Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

Пояснювальна записка

до бакалаврської дипломної роботи на тему

на тему: «Обчислювальна інфраструктура розгортання вебресурсу на хмарній платформі AWS»

Виконав: студент 2 курсу, групи 1КІ-22мс
спеціальності 123 — «Комп'ютерна інженерія»
освітня програма — «Комп'ютерна інженерія»

Риженко А. О

Керівник к.т.н., доц. каф. ОТ

Войцеховська О. А

Рецензент к.т.н., доц. каф. ПЗ

Майданюк В. П.

Допущено до захисту
Завідувач кафедри ОТ
д.т.н., проф. Азаров О. Д.

"10" 06 2024 р.

Вінниця ВНТУ — 2024 рік

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

Освітньо—кваліфікаційний рівень бакалвр

Галузь знань — 12 Інформаційні технології

Спеціальність — 123 Комп'ютерна інженерія

Освітня програма — Комп'ютерна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ
проф., д.т.н. _____ Азаров О. Д.
«6» лютого 2024 року

З А В Д А Н Н Я

НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Риженку Артему Олександровичу

1 Тема роботи: Обчислювальна інфраструктура розгортання вебресурсу на хмарній платформі AWS, керівник роботи Войцеховська Олена Валеріївна, к.т.н., доцент кафедри ОТ, затверджено наказом вищого навчального закладу від «11» березня 2024 року №80.

2 Строк подання студентом роботи 10 червня 2024 року.

3 Вихідні дані до роботи: сучасні технології розгортання вебресурів в хмарному середовищі, контент для наповнення вебдодатку, технології розробки клієнт-серверних вебдодатків, база даних IMDB, API TMDB.

4 Зміст розрахунково-пояснювальної записки: вступ, аналіз сучасних хмарних технологій для розгортання та підтримки вебресурсів, розробка обчислювальної інфраструктури розгортання вебресурсу на AWS, практична реалізація вебдодатку та його розгортання в AWS.

5 Перелік графічного матеріалу: Структурна схема архітектури та процесу розгортання вебдодатку. Етапи розгортання та безпеки на AWS за допомогою

CircleCI. Блок-схема алгоритму розгортання вебзастосунку. Структурна схема сервісів та їх залежностей у docker compose. ER-діаграма бази даних. Етапи роботи вебсерверу Nginx.

6 Консультанти розділів роботи приведені в таблиці 1.

Таблиця 1 — Консультанти роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-3	к.т.н., доц. каф. ОТ Войцеховська О. В.		

7 Дата видачі завдання 12.03.2024 р.

8 Календарний план виконання дипломної роботи приведений в таблиці 2.

Таблиця 2 — Календарний план

№ з/п	Назва етапів дипломної роботи	Строк виконання	Примітка
1	Постановка задачі роботи	12.03.24	Виконано
2	Аналіз середовища розгортання вебресурсу	13.03.24 – 17.03.24	Виконано
3	Обгрунтування вибору інфраструктурних рішень та розробка обчислювальної інфраструктури розгортання вебресурсу	18.03.24 – 29.03.24	Виконано
4	Конфігурація безперервної інтеграції та безперервного постачання (CI/CD)	30.03.24 – 09.04.24	Виконано
5	Розробка структури програмного застосунку	10.04.24 – 16.04.24	Виконано
6	Розробка та наповнення бази даних, а також API для відображення внесених даних	17.04.24 – 25.04.24	Виконано
7	Створення інтерфейсу користувача для взаємодії із даними через Vue.js	26.04.24 – 07.05.24	Виконано
8	Оформлення пояснювальної записки та ілюстративного матеріалу	08.05.24 – 19.05.24	Виконано
9	Перевірка якості виконання бакалаврської роботи та усунення недоліків	20.05.24 – 08.06.24	Виконано

Студент Артем РИЖЕНКО

Керівник роботи к.т.н., доц. Олена ВОЙЦЕХОВСЬКА

АНОТАЦІЯ

УДК 004.7

Риженко А. О. Обчислювальна інфраструктура розгортання вебресурсу на хмарній платформі AWS. Пояснювальна записка містить 97 аркушів, 24 рисунків, 15 лістингів та 40 посилань.

В бакалаврській дипломній роботі створено інфрастрктуру для розгортання вебресурсу на хмарній платформі AWS. Розглянуто технічні аспекти розгортання вебдодатків на AWS, проаналізовано оптимальний вибір інфраструктурних рішень та інструментів для ефективного впровадження та масштабування вебресурсу. У роботі використано такі технології, як JavaScript із фреймворком Vue.js та Python із фреймворком Django, а також Docker для контейнеризації вебдодатку. Висвітлено ключові аспекти процесу розгортання, включаючи конфігурування серверу, забезпечення безпеки та моніторингу системи.

Ключові слова: обчислювальна інфраструктура, розгортання вебдодатку, Amazon Web Services, EC2, RDS, Django Rest Framework, Docker, PostgreSQL, CircleCI, CI/CD, Nginx.

ABSTRACT

UDC 004.7

Ryzhenko A.O. Computing infrastructure for deploying a web resource on the AWS cloud platform. The explanatory note contains 97 sheets, 24 figures, 15 listings and 40 references.

In this bachelor thesis created an infrastructure for deploying a web resource on the AWS cloud platform. The technical aspects of deploying web applications on AWS are considered, in particular, the optimal choice of infrastructure solutions and tools for the effective implementation and scaling of a web resource is analyzed. The work used technologies such as JavaScript with the Vue.js framework and Python with the Django framework, as well as Docker for containerization of the web application. The key aspects of the deployment process, including server configuration, security, and system monitoring, are covered.

Keywords: computing infrastructure deployment of a web resource, Amazon Web Services, EC2, RDS, Django Rest Framework, Docker, PostgreSQL, CircleCI, CI/CD, Nginx.

ЗМІСТ

ВСТУП

У сучасному цифровому світі розвиток вебдодатків відіграє ключову роль у створенні та поширенні інформації та послуг через Інтернет. Однак, зростання вимог до продуктивності, масштабованості та безпеки цих додатків ставить перед розробниками та інженерами завдання забезпечити ефективне розгортання та управління вебресурсами. Зі збільшенням обсягів даних та зростанням навантаження на сервери виникає необхідність використання новітніх технологій та інфраструктурних рішень для забезпечення стабільної та надійної роботи вебдодатків.

Швидкі темпи розвитку цифрового середовища та зростаючі вимоги до продуктивності, масштабованості та безпеки вебресурсів ставлять перед розробниками та IT-інженерами низку складних завдань. У зв'язку з цим, хмарні сервіси та послуги набувають все більшої популярності, забезпечуючи розробникам широкий спектр інструментів та сервісів для розгортання, масштабування та керування вебресурсами. Хмарні платформи забезпечують розробникам та IT-інженерам широкий спектр інструментів та сервісів для розгортання, масштабування та керування вебресурсами, відповідаючи на сучасні вимоги продуктивності, масштабованості та безпеки. Серед найвідоміших і найпопулярніших хмарних платформ виділяються Amazon Web Services, Microsoft Azure та Google Cloud Platform. Лідером серед них є Amazon Web Services, що пропонує найбільш повний та надійний набір хмарних рішень.

Зростання популярності хмарних технологій сприяє, в свою чергу, підвищенню попиту на кваліфікованих фахівців, які мають досвід розгортання та управління вебресурсами в хмарному середовищі. Такі фахівці можуть ефективно працювати над розробкою, розгортанням та підтримкою вебдодатків у реальних умовах. Отже, проектування інфраструктури розгортання вебресурсу в хмарному середовищі є **актуальною темою** у сучасній інформаційній сфері.

Метою даної роботи є покращення та автоматизація процесу розгортання вебресурсу в хмарному середовищі AWS шляхом реалізації хмарної

обчислювальної інфраструктури із впровадженням безперервного розгортання та інтеграції.

Для досягнення поставленої мети необхідно вирішити такі **завдання**:

— проаналізувати сучасні хмарні технології для розгортання та підтримки вебресурсу;

— розробити інфраструктурне рішення для реалізації обчислювальної інфраструктури;

— розробити структурну схему обчислювальної інфраструктури розгортання вебресурсу;

— налаштувати безпечний доступ до розроблюваної обчислювальної інфраструктури;

— створити та підключити базу даних до обчислювальної інфраструктури;

— сконфігурувати безперервну інтеграцію та доставку вебресурсу на обчислювальну інфраструктуру;

— створити, сконфігурувати та наповнити вебдодаток для його розгортання у обчислювальній інфраструктурі;

— створити та підключити SSL сертифікат для захисту особистої інформації користувачів.

Об'єкт дослідження: процес безперервної інтеграції та безперервного розгортання вебресурсу в хмарному середовищі на платформі AWS.

Предмет дослідження: програмні засоби інтеграції та підтримки вебресурсів в хмарному середовищі та сучасні технології і засоби для створення вебдодатків.

Апробація результатів бакалаврської роботи: опубліковано доповідь на Міжнародній науково-практичній інтернет-конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2024)»

1 АНАЛІЗ СУЧАСНИХ ХМАРНИХ ТЕХНОЛОГІЙ ДЛЯ РОЗГОРТАННЯ ТА ПІДТРИМКИ ВЕБРЕСУРСІВ

1.1 Аналіз сучасного стану ринку хмарних технологій

Хмарні технології стали неодмінною складовою сучасної інформаційно-технологічної індустрії, забезпечуючи компаніям широкий спектр можливостей для зберігання даних, обчислень та розгортання вебдодатків. Розвиток хмарних платформ відбувався від початкових послуг з віртуалізації до наборів інтегрованих інструментів та сервісів для будь-яких потреб.

Однією з ключових характеристик хмарних платформ є можливість легко масштабувати обчислювальні ресурси залежно від потреб користувача. Це дозволяє компаніям оперативно збільшувати або зменшувати витрати на обладнання та інфраструктуру, реагуючи на зміну обсягів користувацького трафіку або потреби в обробці даних. Завдяки цьому компанії можуть уникати зайвих витрат на підтримку надлишкових потужностей у періоди низької активності та забезпечувати необхідний рівень ресурсів у періоди пікових навантажень.

Гарантія високої доступності та надійності сервісів досягається завдяки географічно розподіленим центрам даних і автоматичному резервуванню даних, це означає, що навіть у випадку виходу з ладу одного з центрів даних, сервіси залишаються доступними, оскільки дані автоматично переміщуються до іншого центру. Така архітектура забезпечує безперервність та мінімізує ризик втрати даних.

Модель оплати за використання, яку пропонують хмарні платформи, дозволяє користувачам платити лише за реально використані ресурси, це робить витрати на хмарні сервіси прогнозованими та ефективними, оскільки компанії можуть точно розраховувати свої витрати залежно від обсягу споживаних ресурсів. Така модель особливо вигідна для стартапів та малих бізнесів, які можуть уникати великих початкових інвестицій у власну інфраструктуру.

Різноманіття сервісів, що пропонують хмарні платформи включає зберігання даних, віртуальні машини, сервіси штучного інтелекту, аналіз даних та інші. Це дозволяє компаніям використовувати сучасні технології без необхідності

розробляти та підтримувати їх самостійно. Внаслідок цього компанії можуть зосередитися на своїх основних завданнях, залишаючи технічні аспекти надійним провайдерам хмарних послуг.

Широкий спектр інструментів та сервісів для захисту даних, які пропонують хмарні платформи, включає шифрування, контроль доступу та моніторинг безпеки, це дозволяє забезпечити високий рівень захисту конфіденційної інформації та відповідати вимогам регуляторних органів. Також компанії можуть використовувати інструменти для моніторингу безпеки та виявлення потенційних загроз у режимі реального часу, що підвищує загальний рівень захищеності даних.

Розвиток хмарних технологій продовжується, і хмарні платформи продовжують розширювати свої можливості, щоб задовольнити різноманітні потреби користувачів у цифровому світі.

Хмарні технології виникли у відповідь на збільшення обсягів даних та потреби в їх обробці, а також на зростання вимог до доступності та надійності сервісів. Їхнє зростання відбувалося паралельно з розвитком інтернет-технологій та вирішенням проблем, пов'язаних зі складністю та вартістю утримання власних обчислювальних ресурсів у компаній. Основні характеристики хмарних платформ, такі як гнучкість, масштабованість та оплата за використання, роблять їх привабливими для різних секторів індустрії. Бізнес-процеси стають більш ефективними завдяки зниженню витрат на обладнання та інфраструктуру, а також завдяки швидкому розгортанню та масштабуванню додатків.

Крім того, на масштаби використання хмарних технологій як в нашій країні, так і в глобальному масштабі значно вплинула російська збройна агресія проти України. Конфлікт призвів до численних змін в IT-індустрії, включаючи прискорення переходу до хмарних сервісів для забезпечення безперервності бізнесу та захисту даних.

Під час збройної агресії, багато українських компаній зіткнулися з необхідністю швидко перемістити свої дані та обчислювальні ресурси у безпечні регіони. Хмарні платформи стали важливим інструментом для досягнення цієї мети, дозволяючи організаціям швидко масштабувати свої інфраструктури та

забезпечувати доступ до необхідних сервісів навіть у кризових умовах. Такі можливості хмарних платформ, як гнучкість та висока доступність, стали критичними для збереження операційної діяльності підприємств.

Крім того, збройна агресія викликала підвищену увагу до безпеки даних. В умовах зростання кіберзагроз, компанії були змушені посилювати свої системи захисту, використовуючи сучасні інструменти шифрування, контроль доступу та моніторинг безпеки, які пропонують хмарні платформи. Це дозволило мінімізувати ризики втрати чи компрометації конфіденційної інформації.

Також варто зазначити, що міжнародні хмарні провайдери, реагуючи на кризу, почали пропонувати спеціальні умови для українських компаній та урядових установ, це безплатне або знижене використання ресурсів, що допомогло багатьом організаціям підтримувати свою діяльність у важкі часи [2].

Окрім внутрішніх змін, збройна агресія в Україні вплинула і на глобальне розуміння важливості хмарних технологій. Багато міжнародних компаній переглянули свої стратегії щодо резервування даних та планів на випадок надзвичайних ситуацій, включаючи можливість швидкого переходу до хмарних рішень, це призвело до зростання інвестицій у хмарну інфраструктуру та розвиток нових сервісів, орієнтованих на підвищення стійкості до кризових ситуацій [3].

Таким чином, збройна агресія в Україні підкреслила важливість хмарних технологій для забезпечення гнучкості, безпеки та стійкості бізнесу у надзвичайних умовах. Впровадження та розвиток хмарних платформ продовжують залишатися ключовими аспектами для організацій у всьому світі, сприяючи їхньому адаптації до нових викликів та забезпеченню стабільного функціонування.

1.2 Аналіз особливостей розгортання вебдодатків з використанням хмарних технологій

Тенденції в розгортанні вебдодатків постійно змінюються під впливом технологічного прогресу та вимог користувачів. Останнім часом спостерігається стрімке зростання популярності хмарних рішень для розгортання вебдодатків. Хмарні платформи, такі як Amazon Web Services (AWS), Google Cloud Platform

(GCP) та Microsoft Azure, надають широкий спектр послуг, що дозволяє розробникам ефективно керувати та масштабувати свої додатки. Особливістю хмарних рішень є гнучкість та масштабованість: вони дозволяють розгортати вебдодатки в будь-якому масштабі, від невеликих стартапів до великих корпоративних проектів.

Однією з ключових тенденцій в розгортанні вебдодатків є використання контейнеризації. Контейнери, наприклад Docker, забезпечують ізоляцію та переносимість додатків, дозволяючи розробникам створювати консистентні середовища розгортання, незалежно від операційної системи та інфраструктури. Іншою важливою тенденцією є використання інструментів інфраструктури як коду (Infrastructure as Code, IaC), таких як Terraform або AWS CloudFormation. Ці інструменти дозволяють автоматизувати процес розгортання та конфігурування інфраструктури, зменшуючи час та ризики людських помилок [4].

Крім того, з'являється все більше інструментів для керування контейнеризованими додатками. Kubernetes надає засоби для оркестрації та автоматизації розгортання, масштабування та керування контейнеризованими додатками у хмарних середовищах. Ці технології дозволяють розробникам створювати високопродуктивні та стійкі вебдодатки, які можуть легко масштабуватися для відповіді на зростаючі потреби користувачів. Таким чином, розгортання вебдодатків на хмарних платформах із використанням сучасних підходів стає дедалі більш привабливим варіантом для розробників та підприємств, які прагнуть досягти успіху в цифровому світі [5].

У процесі розгортання вебресурсів на хмарних платформах виникає низка питань, на які необхідно звернути увагу ще до початку цього процесу, де одне з найголовніших — це безпека даних. Оскільки вебдодатки зберігають та обробляють важливу користувацьку інформацію, таку як особисті дані або фінансова інформація, важливо забезпечити, щоб ці дані були захищені від несанкціонованого доступу та зловживань. Це включає в себе відповідність регуляторним вимогам, які можуть різнитися в різних країнах та регіонах, а також застосування кращих практик забезпечення безпеки даних.

Ще одним важливим моментом є масштабованість і продуктивність. Підвищення обсягу користувацького трафіку або обробка великих обсягів даних може призвести до перевантаження інфраструктури. Забезпечення ефективного масштабування вебдодатків, яке дозволяє вирішувати зростаючі потреби користувачів, вимагає ретельного проектування архітектури, використання відповідних технологій масштабування та моніторингу, а також постійного аналізу та оптимізації ресурсів.

Організаційні аспекти також відіграють важливу роль. Розгортання вебдодатків на хмарних платформах може потребувати зміни в організаційній культурі, процесах розробки та управління, а також навичках персоналу. Важливо забезпечити відповідну підтримку, навчання та розвиток команди, а також впровадження ефективних процесів управління проектами та співпраці між командами розробників, DevOps та адміністраторів.

Отже, розгортання вебдодатків на хмарних платформах включає в себе багато складних викликів, які вимагають комплексного підходу та постійного удосконалення.

1.3 Порівняльний аналіз сучасних хмарних платформ

Порівняльний аналіз хмарних платформ є важливим етапом при виборі інфраструктури для розгортання вебдодатків та послуг. Основні хмарні платформи, такі як Amazon Web Services (AWS), Google Cloud Platform (GCP) та Microsoft Azure, пропонують широкий спектр сервісів (рисунок 1.1) та можливостей, але мають свої унікальні особливості, які варто враховувати при прийнятті рішення.

AWS є лідером на ринку хмарних платформ та відомий широким набором сервісів, включаючи обчислювальні ресурси, зберігання даних, бази даних, інструменти штучного інтелекту та машинного навчання, а також інструменти для розгортання та управління інфраструктурою [6].

Google Cloud Platform відомий своєю швидкістю та ефективністю, особливо у сфері аналізу даних та штучного інтелекту. GCP також має широкий спектр

інструментів для розробки та розгортання вебдодатків, а також великий вибір сервісів для управління даними та обробки даних [7].

Microsoft Azure відомий своєю інтеграцією з існуючими продуктами Microsoft та широкими можливостями для підтримки корпоративних вебдодатків та сервісів. Azure пропонує широкий спектр інструментів для розробки, тестування, розгортання та моніторингу вебдодатків, а також різноманітні послуги для обробки даних та аналітики [8].

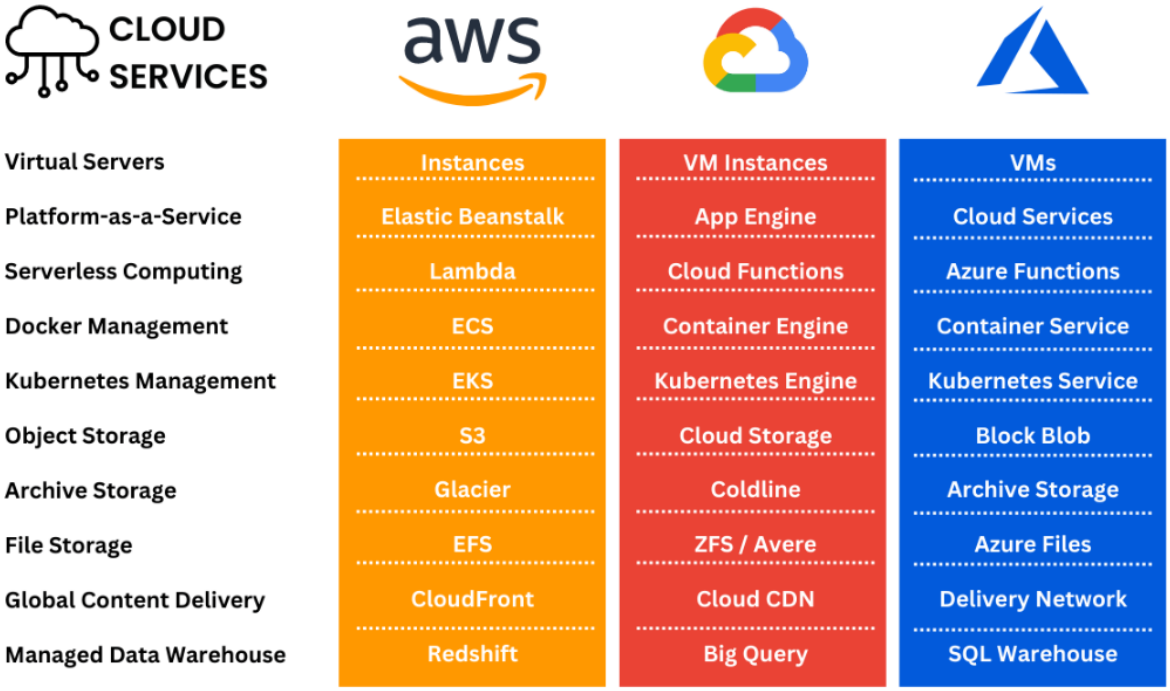

Рисунок 1.1 — Перелік сервісів, які надають хмарні платформи

Порівняльні характеристики провідних хмарних платформ наведено в таблиці 1.1.

Крім найбільш відомих, існують і інші хмарні платформи, такі як IBM Cloud, Oracle Cloud Infrastructure та Alibaba Cloud, які також пропонують свої унікальні сервіси та можливості. Наприклад, IBM Cloud спеціалізується на рішеннях для підприємств, таких як блокчейн та інтернет речей (IoT), тоді як Oracle Cloud Infrastructure надає високопродуктивні обчислювальні та зберігальні ресурси для корпоративних клієнтів.

