МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра інформаційних технологій

«До захисту допущено»
В.о. завідувача кафедри

_____ Світлана ВАЩЕНКО

____ _____ 2024 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня магістр

зі спеціальності 122 «Комп'ютерні науки»_____,

освітньо-наукової програми «Інформаційні технології проектування»_____

на тему: Інформаційна технологія забезпечення робастності вебдодатків

Здобувача групи ІТ.м-21н Молчанова Дмитра Андрійовича
(шифр групи) (прізвище, ім'я, по батькові)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

_____ Д.А. Молчанов
(підпис) (Ім'я та ПРІЗВИЩЕ здобувача)

Керівник к.т.н., доц. Федотова Н.А. _____
(посада, науковий ступінь, вчене звання, Ім'я та ПРІЗВИЩЕ) (підпис)

Суми – 2024

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра інформаційних технологій
Спеціальність 122 «Комп'ютерні науки»
Освітньо-професійна програма «Інформаційні технології проектування»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри ІТ

Світлана ВАЩЕНКО
«_____» _____ 2024 р.

ЗАВДАННЯ

на кваліфікаційну роботу магістра студентові

Молчанову Дмитру Андрійовичу

(прізвище, ім'я, по батькові)

1 Тема кваліфікаційної роботи Інформаційна технологія забезпечення робастності вебдодатків

затверджена наказом по університету від «01» лютого 2024 р. № 0096-VI

2 Термін здачі студентом кваліфікаційної роботи «10» ___травня___ 2024 р.

3 Вхідні дані до кваліфікаційної роботи літературні джерела з проблема забезпечення життєздатності вебдодатків, проекти для демонстрації підтримки

4 Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) аналіз предметної області, постановка задачі, методи дослідження, проектування інформаційної технології забезпечення робастності вебдодатків, розробка інформаційної технології

5 Перелік графічного матеріалу (з точним зазначенням обов'язкових слайдів презентації) актуальність, мета та задачі, задачі дослідження, огляд існуючих аналогів, результати проведеного аналізу аналогів, функціональні вимоги, засоби реалізації, структурно-функціональне моделювання, моделювання варіантів використання, архітектура інформаційної технології, схема реалізованої бази даних засобами PostrgeSQL, демонстрація роботи програмного продукту, апробація, висновок.

6. Консультанти випускної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

Дата видачі завдання _____.

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Підготовка специфікації	11.09.23 – 09.11.23	
2	Розробка інформаційної технології забезпечення робастності вебдодатків	10.11.23 – 15.02.24	
3	Тестування	16.02.24 – 15.03.24	
4	Впровадження в дію	18.03.24 – 05.04.24	
5	Налаштування правильної роботи інформаційної технології забезпечення робастності вебдодатків	08.04.24 – 05.05.24	

Магістрант _____ Дмитро МОЛЧАНОВ

Керівник роботи _____ к.т.н., доц. Наталія ФЕДОТОВА

АНОТАЦІЯ

Тема кваліфікаційної роботи магістра – «Інформаційна технологія забезпечення робастності вебдодатків».

Склад пояснювальної записки:

- вступ;
- 4 основних розділи;
- висновок;
- список використаних джерел, який налічує 39 посилань;
- 2 додатки.

Загальний обсяг магістерської роботи налічує 99 сторінок, з яких 61 сторінок було відведено основному тексту, 5 сторінок – списку використаних джерел, 27 сторінок виділено для додатків А та Б.

Темою магістерської дисертації було обрано створення інформаційної технології для забезпечення робастності вебдодатків. Під час роботи було проаналізовано предметну область, визначено мету й актуальність розробки. Було обрано відповідні технології для розробки, поставлені цілі проєкту, визначено основні завдання й описано використану методологію дослідження,. Робота включала в себе процес проектування системи на різних етапах з урахуванням потреб бізнесу і можливих сценаріїв застосування програмного продукту. Було підготовлено програмні компоненти для подальшої демонстрації.

Результатом виконаного дослідження стала створена інформаційна технологія забезпечення робастності вебдодатків. Використання даної системи сприятиме покращенню процесу управління проектами та співробітниками, завдяки автоматизації процесу розподілу завдань поміж персоналу.

Використання розробленої інформаційної системи сприятиме підвищенню ефективності як приватних вебдодатків, так і складних інфраструктур, контролюючи навантаження на робочу частину системи. Як

результат, це призведе до зменшення рівню стресу в команді та покращить надійність системи в цілому.

Ключові слова: AUTOMATION, KUBERNETES, CONTAINER, DOCKER, AUTOHEALING, RESOURCE MANAGEMENT, AUTOSCAILING.

ЗМІСТ

ВСТУП

Актуальність.

Інформаційні технології знаходяться в постійному стані розвитку, а споживачі сьогодення вимагають від вебдодатків не лише високої швидкості та функціональності, але й максимальної надійності та стійкості до різних викликів. У цьому контексті, забезпечення робастності вебдодатків, тобто їх здатність працювати ефективно та надійно в умовах різноманітних збоїв і атак, стає надзвичайно важливою задачею для розробників та системних адміністраторів.

Інформаційні системи невпинно стають складнішими й вимагають більше ресурсів, зусиль та координації. Використання сучасних інструментів та підходів стає ключовим елементом у забезпеченні робастності вебдодатків. Kubernetes, як масштабована та гнучка платформа для оркестрації контейнеризованих додатків, разом з принципами DevOps, що спрямовані на автоматизацію процесів розробки, впровадження та експлуатації, відіграють важливу роль у цьому процесі.

Актуальність дослідження цієї проблематики обумовлена не лише зростанням обсягів та складності вебдодатків, але й зростанням загроз для їх безпеки та надійності. Спостерігається збільшення числа кібератак, які спрямовані на вебдодатки з метою витоку конфіденційної інформації, втрати доступу до даних чи навіть збоїв у роботі системи. Відсутність адекватних заходів забезпечення безпеки та робастності може призвести до серйозних фінансових втрат, порушення репутації та втрати довіри користувачів.

Враховуючи ситуацію в окремих європейських країнах, а саме Німеччину, де використання сучасних систем для підтримки життєдіяльності не є популярним, можна побачити на живому прикладі як відсутність підтримки впливає на ведення бізнесу. Деякі бюрократичні алгоритми (зокрема робота міграційної служби, банкова система тощо) були перенесені в онлайн, але не

забезпечені необхідною підтримкою. Це призводить до періодичного виходу з ладу цілих систем на довгий період (до місяця).

Отже, розробка інформаційної технології забезпечення робастності вебдодатків є актуальною, а використання систем оркестрації та практик DevOps – необхідними.

Тема дослідження. Інформаційна технологія забезпечення робастності вебдодатків.

Об'єкт дослідження. Алгоритм налаштування інформаційної системи з використанням інструментарію Kubernetes, задля забезпечення її неперервної та стабільної роботи.

Предмет дослідження. Інформаційна технологія забезпечення робастності вебдодатків.

Наукова новизна даного дослідження полягає у впровадженні інформаційної технології забезпечення робастності вебдодатків з використанням інструментарію Kubernetes. Цей підхід є відмінним від традиційних методів розробки та підтримки вебдодатків, оскільки він базується на використанні контейнеризації та автоматизації розгортання та управління.

Мета. Розробка інформаційної технології забезпечення робастності вебдодатків.

Для досягнення мети проекту необхідно виконати наступні задачі:

- аналіз проблемної області;
- огляд сучасних публікацій;
- визначення актуальності та цільової аудиторії використання розробленої технології;
- аналіз аналогів інформаційних систем;
- визначення технологій розробки інформаційної системи;
- проектування алгоритму налаштування інформаційної системи;
- реалізація структури і функціоналу інформаційної технології;
- тестування алгоритму.

Практична цінність. Оптимізація процесу розробки, тестування та підтримки вебдодатків або цілих інформаційних систем, з використанням практик можливостей Kubernetes по автоматизації розгортання, масштабування та управління контейнеризованими системами. Введення в дію розробленої інформаційної технології потенційно посприяє підвищенню продуктивності роботи всередині команди розробки, тестування та підтримки. Це допоможе мінімізувати перевантаження співробітників та уникнути неочікуваних ситуацій, які несуть загрозу життєдіяльності системи, і в результаті знизити рівень стресу як для замовника, так і для розробників.

Гіпотеза дослідження. Впровадження інформаційної технології забезпечення робастності вебдодатків може значно підвищити ефективність підтримки проектів, забезпечуючи постійний нагляд за станом вебдодатку та оптимізацію використання ресурсів. Дослідження в галузі використання цієї технології дають змогу виявити найкращі способи впровадження та підтримки методології DevOps, що потенційно підвищить продуктивність і сприятиме успішному завершенню проекту.

1 Аналіз предметної області

1.1 Огляд останніх досліджень і публікацій

Робастність вебдодатків стала однією з найактуальніших і найважливіших тем у сфері інформаційної безпеки. Вебдодатки повсюдно використовуються для обробки конфіденційних даних, таких як особиста інформація користувачів, банківські дані та комерційні таємниці. У таких операціях порушення життєздатності вебдодатків може призвести до витоку даних, порушення конфіденційності і навіть втрати репутації компанії. Тому забезпечення безпеки веб-застосунків є критично важливим аспектом [1].

Kubernetes – це платформа з відкритим вихідним кодом для управління контейнеризованими робочими навантаженнями та супутніми службами. Згідно з [2] її основні характеристики – кросплатформеність, розширюваність, успішне використання декларативної конфігурації та автоматизації. Вона має гігантську, швидкопрогресуючу екосистему.

DevOps – це набір практик та принципів, які покращують продуктивність розробки програмного забезпечення. DevOps скорочує життєвий цикл розробки програмного забезпечення (SDLC), щоб виробляти програмне забезпечення високої якості зі стабільною та послідовною доставкою коду. DevOps стоїть за ‘розробка’ та ‘операції’, маючи на меті покращити швидкість розробки програмних застосунків.

Використання Kubernetes та DevOps може принести ряд переваг, таких як:

– **Немає простою часу для розгортання:** Kubernetes дозволяє розгортати застосунки без простою часу;

– **Масштабованість:** Kubernetes дозволяє легко масштабувати застосунки відповідно до потреб;

– **Інфраструктура та конфігурація як код:** Kubernetes та DevOps дозволяють керувати інфраструктурою та конфігурацією як кодом, що полегшує автоматизацію та забезпечує консистентність;

– **Співпраця між функціональними групами:** Kubernetes та DevOps сприяють співпраці між розробниками, операторами та іншими учасниками команди.

Використання Kubernetes та DevOps може значно покращити робастність вебдодатків, забезпечуючи стабільність, масштабованість та безпеку [3].

У статті "7 Reasons Kubernetes Is Important for DevOps" [3] розглянуто важливість Kubernetes для DevOps. Автори наголошують на перевагах Kubernetes, таких як немає простою часу для розгортання, масштабованість, інфраструктура та конфігурація як код, а також співпраця між функціональними групами.

У статті "How Kubernetes is Transforming DevOps and 6 Best Practices" розглянуто та надано рекомендації, як Kubernetes трансформує DevOps, і наведено 6 кращих практик:

– Впровадження робочих процесів на основі Git (GitOps);

– Використання **Blue/Green** або **Canary** схем розгортання;

– Реліз та підтримка контейнерів, ідентичних протестованим;

– Зберігання та шифрування секретів поза контейнерами (наприклад, в репозиторії GitHub);

– Скан та тестування образів контейнерів на предмет потенційних вразливостей;

– Використання Infrastructure as a Code (IaC) для автоматичного налаштування IT-інфраструктури.

Автори обговорюють, як Kubernetes допомагає управляти великою кількістю контейнерів як частиною кластера, автоматизуючи багато аспектів їхнього життєвого циклу [4].

У статті "Best of 2021 – DevOps and Kubernetes: A Perfect Match?" розглянуто, як DevOps та Kubernetes можуть ідеально поєднуватися[5]. Автори

вказують, що DevOps – це стратегія розробки програмного забезпечення, яка об’єднує команди розробки та експлуатації в одну цілісну одиницю. Kubernetes, з іншого боку, – це платформа з відкритим вихідним кодом, розроблена для управління контейнерним розгортанням в масштабі.

Автор намагається дослідити взаємозв’язки між корпоративним DevOps, гнучкою культурою, роллю контейнерів у CI/CD конвеєрах та інтеграцією Kubernetes у конвеєр DevOps. Також в тексті розглянуто проблеми, які виникають при роботі команд розробки та експлуатації в ізольованих умовах, і як DevOps може допомогти вирішити деякі з цих проблем, такі як нерозуміння процесів іншою командою. Однак, підкреслено, що культурних змін, які вимагає DevOps, не достатньо, щоб подолати всі проблеми, які існують у командах, що працюють в ізольованих умовах.

Стаття також розглядає роль контейнерів у CI/CD на рівні підприємства та як Kubernetes може допомогти в переході інфраструктури до публічних хмар, таких як Azure або AWS.

1.2 Аналіз інформаційних систем

Сучасні реалії IT-бізнесу призводять до постійного збільшення масштабів інформаційних систем, що в свою чергу створює потребу в додатковому налагодженні основних процесів життєвого циклу ПЗ. Такі важливі речі, як планування та керування ресурсами потребують постійного моніторингу та втручання, які тільки ускладнюють та збільшують вартість підтримки наявних додатків.

Для таких випадків були створені інформаційні рішення, які допомагають спрощували планування змін використання ресурсів, щоб запобігти не раціональному використанню ресурсів серверів. Такі системи не є розповсюдженими серед IT-компаній, але стрімко набирають популярність за рахунок свого потенціалу та широкого ком'юніті.

В цілях порівняння було відібрано три інформаційні системи оркестрації контейнерів: «Docker Swarm», «RedHat OpenShift» та «Rancher».

Docker swarm – абстракція Docker, що стирає межі між різними машинами. Аналогічний Docker engine, але працюючий у кластері (рис. 1.1). Кілька контейнерів-воркерів об'єднуються в сервіс, обслуговує який Swarm менеджер. Swarm менеджер розміщує контейнери з додатком на вільний хостах і приймає команди на управління кластером. Також Swarm працює як балансувальник навантаження між кількома "воркерами", рівномірно розподіляючи запити, що надходять з будь-якої зі сторін кластера [6].

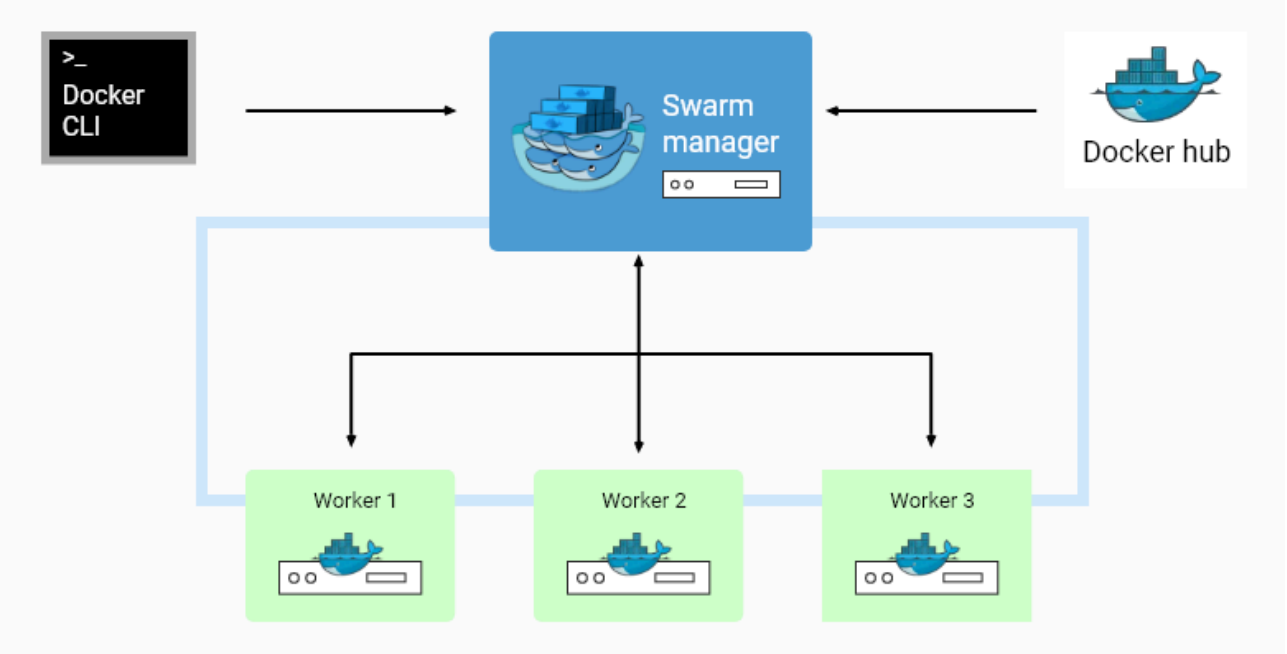


Рисунок 1.1 – Схематичний принцип дії Docker Swarm

Джерело: [6]

Серед основних об’єктів додатку можна виділити такі:

– Manager – управляючий вузол (управляюча нода), яка розподіляє задачі між Worker-нодами;

– Worker – нода, яка виконує визначені задачі;

– Docker CLI – командний інтерфейс для взаємодії користувача з Swarm;

– Docker Hub – репозиторій готових образів.

Плюсами використання Docker Swarm є його відносна простота роботи та висока швидкість освоєння для тих, хто вже мав досвід роботи з Docker-контейнерами. Також важливим аспектом є те, що Docker Swarm вже вбудований в Docker Engine, тому користувачу не потрібно встановлювати додаткове програмне забезпечення.

Найголовнішими мінусами виступають менш гнучкий функціонал, у порівнянні з іншими додатками, та необхідність мати, як мінімум базові, навички працювати з Docker.

Наступна платформа для розробки та запуску контейнеризованих додатків – **OpenShift Container Platform** [7]. Ця платформа призначена для того, щоб дозволити додаткам та дата-центрам, які їх підтримують, розширюватися від кількох до тисяч машин, які обслуговують мільйони клієнтів.

OpenShift Container Platform працює на основі Kubernetes, тому вона використовує аналогічну технологію, яка дозволяє розробникам та IT-організаціям надавати хмарні платформи для розгортання додатків на безпечних та масштабованих ресурсах. Ця система вимагає мінімальної конфігурації та управління.

Container Platform використовує так звану “архітектуру мікросервісів”, що означає, що великі додатки розбиваються на менші частини, які можуть працювати незалежно одна від одної (рис. 1.2). Всі дані про ці мікросервіси зберігаються в надійному сховищі, названому etcd. Цими мікросервісами можна керувати за допомогою REST API, які відкриваються на кожному з основних об’єктів.

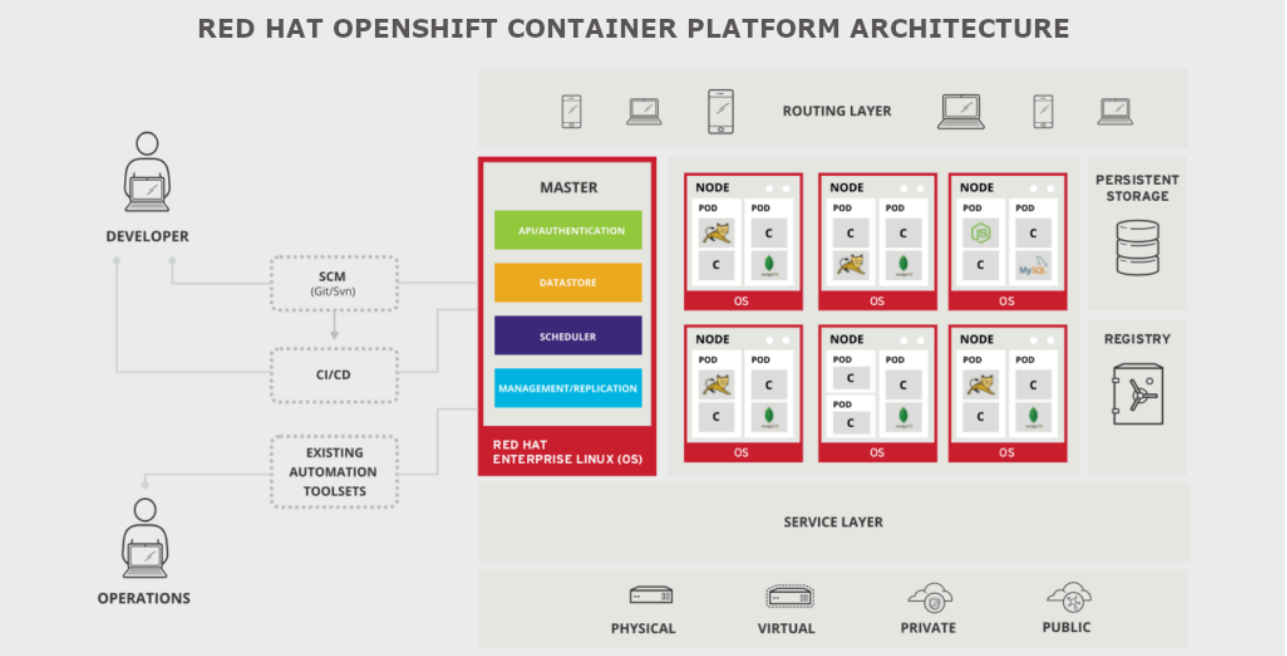


Рисунок 1.2 – Схематична архітектура OpenShift Container Platform

Джерело: [7]

Основними плюсами даної системи є:

– OpenShift базується на Kubernetes, тому він має ті ж технології;

– Дозволяє додаткам та центрам обробки даних, які їх підтримують, розширюватися на необмежену кількість фізичних машин;

– OpenShift надає розробникам та IT-організаціям хмарні платформи додатків, які можна використовувати для розгортання додатків на безпечних та масштабованих ресурсах.

Основними мінусами Container Platform є:

– OpenShift може бути складнішим для налаштування та управління в порівнянні з іншими платформами;

– Вимагає більше ресурсів для безпосереднього запуску.

Останнім інструментом для порівняння було обрано Rancher.

Rancher – це інструмент керування Kubernetes для розгортання та запуску кластерів будь-де та на будь-якому провайдері. Rancher може надавати Kubernetes від хостинг-провайдера, надавати обчислювальні вузли, а потім встановлювати на них Kubernetes, або імпортувати існуючі кластери Kubernetes, що працюють будь-де [8].

Rancher додає значну цінність до Kubernetes, по-перше, централізуючи автентифікацію та контроль доступу на основі ролей (RBAC) для всіх кластерів, надаючи глобальним адміністраторам можливість керувати доступом до кластерів з одного місця. По-друге, він забезпечує детальний моніторинг та оповіщення кластерів та їх ресурсів, надсилає журнали зовнішнім провайдерам та інтегрується безпосередньо з Helm через каталог додатків. Надає можливість підключити зовнішній pipeline CI/CD. В разі відсутності, Rancher має можливість підключення Fleet (він дозволяє користувачам легко керувати **кластерами** так, ніби вони є **одним кластером**), щоб допомогти користувачу автоматично розгортати і оновлювати робочі навантаження.

Схематична архітектура Rancher наведена на рисунку 1.3.

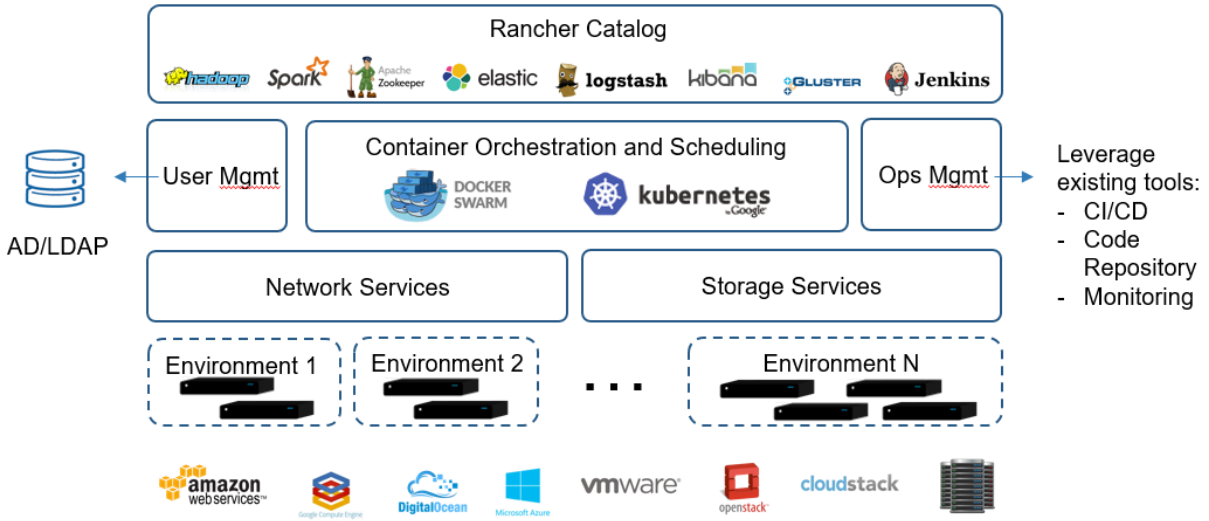


Рисунок 1.3 – Схематична архітектура Rancher

Джерело: [8]

Основними плюсами даної системи є:

– Наявність каталогу готових до запуску сервісів і додатків;

– Зрозумілий веб-інтерфейс;

– Можливість швидко запускати інші рішення для оркестрації, такі як Kubernetes і Docker Swarm.

