що реалізують складну логіку взаємодії з API Kubernetes та великими мовними моделями.

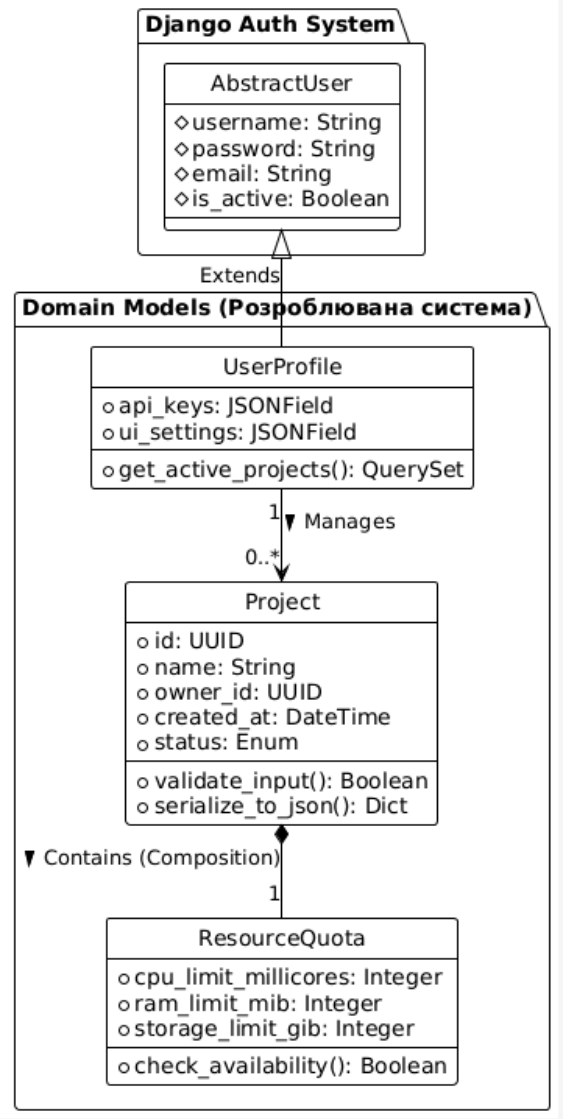


Рис. 2.5 – UML-діаграма класів доменної моделі системи (основні сутності)

Центральним елементом діаграми є клас Project, який агрегує в собі всю інформацію про конфігурацію розгортання. Цей клас містить атрибути, що описують назву проєкту, ідентифікатор власника, дату створення та статус активності. Ключовими методами класу є процедури валідації вхідних даних та методи серіалізації для передачі інформації у форматі JSON. Клас Project пов’язаний відношенням композиції з класом ResourceQuota, який інкапсулює

параметри обмеження ресурсів (CPU, RAM). Така декомпозиція дозволяє гнучко керувати політиками доступу до обчислювальних потужностей без зміни основної сутності проєкту. Користувачі системи представлені класом UserProfile, який успадковує базовий функціонал від стандартного класу автентифікації Django, розширюючи його атрибутами для зберігання API-ключів.

Для реалізації взаємодії з кластером оркестрації спроєктовано окремий шар абстракції, представлений класом K8sConnector. Цей клас реалізує патерн «Адаптер», перетворюючи внутрішні команди системи у виклики до API Kubernetes. Він містить методи create_deployment(), scale_replicas() та get_pod_status(), які приймають на вхід об’єкти конфігурації та повертають стандартизовані відповіді, ізолюючи решту системи від складності низькорівневої комунікації [18]. Логіка моніторингу реалізована через клас MetricsAggregator, який відповідає за збір телеметрії та її підготовку для візуалізації.

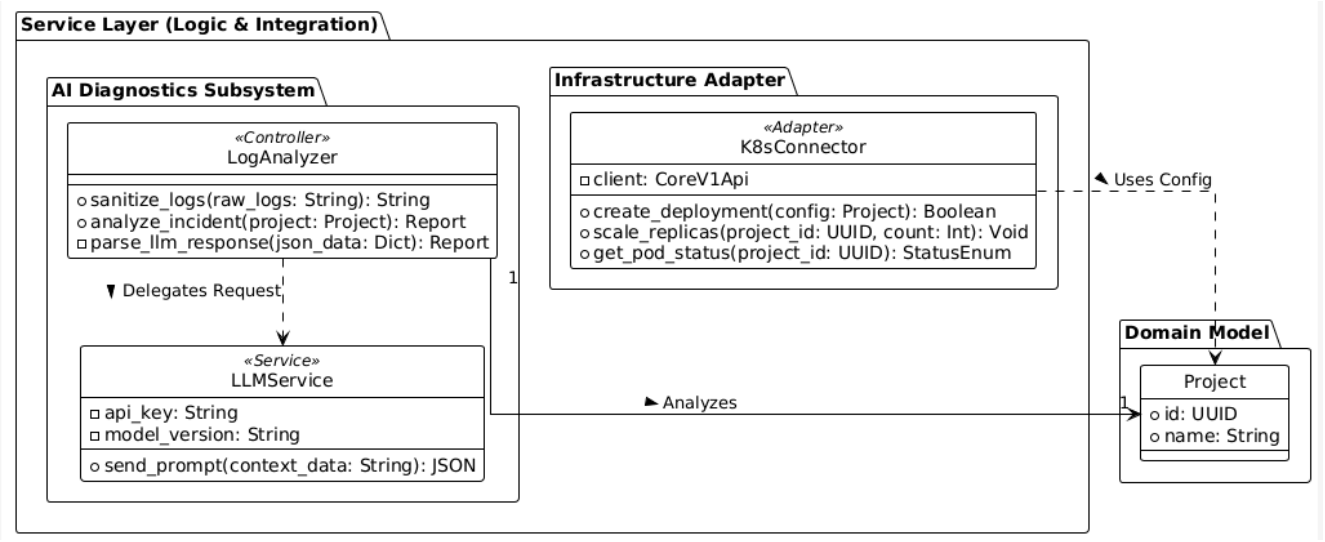


Рис. 2.6 – UML-діаграма класів доменної моделі системи (основні сутності)

Окрему гілку ієрархії складають класи, відповідальні за інтелектуальний аналіз. Клас LogAnalyzer виступає контролером процесу діагностики. Він взаємодіє з класом LLMService, який інкапсулює логіку відправки запитів до нейромережі. LogAnalyzer містить методи для попередньої обробки тексту (санітизації) та методи парсингу відповідей від штучного інтелекту. Зв’язок між

LogAnalyzer та Project реалізовано через асоціацію, що дозволяє запускати аналіз для конкретного екземпляра розгортання. Така структура класів забезпечує високу модульність системи, дозволяючи незалежно модифікувати логіку роботи з AI або Kubernetes без впливу на моделі даних, що спрощує подальший супровід та розширення функціоналу програмного комплексу.

2.5 Вибір мови та середовища розробки

Підбір технологічного стека для створення системи здійснено на основі оцінювання функціональних специфікацій та необхідності забезпечення високої продуктивності розробки та сумісності з сучасними стандартами індустрії хмарних обчислень. Основним критерієм відбору стала здатність технологій забезпечувати ефективну взаємодію між веб-інтерфейсом, системними компонентами керування інфраструктурою та модулями штучного інтелекту. Враховуючи мультипарадигмальний характер проєкту, як основну мову програмування обрано Python. Це рішення зумовлене лідируючими позиціями мови у сферах автоматизації системного адміністрування (DevOps) та машинного навчання. Високий рівень абстракції, лаконічний синтаксис та наявність обширної екосистеми бібліотек, таких як kubernetes для роботи з API кластера та спеціалізованих клієнтів для інтеграції з LLM, дозволяють мінімізувати обсяг коду та зосередитися на реалізації бізнес-логіки. Крім того, динамічна типізація Python сприяє швидкому прототипуванню, що є критично важливим в умовах ітеративного процесу розробки [19].

У якості каркаса для побудови серверної частини веб-застосунку перевагу надано платформі Django. Застосований у ній архітектурний патерн MVT (Model-View-Template) забезпечує чітке розмежування логіки обробки даних та їх відображення. Вирішальним фактором на користь Django стала наявність потужного вбудованого механізму ORM (Object-Relational Mapping), який абстрагує роботу на рівні СКБД, уможливлюючи маніпуляції записами як об’єктами Python. Це суттєво підвищує безпеку системи, автоматично нівелюючи

ризики типових атак, зокрема ін’єкцій SQL-коду [20]. Інтегрована система адміністрування Django значно скорочує час на розробку інтерфейсів для керування користувачами та правами доступу, що дозволяє перерозподілити ресурси на реалізацію специфічних функцій моніторингу.

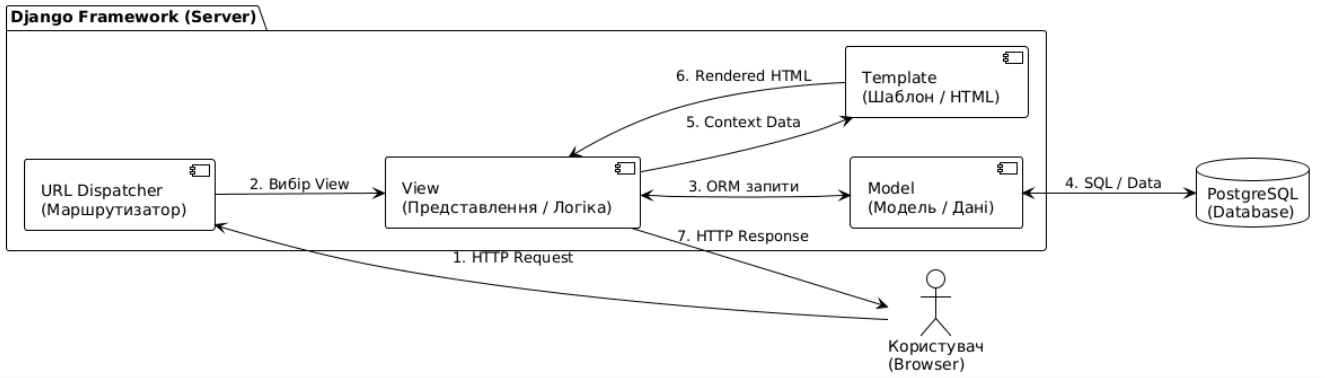


Рис. 2.7 – Схема реалізації патерну Model-View-Template (MVT) у системі

Для створення клієнтської частини інтерфейсу використовуються стандартизовані технології HTML5 та CSS3, доповнені мовою програмування JavaScript. Використання JavaScript є необхідним для реалізації асинхронної взаємодії з сервером (AJAX), що дозволяє оновлювати графіки моніторингу та статуси подів у динамічному режимі, усуваючи необхідність повного оновлення веб-документа. З метою візуалізації даних обрано бібліотеку Chart.js, яка забезпечує гнучкість у побудові графіків навантаження на основі JSON-даних, отриманих від бекенду. Для прискорення верстки та забезпечення повної адаптивності інтерфейсу на різних типах пристроїв доцільно інтегрувати CSS-фреймворк, наприклад Tailwind CSS, що дозволяє ефективно керувати стилями за допомогою утилітарних класів. Окрему увагу приділено розробці клієнтської логіки для діалогового вікна з LLM, яка забезпечує зручне потокове відображення текстових рекомендацій та звітів у реальному часі.