Таблиця 1.1 — Основні відмінності трьох провідних хмарних платформ

Характеристика	AWS	Azure	GCP
Дата запуску	2006	2010	2008
Поширеність	Найбільша кількість сервісів та регіонів	Сильна інтеграція з продуктами Microsoft	Інноваційні рішення в галузі обробки даних та машинного навчання
Продукти та сервіси	Найширший спектр сервісів та рішень	Глибока інтеграція з Windows Server та інструментами Microsoft	Сильні інструменти для аналітики та Big Data
Розрахункові моделі	Гнучкі опції оплати та розрахунків	Хороші варіанти для підприємств, що використовують Microsoft	Зручні для стартапів та бізнесів, що працюють з великими даними
Безпека	Широкий спектр інструментів безпеки та відповідності	Переваги для користувачів Microsoft Active Directory	Міцні заходи безпеки, акцент на шифрування даних
Регіональна доступність	Найбільше географічне покриття	Широка мережа датацентрів, особливо в Європі та США	Добре покриття в ключових регіонах, швидко розширюється
Цільова аудиторія	Підприємства всіх розмірів, від стартапів до великих корпорацій	Корпорації, які вже використовують продукти Microsoft	Стартапи, аналітичні компанії, організації, які фокусуються на обробці даних

При порівнянні хмарних платформ слід враховувати такі фактори, як доступність сервісів у конкретному регіоні, цінова політика, рівень безпеки та дотримання різних стандартів та вимог щодо захисту даних. Також важливо врахувати функціональні можливості, такі як інтеграція з іншими сервісами та інструментами, масштабованість, продуктивність та підтримка користувачів.

Окрім того, важливим фактором є екосистема партнерів та додаткових сервісів, які підтримуються кожною хмарною платформою. Наявність широкого вибору партнерів та інтегрованих сервісів може сприяти швидкому розвитку та впровадженню проектів, а також забезпечити доступ до спеціалізованих послуг та експертної підтримки.

На 2023 рік, частка світового ринку хмарних платформ розподіляється на Amazon Web Services із 32%, Microsoft Azure із 23% та Google Cloud Platform із 10% та інші провайдери із 35% [9].

З рисунку 1.2 видно що домінуюче положення на ринку займає AWS, однак конкуренція з боку Azure та GCP залишається сильною. Важливо зазначити, що кожна з платформ має свої переваги та сильні сторони, які можуть бути вирішальними при виборі провайдера для конкретних потреб.

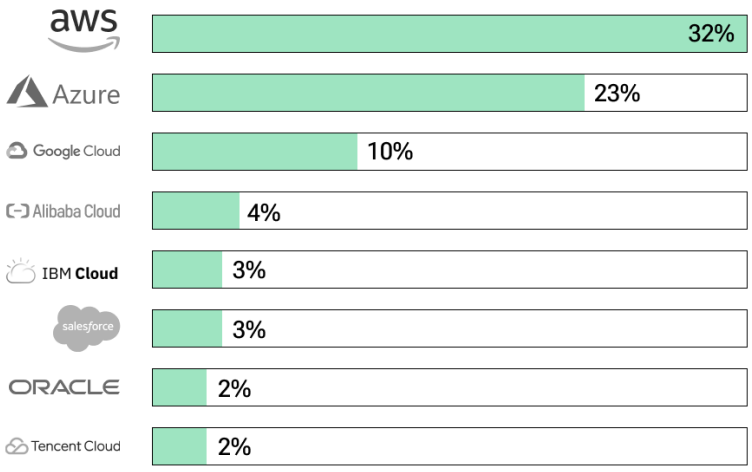

Рисунок 1.2 — Найбільші провайдери на глобальному ринку хмарних технологій

В Україні існують кілька хмарних провайдерів, які пропонують різні рішення для бізнесу, із них GigaCloud, що пропонує IaaS та PaaS рішення, зосереджуючись на малих та середніх підприємствах, а також відома своєю надійністю та гнучкими умовами для масштабування. DataGroup, яка спеціалізується на забезпеченні хмарної інфраструктури для великих підприємств та державних установ і пропонує високий рівень безпеки та підтримки клієнтів. De Novo пропонує хмарні рішення з високим рівнем безпеки, що підходять для фінансових та урядових організацій, відома своєю експертизою у сфері обробки даних та забезпечення відповідності

регуляторним вимогам. UA-Cloud зосереджується на забезпеченні надійної хмарної інфраструктури для малого та середнього бізнесу і пропонує конкурентоспроможні ціни та індивідуальний підхід до кожного клієнта.

AWS, як один з провідних світових хмарних провайдерів, є англомовною структурою з англомовною підтримкою. Це може бути викликом для деяких українських підприємств, які віддають перевагу локалізованому обслуговуванню. На відміну від AWS, українські хмарні провайдери забезпечують підтримку рідною мовою, що може спростити комунікацію та вирішення технічних питань.

При виборі хмарного провайдера важливо враховувати не лише світових гігантів, але й локальних провайдерів, які можуть запропонувати конкурентоспроможні умови та індивідуальний підхід до відповідних потреб. Локальні провайдери часто мають кращі умови для місцевих підприємств завдяки розумінню ринку та швидшому реагуванню на запити клієнтів.

1.4 Поняття CI/CD та принцип його роботи

CI/CD (Continuous Integration/Continuous Delivery або Continuous Deployment) — це методологія розробки програмного забезпечення, яка акцентує увагу на частих і надійних випусках змін у коді. Вона включає в себе автоматизацію процесів інтеграції (рисунок 1.3), тестування та доставки (або розгортання) програмного забезпечення. Метою CI/CD є підвищення ефективності розробки, покращення якості продукту та зменшення часу, необхідного для доставки нових функцій користувачам [10].

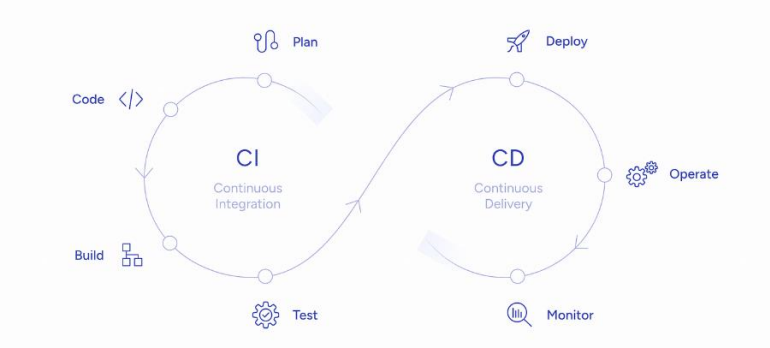

Рисунок 1.3 — Схема життєвого циклу CI/CD

Continuous Integration (CI) — це практика регулярної інтеграції змін у коді від усіх членів команди в основну гілку репозиторію. Вона забезпечує безперервне вдосконалення та стабільність проекту завдяки автоматизації та регулярній перевірці коду.

Кожне внесення змін до коду запускає автоматизовані тести для перевірки коректності змін, це дозволяє одразу виявити помилки та забезпечити, що нові зміни не порушують функціональність існуючого коду. Автоматизація тестування значно скорочує час на ручну перевірку та підвищує якість програмного забезпечення.

Код збирається автоматично, що дозволяє виявити і виправити проблеми на ранніх етапах. Автоматичне збирання забезпечує своєчасне виявлення конфліктів і помилок, які можуть виникати під час інтеграції змін, що спрощує процес розробки та робить його більш ефективним.

Регулярна інтеграція допомагає зменшити кількість конфліктів при злитті гілок. Часті внесення змін та інтеграція сприяють кращій синхронізації роботи команди, зменшуючи ризик виникнення великих конфліктів при об'єднанні різних гілок коду. Воно забезпечує більш гладкий та керований процес розробки.

Continuous Integration є ключовим компонентом сучасних методологій розробки програмного забезпечення, таких як DevOps та Agile. Вона сприяє більш ефективному управлінню проектом, підвищує продуктивність команди та забезпечує високу якість кінцевого продукту.

Continuous Delivery (CD) розширює практику Continuous Integration (CI), включаючи автоматизацію доставки змін до середовища, наближеного до реального.

Після успішного тестування зміни автоматично розгортаються в середовищах тестування або стадійного розгортання, це забезпечує постійну перевірку та інтеграцію змін у реалістичних умовах, що допомагає виявляти можливі проблеми на ранніх етапах. Такий підхід мінімізує ризики та підвищує якість кінцевого продукту.

Продукт завжди готовий до випуску в реальне середовище. Завдяки постійній автоматизації та тестуванню, зміни можуть бути швидко і безпечно випущені в експлуатацію, що дозволяє швидко реагувати на потреби бізнесу та користувачів, забезпечуючи більш швидке впровадження нових функцій та виправлення помилок.

Continuous Delivery є наступним кроком після Continuous Integration, забезпечуючи безперервний потік змін від розробників до кінцевих користувачів. Вона сприяє гнучкості та адаптивності бізнесу, дозволяючи швидко реагувати на зміни на ринку та потреби клієнтів.

Continuous Deployment (CD) — це подальше розширення практики Continuous Delivery, де кожна зміна, що пройшла всі етапи автоматизованого тестування, автоматично розгортається в продуктивне середовище.

Зміни автоматично доставляються користувачам без ручного втручання. Воно забезпечує безперервний потік оновлень та нових функцій, що виходять у продуктивне середовище одразу після проходження всіх необхідних перевірок. Такий підхід мінімізує час від розробки до впровадження та підвищує швидкість реакції на потреби ринку.

Часті і невеликі зміни дозволяють швидко виявляти і виправляти помилки. Завдяки малим поступовим оновленням легше визначити джерело проблеми, що виникла після впровадження змін. Завдяки цьому проходить зменшення час на діагностику та виправлення помилок, що забезпечує стабільність і надійність продукту.

Continuous Deployment є найвищим рівнем автоматизації процесу розгортання, що дозволяє компаніям підтримувати постійний ритм оновлень та швидко реагувати на зворотний зв'язок від користувачів. Ця практика сприяє більш тісній інтеграції розробки та експлуатації, що є основою методології DevOps, забезпечуючи високу продуктивність і якість програмного забезпечення.

Однією з головних переваг CI/CD є автоматичне тестування і постійна інтеграція, що допомагають виявити помилки на ранніх етапах розробки. Завдяки цьому, розробники можуть швидко реагувати на проблеми, вносячи необхідні

виправлення до того, як ці помилки вплинуть на кінцевих користувачів або призведуть до серйозних збоїв у системі.

Автоматизація процесів зменшує час на ручну роботу і дозволяє швидше доставляти нові функції. Це означає, що розробники можуть зосередитися на написанні коду та вдосконаленні продукту, а не витрачати час на рутинні завдання, такі як компіляція, тестування та розгортання. Автоматизовані процеси також знижують ймовірність людських помилок, що підвищує загальну якість програмного забезпечення.

Часті малі зміни простіше тестувати і розгортати, що знижує ризик серйозних проблем у продуктивному середовищі. Завдяки цьому підходу, кожна зміна може бути ретельно перевірена в ізольованому середовищі, перш ніж вона буде інтегрована в основну гілку коду, це дозволяє забезпечити стабільність та надійність системи, навіть під час постійних оновлень.

CI/CD забезпечує можливість швидко і надійно випускати нові версії продуктуе особливо важливо в сучасному конкурентному середовищі, де швидкість виходу на ринок є ключовим фактором успіху. Постійне оновлення продукту дозволяє компаніям оперативно реагувати на зворотний зв'язок від користувачів, впроваджувати нові функції та виправляти помилки, забезпечуючи при цьому високу якість та стабільність програмного забезпечення.

Існує багато інструментів для впровадження CI/CD, кожен з яких має свої особливості та переваги, наприклад CircleCI, котрий є простим у налаштуванні інструментом для CI/CD, який підтримує паралельне виконання завдань. CircleCI інтегрується з популярними системами контролю версій та хмарними платформами, що робить його зручним для різних типів проектів.

Jenkins є потужним і гнучким інструментом з відкритим кодом для CI/CD, що має велику кількість плагінів. Завдяки своїй розширюваності, Jenkins може бути налаштований для будь-яких потреб проекту. Він підтримує різні мови програмування та платформи, що робить його універсальним рішенням для великих і складних проектів.

GitLab CI інтегрований з GitLab і підтримує Docker та автоматизацію пайплайнів. Це дозволяє легко налаштовувати процеси CI/CD без необхідності використовувати додаткові сервіси. GitLab CI забезпечує безшовну інтеграцію з репозиторіями коду, що спрощує управління версіями та автоматизацію розгортання.

Travis CI є легким у налаштуванні інструментом, особливо популярним для відкритих проектів. Він пропонує простий у використанні інтерфейс та інтеграцію з GitHub, що робить його зручним для невеликих команд та проектів з відкритим вихідним кодом. Travis CI підтримує автоматичне тестування та розгортання, що допомагає забезпечити стабільність коду.

GitHub Actions є інструментом для автоматизації робочих процесів, інтегрованим в GitHub, що дозволяє створювати складні пайплайни безпосередньо в репозиторії коду, використовуючи YAML-файли. GitHub Actions підтримує широкий спектр подій, таких як коміти, пулл-запити та релізи, що робить його потужним інструментом для автоматизації різних етапів розробки.

Bamboo — інструмент від Atlassian, який забезпечує інтегрований CI/CD для зручного впровадження процесів розробки. Bamboo підтримує різні системи контролю версій, включаючи Git та Mercurial, і може інтегруватися з іншими інструментами Atlassian, такими як JIRA та Bitbucket. Це робить його ідеальним рішенням для команд, які використовують екосистему Atlassian.

У даному проекті обрано CircleCI, так як він пропонує простий та інтуїтивно зрозумілий процес налаштування, що дозволяє швидко інтегрувати його в будь-який проект. Завдяки підтримці Docker-контейнерів, використовувлись спеціалізовані середовища для різних етапів збірки та тестування, що підвищує загальну ефективність та масштабованість процесів.

Однією з ключових переваг є підтримка паралельного виконання завдань. Це дозволяло значно скоротити час на виконання збірок та тестів, розподіляючи їх між кількома виконавчими середовищами. Швидке виконання CI/CD процесів забезпечує оперативність розробки та впровадження змін.

Також CircleCI надає розширені можливості для аналітики та моніторингу процесів CI/CD, що дозволяє відстежувати продуктивність збірок, виявляти вузькі місця та оптимізувати процеси для забезпечення максимальної ефективності. Детальні звіти та інтеграція з інструментами моніторингу допомагають підтримувати високу якість програмного забезпечення.

Дане середовище постійно розвивається та оновлюється, впроваджуючи нові функції та покращення, що забезпечує актуальність інструменту та можливість використовувати найновіші технології та методології у процесах розробки.

Таким чином, CircleCI був обраний за його простоту, гнучкість, підтримку паралельного виконання завдань, інтеграцію з популярними інструментами та розширені можливості аналітики, що робить його ідеальним вибором для впровадження CI/CD у проекті.

2 РОЗРОБКА ОБЧИСЛЮВАЛЬНОЇ ІНФРАСТРУКТУРИ РОЗГОРТАННЯ ВЕБРЕСУРСУ НА AWS

2.1 Обгрунтування вибору інфраструктурних рішень для розроблюваної обчислювальної інфраструктури

Для розгортання вебресурсу вибрано хмарну платформу Amazon Web Services, що зумовлено її високою надійністю, масштабованістю та безпекою.

При виборі інфраструктурних рішень для реалізації обчислювальної інфраструктури необхідно враховувати кілька ключових аспектів, які сприяють ефективному використанню ресурсів та забезпечують оптимальну продуктивність додатку.

Сервіс Amazon Elastic Compute Cloud (EC2) надає масштабовані віртуальні сервери, що дозволяють запускати та керувати додатками у хмарі. Використання EC2 забезпечує гнучкість у виборі обчислювальних ресурсів, таких як типи екземплярів (наприклад, t2, t3, m5), що дозволяє оптимізувати витрати та продуктивність. EC2 екземпляри також забезпечують можливість автоматичного масштабування (Auto Scaling) для адаптації до змін навантаження. Це означає, що система може автоматично збільшувати або зменшувати кількість запущених інстанцій в залежності від поточного навантаження, забезпечуючи стабільну роботу додатку при пікових навантаженнях та оптимізуючи витрати під час низького навантаження. Додатково, Amazon EC2 підтримує різноманітні операційні системи та налаштування, що дозволяє адаптувати середовище до специфічних потреб проекту [11].

Сервіс Amazon Relational Database Service (RDS) спрощує управління реляційними базами даних, такими як PostgreSQL, MySQL, MariaDB, Oracle та SQL Server. Використання RDS дозволяє автоматизувати завдання адміністрування баз даних, такі як резервне копіювання, відновлення після збоїв, оновлення програмного забезпечення та масштабування. RDS забезпечує високу доступність та надійність за допомогою опцій Multi-AZ (мульти-аварійних зон), що забезпечує безперервність роботи навіть при збоях у одній з зон доступності. Це критично важливо для додатків, що вимагають постійного доступу до даних, мінімізуючи

ризик простоїв та втрати даних. Крім того, RDS пропонує автоматичне масштабування сховищ та обчислювальних потужностей, що дозволяє динамічно адаптуватися до змін обсягів даних та навантаження [12].

Сервіс Amazon CloudWatch забезпечує моніторинг та управління ресурсами AWS та додатками. CloudWatch збирає та аналізує метрики продуктивності, журнали та сповіщення, що дозволяє своєчасно реагувати на проблеми та забезпечувати стабільну роботу додатку. Моніторинг включає відстеження використання процесора, пам’яті, дискового простору та мережевого трафіку. Завдяки цьому розробники можуть оперативно виявляти та вирішувати проблеми, оптимізувати ресурси та планувати майбутні потреби в інфраструктурі. Також CloudWatch дозволяє налаштовувати сповіщення та автоматичні дії у відповідь на певні події, що підвищує загальну надійність системи [13].

Безпека в AWS забезпечується шляхом використання ролей та політик, які визначають права доступу до ресурсів. Групи безпеки (Security Groups) забезпечують контроль доступу на рівні мережі, дозволяючи створювати правила для управління EC2 екземплярів. Використання SSL сертифікатів забезпечує шифрування даних під час передачі між клієнтом та сервером, що захищає від перехоплення даних. AWS надає засоби для моніторингу безпеки та виявлення загроз, включаючи AWS Shield для захисту від DDoS атак та AWS GuardDuty для інтелектуального виявлення загроз. AWS також відповідає багатьом міжнародним стандартам безпеки та регуляторним вимогам, що додає додатковий рівень довіри до використання цієї платформи.

Платформа GitHub є однією з найпопулярніших платформ для розробки програмного забезпечення, підтримуваною великою спільнотою розробників. Завдяки цьому, користувачі мають легкий доступ до численних ресурсів, таких як документація, приклади коду та відкриті проекти, це сприяє швидкому вирішенню проблем та обміну досвідом серед розробників.

GitHub добре інтегрується з різними інструментами CI/CD, такими як CircleCI, Jenkins та GitHub Actions, що дозволяє автоматизувати процеси збірки,

тестування та розгортання без необхідності налаштування додаткових інтеграцій. Така інтеграція забезпечує безперервний потік розробки та підтримки проекту.

Забезпечення потужного контролю версій та інструментів для управління кодом, включаючи pull-запити, огляди коду та спільну роботу над гілками, є однією з ключових функцій GitHub, це допомагає підтримувати високу якість коду та сприяє ефективній співпраці між членами команди. Розробники можуть легко відстежувати зміни, обговорювати їх та інтегрувати нові функції [14].

Сервіс пропонує розширені функції безпеки, такі як двофакторна аутентифікація, управління доступом на основі ролей та сканування вразливостей коду, що допомагає забезпечити безпеку проекту та відповідність вимогам безпеки. Завдяки цим функціям розробники можуть захистити свій код від несанкціонованого доступу та потенційних загроз.

Використовуючи потужні можливості інтеграції GitHub з AWS, розробники можуть налаштувати процес безперервної інтеграції та доставки (CI/CD). Це дозволяє автоматично розгортати зміни в коді на інфраструктурі AWS одразу після їх затвердження в GitHub.

Таким чином, використовуючи інтеграцію GitHub для управління кодом та AWS для розгортання та масштабування додатку, забезпечується надійне та ефективне рішення для розробки та підтримки сучасних вебдодатків.

2.2 Розробка структурної схеми обчислювальної інфраструктури розгортання вебресурсу

Для забезпечення надійного та масштабованого розгортання вебдодатку на хмарній платформі AWS була розроблена структурна схема обчислювальної інфраструктури (Додаток Б). У цій архітектурі задіяно кілька ключових компонентів та сервісів, що працюють у тісній взаємодії для досягнення найкращих результатів.

Сховище коду GitHub Repository слугує джерелом вихідного коду, де розробники можуть ініціювати процес розгортання через git push.

Система CI/CD CircleCI автоматично запускає процес розгортання при оновленні коду в репозиторії.

У хмарній платформі важливу роль відіграють групи безпеки Security group та ролі безпеки IAM Security Role, які регулюють правила доступу до ресурсів.

Віртуальний сервер EC2 використовує для запуску Docker контейнери, які містять вебдодаток. Платформа Docker забезпечує контейнеризацію для зручного розгортання.

Для забезпечення безпеки передачі даних використовується сертифікат SSL.

Веб-сервер і зворотній проксі Nginx обробляє запити до додатка, а JavaScript клієнт взаємодіє з сервером через API.

Web Server Gateway Interface (WSGI) сервер Gunicorn виконує запити до веб-фреймворка Django, який реалізує бізнес-логіку додатка.

База даних PostgreSQL використовується для зберігання даних додатка, а керований сервіс баз даних RDS забезпечує надійне управління базою даних у хмарі AWS.

На початковому етапі вихідний код зберігається у репозиторії на GitHub. При кожному внесенні змін до коду (git push) автоматично ініціюється процес CI/CD через систему CircleCI. CircleCI виконує серію завдань, включаючи тестування, збірку та розгортання додатку, що забезпечує стабільність та якість релізів. Після успішного виконання всіх етапів CircleCI здійснює розгортання додатку на хмарній платформі AWS (рисунок 2.1).

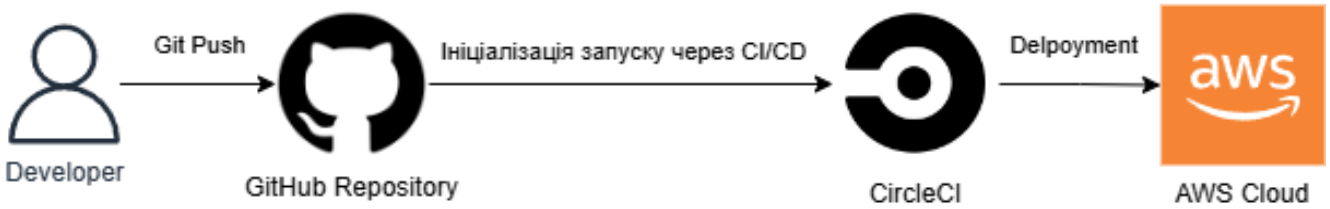

Рисунок 2.1 — Автоматизований процес розгортання через CI/CD на AWS

Вебдодаток розгортається на екземплярах EC2 у середовищі AWS. Для забезпечення безпеки та контролю доступу використовується Security Group, яка регулює правила доступу до серверів, а також IAM Security Role для управління

доступами та безпекою. На екземплярах EC2 запускаються Docker контейнери, що містять різні сервіси вебдодатку (рисунок 2.2), такі як вебсервер, база даних та інші компоненти.

