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

Бакалаврська дипломна робота на тему:

ПРОГРАМНИЙ МОДУЛЬ РОЗГОРТАННЯ СЕРВЕРНОЇ ІНФРАСТРУКТУРИ ДЛЯ ДОДАТКУ ПЛАНУВАННЯ РЕСУРСІВ ПІДПРИЄМСТВА

Виконав: студент 4 курсу, групи 2КН-20б

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

Заяц В.В.
(прізвище та ініціали)

Керівник: к.т.н., доцент каф. КН
Сілагін О.В.
(прізвище та ініціали)

«07» 06 2024 р.

Рецензент:
(прізвище та ініціали)

«07» 06 2024 р.

Допущено до захисту
Зав. кафедри Яровий А. А.
«07» 06 2024 р.

Вінниця ВНТУ – 2024 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.
«04» 03 2024р.

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Зайцю Владиславу Валерійовичу

1. Тема роботи: Програмний модуль розгортання серверної інфраструктури для додатку планування ресурсів підприємства
Керівник роботи: Сілагін Олексій Віталійович, к.т.н., доц. каф. КН.
Затверджені наказом вищого навчального закладу "11" 03 2024 року №80
2. Термін подання студентом роботи 27 05 2024 р.
3. Вихідні дані до роботи : Веб-додаток для розгортання серверної інфраструктури; API для взаємодії з системними характеристиками; Інтерфейс взаємодії веб-додатку та Docker; Розгортання не більше 6 хв; Функції управління серверною інфраструктурою.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): Обґрунтування доцільності створення програмного модулю розгортання серверної інфраструктури для додатку планування ресурсів підприємства; Аналіз сучасних технологій для розгортання серверної інфраструктури; Проектування структури та компонентів програмного модулю; Програмна реалізація модулю розгортання інфраструктури для додатку планування ресурсів підприємства; Тестування програмного модулю розгортання серверної інфраструктури для додатку планування ресурсів підприємства.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень):
Схема алгоритму роботи програмного модулю; UML-діаграма компонентів програмного модулю; Системні характеристики середовища; Список всіх Docker Compose проєктів; Форма додавання нового екземпляру для розгортання; Форма інсталяції проксі-сервера Traefik; Список Docker контейнерів, які належать певному екземпляру; Панель керування та логи екземпляру.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завда прий
1-3	Сілагін О.В., к.т.н., доц. каф. КН		

7. Дата видачі завдання 07 03 2024

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)		При
		Початок	Кінець	
1	Обгрунтування доцільності створення програмного модулю розгортання серверної інфраструктури для додатку планування ресурсів підприємства	06.05.24	08.05.24	
2	Аналіз предметної області розгортання серверної інфраструктури	09.05.24	13.05.24	
3	Проектування структури та компонентів програмного модулю	14.05.24	18.05.24	
4	Програмна реалізація модулю розгортання інфраструктури для додатку планування ресурсів підприємства	19.05.24	24.05.24	
5	Тестування програмного модулю розгортання серверної інфраструктури для додатку планування ресурсів підприємства.	25.05.24	30.05.24	
6	Оформлення пояснювальної записки	31.05.24	06.06.24	

Студент _____ **Заяц В.В.**
(підпис) (прізвище та ініціали)

Керівник БДР _____ **Сілагін О.В.**
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Бакалаврська дипломна робота складається з 83 сторінки формату А4, на яких є 25 рисунків, 3 таблиці список використаних джерел містить 22 найменування.

Бакалаврська дипломна робота присвячена розробці програмного модуля для розгортання серверної інфраструктури додатку планування ресурсів підприємства. У цій бакалаврській роботі розроблено модуль, який забезпечує автоматизоване розгортання, налаштування та моніторинг серверної інфраструктури ERPNext з використанням сучасних технологій контейнеризації та автоматизації, таких як Docker, Docker Compose, Python, Django.

Ключові слова: програмний модуль, автоматизація розгортання, серверна інфраструктура, Docker, додаток планування ресурсів підприємства, Python.

ABSTRACT

The bachelor's thesis consists of 83 A4 pages, containing 25 images, 3 tables, and a list of references comprising 22 sources.

The bachelor's thesis is dedicated to the development of a software module for deploying the server infrastructure of an enterprise resource planning application. In this bachelor's thesis, a module is developed that provides automated deployment, configuration, and monitoring of the ERPNext server infrastructure using modern containerization and automation technologies such as Docker, Docker Compose, Python, and Django.

Keywords: software module, deployment automation, server infrastructure, Docker, enterprise resource planning application, Python.

ЗМІСТ

ВСТУП

У сучасному світі системи планування ресурсів підприємства (ERP) є невід'ємною частиною ефективного управління бізнес-процесами. ERP-системи дозволяють підприємствам автоматизувати багато аспектів їхньої діяльності, включаючи фінансове управління, управління людськими ресурсами, виробництво, постачання та інші. Проте, налаштування та обслуговування ERP-систем є складними та ресурсоємними завданнями, що вимагають значних витрат часу та людських ресурсів.

Сучасні технології дозволяють значно спростити ці процеси за допомогою автоматизації. Автоматизація розгортання серверної інфраструктури для ERP-систем забезпечує зниження трудомісткості та підвищення надійності роботи IT-інфраструктури. Вона дозволяє уникнути помилок, пов'язаних з людським фактором, та забезпечити стабільну роботу системи навіть у випадку збоїв. Однією з найбільш ефективних технологій для автоматизації є контейнеризація, зокрема використання Docker та Docker Compose. Ці інструменти дозволяють створювати ізольовані середовища для додатків, що забезпечує їх стабільну роботу незалежно від середовища виконання. Docker дозволяє швидко розгортати додатки, переносити їх між різними середовищами та масштабувати за потреби. Docker Compose, в свою чергу, спрощує управління конфігурацією багатоконтейнерних додатків.

Все це свідчить про актуальність дослідження та розробки програмного модуля для автоматизованого розгортання серверної інфраструктури ERP-системи. Такий модуль значно підвищить ефективність управління ресурсами підприємства, забезпечуючи швидке та надійне розгортання, легкість у підтримці та моніторингу, а також високу гнучкість та масштабованість системи.

Метою дослідження є пришвидшення процесу розгортання серверної інфраструктури для додатку планування ресурсів підприємства за рахунок автоматизації.

Об’єкт дослідження – процес розгортання та управління серверною інфраструктурою для ERP-систем.

Предмет дослідження – програмні засоби для автоматизації розгортання та управління серверною інфраструктурою ERP-системи ERPNext.

Задачі дослідження:

1. Аналіз та порівняння сучасних технологій для розгортання серверної інфраструктури.
2. Проектування структури та компонентів програмного модулю
3. Програмна реалізація модулю розгортання інфраструктури для додатку планування ресурсів підприємства

Результати, одержані в ході виконання бакалаврської дипломної роботи, а саме, конфігураційні файли, алгоритми та скрипти розгортання, плануються до впровадження в розробку ТОВ «ІКРОК», про що є відповідна довідка в Додатку Д

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ РОЗГОРТАННЯ СЕРВЕРНОЇ ІНФРАСТРУКТУРИ

1.1 Обґрунтування доцільності створення програмного модуля для розгортання серверної інфраструктури

У сучасному світі розгортання та управління серверною інфраструктурою є одним із ключових аспектів ефективного управління IT-ресурсами. Існує кілька популярних технологій, які використовуються для автоматизації та оркестрації серверних інфраструктур, таких як Docker, Kubernetes, Ansible та Jenkins. Кожна з цих технологій має свої унікальні особливості, переваги та обмеження. У цьому розділі ми детально розглянемо кожну з них з технічної точки зору, а також порівняємо їх функціональність і можливості.

Docker є однією з найбільш популярних і широко використовуваних технологій для контейнеризації додатків [1]. Він дозволяє розробникам створювати, розгортати та керувати додатками у вигляді контейнерів, які є легкими і портативними середовищами для виконання програмного забезпечення. Використовуючи Docker, ви можете створювати ізольовані середовища для своїх додатків, що дозволяє уникнути конфліктів між залежностями та забезпечити стабільну роботу додатків у будь-якому середовищі.

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 [1]. 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 [1].

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 [3].

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 [1]. 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 [1].

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 [3].

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 [3]. 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 [3]. 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 [3]. 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 проектів, але для великих розподілених систем може знадобитися використання додаткових інструментів оркестрації, таких як Kubernetes [2]. 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.

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). Kubernetes автоматизує розгортання, масштабування та управління контейнеризованими додатками, дозволяючи з легкістю керувати великими кластерами контейнерів [2]. Kubernetes складається з кількох основних компонентів.

Основним елементом є Kubernetes Cluster, який складається з одного або більше майстер-вузлів (master nodes) та робочих вузлів (worker nodes) [2]. Майстер-вузли керують роботою кластера, виконують розподіл завдань і керують станом кластеру, тоді як робочі вузли виконують додатки та забезпечують їх працездатність. Одним з основних елементів в Kubernetes є Pods – найменша і найпростіша одиниця в Kubernetes, яка може містити один або кілька контейнерів, що поділяють одне і те ж середовище [8]. Pods надають механізм ізоляції для контейнерів і дозволяють їм взаємодіяти один з одним через спільну мережу та файлову систему.

Ще одним важливим компонентом є Services. Services в Kubernetes забезпечують стабільні IP-адреси та DNS-імена для груп Pods. Вони

дозволяють додаткам, що працюють у різних Pods, взаємодіяти один з одним, забезпечуючи стабільний доступ навіть у разі перезавантаження або переміщення Pods між вузлами. Deployments є іншим ключовим компонентом Kubernetes, який відповідає за масштабування та оновлення Pods. Deployments дозволяють легко розгортати нові версії додатків без переривання роботи, автоматично замінюючи старі Pods новими. Це забезпечує безперервну роботу додатків під час оновлення та дозволяє швидко реагувати на зміни вимог [2]. Крім того, Kubernetes надає інструменти для управління конфігураціями та секретними даними. ConfigMaps та Secrets дозволяють зберігати та керувати конфігураційними даними та секретами, такими як паролі або ключі API. Вони забезпечують безпечне зберігання та передачу конфіденційних даних, що є критично важливим для забезпечення безпеки додатків.

Kubernetes має багато переваг, які роблять його одним з найпотужніших інструментів для оркестрації контейнерів. По-перше, Kubernetes забезпечує автоматизацію управління контейнерами. Це означає, що багато аспектів управління контейнерами, включаючи розгортання, масштабування, оновлення та відновлення, автоматизовані. Це значно знижує складність управління великими кластерами контейнерів і забезпечує стабільну роботу додатків [9].

По-друге, Kubernetes забезпечує високу доступність. Kubernetes автоматично переміщує контейнери між вузлами у разі збоїв, забезпечуючи безперервну роботу додатків. Це гарантує, що ваші додатки залишатимуться доступними навіть у разі апаратних або програмних збоїв. По-третє, Kubernetes забезпечує масштабованість. Kubernetes легко масштабується для підтримки великих розподілених систем, дозволяючи керувати тисячами контейнерів. Це особливо важливо для великих підприємств та організацій, які потребують гнучкості та можливості швидко адаптуватися до змінних вимог.

Також Kubernetes надає розширюваність завдяки підтримці плагінів та інших розширень. Це дозволяє інтегрувати різні інструменти та технології, що забезпечує створення комплексних рішень для управління інфраструктурою [2].

Незважаючи на численні переваги, Kubernetes має деякі недоліки. Один з основних недоліків – це складність налаштування і управління. Kubernetes є потужною системою з багатьма функціями, але це також означає, що його налаштування може бути складним і вимагати значних знань та досвіду. Налагодження Kubernetes-кластеру, забезпечення його безпеки та управління можуть вимагати значних зусиль і ресурсів. Ще одним недоліком є високі вимоги до ресурсів [2]. Kubernetes потребує значних обчислювальних ресурсів для забезпечення своєї роботи, що може бути проблемою для невеликих компаній або проектів з обмеженими ресурсами. Керування великими кластерами Kubernetes також може вимагати значних витрат на інфраструктуру.

Також варто зазначити, що Kubernetes може бути надлишковим для невеликих проектів. Для невеликих додатків або проектів, що не потребують масштабованості та високої доступності, Kubernetes може бути занадто складним і громіздким рішенням. У таких випадках простіші інструменти, такі як Docker Compose, можуть бути більш доцільними.

Ansible є інструментом автоматизації IT-процесів з відкритим вихідним кодом, який використовується для управління конфігураціями, розгортання додатків та автоматизації повторюваних завдань [4]. Він дозволяє автоматизувати складні IT-процеси з мінімальними витратами на налаштування.

Основними компонентами Ansible є Playbooks, Modules та Inventory. Ansible Playbooks – це сценарії автоматизації, написані на YAML, які

визначають завдання для виконання. Playbooks дозволяють описувати конфігурації системи, інфраструктури та розгортання додатків. Вони надають гнучкість і можливість деталізувати процеси автоматизації, що робить їх потужним інструментом для керування IT-інфраструктурою [4].

Ansible Modules – це окремі блоки коду, які виконують конкретні завдання, такі як управління пакетами, користувачами, службами та іншими компонентами системи. Модулі забезпечують широкий спектр функціональності, що дозволяє автоматизувати різні аспекти управління IT-інфраструктурою. Вони можуть бути легко розширені та налаштовані для задоволення специфічних вимог [4]. Inventory – це файл, який містить список вузлів (серверів), якими керує Ansible. Інвентаризація дозволяє групувати вузли за певними критеріями, що спрощує управління та забезпечує гнучкість у налаштуванні завдань автоматизації. За допомогою Inventory ви можете визначити, на яких вузлах виконуються певні завдання, що забезпечує точність і контроль над процесами автоматизації.

Ansible має кілька ключових переваг, які роблять його потужним інструментом для автоматизації IT-процесів. По-перше, Ansible забезпечує простоту використання. Ansible не потребує встановлення спеціальних агентів на керованих вузлах, що спрощує його інтеграцію та використання. Це дозволяє швидко розпочати автоматизацію без додаткових витрат на налаштування. По-друге, Ansible забезпечує гнучкість. Ansible підтримує широкий спектр модулів для різних завдань, що дозволяє автоматизувати різноманітні IT-процеси. Ви можете використовувати Ansible для автоматизації всього, від простих завдань конфігурації до складних процесів розгортання додатків. По-третє, Ansible забезпечує інтеграцію. Ansible легко інтегрується з іншими інструментами та системами, що забезпечує високу

гнучкість у використанні. Це дозволяє створювати комплексні рішення для автоматизації, які включають різні інструменти та технології [4].

Незважаючи на численні переваги, Ansible має деякі недоліки. По-перше, Ansible може бути менш ефективним для великих інфраструктур. Хоча Ansible добре працює для невеликих і середніх проектів, для великих розподілених систем його продуктивність може бути обмеженою. Це пов'язано з тим, що Ansible виконує завдання послідовно, що може призводити до затримок у великих середовищах. Ansible має обмежену підтримку стану. Ansible не зберігає стан своїх операцій, що означає, що він не може автоматично визначати, які завдання вже виконані. Це може призводити до надлишкових операцій, якщо ви не налаштуєте свої Playbooks належним чином [4]. Також Ansible може вимагати значних зусиль на підтримку. Хоча Ansible є потужним інструментом, його налаштування та підтримка можуть вимагати значних зусиль, особливо для складних середовищ. Це може включати написання та підтримку складних Playbooks та модулів, а також управління великими інвентаризаційними файлами.

Jenkins є сервером автоматизації з відкритим вихідним кодом, який використовується для автоматизації процесів побудови, тестування та розгортання програмного забезпечення [5]. Jenkins підтримує інтеграцію з численними плагінами, що дозволяє розширювати його функціональність. Jenkins є сервером автоматизації з відкритим вихідним кодом, який використовується для автоматизації процесів побудови, тестування та розгортання програмного забезпечення. Jenkins підтримує інтеграцію з численними плагінами, що дозволяє розширювати його функціональність [5].

Основними компонентами Jenkins є Jenkins Master, Jenkins Agents та Jenkins Pipelines. Jenkins Master – це основний сервер, який управляє завданнями автоматизації та розподіляє їх на агентів. Майстер відповідає за

планування завдань, управління збірками та моніторинг. Він також забезпечує користувацький інтерфейс, де можна налаштовувати завдання, переглядати журнали та звіти. Jenkins Agents – це вузли, які виконують завдання, делеговані майстром. Агенти можуть бути налаштовані на різних платформах, що забезпечує масштабованість і гнучкість системи. Вони виконують завдання збірки, тестування та розгортання, розвантажуючи майстер-сервер і забезпечуючи паралельне виконання завдань [5].