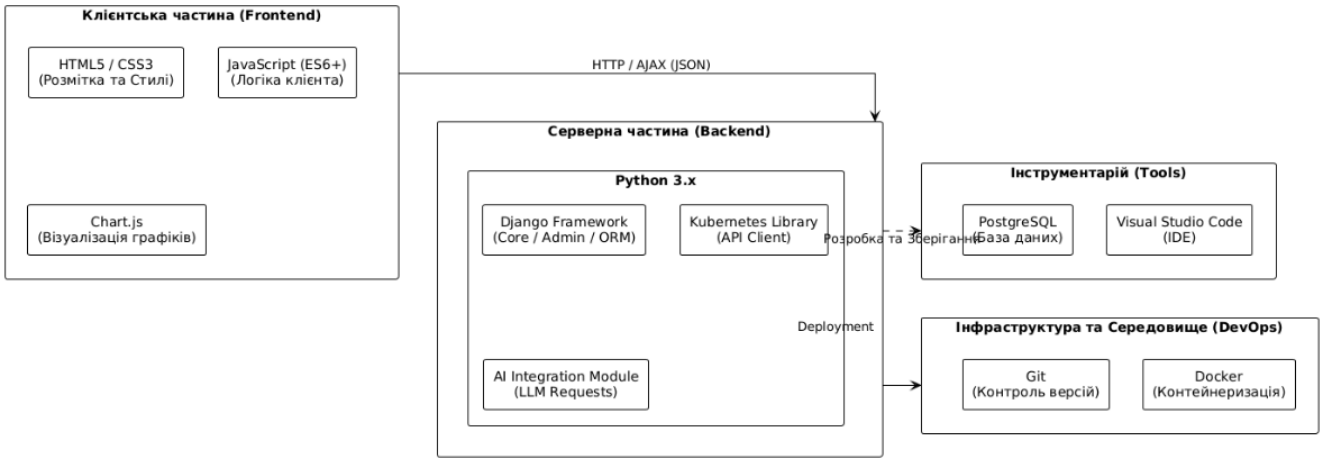


Рис. 2.8 – Структурна схема технологічного стека програмного комплексу

Середовищем розробки (IDE) обрано Visual Studio Code, що пояснюється його модульністю, легкою вагою та широкою підтримкою необхідних розширень для Python, Docker та Kubernetes. Інтегрований термінал та засоби налагодження дозволяють ефективно працювати з контейнеризованим оточенням. Для забезпечення контролю версій та організації командної роботи використовується система Git. Вона дозволяє вести історію змін, створювати окремі гілки для розробки нових функцій та безпечно інтегрувати їх в основну кодову базу. Процес розгортання та тестування самої системи здійснюється у середовищі Docker, що гарантує ідентичність оточення на етапах розробки та експлуатації, усуваючи проблеми залежностей. Таким чином, сформований технологічний стек є збалансованим та повністю відповідає поставленим завданням проєктування високонавантаженої системи керування. Додатково налаштовано засоби статичного аналізу коду (linters) та форматування, що забезпечує дотримання єдиного стилю кодування (PEP 8) ще на етапі написання програмних модулів

2.6 Реалізація основних класів та методів

Програмна реалізація спроєктованої архітектури здійснюється шляхом написання програмного коду на мові Python із використанням об’єктно-орієнтованого підходу, що дозволяє трансформувати теоретичні моделі у функціональні компоненти системи. Основу програмної логіки складають моделі

даних, визначені у файлі models.py фреймворку Django, які виступають відображенням сутностей бази даних. Реалізація класу Project передбачає визначення полів для зберігання метаданих, таких як унікальний ідентифікатор UUID, назва, опис та посилання на репозиторій. Критично важливим аспектом є перевизначення методу save даного класу, в якому реалізовано логіку валідації вхідних параметрів перед записом у базу даних. Це включає перевірку коректності формату URL репозиторію та відповідність виділених ресурсів глобальним лімітам системи. Для зберігання динамічних конфігураційних даних, таких як змінні середовища, застосовано поле типу JSONField, що забезпечує гнучкість структури даних без необхідності зміни схеми таблиці при додаванні нових параметрів. Для зберігання динамічних конфігураційних даних, таких як змінні середовища, застосовано поле типу JSONField, що забезпечує гнучкість структури даних без необхідності зміни схеми таблиці при додаванні нових параметрів.

```python
class K8sManager(models.Model):
    server = models.ForeignKey('servers.Server', on_delete=models.CASCADE, related_name='ku
    k8s_module_type = models.CharField(max_length=20, choices=MODULES_CHOICES, default=None
    active = models.BooleanField(default=False, verbose_name=_('Active'))

    def get_clusters(self):
        """..."""
        module_commands = {...}

        resource_commands = {...}

        try:
            if not self.k8s_module_type or self.k8s_module_type not in module_commands:
                return []

            command = module_commands[self.k8s_module_type]['command']
            parser = module_commands[self.k8s_module_type]['parser']

            output = self.server.execute_command(command)
            if not output:
                return []

            # Get additional resources for kubernetes modules
            data = parser(output)
            if self.k8s_module_type in ['minikube', 'k3s', 'kubeadm', 'microk8s']:
```

Рис. 2.9 – Частина коду для взаємодії з кубернетесом

Взаємодія з кластером Kubernetes інкапсульована у сервісному класі K8sManager, розміщеному в окремому модулі бізнес-логіки. Конструктор цього класу ініціалізує з’єднання з API-сервером кластера, використовуючи конфігураційний файл або сервісний токен. Одним із ключових методів є create_deployment, який приймає об’єкт конфігурації проєкту та виконує його трансформацію у специфікацію V1Deployment бібліотеки kubernetes-client. У тілі методу реалізовано алгоритм побудови маніфесту, що включає визначення контейнерів, портів та ресурсних квот. Для забезпечення відмовостійкості виклики до API загорнуто у блоки обробки виключень, що дозволяє коректно обробляти помилки мережі або відмови кластера, повертаючи системі стандартизовані повідомлення про помилку [21]. Аналогічним чином реалізовано методи delete_deployment та scale_deployment, які забезпечують керування життєвим циклом застосунків.

Реалізація функціоналу моніторингу зосереджена у класі MetricsService. Цей клас містить методи для асинхронного опитування Metrics API Kubernetes. Метод fetch_pod_metrics виконує HTTP-запити до відповідних кінцевих точок кластера, отримуючи "сирі" дані у форматі JSON. Далі відбувається парсинг отриманих значень, конвертація одиниць виміру (наприклад, з nanocores у millicores) та агрегація даних за часовими інтервалами. Оброблена інформація передається на фронтенд для візуалізації або зберігається в базі даних для історичного аналізу.

Особливу увагу приділено реалізації класу AIAnalyzer, який відповідає за інтеграцію з великою мовною моделлю. Основний метод цього класу, analyze_error_log, реалізує багатоступеневий конвеєр обробки даних. На першому етапі відбувається санітизація вхідного тексту логів за допомогою регулярних виразів для видалення конфіденційної інформації, такої як IP-адреси, паролі або токени доступу. Наступним кроком є формування контекстного промпту, який об’єднує опис помилки, очищені логи та інструкції для моделі. Взаємодія з зовнішнім API LLM реалізована через асинхронний клієнт, що дозволяє уникнути блокування основного потоку виконання програми під час очікування відповіді від

нейромережі. Отримана відповідь проходить через метод парсингу, який структурує текст у формат, придатний для відображення користувачеві.

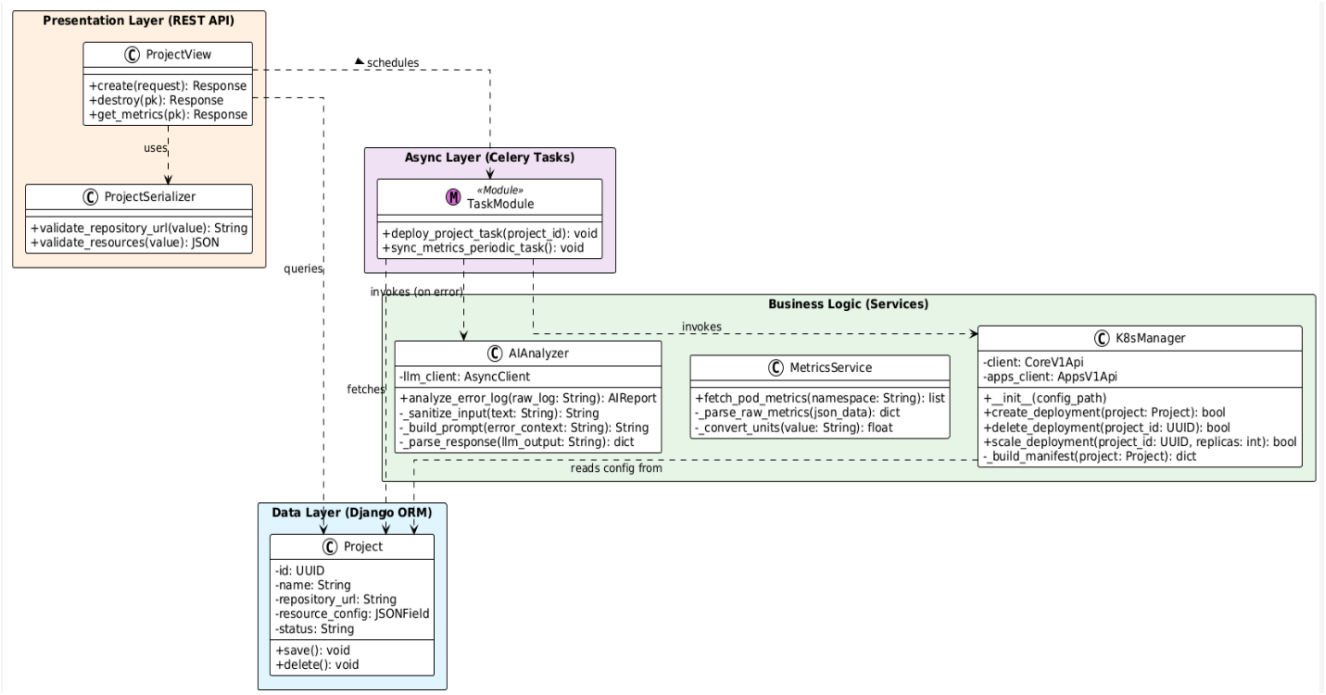


Рис. 2.10 – Схема архітектурної взаємодії компонентів системи

Для організації взаємодії між клієнтською частиною та описаною бізнес-логікою розроблено набір REST-контролерів у модулі views.py. Використання Django REST Framework дозволило стандартизувати процес обробки HTTP-запитів. Кожен контролер (View) відповідає за прийом вхідних даних, їх валідацію за допомогою серіалізаторів та ініціювання відповідних фонових завдань. Наприклад, при надходженні POST-запиту на створення проєкту, серіалізатор перевіряє коректність JSON-структури, після чого контролер не виконує розгортання синхронно, а лише створює запис у базі даних зі статусом «Pending» та передає ID нового запису у чергу Celery. Це забезпечує миттєвий відгук API (код 202 Accepted) та запобігає тайм-аутам браузера.

Окрему увагу приділено забезпеченню цілісності даних на рівні ORM. У методах моделей переписано стандартні методи delete, щоб забезпечити каскадне видалення ресурсів не тільки з бази даних, але й з кластера Kubernetes. Для цього використовуються сигнали Django (signals): перед видаленням об’єкта проєкту

спрацьовує обробник pre_delete, який викликає метод delete_deployment класу K8sManager. Така архітектура гарантує, що при видаленні запису з панелі керування у кластері не залишатиметься «сміттєвих» ресурсів (orphaned resources), які споживають обчислювальну потужність

Для забезпечення високої продуктивності веб-інтерфейсу, важкі обчислювальні операції та мережеві запити винесено у фонові завдання за допомогою бібліотеки Celery. У модулі tasks.py реалізовано функції-обгортки для методів класів K8sManager та AIAnalyzer. Наприклад, завдання deploy_project_task приймає ідентифікатор проєкту, завантажує необхідні дані з бази та викликає відповідний метод сервісного класу. Використання декоратора @shared_task дозволяє планувальнику Celery розпізнавати ці функції та ставити їх у чергу виконання брокера повідомлень Redis. Такий підхід гарантує, що інтерфейс користувача залишається чуйним навіть при виконанні тривалих операцій розгортання або аналізу великих масивів логів [22].

2.7 Розробка інтерфейсу користувача

Етап розробки інтерфейсу користувача (UI) став ключовим у процесі створення системи, оскільки саме якість графічної оболонки визначає ефективність взаємодії адміністратора зі складними серверними модулями. Проєктування візуальної частини базувалося на концепції людино-орієнтованого дизайну (Human-Centered Design), пріоритетом якого є мінімізація когнітивного навантаження та зниження ймовірності помилок оператора [23].

Архітектура головної сторінки реалізована у форматі інтерактивної інформаційної панелі (Dashboard). Її основне завдання — надати миттєвий огляд стану інфраструктури ("helicopter view"). Інтерфейс розділено на логічні зони: навігаційний сайдбар для швидкого перемикання між модулями, верхня панель з налаштуваннями профілю та основна робоча область. У цій області консолідуються критичні метрики: статус кластера, активність сервісів та останні системні сповіщення. Кольорова гама підібрана з метою візуального виділення

критичних станів (червоний/помаранчевий маркери) та не втомлювати зір при тривалій роботі.