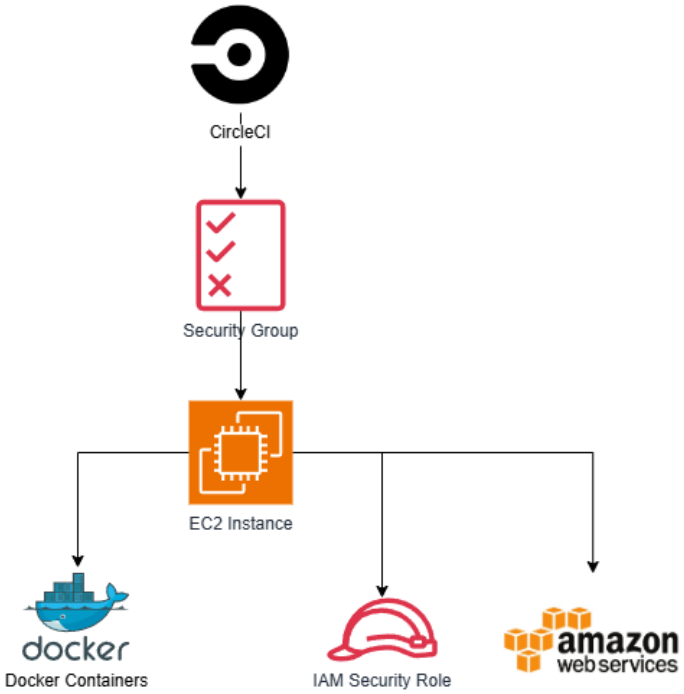

Рисунок 2.2 — Структура розгортання та безпеки на AWS за допомогою CircleCI

Nginx виступає у ролі реверсного проксі та маршрутизатора запитів, розподіляючи трафік між різними сервісами додатку. Запити до API, адміністративної панелі та документації (Swagger, Redoc) переспрямовуються на сервер Django, який працює на порту http://web:8000. Це реалізується за допомогою директиви location ~ ^/(api|admin|swagger|redoc). Водночас запити до основної частини додатку переспрямовуються на клієнтський інтерфейс, реалізований на Vue.js, за адресою http://client:3000. Ця маршрутизація здійснюється через директиву location /, що забезпечує ефективний розподіл запитів між фронтенд і бекенд частинами додатку.

На сервері Django, який працює разом з WSGI сервером Gunicorn, обробляються всі запити бізнес-логіки та взаємодії з базою даних. Django забезпечує виконання основних функцій додатку, включаючи обробку даних користувачів, авторизацію та інші серверні операції. Вебсервер Gunicorn допомагає

оптимізувати продуктивність та забезпечити надійну роботу додатку у продакшн середовищі.

Клієнтська частина додатку реалізована на JavaScript з використанням фреймворку Vue.js, який забезпечує інтерактивність та динамічну взаємодію з користувачами. Vue.js дозволяє створювати реактивні компоненти, що автоматично оновлюються при зміні даних, покращуючи користувацький досвід та підвищуючи ефективність роботи додатку.

Для зберігання та обробки даних використовується база даних PostgreSQL, що розгорнута на сервісі RDS. PostgreSQL забезпечує надійне зберігання даних та потужні можливості для обробки запитів. Дані передаються між серверами за допомогою захищених з'єднань, що використовують SSL сертифікати, які забезпечують шифрування та безпеку передачі інформації (Додаток В).

Таким чином, розроблена обчислювальна інфраструктура для розгортання вебресурсу забезпечує високу ефективність, безпеку та масштабованість додатку, дозволяючи йому надійно обслуговувати користувачів та забезпечувати безперебійну роботу у виробничому середовищі.

2.3 Обґрунтування вибору обчислювальних потужностей та створення EC2 екземпляру

Процес створення EC2 екземплярів для розгортання вебдодатку включає кілька важливих етапів, які забезпечують налаштування обчислювальних ресурсів, мережі та безпеки. У рамках безкоштовного пробного періоду AWS використано тип екземпляру t3.micro, який надає достатню потужність для розгортання та тестування додатку. Екземпляр типу t3.micro забезпечує дві віртуальні CPU та 1 ГБ оперативної пам'яті (таблиця 2.1), що достатньо для базового розгортання додатку [15].

Для вибору образу AMI в AWS запропоновано кілька операційних систем, що продемонстровані на рисунку 2.3. Вибір образу залежить від вимог додатку та середовища розробки.

Таблиця 2.1 — Характеристики екземпляру AWS t3.micro

Параметр	Значення
vCPU	2
Пам'ять (RAM)	1 ГБ
Базова швидкість процесора	2.5 ГГц
Мережевий пропускний канал	До 5 Гбіт/с
Тип зберігання	EBS
Підтримка burstable performance	Так
Вартість за годину (on-demand)	$0.0104

У даному проекті обрано Ubuntu Server, оскілки це одна з найпопулярніших операційних систем для серверів, яка забезпечує стабільність та надійність. Крім того, Ubuntu має широку підтримку спільноти та документацію, що спрощує налаштування та вирішення проблем. Також, дана операційна система сумісна з багатьма інструментами та бібліотеками, що використовуються в сучасній веброзробці, що робить її ідеальним вибором для даного проекту.

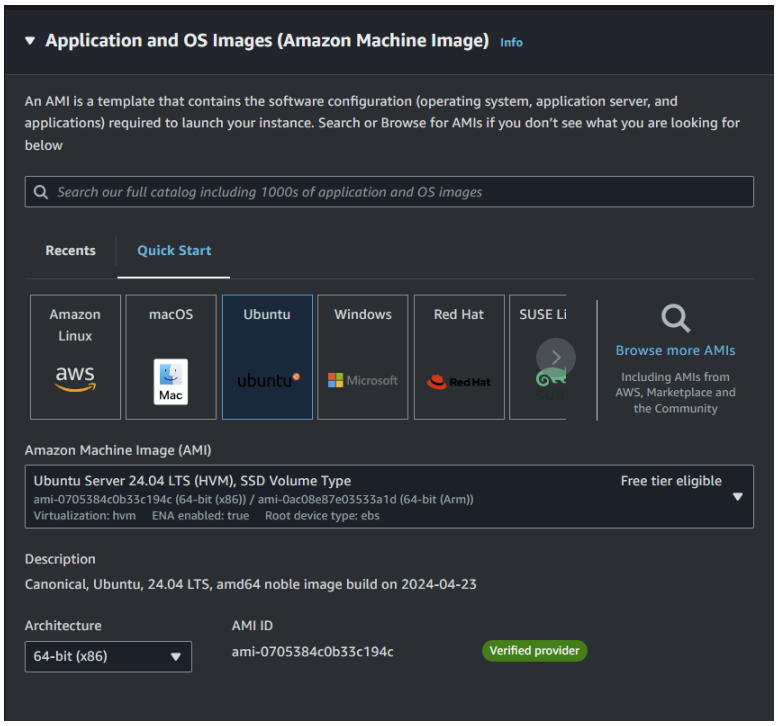

Рисунок 2.3 — Вибір образу програм та ОС

Вибір VPC (Virtual Private Cloud) є важливим етапом у налаштуванні мережі (рисунок 2.4). Можна обрати існуючу VPC або створити нову для екземпляру. VPC забезпечує ізоляцію та контроль над мережею, дозволяючи налаштовувати правила доступу та захисту для обчислювальних ресурсів.

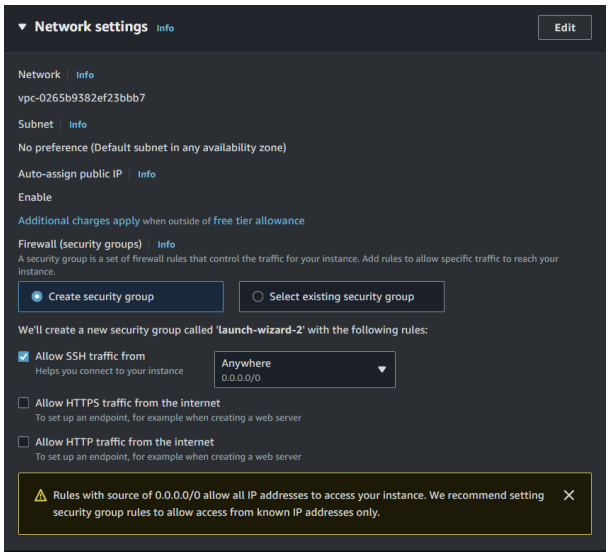

Рисунок 2.4 — Налаштування мережі та конфігурація безпеки

Після цього необхідно вибрати підмережу, в якій буде розміщено екземпляр. Це визначає, які IP-адреси будуть доступні для екземпляру та які мережеві ресурси він зможе використовувати. Вибір підмережі також впливає на доступність та масштабованість додатку, забезпечуючи ефективний розподіл трафіку.

Також важливим аспектом є налаштування інтернет-шлюзу (Internet Gateway). Інтернет-шлюз забезпечує підключення до Інтернету для ресурсів, розміщених у VPC. Це дозволяє екземплярам в VPC здійснювати вихідні з'єднання до Інтернету та отримувати вхідний трафік, необхідний для роботи вебдодатків та інших сервісів.

Створено Security Group, яка визначає правила доступу до екземпляру. Security Group дозволяє налаштувати правила вхідного та вихідного трафіку, забезпечуючи контроль за тим, які порти та IP-адреси мають доступ до екземпляру. Це допомагає захистити ресурси від несанкціонованого доступу та забезпечити безпеку мережевих з'єднань. Також створено та вибрано пару ключів (Key Pair) для SSH доступу до екземпляру. Приватний ключ завантажено та збережено у

безпечному місці, щоб забезпечити безпечний та зручний доступ до серверу. Використання SSH ключів дозволяє значно підвищити рівень безпеки порівняно з традиційними паролями, знижуючи ризик несанкціонованого доступу до серверу.

В даній роботі використано блокове сховище EBS (Elastic Block Store) з об'ємом 30 ГБ, що є достатнім для початкового розгортання (рисунок 2.5). EBS забезпечує надійне та масштабоване сховище, яке можна легко адаптувати до змінних потреб додатку.

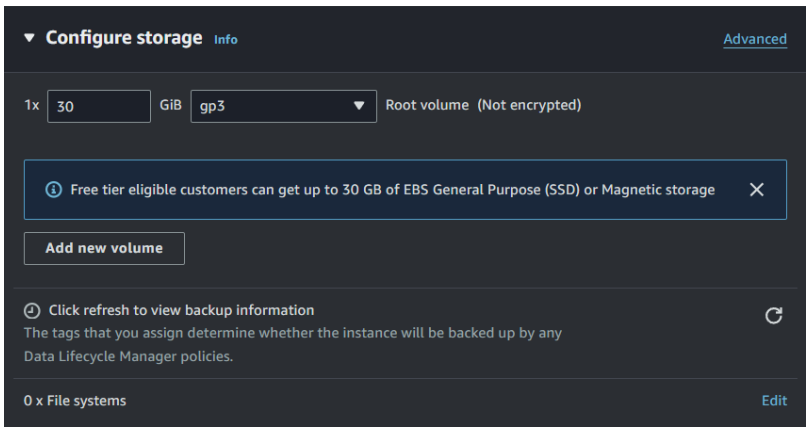

Рисунок 2.5 — Конфігурування сховища екземпляру

Після успішного запуску екземпляру AWS надає його унікальний ідентифікатор (Instance ID), який використовується для подальшого управління та моніторингу ресурсу. Цей ідентифікатор є ключовим для виконання операцій, таких як підключення до екземпляру, зміна його налаштувань або відстеження його стану.

Після запуску екземпляру є можливість використати AWS Management Console для спостереження за його станом, продуктивністю та доступністю. AWS Management Console надає зручний інтерфейс для моніторингу різних метрик та управління ресурсами, що допомагає своєчасно виявляти та вирішувати проблеми.

Для підключення до екземпляру використано SSH за допомогою команди формату «ssh -i /path/to/key.pem ubuntu@instance-public-dns», це дозволяє безпечно підключитися до сервера і виконувати необхідні дії з його налаштування та управління.

Після підключення до екземпляру налаштовано необхідне програмне забезпечення та середовище для запуску вебдодатку. Це включає встановлення веб-серверів, баз даних, інтерпретаторів мов програмування та інших компонентів, необхідних для коректної роботи додатку. Налаштування середовища забезпечує готовність екземпляру до обробки запитів та надання послуг користувачам.

Створення та налаштування EC2 екземплярів є важливим етапом у підготовці інфраструктури для розгортання вебдодатку. Це забезпечує необхідні обчислювальні ресурси, мережеву конфігурацію та безпеку для стабільної та ефективної роботи додатку.

2.4 Налаштування безпечного доступу до розроблювальної обчислювальної інфраструктури

Налаштування безпеки є критично важливим етапом у підготовці середовища для розгортання вебдодатку на AWS. Він включає налаштування ролей та політик IAM (Identity and Access Management), керування доступом до ресурсів через Security Groups, а також забезпечення захисту даних та мережевих з'єднань [18].

Для управління доступом до ресурсів AWS використовується IAM. У даному випадку створено користувача ec2-viewer, який входить до групи container-read. Ця група має відповідні дозволи для доступу до ресурсів, пов'язаних з контейнерами та EC2 екземплярами (рисунок 2.6).

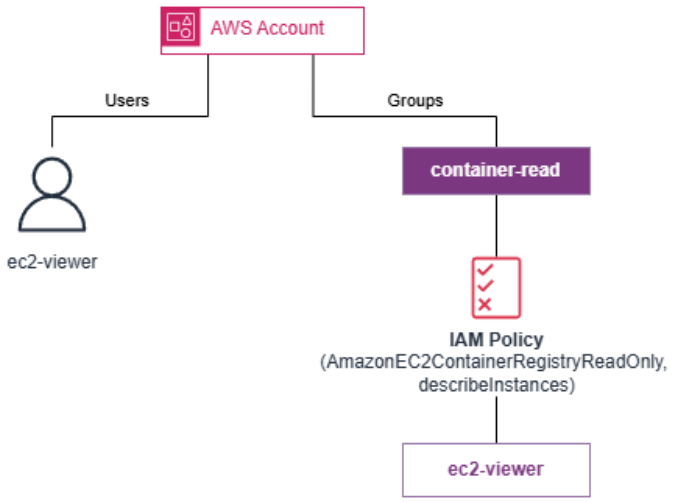

Рисунок 2.6 — Архітектурна діаграма конфігурації безпеки

Група container-read має політику AmazonEC2ContainerRegistryReadOnly, яка надає права на читання ресурсів у Amazon ECR (Elastic Container Registry) та наведена у лістингу 2.1.

Лістинг 2.1 — Політика для доступу до Amazon ECR

```
{"Version": "2012-10-17",
    "Statement": [{
            "Effect": "Allow",
            "Action": [
                "ecr:GetAuthorizationToken",
                "ecr:BatchCheckLayerAvailability",
                "ecr:GetDownloadUrlForLayer",
                "ecr:GetRepositoryPolicy",
                "ecr:DescribeRepositories",
                "ecr:ListImages",
                "ecr:DescribeImages",
                "ecr:BatchGetImage",
                "ecr:GetLifecyclePolicy",
                "ecr:GetLifecyclePolicyPreview",
                "ecr:ListTagsForResource",
                "ecr:DescribeImageScanFindings"],
            "Resource": "*"}]}
```

Також існує політика для доступу до EC2 екземплярів. Дозволи для опису екземплярів та їх статусу визначено у лістингу 2.2

Лістинг 2.2 — Політика для доступу до EC2 екземплярів (describeInstances)

```
{"Version": "2012-10-17",
 "Statement": [{
        "Sid": "VisualEditor0",
        "Effect": "Allow",
        "Action": [
            "ec2:DescribeInstances",
            "ec2:DescribeInstanceStatus"],
        "Resource": "*"}]}
```

Security Groups використовуються для обмеження вхідного та вихідного трафіку до EC2 екземплярів (рисунок 2.7). Вони виконують роль віртуальних брандмауерів, які контролюють доступ до екземплярів на рівні портів і протоколів.

SSH доступ (порт 22) відкритий лише для обмеженого діапазону IP-адрес, що забезпечує захищений доступ до екземпляру через SSH, це важливо для запобігання несанкціонованому доступу до системи.

HTTP доступ (порт 80) відкритий для всіх, що дозволяє користувачам отримати доступ до вебдодатку через HTTP, це стандартний порт для незашифрованого вебтрафіку.

HTTPS доступ (порт 443) також відкритий для всіх, що забезпечує захищений доступ до вебдодатку через HTTPS. Використання цього порту дозволяє шифрувати дані, що передаються між користувачами та вебдодатком, підвищуючи безпеку сервісу.

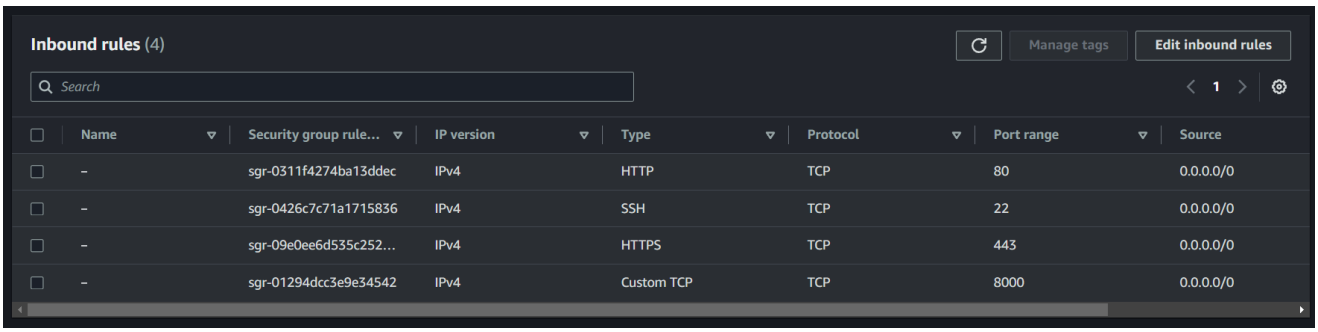

Рисунок 2.7 — Вигляд вікна правил доступу до екземпляру

2.5 Створення та підключення бази даних до інфраструктури

Amazon Web Services пропонує різноманітні рішення для управління базами даних у хмарі. Одним з найбільш популярних сервісів є Amazon Relational Database Service (RDS), який спрощує налаштування, експлуатацію та масштабування реляційних баз даних у хмарі. Він забезпечує економічно ефективні та конфігуровані ресурси баз даних, автоматизує рутинні завдання адміністрування, такі як апаратне забезпечення, налаштування баз даних, оновлення та резервне копіювання [19].

Основні переваги використання Amazon RDS включають легкість використання, оскільки RDS забезпечує просте розгортання баз даних через AWS Management Console, API або інструменти командного рядка. Налаштування та запуск бази даних можна здійснити за лічені хвилини. RDS також автоматизує багато рутинних завдань адміністрування, включаючи резервне копіювання, оновлення програмного забезпечення бази даних, моніторинг і масштабування. Додатково, AWS RDS дозволяє легко масштабувати обсяги зберігання та обчислювальні ресурси бази даних для задоволення змінних потреб. Для забезпечення високої доступності та стійкості до відмов, RDS підтримує автоматичне резервне копіювання, точки відновлення.

Для розгортання вебдодатку обрано базу даних PostgreSQL на платформі RDS. PostgreSQL є відкритою базою даних, що означає відсутність ліцензійних витрат і доступ до великої спільноти розробників та підтримки. Також вона відома своєю можливістю розширення, підтримуючи широкий спектр типів даних, індексів і функцій, які можуть бути адаптовані під специфічні потреби додатків.

Крім того, PostgreSQL підтримує стандарти SQL, що забезпечує сумісність і переносимість додатків, написаних для інших реляційних баз даних. Її міцна архітектура забезпечує надійність і відновлюваність даних, а підтримка транзакцій з ACID-властивостями (атомарність, консистентність, ізольованість, довговічність) робить її надійним вибором для критично важливих додатків. Також PostgreSQL здатна ефективно працювати з великими обсягами даних та складними запитами, що робить її ідеальним вибором для вебдодатків, які потребують надійного і продуктивного управління даними.

Під час створення RDS екземпляру обрано тип екземпляру бази даних, що відповідає вимогам додатку. Для розгортання та тестування використано db.t3.micro (рисунок 2.8), який надається у AWS для безплатного пробного використання. Також обрано кількість vCPU та пам'яті відповідно до очікуваного навантаження на базу даних (таблиця 2.2).

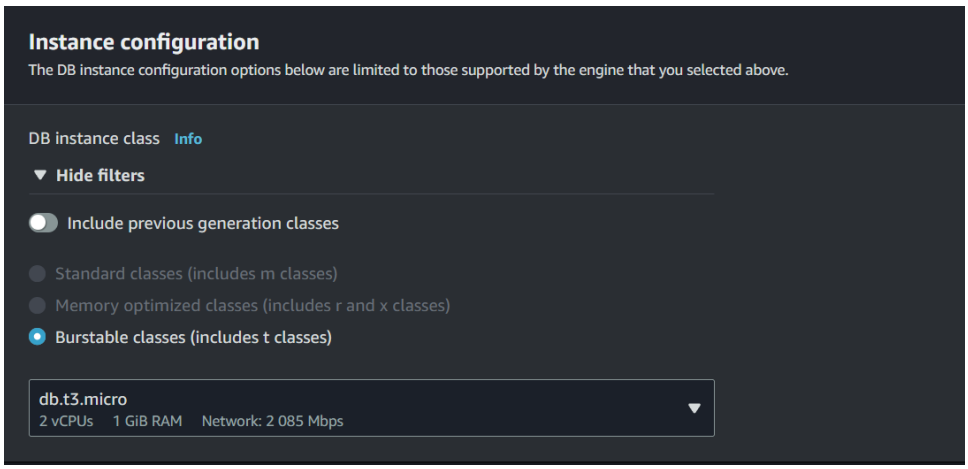

Рисунок 2.8 — Вибір класу екземпляру бази даних

Таблиця 2.2 — Характеристики екземпляру AWS RDS db.t3.micro

Параметр	Значення
vCPU	2
Пам'ять (RAM)	1 ГБ
Базова швидкість процесора	2.5 ГГц
Мережевий пропускний канал	До 5 Гбіт/с
Тип зберігання	EBS
Підтримка burstable performance	Так
Автоматичне резервне копіювання	Так
Підтримка Multi-AZ	Так (за вибором)
Підтримка збереження снапшотів	Так
Сумісність з різними СУБД	MySQL, PostgreSQL, MariaDB, Oracle, SQL Server

Обрано версію PostgreSQL 16.1, яка відповідає вимогам додатку (рисунок 2.9). Також вибір версії обумовлений налаштуваннями вебдодатку.

