Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук

Бакалаврська кваліфікаційна робота на тему:

РОЗРОБКА WEB-ДОДАТКУ ДЛЯ КЕРУВАННЯ СЕРВЕРНИМ СЕРЕДОВИЩЕМ КОРПОРАТИВНОЇ ІНФОРМАЦІЙНОЇ СИСТЕМИ

Виконала: студентка 4 курсу, групи 2КН-21б

спеціальності 122 Комп'ютерні науки
(шифр і назва напрямку підготовки, спеціальності)

Ткаченко Д.О.
(прізвище та ініціали)

Керівник: к.т.н., доц, асистент каф.КН
Богач І.В.
(прізвище та ініціали)

«04» червня 2025 р.

Рецензент: д.т.н., професор каф.АІІТ
Квєтний Р.Н.
(прізвище та ініціали)

«05» червня 2025 р.

Допущено до захисту
Зав. кафедри Яровий А.А.
«06» червня 2025 р.

Вінниця ВНТУ – 2025 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук
Рівень вищої освіти перший (бакалаврський)
Галузь знань – 12 Інформаційні технології
Спеціальність – 122 Комп'ютерні науки
Освітньо-професійна програма – Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
проф., д.т.н. Яровий А.А.
«05» травня 2025р.

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Ткаченко Дар'ї Олексіївні

1. Тема роботи: Розробка WEB-додатку для керування серверним середовищем корпоративної інформаційної системи.
Керівник роботи: Богач Ілона Віталіївна, к.т.н., доцент, асистент каф.КН
Затверджені наказом вищого навчального закладу "20" березня 2025року №97
2. Термін подання студентом роботи 30 травня 2025 р
3. Вихідні дані до роботи: об'єктно-орієнтована мова програмування; API для взаємодії з системними характеристиками; Інтерфейс взаємодії веб-додатку та Docker; інтерфейс користувача має бути інтуїтивно зрозумілий, масштабована та продуктивна інфраструктура
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): Аналіз предметної області моніторингу та керування серверної інфраструктури; Проєктування структури та компонентів web-додатку; Програмна реалізація web-додатку для керування серверним середовищем корпоративної інформаційної системи.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень): UML-діаграма компонентів WEB-додатку для керування серверним середовищем корпоративної інформаційної системи; Загальна структурна схема компонентів WEB-додатку для керування серверним середовищем корпоративної інформаційної системи; алгоритм роботи WEB-додатку для керування серверним середовищем корпоративної інформаційної системи; відображення показників, які вказують відсоток завантаження swap та загального простору диску; відображення системних параметрів операційної системи; відображення показників обсягу використаної RAM та загального заповнення пам'яті; відображення показників кількості запущених процесів та часу роботи серверу; загальний вигляд інтерфейсу, що відображає розгорнуті проєкти; приклад роботи програми.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-3	Богач І.В., к.т.н., доц!, асистент каф.КН		

7. Дата видачі завдання 05 травня 2025 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва та зміст етапу	Термін виконання		Примітка
		Початок	Закінчення	
1	Аналіз предметної області моніторингу та керування серверної інфраструктури	05.05.25	07.05.25	викон.
2	Проєктування структури та компонентів WEB-додатку для керування серверним середовищем корпоративної інформаційної системи	08.05.25	11.05.25	викон.
3	Програмна реалізація WEB-додатку для керування серверним середовищем корпоративної інформаційної системи	12.05.25	15.05.25	викон.
4	Тестування WEB-додатку для керування серверним середовищем корпоративної інформаційної системи	16.05.25	28.05.25	викон.
5	Оформлення пояснювальної записки	25.05.25	29.05.25	викон.
6	Попередній захист БКР та доопрацювання	30.05.25	01.06.25	викон.
7	Захист БКР	02.06.25	04.06.25	викон.

Студент _____ **Ткаченко Д.О.**

(підпис) (прізвище та ініціали)

Керівник БКР _____ **Богач І.В.**

(підпис) (прізвище та ініціали)

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук

Бакалаврська кваліфікаційна робота на тему:

РОЗРОБКА WEB-ДОДАТКУ ДЛЯ КЕРУВАННЯ СЕРВЕРНИМ СЕРЕДОВИЩЕМ КОРПОРАТИВНОЇ ІНФОРМАЦІЙНОЇ СИСТЕМИ

Виконала: студентка 4 курсу, групи 2КН-21б

спеціальності 122 Комп'ютерні науки
(шифр і назва напрямку підготовки, спеціальності)

Ткаченко Д.О.
(прізвище та ініціали)

Керівник: к.т.н., доцент
Богач І.В.
(прізвище та ініціали)

«____»_____2025 р.

Рецензент: д.т.н., професор каф.АІІТ
Квєтний Р.Н.
(прізвище та ініціали)

«____»_____2025 р.

Допущено до захисту
Зав. кафедри _____
«____»_____2025 р.

Вінниця ВНТУ – 2025 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук
Рівень вищої освіти перший (бакалаврський)
Галузь знань – 12 Інформаційні технології
Спеціальність – 122 Комп'ютерні науки
Освітньо-професійна програма – Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
проф., д.т.н. Яровий А.А.
«____»_____2025р.

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ
Ткаченко Дар'ї Олексіївні

1. Тема роботи: Розробка WEB-додатку для керування серверним середовищем корпоративної інформаційної системи.
Керівник роботи: Богач Ілона Віталіївна, к.т.н., доц.
Затверджені наказом вищого навчального закладу "__"_____20__року №__
2. Термін подання студентом роботи_____ _____
3. Вихідні дані до роботи: об'єктно-орієнтована мова програмування; API для взаємодії з системними характеристиками; Інтерфейс взаємодії веб-додатку та Docker; інтерфейс користувача має бути інтуїтивно зрозумілий, масштабована та продуктивна інфраструктура
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): Аналіз предметної області моніторингу та керування серверної інфраструктури; Проєктування структури та компонентів web-додатку; Програмна реалізація web-додатку для керування серверним середовищем корпоративної інформаційної системи.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень): UML-діаграма компонентів WEB-додатку для керування серверним середовищем корпоративної інформаційної системи; Загальна структурна схема компонентів WEB-додатку для керування серверним середовищем корпоративної інформаційної системи; алгоритм роботи WEB-додатку для керування серверним середовищем корпоративної інформаційної системи; відображення показників, які вказують відсоток завантаження swap та загального простору диску; відображення системних параметрів операційної системи; відображення показників обсягу використаної RAM та загального заповнення пам'яті; відображення показників кількості запущених процесів та часу роботи серверу; загальний вигляд інтерфейсу, що відображає розгорнуті проєкти; приклад роботи програми.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-3	Богач І.В., к.т.н., доц., ассистент каф.КН		

7. Дата видачі завдання_____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва та зміст етапу	Термін виконання		Примітка
		Початок	Закінчення	
1	Аналіз предметної області моніторингу та керування серверної інфраструктури			
2	Проєктування структури та компонентів WEB-додатку для керування серверним середовищем корпоративної інформаційної системи			
3	Програмна реалізація WEB-додатку для керування серверним середовищем корпоративної інформаційної системи			
4	Тестування WEB-додатку для керування серверним середовищем корпоративної інформаційної системи			
5	Оформлення пояснювальної записки			
6	Попередній захист БКР та доопрацювання.			
7	Захист БКР			

Студент _____ Ткаченко Д.О.

(підпис) (прізвище та ініціали)

Керівник БКР_____ Богач І.В.

(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Бакалаврська кваліфікаційна робота складається з 93 сторінок формату А4, на яких є 20 рисунків, 1 таблиця, список використаних джерел містить 28 найменувань.

Робота присвячена розширенню фунціональних можливостей WEB-додатку для керування серверним середовищем корпоративної інформаційної системи.

У загальній частині роботи на основі аналізу існуючих рішень, була доведена доцільність розробки такого WEB-додатоку, розроблено алгоритм роботи та UML-діаграму компонентів WEB-додатку, які спрощують розробку та розуміння структури програмного рішення.

В результаті роботі розроблено WEB-додаток для керування серверним середовищем корпоративної інформаційної системи, який забезпечує автоматизоване розгортання, налаштування та моніторинг серверної інфраструктури з використанням сучасних технологій контейнеризації та автоматизації, таких як Docker, Docker Compose, Python, Django.

Ключові слова: WEB-додаток, автоматизація розгортання, серверна інфраструктура, моніторинг, Docker, Python.

ABSTRACT

Bachelor’s Qualification Thesis consists of 93 A4 pages and includes 20 figures, 1 table, and a list of 28 references.

The thesis is dedicated to expanding the functionality of a web application designed for managing the s erver environment of a corporate information system.

In the general part of the thesis, based on the analysis of existing solutions, the feasibility of developing such a web application was substantiated. An operational algorithm and a UML component diagram of the web application were created to facilitate the development process and improve understanding of the system’s architecture.

As a result, a web application was developed for managing the server environment of a corporate information system. It provides automated deployment, configuration, and monitoring of the server infrastructure using modern containerization and automation technologies such as Docker, Docker Compose, Python, and Django.

Keywords: web application, deployment automation, server infrastructure, monitoring, Docker, Python.

ЗМІСТ

ВСТУП

Актуальність роботи. Загальновідомо, що стрімкий розвиток інформаційних технологій та залежність бізнесу від IT–рішень призводить до ускладнення структур корпоративних інформаційних систем. Такі системи дозволяють автоматизовувати управління внутрішніми процесами, забезпечують обробку великих обсягів даних та багато інших аспектів діяльності підприємств. Для забезпечення виконання цих завдань стабільність, керованість та масштабованість серверних інфраструктур стають критично важливими.

У відповідь на потребу в ефективному вирішенні завдань керування інфраструктурою, на перший план виходять сучасні технології, які забезпечують автоматизацію рутинних процесів. Завдяки таким підходам суттєво зменшуються трудові витрати, знижується ймовірність похибок, які пов’язані з людським фактором. Одним із найрезультативнішим рішенням у цій сфері є використання технологій контейнеризації, насамперед Docker Compose та Docker. Завдяки цим інструментам можливе створення ізольованих й керованих середовища, які легко масштабуються і забезпечують гнучкість у налаштуванні. Проте, автоматизація - це лише одна частина ефективного керування інфраструктурою. Такою ж важливою є можливість постійного моніторингу та аналізу системних метрик. Відслідковування системних параметрів дозволяє своєчасно виявляти аномалії, прогнозувати ризики та забезпечувати стабільну й безперебійну роботу всієї інформаційної системи.

Актуальність теми бакалаврської роботи зумовлена необхідністю підвищення ефективності адміністративних процесів, забезпечення стабільності та зниження технічних збоїв. Тому розробка додатку сприятиме підвищенню ефективності керування ресурсами підприємств.

Метою дослідження є розширення фунціональних можливостей додатку для керування серверним середовищем корпоративної інформаційної системи.

Об’єкт дослідження – процес розгортання, моніторингу та управління серверним середовищем корпоративної інформаційної системи.

Предмет дослідження – програмні засоби для моніторингу та керування стану серверної інфраструктури.

Задачі дослідження:

1. Провести аналіз предметної області моніторингу та керування серверної інфраструктури.

2. Спроєктувати структуру та компоненти WEB-додатку для керування серверним середовищем корпоративної інформаційної системи.

3. Програмно реалізувати WEB-додаток для керування серверним середовищем корпоративної інформаційної системи.

4. Виконати тестування програми та провести аналіз отриманих результатів;

5. Розробити інструкцію користувача.

Апробація роботи. Результати роботи було представлено на LIV Всеукраїнській науково-технічній конференції підрозділів Вінницького національного технічного університету (2025) [1] та на Міжнародній науково-практичній Інтернет-конференції студентів, аспірантів та молодих науковців «Молодь в науці: дослідження, проблеми, перспективи (МН-2025)» [2].

Публікації. електронні тези конференції LIV Всеукраїнській науково-технічній конференції підрозділів Вінницького національного технічного університету (2025) [1] та Міжнародній науково-практичній Інтернет-конференції студентів, аспірантів та молодих науковців «Молодь в науці: дослідження, проблеми, перспективи (МН-2025)» [2].

Впровадження. Результати, одержані в ході виконання бакалаврської кваліфікаційної роботи, а саме, розроблені алгоритми, інтерфейсні рішення та програмна реалізація, плануються до впровадження в розробку ТОВ «ЕСЛАЙФ», (Додаток Д) та подана заявка на отримання свідоцтва на авторське право.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ МОНІТОРИНГУ ТА КЕРУВАННЯ СЕРВЕРНОЇ ІНФРАСТРУКТУРИ

1.1 Обґрунтування доцільності розробки WEB-додатку

У контексті сучасного розвитку інформаційних технологій ефективне розгортання, подальше керування та моніторинг серверною інфраструктурою виступає однією з основних передумов стабільного функціонування корпоративних IT-систем. З огляду на зростаючу складність обчислювальних середовищ та необхідність забезпечення високого рівня доступності й масштабованості, все більшого поширення набувають засоби автоматизації процесів розгортання та оркестрації серверних компонентів. Серед найбільш затребуваних технологій у цій сфері варто виокремити такі інструменти, як Frappe Cloud, Railway та Render. Кожен із зазначених інструментів має характерний набір функцій, власну архітектурну специфіку, а також низку переваг і недоліків залежно від цілей і вимог до IT-інфраструктури. У межах цього розділу буде здійснено технічний аналіз зазначених систем, із подальшим порівнянням їх функціональних можливостей, сценаріїв використання та рівня придатності до інтеграції в системи корпоративного рівня.

Розвиток цифрових інструментів та зростаюча залежність підприємств від інформаційних систем зумовили появу нового класу викликів, пов’язаних з управлінням інфраструктурою. Більшість сучасних компаній прагнуть досягти високого рівня гнучкості, автоматизації та стійкості, що ставить під сумнів ефективність традиційних підходів до адміністрування серверів. В умовах, коли сервіси функціонують у режимі 24/7, а кожна секунда простою може обернутися фінансовими втратами чи втратою довіри клієнтів, виникає об’єктивна необхідність у системах, здатних забезпечити оперативний контроль над інфраструктурою, включно з моніторингом, автоматичним масштабуванням та засобами для резервного копіювання [1].

Сучасні обчислювальні середовища часто характеризуються багаторівневою структурою, яка охоплює не лише фізичні або віртуальні сервери, але й численні сервіси, контейнери, бази даних, проксі-рішення, механізми маршрутизації та сертифікації. У таких умовах ефективна оркестрація всіх компонентів є надзвичайно важливою, а засоби централізованого управління — критично необхідними. Зокрема, автоматизовані інструменти дають змогу істотно скоротити час на розгортання нових інстанцій, уніфікувати конфігураційні процеси та зменшити ризик людських помилок.

Одним із базових факторів надійності серверної інфраструктури виступає її здатність до самостійного відновлення або швидкого повернення до робочого стану у разі збоїв. Тут особливо важливу роль відіграє резервне копіювання, що має бути інтегрованим у загальну архітектуру інфраструктурного рішення та забезпечувати як планове збереження конфігурацій, так і можливість екстреного відновлення. У поєднанні з механізмами моніторингу це дозволяє своєчасно виявляти аномалії, перегріви, перевантаження або інші критичні події, що потенційно можуть призвести до збоїв.

Ще одним вагомим фактором є потреба у гнучкій системі маршрутизації запитів, яка дозволяє динамічно керувати підключеннями до окремих сервісів, а також забезпечує підтримку шифрування даних на транспортному рівні. У цьому контексті важливою є роль реверсивного проксі, такого як Traefik, що здатний не лише оптимізувати трафік, а й автоматично оновлювати сертифікати без втручання адміністратора, чим значно знижує навантаження на технічний персонал.

У контексті побудови сучасних IT-систем, що потребують гнучкого масштабування, високої доступності та швидкого розгортання, ключову роль відіграють технології контейнеризації [5]. Саме завдяки їм стало можливим значно оптимізувати процеси керування серверним середовищем, скоротити час налаштування інфраструктури та зменшити залежність від конкретного апаратного або програмного середовища. Контейнеризація відкрила нові підходи до розробки та експлуатації програмного забезпечення, дозволяючи ефективно

ізолювати сервіси, забезпечувати їхню взаємну незалежність та гарантовану стабільність.

Однією з найбільш вживаних технологій, що здійснила значний вплив на методи розгортання та ізоляції програмного забезпечення, є Docker — платформа для контейнеризації, яка дозволяє створювати легковагові, переносні середовища виконання [3]. В основі її функціонування лежить концепція контейнерів — ізольованих одиниць виконання, які забезпечують відокремлення середовища для кожного додатку.

Доволі важливою перевагою контейнеризації є швидкість запуску та можливість гнучкого масштабування. Контейнери запускаються доволі швидко, що критично для сучасних сценаріїв CI/CD, де тестування та розгортання нових версій додатків повинно відбуватися практично миттєво. Ізоляція середовищ дозволяє розгортати одночасно декілька версій одного й того самого додатку без ризику взаємного впливу, що є неможливим у традиційних монолітних архітектурах.

Docker складається з кількох ключових компонентів, які забезпечують його потужність і гнучкість. Першим з них є Docker Engine – основний компонент Docker, який дозволяє створювати та запускати контейнери [3]. Docker Engine складається з сервера (демона), API та клієнта. Сервер Docker відповідає за управління контейнерами, API забезпечує взаємодію між компонентами, а клієнт дозволяє користувачам взаємодіяти з Docker за допомогою командного рядка.

Іншим важливим компонентом є Docker Images – шаблони, з яких створюються контейнери. Образи Docker містять усі необхідні для роботи додатка залежності та налаштування. Використовуючи Docker Images, ви можете легко створювати та розгортати контейнери, забезпечуючи повторюваність і надійність розгортання додатків [3].

Контейнери, створені з образів Docker, є виконуваними інстанціями, що працюють в ізольованих середовищах. Docker Containers ізольовані один від одного та від операційної системи, що забезпечує безпеку та стабільність. Це

дозволяє уникнути конфліктів між додатками та забезпечити стабільну роботу навіть у випадку збоїв в інших контейнерах.

Одним з найважливіших аспектів Docker є Docker Hub – публічний репозиторій Docker Images, де користувачі можуть завантажувати та зберігати свої образи. Docker Hub значно спрощує обмін образами між розробниками та командами, забезпечуючи легкий доступ до попередньо налаштованих образів [4].

Docker має багато переваг, які роблять його одним з найпопулярніших інструментів для контейнеризації додатків. По-перше, Docker забезпечує легкість та портативність. Контейнери Docker легкі та швидкі, що дозволяє їх швидко запускати та переносити між різними середовищами. Це особливо важливо для розробників, які можуть легко перенести свої додатки з локального середовища на тестові або продуктивні сервери без необхідності повторного налаштування [5]. По-друге, Docker забезпечує ізоляцію додатків. Кожен контейнер працює в ізольованому середовищі, що підвищує безпеку та стабільність. Ізоляція дозволяє уникнути конфліктів між додатками та забезпечує стабільну роботу навіть у випадку збоїв в інших контейнерах. Це також означає, що ви можете запускати кілька версій одного й того ж додатка на одному сервері без ризику конфліктів. По-третє, Docker забезпечує швидкість розгортання. Docker дозволяє розгортати додатки за лічені секунди, що значно скорочує час введення в експлуатацію. Це особливо корисно для середовищ, де потрібне швидке масштабування або часті розгортання нових версій додатків [3].

Значний вплив на використання Docker в продакшн середовищах зіграв плагін Docker Compose, який є інструментом для визначення та запуску багатоконтейнерних Docker-додатків. Docker Compose є інструментом, з допомогою якого можна легко керувати конфігурацією кількох контейнерів, визначаючи їх у простому форматі YAML. Docker Compose дозволяє запускати всі необхідні контейнери одночасно, визначати їх залежності і конфігурацію мережі, що значно спрощує процеси налаштування та управління складними додатками [3].

Основним компонентом Docker Compose є файл docker-compose.yml, який містить конфігурацію всіх служб, мереж і томів, що використовуються в додатку [3]. Використовуючи цей файл, ви можете описати всі аспекти вашої інфраструктури в одному місці, що полегшує управління та забезпечує повторюваність процесу розгортання. Команди Docker Compose, такі як docker-compose up, docker-compose down, дозволяють легко запускати, зупиняти та керувати життєвим циклом ваших додатків [4].

Docker Compose має кілька ключових переваг, які роблять його незамінним інструментом для управління багатоконтейнерними додатками. По-перше, Docker Compose забезпечує простоту використання. Використовуючи файл docker-compose.yml, ви можете легко визначити конфігурацію всіх контейнерів, мереж та томів вашого додатка [4]. Це значно спрощує процес налаштування та управління складними додатками, дозволяючи зосередитися на розробці, а не на інфраструктурі. По-друге, Docker Compose забезпечує масштабованість. Використовуючи прості команди, ви можете легко масштабувати ваші служби, додаючи або видаляючи контейнери залежно від навантаження. Це дозволяє оперативно реагувати на зміну вимог і забезпечує високу доступність ваших додатків. По-третє, Docker Compose забезпечує інтеграцію з Docker. Docker Compose безшовно інтегрується з Docker, що дозволяє використовувати всі можливості Docker для управління контейнерами [4]. Це забезпечує додаткові можливості для керування вашими додатками, такі як моніторинг, логування та автоматичне відновлення у випадку збоїв.

Наприклад, для розгортання локального середовища ERPNext для розробки з використанням Docker Compose, можна створити файл docker-compose.yml, який містить конфігурацію для різних служб, таких як база даних (MariaDB), сервер ERPNext та інші необхідні компоненти [4]. Цей файл дозволяє швидко та легко розгортати всю інфраструктуру ERPNext однією командою, забезпечуючи зручність та повторюваність процесу.

Це демонструє, як Docker Compose спрощує налаштування та управління складними додатками, дозволяючи легко масштабувати та керувати

залежностями між службами. Kubernetes є однією з найпотужніших систем оркестрації контейнерів з відкритим вихідним кодом, яка була розроблена Google і зараз підтримується спільнотою Cloud Native Computing Foundation (CNCF) [2]. Kubernetes автоматизує розгортання, масштабування та управління контейнеризованими додатками, дозволяючи з легкістю керувати великими кластерами контейнерів.

Незважаючи на численні переваги, Docker та Docker Compose мають деякі недоліки. Один з них – це обмеження щодо управління складними інфраструктурами. Docker та Docker Compose добре підходять для невеликих та середніх проєктів, але для великих розподілених систем може знадобитися використання додаткових інструментів оркестрації, таких як Kubernetes [4]. Ще одним недоліком є складність у налаштуванні та підтримці складних мережевих конфігурацій. Docker забезпечує базові можливості для налаштування мереж, але для складних мережевих архітектур можуть знадобитися додаткові інструменти та конфігурації. Це може ускладнити процес розгортання та управління інфраструктурою. Також варто зазначити, що Docker потребує певних ресурсів на хост-машині. Хоча контейнери легші за віртуальні машини, вони все одно споживають ресурси, що може вплинути на продуктивність системи при великій кількості контейнерів.

Після налаштування контейнеризованого середовища за допомогою Docker та Docker Compose постає необхідність у створенні ефективної системи моніторингу, яка дозволяє в режимі реального часу відстежувати стан інфраструктури, аналізувати навантаження та оперативно реагувати на будь-які збої. Особливо це актуально в умовах динамічного масштабування та високих вимог до безперебійної роботи систем. У таких випадках доцільно впроваджувати спеціалізовані платформи спостереження, які забезпечують глибоку аналітику, візуалізацію метрик та централізовану обробку журналів. Одним із найефективніших інструментів для цього є Grafana Cloud.

Grafana Cloud — це повністю керована хмарна платформа для спостереження (observability), яка надає централізований доступ до метрик,

логів, трасувань і профілів додатків та інфраструктури. Вона ідеально підходить для хмарно-орієнтованих середовищ і дозволяє об'єднувати дані з різних джерел, забезпечуючи повний огляд системи та швидке виявлення причин збоїв.

