НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено:

Завідувач кафедри

_____ Наталія АУШЕВА

«___»_____2022 р.

Дипломна робота

на здобуття ступеня бакалавра

спеціальності 122 «Комп'ютерні науки »

освітня програма «Комп'ютерний моніторинг та геометричне моделювання процесів і систем»

на тему: «Автоматизація процесів розгортання та неперервної інтеграції клієнт-серверної системи»

Виконав:
студент IV курсу, групи ТР-82
Шкребко Данило Ігорович _____

Керівник:
Доцент
Гурін Артем Леонідович _____

Рецензент:

_____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2022

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший

Спеціальності 122 «Комп’ютерні науки»

Освітня програма «Комп’ютерний моніторинг та геометричне моделювання процесів і систем»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Наталія АУШЕВА
(підпис)
”___”_____2022р.

ЗАВДАННЯ
на дипломну роботу студенту

Шкребку Данилу Ігоровичу
(прізвище, ім’я, по батькові)

1. Тема роботи Автоматизація процесів розгортання та неперервної інтеграції клієнт-серверної системи

керівник роботи Гурін Артем Леонідович, старший викладач
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від ”08” червня 2022р.
№ _____

2. Строк подання студентом роботи ”10” червня 2022р.
3. Вихідні дані до роботи мова програмування JavaScript, TypeScript, середовище розробки Visual Studio Code
4.Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) проаналізувати аналогічні системи до розроблюваної, модернізувати та розробити зручну систему, яка відповідає сучасним вимогам

5. Перелік ілюстративного матеріалу

Наявні програмні комплекси, інфраструктура програмної системи, демонстрація роботи клієнт-серверної системи, сценарій автоматизації процесів інтеграції та розгортання системи, інтерфейс користувача

7. Дата видачі завдання ”17” жовтня 2021 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Затвердження теми роботи	25.05.2022	
2.	Вивчення та аналіз задачі	02.05.2022 - 03.05.2022	
3.	Розробка архітектури та загальної структури системи	04.05.2022 - 05.05.2022	
4.	Розробка структур окремих підсистем	06.05.2022 - 08.05.2022	
5.	Програмна реалізація системи	09.05.2022 - 15.05.2022	
6.	Оформлення пояснювальної записки	16.05.2022 - 22.05.2022	
7.	Захист програмного продукту	23.05.2022 - 27.05.2022	
8.	Передзахист	23.05.2022 - 27.05.2022	
9.	Захист	20.06.2022 - 30.06.2022	

Студент _____ Шкребко Д.І.
(підпис) (прізвище та ініціали,)

Керівник роботи _____ Гурін А.Л.
(підпис) (прізвище та ініціали,)

АНОТАЦІЯ

Об’єктом розгляду в даній роботі є клієнт-серверна системи та інфраструктура для розгортання та неперервної інтеграції коду. Предмет роботи – створення клієнт-серверної системи та інфраструктури для автоматизації процесів розгортання та неперервної інтеграції програмного продукту.

Метою дипломної роботи є створення клієнт-серверної системи та інфраструктури розгортання та неперервної інтеграції коду для забезпечення інструменту для ефективної роботи розробників програмного продукту. Важливою частиною роботи є налагодження інфраструктури та забезпечення автоматизації процесів.

Для досягнення мети були вирішені наступні задачі:

1. Розроблено клієнт-серверну систему.
2. Проведено аналіз рішення.
3. Спроектовано інфраструктуру .
4. Впроваджено методи для розгортання та неперервної інтеграції коду.
5. Автоматизовано процеси.
6. Розроблено та протестовано систему.

Ключові слова: клієнт-сервер, неперервна інтеграція, розгортання, веб-портал, JavaScript, Node.js, TypeScript.

Записка містить 93 сторінок, 27 рисунків та 18 посилань.

ANNOTATION

The subject of this paper is the client-server system and infrastructure for the deployment and continuous integration of code. The subject of work is the creation of a client-server system and infrastructure to automate the processes of deployment and continuous integration of the software product.

The aim of the thesis is to create a client-server system and infrastructure deployment and continuous code integration to provide a tool for effective work of software developers. An important part of the work is to establish the infrastructure and automate processes.

To achieve this goal, the following tasks were solved:

1. Developed client-server system.
2. The analysis of the decision is carried out.
3. Infrastructure designed.
4. Methods for deployment and continuous code integration have been introduced.
5. Automated processes.
6. Developed and tested the system.

Keywords: client-server, continuous integration, deployment, web portal, JavaScript, Node.js, Typescript.

The note contains 93 pages, 27 images and 18 references.

ВСТУП

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API – Application Program Interface;

UI – User interface;

JSON – JavaScript Object Notation;

AWS – Amazon Web Services;

CDK – Cloud Development Kit ;

CLI – Command Language Interface;

S3 – Simple Storage Service;

Bucket - об’єкт сховища Simple Storage Service;

CI/CD – Continuous Integration/Continuous Delivery;

СУБД – Система управління базами даних;

HTTP – HyperText Transfer Protocol;

IaC – Infrastructure as Code.

ВСТУП

Процеси розробки та випуску програмного забезпечення можуть бути значно ускладненими, особливо з урахуванням того, що програми, групи та інфраструктура розгортання на базі власних ресурсів ускладнюються з кожним днем. Нерідко проблеми стають більш вираженими зі зростанням проектів. Для швидкої та послідовної розробки, тестування та випуску програмного забезпечення було виведено три взаємопов'язані, але різні стратегії управління та автоматизації цих процесів.

Неперервна інтеграція спрямована на інтеграцію роботи окремих розробників до основного репозиторію (в загальному випадку – кілька разів на день). Підхід такого типу дозволяє заздалегідь виявити можливі помилки та прискорити спільну розробку. Неперервна доставка пом'якшує процеси розгортання або випуску, автоматизацію етапів, необхідних для розгортання побудови застосунка (можливості безпечно випускати код у будь-який час). Неперервне розгортання продовжує цю лінію, автоматично розгортаючи кожен новий або змінений фрагмент коду.

Неперервна інтеграція (Continuous Integration, CI) та неперервне постачання (Continuous Delivery, CD) є культурою, набором принципів і практик, які дозволяють розробникам частіше і надійніше розгортати зміни програмного забезпечення.

CI/CD – це одна з DevOps-практик [18]. Вона також відноситься і до agile-практиків: автоматизація розгортання дозволяє розробникам зосередитися на реалізації бізнес-вимог, якості коду та безпеки.

Неперервна інтеграція — це методологія розробки та набір практик, за яких код вносяться невеликі зміни з частими комітами. І оскільки більшість сучасних додатків розробляються з використанням різних платформ та інструментів, то виникає потреба в механізмі інтеграції та тестуванні змін, що вносяться.

З технічної точки зору, мета CI - забезпечити послідовний та автоматизований спосіб складання, пакування та тестування додатків. При налагодженому процесі неперервної інтеграції розробники з більшою ймовірністю робитимуть часті комміти, що, у свою чергу, сприятиме покращенню комунікації та підвищенню якості програмного забезпечення.

Неперервне постачання починається там, де закінчується неперервна інтеграція. Вона автоматизує розгортання додатків у різні оточення: більшість розробників працюють як із продакшн-оточенням, так і з середовищами розробки та тестування.

Інструменти CI/CD допомагають настроювати специфічні параметри оточення, які конфігуруються під час розгортання. А також CI/CD-автоматизація виконує необхідні запити до веб-серверів, баз даних та інших сервісів, які можуть потребувати перезапуску або виконання будь-яких додаткових дій при розгортанні програми.

Неперервна інтеграція та неперервне постачання потребують неперервного тестування, оскільки кінцева мета — розробка якісних додатків. Неперервне тестування часто реалізується у вигляді набору різних автоматизованих тестів (регресійних, продуктивності та інших), що виконуються в CI/CD-конвеєрі.

Зріла практика CI/CD дозволяє реалізувати неперервне розгортання: при успішному проходженні коду через CI/CD-конвеєр, збірки автоматично розгортаються у продакшн-оточенні. Команди, які практикують неперервне постачання, можуть дозволити собі щоденне або навіть щогодинне розгортання. Хоча тут варто відзначити, що неперервне постачання підходить не для всіх додатків.

Для розробки інфраструктури для клієнт-серверної системи було використано мову програмування TypeScript для процесів розгортання та неперервної інтеграції рішення, з мовами розмітки та стилів HTML та CSS для створення інтерактивного користувацького інтерфейсу та платформу Node.js з якої складається серверна частина. В якості середовища програмування слугує Visual Studio Code.

1 ПОСТАНОВКА ЗАДАЧІ РОЗРОБКИ КЛІЄНТ-СЕРВЕРНОЇ СИСТЕМИ ТА ІНФРАСТРУКТУРИ ДЛЯ АВТОМАТИЗАЦІЇ ПРОЦЕСІВ РОЗГОРТАННЯ ТА НЕПЕРЕРВНОЇ ІНТЕГРАЦІЇ

Створення клієнт-серверної реалізації веб-порталу[12] книжної полиці та інфраструктури для розробників, що забезпечить ефективність процесів розгортання та неперервної інтеграції коду клієнт-серверної системи.

1.1 Задачі поставлені перед системою

До задач розроблюваної інфраструктури були такі наступні вимоги:

- Розроблення клієнт-серверної архітектури, для аналізу та обробки вхідних даних користувача.
- Маніфестація та створення інфраструктури для системи, користуючись підходом для управління та опису інфраструктури;
- Впровадження засобів для неперервної інтеграції коду до системи;
- Автоматизація процесів для оптимізації циклу розробки програмного продукту.

1.2 Компоненти програмної системи

Для виконання поставлених перед системою задач передбачається розробка наступних компонентів та підсистем:

- Клієнт-серверне рішення – веб-портал.
- Інфраструктура для клієнт-серверної системи.
- Автоматизації процесів розгортання та неперервної інтеграції коду до системи.

Система з користувацьким інтерфейсом – книжний веб-портал, який надає можливість обробляти вхідні дані, вносити і зчитувати інформацію з бази даних у режимі реального часу, створювати, оновлювати та видаляти окремі об’єкти та їх складові.

Інфраструктура для клієнт-серверної системи спроектована, користуючись підходом для управління та опису інфраструктури IaC.

Сценарій для розгортання та неперервної інтеграції коду до системи, а також автоматизація процесів прописан за допомогою платформи для неперервної інтеграції та неперервної доставки коду GitHub Actions.

1.3 Користувачі системи

Потенційними користувачами розробленої під час виконання даної роботи системи є розробники та UI дизайнери клієнт-серверних рішень, а також кінцевий користувач, що зможе мати доступ до найновішої випущеної версії книжної веб полиці.

Користь для розробників:

- Зручність процесу інтеграції нового коду до системи, за рахунок автоматизації.
- Швидкість розгортання нових функціональних рішень.
- Ефективність роботи через миттєве відображення змін на сервері.

Користь для звичайних користувачів:

- Зручний веб-портал для любителів літератури.
- Змога швидко отримати доступ до останньої версії продукту.
- Можливість своєчасного зворотнього зв’язку від розробників при виявленні проблем з системою.

2 ОГЛЯД ТА АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ДЛЯ АВТОМАТИЗАЦІЇ ПРОЦЕСІВ РОЗГОРТАННЯ ТА НЕПЕРЕРВНОЇ ІНТЕГРАЦІЇ КЛІЄНТ-СЕРВЕРНОЇ СИСТЕМИ

2.1 Поняття Хмарних Технологій та їх види

Хмарні технології – технології розподіленої обробки цифрових даних, за допомогою яких комп'ютерні ресурси надаються інтернет-користувачеві як онлайн-сервіс. Програми запускаються та видають результати роботи у вікні web-браузера на локальному ПК. При цьому всі необхідні для роботи програми та їх дані знаходяться на віддаленому інтернет-сервері та тимчасово кешуються на стороні клієнта: на ПК та ін.

Перевага технології в тому, що користувач має доступ до власних даних, але не повинен піклуватися про інфраструктуру, операційну систему та програмне забезпечення, з якими він працює. Слово «хмара» — це метафора, що уособлює складну інфраструктуру, приховує всі технічні деталі [5].

Є такі категорії хмарних технологій:

Публічна хмара – одночасний доступ багатьох користувачів до IT-інфраструктури. Але можливості керувати та обслуговувати цю хмару у користувачів немає, вся відповідальність покладена на її власника. Абонентом запропонованих сервісів може стати будь-яка компанія чи приватна особа.

Приватна хмара – IT-інфраструктура, яку контролює та експлуатує лише один абонент у власних інтересах. Інфраструктура управління приватною хмарою може розміщуватися або в приміщеннях користувача, або у зовнішнього оператора або частково у користувача та оператора.

Гібридна хмара - це IT-інфраструктура, в якій об'єднані найкращі якості публічної та приватної хмари. Така композиція є унікальними об'єктами, пов'язаними між собою стандартизованими або власними технологіями, які дозволяють переносити дані або програми між компонентами.

Існує кілька рівнів хмарних обчислень:

Низький рівень "Інфраструктура як послуга" (IaaS, infrastructure as a service). Користувачі отримують базові обчислювальні ресурси: процесори та пристрої для зберігання інформації – і використовують їх для створення власних операційних систем та додатків. Споживач не управляє базовою інфраструктурою хмари, але має контроль над операційними системами, системами зберігання, розгорнутими програмами. Можливий обмежений контроль вибору мережевих компонентів (наприклад, хост із мережевими екранами).

Наступний рівень "Платформа як послуга" (PaaS, platform as a service). Користувачі мають можливість встановлювати власні програми на платформі, що надається провайдером послуги. Користувач не керує базовою інфраструктурою хмари: мережами, серверами, операційними системами та системами зберігання даних, але має контроль над розгорнутими програмами та деякими параметрами конфігурації середовища хостингу.

Вищий рівень хмарних обчислень "Програмне забезпечення як послуга" (SaaS, software as a service). У «хмарі» зберігаються як дані, а й пов'язані із нею програми, а користувачеві до роботи потрібен лише веб-браузер. Споживач користується програмами провайдера, який працює в хмарній інфраструктурі. При цьому користувач не управляє базовою інфраструктурою хмари - мережами, серверами, операційними системами, системами зберігання, а також індивідуальними налаштуваннями програм за винятком деяких налаштувань конфігурації програми.

2.2 Існуючі рішення хмарних платформ

На даний момент існує багато платформ, що використовують хмарні технології, забезпечують функціонал, необхідний для автоматизації процесів розгортання та неперервної інтеграції клієнт-серверної системи.

Хмарні платформи – платформи для хмарних обчислень, що є готовим програмним та апаратним забезпеченням, що здається в оренду через Інтернет для розгортання, розробки, тестування своїх додатків.

Ідея створення хмарних платформ була сформована ще за часів появи Інтернету, проте перша хмарна платформа з'явилася лише у 2006 році під назвою «Amazon Web Services» (хоча надавати доступ до обчислювальних ресурсів через Інтернет компанія почала ще 2002 року). Саме в 2006 році Amazon запропонувала понад 50 різних послуг у 14 географічних регіонах.

Наступною великою платформою стала система Microsoft Azure, що з'явилася в 2010 (зараз посідає друге почесне місце за кількістю користувачів та послуг після Amazon). У 2011 році у світі був представлений третій великий гравець на ринку хмарних обчислень - Google Cloud Platform. Через рік після цього почалося активне створення нових платформ та сервісів від різних компаній, і як наслідок – зниження цін на оренду платформ. Були створені такі сервіси як: i-Teco OpenStack Cloud, Cloud OS від Azure, Jelastic для PHP та Java додатків та інші.

Сьогодні хмарні платформи, як ніколи, користуються популярністю. Основними перевагами використання хмарних платформ є швидкість створення нових додатків, гнучкість та масштабованість системи.

При використанні хмарних платформ зі сторони клієнтів необхідна тільки організація робочого місця (комп'ютер/ноутбук/планшет) причому незалежно від розташування. Єдиною умовою є наявність виходу до мережі Інтернет. Клієнт (користувач хмарних платформ) займається лише адмініструванням власних програм, ПЗ.

За всі інші аспекти, а це: адміністрування баз даних, віртуалізація, мережеве та серверне обладнання, адміністрування операційних систем, обслуговування інфраструктури дата-центру – відповідає постачальник хмарної платформи, який часто називають хмарним провайдером. Крім того, крім платформи для запуску додатків, хмарна платформа пропонує ще оренду хмарного сховища і підтримку таких технологій як машинне навчання, штучний інтелект, Big Data.

Для забезпечення безпеки найчастіше використовується окремий VPN та адреса мережі, власний брандмауер, гнучкі налаштування DNS.

Ціна за оренду хмарної платформи зазвичай складається з плати за обчислювальні потужності (включаючи обслуговування обладнання), вартості ліцензії, використовуваного програмного забезпечення/операційної системи/систем віртуалізації та плати за послуги хмарного провайдера як постачальника хмарної платформи.

Хмарні платформи використовуються компаніями в багатьох сферах, наприклад для систем розумного страхування, хмарної телеметрії, usability-тестування власного програмного забезпечення або сайтів, систем онлайн-освіти, бронювання готелів або оренди житла, системи ідентифікації в онлайн-банках і т.д.

2.2.1 Amazon Web Services

У 2006 році Amazon Web Services (AWS) [16] почала пропонувати послуги IT-інфраструктури підприємствам через систему веб-сервісів, які зараз широко відомі як хмарні обчислення. Однією з ключових переваг хмарних обчислень є можливість замінити початкові капітальні витрати на інфраструктуру низькими змінними витратами, які масштабуються відповідно до потреб вашого бізнесу. Завдяки хмарним обчисленням компаніям більше не потрібно планувати та закуповувати сервери та іншу IT-інфраструктуру на тижні чи місяці наперед. Замість цього вони можуть миттєво розгорнути сотні чи тисячі серверів за лічені хвилини та швидше отримати результати.

Сьогодні Amazon Web Services забезпечує високонадійну, масштабовану та недорогу інфраструктурну платформу.

AWS надає величезну глобальну хмарну інфраструктуру, яка дозволяє швидко вводити інновації, експериментувати та повторювати. Замість того, щоб чекати тижнями чи місяцями на апаратне забезпечення, можна миттєво розгорнути нові програми, миттєво збільшити масштаби в міру зростання робочого навантаження та миттєво зменшити відповідно до потреб. Незалежно від того, чи потрібен один віртуальний чи тисячі, чи потрібні вони на кілька годин чи цілодобово, все одно сплата лише за те, що використовується.