Налаштування параметрів зберігання для бази даних на платформі RDS включає вибір типу сховища та його початковий розмір. Для забезпечення високої продуктивності обрано тип зберігання «General Purpose (SSD)» (рисунок 2.10). Цей

тип сховища забезпечує відмінне співвідношення вартості та продуктивності, підходить для широкого спектру робочих навантажень і забезпечує низьку латентність доступу до даних.

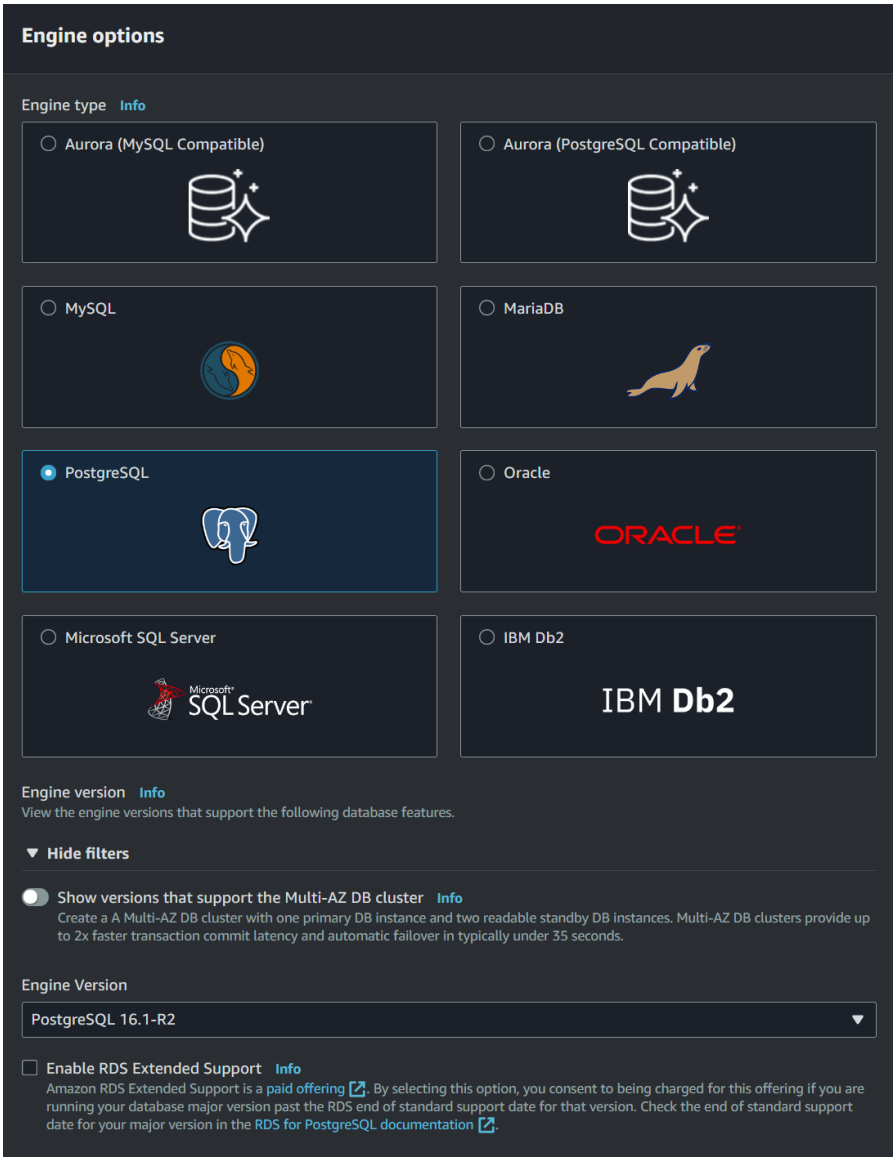

Рисунок 2.9 — Варіанти вибору системи керування базами даних

Також вказано початковий розмір сховища у 20 ГБ (див. рисунок 2.10). Цей розмір є достатнім для більшості додатків на початковому етапі, забезпечуючи необхідний простір для зберігання даних та логів без зайвих витрат. Розмір сховища можна збільшувати в міру зростання обсягів даних та потреб додатка, що дозволяє гнучко реагувати на зміни навантаження і зберігати оптимальну продуктивність.

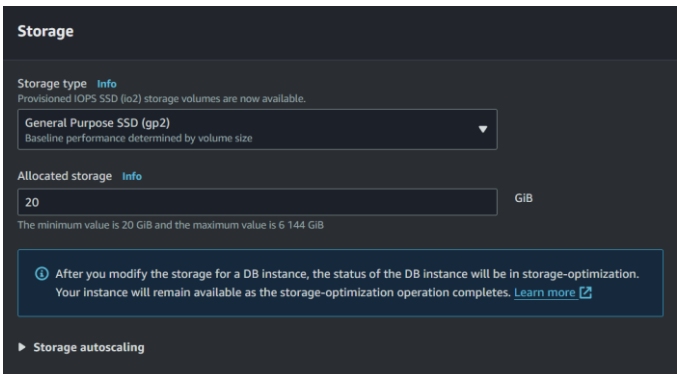

Рисунок 2.10 — Вибір типу та розміру сховища

Обрано VPC, у якій буде розміщено RDS екземпляр. Це повинна бути та сама VPC, що й для EC2 екземпляру, щоб забезпечити легку взаємодію між компонентами додатку.

Обрано Security Group для RDS, яка надає доступ до бази даних з власних EC2 екземплярів. Наприклад, дозволено вхідний трафік на порт 5432 (стандартний порт PostgreSQL) з IP-адрес EC2 екземплярів (рисунок 2.11).

Вказано ідентифікаційні параметри для RDS екземпляру, такі як ім'я бази даних, ім'я користувача адміністратора та пароль. Дані збережено у безпечному місці для подальшого використання.

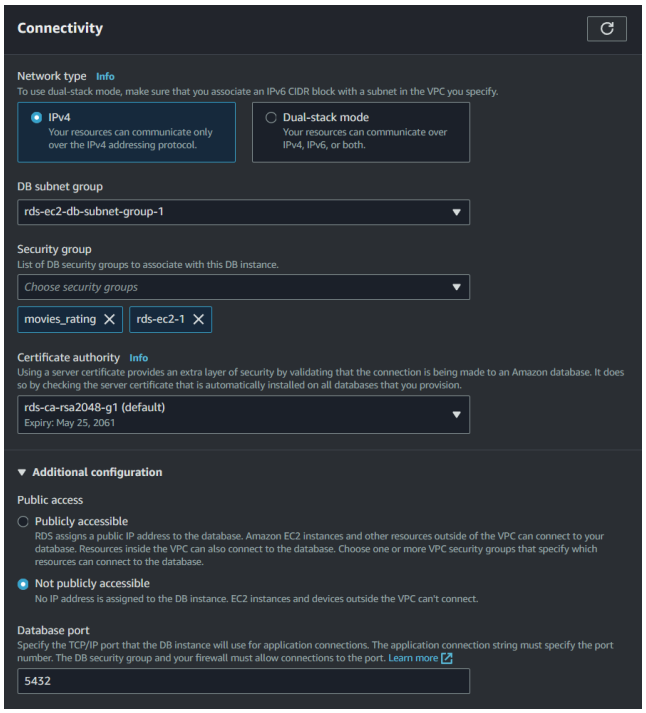

Рисунок 2.11 — Налаштування підключення до бази даних

Увімкнено автоматичне резервне копіювання для RDS екземпляру, що дозволить AWS автоматично створювати щоденні резервні копії бази даних (рисунок 2.12). Період зберігання резервних копій 7 днів, так як використовується тимчасова безплатна версія AWS. Це забезпечить можливість відновлення даних у разі збоїв або помилок.

Налаштовано автоматичне створення точок відновлення (snapshots) для додаткової безпеки, це дозволяє створювати повні знімки стану бази даних у будь-який момент часу (рисунок 2.13). Також можна створювати ручні знімки для збереження важливих версій бази даних перед значними оновленнями або змінами.

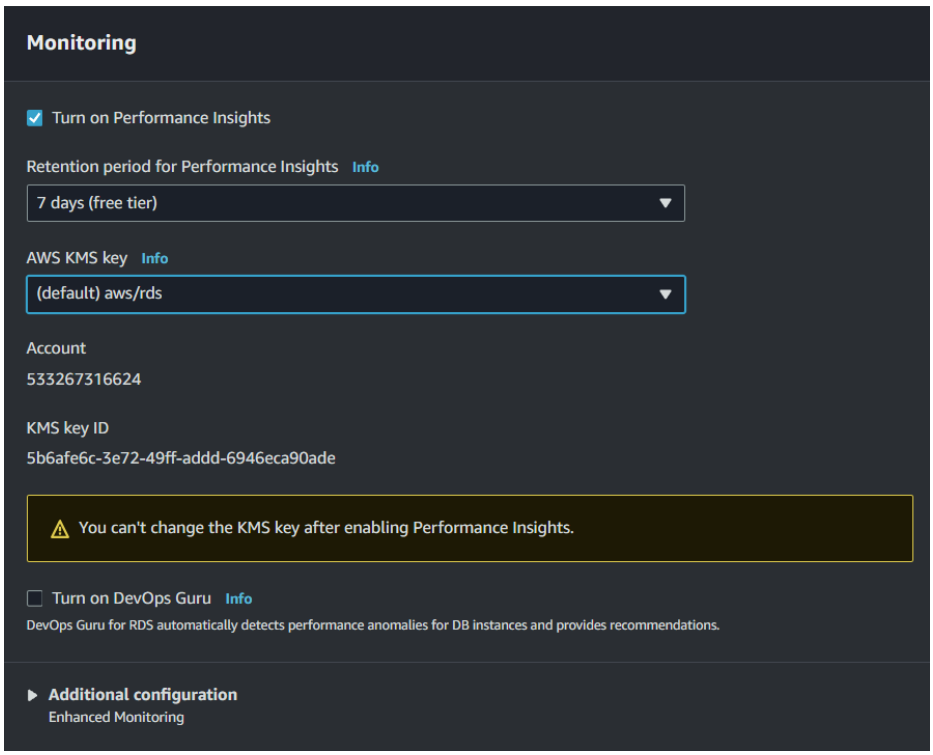

Рисунок 2.12 — Налаштування резервного копіювання бази даних

Налаштовано параметри автоматичного масштабування бази даних відповідно до зміни навантаження, це може включати збільшення об'єму зберігання та збільшення кількості vCPU та пам'яті.

Збережені ідентифікаційні дані використано для підключення вебдодатку до бази даних. Дуже важливо переконатись, що всі налаштування з'єднання відповідають параметрам безпеки та продуктивності.

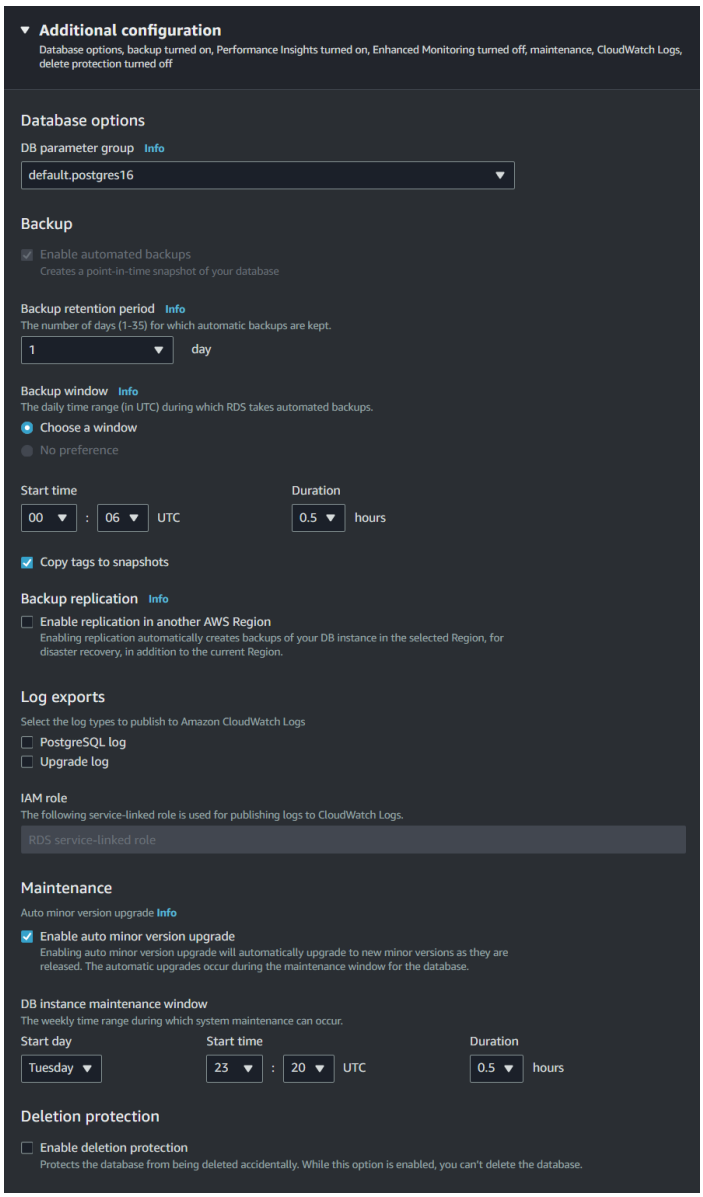

Рисунок 2.13 — Налаштування додаткової конфігурації

2.6 Конфігурування безперервної інтеграції та доставки

Для налаштування CI/CD у проекті використовується CircleCI, що дозволяє інтегрувати автоматизоване тестування та розгортання додатку кожного разу при внесенні змін у репозиторій.

Конфігураційний файл config.yml (Додаток Г) для CircleCI визначає всі необхідні завдання (jobs) та послідовність їх виконання (workflows) [20].

Завдання Prospector необхідне для статичного аналізу коду за допомогою Prospector, це завдання відіграє важливу роль у забезпеченні якості коду та дотримання стандартів програмування.

Першим етапом є завантаження коду з репозиторію, це забезпечує актуальність коду для подальшого аналізу, гарантуючи, що всі останні зміни будуть перевірені. Далі проходить встановлення залежностей. Prospector вимагає певних бібліотек та інструментів для своєї роботи, тому важливо переконатися, що всі необхідні залежності встановлені перед запуском аналізу. Завершальний етап це запуск Prospector для перевірки якості коду. Prospector аналізує код, виявляючи потенційні помилки, недоліки та невідповідності стандартам, це дозволяє розробникам швидко виявляти та виправляти проблеми, підвищуючи загальну якість проекту.

Завдання Tests необхідне для запуску тестів та перевірки покриття коду тестами. Спочатку проходить завантаження коду з репозиторію, це гарантує, що тестування буде проводитися на актуальній версії коду, включаючи всі останні зміни та оновлення. Далі поводиться встановлення залежностей. Для успішного запуску тестів необхідно переконатися, що всі необхідні бібліотеки та інструменти встановлені, щоб середовище тестування було налаштовано правильно. Після цього відбувається запуск тестів з використанням інструменту coverage для вимірювання покриття коду тестами, це дозволяє визначити, яка частина коду перевіряється тестами, і виявити області, які потребують додаткового тестування. Завершальним етапом є генерація звіту про покриття коду тестами. Звіт надає детальну інформацію про результати тестування, включаючи відсоток покриття та конкретні ділянки коду, які були або не були протестовані. Це допомагає розробникам покращити тестування та забезпечити високу якість коду.

Завдання Deploy відповідає за розгортання додатку на середовище розробки. Після завантаження коду та встановлення додаткових бібліотек для розгортання, відбувається додавання SSH ключів для доступу до серверів, що дозволяє автоматизованим процесам підключатися до віддалених серверів без необхідності вручну вводити паролі, забезпечуючи безпеку та зручність. Далі виконується запуск коду розгортання. Код виконує всі необхідні команди для розгортання додатку на сервері, включаючи копіювання файлів, налаштування середовища та

запуск необхідних сервісів. Це забезпечує готовність додатку до використання в середовищі розробки.

Робочий процес (workflow) визначає послідовність виконання завдань (рисунок 2.14). У даному випадку, робочий процес включає три завдання: prospector, tests та deploy_dev. Завдання deploy_dev виконується лише після успішного завершення завдань prospector та tests, що гарантує, що розгортання виконується лише за умови, що код пройшов всі необхідні перевірки.

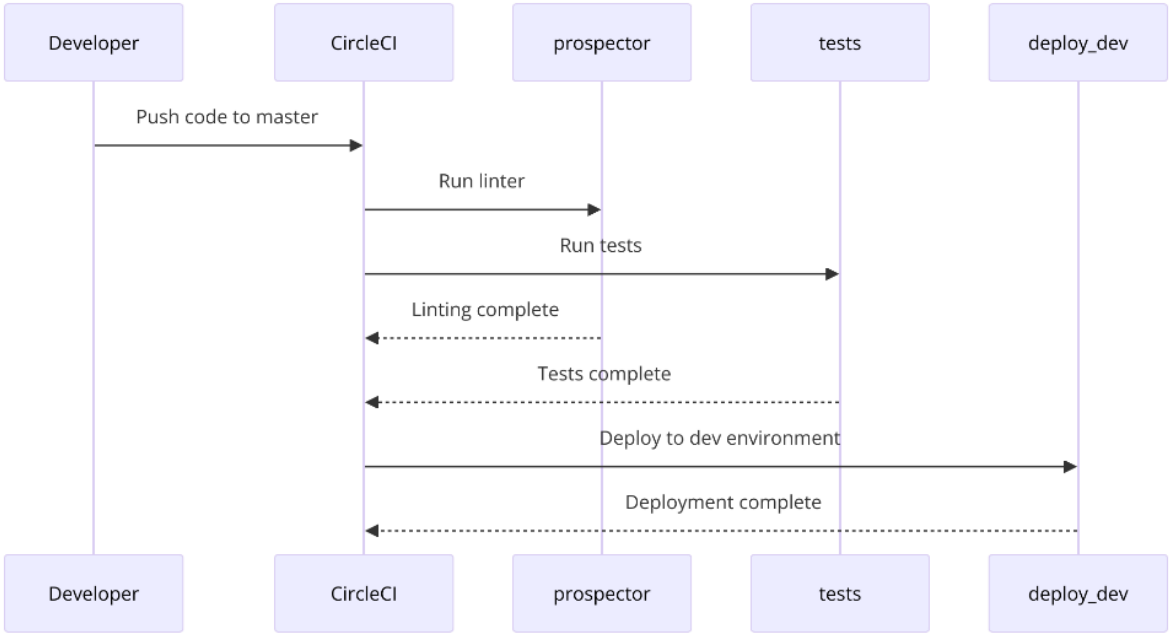

Рисунок 2.14 — Діаграма послідовності процесу конфігурування CircleCI

Впровадження CI/CD у даному проекті приносить ряд значних переваг, що підвищують ефективність розробки та якість кінцевого продукту.

Завдання автоматично запускають тести, що забезпечує постійну перевірку якості коду при кожному внесені змін, що дозволяє вчасно виявляти та виправляти помилки, забезпечуючи стабільність та надійність додатку на всіх етапах розробки.

Автоматичне розгортання дозволяє швидко впроваджувати нові функції та виправлення, що сприяє оперативному реагуванню на потреби користувачів та швидшому випуску оновлень.

Використання інструментів для статичного аналізу коду та покриття тестами допомагає підтримувати високу якість коду. Ці інструменти дозволяють виявляти

потенційні проблеми, дотримуватися стандартів програмування та забезпечувати належне покриття тестами, що знижує ризик помилок у виробничому середовищі.

Зберігання ключів та конфіденційних даних у CircleCI забезпечує їх захищеність, зменшуючи ризик витоку важливої інформації. CircleCI надає засоби для безпечного зберігання та використання конфіденційних даних, що підвищує загальну безпеку проекту.

Впровадження CI/CD у даному проекті забезпечує більш ефективний та надійний процес розробки, сприяючи швидшому впровадженню змін та підтриманню високої якості програмного забезпечення.

Файл deploy.py (Додаток Д) виконує розгортання коду на EC2 екземплярах. Він використовує бібліотеки pexpect для автоматизації SSH сесій та boto3 для взаємодії з AWS. Блок-схему алгоритму розгортання вебзатсосунку наведено у додатку Е.

Клас Pexpect створює SSH сесію з екземпляром, дозволяючи виконувати команди на віддаленому сервері [21].

Функція find_instances() використовує boto3 для пошуку EC2 екземплярів, які відповідають заданим критеріям (наприклад, стан running і тег proj: diploma-project) [22].

Функція deploy_host(instance) виконує команди на віддаленому сервері, такі як зупинка поточних контейнерів, отримання нової версії коду з репозиторію та запуск нових контейнерів.

Функція deploy() знаходить всі відповідні екземпляри та викликає функцію deploy_host() для кожного з них.

2.7 Створення та підключення SSL сертифікатів для вебдодатку

SSL сертифікати забезпечують безпечне з'єднання між сервером та клієнтами, шифруючи дані, що передаються. Сертифікат створено через спеціальний сервіс — ZeroSSL, його використання дозволяє легко отримати і налаштувати SSL сертифікати для вебдодатку [23].

ZeroSSL пропонує інтуїтивно зрозумілий інтерфейс для створення та керування SSL сертифікатами, це робить процес отримання та налаштування сертифікатів простим і зрозумілим навіть для користувачів з обмеженим досвідом у цій сфері.

Однією з головних переваг є можливість отримання безкоштовних сертифікатів на 90 днів, що дозволяє малим та середнім підприємствам, а також особистим вебсайтам забезпечити безпеку з'єднань без додаткових витрат.

ZeroSSL пропонує варіативність методів підтвердження володіння доменом, включаючи підтвердження через email, DNS та HTTP, це дає користувачам гнучкість у виборі найбільш зручного та підходящого для них методу, забезпечуючи додатковий рівень зручності та безпеки при встановленні сертифікатів.

Після реєстрації домену, його необхідно вказати при створенні сертифікату, у даному випадку це diploma.ryzhenko.com. Також обрано безкоштовний тип сертифіката, який реєструється на 90 днів.

ZeroSSL пропонує кілька методів підтвердження володіння доменом.

Перший метод включає підтвердження запиту, що надісланий на один із запропонованих email-адрес. Це швидкий і зручний спосіб, який дозволяє підтвердити володіння доменом шляхом натискання на посилання в електронному листі.

Другий метод полягає в створенні CNAME запису у налаштуваннях DNS домену. Цей спосіб є корисним для тих, хто має доступ до налаштувань DNS і може швидко додати необхідний запис для підтвердження володіння доменом.

Третій метод передбачає завантаження спеціального файлу у кореневий каталог вебсайту. Цей спосіб зручний для користувачів, які мають доступ до файлів вебсайту і можуть легко додати файл для підтвердження.