Основними мінусами Rancher є:

– Складність освоєння та високий поріг входження;

– Необхідність мати навички роботи з Docker-образами.

Результатом здійснення порівняння аналогів є таблиця 1.1.

Таблиця 1.1 – Порівняльна таблиця аналогів інформаційних систем

Характеристика	**Docker Swarm**	**OpenShift Container Platform**	**Rancher**
Опис	Docker Swarm - це рішення для кластеризації та	OpenShift - це безкоштовна, автоматично масштабована	Rancher - це проект з відкритим вихідним кодом,

	оркестрації Docker контейнерів.	платформа від Red Hat для додатків.	який надає повну платформу для роботи Docker в производстве.
Вбудованість	Вбудований в Docker Engine, тому вам не потрібно встановлювати додаткове програмне забезпечення.	Базується на Kubernetes, тому вона має ті ж технології.	Надає інфраструктурні служби, такі як багатовузлові мережі, глобальну та локальну балансировку навантаження та моментальні знімки томів.
Масштабованість	Надає можливість масштабувати додатки, розподіляючи контейнери між вузлами.	Дозволяє додаткам та дата-центрам, які їх підтримують, розширюватися до тисяч фізичних машин	Дозволяє керувати контейнерами на різних хостах, що надає можливість масштабувати додатки.
Відновлення	Автоматично відновлює контейнери, якщо вони зупиняються.	Механізми відновлення аналогічні autohealing від Kubernetes	Має вбудовані механізми відновлення, які допомагають забезпечити неперервність роботи додатків.
Гнучкість	Docker Swarm може бути менш гнучким у порівнянні з іншими платформами, такими як Kubernetes.	OpenShift може бути складнішим для налаштування та управління в порівнянні з іншими платформами.	Rancher надає гнучкість у виборі між кількома оркестраторами контейнерів, включаючи Kubernetes, Mesos та Docker Swarm.

Ресурси	Docker Swarm вимагає менше ресурсів для запуску в порівнянні з іншими платформами.	OpenShift може вимагати більше ресурсів для запуску.	Rancher може вимагати більше ресурсів для запуску в порівнянні з іншими платформами.

Джерело: побудовано автором на основі даних зі статей [6-8]

Проаналізувавши інформацію з таблиці 1.1 можна визначити, що всі системи тим чи іншим чином відносяться до Kubernetes. Отже, інформаційному рішенню необхідно мати наступні риси: зручний інтерфейс керування, забезпечення основними функціями Kubernetes (масштабування, відновлення, балансування), планування використання ресурсів, а також моніторинг життєвого стану завантажених додатків.

2 Постановка задачі та методи дослідження

2.1 Мета та задачі дослідження

Мета магістерської кваліфікаційної роботи – розробка інформаційної технології забезпечення робастності вебдодатків. Розроблене інформаційне рішення буде корисним для компаній сектору IT, зокрема для груп розробки, тестування та підтримки. Розроблена технологія потенційно буде корисною для команд невеликого масштабу, які працюють над довгостроковим проектом і хочуть з самого старту мати налаштоване автоматизоване оточення. Втілення розробленого алгоритму та подальше його використання в IT компанії дозволяє автоматизувати процес розгортання нових версій вебдодатку, та оптимізує процес розробки програмного забезпечення та процеси всередині команди розробників.

Розроблена інформаційна технологія має наступні вимоги по функціональності:

- використання декларативної конфігурації;
- автоматичне самовідновлення, масштабування та моніторинг;
- проведення автоматичного розподілу навантаження між контейнерами;
- забезпечення секретності важливих даних.

Розроблена інформаційна технологія має супроводжуватись інтерфейсом, зрозумілим для розробника. Стан системи повинен відображатися в режимі реального часу.

Процес виконання мети проекту складається з наступних етапів:

- аналіз аналогічних інформаційних рішень;
- визначення технологій розробки інформаційної системи;
- розробка та реалізація структури інформаційної технології;
- тестування результатів розробки.

Після визначення цілей проекту було складено план робіт за проектом, детальна інформація про який наведена в Додатку А.

2.2 Методи дослідження

Для вирішення основної задачі системи, а саме підтримки робастності вебдодатків, було проведено дослідження різних методів, представлених в інформаційних джерелах.

Першим методом є автоматизація тестування. Цей процес включає в себе створення тестових сценаріїв, які автоматично перевіряють функціональність та стабільність вашого вебдодатку [22]. Це може включати:

- **Юніт-тестування:** Перевірка окремих компонентів програмного коду на коректність;
- **Інтеграційне тестування:** Перевірка взаємодії між різними компонентами вебдодатку;
- **Навантажувальне тестування:** Перевірка роботи вебдодатку під великим навантаженням;
- **Тестування безпеки:** Перевірка вебдодатку на наявність потенційних вразливостей.

Другим методом є використання контейнерів (наприклад, Kubernetes). Контейнери дозволяють ізолювати додатки та їх залежності в самостійні умови виконання. Це забезпечує консистентність середовища під час розробки або тестування. Kubernetes дозволяє автоматизувати розгортання, масштабування та управління контейнерізованими додатками, що підвищує їх робастність та швидкодію команди розробки [23]. Ось декілька ключових аспектів:

– **Оркестрація контейнерів:** Kubernetes дозволяє управляти життєвим циклом контейнерів, включаючи розгортання, оновлення, масштабування, моніторинг статусу та видалення;

– **Сервісне виявлення та балансування навантаження:** Kubernetes може автоматично виявляти та балансувати навантаження між контейнерами, що підвищує доступність та стабільність роботи вебдодатку;

– **Автоматичне масштабування:** Kubernetes може автоматично горизонтально (збільшуючи кількість працюючих копій контейнерів) або вертикально (збільшуючи обсяг виділених ресурсів) масштабувати вебдодаток відповідно до навантаження.

Найпопулярнішою практикою є використання принципів DevOps: DevOps об’єднує розробку та експлуатацію, що сприяє швидкому циклу розробки та неперервній інтеграції [24]. Це може включати в себе:

– **Неперервна інтеграція (CI):** Це процес, який включає автоматичне збирання та тестування програмного коду при кожному коміті.

– **Неперервна доставка (CD):** Це процес, який включає автоматичне розгортання оновленого коду в продуктивне середовище.

– **Моніторинг та журналювання:** Це включає в себе збір та аналіз логів та метрик роботи вебдодатку для виявлення та вирішення проблем.

Отже, проаналізувавши представлені методи, було прийнято рішення об’єднати все в один алгоритм налаштування підтримки вебдодатків. Основними складностями виступають теоретична підготовка (зокрема вивчення основ мови HCL), а також процес налаштування середовища.

HashiCorp Configuration Language (HCL) – це мова конфігурації, створена компанією HashiCorp для її інструментів автоматизації хмарної інфраструктури, таких як Terraform. HCL розроблена таким чином, щоб бути зручною як для людей, так і для машин, і вона візуально схожа на JSON з додатковими вбудованими структурами даних та можливостями [9].

2.3 Вибір технологій

Після того, як був проведений аналіз актуальності роботи та визначений тип програмного забезпечення, треба визначитись з технологією реалізації даного інформаційного рішення.

Для реалізації інформаційної технології було обрано наступні технології:

– **Kubernetes**: основна технологія, для управління контейнеризованими додатками згідно з документацією [25];

– **Docker**: інструментарій для збірки, тестування та завантаження робочих образів на Docker Hub, а також хостування кластеру Kubernetes [26];

– **GitHub**: використовується для хостингу програмного коду [27];

– **GitHub Actions**: платформа безперервної інтеграції та безперервного постачання (CI/CD), що дає змогу автоматизувати конвеєр збирання, тестування та розгортання [28];

– **K9s**: надає термінальний інтерфейс для взаємодії з кластерами Kubernetes [29];

– **Kubernetes Dashboard:** інтегрована програмна система візуалізації даних, орієнтована на дані систем IT-моніторингу [39];

– **Terraform:** використовується для декларативного оголошення інфраструктури як для хмарного, так і для локального оточення [35];

– **Minikube:** локальний кластер Kubernetes. Може бути розгорнутий як віртуальна машина, як Docker-контейнер тощо.

Вибір цих конкретних технологій обґрунтовується їхньою здатністю ефективно вирішувати поставлені завдання у контексті розробки інформаційної технології забезпечення робастності вебдодатків. Ось основні причини вибору:

- Kubernetes та Docker: Забезпечують масштабованість і надійність вебдодатків завдяки контейнеризації і оркестрації. Дозволяють швидко розгортати та масштабувати додатки, зменшуючи час та витрати на розробку та підтримку.

– **GitHub та GitHub Actions:** Забезпечують зручне керування версіями коду та автоматизацію процесів CI/CD, що сприяє швидкому та надійному випуску нових версій програмного забезпечення.

– **Terraform:** Дозволяє декларативно описувати інфраструктуру як код, що спрощує розгортання та управління інфраструктурою Kubernetes.

– **K9s та Kubernetes Dashboard:** Надають інтуїтивно зрозумілий інтерфейс для взаємодії та моніторингу кластерів Kubernetes, що полегшує адміністрування системи.

– **Minikube:** Дозволяє легко та швидко розгорнути локальний кластер Kubernetes для розробки та тестування безпосередньо на робочій станції розробника.

3 Моделювання та проектування

Наступним етапом, після детального аналізу предметної області, визначення актуальності та цілей розробки, є розроблення сформульованого інформаційного рішення. У процесі роботи було розроблено IDEF0-діаграму, що деталізує послідовність процесів, пов'язаних зі створенням інфраструктури вебдодатка.

3.1 Структурно-функціональне моделювання

Для створення функціональних моделей інформаційних систем, що розробляються, використовують різні методи структурного та об'єктно-орієнтованого аналізу. Одним із найважливіших методів є IDEF0, який використовується для ефективного моделювання.

Діаграма IDEF0 – це інструмент функціонального моделювання, що використовується для візуалізації робочих процесів і систем. Ця нотація фокусується на функціональних аспектах системи або процесу і наочно показує, як функції взаємодіють одна з одною та із зовнішнім середовищем [10].

Діаграма IDEF0 складається з наступних елементів:

- механізми;
- вхідні дані;
- вихідні дані;
- управління.

Діаграми IDEF0 часто використовують для аналізу наявних процесів з метою виявлення можливостей їхньої оптимізації або для розроблення нових систем і процесів. Діаграми даного типу демонструють взаємозв’язки між ключовими функціями інформаційних технологій.

Процес «Налаштування системи підтримки робастності вебдодатків» має наступні дані:

– вхідні дані: вихідний код вебдодатку, інформація про інфраструктуру вебдодатку;

– вихідні дані: налаштована інфраструктура вебдодатку;

– управління: обмеження ресурсів та бюджету, вимоги до надійності системи, вимоги до процесу розгортання системи;

– механізми: інструменти DevOps, апаратне забезпечення, користувач, середовище Kubernetes.

На рисунку 3.1 зображена контекстна діаграма IDEF0 інформаційної технології з підтримки робастності вебдодатків.

Рисунок 3.1 – Контекстна діаграма IDEF0

Джерело: побудовано автором

Для деталізації процесів інформаційної технології було виконано декомпозицію першого рівня для процесу налаштування інфраструктури для підтримки робастності вебдодатків (рис. 3.2).

Дані для діаграми наступні:

- вхідні дані: вихідний код вебдодатку, інформація про інфраструктуру вебдодатку;
- вихідні дані: налаштована інфраструктура вебдодатку;
- управління: обмеження ресурсів та бюджету, вимоги до надійності системи, вимоги до процесу розгортання системи;
- механізми: інструменти DevOps, апаратне забезпечення, користувач, середовище Kubernetes.

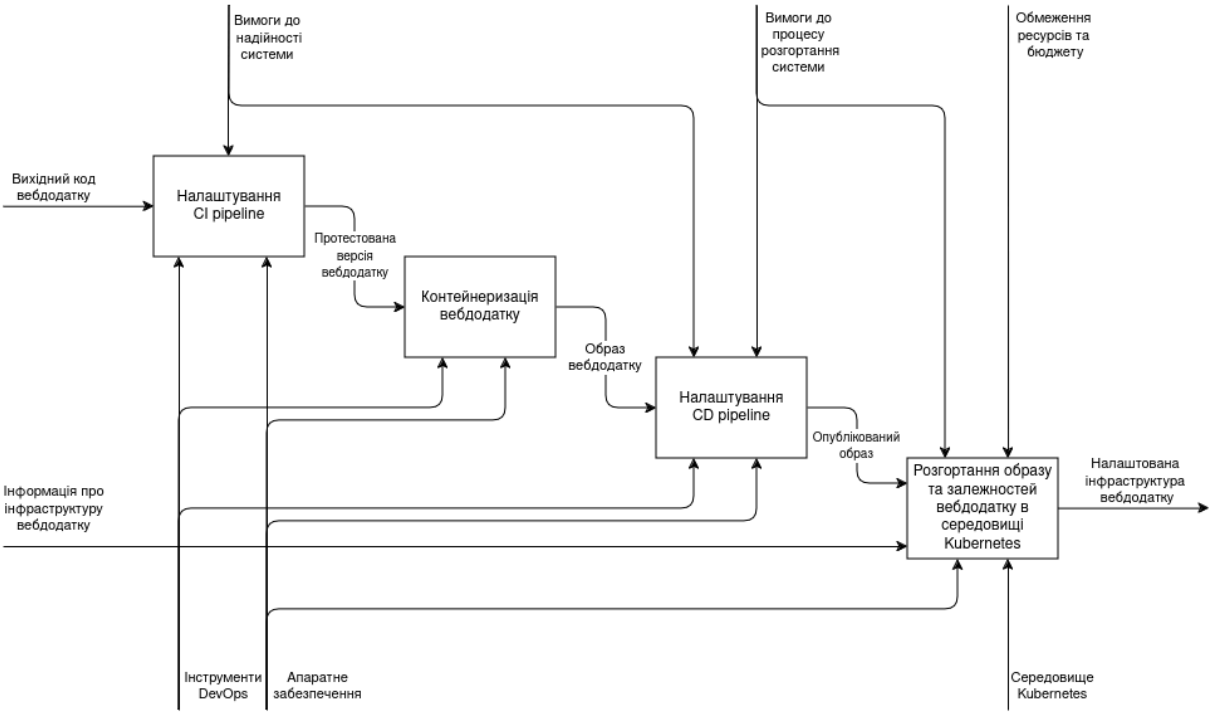


Рисунок 3.2 – Діаграма декомпозиції першого рівня

Джерело: побудовано автором

3.2 Use Case діаграма

Наступним етапом, після моделювання IDEF0 та діаграм декомпозиції, є проектування діаграми Use Case. Діаграма варіантів використання це важливий інструмент галузі проектування інформаційних рішень. Діаграми даного типу допомагають у візуалізації функціональних вимог системи і відносини між фактичними акторами та системою [11].

Основними компонентами діаграми варіантів використання є:

- актори;
- відносини;
- межі системи;
- варіанти використання.

Діаграми Use Case сприяють розумінню функціоналу системи та допомагають визначити основні вимоги користувачів. Це сприяє комунікації розробників з зацікавленими сторонами.

Інформаційна технологія забезпечення робастності вебдодатків виділяє наступні use case-и:

- **масштабування**: користувач може змінювати кількість реплік вебдодатків як власноруч, так і за допомогою завчасно прописаної стратегії у відповідному ресурсі (Deployment або ReplicaSet) Kubernetes [15];
- **моніторинг**: користувач має можливість використовувати інструменти моніторингу (наприклад, Prometheus[12], Grafana[13], Kubernetes Dashboard [39]) для відстеження стану вебдодатків або Kubernetes кластера в цілому;
- **аварійне відновлення**: користувач здатен власноруч відновлювати контейнери вебдодатків або налаштувати необхідні перевірки життєздатності [14]. В додаток до цього, Kubernetes, за відсутності прописаних стратегій, автоматично створює копію вебдодатку, який подає ознаки аварійного завершення;
- **безперервна інтеграція**: користувач може прописати необхідні тести для перевірки стабільності нового коміту[18] вебдодатку[16];
- **безперервне розгортання**: користувач має можливість налаштувати стратегію контейнеризації та опублікування готового образу вебдодатку [17].

Акторами діаграми варіантів використання є користувач, налаштоване середовище Kubernetes та прописані стратегії CI/CD в GitHub Actions. Діаграма використання інформаційної технології забезпечення робастності вебдодатків зображена на рисунку 3.3.

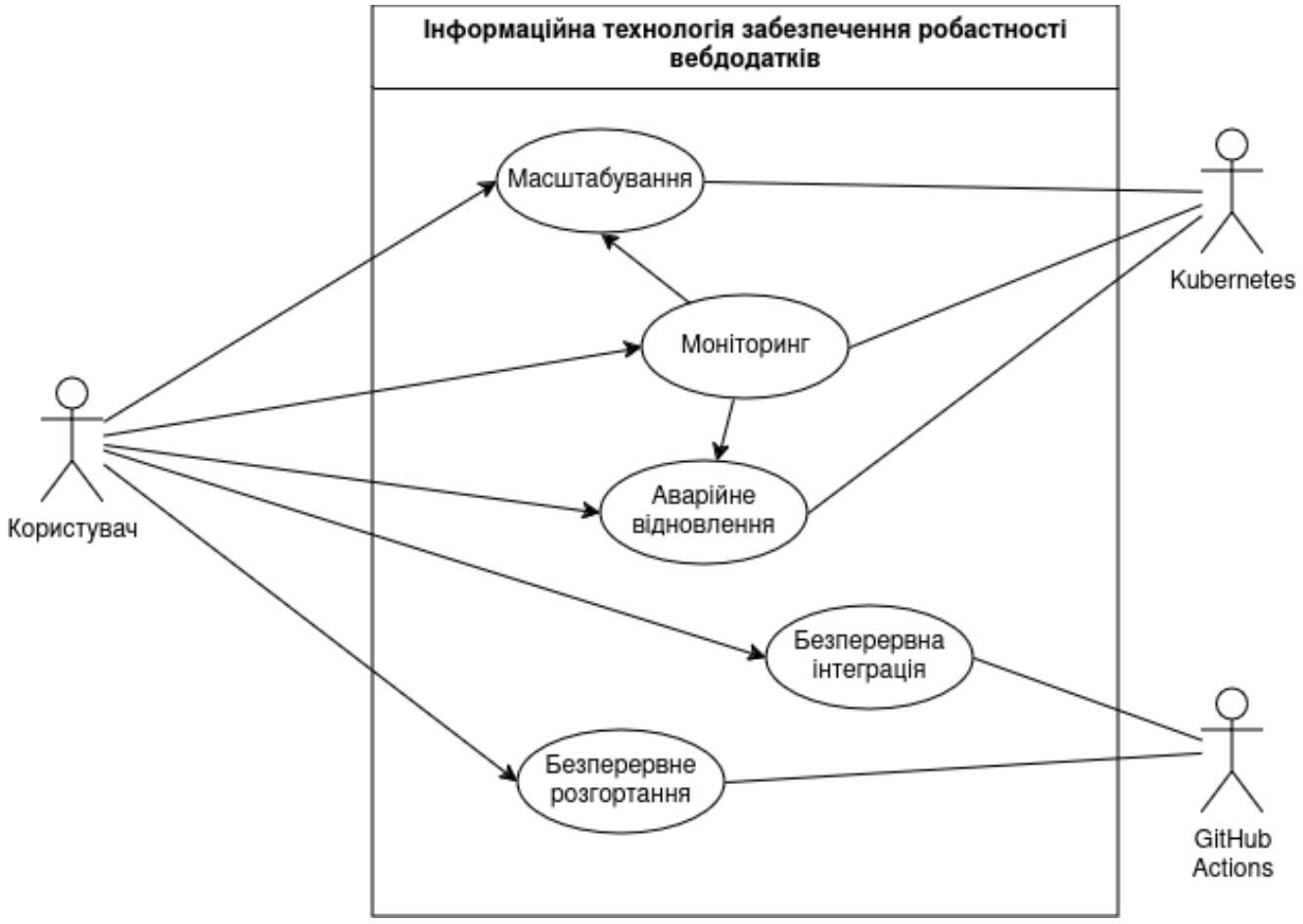


Рисунок 3.3 – Use Case діаграма

Джерело: побудовано автором

3.3 Діаграми послідовності

На рис. 3.4 наведено діаграма послідовності для масштабування ресурсів Kubernetes. Діаграма послідовності відображає процес автоматичного масштабування у Kubernetes та взаємодію між різними компонентами системи. Користувач, взаємодіючи з Kubernetes API, ініціює процес автоматичного масштабування. Kubernetes API відправляє запит до Metrics Server для отримання актуальних метрик, таких як завантаження CPU або використання оперативної пам'яті.

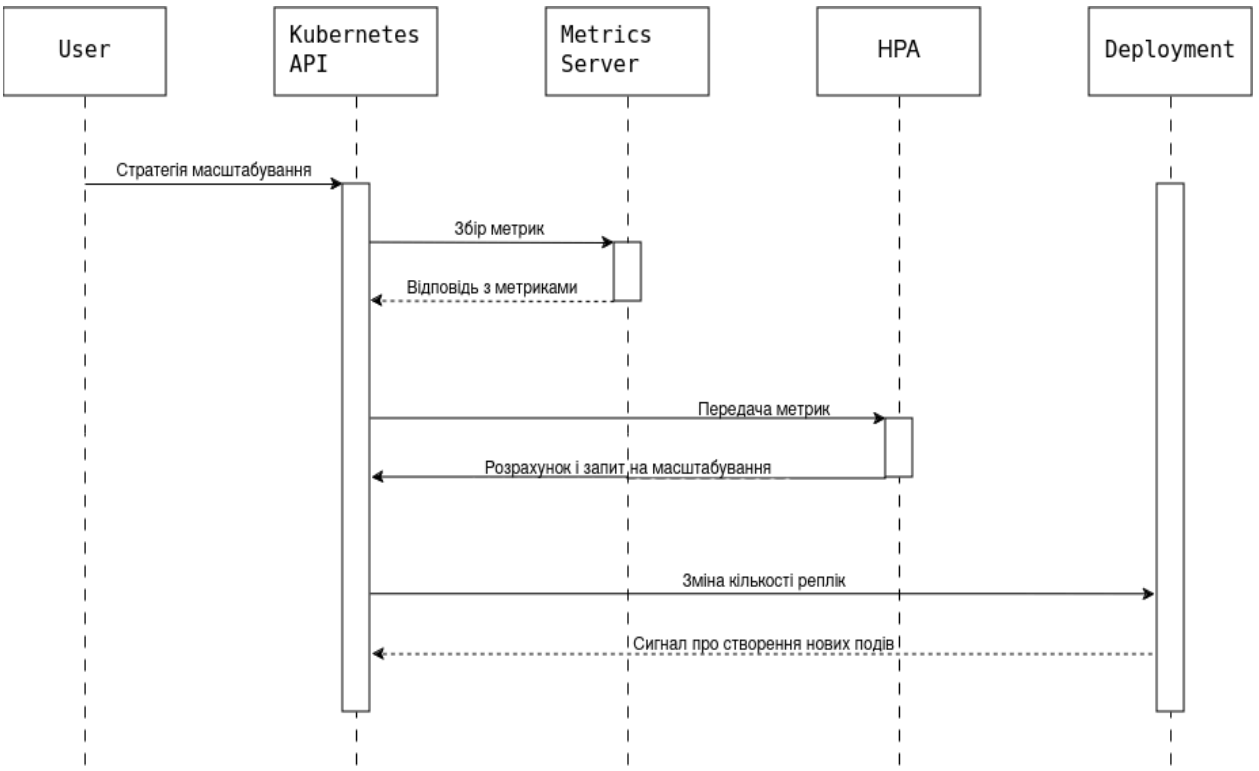


Рисунок 3.4 – Діаграма послідовності для масштабування ресурсів Kubernetes

Джерело: побудовано автором

Після збирання метрик Metrics Server передає їх до Horizontal Pod Autoscaler (HPA), який використовує ці дані для розрахунку необхідної кількості реплік (інстансів) додатка. На основі заданих умов автоматичного масштабування HPA визначає, чи потрібно збільшити або зменшити кількість реплік.