Платформа AWS не залежить від мови та операційної системи. Обрана платформа розробки або модель програмування є тою, що найбільше підходить для поставленої задачі. Є можливість обрати, які послуги ви використовуєте, одну чи кілька, вибрати спосіб їх використання. Ця гнучкість дозволяє зосередитися на інноваціях, а не на інфраструктурі.

AWS — це безпечна, довговічна технологічна платформа з визнаними в галузі сертифікатами та аудитами. Послуги та центри обробки даних мають кілька рівнів оперативної та фізичної безпеки для забезпечення цілісності та безпеки даних.

Ключова інфраструктурна послуга – служба оренди віртуальних серверів EC2. Передплатникам надаються віртуальні машини, що працюють на гіпервізорі Xen та

власному варіанті KVM, доступний вибір різних за обчислювальною потужністю машин, а також машин з доступом до спеціалізованого обладнання (відеокарт для GPGPU, програмованих вентильних матриць). EC2 тісно інтегрована з іншими інфраструктурними послугами хмари, насамперед — Elastic File System, що забезпечує файлову систему, що приєднується до віртуальних машин, Elastic Block Store (EBS), що приєднують до віртуальних машин томи як блокові пристрої, і S3, забезпечує сховище великого обсягу.

Серед інших інфраструктурних послуг - Route 53 (керований хмарний DNS), VPC (засіб створення в рамках ізольованої VPN групи хмарних сервісів), Elastic Load Balancing (балансувальник трафіку між віртуальними машинами), служба Glacier забезпечує довготривале («холодне») зберігання даних, а CloudFront – мережа доставки контенту. Низка послуг забезпечують автоматизоване управління інфраструктурою, що розміщується в AWS, серед таких – CloudFormation, OpsWorks, CloudWatch.

У хмарі широко представлені хмарні СУБД різних категорій. Серед доступних NoSQL-систем - Amazon SimpleDB, DynamoDB, резидентна СУБД ElastiCache, графова СУБД Neptune. У рамках послуг Amazon Relational Database Service (RDS) передплатники можуть розгортати хмарні бази під управлінням популярних реляційних СУБД - MySQL, Oracle Database, Microsoft SQL Server і PostgreSQL, при цьому також доступна реляційна СУБД Amazon Aurora, що масштабується, сумісна з MySQL і PostgreSQL. Аналітична масово-паралельна реляційна СУБД ParAccel, адаптована для хмарної інфраструктури, надається під торговою маркою Amazon Redshift.

Служба Amazon Athena дозволяє проводити аналіз даних в Amazon S3, використовуючи стандартний SQL (із застосуванням Presto), причому для її роботи не потрібно виділених обчислювальних потужностей (використовується стратегія безсерверних обчислень), а передплатники оплачують лише кількість мегабайтів, оброблених у рамках виконаних запитів. Служба Elastic MapReduce дозволяє передплатникам створювати Hadoop-кластери, оснащені відповідною екосистемою продуктів класу великих даних (в тому числі Spark, Hive, HBase, Presto). Інструмент

QuickSight надає передплатникам можливості візуального аналізу даних, розміщених у службах AWS. Amazon Elasticsearch Service забезпечує хмарний доступ до стеку з пошукової системи Elasticsearch та Kibana. Служба Amazon Machine Learning забезпечує передплатникам доступ до інструментарію машинного навчання.

Серед послуг класу зв'язуючого програмного забезпечення є брокер повідомлень Amazon Kinesis (близький по можливостях Apache Kafka), служба черг SQS та служба повідомлень SNS.

Засіб розгортання програм за шаблоном «функція як послуга» за допомогою стратегії безсерверних обчислень — AWS Lambda; Elastic Kubernetes Service надає можливість розгортання програм у контейнерній інфраструктурі під керуванням Kubernetes.

2.2.2 Google Cloud Platform

(GCP) що пропонує Google [14], — це набір послуг хмарних обчислень, які працюють на тій самій інфраструктурі, що Google використовує внутрішньо для своїх продуктів для кінцевих користувачів, таких як Пошук Google, Gmail, Google Drive і YouTube. Поряд із набором інструментів керування, він надає низку модульних хмарних сервісів, включаючи обчислення, зберігання даних, аналітику даних та машинне навчання.

Google Cloud Platform надає інфраструктуру як послугу, платформу як послугу та безсерверні обчислювальні середовища.

У квітні 2008 року Google анонсувала App Engine, платформу для розробки та розміщення веб-додатків у центрах обробки даних, якими керує Google, яка стала першою службою хмарних обчислень від компанії. Служба стала загальнодоступною в листопаді 2011 року. З моменту анонсу App Engine Google додала до платформи кілька хмарних сервісів.

Google Cloud Platform є частиною Google Cloud, яка включає загальнодоступну хмарну інфраструктуру Google Cloud Platform, а також Google Workspace (G Suite),

корпоративні версії ОС Android і Chrome, а також інтерфейси програмування програм (API) для комп’ютера. навчання та картографічні послуги підприємства.

Серед послуг що надає Google Cloud Platform:

App Engine – платформа як послуга для хостингу програм.

BigQuery - інфраструктура як послуга, що масштабується аналітика баз даних.

BigTable - інфраструктура як послуга, що масштабується базою даних NoSQL.

Cloud AutoML – набір продуктів для машинного навчання, які дозволяє розробникам з обмеженим досвідом роботи в галузі машинного навчання використовувати технології навчання та створення нейронних мереж.

Cloud Datastore - документоорієнтована хмарна база даних.

Cloud Pub/Sub — послуга для публікації та підписки на потоки даних та повідомлення. Програми можуть обмінюватися даними через публікацію/підписку, без прямого обміну повідомленнями.

Compute Engine – інфраструктура як послуга, що надає віртуальні машини.

Kubernetes Engine — система автоматичного розгортання, масштабування та керування програмами в контейнерах для Kubernetes.

Google Genomics - аналіз геномів у хмарі

Google Video Intelligence

Cloud Vision

Storage — інфраструктура як послуга, надає онлайн REST-доступ до файлів та змісту сховищ даних.

2.2.3 Microsoft Azure

Раніше відома як Windows Azure [15], є загальнодоступною платформою хмарних обчислень Microsoft. Вона надає цілий ряд хмарних послуг, включаючи обчислення, аналітику, зберігання та мережу. Користувачі можуть обирати з ряду цих

служб для розробки та масштабування нових програм або запуску існуючих програм у загальнодоступній хмарі.

Платформа Azure має на меті допомогти підприємствам справлятися з проблемами та досягати своїх організаційних цілей. Вона пропонує інструменти, які підтримують усі галузі, включаючи електронну комерцію, фінанси та різноманітні компанії зі списку Fortune 500, і сумісний з технологіями з відкритим кодом. Це надає користувачам гнучкість у використанні бажаних інструментів і технологій. Крім того, Azure пропонує 4 різні форми хмарних обчислень: інфраструктура як послуга (IaaS), платформа як послуга (PaaS), програмне забезпечення як послуга (SaaS) і безсерверна платформа.

Оскільки Microsoft Azure пропонує численну кількість послуг, варіанти її використання надзвичайно різноманітні. Запуск віртуальних машин або контейнерів у хмарі є одним із найпопулярніших видів використання Microsoft Azure. Ці обчислювальні ресурси можуть розміщувати компоненти інфраструктури, такі як сервери системи доменних імен (DNS); Служби Windows Server, такі як інформаційні служби Інтернету (IIS); або сторонні програми. Microsoft також підтримує використання сторонніх операційних систем, таких як Linux.

Azure також зазвичай використовується як платформа для розміщення баз даних у хмарі. Microsoft пропонує безсерверні реляційні бази даних, такі як Azure SQL, і нереляційні бази даних, такі як NoSQL.

Крім того, платформа часто використовується для резервного копіювання та аварійного відновлення. Багато організацій використовують сховище Azure як архів, щоб задовольнити свої вимоги щодо довгострокового збереження даних.

3 ЗАСОБИ РОЗРОБКИ ПРОГРАМНОГО ПРОДУКТУ

При створенні інфраструктури проекту клієнт-серверної системи потрібно завжди зважати на досвід розробників під час роботи над програмним продуктом, тому необхідно уявити кожну потребу спеціаліста при розробці та продумати кожне функціональне рішення, інфраструктуру та досвід роботи з нею, адаптивність для різних систем та середовищ розробки.

Клієнт-серверне рішення використовує стандартизовану мову розмітки документів для перегляду веб-сторінок у браузері HTML, формальну мова опису зовнішнього вигляду веб-сторінки CSS, мову сценаріїв для надання інтерактивності веб-сторінкам JavaScript та багатофункціональний набір інструментів інтерфейсу Bootstrap.

Рішення від Amazon Web Services забезпечують інструментами для побудови інфраструктури та забезпечення бази для зберігання файлів розгорнутої системи. Насамперед, такі сервіси, як AWS CDK (Cloud Development Kit) для створення інфраструктури системи, служба для зберігання об’єктів Amazon S3 (Amazon Simple Storage Service), шо містить файли сайту. Налагоджено процеси інтеграції коду до розгорнутої інфраструктури, користуючись системою контролю версій Git та платформою для неперервної інтеграції та неперервної доставки коду GitHub Actions.

3.1 JavaScript

Динамічна, об'єктно-орієнтована прототипна мова програмування. Ця мова є реалізацією стандарту ECMAScript. Частіше за все використовується при створенні сценаріїв веб сторінок, це надає можливість на боці клієнта (пристрій кінцевого користувача) взаємодіяти з користувачем, керувати браузером, асинхронно обмінюватися даними з сервером, змінювати структуру та зовнішній вигляд веб сторінки [8].

JavaScript класифікують як прототипну , скриптовану мову програмування з динамічною типізацією. Крім прототипної, JavaScript частково підтримує інші парадигми програмування, наприклад імперативну та частково функціональну і деякі відповідні архітектурні властивості, зокрема: динамічна та слабка типізація, автоматичне керування пам'яттю, прототипне наслідування, функції як об'єкти першого класу.

Мова JavaScript використовується для:

- написання сценаріїв веб сторінок для надання їм інтерактивності
- створення односторінкових та прогресивних веб застосунків (React, AngularJS, Vue.js)
- програмування на боці сервера (Express.js)
- стаціонарних застосунків (Electron)
- мобільних застосунків (React Native)
- сценаріїв в прикладних програмах (наприклад, в програмах зі складу Adobe Creative Suite чи Apache JMeter)
- всередині PDF-документів тощо.

3.2 Середовище Node.js

Асинхронне кероване подіями середовище виконання JavaScript, Node.js розроблено для створення масштабованих мережевих додатків, на відміну від сьогоднішньої більш поширеної моделі паралельності, в якій використовуються потоки ОС. Мережа на основі потоків є відносно неефективною та дуже важкою у використанні. Крім того, користувачі Node.js не турбуються про блокування процесу, оскільки немає блокування. Майже жодна функція в Node.js безпосередньо не виконує введення-виведення, тому процес ніколи не блокується, за винятком випадків, коли ввод-вивід виконується за допомогою синхронних методів стандартної бібліотеки Node.js.

Node.js [9] схожий за дизайном на такі системи, як Ruby's Event Machine і Python's Twisted, і під впливом таких систем Node.js розширює модель подій. Він представляє цикл подій як конструкцію під час виконання, а не як бібліотеку. В інших системах завжди існує блокуючий виклик для початку циклу подій. Зазвичай поведінка визначається за допомогою зворотних викликів на початку сценарію, а в кінці сервер запускається за допомогою блокуючого виклику, наприклад EventMachine::run(). У Node.js немає такого виклику циклу start-the-event. Node.js просто входить у цикл подій після виконання сценарію введення. Node.js виходить з циклу подій, коли більше немає зворотних викликів для виконання. Ця поведінка схожа на JavaScript у браузері — цикл подій прихований від користувача.

Node.js добре підходить для створення веб-бібліотеки або фреймворку, дає змогу скористатися перевагами кількох ядер у своєму середовищі. Дочірні процеси можуть бути створені за допомогою API child_process.fork() і розроблені так, щоб до них легко можна було звернутись. На цьому ж інтерфейсі побудований модуль кластера, який дозволяє спільно використовувати сокети між процесами, щоб уможливити балансування навантаження на ядра.

3.3 Фреймворк Bootstrap

Bootstrap — це найпопулярніший фреймворк HTML, CSS і JavaScript для розробки адаптивного веб-сайту, зручного для мобільних пристроїв. Інтерфейсний фреймворк, який використовується для простішої та швидкої веб-розробки [17]. Він включає в себе шаблони дизайну на основі HTML і CSS для типографіки, форм, кнопок, таблиць, навігації, модулів, каруселі зображень та багатьох інших елементів. Він також може використовувати плагіни JavaScript, це полегшує створення адаптивних дизайнів.

Основні інструменти Bootstrap:

Сітка — наперед задані розміри колонок, що можна одразу використати, наприклад, ширина колонки сто сорок пікселів відноситься до класу .span2 (.col-md-2 в 3-ій версії фреймворку), що можна використати в CSS описанні документа;

Шаблони – фіксований або адаптивний шаблон документа;

Типографіка - описання шрифтів, визначення певних класів для цих шрифтів, таких як код, цитати т.і.;

Медіа — надає фіксоване керування зображеннями та відео;

Таблиці — є засобами для оформлення таблиць, додавання функціональності сортування;

Форми – є класами, що дозволяють описання форм та деяких подій, які з ними відбуваються;

Навігація – це класи оформлення панелей, вкладок, для оформлення переходу по сторінках, меню та панелі інструментів;

Алерти – оформлення діалогових підказок, вікон та спливаючих вікон.

3.4 Онлайн-сервіс AWS Cloud Development Kit

Фреймворк розробки програмного забезпечення з відкритим кодом для визначення ресурсів хмарних додатків за допомогою цілого ряду мов програмування.

Надання хмарних додатків може бути складним процесом, який вимагає виконання дій вручну, написання користувацьких сценаріїв, підтримки шаблонів або вивчення мов домену. AWS CDK використовує сильні сторони розробки за рахунок вже знайомих мов програмування для моделювання ваших програм. Він надає високорівневі компоненти, які називаються конструкціями, що попередньо налаштовують хмарні ресурси з перевіреними параметрами за замовчуванням, тим самим дає змогу легко створювати хмарні програми. AWS CDK забезпечує ресурси для програмного рішення безпечним, повторюваним способом через AWS CloudFormation. Він також дозволяє створювати та ділитися своїми власними конструкціями, що враховують вимоги вашої організації, допомагаючи прискорити нові проекти.

Cdktf надає конструкції CDK для визначення файлів стану Terraform HCL у TypeScript і Python. Для користувачів Kubernetes проект cdk8s дозволяє використовувати конструкції CDK для визначення конфігурації Kubernetes в TypeScript, Python і Java. Крім того, cdk8s можна використовувати для визначення інфраструктури Kubernetes, яка працює в будь-якому місці, і може використовуватися з бібліотекою конструкцій Amazon Elastic Kubernetes Service (Amazon EKS) AWS CDK.

3.5 Служба Amazon S3

Amazon Simple Storage Service (Amazon S3) — це служба зберігання об’єктів, яка пропонує провідну в галузі масштабованість, доступність даних, безпеку та

продуктивність. Клієнти будь-яких розмірів і галузей можуть використовувати Amazon S3 для зберігання та захисту будь-якої кількості даних для різних випадків використання, таких як веб-сайти, мобільні додатки, резервне копіювання та відновлення, архівування, корпоративні програми, пристрої Інтернету речей та великих масивів даних.

Amazon S3 надає функції керування, щоб оптимізувати, організувати та налаштувати доступ до даних відповідно конкретних вимог бізнесу, організації та відповідності.

Служба пропонує цілий ряд класів зберігання, розроблених для різних випадків використання. Наприклад, є можливість зберігати критично важливі виробничі дані в S3 Standard для частого доступу, заощаджувати витрати, зберігаючи дані до рідкого доступу в S3 Standard-IA або S3 One Zone-IA, а також архівувати дані з найнижчими витратами в S3 Glacier Instant Retrieval, Гнучкий пошук S3 Glacier і глибокий архів S3 Glacier.

Можна зберігати дані зі змінюваними або невідомими шаблонами доступу в S3 Intelligent-Tiering, який оптимізує витрати на зберігання, автоматично переміщуючи дані між чотирма рівнями доступу, коли шаблони доступу змінюються. Ці чотири рівні доступу включають два рівні доступу з низькою затримкою, оптимізовані для частого та нечастого доступу, і два рівні доступу до архіву, розроблені для асинхронного доступу до даних, до яких рідко звертаються.

Amazon S3 має функції керування сховищем, за допомогою яких можна керувати витратами, відповідати нормативним вимогам, зменшувати затримки та зберігати кілька окремих копій даних для відповідності вимогам:

Життєвий цикл S3 – політика життєвого циклу, щоб керувати об’єктами та ефективно зберігати їх протягом усього життєвого циклу. Є можливість переносити об’єкти в інші класи зберігання S3 або об’єкти, термін дії яких закінчився.

Блокування об’єктів S3 – запобігає видаленню або перезапису об’єктів Amazon S3 протягом фіксованого періоду часу або на невизначений термін. Є можливість використовувати Object Lock, щоб задовольнити нормативні вимоги, які вимагають сховища з можливістю запису write-once-read-many (WORM) або просто

додати ще один рівень захисту від змін і видалення об’єктів.

Реплікація S3 – реплікація об’єктів та їх відповідні метадані та теги об’єктів в один або кілька сегментів призначення в тих самих або різних регіонах AWS для зменшення затримки, відповідності, безпеки та інших випадків використання.

Пакетні операції S3 – керування мільярдами об’єктів у масштабі за допомогою одного запиту API S3 або кількох кліків на консолі Amazon S3. Можливість використовувати пакетні операції для виконання таких операцій, як копіювання, виклик функції AWS Lambda та відновлення для мільйонів або мільярдів об’єктів.

