Міністерство освіти і науки України
Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра мережевих та інтернет технологій

ЗАТВЕРДЖУЮ
завідувач кафедри
мережевих та інтернет технологій
_____ Ю.В. Кравченко
«_____»_____2021 року

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

галузі знань 17 «Електроніка та телекомунікації»
за спеціальністю 172 «Телекомунікації та радіотехніка»

на тему:

Методика створення та розгортання програм на платформі AWS з використанням шифрування SSL

Виконав: студент групи МІТ - 41

Бойко Дмитро Володимирович _____
(прізвище ім'я по-батькові) (підпис)

Керівник: асистент кафедри мережевих та інтернет технологій

асистент, Герасименко К.В. _____
(посада, прізвище ім'я по-батькові) (підпис)

Київ 2020

Міністерство освіти і науки України
Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра мережевих та інтернет технологій

ЗАТВЕРДЖУЮ
завідувач кафедри
мережевих та інтернет технологій
_____ Ю.В. Кравченко
«_____»_____2021 року

ЗАВДАННЯ
НА ДИПЛОМНУ РОБОТУ

Здобувачу вищої освіти Бойко Дмитро Володимирович
(прізвище, ім'я, по батькові)

1. Тема роботи:

«Методика створення та розгортання програм на платформі AWS з використанням шифрування SSL»

затверджена на засіданні кафедри МІТ «11» грудня 2020 р. протокол № 6
2. Термін здачі закінченої роботи «30» травня 2021р
3. Вихідні дані до проекту (роботи)

Додаток із SSL сертифікатом на платформі AWS

4. Зміст пояснювальної записки (перелік питань, що їх потрібно розробити, обсяг – 35-50 стор.)

1. Аналіз предметної галузі. Опис та аналіз розробки додатка
2. Проектування додатка в хмарному провайдері з шифруванням
3. Розробка та розгортання додатка

5. Перелік графічного матеріалу 8-10 слайдів

Налаштування середовища для додатка
Налаштування додатка
Результат розробки програмних засобів
Результат розробки додатку із SSL шифруванням

Дата видачі завдання
Керівник роботи асист. каф. МІТ Герасименко К.В.
(підпис) (посада, прізвище, ім'я, по батькові)

Завдання прийняв до виконання Бойко Дмитро Володимирович
(підпис) (прізвище, ім'я, по батькові)

КАЛЕНДАРНИЙ ПЛАН ВИКОНАННЯ РОБОТИ

Номер	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Підготовчий	28.01.2021	
2	Розділ 1	01.03.2021	
3	Розділ 2	01.04.2021	
4	Розділ 3	01.05.2021	
5	Доповідь та слайди	27.05.2021	
6	Пояснювальна записка	30.05.2021	

Здобувач вищої освіти _____

_____ (підпис) (прізвище, ім'я, по батькові)

Керівник _____

(підпис) (прізвище, ім'я, по батькові)

РЕФЕРАТ

Пояснювальна записка: 56с., 37рис.,15 джерел.

Об’єкт дослідження: розробка та розгортання додатка

Предмет дослідження: процес проектування та реалізації додатка в хмарному провайдері із SSL шифруванням

Мета роботи (проекту): розробити додаток та розгорнути в хмарному провайдері із SSL шифруванням.

Методи дослідження: системний підхід, методи порівняння, узагальнення.

У спеціальній частині дана характеристика сучасного стану технологій безперервної інтеграції та розгортання.

В роботі проведено аналіз існуючих технологій та підходів автоматизації до процесу виконання та оцінювання лабораторних та практичних робіт.

Запропоновано використати стек технологій безперервної інтеграції та розгортання, а також застосунок для створення користувача та порівняння разом з реляційною СУБД.

Розроблено застосунок для перегляду статистики та порівняння робіт студентів.

Розроблено програмне забезпечення та скрипти налаштування конвеєрів.

Практичне значення роботи полягає у створенні програмного забезпечення, для різних компаній та в створенні шифрування для додатка.

Результати здійснених у дипломному проекті досліджень можуть бути використані на першому етапі реалізації системи автоматизації перевірки лабораторних робіт факультету інформаційних технологій, а також у інших системах автоматизації перевірки різних факультетів, де викладається програмування та потрібна автоматизована перевірка робіт студентів.

Галузь використання – програмування, IT-сфера, забезпечення ПО для корпорацій.

Напрямки подальшого розвитку роботи включають інтеграцію розробленого застосунку та програмного забезпечення і скриптів налаштування конвеєрів, а також тестування розробленої системи автоматизації та її компонентів у процесі використання.

Ключові слова: ОДНОСТОРІНКОВИЙ ДОДАТОК, СИСТЕМА УПРАВЛІННЯ БАЗАМИ ДАНИХ, РІВЕНЬ ЗАХИЩЕНИХ СОКЕТІВ, МЕНЕДЖЕР ПАКЕТОВ НОД, МОВА ОПИСУ ДАНИХ, МОВА МАНІПУЛЮВАННЯ ДАНИМИ, РОЗГОРТАННЯ БАЗИ ДАНИХ, НАБІР ЗАСОБІВ РОЗРОБКИ.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

AWS - Amazon Web Services(Хмарна платформа Амазон).

SPA - Single-Page Application(Односторінковий додаток).

SDK - Software Development Kit(Набір засобів розробки).

API - Application Programming Interface(Інтерфейс програмування програм).

SSL - Secure Sockets Layer(Рівень захищених сокетів).

HTML - Hyper Text Markup Language(Мова гіпертекствої розмітки).

VM - Virtual Machine(Віртуальна машина).

NPM - Node Package Manage(Менеджер пакетов нод).

DB – Database (База даних).

СУБД – Система Управління Базами Даними.

DML – Data Manipulation Language (Мова маніпулювання даними).

DDL – Data Definition Language (Мова опису даних).

ВСТУП

За даними GitHub, найбільшого сховища програмного забезпечення з відкритим кодом, JavaScript досі є найпопулярнішою мовою програмування у світі. Більше проектів написано на JavaScript. Навіть проекти, зазвичай не пов'язані з Інтернетом, такі як машинне навчання та криптовалюти, часто використовують JavaScript. Мова програмування JavaScript надзвичайно потужна і здатна, але на додаток до мови існують framework, включаючи React та Node, які додають можливості мови, роблячи її ще кращою. На додачу до цього, TypeScript тепер став стандартом для великих проектів JavaScript. Він надає мовні функції, які роблять кодування за допомогою JavaScript більш продуктивним та краще підходить для великих додатків. Сучасна веб-розробка надзвичайно просунулася за ці роки. Раніше клієнтський код, як правило, означав статичний HTML і CSS, мабуть, крихітний фрагмент JavaScript. І back-end, як правило, писався зовсім іншою мовою, такою як PHP або CGI-скрипти. Однак зараз прийнято писати всю програму, від клієнта до сервера, використовуючи лише JavaScript та пов'язані з ним ф. Ця можливість писати наші програми лише однією мовою забезпечує величезні переваги під час розробки. Крім того, надійні та зрілі механізми роблять повне стекове програмування в JavaScript конкурентоспроможним з будь-якою іншою платформою. Ми навчимось використовувати силу JavaScript для створення повних повнотекстових веб-додатків із SSL сертифікатом. Ми посилимо цю потужність за допомогою TypeScript, ще однієї потужної мови першої десятки. Потім, використовуючи такі інструменти, як React, Redux, Node, Express та GraphQL, ми створимо реалістичну, повнофункціональну веб-програму з найкращими практиками, яка надасть вам усі знання, необхідні для створення сучасних повнотекстових веб-додатків. І як тільки наша програма буде завершена, ми розгорнемо її на хмарних сервісах AWS, найпопулярнішому та багатофункціональному постачальнику хмарних послуг у світі із шифруванням для безпеки

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Опис та аналіз розробки додатка

На даний момент, у нашому світі розробка повного стеку та способи використання сучасних веб-технологій для створення готових до виробництва додатків для розгортання на AWS з використанням SSL шифруванням є затребуваним.

Встановлено, що практичний підхід до впровадження сучасних веб-технологій та пов'язаних з ними методологій для створення повнофункціональних програм є актуальним та не уступає іншим підходам. Глибоке розуміння TypeScript та його використання для створення високоякісних веб-програм – є необхідним. Наступні дослідження заглиблюються в розробку на стороні клієнта за допомогою React та за допомогою нового API Hooks та Redux. Далі проведемо дослідження з розробкою на стороні сервера за допомогою Express, включаючи аутентифікацію за допомогою сеансів на основі Redis та доступ до баз даних за допомогою TypeORM. Пізніше необхідно дізнатись, як будувати схеми GraphQL та інтегрувати їх у React за допомогою хуків. Нарешті, ми зосередились на тому, як розгорнути свою програму на сервері NGINX у хмарного провайдера AWS.

Проведемо дослідження, як створювати та розгортати повні високопродуктивні веб-програми за допомогою React, Node та GraphQL та як розгортати на AWS.

React встановлює стандарт для створення високопродуктивних веб-додатків на стороні клієнта. Node.js - це масштабований сервер додатків, який використовується на тисячах веб-сайтів, тоді як GraphQL стає стандартним способом великих веб-сайтів надавати дані та послуги своїм користувачам. Разом ці технології, підсилені можливостями TypeScript, забезпечують передовий стек для повної розробки веб-додатків.

Проекти Full-Stack Application проведуть нас до процесу підготовки середовища розробки для веб-розробки, створення базового скелетного додатка та його розширення для створення шести різних веб-програм.

Основні характеристики:

- Розуміння архітектури React та односторінкові програми;
- Створення сучасного веб-API для вашого SPA, використовуючи Node.js, Express та GraphQL;
- Отримання чіткого та практичного розуміння того, як створити повний повнотекстовий додаток;

Чому ми навчимось:

- Дослідження про найважливіші функції TypeScript та про те, як їх можна використовувати для поліпшення якості та ремонтопридатності коду;
- Що таке React Hooks і як створювати програми React, використовуючи їх;
- Реалізація управлінням станом для вашого додатку React за допомогою Redux;
- Налаштування експрес-проект за допомогою TypeScript та GraphQL з нуля;
- Створення повнофункціональний онлайн-додаток для форуму за допомогою React та GraphQL;
- Додавання автентифікацію до веб-програми за допомогою Redis;
- Збереження та отримання даних з бази даних Postgres за допомогою TypeORM;
- Налаштування NGINX у хмарі AWS для розгортання та обслуговування додатка.

1.2 Дослідження TypeScript та спосіб покращення JavaScript

Досліджено, що JavaScript - надзвичайно популярна і потужна мова. За даними GitHub, це найпопулярніша мова у світі (так, використовується навіть більше, ніж

Python), а нові функції ES6 + продовжують додавати корисні можливості. Однак для розробки великих додатків набір функцій вважається неповним. Ось чому був створений TypeScript. Було встановлено, що TypeScript має безліч передових функцій, що приносять користь розробникам. Головними серед них є можливості статичного набору тексту та об’єктно-орієнтоване програмування (ООП).

TypeScript насправді два окремі, але пов’язані технології - мова та компілятор:

– Мова - це багатофункціональна статично набрана мова програмування, яка додає справжні об’єктно-орієнтовані можливості до JavaScript;

– Компілятор перетворює код TypeScript у власний JavaScript, але також надає програмісту допомогу в написанні коду з меншою кількістю помилок.

Простережено, що TypeScript дозволяє розробнику розробляти програмне забезпечення вищої якості. Поєднання мови та компілятора покращує можливості розробника. Використовуючи TypeScript, розробник може писати код, який легше зрозуміти та рефакторувати та містить менше помилок. Крім того, це додає дисципліни до робочого процесу розробки, змушуючи виправляти помилки ще в процесі розробки.

TypeScript - це технологія часу розробки. Немає компоненту виконання, і жоден код TypeScript ніколи не працює на жодному комп'ютері. Натомість компілятор TypeScript перетворює TypeScript у JavaScript, і цей код потім розгортається та запускається на браузерах або серверах. Можливо, Microsoft розглядала можливість розробки середовища виконання для TypeScript. Однак, на відміну від ринку операційних систем, Microsoft не контролює орган стандартів ECMAScript (групу, яка вирішує, що буде в кожній версії JavaScript). Натомість Microsoft вирішила створити інструмент, який підвищує продуктивність розробника JavaScript та якість коду[1].

Тоді, якщо TypeScript не має середовища виконання, як розробники отримують запущений код? TypeScript використовує процес, який називається транпіляцією. Транспіляція – це метод, коли код з однієї мови "компілюється" або перетворюється на іншу мову. Це означає, що весь код TypeScript в кінцевому підсумку перетворюється на код JavaScript, перш ніж він остаточно розгортається і

запускається. Розробка великих додатків потребує більших потреб, ніж розробка браузера, для якої вперше було створено JavaScript. На високому рівні майже всі великі мови розробки додатків, такі як Java, C ++, C # тощо, забезпечують можливість статичного введення та можливості ООП. Однак реалізація JavaScript обмежена як з точки зору доступних мовних можливостей, так і дизайну. Дослідимо, як JavaScript виконує ООП і як TypeScript покращує можливості JavaScript.

1.2.1 Об’єктно-орієнтоване програмування в TypeScript

Однак реалізація JavaScript обмежена як з точки зору доступних мовних можливостей, так і дизайну. Розглянемо, як JavaScript виконує ООП і як TypeScript покращує можливості JavaScript.

Основні принципи ООП:

- Інкапсуляція;
- Абстракція;
- Наслідування;
- Поліморфізм.

Коротший спосіб інкапсуляції - це приховування інформації. У кожній програмі ви будете мати дані та функції, які дозволять вам щось робити з цими даними. Коли ми використовуємо інкапсуляцію, ми беремо ці дані та поміщаємо їх у своєрідний контейнер. Цей контейнер відомий як клас у більшості мов програмування, і в основному він захищає ці дані, так що ніщо поза контейнером не може їх змінити або переглянути. Цей метод роботи з об'єктними даними дозволяє суворо контролювати те, що відбувається з цими даними, з одного місця в коді, замість того, щоб їх розповсюджувати по багатьох місцях у великій програмі - що може бути громіздким і складним в обслуговуванні.