Jenkins Pipelines – це опис процесів автоматизації за допомогою DSL (Domain Specific Language) або графічного інтерфейсу. Pipelines дозволяють створювати складні процеси CI/CD (безперервна інтеграція та розгортання), що автоматизують всі етапи розробки програмного забезпечення. Pipelines забезпечують гнучкість і масштабованість, дозволяючи налаштовувати процеси, що відповідають специфічним потребам проекту [5].

Однією з основних платформ для розгортання ERPNext є Frappe Cloud. Frappe Cloud є хмарним сервісом, розробленим компанією Frappe Technologies, який дозволяє користувачам швидко і без зайвих зусиль розгортати, управляти та масштабувати інстанції ERPNext [11]. Ця платформа пропонує широкий спектр можливостей та інтеграцій, що робить її привабливою для підприємств, які прагнуть використовувати ERPNext без необхідності встановлення та налаштування власної серверної інфраструктури.

Основною перевагою Frappe Cloud є її простота у використанні. Користувачі можуть розгорнути інстанцію ERPNext буквально за кілька хвилин, використовуючи інтуїтивно зрозумілий інтерфейс [11]. Це значно спрощує процес для тих, хто не має глибоких технічних знань або досвіду у роботі з серверними системами та контейнерами. Крім того, Frappe Cloud

забезпечує автоматичне оновлення системи, що гарантує користувачам доступ до найновіших функцій та безпекових оновлень без додаткових зусиль.

Ще однією перевагою Frappe Cloud є вбудовані механізми резервного копіювання та відновлення даних [11]. Користувачі можуть бути впевнені, що їхні дані надійно зберігаються та можуть бути легко відновлені у разі потреби. Це значно підвищує безпеку і надійність роботи з системою, що є критично важливим для підприємств, які обробляють велику кількість чутливої інформації.

Однак, Frappe Cloud має і свої недоліки. Один із основних недоліків полягає у відносно високій вартості послуг. Хоча початкові тарифи можуть здатися доступними, витрати можуть значно зрости з підвищенням обсягу використання ресурсів або додаванням нових інстанцій. Це може бути стримуючим фактором для малих і середніх підприємств, які прагнуть мінімізувати свої витрати. Крім того, Frappe Cloud обмежує користувачів високою вимогливість до ресурсів сервера та довгим часом розгортання. Хоча платформа пропонує широкий набір інтеграцій і налаштувань, користувачі все одно залежать від інфраструктури і політик Frappe Cloud. Це означає, що деякі специфічні вимоги або налаштування можуть бути складними або навіть неможливими для реалізації на цій платформі.

Таким чином основний аналог для порівняння обрано Frappe Cloud, оскільки через обмежену швидкість розгортання це може повпливати на якість бізнес рішень.

1.2 Аналіз існуючого програмного забезпечення для розгортання серверної інфраструктури

Docker та Kubernetes часто розглядаються як взаємодоповнюючі технології, хоча вони мають різні цілі [9]. Docker фокусується на контейнеризації додатків, що дозволяє пакувати додаток разом з усіма його залежностями в один контейнер. Ці контейнери легкі, портативні та можуть бути легко переміщені між різними середовищами, що забезпечує консистентність роботи додатків. Docker також включає Docker Compose, який дозволяє легко керувати багатоконтейнерними додатками, використовуючи простий формат YAML для опису конфігурації всіх служб додатка.

З іншого боку, Kubernetes є більш потужним інструментом, орієнтованим на оркестрацію контейнерів. Він автоматизує розгортання, масштабування та управління контейнеризованими додатками, забезпечуючи високу доступність та стійкість до збоїв [8]. Kubernetes може керувати тисячами контейнерів у великих розподілених системах, автоматично переміщуючи контейнери між вузлами для забезпечення стабільної роботи додатків навіть у разі збоїв. Однак Kubernetes має складнішу архітектуру і потребує більше зусиль на налаштування та управління. Це може бути викликом для невеликих команд або проектів з обмеженими ресурсами. У таких випадках Docker та Docker Compose можуть бути більш доцільними через їхню простоту і легкість у використанні.

Ansible відрізняється від Docker та Kubernetes тим, що зосереджується на автоматизації IT-процесів, таких як управління конфігураціями та розгортання додатків. Ansible використовує простий формат YAML для написання сценаріїв автоматизації, відомих як Playbooks. Він не потребує встановлення спеціальних агентів на керованих вузлах, що спрощує його

інтеграцію та використання [7]. Ansible дозволяє автоматизувати складні IT-процеси з мінімальними витратами на налаштування, забезпечуючи гнучкість і можливість деталізувати процеси автоматизації.

Однією з головних переваг Ansible є його простота. Однак, для великих розподілених систем продуктивність Ansible може бути обмеженою, оскільки він виконує завдання послідовно, що може призводити до затримок. Ansible також має обмежену підтримку стану, що означає, що він не може автоматично визначати, які завдання вже виконані, і може виконувати надлишкові операції, якщо сценарії автоматизації не налаштовані належним чином.

Jenkins є сервером автоматизації, який використовується для автоматизації процесів побудови, тестування та розгортання програмного забезпечення. Jenkins підтримує інтеграцію з численними плагінами, що дозволяє розширювати його функціональність та адаптувати до будь-яких потреб. Основними компонентами Jenkins є Jenkins Master, який управляє завданнями автоматизації, та Jenkins Agents, які виконують завдання збірки, тестування та розгортання. Jenkins Pipelines дозволяють створювати складні процеси CI/CD (безперервна інтеграція та розгортання), що автоматизують всі етапи розробки програмного забезпечення [5].

Однією з ключових переваг Jenkins є його гнучкість і розширюваність. Завдяки підтримці тисяч плагінів, Jenkins можна легко адаптувати до будь-яких потреб, інтегруючи з різними інструментами та технологіями для створення комплексних рішень для автоматизації. Jenkins також забезпечує безперервну інтеграцію і розгортання, дозволяючи налаштувати процеси CI/CD, що автоматизують всі етапи розробки, включаючи збірку, тестування та розгортання.

Однак, як і у випадку з Kubernetes, налаштування і підтримка Jenkins можуть бути складними та вимагати значних зусиль і ресурсів. Jenkins також

потребує значних обчислювальних ресурсів для забезпечення своєї роботи, особливо при великій кількості одночасно виконуваних завдань. Це може бути проблемою для невеликих компаній або проектів з обмеженими ресурсами [10].

Docker є чудовим вибором для контейнеризації додатків, забезпечуючи портативність та ізоляцію. Він легкий у використанні і дозволяє швидко створювати та запускати контейнери. Docker Compose додає додаткові можливості для управління багатоконтейнерними додатками, що робить його ідеальним для невеликих та середніх проектів. Однак для великих розподілених систем Docker може бути недостатньо потужним, і тут на допомогу приходить Kubernetes.

Kubernetes забезпечує автоматизацію управління контейнерами у великих масштабах. Він надає потужні інструменти для оркестрації контейнерів, забезпечуючи високу доступність, масштабованість та стійкість до збоїв. Однак Kubernetes має складнішу архітектуру і потребує більше зусиль на налаштування та управління, що може бути викликом для невеликих команд або проектів з обмеженими ресурсами.

Ansible, на відміну від Docker та Kubernetes, зосереджується на автоматизації IT-процесів. Він забезпечує простоту використання та гнучкість, дозволяючи автоматизувати управління конфігураціями та розгортання додатків. Ansible не потребує встановлення агентів на керованих вузлах, що спрощує його інтеграцію та використання. Однак продуктивність Ansible може бути обмеженою для великих розподілених систем, і він має обмежену підтримку стану.

Jenkins є сервером автоматизації, який забезпечує безперервну інтеграцію і розгортання. Завдяки підтримці тисяч плагінів, Jenkins можна легко адаптувати до будь-яких потреб, інтегруючи з різними інструментами та

технологіями для створення комплексних рішень для автоматизації. Однак налаштування і підтримка Jenkins можуть бути складними, і він потребує значних обчислювальних ресурсів для забезпечення своєї роботи.

Кожна з розглянутих технологій має свої сильні та слабкі сторони, і вибір між ними залежить від конкретних вимог проекту. Docker та Docker Compose є чудовим вибором для невеликих та середніх проектів, де важлива портативність та легкість використання. Kubernetes підходить для великих розподілених систем, де потрібна автоматизація управління контейнерами у великих масштабах. Ansible є потужним інструментом для автоматизації IT-процесів, забезпечуючи простоту та гнучкість у налаштуванні та розгортанні додатків. Jenkins забезпечує безперервну інтеграцію і розгортання, дозволяючи автоматизувати всі етапи розробки програмного забезпечення.

1.3 Постановка задач на створення програмного модулю розгортання серверної інфраструктури для додатку планування ресурсів підприємства

Для розробки програмного модуля розгортання серверної інфраструктури для додатку планування ресурсів підприємства (ERPNext) необхідно виконати ряд послідовних кроків. Основна мета —пришвидшити процес розгортання серверної інфраструктури для управління інстанціями ERPNext, включаючи їх розгортання, моніторинг, резервне копіювання та відновлення. Завдання можна розділити на наступні етапи:

- Розробка структури компонентів - розробити структуру системи, яка включатиме основні компоненти програмного модулю
- Розробка алгоритму роботи програмного модуля – розробити алгоритм роботи програмного модулю, який повинен забезпечувати ініціалізацію веб-додатку та налаштування взаємодії між компонентами.

- Обґрунтування вибору мови та середовища програмування – обґрунтувати мову програмування та середовище розробки, зі допомогоб яких буде створено програмний модуль
- Програмна реалізація модуля – реалізувати програмний модуль розгортання серверно інфраструктури для додатку планування ресурсів підприємтсва

Таким чином, постановка задач для розробки програмного модуля включає визначення структури компонентів, розробку алгоритму роботи, обґрунтування вибору мови та середовища програмування, використання фреймворків та бібліотек, а також програмну реалізацію модуля, що забезпечить створення ефективного та надійного рішення для розгортання та управління інстанціями ERPNext.

1.4 Висновки

У першому розділі проведено детальний огляд та аналіз сучасних технологій для розгортання серверної інфраструктури. Розглянуто чотири основні інструменти: Docker, Kubernetes, Ansible та Jenkins, кожен з яких має свої унікальні переваги та недоліки.

Docker та Docker Compose є чудовим вибором для контейнеризації додатків, забезпечуючи портативність, ізоляцію та легкість використання [3]. Frappe Cloud було обрано як основний аналог для розгортання ERPNext інфраструктури.

2 ПРОЕКТУВАННЯ СТРУКТУРИ ТА КОМПОНЕНТІВ ПРОГРАМНОГО МОДУЛЮ

2.1 Проектування структури компонентів програмного модулю розгортання серверної інфраструктури для додатку планування ресурсів підприємства

Розробка програмного модуля для розгортання серверної інфраструктури ERP-системи ERPNext базуватиметься на використанні веб-додатку, створеного на основі фреймворку Django [12]. Цей модуль надасть зручний інтерфейс для управління інстанціями додатку, надаючи можливість розгортати нові інстанції, керувати існуючими, а також виконувати резервне копіювання та відновлення даних. Основними компонентами цього модуля будуть панель управління інстанціями, механізм відображення логів, а також інструменти для моніторингу завантаженості серверів. Розглянемо детально кожен аспект розробки цього модуля.

Головна сторінка веб-додатку буде містити дашборд, який надасть користувачеві загальний огляд стану серверної інфраструктури. Цей дашборд відображатиме інформацію про завантаженість серверів, на яких розгорнуті інстанції ERPNext. Зокрема, він показуватиме поточний рівень завантаженості CPU, використання пам'яті, а також кількість активних інстанцій. Для цього будуть використані інтегровані бібліотеки для збору та візуалізації метрик.

На сторінці управління інстанціями буде представлено список всіх розгорнутих інстанцій ERPNext. Кожна інстанція буде відображатися у вигляді картки або рядка таблиці, яка міститиме основну інформацію про інстанцію, включаючи її ім'я, стан (активна, зупинена), і поточний статус

(завантаженість CPU, використання пам'яті тощо). Користувач зможе вибрати будь-яку інстанцію для детального перегляду та управління.

При виборі конкретної інстанції користувач потраплятиме на детальну сторінку, де буде надано повну інформацію про цю інстанцію. Тут відображатимуться логи роботи інстанції, що дозволить швидко діагностувати можливі проблеми. Крім того, на цій сторінці будуть розміщені кнопки для керування інстанцією:

Запуск інстанції: Кнопка для запуску зупиненої інстанції. Після натискання буде викликано відповідний Docker-команду для запуску контейнера.

Зупинка інстанції: Кнопка для зупинки активної інстанції. Після натискання буде викликано Docker-команду для зупинки контейнера.

Резервне копіювання бази даних: Кнопка для створення бекапу бази даних інстанції. Після натискання буде виконано відповідний скрипт для збереження бекапу бази даних у безпечне місце.

Відновлення бази даних: Кнопка для відновлення бази даних з резервної копії. Після натискання буде викликано скрипт для відновлення бази даних з попередньо збереженого бекапу.

Головна сторінка веб-додатку є центральною точкою входу для користувача і надає загальний огляд стану серверної інфраструктури. Ця сторінка відображає ключові метрики, які дозволяють користувачам швидко оцінити поточну ситуацію та приймати обґрунтовані рішення. Для реалізації цієї сторінки в рамках Django створюється представлення (view), яке взаємодіє з JS для збору необхідних даних про завантаженість CPU, використання пам'яті та дискового простору на серверах [15]. Дані, отримані відj JS, передаються у шаблон (template), де вони візуалізуються за допомогою бібліотек для побудови графіків, таких як Chart.js або D3.js.

Цей підхід дозволяє створити інтерактивні графіки, які динамічно відображають актуальну інформацію. Користувач може бачити, як змінюється завантаженість серверів у реальному часі, що дозволяє швидко реагувати на можливі проблеми або перевантаження системи.

Сторінка управління інстанціями надає користувачам можливість переглядати всі розгорнуті інстанції ERPNext та керувати ними [22]. На цій сторінці відображається список всіх інстанцій у вигляді таблиці або карток, кожна з яких містить основну інформацію про інстанцію, таку як її ім'я, поточний стан (активна або зупинена), та основні метрики завантаженості (CPU, пам'ять, диск).

Для кожної інстанції також передбачені кнопки для виконання основних операцій: запуск, зупинка, резервне копіювання та відновлення бази даних. Взаємодія з Docker-контейнерами відбувається через виклики до Docker API, що забезпечує надійність і ефективність процесів [16]. Користувач може швидко запускати або зупиняти інстанції, створювати резервні копії або відновлювати дані з бекапів без необхідності вручну виконувати команди в командному рядку. Коли користувач вибирає конкретну інстанцію зі списку, він потрапляє на детальну сторінку, де представлена більш детальна інформація про обрану інстанцію. На цій сторінці відображаються логи роботи інстанції у реальному часі, що дозволяє користувачам моніторити стан інстанції та швидко діагностувати можливі проблеми.

Крім логів, на детальній сторінці також розміщені кнопки для керування інстанцією: запуск, зупинка, створення бекапу та відновлення з резервної копії. Всі ці операції виконуються через виклики до Docker API, що забезпечує простоту та ефективність управління.