Amazon S3 надає функції для аудиту та керування доступом до сегментів і об’єктів. За замовчуванням сегменти S3 та об’єкти в них є приватними. Доступ надано лише до ресурсів S3, які створюються. Щоб надати детальні дозволи до ресурсів, які підтримують певний варіант використання, або для аудиту дозволів ресурсів Amazon S3, можна скористатися наведеними нижче функціями:

S3 Block Public Access – блокувати публічний доступ до сегментів і об’єктів S3. За замовчуванням параметри Блокувати публічний доступ увімкнено на рівні облікового запису та сегмента.

AWS Identity and Access Management (IAM) – створення користувачів IAM для облікового запису AWS, щоб керувати доступом до ресурсів Amazon S3. Наприклад, є можливість використовувати IAM з Amazon S3, щоб керувати типом доступу користувача або групи користувачів до сегмента S3, яким володіє обліковий запис AWS.

Політики сегментів – використання мови політики на основі IAM, щоб налаштувати дозволи на основі ресурсів для сегментів S3 та об’єктів у них.

Точки доступу Amazon S3 – налаштування іменованих кінцевих точок мережі за допомогою спеціальних політик доступу, щоб керувати доступом до даних у масштабі для спільних наборів даних у Amazon S3.

Списки контролю доступу (ACL) – надання дозволу на читання та запис для окремих сегментів і об’єктів авторизованим користувачам. Як правило, рекомендовано використовувати політики на основі ресурсів S3 (політики сегментів і політики точок доступу) або політики IAM для контролю доступу замість ACL.

ACL — це механізм контролю доступу, який передував політикам на основі ресурсів та IAM.

3.6 Веб-сервіс Amazon CloudFront

Веб-сервіс, який пришвидшує розповсюдження статичного та динамічного веб-вмісту, такого як .html, .css, .js та файли зображень. CloudFront доставляє дані через всесвітню мережу центрів обробки даних, які називаються граничними місцями. Коли користувач запитує вміст даних, що обслуговуються за допомогою CloudFront, запит направляється до граничного розташування, яке забезпечує найнижчу затримку (часову затримку), щоб вміст доставлявся з найкращою можливою продуктивністю.

Якщо дані знаходяться не в цьому граничному місці, CloudFront отримує їх з визначеного джерела, наприклад, сегмента Amazon S3, каналу MediaPackage або HTTP-сервера (наприклад, веб-сервер), яке було ідентифіковано як джерело остаточної версії даних.

CloudFront пришвидшує розповсюдження збережених даних, перенаправляючи кожен запит користувача через магістральну мережу AWS до граничного розташування, яке може найкраще обслуговувати вміст. Як правило, це граничний сервер CloudFront, який забезпечує найшвидшу доставку до переглядача. Використання мережі AWS різко зменшує кількість мереж, через які повинні проходити запити користувачів, що покращує продуктивність. Користувачі отримують меншу затримку — час, необхідний для завантаження першого байта файлу — і вищі швидкості передачі даних.

Також, це забезпечує підвищену надійність та доступність, оскільки копії файлів (також відомих як об’єкти) тепер зберігаються (або кешуються) у кількох периферійних місцях по всьому світу.

Якщо дані уже знаходяться на межі з найменшою затримкою, CloudFront доставить негайно.

Мережа доставки вмісту Amazon CloudFront широко масштабується та поширюється по всьому світу. Мережа CloudFront має понад 310 точок присутності (300+ Edge локацій і 13 регіональних кешів середнього рівня) у 90+ містах у 47 країнах і використовує високостійку приватну магістральну мережу для чудової продуктивності та доступності для кінцевих користувачів.

Amazon CloudFront — це високобезпечна CDN, яка забезпечує захист на рівні мережі та додатків. Ваш трафік і програми отримують переваги завдяки різноманітним вбудованим захистам, таким як Amazon Shield Standard, без додаткових витрат.

Amazon CloudFront інтегровано зі службами Amazon Web Services, такими як Amazon S3, Amazon EC2 і Elastic Load Balancing. Усі вони доступні через ту саму консоль, і всі функції в CDN можна програмно налаштувати за допомогою пакетів SDK або консолі керування Amazon Web Services. Нарешті, якщо використовуються вихідні веб-служби Amazon, такі як Amazon S3, Amazon EC2 або Elastic Load Balancing, відсутня плата за дані, що передаються між цими службами та CloudFront.

3.7 Платформа GitHub Actions

Платформа неперервної інтеграції та неперервної доставки (CI/CD), яка дозволяє автоматизувати конвеєр складання, тестування та розгортання. Надає можливість створювати робочі процеси, які створюють і тестують кожен запит на витяг до вашого сховища, або розгортати об’єднані запити на витяг у виробництво.

GitHub Actions [13] виходить за рамки просто DevOps і дозволяє запускати робочі процеси, коли у сховищі відбуваються інші події. Наприклад, можна запустити робочий процес, щоб автоматично додавати відповідні мітки щоразу, коли хтось створює новий запрос у сховищі.

GitHub надає віртуальні машини Linux, Windows і macOS для запуску робочих процесів, або можна розмістити власні у центрі обробки даних або хмарній інфраструктурі.

Можна налаштувати робочий процес GitHub Actions так, щоб він запускався, коли у сховищі відбувається подія, наприклад, відкривається запит на витяг або створюється проблема. Робочий процес містить одне або кілька завдань, які можуть виконуватися в послідовному порядку або паралельно. Кожне завдання виконуватиметься всередині власної віртуальної машини або всередині контейнера і має один або кілька кроків, які запускають визначений сценарій або дію, що може спростити ваш робочий процес.

GitHub Actions дає змогу створювати власні робочі процеси життєвого циклу розробки програмного забезпечення (SDLC) безпосередньо у сховищі GitHub.

Потрібно налаштувати дії GitHub за допомогою синтаксису YAML і зберегти їх як файли робочого процесу у сховищі. Робочі процеси — це спеціальні автоматизовані процеси, які можна налаштувати у сховищі для створення, тестування, упаковки, випуску або розгортання будь-якого проекту на GitHub. Є можливість писати окремі завдання, які називаються діями, і об’єднувати їх, щоб створити індивідуальний робочий процес. Після того, як успішно створено файл робочого процесу YAML і запущено робочий процес, з’являються журнали збірки, результати тестів, артефакти та статуси для кожного кроку робочого процесу. Його можна налаштувати для створення проекту на різних платформах розробки.

GitHub Actions допоможе автоматизувати робочі процеси розробки програмного забезпечення там же, де зберігається код і йде робота над запитами та проблемами на витяг, і щоразу, коли це відбувається, він намагатиметься створити проект.

3.8 Середовище розробки VS Code

Редактор вихідного коду, що доступний для Windows, macOS та Linux. Він поставляється з вбудованою підтримкою таких мов, як JavaScript, TypeScript і Node.js та має велику екосистему з розширенням для інших мов (C++, C#, Java, Python, PHP, Go) і середовищ виконання [6].

Visual Studio Code – це зручний редактор, тому що має велику кількість розширень для поліпшення роботи. Велику популярність мають розширення пов’язані з веб технологіями розробки та не тільки. Нижче наведений список переваг цього середовища розробки:

- Інтеграція TypeScript та JavaScript з фреймворками;
- Виконання функції автодоповнення тексту кода для мов програмування JavaScript, HTML [10] і CSS [11];
- Середовище містить в собі засоби Live Edit – зміни занесені в код миттєво будуть відображатися в браузері;
- Підтримка роботи з різними мовами програмування та розмітки, наприклад JavaScript, CSS і HTML;;
- Підтримка роботи з Git (системою контролю версій) — дозволяє відразу заносити зміни до віддаленого репозиторію без сторонніх утиліт.

Висновки до розділу

Відповідно до поставленої задачі та провівши аналіз різних програмних рішень було обрано мову програмування JavaScript та платформу Node.js для побудови клієнт-серверної реалізації веб-порталу книжкової полиці, JavaScript — доцільний інструмент для розробки веб-орієнтованого рішення, а Node.js – містить всі необхідні компоненти та інструменти для обробки переданих даних користувачів в браузері, створення клієнт-серверної архітектури.

Для описання та побудови інфраструктури для клієнт-серверної системи було обрано мову програмування TypeScript, а також програмну платформу GitHub Actions – для написання сценарію розгортання системи, неперервної інтеграції коду та автоматизації цих процесів.

До переваг даної мови належать:

- Структуризація, типізація;
- Можливість використання для багатьох сервісів, що надають послуги для побудови інфраструктури;
- Підтримка багатьма популярними IDE.
- Швидкість розробки.
- До недоліків мови належать:
- Не всі браузери підтримують налагодження TypeScript у консолі без зайвих налаштувань;
- Неявна статична типізація. Завжди можна описати тип як any, що фактично відключить приведення до конкретного типу цієї змінної.

Приведені недоліки даної мови програмування не є значними і не мають впливу на побудову інфраструктури для клієнт-серверної системи, таки чином TypeScript була обрана для реалізації цього рішення.

4 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

Основа розробки програмного рішення поділяється на дві складові: побудова клієнт-серверного рішення для веб-порталу книжкової полиці, створення інфраструктури системи для розробників та автоматизація процесів розгортання та неперервної інтеграції клієнт-серверної системи.

Розробку програмного продукту умовно можна розбити на три частини:

- реалізація клієнт-серверної системи;
- побудова інфраструктури системи;
- автоматизації процесів розгортання та неперервної інтеграції коду.

4.1 Опис архітектури клієнт-серверної системи

1. Сторінка Авторизації – її переглядає користувач коли заходить на сайт, на ній відображено форми авторизації та можливість перейти до сторінки реєстрації.
2. Сторінка Реєстрації – дає створити аккаунт на даному ресурсі.
3. Головна сторінка – перша сторінка, що бачить користувач, коли авторизується у системі, саме книжна полиця за переліком категорій, та можливістю переглянути список бажаного до прочитання.
4. Сторінки Категорій – відображається список книг за даною категорією.
5. Сторінка Книги – інформація про літературу, відео-матеріали, присутня можливість додати книгу до бажаного.
6. Інформація вноситься у базу даних .json, там вона неперервно обробляється та оновлюється.

На сторінках пунктів 3-5 присутня строка пошуку для зручного користування системою.

4.2 Опис інфраструктури

Система включає в себе такі компоненти:

1. Користувацький інтерфейс клієнт-серверної системи;
2. Серверна частина;
3. Інфраструктура, що забезпечує розгортання та інтеграцію коду;
4. Сценарій для автоматизації процесів;
5. Сервер для взаємодії компонентів системи.

Перші два компоненти являють собою частини клієнт-серверної системи. Проаналізувавши їх можна побудувати інфраструктуру, що дасть можливість розробникам швидко та ефективно додавати нові функції до системи, а також забезпечує UI дизайнерів інструментом для швидкої інтеграції своїх робіт до готового програмного продукту.

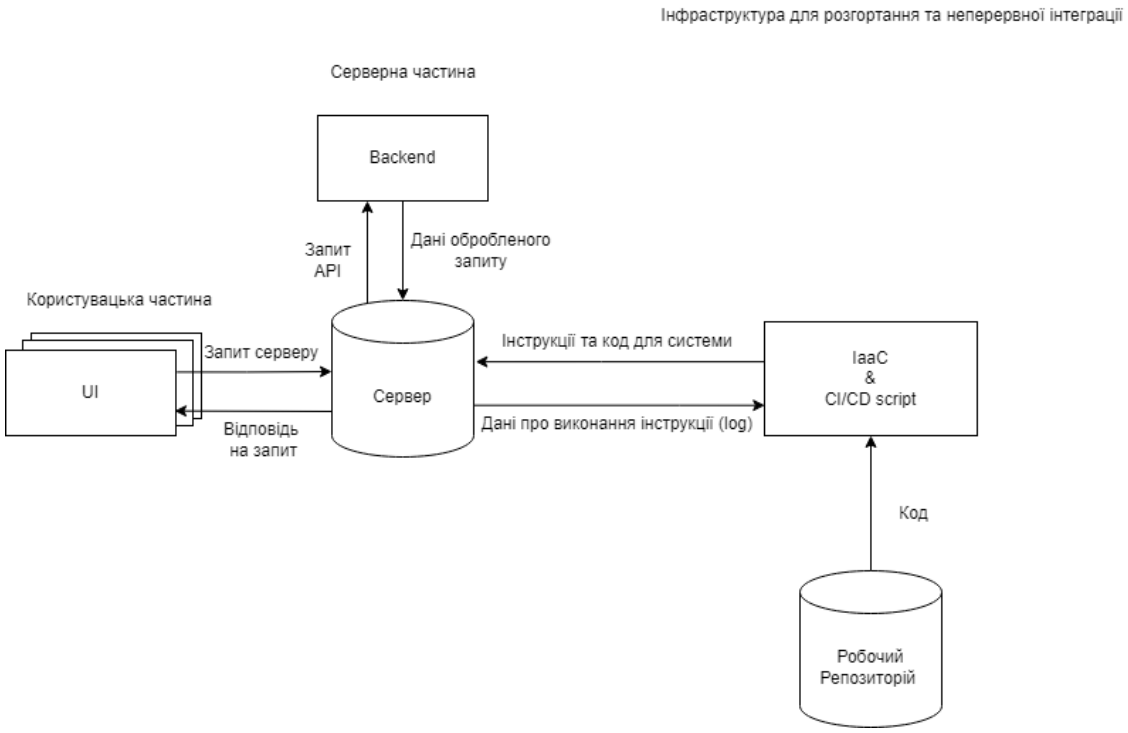

Рисунок 4.1 – Блок-схема Інфраструктури

Користувацька частина рішення представляє собою HTML файли та CSS розмітки, а також JavaScript код для функціональної частини системи.

Серверна частина реалізована, використовуючи Node.js. Вона складається з таких умовних підчастин:

- модуль навігації;
- модуль обробки даних та їх збереження.

Інфраструктура побудована на базі Amazon CDK. Її підчастини включають в себе такі модулі:

- стек CloudFormation для реалізації моделювання та розподілення ресурсів AWS.
- сховище Simple Storage Service Bucket, що зберігає дані для серверу системи.
- сервіс для доставки контенту CloudFront, що дозволяє поширювати контент для кінцевих користувачів із мінімальними затримками та високою швидкістю передачі даних.

Сценарій прописан за допомогою скрипту на базі платформи GitHub Actions. Його підчастини включають в себе такі робочі процеси:

- ініціалізація віртуальної машини на базі Ubuntu з певними параметрами.
- перевірка сховища GitHub репозиторію, щоб робочий процес міг отримати до нього доступ.
- завантаження шифрованих облікових даних Amazon з репозиторію.
- встановлення потрібних для побудови інфраструктури модулів.
- виконання інструкції по інтеграції коду та розгортанню системи.
- перевірка побудованого рішення.

4.2.1 Модуль навігації

Модуль навігації описує відповіді програмної платформи Node.js на запити, що надходять з клієнтської сторони на конкретну URL-адресу. Якщо запит відповідає певному маршруту, викликається для обробки запиту відповідний обробник.

Тобто даний модуль виконує роль посередника між запитами клієнта та направленням йому відповідної інформації або передачу даних іншим компонентам.

4.2.2 Модуль обробки даних та їх збереження

Модуль обробки даних виконує функцію завантаження, обробки та внаслідок збереження даних. Описати функціонал даного компонента можна на прикладі завантаження статичного вмісту для сторінки:

1. Прийняття файла через інтерфейс вебу;
2. Обробка цього файла на сервері, початок роботи даного модуля: перейменування, зменшення роздільної здатності, зменшення розмірів вмісту, зменшення розміру файлу;
3. Створення структури директорій для файлу, що надійшов;
4. Збереження файла на сервері в сформованій структурі;
5. Запис даних про розміщення файла до бази даних.

4.2.3 Стек CloudFormation

Шаблони CloudFormation є текстовими файлами у форматі JSON або YAML, що складаються з п'яти типів елементів:

1. необов'язковий перелік параметрів шаблону (вхідні значення, що передаються в момент створення стека);

2. необов'язковий список вихідних значень (наприклад, повна URL-адреса інтернет-програми);

3. необов'язковий список таблиць даних, використовуваний перегляду статичних значень конфігурації (наприклад, імен AMI);

4. список ресурсів AWS та їх значень конфігурації;

5. номер версії формату файлу шаблону.

Для налаштування різних аспектів шаблону під час виконання, коли стек вже буде створено, можна використовувати параметри. При створенні стека AWS CloudFormation Консоль керування AWS автоматично генерує та відображає спливаюче діалогове вікно, в якому можна редагувати значення параметрів.
У випадку програмного рішення для клієнт-серверної системи було використано маніфест інфраструктури за допомогою AWS CDK, що дозволяє передавати всі параметри до інфраструктури автоматично при розгортанні сховища користуючись підходом для управління та опису інфраструктури IaC.

4.2.4 Сховище Amazon Simple Storage Service

За допомогою об’єкта цієї служби - Bucket, проходить ініціалізація сховища з властивостями масштабованості, доступності даних, безпеки та продуктивності. Завдяки ефективним класам зберігання та простим у використанні функціям керування можна оптимізувати витрати, упорядкувати дані та налаштувати точні засоби контролю доступу, щоб відповідати вимогам системи.
Реалізація клієнт-серверної системи має статичний вміст серед файлів готового програмного продукту. Статичний вміст системи включає такі файли, як зображення, сценарії, як-от .css або js.

Простий і гнучкий підхід до статичного вмісту, доступного в Інтернеті, — це зберігання його в Amazon S3 Bucket. S3 простий у налаштуванні та використанні, він призначений для зберігання та отримання будь-якої кількості файлів або об’єктів з будь-якої точки Інтернету.

Розміщуючи свої файли S3 у хмарі, багато чого стає набагато простіше. По-перше, не потрібно планувати та виділяти певну кількість місця для зберігання, оскільки сегменти S3 автоматично масштабуються. Крім того, оскільки S3 є безсерверною службою, не потрібно самостійно керувати або виправляти сервери, які зберігають файли. Нарешті, навіть якщо потрібен сервер для програми (наприклад, при наявності динамічної програми), сервер може бути меншим, оскільки йому не потрібно обробляти запити на статичний вміст.

4.2.5 Сервіс для доставки контенту CloudFront

Зберігання статичного вмісту за допомогою S3 надає багато переваг. Але щоб допомогти оптимізувати продуктивність і безпеку програми, ефективно керуючи витратами треба налаштувати Amazon CloudFront для роботи з сегментом S3 для обслуговування та захисту вмісту. CloudFront — це служба мережі доставки вмісту (CDN), яка забезпечує статичний і динамічний веб-контент, відеопотоки та API по всьому світу, безпечно та в масштабі.