Після цього HPA відправляє відповідний запит на масштабування назад до Kubernetes API. Kubernetes API, отримавши цей запит, змінює кількість реплік у Deployment. Нарешті, Deployment виконує відповідні дії для створення або видалення нових подів відповідно до запиту на масштабування, забезпечуючи таким чином підтримку оптимального рівня завантаження та ресурсів у кластері Kubernetes.

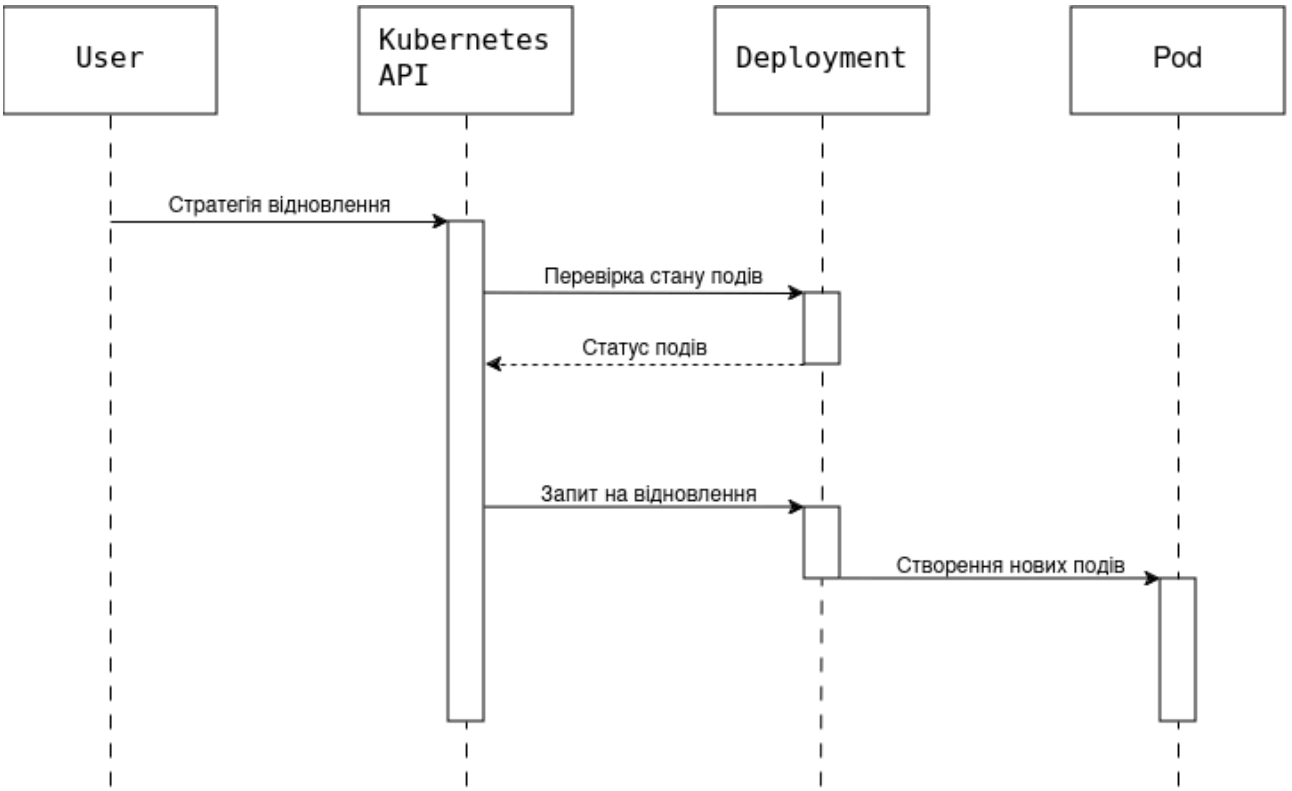


Рисунок 3.5 - Діаграма послідовності для самовідновлення ресурсів Kubernetes

Джерело: побудовано автором

Діаграма відображає, як користувач взаємодіє з Kubernetes API для запиту на відновлення подів. Kubernetes API перевіряє стан подів в Deployment і отримує їх статус. Якщо поди потребують відновлення, Kubernetes API відправляє запит на відновлення до Deployment, який потім створює нові поди.

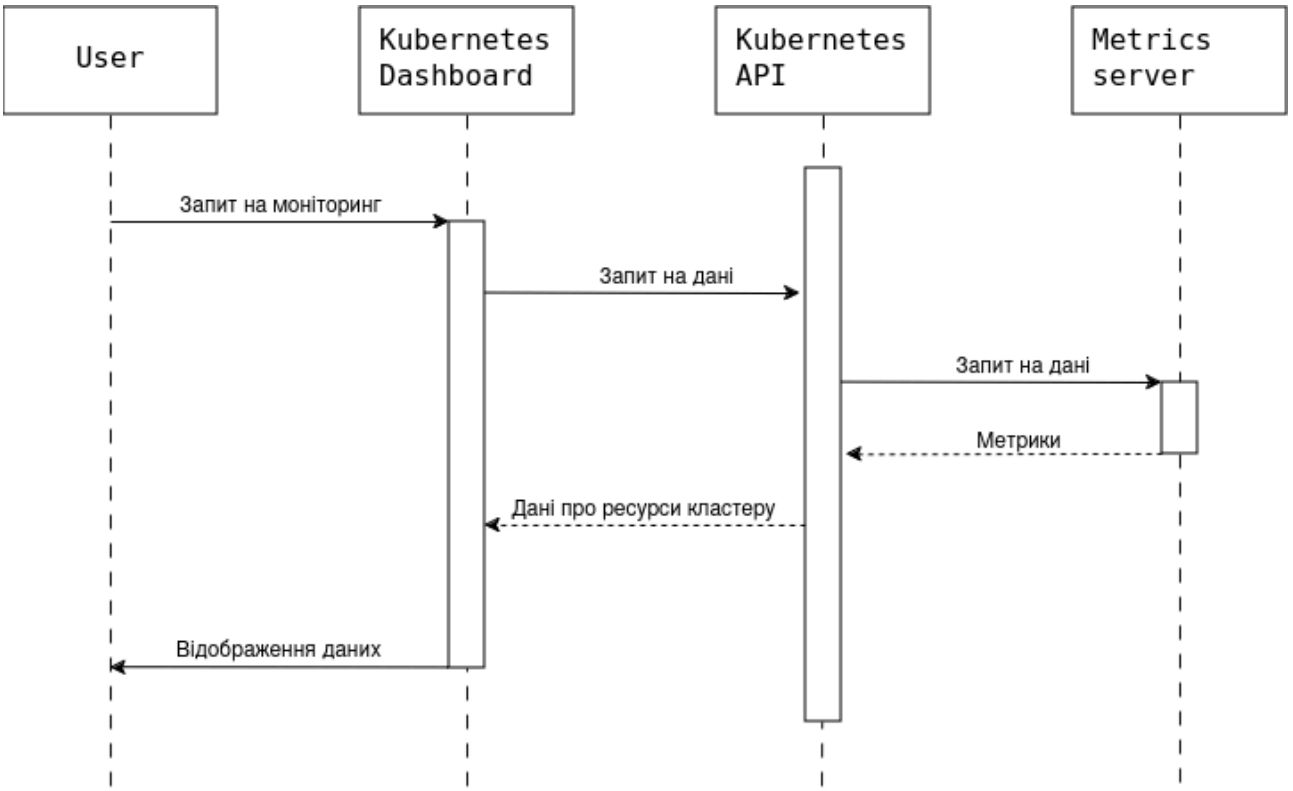


Рисунок 3.6 - Архітектура демонстраційного додатку

Джерело: побудовано автором

Діаграма відображає, як користувач взаємодіє з Kubernetes Dashboard для моніторингу ресурсів кластеру. Kubernetes Dashboard відправляє запит на дані до Kubernetes API, який отримує дані про ресурси кластеру від Metrics Server. Потім Kubernetes API відправляє ці дані назад до Kubernetes Dashboard, який відображає ці дані користувачу.

4 Розробка інформаційної технології забезпечення робастності вебдодатків

4.1 Реалізація демонстраційного додатку

Процес розробки інформаційної технології включає в себе підготовку програмного коду (розбиття на модулі, покриття тестами, перевірка скриптів збірки), налаштування CI/CD пайплайнів (створення маніфесту GitHub Actions та завантаження коду в репозиторій) та розгортання (створення декларативної конфігурації інфраструктури в середовищі Kubernetes). Архітектура інформаційної технології зображена на малюнку 4.1.

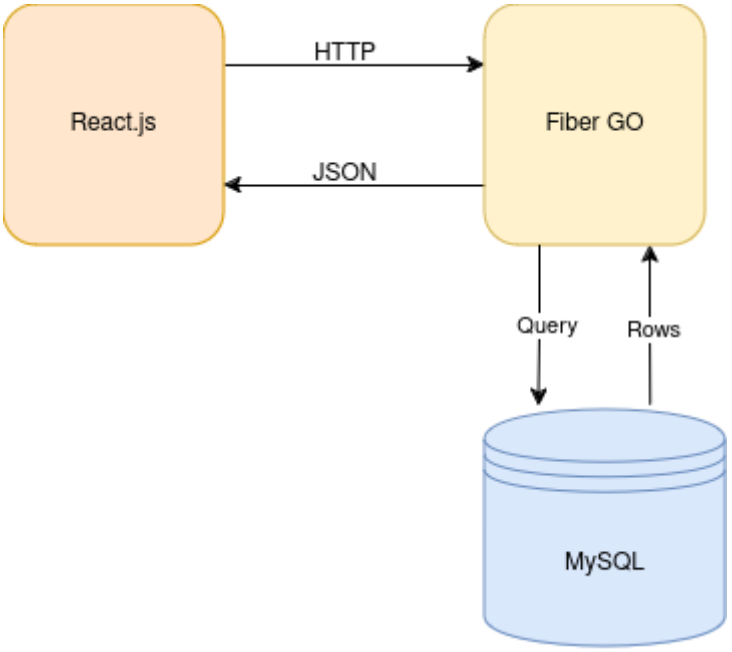


Рисунок 4.1 - Архітектура демонстраційного додатку

Джерело: побудовано автором

Демонстраційний додаток був складається з 3 модулів. За front-end частину відповідає JavaScript бібліотека React.js, засобами якої був створений додаток To-do-list [32]. Back-end побудований з використанням мікрофреймворку Fiber. Fiber – це веб-фреймворк Go, побудований на основі

Fasthttp, найшвидшого HTTP-рушія для Go [33]. Системою керування реляційною базою даних було обрано MySQL [34].

Базу даних було створено згідно мінімальним потребам додатку. На рисунку 4.2 представлена структура таблиці БД.

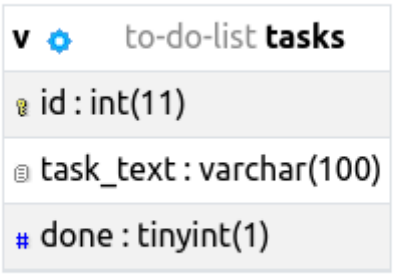


Рисунок 4.2 - Структура таблиці завдань

Джерело: побудовано автором (знімок екрану)

На рисунку 4.3 наведено вигляд інтерфейсу вебдодатку.

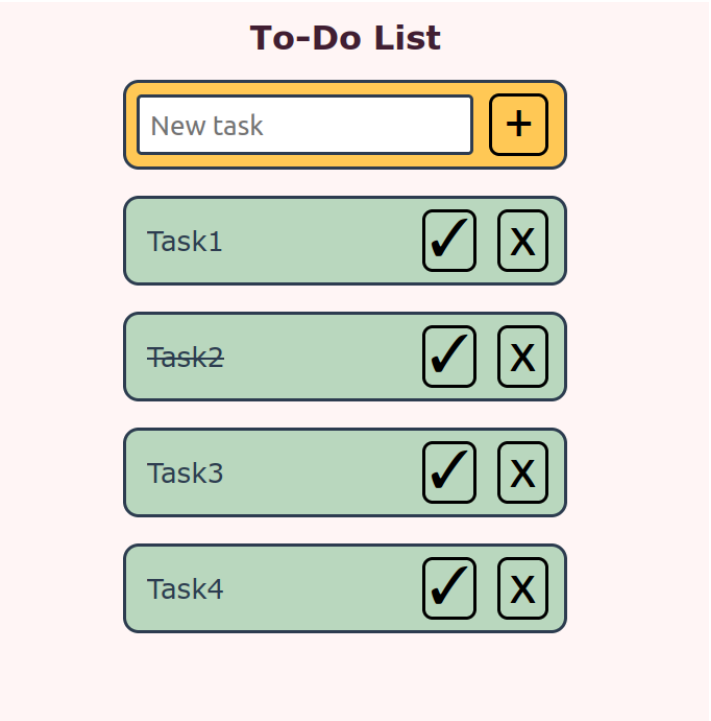


Рисунок 4.3 - Інтерфейс демонстраційного вебдодатку

Джерело: побудовано автором (знімок екрану)

Структура вебсторінки розбита на компоненти і описана в кожному файлі окремо (рис. 4.4). Приклад коду компонента наведено на рисунку 4.5.

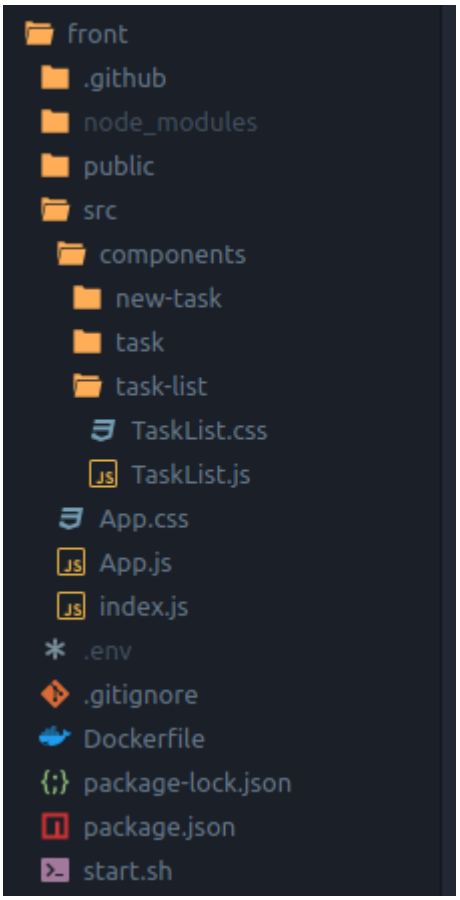


Рисунок 4.4 – Структура директорії front-end частини вебдодатку

Джерело: побудовано автором (знімок екрану)

Рисунок 4.5 – Приклад програмного коду React компонента

Джерело: побудовано автором (знімок екрану)

Back-end частина, яка використовується для взаємодії з базою даних (БД) MySQL була реалізована мовою Golang з використанням мікрофреймворку Fiber. Алгоритм взаємодії з fiber наступний:

1. Фронт-енд відправляє HTTP-запит до сервера, на якому запущений Fiber;
2. Fiber приймає цей запит і перенаправляє його до відповідного обробника маршруту, засновуючись на HTTP-методі (GET, POST тощо) та шляху запиту (наприклад, /users);
3. Визначений обробник маршруту виконує необхідні операції з базою даних. У даному випадку, виконується запит до БД MySQL для отримання всіх записів з таблиці завдань;
4. Після отримання даних з бази даних, Fiber конвертує ці дані в формат JSON і відправляє їх назад як відповідь на оригінальний HTTP-запит;
5. Фронт-енд отримує цю відповідь і використовує дані JSON для оновлення інтерфейсу користувача.

Маршрути запитів, методи та їх обробники описані в окремому файлі (рис. 4.6). Приклад обробника для отримання усіх запитів наведено на рисунку 4.7.

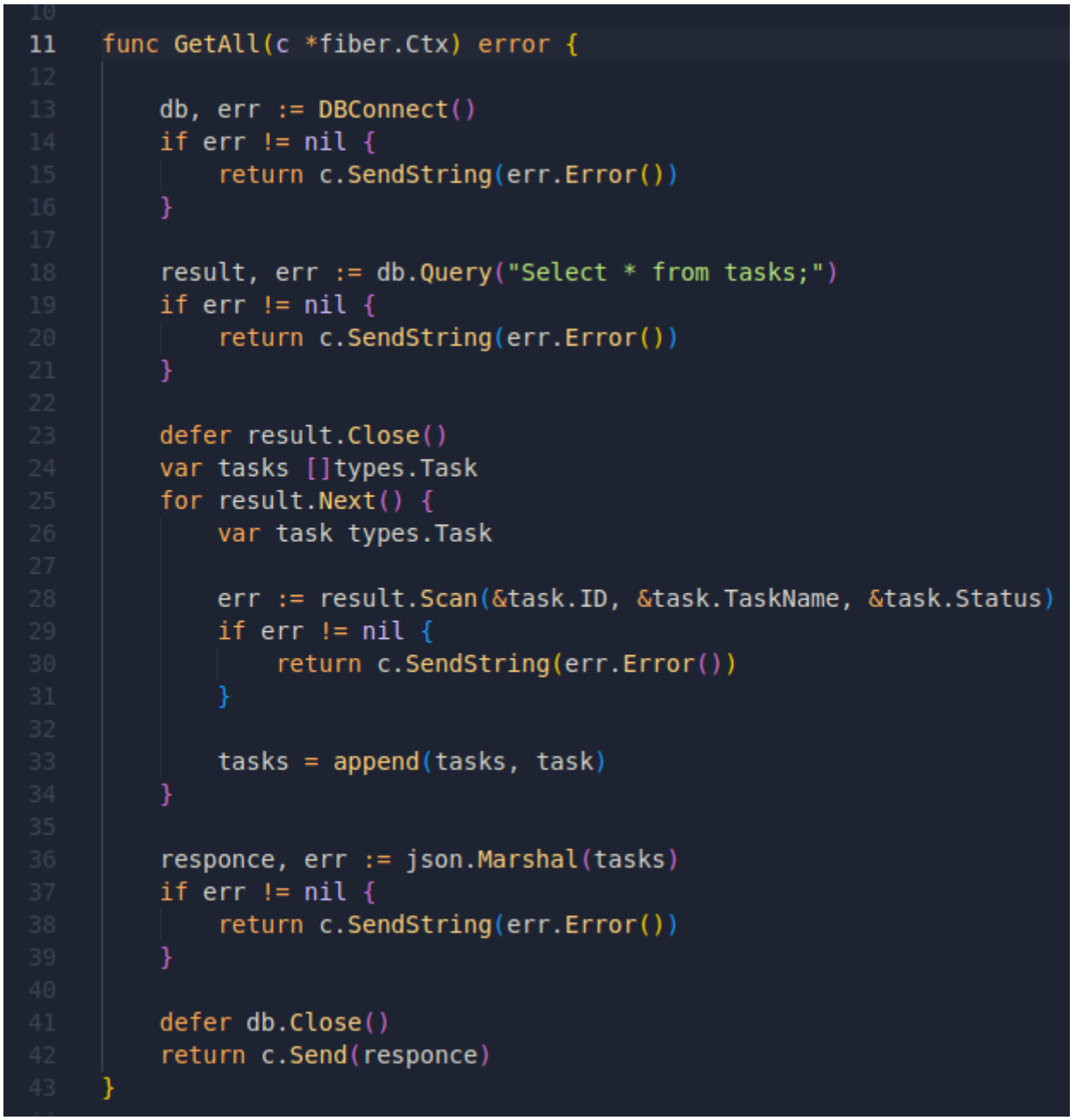
routes.go

back > pkg > routes > routes.go > PublicRoutes

```go
package routes

import (
    "github.com/gofiber/fiber/v2"

    "mysql-controller/pkg/commands"
)

func PublicRoutes(a *fiber.App) {
    route := a.Group("/api/v1")

    route.Get("/getall", commands.GetAll)
    route.Get("/get/:id", commands.Get)
    route.Post("/insert/:taskName", commands.Insert)
    route.Post("/delete/:id", commands.Delete)
    route.Post("/update/:id.:status", commands.Update)
}
```

Рисунок 4.6 – Оголошені шляхи запитів та їх обробники

Джерело: побудовано автором (знімок екрану)



```
func GetAll(c *fiber.Ctx) error {

    db, err := DBConnect()
    if err != nil {
        return c.SendString(err.Error())
    }

    result, err := db.Query("Select * from tasks;")
    if err != nil {
        return c.SendString(err.Error())
    }

    defer result.Close()
    var tasks []types.Task
    for result.Next() {
        var task types.Task

        err := result.Scan(&task.ID, &task.TaskName, &task.Status)
        if err != nil {
            return c.SendString(err.Error())
        }

        tasks = append(tasks, task)
    }

    responce, err := json.Marshal(tasks)
    if err != nil {
        return c.SendString(err.Error())
    }

    defer db.Close()
    return c.Send(responce)
}
```

Рисунок 4.5 – Приклад обробника для отримання всіх завдань з таблиці БД

Джерело: побудовано автором (знімок екрану)

4.2 Підготовка коду вебдодатку

Програмний код готового вебдодатку було розподілено у 3 директорії: front, back, db (рис. 4.6). Директорія terraform відповідає за декларативне

описання бажаного кінцевого стану інфраструктури та містить в собі файли з розширенням .tf.

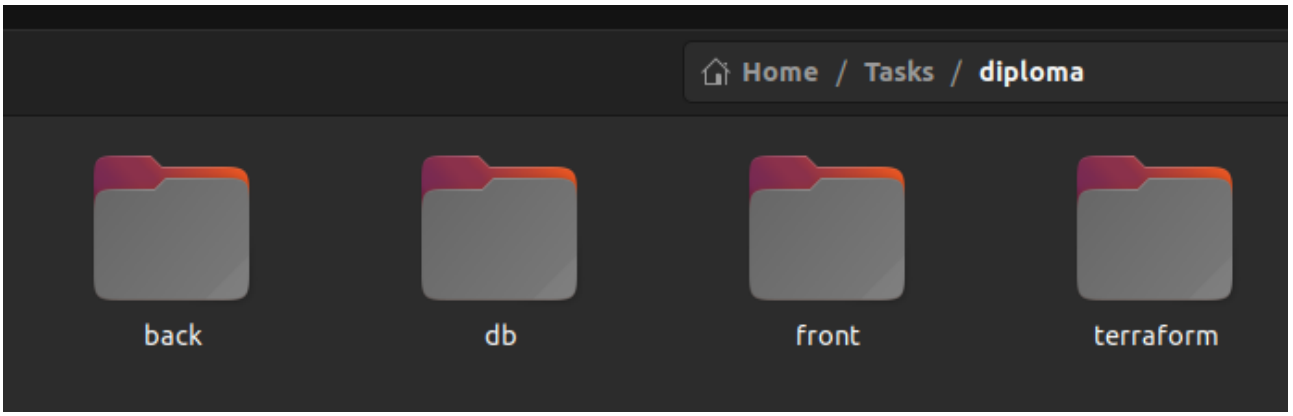


Рисунок 4.6 – Відповідні директорії для кожного модуля інформаційної технології

Джерело: побудовано автором (знімок екрану)

Для всіх функціональної частини вебдодатку були створені тести, для перевірки коректності виконання всіх програмних компонентів. Під час цього етапу було використано інструментарій фреймворку для написання тестів мовою GO – Testify [36].

Тестування додатків мовою є дає можливість використати команду “go test -v -cover ./…”. Флаг “-cover” показує відсоток покриття тестами усього наявного функціоналу (рис. 4.7).

На рисунку 4.8 представлено приклад тесту для перевірки функції GetAll.

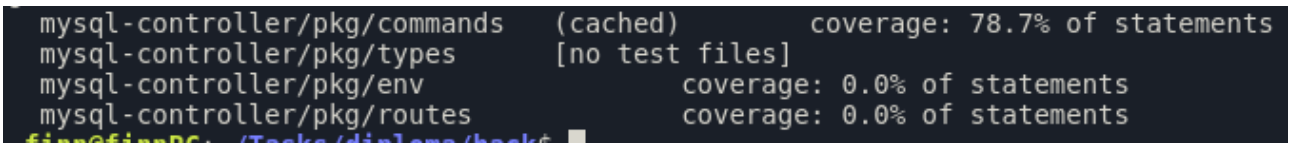


Рисунок 4.7 – Вивід запуску тестів з використанням -cover

Джерело: побудовано автором (знімок екрану)

routes.go | get-all_test.go

back > pkg > commands > get-all_test.go > TestGetAll

run package tests | run file tests

```go
package commands_test

import (
    "encoding/json"
    "io"
    "mysql-controller/pkg/commands"
    "mysql-controller/pkg/types"
    "net/http/httptest"
    "testing"

    "github.com/gofiber/fiber/v2"
    "github.com/stretchr/testify/assert"
)

run test | debug test
func TestGetAll(t *testing.T) {

    assert := assert.New(t)
    app := fiber.New()

    app.Get("/tasks", commands.GetAll)

    req := httptest.NewRequest("GET", "/tasks", nil)
    resp, err := app.Test(req)

    assert.Nil(err)
    assert.Equal(200, resp.StatusCode)

    body, err := io.ReadAll(resp.Body)
    assert.Nil(err)

    var tasks []types.Task
    err = json.Unmarshal(body, &tasks)
    assert.Nil(err)

    var mock []types.Task
    assert.IsType(mock, tasks)
}
```

Рисунок 4.8 – Програмний код для тестування функції GetAll

Джерело: побудовано автором (знімок екрану)

Перед наступним етапом необхідно виконати тестування всіх програмних компонентів, використання яких має вплив на робастність вебдодатку та збірку кожного з модулів. В разі виникнення помилок – відкоригувати програмний код.