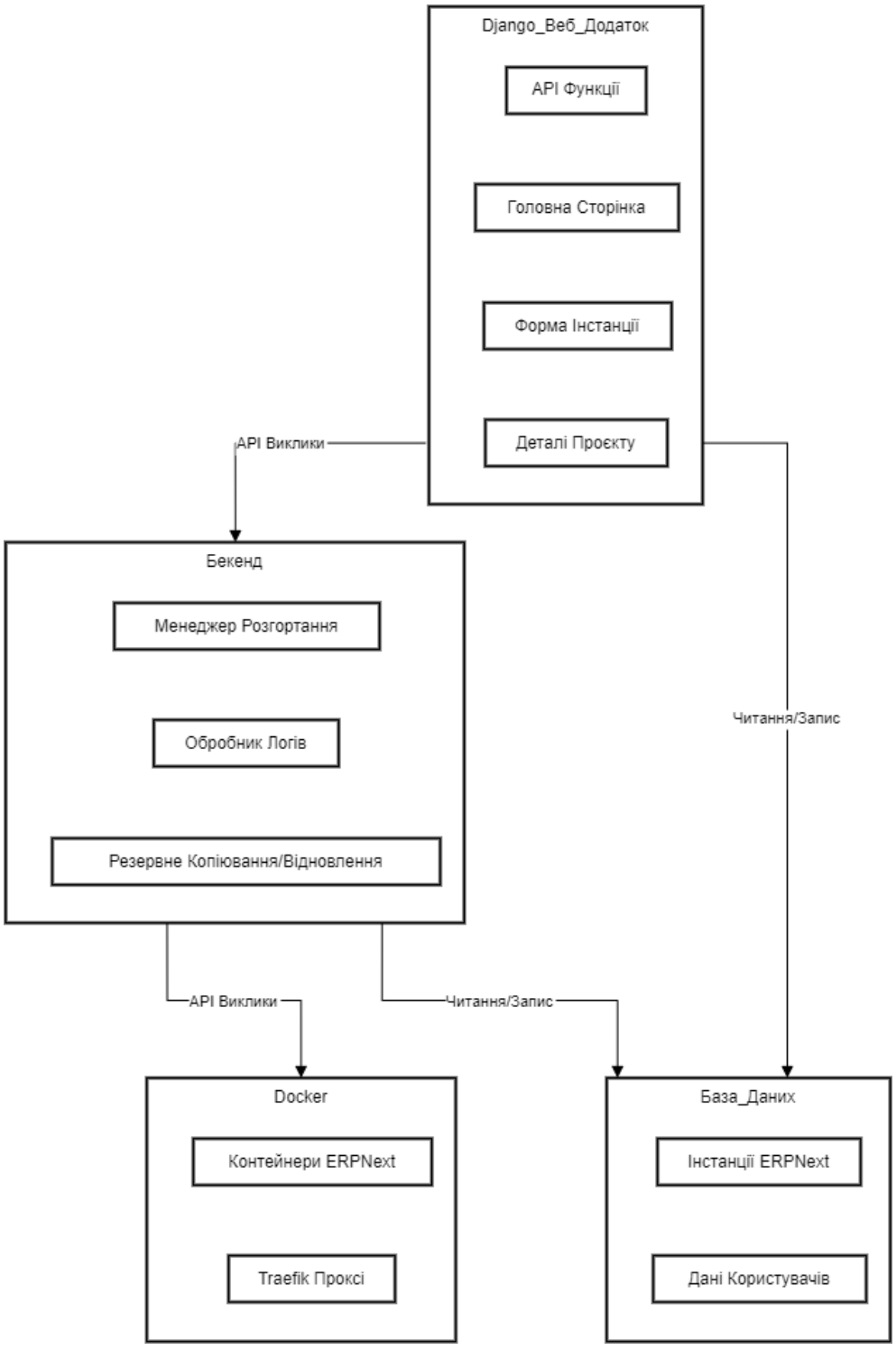

Рисунок 2.1 – UML-діаграма компонентів програмного модулю розгортання серверної інфраструктури для додатку планування ресурсів підприємтсва

2.2 Розробка алгоритму роботи програмного модулю розгортання серверної інфраструктури для додатку планування ресурсів підприємства

Розробка програмного модуля для керування інстанціями ERP-системи ERPNext включає детальне планування всіх операцій, які можуть виконуватися користувачем. Модуль має забезпечити зручний інтерфейс для створення, запуску, зупинки інстанцій, а також для резервного копіювання та відновлення даних. Розглянемо покроковий алгоритм роботи модуля на рівні програмного забезпечення.

Коли користувач подає запит на створення нової інстанції, веб-додаток приймає цей запит та розпочинає процес ініціалізації. Спочатку генерується унікальний ідентифікатор для нової інстанції, щоб її можна було легко ідентифікувати серед інших інстанцій. Далі формується конфігураційний файл Docker Compose, який містить усі необхідні налаштування для створення контейнерів для нової інстанції.

Конфігураційний файл зберігається на сервері, після чого виконується команда Docker Compose для створення і запуску нових контейнерів. Після успішного запуску контейнерів інформація про нову інстанцію зберігається в базі даних.

Коли користувач подає запит на запуск інстанції, веб-додаток приймає цей запит та перевіряє статус інстанції в базі даних. Якщо інстанція зупинена, виконується команда Docker для запуску відповідного контейнера. Після успішного запуску контейнера оновлюється статус інстанції в базі даних, щоб відобразити її активний стан.

Коли користувач подає запит на зупинку інстанції, веб-додаток приймає цей запит та перевіряє статус інстанції в базі даних. Якщо інстанція активна, виконується команда Docker для зупинки відповідного контейнера. Після

успішного зупинки контейнера оновлюється статус інстанції в базі даних, щоб відобразити її зупинений стан.

Коли користувач подає запит на створення резервної копії бази даних інстанції, веб-додаток приймає цей запит і виконує команду Docker для запуску процесу створення резервної копії всередині контейнера. Після створення резервної копії вона переноситься до безпечного місця для зберігання. Інформація про останній бекап оновлюється в базі даних.

Коли користувач подає запит на відновлення бази даних з резервної копії, веб-додаток приймає цей запит і виконує команду Docker для запуску процесу відновлення бази даних з резервної копії всередині контейнера. Після успішного відновлення оновлюється статус інстанції в базі даних.

Моніторинг стану інстанцій здійснюється за допомогою Prometheus, який збирає метрики про стан контейнерів і серверів, на яких вони запущені. Дані візуалізуються на дашборді за допомогою JS, що дозволяє користувачам отримувати актуальну інформацію про завантаженість серверів та стан інстанцій у реальному часі.

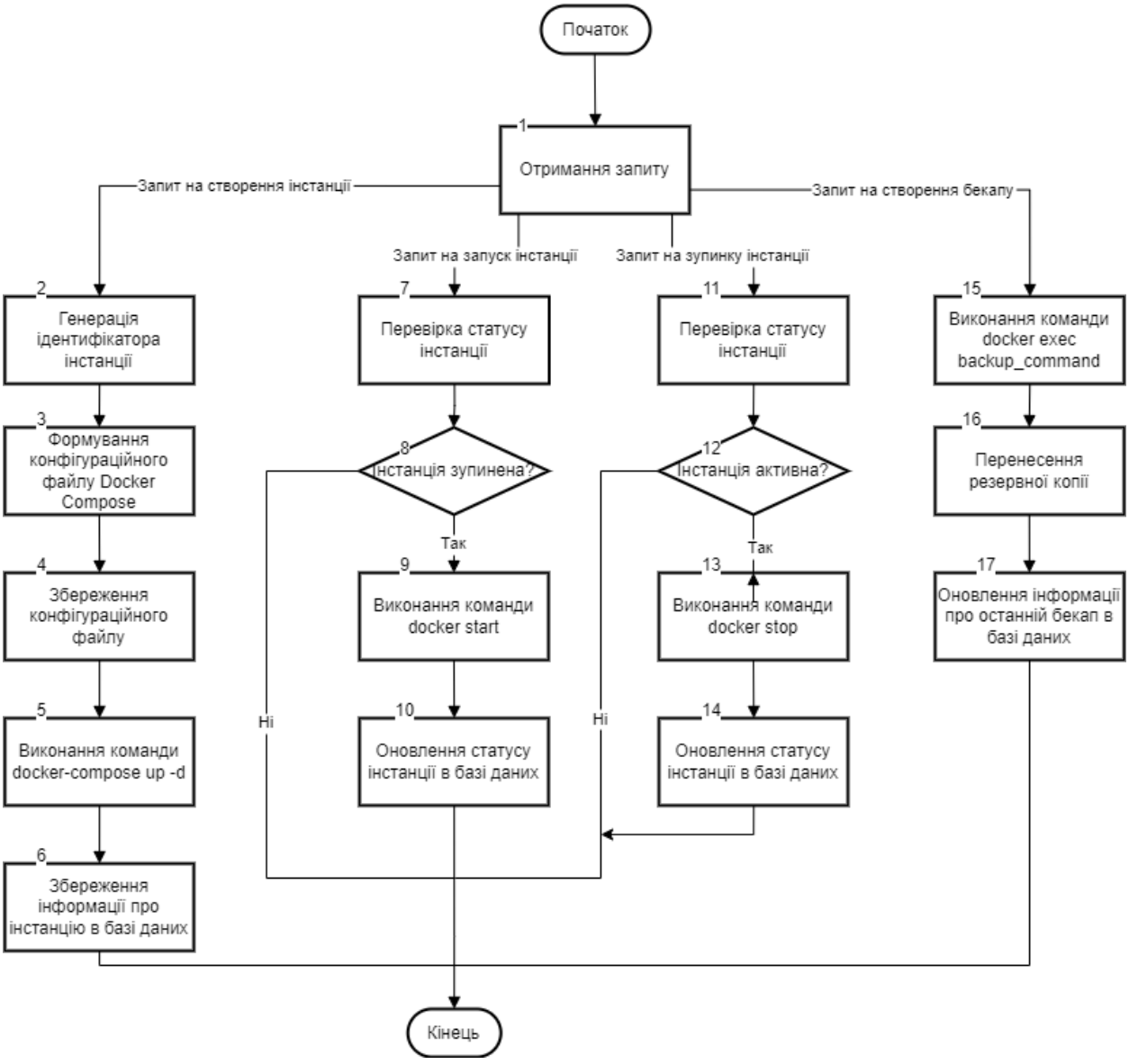

Рисунок 2.1 – Алгоритм роботи програмного модулю розгортання серверної інфраструктури для додатку планування ресурсів підприємства

2.3 Висновки

У цьому розділі детально розглянуто та описано основні компоненти системи для розгортання серверної інфраструктури ERP-системи ERPNext, зокрема модуль керування інстанціями. Описано структуру та взаємодію між компонентами системи, а також принципи роботи кожного з них.

Використання Docker і Docker Compose дозволяє забезпечити ефективну контейнеризацію додатків, ізоляцію середовища виконання та швидке розгортання інстанцій. Docker забезпечує можливість створення і запуску контейнерів з усіма необхідними залежностями, а Docker Compose спрощує управління багатоконтейнерними додатками, дозволяючи легко налаштовувати мережі та томи для зберігання даних.

Інтерфейс, розроблений на основі Django, надає зручний та інтуїтивний спосіб взаємодії користувачів з системою. Користувачі можуть створювати нові інстанції, запускати і зупиняти існуючі, а також здійснювати резервне копіювання та відновлення баз даних через зручні форми та панелі керування.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ МОДУЛЮ РОЗГОРТАННЯ ІНФРАСТРУКТУРИ ДЛЯ ДОДАТКУ ПЛАНУВАННЯ РЕСУРСІВ ПІДПРИЄМСТВА

3.1 Обґрунтування вибору мови та середовища програмування

Python — це високорівнева мова програмування загального призначення [13]. Її дизайн фокусується на коді, який легко читається, що досягається завдяки простому і зрозумілому синтаксису. Python підтримує різні парадигми програмування, включаючи об’єктно-орієнтоване, функціональне та процедурне програмування.

Однією з найбільших переваг Python є велика стандартна бібліотека, яка забезпечує широкий спектр модулів і пакетів для вирішення різноманітних завдань, починаючи від обробки тексту і завершуючи взаємодією з операційною системою. Крім того, Python має активну і велику спільноту розробників, що сприяє постійному розвитку мови та доступності великої кількості додаткових бібліотек.

Технічні характеристики Python:

- Легкочитність і простота коду: Один з основних принципів дизайну Python — це його легкочитність. Синтаксис мови нагадує англійську мову, що робить код інтуїтивно зрозумілим і легким для читання. Це особливо важливо для командної роботи та підтримки коду в довгостроковій перспективі.
- Багатоплатформність: Python є кросплатформною мовою програмування, що дозволяє запускати програми на різних операційних системах, таких як Windows, MacOS і Linux, без значних змін у коді.

- Широка стандартна бібліотека: Python постачається з великою кількістю модулів, які дозволяють вирішувати типові завдання, такі як робота з файлами, мережеве програмування, маніпуляція з даними та інше, без необхідності встановлення додаткових пакетів.
- Підтримка інтеграції з іншими мовами: Python легко інтегрується з такими мовами, як C, C++, Java, що дозволяє використовувати його в гетерогенних середовищах і взаємодіяти з існуючими системами.
- Масштабованість: Хоча Python часто використовується для написання скриптів і невеликих програм, він також підходить для розробки масштабних і складних додатків завдяки своїй гнучкості та можливостям модульного програмування.

Однією з важливих характеристик Python є його модульність та розширюваність. Кожен компонент системи може бути реалізований як окремий модуль, що взаємодіє з іншими модулями через чітко визначені інтерфейси. Це полегшує процес додавання нових функцій та адаптації існуючої системи до нових вимог. У контексті нашого проекту це означає, що ми можемо легко додавати нові функції для керування Docker контейнерами або інтеграції з іншими сервісами без необхідності значних змін у кодовій базі. Крім того, велика кількість доступних бібліотек та інструментів дозволяє швидко знаходити готові рішення для багатьох задач, що знижує витрати часу та ресурсів на розробку.

Python є одним із найпопулярніших виборів для розробників, коли йдеться про написання скриптів для автоматизації та взаємодії з операційними системами. Його синтаксис, структура та широкі можливості роблять цю мову ідеальним інструментом для створення надійних та ефективних скриптів.

Завдяки зрозумілому та логічному синтаксису, скрипти на Python легко читати та підтримувати. Це особливо важливо для автоматизації задач, де часто потрібно швидко писати, тестувати та модифікувати код. Читаємий код полегшує виявлення та виправлення помилок, що є критично важливим для скриптів, які взаємодіють з операційною системою та її компонентами.

Скрипти можуть бути розбиті на модулі та пакети, що сприяє організації коду та полегшує його підтримку. Це особливо корисно для великих проектів з автоматизації, де важливо мати чітку структуру та розділити функціональність на окремі частини. Модульність Python дозволяє легко додавати нові функції до існуючих скриптів без значних змін у кодовій базі, що забезпечує масштабованість та зручність розробки.

3.2 Використання фреймворків та бібліотек

Django – це потужний веб-фреймворк для Python, який дозволяє швидко створювати надійні, масштабовані та безпечні веб-додатки [12]. Django розроблений з акцентом на простоту та швидкість розробки, що дозволяє розробникам зосередитися на написанні логіки додатка, а не на вирішенні інфраструктурних питань. Однією з найбільших переваг Django є його підхід «Batteries Included», що означає наявність великої кількості вбудованих компонентів та інструментів, які готові до використання з коробки. Це включає систему автентифікації користувачів, адміністративний інтерфейс, ORM (Object-Relational Mapping) для роботи з базами даних, маршрутизацію URL, валідацію форм та багато іншого. Такий підхід значно скорочує час розробки та спрощує процес інтеграції різних компонентів.

Однією з важливих особливостей Django є його ORM, що дозволяє працювати з базами даних через об’єктно-орієнтований підхід.

Завдяки ORM, розробники можуть маніпулювати даними в базі даних, використовуючи Python-класи та методи, замість написання сирого SQL. Це не лише підвищує продуктивність роботи, але й забезпечує безпеку та захист від типових атак, таких як SQL-ін'єкції. ORM також полегшує процес міграції даних, дозволяючи легко змінювати структуру бази даних і автоматично застосовувати ці зміни до реальної бази.

Також Django включає потужний адміністративний інтерфейс, який генерується автоматично на основі моделей даних. Це дає можливість швидко створювати зручний інтерфейс для управління даними, що значно полегшує роботу адміністраторам та розробникам. Адміністративний інтерфейс можна легко налаштовувати та розширювати, додаючи нові функції та змінюючи існуючі.

Безпека є ще одним ключовим аспектом, на який звертає увагу Django. Фреймворк надає вбудовані механізми для захисту від поширених веб-загроз, таких як CSRF (Cross-Site Request Forgery), XSS (Cross-Site Scripting), і SQL-ін'єкції. Це дозволяє розробникам зосередитися на логіці додатку, не турбуючись про основні аспекти безпеки, які автоматично обробляються фреймворком. Маршрутизація запитів в Django також реалізована з великою увагою до простоти та гнучкості. Використовуючи зрозумілі та інтуїтивні правила маршрутизації, розробники можуть легко налаштовувати URL-и для своїх додатків, забезпечуючи зручну навігацію для користувачів. Django дозволяє створювати як прості маршрути, так і складні схеми URL, підтримуючи всі необхідні функціональні можливості для створення сучасних веб-додатків.

Фреймворк дозволяє легко інтегрувати сторонні бібліотеки та додатки, завдяки чому можна швидко додавати нові функції до проекту.

Django також підтримує можливість створення багаторазових компонентів та модулів, що сприяє повторному використанню коду та зменшує кількість роботи при розробці нових функціональностей [12].

Django орієнтований на швидкий розвиток та розгортання веб-додатків. Він надає вбудовані інструменти для розгортання, тестування та обслуговування додатків. Вбудована система управління конфігурацією та налаштувань дозволяє легко адаптувати додаток до різних середовищ, забезпечуючи плавний процес переходу від розробки до продуктивного використання.