CloudFront обслуговує вміст через всесвітню мережу центрів обробки даних під назвою Edge Locations. Використання периферійних серверів для кешування та обслуговування вмісту покращує продуктивність, надаючи вміст ближче до місця розташування користувача. CloudFront має периферійні сервери в різних місцях [7] по всьому світу, як можна побачити на наступній карті (рисунок 4.2):

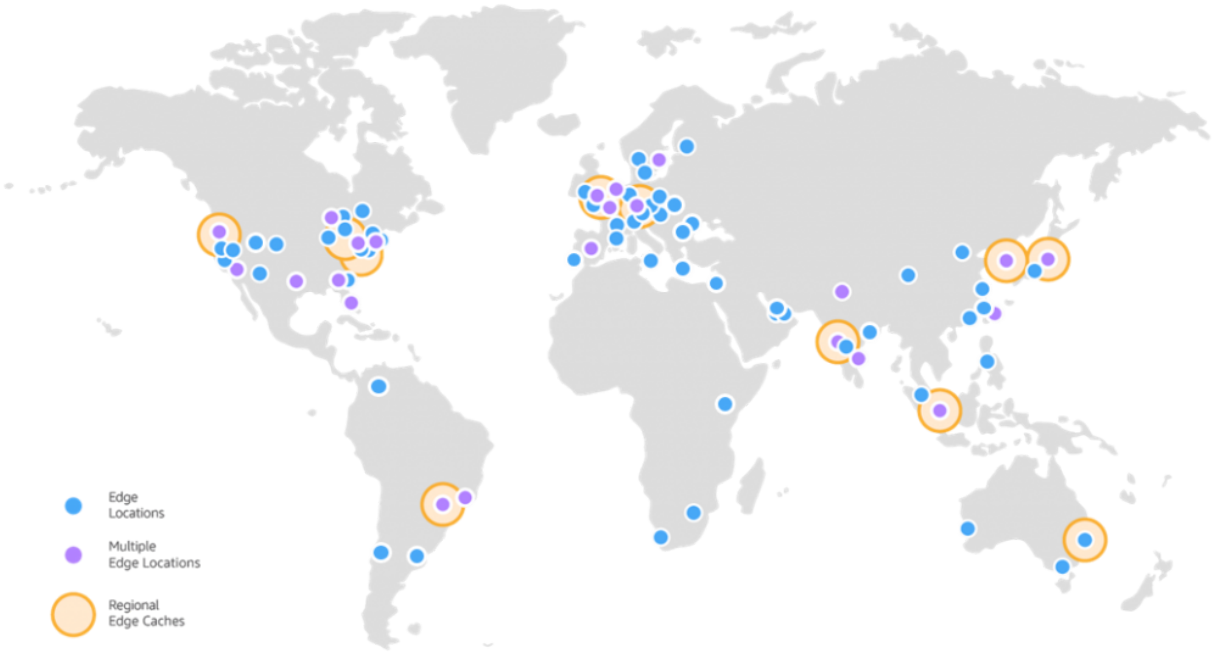

Рисунок 4.2 – Edge Locations для служби CloudFront

4.3 Сценарій роботи користувачів інфраструктури

На діаграмі прецедентів (рисунок 4.3) можна побачити та ознайомитися з усіма доступними діями для користувача залежно від його ролі, а також відношення між акторами та прецедентами в розробленій інфраструктурі.

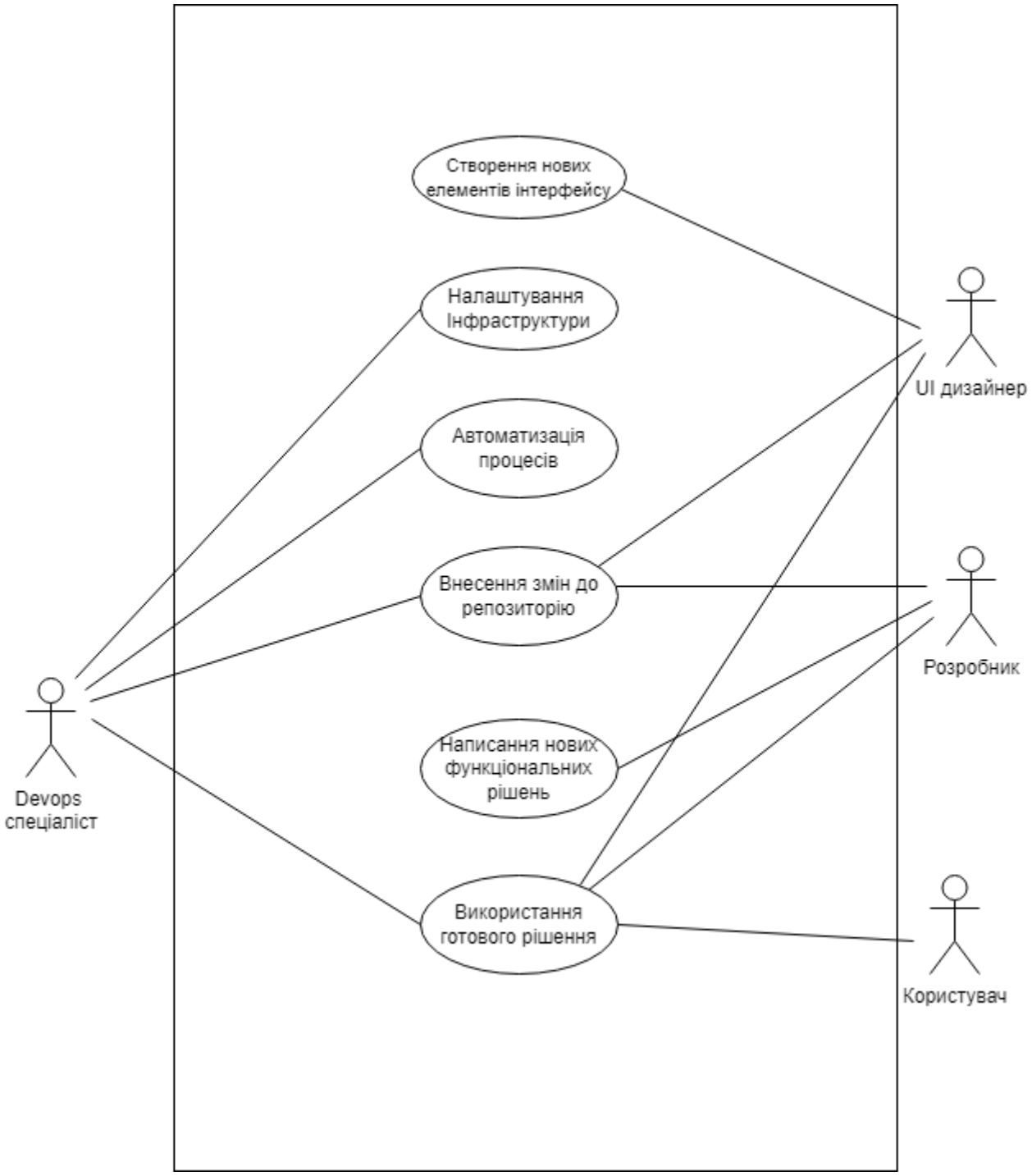

Рисунок 4.3 – Діаграма прецедентів системи

У системному рішенні присутні такі ролі:

1. Користувач – має обмежені права, а саме може переглядати дані клієнтської сторони реалізованої системи, має доступ до користувацького функціоналу рішення.

2. UI дизайнер – наділений правами користувача, а також має доступ до репозиторію для внесення змін до інтерфейсу продукту.
3. Розробник – займається розробкою нового функціоналу, має доступ до репозиторію та сценарію автоматизації процесів неперервної інтеграції коду до системи
4. Devops – аналізує програмне рішення та займається побудовою відповідної до задач інфраструктури, що забезпечить стабільну роботу клієнт-серверної системи та доступ користувачів до неї. Займається написанням сценарію для автоматизації процесів розгортання системи та неперервної інтеграції коду до рішення.

Кожен з акторів окрім Користувача має доступ до даних репозиторію, що складаються з файлів серверної частини системи, клієнтської частини (рисунок 4.4) та файлів сценарію, який забезпечує процеси автоматизації розгортання системи та неперервної інтеграції коду до рішення.

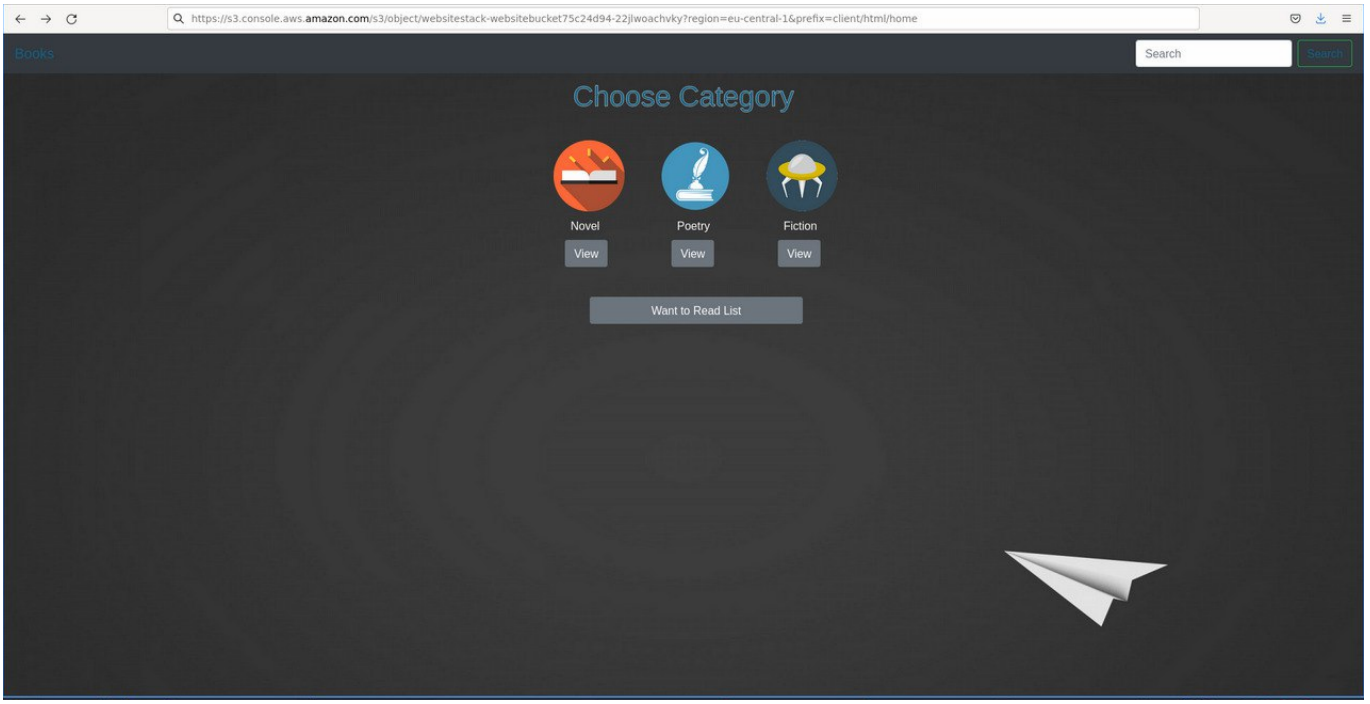

Рисунок 4.4 – Вигляд клієнтської частини розгорнутого рішення

4.3.1 Діаграма станів

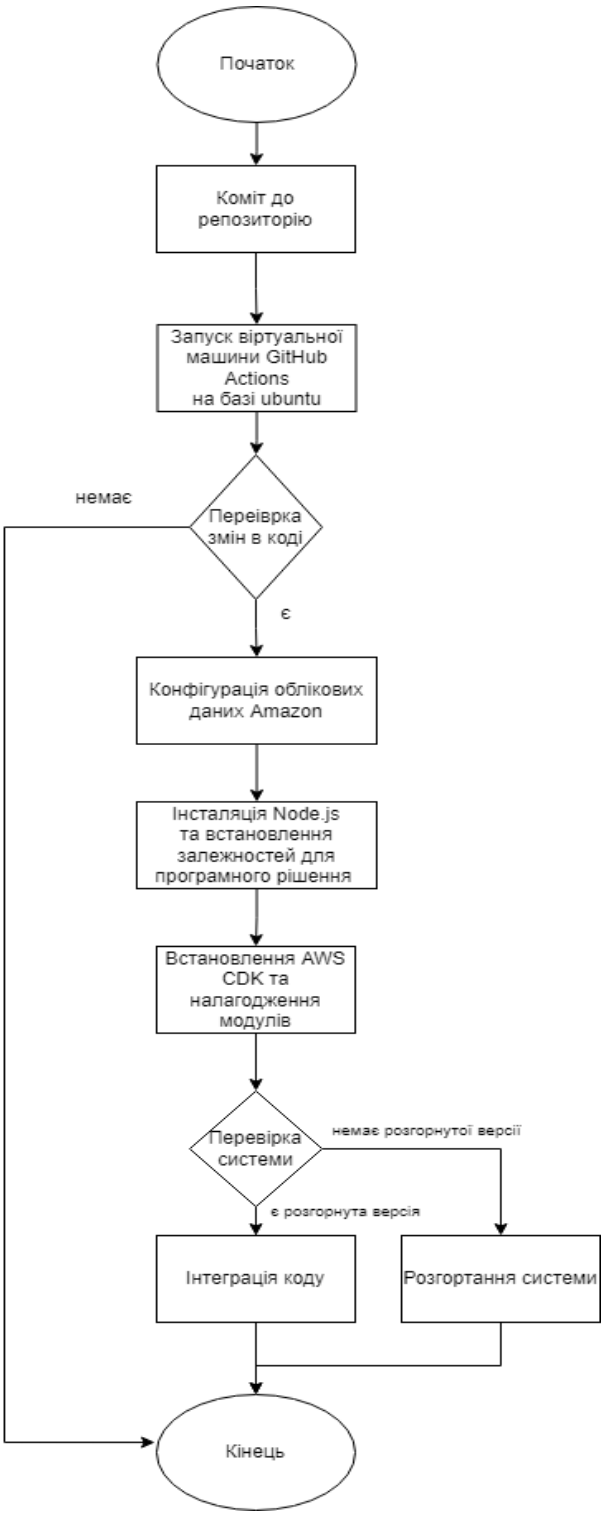

Рисунок 4.5 Діаграма станів

Використовують систему: Devops, Розробник, UI дизайнер, Користувач.

Devops, який відповідає за:

Створення інфраструктури,

Написання робочих процесів для розгортання системи, впровадження методів інтеграції коду,

Автоматизація цих процесів.

Розробник, який відповідає за:

Створення програмного коду системи,

Перевірку логіки рішення,

Оновлення версії коду на репозиторії.

UI дизайнер, відповідає за:

Створення інтерфейсу продукту,

Оновлення масиву даних на сайті,

Оновлення файлів елементів інтерфейсу на репозиторії.

Користувач, який може:

Переглядати інформацію на сайті.

Висновки до розділу

У даному розділі було розглянуто технології для побудови інфраструктури для клієнт-серверної системи, а також написання сценарію для автоматизації процесів розгортання та неперервної інтеграції коду програмного рішення. В результаті, для реалізації системи обрано було мову програмування TypeScript для маніфестації та побудови інфраструктури для клієнт-серверної системи на основі мови JavaScript для динамічного та статичного вмісту UI частини сторінок та платформи NodeJS на

серверній частині. Середовищем для розробки інфраструктури був обран Visual Studio Code через підтримку TypeScript. Написання сценаріїв для автоматизації процесів розгортання та неперервної інтеграції коду реалізована за допомогою платформи для неперервної інтеграції та неперервної доставки коду GitHub Actions.

5 МЕТОДИКА РОБОТИ З СИСТЕМОЮ

В цьому розділі описуються вимоги до системи користувача та розробника для коректної роботи системи. А також надається інструкція розробникам для роботи з інфраструктурою.

5.1 Системні вимоги

Користувач зможе отримати доступ до клієнт-серверної системи навіть з мінімальною конфігурацією комп’ютера. Важливою умовою для коректного відображення веб сторінок та функціоналу сайту є використання однієї з наведених версій браузеру:

Microsoft Edge – версії 16 або вище;

Internet Explorer – версії 12 або вище;

Firefox – версії 55 або вище;

Chrome – версії 52 або вище;

Opera – версії 39 або вище.

Safari – версії 11 або вище;

Для коректної роботи розробника з інфраструктурою необхідне виконання

наступних мінімальних системних вимог:

1. Об’єм оперативної пам’яті (RAM) – 1024 МБ для систем на базі Linux, 2048 МБ для Windows.
2. Операційна система Windows 7/10/11, Linux, MacOS.
3. Процесор Intel Pentium 4 або більш пізньої версії з підтримкою SSE3.

На комп’ютері повинне бути встановлене наступне програмне забезпечення:

Інтернет браузер: Google Chrome/Opera/Mozilla Firefox/Safari.

Система контролю версій Git.

5.2 Сценарій роботи користувача з інтерфейсом

Ініціалізація системи користувачем відбувається при переході за відповідним посиланням (url адресою сайту), що було вказане при налаштуванні сервера.

5.2.1 Авторизація користувача

На сторінці авторизації (рисунок 5.1), користувач може побачити поля для вводу даних про обліковий запис в клієнт-серверній системі веб-порталу книжкової полиці. Також присутня можливість зареєструватися (рисунок 5.3) на даному ресурсі натиснувши кнопку “ I don’t have an account”(рисунок 5.2) та перейшовши до форми реєстрації облікового запису.