4.3 Налаштування CI/CD пайплайнів

На етапі створення CI/CD пайплайнів (послідовності виконання визначених дій) було використано Github Actions. Спочатку програмний код було завантажено до репозиторію Github (рис. 4.9).

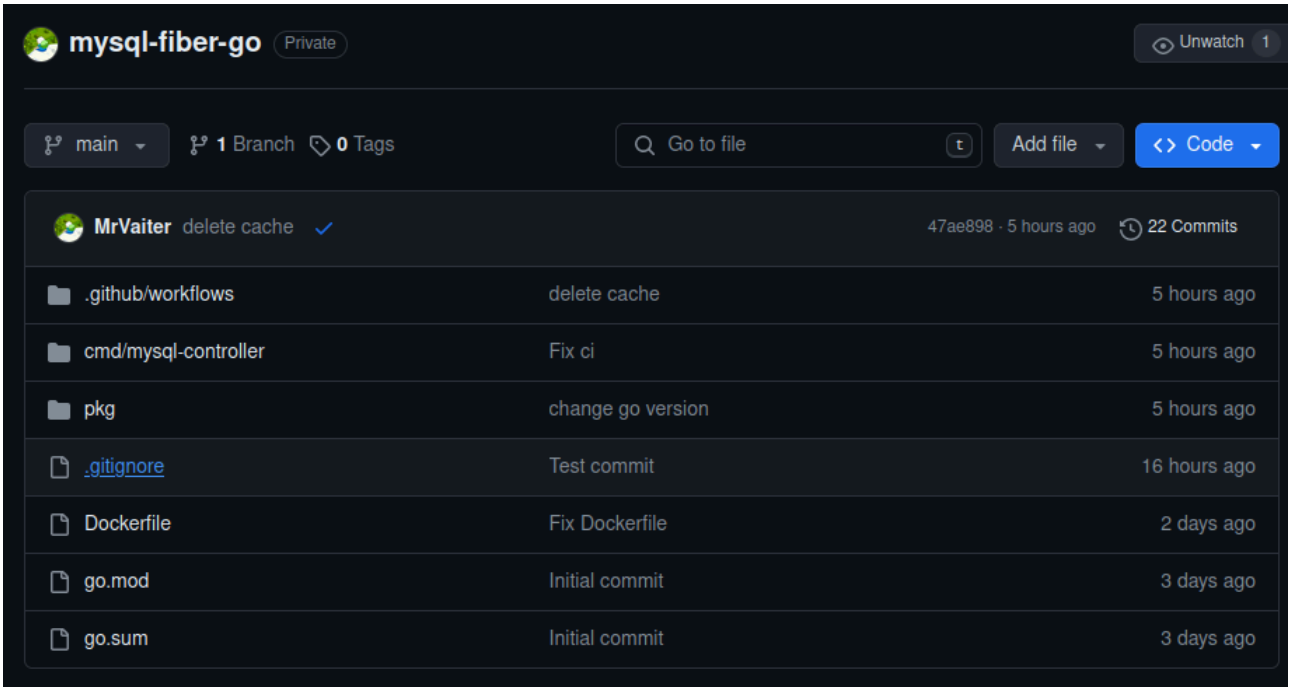


Рисунок 4.9 – Програмний код back-end частини ведодатку в приватному репозиторії

Джерело: побудовано автором (знімок екрану)

Далі необхідно створити файл з розширенням .yaml в якому буде описана конфігурація та послідовність робочого процесу GitHub Action (рис. 4.10).

Шлях до файлу наступний: з корня директорії ***./github/workflows/***. Також можна обрати завчасно створений шаблон в розділі Actions (рис. 4.11).

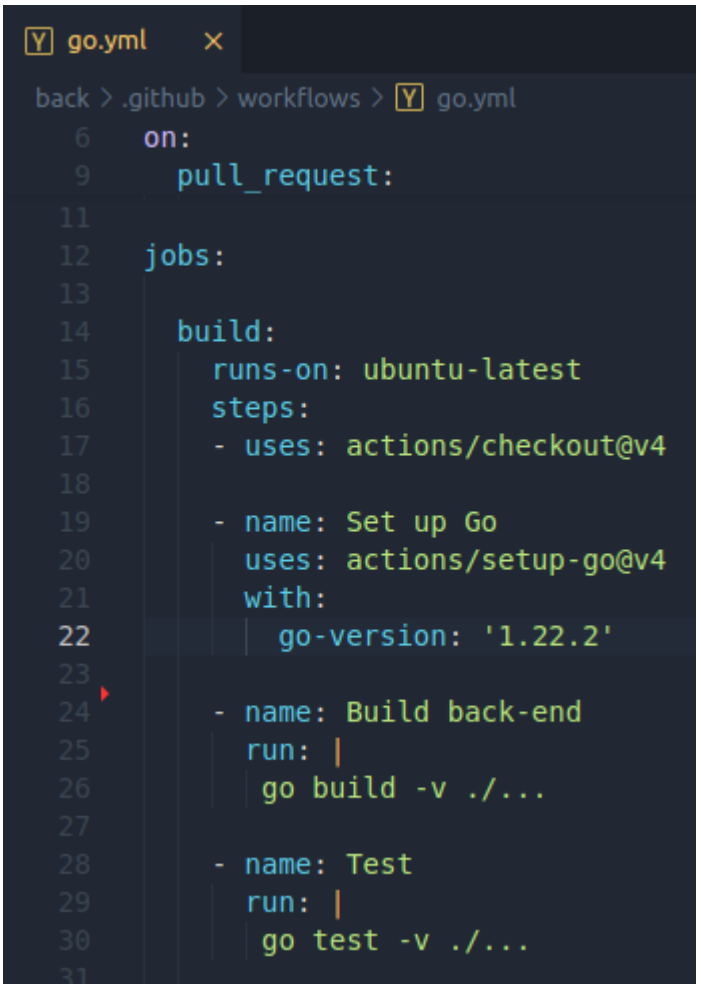


Рисунок 4.10 – Файл робочого процесу для додатків написаних мовою GO

Джерело: побудовано автором (знімок екрану)

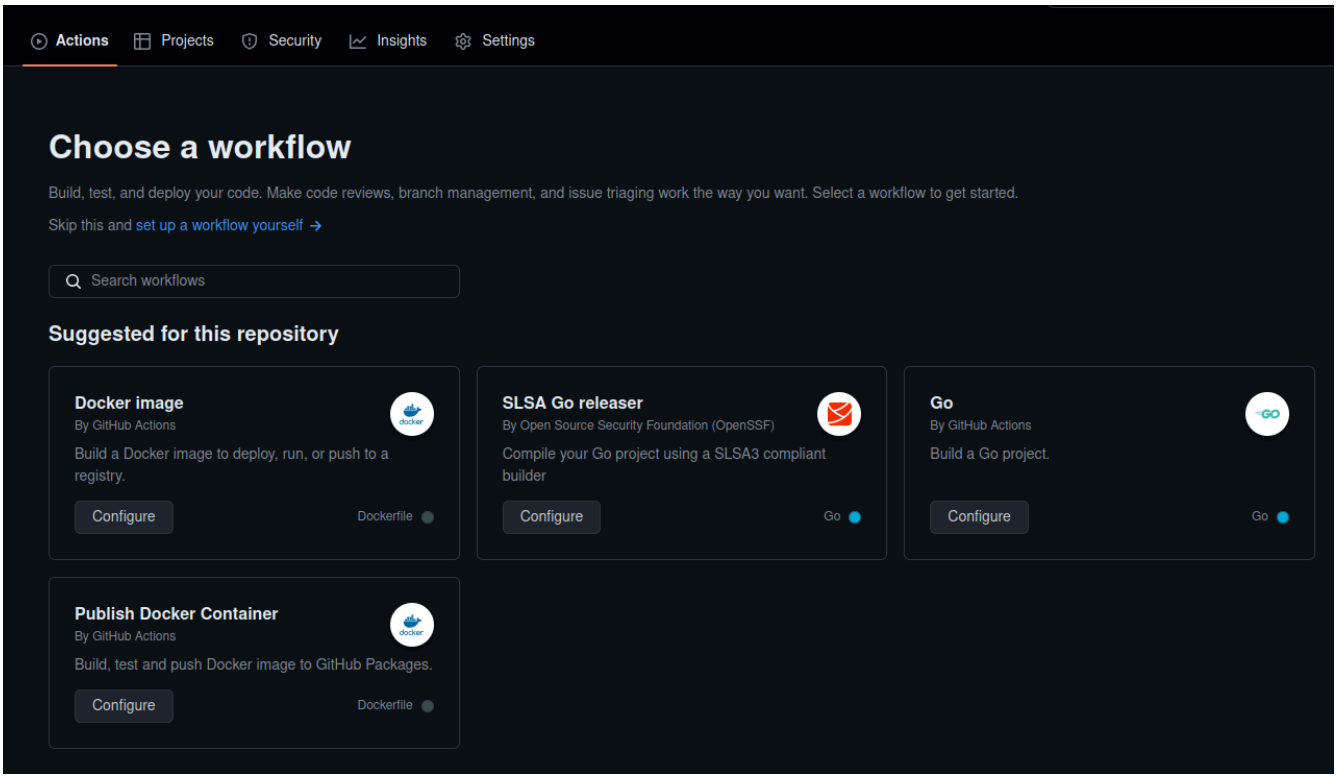
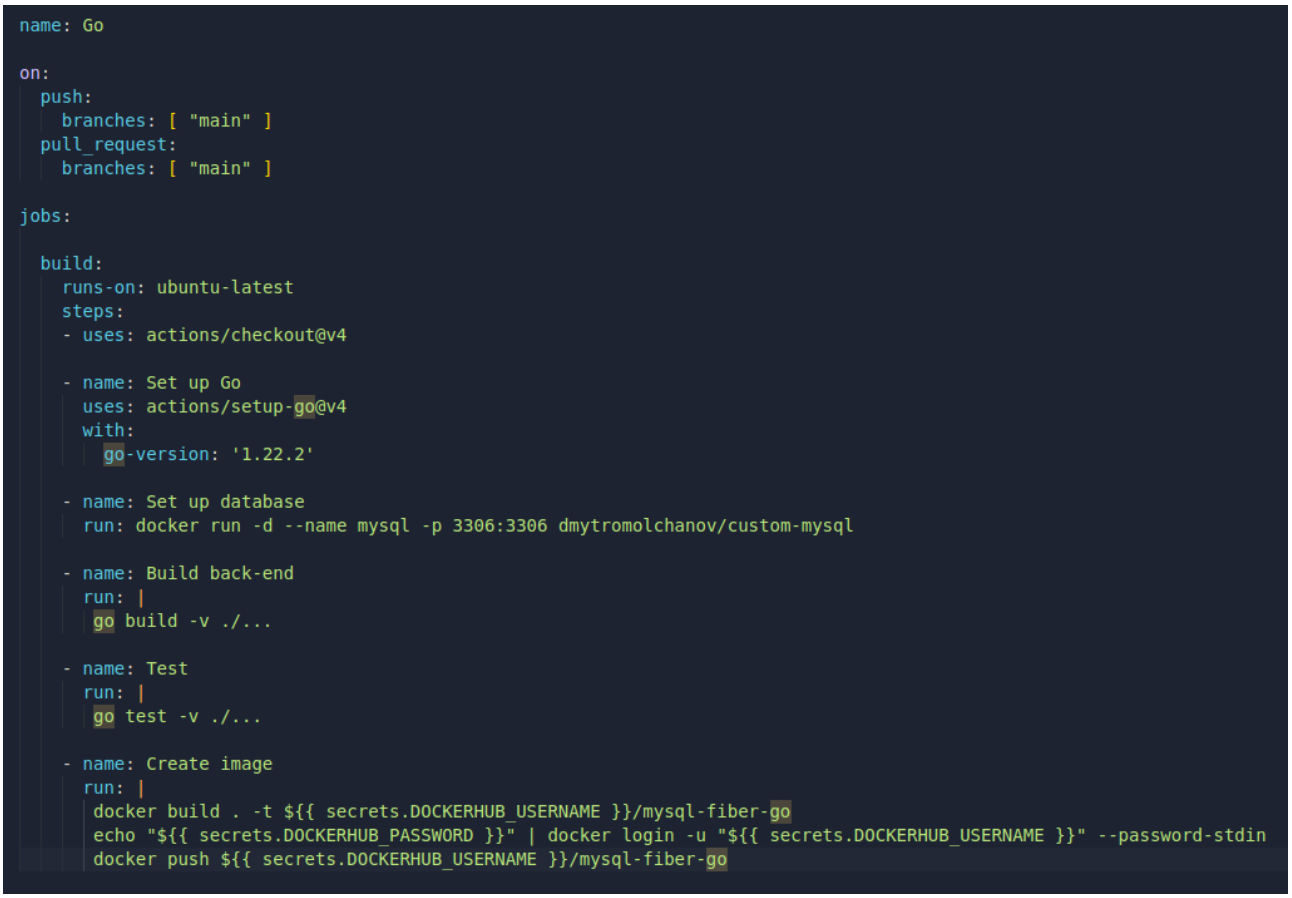


Рисунок 4.10 – Шаблони, запропоновані GitHub

Джерело: побудовано автором (знімок екрану)

У даному випадку, необхідно протестувати функціонал для взаємодії з базою даних, тому необхідно додати контейнер mysql безпосередньо в робочий процес. В разі успішного тестування, необхідно створити образ та завантажити його в персональний Docker Hub репозиторій. Відредагований .yaml файл представлений на рисунку 4.11.



```yaml
name: Go

on:
  push:
    branches: [ "main" ]
  pull_request:
    branches: [ "main" ]

jobs:

  build:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v4

    - name: Set up Go
      uses: actions/setup-go@v4
      with:
        go-version: '1.22.2'

    - name: Set up database
      run: docker run -d --name mysql -p 3306:3306 dmytromolchanov/custom-mysql

    - name: Build back-end
      run: |
       go build -v ./...

    - name: Test
      run: |
       go test -v ./...

    - name: Create image
      run: |
       docker build . -t ${{ secrets.DOCKERHUB_USERNAME }}/mysql-fiber-go
       echo "${{ secrets.DOCKERHUB_PASSWORD }}" | docker login -u "${{ secrets.DOCKERHUB_USERNAME }}" --password-stdin
       docker push ${{ secrets.DOCKERHUB_USERNAME }}/mysql-fiber-go
```

Рисунок 4.11 – Файл робочого процесу для тестування взаємодії з БД MySQL

Джерело: побудовано автором (знімок екрану)

Чутливі дані (логін, пароль, айді тощо) необхідно оголошувати в secrets. Секрети створюються для кожного репозиторію окремо. Для цього необхідно Перейти в Settings -> Security -> Secrets and variables -> Actions. Створені секрети показані на рисунку 4.12.

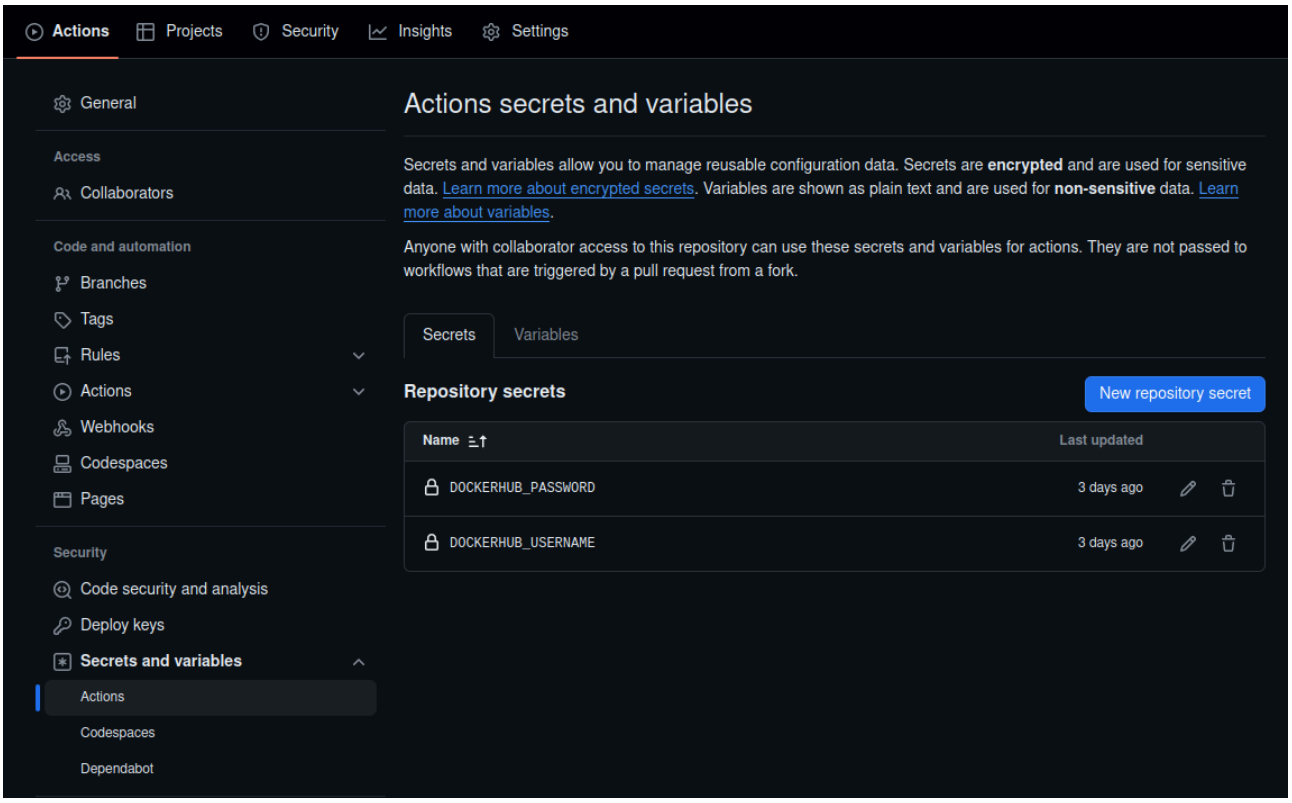


Рисунок 4.12 – Створені секрети для репозиторію mysql-fiber-go

Джерело: побудовано автором (знімок екрану)

Після збереження змін у коді та завантаження нового коміту, активується прописаний тригер (в даному випадку - пуш в гілку main) і починає виконуватися пайплайн. Процес та результат виконання пайплайнів показано на вкладці Actions (рис. 4.13-4.15).

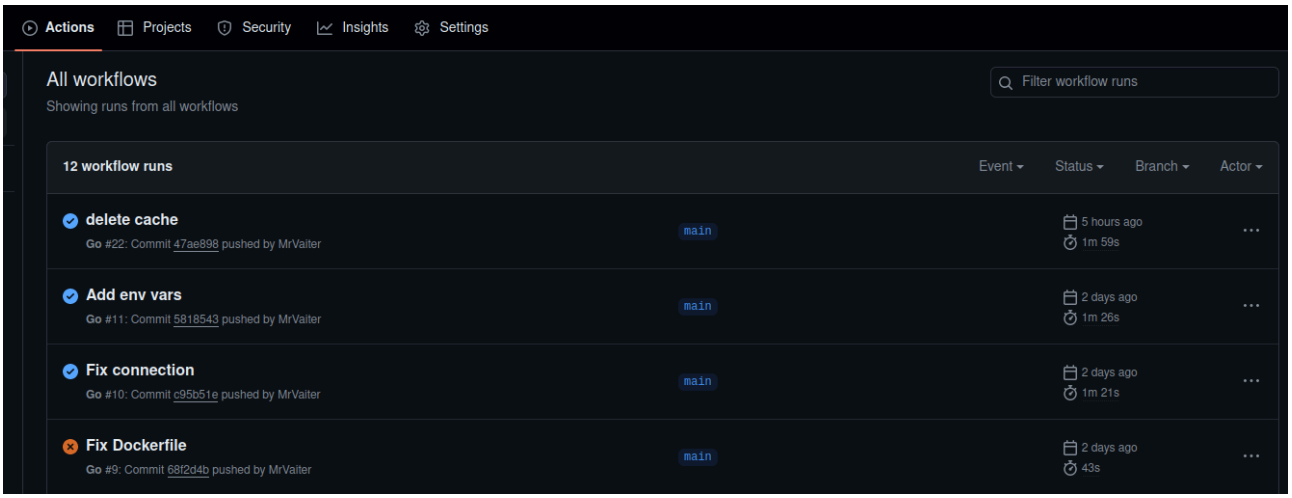


Рисунок 4.13 – Список робочих процесів відповідно до кожного коміта

Джерело: побудовано автором (знімок екрану)

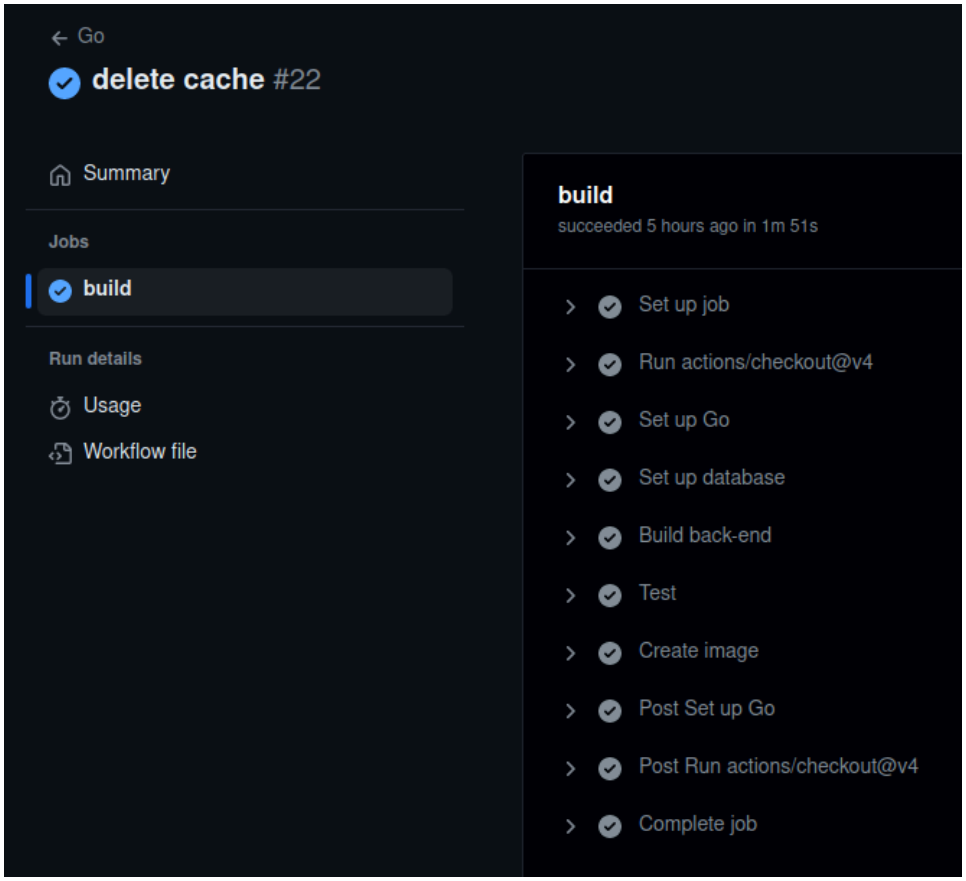


Рисунок 4.14 – Етапи робочого процесу

Джерело: побудовано автором (знімок екрану)

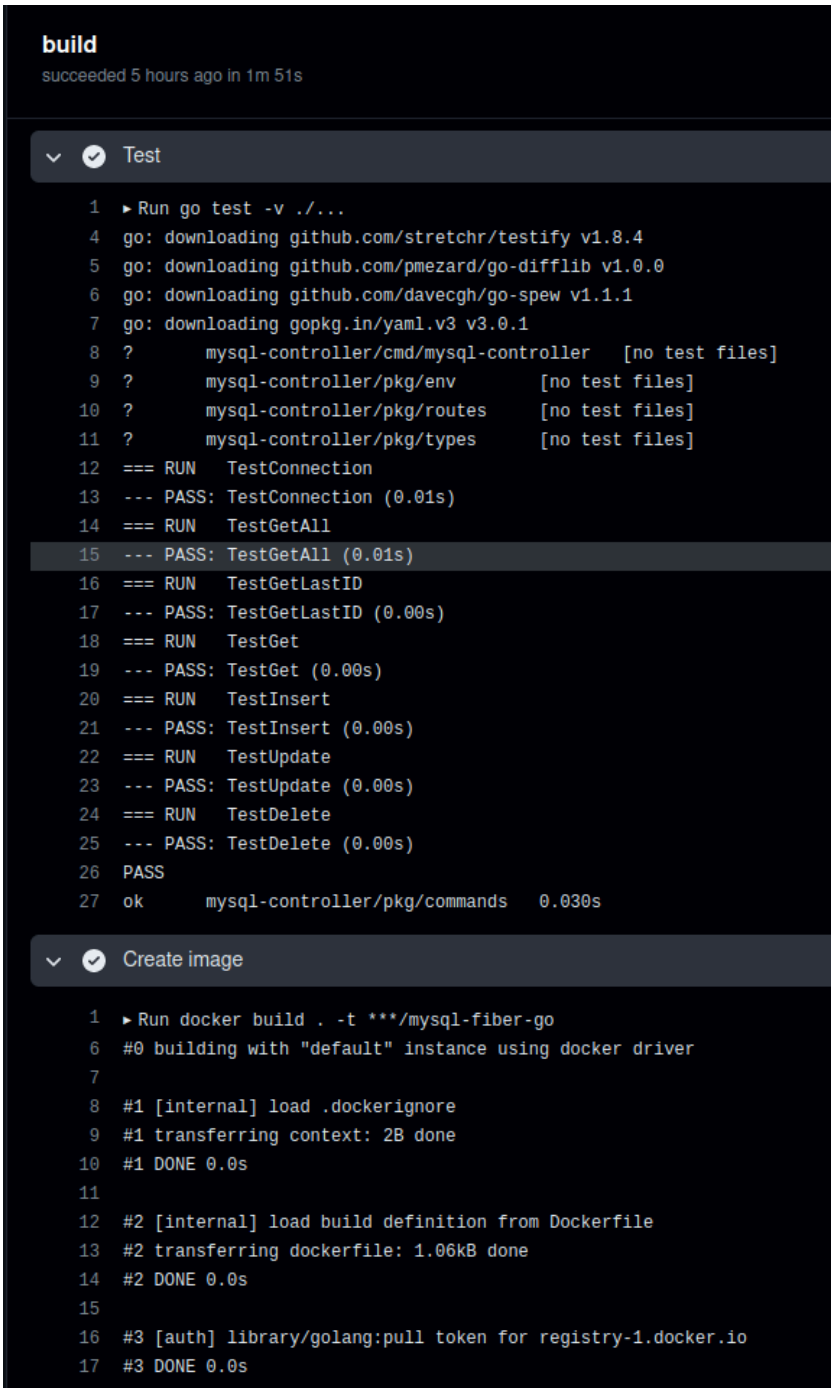


Рисунок 4.15 – Логи під час виконання кожного з етапів

Джерело: побудовано автором (знімок екрану)

Якщо на якомусь з етапів пайплайн отримує помилку, то робочий процес далі не йде, а пайплайн завершується зі статусом “Failure” (рис. 4.16-4.17).