Вибір Django обумовлений його численними технічними перевагами, включаючи високу продуктивність, безпеку, гнучкість та простоту використання. Використання цього фреймворку дозволяє значно скоротити час розробки, підвищити якість коду та забезпечити надійну роботу веб-додатку, що є критично важливим для успішної реалізації проекту розгортання серверної інфраструктури для додатку планування ресурсів підприємства.

python_on_whales є сучасною бібліотекою для Python, яка забезпечує зручний інтерфейс для роботи з Docker [14]. Вона надає високоабстрактні засоби для керування Docker контейнерами, мережами, томами та іншими компонентами. Це дозволяє розробникам взаємодіяти з Docker API на високому рівні, спрощуючи процес написання скриптів для розгортання та управління контейнерами. Особливо корисним функціоналом є широка взаємодія з плагіном Docker Compose, що дозволяє керувати складними багатоконтейнерними додатками. Це особливо важливо для проєкту, де кожен екземпляр ERPNext розгортається як окремий Docker Compose проєкт.

Використовуючи python_on_whales, ми можемо легко розгортати нові екземпляри, керувати існуючими та забезпечувати резервне копіювання та відновлення даних [14]. Це забезпечує високу гнучкість та ефективність управління контейнеризованими додатками.

psutil є потужною бібліотекою для Python, яка дозволяє зчитувати інформацію про системні ресурси та процеси. Вона надає інструменти для моніторингу CPU, пам'яті, дисків, мережі, датчиків та інших параметрів системи. Це робить psutil надзвичайно корисною для задач моніторингу та управління ресурсами. Однією з ключових переваг psutil є її здатність працювати на різних платформах, включаючи Windows, Linux та macOS. Це забезпечує крос-платформну сумісність, що важливо для проектів, які повинні функціонувати в різних середовищах. psutil дозволяє отримувати детальну інформацію про процеси, включаючи їх ідентифікатори, статус, використання CPU та пам'яті, що є критично важливим для моніторингу продуктивності системи та виявлення проблем.

Завдяки простому та інтуїтивно зрозумілому API, psutil дозволяє легко інтегрувати функціональність моніторингу в додаток. Бібліотеку буде використовуватись для отримання інформації про завантаження CPU, кількість активних процесів, використання оперативної пам'яті та інші метрики, які необхідні для відображення системних характеристик на нашому веб-інтерфейсі. Це забезпечує користувачам можливість отримувати актуальну інформацію про стан системи та приймати відповідні рішення.

os є стандартною бібліотекою Python, яка надає засоби для взаємодії з операційною системою. Вона забезпечує функціональність для роботи з файловою системою, управління процесами та виконання системних команд. os є незамінною частиною стандартної бібліотеки Python, яка використовується для виконання багатьох базових операцій.

В бібліотеці є можливість роботи з файловою системою. Вона дозволяє створювати, видаляти, перейменовувати та переміщувати файли та директорії. Це особливо корисно для написання скриптів, які повинні виконувати операції з файлами, такі як збереження логів, резервне копіювання даних або створення тимчасових файлів. os також забезпечує засоби для управління середовищем виконання, включаючи доступ до змінних середовища, інформації про поточний процес та його нащадки. Це дозволяє отримувати інформацію про налаштування системи та конфігурацію, що може бути корисним для налаштування поведінки додатку в залежності від середовища виконання.

3.3 Програмна реалізація модулю розгортання серверної інфраструктури для додатку планування ресурсів підприємства

У цьому розділі детально розглянемо програмну реалізацію модуля, який автоматизує розгортання серверної інфраструктури для додатку планування ресурсів підприємства (ERPNext). Цей модуль надає можливість запускати, зупиняти, моніторити та керувати Docker контейнерами, що забезпечують роботу ERPNext екземплярів. Зокрема, ми зосередимося на ключових функціях модулю.

Для початку ми створили Django проект, який слугує основою для нашого веб-додатку. Ми підключили необхідні бібліотеки, такі як docker, os, psutil, crypt, platform, python_on_whales, cpuinfo, а також стандартні бібліотеки Django. Це забезпечило нас усіма необхідними інструментами для роботи з Docker контейнерами та операційною системою.

Бекенд розділений на два блоки функцій: API-функції, які передають дані на фронтенд сторону сайту; утиліти, які виконують операції на сайті, наприклад, розгортка нового екземпляру ERPNext

Розглянемо основні API-функції, які опрацьовують дані системи про CPU, оперативну пам’ять та пам’ять дисків і відсилають обчислені дані на фронтенд частину, яка зі свого боку обробляє їх за допомогою JavaScript. Після цього, оброблені дані потрапляють на сторінку.

Функція get_cpu_load відповідає за отримання поточного завантаження CPU. Вона використовує бібліотеку psutil для збору даних про завантаження процесора за останню секунду і повертає цю інформацію у форматі JSON. Це дозволяє користувачам швидко отримувати актуальну інформацію про стан процесора їхньої системи через веб-інтерфейс.

```
def get_cpu_load(request):
    context = {'cpu_load': psutil.cpu_percent(interval=1)}
    return JsonResponse(context)
```

Функція get_system_info збирає інформацію про систему, включаючи марку процесора, кількість ядер, обсяг оперативної пам'яті та операційну систему. Ці дані отримуються за допомогою бібліотек psutil, platform та cpuinfo, і повертаються у форматі JSON.

На рисунку 3.1 зображено показник, який вказує відсоток завантаження процесора поточного середовища.

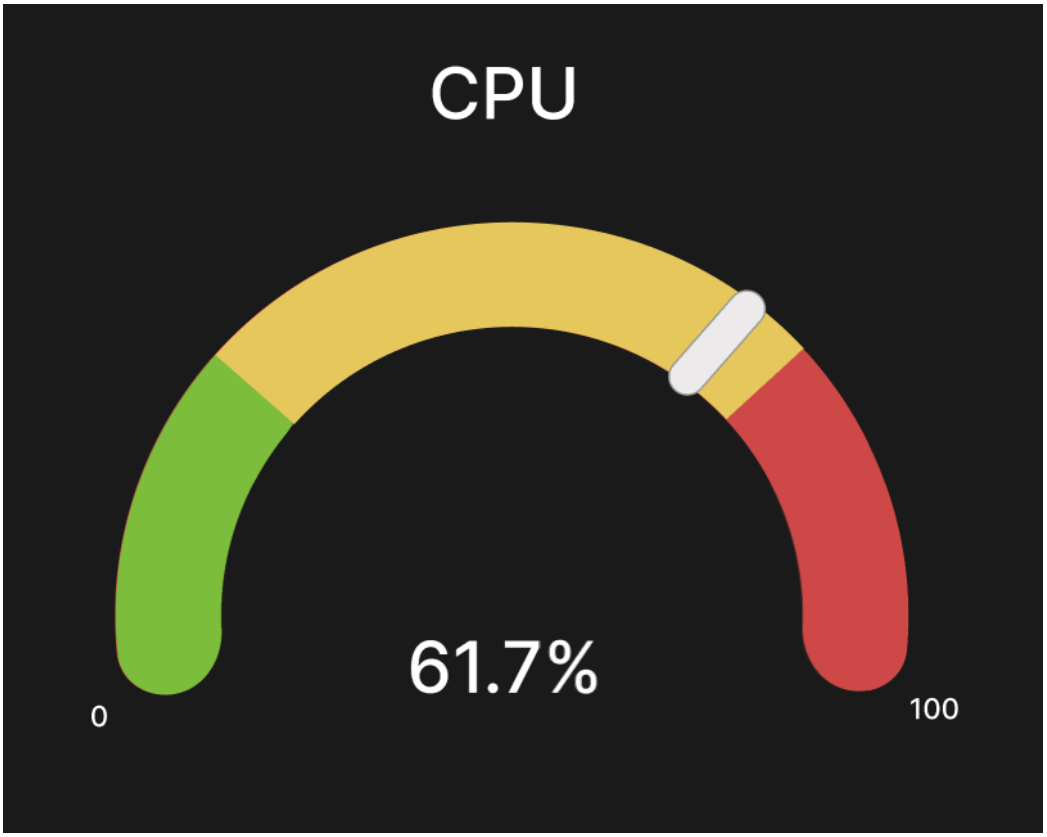

Рисунок 3.1 – Показник завантаження процесора

```
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 таблицю системних параметрів середовища.

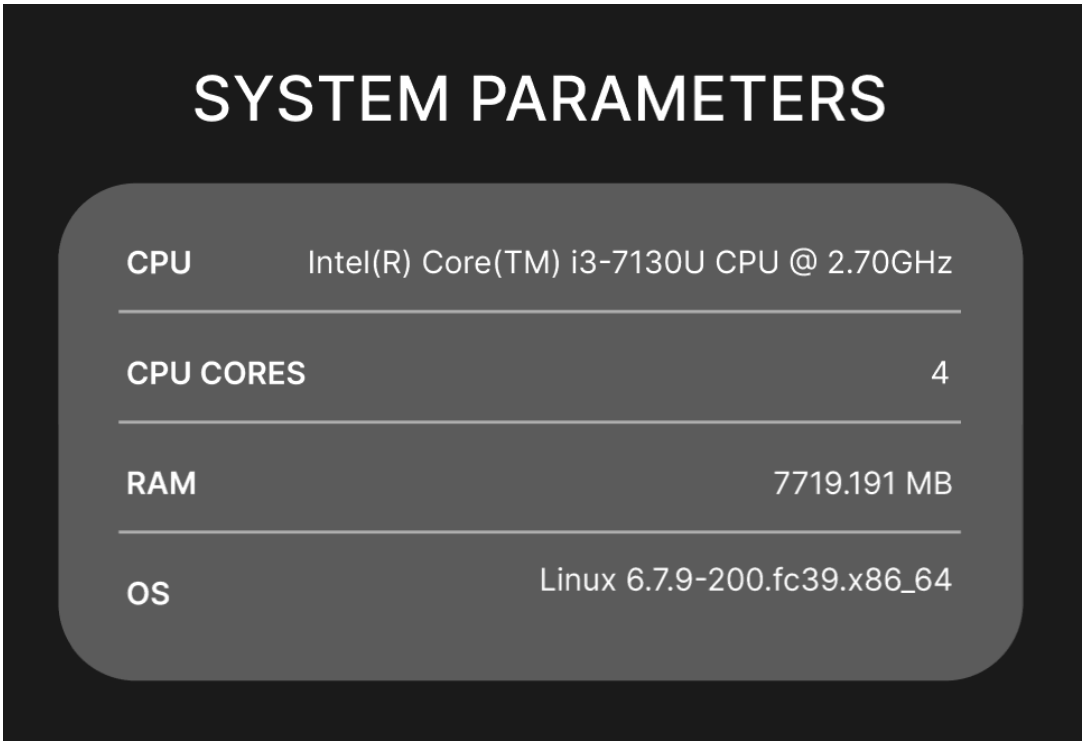

Рисунок 3.2 – Таблиця системних параметрів

Функція get_memory_info збирає інформацію про використання оперативної пам'яті та дискового простору. Вона обчислює загальний та використаний обсяг пам'яті, а також залишкову ємність. Отримані дані повертаються у форматі JSON, що забезпечує користувачів актуальною інформацією про стан системної пам'яті та дисків.

```
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 прикладі install_instance розглянемо функції-утиліти, які виконують основні операції над екземплярами ERPNext, звертаючись до Docker

Такі функції описують операції створення нового екземпляру, видалення екземпляру, операції над контейнерами, внутрішній бекап даних з певного сайту та рестор даних на певному сайті.

Функція install_instance є однією з ключових утиліт для розгортання нового екземпляру ERPNext. Вона включає декілька основних кроків:

- Перевірка наявності необхідних репозиторіїв та їх клонування: Якщо репозиторій frappe_docker не існує, він клонуються з GitHub:

```
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')
```

- Налаштування Traefik: Якщо Traefik ще не налаштований, створюються відповідні конфігураційні файли та запускається Docker Compose проект для Traefik:

```
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 налаштування проекту: Створюються директорії для нового проекту, копіюються та модифікуються конфігураційні файли для MariaDB та ERPNext:

```
    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 Docker Compose проектів: Запускаються Docker Compose проекти для MariaDB та ERPNext з налаштуванням відповідних змінних середовища.

```
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)
```

Ініціалізація нового сайту ERPNext: Виконується команда для створення нового сайту ERPNext з вказаними параметрами:

```
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)
```

На рисунку 3.3 зображено вигляд таблиці всіх інстальованих проєктів на середовищі користувача.

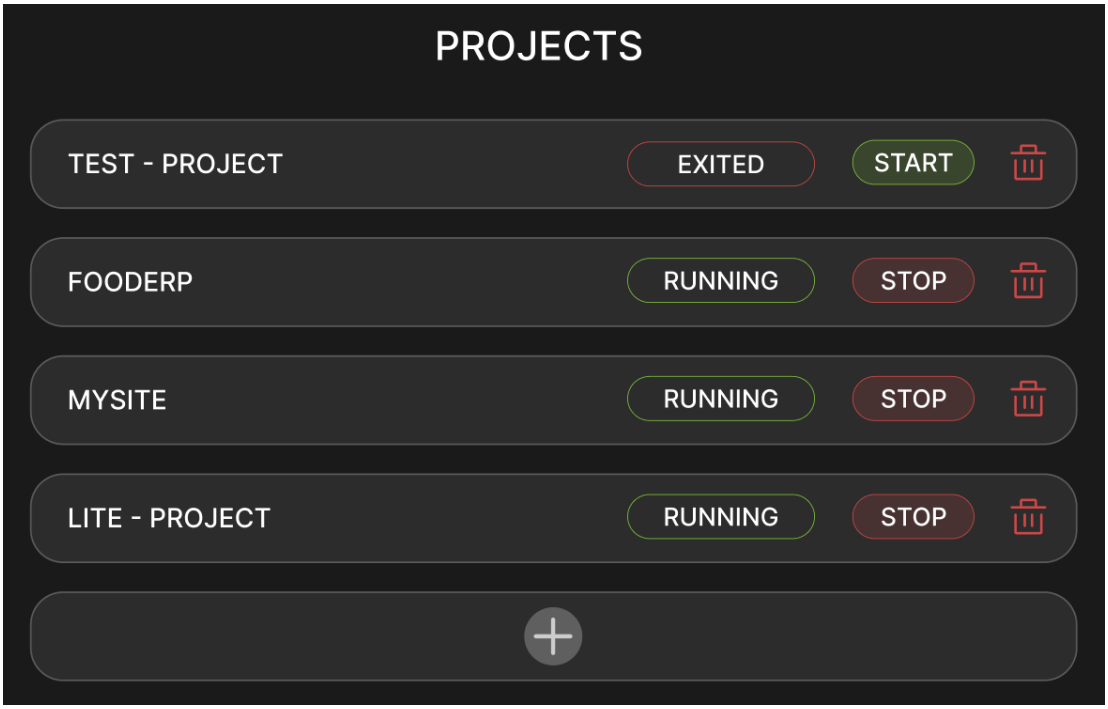

Рисунок 3.3 – Таблиця інстальованих проєктів

3.4 Тестування роботи програмного модулю розгортання серверної інфраструктури для додатку планування ресурсів підприємства

На головній сторінці користувач бачить системні характеристики свого середовища. Системні характктеристики середовища зображено на рисунку 3.4

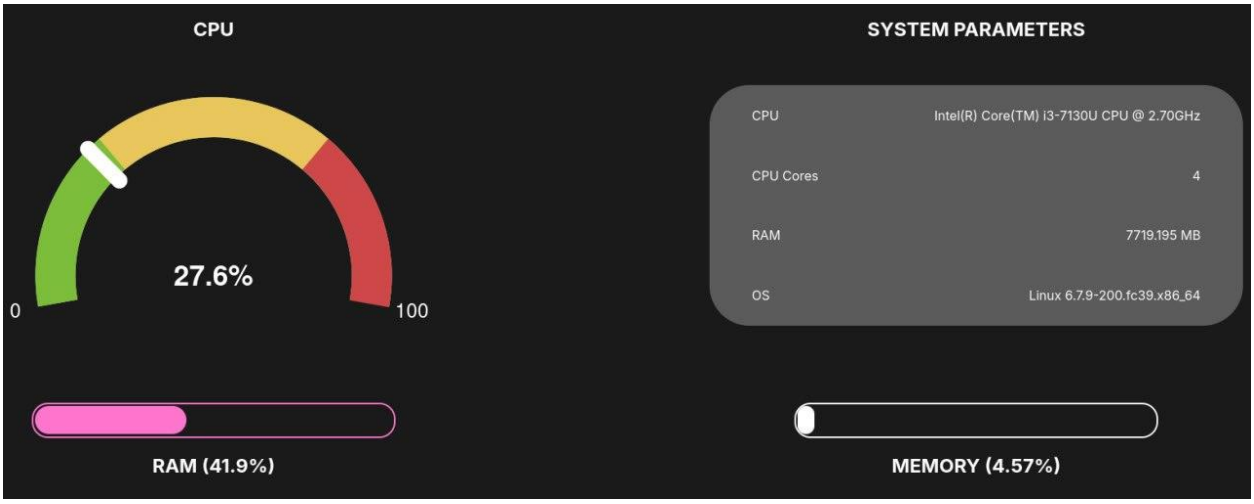

Рисунок 3.4 – Системні характеристики середовища

Далі, на головній сторінці, можна проглянути всі Docker Compose проєктів, розгорнутих на вашому середовищі. Список всіх Docker Compose проєктів зображено на рисунку 3.5