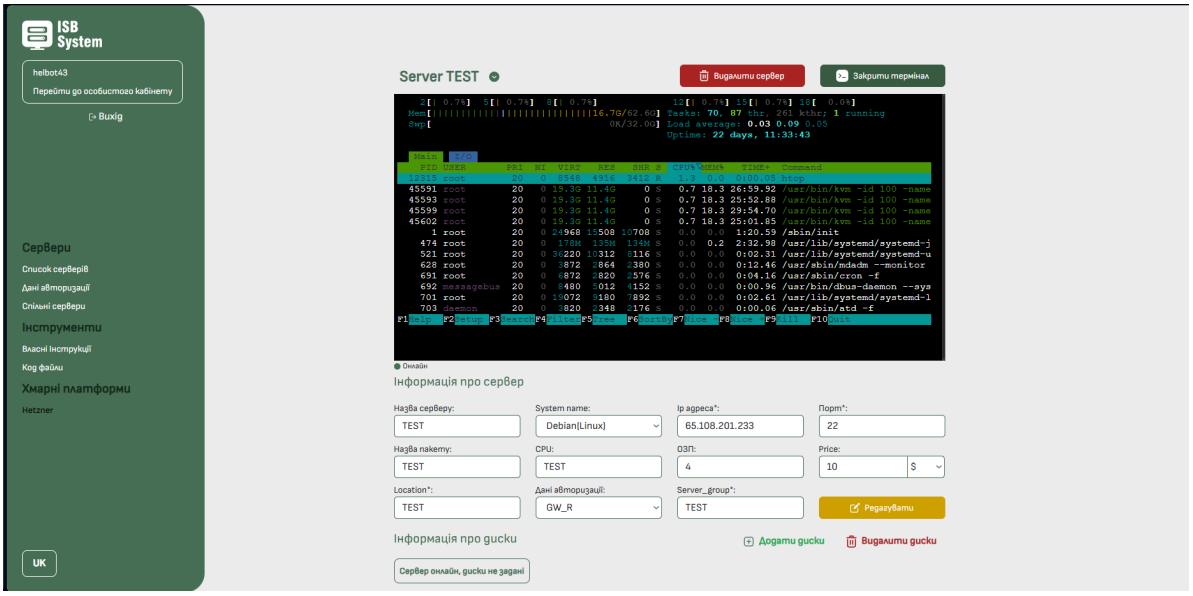


Рис. 2.11 – Інтерфейс користувача

Програмна реалізація клієнтської частини виконана на базі технологій, що забезпечують баланс між продуктивністю та зручністю підтримки коду. Основою слугує шаблонний двигун Django Template Language (DTL). Цей інструмент дозволив реалізувати принцип успадкування шаблонів, де базовий каркас сторінки (header, footer, підключення скриптів) визначається в одному файлі, а унікальний контент конкретних розділів динамічно підставляється у відповідні блоки. Такий підхід суттєво зменшує дублювання коду та спрощує масштабування проєкту.

Верстка компонентів виконана з використанням семантичного HTML5, що покращує доступність інтерфейсу. Для стилізації використано можливості CSS3, зокрема специфікації Flexbox для одновимірного розташування елементів (навігаційні меню, кнопки) та Grid Layout для створення складних двовимірних сіток (розташування віджетів на дашборді). Завдяки медіа-запитам (@media queries) реалізовано повну адаптивність: інтерфейс автоматично перебудовується залежно від розширення екрана, забезпечуючи повноцінне керування системою з мобільних пристроїв.

```
<section class="servers_page offset-md-3 col-md-8 col-12">
    <section class="server_detail" data-server_id="{{ server.id }}">
        <div class="information_block">
            <div class="d-flex justify-content-between mb-2">
                <div class="fs-5 text-primary">
                    {% trans "Інформація про диски" %}
                </div>
                {% if not read_only %}
                    <div class="d-flex gap-2">
                        <div class="open_modal_object btn d-flex fw-bold text-success align-items-center border-bottom-hover"
                             data-url="{% url "modal_add" model_name="ServerDisk" %}">
                            <img class="m-mobile me-2 d-flex" src="/media/icons/green_add_icon.svg" alt="{% trans "Додати" %}">
                            <div class="d-none d-md-block">
                                {% trans "Додати диски" %}
                            </div>
                        </div>
                        <div class="start_delete_disks btn d-flex fw-bold text-danger align-items-center border-bottom-hover">
                            <img class="m-mobile me-2 d-flex" src="/media/icons/red_trash.svg" alt="{% trans "Видалити" %}">
                            <div class="d-none d-md-block ">
                                {% trans "Видалити диски" %}
                            </div>
                        </div>
                    </div>
                {% endif %}
```

Рис. 2.12 – Частина коду інтерфейсу користувача

Критично важливим аспектом для DevOps-інструментів є візуалізація метрик у реальному часі. Підсистема моніторингу розроблена з використанням бібліотеки Chart.js, яка використовує елемент HTML5 Canvas для рендерингу високопродуктивних графіків. На сторінці деталізації проєкту відображаються тренди використання процесорного часу (CPU) та оперативної пам'яті (RAM), а також статистика звернень до мовних моделей.

Щоб уникнути надмірного навантаження на мережу та покращити UX, замість повного перезавантаження сторінки застосовано технологію AJAX (через Fetch API). Клієнтський JavaScript-сценарій працює у фоновому режимі, виконуючи періодичні "опитування" (polling) API-ендпоінтів сервера. Отримані дані у форматі JSON автоматично обробляються, і графіки плавно оновлюються. Це дозволяє адміністратору спостерігати за динамікою процесів без переривання роботи та мерехтіння інтерфейсу [24].

Рис. 2.13 – Графіки навантаження на сервер та використання ллм

Інноваційною складовою системи є модуль інтелектуального аналізу інцидентів, реалізований шляхом інтеграції API великих мовних моделей безпосередньо у розділ перегляду системних журналів. Цей функціонал дозволяє автоматизувати рутинний процес первинної діагностики: замість ручного пошуку рішень у документації чи на форумах, користувач має можливість виділити критичний фрагмент тексту помилки або стек викликів (stack trace) та ініціювати його глибокий аналіз одним кліком. Такий підхід перетворює роботу з "сирими" логами на інтерактивний процес отримання експертних рекомендацій, суттєво зменшуючи час реакції на збої.

Для підвищення релевантності результатів система не просто передає виділений текст моделі, а збагачується його спеціалізованим системним контекстом (system prompt). Цей прихований від користувача шар налаштовує нейромережу на роль кваліфікованого DevOps-інженера, встановлюючи жорсткі вимоги до структури відповіді. Завдяки цьому алгоритм фокусується на технічних

аспектах проблеми, відсіюючи зайву інформацію та генеруючи рішення, адаптовані до специфіки серверного середовища.

Взаємодія з модулем реалізована через асинхронний виклик, що забезпечує чуйність інтерфейсу: під час обробки запиту відображається індикатор прогресу, а результат повертається у вигляді структурованого звіту в модальному вікні, що перекриває основний контент. Звіт містить детальний опис проблеми, аналіз причин виникнення та конкретні команди для її усунення. Для зручності сприйняття застосовано парсинг Markdown, який забезпечує коректне форматування тексту та синтаксичне підсвічування фрагментів коду.

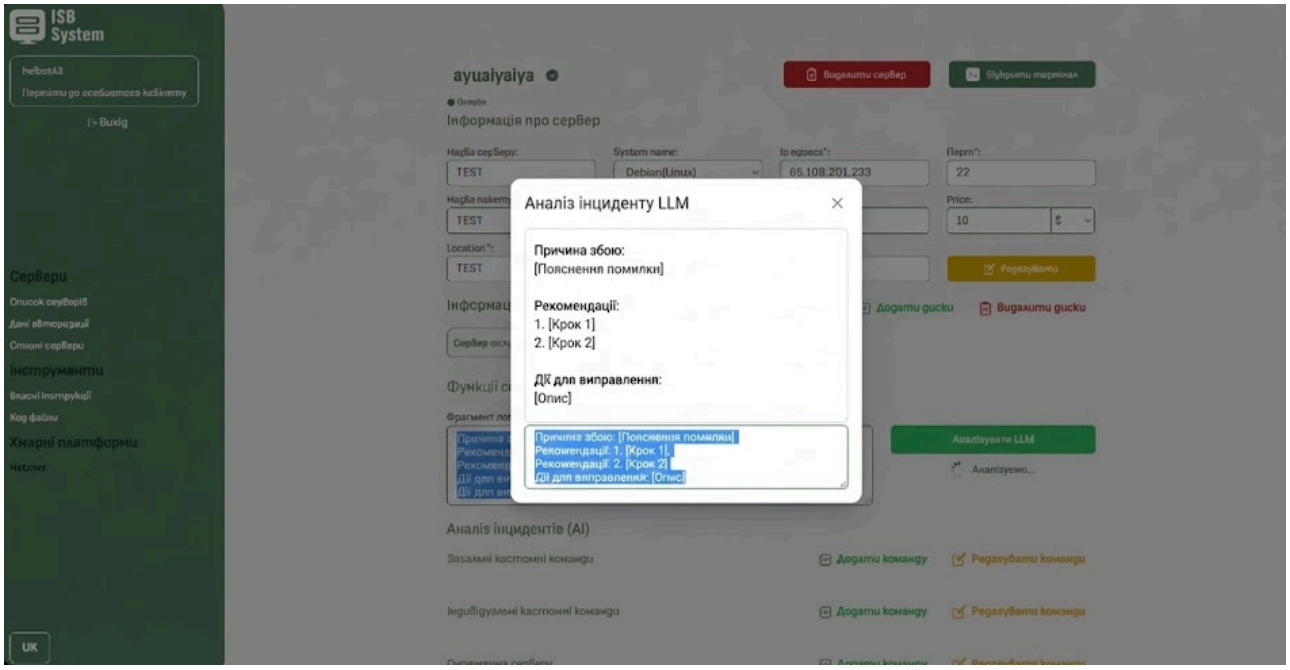


Рис. 2.14 – приклад звіту від ллм

Загальна ергономіка системи (UX) підсилена низкою допоміжних механізмів. Впроваджено систему спливаючих сповіщень (toast notifications), які надають ненав’язливий зворотний зв’язок про результати дій (наприклад, "Налаштування збережено", "Помилка з’єднання"). Також реалізовано дворівневу валідацію форм: первинна перевірка відбувається на стороні клієнта за допомогою атрибутів HTML5 та JavaScript. Це дозволяє миттєво підсвічувати помилки.

3 ТЕСТУВАННЯ ВПРОВАДЖЕННЯ ТА ПІДТРИМКА

Етап тестування передбачає виконання комплексних модульних та навантажувальних випробувань для верифікації відмовостійкості мікросервісів, після чого здійснюється автоматизоване розгортання системи в кластері Kubernetes за допомогою CI/CD пайплайнів. Підтримка експлуатації базується на безперервному відстеженні метрик продуктивності та періодичному оновленні контексту для LLM, що дозволяє адаптувати алгоритми штучного інтелекту до нових сценаріїв навантаження та гарантувати стабільність роботи серверів.

3.1 Тестування програмної системи

Тестування розробленого програмного комплексу є критично важливою фазою життєвого циклу проєкту, спрямованою на верифікацію відповідності реалізованого функціоналу первинним вимогам та забезпечення надійності роботи системи в умовах реальної експлуатації. Враховуючи складну гетерогенну архітектуру, що поєднує веб-інтерфейс, засоби оркестрації контейнерів та модулі штучного інтелекту, процес перевірки якості вимагає комплексного підходу. Головною метою цього етапу є виявлення та усунення програмних дефектів, перевірка коректності взаємодії між ізольованими компонентами системи та оцінка стійкості до відмов при роботі із зовнішніми API. Тестування дозволяє мінімізувати ризики виникнення критичних збоїв під час керування інфраструктурою та гарантувати точність роботи інтелектуальних алгоритмів діагностики. Стратегія верифікації реалізується за принципом багаторівневого тестування, починаючи з модульних тестів (Unit Testing) для перевірки ізольованої бізнес-логіки бекенду на Python, та завершуючи комплексним системним тестуванням розгорнутого середовища.

3.1.1 Види та план тестування

Стратегія забезпечення якості розроблюваної системи базується на багаторівневій моделі тестування, яка передбачає послідовну перевірку компонентів від найнижчого рівня коду до повної системної інтеграції. План тестування структурований таким чином, щоб локалізувати потенційні помилки на ранніх стадіях, зменшуючи вартість їх виправлення. Визначено чотири основні види тестування, які застосовуються в рамках даного проєкту: модульне, інтеграційне, системне та навантажувальне. Кожен вид має чітко окреслену область покриття та специфічні інструменти реалізації. Крім того, для забезпечення оперативного зворотного зв'язку більшість тестових сценаріїв буде інтегровано в автоматизований конвеєр CI/CD, що мінімізує вплив людського фактора. Завершальним етапом стратегії стане прийомочне тестування (UAT), яке дозволить підтвердити відповідність продукту бізнес-вимогам [25].