Абстракція пов’язана з інкапсуляцією. Використовуючи абстракцію, ви приховуєте внутрішню реалізацію управління даними та надаєте більш спрощений інтерфейс до зовнішнього коду. Перш за все, це робиться для того, щоб викликати "нещільне зчеплення". Це означає, що бажано, щоб код, який відповідає за один набір даних, був незалежним та відокремленим від іншого коду. Таким чином, можна змінити код в одній частині програми без негативного впливу на код в іншій частині. Абстрагування для більшості мов ООП вимагає використання механізму, що забезпечує спрощений доступ до об’єкта, не виявляючи внутрішньої роботи цього об’єкта. Для більшості мов це або інтерфейс, або абстрактний клас. Ця можливість є надзвичайно важливим у створенні згаданої раніше "вільної муфти" і дозволяє легше модифікувати та підтримувати код. JavaScript не підтримує інтерфейси або абстрактні класи, тоді як TypeScript підтримує обидві функції[2].

Наслідування - це повторне використання коду. Наприклад, якщо вам потрібно було створити об’єкти для декількох типів транспортних засобів - автомобіля, вантажівки та човна - було б неефективно писати окремий код для кожного типу транспортного засобу. Було б краще створити базовий тип, який має основні атрибути всіх транспортних засобів, а потім повторно використовувати цей код для кожного конкретного типу транспортного засобу.

Поліморфізм пов’язаний із успадкуванням. У поліморфізмі можна створити об'єкт, якому можна встановити один із будь-якої кількості можливих типів, які успадковуються від одного і того ж базового походження. Ця можливість корисна для сценаріїв, коли необхідний тип неможливо дізнатись відразу, але його можна встановити під час виконання, коли виникнуть відповідні обставини.

1.3 Особливості односторінкових концепцій додатків

Single-Page Application - цей стиль програмування веб-додатків є відносно новим в історії веб-розробки. Зараз його використання є звичною практикою для будівництва великих будівель складні веб-програми, які мають відчувати себе

рідними робочими або мобільними програмами. Було досліджено, що React може допомогти побудувати стиль програми ефективно та ефективним способом.

1.3.1 Переваги та властивості SPA

Використовуючи методи застосування SPA, ми змусимо нашу програму реагувати і виглядати так, ніби вона була встановлена на пристрої. Веб-програми в класичному стилі можуть здаватися млявими, оскільки будь-які зміни на сторінці вимагають зворотного зв'язка із сервером, щоб отримати новий екран. Однак програми у стилі SPA перемальовують частини екрану негайно, не чекаючи повернення нового файлу з сервера. Програма SPA є власним додатком для пристроїв. Створення програм SPA є досить складним, з багатьма компонентами та бібліотеками потрібно використовувати[3].

Є певні особливості та вимоги, які завжди будуть загальними для всіх SPA програм:

- Додаток живе лише на одній HTML-сторінці. На відміну від стандартних програм HTML, які використовують окремі сторінки для відображення різних екранів, перша сторінка є єдиною сторінкою, яка коли-небудь завантажується в програму SPA;
- Замість статичних HTML-файлів JavaScript динамічно відображає екран. Отже, HTML-сторінка, яка вперше завантажується, насправді майже повністю позбавлена вмісту. Але те, що він матиме, - це кореневий елемент всередині тегу body, який стає контейнером для всієї програми, яка знову відображається в режимі реального часу, коли користувач взаємодіє з додатком;
- Всі скрипти та файли, необхідні для запуску програми, зазвичай завантажуються на початку, під час отримання основного файлу HTML.

Однак цей метод змінюється, і більшість програм завантажують лише файл сценарію базового рівня, а потім завантажують інші сценарії за запитом;

– Маршрутизація URL-адрес обробляється для SPA. У програмах SPA існує певний механізм, який використовується, залежно від обраної вами структури, для створення віртуальної маршрутизації. Віртуальна маршрутизація просто означає, що, хоча користувачеві здається, що здійснюються різні виклики до різних URL-адрес на стороні сервера, насправді вся "маршрутизація" відбувається лише у клієнтському браузері з метою здійснення логічних переходів на різні екрани.

1.3.2 Характеристика React

Обгрунтовано, шо файл HTML містить код, який браузер використовує для створення логічного дерева, що називається об'єктна модель документа (DOM). Це дерево представляє всі елементи HTML у файлі відповідно до їх порядку та відносно інших елементів у структурі. Усі веб-сайти мають на своїх сторінках структуру DOM, незалежно від того, використовують вони стиль SPA чи ні. Однак React унікальними способами використовує DOM, щоб допомогти створювати додатки.

React має дві основні конструкції:

– React підтримує власний віртуальний DOM під час виконання. віртуальна DOM містить елемент, який не міститься в DOM браузера - React надішле інструкції DOM браузера для створення цього елемента так, щоб DOM браузера та віртуальний DOM збігалися. Цей процес додавання, редагування або видалення елементів відомий як фаза коміту;

– React додаток складається з багатьох компонентів, і в кожному компоненті може бути якийсь локальний стан (тобто дані). Якщо ці дані зміняться з

будь-якої причини, React ініціює процес узгодження та внесе зміни до DOM, якщо це необхідно.

1.4 Особливості React та використання за допомогою хуків

Простежено, що розробку за допомогою React Hooks зі старішим стилем на основі класів є кращим способом розвитку в React. Показано, що найкращі практики кодування за допомогою хуків, щоб приносить розробнику найкращий якісний код.

1.4.1 Обмеження та проблеми зі старими компоненти стилю класу

Показано, методи повторного використання коду стилю: успадкування та методи життєвого циклу, хоча і були продуктивні, але в кінцевому рахунку не забезпечують хороших можливостей повторного використання коду та структури компонентів. Написання коду із компонентами класу, є дуже важливі компоненти, що базуються на класах, оскільки більшість існуючих кодів React використовують класи, оскільки хуки все ще дещо нові. Додаток React складається з багатьох окремих структур, які називаються компонентами. При використанні стилю на основі класів ці компоненти є класами JavaScript ES6, які успадковуються від React Component. Компонент - це, в основному, машина, яка може містити дані, що називаються станом, і компонент видаватиме HTML через мову, що називається JSX, на основі змін у цих даних. Дочірні компоненти - це просто інші компоненти React, які були вбудовані у функцію візуалізації батьківського компонента.

Компонент класу повинен успадковуватись від об’єкта React Component. Простежено, що він отримує всі можливості компонента React, включаючи функції життєвого циклу. Ці функції є обробниками подій, які надає React, що дозволяють

розробникам підключатись до подій, що відбуваються в певний час протягом життя компонента React.

1.4.2 Стани в React

У компоненті класу React є єдине поле, яке називається “стан”. Це поле є об'єктом, який може містити будь-яку кількість властивостей, що описують пов'язаний компонент. Функції не повинні застосовуватись до стану. Зміни стану - це те, що компонент відображає в React, а компоненти містять лише елементи інтерфейсу для себе, що є хорошим способом підтримувати поділ проблем та чисту практику кодування. Зміни стану у компонентах на основі класу ініціюються функцією setState. Ця функція приймає параметр, який є вашим новим станом, і React згодом асинхронно оновить ваш стан. Це означає, що фактична зміна стану відбувається не відразу, а контролюється системою React. На додаток до стану існує можливість ділитися станом компонента за допомогою реквізиту. Реквізити - це властивості стану, які передаються дочірнім компонентам компонента.

1.5 Аналіз технологій Redux та React Router

Досліджено, що використовуючи глобальний стан Redux, ми можемо зменшити велику кількість типових кодів та впорядкувати програму. React Router - це також найпопулярніший інструмент для управління маршрутизацією URL-адрес на стороні клієнта. Клієнтська маршрутизація URL-адрес дозволяє програмі SPA поводитись звично для користувачів, очікуючи веб-програми класичного стилю, яка вказує, де вони перебувають у програмі. Обидві ці технології необхідні для створення SPA-додатків, які виглядають як стандартні веб-програми, які допомагають синхронізувати дані маршрутизації із сховищем та доступ до нього,

також підтримують налагодження подорожей у часі для змін маршруту в розробницьких інструментах Redux.

1.5.1 Аналіз Redux станів

Стан на основі компонентів може бути обмежувальним. Бувають випадки, коли стан не є специфічним для компонента або навіть для ієрархії компонентів. Іноді стан може бути необхідним для кількох компонентів або інших некомпонентних служб, що складають заявку. На додаток до цього, стан у React передається лише одним способом, від батьківського до наслідуваного. І це обмежує можливість використання стану у React. Отже, Redux забезпечує механізм не лише спільного використання стану, але також дозволяє вводити та оновлювати стан із будь-якого компонента за необхідності. У типовому додатку корпоративного класу ми завжди матимемо автентифікацію. І як тільки аутентифікація користувача відбувається, ми можемо отримати певні дані про користувача - наприклад, повне ім’я користувача, ідентифікатор користувача, електронну пошту тощо[4].

Виявлено, що було б корисно мати можливість зберігати ці дані на клієнті лише в одному місці та ділитися ними з будь-яким необхідним компонентом. Таким чином, якщо ці дані колись оновляться, ми можемо бути впевнені, що всі компоненти, незалежно від того, в якому розділі програми вони знаходяться, отримають останні актуальні дані. Це те, що Redux може зробити для нашої програми. Redux - це служба зберігання даних, яка зберігає всі загальнодоступні дані в нашому додатку React. Redux надає не тільки саме інформацію, але й основні функції, необхідні для додавання, видалення та спільного використання цих даних. Однак одна відмінність від стану React полягає в тому, що стан Redux не обов'язково ініціює оновлення інтерфейсу. Це, звичайно, може, якщо ми хочемо це

зробити, але немає явної необхідності робити це. Щоб ваш компонент мав доступ до стану, вам слід чітко вказати, до якого стану він отримає доступ.

1.5.1 Характеристика React Router

React Router є найбільш часто використовуваною структурою маршрутизації в React. Маршрутизація, як досліджено, є вивчення концепцій додатків на одній сторінці та Як React використовує їх, є актуально у веб-розробці. Це функція, яку очікують користувачі веб-додатків, тому навчитися користуватися нею для нашого додатку React є обов’язковою вимогою.

Маршрути в React Router - це просто компоненти React Router, які містять наші власні компоненти програми, і ці компоненти, в свою чергу, представляють наші екрани. Іншими словами, маршрут у React Router є логічним поданням віртуального розташування (під віртуальним розташуванням я маю на увазі URL-адресу, яка є лише міткою і насправді не існує на жодному сервері). "Маршрутизатори" в React Router виконують роль батьківських компонентів, а наші компоненти візуалізації екрану діють як дочірні.

1.6 Аналіз компонентів Server-Side Development з Node.js та Express

Виявлено, що Node може допомогти нам створити ефективні веб-служби та встановлено, що взаємозв'язок між Node і Express можемо використовувати їх для створення нашого веб-API.

1.6.1 Характеристика Node технології

Node - одна з найпопулярніших у світі інструментів JavaScript. Він використовується як основна технологія для мільйонів веб-сайтів. Він дуже швидкий, і при використанні з такими речами, як кластеризація та робочі потоки, дозволяє масштабувати. Крім того, оскільки він використовує JavaScript, він дозволяє створювати повнофункціональний додаток, спереду назад, використовуючи лише одну мову. Усі ці характеристики роблять Node приголомшливим вибором, якщо ви націлюєтесь на Інтернет. Визначено, що Node не є серверною структурою. Це насправді середовище загального користування, а не просто веб-сервер. Вузол забезпечує JavaScript такими можливостями, яких він зазвичай не має, наприклад, можливістю отримати доступ до файлової системи та приймати вхідні мережеві з'єднання. Браузер також є середовищем виконання для нашого коду JavaScript (а також HTML і CSS). Браузер працює, маючи основний механізм JavaScript, який забезпечує базові функції мови JavaScript. Сюди входить інтерпретатор мови, який читає наш код для дійсного JavaScript, і віртуальна машина, яка запускає наш код на різних пристроях. Над цим ядром браузер забезпечує захищений контейнер пам’яті для запуску програм, пісочниці. Але він також надає додаткові можливості JavaScript, загально відомі як веб-API (не на стороні сервера, а на рівні браузера). Веб-API доповнює базовий механізм JavaScript, надаючи такі речі, як об'єктна модель документа (ОМД), щоб код JavaScript мав доступ до документа HTML і маніпулював ним. Він забезпечує такі дзвінки, як fetch, що дозволяють здійснювати асинхронні мережеві дзвінки на інші машини, а також WebGL для графіки та багато іншого[5].

Node - це середовище виконання загального користування, орієнтоване на високу продуктивність та масштабованість. За допомогою Node можна створювати багато типів додатків, включаючи сценарії керування комп’ютером та програми терміналів. Але можливості масштабування Node роблять його також придатним як веб-сервер. Node має багато можливостей, які роблять його дуже придатним для виконання програм.

1.6.2 Аналіз техології Express та її проміжні виклики

Окреслено, що Node безпосередньо має незручне та громіздке розробку. Наявність простішого у використанні API зробить нас більш продуктивними. Це те, що намагається зробити Express. Простежено, що Express може допомогти нам легше писати код для наших програм Node.

Express не є автономною структурою сервера JavaScript. Це слой коду, який знаходиться на вершині Node і, отже, використовує Node, щоб спростити розробку серверів JavaScript з Node. Так само, як Node, має свої основні можливості, а потім деякі додаткові функції через пакети залежностей. Express також має свої основні здібності, а також багату екосистему проміжного програмного забезпечення, що забезпечує додаткові можливості[6].