Платформа Grafana Cloud побудована на основі відкритих технологій, таких як Prometheus для збору метрик, Loki для агрегації логів, Tempo для трасування та Mimir для масштабованого зберігання метрик. Це дозволяє користувачам легко інтегрувати існуючі інструменти та уникати прив'язки до конкретного постачальника. Завдяки гнучкій архітектурі, Grafana Cloud підтримує понад 100 зовнішніх джерел даних, включаючи Elasticsearch, Amazon CloudWatch та інші.

Однією з ключових переваг Grafana Cloud є можливість швидкого запуску та масштабування без необхідності самостійного обслуговування інфраструктури. Користувачі отримують доступ до попередньо налаштованих дашбордів, інтеграцій для Kubernetes, інструментів для тестування продуктивності (k6) та управління інцидентами (Grafana Incident, OnCall). Це дозволяє зменшити час на розгортання та забезпечити надійний моніторинг систем.

Grafana Cloud також пропонує розширені можливості безпеки та управління доступом, включаючи підтримку SSO/SAML/LDAP, рольову модель доступу (RBAC) та можливість безпечного запиту приватних даних через Private Data Source Connect. Це забезпечує контроль над тим, хто і до яких даних має доступ, що особливо важливо для корпоративних середовищ.

Завдяки автоматичним оновленням, високій доступності та підтримці 24/7, Grafana Cloud дозволяє командам зосередитися на аналізі та оптимізації своїх систем, не витрачаючи ресурси на обслуговування інфраструктури спостереження. Це робить її ефективним рішенням для забезпечення надійності та продуктивності сучасних додатків та інфраструктур.

Загалом, інтеграція усіх вищенаведених функціональних компонентів у єдине середовище керування створює додаткову цінність для підприємств, які прагнуть до спрощення технічного обслуговування, підвищення безпеки та

мінімізації часу реакції на інфраструктурні загрози. Такі інструменти стають не просто зручним доповненням до IT-ландшафту компанії, а стратегічно важливим ресурсом, що дозволяє гнучко адаптуватися до змін, швидко масштабувати сервіси відповідно до бізнес-потреб та ефективно управляти ризиками, пов’язаними з цифровою інфраструктурою.

1.2 Аналіз відомих програмних рішень

На сучасному етапі розвитку IT-інфраструктури існує чимало інструментів і платформ, які пропонують різноманітні підходи до автоматизації розгортання, моніторингу та керування серверними середовищами. Кожне з рішень має власні особливості реалізації, функціональні можливості та сфери застосування.

З метою обґрунтування архітектури власної системи доцільно розглянути вже існуючі технології, які зарекомендували себе на практиці. У цьому розділі буде здійснено порівняльний аналіз найпоширеніших інструментів, зокрема таких як Frappe Cloud, Railway, Render. Оцінка буде здійснюватися за низкою критичних параметрів, що мають безпосереднє значення для реалізації цілей цього дослідження: моніторинг, зручність використання, гнучкість налаштувань, підтримка масштабованості, наявність механізмів автоматизації та візуалізації.

Такий аналіз дозволить сформувати обґрунтовану позицію щодо вибору технологій, які найкраще відповідають потребам розробки системи управління серверним середовищем корпоративної інформаційної системи.

Одним із провідних хмарних рішень, що орієнтовані на швидке розгортання корпоративних інформаційних систем, є Frappe Cloud — платформа, створена компанією Frappe Technologies [7]. Цей сервіс надає можливість запуску та масштабування інстанцій без потреби глибокого технічного втручання з боку користувача. Завдяки попередньо налаштованому середовищу та вбудованим інструментам керування, Frappe Cloud дозволяє суттєво скоротити час на початкове налаштування системи. Широка підтримка інтеграцій, зручність у користуванні та автоматизоване адміністрування роблять

цю платформу особливо привабливою для організацій, які не мають власної серверної інфраструктури або бажають зменшити витрати на її обслуговування [12]. Загальний вигляд інтерфейсу Frappe Cloud на рисунку 1.1.

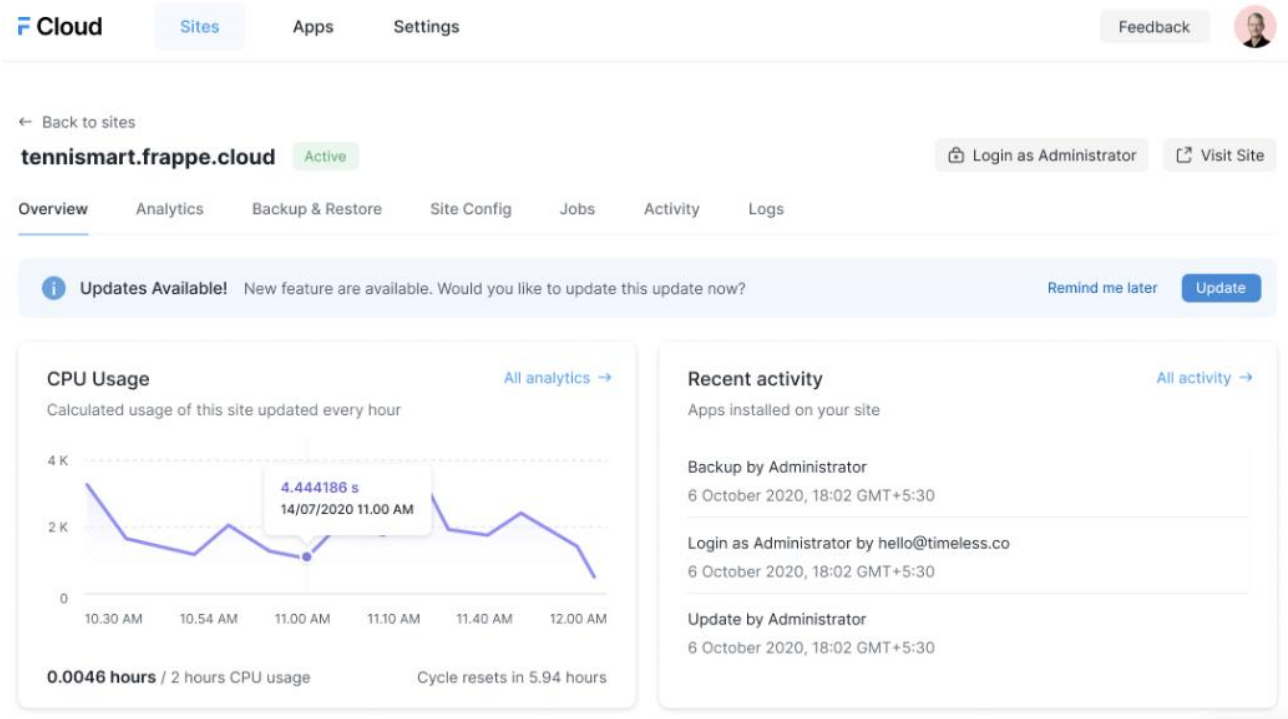

Рисунок 1.1 – Загальний вигляд інтерфейсу Frappe Cloud

Головною перевагою платформи Frappe Cloud є її орієнтація на зручність і доступність для кінцевого користувача. Завдяки простому та логічному інтерфейсу, розгортання екземпляру корпоративної інформаційної системи може бути виконане всього за кілька хвилин [8]. Такий підхід істотно полегшує процес налаштування для фахівців, які не мають глибоких знань у сфері адміністрування серверів або роботи з контейнерними технологіями. Окрім цього, система забезпечує автоматичне оновлення функціоналу та безпекових компонентів, що дозволяє завжди використовувати актуальні версії програмного забезпечення без ручного втручання чи додаткових дій з боку користувача.

Попри низку переваг, використання Frappe Cloud не позбавлене обмежень. Одним із ключових недоліків цієї хмарної платформи є порівняно висока вартість обслуговування [8]. Хоча початкові цінові плани виглядають

привабливо, зі зростанням обсягу ресурсів або появою потреби у створенні додаткових інстанцій загальна сума витрат може суттєво зрости. Для невеликих компаній чи стартапів це іноді стає критичним фактором, який стримує масштабування.

Також серед недоліків важливо зазначити про те, що користувачі не мають доступу до повноцінного моніторингу системних ресурсів. Це ускладнює своєчасне реагування на перевантаження або діагностику несправносте.

Крім того, певні технічні аспекти Frappe Cloud можуть створювати обмеження для користувачів. Наприклад, розгортання інстанцій іноді займає більше часу, ніж очікується, а вимоги до ресурсів сервера залишаються досить високими. Хоча система підтримує інтеграції та налаштування, користувачі все одно залишаються залежними від внутрішньої інфраструктури платформи та її політик. Це ускладнює реалізацію нестандартних конфігурацій або індивідуальних підходів, що можуть бути важливими для окремих проєктів або специфічних бізнес-процесів.

Render — це сучасна хмарна платформа, яка надає послуги типу Platform as a Service (PaaS) для розгортання, хостингу та масштабування вебзастосунків. Основна мета цієї платформи — максимально спростити процес запуску серверної інфраструктури без потреби в налаштуванні низькорівневих параметрів [10]. Render орієнтована як на індивідуальних розробників, так і на малі та середні команди, пропонуючи інтерфейс, що дозволяє швидко та інтуїтивно розгортати проєкти з підтримкою широкого спектру технологій, включаючи Docker-контейнери, статичні сайти, фонові задачі та бази даних.

Платформа забезпечує автоматичне безперервне розгортання (CI/CD) з прив’язкою до Git-репозиторіїв (GitHub або GitLab), дозволяючи запускати оновлення одразу після кожного коміту. Вона підтримує розгортання без простоїв, приватні внутрішні мережі, автоматичне масштабування залежно від поточного навантаження, а також надає базовий захист [10]. Загальний вигляд інтерфейсу Render наведено на рисунку 1.2.

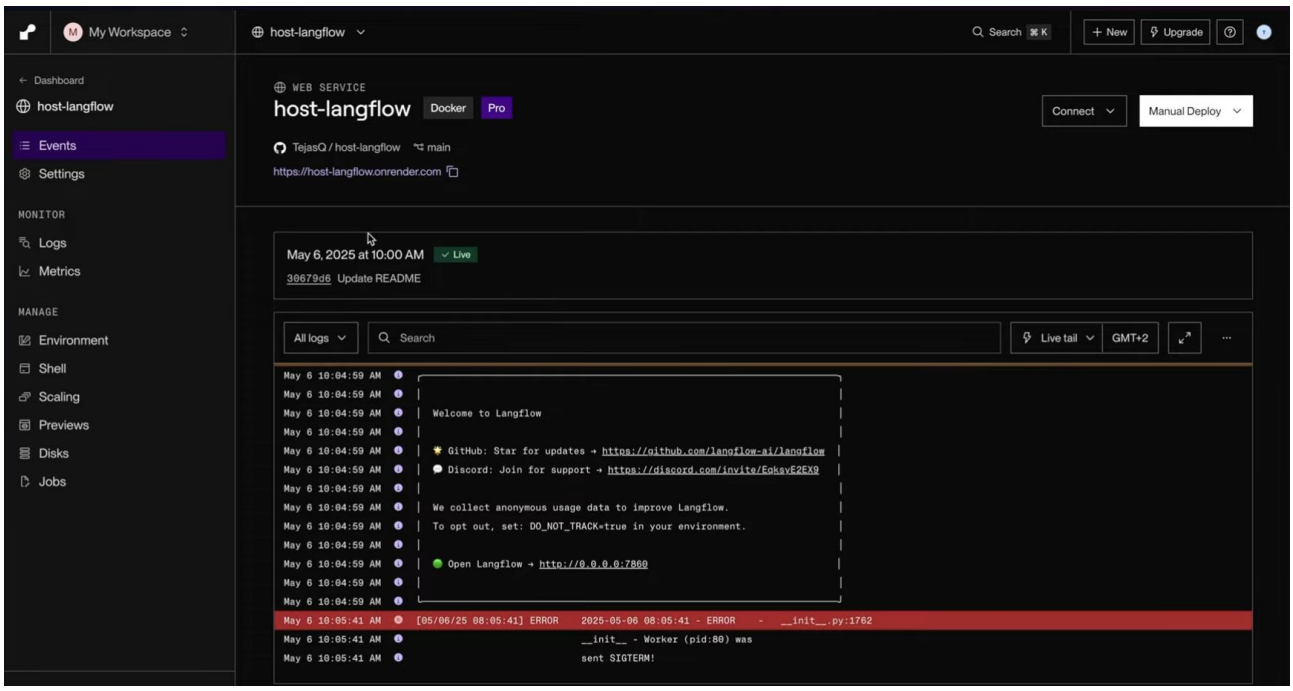

Рисунок 1.2 – Загальний вигляд інтерфейсу Render

Попри свої численні переваги, Render має низку обмежень, які необхідно враховувати, особливо в контексті побудови систем із підвищеними вимогами до моніторингу та контролю за ресурсами. Одним із ключових недоліків є обмежені можливості моніторингу серверного середовища. Платформа надає базову інформацію про роботу сервісів — журнал подій (логи), статуси розгортань, а також загальні метрики споживання ресурсів (CPU, пам’ять)[]. Однак цього функціоналу недостатньо для повноцінного контролю продуктивності у великих або критичних системах, де потрібні розширені механізми збору телеметрії, кастомні метрики, створення власних дашбордів та система сповіщень.

Ще одним аспектом, який варто враховувати, є обмежена підтримка кастомізації процесів розгортання та налаштування середовища виконання. Хоча Render надає базову гнучкість через використання Docker-контейнерів та конфігураційного файлу render.yaml, можливості тонкого налаштування залишаються обмеженими. Наприклад, не передбачено прямого доступу до інфраструктурного рівня, що унеможливлює налаштування нестандартних параметрів мережі, балансування навантаження або встановлення специфічних

служб моніторингу. У випадках, коли проєкт вимагає нестандартних рішень або інтеграції з власними DevOps-інструментами, Render може виявитися недостатньо гнучким і не відповідати вимогам щодо масштабованості та керованості на рівні інфраструктури. Також ваото додати те що тут відсутня можливість бекапу та відновлення даних в будь – який момент, особливо в дешевших підписках.

Railway — це сучасна хмарна платформа, орієнтована на спрощення процесів створення, розгортання та підтримки вебзастосунків. Вона позиціонує себе як інструмент для «розробників, які не хочуть бути DevOps-фахівцями», і надає інтуїтивно зрозумілий інтерфейс для керування проєктами, сервісами, середовищами виконання та конфігураціями без необхідності глибокого занурення в інфраструктурні деталі[10].

Railway підтримує розгортання як із репозиторіїв Git (GitHub, GitLab), так і через локальні CLI-інструменти. Це дозволяє швидко запускати серверні застосунки, бази даних або фронтенд-проєкти. Користувач може обрати із попередньо налаштованих шаблонів (наприклад, PostgreSQL, Redis, Express.js, Next.js тощо), або створити власний стек за допомогою Docker або Buildpacks.

Вигляд інтерфейсу Railway наведено на рисунку 1.3.

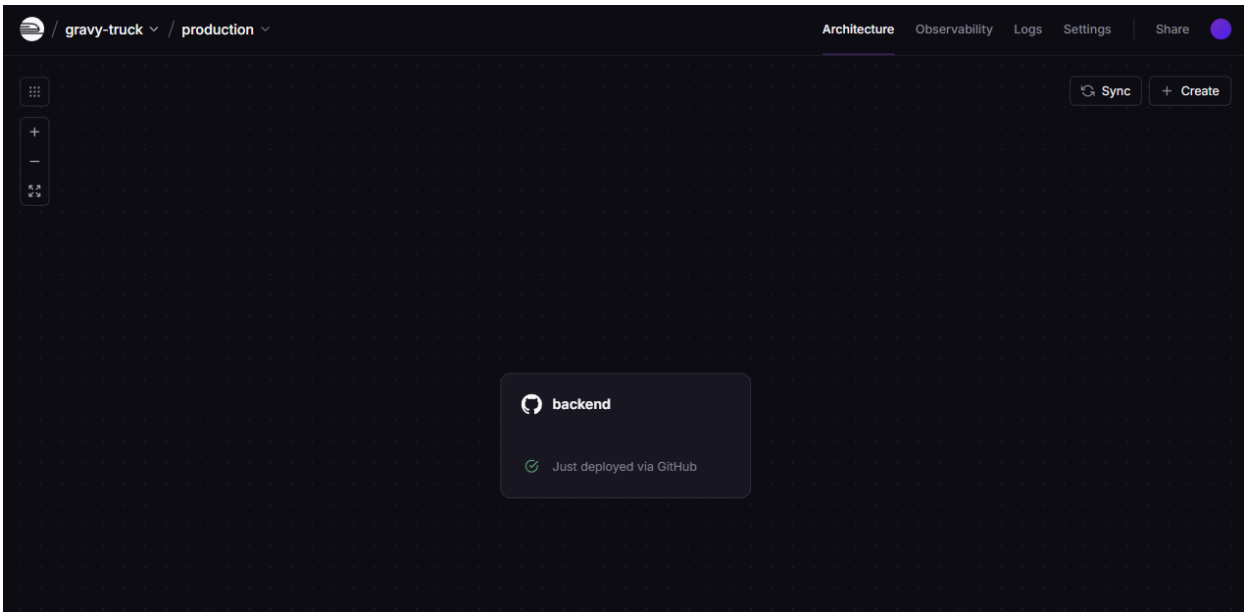

Рисунок 1.3 – Вигляд інтерфейсу Railway

Попри велику кількість позитивних аспектів, Railway має й обмеження. Одним із ключових недоліків платформи є відносно обмежені функції моніторингу та відсутність розширеної аналітики продуктивності. Хоча платформа надає базові логи, інформацію про статуси деплою та споживання ресурсів, цього часто недостатньо для повноцінного аналізу продуктивності або виявлення причин збоїв. Railway не підтримує нативну інтеграцію з професійними системами моніторингу на кшталт Prometheus або Grafana, а також не дозволяє створювати кастомні метрики чи систему алертів.

Провівши порівняльний аналіз сучасних платформ для розгортання та керування серверною інфраструктурою — зокрема Frappe Cloud, Railway та Render — можна дійти висновку, що кожна з них має ряд переваг, серед яких автоматизоване розгортання, зручна інтеграція з репозиторіями, базова масштабованість та простота використання. Однак, попри загальну зручність та автоматизацію, всі розглянуті платформи мають суттєві обмеження, які унеможливлюють їх повноцінне застосування. Найбільш критичним недоліком є недостатньо розвинений функціонал моніторингу: користувачі мають доступ лише до базових логів, статусів сервісів і поверхневих метрик. Крім того, функціональність резервного копіювання в аналізованих платформах або відсутня, або реалізована у вигляді фіксованих політик без можливості гнучкого налаштування.

1.3 Постановка задач та формулювання вимог

Для реалізації веб-додатку для керування серверним середовищем корпоративної інформаційної системи, необхідно дотримуватись чітко визначеної послідовності дій. Чітке формулювання завдань є ключовим етапом у процесі розробки програмного забезпечення, оскільки саме воно визначає загальний напрям проєкту, дозволяє уникнути неузгодженостей і забезпечує послідовність дій.

Формування вимог до майбутньої системи повинно базуватися на глибокому розумінні особливостей її використання у реальному середовищі. Важливо враховувати не лише технічні характеристики, а й зручність для кінцевого користувача й швидкість виконання операцій. Такий підхід дозволяє забезпечити узгодженість між очікуваним функціоналом і практичними потребами, що, у свою чергу, впливає на успішність впровадження розробленого рішення.

Основною метою є розширення функціональних можливостей додатку для керування серверним середовищем корпоративної інформаційної системи, а тобто створення інструменту, що забезпечує повноцінну підтримку основних функцій: розгортання середовищ, спостереження за станом інфраструктури, а також організацію процесів збереження й відновлення даних. Для досягнення цієї мети завдання доцільно структурувати за окремими етапами, кожен з яких охоплює конкретну сферу розробки та впровадження функціоналу:

- Проєктування та розробка структури компонентів — визначення загальної логіки побудови системи, виділення основних функціональних модулів і встановлення зв’язків між ними.

- Розробка алгоритму функціонування — створення логічної моделі роботи програмного модуля, що описує ініціалізацію додатку та взаємодію його частин.

- Обґрунтування технологічного стеку — аргументація вибору мови програмування, середовища розробки та допоміжних інструментів, необхідних для реалізації системи.

- Програмна реалізація веб-додатку — безпосереднє створення програмного забезпечення, яке забезпечує автоматизоване розгортання інфраструктури та її подальше обслуговування.

- Тестування функціоналу — перевірка працездатності системи, виявлення помилок, оцінка стабільності роботи та відповідності розробленого додатку вимогам.

Важливим є визначення функціональних вимог веб-додатку для керування серверним середовищем корпоративної інформаційної системи. Цей додаток повинен забезпечувати створення, запуск і зупинку інстанцій, а також резервне копіювання й відновлення даних. Обов’язковою є підтримка моніторингу ключових параметрів системи в реальному часі, зокрема навантаження на CPU, використання пам’яті, дискового простору та мережевого трафіку. Додаток також має взаємодіяти з проксі-сервером Traefik, забезпечуючи при цьому зручний та зрозумілий інтерфейс.

Отже, процес формування технічного завдання на розробку програмного модуля охоплює побудову архітектурної структури системи, проєктування логіки її функціонування, вибір відповідного інструментарію — мови програмування, середовища та допоміжних технологій, а також реалізацію ключових функцій, що відповідають за створення, запуск і обслуговування серверних інстанцій. У результаті має бути створене цілісне та стабільне програмне рішення, здатне автоматизувати керування інфраструктурою інформаційної системи.

1.4 Висновки до розділу 1

В цьому розділі в результаті проведеного аналізу було обґрунтовано актуальність створення веб-додатку для керування серверним середовищем корпоративної інформаційної системи в умовах зростаючих вимог до автоматизації, масштабованості та безперервної доступності IT-інфраструктури. Було досліджено специфіку сучасних підходів до розгортання, моніторингу та адміністрування серверів, розглянуто можливості контейнеризації як ключової технології для створення ізольованих середовищ, а також охарактеризовано інструменти, які забезпечують централізоване управління інстанціями та сервісами.

Здійснено порівняльний аналіз популярних хмарних платформ, таких як Frappe Cloud, Render і Railway, що дозволило виявити їх переваги та обмеження,

зокрема недостатню гнучкість моніторингу, обмежений контроль над інфраструктурою та відсутність гнучких механізмів резервного копіювання.

Визначено основні технічні та функціональні вимоги до майбутнього програмного модуля, сформовано послідовність етапів його реалізації, а також акцентовано на важливості чіткого формування задач проєкту. Усе це створює підґрунтя для проєктування власного рішення, що враховуватиме виявлені недоліки існуючих систем і відповідатиме актуальним потребам адміністрування серверної інфраструктури.

2 ПРОЄКТУВАННЯ СТРУКТУРИ ТА КОМПОНЕНТІВ WEB-ДОДАТКУ ДЛЯ КЕРУВАННЯ СЕРВЕРНИМ СЕРЕДОВИЩЕМ КОРПОРАТИВНОЇ ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1 Проєктування структури компонентів WEB-додатку

Проєктування Web-додатку для керування серверним середовищем корпоративної інформаційної системи передбачає реалізацію інтерфейсу, який дозволить адміністраторам ефективно здійснювати керування та моніторинг над різними аспектами серверної інфраструктури. Як базовий фреймворк для створення такого додатку було обрано Django. [13]. Додаток буде забезпечувати повноцінне управління інстанціями корпоративного програмного забезпечення: запуск нових серверних середовищ, оновлення та контроль за вже активними інстанціями, а також здійснення бекапу і відновлення резервої копії у разі збою чи втрати інформації. Також важливу роль будуть відігравати засоби для моніторингу актуального стану серверів. Всі ці функції будуть об’єднані в єдину систему керування зі зручним інтерфейсом користувача. Надалі буде деталізовано розглянуто кожен компонент даного веб - додатку. UML – діаграму програмних компонентів web- додатку зображено на рисунку 2.1.