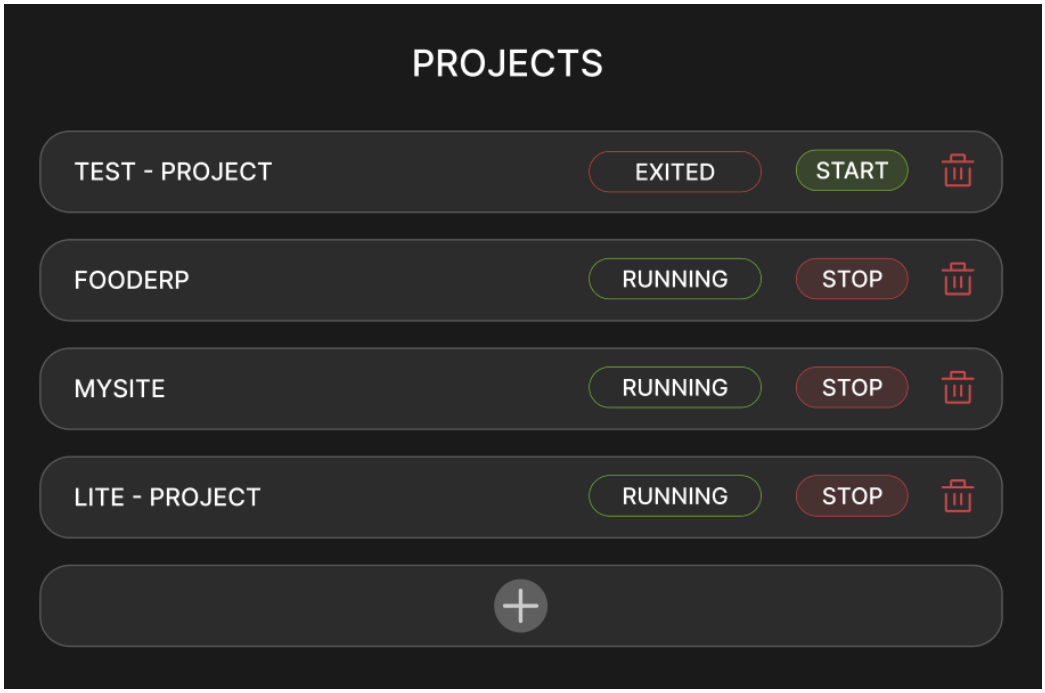

Рисунок 3.5 – Список всіх Docker Compose проєктів

Для того, щоб створити новий екземпляр сайту користувач має натиснути на блок зі знаком «+». Після цього потрібно буде ввести дані вашого нового екземпляру в формі інсталяції. Після отримання даних із форми, програмний модуль почне розгортання інстанції ERPNext як новий Docker Compose проєкт на середовище користувача. Форма додавання нового екземпляру для розгортання зображена на рисунку 3.6

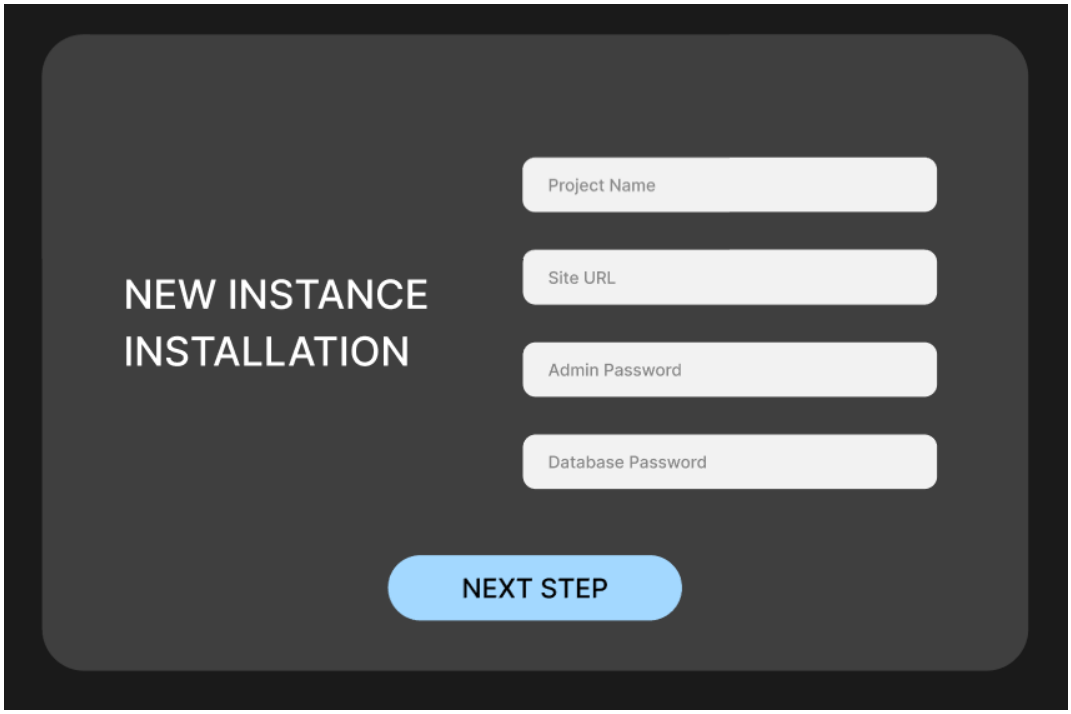

Рисунок 3.6 - Форма додавання нового екземпляру для розгортання

Форма інсталяції проксі-сервера Traefik зображена на рисунку 3.7

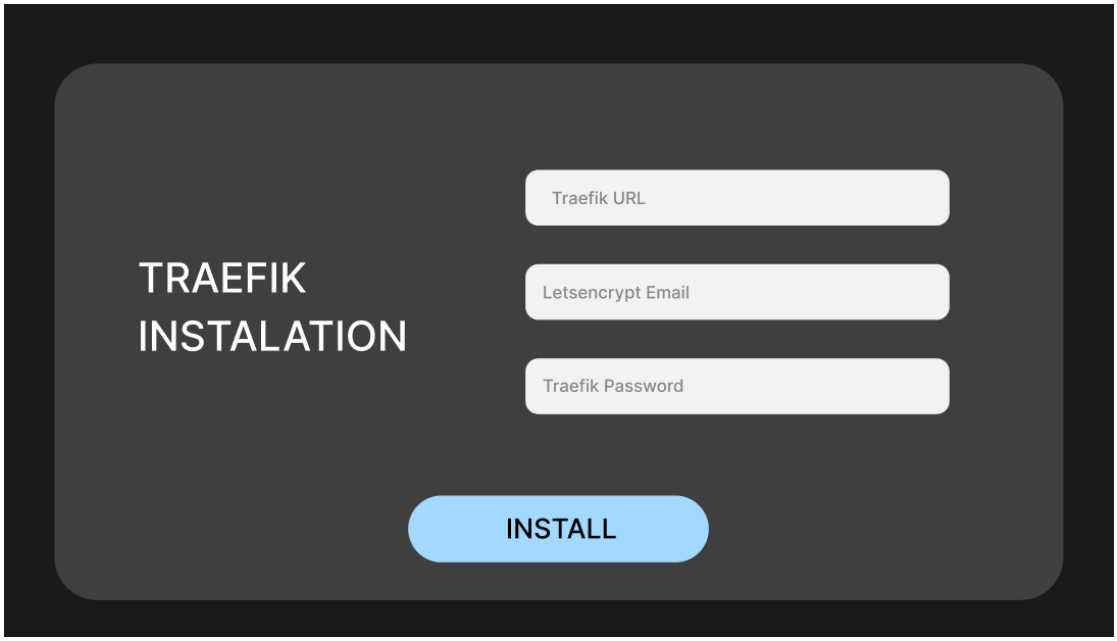

Рисунок 3.7 - Форма інсталяції проксі-сервера Traefik

Перейшовши на певний Docker Compose проєкт, користувач отримає доступ до сторінки проєкту, на якій розміщено список всіх контейнерів цього проєкту. Тут можна побачити інформацію про контейнери, статус та зупинити або запустити контейнер. Список Docker контейнерів, які належать певному екземпляру зображено на рисунку 3.8

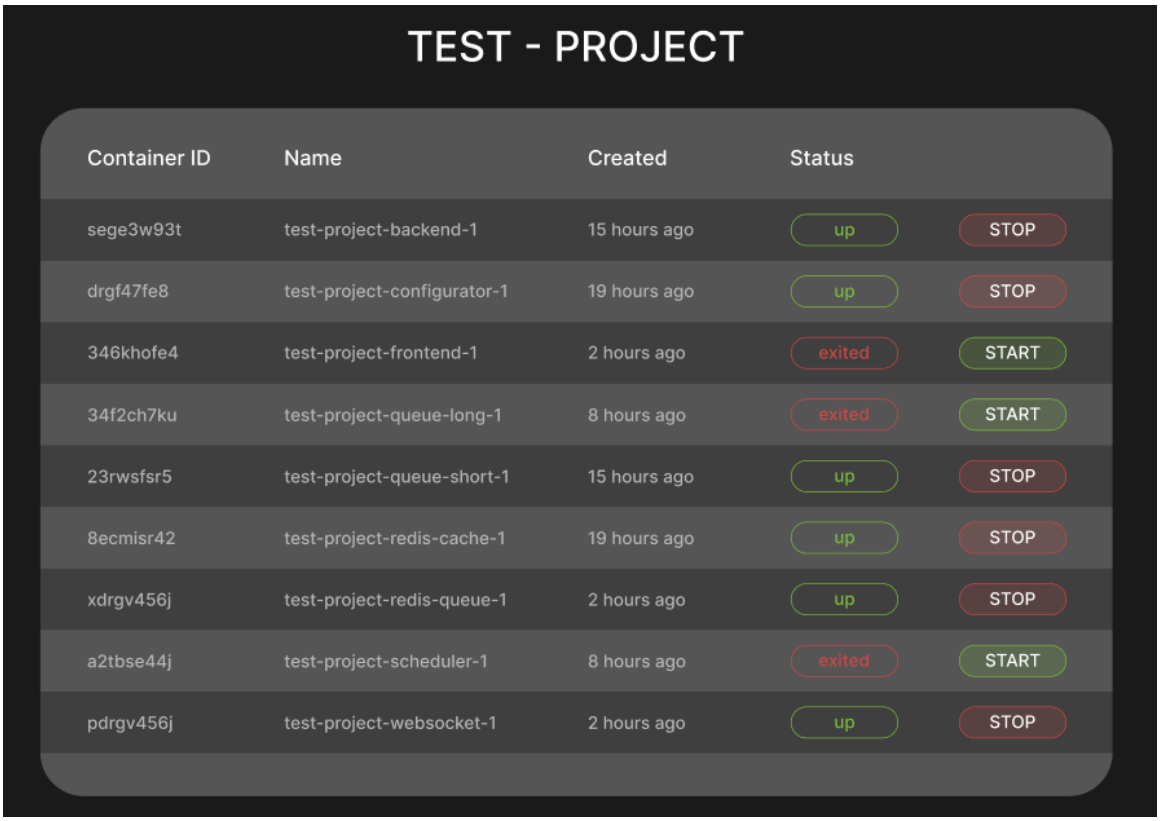

Рисунок 3.8 - Список Docker контейнерів, які належать певному екземпляру

Наступним елементом цієї сторінки є панель для керування проєктом. На панелі розташовані кнопки Start/Stop (в залежності від статусу проєкта), Backup (виконати процедуру, яка збереже поточний стан бази даних сайту) і Restore (виконати процедуру відновлення бази даних з обраного бекап-файлу). Після панелі керування знаходиться вікно з логами проєкту, для того, щоб слідкувати запити та проблеми на сайті. Панель керування та логи екземпляру зображені на рисунку 3.9

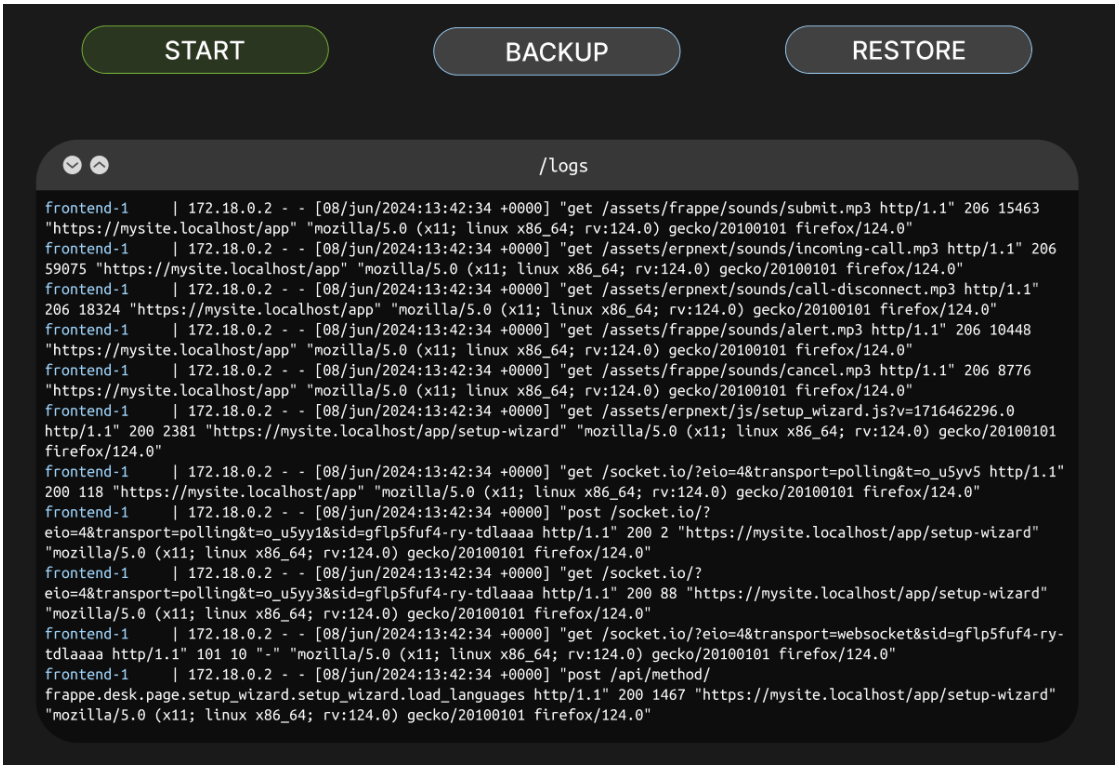

Рисунок 3.9 – Панель керування та логи екземпляру

3.5 Аналіз числових параметрів роботи програмного модулю розгортання серверної інфраструктури для додатку планування ресурсів підприємства

Під час роботи програмного модуля для розгортання серверної інфраструктури ERPNext було проведено аналіз використання ресурсів системи. Для порівняння, аналогічний аналіз було виконано для стандартного розгортання ERPNext на основі Frappe Cloud, сервісу, який надає розгортання ERPNext інстанцій на віддаленому сервері. Результати тестування наведено у табл. 3.1.

Таблиця 3.1 – Результати тестування запропонованого програмного модуля

Показник	Значення
Максимальний показник використання оперативної пам’яті	1536.78 МБ
Максимальний показник використання ресурсів процесора	32.47%

Таблиця 3.2 – Результати тестування стандартного розгортання ERPNext з Frappe Cloud

Показник	Значення
Максимальний показник використання оперативної пам’яті	2048.54 МБ
Максимальний показник використання ресурсів процесора	47.83%

Для доведення досягнення мети дослідження потрібно порівняти швидкість розгортання розробленого програмного модуля з показниками стандартного розгортання ERPNext з Frappe Cloud (див. табл. 3.2). Таке порівняння наведено у табл. 3.3.