Express - це лише програма, яка являє собою серію проміжних викликів(Рис. 1.6.1)

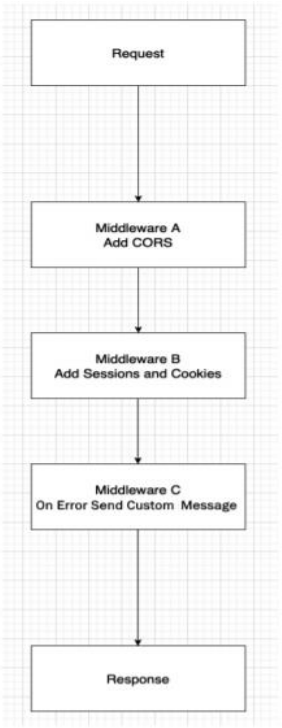

Рисунок 1.6.1 – Проміжні виклики Express

1.7 Дослідження веб-технології GraphQL

Показано, що GraphQL, одна з найпопулярніших веб-технологій, що використовуються в даний час. Багато великих компаній прийняли GraphQL для своїх API, включаючи такі компанії, як Facebook, Twitter, New York Times та GitHub. GraphQL є настільки популярним, тому що він працює внутрішньо, і ми можемо скористатися його багатьма можливостями.

GraphQL має декілька основних характеристик:

- GraphQL - це стандарт схеми даних, розроблений Facebook. Він забезпечує стандартну мову для визначення даних, типів даних та відповідних запитів даних. Там немає коду, але ви все одно можете побачити, які типи та запити доступні;
- GraphQL працює на різних платформах, інструментах та мовах. При створення API за допомогою GraphQL, для опису наших даних, їх типів та запитів буде використана та сама мова GraphQL, незалежно від того, яку мову програмування чи операційну систему ми використовуємо. Наявність послідовного та надійного представлення даних у найрізноманітніших системах та платформах – це добре для клієнтів та систем. Але це також корисно для програмістів, оскільки ми можемо продовжувати використовувати нашу звичайну мову програмування та основи вибору;
- GraphQL повертає контроль тим, хто звертається до абонента. У стандартній веб-службі саме сервер контролює, які поля даних будуть повернуті. Однак в API GraphQL саме клієнт визначає, які поля він хотів б отримати. Це надає клієнтам кращий контроль і зменшує використання пропускної здатності та вартість.

Встановлено, що існує два основних способи використання кінцевої точки GraphQL. Один - як шлюз для консолідації інших служб даних, а другий - як основний сервіс веб-API, який безпосередньо отримує дані з сховища даних і надає

їх клієнтам. Схема GraphQL(Рис 1.7.1), яка використовується як шлюз для інших даних.

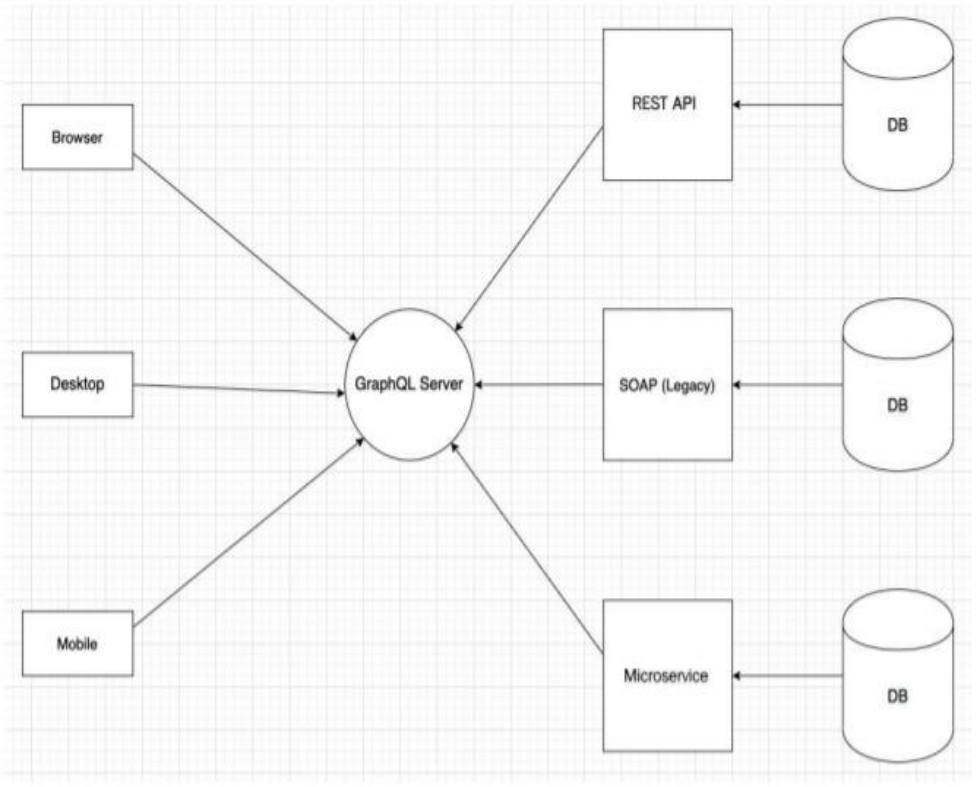

Рисунок 1.7.1 – Схема GraphQL

GraphQL виступає єдиним основним сервером для всіх клієнтів. Він добре працює завдяки своїй мові, заснований на стандартах, який підтримується в широкому спектрі систем. Це означає, що вам не потрібно переписувати всю програму. GraphQL має власну мову даних, що означає, що її можна використовувати незалежно від середовища сервера, мови програмування програм або операційної системи. Ця гнучкість дозволяє GraphQL бути потужним засобом обміну даними в організації або навіть в Інтернеті.

1.8 Аналіз хмарного провайдера AWS

Досліджено, що Amazon Web Services (AWS) - одна з найпопулярніших та найефективніших хмарних платформ для адміністрування та розгортання додатків,

щоб зробити їх стійкими та надійними. AWS має багато розширених концепцій хмарного адміністрування для розгортання, управління та експлуатації високодоступних систем. Він пропонує сотні послуг з тисячами функцій, які допомагають створити масштабовані хмарні рішення для різних додатків та взаємодіє з ними через API[7].

API - це інтерфейс, що дозволяє двом незалежним компонентів програмного забезпечення обмінюватися інформацією. API грає роль посередника між внутрішніми і зовнішніми програмними функціями, забезпечуючи настільки ефективний обмін інформацією, що кінцеві користувачі зазвичай його просто не помічають.

Amazon Web Services (AWS) змінив спосіб системного адміністрування. Подумайте про дохмарну еру, коли, якби ми планували створити новий центр обробки даних, він проходив би місяць планування, що включало б вибір місця, замовлення обладнання, налаштування мережевої інфраструктури (наприклад, маршрутизаторів та комутаторів) ; і список можна продовжувати і продовжувати. За допомогою AWS налаштування нового центру обробки даних можна виконати за допомогою декількох клацань миші або за допомогою викликів інтерфейсу прикладного програмування (API).

В AWS є комплект для розробки програмного забезпечення Python (SDK) та багатофункціональний об’єктно-орієнтований API, який забезпечує низькорівневий доступ до служб AWS. CloudFormation та Terraform - ці інструменти можна використовувати для автоматизації вашої інфраструктури AWS, але між ними є незначна різниця. CloudFormation, з одного боку, є власним рішенням AWS, тоді як Terraform - це проект з відкритим кодом. Інша ключова різниця між ними полягає в тому, що Terraform підтримує інших хмарних провайдерів, таких як Google Cloud та Azure, тоді як CloudFormation є рідною для AWS.

1.9 Аналіз шифрування даних за допомогою SSL/TLS протоколів

Ми живемо у світі, який дедалі більше зв’язується. Протягом останнього десятиліття 20-го століття Інтернет став популярним і назавжди змінив спосіб життя. Сьогодні ми покладаємось на наші телефони та комп’ютери для спілкування, купівлі товарів, оплати рахунків, подорожей, роботи тощо. Багато з нас, постійно вмикаючи пристрої в кишенях, не підключаються до Інтернету, ми є Інтернетом. Вже зараз телефонів більше, ніж у людей. Кількість смартфонів вимірюється мільярдами і швидко зростає. Тим часом тривають плани підключення всіляких пристроїв до однієї мережі. Всі пристрої, підключені до Інтернету, мають одне спільне - вони покладаються на протоколи SSL (Secure Socket Layer) і TLS (Transport Layer Security) для захисту інформації, що передається[8].

Коли спочатку розроблявся Інтернет, про безпеку мало замислювалися. Як результат, основні комунікаційні протоколи за своєю суттю є небезпечними та покладаються на чесну поведінку всіх залучених сторін. Це могло спрацювати ще в ті часи, коли Інтернет складався з невеликої кількості вузлів - переважно університетів -, але сьогодні повністю розпадається, коли всі перебувають у мережі. SSL і TLS - це криптографічні протоколи, призначені для забезпечення безпечного зв'язку через незахищену інфраструктуру. Це означає, що, якщо ці протоколи розгорнуті належним чином, ви можете відкрити канал зв'язку для довільної служби в Інтернеті, бути досить впевненими, що спілкуєтесь із правильним сервером, і безпечно обмінюватися інформацією, знаючи, що ваші дані виграли не потрапити в чужі руки і що його отримають цілим. Ці протоколи захищають лінію зв'язку або транспортний рівень, звідки походить назва TLS.

Безпека - не єдина мета TLS/SSL. Ці протоколи насправді мають чотири основні цілі, перелічені у порядку пріоритету:

– Криптографічна безпека: забезпечення безпечного спілкування між будь-якими двома сторонами, які бажають обмінятися інформацією;

- Сумісність : незалежні програмісти повинні мати можливість розробляти такі програми та бібліотеки здатні спілкуватися між собою за допомогою загальних криптографічних параметрів;
- Розширюваність: TLS є фактично основою для розробки та розгортання криптографічних протоколів. Його важливою метою є бути незалежним від фактичного криптографічні примітиви (наприклад, шифри та функції хешування), що дозволяють мігрувати з одного примітиву на інший без необхідності створювати нові протоколи;
- Ефективність: кінцевою метою є досягнення всіх попередніх цілей за прийнятних витрат, зменшення витратних криптографічних операцій до мінімуму та надання схеми кешування сеансу, щоб уникнути їх при подальших з'єднаннях.

Висновки до розділу 1

У цьому розділі ми дізналися про додаткові залежності пакета NPM, які ми будемо використовувати для побудови нашого додатку. Ці інструменти добре використовуються громадою, а тому добре перевірені та надійні. Використання пакетів з екосистеми Node - одна з найцінніших переваг Node. Це позбавляє нас від необхідності самостійно писати, тестувати та підтримувати цей додатковий код. У наступному розділі ми детально розглянемо те, що будуватимемо. Ми побачимо, якими будуть різні компоненти нашого додатку, і ми почнемо писати код до нашого додатку React.

2 ПРОЕКТУВАННЯ ДОДАТКА В ХМАРНОМУ ПРОВАЙДЕРІ З ШИФРУВАННЯМ

Процес проектування додатка є основною та важливою частиною, сам процес призначений для опису рішення, яке повинно відповідати функціональним вимогам і вимогам ефективності, а також обмеженням того середовища, в якій вона буде працювати. Раніше зібрані вимоги уточнюються і поліпшуються, щоб задовольняти можливим технологічним обмеженням. Раніше ми згадували основні технології, які необхідні для створення сучасного веб-додатка. Розглянемо проектування цих технологій окремо кожну із них.

2.1 Проектування технології TypeScript

Як було сказано раніше, TypeScript компілюється, а не інтерпретується так, як JavaScript. У випадку TypeScript люди часто говорять про транспіляцію, а не про компіляцію, оскільки компілятор TypeScript насправді виконує перетворення з джерела в джерело. Другий ключовий момент полягає в тому, що TypeScript - це набір JavaScript, як показано на наступному рисунку (рис 2.1).

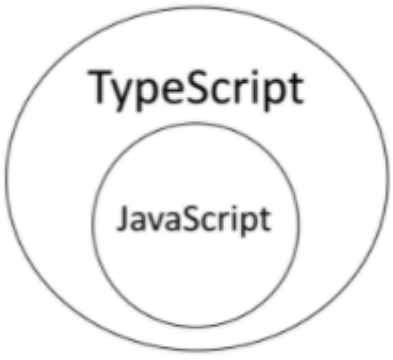

Рисунок 2.1 – Набір набір JavaScript

Оскільки будь-який код JavaScript також є дійсним кодом TypeScript, це також означає, що ви можете оголошувати змінні, не вказуючи їх тип, а пізніше призначати їм різні типи (наприклад, числа, рядки тощо). Хоча той факт, що визначення типів є необов'язковим за замовчуванням у TypeScript, не означає, що вам слід уникати визначення типів. Натомість TypeScript підсвітчує код, коли ви використовуєте його систему типів. TypeScript дозволяє чітко вказати тип ваших змінних та функцій. Крім того, він також має дуже потужний висновок типу, підтримку класів, дженериків, переліків, мікшинів та багатьох інших цікавих речей.

2.1.1 Переваги TypeScript та його розробки

Проведення власних досліджень, було визначено що TypeScript не має на меті замінити JavaScript; натомість він спрямований на поліпшення життя розробників, забезпечуючи більш потужну мову, яка генерує чистий і простий код JavaScript. Певним чином, можна розглядати TypeScript, як засіб перевірки якості коду для JavaScript на стероїдах[9].

Друга велика перевага полягає в тому, що TypeScript дозволяє використовувати найновіші функції ECMAScript прямо зараз, незалежно від того, націлено ви на останню версію Node.js або Internet Explorer 11 (бідна душа!). Це чудово, оскільки це означає, що вам не потрібно чекати, поки все буде підтримано в цільовому середовищі.

Можна дуже легко налаштувати компілятор TypeScript для створення коду, сумісного з ES3-, ES5-, ES2015-, ES2016-, ES2017-, ES2018-, ES2019- або ESNext.

Щоб було зрозуміліше, візьмемо приклад. У TypeScript ви можете створювати класи, перекладаючи свій код у ES3-сумісний код, хоча класи були введені лише в ES2015. Компілятор TypeScript виконує перетворення лише для використання мовних конструкцій, що існували у цій версії ECMAScript. Коли TypeScript робить це, ми часто говоримо про випромінювання нижчого рівня. Є багато можливостей

останніх версій ECMAScript, які можна використовувати з TypeScript під час націлювання на ES5.