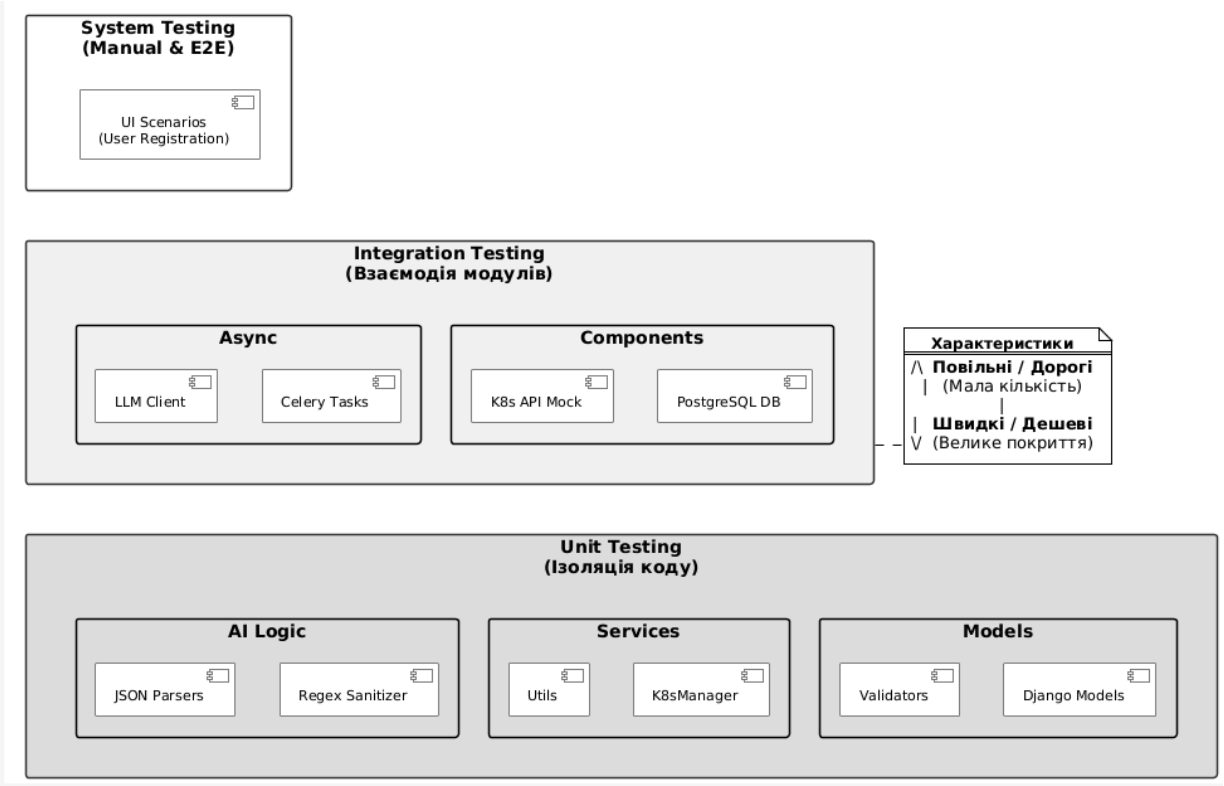


Рис. 3.1 – Піраміда тестування розроблюваної системи

Першим етапом плану є проведення модульного тестування (Unit Testing), об'єктом якого виступають окремі класи, функції та методи системи, ізольовані

від зовнішніх залежностей. Основна увага приділяється перевірці коректності роботи бізнес-логіки моделей Django, методів валідації вхідних даних та алгоритмів парсингу відповідей від LLM. Для цього використовуються заглушки (mock-об’єкти), що імітують поведінку бази даних та зовнішніх API, дозволяючи перевірити поведінку коду в детермінованих умовах [26].

Окремий наголос у рамках модульного тестування робиться на перевірці стійкості коду до граничних випадків (edge cases) та обробці виключних ситуацій. Оскільки вихідні дані від LLM можуть бути варіативними, критично важливо гарантувати, що внутрішні парсери коректно обробляють пошкоджені JSON-структури або неповні відповіді, не викликаючи падіння сервісу. Також тестується логіка формування динамічних промптів: перевіряється, чи вірно підставляються метрики та контекст інциденту у шаблони запитів перед їх відправкою до моделі, що забезпечує точність подальшого аналізу.

```
tests/infra/k8s/test_cluster.py::test_connection_to_cluster_control_plane PASSED          [  7%]
tests/infra/k8s/test_deployments.py::test_monitoring_agent_daemonset_readiness PASSED          [ 15%]
tests/infra/k8s/test_hpa.py::test_horizontal_pod_autoscaling_metrics PASSED          [ 23%]
tests/infra/helm/test_charts.py::test_prometheus_config_map_validation PASSED          [ 30%]
tests/migration/test_etl.py::test_legacy_sql_to_timeseries_migration PASSED          [ 38%]
tests/migration/test_compatibility.py::test_api_v1_backward_compatibility PASSED          [ 46%]
tests/ai/llm_core/test_provider.py::test_llm_api_token_validation PASSED          [ 53%]
tests/ai/llm_core/test_tokenizer.py::test_log_chunking_token_limit PASSED          [ 61%]
tests/ai/vector_db/test_embeddings.py::test_log_entry_vectorization_speed PASSED          [ 69%]
tests/ai/agents/test_analyst.py::test_anomaly_detection_inference_accuracy PASSED          [ 76%]
tests/ai/agents/test_summarizer.py::test_incident_report_generation_hallucination_check PASSED          [ 84%]
tests/integration/test_pipeline.py::test_end_to_end_log_flow_to_dashboard PASSED          [ 92%]
tests/notifications/test_slack.py::test_smart_alert_formatting PASSED          [100%]

================== 12 passed, 1 skipped in 7.93s ==================


** Process exited - Return Code: 0 **
```

Рис. 3.2 – Результат проходження юніт тестів

Наступним кроком є інтеграційне тестування, метою якого є перевірка взаємодії між різними модулями системи та зовнішніми сервісами. На цьому етапі перевіряється коректність передачі даних між веб-застосунком та базою даних PostgreSQL, надійність роботи черги завдань Celery та правильність виконання запитів до API Kubernetes. Особлива увага приділяється сценаріям, де система

виступає клієнтом для сервісів штучного інтелекту, перевіряючи обробку тайм-аутів та помилок мережі.

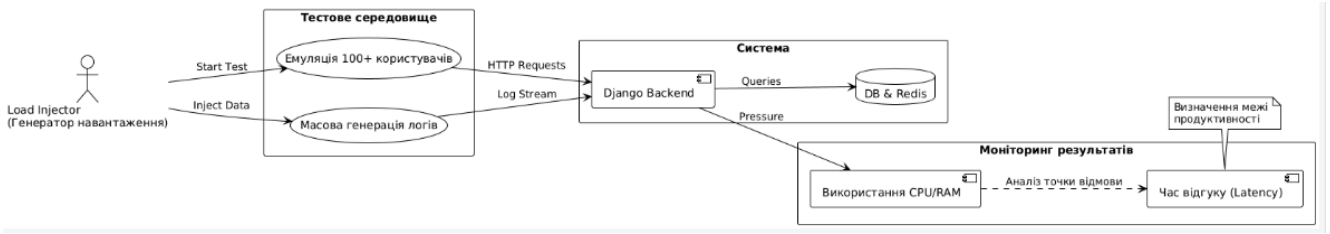


Рис. 3.3 – Логічна схема проведення навантажувального тестування

Системне тестування виконується на повністю розгорнутому програмному комплексі та імітує реальні сценарії роботи користувача. Воно охоплює перевірку функціональних вимог через графічний інтерфейс, включаючи процеси реєстрації, створення проєктів, розгортання контейнерів та отримання аналітичних звітів. Цей етап дозволяє оцінити узгодженість роботи всіх компонентів та зручність інтерфейсу.

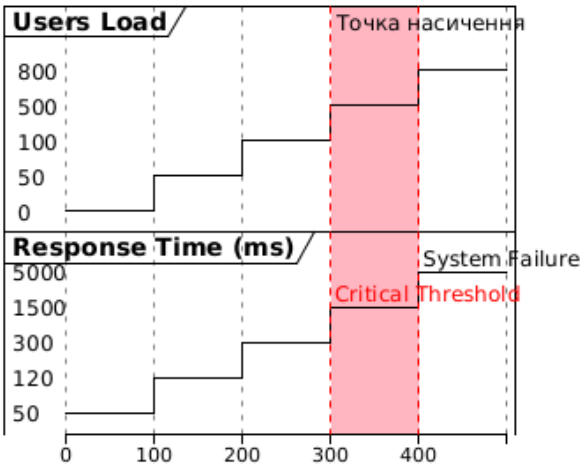


Рис. 3.4 – Графік результатів навантажувального тестування

Завершальним етапом плану є навантажувальне тестування, спрямоване на визначення меж продуктивності системи та її поведінки під час пікових навантажень. Моделюється ситуація конкурентної активності масиву користувачів і генерації великого потоку логів для аналізу, що дає змогу локалізувати лімітуючі

фактори продуктивності, проаналізувати ефективність кешування та перевірити роботу механізмів масштабування. Реалізація цього плану забезпечує всебічну перевірку якості програмного продукту перед його впровадженням у експлуатацію.

3.1.2 Розробка тестових сценаріїв

Розробка тестових сценаріїв є процесом формалізації умов перевірки програмного продукту, що базується на специфікаціях вимог та визначених раніше варіантах використання. Кожен сценарій являє собою документовану сукупність операцій, необхідних для валідації конкретної функції, із зазначенням вхідних даних, очікуваного результату та критеріїв успішності проходження тесту. У межах даної роботи тестові сценарії розподілено на групи відповідно до архітектурних модулів системи: керування доступом, взаємодія з інфраструктурою Kubernetes, моніторинг ресурсів та інтелектуальний аналіз логів. Такий підхід забезпечує повне покриття функціоналу та дозволяє систематизувати процес виявлення дефектів [27].

Перша група сценаріїв фокусується на перевірці підсистеми автентифікації та авторизації. Базовий позитивний сценарій передбачає введення валідних облікових даних користувачем, що має призводити до генерації токена доступу та перенаправлення на головну панель керування. Негативні сценарії включають спроби входу з некоректним паролем, використання заблокованого облікового запису або спроби доступу до адміністративних розділів користувачем з обмеженими правами. Критерієм успішності у цих випадках є коректна відмова системи у доступі із виведенням відповідного повідомлення про помилку без розкриття деталей внутрішньої реалізації механізму безпеки. Окрім цього, сценарії охоплюють перевірку життєвого циклу сесії, зокрема коректність автоматичного завершення роботи при закінченні терміну дії токена (token expiration). Також окрема увага приділяється тестуванню полів вводу на стійкість

до поширених вразливостей, таких як SQL-ін’єкції та XSS-атаки, що забезпечує надійність захисного контуру веб-застосунку [28].