Таблиця 3.3 – Порівняння швидкості розгортання розробленого програмного модуля та стандартного розгортання ERPNext

Показник	Стандартне розгортання ERPNext з Frappe Cloud	Програмний модуль
Середній час розгортання одного екземпляру	15 хв	5 хв

Із табл. 3.3 видно, що розроблений програмний модуль значно швидше розгортає інстанції ERPNext. Час розгортання зменшився на 10 хвилин (тобто на 40% порівняно зі стандартним розгортанням з Frappe Cloud). Тобто мета роботи - підвищення швидкості розгортання серверної інфраструктури ERPNext шляхом використання власних алгоритмів - досягнута.

3.6. Висновок

У цьому розділі детально розглянуто та описано основні компоненти програмної реалізації системи для розгортання серверної інфраструктури додатку керування ресурсами підприємства ERPNext, зокрема модуль керування інстанціями. Описано структуру та взаємодію між функціями.

API функції забезпечують зручний спосіб отримання ключових системних характеристик та показників продуктивності. Ці функції використовують потужні бібліотеки для збору даних про завантаженість CPU, системні ресурси та використання пам'яті, повертаючи інформацію у форматі JSON. Це дозволяє легко інтегрувати ці дані у веб-інтерфейс та забезпечити користувачів актуальною інформацією про стан системи.

Утиліти автоматизують процес розгортання нових інстанцій ERPNext. Ці функції виконують основні операції, включаючи клонування необхідних репозиторіїв, налаштування та запуск Docker контейнерів, конфігурацію мережі та створення нових сайтів ERPNext.

ВИСНОВОК

У роботі було створено програмний модуль для розгортання серверної інфраструктури ERPNext. Проаналізовано інструменти та стандартні методи розгортання ERPNext, зокрема Frappe Cloud. У другому розділі розроблено структуру, архітектуру модуля, інтерфейси користувача для управління інстанціями. У третьому розділі обґрунтовано вибір мов програмування та середовищ розробки, створено програмну реалізацію модуля. Аналіз показав, що модуль використовує на 25% менше оперативної пам’яті та на 32.1% менше ресурсів процесора порівняно зі стандартним розгортанням. Швидкість розгортання інстанції збільшилася на 40%.

Всі задачі, які були поставлені на початку виконання дипломної роботи, були виконані в повному об'ємі, а саме:

Обґрунтувано доцільность створення програмного модулю розгортання серверної інфраструктури для додатку планування ресурсів підприємства;

Проведено аналіз сучасних технологій для розгортання серверної інфраструктури;

Спроектовано структуру компонентів програмного модулю;

Програмно реалізовано модуль розгортання інфраструктури для додатку планування ресурсів підприємства;

Проведено тестування програмного модулю розгортання серверної інфраструктури для додатку планування ресурсів підприємства.

Тобто мета роботи - підвищення швидкості розгортання досягнута у порівнянні з аналогічними рішеннями по розробці програмних продуктів для розгортання серверної інфраструктури ERP систем. ас розгортання зменшено з 15 до 5 хв.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. 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: 20.05.2024). – Назва з екрана.

2. Kubernetes Documentation [Електронний ресурс] – Режим доступу: https://kubernetes.io/docs/home/ (дата звернення: 20.05.2024). – Назва з екрана.

3. Turnbull J. The Docker Book: Containerization is the new virtualization [Електронний ресурс] / James Turnbull // James Turnbull. – 2014. – Режим доступу: https://dockerbook.com (дата звернення: 20.05.2024). – Назва з екрана.

4. 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: 20.05.2024). – Назва з екрана.

5. 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: 20.05.2024). – Назва з екрана.

6. What is containerization? - AWS [Електронний ресурс] – Режим доступу: https://aws.amazon.com/what-is/containerization/#:~:text=Containerization%20is%20a%20software%20deployment,matched%20your%20machine's%20operating%20system. (дата звернення: 20.05.2024). – Назва з екрана.

7. Ansible Documentation [Електронний ресурс] – Режим доступу: https://docs.ansible.com/ansible/latest/index.html (дата звернення: 20.05.2024). – Назва з екрана.

8. 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: 20.05.2024). – Назва з екрана.

9. Understanding virtualization [Електронний ресурс] / Red Hat, Inc. – 2021. – Режим доступу: https://www.redhat.com/en/topics/virtualization (дата звернення: 20.05.2024). – Назва з екрана.

10. 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: 20.05.2024). – Назва з екрана.

11. Frappe Cloud – The Official Cloud Platform for Frappe Apps [Електронний ресурс] – Режим доступу: https://frappecloud.com/

12. Django documentation [ЕлеДктронний ресурс] – Режим доступу: https://docs.djangoproject.com/en/5.0/

13. The Python Standard Library — Python 3.12.3 documentation [Електронний ресурс] – Режим доступу: https://docs.python.org/3/library/index.html

14. Python on whales [Електронний ресурс] – Режим доступу: https://gabrieldemarmiesse.github.io/python-on-whales/

15. JavaScript – MDN Web Docs – Mozilla [Електронний ресурс] – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/JavaScript

16. Develop with Docker Engine API [Електронний ресурс] – Режим доступу: https://docs.docker.com/engine/api/

17. How to Build an API in Python [Електронний ресурс] – Режим доступу: https://blog.postman.com/how-to-build-an-api-in-python/

18. os — Miscellaneous operating system interfaces [Електронний ресурс] – Режим доступу: https://docs.python.org/3/library/os.html

19. psutil documentation — psutil 6.0.0 documentation [Електронний ресурс] – Режим доступу: https://psutil.readthedocs.io/en/latest/

20. HTML: HyperText Markup Language – MDN Web Docs [Електронний ресурс] – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/HTML

21. CSS: Cascading Style Sheets - MDN Web Docs [Електронний ресурс] – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/CSS

22. 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

Додаток А (обов'язковий)

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Програмний модуль розгортання серверної інфраструктури для додатку планування ресурсів підприємства

Тип роботи: бакалаврська дипломна робота
(БДР, МКР)

Підрозділ кафедра комп'ютерних наук, ФІТА
(кафедра, факультет)

Показники звіту подібності Unicheck

Оригінальність 98,92% Схожість 1,08%

Аналіз звіту подібності (відмітити потрібне):

✓ Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.

□ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.

□ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку Озеранський В.С.

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи Заяц В.В.

Керівник роботи Сілагін О.В.