2.1.2 Властивості та функції TypeScript

Однією з найкращих функцій TypeScript є потужна система шрифтів, додана поверх JavaScript. Це одне повинно сподобатися будь-якому розробнику, який звик працювати з сильно набраними мовами і, отже, звик користуватися великою підтримкою IDE.

Коли ви розробляєте за допомогою C # або Java із надійним редактором / IDE, ви отримуєте потужне автоматичне заповнення, підтримку рефакторингу, доступ до документації, підказки в контексті та попередження та помилки - все безкоштовно. TypeScript забезпечує той самий досвід розробника для екосистеми JavaScript і дозволяє розробникам скористатися високопродуктивним середовищем розробки. Це чудово підходить для всіх розробників, а не лише для більших команд.

Незалежно від того, чи комфортніше статичний або динамічний набір тексту, TypeScript порадує вас, оскільки типи в TypeScript необов'язкові, а TypeScript має дуже потужні можливості виведення тексту. Більше того, вивід типу покращується з кожним новим випуском компілятора.

Простіше кажучи, TypeScript, безумовно, допоможе вам виявити помилки раніше, і це допоможе вам краще організувати та структурувати свій код, незалежно від того, працюєте над невеликим додатком чи великим проектом.

Було приведено достатньо агрументів до використання цієї мови, TypeScript не є універсальним рішення. Тому було встановлено, що TypeScript необхідно в більшості використовувати в таких випадках:

– У разі великого додатка. TypeScript відмінно підійде, якщо ваше додаток має велику архітектуру і купу коду, а особливо якщо над цим додатком працюють кілька розробників;
– У разі, коли ви і ваша команда знайомі з ЯП зі статичної типізацією. Команді варто дивитися в бік TypeScript, коли вони хочуть розробляти додаток на JavaScript і вже знайомі з Java або C #, адже ті є мовами зі статичної типізацією.

2.2 Створення програм з функціями ES6 +

Є деякі важливі особливості JavaScript у його останній формі ES6 + (я додав знак плюс для позначення ES6 і пізніше). Важливо розуміти, що, хоча ця книга використовує TypeScript, але ці дві мови доповнюють один одного. Переглянемо масштаб змінних та новий const і додамо ключові слова. Дослідження нових функцій мапів та async await покращує роботу додатку.

Функція “arrow” були новим доповненням до ES6. В основному вони слугують двом основним цілям:

– Вони скорочують синтаксис функцій запису;
– Вони також автоматично роблять безпосереднім батьківським обсягом, цей об'єкт, батьківським елементом функції стрілки.

У TypeScript об’єкт, екземпляр об’єкта власника, до якого належать властивості та методи членів, може змінюватися залежно від контексту виклику. Отже, коли функція викликається безпосередньо - наприклад, MyFunction () - батьком це буде викликаючою функцією; тобто, поточний обсяг - це об’єкт. Для браузерів це, як правило, об'єкт вікна. Однак у TypeScript функції також можуть використовуватися як конструктори об'єктів - наприклад, new MyFunction (). У цьому випадку цим об’єктом усередині функції буде екземпляр об’єкта, створений із нового конструктора MyFunction. Приклад при використанню цього коду:

function MyFunction () { console.log(this); } MyFunction(); let test = new MyFunction(); Необхідно скомпілювати код, а потім запустити його, ви побачите результат(Рис. 2.2)

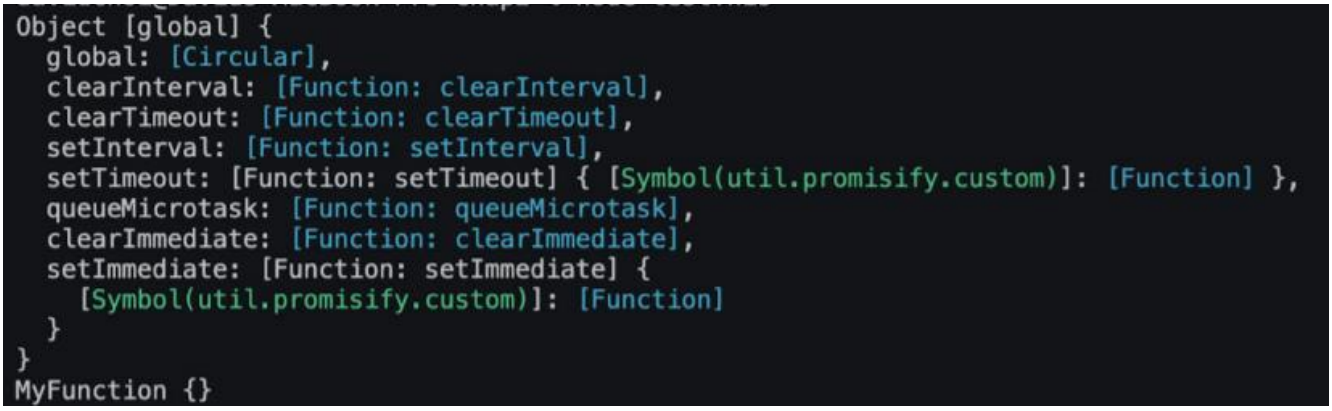

Рисунок 2.2 – Екземпляр конструктора MyFunction

Мап - це набір пар ключ-значення. Іншими словами, це словник. Кожен учасник Мап має унікальний ключ. Необхідно створити новий файл з назвою mapCollection.js і додати наступний код: const mappedEmp = new Map(); mappedEmp.set('linda', { fullName: 'Linda Johnson', id: 1 }); mappedEmp.set('jim', { fullName: 'Jim Thomson', id: 2 }); mappedEmp.set('pam', { fullName: 'Pam Dryer', id: 4 });console.log(mappedEmp);console.log('get', mappedEmp.get('jim')); console.log('size', mappedEmp.size); for(let [key, val] of mappedEmp) { console.log('iterate', key, val); }.

Обґрунтовано, що деякі виклики дуже схожі на Set. Однак однією відмінністю є цикл ітерацій внизу, який використовує масив для позначення ключа та значення(Рис. 2.3).

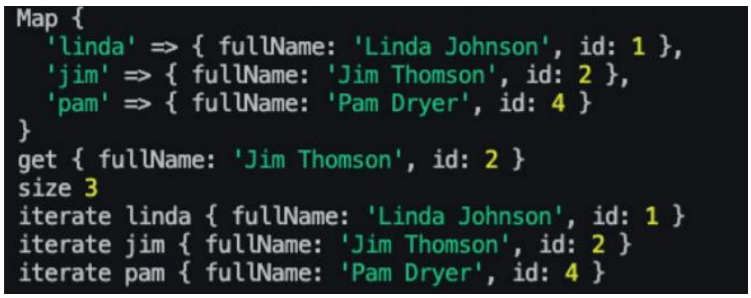

Рисунок 2.3 – ключ-значення mapCollection

Перш ніж пояснювати асинхронність і очікування, необхідно дізнатись, що таке асинхронний код. У більшості мов код, як правило, синхронний, що означає,

що оператори виконуються один за одним. Якщо у вас є оператори A, B і C, оператор B не може виконуватися до завершення оператора A, а оператор C не може виконуватися до завершення оператора B. Однак в асинхронному програмуванні, якщо оператор A є асинхронним, він запускається, але відразу після цього починається оператор B. Отже, твердження B ніколи не чекає, поки A завершиться, перш ніж воно запуститься. Це чудово для продуктивності, але ускладнює читання та виправлення коду. async очікують у спробах JavaScript вирішити деякі з цих труднощів. Очікування про асинхронізацію очікує. Отже, асинхронне програмування забезпечує більш високу продуктивність, оскільки оператори можуть працювати одночасно, не чекаючи один одного. Однак, щоб зрозуміти асинхронне програмування, нам слід спочатку зрозуміти зворотні виклики. Зворотні виклики є основною особливістю програмування Node.js з самого початку, тому це важливо розуміти. Перш ніж чекати асинхронізації, асинхронний код обробляється за допомогою Promises. Promises - це об’єкт із затриманим завершенням у якийсь невизначений час. Тут асинхронне очікування може допомогти. Він робить дві основні речі: очищає код, робить його простішим та меншим, а також полегшує виконання коду, оскільки він виглядає як синхронний код. Приклад коду: async function delayedResult() { return new Promise((resolve, reject) => { setTimeout(() => { resolve('I completed successfully'); }, 500); }); } (async function execAsyncFunc() { const result = await delayedResult(); console.log(result); })();

Цей код має функцію, яка називається delayedResult, яка, як бачите, перед собою має префікс async. Префікс функції перед асинхронізацією повідомляє час виконання, що ця функція поверне Promises і тому повинна оброблятися асинхронно. Після delayedResult ми бачимо функцію execAsyncFunc, яка одночасно оголошується і виконується. Якщо ви не знайомі з нею, ця можливість називається виразом функції відразу ж викликаного. Функція execAsyncFunc також підтримує функцію асинхронізації, і, як бачите, вона внутрішньо використовує ключове слово await[10]. Ключове слово await повідомляє час виконання, що ми збираємося виконати асинхронну функцію, тому воно повинно чекати від нашого імені, а потім,

після завершення оператора, натомість дасть нам фактичне значення, що повертається(Рис. 2.4)

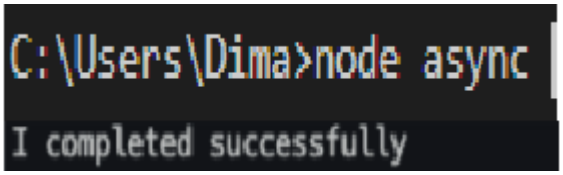

Рисунок 2.4 – Вивід async

2.3 React концепції односторінкового додатку

По суті, сучасний додаток React потребує декількох базових функцій, щоб функціонувати. Досліджено, що npm необхідний щоб допомогти нам керувати залежностями програми. Npm - це сховище, яке дозволяє нам завантажувати залежності з відкритим кодом із центрального магазину та використовувати їх у нашому додатку. Також знадобиться інструмент для виконання того, що називається групуванням. Система зв'язку - це послуга, яка об'єднує всі наші файли скриптів та активи, такі як файли CSS, і зводить їх до єдиного набору файлів. Процес мініфікації видаляє пробіли та інший непотрібний текст із наших сценаріїв, щоб файли, які в кінцевому підсумку завантажуються в браузери користувачів, були якомога меншими. Цей менший розмір корисного навантаження покращує час запуску програми та покращує взаємодію з користувачем. Система комплектування, яка використовується, називається webpack, і ми обрали її, оскільки це галузевий стандарт для групування програм React. Крім того, ми можемо використовувати вбудовану систему сценаріїв npm і створювати сценарії для автоматизувати частину нашої роботи. Наприклад, можливо створити сценарії, які запускатимуть наш тестовий сервер, запускати наші тести та створювати остаточну версію нашого додатку.

2.4 Дослідження React за допомогою хуків

Наступне зображення - це хороший огляд викликів життєвого циклу в компоненті React на основі класу. Крім того, існує також кілька застарілих функцій, які не згадуються на схемі, такі як componentWillReceiveProps, які були повністю усунені, оскільки спричиняли такі проблеми, як небажані візуалізації та нескінченні цикли(Рис. 2.5).

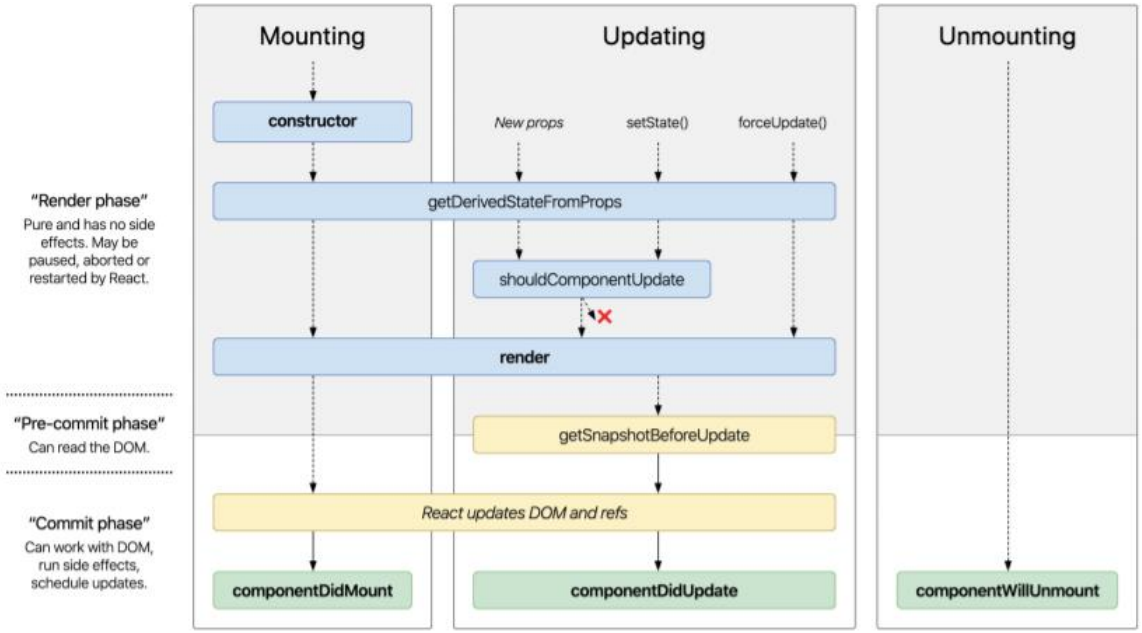

Рисунок 2.5 - Життєвий цикл компонента React

По-перше, можна бачити, що є монтування, оновлення та демонтування. Монтування - це просто створення і ініціалізація компонента, а потім додавання ініціалізованого компонента у віртуальний React DOM. Оновлення стосується рендерингу. Саме тоді змінюється стан і інтерфейс користувача повинен бути оновлений. Демонтування відбувається, коли компонент більше не використовується і його потрібно видалити з DOM[11].