Головна сторінка веб-додатку для керування серверним середовищем корпоративної інформаційної системи виконуватиме функцію інформаційного дашборду, що забезпечуватиме візуальний контроль за ключовими параметрами продуктивності серверів у режимі реального часу. Даний дашборд стане центральним елементом інтерфейсу, дозволяючи системним адміністраторам оперативно оцінювати стан інфраструктури та приймати обґрунтовані рішення щодо її подальшої експлуатації або масштабування. Він буде містити такі метрики: поточне навантаження на центральний процесор(CPU), обсяг використаної та доступної оперативної пам’яті(RAM), кількість запущених процесів, швидкість вхідного та вихідного мережевого трафіку, тривалість роботи серверу та технічні характеристики серверного середовища.

Інтерфейс керування екземплярами, реалізований у складі веб-застосунку, забезпечує повний доступ до адміністрування кожного розгорнутого середовища корпоративної інформаційної системи. У межах цієї сторінки користувачу демонструється впорядкований перелік інстанцій у вигляді таблиці або адаптивних карткових блоків, де для кожного екземпляра виводяться основні параметри: унікальна назва, актуальний статус (запущено або призупинено) та інші супутні характеристики. Крім перегляду, користувач має можливість перейти до детальної інформації про вибраний екземпляр для виконання подальших дій.

В додатку передбачена можливість перегляду про кожну інстанцію корпоративної інформаційної системи. Коли здійснюється вибір однієї з наявних інстанцій, користувач потраплятиме на сторінку, яка надасть розширену відомість про її поточний стан та супутні процеси. Ця сторінка відіграє ключову роль в аналізі та оперативному керуванні інстанцією. На сторінці відображатиметься перелік контейнерів разом із такими параметрами: поточний статус кожного контейнера, час створення, ім’я та унікальний ідентифікатор. Окрім інформаційного блоку, на сторінці також містяться елементи управління, які дозволяють виконувати такі основні операції:

Запуск: активує контейнер, що перебуває у стані зупинки, реалізується шляхом виклику відповідної Docker-команди.

Зупинка: зупиняє виконання активоного контейнера, при натисканні викликатиметься відповідна команда для зупинки.

Створення резервної копії: кнопка за допомогою якої виконується резервне копіювання, при натисканні ініціюється скрипт для зберіганняння актуального стану у захищену директорію.

Відновлення з резервнох копії: після натичкання виконується відкат бази даних до попереднього стану на основі наявного бекапу, шляхом виклику відповідного сценарію.

Головна сторінка веб-додатку виступає основним елементом взаємодії користувача з системою та є основною точкою входу до інтерфейсу керування

серверною інфраструктурою. Її функціональне призначення полягає у наданні узагальненого огляду технічного стану серверів, на яких розгорнуті компоненти корпоративної інформаційної системи. Завдяки зручному й наочному поданню основних метрик продуктивності, сторінка забезпечує можливість оперативного прийняття обґрунтованих адміністративних рішень. З технічної точки зору, реалізація цієї сторінки в межах фреймворку Django передбачає створення відповідного представлення (view), яке обробляє запити та передає дані до клієнтської частини. Збір і первинна обробка метрик здійснюється за допомогою JavaScript-компонентів, які надсилають асинхронні запити до API сервера або безпосередньо взаємодіють із системними службами для отримання актуальних значень [16]. Отримані дані передаються у шаблон (template), де відбувається їх подальша візуалізація. Для цього застосовуються спеціалізовані бібліотеки для будування графіків таких як Chart.js або D3.js, які дозволяють створювати інтерактивні й адаптивні графічні компоненти, що полегшують аналіз інформації.

Такий підхід дає змогу реалізувати інтерактивні графіки, які оновлюються у режимі реального часу та відображають актуальні показники. Завдяки цьому користувач має можливість спостерігати за змінами навантаження на сервери в поточний момент і своєчасно реагувати у разі виявлення перевантажень або потенційних збоїв у роботі системи.

Сторінка для безпосереднього управління інстанціями у веб-додатку надає користувачу зручний інтерфейс для перегляду та адміністрування всіх розгорнутих екземплярів. Представлення інстанцій реалізоване у форматі таблиці або карткового відображення, де кожен елемент містить основну інформацію — назву інстанції, її актуальний стан (активна чи призупинена), а також базові метрики навантаження.

Кожна інстанція має інтерактивні елементи управління, що дозволяють запускати та зупиняти сервери, створювати резервні копії даних і відновлювати їх із бекапів. Усі ці дії реалізуються шляхом взаємодії з Docker API, що дозволяє здійснювати надійне та ефективне виконання операцій [17]. Користувач не

потрібно вводити команди в терміналі — весь процес управління відбувається через інтуїтивно зрозумілий інтерфейс.

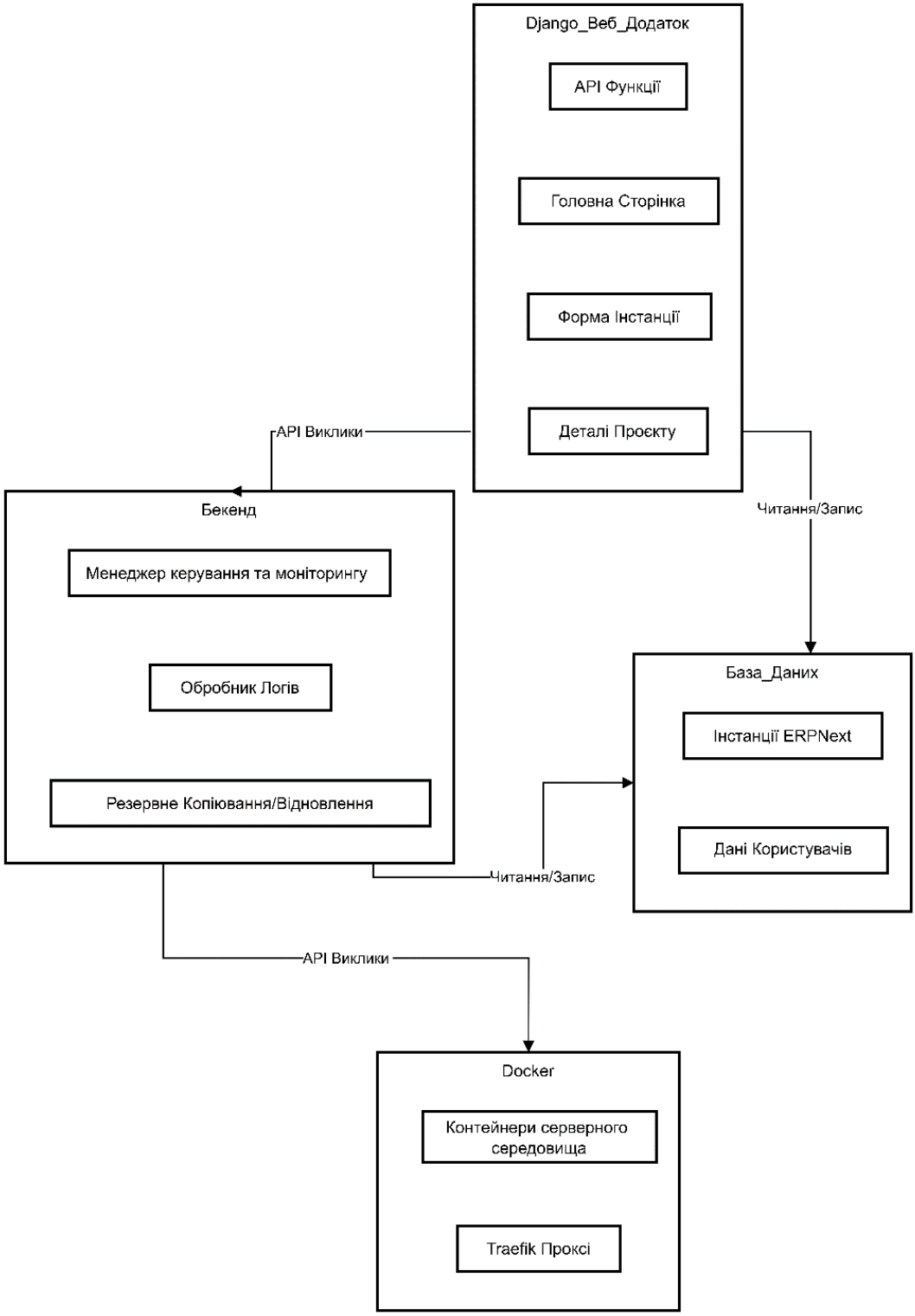

Рисунок 2.1 – UML-діаграма компонентів WEB-додатку для керування серверним середовищем корпоративної інформаційної системи

Після вибору певної інстанції користувач переходить на сторінку з розширеною інформацією, де детально описано конфігурацію інстанції та перелік її контейнерів. На заданій сторінці також доступні всі основні інструменти керування: запуск, зупинка, створення резервної копії та відновлення з наявного бекапу. Всі операції, як і раніше, виконуються за допомогою взаємодії з Docker API, що дозволяє підтримувати високу швидкість і зручність адміністрування серверного середовища.

2.2 Проєктування клієнтської частини WEB-додатку

Клієнтська частина web-додатку є важливим компонентом системи, яка забезпечує взаємодію користувача з серверною логікою керування інстанціями ERPNext. Основним завданням клієнтського інтерфейсу є надання інтуїтивно зрозумілого, зручного та функціонального доступу до всіх функцій системи: створення, запуску, зупинки інстанцій, резервного копіювання, моніторингу стану.

Клієнтська частина реалізована з використанням стандартного стеку: HTML, CSS та JavaScript. Для побудови динамічних візуалізацій метрик (CPU, пам’ять, диск) використано бібліотеки Chart.js або D3.js, з допомогою яких можна створювати інтерактивні графіки та діаграми. JavaScript використовується для ініціалізації запитів до бекенду, обробки відповідей, оновлення DOM та інтерактивної взаємодії з елементами UI у реальному часі. Для стилізації застосовано CSS, що забезпечує адаптивність інтерфейсу та зручну верстку на всіх типах пристроїв.

Процес проєктування інтерфейсу користувача для web-додатку здійснюється поетапно з урахуванням принципів зручності, інтуїтивності та функціональності. Першим важливим етапом є прототипування системи у вигляді схем. Такий етап допомагає визначити майбутню структуру інтерфейсу, логіку взаємодії користувач-система, а також запезпечують узгодженість під час реалізації функціональних можливостей.

Принципи які були використані під час проєктування інтерфейсу:

1. Зрозумілість та інтуїтивність - нтерфейс повинен бути простим для розуміння, навіть для користувача без технічної підготовки. Кнопки, іконки та навігація мають бути логічними та самодостатніми — користувач має одразу розуміти, що і як працює.

2. Послідовність - всі елементи інтерфейсу мають виглядати та поводитися однаково на всіх сторінках застосунку. Це стосується кольорової гами, шрифтів, іконок, стилю кнопок і взаємодій. Завдяки цьому користувач швидко адаптується до системи.

3. Зворотний зв'язок - кожна дія користувача має супроводжуватись візуальним підтвердженням. Це допомагає уникати плутанини та підвищує довіру до системи.

4. Контекстна взаємодія - кнопки дій для кожної інстанції (запуск, зупинка, резервне копіювання) з’являються відповідно до її поточного стану. Це знижує ризик помилок та покращує навігацію.

5. Візуалізація метрик у реальному часі - за допомогою інтерактивних графіків користувач може швидко оцінити поточну завантаженість серверів (CPU, RAM, диск), що підвищує оперативність прийняття рішень. Усі дії з інстанціями реалізовано за допомогою стандартизованих сценаріїв, що мають однакову логіку виконання — це підвищує зручність у використанні системи.

У процесі розробки клієнтської частини web-додатку було забезпечено дотримання сучасних веб-стандартів, таких як HTML5, CSS3 та JavaScript, що гарантує коректну роботу інтерфейсу. Це дозволяє досягти високого рівня сумісності та стабільності при використанні системи. Для стилізації інтерфейсу використано фреймворк Bootstrap, який завдяки вбудованій системі сіток та готовим компонентам значно пришвидшує розробку адаптивних веб-сторінок.

Клієнтська частина активно взаємодіє з бекендом за допомогою HTTP-запитів до REST API, реалізованого на основі Django. Усі ключові дії користувача — запуск, зупинка, створення резервної копії або відновлення інстанції — здійснюються через виклики відповідних ендпоінтів. Отримані

відповіді від серверної частини обробляються JavaScript-кодом, який динамічно оновлює інтерфейс без необхідності перезавантаження сторінки. Така інтеграція клієнта з сервером забезпечує інтерактивність, швидкодію та гнучкість системи. На рисунку 2.2 зображено загальну структурну схема компонентів WEB-додатку для керування серверним середовищем корпоративної інформаційної системи.

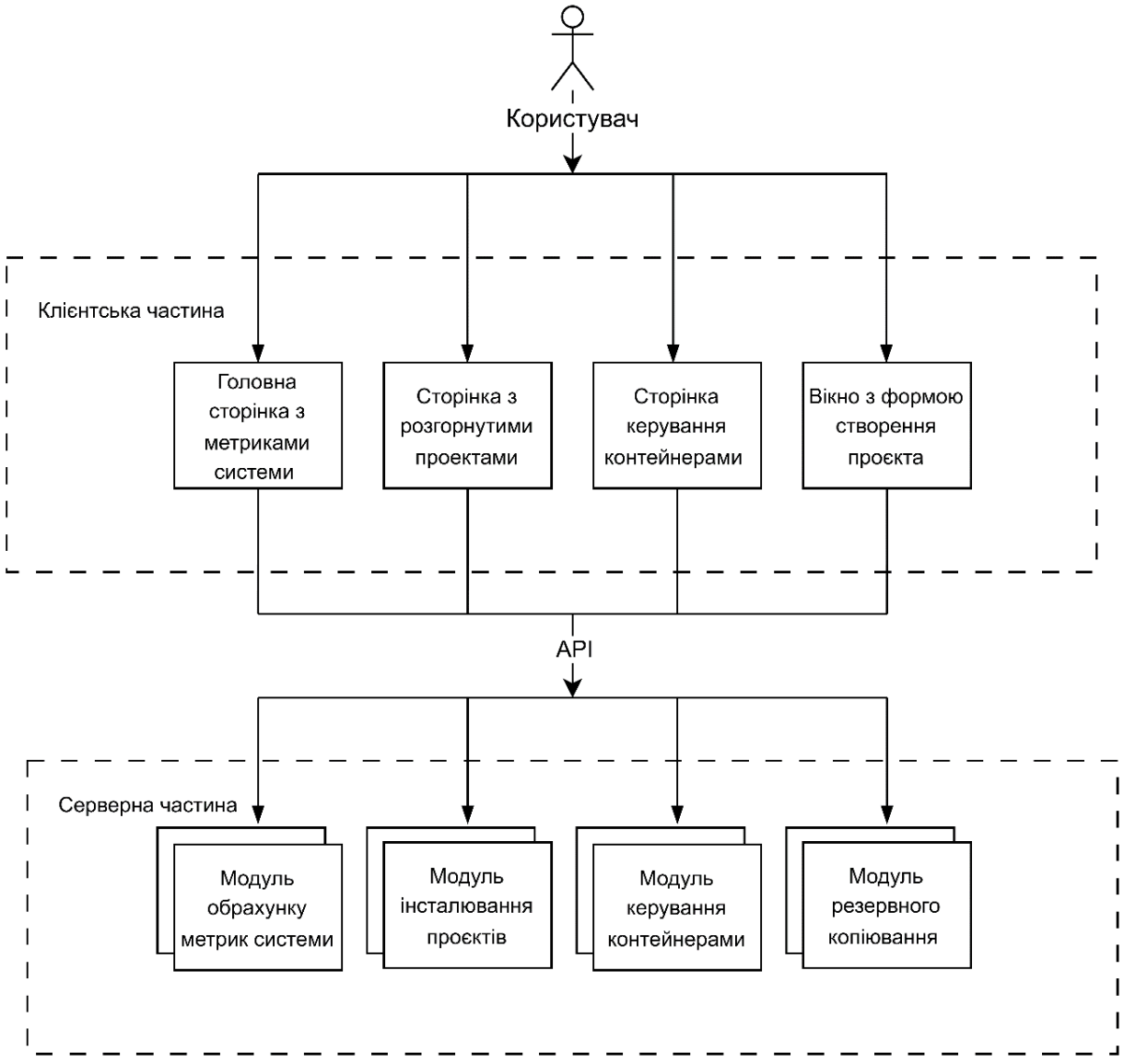

Рисунок 2.2 - Загальна структурна схема компонентів WEB-додатку для керування серверним середовищем корпоративної інформаційної системи

Описаний на рисунку структурний макет ілюструє загальну взаємодію користувача з веб-додатком, а також взаємозв’язок між клієнтською та

серверною частинами через API. Він дає змогу наочно уявити, яким чином організовано логіку інтерфейсів та функціональних блоків, зокрема які саме компоненти відповідають за відображення інформації на стороні користувача, та які серверні модулі опрацьовують запити.

У верхній частині діаграми зображено клієнтську частину додатку, що складається з кількох основних інтерфейсних екранів. До них належить головна сторінка з метриками системи, сторінка зі списком розгорнутих проєктів, інтерфейс для керування контейнерами, а також форма створення нового проєкту. Усі ці елементи — це візуальна складова, з якою безпосередньо взаємодіє користувач через браузер. На кожному з етапів — від моніторингу до створення інстанції — відправляються запити до API, які обробляються на серверній частині системи.

API виступає проміжною ланкою між клієнтським інтерфейсом і функціональними модулями, що реалізовані на сервері. Залежно від запиту, активується відповідний серверний компонент: обчислення метрик, інсталяція проєктів, керування контейнерами або резервне копіювання. Кожен з модулів ізольовано відповідає за конкретну функцію, забезпечуючи розділення відповідальностей і зручність у розширенні або обслуговуванні системи.

Такий підхід до проєктування клієнтської частини дозволяє чітко структурувати інтерфейс користувача, зробити його логічним і передбачуваним. Кожен екран пов’язаний з певною функціональністю і реагує на взаємодію шляхом виклику API-запитів, що у свою чергу гарантує динамічне оновлення даних без повного перезавантаження сторінки. Уся система функціонує як єдиний узгоджений механізм, де кожен елемент — від кнопки на інтерфейсі до серверної обробки — відіграє визначену роль у загальному процесі керування серверним середовищем.

2.3 Розробка схеми алгоритму роботи WEB-додатку

Розробка додатку для керування серверним середовищем корпоративної інформаційної системи потребує чіткого структурування всіх дій, які може здійснювати користувач у межах веб-додатку. Веб - додаток має надавати простий та зрозумілий інтерфейс користувача щоб створювати, запускати та зупиняти середовища, переглядати метрики системи, а також можливість бекапу та відновлення даних.

На рівні програмної реалізації процес побудований за покроковим алгоритмом, розглянемо його. Уразі якщо користувачем буде надіслано запит для створеня нового середовища, система обробляє цей запит і розпочинається процедуру ініціалізації. На початковому етапі створення інстанції виконується генерація унікального ідентифікатора, що слугує основою для подальшого розпізнавання цього середовища серед інших активних екземплярів. Після цього система автоматично формує файл з конфігураціями Docker Compose, у якому визначено всі ключові параметри — від назв сервісів до налаштувань мережі й змінних середовища — необхідні для коректного запуску контейнерів, що забезпечують функціонування новоствореної інстанції.

Після формування файлу конфігурацій, він буде зберігатися безпосередньо на сервері. Далі ініціюється виконання відповідної команди Docker Compose, яка відповідає за створення та запуск новостворених контейнерів. У разі вдалого розгортання, інформація про нове серверне середовище вноситься до бази даних — це дає змогу здійснювати подальше управління цим екземпляром через інтерфейс веб-додатку.

При поданні запиту на запуск середовища, система спочатку перевіряє його поточний статус у базі даних. У випадку, коли серверне середовище перебуває у стані зупинки, викликається Docker-команда, яка активує відповідний контейнер. Після підтвердження успішного запуску, статус цього середовища оновлюється у базі даних і відображається як активний.

Аналогічно, при запиті на зупинення середовища, додаток спочатку зчитує його поточний стан. Якщо середовище працює, ініціюється команда зупинення конкретного контейнера, після чого база даних оновлюється відповідно до нового статусу — з активного на зупинений. Така логіка забезпечує цілісність даних та дозволяє точно відстежувати стан кожного середовища в системі.

У разі, коли користувачем ініціюється створення бекапу бази даних певного серверного середовища, веб-додаток отримує відповідний запит та ініціює виконання команди Docker, яка запускає процес резервного копіювання безпосередньо всередині контейнера. У результаті успішного створення бекапу файл резервної копії переміщується до захищеного сховища, призначеного для збереження важливих даних. У базі даних оновлюється інформація щодо часу останнього резервного копіювання, що дозволяє відстежувати актуальність збережених копій.

Аналогічно, при запиті на відновлювання даних із попередньо створеного бекапу, веб-додаток обробляє запит і запускає відповідну Docker-команду, яка ініціює процедуру відновлювання бази даних усередині контейнера. По завершенню цього процесу та підтвердження його успішності система оновлює статус відповідного середовища в базі даних, фіксуючи його як відновлене.

Моніторинг стану серверних середовищ реалізується за рахунок бекенд-частини веб-додатку, яка відповідає за збір і обробку внутрішньої технічної інформації. Для цього використовуються вбудовані функції, що дозволяють отримувати метрики щодо стану як самих контейнерів, так і серверів, де вони працюють. Отримані дані надсилаються до клієнтської частини, де за допомогою JavaScript відбувається їхня динамічна візуалізація на дашборді. Такий підхід дає змогу користувачеві у реальному часі отримувати відомості про навантаження на інфраструктуру, а також оперативно оцінювати статус кожного активного середовища.

На Рисунку 2.3 зображено алгоритм роботи WEB-додатку для керування серверним середовищем корпоративної інформаційної системи.

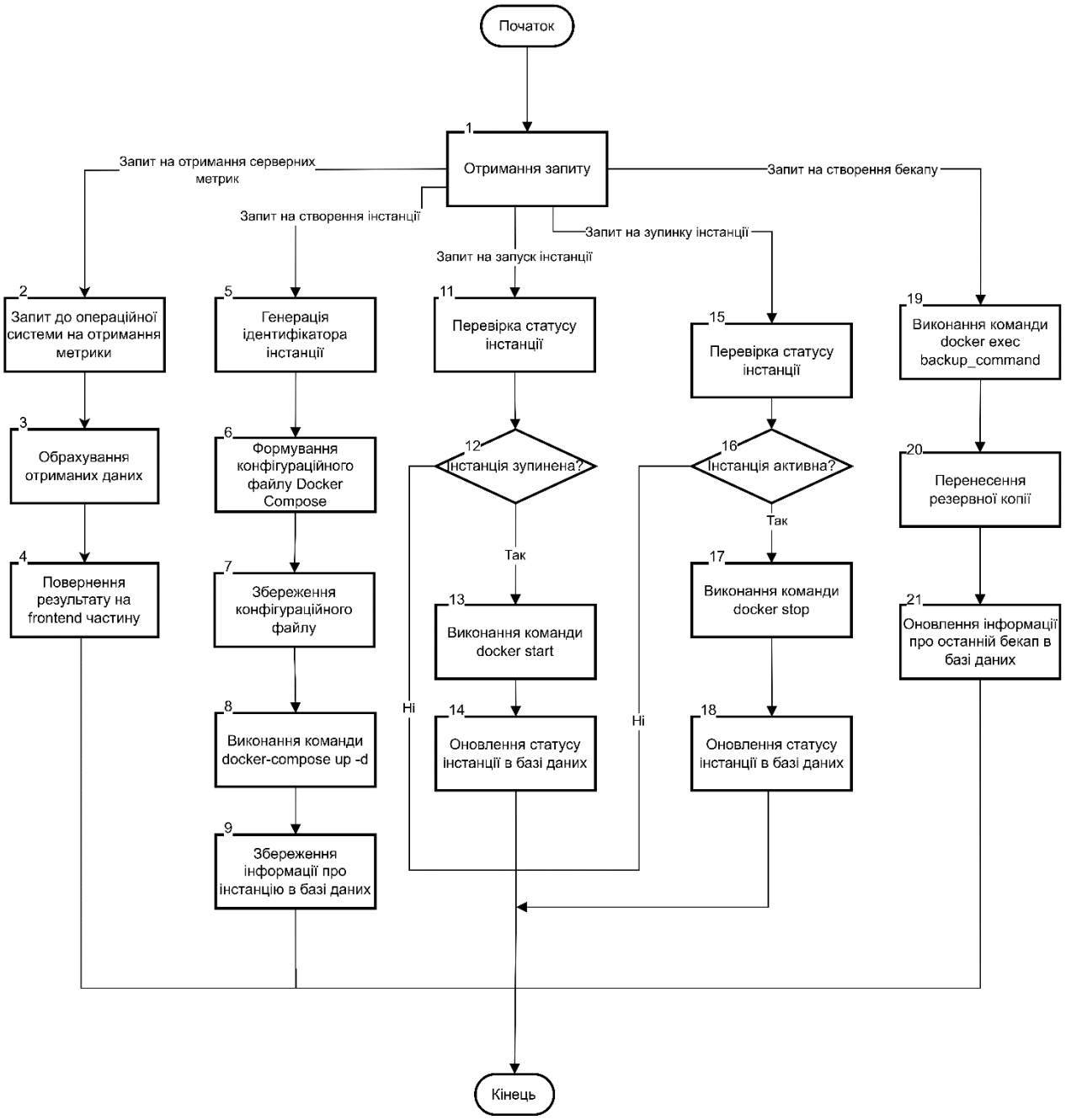

Рисунок 2.3 – Алгоритм роботи WEB-додатку для керування серверним середовищем корпоративної інформаційної системи

Алгоритм роботи веб-додатку розпочинається з обробки вхідного запиту, який може стосуватись створення нового серверного середовища, його запуску, зупинки або ініціації процесу резервного копіювання даних. У разі створення середовища система спочатку генерує унікальний ідентифікатор, після чого

формується файл із конфігураціями Docker Compose, який зберігається на сервері та використовується для запуску відповідних контейнерів.