Додаток Б (обов’язковий)
Лістинг 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()
```

Додаток В (обов'язковий)

ІЛЮСТРАТИВНА ЧАСТИНА

« ПРОГРАМНИЙ МОДУЛЬ РОЗГОРТАННЯ СЕРВЕРНОЇ ІНФРАСТРУКТУРИ ДЛЯ ДОДАТКУ ПЛАНУВАННЯ РЕСУРСІВ ПІДПРИЄМСТВА»

Виконав: студент 4 курсу,групи 2КН-20б
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Заяц В.В
(прізвище та ініціали)

Керівник: к.т.н., доцент каф.КН
Сілагін О.В.
(прізвище та ініціали)
«07» 02 2024 р.

Вінниця ВНТУ - 2024 рік

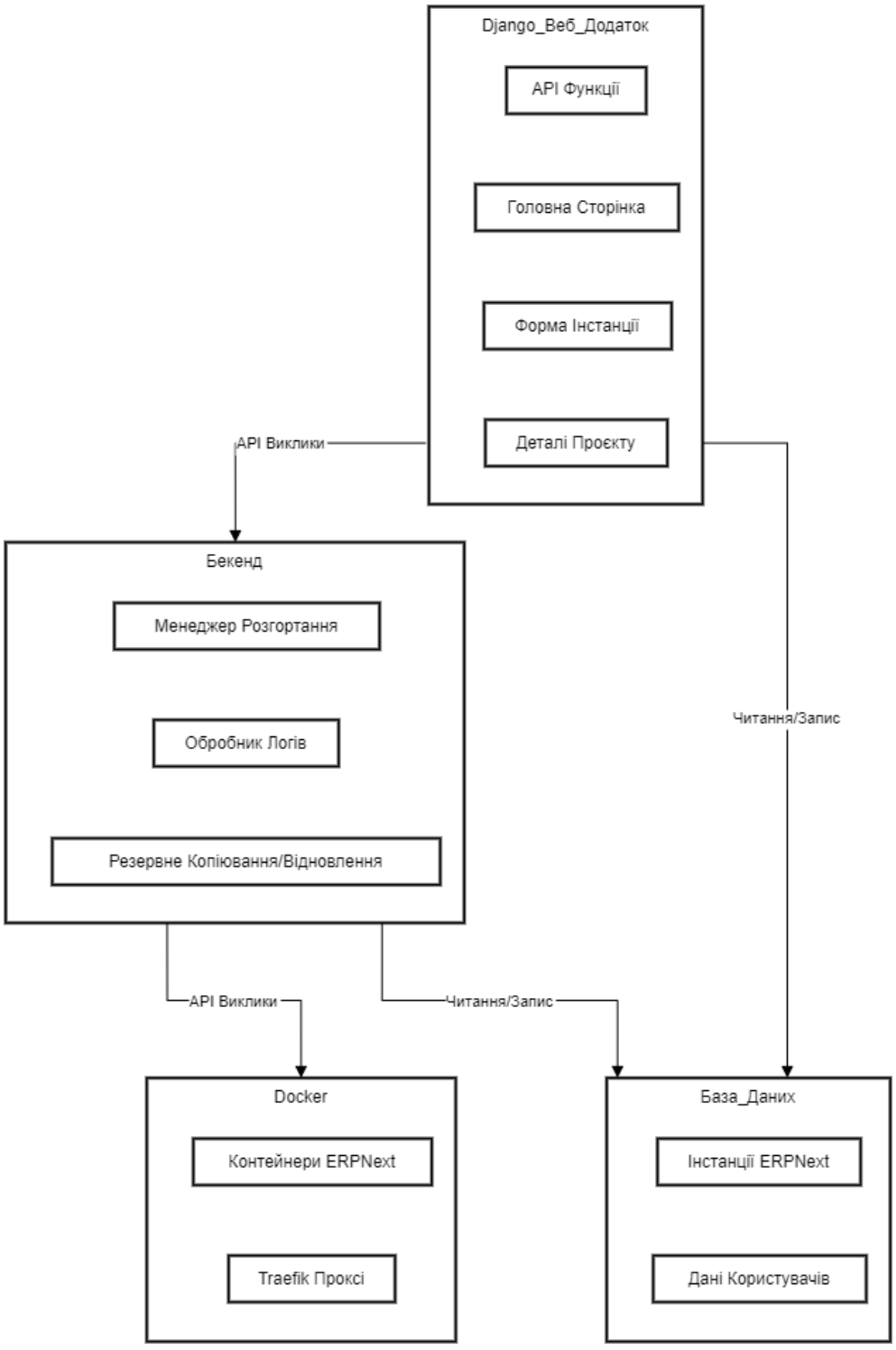

Рисунок В.1 – UML-діаграма компонентів програмного модулю розгортання серверної інфраструктури для додатку планування ресурсів підприємтсва

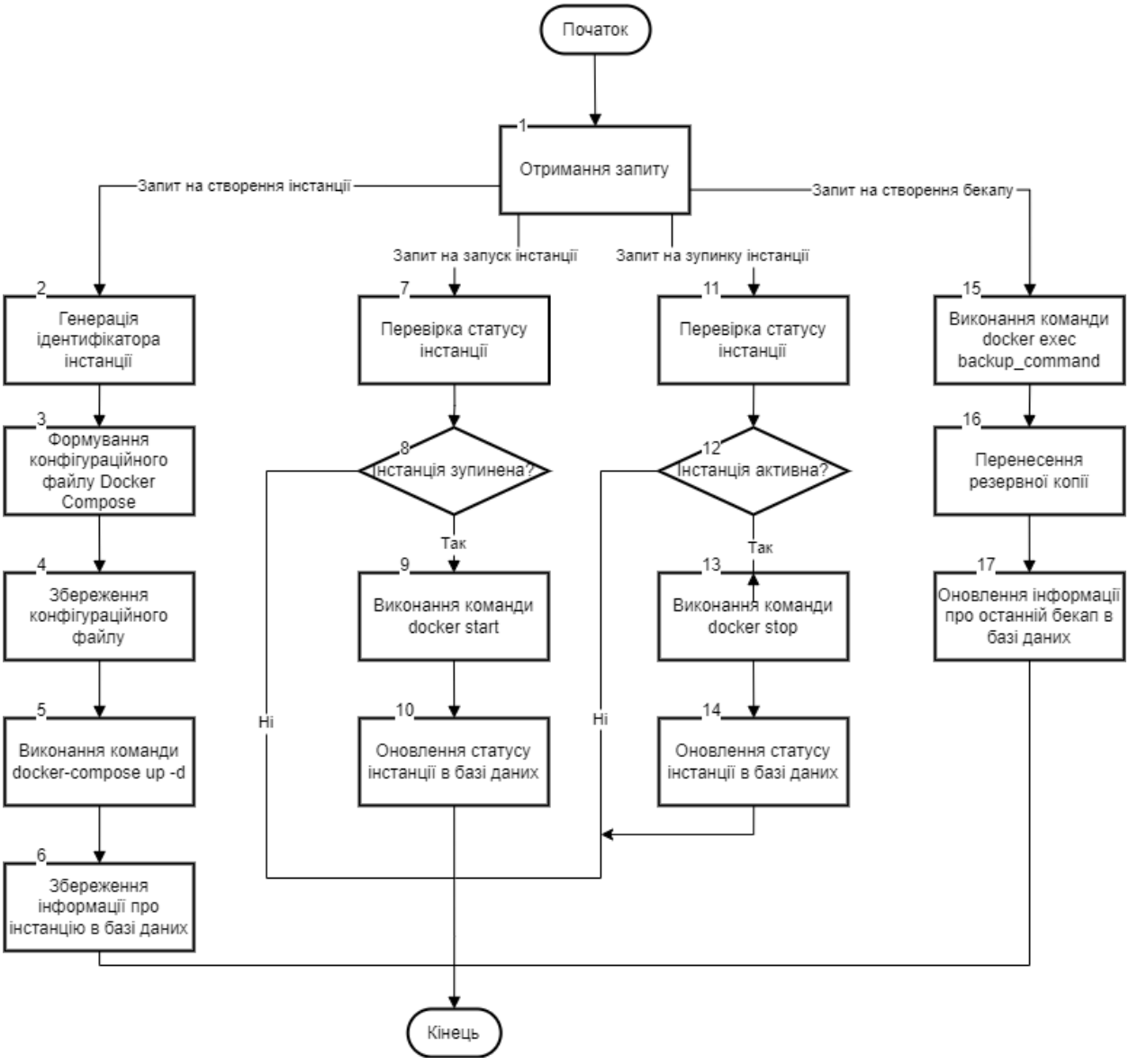

Рисунок В.2 – Алгоритм роботи програмного модулю розгортання серверної інфраструктури для додатку планування ресурсів підприємства

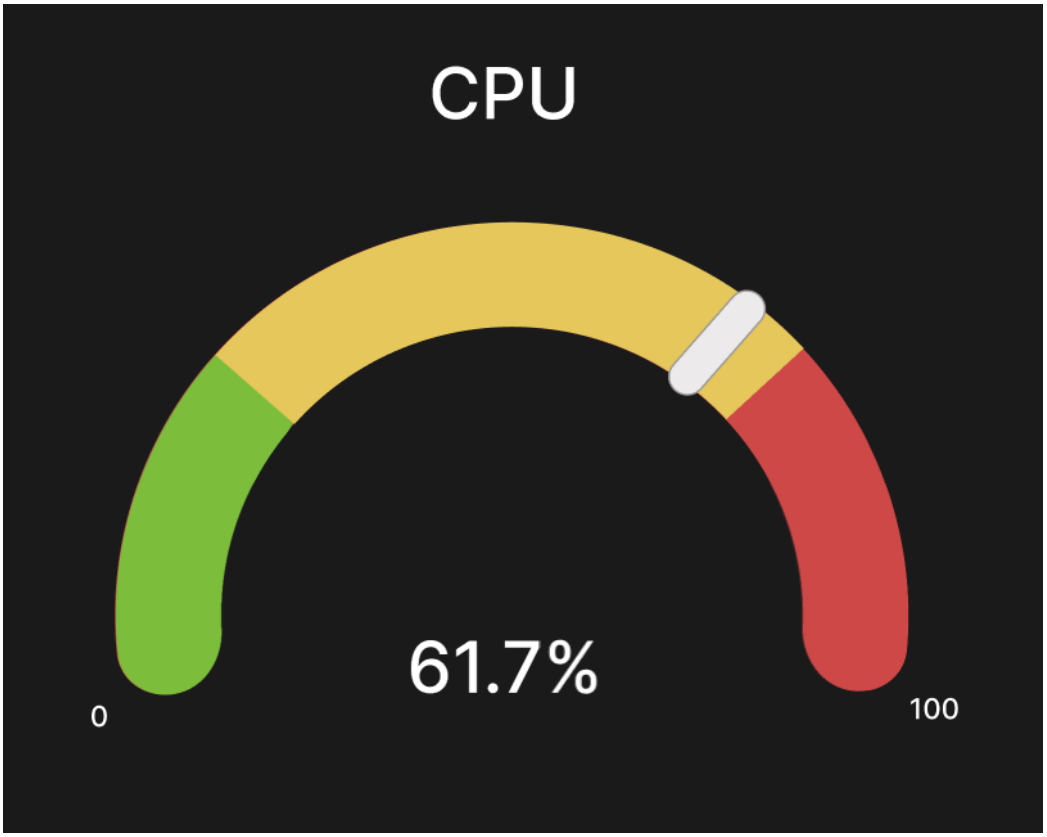

Рисунок В.3 – Показник завантаження процесора

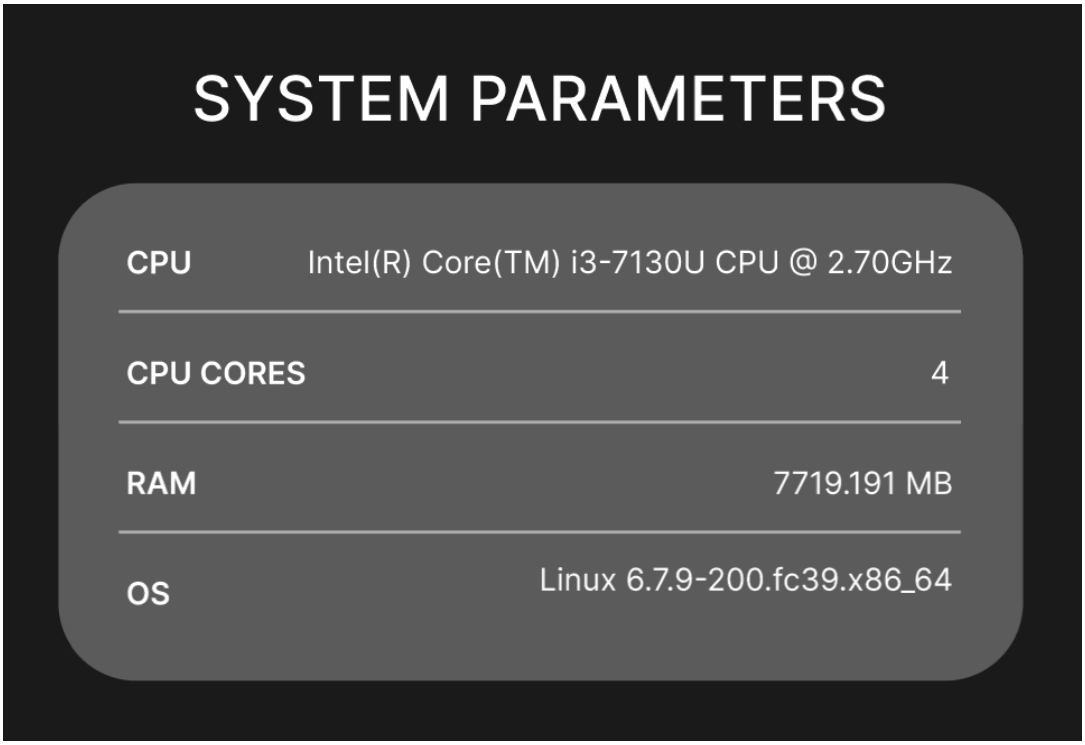

Рисунок В.4 – Таблиця системних параметрів

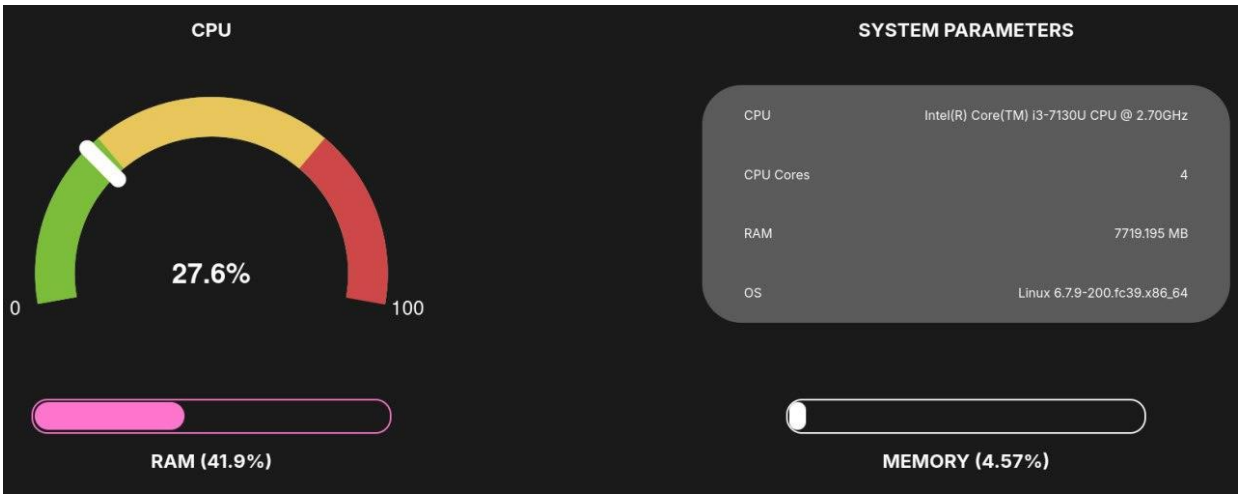

Рисунок В.5 – Системні характеристики середовища

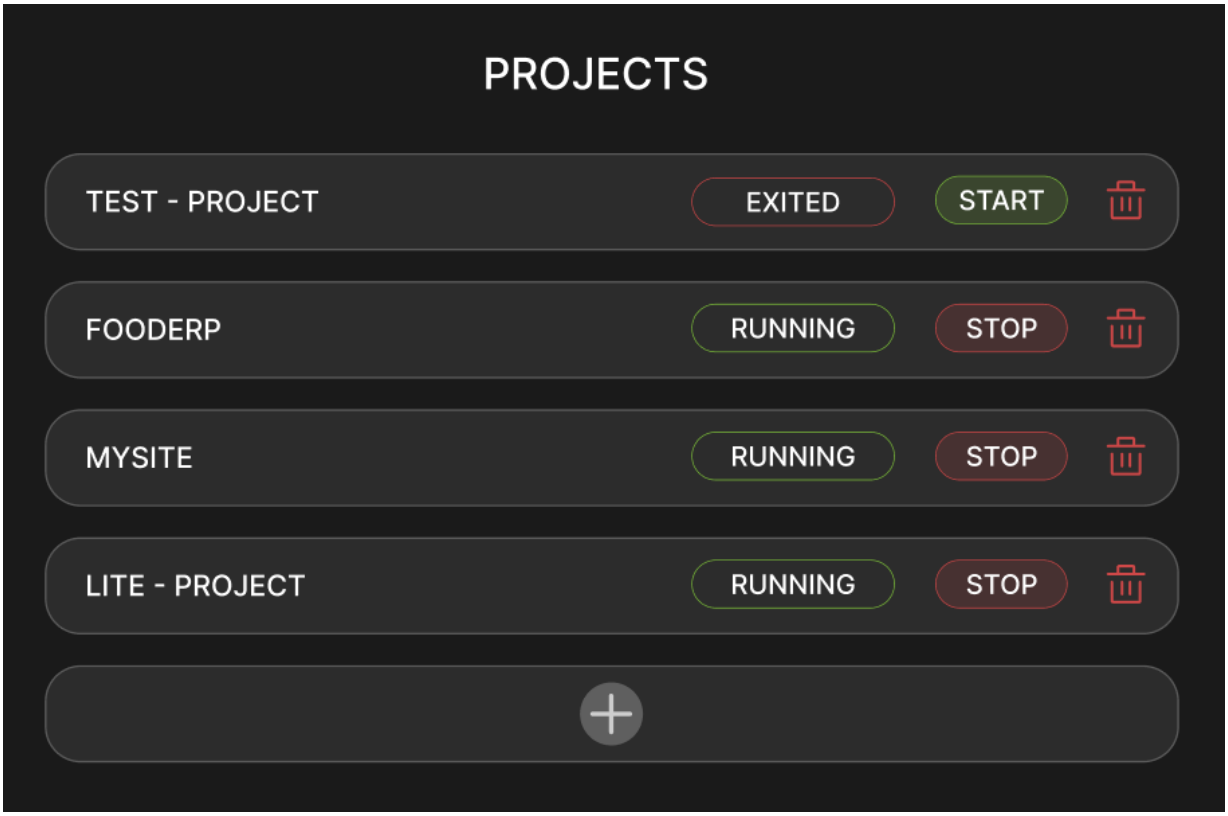

Рисунок В.6 – Список всіх Docker Compose проєктів

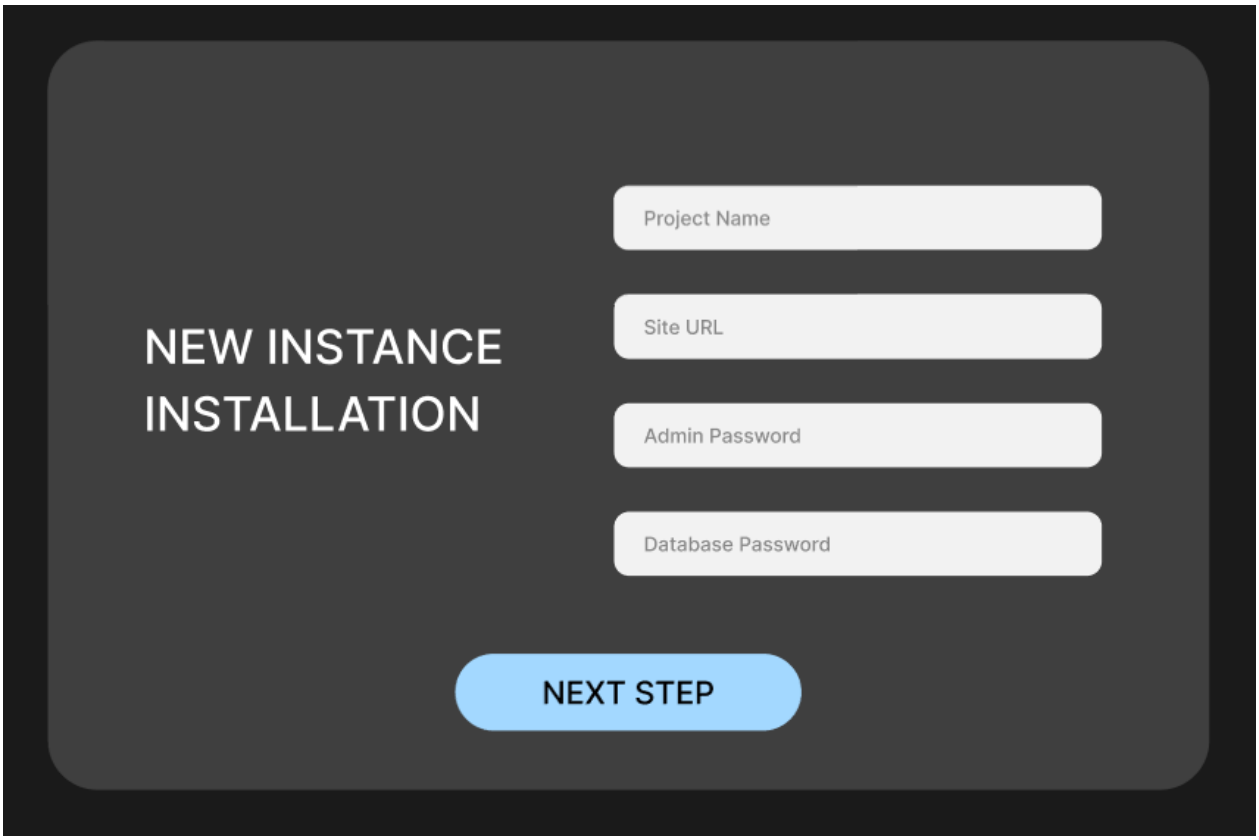

Рисунок В.7 - Форма додавання нового екземпляру для розгортання

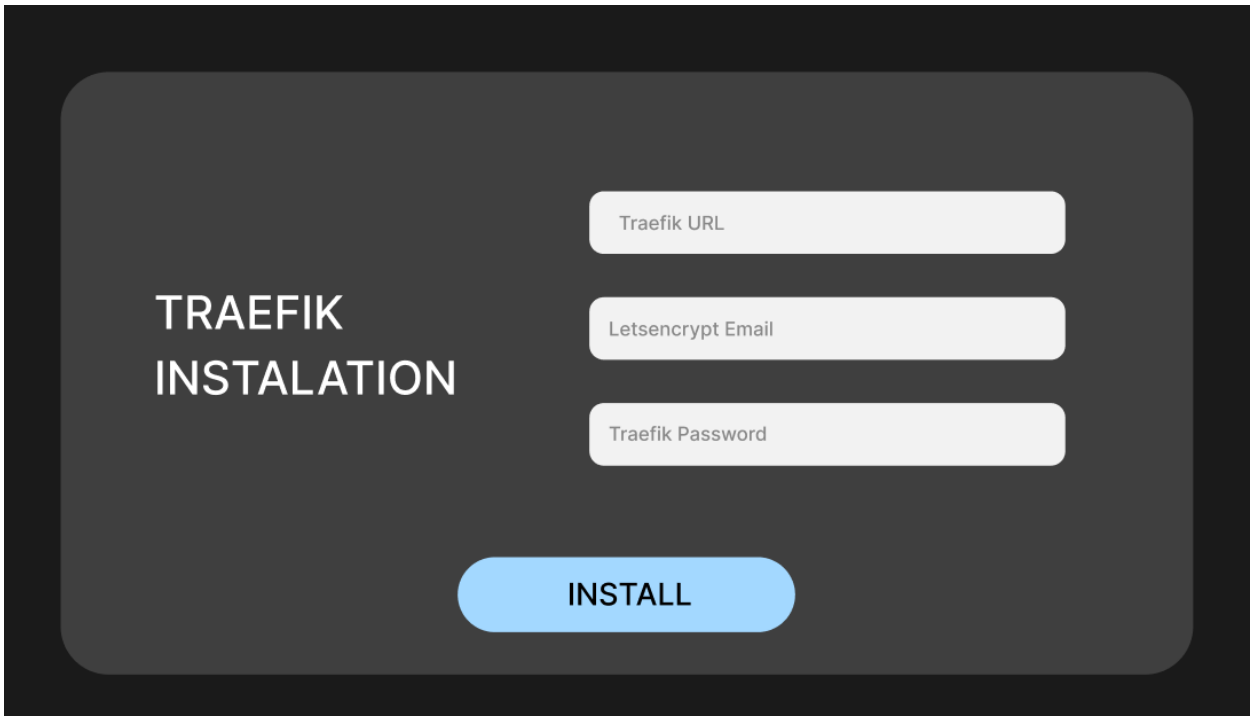

Рисунок В.8 - Форма інсталяції проксі-сервера Traefik

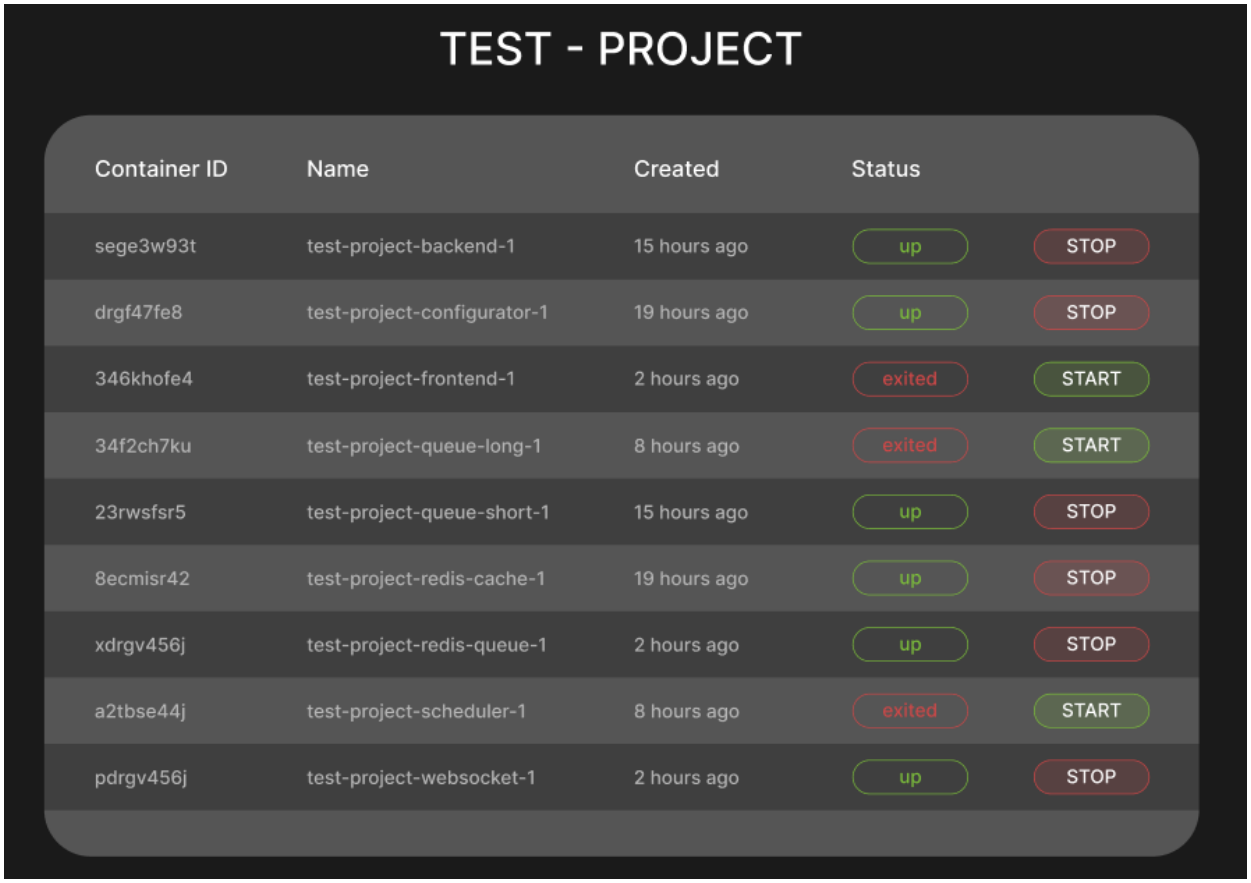

Рисунок В.9 - Список Docker контейнерів, які належать певному екземпляру

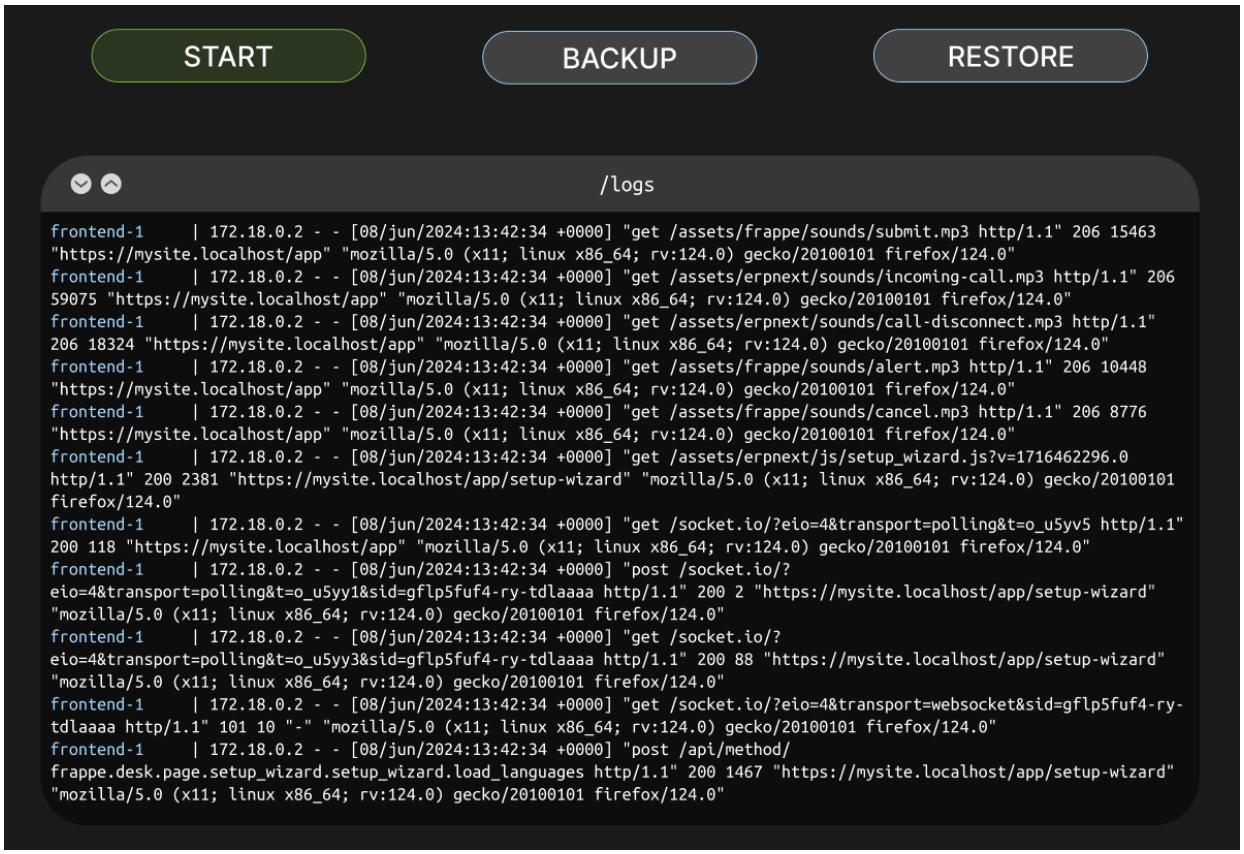

Рисунок В.10 – Панель керування та логи екземпляру