Ці різноманітні методи підтвердження забезпечують гнучкість та зручність у виборі найкращого варіанту для кожного користувача, що робить процес отримання SSL сертифікатів максимально простим і швидким.

У даній роботі підтвердження проходило через CNAME запис у налаштуваннях DNS домену, так як він був найефективніший серед запропонованих.

Після чого завантажено та скопійовано на сервер сертифікат, приватний ключ та CA-bundle (комплект сертифікатів від сертифікаційного центру).

Сконфігуровано файл вебсервера (Nginx) для використання SSL сертифікату, а саме додано блоки серверу, щоб включити SSL сертифікат і ключі та налаштовано перепосилання з HTTP на HTTPS для забезпечення безпеки всіх з'єднань (лістинг 2.3).

Лістинг 2.3 — Файл конфігурування вебсервера

```
server {
    listen      80;
    server_name _;
    charset     utf-8;
    include     /etc/nginx/mime.types;
    root /;
    return 301 https://$host$request_uri;}
server {
    listen               443 ssl;
    ssl_certificate      /etc/ssl/certificate.crt;
    ssl_certificate_key  /etc/ssl/private.key;
    server_name  _;}
```

Після конфігурування перезапущено docker контейнери, щоб зміни у конфігурації вебсерверу прийшли в дію, та забезпечити підключення до сайту через https.

Цей процес дозволяє швидко та ефективно налаштувати SSL сертифікати для вебдодатку, забезпечуючи надійне та безпечне з'єднання з користувачами.

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ ВЕБДОДАТКУ ТА ЙОГО РОЗГОРТАННЯ В AWS

3.1 Вибір середовища розробки для реалізації вебдодатку

Для ефективної розробки вебдодатку використовуються інтегровані середовища розробки (IDE), які забезпечують широкий спектр інструментів для написання, тестування та налагодження коду. Як середовища розробки для реалізації вебдодатку використано IntelliJ IDEA та PyCharm від компанії JetBrains.

IntelliJ IDEA є одним з найпотужніших та найпопулярніших середовищ розробки, особливо серед розробників, які працюють з мовами програмування на базі JVM, такими як Java. Однак, IntelliJ IDEA також підтримує багато інших мов програмування, включаючи JavaScript та фреймворк Vue.js, які використовувалися у цьому проекті для розробки фронтенд частини додатку [24].

Однією з головних переваг IntelliJ IDEA є надання інтелектуальних підказок, автозавершення коду та рефакторинг, ці функції допомагають розробникам писати чистий та зрозумілий код, значно скорочуючи час на написання та перевірку програмного забезпечення.

Вбудовані інструменти для налагодження дозволяють легко виявляти та виправляти помилки в коді. IntelliJ IDEA підтримує інтеграцію з популярними фреймворками для тестування, що робить процес тестування коду більш ефективним та зручним.

Інтеграція з Git та іншими системами контролю версій значно спрощує процес управління версіями коду та командної роботи над проектом, це дозволяє розробникам легко відстежувати зміни, працювати над різними гілками коду та забезпечувати цілісність проекту.

IntelliJ IDEA також підтримує інтеграцію з базами даних, що дозволяє розробникам легко управляти базами даних, виконувати SQL-запити та аналізувати результати, це робить роботу з базами даних більш ефективною та зручною, забезпечуючи швидкий доступ до необхідної інформації.

PyCharm є спеціалізованим середовищем розробки для мови програмування Python, яке також розроблено компанією JetBrains. PyCharm використовувалося для

розробки бекенд частини додатку, яка реалізована на основі фреймворку Django [25].

Однією з головних переваг PyCharm є розширені можливості для написання, налагодження та тестування Python-коду. Середовище підтримує популярні фреймворки, такі як Django, Flask та інші, що робить його зручним для розробників Python.

Інтелектуальні інструменти для рефакторингу дозволяють безпечно та швидко змінювати структуру коду, забезпечуючи його чистоту та підтримку, це значно спрощує процес оновлення та оптимізації коду без ризику внесення помилок.

PyCharm, як і IntelliJ IDEA, підтримує інтеграцію з Git та іншими системами контролю версій, це полегшує спільну роботу над проектом, забезпечуючи можливість відстежувати зміни та управляти версіями коду в режимі реального часу.

Також PyCharm підтримує інтеграцію з популярними фреймворками для тестування, такими як pytest, це дозволяє автоматизувати процес тестування та забезпечувати високу якість коду, зменшуючи кількість помилок та підвищуючи ефективність розробки.

Обидва середовища розробки IntelliJ IDEA та PyCharm забезпечують високу продуктивність роботи завдяки інтелектуальним підказкам коду, автозавершенню, інтеграції з базами даних та різним інструментам для тестування та налагодження, що дозволяє розробникам швидше знаходити та виправляти помилки, покращувати якість коду та забезпечувати безперервний розвиток проекту.

Крім того, використання таких потужних IDE як IntelliJ IDEA та PyCharm дозволяє розробникам зосередитися на логіці додатку та бізнес-вимогах, а не на технічних аспектах налаштування середовища розробки, що значно підвищує ефективність роботи.

3.2 Використання системи контролю версій Git та організація процесу відправки змін на GitHub

Git є однією з найпопулярніших систем контролю версій, яка використовується для управління та відстеження змін у коді програмних проектів. Вона дозволяє розробникам працювати над спільними проектами, зберігати історію змін, об'єднувати різні версії коду та відновлювати попередні версії у разі необхідності. Завдяки розподіленій природі Git, кожен розробник має повну копію історії проекту, що підвищує надійність та безпеку даних [26].

Однією з основних операцій у Git є запис змін (commit), який зберігає зміни у локальному репозиторії, це дозволяє фіксувати прогрес роботи та забезпечує можливість повернутися до конкретного стану проекту в майбутньому. Записи змін є ключовими елементами історії проекту, оскільки вони містять інформацію про зміни, автора змін та час їх внесення.

Для спільної роботи над проектом розробники використовують віддалені репозиторії, зокрема популярною платформою для цього є GitHub. Він надає інструменти для хостингу репозиторіїв, спільної роботи над кодом, керування запитами на злиття (pull requests) та відстеження помилок. Віддалений репозиторій на GitHub є центральним місцем, де зберігається найактуальніша версія проекту та де розробники можуть обмінюватися змінами [27].

Процес відправки змін з локального репозиторію на віддалений репозиторій GitHub називається push. Після того, як розробник завершив роботу над певною функцією або виправленням, він виконує commit, а потім використовує команду git push, щоб передати свої зміни на віддалений сервер, що дозволяє іншим учасникам проекту отримати доступ до останніх оновлень та інтегрувати їх у свої локальні копії (рисунок 3.1).

Для забезпечення якісного злиття змін та уникнення конфліктів, розробники часто використовують pull-запити (pull requests) на GitHub. Коли розробник вважає свою роботу завершеною і готовою до інтеграції у головну гілку проекту, він створює pull-запит. Інші учасники проекту можуть переглянути запропоновані

зміни, обговорити їх та внести корективи перед остаточним злиттям, це забезпечує контроль якості коду та сприяє співпраці у команді.

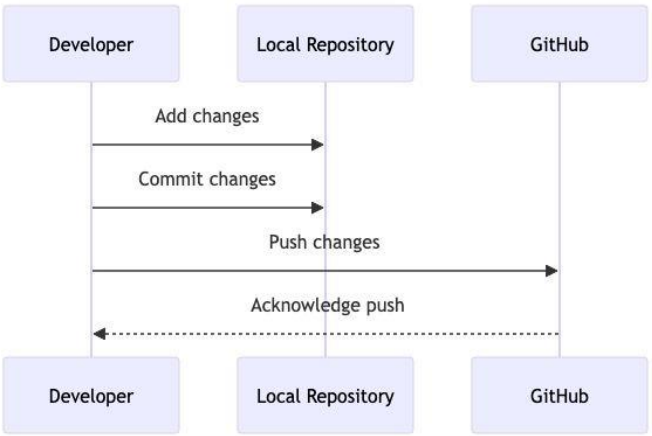

Рисунок 3.1 — Діаграма послідовності процесу завнтаження даних на GitHub

Таким чином, використання Git та GitHub створює ефективну систему для управління версіями коду, спільної роботи над проектами та забезпечення високої якості програмного забезпечення. Завдяки цим інструментам розробники можуть легко відстежувати зміни, співпрацювати з іншими та зберігати цілісність своїх проектів.

3.3 Побудова архітектури вебдодатку

У даній роботі вебдодаток є сервісом, який був розроблений для прикладу, щоб використати для розгортання в розробленій інфраструктурі. Визначено модульну архітектуру, яка забезпечує ізоляцію компонентів та полегшує процеси розгортання та масштабування.

Даний вебдодаток призначений для управління базою даних фільмів, осіб, які брали участь у їх створенні, та їх рейтингами. Він надає користувачам можливість переглядати, додавати, оновлювати та видаляти інформацію про фільми, осіб (режисерів, акторів тощо), та їхні оцінки. Основними функціональними модулями є модулі для роботи з фільмами, особами, зв'язками між ними, а також рейтинговою системою.

Модуль фільмів дозволяє користувачам переглядати список фільмів з фільтрацією та сортуванням за різними параметрами, такими як рік виходу, жанри та середня оцінка. Користувачі можуть також переглядати деталі конкретного фільму, включаючи його назву, жанри, режисерів та постер.

Модуль осіб надає можливість керувати інформацією про людей, які беруть участь у створенні фільмів, зокрема акторів та режисерів

Модуль зв’язків між фільмами та особами дозволяє відображати та керувати зв'язками між фільмами та особами, які працювали над ними. Це включає інформацію про ролі, які виконували особи у фільмах, такі як акторські ролі або режисерські обов'язки.

Модуль рейтингів забезпечує можливість для перегляду, додавання та оновлення рейтингів фільмів. Користувачі можуть переглядати середні оцінки фільмів та кількість голосів, які вони отримали. Це дозволяє користувачам отримувати уявлення про популярність та якість фільмів.

Додаток також включає модуль для управління користувачами та їх аутентифікацією. Користувачі можуть реєструватися, вводячи електронну пошту, ім'я користувача та пароль, а також переглядати інформацію про свій обліковий запис. Цей модуль забезпечує безпеку та контроль доступу до можливостей додатка.

Даний вебдодаток є повноцінною платформою для управління інформацією про фільми та забезпечує користувачам зручний інтерфейс для взаємодії з базою даних фільмів, їх рейтингів та осіб, що брали у них участь.

Для контейнеризації використовується Docker, що дозволяє легко управляти сервісами та їх залежностями (Додаток Ж). Використання Docker в даному вебдодатку забезпечує ряд значних переваг, які підвищують ефективність та надійність розробки та розгортання.

Кожен сервіс додатку, такий як база даних, кеш, вебсервер, клієнтський інтерфейс, працює в окремому контейнері. Це гарантує, що конфлікти між залежностями різних сервісів виключені, і кожен сервіс має своє незалежне середовище виконання. Ізоляція сервісів підвищує стабільність та безпеку додатку.

Docker дозволяє швидко розгорнути додаток на будь-якому сервері або хмарній платформі. Контейнери можна легко перенести з одного середовища в інше, що спрощує як розробку, так і виробниче серидовище. Це забезпечує гнучкість у виборі інфраструктури та скорочує час розгортання.

Використовуючи Docker, можна легко масштабувати сервіси додатку. Наприклад, якщо навантаження на вебсервіс зростає, можна запустити додаткові контейнери цього сервісу, розподіляючи навантаження між ними. Це дозволяє ефективно управляти ресурсами та забезпечувати високу доступність додатку.

Усі залежності додатку визначаються у Dockerfile та docker-compose.yml, що спрощує управління ними та гарантує, що всі розробники та сервери використовують однакові версії бібліотек та інструментів.

Взаємозв'язок між Dockerfile та docker-compose.yml у проекті полягає у їхньому спільному використанні для створення та оркестрації багатоконтейнерного додатка. Dockerfile визначає, як створити образ для сервісу web, включаючи всі необхідні залежності, конфігурації та команди для запуску додатка. Це базовий крок, який забезпечує створення стандартизованого та відтворюваного середовища для даного додатка.

Dockerfile використовується у docker-compose.yml для побудови сервісу web. У файлі docker-compose.yml описані всі сервіси, які складають додаток, включаючи базу даних (db), кешуючий сервер (redis), клієнтський інтерфейс (client) та зворотній проксі сервер (nginx). Docker Compose використовує образ, створений за допомогою Dockerfile, для запуску сервісу web, забезпечуючи його налаштування та інтеграцію з іншими сервісами [28].

У файлі docker-compose.yml також визначені залежності між сервісами, що гарантує правильний порядок їх запуску. Наприклад, сервіс web залежить від db та redis, що означає, що ці сервіси будуть запущені перед запуском web. Це важливо для забезпечення коректної роботи додатка, оскільки веб-сервіс потребує доступу до бази даних і кешу для виконання своїх функцій.

Крім того, Docker Compose додає додаткові конфігурації до кожного сервісу, такі як об'єми для зберігання даних і статичних файлів, порти для доступу до

сервісів та змінні середовища, що налаштовують поведінку додатка. Це дозволяє легко керувати конфігураціями і змінювати їх без необхідності редагувати Dockerfile.

У додатку И подано структурну схему сервісів і їх залежностей у середовищі docker compose для вебдодатка. Кожен сервіс має свої специфічні ролі та взаємодії з іншими компонентами системи, що забезпечує злагоджену роботу додатка.

Сервіс «nginx: 1.25.4» відповідає за обробку HTTP-запитів до вебдодатка. Він використовує том static для зберігання статичних файлів, таких як HTML, CSS, JavaScript, та зображення. Конфігураційні файли Nginx зберігаються в об'ємі config/nginx.conf. Цей сервіс залежить від сервісу client, тобто він запускається після того, як клієнтська частина додатка буде готова.

Том static використовується для зберігання статичних файлів, до яких мають доступ як nginx, так і web. Це забезпечує централізоване зберігання статичних ресурсів, що полегшує їхнє оновлення і використання.

Сервіс «client: Node 21.6.2» відповідає за клієнтську частину вебдодатка. Він також використовує том static для зберігання статичних файлів, які генерує фронтенд. Цей сервіс залежить від вебсервісу web, тобто він запускається після того, як серверна частина додатка буде готова. Для зберігання специфічних даних клієнта використовується том client.

«db: Postgres 16.1» є основною базою даних, де зберігаються всі постійні дані вебдодатка, включаючи інформацію про фільми, користувачів та їхні рейтинги. Для зберігання даних бази використовується том pgdata. Цей сервіс також залежить від вебсервісу web [29].

Redis використовується як кешуючий сервер, який допомагає зберігати тимчасові дані для швидшого доступу. Він відіграє важливу роль у зниженні навантаження на базу даних і підвищенні продуктивності додатка. Сервіс redis залежить від вебсервісу web [30].

Сервіс «web: movies_rating_web» є центральною серверною частиною додатка. Він забезпечує взаємодію між клієнтською частиною, базою даних та кешуючим сервером. Вебсервіс використовує том static для доступу до статичних

файлів і том pgdata для зберігання даних у базі Postgres. Він залежить від сервісів redis та db, тобто запускається після того, як ці сервіси будуть готові [31].

Ця структура сервісів забезпечує модульність і гнучкість вебдодатка, дозволяючи легко масштабувати і управляти кожним компонентом окремо.

Використання Dockerfile (лістинг 3.1) дозволяє визначити всі залежності та команди для створення середовища, необхідного для запуску вебсервісу на Python з використанням фреймворку Django.

Лістинг 3.1 — Код файлу Dockerfile

```
FROM python:3.10
ADD requirements.txt /requirements.txt
RUN apt update; apt install -y gettext; rm -rf /var/apt/cache
RUN pip install -r /requirements.txt
ADD . /src
WORKDIR /src
CMD pip install -r requirements.txt && invoke runit
```

Перший рядок коду визначає базовий образ для контейнера. Використовується офіційний образ Python версії 3.10, що забезпечує стабільне середовище для виконання Python-коду, оскільки офіційні образи ретельно підтримуються та містять всі необхідні компоненти для роботи з Python.

Наступний рядок копіює файл requirements.txt з поточного каталогу на хост-машині в контейнер, це дозволяє контейнеру мати доступ до списку залежностей, які необхідно встановити для роботи додатку.

Оновлення пакетів та встановлення утиліт призначена для оновлення списку пакетів та встановлення утиліти gettext, яка може використовуватися для локалізації. Після встановлення утиліти очищується кеш apt, щоб зменшити розмір образу. Очищення кешу допомагає оптимізувати використання дискового простору в контейнері.

Четвертий рядок встановлює всі необхідні залежності, визначені у файлі requirements.txt, використовуючи pip, це гарантує, що всі залежності, необхідні для роботи додатку, будуть встановлені в контейнері.

Далі проводиться копіювання всіх файли з поточного каталогу на хост-машині в контейнер в каталог /src, що дозволяє контейнеру мати доступ до всіх

файлів додатку, включаючи вихідний код, конфігураційні файли та інші необхідні ресурси.

Через команду WORKDIR /src усі наступні команди будуть виконуватися з цього каталогу, що забезпечує зручність і організованість при виконанні команд.

В останньому рядку команда pip install -r requirements.txt переконується, що всі залежності актуальні перед запуском додатку, а команда invoke runit запускає додаток відповідно до налаштувань, визначених в проекті.

Dockerfile забезпечує повний процес налаштування контейнера, від визначення базового образу до запуску вебдодатку, гарантуючи стабільність і надійність середовища виконання.

3.3.1 Конфігурування бази даних вебдодатку

База даних є одним з ключових компонентів вебдодатку, оскільки вона відповідає за зберігання та управління даними. У архітектурі використовується PostgreSQL, одна з найпопулярніших систем керування базами даних з відкритим вихідним кодом. PostgreSQL відома своєю надійністю, стабільністю та розширюваністю.

Використовується офіційний Docker образ PostgreSQL версії 16.1, що гарантує використання останньої стабільної та безпечної версії програмного забезпечення. Конфігурація бази даних здійснюється через файл .env, що містить змінні середовища для налаштування бази даних. Цей файл включає такі змінні, як POSTGRES_USER, POSTGRES_PASSWORD, та POSTGRES_DB, які визначають ім'я користувача, пароль та ім'я бази даних відповідно.

База даних PostgreSQL працює на внутрішньому порту 5432 контейнера, ззовні він переспрямований на цей ж порт хост-машини. Це дозволяє уникнути конфліктів з іншими сервісами, які можуть використовувати стандартний порт PostgreSQL.

Для забезпечення стійкості даних використовується Docker-об'єм pgdata, який монтується до /var/lib/postgresql/data всередині контейнера. Це означає, що всі дані бази зберігаються поза межами контейнера, і не будуть втрачені при

перезапуску чи видаленні контейнера. Використання об'ємів дозволяє легко створювати резервні копії даних та забезпечувати відмовостійкість.

Контейнер PostgreSQL налаштований так, що інші сервіси (наприклад, вебсервіс Django) можуть легко підключитися до бази даних. У конфігураційному файлі Docker Compose визначено, що вебсервіс залежить від сервісу db. Це гарантує, що база даних буде запущена до старту вебсервісу.

ER-діаграма бази даних наведена у додатку К. У базі даних є чотири основні таблиці: movies_movie, movies_person, movies_rating та movies_personmovie. Кожна з них зберігає різні типи даних, пов'язані з фільмами, особами, рейтингами та участю осіб у фільмах, забезпечуючи повний набір інформації для кінематографічної бази даних. Ці таблиці спільно забезпечують повну структуру для зберігання та обробки кінематографічних даних, дозволяючи ефективно управляти інформацією.

Доступ до бази даних обмежений за допомогою змінних середовища для налаштування пароля та імені користувача. Додатково можна налаштувати правила брандмауера та мережевої ізоляції, щоб забезпечити доступ до бази даних лише з певних IP-адрес або контейнерів.

Використання PostgreSQL як бази даних у контейнеризованому середовищі Docker дозволяє забезпечити надійне та гнучке управління даними, спрощує процеси резервного копіювання та відновлення, а також забезпечує високу доступність і відмовостійкість додатку.

3.3.2 Покращення продуктивності вебдодатку шляхом кешування даних

Для кешування даних та покращення продуктивності вебдодатку використовується сервіс Redis. Це дозволяє значно зменшити навантаження на базу даних та прискорити доступ до часто використовуваних даних. Кешування особливо корисне для зберігання результатів запитів, які часто повторюються, а також для збереження сесій користувачів та тимчасових даних.

Redis — це високопродуктивна, відкритокодова, in-memory система управління базами даних, яка підтримує різні структури даних, такі як рядки, хеші,

списки, множини та інші. Завдяки своїй in-memory архітектурі, Redis забезпечує дуже швидкий доступ до даних, що робить його ідеальним вибором для кешування.

У даному проекті Redis працює в окремому Docker контейнері, що дозволяє ізолювати його від інших сервісів та забезпечує гнучкість у розгортанні. Контейнер запускається з використанням офіційного образу redis. Налаштування Redis за замовчуванням є достатніми для більшості випадків використання, але за потреби його конфігурацію можна легко змінити через Docker Compose.

Однією з основних переваг використання Redis є зменшення кількості звернень до бази даних. Завдяки зберіганню часто запитуваних даних в пам'яті, Redis дозволяє отримувати їх набагато швидше, ніж із традиційної бази даних, а це в свою чергу значно прискорює час відповіді на запити користувачів, покращуючи загальний досвід взаємодії з додатком.

Ще однією важливою перевагою є зниження навантаження на основну базу даних. Використання кешу дозволяє розвантажити основну базу даних, що покращує її загальну продуктивність та стійкість, це особливо корисно під час пікових навантажень, коли кількість одночасних запитів може значно збільшуватися.

Крім того, Redis легко масштабується, це дозволяє обробляти зростаючі обсяги даних та збільшення навантаження без значних змін в архітектурі додатку. Легка масштабованість забезпечує гнучкість та адаптивність системи до змінних умов експлуатації, дозволяючи підтримувати високу продуктивність навіть при збільшенні числа користувачів та обсягу даних.

У Django Redis інтегровано за допомогою бібліотеки django-redis, яка дозволяє використовувати Redis як бекенд для кешування. Налаштування Redis у Django здійснюється через файл docker.py (лістинг 3.2), де конфігурується система кешування.

Лістинг 3.2 — Фрагмент коду docker.py із конфігуруванням Redis

```
CACHES = {
    'default': {
        'BACKEND': 'django.core.cache.backends.redis.RedisCache',
        'LOCATION': 'redis://redis:6379',}}
```

3.3.3 Програмна реалізація архітектури та налаштування Django вебсервісу

Вебсервіс у цій архітектурі реалізований за допомогою фреймворку Django. Він виконує функції бекенду додатку, обробляє запити від фронтенду, взаємодіє з базою даних та іншими сервісами, такими як Redis, для забезпечення бізнес-логіки додатку. Вебсервіс також відповідає за надання RESTful API, яке дозволяє фронтенду та іншим клієнтам взаємодіяти з додатком [32].