Висновок до розділу 2

У цьому розділі ми охопили дуже великий обсяг інформації. Ми дізналися про компоненти, що базуються на заняттях. Ми також дізналися про компоненти на основі хука, які простіші та легші для повторного використання. Зараз ми знаємо основи програмування React. Тепер ми можемо створювати власні компоненти React і починати будувати наш додаток! У наступному розділі ми дізнаємося про інструментарій навколо React. Ми поєднаємо отримані тут знання з інформацією про інструменти, і це допоможе нам написати чистий, чуйний код. Ми також дізналися про деякі обмеження давнього стилю веб-розробки та про те, як програми SPA намагаються їх подолати. Ми побачили, як головний драйвер SPA-програми полягає у тому, щоб веб-програми поводились більше як власні програми. Нарешті, ми отримали вступ до компонентів для розробки та побудови React. У наступному розділі ми спиратимемось на ці знання та підемо в побудову компонентів React більш глибоко. Ми розглянемо компоненти, що базуються на класах, і порівняємо та порівняємо їх із новими компонентами у стилі Хук. Те, що ми до цього часу дізналися про веб-розробку та веб-розробку на основі React, допоможе нам краще зрозуміти цей наступний розділ.

3 РОЗРОБКА ТА РОЗГОРТАННЯ ДОДАТКА

Після того, як додаток буде завершено, його слід розгорнути, перш ніж його можна буде використовувати. Існує багато варіантів на вибір, включаючи використання власної інфраструктури. Однак сьогодні більшість компаній рекомендують користуватися послугами хмарного провайдера, щоб зменшити свої витрати, пов'язані з IT. AWS є стандартом, що стосується хмарних провайдерів. Ми налаштуємо наші служби додатків Redis, Postgres та NGINX поверх Linux VM.

3.1 Технічні вимоги

Для розробки додатку необхідно глибоко розуміти такі веб-технології як: TypeScript, Redis, React, Express та GraphQL, хмарний провайдер AWS та SSL. Також необхідно використовувати Node та Visual Studio Code. Репозиторій GitHub доступний за адресою https://github.com/dimon12091/Diplom_Web-application_AWS.

Потрібно базові налаштувань на вашій машині для розробки:

- Створити папку practice, потім скопіювати папки super-forum-server та super-forum-client з вихідного коду папки test;
- Якщо node_modules та package-lock.json скопіюються, видаліть ці папки і файли;
- Перейти у папку practice/super-forum-server і виконати команду: npm install;
- Перейти у папку practice/super-forum-client та виконайте команду: npm install.

3.2 Налаштування Ubuntu Linux на AWS Cloud

Необхідно створити обліковий запис AWS. Процес досить простий, оскільки існуючий образ Ubuntu Linux вже буде доступний для використання.

Після входу в систему(Рис. 3.1) є поточним порталом AWS. Стартова сторінка часто змінюється, тому вигляд може відрізнятися:

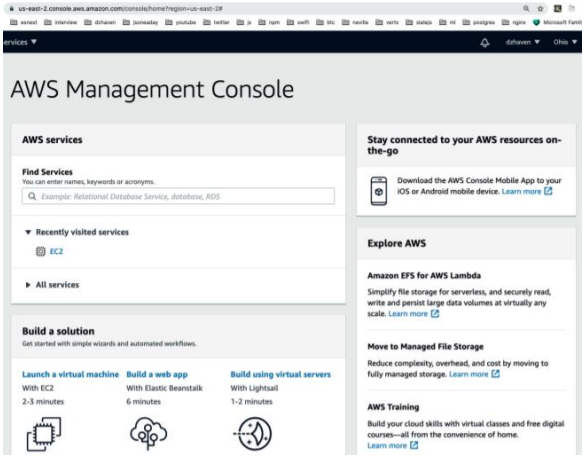

Рисунок 3.1 – Стартова сторінка AWS

В стартовому вікні необхідно найти сервіс EC2 для створення віртуальної машини та вибрати віртуальну машину, яку ми хочемо запустити(Рис. 3.2).

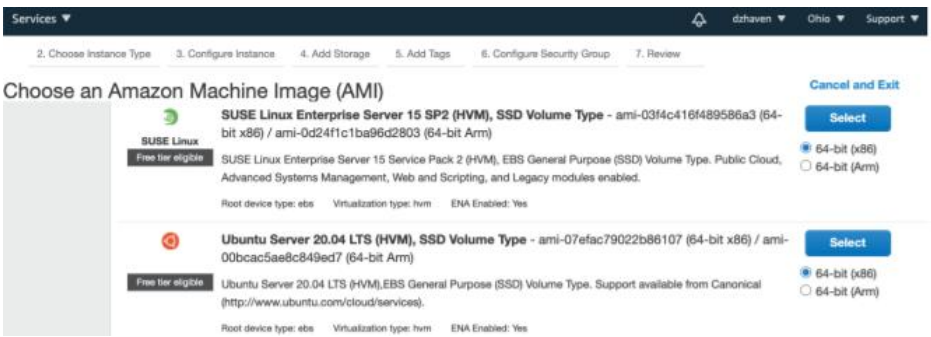

Рисунок 3.2 – Вибір віртуальної машини

Виберемо образ Ubuntu 20.04 LTS, далі перейшовши в наступний розділ треба вибрати образ із 1 vCPU та 2 ГБ пам'яті(Рис. 3.3). Зауважте, що EBS - це специфічна для AWS оптимізація продуктивності для зберігання. Давайте спростимо ситуацію, зберігаючи налаштування за замовчуванням і після вибору натисніть кнопку - переглянути та запустити вниз екрана[12].

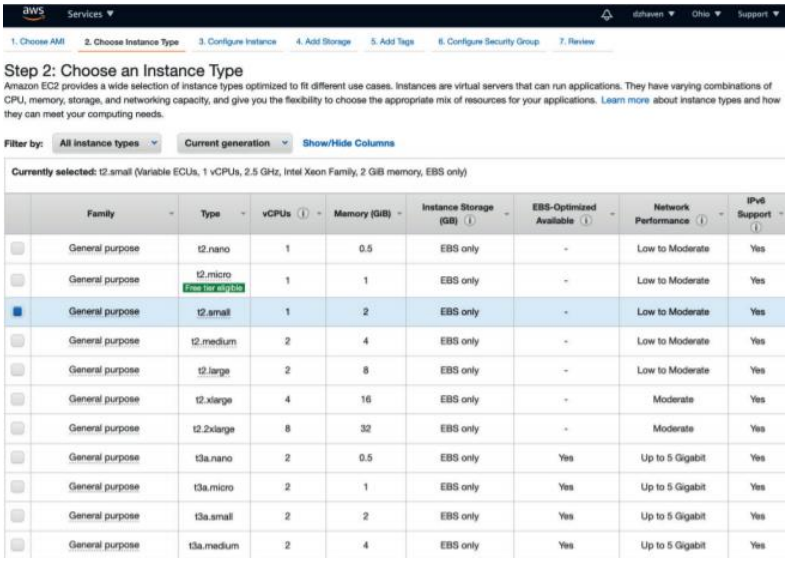

Рисунок 3.3 – Тип віртуальної машини

Натисніть продовжити, щоб перейти у наступний розділ для вибору ключа(Рис. 3.4). Набір ключів шифрування необхідний для використання з SSH, один для вас і один для AWS, завдяки якому можна заходити на віртуальну машину по захищеному з'єднанню. Завантажте ключ та збережіть у надійному місці. Натисніть кнопку запустити екземпляр, щоб продовжити.

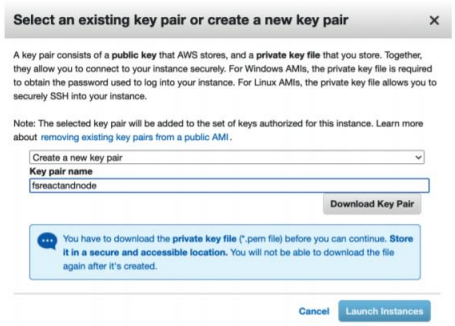

Рисунок 3.4 – Вибір або створення ключа для безпечного підключення

Після завершення побачите екран запуску стану. Натисніть кнопку(Рис. 3.5) - переглянути VM's, щоб перейти на сторінку з вашими віртуальними машинами.

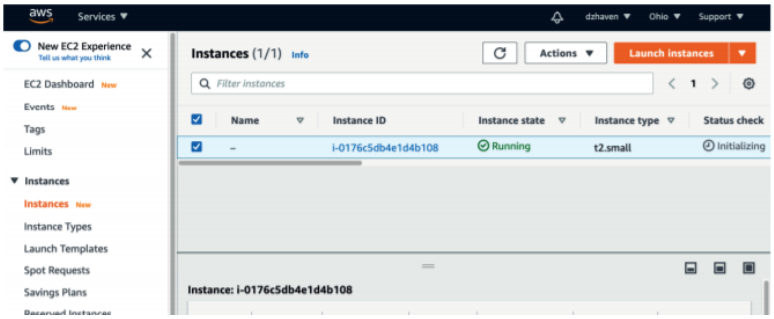

Рисунок 3.5 – Сторінка з VM's

Натисніть на Ідентифікатор VM, і отримаєте екран зведення екземпляра. Можна побачити такі факти, як стан запущеного екземпляра, загальнодоступна IP-адреса та загальнодоступне ім'я DNS. Біля верхнього правого кута цього екрана ви побачите - кнопку підключення. Натисніть на неї, щоб відкрити екран Connect to instance. Перша вкладка - EC2 Instance Connect, який є терміналом, наданим нам AWS. Натисніть кнопку Connect, і ми побачимо термінал до нашого сервера Ubuntu всередині нашого браузера(Рис. 3.6).

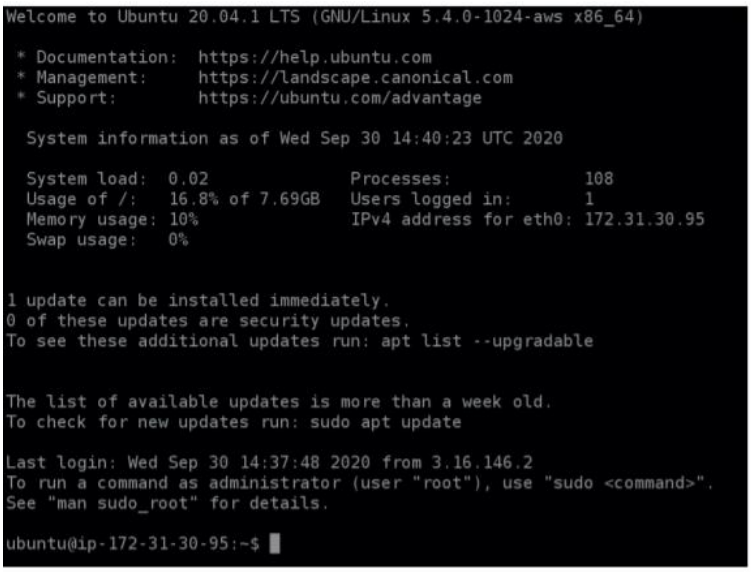

Рисунок 3.6 – AWS EC2-підключення

Це один із способів підключення до VM, через веб інтерфейс. Інший спосіб підключення – через SSH. Поверніться до екрана Connect to instance та виберіть вкладку, клієнт SSH та використайте ключ шифрування який недавно скачували та підключіться за допомогою нього(Рис. 3.7)[13].

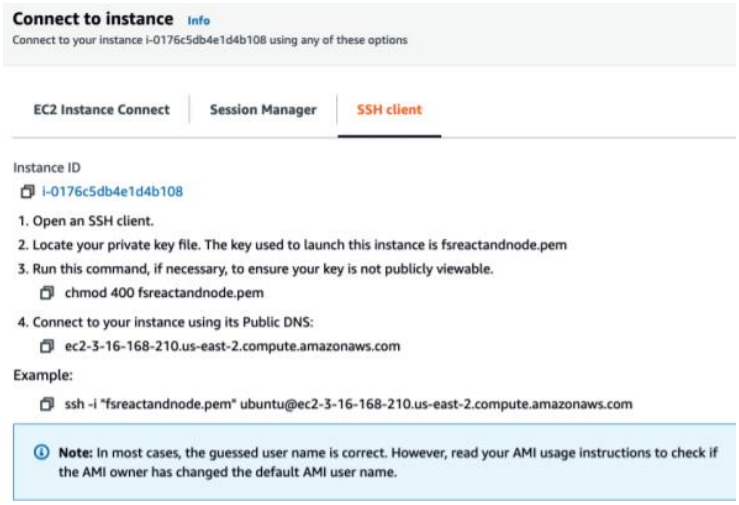

Рисунок 3.7 – Підключення по SSH

3.3 Налаштування Redis, Postgres та Node на VM Ubuntu

У цьому розділі ми встановимо наші основні вимоги на наш сервер Linux. Ми вже проводили налаштування та конфігурацію Redis, налаштовували стан сеансу за допомогою Express та Redis, але зробимо це тепер у хмарному провайдері.

3.3.1 Налаштування Redis

На своєму терміналі зайдіть на сервер і запустіть дві команди: sudo apt update; sudo apt install redis-server.

Apt - це інструмент упаковки програмних залежностей для таких дистрибутивів Linux, як Ubuntu та Debian. Після завершення обновлень, відкрийте файл redis.conf таким чином: sudo nano /etc/redis/redis.conf .

Знайдіть запис requirepass, прокоментуйте його, а потім додайте власний пароль. Попередження Пароль у файлі super-forum-server/devconfig/.env папки вихідного коду для змінної REDIS_PASSWORD повинен відповідати паролю, який ви ввели у своєму файлі redis.conf. Далі знайдіть контрольований запис і встановіть для нього значення systemd. Тепер збережіть і вийдіть. Тепер давайте перезапустимо наш сервер Redis, щоб прийняти нові налаштування: sudo systemctl перезапустіть redis.service Якщо треба зупинити службу, ми можемо виконати команду: sudo systemctl stop redis.service Якщо треба запустити службу, ми запускаємо команду: sudo systemctl start redis.service[14]. Якщо запустити команду - sudo systemctl status redis, вона покаже, чи працює Redis належним чином(Рис. 3.8).