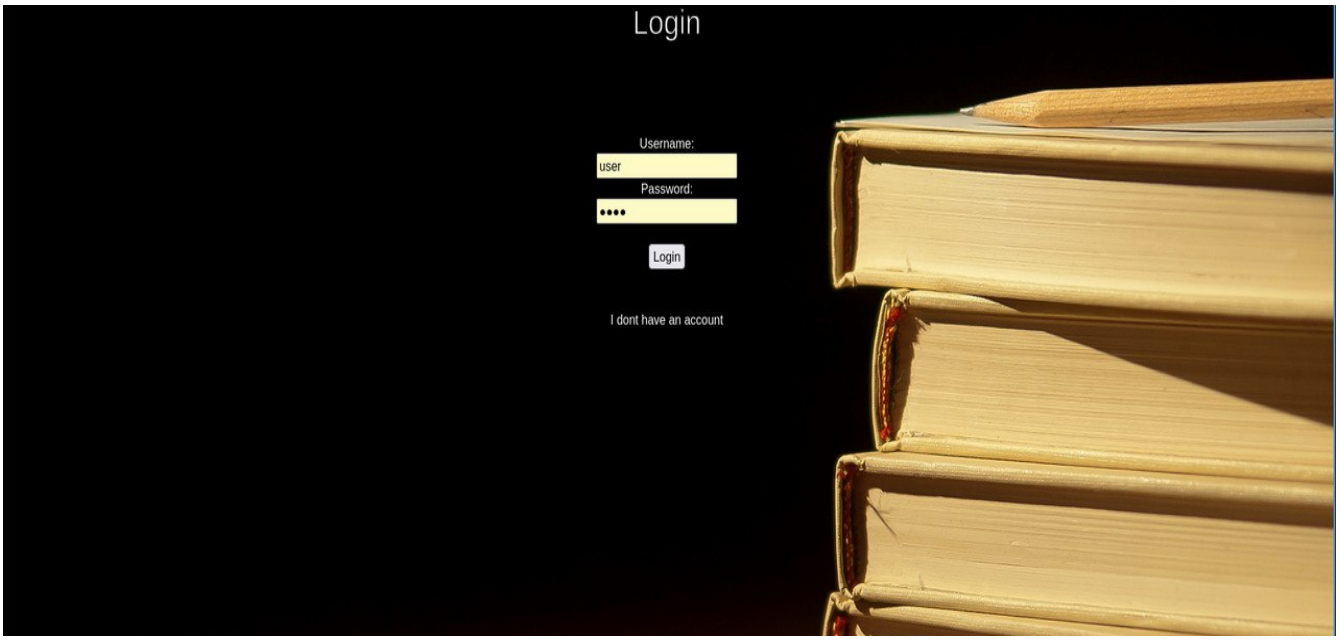

Рисунок 5.1 – Сторінка авторизації

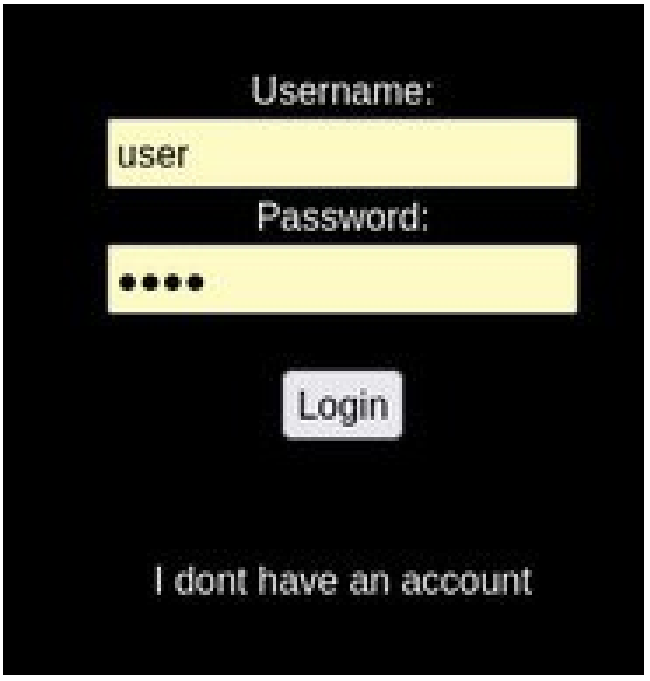

Рисунок 5.2 – Перехід до реєстрації

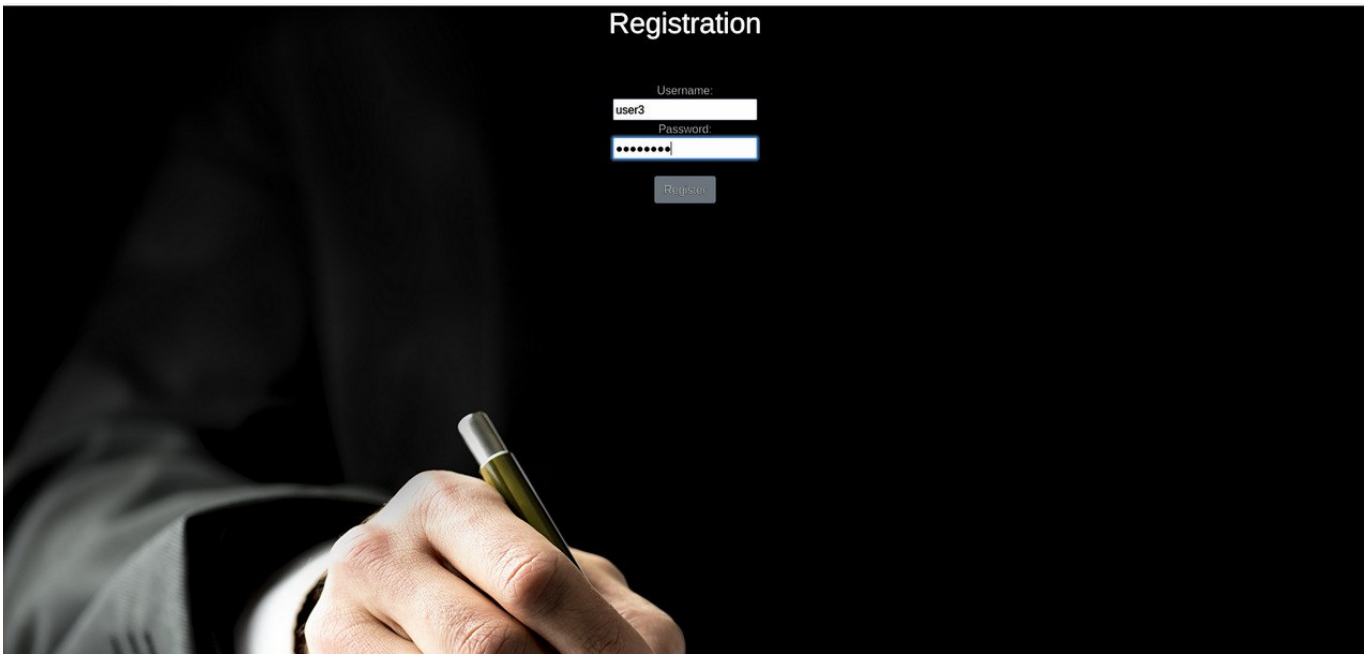

Рисунок 5.3 – Сторінка Реєстрації

5.2.2 Головна Сторінка

На головній сторінці (рисунок 5.4) користувач може побачити останню інформацію про категорії з літературою, що доступні на даному веб-порталі книжної полиці, натиснувши на кнопку “View” під назвою категорії (рисунок 5.5) - можна перейти до літератури. Перехід до списку бажаного (перелік книг, що читач додавав для подальшого читання) реалізован через кнопку “Want to Read List” (рисунок 5.6). Також є можливість одразу шукати певний твір за його назвою , користуючись пошуковою строкою (рисунок 5.7) у правому верхньому кутку системи.

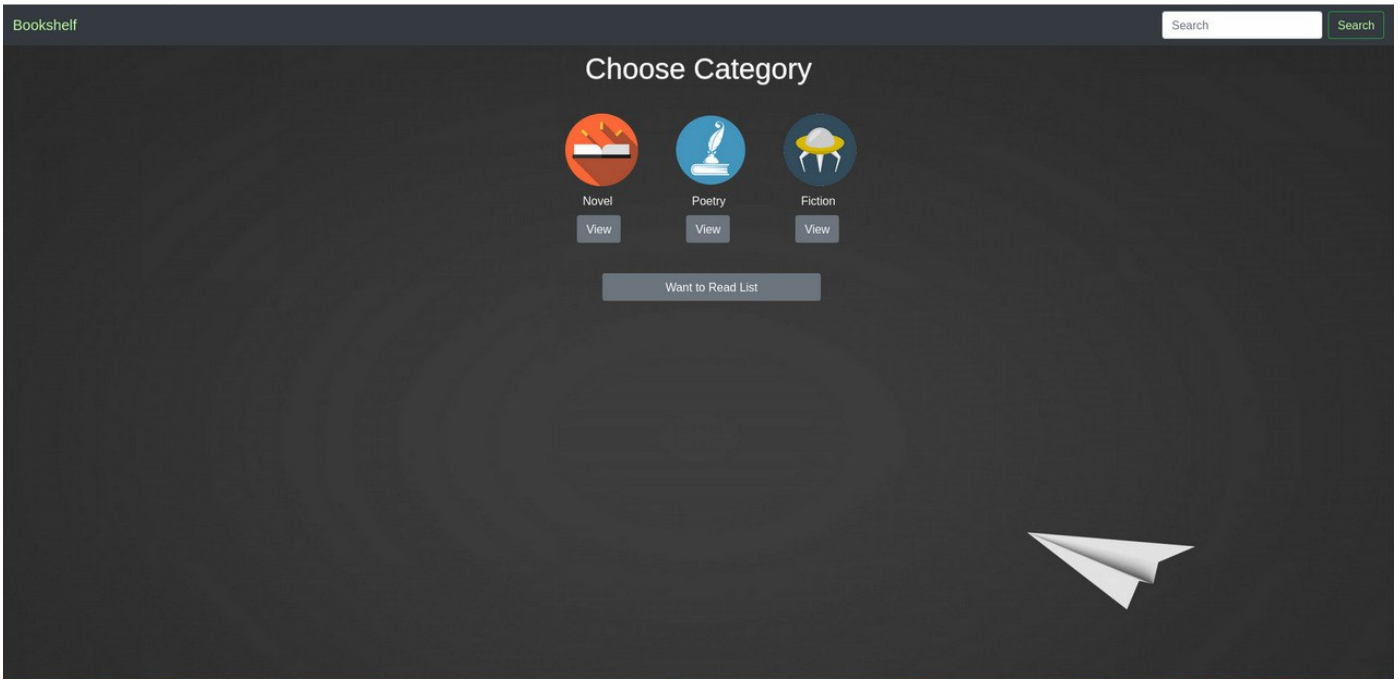

Рисунок 5.4 – Головна сторінка

Рисунок 5.5 – Вибір категорії

Рисунок 5.6 – Перехід до списку бажаного до прочитання

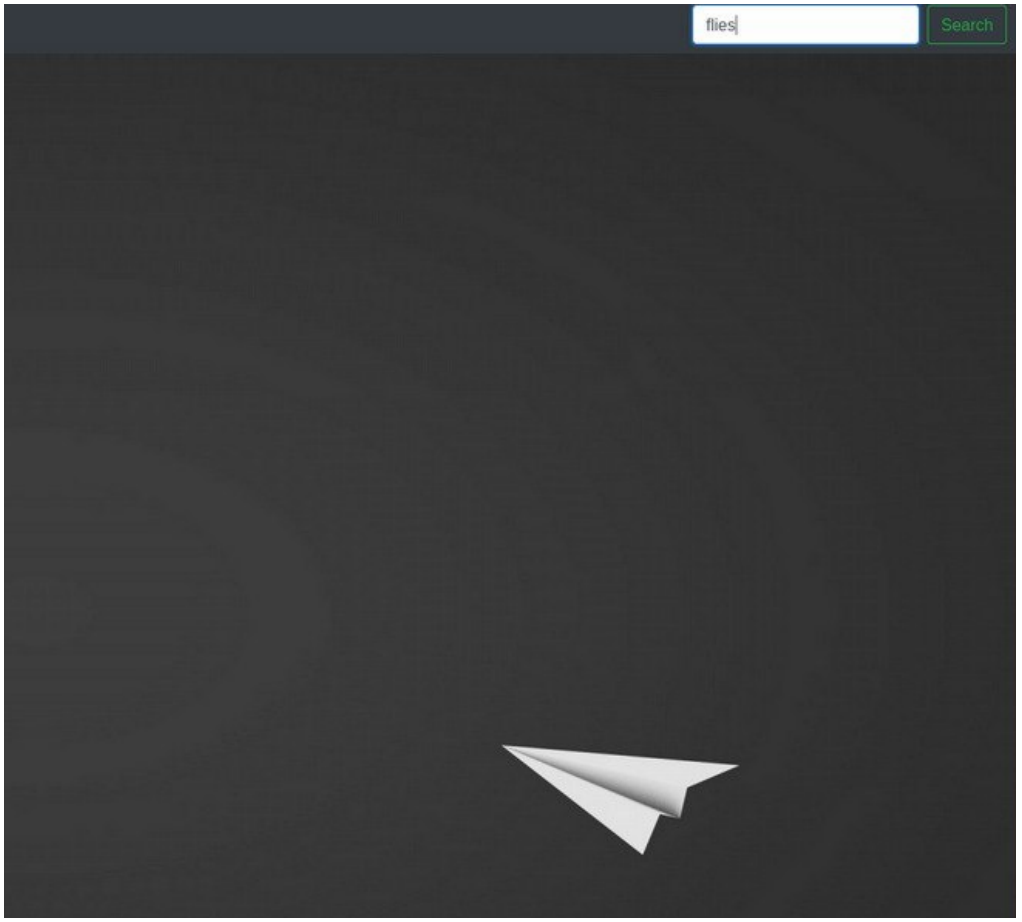

Рисунок 5.7 – Пошук книг

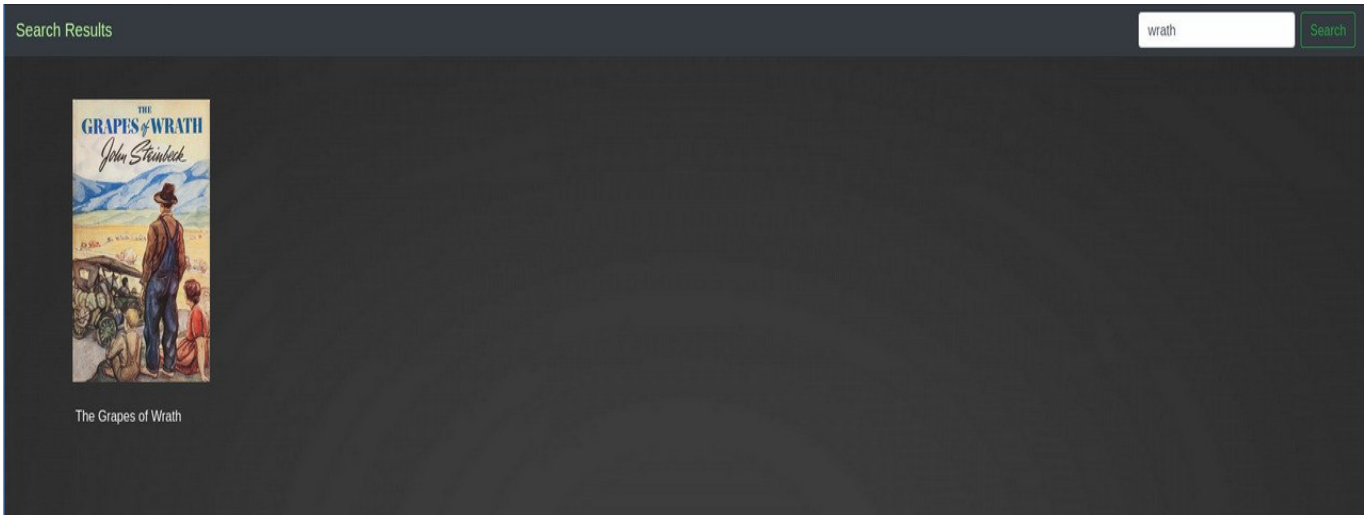

Рисунок 5.8 – Приклад пошуку

5.2.3 Сторінка Категорій

На сторінці певної категорії з літератури – користувач може побачити останню інформацію про книги (рисунок 5.9), що доступні на даному веб-порталі книжної полиці, натиснувши на кнопку “View” під назвою книги - можна перейти до літератури.

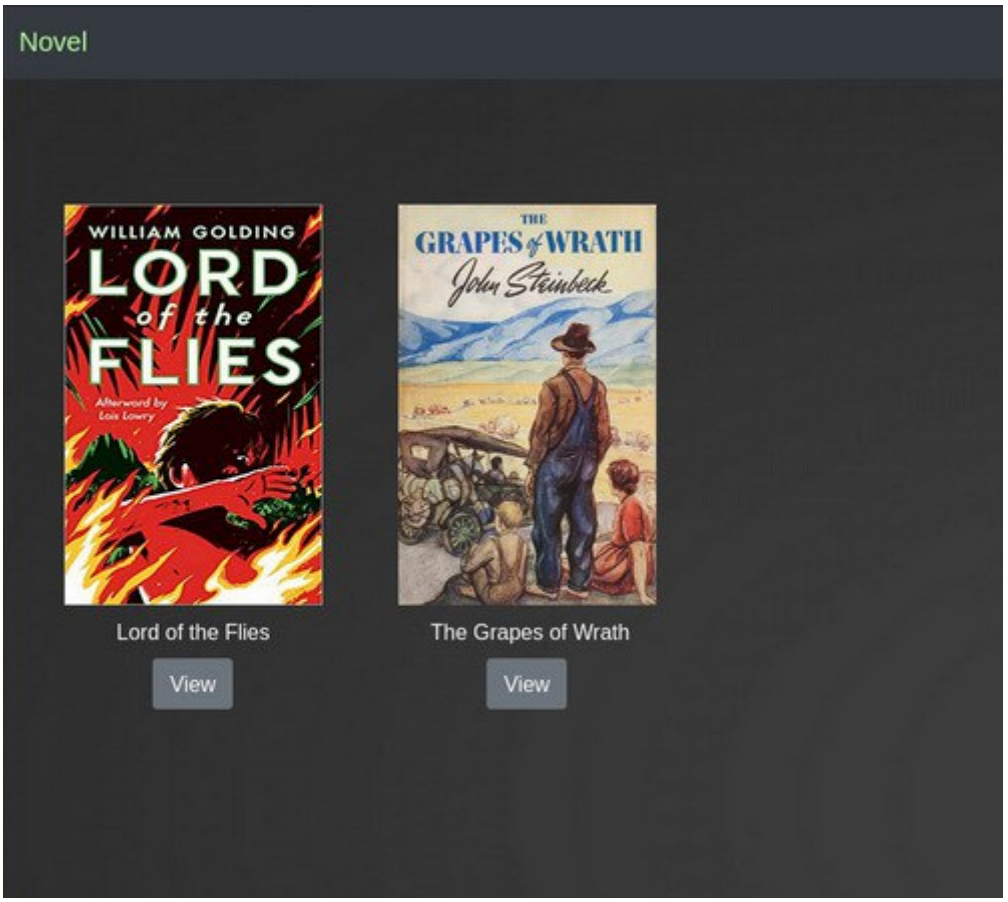

Рисунок 5.9 – Перелік літератури

5.2.4 Сторінка Книги

На сторінці певної твору з літератури – користувач може побачити останню інформацію про книгу (рисунок 5.10), що доступна на даному веб-порталі книжної полиці, переглянути відеоматеріал (рисунок 5.11) про сюжет даного твору та додати книгу до списку бажаного до прочитання користуючись кнопкою “Want to Read List” (рисунок 5.12).