Файл base.py містить основні налаштування Django для проекту movies_rating. Він генерується під час створення проекту за допомогою команди django-admin startproject і містить налаштування, необхідні для роботи проекту в середовищі розробки [33].

Файл починається з імпорту необхідних модулів і визначення базового каталогу проекту. BASE_DIR визначає кореневий каталог проекту, що дозволяє легко задавати шляхи до інших файлів і директорій у проекті.

Параметр SECRET_KEY зберігає секретний ключ для шифрування даних, який повинен бути захищений і не публікуватися у відкритому доступі, це значення отримується з простору (os.environ), що забезпечує додатковий рівень безпеки. У режимі розробки може використовуватися значення за замовчуванням, проте для виробничого середовища обов’язково використовувати унікальний та захищений ключ.

Параметр DEBUG визначає, чи включений режим налагодження. У режимі розробки значення True дозволяє відображати детальну інформацію про помилки, що допомагає розробникам швидко знаходити та виправляти проблеми в коді. У виробничому середовищі цей параметр повинен бути встановлений на False, щоб уникнути розкриття внутрішніх деталей системи.

ALLOWED_HOSTS містить список дозволених хостів, які можуть підключатися до проекту. На етапі розробки цей список зазвичай залишається порожнім, що дозволяє підключатися до додатку з будь-якого хоста. Однак у виробничому середовищі важливо вказати конкретні хости або домени, з яких дозволено доступ до додатку, щоб запобігти несанкціонованому доступу.

Секція INSTALLED_APPS містить список додатків, які встановлені в проекті. Ці додатки включають стандартні додатки Django, а також додатки для REST API (rest_framework, drf_yasg), соціальної автентифікації (social_django), фільтрації (django_filters), та користувацькі додатки (apps.authentication, apps.movies).

MIDDLEWARE визначає список проміжних програм, які обробляють запити до того, як вони будуть передані до додатків. middleware у проекті включають засоби безпеки, сесії, захист від CSRF атак, аутентифікацію, повідомлення та захист від клікджекингу.

Налаштування у лістингу 3.3 визначають конфігурацію URL та WSGI додатку.

Параметр ROOT_URLCONF вказує на модуль, що містить конфігурацію URL для проекту. Це визначення дозволяє Django знати, який модуль використовувати для пошуку шляхів URL та обробки запитів. Цей модуль зазвичай містить всі маршрути додатку та прив'язує їх до відповідних представлень (views).

Параметр WSGI_APPLICATION вказує на модуль, що містить налаштування WSGI додатку. WSGI є стандартом для Python вебдодатків і забезпечує точку входу для WSGI-сумісних вебсерверів. Це дозволяє вебсерверу взаємодіяти з Django додатком, обробляючи вхідні HTTP запити та відправляючи відповіді.

Лістинг 3.3 — Конфігурація URL та WSGI додатку

```
ROOT_URLCONF = 'config.urls'
WSGI_APPLICATION = 'config.wsgi.application'
```

Потрібно також налаштувати методи автентифікації та валідації паролів.

Параметр LOGIN_URL вказує URL для сторінки входу, це значення використовується, коли користувачі неавторизовані і їм необхідно виконати вхід для доступу до захищених сторінок або функцій додатку.

Параметр AUTH_USER_MODEL визначає користувацьку модель для аутентифікації, що дозволяє використовувати кастомізовану модель користувача замість стандартної, наданої Django. Вона забезпечує гнучкість у визначенні атрибутів та поведінки користувачів.

Параметр AUTH_PASSWORD_VALIDATORS містить список валідаторів паролів для забезпечення безпеки. Ці валідатори перевіряють, чи відповідає пароль встановленим критеріям безпеки, таким як мінімальна довжина, складність та унікальність, що допомагає захистити додаток від використання слабких паролів.

Налаштування Django REST Framework визначають параметри для роботи з API (лістинг 3.4). Ці налаштування включають класи аутентифікації, класи дозволів, класи пагінації та бекенди для фільтрації та пошуку.

Лістинг 3.4 — Фрагмент коду base.py із конфігуруванням фреймворку

```
REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': (
        'rest_framework.authentication.SessionAuthentication',
       'rest_framework_simplejwt.authentication.JWTAuthentication',),
    'DEFAULT_PERMISSION_CLASSES': (
        'rest_framework.permissions.IsAuthenticated',),
    'DEFAULT_PAGINATION_CLASS':
'rest_framework.pagination.LimitOffsetPagination',
    'PAGE_SIZE': 18,
    'DEFAULT_FILTER_BACKENDS': (
        'django_filters.rest_framework.DjangoFilterBackend',
        'rest_framework.filters.SearchFilter'),}
```

Файл urls.py визначає конфігурацію маршрутизації для проекту Django. Він містить усі URL-шляхи, які обробляються додатком, та прив'язує їх до відповідних представлень (views). У роботі цей файл містить налаштування для адміністративної панелі, аутентифікації, API для фільмів та соціальної автентифікації.

Спочатку імпортуються необхідні модулі, які включають стандартні модулі Django для маршрутизації (path, include, re_path), модулі для роботи з адміністративною панеллю (admin) та REST Framework (permissions). Також імпортуються модулі drf_yasg для генерації документації API.

Для генерації документації API використовується Swagger, де schema_view визначає параметри для генерації документації API, включаючи заголовок, версію, опис, умови використання, контактну інформацію та ліцензію.

Основні маршрути додатку визначаються блоком urlpatterns, що дозволяє організувати доступ до різних функціональних частин додатку.

Маршрут «admin/» відповідає за доступ до адміністративної панелі Django. Ця панель дозволяє адміністраторам керувати даними додатку, створювати, редагувати та видаляти об'єкти, а також виконувати інші адміністративні задачі. Адміністративна панель є важливим інструментом для управління контентом і користувачами.

Маршрут «api-auth/» включає стандартні URL для аутентифікації, які надаються Django REST Framework. Він забезпечує базові механізми аутентифікації користувачів для доступу до API, такі як входження, вихід і управління сесіями. Цей маршрут значно спрощує реалізацію аутентифікації у додатку.

Маршрут «api/v1/auth/» включає URL для аутентифікації користувачів, визначені у додатку authentication. Він забезпечує можливості для реєстрації, входу, виходу користувачів та управління акаунтами, а також відповідає за всі операції, пов'язані з обробкою облікових записів користувачів.

Маршрут «api/v1/movies/» включає URL для роботи з фільмами, визначені у додатку movies. Він дозволяє здійснювати CRUD-операції (створення, читання, оновлення, видалення) з об'єктами фільмів у базі даних. Ще він забезпечує доступ до даних фільмів і є ключовим для функціональності додатку, що стосується управління фільмами.

Маршрути із re_path відповідають за генерацію документації API у форматах Swagger та ReDoc. Ці маршрути надають доступ до інтерактивної документації API, що полегшує розробникам розуміння та використання доступних кінцевих точок API. Документація у форматі Swagger та ReDoc дозволяє легко тестувати API та забезпечує зручний інтерфейс для ознайомлення з його можливостями.

Файл urls.py є центральним місцем для визначення всіх маршрутів у проекті Django. Він включає конфігурацію для адміністративної панелі, аутентифікації користувачів, REST API для фільмів, а також маршрути для генерації документації

API за допомогою Swagger та ReDoc. Використання include дозволяє легко розбити маршрутизацію на логічні блоки, що покращує структуру та читабельність коду.

3.3.4 Інтеграція та управління даними у застосунках movies та authentication

Застосунок movies в рамках вебдодатку на Django відповідає за управління даними про фільми, осіб, пов'язаних з фільмами, та рейтинги фільмів. Він включає моделі, серіалізатори, представлення (views) та маршрутизацію для роботи з цими даними через REST API.

Моделі застосунку movies визначають структуру таблиць бази даних та взаємозв'язки між ними, забезпечуючи зручний спосіб зберігання і обробки інформації про фільми, особи та рейтинги.

Movie — модель, що представляє фільм, включає поля для ідентифікатора IMDB, типу фільму, назви, вікових обмежень, року випуску, жанрів та постеру. Ця модель зберігає основну інформацію про кожен фільм у базі даних, надаючи структуроване сховище для різноманітних атрибутів фільмів.

Person — модель, що представляє особу (актора, режисера тощо), включає поля для ідентифікатора IMDb, імені, дати народження та смерті, відомих фільмів та фото. Вона дозволяє зберігати та обробляти інформацію про людей, які брали участь у створенні фільмів, забезпечуючи можливість відстеження внеску кожної особи в кінематограф.

PersonMovie — модель, що представляє зв'язок між фільмом та особою, включає поля для ідентифікаторів фільму та особи, порядкового номера, категорії, роботи та персонажів. Ця модель дозволяє відстежувати, які особи були залучені до створення конкретних фільмів та в якій ролі, що є важливим для повного розуміння складу та участі команди в кожному проекті.

Rating — модель, що представляє рейтинг фільму, включає поля для ідентифікатора фільму, середнього рейтингу та кількості голосів. Вона зберігає інформацію про рейтинги, надані користувачами, що дозволяє оцінювати популярність та якість фільмів на основі зібраних даних. Це забезпечує зворотний

зв'язок від аудиторії та допомагає користувачам обирати фільми для перегляду на основі оцінок інших глядачів.

Серіалізатори використовуються для перетворення даних моделей у JSON формат для передачі через API, забезпечуючи структуроване та читабельне представлення даних (Додаток Л) [34].

MovieSerializer — цей серіалізатор перетворює дані моделі Movie та включає додаткові поля directors та average_rating, які обчислюються методами get_directors та get_average_rating відповідно. Поле directors збирає імена режисерів, пов'язаних з фільмом, а поле average_rating обчислює середній рейтинг фільму, якщо він існує. Це забезпечує комплексне представлення фільму з додатковими обчислюваними даними, що підвищує інформативність відповіді API.

PersonSerializer — серіалізатор для моделі Person, який перетворює дані про особу, включаючи її ідентифікатор IMDb, ім'я, дати народження та смерті, відомі фільми та фото. Це дозволяє передавати повну інформацію про особу у форматі JSON, що є необхідним для відображення даних про акторів, режисерів та інших учасників кінематографу.

Серіалізатор PersonMovieSerializer обробляє дані про зв'язки між фільмами та особами, включаючи деталі про роль особи у фільмі. Він також використовує вкладений серіалізатор PersonSerializer для представлення даних про особу. Це дозволяє отримувати детальну інформацію про участь кожної особи у створенні фільму, включаючи роль та вклад.

RatingsSerializer — серіалізатор для моделі Rating, який обробляє дані про рейтинги фільмів, включаючи середній рейтинг та кількість голосів. Це забезпечує передачу даних про оцінки фільмів, що дозволяє користувачам бачити рейтинг фільму та кількість людей, які проголосували, що може вплинути на їх рішення про перегляд.

Представлення визначають логіку обробки запитів до API, забезпечуючи взаємодію між користувачами та даними додатку. У застосунку movies використовуються класи представлень на основі Django REST Framework, які дозволяють ефективно обробляти різні типи запитів (Додаток М) [35].

MovieListCreateView — представлення для створення та отримання списку фільмів з підтримкою фільтрації за жанрами, пошуку за назвою та ID IMDb, а також сортування за середнім рейтингом та ID. Це дозволяє користувачам отримувати відповідний перелік фільмів, додавати нові фільми та виконувати пошук і сортування для зручного перегляду.

MoviesRetrieveUpdateDestroyView — представлення для отримання, оновлення та видалення конкретного фільму. Це дозволяє здійснювати повний цикл управління інформацією про окремі фільми, включаючи їхню модифікацію та видалення.

PersonListCreateView — представлення для створення та отримання списку осіб з підтримкою сортування. Користувачі можуть додавати нових осіб та переглядати існуючі, сортувати їх за різними параметрами для зручності навігації.

PersonsRetrieveUpdateDestroyView — представлення для отримання, оновлення та видалення конкретної особи. Це надає можливість керувати даними про окремих осіб, редагувати їх інформацію та видаляти, якщо необхідно.

PersonMovieListCreateView — представлення для створення та отримання списку зв'язків між фільмами та особами з підтримкою пошуку за ID фільму. Це дозволяє додавати нові зв'язки, переглядати існуючі та шукати зв'язки за конкретними фільмами.

PersonMoviesRetrieveUpdateDestroyView — представлення для отримання, оновлення та видалення конкретного зв'язку між фільмом та особою. Це забезпечує можливість детального управління кожним зв'язком, включаючи його редагування та видалення.

RatingsListCreateView — представлення для створення та отримання списку рейтингів з підтримкою пошуку за ID фільму. Користувачі можуть додавати нові рейтинги та переглядати наявні, здійснюючи пошук за конкретними фільмами для зручного доступу до оцінок.

RatingsRetrieveUpdateDestroyView — представлення для отримання, оновлення та видалення конкретного рейтингу. Це дозволяє повністю керувати

інформацією про рейтинги, забезпечуючи можливість їхнього редагування та видалення.

Маршрутизація визначає URL-шляхи для представлень. Маршрути включають URL-шляхи для роботи з фільмами, особами, зв'язками між фільмами та особами, а також рейтингами.

Застосунок movies дозволяє зручно та ефективно управляти інформацією про фільми та пов'язаних з ними осіб, а також зберігати та обробляти рейтинги фільмів. Використання Django та Django REST Framework забезпечує високу гнучкість, масштабованість та зручність інтеграції з іншими компонентами та сервісами вебдодатку. Також застосунок надає всі необхідні інструменти для реалізації CRUD-операцій та забезпечення взаємодії через REST API, що робить його важливим елементом будь-якої кінематографічної платформи чи бази даних.

Застосунок authentication у вебдодатку на Django відповідає за управління користувачами, автентифікацію та авторизацію. Він включає моделі користувачів, серіалізатори, представлення (views) та маршрутизацію для роботи з користувачами через REST API. Основні компоненти проекту authentication включають моделі, серіалізатори, представлення та маршрутизацію.

У проекті authentication використовуються три моделі, а саме UserManager, User та UserToken.

UserManager є користувацьким менеджером для створення користувачів та суперкористувачів. Він забезпечує методи для створення нових облікових записів, встановлення паролів та налаштування можливостей суперкористувача. Це дозволяє ефективно керувати обліковими записами користувачів, забезпечуючи гнучкість і контроль над процесом створення та адміністрування облікових записів.

User є моделлю користувача, яка розширює стандартну модель AbstractUser та використовує електронну пошту як унікальний ідентифікатор. Це дозволяє забезпечити унікальність кожного користувача на основі його електронної пошти, що спрощує процес аутентифікації та управління обліковими записами. Використання електронної пошти як унікального ідентифікатора також покращує безпеку та зручність для користувачів.

UserToken є моделлю для зберігання токенів користувачів, які можуть використовуватися для різних цілей, таких як відновлення паролю або підтвердження електронної пошти. Це забезпечує безпечне зберігання тимчасових токенів, необхідних для виконання певних операцій з обліковими записами користувачів. Наявність такої моделі дозволяє реалізувати різноманітні функціональні можливості, пов'язані з безпекою та управлінням доступом.

У застосунку authentication використано два серіалізатора, а саме UserSerializer для аутентифікації та RegistrationUserSerializer для реєстрації (див. Додаток Л).

UserSerializer — серіалізатор для моделі User, який містить поля для електронної пошти та імені користувача. Цей серіалізатор забезпечує передачу основної інформації про користувача, що необхідна для аутентифікації та управління обліковими записами.

RegistrationUserSerializer — серіалізатор для реєстрації користувачів, який включає додаткову логіку для перевірки паролів, та забезпечує правильність введених паролів, перевіряючи їх відповідність встановленим критеріям безпеки, що допомагає уникнути потенційних проблем з безпекою та забезпечує коректну реєстрацію нових користувачів.

Представлення визначають логіку обробки запитів до API, забезпечуючи взаємодію між користувачами та системою. У проекті authentication використовуються класи представлень на основі Django REST Framework (див. Додаток М).

CurrentUserDetailsView — це представлення для отримання даних поточного користувача. Воно використовує серіалізатор UserSerializer і повертає дані аутентифікованого користувача, що дозволяє отримати інформацію про поточного користувача після успішної аутентифікації.

UserRegistrationView — це представлення для реєстрації нових користувачів. Воно використовує серіалізатор RegistrationUserSerializer і дозволяє будь-якому користувачеві, навіть неаутентифікованому, створювати нові облікові записи. Це

представлення забезпечує процес реєстрації, включаючи валідацію та збереження нових користувачів у системі.

Маршрутизація визначає URL-шляхи для представлень, забезпечуючи доступ до них через HTTP-запити.

TokenObtainPairView — стандартне представлення Django REST Framework Simple JWT для отримання токенів доступу та оновлення. Воно дозволяє користувачам отримувати пару токенів (access і refresh) після успішної аутентифікації.

TokenRefreshView — представлення для оновлення токенів. Це дозволяє користувачам отримувати новий токен доступу, використовуючи токен оновлення, коли поточний токен доступу закінчився.

CurrentUserDetailsView — маршрут для отримання інформації про поточного користувача. Цей маршрут забезпечує доступ до даних аутентифікованого користувача через відповідне представлення.

UserRegistrationView — маршрут для реєстрації нового користувача. Цей маршрут надає інтерфейс для створення нових облікових записів через відповідне представлення, підтримуючи процес реєстрації користувачів у системі.

Застосунок authentication забезпечує можливості управління користувачами, автентифікації та авторизації у вебдодатку. Він використовує потужні можливості Django та Django REST Framework для реалізації цих завдань. Завдяки чітко визначеним моделям, серіалізаторам, представленням та маршрутизації, проект authentication є гнучким і легко розширюваним, що дозволяє інтегрувати його у більш складні вебдодатки.

3.3.5 Реалізація клієнтської частини вебдодатку

Клієнтський інтерфейс вебдодатку реалізований за допомогою JavaScript з використанням сучасного фреймворку Vue.js, який забезпечує створення інтерактивних та динамічних вебсторінок. Основними компонентами клієнтського інтерфейсу є HTML для структури вебсторінок, CSS для стилізації, та JavaScript для логіки і взаємодії з користувачем [36].

Для створення базової структури вебсторінок використовується HTML. Він дозволяє визначати елементи, такі як заголовки, параграфи, форми, кнопки, таблиці тощо. HTML є основою будь-якої вебсторінки, забезпечуючи організацію контенту та структуру документа.

За стилізацію вебсторінок відповідає CSS, надаючи їм привабливого вигляду. Він визначає кольори, шрифти, розміри, відступи та інші візуальні характеристики елементів на сторінці. CSS дозволяє робити вебсторінки естетично привабливими та забезпечувати гармонійний вигляд усіх компонентів.

Інтерактивність та динамічну взаємодію з користувачем забезпечує JavaScript. Використання бібліотеки Vue.js дозволяє створювати компоненти, які можуть повторно використовуватися, та забезпечує реактивність додатку, де зміни в даних автоматично відображаються на сторінці без необхідності перезавантаження. JavaScript разом з Vue.js надає додатку динамічні можливості, покращуючи користувацький досвід за рахунок інтерактивності та швидкої реакції на дії користувача.

Для розробки клієнтського інтерфейсу використовується контейнер з образом Node.js версії 21.6.2. Весь код клієнтської частини зберігається в локальному каталозі ./client, який монтується в контейнер під час запуску.

Під час запуску контейнера виконується команда npm i, яка встановлює всі необхідні залежності, зазначені у файлі package.json.

Після встановлення залежностей виконується команда npm run dev, яка запускає режим розробки. В цьому режимі включається локальний сервер для розробки, що забезпечує гаряче перезавантаження (hot-reloading), завдяки якому зміни в коді автоматично відображаються у браузері без необхідності ручного перезавантаження сторінки.

Клієнтський інтерфейс використовує переваги Vue.js для створення реактивних додатків. Однією з основних особливостей Vue.js є компонентний підхід. Код організований у вигляді компонентів, що дозволяє легко повторно використовувати їх у різних частинах додатку, що підвищує модульність і

підтримуваність коду, дозволяючи розробникам створювати ізольовані, незалежні частини додатку.

Також Vue.js забезпечує автоматичне оновлення інтерфейсу при зміні даних у моделі, це спрощує синхронізацію між користувацьким інтерфейсом та бізнес-логікою, знижуючи необхідність ручного оновлення елементів інтерфейсу. Автоматичне оновлення інтерфейсу гарантує, що користувачі завжди бачать актуальну інформацію, що покращує загальний досвід використання додатку.

Використання директив Vue.js дозволяє динамічно оновлювати DOM-елементи на основі змін в даних, забезпечуючи легке впровадження складних функціональних можливостей інтерфейсу, таких як умовне відображення, цикли та обробка подій, без необхідності писати зайвий код. Директиви Vue.js роблять код більш зрозумілим і спрощують його підтримку.

Клієнтський інтерфейс взаємодіє з бекендом, реалізованим на Django, через RESTful API. Запити до серверу відправляються за допомогою функції fetch, яка дозволяє легко працювати з HTTP-запитами, дозволяючи клієнтському інтерфейсу отримувати, відправляти та оновлювати дані, зберігаючи додаток динамічним та інтерактивним [37].

Завдяки використанню сучасних технологій та інструментів, клієнтський інтерфейс забезпечує високий рівень зручності для користувачів, швидку реакцію на їх дії та приємний користувацький досвід.

3.3.6 Налаштування вебсерверу для роботи із вебдодатком

Як зворотний проксі-сервер використовується вебсервер Nginx, забезпечуючи маршрутизацію запитів між різними сервісами додатку, а також безпеку та оптимізацію продуктивності [38]. Етапи роботи вебсерверу подано у додатку Н.

Nginx налаштований для роботи як через HTTP (порт 80), так і через HTTPS (порт 443). У конфігуруванні серверу, запити на порт 80 автоматично перенаправляються на HTTPS, що забезпечує захищене з'єднання. SSL-сертифікати

та ключі зберігаються у відповідних директоріях /etc/ssl/certificate.crt та /etc/ssl/private.key.

Статичні файли, такі як зображення, CSS та JavaScript, обслуговуються безпосередньо Nginx для зменшення навантаження на бекенд-сервіс. Це реалізується через директиви location /static та location /media, які вказують на відповідні директорії.

Nginx маршрутизує запити на різні сервіси додатку, забезпечуючи правильний розподіл трафіку між бекендом і фронтендом [39].

Запити до API, адміністративної панелі та документації (Swagger, Redoc) переспрямовуються на сервер Django за адресою http://web:8000. Ця директива гарантує, що всі запити, які відповідають вказаним шляхам, перенаправляються на бекенд-сервер, де їх обробляє Django.