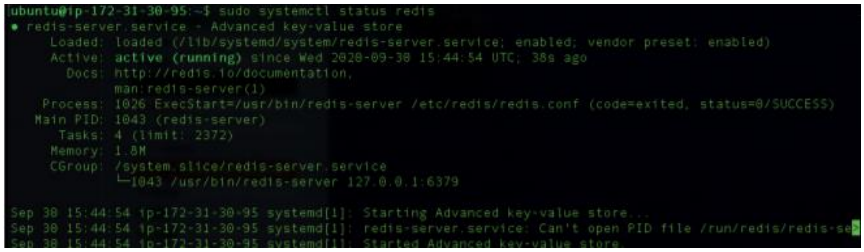

Рисунок 3.8 – Статус Redis сервісу

3.3.2 Налаштування Postgres

Встановимо Postgres для нашого додатку: sudo apt install postgresql. Перевіримо, чи вона працює, запустивши команду(Рис. 3.9).

Рисунок 3.9 – Термінал postgres

Роль postgres, показана в команді, - це загальний обліковий запис адміністратора, створений за замовчуванням у Postgres. В основному ми робимо так, щоб наш вхід в обліковий запис Linux діяв так, ніби це тимчасово обліковий запис postgres, використовуючи -i в команді. -u вказує, яку роль ми використовуємо. Отже, зараз ми працюємо як користувач postgres @, як було показано на скріншоті. Якби ми не працювали як Postgres, нам потрібно було б додати до будь-яких команд Postgres префікс sudo -u postgres. Але оскільки ми виконуємо роль Postgres, ми можемо просто запустити команду(Рис. 3.10). Команда createuser –-interactive створює нового користувача.

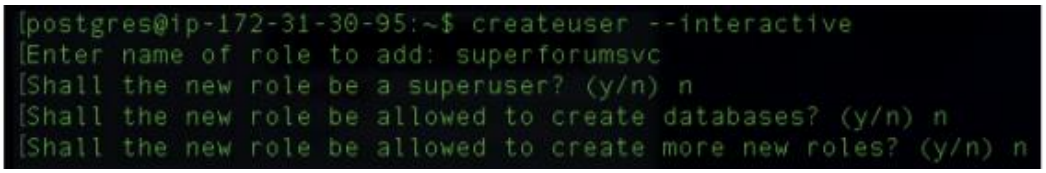

Рисунок 3.10 – Створення нового користувача

Я встановив ім'я користувача на superforumsvc. Тепер ми дамо нашому новому користувачеві пароль - psql. Потім я вводжу запит SQL для зміни пароля користувача superforumsvc. Тепер давайте створимо базу даних для програми. Спочатку закрийте команду psql, а потім створіть базу даних таким чином: \ q createdb -O superforumsvc SuperForum. Ця команда створює базу даних і робить роль superforumsvc власником нової бази даних. Тепер додамо наш ThreadCategory за замовчуванням до нашої бази даних. У проекті супер-форум-сервер ви знайдете файл utils / InsertThreadCategories.txt(Рис. 3.11).

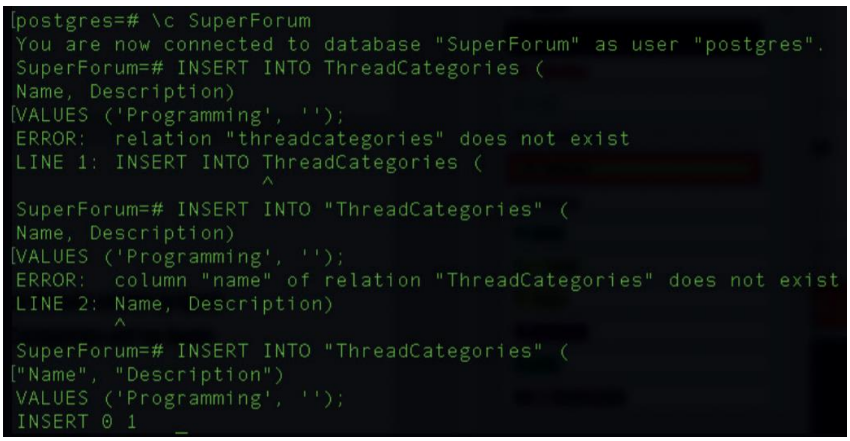

Рисунок 3.11 – Вставка Thread Category

По-перше, ви повинні бути у правильній базі даних. Отже, знову використовуйте \c для цього. Зверніть увагу, що ім’я бази даних чутливе до регістру. Потім переконайтеся, що в іменах таблиць та полів навколо них є подвійні лапки.

3.3.3 Налаштування Node

- Запустіть наступну команду: sudo apt install nodejs;
- Тепер запустіть: node -v, щоб перевірити номер версії вашого Node; Ваша версія Node має бути принаймні 12 або вище. Якщо ні, то виконайте команду: curl -sL https://deb.nodesource.com/setup_12.x | sudo -E bash -, а потім: sudo apt install nodejs;
- Встановити NPM, необхідно виконати команду: sudo apt install npm;
- Тепер потрібно встановити спосіб управління нашим сервером Node, тобто вимкнути його та автоматично перезапустити. Отже, ми будемо використовувати pm2, який на сьогодні є одним з найпопулярніших способів управління Node: sudo npm install -g pm2.

3.4 Налаштування та розгортання програми на NGINX із SSL шифруванням

У цьому розділі встановимо та налаштуємо наш додаток для використання NGINX. NGINX - це дуже популярний високопродуктивний веб-сервер, зворотний проксі-сервер і балансувач навантаження. Його поважають за високу продуктивність, а також за можливості обробляти різні конфігурації для сайтів, що використовують кілька серверів. Ми використаємо його для обслуговування двох сайтів. Один буде обслуговувати наш клієнт React, а інший - наш сервер GraphQL Express. Весь трафік нашого сайту спочатку буде спрямований на NGINX, а потім він перенаправить ці запити у відповідну частину нашої програми. Почнемо з установки NGINX:

- Підключіться по SSH на ваш сервер, як показано раніше в розділі Налаштування Ubuntu Linux на AWS Cloud та запустіть команди для встановлення NGINX: sudo apt update; sudo apt install nginx;
- Тепер, коли встановлено NGINX, давайте створимо папку для зберігання наших серверних файлів: sudo mkdir /var/www/superforum; sudo mkdir /var/www/superforum/server.

Як випливає з назви, каталог /var/www є місцем розташування веб-файлів за замовчуванням.

3.4.1 Створення super-forum-server

У цьому розділі ми створимо процес побудови та розгортання коду нашого сервера. Добре мати стандартизований процес розгортання, щоб ваші розгортання були послідовними та надійними.

Перш ніж ми зможемо розпочати копіювання наших файлів, нам потрібно виконати деякі базові налаштування та побудувати наш серверний проект.

Відкрийте проект super-forum-server у VSCode. Якщо ви подивитесь на розділ сценаріїв файлу package.json, то побачите, що у нас є новий сценарій під назвою build. Це скомпілює наш серверний код і належним чином упакує його для розподілу в папку dist. Тепер, щоб ця команда працювала, нам потрібно буде спочатку встановити деякі пакети NPM глобально. Запустіть команду на машині розробника, а не на сервері Ubuntu: sudo npm i -g del-cli cpy-cli.Пакет del-cli - це універсальна команда видалення з командного рядка. Це означає, що незалежно від того, чи є ваша машина для розробки Linux, Mac або Windows, ця команда буде працювати однаково. Подібним чином пакет cpy-cli дозволяє універсально копіювати файли та папки. Ми використовуємо ці команди, щоб мати одну команду сценарію NPM, яка працюватиме однаково в усіх операційних системах розробників. Сценарій збірки спочатку видаляє папку dist, щоб ми кожного разу починали знову. Потім він копіює вміст dev-config у dist та окремо копіює файл .env у dist. І, нарешті, він запускає компілятор TypeScript. Зауважемо, у нас також є нова папка під назвою dev-config. Ця папка буде містити файли, пов’язані з конфігурацією, які в підсумку будуть скопійовані в нашу папку dist за допомогою сценарію збірки. Файлами в цій папці є файл .env для глобальної конфігурації, файл ormconfig.js, для конфігурації TypeORM та наш файл package.json[15].

На жаль, здається, є якась помилка з останніми пакетами Express NPM, тому нам доведеться встановити ще одну залежність пакета NPM. Запустіть цю команду на машині розробки: npm i -D @types/express-serve-static-core. Зверніть увагу на одне. У файлі super-forum-server/src/index.ts я додав нову функцію loadEnv, розташовану вгорі файлу. Цей файл буде мати справу з відносною різницею у шляху .env файлу між середовищем розробки та сервера, використовуючи змінну Node __dirname.

Зверніть увагу на одне. У файлі super-forum-server/src/index.ts я додав нову функцію loadEnv, розташовану вгорі файлу. Цей файл буде мати справу з відносною різницею .env файлу між середовищем розробки та сервера, використовуючи змінну Node __dirname. Також підправив файл super-forum-

server/dev-config/ormconfig.js, щоб він використовував __dirname для шляху до сутностей TypeORM;

Зараз спробуємо запустити наш сценарій збірки. Запустіть команду на машині розробки: npm run build. Запуск цього повинен створити папку dist(Рис. 3.12);

Рисунок 3.12 – Створення папки dist

Тепер спробуємо скопіювати нашу папку dist на наш сервер. На терміналі вашої машини розробки запустіть цю команду з відповідними вам конфігураціями: scp -i <you pem path> -r <source folder>/* <username>@<ip>:<dest folder>. Знову увійдіть до сеансу Ubuntu SSH і запустіть наступну команду: sudo chmod -R 777 /var/www/superforum/server;

Тепер скопіюйте файли, використовуючи ту саму команду scp з вашого терміналу розробки(Рис. 3.13);

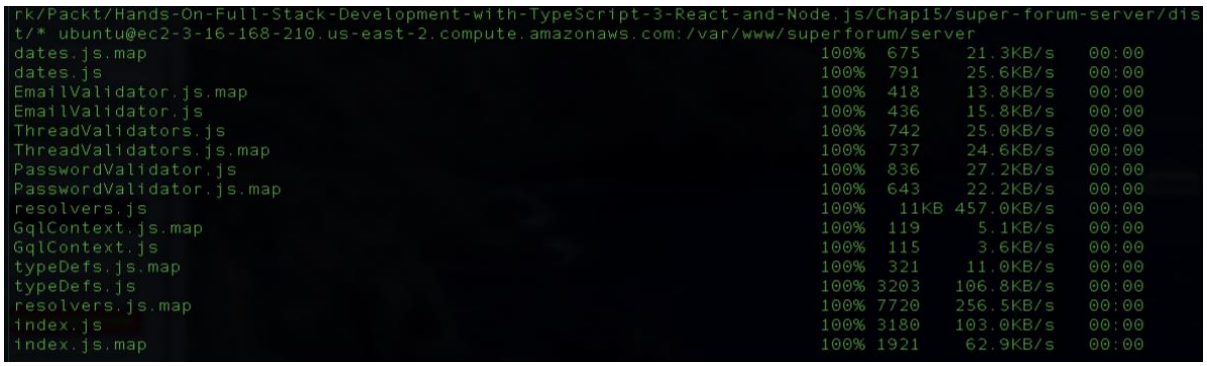

Рисунок 3.13 – scp копіювання

Тепер перевірте, чи всі файли конфігурації скопійовані на сервер, заглянувши в папку сервера(Рис. 3.14).

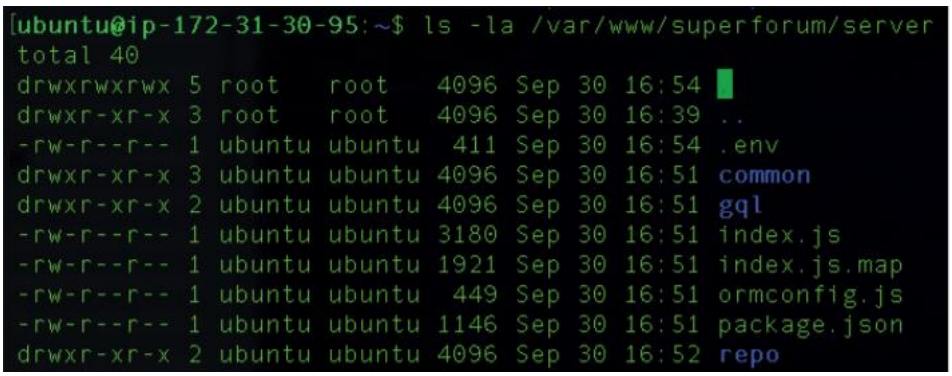

Рисунок 3.14 – Перевірка папки сервера

Якщо файл .env відсутній, потрібно вручну скопіювати його за допомогою наступної команди. Зверніть увагу, це проблема з Mac, де файли .env з певних причин не видно: scp -i <your pem path> <your path>/.env <username><yourserverpath>/.env. Тепер ми повинні знову закрити наші дозволи за допомогою цієї команди: sudo chmod -R 755 /var/www/superforum/server;

Тепер, на термінальному сеансі SSH на сервері Ubuntu, cd /var/www/ superforum/server: npm install;

Тепер нам потрібно налаштувати нашу систему pm2, щоб вона керувала нашим сервером Node. Виконайте цю команду: pm2 startup; Ця команда повідомляє нам конкретні налаштування, які потрібні нашому поточному користувачеві для того, щоб налаштувати використання pm2 з systemd та запуск нашого Node-сервера під час перезапуску сервера. Systemd знову є нашим контролером служб Ubuntu(Рис. 3.15)

Рисунок 3.15 – pm2 запуск

Отже, скопіюйте та вставте команду, починаючи з sudo, а потім запустіть її у своєму SSH-сеансі вашого сервера Ubuntu(Рис. 3.16)