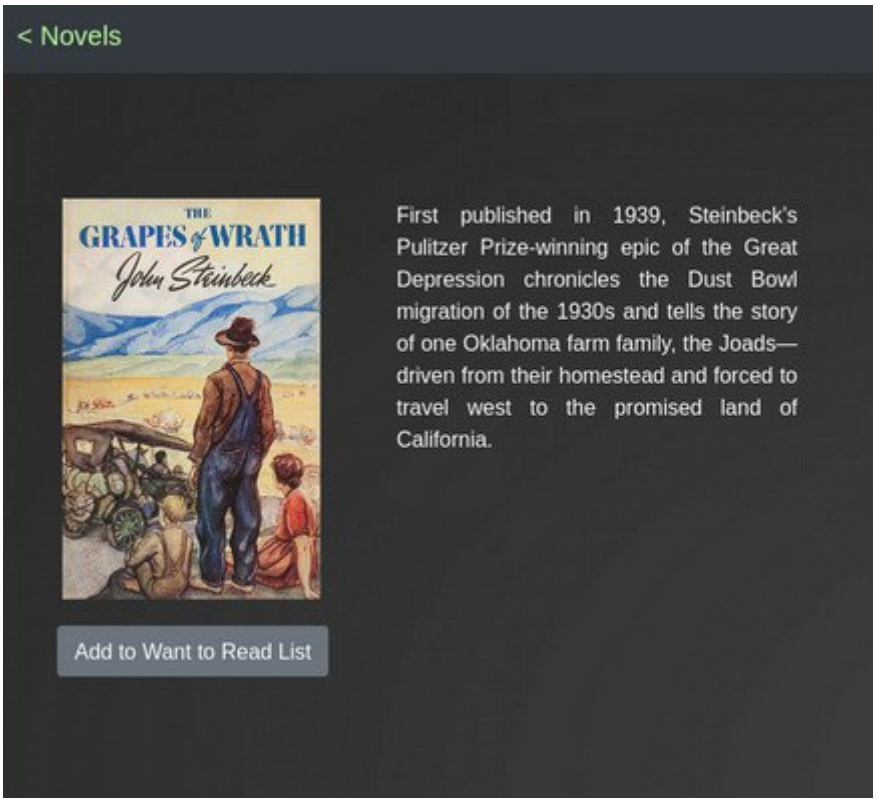

Рисунок 5.10 – Інформація про твір

Рисунок 5.11 – Відеоматеріал до літератури

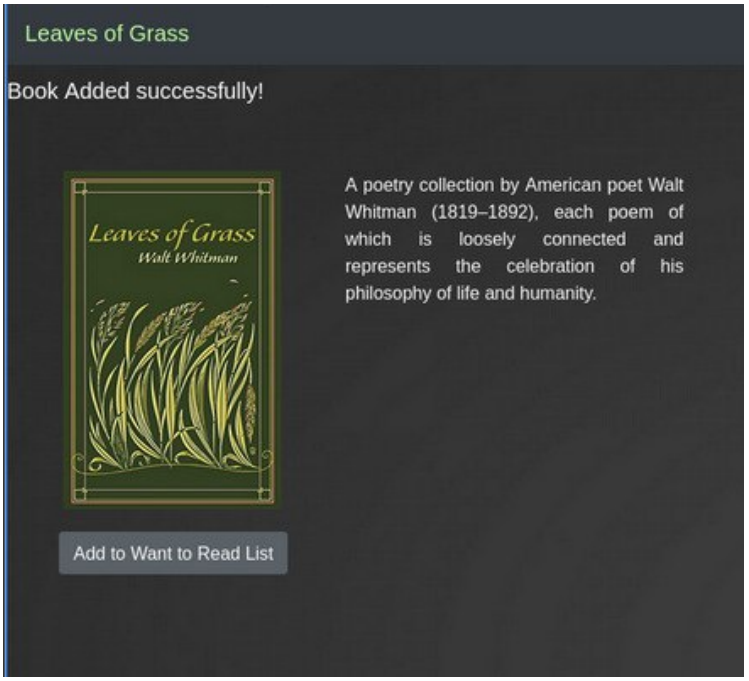

Рисунок 5.12 – Приклад додавання нової книги до списку

Рисунок 5.13 – Приклад додавання книги, що вже в списку

5.2.5 Адміністрування

На сторінці авторизації (рисунок 5.1), користувач може побачити поля для вводу даних про обліковий запис в клієнт-серверній системі веб-порталу книжкової полиці, в такий спосіб адміністратор ресурсу також потрапляє до головної сторінки з переліком категорій ввівши облікові дані адміністратора (рисунок 5.14).

Рисунок 5.14 – Авторизація адміністратора системи

Загалом, можливості адміністратора включають в себе всі функції, які може проводити звичайний користувач та не обмежуються на цьому. В руках адміністратора - створення категорій літератури, додавання творів та матеріалів до них (рисунок 5.15).

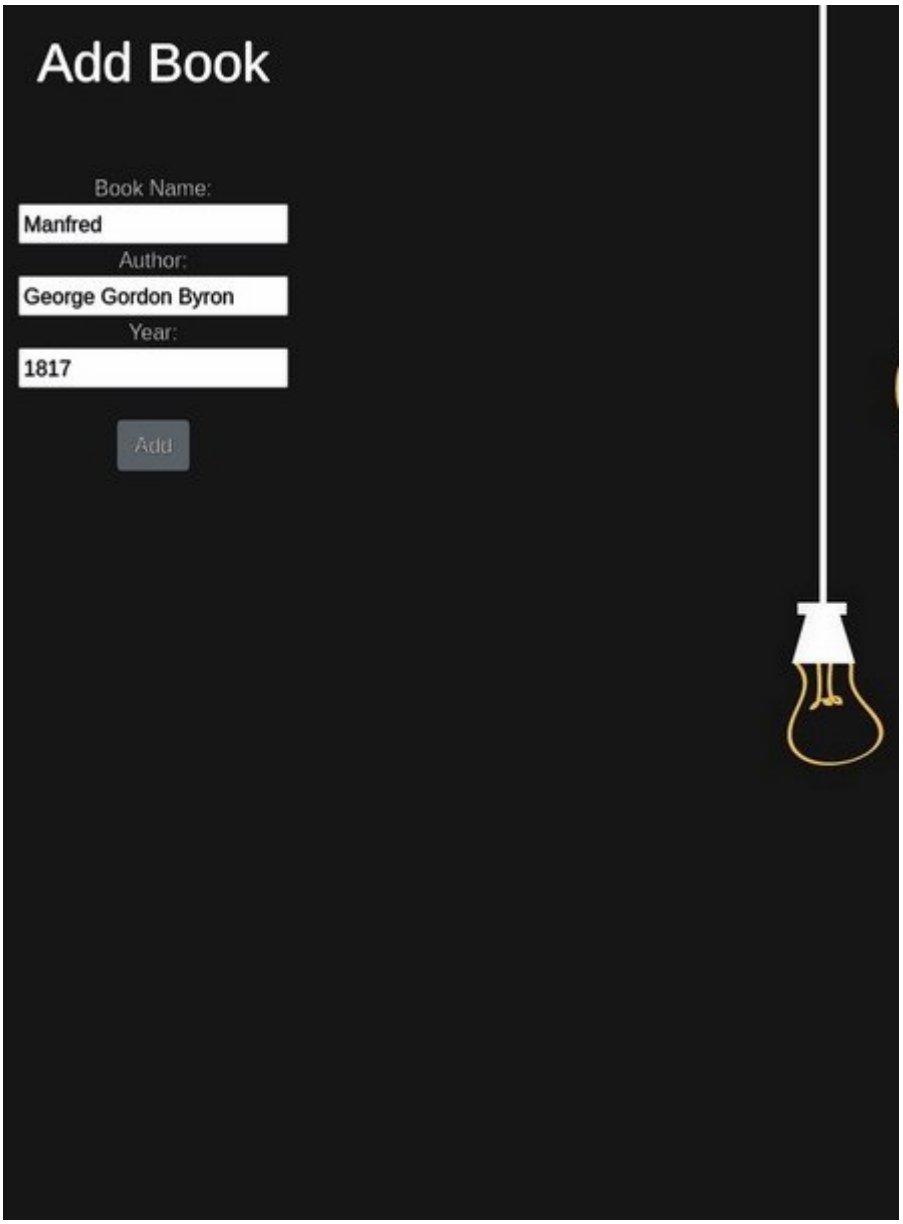

Рисунок 5.15 – Авторизація адміністратора системи

5.3 Сценарій роботи Інфраструктури

Клієнт-серверне рішення використовує стандартизовану мову розмітки документів для перегляду веб-сторінок у браузері HTML, формальну мова опису зовнішнього вигляду веб-сторінки CSS, мову сценаріїв для надання інтерактивності веб-сторінкам JavaScript та багатофункціональний набір інструментів інтерфейсу Bootstrap . Для побудови рішення та інтеграції до інфраструктури системи використано GitHub Actions.

На вході маємо файли клієнт-серверної системи, збережені в репозиторії, що використовують розробники (рисунок 5.16)

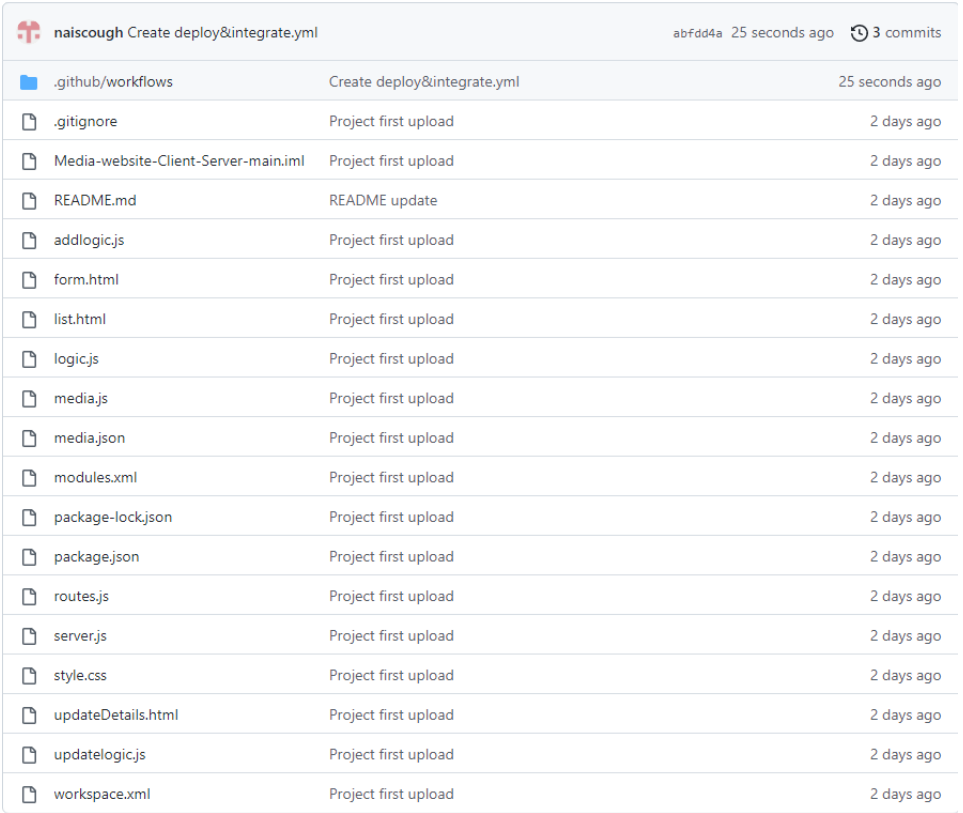

Рисунок 5.16 – Репозиторій з файлами клієнт серверної системи

Скрипт робочих процесів дозволяє налагоджувати залежності та підіймати рішення вже з інтегрованим до інфраструктури кодом при кожному коміті розробників до головної гілки репозиторія. Ці процеси відбуваються за допомогою віртуальних машин Git, що виконують приведені кроки (рисунок 5.17).

```
name: CI/CD


on:
  push:
    branches: [ master ]

jobs:
  build-and-deploy:
    name: Build and deploy
    runs-on: ubuntu-latest
    steps:
      - name: Checkout Code
        uses: actions/checkout@v2.3.1

      - name: Configure AWS Credentials
        uses: aws-actions/configure-aws-credentials@v1
        with:
          aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
          aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
          aws-region: us-east-2

      - name: Install Node.js
        uses: actions/setup-node@v1
        with:
          node-version: '12.x'

      - name: Install Project Dependencies
        run: npm install

      - name: Build project
        run: npm start

      - name: Install AWS CDK
        run: 'sudo npm install -g aws-cdk'

      - name: CDK Bootstrap
        run: yarn add @aws-cdk/aws-s3 @aws-cdk/aws-s3-deployment @aws-cdk/aws-cloudfront ; cdk bootstrap
        working-directory: cdk

      - name: CDK Synth
        run: cdk synth
        working-directory: cdk

      - name: CDK Deploy
        run: cdk deploy --require-approval never
        working-directory: cdk

      - name: Verify build
        run:  ls -la public
```