При надходженні запиту на запуск середовища, веб-додаток перевіряє його поточний стан у базі даних. Якщо середовище перебуває у стані зупинки, ініціюється виконання команди docker start. Аналогічно, у випадку зупинення — якщо середовище є активним, ініціюється виконання команди docker stop.

Запит на створення бекапу містить виконання такої команди як docker exec, яка викликає необхідний скрипт для зберігання бази даних. Після цього файл бекапу переноситься до безпечного сховища, а в базі даних оновлюється інформація про час останнього резервного копіювання.

Крім того, веб-додаток може обробляти запити на отримання метрик. У цьому випадку запускається механізм збору технічних показників стану серверного середовища — зокрема, рівня навантаження на процесор, використання оперативної пам’яті, обсягу вільного місця на диску та кількості активних процесів. Зібрані метрики передаються до клієнтської частини для подальшої візуалізації на дашборді в режимі реального часу.

2.4 Висновки до розділу 2

У цьому розділі було детально проаналізовано ключові компоненти веб-додатку, призначеного для керування серверним середовищем корпоративної інформаційної системи. Основну увагу приділено модулю управління серверними інстанціями, принципам його побудови та взаємодії між складовими частинами системи. Наведено UML-діаграму програмних компонентів і схему алгоритму роботи модуля, що дозволяє чітко уявити логіку функціонування на концептуальному рівні. Спроєктована структура системи є простою, гнучкою та зручною для подальшої реалізації й підтримки.

Контейнеризація за допомогою Docker та організація багатокомпонентного середовища через Docker Compose стали ключовими технологічними елементами у побудові серверної інфраструктури. Docker дає змогу ізолювати

сервіси в незалежних середовищах, що містять повний стек необхідних залежностей, тоді як Docker Compose дозволяє описати конфігурацію кількох пов’язаних контейнерів у єдиному YAML-файлі, включаючи параметри взаємодії між ними, мережеву топологію та правила збереження даних. Інтеграція з Docker API забезпечує високий рівень автоматизації при виконанні типових операцій — запуску, завершенні, зборі інформації про статус контейнерів, а також дій, пов’язаних з обробкою даних.

Фронтенд додатку реалізовано за допомогою Django у поєднанні з JavaScript, що дало змогу створити інтуїтивно зрозумілий веб-інтерфейс із широким набором можливостей. Через графічну оболонку користувач має змогу управляти життєвим циклом серверних середовищ: ініціювати створення нових інстанцій, контролювати їх стан, отримувати детальні технічні метрики, взаємодіяти з окремими контейнерами, виконувати операції зі збереження та відновлювання даних, а також переглядати журнали подій. Завдяки інтеграції з сучасними веб-технологіями, інтерфейс системи поєднує простоту використання з гнучкістю управління, що дозволяє ефективно застосовувати його у бізнес-середовищах із підвищеними вимогами до надійності та масштабованості.

.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ WEB–ДОДАТКУ ДЛЯ КЕРУВАННЯ СЕРВЕРНИМ СЕРЕДОВИЩЕМ КОРПОРАТИВНОЇ ІНФОРМАЦІНОЇ СИСТЕМИ

3.1 Обґрунтування вибору мови та середовища програмування

Python є високорівневою мовою програмування, яка вирізняється простотою у використанні та високою читабельністю коду [14]. Її синтаксис наближений до природної мови, що значно полегшує написання й супровід програмного забезпечення. Завдяки підтримці кількох парадигм програмування — об’єктно-орієнтованої, процедурної та функціональної — Python залишається універсальним інструментом для розв’язання широкого спектра задач.

Серед основних переваг мови варто відзначити розгалужену стандартну бібліотеку, яка охоплює безліч модулів для обробки даних, роботи з файлами, мережевої взаємодії, багатопотоковості та інших напрямів. Python також має велике й активне ком’юніті фахівців та розробників, яке постійно поповнює екосистему новими бібліотеками, фреймворками та інструментами, що значно розширює можливості мови в різних галузях — від автоматизації до штучного інтелекту та веброзробки.

Мова Python має низку технічних характеристик, які забезпечують її популярність і універсальність у найрізноманітніших галузях розробки:

- Простота синтаксису та зручність читання коду. Однією з ключових концепцій у філософії Python є те, що код має бути максимально зрозумілим навіть для тих, хто бачить його вперше. Завдяки синтаксису, наближеному до природної мови, програмний код читається легко, що суттєво спрощує підтримку й колективну розробку.

- Мультиплатформність. Python є незалежною від операційної системи, що означає можливість запуску одного і того ж коду на різних платформах — Windows, Linux, macOS — без необхідності адаптації або внесення змін у структуру програми.

- Розширена базова бібліотека. Python має потужний набір вбудованих модулів, що дозволяє розробникам вирішувати велику кількість типових задач «із коробки». Це охоплює роботу з файлами, мережею, системними процесами, структурованими даними тощо — без додаткових залежностей.

- Гнучка інтеграція з іншими мовами. Мова програмування Python чудово поєднується з кодом, написаним іншими мовами — зокрема, C, C++ і Java. Це відкриває можливості для використання Python як зв'язувального шару або інтерфейсу у вже наявних рішеннях, розроблених іншими засобами.

- Придатність до масштабування. Незважаючи на те, що мова Python часто асоціюється з автоматизацією та написанням скриптів, його архітектурна гнучкість дозволяє створювати й повноцінні масштабовані застосунки, включаючи веб-сервіси, системи обробки великих даних і навіть промислові програмні продукти.

Одною з визначальних переваг Python як мови програмування є її високий рівень модульності та розширюваності. Архітектура Python дозволяє розробляти систему у вигляді окремих, незалежних модулів, які взаємодіють між собою за допомогою чітко окреслених інтерфейсів. Такий підхід значно спрощує підтримку коду та розширення функціоналу: у разі потреби можна додати нову підсистему або змінити існуючу, не порушуючи цілісності всієї програми. У межах реалізації веб-додатку для керування серверним середовищем корпоративної інформаційної системи це дає змогу зручно впроваджувати додаткові функції, пов'язані з керуванням Docker-контейнерами, обробкою системних метрик або інтеграцією з зовнішніми сервісами, без наявності потреби серйозної переробки існуючої кодової бази. Окрім того, розвинена екосистема Python забезпечує розробників великим вибором бібліотек і фреймворків, що дозволяє оперативно шукати та знаходити готові вирішення для типових задач. Це, у свою чергу, сприяє оптимізації витрат часу та ресурсів на етапах проєктування й реалізації програмного продукту.

Python часто обирають як основний інструмент для створення скриптів, орієнтованих на автоматизацію рутинних завдань і взаємодію з операційною системою. Він має достатньо просту структуру, яка є логічною побудова коду, що робить мову надзвичайно зручною для реалізації та написання надійних і дієвих автоматизованих рішень.

Завдячуючи синтаксису, який легко читається та інтуїтивно зрозумілий, Python-скрипти не лише швидко пишуться, а й легко адаптуються до нових вимог. Це вкрай суттєво у випадках, коли необхідно оперативно тестувати, налагоджувати або змінювати поведінку скрипта. Висока читабельність коду також значно полегшує діагностику та усунення помилок, що єдоволі важливим під час роботи з компонентами операційної системи, доступами до файлів, п Python надає розробникам гнучкі засоби для структуризації коду шляхом поділу його на модулі та пакети, що особливо актуально при створенні великих автоматизованих рішень. Такий підхід сприяє логічному упорядкуванню функціональності, дозволяючи розділити складну систему на менші, ізольовані частини, кожна з яких виконує конкретну задачу. Це значно полегшує підтримку та розширення коду в майбутньому.

Модульна архітектура Python створює умови для поетапного розширення можливостей скриптів — нові функції можуть додаватись без необхідності суттєвої переробки основної логіки. Завдяки цьому досягається масштабованість, що є важливою перевагою під час розробки довготривалих або складних автоматизаційних проєктів, де гнучкість і простота інтеграції нових елементів мають вирішальне значення.роцесами чи мережею.

У процесі розробки програмного забезпечення вибір середовища розробки відіграє важливу роль, адже саме від нього залежить зручність роботи розробника, ефективність процесу написання коду та можливості інтеграції з додатковими інструментами. Середовище розробки — це не просто текстовий редактор, а комплексний інструмент, який повинен забезпечити максимально комфортні умови для створення, налагодження, тестування та підтримки програмного продукту. Особливо це актуально в межах реалізації складних веб-

додатків, які включають багатокомпонентну архітектуру, взаємодію з контейнерами, інтеграцію з API та використання сторонніх бібліотек.

Вибір середовища розробки повинен базуватися на кількох ключових факторах: підтримці потрібної мови програмування, наявності розширень та плагінів, що дозволяють інтегрувати додатковий функціонал, зручності інтерфейсу, можливості працювати з системами контролю версій, а також підтримці роботи з контейнерами та терміналом. Важливо, щоб середовище було не тільки функціональним, але й легким у налаштуванні, стабільним і таким, що підтримується великою спільнотою, адже це значно полегшує вирішення можливих проблем.

У межах цієї роботи було обрано Visual Studio Code (VS Code) — потужне, універсальне та надзвичайно популярне середовище розробки, яке відповідає всім вищезазначеним критеріям. VS Code — це безкоштовний редактор коду з відкритим вихідним кодом, розроблений компанією Microsoft. Він підтримує численні мови програмування, зокрема Python, що є основною мовою для створення веб-додатку у цьому проєкті. Окрім того, завдяки інтегрованому терміналу, підтримці Git, численним розширенням і високій швидкодії, VS Code ідеально підходить для роботи над проєктами, що базуються на фреймворку Django та використовують контейнеризацію через Docker.

Середовище VS Code надає низку можливостей, які мають вирішальне значення для реалізації веб-додатку керування серверним середовищем корпоративної інформаційної системи. Завдяки плагінам, таким як Python, Docker, Django, YAML та інші, розробник може швидко працювати з відповідними конфігураціями, відлагоджувати код, запускати сервери безпосередньо з редактора та взаємодіяти з Docker-контейнерами у візуальному форматі. Це забезпечує тісну інтеграцію між кодом та інфраструктурними компонентами, що суттєво прискорює процес розробки й знижує ймовірність помилок.

Особливо корисною функцією VS Code є можливість налаштування робочого простору під специфіку проєкту. Наприклад, для даної роботи було

створено окреме середовище з прив’язкою до віртуального оточення Python, визначено шаблони запуску серверів та налаштовано автоматичне форматування коду згідно з PEP8. Крім того, за допомогою розширень стало можливим здійснювати статичний аналіз коду, перевіряти його на наявність помилок і потенційних вразливостей ще до запуску.

Інтеграція з Git дозволяє ефективно управляти змінами у коді, контролювати версії та працювати в команді, що є критично важливим для масштабних проєктів. У випадку потреби, VS Code також підтримує підключення до віддалених серверів через SSH, що дає змогу розгортати й тестувати код безпосередньо на хостинговому середовищі. Це особливо зручно, коли йдеться про управління серверними інстанціями та перевірку стабільності їх роботи.

Таким чином, Visual Studio Code є ідеальним вибором для розробки веб-додатку, який вимагає гнучкості, інтеграції з інструментами контейнеризації, роботи з Django та забезпечення стабільної та безперервної розробки. Його функціональність, розширюваність та дружній інтерфейс створюють ефективне середовище, здатне задовольнити потреби як індивідуального розробника, так і цілої команди. В умовах створення складного програмного рішення, орієнтованого на автоматизацію серверної інфраструктури, саме така інструментальна платформа дозволяє досягти високої якості, стабільності та зручності у розробці.

3.2 Обгрунтування вибору фреймворків та бібліотек

Django є одним із найпотужніших веб-фреймворків для мови програмування Python, який забезпечує швидку та ефективну розробку надійних, масштабованих і безпечних веб-застосунків [13]. Основна концепція, закладена в архітектуру фреймворку, - це спрощення розробницького процесу шляхом абстрагування від рутинних задач, пов’язаних із інфраструктурою. Завдяки цьому розробник може зосередитись на реалізації бізнес-логіки проєкту.

Однією з характерних рис Django є філософія «Batteries Included», яка передбачає забезпечення широкого набору готових інструментів і функціональностей у базовій комплектації. До них належать: вбудована система аутентифікації, зручний управлінський інтерфейс, Object-Relational Mapping (ORM) для взаємодії з базами даних, механізми маршрутизації, перевірки форм, обробки запитів тощо. Така повнота функціоналу «з коробки» дозволяє значно скоротити час на інтеграцію зовнішніх бібліотек, зменшити кількість залежностей і підвищити загальну цілісність проєкту.

Django також вирізняється добре продуманою структурою, високим рівнем безпеки та активною спільнотою, що робить його одним з найкращих рішень для побудови як невеликих веб-додатків, так і складних корпоративних систем.

Також важливою складовою функціоналу фреймворку Django є вбудований ORM (Object-Relational Mapping), який забезпечує зручну та безпечну взаємодію з базами даних. Завдячуючи цьому інструменту розробники можуть працювати з даними, використовуючи об'єктно-орієнтований підхід: таблиці бази даних відображаються у вигляді Python-класів, а записи — у вигляді об’єктів цих класів. Такий підхід дозволяє уникнути прямого написання SQL-запитів, що не лише підвищує швидкість розробки, але й знижує ризики, пов’язані з помилками та вразливостями, зокрема SQL-ін’єкціями.

Крім того, ORM Django значно спрощує керування міграціями, дозволяючи легко вносити зміни до структури бази даних шляхом редагування моделей. Зміни застосовуються автоматично за допомогою спеціальних інструментів командного рядка, що мінімізує людський фактор і сприяє підтримці узгодженості структури даних.

Ще однією важливою перевагою Django є адміністративна панель, яка генерується в автоматичному режимі на основі описаних моделей. Вона надає зручний веб-інтерфейс для перегляду, редагування та керування даними без потреби створювати окрему систему адміністрування. Адмін-панель легко налаштовується — можна змінювати вигляд, розширювати функціональність, додавати нові елементи інтерфейсу, що робить її особливо корисною як для

розробників, так і для технічного персоналу, який працює з даними у продакшн-середовищі.

Питання безпеки є одним із пріоритетних напрямів у розробці фреймворку Django, і саме тому він пропонує широкий спектр вбудованих засобів захисту від типових загроз. До них належать, зокрема, механізми протидії CSRF, XSS та SQL-ін’єкціям. Завдяки цим засобам основні аспекти безпеки обробляються автоматично, щонадає змогу розробникам сконцентруватися на реалізації прикладної логіки без необхідності вручну впроваджувати типові захисні механізми.

Окрему увагу в Django приділено маршрутизації запитів - система побудована таким чином, щоб бути водночас простою в налаштуванні й достатньо гнучкою для реалізації як базових, так і складних сценаріїв. Завдяки зрозумілим правилам конфігурації URL-адрес, розробник може швидко створювати інтуїтивну навігаційну структуру веб-застосунку, що покращує користувацький досвід і спрощує обслуговування [13].

Ще однією сильною стороною Django є легкість інтеграції з зовнішніми бібліотеками та застосунками. Фреймворк підтримує підключення сторонніх пакетів через стандартний механізм додатків, що дозволяє швидко розширювати функціональність проєкту без суттєвого втручання в основний код. Такий підхід забезпечує високу адаптивність і гнучкість під час розвитку та масштабування програмного рішення.

Однією з важливих переваг Django є підтримка створювання багаторазових компонентів і модулів, які дозволяють ефективно організовувати структуру проєкту та суттєво скорочувати час на розробку нових функціональних блоків [13]. Завдяки можливості повторного використання коду, розробники можуть будувати масштабовані рішення, уникаючи дублювання логіки та зменшуючи ймовірність помилок.

Фреймворк зорієнтований на пришвидшений цикл розробки та розгортки, надаючи розробнику повний набір вбудованих інструментів для запуску, проведення тестів та підтримки веб-застосунків. Django дозволяє ефективно

керувати конфігураціями, завдяки чому легко адаптовувати застосунок до різних середовищ - наприклад, розробницького, тестового або продуктивного. Така гнучкість конфігураційного механізму сприяє плавному переходу від етапу розробки до реального користання, забезпечуючи стабільність і контрольованість процесу розгортання.

Вибір фреймворку Django як основи для реалізації веб-додатку зумовлений сукупністю його технічних переваг: високою продуктивністю, розвиненою системою безпеки, архітектурною гнучкістю та зручністю у використанні. Застосування цього інструменту дозволить суттєво скорочувати терміни розробки, забезпечити структуровану та підтримувану кодову базу, а також реалізувати стабільну й захищену роботу застосунку. У контексті створення системи для керування серверним середовищем корпоративної інформаційної системи, ці характеристики є надзвичайно важливими, оскільки гарантують надійність і масштабованість рішення, що прямо впливає на ефективність усього IT-інфраструктурного процесу.

python_on_whales - це сучасна Python-бібліотека, за допомогою якої забезпечується зручний високорівневий інтерфейс для взаємодії з Docker. Вона дозволяє керувати томами, контейнерами, мережами та іншими ресурсами через зрозумілі Python-методи, що значно спрощує розробку скриптів для автоматизації процесів [14]. Важливою перевагою є повна підтримка Docker Compose, що робить бібліотеку особливо корисною для проєктів, у яких кожне серверне середовище можна розгортати як окремий багатоконтейнерний застосунок.

З використанням python_on_whales розробник отримує можливість зручно запускати нові інстанції, керувати вже існуючими середовищами, а також виконувати бекап і відновлювання даних у разі потреби [14]. Це рішення забезпечує гнучкий та продуктивний підхід до керування контейнеризованими сервісами.

psutil - це функціонально насичена бібліотека для Python, за допомогою якої можна отримувати детальні дані про стан системних ресурсів та активні

процеси. Ця бібліотека надає засоби для відстеження завантаження CPU, обсягу використаної пам’яті, роботи з дисками, мережевої активності, а також температурних та інших апаратних показників [20]. Такий набір можливостей робить бібліотеку доволі корисною для реалізації систем моніторингу та керування ресурсами. Суттєвою перевагою psutil є її кросплатформність — вона коректно працює в середовищах macOS, Linux та Windows, що дозволяє використовувати її в універсальних рішеннях. Бібліотека дає змогу зчитувати ключові характеристики процесів, зокрема PID, статус, навантаження на CPU і споживання оперативної пам’яті. Ця інформація є критично важливою для контролю за стабільністю роботи системи та своєчасного виявлення потенційних проблем.

Завдяки зрозумілому та зручному API, psutil надає можливість без зайвих складнощів інтегрувати функціонал моніторинга в програмне забезпечення. У рамках нашого проєкту бібліотека буде використана для збору даних про навантаження на процесор, використання RAM, кількість виконуваних процесів, а також інші ключові метрики. Отримана інформація відображатиметься у веб-інтерфейсі, що дає змогу користувачам оперативно оцінювати технічний стан системи та своєчасно реагувати на зміни.

os - це стандартний модуль мови Python, що надає інструменти для взаємодії з функціями ОС[19]. Він дозволяє працювати з файловою структурою, запускати та контролювати процеси, а також виконувати системні команди. Бібліотека os є важливою складовою стандартного набору Python і широко застосовується для реалізації базової функціональності в багатьох проєктах.

Модуль os містить розширений набір функцій для роботи з системою файлів. З його допомогою можна перейменовувати, створювати, видаляти або переміщувати не тільки окремі файли, але і цілі директорії. Це ефективно під час розробки скриптів, які мають автоматизувати дії з файлами — наприклад, зберігання логів, формування резервних копій чи створення тимчасових службових файлів. Крім того, бібліотека надає інструменти для взаємодії з виконувальним середовищем, зокрема доступ до змінних оточення, даних про

поточний процес і дочірні процеси. Це дозволяє адаптувати поведінку застосунку відповідно до конфігурації операційної системи, що є корисним для гнучкого налаштування в різних середовищах.

3.3 Програмна реалізація WEB-додатку

У цьому розділі буде розглянуто програмну реалізацію веб-додатку, призначеного для автоматизованого керування та моніторингу серверним середовищем корпоративної інформаційної системи. Розроблений функціонал забезпечує запуск, зупинку, моніторинг і адміністрування контейнерів Docker, які відповідають за роботу окремих інстанцій системи. Основна увага приділиться головним можливостям веб-інтерфейсу.

На початковому етапі було створено Django-проєкт, який став основою для реалізації веб-додатку. До проєкту були підключені всі потрібні для розробки бібліотеки: docker, psutil, os, crypt, python_on_whales, cpuinfo, platform, а також стандартизовані компоненти фреймворку Django. Цей набір інструментів забезпечить повноцінну взаємодію з Docker-контейнерами та ОС.

Архітектура бекенду структурована на два основні блоки: API-функції, що відповідають за передачу даних на фронтенд частину інтерфейсу, та утилітарні функції, які виконують прикладні дії — зокрема, ініціалізацію нового середовища на базі окремого екземпляра системи.

Далі розглядаються основні API-функції, які відповідають за обробку системної інформації, зокрема даних щодо використання swap-пам’яті, оперативної пам’яті, дискового простору, кількості активних процесів, тривалості роботи сервера, а також швидкості мережевого завантаження та вивантаження. Отримані значення обробляються на серверній стороні, після чого передаються у фронтенд-частину веб-додатку. Там дані опрацьовуються засобами JavaScript, і результат виводиться на сторінку у вигляді графіків або текстових показників.

Функція get_swap_stats дозволяє отримувати поточні дані про стан swap-пам’яті. З використанням можливостей бібліотеки psutil визначається як загальний обсяг виділеного простору для swap, так і фактично використане значення. Крім того, обчислюється відсоткове співвідношення зайнятої пам’яті. Отримані значення передаються у фронтенд у форматі JSON і можуть бути відображені на дашборді для моніторингу стану оперативної та допоміжної пам’яті системи.

```
def get_swap_stats(request):
    swap = psutil.swap_memory()
    data = {
        'swap_total': swap.total,
        'swap_used': swap.used,
        'swap_percent': swap.percent
    }
    return JsonResponse(data)
```

Функція get_disk_stats для збору інформації про дисковий простір повертає основні метрики, зокрема загальний обсяг накопичувача, використане місце та його заповненість у відсотках. Це дозволяє на стороні клієнта реалізувати візуальний контроль за завантаженістю системи та попереджати користувача про можливу нестачу вільного місця.

```
def get_disk_stats(request):
    disk = psutil.disk_usage('/')
    data = {
        'disk_total': disk.total,
        'disk_used': disk.used,
        'disk_percent': disk.percent,
    }
```

```
    return JsonResponse(data)
```

На рисунку 3.1 зображено показники, які вказують відсоток завантаження swap та загального простору диску.