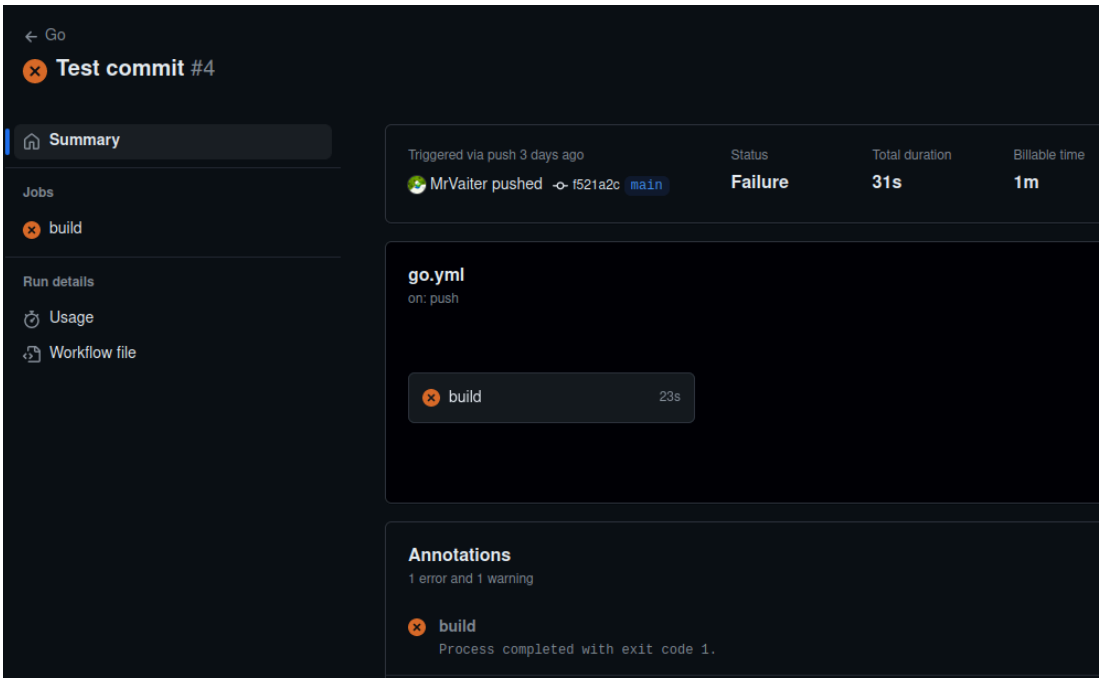


Рисунок 4.16 – Тестовий приклад робочого процесу з помилкою

Джерело: побудовано автором (знімок екрану)

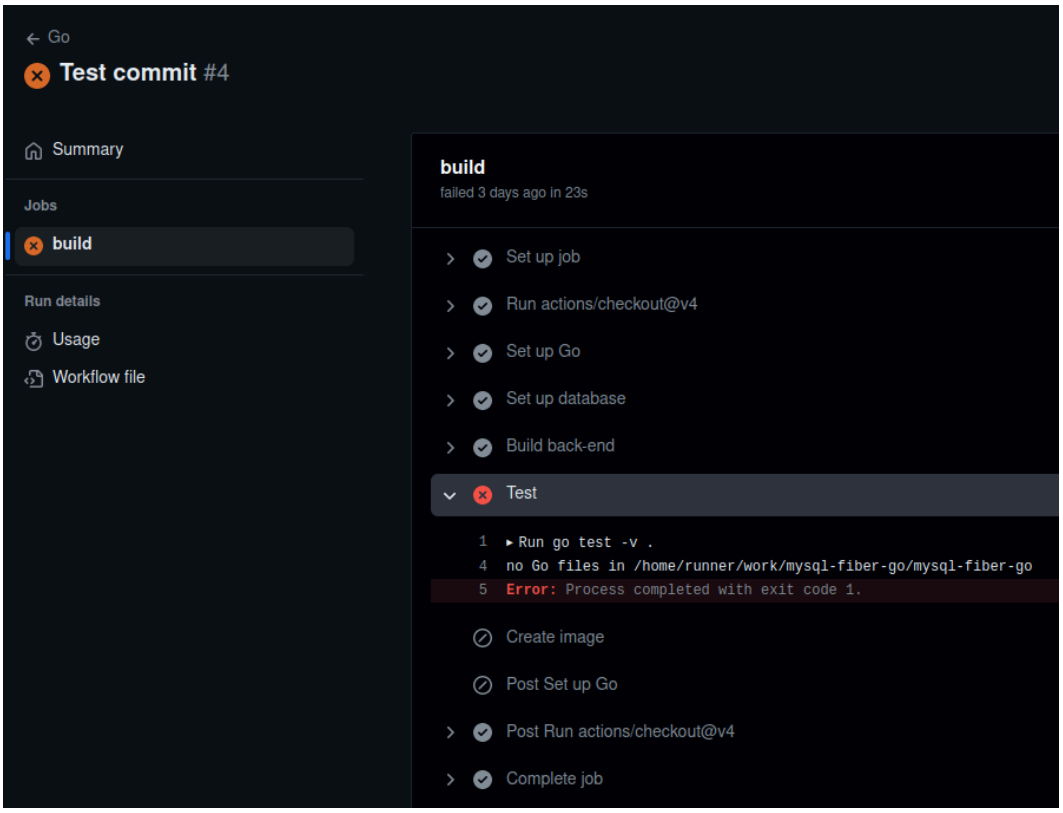


Рисунок 4.17 – Робочий процес показує на якому етапі сталась помилка

Джерело: побудовано автором (знімок екрану)

У разі успішного проходження всіх етапів робочого процесу, новостворений Docker-образ завантажується на Docker Hub (рис. 4.18). Для цього необхідно прописати Dockerfile в кореневій папці (рис 4.19-4.20). Це файл для попередньої роботи, набір інструкцій, який потрібний для запису образу. Він містить інформацію про операційну систему, обрану платформу, фреймворки, бібліотеки, інструменти, які потрібно встановити, та команди, які потрібно виконати. Для зменшення розміру Docker-образів (в 100 разів), було використано Multi-stage build [37] (рис. 4.21).

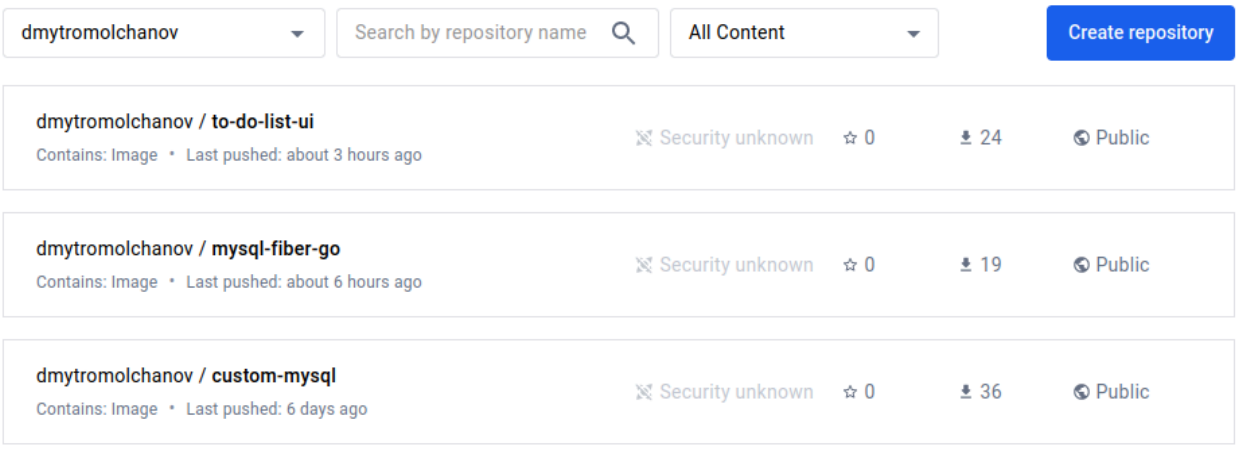


Рисунок 4.18 – Готові образи модулів вебдодатку

Джерело: побудовано автором (знімок екрану)

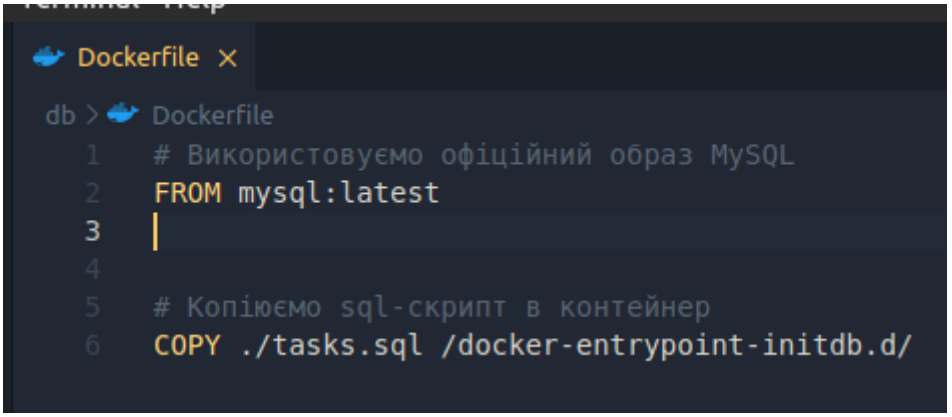


Рисунок 4.19 – Dockerfile для БД MySQL зі sql-скриптом для додавання базових даних в таблицю

Джерело: побудовано автором (знімок екрану)

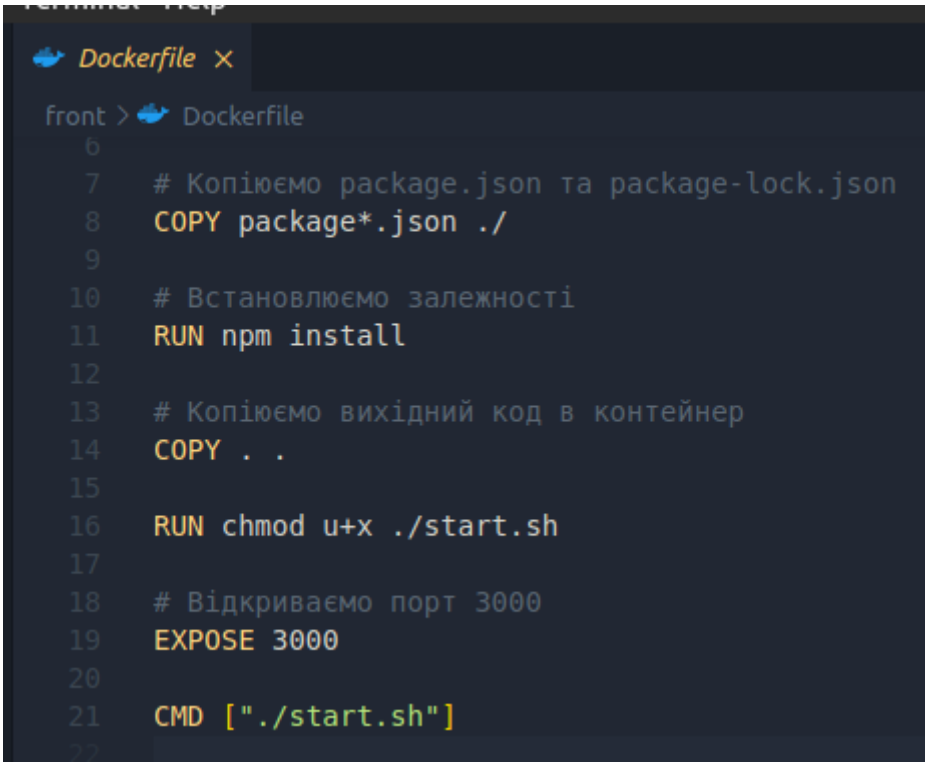


Рисунок 4.20 – Dockerfile для front-end частини з bash-скриптом для збірки та запуску

Джерело: побудовано автором (знімок екрану)

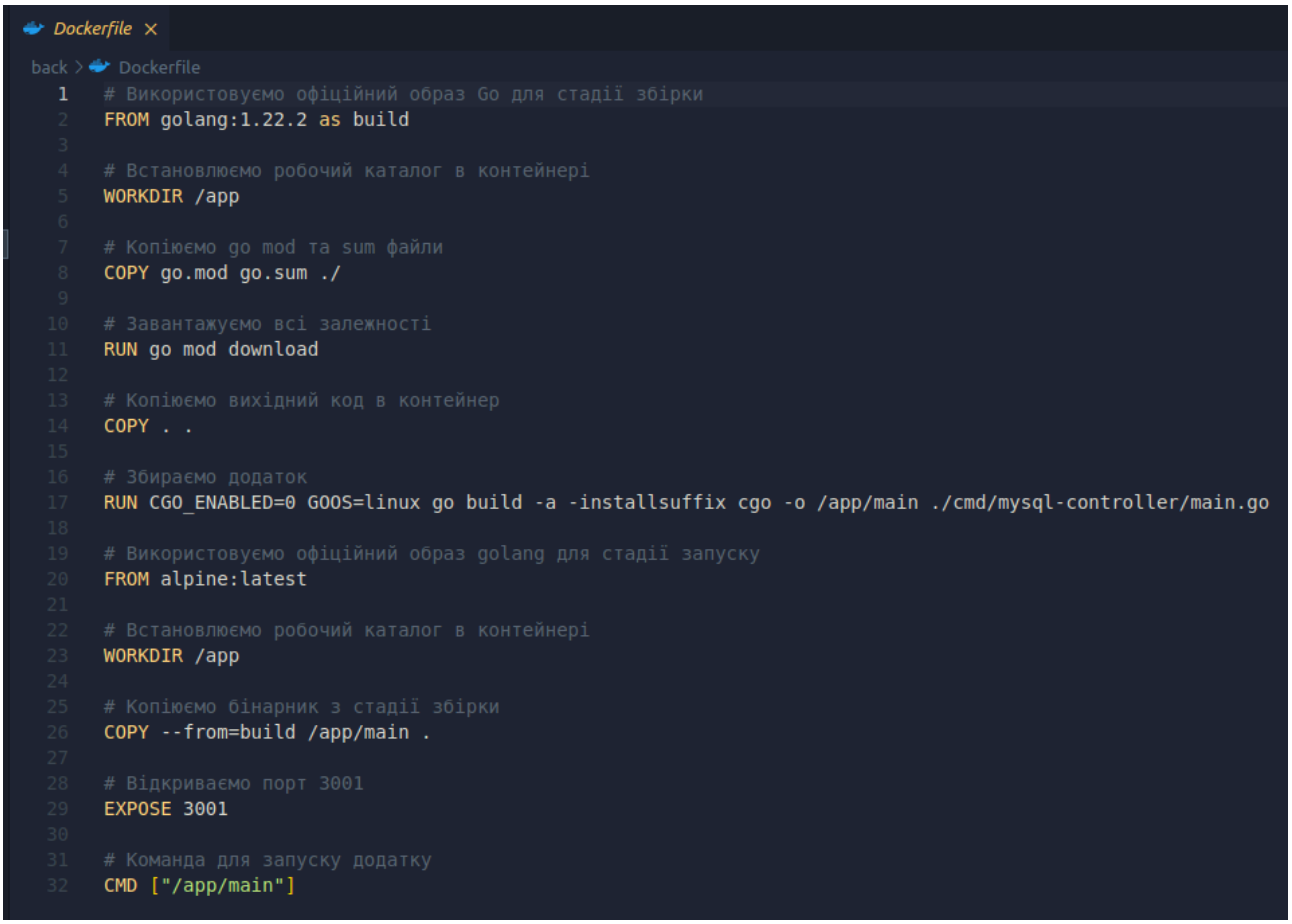


Рисунок 4.21 – Dockerfile для back-end модуля з використанням multi-stage build

Джерело: побудовано автором (знімок екрану)

4.4 Декларативний опис інфраструктури

Отримавши повністю протестовані Docker-образи, декларативно описуємо інфраструктуру інформаційної системи. Для цього мовою HCL (HashiCorp Configuration Language) описано необхідні ресурси Kubernetes, розподілені в відповідні файли Terraform. Було створено:

1. **main.tf:** в ньому описані основні модулі та їх залежності (рис. 4.22);
2. **providers.tf:** містить список усіх використаних провайдерів, їх версії та шляхи до файлів конфігурації (рис. 4.23);

3. **terraform.tfstate та terraform.tfstate.backup:** містять поточний стан інфраструктури та дані для відновлення минулого стану. Створюються автоматично після команди ініціалізації тераформу командою **terraform init**.

Рисунок 4.22 – Декларативний опис модулів Terraform

Джерело: побудовано автором (знімок екрану)

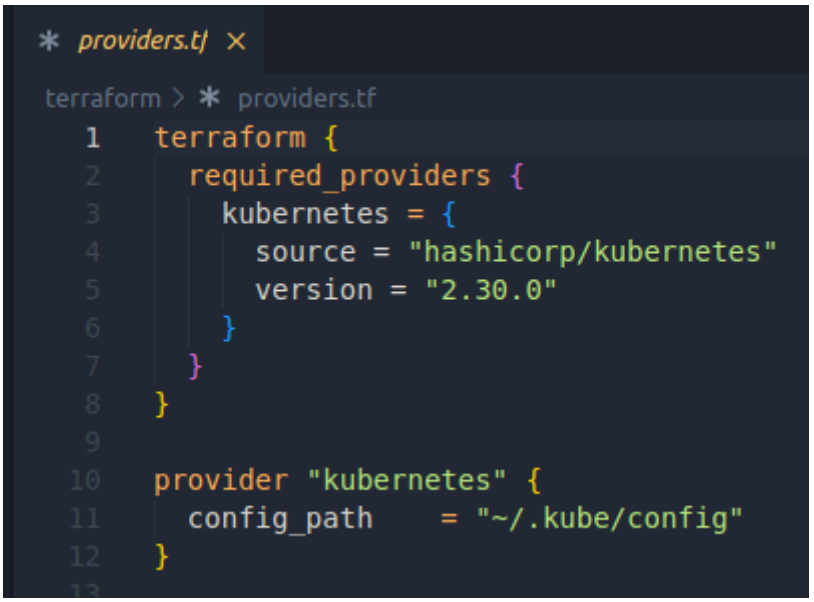


Рисунок 4.23 – Опис провайдерів Terraform

Джерело: побудовано автором (знімок екрану)

Для кожного з модулів був створений визначений набір ресурсів:

1. **variables.tf:** файл, який містить назви змінних та їх значення (рис 4.24);
2. **namespace.tf:** містить опис ресурсу простору назв (namespace), який використовується для відмежовування від ресурсів інших модулів (рис. 4.25);
3. **deployment.tf:** описує Kubernetes Deployment. Це особлива структура, яка містить назву контейнерів, кількість одночасних копій, який порт необхідно відкрити та який образ (image) використати (рис. 4.26);
4. **service.tf:** описує сервіс в середовищі Kubernetes. Він необхідний для взаємодії різних ресурсів між собою (Pod, Deployment, ReplicaSet) або надання можливості зовнішніх підключень (рис. 4.27).

Рисунок 4.24 – Опис змінних Terraform

Джерело: побудовано автором (знімок екрану)

namespace.tf

terraform > modules > mysql > namespace.tf

```
resource "kubernetes_namespace" "mysql" {
  metadata {
    name = var.namespace
  }
}
```

Рисунок 4.25 – Опис простору імен Kubernetes

Джерело: побудовано автором (знімок екрану)

deployment.tf

terraform > modules > mysql > deployment.tf

```
resource "kubernetes_deployment" "mysql" {
  metadata {
    name      = "to-do-list-db"
    namespace = var.namespace
  }
  spec {
    replicas = 1
    selector {
      match_labels = {
        app = var.label
      }
    }

    template {
      metadata {
        labels = {
          app = var.label
        }
      }
      spec {
        container {
          image = "dmytromolchanov/custom-mysql"
          name  = "database"
          port {
            container_port = 3306
          }
        }
      }
    }
  }
}
```

Рисунок 4.26 – Опис деплойменту Kubernetes

Джерело: побудовано автором (знімок екрану)

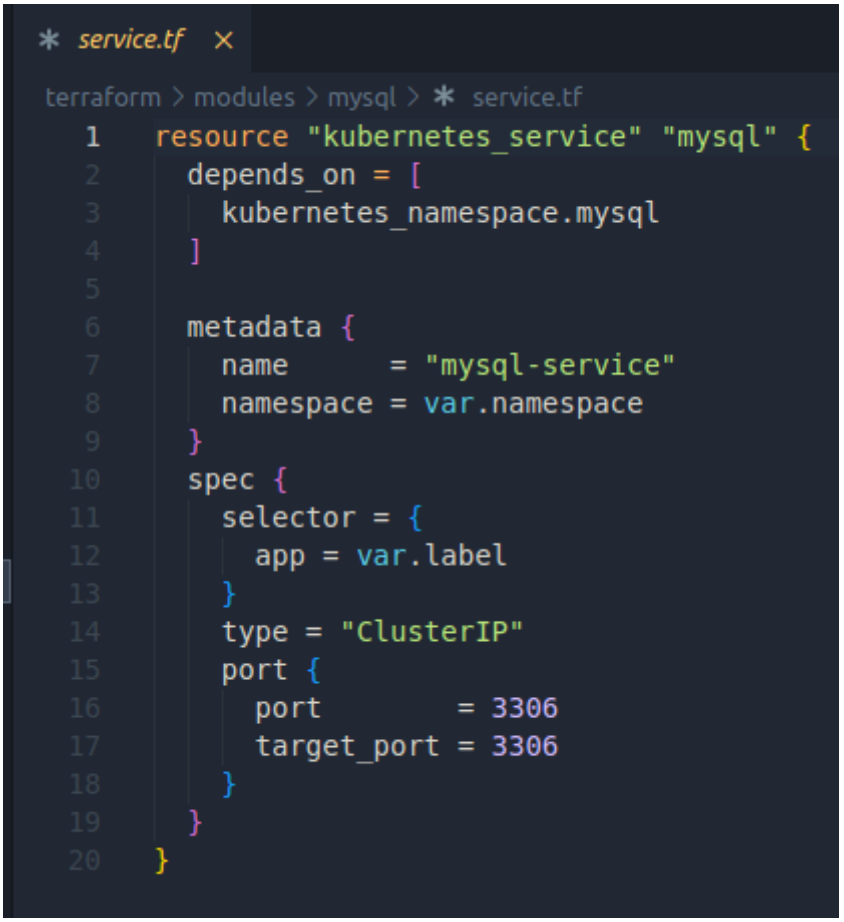


Рисунок 4.27 – Опис сервісу Kubernetes

Джерело: побудовано автором (знімок екрану)

Декларативний опис всіх модулів наведено в додатку Б.