Рисунок 5.17 – Приклад робочих процесів, що виконує скрипт GitHub Actions

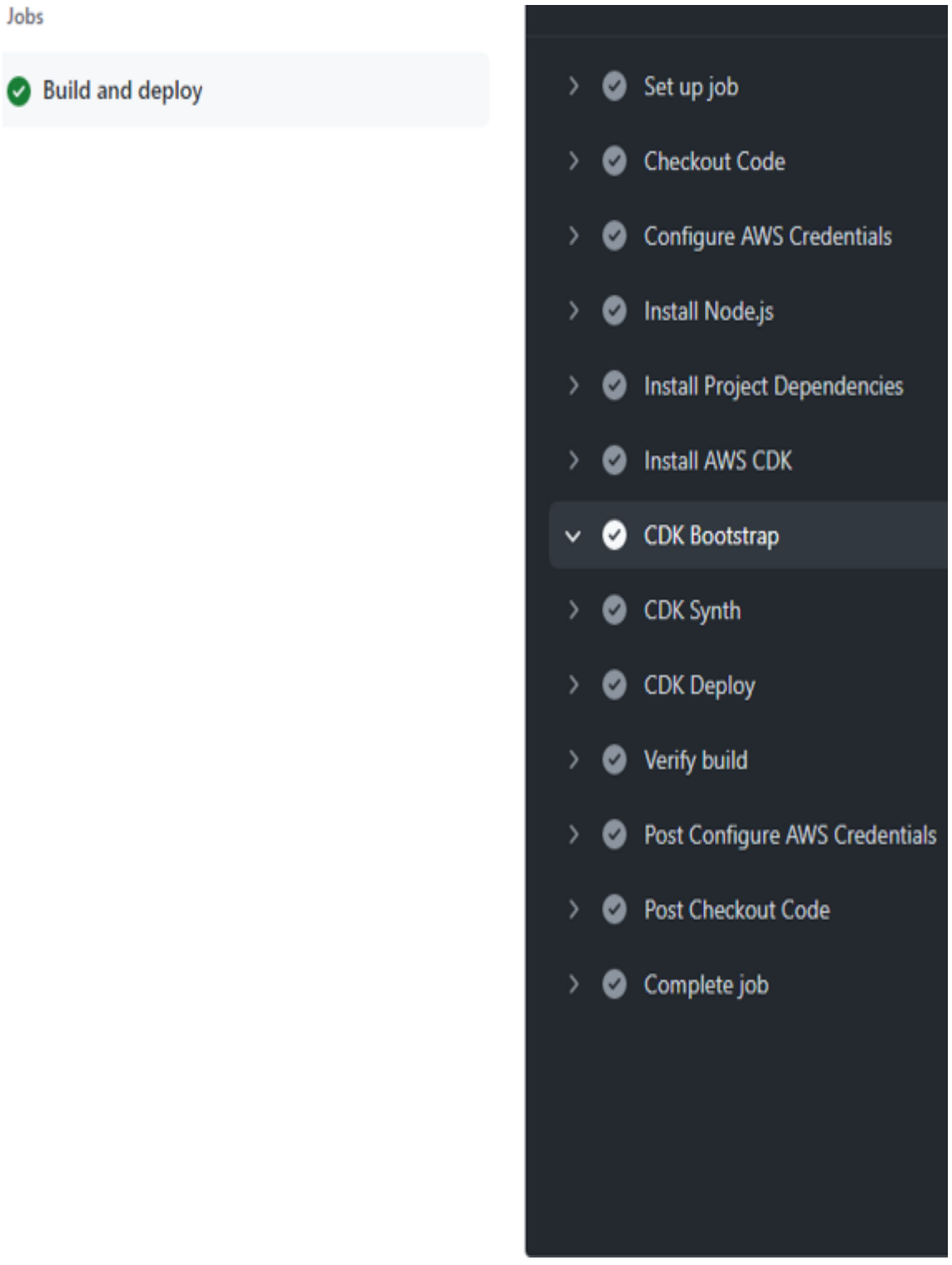

Рисунок 5.18 – Log виконаних робочих процесів GitHub Actions

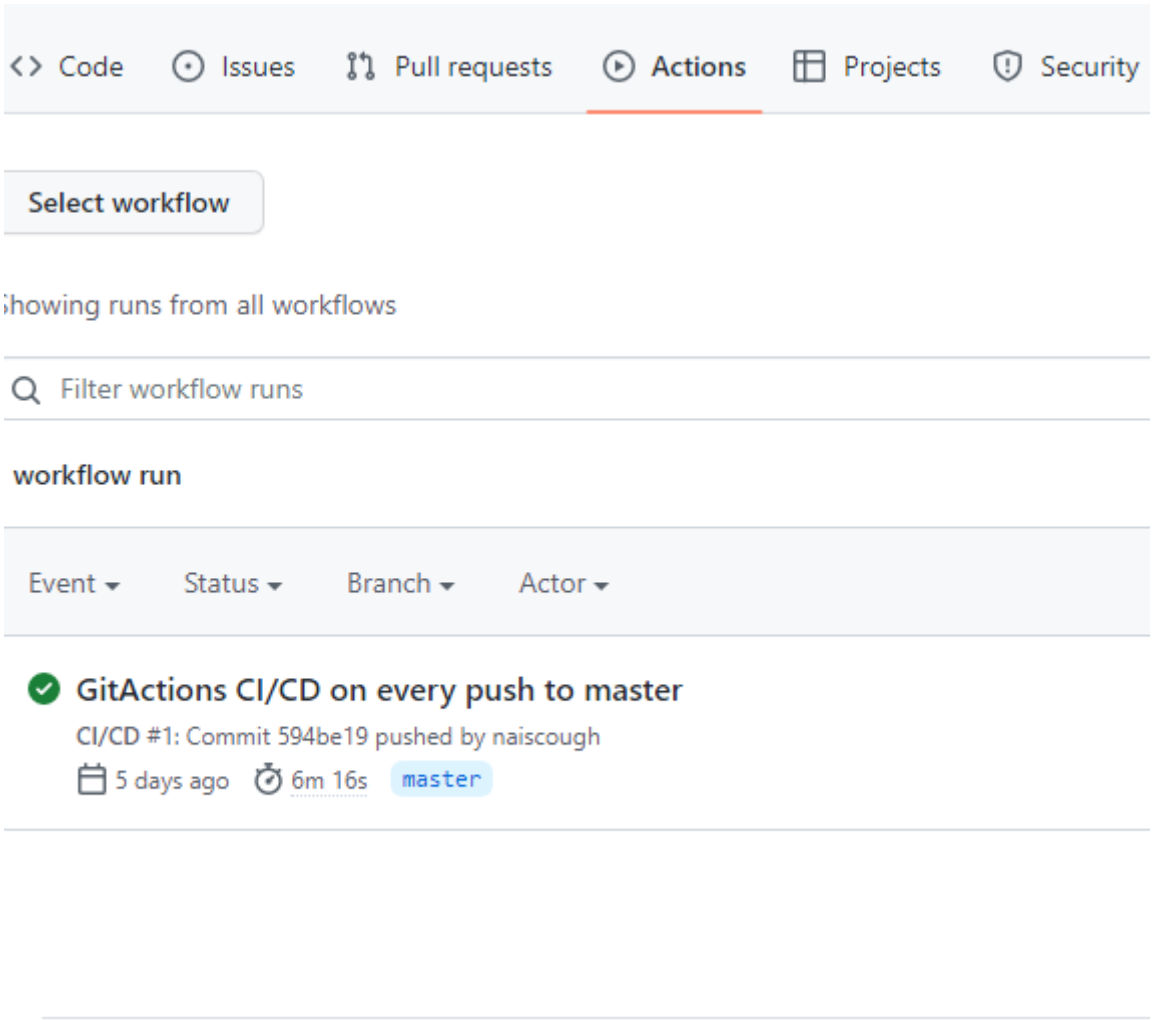

Рисунок 5.19 – Робочі процеси та стутус їх виконання

За допомогою маніфестів на базі Amazon CDK створюється інфраструктура через стек в CloudFormation (рисунок 5.20), що розгортає клієнт-серверну систему на сервері Amazon з використанням Amazon S3 та CloudFront.

Інтегровані до інфраструктури файли зберігаються в сховищі Amazon S3 (рисунок 5.21).

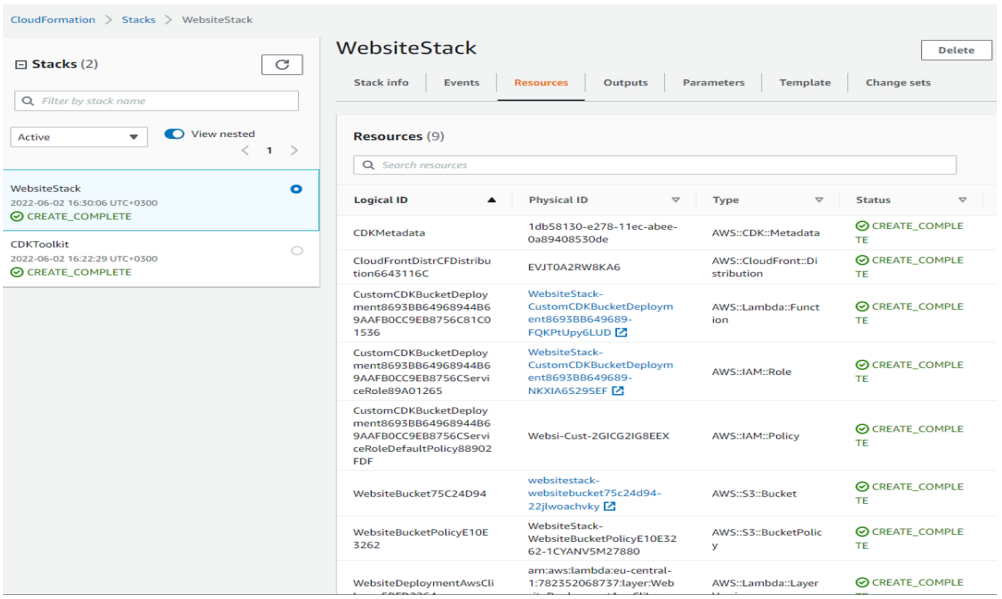

Рисунок 5.20 – Стек CloudFormation для розгортання клієнт-серверної системи

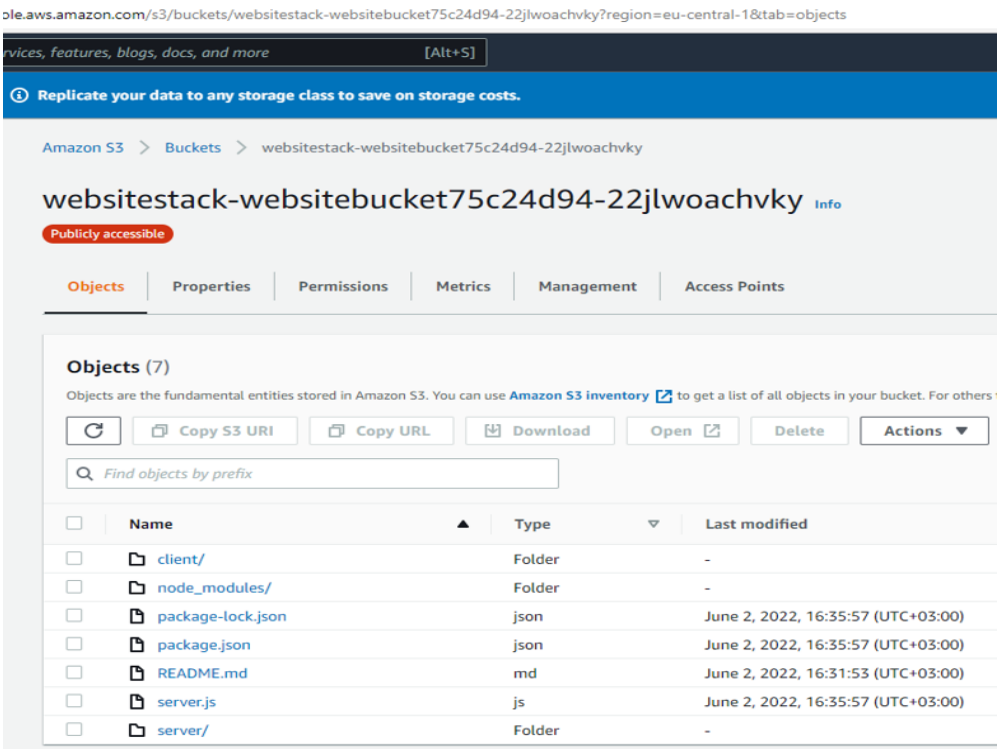

Рисунок 5.21 – файли розгорнутої системи у сховищі Amazon S3

Amazon Web Services забезпечують побудову інфраструктури та зберігання файлів розгорнутої системи (рисунок 4.2.2(6)). Процеси інтеграції коду до розгорнутої інфраструктури налагодженні, користуючись системою контролю версій Git та платформою для безперервної інтеграції (рисунок 4.2.2(7))та безперервної доставки коду GitHub Actions.

Доступ до файлів через Amazon S3 та CloudFront, що забезпечують постійний доступ до даних сайту, з будь-якої точки світу з мінімальною затримкою.

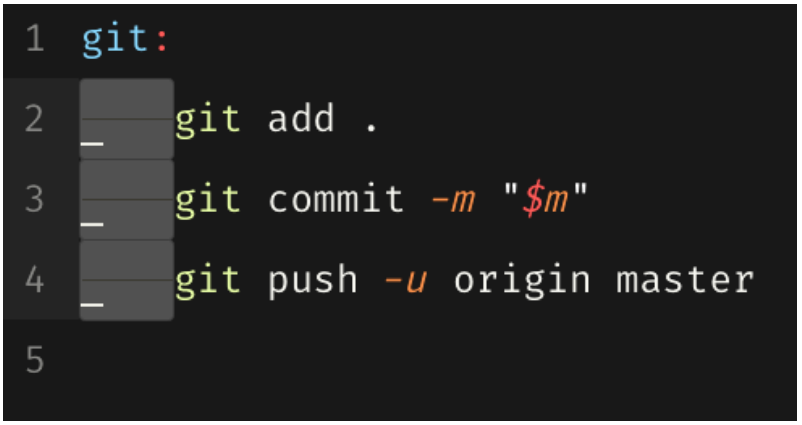

Рисунок 5.21 Приклад

Завдяки автоматизації процесів розгортання та інтеграції коду, розробникам достатньо лише завантажити вміст локального репозиторію у віддалений репозиторій, тобто передати коміти з локального репозиторію у віддалений.

ВИСНОВКИ

У результаті виконання дипломної роботи було створено клієнт-серверну реалізацію веб-порталу книжної полиці та побудовано стабільну інфраструктуру, налагоджено процеси розгортання системи та неперервної інтеграції коду. Розроблена система працює комплексно з усіма модулями клієнт-серверної системи і надає розробникам необхідні функції для впровадження нового функціоналу до програмного рішення.

Під час створення системи було проаналізовано та вивчено методику створення клієнт-серверних систем, методику розробки та впровадження інфраструктури. Були покращені навички розробки програмних рішень.

Для побудови системи було використано інструменти JavaScript та Node.js для клієнт-серверної системи книжкового веб порталу, сервіси Amazon, що дуже спростило процес розгортання інфраструктури, для неперервної інтеграції коду було використано платформу для автоматизації GitHub Actions.

Розроблений програмний продукт включає в себе наступний функціонал:

- розгортання інфраструктури клієнт-серверної системи;
- автоматична інтеграція коду до програмного продукту;
- безпечне зберігання даних на сервері;

Розроблену платформу можна використовувати для розробки програмних продуктів, а методи – впроваджені для будь-якою клієнт-серверної системи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Jennifer Davis, Ryn Daniels Effective DevOps: Building a Culture of Collaboration, Affinity, and Tooling.
   Gravenstein Highway North, Sebastopol : O'Reilly Media, 2015
2. Дюваль Поль М., Матиас Стивен Непрерывная интеграция. Улучшение качества программного обеспечения и снижение риска.
   Киев: Вильямс, 2008
3. Домингус Джастин, Арундел Джон Kubernetes для DevOps. Развертывание, запуск и масштабирование в облаке.
   Санкт-Петербург : Питер, 2019
4. Вехен Джульен Безопасный DevOps. Эффективная эксплуатация систем.
   Санкт-Петербург : Питер, 2020
5. What Is Cloud Computing?: веб-сайт. URL: https://www.salesforce.com/ca/cloud-computing/ (дата звернення:10.02.2022).
6. Документація Visual Studio Code: веб-сайт. URL: https://code.visualstudio.com/docs (дата звернення: 17.02.2022).
7. Amazon website: веб-сайт. URL: https://aws.amazon.com/blogs/networking-and-content-delivery/amazon-s3-amazon-cloudfront-a-match-made-in-the-cloud/ (дата звернення: 13.04.2022).
8. JavaScript: веб-сайт. URL: https://developer.mozilla.org/ru/docs/Web/JavaScript (дата звернення: 8.03.2022).
9. Документація Node.js: веб-сайт. URL: https://nodejs.org/en/docs/ (дата звернення: 13.02.2022).
10. Документація HTML: веб-сайт. URL: https://developer.mozilla.org/ru/docs/Web/HTML (дата звернення: 17.03.2022).

11. Документація CSS: веб-сайт. https://developer.mozilla.org/ru/docs/Web/CSS (дата звернення: 17.03.2022).
12. Web portal: публікація URL: https://www.liferay.com/resources/l/web-portal (дата звернення: 3.03.2022).
13. Документація GitHub Actions веб-сайт. URL: https://docs.github.com/en/actions/learn-github-actions/essential-features-of-github-actions (дата звернення: 20.03.2022).
14. Google Cloud Platform: веб-сайт. URL: https://cloud.google.com/ (дата звернення: 8.03.2022).
15. Microsoft Azure: веб-сайт. URL: https://azure.microsoft.com/ (дата звернення: 8.03.2022).
16. Amazon Web Services: веб-сайт. URL: https://aws.amazon.com/ (дата звернення: 8.03.2022).
17. Bootstrap: веб-сайт. URL: https://getbootstrap.com/ (дата звернення: 8.03.2022).
18. CI/CD: стаття URL: https://habr.com/ru/company/otus/blog/515078/ (дата звернення: 5.02.2022).

ДОДАТОК А

Автоматизація процесів розгортання та неперервної інтеграції клієнт-серверної системи

Лістинг програми

УКР.НТУУ”КПІ імені Ігоря Сікорського”_ТЕФ_АПЕПС_ТР-82

Аркушів 26

Київ 2022

app.js:

```
var express = require('express');
var path = require('path');
var fs = require('fs');

var app = express();



// view engine setup
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'ejs');



app.use(express.json());
app.use(express.urlencoded({ extended: false }));
app.use(express.static(path.join(__dirname, 'public')));

const session = require('express-session');
app.use(session({
  secret:'secret-key',
  resave: false,
  saveUninitialized: false
}))



app.post('',function(req,res){
  var user = req.body;
  var allUsers = fs.readFileSync("users.json");
  allUsers = JSON.parse(allUsers);
```

```
  var flag = false;
  for(i in allUsers)
  {

    if(user.username==allUsers[i].username && user.password == allUsers[i].password)
    {
      flag=true;
      req.session.username=user.username;
      res.redirect("/home");
    }
  }
  if(!flag)
    res.render("login",{error:"Wrong username or password. Please try again!"});

}
)

app.get('/', function(req,res){

  res.render("login",{error:""});

});

app.get('/home', function(req,res){
  if(req.session.username)
   res.render('home')
   else
   res.redirect("/");
});
```

```
app.get('/dune', function(req,res){
  if(req.session.username)
  res.render('dune',{message:""})
  else
   res.redirect("/");
});

app.get('/fiction', function(req,res){
  if(req.session.username)
  res.render('fiction',{message:""})
  else
   res.redirect("/");
});

app.get('/flies', function(req,res){
  if(req.session.username)
  res.render('flies',{message:""})
  else
   res.redirect("/");
});

app.get('/grapes', function(req,res){
  if(req.session.username)
  res.render('grapes',{message:""})
  else
   res.redirect("/");
});
```

```
app.get('/leaves', function(req,res){
  if(req.session.username)
  res.render('leaves',{message:""})
  else
   res.redirect("/");
});

app.get('/mockingbird', function(req,res){
  if(req.session.username)
  res.render('mockingbird',{message:""})
  else
   res.redirect("/");
});

app.get('/novel', function(req,res){
  if(req.session.username)
  res.render('novel')
  else
   res.redirect("/");
});

app.get('/poetry', function(req,res){
  if(req.session.username)
  res.render('poetry')
  else
   res.redirect("/");
});

app.get('/readlist', function(req,res){
```

```
  if(req.session.username)
  {
    var userReadList=[];
    var allUsers = fs.readFileSync("users.json");
  allUsers = JSON.parse(allUsers);
  for(u in allUsers)
  {
    if(req.session.username. localeCompare(allUsers[u].username)==0)
    {
     userReadList=allUsers[u].readList;

    }
  }
    res.render('readlist', {list:userReadList})
  }
  else
   res.redirect("/");
});

app.get('/registration', function(req,res){
  res.render('registration',{error:""})
});

app.get('/searchresults', function(req,res){
  if(req.session.username)
  res.render('searchresults')
  else
  res.redirect("/");
});
```

```
app.get('/sun', function(req,res){
  if(req.session.username)
  res.render('sun',{message:""})
  else
   res.redirect("/");
});

app.post('/register',function(req,res){

  var newUser = req.body;
  newUser.readList=[];
  var allUsers = fs.readFileSync("users.json");
  if(allUsers.length == 0)
  {
    var s = JSON.stringify(newUser);
    fs.writeFileSync('users.json',s);
    return;
  }
  allUsers = JSON.parse(allUsers);
  var oldUser = alreadyRegistered(allUsers,newUser);
  for(u in allUsers)
  {
    if(newUser.username==allUsers[u].username)
    {
      oldUser= true;
    }
  }
  if(oldUser)
```

```
    res.render("registration",{error:"Username already in use. Please Enter another
one"});
  else
  {
    allUsers.push(newUser);
    var s = JSON.stringify(allUsers);
    fs.writeFileSync('users.json',s);
    res.redirect("/");
  }
}
)

app.post('/search',function(req,res){

  res.render("searchresults",{searchValue:req.body.Search});
    }

)

app.post('/flies',function(req,res){

  var alreadyExists = addToList("Lord of the Flies",req);
  if(alreadyExists)
    res.render("flies",{message:"The book is already in your readlist!"});
  else
    res.render("flies",{message:"Book Added successfully!"});
    }
)
```

```
app.post('/grapes',function(req,res){

  var alreadyExists = addToList("The Grapes of Wrath",req);
  if(alreadyExists)
    res.render("grapes",{message:"The book is already in your readlist!"});
  else
    res.render("grapes",{message:"Book Added successfully!"});
    }

)
app.post('/leaves',function(req,res){

  var alreadyExists = addToList("Leaves of Grass",req);

  if(alreadyExists)
    res.render("leaves",{message:"The book is already in your readlist!"});
  else
    res.render("leaves",{message:"Book Added successfully!"});
    }

)

app.post('/sun',function(req,res){

  var alreadyExists = addToList("The Sun and Her Flowers",req);
  if(alreadyExists)
    res.render("sun",{message:"The book is already in your readlist!"});
  else
    res.render("sun",{message:"Book Added successfully!"});
```

```
    }

)

app.post('/dune',function(req,res){

  var alreadyExists = addToList("Dune",req);
  if(alreadyExists)
    res.render("dune",{message:"The book is already in your readlist!"});
  else
    res.render("dune",{message:"Book Added successfully!"});
    }

)

app.post('/mockingbird',function(req,res){

  var alreadyExists = addToList("To Kill a Mockingbird",req);
  if(alreadyExists)
    res.render("mockingbird",{message:"The book is already in your readlist!"});
  else
    res.render("mockingbird",{message:"Book Added successfully!"});
    }

)
function addToList(bookName,req){

  var alreadyExists=false;
  var allUsers = fs.readFileSync("users.json");
```

```
  allUsers = JSON.parse(allUsers);
  for(u in allUsers)
  {
    if((req.session.username). localeCompare(allUsers[u].username)==0)
    {
    if(!allUsers[u].readList.includes(bookName))
      allUsers[u].readList.push(bookName);
    else
      alreadyExists=true;
     var s = JSON.stringify(allUsers);
     fs.writeFileSync('users.json',s);
     return alreadyExists;
    }
  }



}


function alreadyRegistered(allUsers,newUser)
{
  for(u in allUsers)
  {
    if(newUser.username==allUsers[u].username)
    {
      return true;
    }
  }
  return false;
```

```
}


app.listen(process.env.PORT || 3000);
```