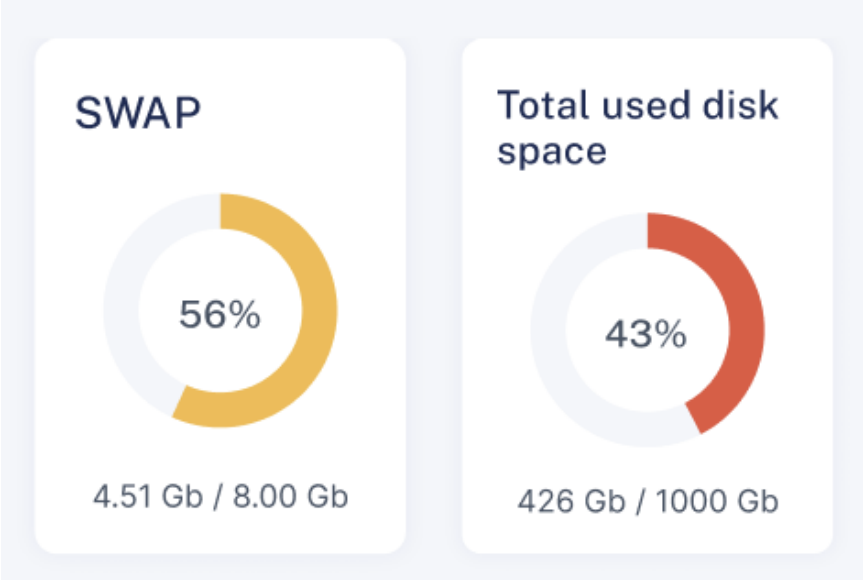

Рисунок 3.1 – Відображення показників, які вказують відсоток завантаження swap та загального простору диску

Функція get_system_info використовується для збору загальної інформації про апаратне та програмне забезпечення системи. Вона отримує дані про модель центрального процесора, обсяг оперативної пам’яті, кількість доступних ядер, а також тип та версію операційної системи. Для реалізації цього функціоналу залучаються бібліотеки platform, psutil і cpuinfo. Результати обробки повертаються у JSON-форматі, що дозволяє легко передавати їх у клієнтську частину веб-додатку.

```
def get_system_info(request):
    context = {
        "cpu": get_cpu_info()['brand_raw'],
        "cpu_cores": psutil.cpu_count(),
        "ram": f'{psutil.virtual_memory().total / (1024**2):.3f} MB',
```

```
        "os": f'{platform.system()} {platform.release()}'
    }
    return JsonResponse(context)
```

На рисунку 3.2 зображено інтерфейс з відображенням параметрів ОС на головній сторінці з дашбордом.

Рисунок 3.2 – Відображення системних параметрів операційної системи

Функція get_memory_info відповідає за збір показників щодо стану RAM та накопичувача. У її межах обчислюються загальний обсяг та поточне використання як RAM, так і простору диску, включаючи розрахунок заповненості. Зібрана інформація повертається у JSON-форматі, це дозволяє виводити на веб-інтерфейсі актуальні дані про навантаження на оперативну пам’ять та зайнятий простір на диску.

```
    def get_memory_info(request):
    used = 0
    total = 0
    for disk in psutil.disk_partitions():
```

```
        used += psutil.disk_usage(disk.mountpoint).used
        total += psutil.disk_usage(disk.mountpoint).total
    context = {
        'ram_used': f'{psutil.virtual_memory().used / (1024**3):.3f} GB',
        'ram_remnant': psutil.virtual_memory().percent,
        'used': f'{used / (1024**3):.3f} GB',
        'total': f'{total / (1024**3):.3f} GB',
        'drive_remnant': f'{used * 100 / total:.3f} %'
    }
    return JsonResponse(context)
```

На рисунку 3.3 зображено візуалізація обсягу використаної RAM та загального заповнення

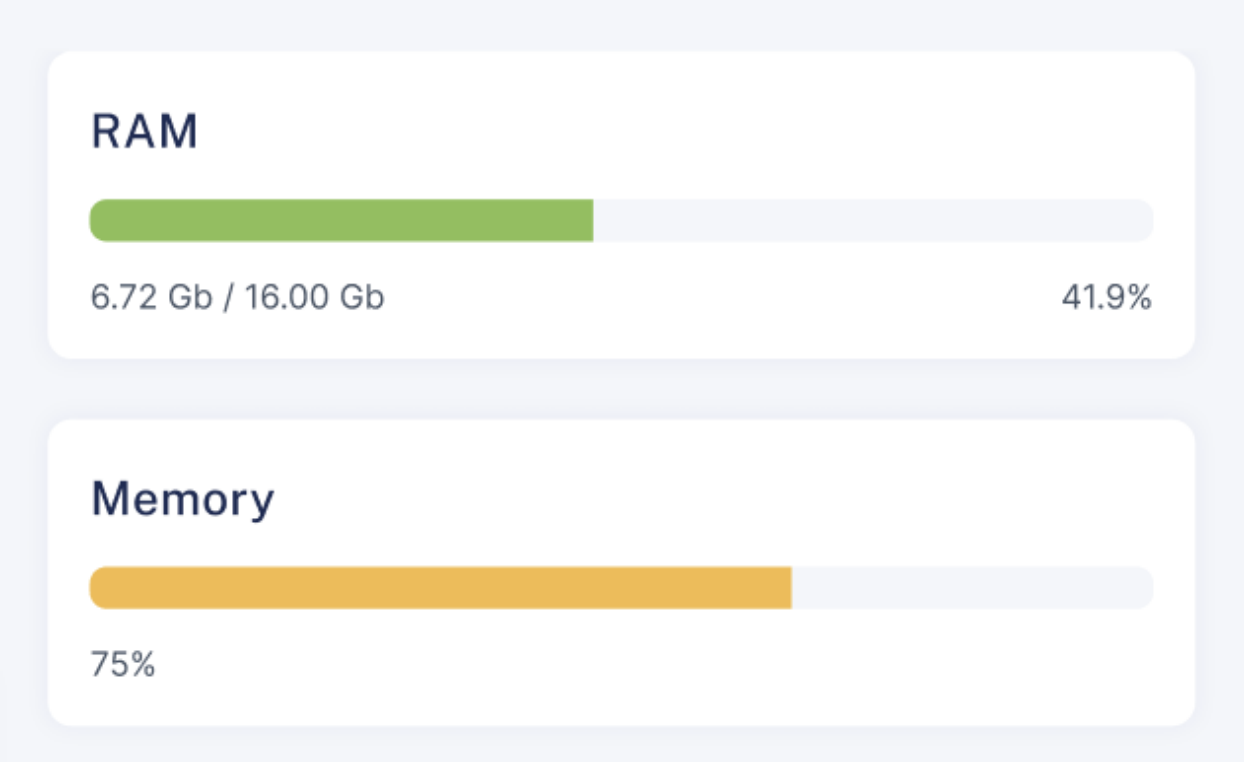

Рисунок 3.3 - Відображення показників обсягу використаної RAM та загального заповнення пам’яті

Окрема функція реалізована для визначення кількості активних процесів у системі. Вона використовує можливості бібліотеки psutil, зокрема метод psutil.pids(), який повертає список ідентифікаторів усіх поточних процесів. Шляхом підрахунку довжини цього списку визначається загальна кількість запущених екземплярів. Це дозволяє користувачеві оперативно оцінити навантаження на систему з боку програмного забезпечення.

Ще одна функція відповідає за отримання інформації про тривалість безперервної роботи серверного середовища. За допомогою методу psutil.boot_time() визначається час останнього перезавантаження системи, після чого шляхом віднімання цього значення від поточного часу обчислюється uptime. Отриманий результат подається у форматі, зручному для сприйняття (години, хвилини), що дає змогу адміністраторам контролювати стабільність роботи серверу.

```
def get_process_count(request):
    process_count = len(psutil.pids())
    return JsonResponse({'process_count': process_count})

def get_uptime(request):
    boot_time = psutil.boot_time()
    current_time = time.time()
    uptime_seconds = int(current_time - boot_time)
    hours = uptime_seconds // 3600
    minutes = (uptime_seconds % 3600) // 60

    return JsonResponse({'uptime': f'{hours}h {minutes}m'})
```

На рисунку 3.4 зображено відображення на дашборді головної сторінки кількість запущених процесів та час роботи серверу.

Дані, що генеруються API-функціями на сервері, передаються до клієнтської частини з допомоги JavaScript-запитів. Під час завантаження або оновлення сторінки ініціюється звернення до відповідного маршруту, де запит оброблюються серверною функцією, яка повертає результат у JSON-форматі. Отримані дані динамічно вносяться в елементи сторінки за допомогою HTML-структур, а стилізація здійснюється через CSS, що формує остаточне візуальне представлення інформації про систему для користувача.

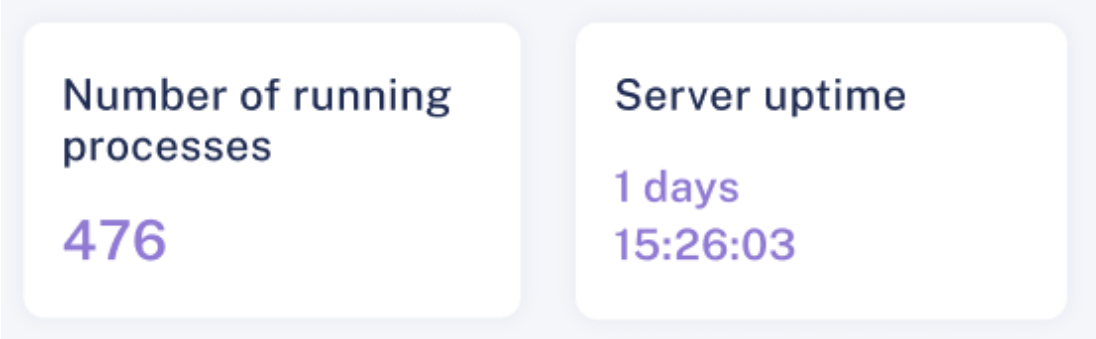

Рисунок 3.4 – Відображення показників кількості запущених процесів та часу роботи серверу

На прикладі функції install_instance можна розглянути принцип роботи утилітних функцій, які реалізують ключові дії з управління екземплярами через Docker.

Ці функціональні блоки реалізують повний цикл управління інстанціями серверного середовища: вони автоматизують створення нових середовищ, забезпечують можливість видалення вже неактуальних, здійснюють контроль над окремими контейнерами, а також відповідають за збереження резервних копій та їх подальше відновлення для окремих сайтів у межах системи.

Функція install_instance виконує одну з центральних ролей у процесі автоматичного розгортання щойно створеного екземпляра. Її робота поділена на кілька послідовних етапів:

- Перевірка на наявність необхідного репозиторію та його клонуванню: На початковому етапі функція install_instance виконує перевірку,

чи існує локальний каталог frappe_docker за вказаним шляхом. Якщо директорія відсутня, ініціюється клонування репозиторію з офіційного джерела GitHub. Це забезпечує наявність усіх необхідних файлів і шаблонів для подальшої роботи з контейнерами.

```
if not os.path.isdir(os.path.join(projects_path, 'frappe_docker')):
os.system(f'git clone https://github.com/frappe/frappe_docker.git {projects_path}/frappe_docker')
```

- Traefik - налаштування. У разі, коли компонент Traefik не був налаштований попередньо, система автоматично створює необхідні файли конфігурацій та ініціює запуск окремого Docker Compose-проєкту для його розгортання. Зокрема, формується директорія для Traefik, генерується .env файл із параметрами доменного імені, електронної пошти та хешованого пароля, після чого виконується запуск сервісу через відповідні YAML-файли конфігурації.

```
  if not os.path.isdir(os.path.join(projects_path, 'traefik')):
os.makedir(os.path.join(projects_path, 'traefik'))
with open(os.path.join(projects_path, 'traefik', 'traefik.env'), 'w') as traefik_env:
    traefik_env.write(f'TRAEFIK_DOMAIN={traefik_domain}\n')
    traefik_env.write(f'EMAIL={traefik_email}\n')

  traefik_env.write(f'HASHED_PASSWORD={traefik_hashed_password}\n')

traefik_compose = DockerClient(
    compose_project_name='traefik',
    compose_env_file=os.path.join(projects_path, 'traefik', 'traefik.env'),
    compose_files=[
```

```
            os.path.join(frappe_docker_path, 'overrides', 'compose.traefik.yaml'),
            os.path.join(frappe_docker_path, 'overrides', 'compose.traefik-
            ssl.yaml')
        ]
    )
    traefik_compose.compose.up(detach=True)
```

- Створення та початкове налаштування проєкту. На цьому етапі здійснюється формування базової структури нового серверного середовища. Створюється окрема директорія для проєкту, у якій розміщуються сконфігуровані файли для підключення до бази даних MariaDB та налаштування змінних середовища.

```
    project_folder = os.path.join(projects_path, project_name)
os.mkdir(project_folder)
with open(os.path.join(project_folder, 'mariadb.env'), 'w') as mariadb_env:
    mariadb_env.write(f'DB_PASSWORD={mariadb_password}\n')
copyfile(os.path.join(frappe_docker_path, 'overrides', 'compose.mariadb-
shared.yaml'), os.path.join(project_folder, 'compose.mariadb-shared.yaml'))
with open(os.path.join(project_folder, 'compose.mariadb-shared.yaml'), 'r+')
as mariadb_yaml:
    lines = mariadb_yaml.readlines()
    mariadb_yaml.seek(0)
    for line in lines:
        if 'mariadb-database' in line:
            line = line.replace('mariadb-database', f'{project_name}-mariadb-
            database')
        mariadb_yaml.write(line)
```

```
copyfile(os.path.join(frappe_docker_path, 'example.env'),
os.path.join(project_folder, f'{project_name}.env'))
  with open(os.path.join(project_folder, f'{project_name}.env'), 'r+') as
  project_env:
    lines = project_env.readlines()
    project_env.seek(0)
    for line in lines:
      if 'DB_PASSWORD=123' in line:
          line = line.replace('DB_PASSWORD=123',
          f'DB_PASSWORD={mariadb_password}')
      if 'DB_HOST=' in line:
          line = line.replace('DB_HOST=', f'DB_HOST={project_name}-
          mariadb-database')
      if 'DB_PORT=' in line:
        line = line.replace('DB_PORT=', f'DB_PORT=3306')
      if 'SITES=erp.example.com' in line:
        line = line.replace('SITES=erp.example.com', f'SITES={site_url}')
      project_env.write(line)
    project_env.write(f'ROUTER={project_name}\n')
    project_env.write(f'BENCH_NETWORK={project_name}\n')
```

- Запуск проєктів через Docker Compose. Після завершення підготовки конфігураційних файлів виконується ініціалізація окремих сервісів контейнерної інфраструктури за допомогою Docker Compose. У першу чергу здійснюється запуск контейнера з базою даних MariaDB, який використовує індивідуальний .env-файл та відповідний YAML-файл конфігурації.

```
mariadb_compose = DockerClient(
    compose_project_name=f'{project_name}-mariadb',
    compose_env_file=os.path.join(project_folder, 'mariadb.env'),
```

```
            compose_files=[os.path.join(project_folder, 'compose.mariadb-
    shared.yaml')]
        )
        mariadb_compose.compose.up(detach=True)
```

- Початкове налаштування нового сайту серверного середовища: після запуску базової інфраструктури здійснюється розгортання основного контейнерного середовища, в межах якого працюватиме корпоративна інформаційна система. Створюється новий сайт шляхом запуску відповідних контейнерів із врахуванням конфігураційного набору, який включає основний файл compose.yaml та низку доповнень, мультибенч-середовища та SSL-підтримки.

```
          project_compose = DockerClient(
            compose_project_name=project_name,
            compose_env_file=os.path.join(project_folder, f'{project_name}.env'),
            compose_files=[
              os.path.join(frappe_docker_path, 'compose.yaml'),
              os.path.join(frappe_docker_path, 'overrides', 'compose.redis.yaml'),
              os.path.join(frappe_docker_path, 'overrides', 'compose.multi-
bench.yaml'),
              os.path.join(frappe_docker_path, 'overrides', 'compose.multi-bench-
ssl.yaml')
            ]
        )
        project_config = project_compose.compose.config(return_json=True)
        yaml_config = dump(project_config, default_flow_style=False,
sort_keys=False)
        with open(os.path.join(project_folder, f'{project_name}.yaml'), 'w') as
project_yaml:
```

```
            project_yaml.write(yaml_config)
        project_compose.compose_files = [os.path.join(project_folder,
f'{project_name}.yaml')]
        project_compose.compose.up(detach=True)
```

Для запуску нового серверного середовища виконується конфігурація проєкту через інтерфейс DockerClient. На цьому етапі визначаються ключові параметри: ім’я проєкту, шляхи до YAML-файлів конфігурації та файлів змінних середовища. Після цього вміст змінних з .env файлів автоматично підставлюються у структуру конфігурації, що дозволяє сформувати повну картину розгортання. Після генерації підсумкового YAML-файлу проєкт запускається як окрема інстанція Docker Compose, яка об’єднує всі контейнери в єдиний керований об’єкт.

На рисунку 3.5 представлено інтерфейс, що відображає перелік усіх розгорнутих проєктів у серверному середовищі.

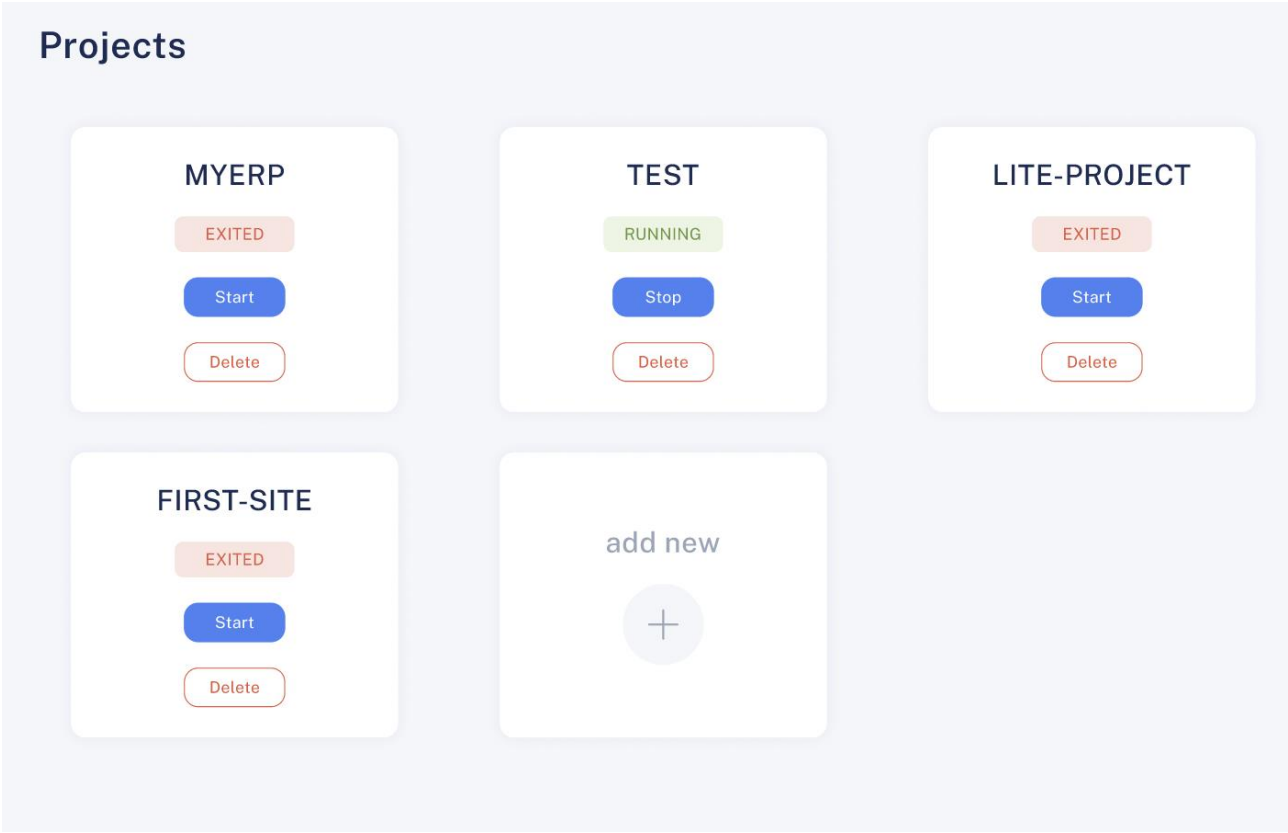

Рисунок 3.5 – Загальний вигляд інтерфейсу, що відображає розгорнуті проєкти

3.4 Тестування роботи WEB-додатку

Після запуску додатку, користувач потрапляє на головну сторінку. На ній можна побачити дашборд з різними метриками, такими як, SWAP, швидкість трафіку, стан пам’яті та навантаженість на процесор, кількість запущених процесів, час роботи серверу та системні характеристики. На зображені 3.6 зображено головна сторінка з метриками системи.

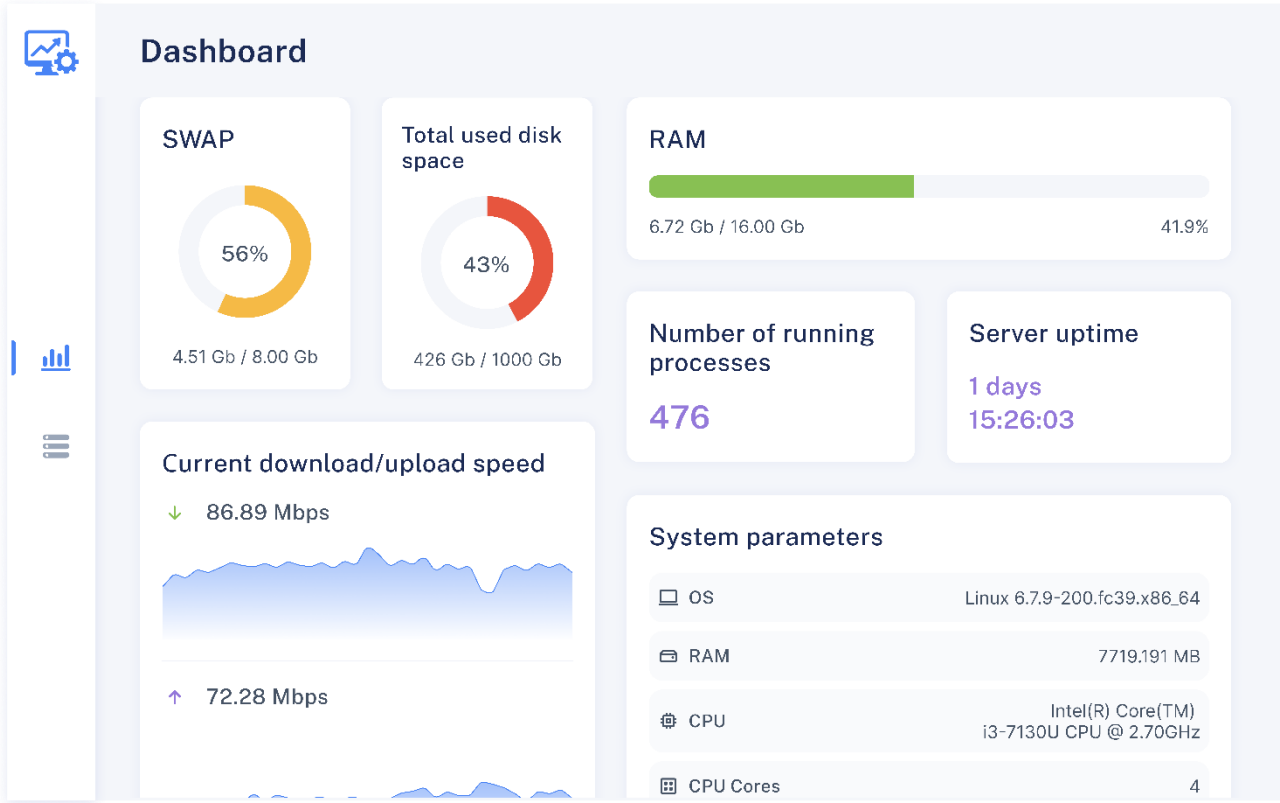

Рисунок 3.6 – Загальний вигляд головної сторінки з метриками системи

Далі перейшовши на іншу вкладку, користувач може побачити всі розгорнуті проєкти в середовищі. В кожного проєкту відображається його назва, стан, є можливість зупинки чи запуску а також видалення проєкту. Сторінку з відображенням всіх проєктів можна побачити на рисунку 3.7.