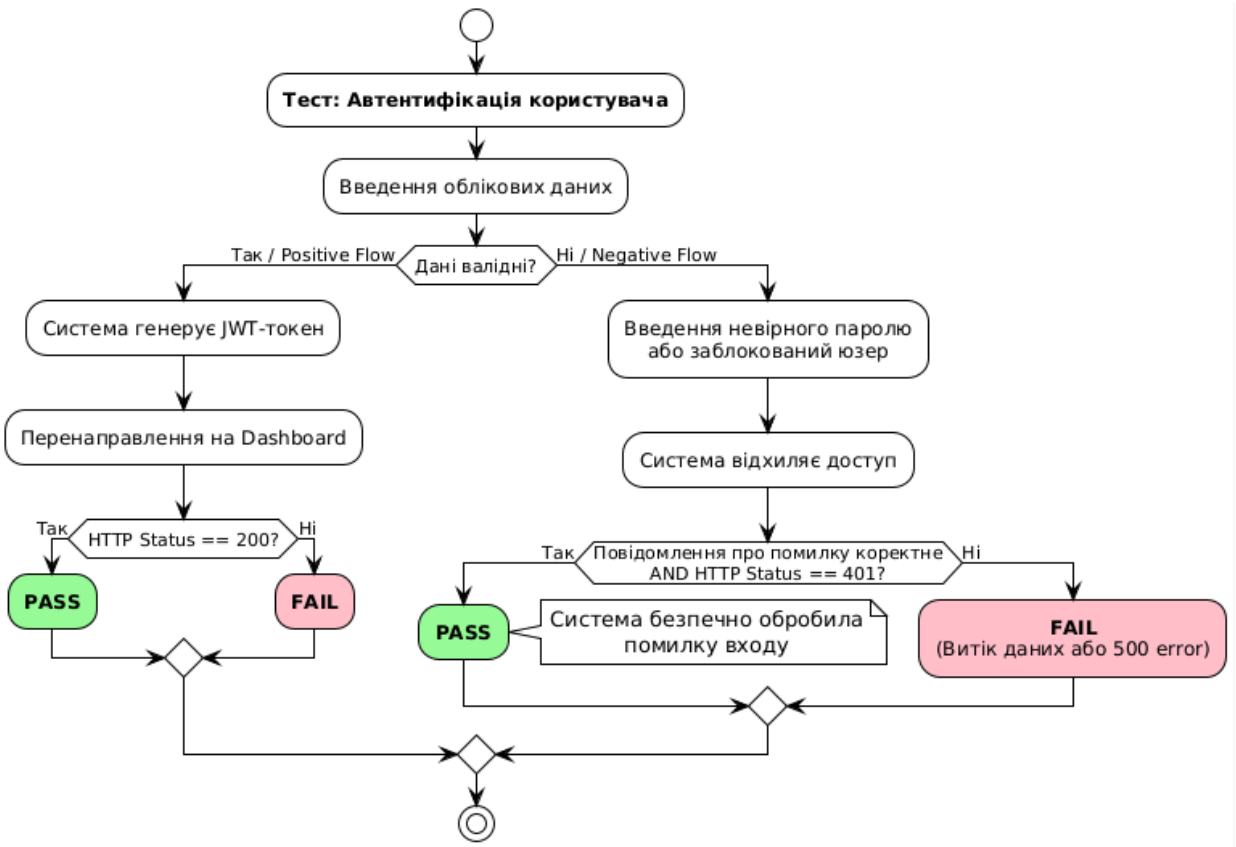


Рис. 3.5 – Діаграма діяльності перевірки сценаріїв автентифікації (позитивний та негативний кейси)

Сценарії перевірки функціоналу розгортання проєктів є найбільш критичними для оцінки працездатності системи. Типовий сценарій створення розгортання включає заповнення форми з вказівкою образу контейнера, змінних середовища та лімітів ресурсів. Очікуваним результатом є створення запису в базі даних PostgreSQL, успішна відправка запиту до API Kubernetes та поява нового об’єкта Deployment у кластері зі статусом «Running». Важливим аспектом є сценарії валідації вхідних даних, які передбачають введення недопустимих значень, наприклад, від’ємних лімітів пам’яті або некоректного формату URL репозиторію. У таких випадках система повинна блокувати відправку форми на рівні фронтенду або повертати структуровану помилку валідації з бекенду.

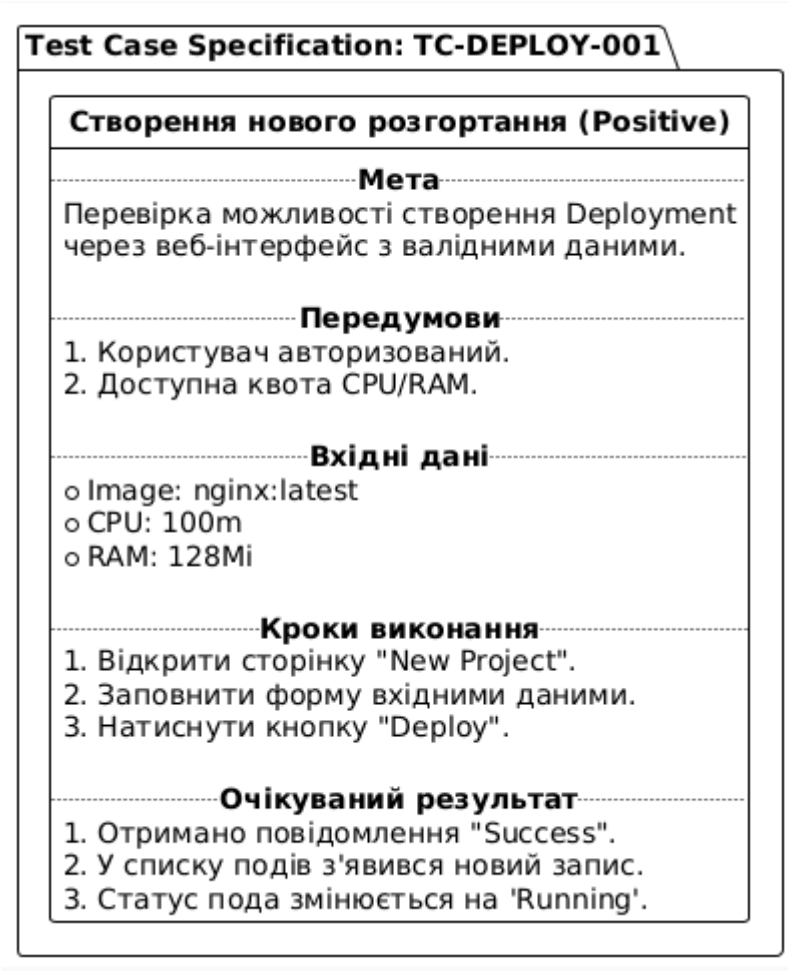


Рис. 3.6 – Приклад формалізованої карти тестового сценарію для функціоналу розгортання

Тестування підсистеми моніторингу вимагає сценаріїв, що перевіряють точність та своєчасність відображення даних. Сценарій передбачає генерацію штучного навантаження на тестовий контейнер та порівняння метрик, відображених на графіках веб-інтерфейсу, з еталонними значеннями, отриманими безпосередньо через консольні утиліти Kubernetes. Також перевіряється реакція інтерфейсу на відсутність даних або розрив з’єднання з сервером метрик, що має супроводжуватися відповідною візуальною індикацією, а не зависанням сторінки.

Окрему категорію складають сценарії тестування інтеграції з великою мовною моделлю. Сценарій починається з ініціації запиту на аналіз фрагмента логу, що містить відому помилку. Перевірці підлягає коректність санітизації даних

перед відправкою (відсутність конфіденційної інформації у запиті до зовнішнього API) та адекватність отриманої відповіді. Система повинна коректно обробляти випадки, коли API моделі недоступний або повертає порожню відповідь, відображаючи користувачеві повідомлення про тимчасову неможливість аналізу, замість системного виключення. Сукупність розроблених сценаріїв формує надійну базу для проведення системного тестування.

3.2 Розгортання програмної системи та системні вимоги

Етап впровадження розробленого програмного комплексу в експлуатацію вимагає чіткого визначення технічних умов функціонування та дотримання регламентованої процедури розгортання компонентів. Системні вимоги поділяються на апаратні, що стосуються обчислювальних потужностей сервера, та програмні, що визначають необхідне середовище виконання. Враховуючи архітектурні особливості системи, побудованої на базі екосистеми Python і платформи Django, бекенд-складова розрахована на роботу під керуванням UNIX-подібних операційних систем, зокрема дистрибутивів Linux (Ubuntu, CentOS, Debian). Для забезпечення стабільної роботи веб-сервера, сервера баз даних і фонових процесів обробки черг, мінімальна рекомендована конфігурація апаратного забезпечення включає двоядерний процесор із тактовою частотою не менше 2 ГГц, 4 ГБ оперативної пам’яті та 20 ГБ вільного дискового простору на твердотільному накопичувачі (SSD) для забезпечення високої швидкості операцій вводу-виводу. Критичною вимогою є наявність стабільного широкосмугового доступу до глобальної мережі з метою обміну даними з API кластера Kubernetes та зовнішніми сервісами великих мовних моделей. З метою забезпечення відмовостійкості системи та збереження цілісності даних необхідно налаштувати процедури регулярного автоматичного резервного копіювання бази даних на незалежні носії або хмарні сховища. Окрім того, для захисту інформаційного периметра взаємодія з усіма зовнішніми вузлами та клієнтськими додатками

повинна здійснюватися виключно через захищені канали зв’язку з використанням протоколів шифрування SSL/TLS.

Програмне оточення сервера повинно включати інтерпретатор мови Python версії 3.10 або новішої, а також систему керування пакунками pip для встановлення залежностей. Як систему керування базами даних обрано PostgreSQL версії 14 і вище, що забезпечує підтримку роботи з JSONB-полями, необхідними для гнучкого зберігання конфігурацій. Для організації черг асинхронних завдань вимагається встановлення брокера повідомлень Redis версії 6 або вище [29].

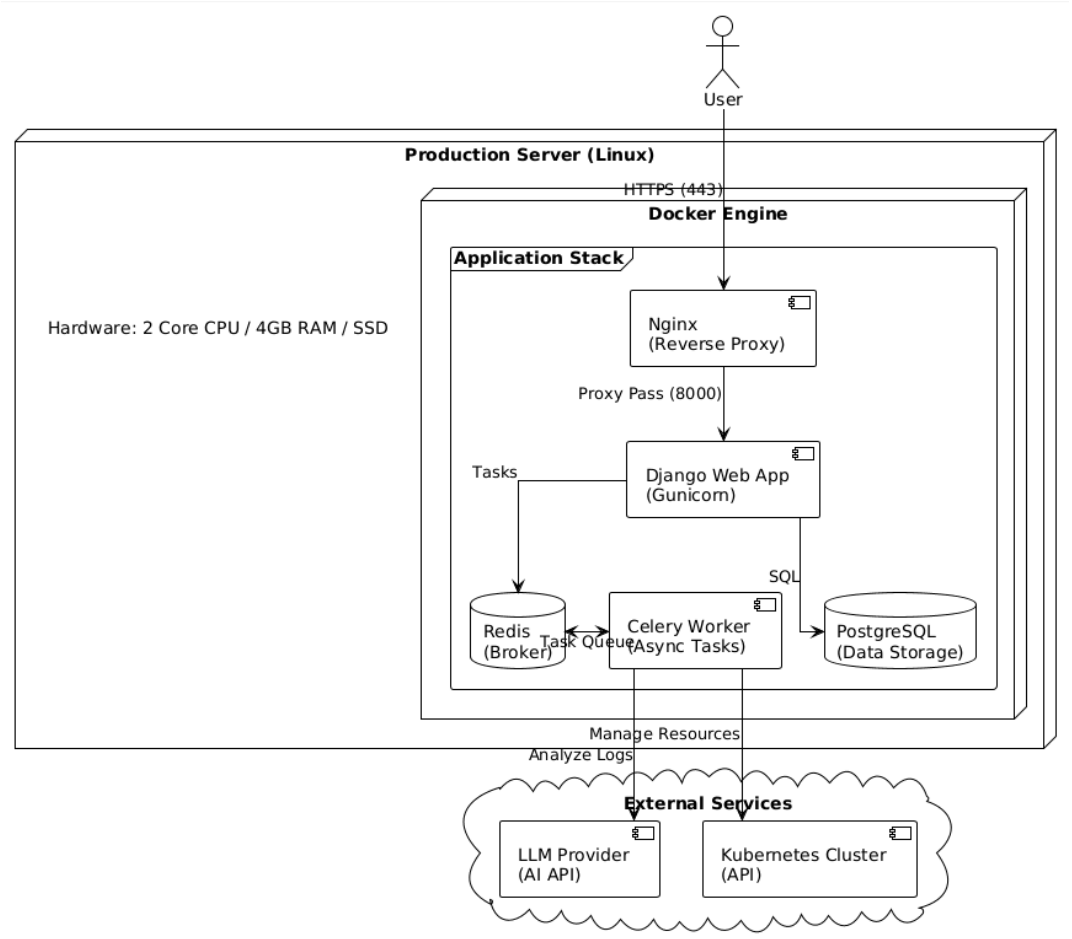


Рис. 3.7 – Діаграма розгортання компонентів системи у середовищі Docker

Процедура розгортання програмної системи реалізована з використанням методології контейнеризації. Процес розпочинається зі створення образу контейнера на основі інструкцій, описаних у Dockerfile. На цьому етапі відбувається перенесення програмних ресурсів, інсталяція пакетних залежностей

та регламентація прав доступу в межах ізольованого середовища. Визначення атрибутів з’єднання зі сховищем даних, приватних ключів API та налаштувань режиму налагодження здійснюється через механізм змінних середовища, що відповідає принципам методології "The Twelve-Factor App" і запобігає витоку конфіденційної інформації [30].