Запити до основної частини додатку переспрямовуються на клієнтський інтерфейс, реалізований на Vue.js, за адресою http://client:3000. Дана директива забезпечує, що всі інші запити, які не відповідають попереднім умовам, перенаправляються на фронтенд-сервер, де їх обробляє Vue.js. Таким чином, Nginx ефективно розподіляє запити між різними сервісами додатку, забезпечуючи їх коректну обробку (Додаток П).

ВИСНОВКИ

У даній роботі розглянуто покращення та автоматизацію процесу розгортання вебресурсу в хмарному середовищі Amazon Web Services з використанням сучасних хмарних технологій та інструментів для безперервної інтеграції та доставки. Проаналізовано різні аспекти вибору технологій, налаштування інфраструктури, а також забезпечення безпеки та ефективності роботи додатку.

У процесі роботи було детально проаналізовано сучасні хмарні технології та їх здатність підтримувати і розгортати вебресурси. Це дало можливість виявити переваги та недоліки основних хмарних платформ, а також особливості їх використання для розгортання вебдодатків. Проведений порівняльний аналіз ринку хмарних технологій дозволив зробити обґрунтований вибір платформи для розгортання вебресурсу.

Платформа Amazon Web Services є найбільш підходящою для розгортання вебресурсу завдяки своїй надійності, масштабованості та широкому спектру доступних сервісів. AWS забезпечує гнучкість у виборі обчислювальних ресурсів, зберігання даних та інструментів для моніторингу та безпеки.

Розроблено та реалізовано структурну схему обчислювальної інфраструктури, що включає вибір необхідних обчислювальних потужностей та створення відповідних EC2 екземплярів на платформі AWS. Цей підхід забезпечив необхідну масштабованість та продуктивність для стабільної роботи вебдодатку. Налаштовано безпечний доступ до інфраструктури, що включало конфігурування правил безпеки та застосування відповідних механізмів захисту даних.

Особливу увагу приділено створенню та підключенню бази даних до обчислювальної інфраструктури, що дозволило забезпечити ефективне зберігання та обробку даних вебдодатку. Конфігурація процесу безперервної інтеграції та доставки (CI/CD) забезпечила автоматизацію розгортання оновлень, зменшивши ризик помилок та прискоривши процес розробки.

Створення та підключення SSL сертифікатів гарантувало безпечну передачу даних між користувачами та сервером, що є критично важливим для захисту

особистої інформації. Реалізовано вебдодаток з використанням фреймворку Django, налаштовано середовище розробки та інтеграцію з системою контролю версій Git.

Покращено продуктивність вебдодатку шляхом кешування даних та конфігурування вебсерверу для його оптимальної роботи. Реалізована архітектура вебдодатку дозволила інтегрувати різні компоненти, такі як застосунки для роботи з фільмами та системою аутентифікації, а також забезпечити ефективне управління даними.

Таким чином, виконано всі поставлені в роботі задачі, що дозволило покращити та автоматизувати процес розгортання вебресурсу в хмарному середовищі AWS та значно підвищило ефективність та надійність розробки та підтримки вебдодатку.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. «Використання CircleCI для розгортання програмного коду з GitHub в хмарному середовищі AWS» / А. О. Риженко, О. В. Войцеховська, Т. В. Лісничук // Матеріали міжнародної науково-технічної конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2024)». Вінниця, 2024 р. [Електронний ресурс]. — Режим доступу до ресурсу: https://conferences.vntu.edu.ua/index.php/mn/mn2024/paper/viewFile/21520/17814

2. «Russian missiles can't destroy the cloud»: Ukraine leader describes emergency migration. URL: https://www.theregister.com/2022/11/30/ukraine_cloud_migration/

3. How technology helped Ukraine resist during wartime вебсайт. URL: https://www.theregister.com/2022/11/30/ukraine_cloud_migration/

4. State of Containers and the Docker Ecosystem. URL: https://www.docker.com/state-of-containers

5. Kubernetes. URL: https://kubernetes.io/docs/concepts/ overview/

6. Amazon Web Services (AWS). "Overview of Amazon Web Services". URL: https://aws.amazon.com/what-is-aws/

7. Google Cloud "Overview of Google Cloud". URL: https://cloud.google.com/

8. Microsoft Azure. "What is Azure?". URL: https://azure.microsoft.com/en-us/overview/what-is-azure/

9. Топ 5 хмарних провайдерів: плюси та мінуси. URL: https://itedu.center/ua/blog/sysadministration/top-5-cloud-providers-advantages-and-disadvantages/

10. What is CI/CD?. URL: https://www.redhat.com/en/topics/devops/what-is-ci-cd

11. Amazon Elastic Compute Cloud Documentation. URL: https://docs.aws.amazon.com/ec2/

12. Amazon RDS and Aurora Documentation. URL: https://docs.aws.amazon.com/rds/

13. Amazon CloudWatch Documentation. URL: https://docs.aws.amazon.com/cloudwatch/

14. Chacon S., Straub B. Pro Git. 2nd Edition. Apress. 2014. 440 c.

15. Setting Up Your Environment. URL: https://docs.aws.amazon.com/AmazonECS/latest/developerguide/get-set-up-for-amazon-ecs.html

16. Launching an Instance. URL: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EC2_GetStarted.html

17. What is Amazon EC2? URL: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html

18. AWS Identity and Access Management (IAM). URL: https://docs.aws.amazon.com/iam/index.html

19. Creating an Amazon RDS DB Instance. URL: https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_CreateDBInstance.html

20. Configuration reference. URL: https://circleci.com/docs/configuration-reference/

21. Pexpect Documentation. URL: https://pexpect.readthedocs.io/en/stable/

22. Boto3 Documentation. URL: https://boto3.amazonaws.com/v1/documentation/api/latest/index.html

23. ZeroSSL. URL: https://zerossl.com/

24. The Leading Java and Kotlin IDE. URL: https://www.jetbrains.com/idea/

25. The Python IDE for data science and web development. URL: https://www.jetbrains.com/pycharm/

26. Git. URL: https://git-scm.com/

27. GitHub: Let's build from here. URL: https://github.com/

28. Docker Compose overview. URL: https://docs.docker.com/compose/

29. PostgreSQL: The World's Most Advanced Open Source Relational Database. URL: https://www.postgresql.org/

30. Redis - The Real-time Data Platform. URL: https://redis.io/

31. Django: The web framework for perfectionists with deadlines. URL: https://www.djangoproject.com/

32. Django REST framewor. URL: https://www.django-rest-framework.org/

33. Django Tutorial Part 3: Using models. URL: https://developer.mozilla .org/en-US/docs/Learn/Server-side/Django/Models

34. Serializers. URL: https://www.django-rest-framework.org/api-guide/serializers/

35. Class-based Views. URL: https://www.django-rest-framework.org/api-guide/views/

36. Vue.js - The Progressive JavaScript Framework. URL: https://vuejs.org/

37. Fetch API. URL: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API

38. NGINX: Advanced Load Balancer, Web Server, & Reverse Proxy. URL: https://www.f5.com/go/product/welcome-to-nginx

39. DeJonghe D. NGINX Cookbook: Advanced Recipes for High-Performance Load Balancing. 2nd Edition. O'Reilly Media. 2022. 203 c. ISBN: 978-1492078487.

40. Kholodkov V. Nginx Essentials. Packt Publishing. 2015. 150 c. ISBN: 978-1785289538.

ДОДАТОК А

Технічне завдання

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ
_____ проф., д.т.н. О.Д. Азаров
«6» травня 2024 р.

ТЕХНІЧНЕ ЗАВДАННЯ
Обчислювальна інфраструктура розгортання вебресурсу на хмарній платформі AWS
08-23.БДР.022.00.000 ТЗ

Керівник роботи: к.т.н., доц. каф. ОТ
_____Войцеховська О.В.
Виконав: студент гр. 1КІ-22мс
_____Риженко А.О.

Вінниця 2024

1 Підстава для виконання дипломної роботи (БДР):

— Тема дипломної роботи " Обчислювальна інфраструктура розгортання вебресурсу на хмарній платформі AWS" актуальна у зв'язку зі зростанням популярності хмарних технологій, що в свою чергу сприяє підвищенню попиту на кваліфікованих фахівців, які мають досвід розгортання та управління вебресурсами в хмарному середовищі. Розробка такого додатка дозволить ефективно працювати над розробкою, розгортанням та підтримкою вебдодатків у реальних умовах.

— наказ про затвердження теми бакалаврської дипломної роботи.

2 Мета і призначення БДР:

— метою роботи є покращення та автоматизація процесу розгортання вебресурсу в хмарному середовищі AWS шляхом реалізації хмарної обчислювальної інфраструктури із впровадженням безперервного розгортання та інтеграці інтеграції;

— призначення розробки — виконання бакалаврської дипломної роботи із подальшою пітримкою та розвитком.

3 Джерела розробки:

— відкрита для некомерційного використання база даних IMDB;

— відкрита для некомерційного використання API TMDB.

4 Технічні вимоги до виконання БДР:

— забезпечити конфігурацію безперервної інтеграції та доставки, щоб автоматизувати процеси розгортання та тестування вебдодатку;

— вибрати відповідні обчислювальні потужності та створити EC2 екземпляр, налаштувати безпечний до нього доступ та створити SSL сертифікати для безпеки даних користувачів, що використовують вебдодаток.

5 Етапи роботи та очікуванні результати приведено в Таблиці А.1.

Таблиця А.1 — Етапи БДР

№ з/п	Назва етапів дипломної роботи	Термін виконання	Очікувані результати
1	Постановка задачі роботи	12.03.24	Задачі дослідження
2	Огляд сучасних хмарних технологій та способів розгортання вебресурсів	13.03.24 — 17.03.24	Аналітичний огляд джерел
3	Аналіз сучасних хмарних технологій для розгортання та підтримки вебресурсів	18.03.24 — 23.03.24	Розділ 1
4	Розробка обчислювальної інфраструктури розгортання вебресурсу на AWS	24.03.24 — 09.04.24	Розділ 2
5	Практична реалізація вебдодатку та його розгортання в AWS	10.04.24 — 07.05.24	Розділ 3
6	Оформлення пояснювальної записки та ілюстративного матеріалу	08.05.24 — 08.06.24	ПЗ, додатки, презентація

6 Матеріали, що подаються до захисту БДР: пояснювальна записка, графічні та ілюстративні матеріали, протокол попереднього захисту роботи на кафедрі, відзив наукового керівника, рецензія опонента, анотації до українською та іноземною мовами, нормоконтроль про відповідність оформлення діючим вимогам.

7 Порядок контролю виконання та захисту БДР: виконання етапів графічної та розрахункової документації контролюється науковим керівником згідно зі встановленими термінами. Захист відбувається на засіданні Державної екзаменаційної комісії, затвердженою наказом ректора.

8 Вимоги до оформлювання та порядок виконання БДР.

8.1 При оформлювання БДР використовуються:

— ДСТУ 3008: 2015 «Звіти в сфері науки і техніки. Структура та правила оформлювання»;

— ДСТУ 8302: 2015 «Бібліографічні посилання. Загальні положення та правила складання»;

— міждержавний ГОСТ 2.104-2006 «Єдина система конструкторської документації. Основні написи»;

— методичні вказівки до виконання бакалаврських кваліфікаційних робіт зі спеціальності 123 «Комп'ютерна інженерія» (освітня програма «Комп'ютерна інженерія»). Кафедра обчислювальної техніки ВНТУ 2022;

— документами на які посилаються у вище вказаних.

8.2 Порядок виконання БДР викладено в «Положення про кваліфікаційні роботи на першому (бакалаврському) рівні вищої освіти СУЯ ВНТУ-03.02.02-П.001.01:21»

ДОДАТОК Б

Архітектура та процес розгортання вебдодатку

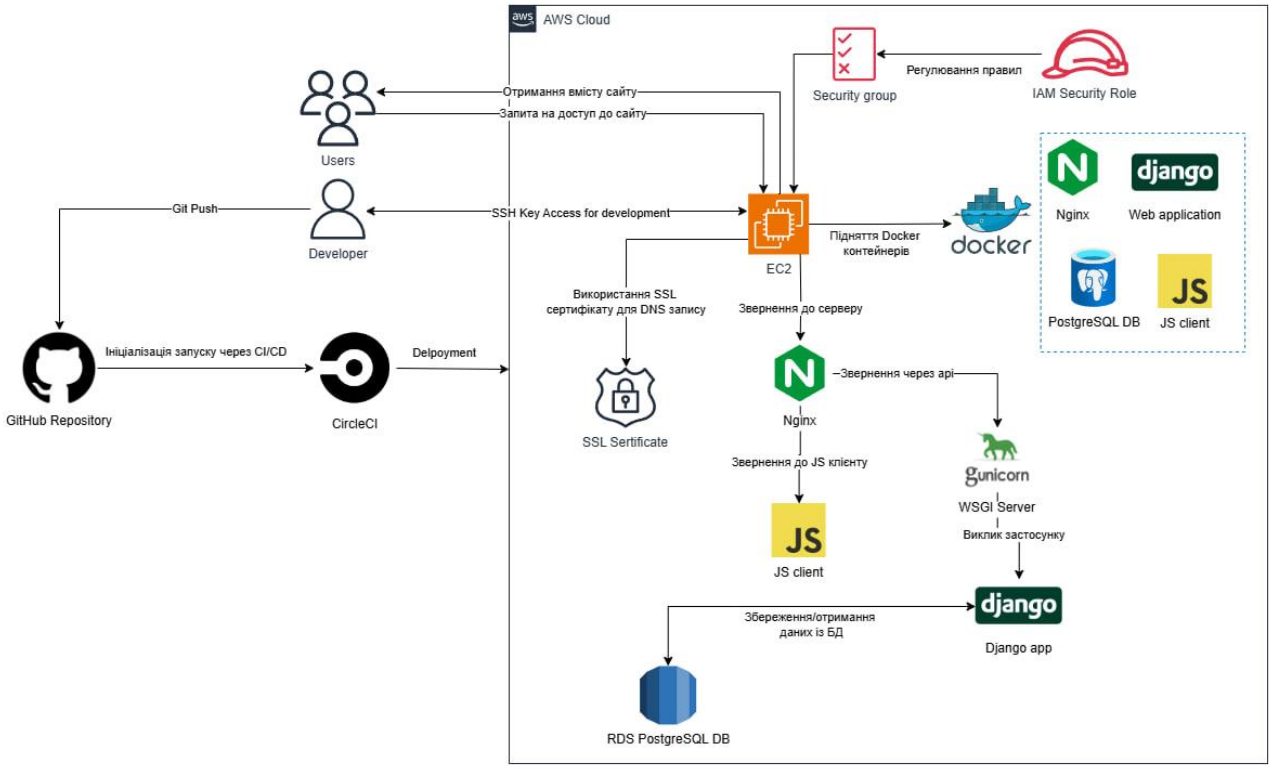

Рисунок Б.1 — Структурна схема архітектури та процесу розгортання вебдодатку

ДОДАТОК В

Розгортання та безпека AWS через CircleCI

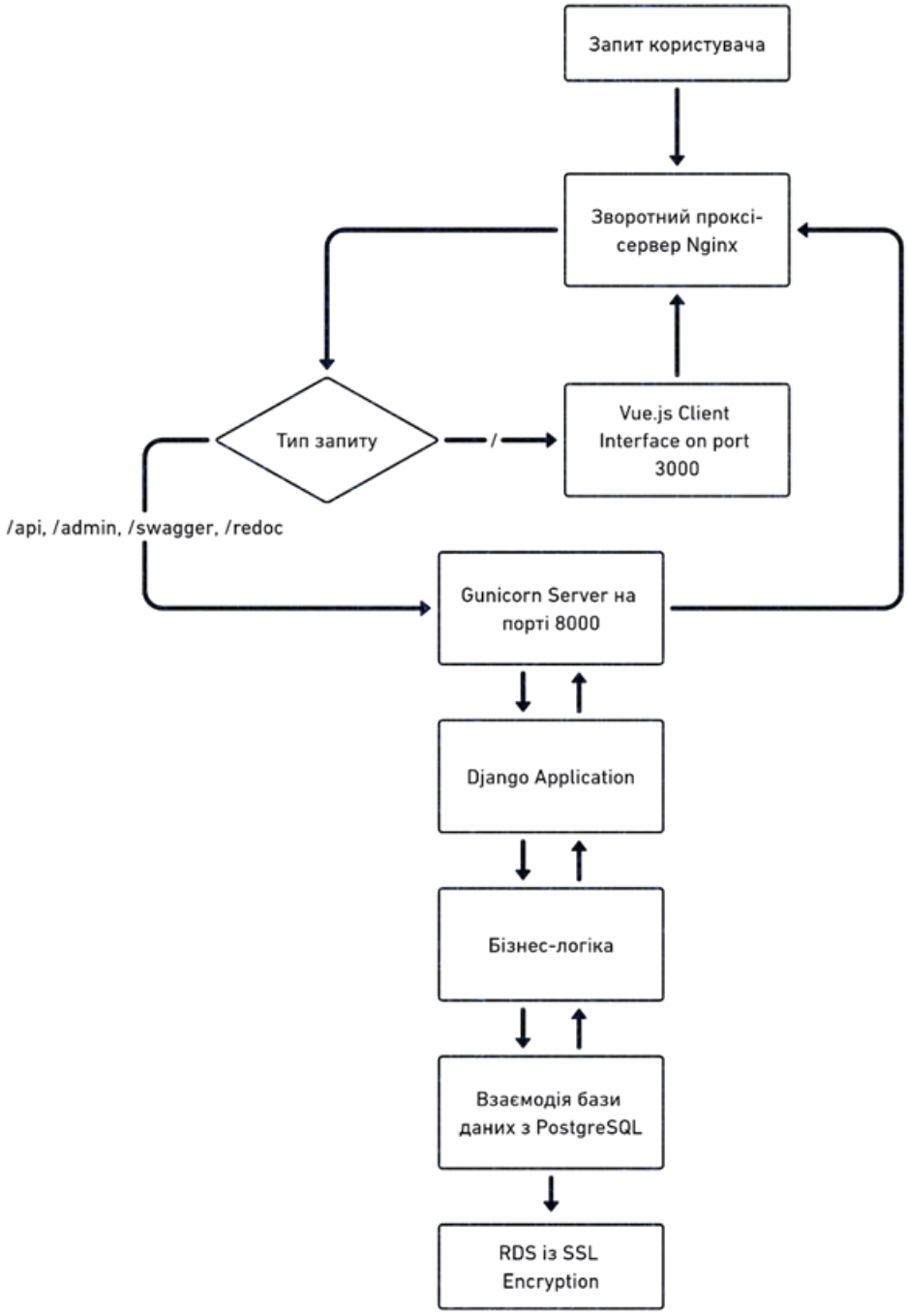

Рисунок В.1 — Етапи розгортання та безпеки на AWS за допомогою CircleCI

ДОДАТОК Г

Конфігураційний файл для CircleCI

```
version: 2.1
jobs:
  prospector:
    docker:
      - image: playmouse/movies_rating-deploy
    steps:
      - checkout
      - restore_cache:
          key: python-dev-requirements
      - run: pip install -r requirements-dev.txt
      - save_cache:
          key: python-dev-requirements-3.9
          paths:
            - /usr/local/lib/python3.9/site-packages
      - run: prospector apps -X
  tests:
    docker:
      - image: playmouse/movies_rating-deploy
    steps:
      - checkout
      - restore_cache:
          key: python-dev-requirements
      - run: pip install -r requirements-dev.txt
      - save_cache:
          key: python-dev-requirements-3.9
          paths:
            - /usr/local/lib/python3.9/site-packages
      - run: coverage run manage.py test
      - run: coverage report --fail-under=80
  deploy_dev:
    docker:
      - image: playmouse/movies_rating-deploy
    steps:
      - checkout
      - run: pip install boto3 pexpect
      - add_ssh_keys:
          fingerprints:
            - "SHA256:MYPMtB73kkNfZuLq8ryxgO2PgKXmSn2XMzq/FC9VLHU"
      - run: python deploy.py
workflows:
  check_and_deploy:
    jobs:
      - prospector
      - tests
      - deploy_dev:
          requires: ['prospector', 'tests']
          filters:
            branches:
              only:
                - master
```

ДОДАТОК Д

Лістинг коду deploy.py

```
import os
import sys
import logging
import pexpect
import boto3
logging.basicConfig(level=logging.INFO)
class Pexpect:
    def __init__(self, host, default_expect='ubuntu@', timeout=300):
        self.host = host
        self.default_expect = default_expect
        self.child = pexpect.spawn('ssh -o StrictHostKeyChecking=no
ubuntu@{}'.format(host), timeout=timeout, encoding='utf-8')
        self.child.logfile = sys.stdout
        self.child.expect(default_expect)
    def cmd(self, cmd, expect=None, timeout=None):
        if not expect:
            expect = self.default_expect
        self.child.sendline(cmd)
        return self.child.expect(expect, timeout=timeout)
    def send_break(self, expect=None):
        if not expect:
            expect = self.default_expect
        self.child.send('\003')
        self.child.expect(expect)
def find_instances():
    ec2 = boto3.resource(service_name='ec2', region_name='eu-north-1')
    running_instances = ec2.instances.filter(Filters=[{'Name': 'instance-
state-name', 'Values': ['running']}, {'Name': 'tag:proj', 'Values':
['diploma-project']},])
    return running_instances
def deploy_host(instance):
    deploy_host = instance.public_ip_address
    docker_compose_file = os.getenv('DOCKER_COMPOSE_FILE', 'docker-
compose.yml')
    docker_compose = 'docker compose -f {}'.format(docker_compose_file)
    expect_value = os.getenv('EXPECT_VALUE', 'ubuntu@')
    deploy_version = os.getenv('CIRCLE_SHA1')
    if not deploy_host or not deploy_version:
        raise ValueError('No env vars provided')
    expect = Pexpect(deploy_host, default_expect=expect_value)
    expect.cmd('pushd {}'.format('movies_rating'))
    expect.cmd('git fetch origin')
    expect.cmd('git reset --hard')
    expect.cmd('git checkout {}'.format(deploy_version))
    expect.cmd('{} stop'.format(docker_compose))
    expect.cmd('chmod +x manage.py')
    expect.cmd('{} up -d'.format(docker_compose))
```

```
    expect.cmd('exit', expect='logout')
def deploy():
    instances = find_instances()
    for instance in instances:
        deploy_host(instance)
if __name__ == '__main__':
    deploy()
```