Додаток Г (довідниковий)
Інструкція користувача

Для початку роботи з програмним модулем необхідно мати встановленим Python та Django Framework зі всіма описаними бібліотеками, завантажити архів та запустити Django проєкт. Коли програмний модуль запустився побачимо першу сторінку сайту.

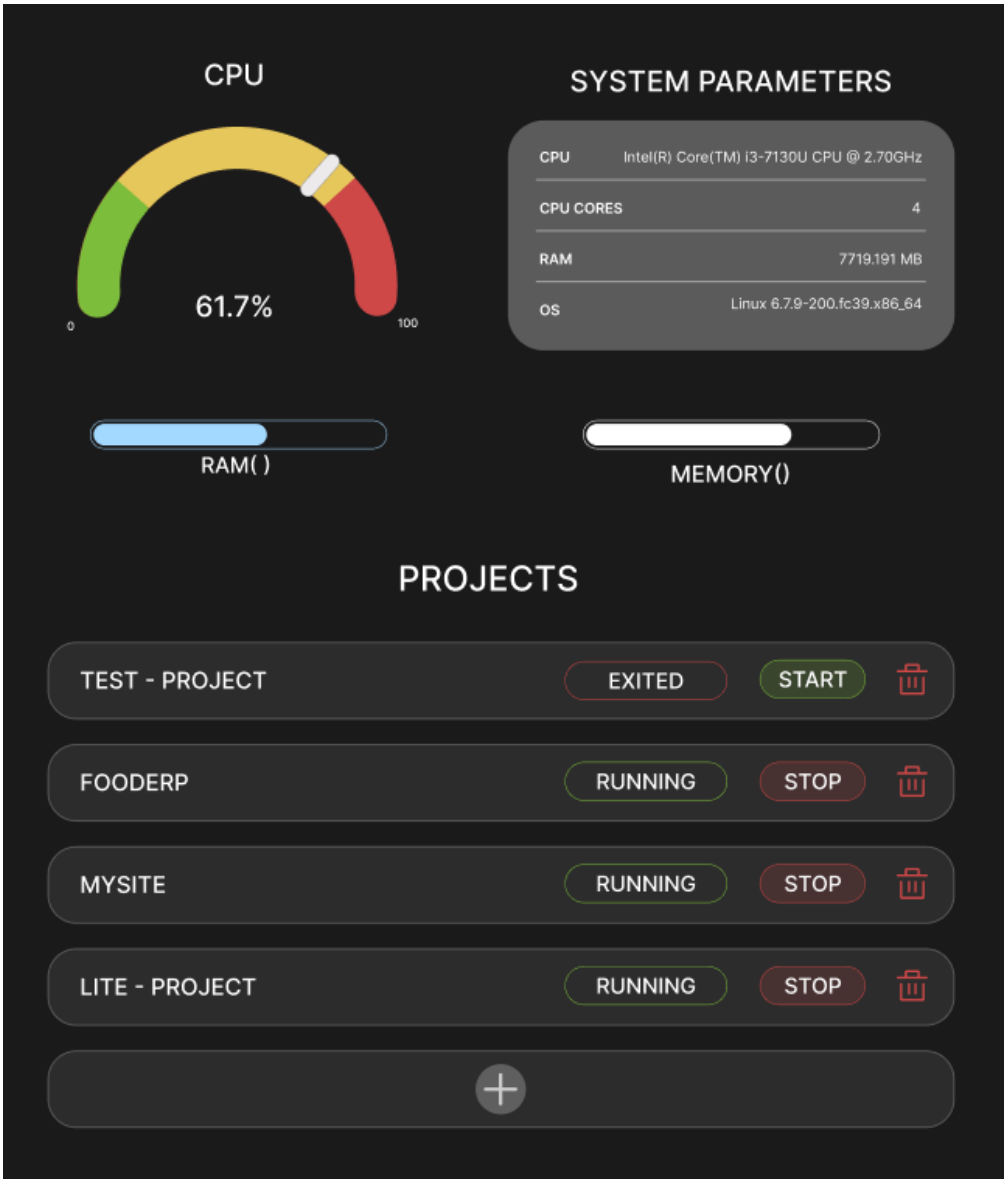

Рисунок Г.1 – Стартова сторінка з системними параметрами і проєктами

Щоб отримати доступ до керування проєктом (контейннри, панель керування, логи) натискаємо на проєкт.

Рисунок Г.2 – Сторінка керування проєктом

Щоб створити новий проєкт натискаємо на блок «+» на стартовій сторінці. Користувач побачить форму для введення даних нового проєкта. Якщо до моменту заповнення даних не встановлено Traefik, то користувач перейде на форму для створення Traefik.

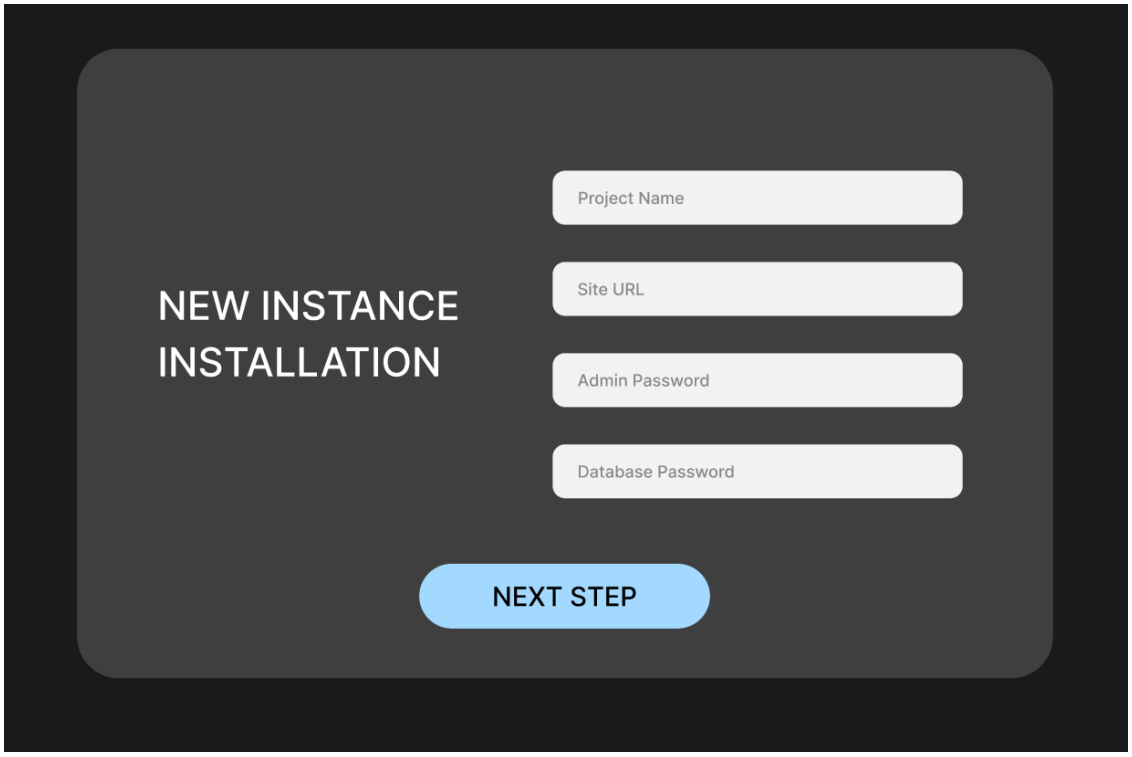

Рисунок Г.3 – Форма для розгортання нового проєкту

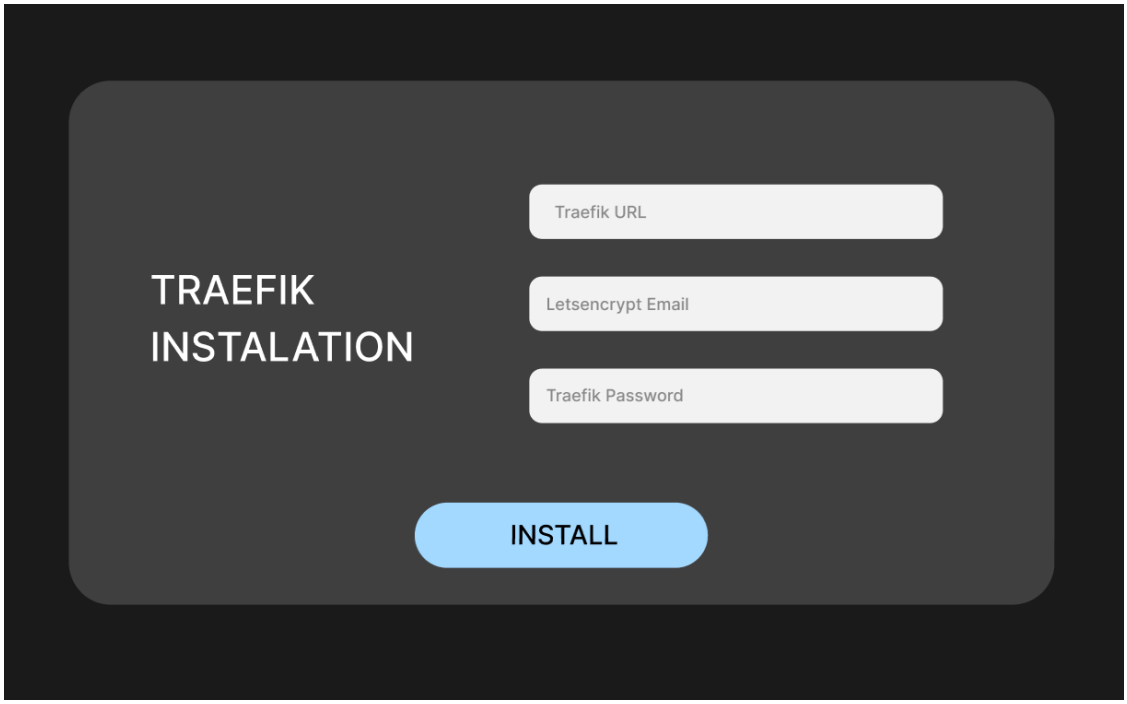

Рисунок Г.4 – Форма для розгортання Traefik

Додаток Д
Акт про впровадження

Затверджую

Директор ТОВ „ІКРОК"

Кашеєв А.А.

«07» червня 2024 р

АКТ

Впровадження результатів бакалаврської кваліфікаційної роботи

Зайця Владислава Валерійовича на тему:

«Програмного модуля розгортання серверної інфраструктури для додатку планування ресурсів підприємства»

Результати бакалаврської кваліфікаційної роботи, а саме алгоритми, скрипти для розгортання серверної інфраструктури.

було впроваджено/було використано в проектах пов'язаних з планування ресурсів підприємства.

Члени комісії: _____ _____

_____ _____

Товариство з обмеженою відповідальністю "ІКРОК" Ідент. код 38135356 Україна м.Вінниця