4.5 Розгортання налаштованої системи

Для запуску локального кластера Kubernetes було використано minikube у взаємодії з докером. Запуск кластеру представлений на рисунку 4.28.

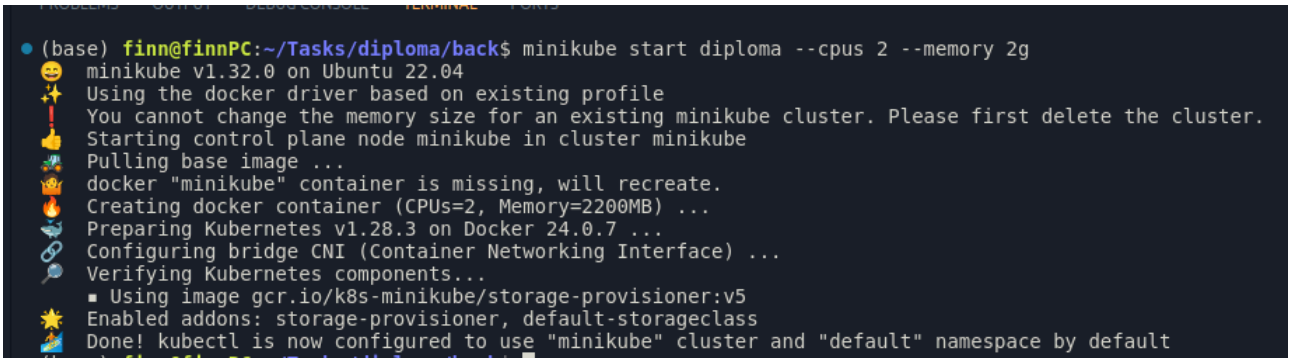


Рисунок 4.28 – Запуск кластера на 2 ядра процесора та 2 гігабайти оперативної пам’яті

Джерело: побудовано автором (знімок екрану)

Після запуску кластеру, в кореневій папці Terraform модулів запускаємо налаштування інфраструктури в середовищі Kubernetes (k8s). За це відповідає команда “terraform apply”. Після представлення запланованого стану системи, треба його затвердити написавши “yes”. Процес запуску представлений на рисунках 4.29-4.30.

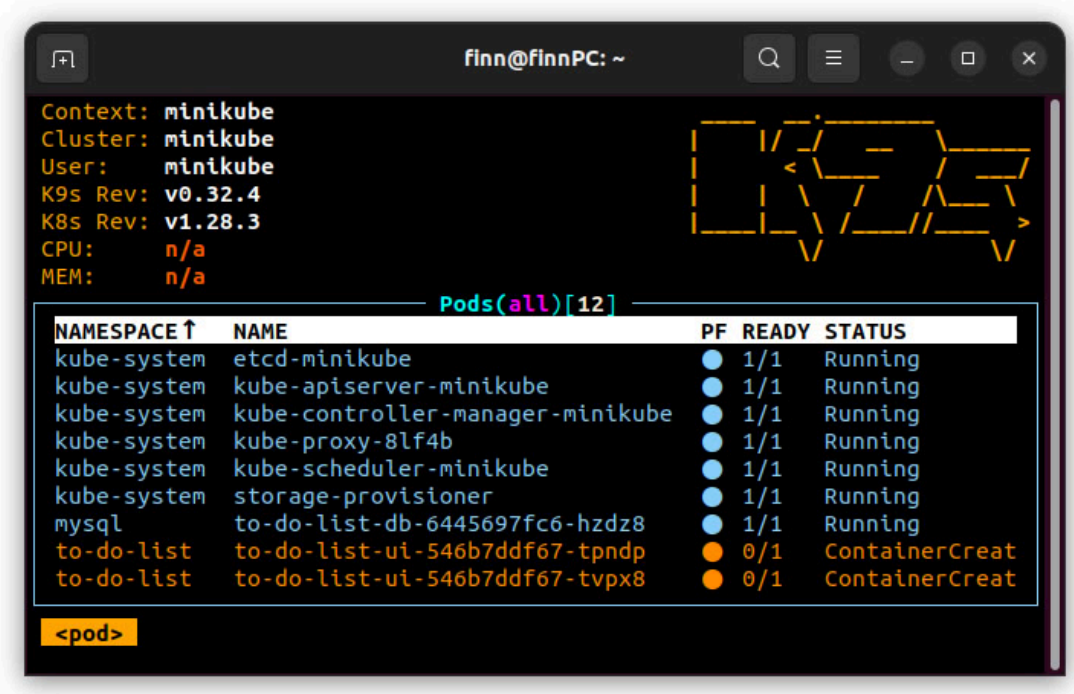


Рисунок 4.29 – Процес створення ресурсів, представлений за допомогою k9s

Джерело: побудовано автором (знімок екрану)

```
Plan: 9 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

module.mysql.kubernetes_namespace.mysql: Creating...
module.mysql.kubernetes_namespace.mysql: Creation complete after 0s [id=mysql]
module.mysql.kubernetes_deployment.mysql: Creating...
module.mysql.kubernetes_service.mysql: Creating...
module.mysql.kubernetes_service.mysql: Creation complete after 0s [id=mysql/mysql-service]
module.mysql.kubernetes_deployment.mysql: Creation complete after 8s [id=mysql/to-do-list-db]
module.fiber.kubernetes_namespace.fiber: Creating...
module.fiber.kubernetes_namespace.fiber: Creation complete after 0s [id=fiber]
module.fiber.kubernetes_deployment.fiber: Creating...
module.fiber.kubernetes_service.fiber: Creating...
module.fiber.kubernetes_service.fiber: Creation complete after 0s [id=fiber/fiber-service]
module.fiber.kubernetes_deployment.fiber: Still creating... [10s elapsed]
module.fiber.kubernetes_deployment.fiber: Creation complete after 15s [id=fiber/to-do-list-back]
module.react.kubernetes_namespace.ui: Creating...
module.react.kubernetes_namespace.ui: Creation complete after 0s [id=to-do-list]
module.react.kubernetes_deployment.ui: Creating...
module.react.kubernetes_service.ui: Creating...
module.react.kubernetes_service.ui: Creation complete after 1s [id=to-do-list/ui-service]
module.react.kubernetes_deployment.ui: Still creating... [3m11s elapsed]
module.react.kubernetes_deployment.ui: Still creating... [3m21s elapsed]
module.react.kubernetes_deployment.ui: Creation complete after 3m26s [id=to-do-list/to-do-list-ui]

Apply complete! Resources: 9 added, 0 changed, 0 destroyed.
```

Рисунок 4.30 – Процес створення ресурсів в середовищі k8s

Джерело: побудовано автором (знімок екрану)

Готова інфраструктура представлена на рисунку 4.31.

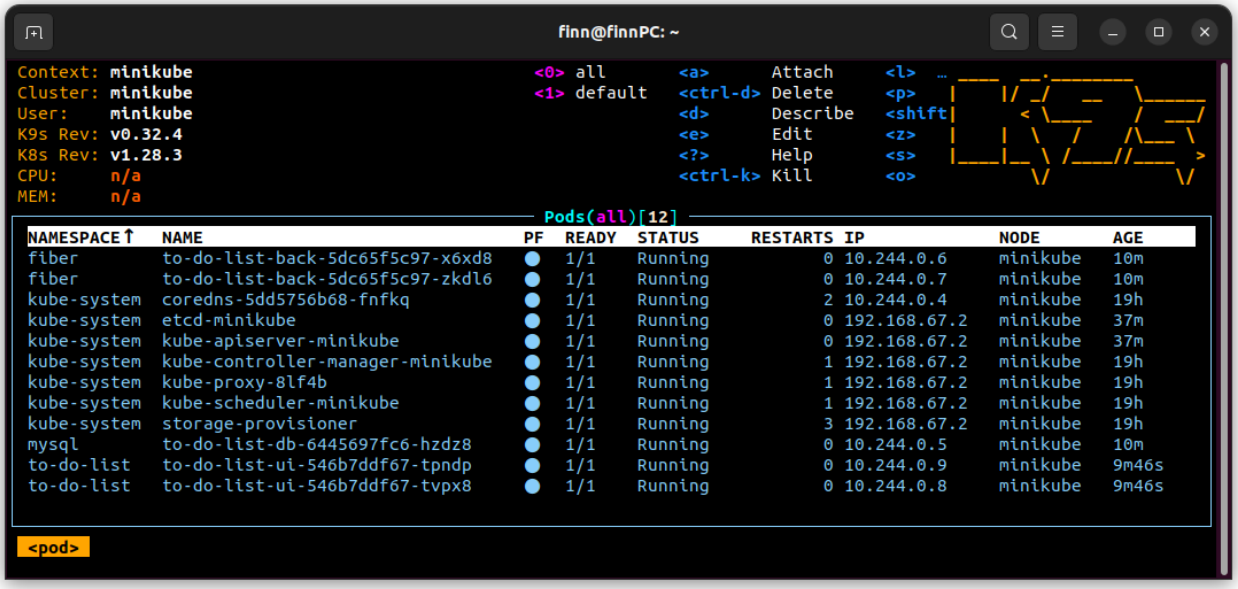


Рисунок 4.31 – Готова інфраструктура в середовищі k8s

Джерело: побудовано автором (знімок екрану)

4.6 Тестування налаштованої системи

Для тестування функції autohealing видалимо один із робочих подів комбінацією клавіш Ctrl + D. Процес видалення поду показано на рисунках 4.32-4.33.

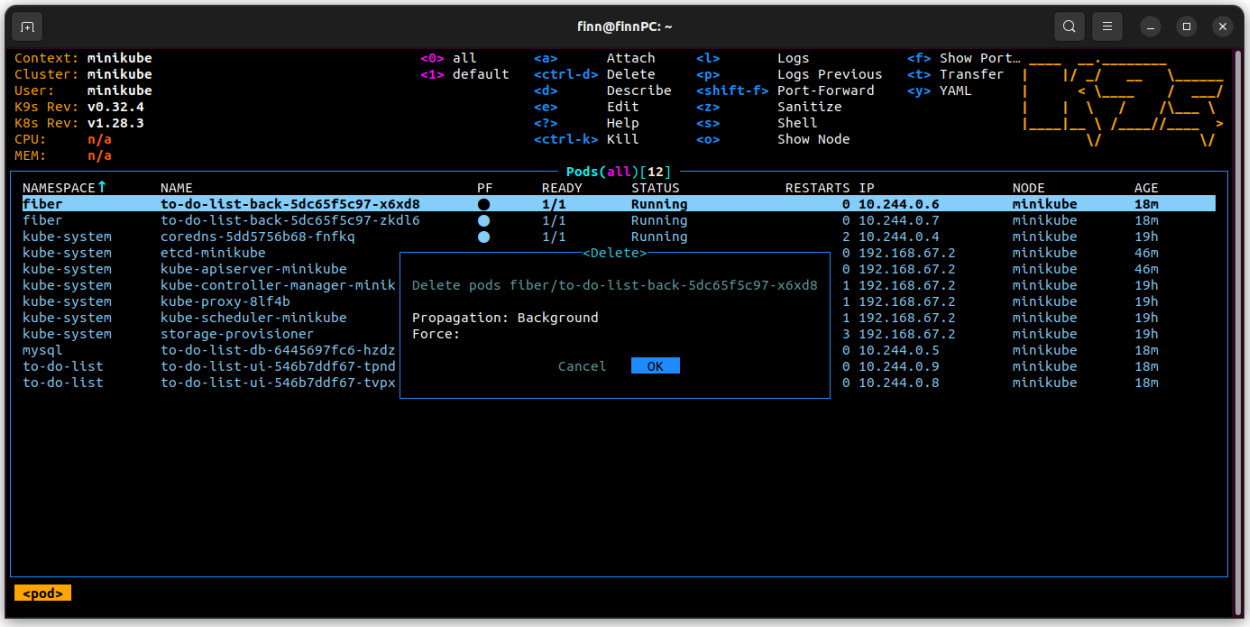


Рисунок 4.32 – Готова інфраструктура в середовищі k8s

Джерело: побудовано автором (знімок екрану)

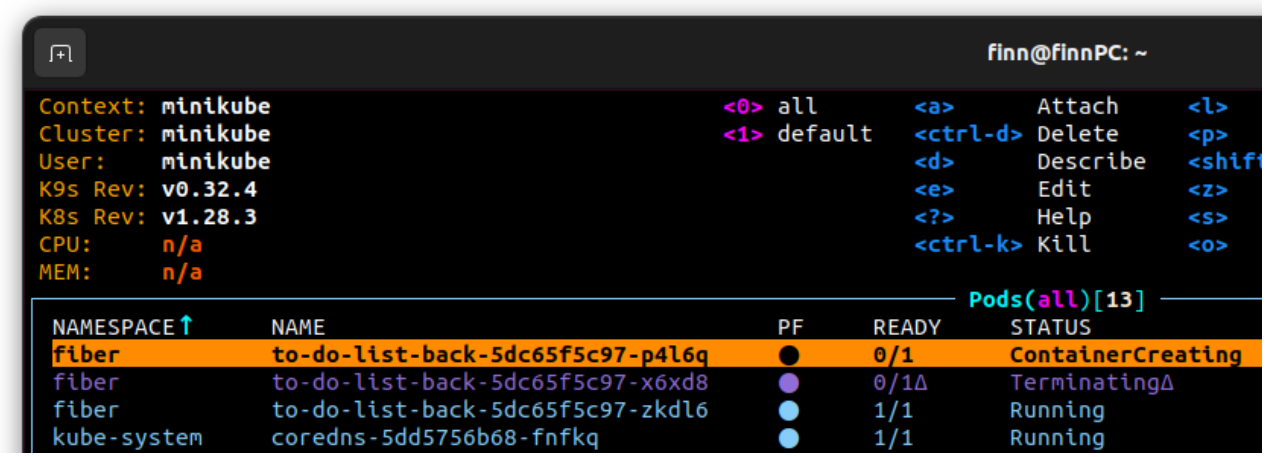


Рисунок 4.33 – Готова інфраструктура в середовищі k8s

Джерело: побудовано автором (знімок екрану)

В момент видалення поду, система Kubernetes моментально створила його копію, забезпечуючи необхідну кількість реплік, прописану в конфігурації Terraform файлу.

Для тестування зв’язку між подами, які знаходяться в різних просторах імен, виконаємо GET запит в одному з подів (контейнерів). Для цього виконуємо команду ssh в один з подів front-end модуля. Для цього треба обрати необхідний контейнер та натиснути кнопку S (рис. 4.34).

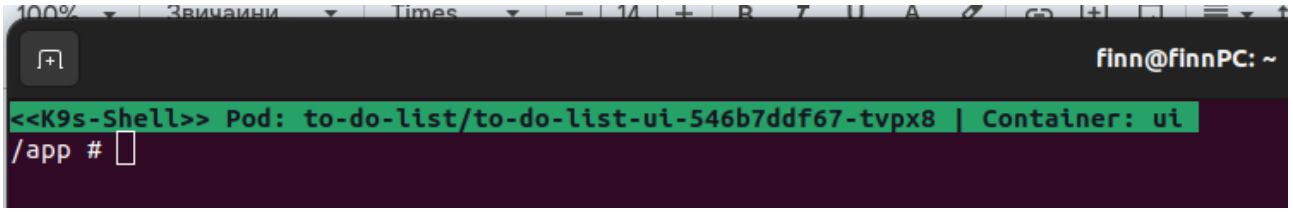


Рисунок 4.34 – Успішне підключення до контейнера

Джерело: побудовано автором (знімок екрану)

Виконаємо GET запит до back-end сервісу, щоб отримати дані через сервіс бази даних. Для цього виконаємо команду:

curl -X GET fiber-service.fiber.svc.cluster.local:3001/api/v1/getall

Результат виконання представлений на рисунку 4.35.

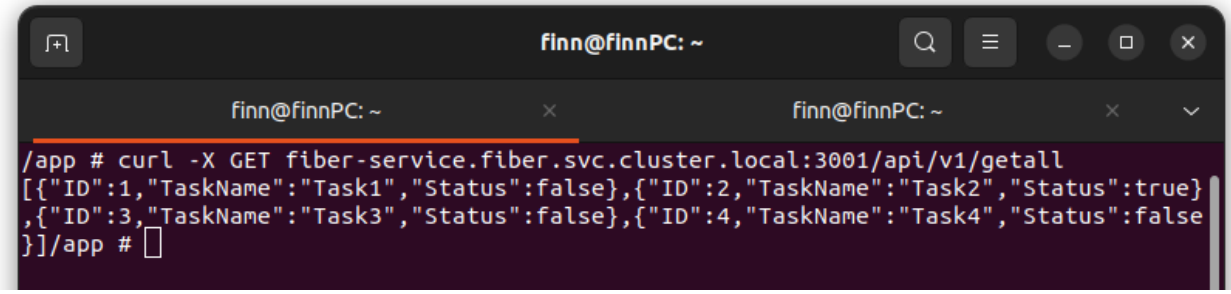


Рисунок 4.35 – Отримані дані від back-end поду

Джерело: побудовано автором (знімок екрану)

Для візуального тестування, треба отримати завчасно налаштований NodePort front-end сервісу, та IP кластеру. Вищевказані дані можна отримати відповідними командами ***kubectl get services -A*** та ***kubectl get nodes -owide***. Результат виконання команд наведений на рисунку 4.36.

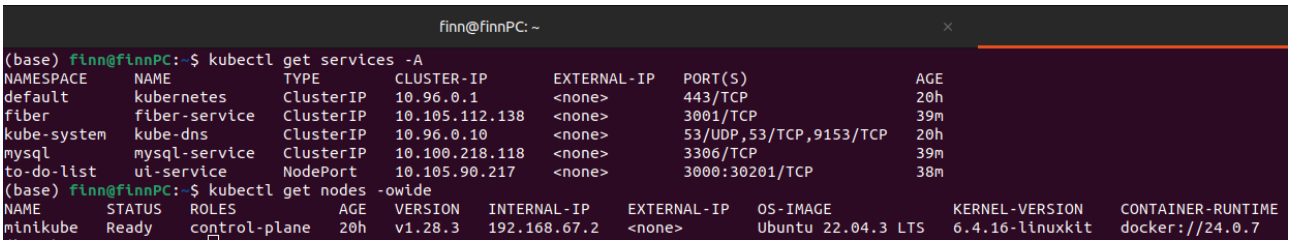


Рисунок 4.35 – Отримані дані від back-end поду

Джерело: побудовано автором (знімок екрану)

Значення в форматі <Cluster IP>:<Node Port> формують адресу сайту, на якому представлений користувацький інтерфейс додатку To-do-list (рис. 4.36).

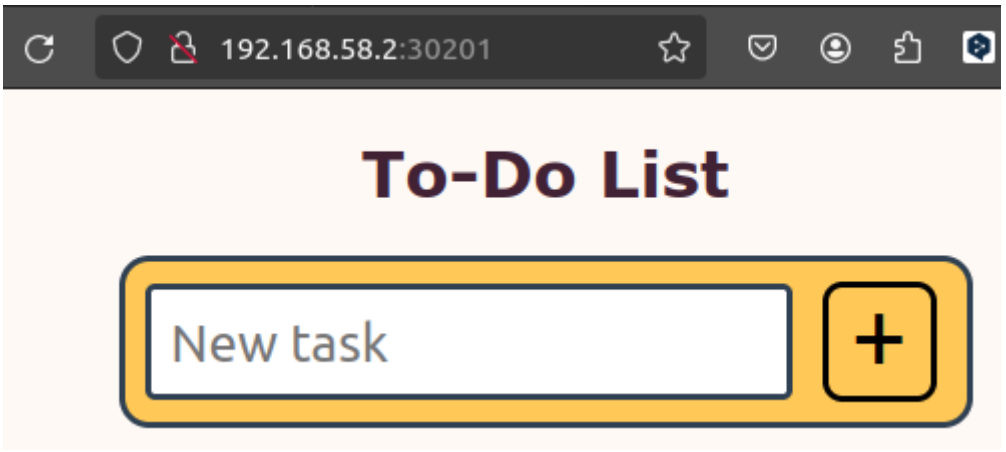


Рисунок 4.35 – Успішний доступ

Джерело: побудовано автором (знімок екрану)

4.7 Моніторинг інфраструктури

Для налаштування моніторингу стану інфраструктури в середовищі Kubernetes, необхідно виконати наступні команди:

- ***minikube -p diploma addons enable metrics-server*** – підключає аддон сервера з метриками до вашого профілю minikube;
- ***minikube dashboard --url -p diploma*** – створює необхідні контейнери в просторі імен kubernetes-dashboard і повертає посилання на dashboard, на якому наявна інформація про всі присутні ресурси.

Інформація про ресурси та стан кластера, подів та сервісів наведені на рисунках 4.36-4.39.