ДОДАТОК Е

Розгортання вебдодатку через deploy.py

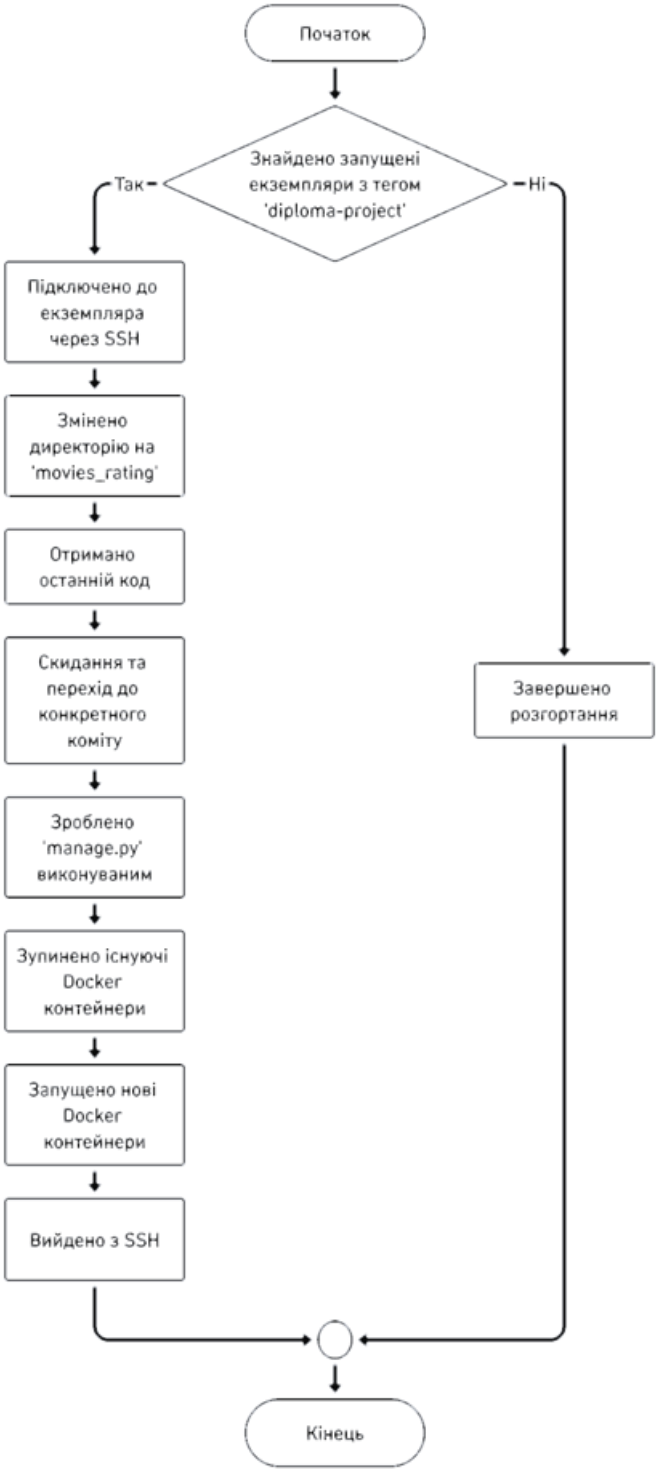

Рисунок Е.1 — Блок-схема алгоритму розгортання вебзастосунку через deploy.py

ДОДАТОК Ж

Лістинг файлу docker-compose.yml

```
version: "3.9"
services:
  db:
    image: postgres:16.1
    env_file:
      .env
    ports:
      - 5432:5432
    volumes:
      - pgdata:/var/lib/postgresql/data
redis:
image: redis
  web:
    build: .
    image: movies_rating_web
    depends_on:
      - db
      - redis
    volumes:
      - .:/src
      - static:/static
    env_file:
      - .env
    command: /bin/bash -c "pip install -r requirements.txt && invoke runit"
  client:
    image: node:21.6.2
    volumes:
      - ./client:/src
    command: /bin/bash -c "cd /src/; npm i; npm run dev"
    ports:
      - 3000:3000
  nginx:
    image: nginx:1.25.4
    networks:
      - default
      - nginx-proxy
    volumes:
      - static:/static:ro
      - ./config/nginx.conf:/etc/nginx/conf.d/default.conf:ro
    depends_on:
      - web
      - client
    ports:
      - "8000:80"
    environment:
      - VIRTUAL_HOST=diploma.ryzhenko.com
```

```
      - VIRTUAL_PORT=80
      - LETSENCRYPT_HOST=diploma.ryzhenko.com
      - LETSENCRYPT_EMAIL=admin@diploma.ryzhenko.com
networks:
  default:
  nginx-proxy:
    name: nginx-proxy
volumes:
  pgdata:
  static:
```

ДОДАТОК И

Сервіси та їх залежності у docker compose

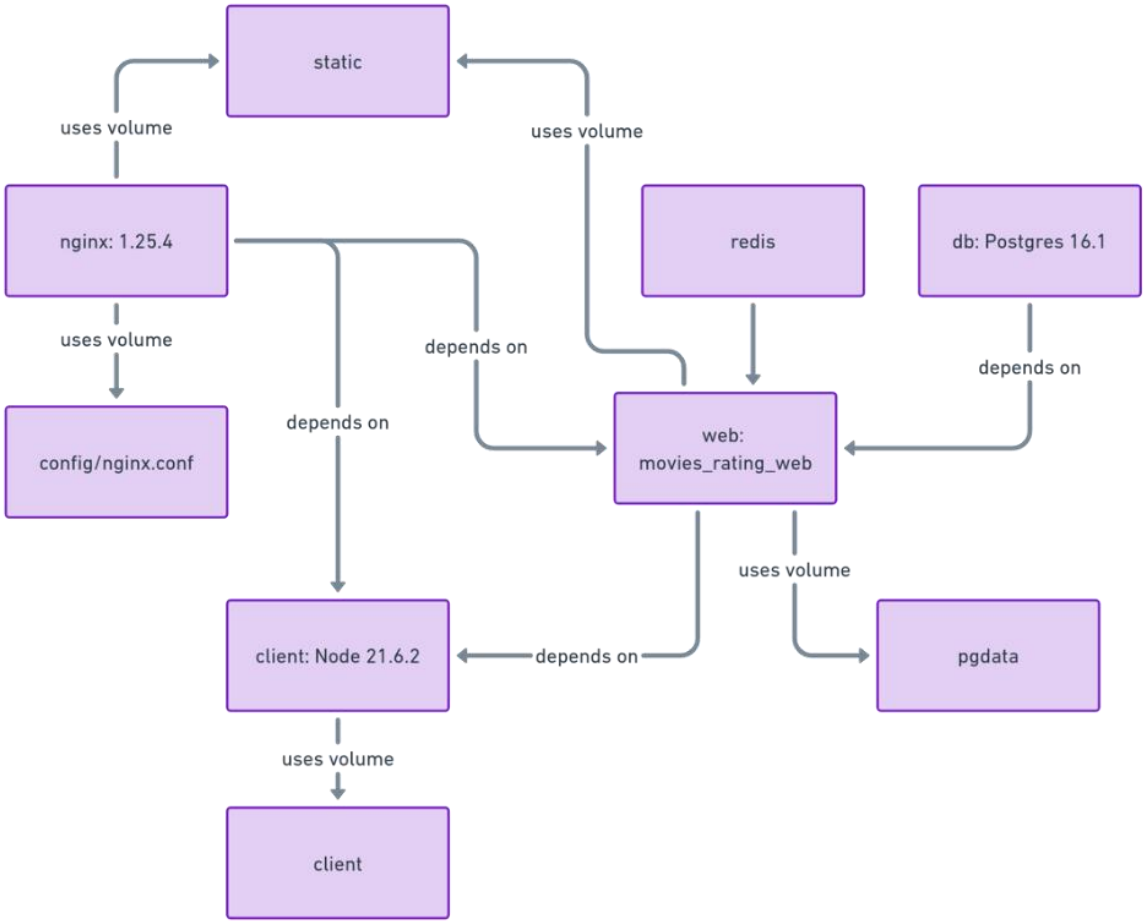

Рисунок И.1 — Структурна схема сервісів та їх залежностей у docker compose

ДОДАТОК К

Діаграма бази даних

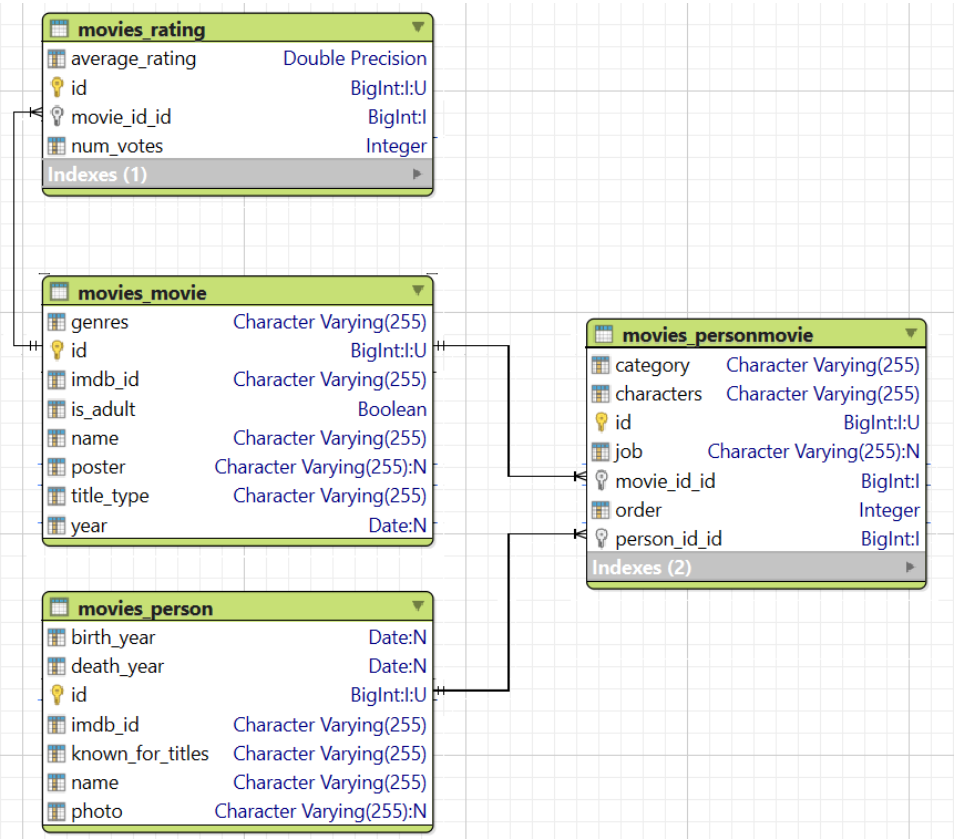

Рисунок К.1 — ER-діаграма бази даних

ДОДАТОК Л

Серіалізатори застосунків movies та authentication

Лістинг серіалізатора застосунку movies.

```
from rest_framework import serializers
from apps.movies.models import Movie, Person, PersonMovie, Rating
class MovieSerializer(serializers.ModelSerializer):
    directors = serializers.SerializerMethodField()
    average_rating = serializers.SerializerMethodField()
    class Meta:
        model = Movie
        fields = ['id', 'imdb_id', 'title_type', 'name', 'is_adult',
'year', 'genres', 'directors', 'average_rating',
                  'poster']
    def get_directors(self, obj):
        return
obj.personmovie_set.filter(category__iexact='director').values_list("person
_id__name", flat=True)
    def get_average_rating(self, obj):
        try:
            return obj.rating_set.values_list("average_rating",
flat=True).get()
        except Rating.DoesNotExist:
            return 0.0
class PersonSerializer(serializers.ModelSerializer):
    class Meta:
        model = Person
        fields = ['id', 'imdb_id', 'name', 'birth_year', 'death_year',
'known_for_titles', 'photo']
class PersonMovieSerializer(serializers.ModelSerializer):
    class Meta:
        model = PersonMovie
        fields = ['id', 'movie_id', 'person_id', 'order', 'category',
'job', 'characters']
    person_id = PersonSerializer()
class RatingsSerializer(serializers.ModelSerializer):
    class Meta:
        model = Rating
        fields = ['id', 'movie_id', 'average_rating', 'num_votes']
```

Лістинг серіалізатора застосунку authentication.

```
from django.contrib.auth import get_user_model
from rest_framework import serializers
User = get_user_model()
class UserSerializer(serializers.ModelSerializer):
    class Meta:
        model = User
        fields = ['email', 'username']
```

```
class RegistrationUserSerializer(serializers.ModelSerializer):
    class Meta:
        model = User
        fields = ['email', 'password', 'password2', 'username']
    password = serializers.CharField(write_only=True)
    password2 = serializers.CharField(write_only=True)

    def create(self, validated_data):

        user = User.objects.create_user(
            email=validated_data['email'],
            username=validated_data['username'],
            password=validated_data['password'])
        password = self.validated_data['password']
        password2 = self.validated_data['password2']
        if password2 != password:
            raise serializers.ValidationError({'detail': 'Password
mismatch'})
        elif len(password) < 6:
            raise serializers.ValidationError({'detail': 'Password is too
small'})
        username = self.validated_data['username']
        if len(username) < 6:
            raise serializers.ValidationError({'detail': 'Username is too
small'})
        user.save()
        return user
```

ДОДАТОК М

Представлення застосунків movies та authentication

Лістинг представлення застосунку movies.

```
from apps.movies.models import Movie, Person, PersonMovie, Rating
from apps.movies.serializers import MovieSerializer, PersonSerializer,
PersonMovieSerializer, RatingsSerializer
from rest_framework import permissions
from rest_framework.filters import OrderingFilter, SearchFilter
from rest_framework.generics import ListCreateAPIView,
RetrieveUpdateDestroyAPIView
class MovieListCreateView(ListCreateAPIView):
    permission_classes = [permissions.AllowAny]
    serializer_class = MovieSerializer
    filter_backends = [OrderingFilter, SearchFilter]
    search_fields = ['^name', '=imdb_id']
    ordering_fields = ['rating__average_rating', 'id']
    def get_queryset(self):
        queryset = Movie.objects.all()
        genres = self.request.query_params.get('genres')
        if genres:
            genres_list = genres.split(',')
            queryset = queryset.filter(genres__contains=genres_list)
            queryset = queryset.filter(genres__len=len(genres_list))
        if order_by := self.request.query_params.get('order_by'):
            queryset = queryset.order_by(order_by)
        return queryset
class MoviesRetrieveUpdateDestroyView(RetrieveUpdateDestroyAPIView):
    permission_classes = [permissions.AllowAny]
    serializer_class = MovieSerializer
    queryset = Movie.objects.all()
class PersonListCreateView(ListCreateAPIView):
    permission_classes = [permissions.AllowAny]
    serializer_class = PersonSerializer
    def get_queryset(self):
        queryset = Person.objects.all()
        if order_by := self.request.query_params.get('order_by'):
            queryset = queryset.order_by(order_by)
        return queryset
class PersonsRetrieveUpdateDestroyView(RetrieveUpdateDestroyAPIView):
    permission_classes = [permissions.AllowAny]
    serializer_class = PersonSerializer
    queryset = Person.objects.all()
class PersonMovieListCreateView(ListCreateAPIView):
    permission_classes = [permissions.AllowAny]
    serializer_class = PersonMovieSerializer
    search_fields = ['=movie_id__id']
    def get_queryset(self):
        queryset = PersonMovie.objects.all()
```

```
        if order_by := self.request.query_params.get('order_by'):
            queryset = queryset.order_by(order_by)
        return queryset
class PersonMoviesRetrieveUpdateDestroyView(RetrieveUpdateDestroyAPIView):
    permission_classes = [permissions.AllowAny]
    serializer_class = PersonMovieSerializer
    queryset = PersonMovie.objects.all()
class RatingsListCreateView(ListCreateAPIView):
    permission_classes = [permissions.AllowAny]
    serializer_class = RatingsSerializer
    search_fields = ['=movie_id__id']
    def get_queryset(self):
        queryset = Rating.objects.all()
        if order_by := self.request.query_params.get('order_by'):
            queryset = queryset.order_by(order_by)
        return queryset
class RatingsRetrieveUpdateDestroyView(RetrieveUpdateDestroyAPIView):
    permission_classes = [permissions.AllowAny]
    serializer_class = RatingsSerializer
    queryset = Rating.objects.all()
```

Лістинг представлення застосунку authentication.

```
from apps.authentication.serializer import UserSerializer,
RegistrationUserSerializer
from django.contrib.auth import get_user_model
from rest_framework import permissions
from rest_framework.generics import RetrieveAPIView, CreateAPIView
User = get_user_model()
class CurrentUserDetailsView(RetrieveAPIView):
    serializer_class = UserSerializer
    def get_object(self):
        return self.request.user
class UserRegistrationView(CreateAPIView):
    model = User
    permission_classes = [permissions.AllowAny]
    serializer_class = RegistrationUserSerializer
    queryset = User.objects.all()
```

ДОДАТОК Н

Лістинг файлу конфігурування Nginx для розготання

```
server {
    listen      80;
    server_name _;
    charset     utf-8;
    include     /etc/nginx/mime.types;
    root /;
    return 301 https://$host$request_uri;
}
server {
    listen               443 ssl;
    ssl_certificate      /etc/ssl/certificate.crt;
    ssl_certificate_key  /etc/ssl/private.key;
    server_name  _;
    location /media {
        alias /media;
    }
    location /static {
        alias /static;}
    location ~ ^/(api|admin|swagger|redoc) {
        proxy_pass       http://web:8000;
        proxy_redirect   http:// $scheme://;
        proxy_set_header Host $http_host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_connect_timeout 600;
        proxy_send_timeout 600;
        proxy_read_timeout 600;
        send_timeout 600;
        add_header 'Access-Control-Allow-Methods' 'GET, POST, PUT, PATCH,
DELETE, OPTIONS' always;
        add_header 'Access-Control-Allow-Headers'
'Accept,Authorization,Cache-Control,Content-Type,DNT,If-Modified-
Since,Keep-Alive,Origin,User-Agent,X-Requested-With' always;
        client_max_body_size 12M;
    }
    location / {
        proxy_pass       http://client:3000;
        proxy_redirect   http:// $scheme://;
        proxy_set_header Host $http_host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_connect_timeout 600;
        proxy_send_timeout 600;
        proxy_read_timeout 600;
        send_timeout 600;
        add_header 'Access-Control-Allow-Methods' 'GET, POST, PUT, PATCH,
DELETE, OPTIONS' always;
        add_header 'Access-Control-Allow-Headers'
'Accept,Authorization,Cache-Control,Content-Type,DNT,If-Modified-
Since,Keep-Alive,Origin,User-Agent,X-Requested-With' always;
        client_max_body_size 12M;
    }
}
```

ДОДАТОК П

Етапи роботи вебсерверу Nginx

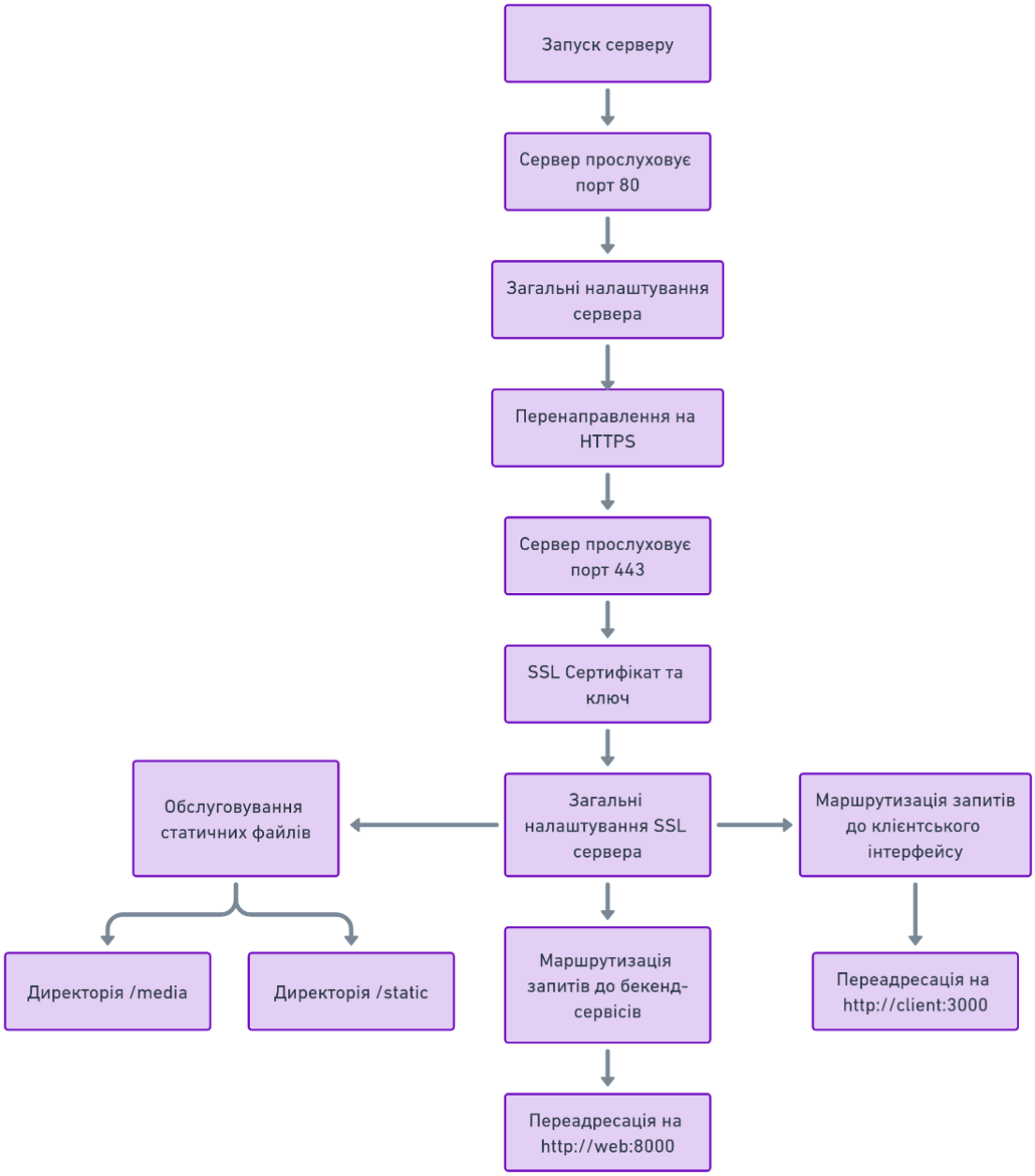

Рисунок П.1 — Етапи роботи вебсерверу Nginx

ДОДАТОК Р

ПРОТОКОЛ

ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Обчислювальна інфраструктура розгортання вебресурсу на хмарній платформі AWS

Тип роботи: бакалаврська дипломна робота

(БДР, МКР)

Підрозділ кафедра обчислювальної техніки

(кафедра, факультет)

Показники звіту подібності Unicheck

Оригінальність 98,4% Схожість 1,6%

Аналіз звіту подібності (відмітити потрібне):

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

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

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

Особа, відповідальна за перевірку _____ Захарченко С.М.

(підпис) (прізвище, ініціали)

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

Автор роботи _____ Риженко А. О.

(підпис) (прізвище, ініціали)

Керівник роботи _____ Войцеховська О. В.

(підпис) (прізвище, ініціали)