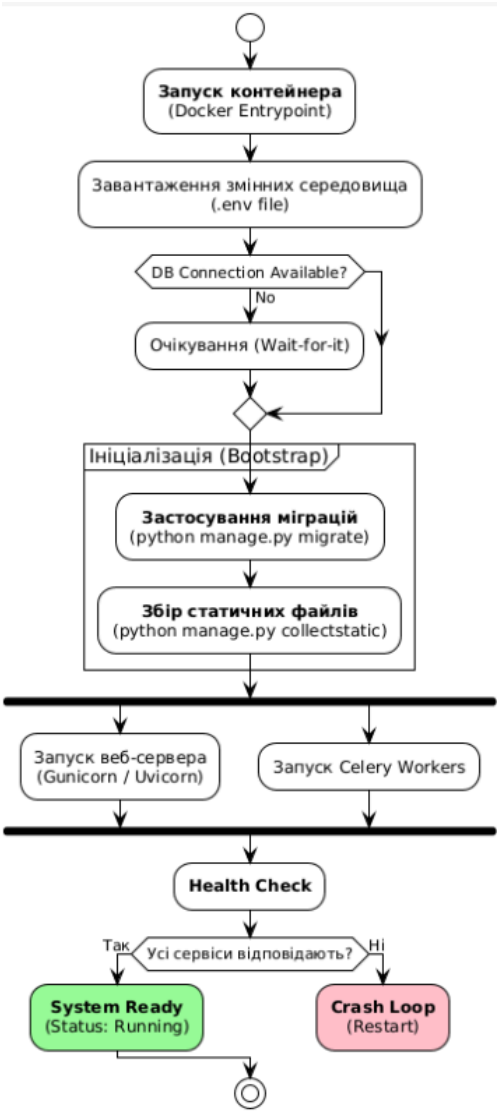


Рис. 3.8 – Алгоритм автоматичної ініціалізації компонентів системи при розгортанні

Безпосередній запуск системи передбачає виконання послідовності операцій ініціалізації. Після старту контейнерів автоматично виконується скрипт застосування міграцій бази даних, який синхронізує структуру таблиць із поточними моделями Django. Наступним кроком є збір статичних файлів (CSS,

JavaScript, зображення) в єдину директорію для їх ефективної роздачі веб-сервером Nginx або іншим реверс-проксі. Паралельно із основним процесом веб-застосунку ініціюються воркери Celery, які встановлюють з’єднання з брокером Redis та починають прослуховування черг завдань. Завершальним етапом розгортання є перевірка працездатності системи (Health Check), яка підтверджує доступність усіх компонентів та готовність до обробки запитів користувачів. Така стратегія розгортання забезпечує високий рівень автоматизації, відтворюваність результатів та мінімізацію часу простою при оновленні версій програмного забезпечення.

Для забезпечення безперервності процесу інтеграції та доставки (CI/CD) налаштовано автоматизований конвеєр, який ініціюється при оновленні основної гілки репозиторію коду. Цей механізм відповідає за автоматичне збирання нових версій образів контейнерів, їх тестування та завантаження до приватного реєстру (Container Registry). Оновлення продуктивного середовища відбувається за стратегією «Rolling Update», що передбачає поступову заміну старих екземплярів контейнерів на нові без переривання обслуговування активних користувацьких сесій. Такий підхід, у поєднанні з налаштованим моніторингом логів та метрик через централізовану систему, уможливлює своєчасну ідентифікацію та нейтралізацію відхилень, що можуть виникнути безпосередньо після розгортання оновлень, гарантуючи стабільність роботи комплексу. У випадку виявлення критичних помилок системою моніторингу, автоматично активується процедура відкату (rollback) до попередньої стабільної версії, що запобігає тривалим збоям у роботі сервісу. Така автоматизація процесів розгортання дозволяє суттєво скоротити цикл доставки програмного забезпечення (Time-to-Market), забезпечуючи при цьому високий рівень відмовостійкості інфраструктури.

3.3 Верифікація програмної системи

Верифікація розробленої програмної системи є заключним етапом контролю якості, спрямованим на офіційне підтвердження відповідності реалізованого

продукту вимогам технічного завдання, сформульованим на етапі аналізу предметної області. Цей процес відрізняється від тестування тим, що фокусується не на пошуку помилок, а на доведенні того, що система побудована правильно і реалізує передбачений спектр завдань у повному обсязі. Процедура верифікації охоплювала перевірку функціональної повноти, оцінку продуктивності, аналіз надійності архітектурних рішень та відповідність стандартам кодування. Для забезпечення об’єктивності результатів використовувався метод співставлення матриці вимог із фактично реалізованими можливостями системи, що дозволило документально зафіксувати досягнення поставлених цілей проєктування [31].

ID	Вимога (Requirement)	Метод верифікації	Результат
REQ-001	Створення K8s Deployment	Інтеграційний тест	Pass
REQ-002	Масштабування подів (Scale)	Системний тест	Pass
REQ-003	Збір метрик (CPU/RAM)	Порівняння з CLI	Pass
REQ-004	AI-аналіз інцидентів	Експертна оцінка	Pass
NFR-001	Час відгуку API < 300ms	Навантажувальний тест	Pass
NFR-002	Відповідність PEP 8	Статичний аналіз	Pass
SEC-001	Шифрування змінних (AES)	Аналіз коду	Pass

\

Рис. 3.9 – Фрагмент матриці відстеження вимог (RTM) із результатами верифікації

У ході верифікації функціональних вимог було підтверджено коректність реалізації механізмів взаємодії з кластером Kubernetes. Встановлено, що модуль керування інфраструктурою безпомилково трансформує користувацькі запити у маніфести розгортання, забезпечуючи створення, масштабування та видалення подів відповідно до заданих параметрів. Перевірка підсистеми моніторингу засвідчила точність збору та агрегації телеметричних даних, при цьому розходження між показниками, відображеними у веб-інтерфейсі, та даними, отриманими безпосередньо через інтерфейс командного рядка кластера, не перевищує допустимої похибки вимірювання. Особливу увагу було приділено верифікації інтелектуального компонента: аналіз вибірки згенерованих великою мовною моделлю звітів показав високий рівень релевантності рекомендацій, що підтверджує ефективність обраного підходу до інтеграції штучного інтелекту в процеси діагностики інцидентів. Додатково проведене тестування обробки

виключних ситуацій продемонструвало здатність системи коректно реагувати на тимчасову недоступність зовнішніх сервісів, запобігаючи критичним збоям у роботі основного додатка. У підсумку, результати функціональної верифікації свідчать про повну відповідність реалізованого програмного забезпечення затвердженим технічним вимогам та очікуванням кінцевих користувачів.

Оцінка нефункціональних характеристик підтвердила здатність системи забезпечувати стабільну роботу під навантаженням. Вимірювання часу відгуку інтерфейсу показало, що завдяки використанню асинхронних черг завдань середній час реакції на дії користувача залишається в межах комфортних значень навіть при виконанні тривалих операцій на стороні сервера. Верифікація підсистеми безпеки підтвердила надійність механізмів автентифікації та захисту конфіденційних даних: змінні середовища зберігаються у зашифрованому вигляді, а права доступу до проєктів коректно ізольовані між різними користувачами. Статичний аналіз програмного коду продемонстрував відповідність стандартам оформлення PEP 8, що є підтвердженням належного рівня архітектури та придатність системи до подальшого супроводу та розширення [32].

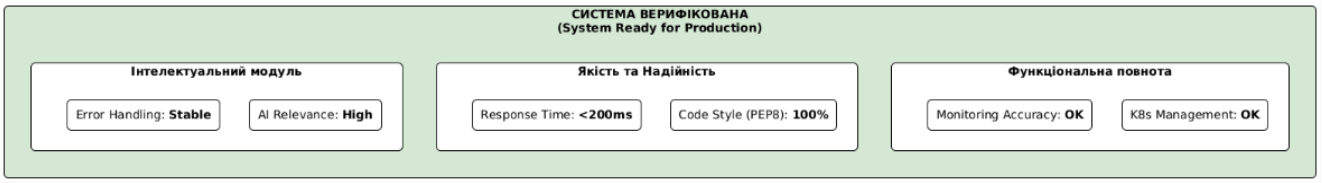


Рис. 3.10 – Підсумкова діаграма результатів верифікації компонентів системи

Таким чином, результати верифікації дозволяють стверджувати, що розроблена система є цілісним, працездатним програмним продуктом, готовим до впровадження у промислову експлуатацію.

4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

Охорона праці при розробці та експлуатації системи зосереджена на забезпеченні ергономічних вимог до робочого місця IT-фахівця, нормуванні зорового навантаження та дотриманні правил електробезпеки при роботі з серверним обладнанням. Заходи безпеки у надзвичайних ситуаціях включають розробку чітких інструкцій щодо евакуації персоналу та впровадження алгоритмів аварійного бекапу даних для захисту інформаційної інфраструктури у випадку пожежі чи техногенних аварій.

4.1 Охорона праці

Організація процесу модернізації системи менеджменту та моніторингу проєктів на серверах, що передбачала інтеграцію технологій Python, Django, Kubernetes та великих мовних моделей (LLM), вимагала створення безпечного виробничого середовища, що відповідає сучасним стандартам ергономіки та гігієни праці. Специфіка роботи розробника в рамках даного проєкту характеризувалася тривалою взаємодією з візуальними терміналами, високим рівнем концентрації уваги при написанні бекенд-коду, налаштуванні оркестрації контейнерів, а також значним інтелектуальним навантаженням під час навчання та інтеграції нейромережевих моделей. З огляду на це, пріоритетним завданням стало забезпечення умов праці, які мінімізують вплив шкідливих виробничих факторів та запобігають розвитку професійних захворювань.

Основним нормативним документом, що регламентує безпеку праці та ергономіку робочого місця при розробці складних програмних комплексів, є «Вимоги до безпеки та захисту здоров’я працівників під час роботи з екранними пристроями», затверджені наказом Міністерства соціальної політики України. Відповідно до положень цього документу, робоче місце інженера було спроектовано так, щоб робочий стіл та крісло дозволяли займати зручну робочу позу та змінювати її протягом робочого дня, а сидіння стільця регулювалося по

висоті [33]. Для роботи з інформаційними панелями моніторингу (Dashboard) та кодом використовувався рідкокристалічний монітор з діагоналлю 27 дюймів, поверхня якого не створювала відблисків, а зображення було стабільним і не мало мерехтінь, що є обов’язковою умовою згідно з вимогами до екрана [33]. Екран розташовувався на відстані, що становила від 600 до 700 міліметрів від очей працівника, що відповідає нормативним вимогам для запобігання перенапруженню зорового аналізатора, а клавіатура розміщувалася так, щоб у користувача була опора для рук і передпліч [33].

Загальна організація безпеки праці на проєкті базувалася на положеннях Закону України «Про охорону праці» [34]. Згідно зі статтею 13 цього Закону, роботодавець зобов’язаний створити на робочому місці умови праці відповідно до нормативно-правових актів, а також забезпечити додержання вимог законодавства щодо прав працівників у галузі охорони праці. Тому перед початком робіт з розгортання серверної інфраструктури було проведено вступний інструктаж з питань охорони праці, а саме робоче місце було організовано з урахуванням вимог електробезпеки: всі струмопровідні частини обладнання були надійно ізольовані, а корпуси комп'ютерної техніки — заземлені, що гарантувало захист від ураження електричним струмом під час експлуатації потужних обчислювальних засобів [34].

Ключовим фактором, що визначає рівень зорового комфорту при роботі з програмним кодом, конфігураційними файлами YAML та логами системи, є освітлення. Параметри світлового середовища на робочому місці були приведені у відповідність до державних будівельних норм ДБН В.2.5-28:2018 «Природне і штучне освітлення». У приміщенні застосовувалася система комбінованого освітлення, де рівень освітленості робочої поверхні підтримувався в межах 500 люкс, що є нормативним показником для робіт високої точності, до яких належить програмування та адміністрування систем [35]. Для обмеження прямої блискучості від джерел світла світильники були розташовані так, щоб їхні світлові потоки не потрапляли безпосередньо в очі працівника. Крім того, світлодіодні джерела світла мали індекс кольоропередачі CRI понад 80, що забезпечувало

комфортне сприйняття візуальної інформації, як того вимагають норми для приміщень з тривалим перебуванням людей [35].

Мікроклімат у приміщенні суттєво впливає на теплообмін організму та загальне самопочуття, особливо враховуючи тепловиділення від обладнання під час тренування або інференсу LLM-моделей. Під час виконання роботи параметри мікроклімату, такі як температура, вологість та швидкість руху повітря, підтримувалися на рівні комфортних значень. Відповідно до «Вимог до безпеки та захисту здоров’я працівників під час роботи з екранними пристроями», на робочих місцях було забезпечено дотримання значень параметрів мікроклімату згідно з чинними санітарними нормами, а надлишкове тепло, що виділяється робочою станцією, компенсувалося ефективною системою вентиляції та не спричиняло дискомфорту працівникам.