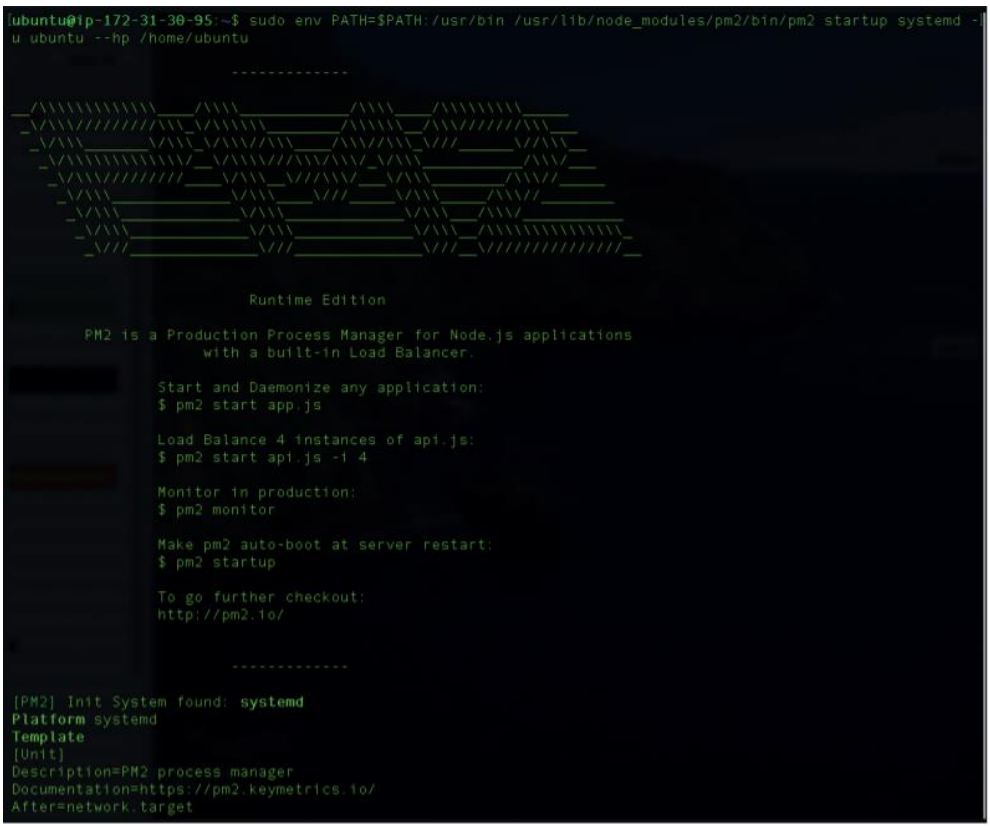

Рисунок 3.16 - Результат запуску запуску pm2

Далі необхідно запустити наш сервер Node наступним чином(Рис. 3.17).

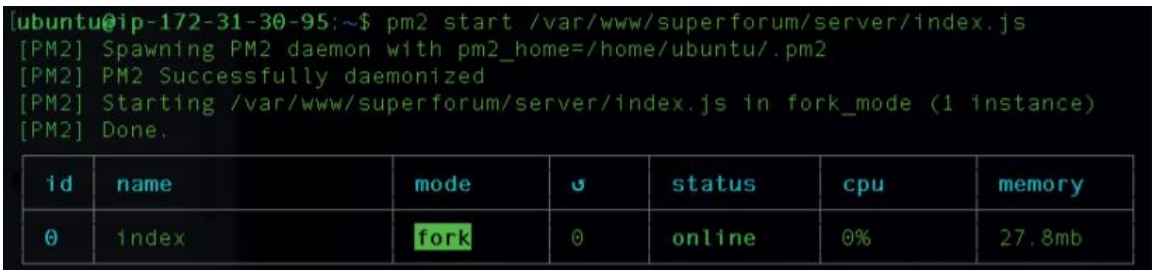

Рисунок 3.17 – Запуск Node-server

Тепер ми можемо зберегти це як частину списку запуску pm2, виконавши команду: pm2 save(Рис. 3.18).

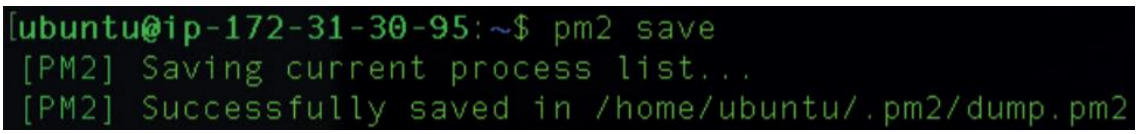

Рисунок 3.18 – Запуск pm2 save

Виконуючи цю операцію збереження, наш сервер Node тепер запускає автоматично при кожному перезавантаженні нашого сервера. У цьому розділі ми створили процес побудови, розгортання та запуску нашого сервера Node.

3.4.2 Створення super-forum-client

Зараз ми маємо зробити подібний процес для нашого проекту на стороні клієнта. Вам слід скопіювати super-forum-client у вашу папку, оскільки це перше, що ми зробили на початку цього розділу.

Тепер необхідно повернутись до сеансу терміналу SSH на сервері Ubuntu і створити папку для клієнтського проекту наступним чином: sudo mkdir /var/www/superforum/client.

Тепер поверніться до вашого терміналу розробки в папці проекту, щоб ми змогли побудувати та розгорнути клієнт. По-перше, нам потрібно виконати незначну корекцію нашого проекту. Ви побачили, що наш серверний проект використовує файл .env для налаштувань. Нам не потрібно нічого, що стосується проекту нашого клієнта. Однак ми повинні мати можливість принаймні встановити URL-адресу сервера GraphQL за необхідності, залежно від середовища розгортання. Відкрийте index.ts через VSCode and обновіть ApolloClient: const client = new ApolloClient({ uri: process.env.REACT_APP_GQL_URL, credentials: "include", cache: new InMemoryCache({ resultCaching: false, }), }). Як бачите, ми додали змінну середовища REACT_APP_GQL_URL, як і те, як ми це робили на своєму сервері.

Отже, давайте запустимо скрипт build-dev: npm run build-dev. Сценарій збірки в клієнтському проекті вже створений для нас як частина createreact-app, але ми змінили його, додавши змінну середовища. Після завершення ви побачите цю папку з назвою build(Рис. 3.20).

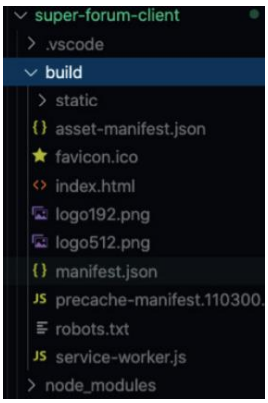

Рисунок 3.20 - Створення папки super-forum-client

Тепер нам просто потрібно тимчасово відкрити папку клієнта сервера, щоб ми могли скопіювати: sudo chmod -R 777 /var/www/superforum/client. Тепер ми можемо розгортати наші файли збірки на стороні клієнта. З вашого терміналу розробки запустіть цю команду, звичайно, зі своїми власними шляхами: scp -i <your pem path> -r <your path><username><yourserverpath>(Рис. 3.21). Та необхідно надати доступ: sudo chmod -R 755 /var/www/superforum/client.

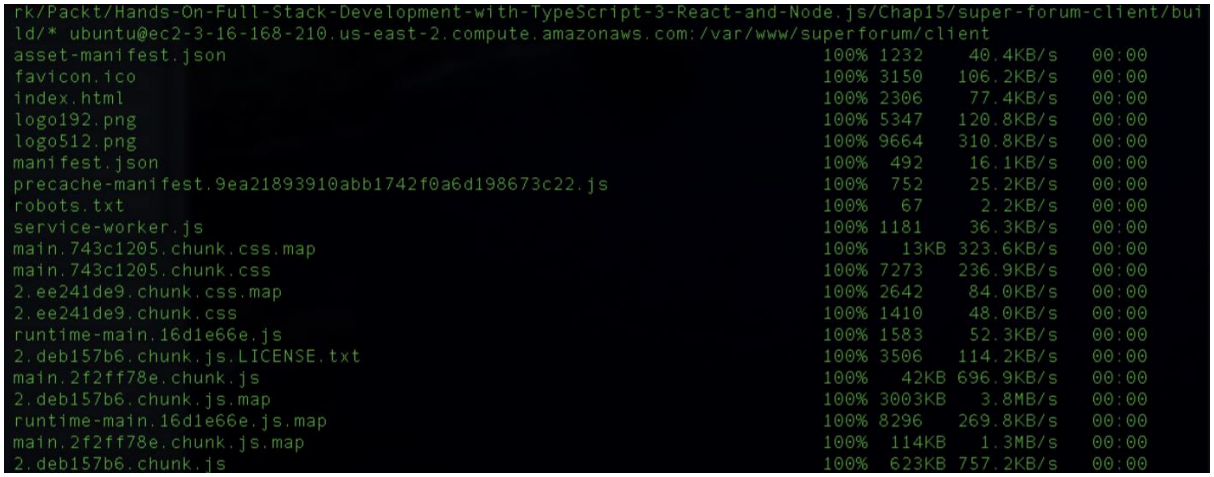

Рисунок 3.21 – Копіювання клієнтських файлів на сервер

3.4.3 Налаштування Nginx

Отже ми зробили багато налаштування нашої збірки сервера, тому тепер ми можемо продовжити, налаштувавши наш встановлений сервер NGINX.

Нам потрібно запустити NGINX під час запуску системи на нашому сервері Ubuntu. Запустіть команду, показану на вашому терміналі SSH, а потім автентифікуйтесь(Рис. 3.22).

Рисунок 3.22 – Ввімкнення NGINX

Тепер перевірте, чи працює NGINX за допомогою команди стану: systemctl status nginx. Тепер нам потрібно відкрити порт 80 на нашому брандмауері AWS VM. Відкрийте браузер на порталі AWS, а потім виберіть Групи безпеки в меню Мережа

та безпека. Тепер виберіть групу, яка не за замовчуванням. Виберіть кнопку редагувати правила вхідних повідомлень, а потім на наступному екрані натисніть кнопку додати правила. Додайте нове правило вхідного зв'язку для HTTP(Рис. 3.23). Вибравши 0.0.0.0/0, ви дозволяєте будь-які IP-адреси, зберегти правила.

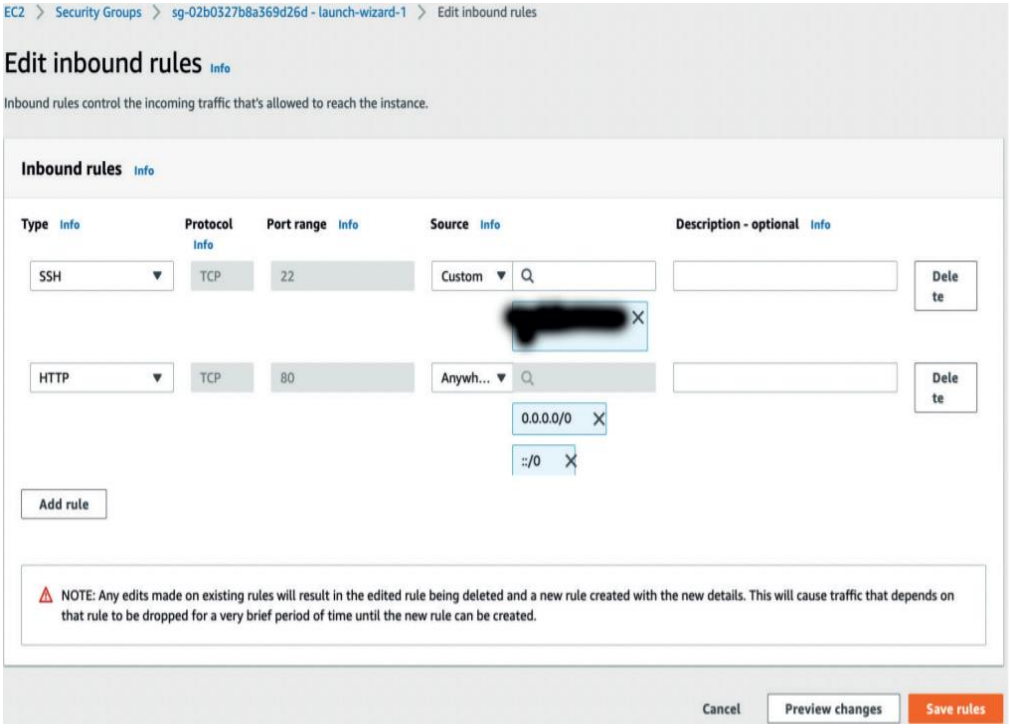

Рисунок 3.23 – Редагування HTTP правила

Зазвичай локальний брандмауер Ubuntu не вмикається. Однак, якщо він ввімкнений, нам також потрібно пропустити трафік до NGINX на брандмауері. Виконайте команду: sudo ufw allow 'Nginx HTTP', sudo ufw status. Запуск попередньої команди повинен дати такий результат(Рис. 3.24).

Рисунок 3.24 – Перевірка правил доступів

Тепер, якщо ми переходимо до нашої URL-адреси разом із нашим браузером, у моєму випадку це ec2-3-16-168-210. us-east-2.compute.amazonaws.com, ваш буде іншим(Рис. 3.25).