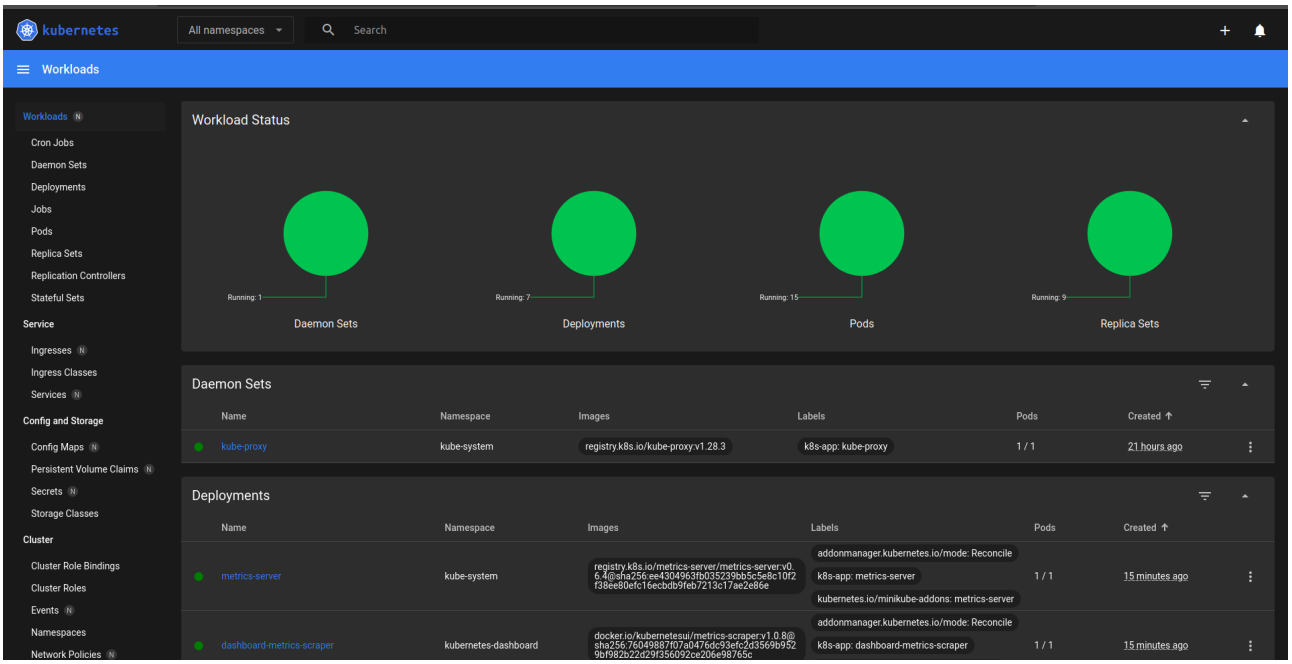


Рисунок 4.36 – Інформація про всі ресурси кластера

Джерело: побудовано автором (знімок екрану)

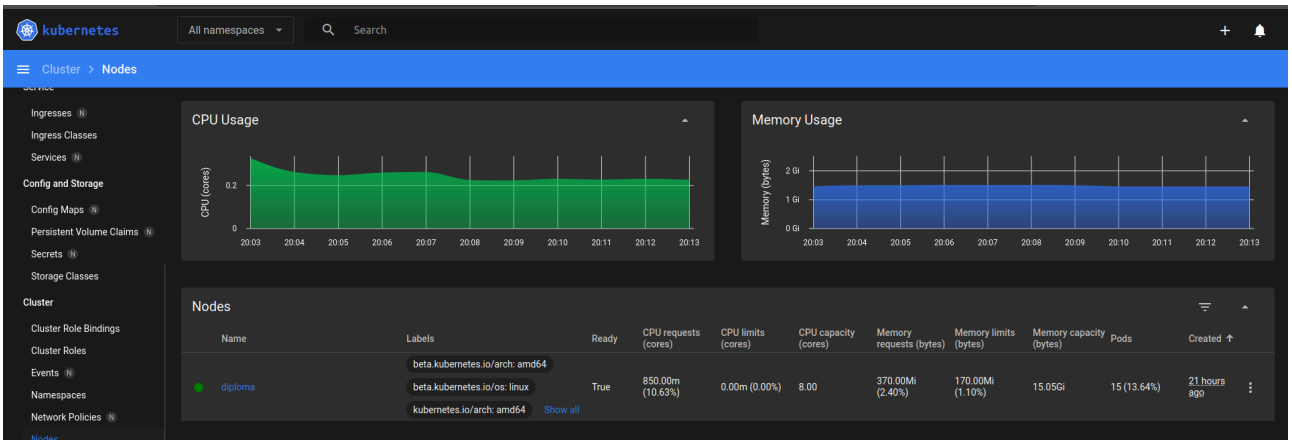


Рисунок 4.37 – Інформація про стан кластера

Джерело: побудовано автором (знімок екрану)

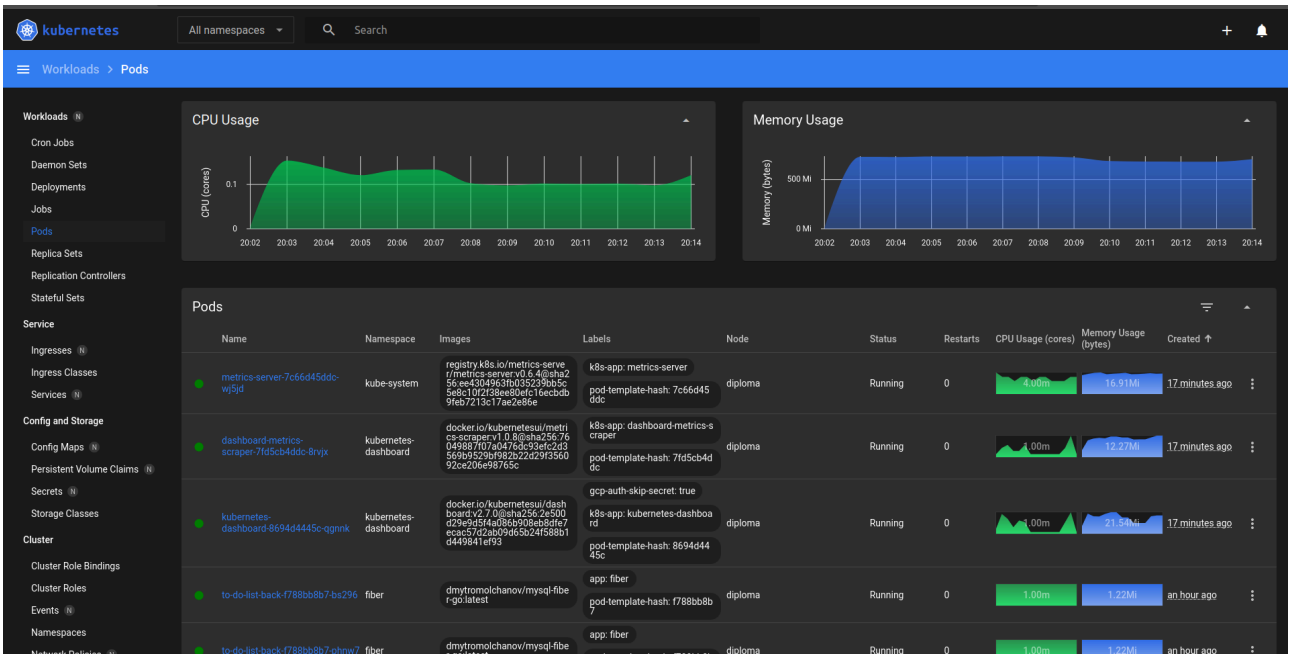


Рисунок 4.38 – Інформація про усі поди кластера

Джерело: побудовано автором (знімок екрану)

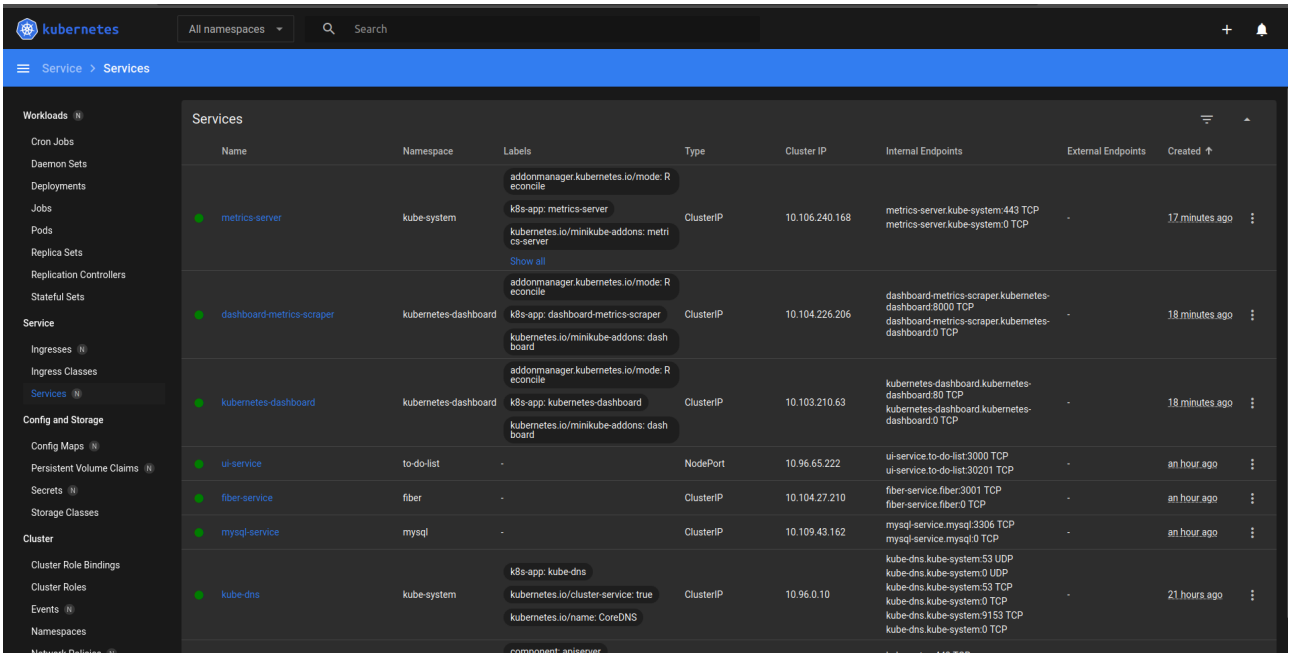


Рисунок 4.39 – Інформація про усі сервіси кластера

Джерело: побудовано автором (знімок екрану)

ВИСНОВОК

У ході виконання магістерської роботи була проведена дослідницька робота та розроблена інформаційна технологія забезпечення робастності вебдодатків. Основною метою роботи було створення інструменту, що дозволяє ефективно керувати та забезпечувати стабільну роботу вебдодатків у різних умовах. Для досягнення цієї мети були вирішені наступні задачі, а саме:

- Проведений аналіз дозволив визначити актуальність застосування інформаційної технології для забезпечення робастності вебдодатків.
- Огляд сучасних публікацій та аналіз аналогів дозволили визначити оптимальний підхід до реалізації інформаційної технології.
- Було розроблено алгоритм налаштування інформаційної системи з використанням інструментарію Kubernetes.
- Реалізовано функціонал інформаційної технології та проведено його успішне тестування.

Отримані результати свідчать про практичну цінність розробленої технології, що полягає у підвищенні продуктивності та надійності роботи вебдодатків. Впровадження даної технології може сприяти оптимізації процесу розробки, тестування та підтримки вебдодатків, а також допомогти уникнути перевантаження співробітників та непередбачених ситуацій, що можуть загрожувати життєдіяльності системи.

Дана інформаційна технологія була розроблена з використанням Kubernetes та Docker для контейнеризації та оркестрації контейнеризованих додатків, GitHub CI/CD для створення CI/CD pipelines, Terraform для декларативної конфігурації, k9s для демонстрації стану Kubernetes кластера та K8s Dashboard для моніторингу стану інформаційної системи.

Отже, робота виявилася успішною у вирішенні поставлених завдань та відповідає поставленим цілям. Впровадження розробленої технології може мати

значний вплив на покращення ефективності та надійності вебдодатків у практичних умовах їх експлуатації.

Наукова новизна даного дослідження полягає у впровадженні інформаційної технології забезпечення робастності вебдодатків з використанням інструментарію Kubernetes. Цей підхід відрізняється від традиційних методів розробки та підтримки вебдодатків, оскільки базується на використанні контейнеризації і автоматизації розгортання та управління. Використання Kubernetes дозволяє ефективно керувати та масштабувати вебдодатки, забезпечуючи їх стабільну роботу в різних умовах і в умовах збільшеної навантаженості. Такий підхід є передовим у контексті сучасних вимог до розгортання та підтримки вебдодатків і може мати значний вплив на подальший розвиток галузі інформаційних технологій.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Безпека вебдодатків: найкращі практики та вразливості. itproger.com. URL: https://itproger.com/ua/news/bezopasnost-veb-prilozheniy-luchshie-praktiki-i-uyazvimosti (дата звернення: 01.04.2024);
2. Що таке Kubernetes?. Kubernetes. URL: https://kubernetes.io/uk/docs/concepts/overview/what-is-kubernetes/ (дата звернення: 01.04.2024);
3. 7 reasons kubernetes for devops is important. Turing Blog. URL: https://www.turing.com/blog/importance-of-kubernetes-for-devops/ (дата звернення: 01.04.2024);
4. How kubernetes is transforming devops and 6 best practices. Aqua. URL: https://www.aquasec.com/cloud-native-academy/kubernetes-101/kubernetes-devops/ (дата звернення: 04.04.2024);
5. Best of 2021 - devops and kubernetes: a perfect match?. Cloud Native Now. URL: https://cloudnativenow.com/features/devops-and-kubernetes-a-perfect-match/ (дата звернення: 04.04.2024);
6. Swarm mode key concepts. Docker Documentation. URL: https://docs.docker.com/engine/swarm/key-concepts/ (дата звернення: 05.04.2024);
7. OpenShift Container Platform overview | Getting started | OpenShift Container Platform 4.10. Home | Official Red Hat OpenShift Documentation. URL: https://docs.openshift.com/container-platform/4.10/getting_started/openshift-overview.html. (дата звернення: 05.04.2024);

8. What is Rancher? | Rancher. What is Rancher? | Rancher. URL: https://ranchermanager.docs.rancher.com/?_gl=1*8vtitv*_ga*MTM0MDYxMDEzMi4xNzEzMjE3NDU5*_ga_Y7SFXF9L00*MTcxMzIxNzQ1OC4xLjEuMTcxMzIxNzQ3Mi40Ni4wLjA
(дата звернення: 05.04.2024);

9. Nayan V. Understanding HashiCorp Configuration Language. Medium. URL: https://proviveknayan.medium.com/understanding-hashicorp-configuration-language-53f3c0f085b7
(дата звернення: 05.04.2024);

10. The Complete Guide To Understand IDEF Diagram | EdrawMax Online. Edrawsoft. URL: https://www.edrawmax.com/article/the-complete-guide-to-understand-idef-diagram.html
(дата звернення: 08.04.2024);

11. IBM Documentation. IBM in Deutschland, Österreich und der Schweiz. URL: https://www.ibm.com/docs/en/rational-soft-arch/9.6.1?topic=diagrams-use-case
(дата звернення: 08.04.2024);

12. What is Prometheus?. New Relic. URL: https://newrelic.com/blog/best-practices/what-is-prometheus
(дата звернення: 08.04.2024);

13. Grafana dashboards overview | Grafana documentation. Grafana Labs. URL: https://grafana.com/docs/grafana/latest/fundamentals/dashboards-overview/
(дата звернення: 08.04.2024);

14. Configure Liveness, Readiness and Startup Probes. Kubernetes. URL: https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-startup-probes/
(дата звернення: 10.04.2024);

15. Autoscaling in Kubernetes. Kubernetes. URL: https://kubernetes.io/blog/2016/07/autoscaling-in-kubernetes/
(дата звернення: 10.04.2024);

16. Features • GitHub Actions. GitHub. URL: https://github.com/features/actions
(дата звернення: 10.04.2024);

17. How to build a CI/CD pipeline with GitHub Actions in four simple steps. The GitHub Blog. URL: https://github.blog/2022-02-02-build-ci-cd-pipeline-github-actions-four-steps/
(дата звернення: 10.04.2024);

18. Git Commit | Atlassian Git Tutorial. Atlassian. URL: https://www.atlassian.com/git/tutorials/saving-changes/git-commit
(дата звернення: 10.04.2024);

19. MindTools | Home. Develop your personal wellbeing and career skills - Mind Tools - Mind Tools. URL: https://www.mindtools.com/a4wo118/smart-goals
(дата звернення: 15.04.2024).

20. What is Work Breakdown Structure (WBS) Diagram? - Edraw. Edraw Software: Unlock Diagram Possibilities. URL: https://www.edrawsoft.com/what-is-work-breakdown-structure-diagram.html
(дата звернення: 15.04.2024);

21. Organization Breakdown Structure (OBS) - Upland Software. URL: https://www.edrawsoft.com/what-is-work-breakdown-structure-diagram.html
(дата звернення: 15.04.2024);

22. Test Automation / M. Polo et al. IEEE Software. 2013. Vol. 30, no. 1. P. 84–89. URL: https://doi.org/10.1109/ms.2013.15
(дата звернення: 08.04.2024);

23. Zhou N., Zhou H., Hoppe D. Containerisation for High Performance Computing Systems: Survey and Prospects. IEEE Transactions on Software

Engineering. 2022. P. 1–20. URL: https://doi.org/10.1109/tse.2022.3229221
(дата звернення: 08.04.2024);

24. DevOps / C. Ebert et al. IEEE Software. 2016. Vol. 33, no. 3. P. 94–100. URL: https://doi.org/10.1109/ms.2016.68
(дата звернення: 08.04.2024);

25. Overview. Kubernetes. URL: (дата звернення: 10.04.2024); https://kubernetes.io/docs/concepts/overview/#why-you-need-kubernetes-and-what-can-it-do Overview of get started. Docker Documentation. URL: https://docs.docker.com/guides/get-started/
(дата звернення: 10.04.2024);

26. About GitHub and Git - GitHub Docs. GitHub Docs. URL: https://docs.github.com/en/get-started/start-your-journey/about-github-and-git (дата звернення: 10.04.2024);

27. Understanding GitHub Actions - GitHub Docs. GitHub Docs. URL: https://docs.github.com/en/actions/learn-github-actions/understanding-github-actions
(дата звернення: 10.04.2024);

28. K9s - Manage Your Kubernetes Clusters In Style. URL: https://k9scli.io/
(дата звернення: 10.04.2024);

29. Argo CD - Declarative GitOps CD for Kubernetes. URL: https://argo-cd.readthedocs.io/en/stable/
(дата звернення: 10.04.2024);

30. About Grafana | Grafana documentation. Grafana Labs. URL: https://grafana.com/docs/grafana/latest/introduction/
(дата звернення: 10.04.2024);

31. Quick Start – React. React. URL: https://react.dev/learn
(дата звернення: 15.04.2024);

32. Welcome . Fiber. URL: https://docs.gofiber.io/
(дата звернення: 15.04.2024);

33. General Information. MySQL Documentation. URL: https://dev.mysql.com/doc/refman/8.0/en/introduction.html
(дата звернення: 15.04.2024);

34. What is Terraform?. Terraform/HashiCorp Developer. URL: https://developer.hashicorp.com/terraform/intro
(дата звернення: 15.04.2024);

35. GitHub - stretchr/testify: A toolkit with common assertions and mocks that plays nicely with the standard library. GitHub. URL: https://github.com/stretchr/testify
(дата звернення: 15.04.2024);

36. Multi-stage builds. Docker Documentation. URL: https://docs.docker.com/build/building/multi-stage/
(дата звернення: 15.04.2024);

37. Welcome!. minikube. URL: https://minikube.sigs.k8s.io/docs/
(дата звернення: 15.04.2024);

38. Dashboard. minikube. URL: https://minikube.sigs.k8s.io/docs/handbook/dashboard/
(дата звернення: 15.04.2024);

39. Gantt Charts | Atlassian. Atlassian. URL: https://www.atlassian.com/agile/project-management/gantt-chart
(дата звернення: 15.04.2024).

ДОДАТОК А

А1. Планування робіт

На сучасному етапі розвитку ІТ-галузі, інформаційні технології все більше потребують автоматизації. Особливо важливою вона є для розробників, які прагнуть оптимізувати робочі процеси та підвищити ефективність команд. Автоматизація рутинних задач, особливо тих, що вимагають значного часу та мануальної праці, відкриває нові можливості для зосередження уваги на більш складних та стратегічних завданнях.

Інформаційна технологія є фундаментальним інструментом для підтримки оперативної розробки та тестування, відіграючи значну роль в успішній реалізації проектів і подальшій їх підтримці.

А2. Деталізація мети методом SMART.

Чітке визначення цілей на етапі концептуального проектування є ключовим фактором для забезпечення ефективного та високоякісного виконання проекту. Використання методу SMART для деталізації цілей проекту дозволяє структурувати та ясно визначити необхідні параметри, що сприяє кращому планування та виконанню [19]. Результати цього процесу відображено у таблиці А.1

Таблиця А.1 – Деталізація мети проекту методом SMART

Specific(Конкретна)	Розробити та імплементувати інформаційну технологію забезпечення робастності вебдодатків, задля оптимізації процесів розробки, тестування та подальшої підтримки.
Measurable(Вимірювана)	Результатом створення дипломного проекту є інформаційна технологія забезпечення робастності вебдодатків.
Achievable(Досяжна)	Для досягнення мети проекту необхідні знання системи Kubernetes, знання в сфері контейнеризації та досвід роботи з Docker, навички побудови CI/CD пайплайнів, уміння створювати декларативну конфігурацію для Terraform або Pulumi та навичок створення документації.
Relevant(Реалістична)	Розроблена інформаційна технологія дозволить автоматизувати процес тестування, контейнеризації та розгортання вебдодатків, покращить ефективність розробки, звільняє час розробників та тестувальників для зосередження на більш складних аспектах проекту.

Time-framed(Обмежена у часі)	Термін досягнення мети проекту визначено з замовником і дорівнює 3 місяці.

Джерело: побудовано автором

А3. Планування змісту робіт.

Work Breakdown Structure(WBS) – це інструмент розподілу проекту на керовані компоненти, що допомагає керівникам проектів і командам у визначенні і структуруванні обсягу робіт. Мета розробки WBS полягає у створенні чіткої, детальної та ієрархічної структури. WBS структурує роботу у формі ієрархії, де на верхньому рівні цієї ієрархії розташовується кінцевий продукт, а кожен наступний, більш нижній рівень, представляє собою більш деталізоване визначення робочих задач. Процес декомпозиції завдань триває до того моменту, поки вони не досягають розміру, який є оптимальним для ефективного управління та контролю, але водночас достатньо великого, щоб мати практичне значення [20]. На рисунку А.1 представлено WBS для проекту розробки інформаційної технології забезпечення робастності вебдодатків.

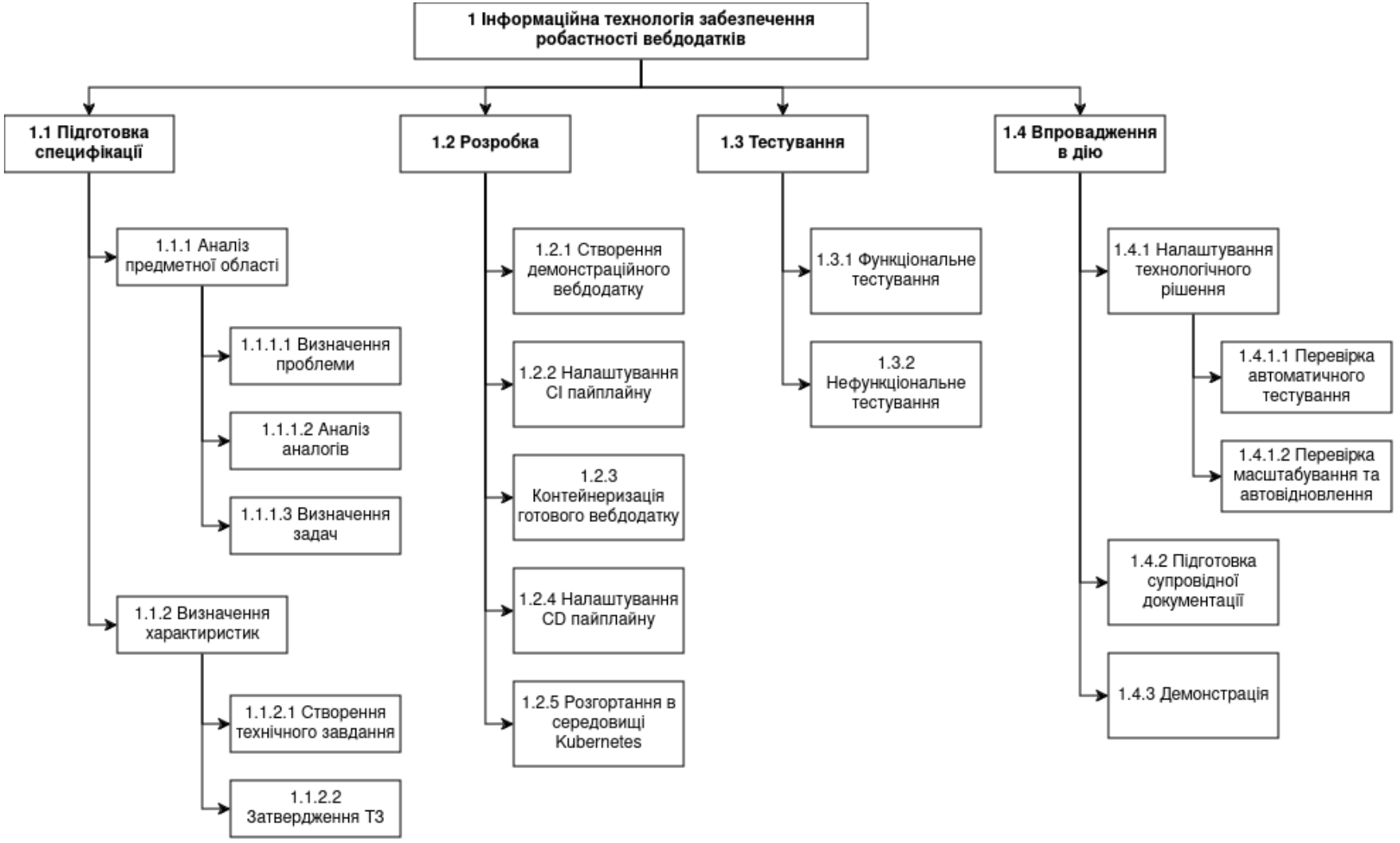


Рисунок А.1 – WBS-структура робіт проекту

Джерело: побудовано автором

А4. Планування структури виконавців.

Розробка організаційної структури виконавців (OBS) є наступним кроком після декомпозиції процесів за допомогою WBS. OBS або Organization Breakdown Structure – це структура, яка застосовується для демонстрації ієрархії команди імплементації проекту, і з’ясовує як організована команда та розподіл ресурсів у відповідності з метою виконання проекту [21]. На рисунку А.2 зображена організаційна структура планування робіт проекту. Дані про учасників проекту подано у таблиці А.2.