Пожежна безпека на об’єкті розробки регламентувалася Кодексом цивільного захисту України. Оскільки робоче місце насичене електронним обладнанням, необхідним для підтримки середовищ віртуалізації, основним заходом профілактики було дотримання правил експлуатації електроустановок для запобігання коротким замиканням. Приміщення було забезпечене первинними засобами пожежогасіння, а саме вуглекислотним вогнегасником, придатним для гасіння електрообладнання під напругою. Відповідно до вимог законодавства, суб’єкт господарювання забезпечив виконання заходів пожежної безпеки, спрямованих на запобігання виникненню пожеж та створення умов для швидкої евакуації людей у разі небезпеки [36].

Окремим важливим аспектом охорони праці стала організація режиму праці та відпочинку для профілактики гіподинамії та зорової втоми, які можуть виникати при тривалому моніторингу метрик кластера Kubernetes. Згідно з «Вимогами до безпеки та захисту здоров’я працівників під час роботи з екранними пристроями», роботодавець зобов’язаний планувати діяльність працівника так, щоб щоденна робота з екраном періодично переривалася. Тому щогодини робилися регламентовані перерви тривалістю 10–15 хвилин, під час яких виконувалися вправи для зняття напруги з очей та розвантаження

опорно-рухового апарату, що дозволило зберегти високу працездатність та ефективність при вирішенні архітектурних завдань протягом усього періоду виконання проєкту.

4.2 Фактори виробничого середовища і їх вплив на життєдіяльність промислово-виробничого персоналу.

В умовах сучасної промисловості професійна діяльність виробничого персоналу все частіше характеризується формуванням специфічного середовища, де домінуючу роль відіграє взаємодія людини зі складними автоматизованими системами управління та контролю. Виробничі процеси передбачають тривале перебування операторів, диспетчерів та інженерів на робочих місцях, оснащених електронними засобами відображення інформації, що супроводжується впливом комплексу шкідливих та небезпечних факторів: від психофізіологічного напруження до електромагнітного випромінювання промислового обладнання. Їхня сукупна дія здатна викликати стійкі функціональні зрушення в організмі працівників. Аналіз впливу виробничого середовища на здоров’я персоналу є необхідною передумовою для розробки ефективних заходів охорони праці.

Першим і найбільш вираженим фактором, що визначає специфіку праці операторів автоматизованих ліній та пультів управління, є високе напруження зорового аналізатора. Моніторинг технологічних процесів, зчитування показників датчиків та робота з технічною документацією вимагають розрізнення дрібних об’єктів та символів в умовах штучного освітлення та світіння індикаторів. Це призводить до інтенсивної експлуатації апарату акомодації та конвергенції ока.

Відповідно до «Мінімальних вимог безпеки та охорони здоров’я під час роботи з екранними пристроями», затверджених наказом Міністерства соціальної політики України, така діяльність класифікується як зорово-напружена та пов’язана з ризиком розвитку загального стомлення. Тривала фіксація погляду на засобах відображення інформації викликає зниження частоти мимовільного моргання, що призводить до порушення стабільності слізної плівки (синдром

«сухого ока»). Постійна необхідність перефокусування погляду між пультом управління, екранами моніторингу та виробничим обладнанням викликає втому циліарного м’яза. У сукупності ці явища формують професійно зумовлені порушення зору, що проявляються затуманенням, болем в очах та головним болем. Нормативні документи наголошують на необхідності суворого дотримання режимів праці та відпочинку для запобігання незворотним змінам зору [37].

Вагомим блоком факторів впливу є ергономічні аспекти організації робочого місця промислового персоналу та вимушена гіпокінезія. Специфіка роботи операторів часто вимагає фіксованої робочої пози (сидячи або стоячи) протягом зміни. Це створює значне статичне навантаження на опорно-руховий апарат, зокрема на хребетний стовп та м’язи тулуба.

Згідно з положеннями національного стандарту ДСТУ EN ISO 9241-5:2018 «Ергономіка взаємодії людина-система», невідповідність параметрів робочого місця (висоти пульта, конструкції крісла) антропометричним даним працівника є ключовим фактором ризику виникнення скелетно-м’язових розладів.

Тривале перебування у вимушеній позі призводить до перерозподілу навантаження на міжхребцеві диски, що створює передумови для розвитку остеохондрозу та протрузій. Крім того, статичне напруження м’язів шиї та плечового поясу, необхідне для спостереження за виробничим процесом, може викликати погіршення церебрального кровообігу, що проявляється запамороченням та зниженням уваги — критично важливого фактора для безпеки виробництва.

Окрему загрозу становить специфічне навантаження на верхні кінцівки при роботі з органами ручного управління (клавіатурами, джойстиками, важелями). Виконання стереотипних рухів може призвести до напруження сухожиль та розвитку тунельних синдромів. Застійні явища у судинній системі нижніх кінцівок також підвищують ризик професійних патологій [38].

Третя група факторів пов’язана з фізичними характеристиками середовища у виробничих приміщеннях, серед яких особливе місце посідають електромагнітні випромінювання та мікроклімат. Промислові об’єкти насичені потужним

електрообладнанням, силовими кабелями, трансформаторами та серверними шафами, які генерують електромагнітні поля (ЕМП) промислової частоти та радіочастотного діапазону.

Відповідно до «Державних санітарних норм і правил захисту населення від впливу електромагнітних випромінювань», персонал, що обслуговує такі системи, піддається хронічному впливу ЕМП. Біологічна дія цих полів впливає на нервову, імунну та ендокринну системи, що може викликати підвищену дратівливість, порушення сну та зниження концентрації уваги.

Робота електронного та електричного обладнання впливає на аеройонний склад повітря у приміщеннях. Електростатичні поля сприяють зміні балансу легких та важких йонів («електричне забруднення» повітря), що може призводити до погіршення легеневого газообміну та зниження імунітету працівників. Зазначені санітарні норми регламентують гранично допустимі рівні напруженості полів, перевищення яких створює пряму загрозу здоров’ю персоналу [39].

Психофізіологічний аспект впливу не можна ігнорувати. Праця промислово-виробничого персоналу часто пов’язана з високою відповідальністю за безаварійну роботу обладнання та безпеку людей. Необхідність швидкого прийняття рішень у нестандартних ситуаціях, монотонність спостереження за приладами або інформаційні перевантаження викликають значне нервово-емоційне напруження. Хронічний стрес виснажує адаптаційні ресурси організму, що проявляється у зниженні надійності дій оператора, збільшенні ймовірності помилок та виробничого травматизму.

Вплив вищезазначених факторів формує загальний рівень працездатності персоналу. Дослідження показують, що робота в несприятливих умовах знижує продуктивність та підвищує ризики аварійних ситуацій. Тому створення оптимального виробничого середовища є не лише вимогою законодавства, але й економічно обґрунтованою необхідністю [40, 41].

ВИСНОВКИ

У дипломній роботі успішно виконано завдання модернізації системи менеджменту та моніторингу проєктів на серверах шляхом розробки комплексного програмного продукту. Результати дослідження засвідчили, що інтеграція сучасних інструментів оркестрації контейнерів із можливостями генеративного штучного інтелекту дозволяє кардинально змінити підходи до системного адміністрування, мінімізуючи ризики, пов’язані з помилками операторів, та посилюючи відмовостійкість інфраструктури.

На етапі системного аналізу предметної області було виявлено критичні недоліки традиційних методів керування серверними потужностями, зокрема складність масштабування та високий поріг входження для нових спеціалістів. На основі цього було сформульовано технічне завдання та визначено ключові вимоги до системи, які включали необхідність централізованого керування життєвим циклом застосунків, автоматизацію розподілу ресурсів та впровадження інтелектуальних засобів діагностики. Моделювання варіантів використання дозволило чітко розмежувати зони відповідальності між ролями адміністраторів та розробників, забезпечивши гранулярний контроль доступу до функціоналу.

В ході проєктування архітектури було аргументовано доцільність застосування мови Python у поєднанні з фреймворком Django. Такий вибір забезпечив високу швидкість розробки завдяки використанню патерну MVT та вбудованих механізмів безпеки. Розроблена схема бази даних, реалізована на PostgreSQL, гарантує цілісність збереження конфігураційних даних проєктів та користувацьких квот, а використання JSON-полів забезпечило необхідну гнучкість для зберігання динамічних параметрів розгортання. Об’єктно-орієнтоване проєктування класів дозволило створити модульну структуру коду, де логіка взаємодії з Kubernetes та модулі штучного інтелекту інкапсульовані в окремі сервіси, що спрощує подальшу підтримку та масштабування системи.

Практична реалізація системи охопила створення надійних механізмів інтеграції з API кластера Kubernetes, що дозволило автоматизувати процеси

створення (Deployment), масштабування та видалення контейнерів безпосередньо через веб-інтерфейс. Ключовим аспектом реалізації стало створення підсистеми інтелектуальної аналітики, що базується на застосуванні великих мовних моделей (LLM) задля опрацювання логів помилок. Це інноваційне рішення дозволило не лише фіксувати інциденти, але й надавати користувачам автоматизовані рекомендації щодо їх усунення, значно скорочуючи час реакції на збої (Mean Time To Repair).

Розробка зручного інтерфейсу користувача з використанням сучасних веб-технологій дозволила візуалізувати складні метрики використання CPU та RAM у зрозумілому графічному вигляді. Впровадження асинхронної обробки важких завдань за допомогою черг Celery та брокера Redis забезпечило високу чуйність системи, дозволяючи виконувати тривалі операції розгортання у фоновому режимі без блокування основного інтерфейсу.

Комплексна стратегія тестування, що включала модульні, інтеграційні та навантажувальні випробування, підтвердила стабільність роботи програмного комплексу. Було верифіковано коректність роботи алгоритмів санітизації даних перед їх відправкою до LLM, а також стійкість системи до пікових навантажень. Процедура розгортання, побудована на принципах контейнеризації Docker, гарантує ідентичність середовищ розробки та експлуатації, спрощуючи процес доставки оновлень.

Підсумовуючи, можна стверджувати, що створена система є повнофункціональним рішенням, яке повністю відповідає поставленим вимогам. Вона вирішує актуальні проблеми автоматизації DevOps-процесів, забезпечує високий рівень безпеки та пропонує інноваційний підхід до моніторингу, що робить її готовою до впровадження у реальних виробничих умовах.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Романюк О. Н., Савчук Т. О. Архітектурні особливості побудови високонавантажених розподілених програмних систем. Вісник Вінницького політехнічного інституту. 2021. № 2. С. 56–62.

2. Bryk O., Mudryk I., Holubovskyi M., Stoianov Y. Machine learning models and methods aspects of processing unstructured data. Proceedings of the 1st International Workshop on Bioinformatics and Applied Information Technologies (BAIT 2024). Zboriv, 2024. P. 64–74.

3. Hou X., Zhao Y., Liu Y., Yang Z. Large Language Models for Software Engineering: A Survey. ACM Transactions on Software Engineering and Methodology. 2024. Vol. 33, Issue 2. P. 1–34.

4. Шевченко В. Л., Шила О. О. Аналіз методів підвищення надійності хмарних обчислювальних середовищ. Системи керування, навігації та зв’язку. 2020. Вип. 2 (60). С. 112–116.

5. Yaroslavivna N., Mykhailivna N., Orestivna L., Mudryk I. Radial-basis neural networks for forecasting the activity of enterprises. European Science. 2023. No. sge17-03. P. 42–48.

6. Петрик М. Р., Мудрик І. Я. Проєктування програмного забезпечення на основі об’єкнто-орієнтованого аналізу вимог та інструментальних засобів розробки IBM Rational Software Architect. Вісник ТНТУ ім. І. Пулюя. 2022.

7. Yuan Z. et al. LLM for DevOps: A Comprehensive Survey of Large Language Models for Software Development and Operations. arXiv preprint arXiv:2403.04944. 2024.

8. Романюк Д., Мудрик І. Я. Вплив сучасних технологій на бухгалтерський облік. Інформаційні моделі, системи та технології : Матеріали XII наук.-техн. конф. Тернопіль : ТНТУ, 2024. С. 188–189.

9. Lee M., Kim S. LogGPT: Exploring ChatGPT for Log-Based Anomaly Detection and Root Cause Analysis. Proceedings of the 34th International Symposium on Software Reliability Engineering (ISSRE). IEEE, 2023. P. 12–23.

10. Sommerville I. Software Engineering. 10th ed. Pearson, 2015. 816 p.

11. Schwaber K., Sutherland J. The Scrum Guide: The Definitive Guide to Scrum: The Rules of the Game. Scrum.org, 2020. 18 p.