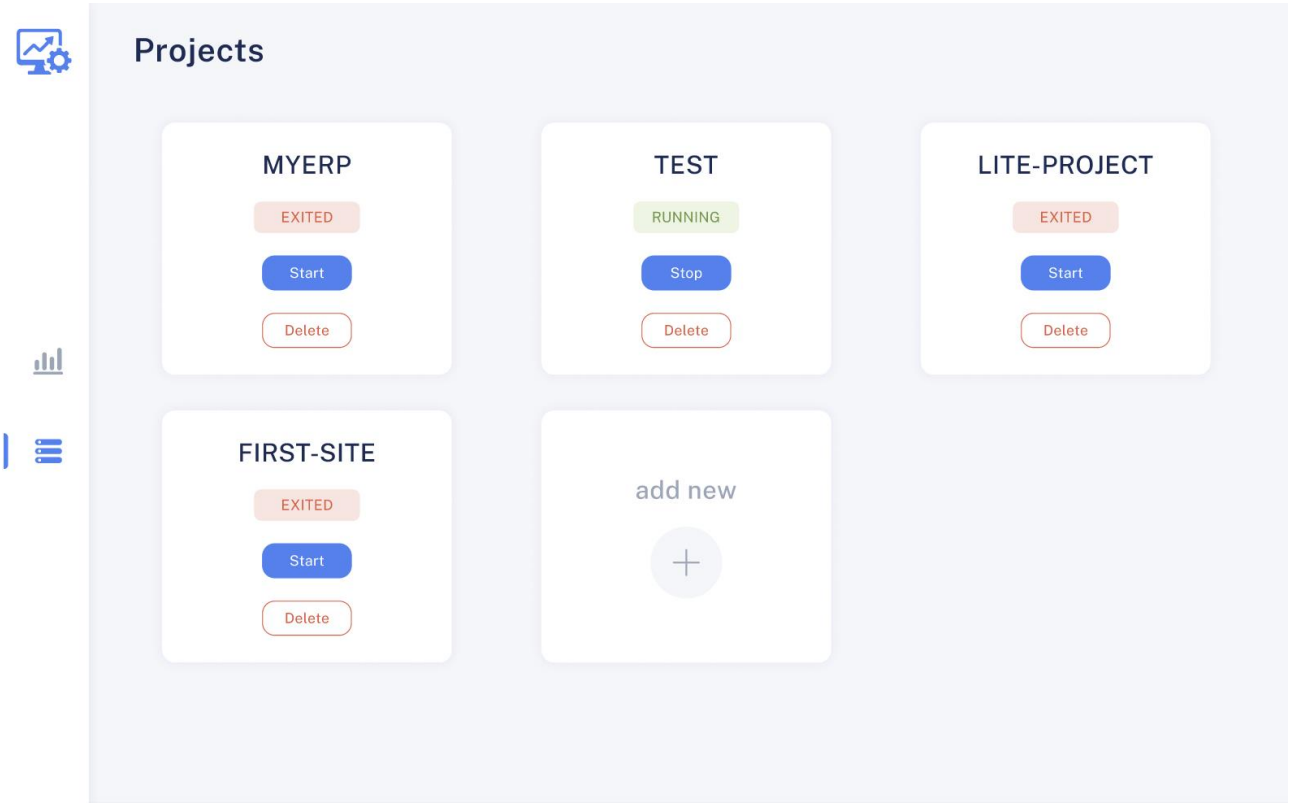

Рисунок 3.7 – Загальний вигляд інтерфейсу з відображенням всіх розгорнутих проєктів в середовищі

На сторінці проєкту можна побачити кнопку ‘add new’ за допомогою якої можна створити інший екземпляр сайту. Після натиснення на даний блок, користувач побачить форму інсталяції. Коли користувач заповнить дану форму додаток почне процес розгортання нової інстанції серверного середовища, як проєкт Docker Compose.

В разі, якщо у середовищі відсутній попередньо налаштований реверсивний проксі Traefik, користувачеві необхідно додатково ввести параметри, необхідні для його конфігурації. Це забезпечує коректне маршрутизування запитів до цільового сервісу після розгортання інстанції. Інтерфейс форми додавання екземпляра, що використовується для налаштування та ініціалізації розгортання, представлено на рисунку 3.8.

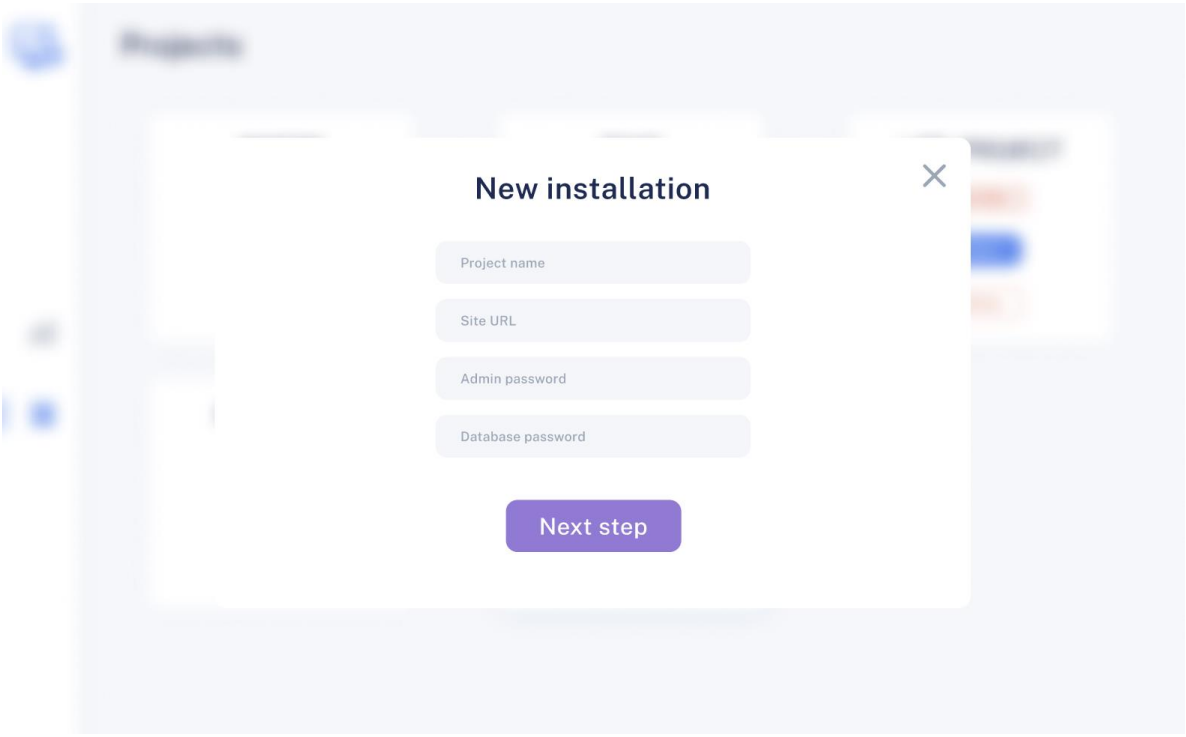

Рисунок 3.8 - Загальний вигляд вікна з формою додавання нового екземпляру

На рисунку 3.9 проілюстровано загальний вигляд вікна з формою конфігурації компонента Traefik.

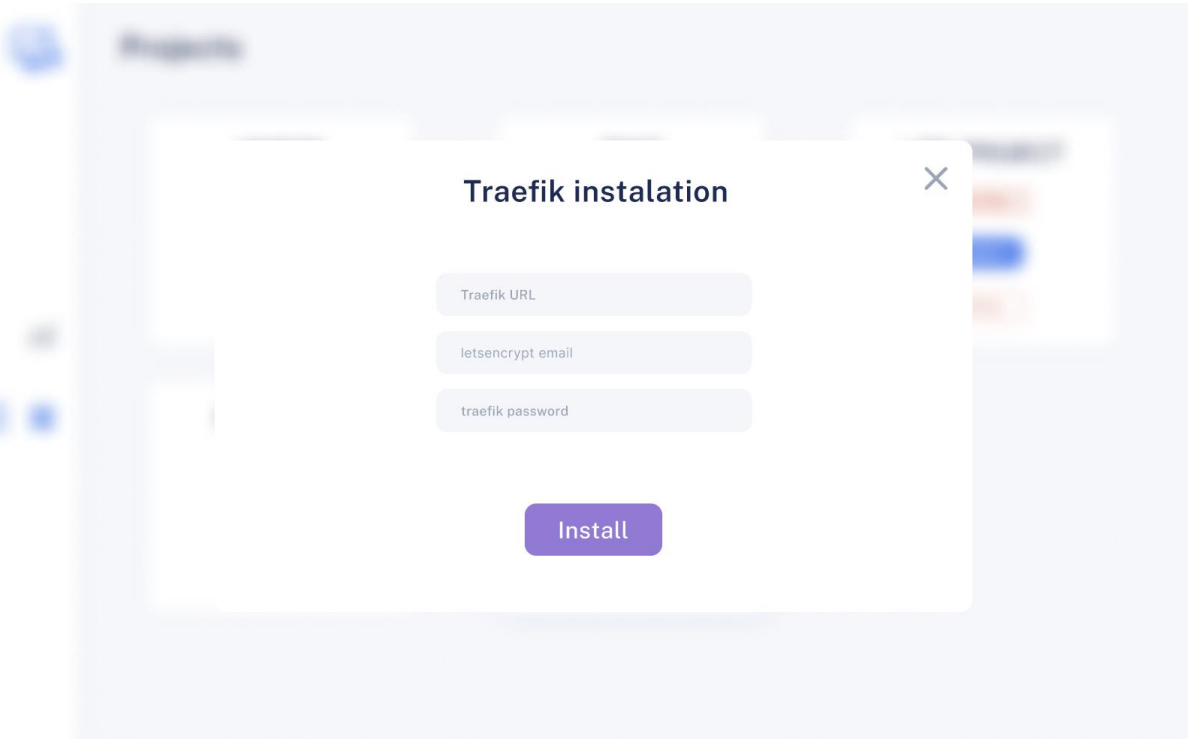

Рисунок 3.9 - Загальний вигляд вікна з формою конфігурації конфігурації Traefik

Коли користувач переходить на конкретрий проєкт, він потрапляє на сторінку попередньо обраного ним проєкту, тут розміщені всі наявні контейнери проєкту. На сторінці відображається інформація про id кожного контейнера, його назву, час з моменту створення, його статус. Також реалізована можливість зупинки чи запуску контейнерів. Сторінка з відображенням деталей проєкту зображена на рисунку 3.10.

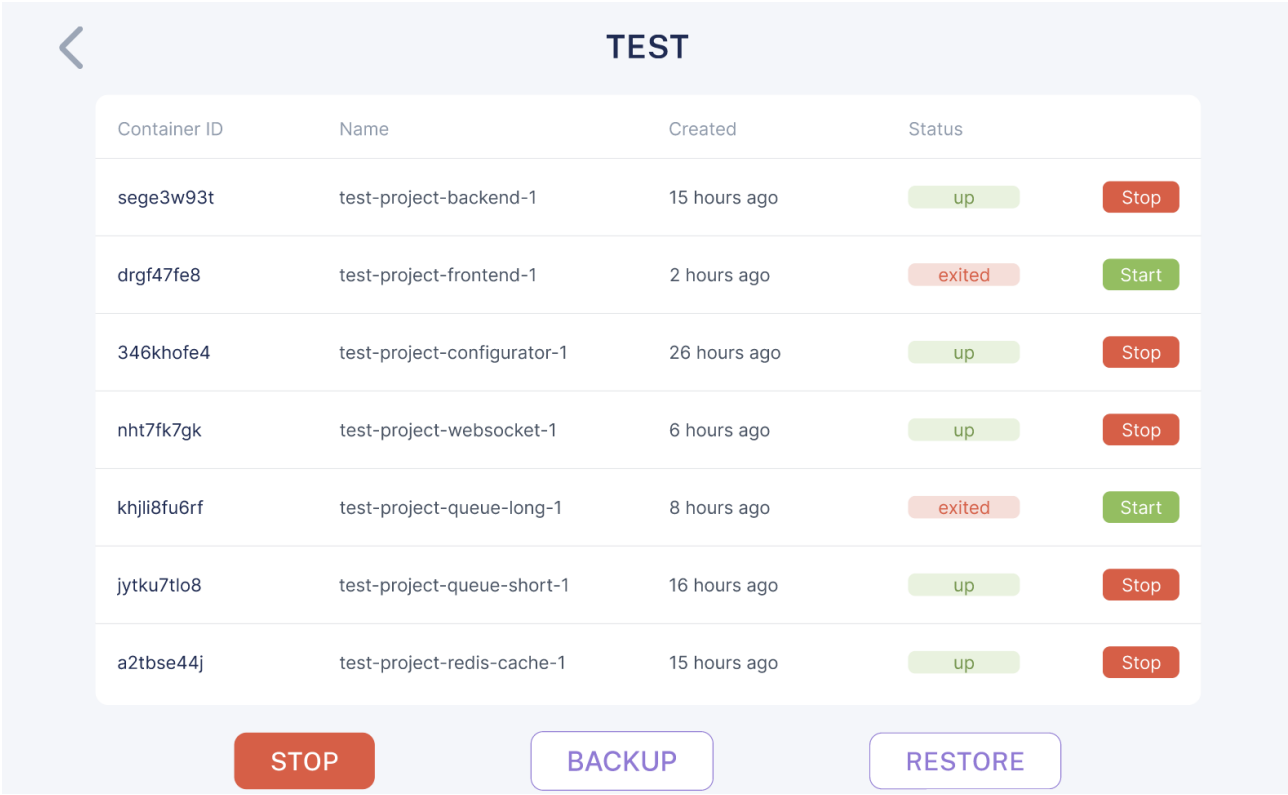

Рисунок 3.10 – Відображення деталей проєкту

Наступним функціональним блоком на сторінці є панель управління окремим проєктом, яка забезпечує швидкий доступ до ключових операцій адміністрування інстанції. Інтерфейс панелі адаптується до поточного стану серверного середовища: якщо проєкт зупинено, з’являється кнопка Start для його запуску; у разі активного стану — доступна кнопка Stop, яка дозволяє призупинити роботу інстанції. Також доступні дії Backup (ініціація створення

резервної копії бази даних сайту) та Restore (відновлення бази з попередньо збереженого бекапу).

Усі ці операції реалізовані через взаємодію з Docker API та виконуються в один клік, без необхідності використання командного рядка чи складної ручної конфігурації. Такий підхід значно підвищує зручність користування додатком та мінімізує ризик помилок під час адміністрування.

3.5 Аналіз результатів

Після завершення етапу тестування реалізованого веб-додатку доцільним є проведення порівняльного аналізу отриманих результатів з характеристиками існуючих рішень, що виконують схожі функції. Такий підхід дозволяє об’єктивно оцінити рівень функціональності, зручності та ефективності розробленої системи у контексті сучасних альтернатив. У межах цього розділу буде здійснено порівняння з трьома попередньо обраними платформами — Frappe Cloud, Render та Railway, які застосовуються для автоматизації розгортання, адміністрування та моніторингу серверних середовищ. Аналіз проводиться за низкою критичних параметрів, що мають безпосереднє значення для керування серверного середовища корпоративної інформаційної системи.

Аналіз здійснюється за такими ключовими критеріями: наявність функцій моніторингу системних ресурсів, можливість резервного копіювання та відновлення даних, зручність користувацького інтерфейсу, швидкість розгортання інстанцій, та контроль над окремими контейнерами. Вибір платформ Frappe Cloud, Render, Railway та власного веб-додатку обумовлений їх популярністю, широким застосуванням у сфері хостингу серверних середовищ, а також наявністю функціоналу, що частково перетинається з реалізованим рішенням. Такий підхід дозволяє порівняти альтернативи й оцінити конкурентоспроможність власної розробки.

В таблиці 3.1 наверено порівняльну характеристику за ключовими критеріями.

Таблиця 3.1 – Порівняння WEB – додатків для керування серверним середовищем корпоративної інформаційної системи.

	Frappe Cloud	Render	Railway	Реалізований web - додаток
Зручність інтерфейсу	+	-	+/-	+
Моніторинг стану системи та ресурсів	+/-	-	+/-	+
Можливість резервного копіювання та відновлення	+	+/-	-	+
Контроль над окремими контейнерами	-	-	-	+
Швидкість розгортання інстанцій	15 хв	20 хв	17 хв	6 хв

Щодо зручності та інтуїтивності інтерфейсу, найкращі результати демонструють Railway та реалізований веб-додаток, які відзначаються інтуїтивно зрозумілим та комфортним у користуванні інтерфейсом. Frappe Cloud також можна вважати достатньо простим у використанні. Натомість Render виявився найменш зручним — його інтерфейс перевантажений і потребує часу для адаптації.

У контексті моніторингу стану системи та ресурсів, жодна з платформ, окрім реалізованого веб-додатку, не забезпечує повноцінного функціоналу. У Frappe Cloud та Railway доступний лише базовий моніторинг, тоді як Render надає логування, однак не підтримує перегляд розширених метрик. Реалізований веб-додаток суттєво вирізняється тим, що включає повноцінний моніторинг із можливістю відстеження всіх необхідних системних показників.

Якщо порівнювати можливість резервного копіювання та відновлення, всі аналізовані сервіси (Frappe Cloud, Render, Railway) мають функції бекапу, але переважно в автоматизованому режимі або з обмеженнями, залежно від тарифу. У випадку реалізованого додатку користувач може створити резервну копію в будь-який момент, що значно розширює можливості керування й підвищує гнучкість у критичних ситуаціях.

Аналіз можливості контролю окремих контейнерів показав відсутність такої опції у всіх розглянутих платформах, крім реалізованого рішення, яке надає безпосередній контроль над кожним окремим контейнером — важлива функціональність для адміністраторів серверного середовища.

Швидкість розгортання інстанцій у реалізованому веб-додатку також суттєво переважає аналоги. Якщо для Frappe Cloud, Render і Railway цей процес займає від 15 до 20 хвилин, то у реалізованому рішенні інстанція готова до роботи вже за 6 хвилин, що свідчить про високу оптимізацію та ефективність процесів розгортання.

У підсумку, проведене порівняння демонструє, що реалізований веб-додаток не лише успішно поєднує зручність інтерфейсу та швидкість розгортання, а й перевершує існуючі сервіси за критично важливими параметрами: розширеним моніторингом, контролем контейнерів та гнучкістю резервного копіювання.

3.6 Висновки до розділу 3

В цьому розділі було здійснено повноцінну реалізацію веб-додатку, призначеного для керування серверним середовищем корпоративної інформаційної системи. Описано ключові етапи побудови додатку, починаючи з вибору мови програмування, фреймворку та бібліотек, і завершуючи практичною реалізацією функціональних модулів. Розроблене рішення поєднує автоматизоване розгортання інстанцій, оперативний моніторинг основних

системних параметрів та гнучке керування сервісами на основі контейнерної архітектури.

API-функціонал веб-додатку забезпечує збір даних про стан ресурсоємних компонентів системи — процесора, оперативної пам’яті, дискового простору, swap-пам’яті, кількості активних процесів, а також часу безперервної роботи та мережевої активності. Усі ці метрики обробляються та відображаються у веб-інтерфейсі в реальному часі. Така реалізація дає змогу забезпечити високий рівень прозорості та доступності критичних показників системи, що є надзвичайно важливим для стабільного адміністрування. Також була приділена увага реалізації утилітарних функцій, які автоматизують створення нових інстанцій корпоративної інформаційної системи. Ці функції охоплюють повний цикл: від перевірки наявності необхідних компонентів — до генерації середовища за допомогою Docker Compose.

Таким чином, побудований додаток демонструє повноцінну готовність до використання в реальних умовах. Його модульна архітектура дозволяє легко масштабувати функціонал, а візуально зручний інтерфейс спрощує виконання рутинних адміністративних дій. Усі реалізовані компоненти працюють узгоджено, що дозволяє швидко реагувати на зміни в інфраструктурі та підтримувати її стабільну роботу.

ВИСНОВКИ

Всі задачі, поставлені для реалізації WEB-додатку для керування серверним середовищем корпоративної інформаційної системи були виконані в повному обсязі, а саме: проведено аналіз предметної області моніторингу та керування серверної інфраструктури; спроєктовано структуру та компоненти WEB-додатку для керування серверним середовищем корпоративної інформаційної системи; програмно реалізувано WEB-додаток для керування серверним середовищем корпоративної інформаційної системи; виконано тестування програми та проведено аналіз отриманих результатів; розроблено інструкцію користувача.

У ході виконання кваліфікаційної роботи було реалізовано web-додаток для керування серверним середовищем корпоративної інформаційної системи з використанням сучасних технологій та підходів до автоматизації. Серверна частина проєкту побудована з використанням фреймворку Django, що забезпечило стабільність і гнучкість при реалізації логіки взаємодії з інстанціями, контейнерами та зовнішніми сервісами. Для реалізації клієнтського інтерфейсу застосовано сучасні веб-технології на базі HTML, CSS і JavaScript, що дало змогу створити зручне, інтуїтивно зрозуміле середовище для взаємодії користувача з системою. Система ефективно інтегрується з Docker та Docker Compose, що забезпечує можливість контейнеризації та простого розгортання серверних інстанцій без глибоких технічних знань.

Під час розробки створено повноцінну архітектуру програмного модуля, що включає функціональні блоки для ініціалізації нових проєктів, керування інстанціями, виконання резервного копіювання та відновлення, а також перегляду системних метрик. Розроблено алгоритм роботи додатку, спроєктовано компоненти системи з використанням UML-діаграм, а також обґрунтовано вибір усіх технологій, інструментів та середовища розробки. Створений функціонал протестовано на відповідність вимогам, а результати порівняно з іншими хмарними рішеннями — Frappe Cloud, Render та Railway.

Тестування показало стабільну роботу програмного модуля за різних сценаріїв, що підтверджує його практичну придатність до реального корпоративного використання. Таким чином, поставлені цілі та завдання були досягнуті в повному обсязі. Створений web-додаток є ефективним інструментом для автоматизованого керування серверною інфраструктурою, який може бути з легкістю інтегрований у внутрішні IT-системи підприємств для підвищення надійності, швидкодії та зручності в адмініструванні.

Мету роботи – розширення функціональних можливостей додатку для керування серверним середовищем корпоративної інформаційної системи досягнено в повноному обсязі за рахунок впровадження модулів автоматизованого розгортання, системного моніторингу, резервного збереження та відновлення даних, а також реалізації інтерфейсу для керування окремими контейнерами з урахуванням зручності та інтуїтивності взаємодії користувача з додатком.

Звіт про виконану роботу було складено відповідно до вимог і методичних вказівок, що гарантує його структурованість, логічність викладення матеріалу та відповідність академічним стандартам [28].