Таблиця А.2 – Виконавці проекту

Роль	**Ім’я**	**Проектна роль**
Розробник	Молчанов Д.А.	Розробка демонстраційного вебдодатку, налаштування CI/CD пайплайнів, контейнеризація на розгортання вебдодатку в середовищі Kubernetes
Тестувальник	Молчанов Д.А.	Проведення тестування програмного рішення.
Проектувальник	Молчанов Д.А	Проектування структури інформаційної системи та її взаємозв’язків.
Керівник проекту	Федотова Н.А.	Створення завдання на розробку проекту, рецензія імплементованого програмного рішення.
Менеджер	Федотова Н.А.	Контроль над дотриманням дедлайну, розподіл ресурсів та задач на розробку проекту.

		Проведення аналізу та збору інформації.

Джерело: побудовано автором

А5. Діаграма Ганта

Розробка календарного плану робіт є ключовою складовою управління проектом. Цей план представляє собою діаграму з розподілом часових рамок для кожного завдання, що сприяє точному визначенню загальної тривалості проекту з урахуванням доступних ресурсів [40].

Календарний графік проекту представлено на рисунку А.3.

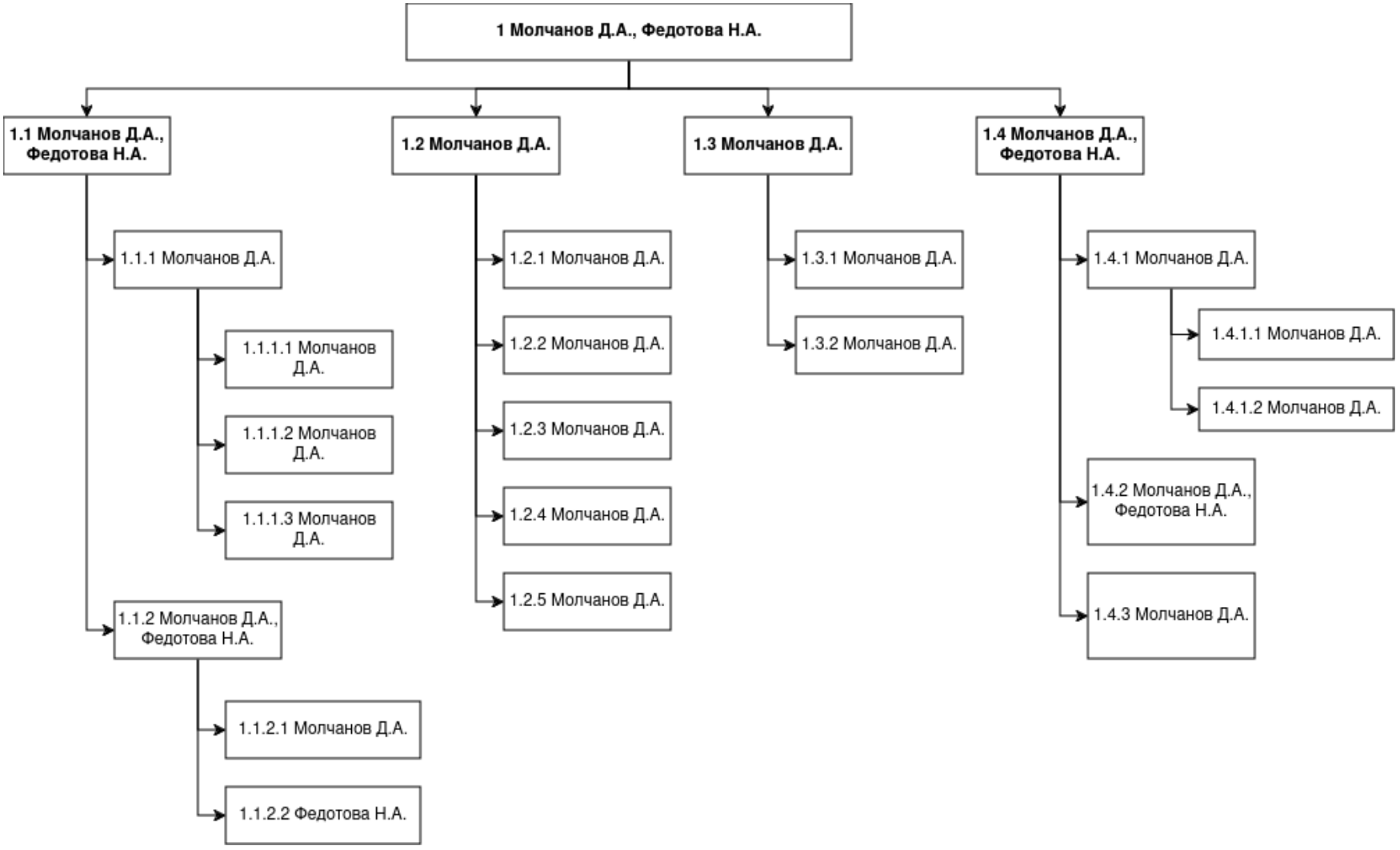
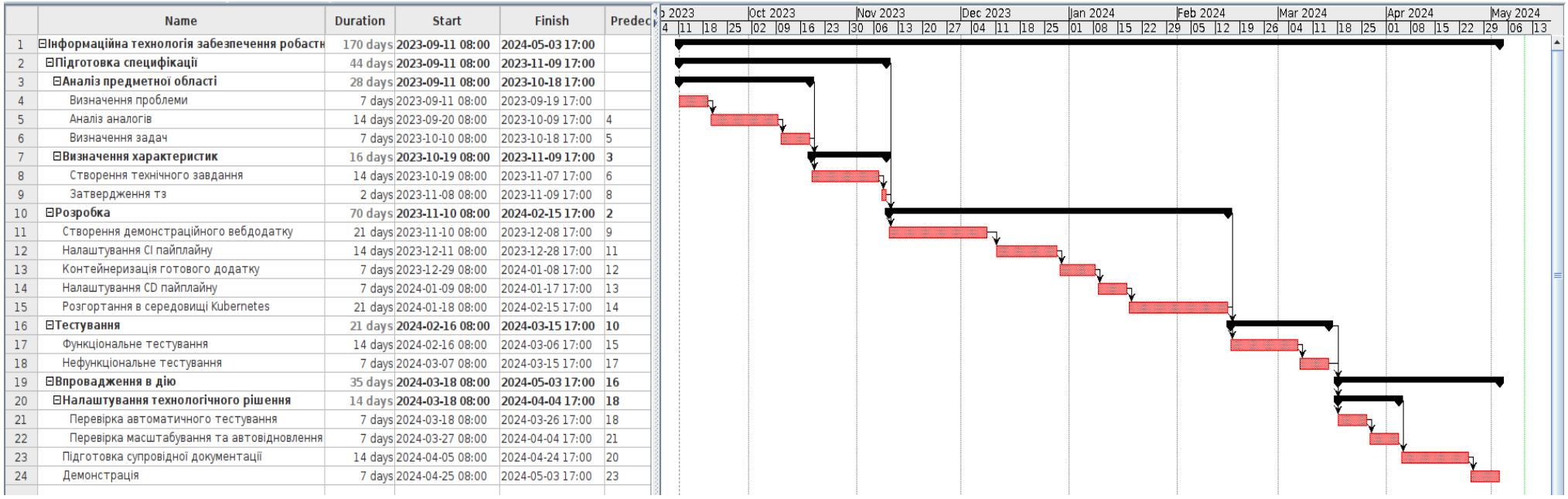


Рисунок А.2 – OBS-структура робіт проекту

Джерело: побудовано автором



	Name	Duration	Start	Finish	Predec
1	**Інформаційна технологія забезпечення робастн**	170 days	**2023-09-11 08:00**	**2024-05-03 17:00**	
2	**Підготовка специфікації**	44 days	**2023-09-11 08:00**	**2023-11-09 17:00**	
3	**Аналіз предметної області**	28 days	**2023-09-11 08:00**	**2023-10-18 17:00**	
4	Визначення проблеми	7 days	2023-09-11 08:00	2023-09-19 17:00	
5	Аналіз аналогів	14 days	2023-09-20 08:00	2023-10-09 17:00	4
6	Визначення задач	7 days	2023-10-10 08:00	2023-10-18 17:00	5
7	**Визначення характеристик**	16 days	**2023-10-19 08:00**	**2023-11-09 17:00**	**3**
8	Створення технічного завдання	14 days	2023-10-19 08:00	2023-11-07 17:00	6
9	Затвердження тз	2 days	2023-11-08 08:00	2023-11-09 17:00	8
10	**Розробка**	70 days	**2023-11-10 08:00**	**2024-02-15 17:00**	**2**
11	Створення демонстраційного вебдодатку	21 days	2023-11-10 08:00	2023-12-08 17:00	9
12	Налаштування CI пайплайну	14 days	2023-12-11 08:00	2023-12-28 17:00	11
13	Контейнеризація готового додатку	7 days	2023-12-29 08:00	2024-01-08 17:00	12
14	Налаштування CD пайплайну	7 days	2024-01-09 08:00	2024-01-17 17:00	13
15	Розгортання в середовищі Kubernetes	21 days	2024-01-18 08:00	2024-02-15 17:00	14
16	**Тестування**	21 days	**2024-02-16 08:00**	**2024-03-15 17:00**	**10**
17	Функціональне тестування	14 days	2024-02-16 08:00	2024-03-06 17:00	15
18	Нефункціональне тестування	7 days	2024-03-07 08:00	2024-03-15 17:00	17
19	**Впровадження в дію**	35 days	**2024-03-18 08:00**	**2024-05-03 17:00**	**16**
20	**Налаштування технологічного рішення**	14 days	**2024-03-18 08:00**	**2024-04-04 17:00**	**18**
21	Перевірка автоматичного тестування	7 days	2024-03-18 08:00	2024-03-26 17:00	18
22	Перевірка масштабування та автовідновлення	7 days	2024-03-27 08:00	2024-04-04 17:00	21
23	Підготовка супровідної документації	14 days	2024-04-05 08:00	2024-04-24 17:00	20
24	Демонстрація	7 days	2024-04-25 08:00	2024-05-03 17:00	23

Рисунок А.3 – Календарний графік проекту

Джерело: побудовано автором

А6. Управління ризиками проекту.

На етапі планування проектних робіт особлива увага приділяється аналізу та управлінню ризиками. Цей процес включає кілька кроків: ідентифікацію ризиків, оцінку їх впливу та ймовірності, розробку стратегій їх зменшення або усунення, моніторинг та контроль за застосуванням цих стратегій. У таблиці А.3 було перелічено ризики даного проекту. Оцінки ризиків надано у таблиці А.4 Таблиця А.5 показує шкалу ризиків за типом, величиною впливу та ймовірністю.

Таблиця А.3 – Ризики проекту.

№ ризику	**Назва (опис) ризику**
1	Проблеми з електроживленням
2	Недостатні апаратні можливості
3	Нестабільний або відсутній інтернет
4	Технічні збої комп’ютерного обладнання
5	Захворювання членів команди розробки
6	Поява альтернативного продукту
7	Недостатній контроль якості
8	Зміна вимог замовника
9	Оновлення залучених бібліотек
10	Поганий тайм-менеджмент

Джерело: побудовано автором

Таблиця А.4 – Результати визначення ймовірності, впливу та рангу ризику проекту.

№ ризику	**Назва (опис) ризику**	**Ймовірність (0,1-0,9)**	**Вплив (0,05-0,8)**	**Ранг**
1	Проблеми з електроживленням	0.1	0.6	0.06
2	Недостатні апаратні можливості	0.1	0.5	0.05
3	Нестабільний або відсутній інтернет	0.3	0.4	0.12
4	Технічні збої комп’ютерного обладнання	0.5	0.2	0.1
5	Захворювання членів команди розробки	0.1	0.5	0.05
6	Поява альтернативного продукту	0.5	0.05	0.025
7	Недостатній контроль якості	0.1	0.6	0.06
8	Зміна вимог замовника	0.1	0.7	0.07
9	Оновлення залучених бібліотек	0.2.	0.4	0.08
10	Поганий тайм-менеджмент	0.2	0.7	0.15

Джерело: побудовано автором

Таблиця А.5 – Шкала оцінювання ризика типом, ймовірністю та величиною впливу.

Оцінка	Ймовірність виникнення	Вплив ризику	Тип ризику
1	Низька	Низький	Прийнятні
2	Середня	Середній	Виправдані
3	Висока	Високий	Недопустимі

Джерело: побудовано автором

Для зменшення негативного впливу потенційних загроз на проект, критично важливо розробити детальний план реагування на ризики. Цей план має включати аналіз та оцінку ефективності запропонованих стратегій зниження ризиків, враховуючи можливі наслідки для проекту. Така оцінка базується на критеріях, визначених у таблиці А.5. В результаті створення плану реагування була сформована матриця, що відображає ймовірність і вплив різних ризиків, що було описано у таблиці А.6. У цій матриці використані кольори для класифікації ризиків: зелений для прийнятних, жовтий для тих, що можна виправдати, та червоний для недопустимих ризиків.

Таблиця А.6 – Матриця ймовірності та впливу

Ймовірність виникнення ризику	**Вплив ризику**					
	0.05	**0.2**	**0.4**	**0.5**	**0.6**	**0.7**
0.5	0.025 **R6**	0.1 **R4**	0.2	0.25	0.3	0.35
0.4	0.02	0.08 **R9**	0.16	0.2	0.24	0.28
0.3	0.015	0.06 **R1, R7**	0.12 **R3**	0.15	0.18	0.21
0.2	0.01	0.04	0.08	0.1	0.12	0.14 **R10**

0.1	0.005	0.02	0.04	0.05 **R2, R5**	0.06	0.07 **R8**

Джерело: побудовано автором

В таблиці А.7 описано класифікацію ризиків за їх рівнем, відповідно до їх індексних значень. Детальний опис ризиків та стратегій їх управління описано у таблиці А.8.

Таблиця А.7 – Шкала оцінювання за рівнем ризику.

№	**Назва**	**Межі**	**Ризики, які входять(номера)**
1	Прийнятні	$0{,}005 \leq R \leq 0{,}05$	2, 5, 6
2	Виправдані	$0{,}05 \leq R \leq 0{,}14$	1, 3, 4, 7, 8, 9, 10
3	Недопустимі	$0{,}14 \leq R \leq 0{,}72$	

Джерело: побудовано автором

Таблиця А.8 – Ризики та стратегії реагування

ID	**Статус ризику**	**Опис ризику**	**Ймовірність виникнення**	**Вплив ризику**	**Ранг ризику**	**План А**	**Тип стратегії реагування**	**План Б**
RS-1	Відкритий	Проблеми з електроживленням	Низька	Високий	0,06	Знайти альтернативне джерело електропостачання (повербанк, зарядна станція…)	Зменшення	Знайти місце, яке має електроживлення (кафе, коворкінг…)
RS-2	Відкритий	Недостатні апаратні можливості	Низька	Високий	0,05	1. Регулярне обслуговування обладнання. 2. Мати запасне обладнання для розподілення задач або позичити робочу станцію	Зменшення	Оновити комплектуючі комп’ютера

Продовження таблиці А.8

RS-3	Відкритий	Нестабільний або відсутній інтернет	Середня	Середній	0.12	Мати декілька джерел інтернету	Прийняття	Знайти найближче публічне джерело інтернету (кафе, коворкінг)
RS-4	Відкритий	Технічні збої комп’ютерного обладнання	Середня	Низький	0.1	Робити резервні копії для потенційного відновлення	Прийняття	Мати запасне обладнання
RS-5	Відкритий	Захворювання членів команди розробки	Низька	Високий	0.05	1. Уникати випадків, які потенційно загрожують здоров’ю 2. Мати розробника, який замінить на деякий час	Зменшення	Віддати частину завдання на аутсорс

Продовження таблиці А.8

RS-6	Відкритий	Поява альтернативного продукту	Висока	Низький	0,025	Уникати розповсюдження важливої інформації. Зберігати код в приватному репозиторії системи керування версіями.	Прийняття	Створити додаткову унікальність або спростити демонстрацію функціоналу для користувачів
RS-7	Відкритий	Недостатній контроль якості	Низька	Високий	0,06	Впровадити періодичні перевірки якості створюваного продукту	Зменшення	Залучити неупереджених експертів для оцінки роботи
RS-8	Новий	Зміна вимог замовника	Низька	Високий	0,07	Регулярне узгодження вимог з замовником	Прийняття	Запровадити гнучку розробку проекту, з можливістю корекції

Продовження таблиці А.8

RS-9	Відкритий	Оновлення залучених бібліотек	Низька	Середній	0,08	Забезпечити модульність проекту. За потреби, змінити проблемну бібліотеку на її аналог	Зменшення	Дослідити можливість змінити мову програмування для використання бажаної бібліотеки
RS-10	Відкритий	Поганий тайм-менеджмент	Низька	Високий	0,15	Створити та затвердити календарний план робіт та чітко його дотримуватися	Прийняття	Знайти розробника, який візьме на себе частину роботи

Джерело: побудовано автором

ДОДАТОК Б

Б1. Лістинг коду файлу робочого процесу для back-end додатку

```
name: Go

on:
  push:
     branches: [ "main" ]
  pull_request:
     branches: [ "main" ]

jobs:

  build:
     runs-on: ubuntu-latest
     steps:
     - uses: actions/checkout@v4

     - name: Set up Go
     uses: actions/setup-go@v4
     with:
     go-version: '1.22.2'

     - name: Set up database
     run: docker run -d --name mysql -p 3306:3306
dmytromolchanov/custom-mysql

     - name: Build back-end
     run: |
```

```
      go build -v ./...


      - name: Test
      run: |
      go test -v ./...


      - name: Create image
      run: |
      docker build . -t ${{ secrets.DOCKERHUB_USERNAME
}}/mysql-fiber-go
      echo "${{ secrets.DOCKERHUB_PASSWORD }}" | docker login -u "${{
secrets.DOCKERHUB_USERNAME }}" --password-stdin
      docker push ${{ secrets.DOCKERHUB_USERNAME }}/mysql-fiber-go
```

Б2. Лістинг коду файлу робочого процесу для front-end додатку

```
name: Node.js CI


on:
  push:
      branches: [ "main" ]
  pull_request:
      branches: [ "main" ]


jobs:
  build:


      runs-on: ubuntu-latest


      strategy:
      matrix:
```

```
      node-version: [22.1.0]

      steps:
      - uses: actions/checkout@v4

      - name: Use Node.js ${{ matrix.node-version }}
      uses: actions/setup-node@v3
      with:
      node-version: ${{ matrix.node-version }}
      cache: 'npm'

      - name: Build
      run: |
      npm ci
      npm run build --if-present

      - name: Create image
      run: |
      docker build . -t ${{ secrets.DOCKERHUB_USERNAME }}/to-do-list-ui
      echo "${{ secrets.DOCKERHUB_PASSWORD }}" | docker login -u "${{
secrets.DOCKERHUB_USERNAME }}" --password-stdin
      docker push ${{ secrets.DOCKERHUB_USERNAME }}/to-do-list-ui
```

Б3. Лістинг коду файлу connect_test.go

```
package commands_test

import (
    "database/sql"
    "mysql-controller/pkg/commands"
    "testing"
```

```
    "github.com/stretchr/testify/assert"
)

func TestConnection(t *testing.T) {
    assert := assert.New(t)

    db, err := commands.DBConnect()
    assert.Nil(err, "Should be nil")

    var mock *sql.DB
    assert.IsType(mock, db)
}
```

Б4. Лістинг коду файлу get_test.go

```
package commands_test

import (
    "encoding/json"
    "io"
    "mysql-controller/pkg/commands"
    "mysql-controller/pkg/types"
    "net/http/httptest"
    "testing"

    "github.com/gofiber/fiber/v2"
    "github.com/stretchr/testify/assert"
)

func TestGet(t *testing.T) {

    assert := assert.New(t)
    app := fiber.New()
```

```
    app.Get("/tasks/:id", commands.Get)

    req := httptest.NewRequest("GET", "/tasks/1", nil)
    resp, err := app.Test(req)

    assert.Nil(err, "Should be nil")
    assert.Equal(200, resp.StatusCode, "Should be successful")

    body, err := io.ReadAll(resp.Body)
    assert.Nil(err)

    var task types.Task
    err = json.Unmarshal(body, &task)

    assert.Nil(err)
    assert.Equal(int64(1), task.ID)
}
```

Б5. Лістинг коду файлу get-all_test.go

```
package commands_test

import (
    "encoding/json"
    "io"
    "mysql-controller/pkg/commands"
    "mysql-controller/pkg/types"
    "net/http/httptest"
    "testing"

    "github.com/gofiber/fiber/v2"
    "github.com/stretchr/testify/assert"
)
```

```
func TestGetAll(t *testing.T) {

    assert := assert.New(t)
    app := fiber.New()

    app.Get("/tasks", commands.GetAll)

    req := httptest.NewRequest("GET", "/tasks", nil)
    resp, err := app.Test(req)

    assert.Nil(err)
    assert.Equal(200, resp.StatusCode)

    body, err := io.ReadAll(resp.Body)
    assert.Nil(err)

    var tasks []types.Task
    err = json.Unmarshal(body, &tasks)
    assert.Nil(err)

    var mock []types.Task
    assert.IsType(mock, tasks)
}
```

Б6. Лістинг Dockerfile для back-end модуля

```
# Використовуємо офіційний образ Go для стадії збірки
FROM golang:1.22.2 as build

# Встановлюємо робочий каталог в контейнері
WORKDIR /app

# Копіюємо go mod та sum файли
```

```
COPY go.mod go.sum ./

# Завантажуємо всі залежності
RUN go mod download

# Копіюємо вихідний код в контейнер
COPY . .

# Збираємо додаток
RUN CGO_ENABLED=0 GOOS=linux go build -a -installsuffix cgo -o
/app/main ./cmd/mysql-controller/main.go

# Використовуємо офіційний образ golang для стадії запуску
FROM alpine:latest

# Встановлюємо робочий каталог в контейнері
WORKDIR /app

# Копіюємо бінарник з стадії збірки
COPY --from=build /app/main .

# Відкриваємо порт 3001
EXPOSE 3001

# Команда для запуску додатку
CMD ["/app/main"]
```

Б7. Лістинг Dockerfile для front-end модуля

```
# Використовуємо офіційний образ Node.js для стадії збірки
FROM node:22.1-alpine as build

# Встановлюємо робочий каталог в контейнері
WORKDIR /app
```

```
# Копіюємо package.json та package-lock.json
COPY package*.json ./

# Встановлюємо залежності
RUN npm install

# Копіюємо вихідний код в контейнер
COPY . .

RUN chmod u+x ./start.sh

# Відкриваємо порт 3000
EXPOSE 3000

CMD ["./start.sh"]
```

Б8. Лістинг Dockerfile для бази даних

```
# Використовуємо офіційний образ MySQL
FROM mysql:latest

# Копіюємо sql-скрипт в контейнер
COPY ./tasks.sql /docker-entrypoint-initdb.d/
```

Б9. Лістинг Terraform коду main.tf

```
module mysql {
  source = "./modules/mysql"
}

module fiber {
  source = "./modules/fiber"
  depends_on = [module.mysql]
```

```
}

module react {
  source = "./modules/react"
  depends_on = [module.fiber]
}
```

Б9. Лістинг Terraform коду providers.tf

```
terraform {
  required_providers {
     kubernetes = {
     source = "hashicorp/kubernetes"
     version = "2.30.0"
     }
  }
}

provider "kubernetes" {
  config_path  = "~/.kube/config"
}
```

Б9. Лістинг Terraform коду fiber/variable.tf

```
variable namespace {
  type = string
  default = "fiber"
}

variable label {
  type         = string
  default      = "fiber"
}
```

Б9. Лістинг Terraform коду fiber/namespace.tf

```
resource "kubernetes_namespace" "fiber" {
  metadata {
     name = var.namespace
  }
}
```

Б9. Лістинг Terraform коду fiber/deployment.tf

```
resource "kubernetes_deployment" "fiber" {
  metadata {
     name      = "to-do-list-back"
     namespace = var.namespace
  }
  spec {
     replicas = 2
     selector {
     match_labels = {
     app = var.label
     }
     }

     template {
     metadata {
     labels = {
          app = var.label
     }
     }
     spec {
     container {
          image = "dmytromolchanov/mysql-fiber-go:latest"
          name  = "back-end"
          port {
```

```
            container_port = 3001
            }

            env {
            name  = "UI_HOST"
            value = "localhost"
            }
            env {
            name  = "UI_PORT"
            value = "3000"
            }
            env {
            name  = "DB_HOST"
            value = "mysql-service.mysql.svc.cluster.local"
            }
            env {
            name  = "DB_PORT"
            value = "3306"
            }
      }
      }
      }
  }
}
```

Б9. Лістинг Terraform коду fiber/service.tf

```
resource "kubernetes_service" "fiber" {
  depends_on = [
     kubernetes_namespace.fiber
  ]

  metadata {
     name      = "fiber-service"
```

```
    namespace = var.namespace
  }
  spec {
    selector = {
    app = var.label
    }
    type = "ClusterIP"
    port {
    port      = 3001
    target_port = 3001
    }
  }
}
```