12. Kim G., Humble J., Debois P., Willis J. The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations. IT Revolution Press, 2016. 480 p.

13. Глибовець М. М., Олецький О. В. Розподілені системи та мережі : навч. посіб. Київ : ВД «Києво-Могилянська академія», 2018. 320 с.

14. Shum H., He X., Li D. From Large Language Models to Intelligent Agents: An Architecture Perspective. Engineering. 2024. Vol. 35. P. 15–28.

15. Берко А. Ю., Верес О. М., Пасічник В. В. Системи баз даних та знань : підручник. Львів : Магнолія-2006, 2018. 680 с.

16. Kleppmann M. Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems. O'Reilly Media, 2017. 616 p.

17. Ткачук В. М. Об'єктно-орієнтоване моделювання програмних систем : навч. посіб. Івано-Франківськ : ІФНТУНГ, 2020. 264 с.

18. Percival H., Gregory B. Architecture Patterns with Python: Enabling Test-Driven Development, Domain-Driven Design, and Event-Driven Microservices. O'Reilly Media, 2020. 300 p.

19. Коноваленко І. В., Марущак Т. В. Програмування мовою Python : навч. посіб. Тернопіль : ТНТУ імені Івана Пулюя, 2016. 268 с.

20. Гнатюк С. О. Сучасні технології веб-програмування : навч. посіб. Київ : НАУ, 2019. 284 с.

21. Gift N., Behrman K. Python for DevOps: Learn Ruthlessly Effective Automation. O'Reilly Media, 2020. 200 p.

22. Мелешко Є. В., Якименко М. С. Розробка веб-застосунків мовою Python : навч. посіб. Київ : КПІ ім. Ігоря Сікорського, 2019. 150 с.

23. Кульпа Ю. С. Проектування користувацьких інтерфейсів : навч. посіб. Тернопіль : ТНТУ, 2019. 164 с.

24. Haverbeke M. Eloquent JavaScript: A Modern Introduction to Programming. 3rd ed. No Starch Press, 2018. 472 p.

25. Сидорченко М. В. Тестування та контроль якості програмних продуктів : конспект лекцій. Київ : НАУ, 2021. 64 с.

26. Percival H. Test-Driven Development with Python: Obey the Testing Goat: Using Django, Selenium, and JavaScript. 2nd ed. O'Reilly Media, 2017. 662 p.

27. Федорович О. Є., Сьомкін М. В. Методи та засоби тестування програмного забезпечення : навч. посіб. Харків : "ХАІ", 2020. 84 с.

28. Stuttard D., Pinto M. The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws. 2nd ed. Wiley, 2011. 912 p.

29. Криворучко О. В. Хмарні технології : навч. посіб. Харків : ХНАДУ, 2020. 180 с.

30. Wiggins A. The Twelve-Factor App. Heroku, 2011. URL: https://12factor.net/ (дата звернення: 16.12.2025).

31. Лавріщева К. М. Програмна інженерія : підручник. Київ : Академперіодика, 2008. 319 с.

32. Van Rossum G., Warsaw B., Coghlan N. PEP 8 – Style Guide for Python Code. Python.org. 2001. URL: https://peps.python.org/pep-0008/ (дата звернення: 16.12.2025).

33. Вимоги до безпеки та захисту здоров’я працівників під час роботи з екранними пристроями : затв. наказом М-ва соц. політики України від 14.02.2018 № 207. URL: https://zakon.rada.gov.ua/laws/show/z0508-18 (дата звернення: 01.12.2025).

34. Про охорону праці : Закон України від 14.10.1992 № 2694-XII : станом на 01 січ. 2025 р. / Верховна Рада України. URL: https://zakon.rada.gov.ua/laws/show/2694-12 (дата звернення: 01.12.2025).

35. ДБН В.2.5-28:2018. Природне і штучне освітлення. Київ : Мінрегіон України, 2018. 133 с.

36. Кодекс цивільного захисту України : Закон України від 02.10.2012 № 5403-VI : станом на 01 січ. 2025 р. / Верховна Рада України. URL: https://zakon.rada.gov.ua/laws/show/5403-17 (дата звернення: 01.12.2025).

37. Мінімальні вимоги безпеки та охорони здоров’я під час роботи з екранними пристроями : затв. наказом М-ва соц. політики України від 14.02.2018 № 207. URL: https://zakon.rada.gov.ua/laws/show/z0406-18 (дата звернення: 08.12.2025).

38. ДСТУ EN ISO 9241-5:2018 (EN ISO 9241-5:1999, IDT; ISO 9241-5:1998, IDT). Ергономічні вимоги до роботи з відеотерміналами в офісі. Частина 5. Вимоги до розташування робочого місця та постави. Київ : ДП «УкрНДНЦ», 2018. 28 с.

39. Державні санітарні норми і правила захисту населення від впливу електромагнітних випромінювань : затв. наказом М-ва охорони здоров’я України від 01.08.1996 № 239 : у редакції наказу МОЗ України від 03.04.2017 № 356. URL: https://zakon.rada.gov.ua/laws/show/z0488-96 (дата звернення: 08.12.2025).

40. Стручок В.С. Методичний посібник для здобувачів освітнього ступеня «магістр» всіх спеціальностей денної та заочної (дистанційної) форм навчання «БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ» / Тернопіль: ФОП Паляниця В. А., –156 с. Отримано з https://elartu.tntu.edu.ua/handle/lib/39196.

41. Стручок В.С. Навчальний посібник «ТЕХНОЕКОЛОГІЯ ТА ЦИВІЛЬНА БЕЗПЕКА. ЧАСТИНА «ЦИВІЛЬНА БЕЗПЕКА»» / Тернопіль: ФОП Паляниця В. А., – 156 с. Отримано з http://elartu.tntu.edu.ua/handle/lib/39424

42. Методичні вказівки до виконання кваліфікаційної роботи магістра для здобувачів спеціальності 121 – Інженерія програмного забезпечення, всіх форм навчання / укладачі Михалик Д.М., Цуприк Г.Б., Бревус В.М., Мудрик І.Я. – Тернопіль: Тернопільський національний технічний університет імені Івана Пулюя, 2024. – 44 с.

ДОДАТКИ

ДОДАТОК А

Тези для XIII науково-технічна конференція «Інформаційні моделі, системи та технології»

УДК 004.41
Лань О. – ст. гр. СПм-61
Тернопільський національний технічний університет імені Івана Пулюя

МОДЕРНІЗАЦІЯ СИСТЕМИ МЕНЕДЖМЕНТУ ТА МОНІТОРИНГУ ПРОЄКТІВ НА СЕРВЕРАХ З ВИКОРИСТАННЯМ ТЕХНОЛОГІЙ Python, Django, Kubernetes ТА LLM

Науковий керівник: д. ф., с. в. Мудрик І. Я.

Lan O.
Ternopil Ivan Puluj National Technical University

MODERNIZATION OF PROJECT MANAGEMENT AND SERVER MONITORING SYSTEM USING Python, Django, Kubernetes AND LLM TECHNOLOGIES

Supervisor: PhD, Senior Lecturer Mudryk I. Y.

Ключові слова: моніторинг серверів, оркестрація, штучний інтелект
Keywords: server monitoring, orchestration, artificial intelligence

Сучасні підходи до інженерії програмного забезпечення вимагають інтеграції процесів управління проєктами з інструментами безперервного моніторингу серверної інфраструктури. Модернізація таких систем передбачає створення адаптивних платформ, здатних не лише фіксувати стан виконання завдань, а й автоматизувати управління обчислювальними ресурсами. Технічну основу запропонованого рішення складає поєднання гнучкості Python і Django, потужності оркестратора Kubernetes[1] та аналітичних можливостей великих мовних моделей (LLM).

У розробленій архітектурі Django виступає центральним вузлом для агрегації даних та реалізації API, тоді як Kubernetes забезпечує автоматизоване масштабування контейнеризованих додатків та контроль метрик у реальному часі. Впровадження LLM трансформує систему в засіб інтелектуальної підтримки, дозволяючи виявляти аномалії в логах, генерувати конфігураційні файли для налаштування середовища та надавати зведені звіти природною мовою. Комплексне застосування зазначених технологій знижує операційні ризики, підвищує прозорість процесів та оптимізує використання серверних потужностей

Література

Сайт Kubernetes [Електронний ресурс] – Режим доступу до ресурсу: https://kubernetes.io

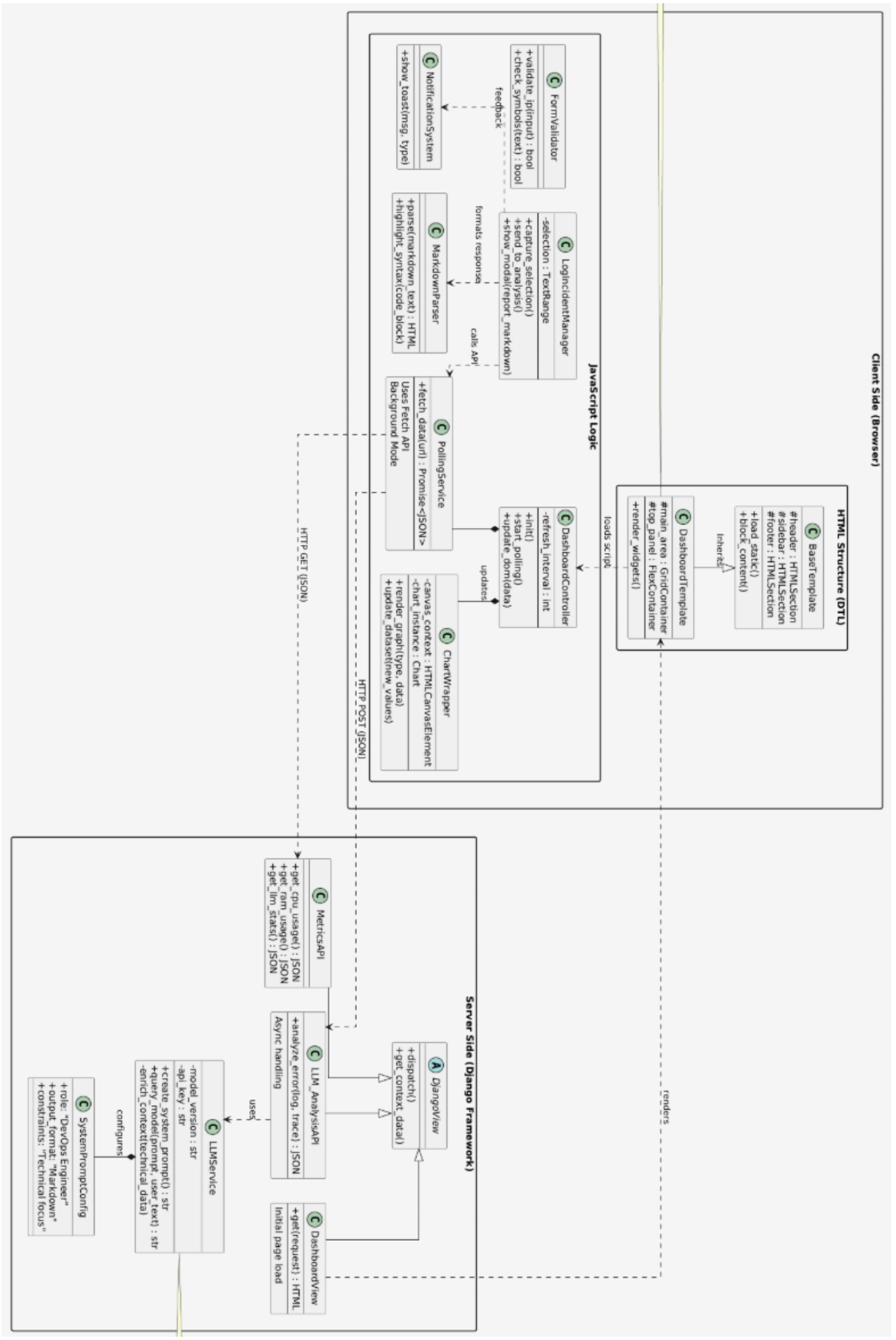
Komodor. K8sGPT: Improving K8s Cluster Management with LLMs [Електронний ресурс]. – Режим доступу: https://komodor.com/learn/k8sgpt-improving-k8s-cluster-management-with-llms/

ДОДАТОК Б
РЕПОЗИТОРІЙ

URL:https://github.com/OrestLemish/-diploma-project.

ДОДАТОК В

Діаграма класів підсистеми користувацького інтерфейсу та моніторингу



ДОДАТОК Г

Діаграма класів Серверної частини і бізнес логіки