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Аналіз передумов розробки веб-додатку для керування серверним середовищем корпоративної інформаційної системи/ Ткаченко Д.О., Богач І.В. // Матеріали LIV науково-технічної конференції підрозділів Вінницького національного технічного університету (НТКП ВНТУ-2025) Збірник доповідей [Електронний ресурс], Вінниця : ВНТУ, 2025. URL: https://conferences.vntu.edu.ua/index.php/all-fksa/all-fksa-2025/paper/view/24456/0. (дата звернення 05.05 2025).

2. РОЗРОБКА WEB-ДОДАТКУ ДЛЯ КЕРУВАННЯ СЕРВЕРНИМ СЕРЕДОВИЩЕМ КОРПОРАТИВНОЇ ІНФОРМАЦІЙНОЇ СИСТЕМИ /Д.О.Ткаченко, І.В.Богач// Матеріали міжнародної науково-практичної Інтернет-конференції студентів, аспірантів та молодих науковців «Молодь в науці: дослідження, проблеми, перспективи (МН-2025)». Збірник доповідей [Електронний ресурс], Вінниця: ВНТУ, 2025. Режим доступу: https://conferences.vntu.edu.ua/index.php/mn/mn2025/paper/view/25414

3. Merkel D. Docker: Lightweight Linux Containers for Consistent Development and Deployment [Електронний ресурс] / Dirk Merkel // Linux Journal. – 2014. – V. 2014(239). – Режим доступу: https://www.linuxjournal.com/content/docker-lightweight-linux-containers-consistent-development-and-deployment (дата звернення: 05.05.2025).

4. Kubernetes Documentation [Електронний ресурс] – Режим доступу: https://kubernetes.io/docs/home/ (дата звернення: 06.05.2025). – Назва з екрана.

5. Turnbull J. The Docker Book: Containerization is the new virtualization [Електронний ресурс] / James Turnbull // James Turnbull. – 2014. – Режим доступу: https://dockerbook.com (дата звернення: 07.05.2025).

6. Hochstein L. Ansible: Up and Running: Automating Configuration Management and Deployment the Easy Way [Електронний ресурс] / Lorin Hochstein // O'Reilly Media. – 2017. – 2nd edition. – Режим доступу:

https://www.oreilly.com/library/view/ansible-up-and/9781491979792/ (дата звернення: 07.05.2025).

7. Smart J. Jenkins: The Definitive Guide: Continuous Integration for the Masses [Електронний ресурс] / John Smart // O'Reilly Media. – 2011. – Режим доступу: https://www.oreilly.com/library/view/jenkins-the-definitive/9781449305352/ (дата звернення: 07.05.2024).

8. What is containerization? - AWS [Електронний ресурс] – Режим доступу: https://aws.amazon.com/whatis/containerization/#:~:text=Containerization%20is%20a%20software%20deployment,matched%20your%20machine's%20operating%20system. (дата звернення: 08.05.2025).

9. Frappe Cloud – The Official Cloud Platform for Frappe Apps [Електронний ресурс] – Режим доступу: https://frappecloud.com/ (дата звернення: 08.05.2025).

10. Duffy J. Cloud Native DevOps with Kubernetes: Building, Deploying, and Scaling Modern Applications in the Cloud [Електронний ресурс] / Justin Garrison, Kris Nova // O'Reilly Media. – 2019. – Режим доступу: https://www.oreilly.com/library/view/cloud-native-devops/9781492040767/ (дата звернення: 07.05.2025).

11. Render - The Official Platform [Електронний ресурс] /Render, Inc. – 2025. – Режим доступу: https://render.com/ (дата звернення: 08.05.2025).

12. Railway 2025 [Електронний ресурс] / Railway Corp. - 2025– Режим доступу: https://railway.com/ (дата звернення: 08.05.2025).

13. Understanding virtualization [Електронний ресурс] / Red Hat, Inc. – 2021. – Режим доступу: https://www.redhat.com/en/topics/virtualization (дата звернення: 08.05.2025).

14. Hightower K., Burns B., Beda J. Kubernetes: Up and Running: Dive into the Future of Infrastructure [Електронний ресурс] / Kelsey Hightower, Brendan Burns, Joe Beda // O'Reilly Media. – 2021. – 3rd edition. – Режим доступу: https://www.oreilly.com/library/view/kubernetes-up-and/9781492092308/ (дата звернення: 08.05.2025).

15. Django documentation [ЕлеДктронний ресурс] – Режим доступу: https://docs.djangoproject.com/en/5.0/ (дата звернення: 15.05.2025).

16. The Python Standard Library — Python 3.12.3 documentation [Електронний ресурс] – Режим доступу: https://docs.python.org/3/library/index.html (дата звернення: 08.05.2025).

17. Python on whales [Електронний ресурс] – Режим доступу: https://gabrieldemarmiesse.github.io/python-on-whales/(дата звернення: 15.05.2025).

18. JavaScript – MDN Web Docs – Mozilla [Електронний ресурс] – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/JavaScript (дата звернення: 15.05.2025).

19. Develop with Docker Engine API [Електронний ресурс] – Режим доступу: https://docs.docker.com/engine/api/(дата звернення: 15.05.2025).

20. How to Build an API in Python [Електронний ресурс] – Режим доступу: https://blog.postman.com/how-to-build-an-api-in-python/(дата звернення: 15.05.2025).

21. os — Miscellaneous operating system interfaces [Електронний ресурс] – Режим доступу: https://docs.python.org/3/library/os.html(дата звернення: 15.05.2025).

22. psutil documentation — psutil 6.0.0 documentation [Електронний ресурс] – Режим доступу: https://psutil.readthedocs.io/en/latest/(дата звернення: 15.05.2025).

23. HTML: HyperText Markup Language – MDN Web Docs [Електронний ресурс] – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/HTML(дата звернення: 15.05.2025).

24. CSS: Cascading Style Sheets - MDN Web Docs [Електронний ресурс] – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/CSS(дата звернення: 15.05.2025).

25. frappe/frappe_docker: Docker images for production and development setups of the Frappe framework and ERPNext [Електронний ресурс] – Режим доступу: https://github.com/frappe/frappe_docker(дата звернення: 15.05.2025).

26. Scaling ERPNext on Kubernetes [Електронний ресурс] – Режим доступу: https://frappe.io/blog/scaling-erpnext-on-kubernetes (дата звернення: 03.06.2025) – Назва з екрана.

27. Khondker F. Microservices with Docker, Flask, and React [Електронний ресурс] / Firoz Khondker // Packt Publishing. – 2023. – Режим доступу: https://www.packtpub.com/product/microservices-with-docker-flask-and-react (дата звернення: 03.06.2025).

28. Методичні вказівки до виконання бакалаврських кваліфікаційних робіт для студентів спеціальності 122 «Комп'ютерні науки» [Електронний ресурс] / уклад.: А. А. Яровий, О. В. Сілагін, І. В. Богач. – Вінниця : ВНТУ, 2023. – (PDF, 54 с.) (дата звернення: 03.06.2025).

ДОДАТКИ

Додаток А (обов'язковий)

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Розробка Web - додатку для керування серверним середовищем корпоративної інформаційної системи

Тип роботи: бакалаврська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ кафедра комп'ютерних наук
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 0,47 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

✓ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

Яровий А.А., зав. каф. КН _____
(прізвище, ініціали, посада) (підпис)

Колесницький О.К., проф. каф КН _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Озеранський В.С.
(підпис) (прізвище, ініціали)

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Богач І.В.
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Ткаченко Д.О.
(підпис) (прізвище, ініціали)

Додаток Б (обов’язковий)

Лістинг основних функцій програми

```
def index(request):
    return render(request, "instances/index.html")


def install_instance(request):
    projects_path = 'instances/projects/'
    project_name = 'test-project'
    mariadb_password = '123'
    site_url = 'mysite.localhost'
    admin_password = 'admin'

    if not os.path.isdir(os.path.join(projects_path, 'frappe_docker')):
        print('no frappe_docker')
        os.system(f'git clone https://github.com/frappe/frappe_docker.git
{projects_path}/frappe_docker')

    frappe_docker_path = os.path.join(projects_path, 'frappe_docker')

    if not os.path.isdir(os.path.join(projects_path, 'traefik')):
        traefik_domain = 'traefik.com'
        traefik_email = 'zayats.v@ikrok.net'
        traefik_password = '123'
        traefik_hashed_password = crypt.crypt(traefik_password,
crypt.mksalt(crypt.METHOD_SHA512))

        os.makedir(os.path.join(projects_path, 'traefik'))
        with open(os.path.join(projects_path, 'traefik', 'traefik.env'), 'w') as traefik_env:
```

```
            traefik_env.write(f'TRAEFIK_DOMAIN={traefik_domain}\n')
            traefik_env.write(f'EMAIL={traefik_email}\n')
            traefik_env.write(f'HASHED_PASSWORD={traefik_hashed_password}\n')
        traefik_compose = DockerClient(
            compose_project_name='traefik',
            compose_env_file=os.path.join(projects_path, 'traefik', 'traefik.env'),
            compose_files=[
                os.path.join(frappe_docker_path, 'overrides', 'compose.traefik.yaml'),
                os.path.join(frappe_docker_path, 'overrides', 'compose.traefik-ssl.yaml')
            ]
        )
        traefik_compose.compose.up(detach=True)

    try:
        project_folder = os.path.join(projects_path, project_name)
        os.mkdir(project_folder)

        with open(os.path.join(project_folder, 'mariadb.env'), 'w') as mariadb_env:
            mariadb_env.write(f'DB_PASSWORD={mariadb_password}\n')

        copyfile(os.path.join(frappe_docker_path, 'overrides', 'compose.mariadb-
shared.yaml'), os.path.join(project_folder, 'compose.mariadb-shared.yaml'))

        with open(os.path.join(project_folder, 'compose.mariadb-shared.yaml'), 'r+') as
mariadb_yaml:
            lines = mariadb_yaml.readlines()
            mariadb_yaml.seek(0)

            for line in lines:
```

```
            if 'mariadb-database' in line:
                line = line.replace('mariadb-database', f'{project_name}-mariadb-
database')
            mariadb_yaml.write(line)


    mariadb_compose = DockerClient(
        compose_project_name=f'{project_name}-mariadb',
        compose_env_file=os.path.join(project_folder, 'mariadb.env'),
        compose_files=[os.path.join(project_folder, 'compose.mariadb-shared.yaml')]
    )


    mariadb_compose.compose.up(detach=True)


    copyfile(os.path.join(frappe_docker_path, 'example.env'),
os.path.join(project_folder, f'{project_name}.env'))
    with open(os.path.join(project_folder, f'{project_name}.env'), 'r+') as
project_env:
        lines = project_env.readlines()
        project_env.seek(0)


        for line in lines:
            if 'DB_PASSWORD=123' in line:
                line = line.replace('DB_PASSWORD=123',
f'DB_PASSWORD={mariadb_password}')
            if 'DB_HOST=' in line:
                line = line.replace('DB_HOST=', f'DB_HOST={project_name}-mariadb-
database')
            if 'DB_PORT=' in line:
                line = line.replace('DB_PORT=', f'DB_PORT=3306')
            if 'SITES=erp.example.com' in line:
```

```
                line = line.replace('SITES=erp.example.com', f'SITES={site_url}')

            project_env.write(line)

        project_env.write(f'ROUTER={project_name}\n')
        project_env.write(f'BENCH_NETWORK={project_name}\n')
project_compose = DockerClient(
        compose_project_name=project_name,
        compose_env_file=os.path.join(project_folder, f'{project_name}.env'),
        compose_files=[
            os.path.join(frappe_docker_path, 'compose.yaml'),
            os.path.join(frappe_docker_path, 'overrides', 'compose.redis.yaml'),
            os.path.join(frappe_docker_path, 'overrides', 'compose.multi-bench.yaml'),
            os.path.join(frappe_docker_path, 'overrides', 'compose.multi-bench-
ssl.yaml')
        ]
    )

    project_config = project_compose.compose.config(return_json=True)
    yaml_config = dump(project_config, default_flow_style=False, sort_keys=False)

    with open(os.path.join(project_folder, f'{project_name}.yaml'), 'w') as
project_yaml:
        project_yaml.write(yaml_config)

    project_compose.compose_files = [os.path.join(project_folder,
f'{project_name}.yaml')]
    project_compose.compose.up(detach=True)
    project_compose.compose.execute(
        service='backend',
```

```
            command=['bench', 'new-site', f'{site_url}', '--no-mariadb-socket', '--mariadb-root-password', f'{mariadb_password}', '--install-app', 'erpnext', '--admin-password', f'{admin_password}'],
            #command=['bench', '--help'],
            workdir='/home/frappe/frappe-bench'
        )

    except Exception as e:
        print('InstallError: There was something wrong during installation proccess')
        print(e)

    return HttpResponse()

ZC.LICENSE = ["your-license-key"];

const CPUGauge = {
    type: 'gauge',
    globals: {
        fontSize: 25,
        backgroundColor: '#1A1A1A'
    },
    plot: {
        size: '100%',
        valueBox: {
            placement: 'center',
            text: '%v%',
            fontSize: 35,
            backgroundColor: '#1A1A1A',
            shadowColor: 'none',
```

```
        rules: [{
            rule: '%v >= 75',
            text: '%v%'
        }, {
            rule: '%v < 75 && %v > 25',
            text: '%v%',
        }, {
            rule: '%v <= 25',
            text: '%v%'
        }]
    }
},
plotarea: {
    marginTop: 50
},
scaleR: {
    borderColor: '#1A1A1A',
    lineColor: '#1A1A1A',
    aperture: 200,
    minValue: 0,
    maxValue: 100,
    step: 10,
    center: {
        visible: false
    },
    tick: {
        visible: false
    },
    item: {
```

```
        offsetR: 0,
        fontColor: 'white',
        rules: [{
            rule: '%i == 9',
            offsetX: 15,
        }]
    },
    labels: ['0', '', '', '', '', '50', '', '', '', '', '100'],
    ring: {
        size: 50,
        rules: [{
            rule: '%v <= 25',
            backgroundColor: '#7CBD3B'
        }, {
            rule: '%v > 25 && %v < 75',
            backgroundColor: '#E8C75B'
        }, {
            rule: '%v >= 70',
            backgroundColor: '#CE4848'
        }]
    }
},
series: [{
    values: [0],
    backgroundColor: 'white',
    borderColor: '#1A1A1A',
    lineColor: '#1A1A1A',
    indicator: [10, 10, 10, 10, 0.75],
    animation: {
```

```
            effect: 2,
            method: 1,
            sequence: 4,
            speed: 900
        }
    }]
};

zingchart.render({
    id: 'gauge-cpu',
    data: CPUGauge,
    height: 500,
    width: '100%'
});

function updateGauge() {
    fetch('/api/get_cpu_load/')
        .then(response => response.json())
        .then(data => {
            zingchart.exec('gauge-cpu', 'setseriesvalues', {
                values: [[data.cpu_load]]
            });
        });
}

setInterval(updateGauge, 60000);

updateGauge();
```

```
function updateTableData() {
    fetch('/api/get_system_info')
        .then(response => response.json())
        .then(data => {
            const table = document.getElementById('info-table');

            updateTable(table, 'CPU', data.cpu);
            updateTable(table, 'CPU Cores', data.cpu_cores);
            updateTable(table, 'RAM', data.ram);
            updateTable(table, 'OS', data.os);
        })
        .catch(error => console.error('Error fetching system info:', error));
}

function updateTable(table, label, value) {
    const rows = table.getElementsByTagName('tr');
    for (let row of rows) {
        if (row.cells[0].textContent === label) {
            row.cells[1].textContent = value;
            break;
        }
    }
}

function fetchMemoryInfo() {
    fetch('/api/get_memory_info')
        .then(response => response.json())
        .then(data => {
            const ramFill = document.getElementById('ram-fill');
```

```
        const ramPercentage = data.ram_remnant;
        ramFill.style.width = ${ramPercentage}%;
        document.getElementById('ram-text').innerHTML = <h2>RAM (${ramPercentage}%)</h2>;

        const memFill = document.getElementById('mem-fill');
        const memPercentage = data.drive_remnant;
        memFill.style.width = ${memPercentage}%;
        document.getElementById('mem-text').innerHTML = <h2>MEMORY (${memPercentage}%)</h2>;
    })
    .catch(error => console.error('Error fetching memory info:', error));
}
```

Додаток В (обов'язковий)

ІЛЮСТРАТИВНА ЧАСТИНА

« РОЗРОБКА WEB – ДОДАТКУ ДЛЯ КЕРУВАННЯ СЕРВЕРНИМ СЕРЕДОВИЩЕМ КОРПОРАТИВНОЇ ІНФОРМАЦІЙНОЇ СИСТЕМИ»

Виконала: студентка 4 курсу,групи 2КН-21б
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Ткаченко Д.О.
(прізвище та ініціали)

Керівник: к.т.н., доцент
Богач І.В.
(прізвище та ініціали)
«03» червня 2025 р.

Вінниця ВНТУ - 2025 рік

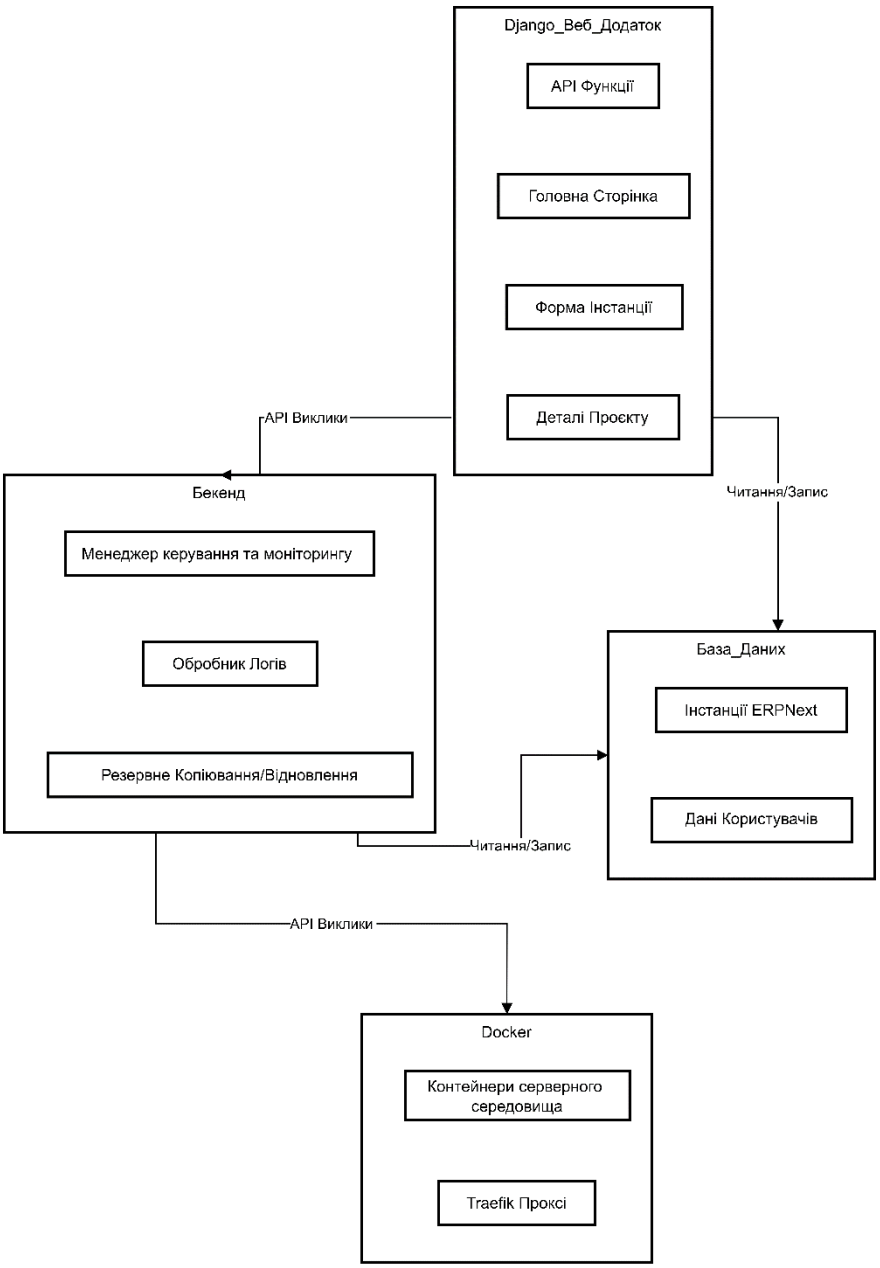

Рисунок В.1 – UML-діаграма компонентів WEB-додатку для керування серверним середовищем корпоративної інформаційної системи

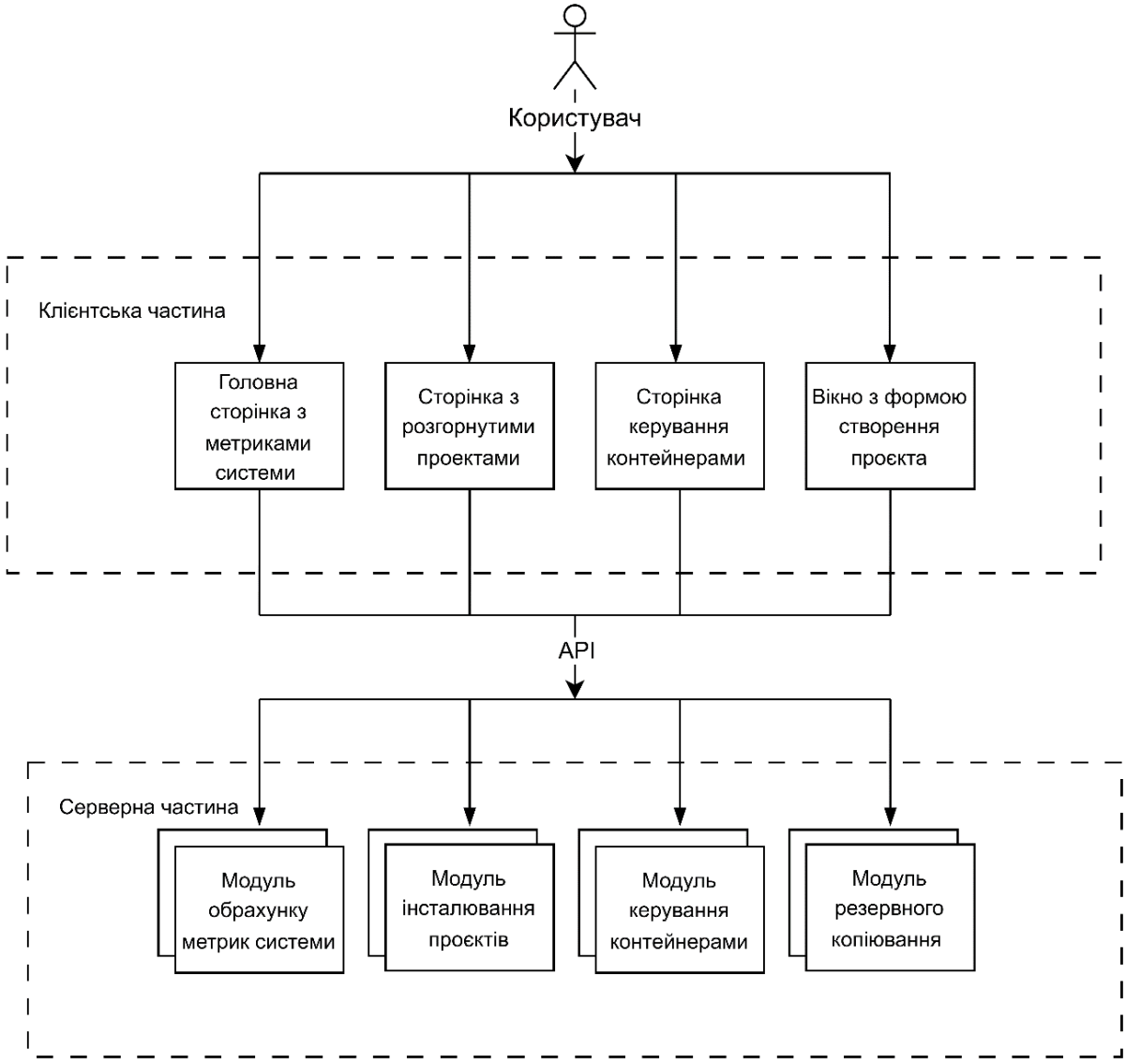

Рисунок В.2 - Загальна структурна схема компонентів WEB-додатку для керування серверним середовищем корпоративної інформаційної системи