home.ejs:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <style>
        body {
            background-image: url("background3.jpg");
            background-repeat: no-repeat;
            background-size: 100%;
        }
        h1 {
            color: white;
            text-align: center;
            -webkit-text-stroke: 1px blue;
        }
        .cont {
            position: relative;
            left: 600px;
            top: 50px;
            color: white;
            -webkit-text-stroke: 0.5px black;
        }
        .container1 {
            position: fixed;
            left: 50px;
            top: 150px;
```

```
            color: white;
        }
        .container2 {
            position: fixed;
            left: 200px;
            top: 150px;
            color: white;
        }
        .container3 {
            position: fixed;
            left: 350px;
            top: 150px;
            color: white;
        }
        .readlist {
            position: fixed;
            left: 80px;
            top: 370px;
        }
    </style>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Books</title>
                                                    <link              rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css"

integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T"
crossorigin="anonymous">
</head>
```

```
<body>
    <nav class="navbar navbar-expand-lg navbar-dark bg-dark mb-2">
        <a class="navbar-brand" style="color: rgb(191, 255, 161)">Books</a>
                  <button  class="navbar-toggler"  type="button"  data-toggle="collapse"
data-target="#navbarSupportedContent"
                           aria-controls="navbarSupportedContent"  aria-expanded="false"
aria-label="Toggle navigation">
            <span class="navbar-toggler-icon"></span>
        </button>
        <div class="collapse navbar-collapse" id="navbarSupportedContent">
            <ul class="navbar-nav mr-auto">
            </ul>
            <form method="POST" action='/search' class="form-inline my-2 my-lg-0">
                       <input  class="form-control  mr-sm-2"  name="Search"  type="search"
placeholder="Search"
                   aria-label="Search">
                                <button  class="btn  btn-outline-success  my-2  my-sm-0"
type="submit">Search</button>
            </form>
        </div>
    </nav>

    <div class="container">
        <h1>Welcome</h1>
    </div>
    <div class="container1">
        <img id="image" src="/novel.png" width="100" height="100">
        <br>
        <label class="ml-4 my-2">Novel</label>
```

```
        <br>
         <button onclick="location.href = '/novel';" id="novel" class="btn btn-secondary
ml-3"> View </button>
    </div>
    <div class="container2">
        <img id="image" src="/poetry.png" width="100" height="100">
        <br>
        <label class="ml-4 my-2">Poetry</label>
        <br>
                <button  onclick="location.href  =  '/poetry';"  id="poetry"  class="btn
btn-secondary ml-3"> View </button>
    </div>
    <div class="container3">
        <img id="image" src="/fiction.png" width="100" height="100">
        <br>
        <label class="ml-4 my-2">Fiction</label>
        <br>
               <button  onclick="location.href  =  '/fiction';"  id="fiction"  class="btn
btn-secondary ml-3"> View </button>
    </div>
    <div class="readlist">
              <button  onclick="location.href  =  '/readlist';"  id="readlist"  class="btn
btn-secondary ml-3" style="width: 300px;">Want to Read List</button>
    </div>
</body>
</html>
```

cdk.ts:

```
import { Stack, StackProps } from 'aws-cdk-lib';
```

```
import { CoreDnsComputeType } from 'aws-cdk-lib/aws-eks';
import { S3 } from 'aws-cdk-lib/aws-ses-actions';
import { Construct } from 'constructs';
import { aws_s3 as s3} from 'aws-cdk-lib';
import { aws_s3_deployment as s3Deployment} from 'aws-cdk-lib';
import { aws_cloudfront as cloudfront} from 'aws-cdk-lib';


export class CdkStack extends Stack {
  constructor(scope: Construct, id: string, props?: StackProps) {
    super(scope, id, props);

    // S3 Bucket ініціалізація
    const bucket = new s3.Bucket(this, "WebsiteBucket", {
      publicReadAccess: true,
      websiteIndexDocument: "list.html"
    });

    // Deployment розгортання сховища

    new s3Deployment.BucketDeployment(this, "WebsiteDeployment", {
      sources: [s3Deployment.Source.asset("../build")],
      destinationBucket: bucket
    });



    // CloudFront Ініціалізація веб-сервісу для доставки контенту
    new cloudfront.CloudFrontWebDistribution(this, 'CloudFrontDistr', {
      originConfigs: [
```

```
        {
          s3OriginSource: {
            s3BucketSource: bucket
          },
          behaviors: [ {isDefaultBehavior: true}]
        }
      ]
    });
  }
}
```

cdk.ts:

```
#!/usr/bin/env node
import 'source-map-support/register';
import * as cdk from 'aws-cdk-lib';
import { CdkStack } from '../lib/cdk-stack';
import { access } from 'fs';


const app = new cdk.App();
new CdkStack(app, 'WebsiteStack', {
  env: {
    region: "eu-central-1"
```

```
  }
});
```

deploy.yml:

```
name: CI/CD



on:
  push:
    branches: [ master ]

jobs:
  build-and-deploy:
    name: Build and deploy
    runs-on: ubuntu-latest
    steps:
      - name: Checkout Code
        uses: actions/checkout@v2.3.1


      - name: Configure AWS Credentials
        uses: aws-actions/configure-aws-credentials@v1
        with:
          aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
          aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
          aws-region: us-east-2


      - name: Install Node.js
        uses: actions/setup-node@v1
```

```
        with:
          node-version: '12.x'

      - name: Install Project Dependencies
        run: npm install

      - name: Build project
        run: npm start

      - name: Install AWS CDK
        run: 'sudo npm install -g aws-cdk'

      - name: CDK Bootstrap
        run: yarn add @aws-cdk/aws-s3 @aws-cdk/aws-s3-deployment @aws-cdk/aws-cloudfront
; cdk bootstrap
        working-directory: cdk

      - name: CDK Synth
        run: cdk synth
        working-directory: cdk

      - name: CDK Deploy
        run: cdk deploy --require-approval never
        working-directory: cdk

      - name: Verify build
        run:  ls -la public
```

cdk.json:

```
{
  "app": "npx ts-node --prefer-ts-exts bin/cdk.ts",
  "watch": {
    "include": [
      "**"
    ],
    "exclude": [
      "README.md",
      "cdk*.json",
      "**/*.d.ts",
      "**/*.js",
      "tsconfig.json",
      "package*.json",
      "yarn.lock",
      "node_modules",
      "test"
    ]
  },
  "context": {
    "@aws-cdk/aws-apigateway:usagePlanKeyOrderInsensitiveId": true,
    "@aws-cdk/core:stackRelativeExports": true,
    "@aws-cdk/aws-rds:lowercaseDbIdentifier": true,
    "@aws-cdk/aws-lambda:recognizeVersionProps": true,
    "@aws-cdk/aws-cloudfront:defaultSecurityPolicyTLSv1.2_2021": true,
    "@aws-cdk-containers/ecs-service-extensions:enableDefaultLogDriver": true,
    "@aws-cdk/aws-ec2:uniqueImdsv2TemplateName": true,
    "@aws-cdk/core:checkSecretUsage": true,
```

```
    "@aws-cdk/aws-iam:minimizePolicies": true,
    "@aws-cdk/core:target-partitions": [
      "aws",
      "aws-cn"
    ]
  }
}
```

package.json:

```
{
  "name": "cdk",
  "version": "0.1.0",
  "bin": {
    "cdk": "bin/cdk.js"
  },
  "scripts": {
    "build": "tsc",
    "watch": "tsc -w",
    "test": "jest",
    "cdk": "cdk"
  },
  "devDependencies": {
    "@types/jest": "^27.5.0",
    "@types/node": "10.17.27",
    "@types/prettier": "2.6.0",
    "aws-cdk": "2.26.0",
    "jest": "^27.5.1",
```

```
    "ts-jest": "^27.1.4",
    "ts-node": "^10.7.0",
    "typescript": "~3.9.7"
  },
  "dependencies": {
    "@aws-cdk/aws-cloudfront": "^1.158.0",
    "@aws-cdk/aws-s3": "^1.158.0",
    "@aws-cdk/aws-s3-deployment": "^1.158.0",
    "aws-cdk-lib": "2.26.0",
    "constructs": "^10.0.0",
    "source-map-support": "^0.5.21"
  }
}
```

tsconfig.json:

```
{
  "compilerOptions": {
    "target": "ES2018",
    "module": "commonjs",
    "lib": [
      "es2018"
    ],
    "declaration": true,
    "strict": true,
    "noImplicitAny": true,
    "strictNullChecks": true,
    "noImplicitThis": true,
    "alwaysStrict": true,
    "noUnusedLocals": false,
```

```
    "noUnusedParameters": false,
    "noImplicitReturns": true,
    "noFallthroughCasesInSwitch": false,
    "inlineSourceMap": true,
    "inlineSources": true,
    "experimentalDecorators": true,
    "strictPropertyInitialization": false,
    "typeRoots": [
      "./node_modules/@types"
    ]
  },
  "exclude": [
    "node_modules",
    "cdk.out"
  ]
}
```

CdkStack.assets.json:

```
{
  "version": "20.0.0",
  "files": {
    "8ad7bbf8be94e05d569da95ddb82511dcc959f25054825394cbb86028ccd1b6a": {
      "source": {
                                                                    "path":
"asset.8ad7bbf8be94e05d569da95ddb82511dcc959f25054825394cbb86028ccd1b6a.zip",
        "packaging": "file"
      },
      "destinations": {
```

```
        "current_account-current_region": {
          "bucketName": "cdk-hnb659fds-assets-${AWS::AccountId}-${AWS::Region}",
                                                                    "objectKey":
"8ad7bbf8be94e05d569da95ddb82511dcc959f25054825394cbb86028ccd1b6a.zip",
                                                                "assumeRoleArn":
"arn:${AWS::Partition}:iam::${AWS::AccountId}:role/cdk-hnb659fds-file-publishing-role-${
AWS::AccountId}-${AWS::Region}"
        }
      }
    },
    "f98b78092dcdd31f5e6d47489beb5f804d4835ef86a8085d0a2053cb9ae711da": {
      "source": {
                                                                "path":
"asset.f98b78092dcdd31f5e6d47489beb5f804d4835ef86a8085d0a2053cb9ae711da",
        "packaging": "zip"
      },
      "destinations": {
        "current_account-current_region": {
          "bucketName": "cdk-hnb659fds-assets-${AWS::AccountId}-${AWS::Region}",
                                                                    "objectKey":
"f98b78092dcdd31f5e6d47489beb5f804d4835ef86a8085d0a2053cb9ae711da.zip",
                                                                "assumeRoleArn":
"arn:${AWS::Partition}:iam::${AWS::AccountId}:role/cdk-hnb659fds-file-publishing-role-${
AWS::AccountId}-${AWS::Region}"
        }
      }
    },
    "0aed4c48289958138c9e30ff59a79a260f8448f4763f7ebc82502750baf74801": {
      "source": {
                                                                "path":
"asset.0aed4c48289958138c9e30ff59a79a260f8448f4763f7ebc82502750baf74801",
        "packaging": "zip"
```

```
      },
      "destinations": {
        "current_account-current_region": {
          "bucketName": "cdk-hnb659fds-assets-${AWS::AccountId}-${AWS::Region}",
                                                                    "objectKey":
"0aed4c48289958138c9e30ff59a79a260f8448f4763f7ebc82502750baf74801.zip",
                                                                "assumeRoleArn":
"arn:${AWS::Partition}:iam::${AWS::AccountId}:role/cdk-hnb659fds-file-publishing-role-${
AWS::AccountId}-${AWS::Region}"
        }
      }
    },
    "cb082f78880d0eeb3761d95ff5ea6e371772c1aea06c61c2afd01d048206308d": {
      "source": {
        "path": "CdkStack.template.json",
        "packaging": "file"
      },
      "destinations": {
        "current_account-current_region": {
          "bucketName": "cdk-hnb659fds-assets-${AWS::AccountId}-${AWS::Region}",
                                                                    "objectKey":
"cb082f78880d0eeb3761d95ff5ea6e371772c1aea06c61c2afd01d048206308d.json",
                                                                "assumeRoleArn":
"arn:${AWS::Partition}:iam::${AWS::AccountId}:role/cdk-hnb659fds-file-publishing-role-${
AWS::AccountId}-${AWS::Region}"
        }
      }
    }
  },
  "dockerImages": {}
```

```
}
```


WebsiteStack.assets.json:


```
{
  "version": "20.0.0",
  "files": {
    "8ad7bbf8be94e05d569da95ddb82511dcc959f25054825394cbb86028ccd1b6a": {
      "source": {
                                                                    "path":
"asset.8ad7bbf8be94e05d569da95ddb82511dcc959f25054825394cbb86028ccd1b6a.zip",
        "packaging": "file"
      },
      "destinations": {
        "current_account-eu-central-1": {
          "bucketName": "cdk-hnb659fds-assets-${AWS::AccountId}-eu-central-1",
                                                                "objectKey":
"8ad7bbf8be94e05d569da95ddb82511dcc959f25054825394cbb86028ccd1b6a.zip",
          "region": "eu-central-1",
                                                              "assumeRoleArn":
"arn:${AWS::Partition}:iam::${AWS::AccountId}:role/cdk-hnb659fds-file-publishing-role-${
AWS::AccountId}-eu-central-1"
        }
      }
    },
    "f98b78092dcdd31f5e6d47489beb5f804d4835ef86a8085d0a2053cb9ae711da": {
      "source": {
                                                                    "path":
"asset.f98b78092dcdd31f5e6d47489beb5f804d4835ef86a8085d0a2053cb9ae711da",
        "packaging": "zip"
      },
```

```
      "destinations": {
        "current_account-eu-central-1": {
          "bucketName": "cdk-hnb659fds-assets-${AWS::AccountId}-eu-central-1",
                                                                    "objectKey":
"f98b78092dcdd31f5e6d47489beb5f804d4835ef86a8085d0a2053cb9ae711da.zip",
          "region": "eu-central-1",
                                                                "assumeRoleArn":
"arn:${AWS::Partition}:iam::${AWS::AccountId}:role/cdk-hnb659fds-file-publishing-role-${
AWS::AccountId}-eu-central-1"
        }
      }
    },
    "0aed4c48289958138c9e30ff59a79a260f8448f4763f7ebc82502750baf74801": {
      "source": {
                                                                        "path":
"asset.0aed4c48289958138c9e30ff59a79a260f8448f4763f7ebc82502750baf74801",
        "packaging": "zip"
      },
      "destinations": {
        "current_account-eu-central-1": {
          "bucketName": "cdk-hnb659fds-assets-${AWS::AccountId}-eu-central-1",
                                                                    "objectKey":
"0aed4c48289958138c9e30ff59a79a260f8448f4763f7ebc82502750baf74801.zip",
          "region": "eu-central-1",
                                                                "assumeRoleArn":
"arn:${AWS::Partition}:iam::${AWS::AccountId}:role/cdk-hnb659fds-file-publishing-role-${
AWS::AccountId}-eu-central-1"
        }
      }
    },
    "045288f3f9bbb83accd1dd75916a88ce95aea3b3f232eae4af7e489e7d1b8a17": {
```

```
      "source": {
        "path": "WebsiteStack.template.json",
        "packaging": "file"
      },
      "destinations": {
        "current_account-eu-central-1": {
          "bucketName": "cdk-hnb659fds-assets-${AWS::AccountId}-eu-central-1",
                                                                    "objectKey":
"045288f3f9bbb83accd1dd75916a88ce95aea3b3f232eae4af7e489e7d1b8a17.json",
          "region": "eu-central-1",
                                                                  "assumeRoleArn":
"arn:${AWS::Partition}:iam::${AWS::AccountId}:role/cdk-hnb659fds-file-publishing-role-${
AWS::AccountId}-eu-central-1"
        }
      }
    }
  },
  "dockerImages": {}
}
```