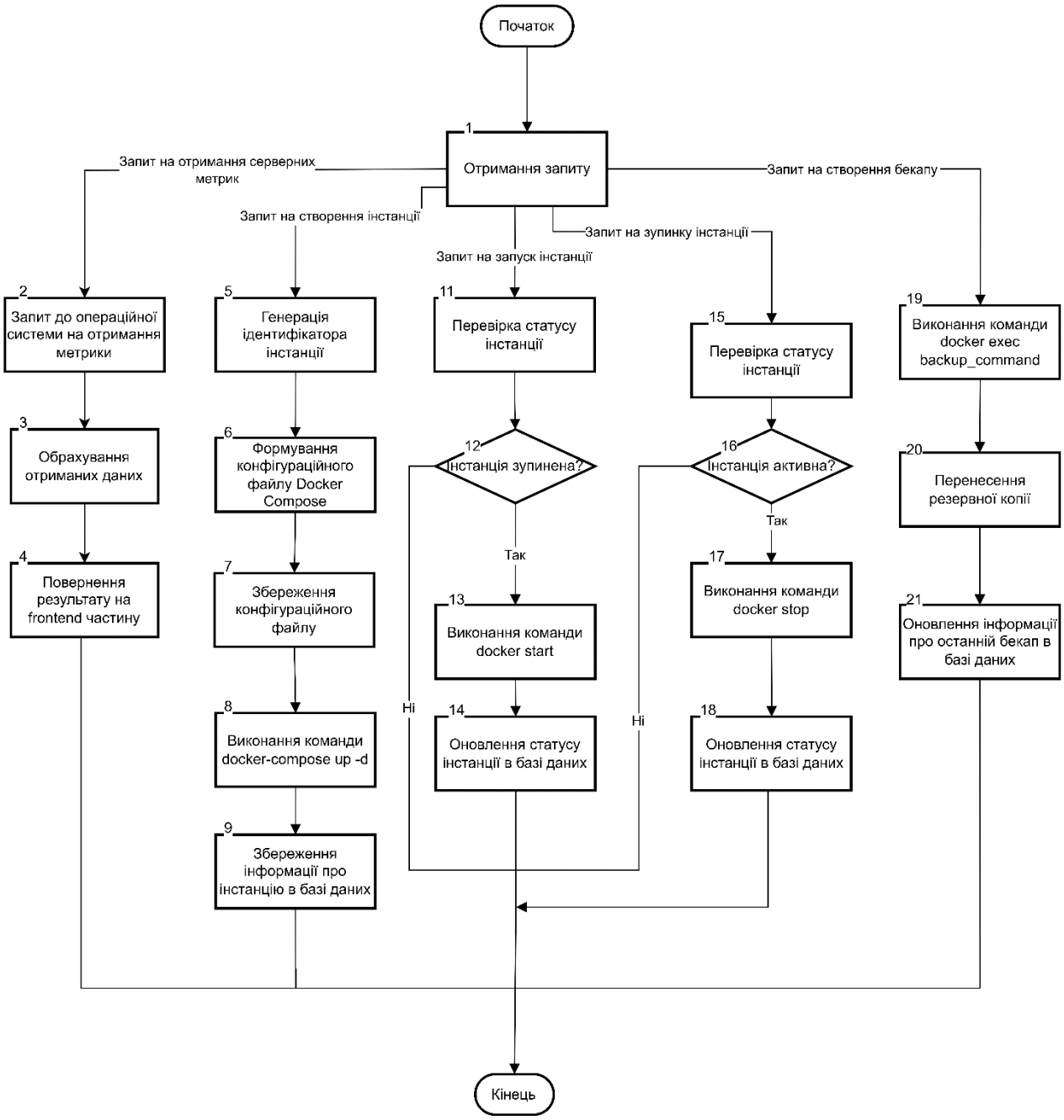

Рисунок В.3 – Алгоритм роботи WEB-додатку для керування серверним середовищем корпоративної інформаційної системи

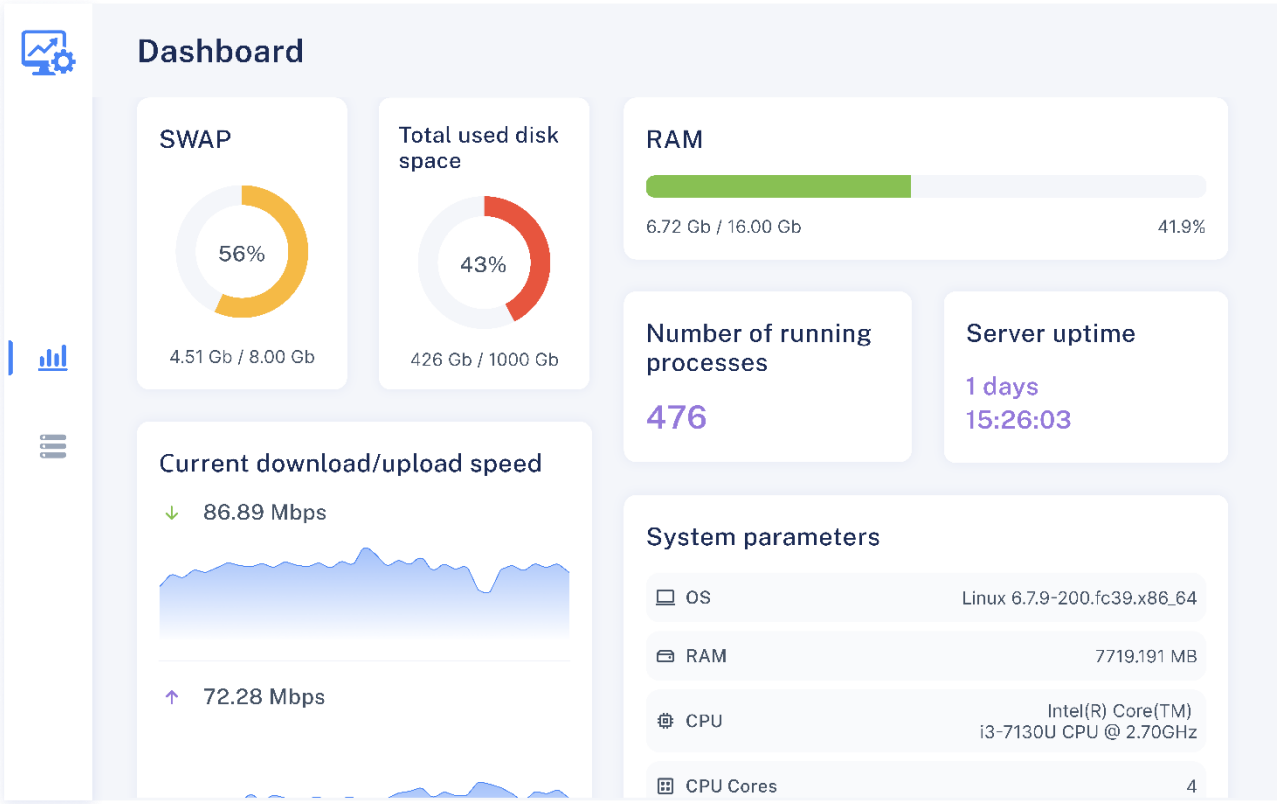

Рисунок В.4 – Стартова сторінка з метриками

Додаток Г (довідниковий)

ІНСТРУКЦІЯ КОРИСТУВАЧА

Щоб розпочати роботу з реалізованим веб-додатком для керування серверним середовищем, необхідно попередньо встановити мову програмування Python та фреймворк Django разом з усіма необхідними бібліотеками, що використовуються в проєкті. Після цього потрібно розпакувати архів із проєктом та виконати запуск Django-сервера. Після успішного старту користувач побачить головну сторінку веб-інтерфейсу, з якої починається взаємодія з додатком.

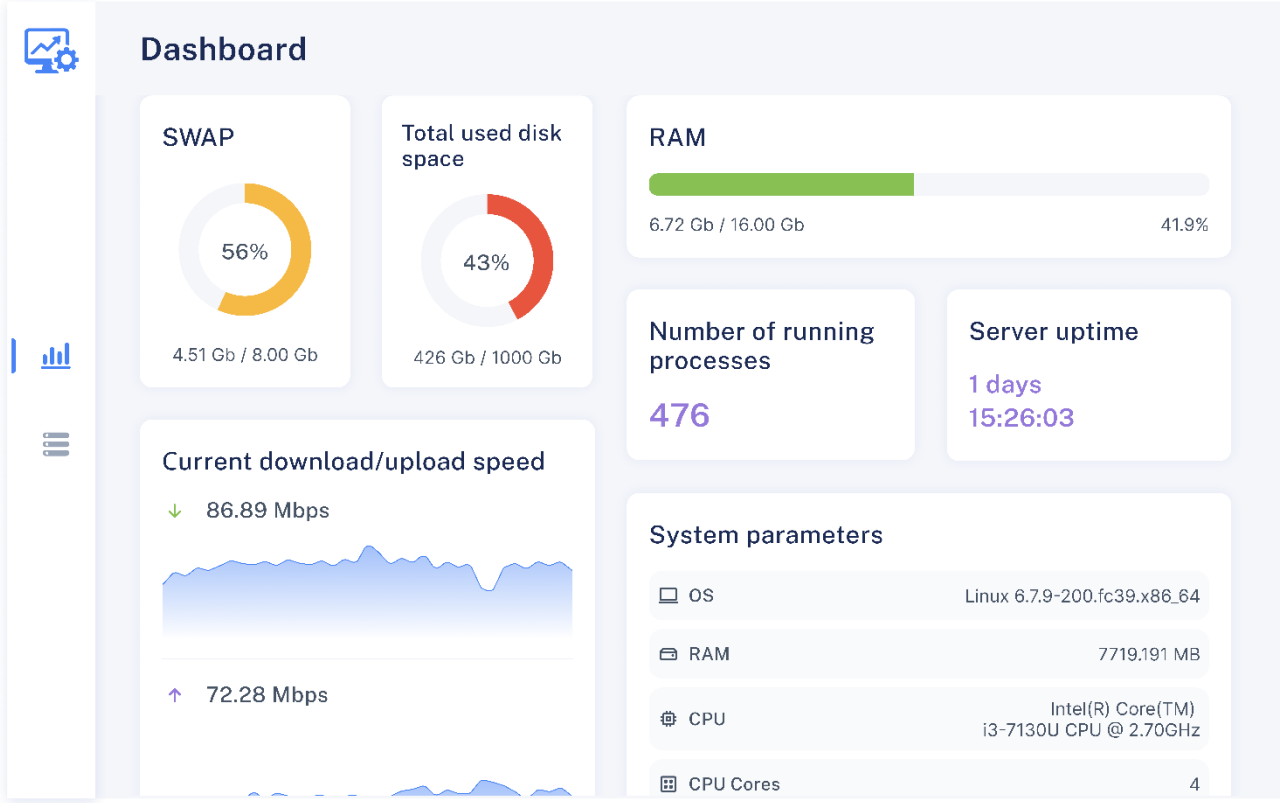

Рисунок Г.1 – Стартова сторінка з метриками

Щоб отримати доступ до проєктів (контейннри, панель керування) на пічній панелі перемикаємося на іншу сторінку за допомогою іконки. Після натиснення відкриється сторінка з проєктами

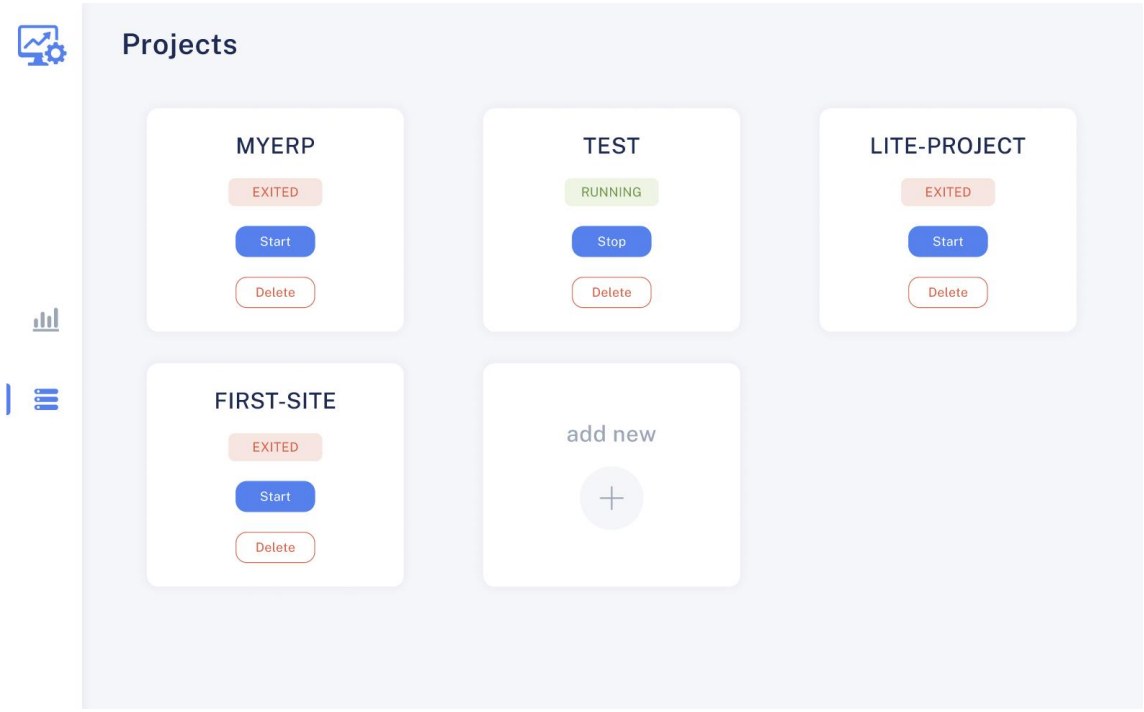

Рисунок Г.2 – Сторінка з розгорнутими проєктами

Для створення нового проєкту необхідно натиснути на елемент «+», розміщений на стартовій сторінці веб-додатку. Після цього відкриється форма, в якій користувач має вказати необхідні дані для ініціалізації нового серверного середовища. Якщо до цього моменту компонент Traefik ще не був налаштований, система автоматично перенаправить користувача на сторінку для конфігурації Traefik, що є обов'язковим етапом перед запуском основного проєкту.

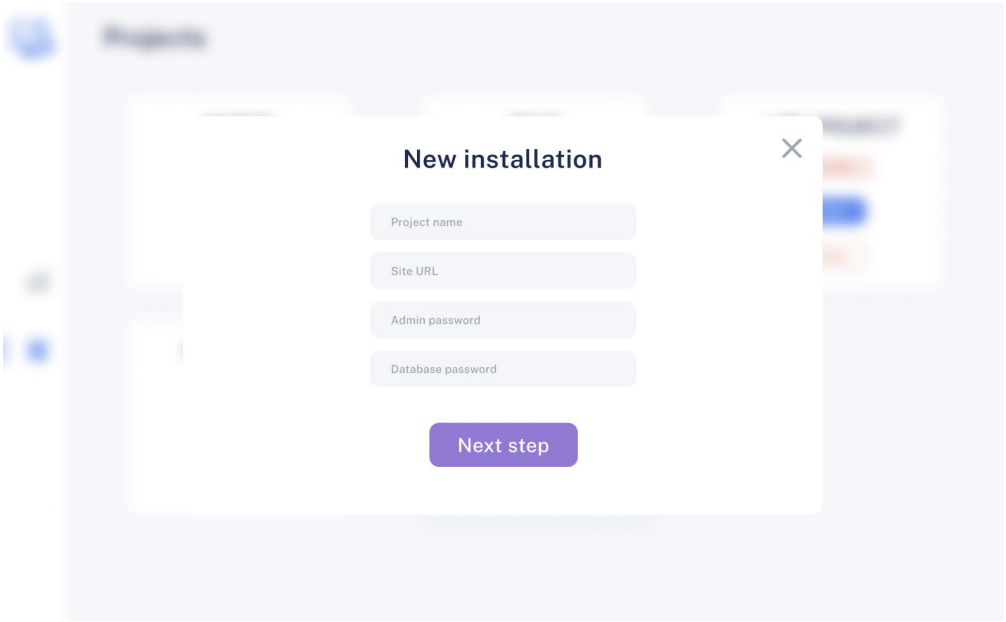

Рисунок Г.3 - Форма додавання нового екземпляру для розгортання

Для розгортання Traefik користувач потрапляє на спеціальну форму, яка містить поля для введення необхідних параметрів проксі-сервера.

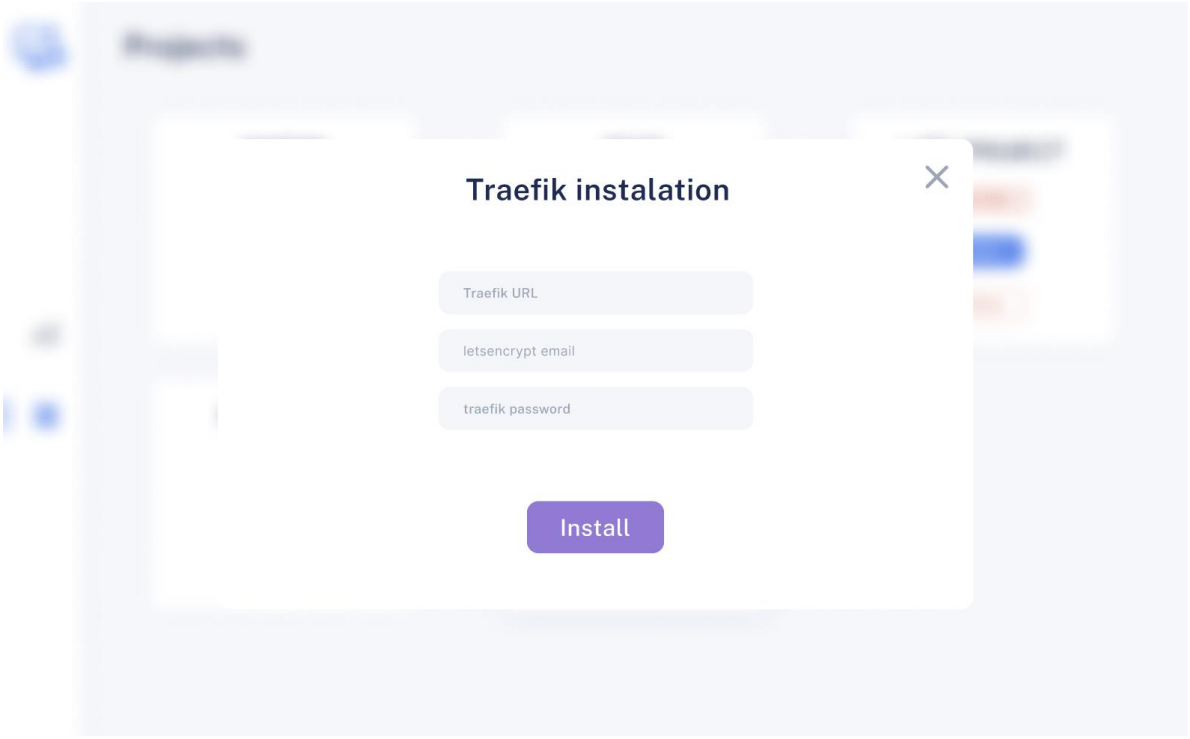

Рисунок Г.4 - Форма конфігурації Traefik

Для того щоб переглянути деталі контретного проєкту, користувачу необхідно натиснути на конкретний проєкт на сторінці з проєктами, після натиснення відкриється стрінка де буде відображено контейнери, їх статуси, дата створення, а також кнопки керування.

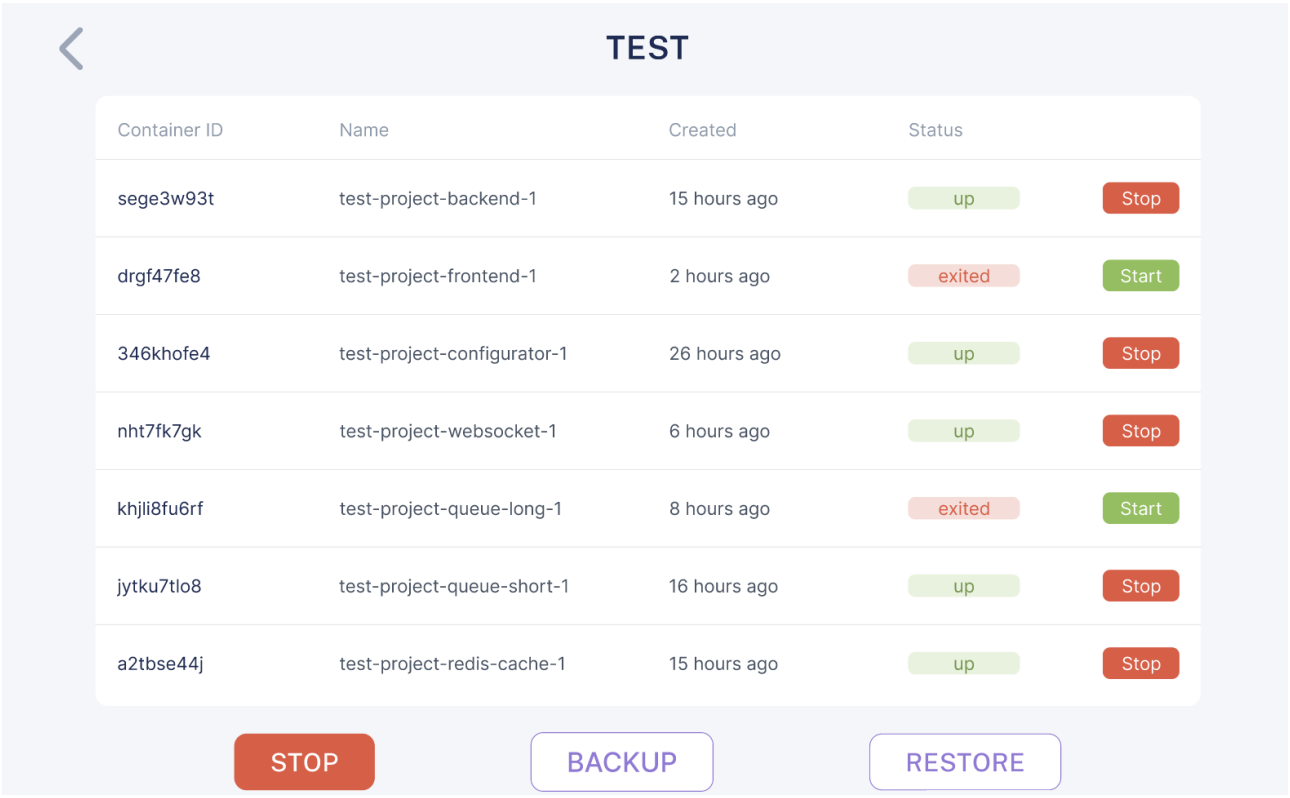

Рисунок Г.5 – Відображення деталей проєкту

Додаток Д

ДОВІДКА ПРО ВПРОВАДЖЕННЯ

21036, Україна; м. Вінниця;
Хмельницьке шосе, 82
+380957922117
olha@slife.guru

За місцем вимоги

Довідка

Видана Ткаченко Дар'ї Олексіївні в тому, що результати, одержані нею у процесі виконання бакалаврської кваліфікаційної роботи на тему «Розробка WEB-додатку для керування серверним середовищем корпоративної інформаційної системи», а саме розроблені алгоритми, інтерфейсні рішення та програмна реалізація, планується використати в розробках ТОВ «ЕСЛАЙФ».

Директор (підпис) Орлова О. О.
(прізвище та ініціали)