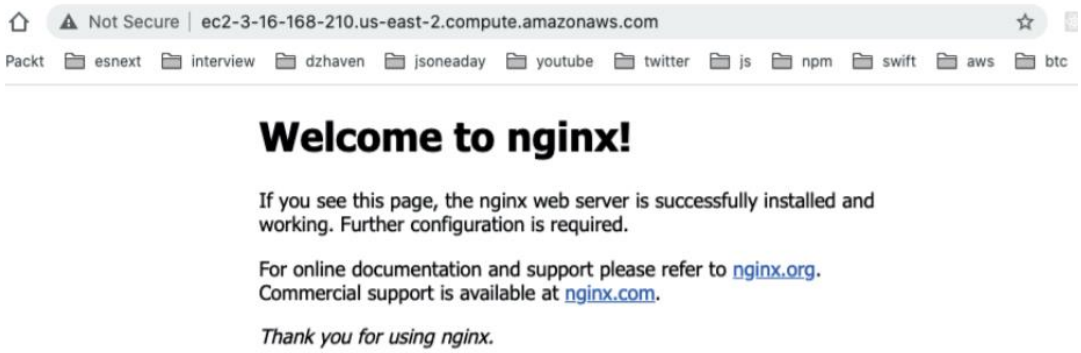

Рисунок 3.25 – Сторінка NGINX за замовчуванням

Отже, очевидно, що наш NGINX встановлений і працює. Тож зараз нам потрібно зробити так, щоб він обслуговував наш сайт. NGINX має два варіанти налаштування сайтів. Один дозволяє нам використовувати файл конфігурації в папці /etc/nginx/conf.d. Інший, який називається Блоки сервера, використовує папку /etc/nginx/sites-available. Ми будемо використовувати метод conf.d. Запустіть: sudo nano /etc/nginx/conf.d/superforum.conf. Тепер це те, що повинен містити ваш файл, знову ж таки із вашими власними шляхами до папки та доменним іменем(Рис. 3.26).

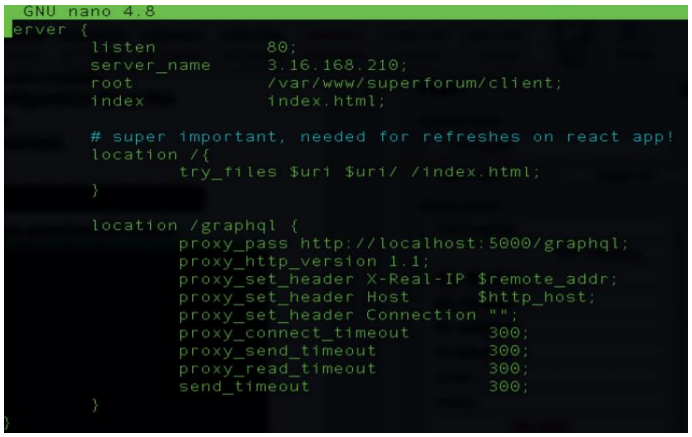

Рисунок 3.26 – Файл конфігурації NGINX

3.4.4 Налаштування SSL шифрування за допомогою Nginx

Установка SSL-сертифіката і перехід до протоколу HTTPS - відмінний спосіб підвищити безпеку з'єднань між браузером і сервером. SSL-з'єднання є вкрай рекомендованим для всіх сайтів, де потрібно для користувача реєстрація. Попри те, що говорять багато застарілі інструкції і керівництва, талон не додає ніякої серйозної навантаження на сервер, і його можна легко і недорого випустити.

Перше, що потрібно зробити ще до того, як переходити до налаштувань Nginx - це замовити SSL-сертифікат. На вибір пропонуються найрізноманітніші сертифікати, серед яких можна особливо виділити наступні:

- EssentialSSL - базовий SSL-сертифікат, що дозволяє перейти до HTTPS за найнижчою ціною. Проводиться тільки перевірка по домену;
- InstantSSL - «золота середина». SSL-сертифікат, добре підходить різним компаніям і організаціям;
- EV SSL - SSL-сертифікат, що гарантує максимальну надійність і захищеність з'єднання. Дозволяє домогтися зеленої рядки в браузері, яка в очах користувачів є доказом авторитетності і безпеки ресурсу.

Тепер, коли у вас є на руках сертифікат, ви можете переходити до технічної сторони - налаштування SSL на Nginx.

Для установки SSL на Nginx потрібно буде виконати наступні кроки.Ви повинні отримати архів з сертифікатом на свою поштову адресу, яка була вказана при замовленні SSL-сертифіката. Архів зазвичай зберігає в собі кореневої, проміжний сертифікат, а також сертифікат для домену. В іншому випадку вам потрібно буде об'єднати в один файл три файли сертифікатів.

У вас повинно бути три файли: comodo_root.crt (кореневий сертифікат), comodo_intermediate.crt (проміжний сертифікат) і vash_domen.crt. Скопіюйте їх разом з файлом .key в довільний каталог на своєму сервері, в якому ви хочете їх зберігати. Вам потрібно буде об'єднати файл з кореневим сертифікатом (comodo_root.crt), файл з проміжним сертифікатом (comodo_intermediate.crt) і файл сертифіката vash_domen.crt в один crt-файл. Робиться це за допомогою наступної команди: cat comodo_root.crt comodo_intermediate.crt vash_domen.crt> bundle.crt

У деяких випадках Comodo надсилають архів, в якому файли проміжного сертифіката і кореневого сертифіката вже об'єднані в один файл. Якщо так, то вам потрібно буде об'єднати його з файлом сертифікату за допомогою наступної команди: cat comodo-bundle.crt vash_domen.crt> bundle.crt.

Переходимо до налаштування Nginx. Щоб згенерувати конфігураційний файл для веб-сервера, можна скористатися сервісом https://mozilla.github.io/server-side-tls/ssl-config-generator/.

Відкриваємо файл віртуального хоста Nginx для сайту, який ви хочете захистити. Якщо ви хочете, щоб ваш сайт був доступний і через незахищене, і через захищене з'єднання, вам потрібно буде додати модуль сервера для кожного типу підключення. Зробіть копію існуючого модуля для незахищеного з'єднання і вставте його під основним кодом. Після чого додайте у файл налаштування вашого сертифікату. В моєму випадку я вимкну сертифікат, щоб не оплачувати вартість послуги.

Давайте подивимось, чи з’являється наш додаток у браузері. По-перше, зупинимо наш сервер Node і перезапустимо його, не використовуючи pm2, щоб ми могли побачити будь-які помилки, які можуть виникати. Запустіть ці команди на своєму терміналі Ubuntu SSH: pm2 stop index, node /var/www/superforum/server/index.js(Рис. 3.27).

Рисунок 3.27 – Запуск Node сервера

Тепер відкрийте браузер і перейдіть на свою IP-адресу, вказану AWS. Потім натисніть кнопку зареєструватись і давайте зареєструємо нового користувача(Рис. 3.28).

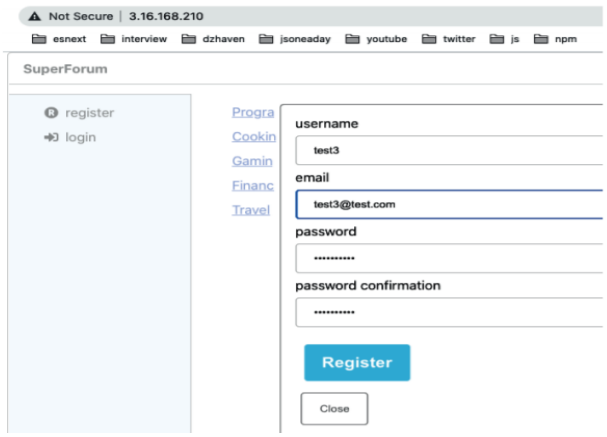

Рисунок 3.28 – Реєстрація нового користувача

Заповніть значення, як вважаєте за потрібне, і натисніть кнопку Зареєструвати. Тепер нам потрібно підтвердити нашого нового користувача. Запустіть команди на терміналі Ubuntu SSH: sudo -u postgres psql, \c SuperForum Update "Users" set "Confirmed" = true. Тепер спробуємо увійти в систему з нашим новим користувачем(Рис. 3.29).

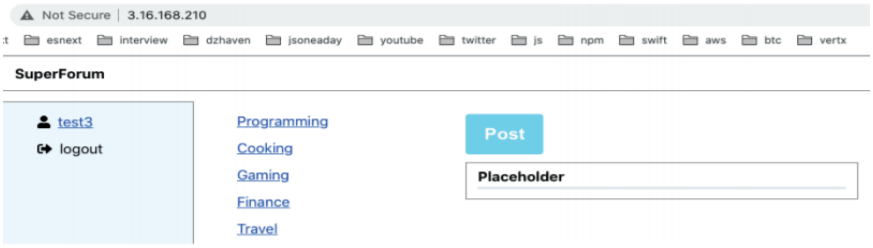

Рисунок 3.29 – Логін нового користувача

Звичайно, на даний момент у нас немає даних, тому тепер ми додамо одну публікацію теми(Рис. 3.30).

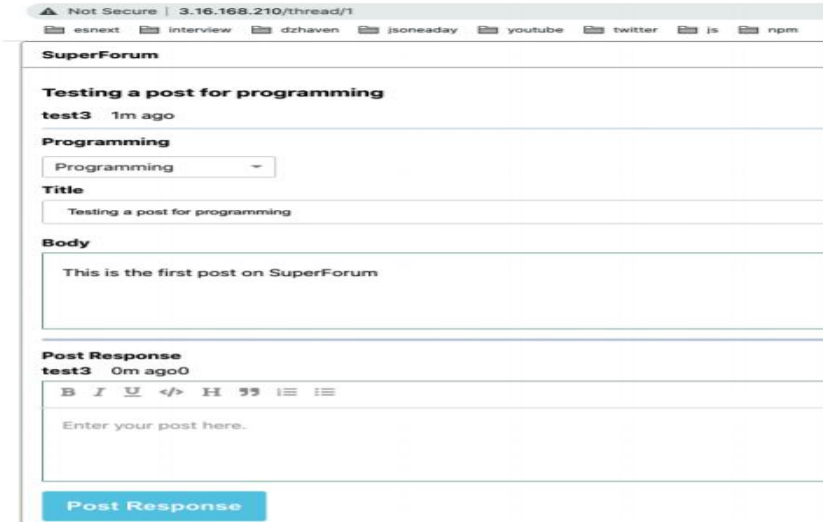

Рисунок 3.30 – Публікація теми

Тепер перейдемо на домашню сторінку нашого додатку(Рис. 3.31).

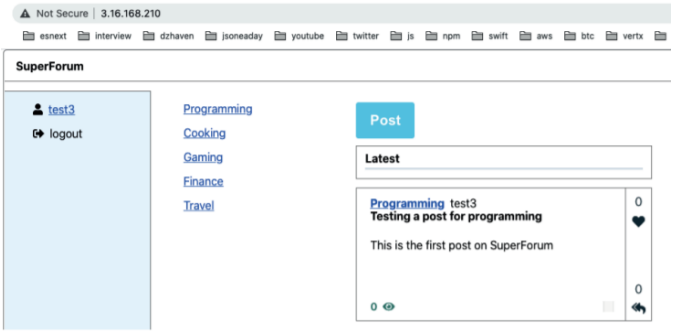

Рисунок 3.31 - Головний екран після першої публікації

3.5 Висновки до розділу 3

У цьому розділі ми завершили налаштування нашої програми за допомогою NGINX та всіх інших наших служб. У цьому розділі ми закріпили свої знання про веб-розробку за допомогою React, Node та GraphQL, нарешті, розгорнувши наш додаток до хмари. Розгорнули наш додаток на хмарі AWS та зрозуміли наскільки на сьогоднішній день це найпопулярніший і широко використовуваний хмарний сервіс. Крім того, зробити це за допомогою NGINX було правильним кроком, оскільки NGINX є дуже продуктивним і надзвичайно популярним у спільноті Node та встановили SSL-сертифікат на Nginx.

ПЕРЕЛІК ПОСИЛАНЬ

1. cyberleninka.ru/ [Електронний ресурс]. – Режим доступу: https://cyberleninka.ru/article/n/rol-udovletvorennosti-trudom-personala-vreshenii-upravlencheskih-zadach-meditsinskoy-organizatsii/viewer
2. hr-director.ru [Електронний ресурс]. – Режим доступу: https://www.hr-director.ru/article/63014-otsenka-udovletvorennosti-personalauznaem-nastroeniya-rabotnikov
3. Дмитрий Котеров, Алексей Костарев PHP. В подлиннике. / Д. Котеров, А. Костарев — Спб.: «БХВ-Петербург», 2005. — 1120 с.
4. lifewire.com [Електронний ресурс]. – Режим доступу: https://www.lifewire.com/what-is-a-web-application-3486637
5. habr.com [Електронний ресурс]. – Режим доступу: https://habr.com/ru/post/450282/
6. en.yeeply.com [Електронний ресурс]. – Режим доступу: https://en.yeeply.com/blog/advantages-and-disadvantages-of-web-appdevelopment/
7. Колисниченко Д. Н. Самоучитель PHP 5 / Д. Н. Колисниченко — СПб.: Наука и Техника, 2007. — 640 с.
8. slideshare.net [Електронний ресурс]. – Режим доступу: https://www.slideshare.net/ssusere5f319/web-66422826
9. wikipedia.org [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/Доменна_система_імен
10. Кейт Джонс. DOM Scripting: Web Design with JavaScript and the Document Object Model. / К. Джонс — Перше, 2005. — 368 с.
11. Что такое SSL-сертификат | Кому и зачем он нужен [Електронний ресурс]. - https://ssl.com.ua/info/what-is-ssl/
12. wikipedia.org [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/PHP 93
13. Современный учебник JavaScript [Електронний ресурс]. – Режим доступу: https://learn.javascript.ru/
14. habr.com [Електронний ресурс]. – Режим доступу: https://habr.com/ru/post/464417/
15. web-creator.ru [Електронний ресурс]. – Режим доступу: https://web-creator.ru/articles/yii

Міністерство освіти і науки України
Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра мережевих та інтернет технологій

ПОДАННЯ
ГОЛОВІ ДЕРЖАВНОЇ ЕКЗАМЕНАЦІЙНОЇ КОМІСІЇ
ЩОДО ЗАХИСТУ БАКАЛАВРСЬКОЇ РОБОТИ

Направляється студент Бойко Д.В. до захисту бакалаврської роботи за спеціальністю 172 «Телекомунікації та радіотехніки» на тему: «Методика створення та розгортання програм на платформі AWS з використанням шифрування SSL».
Бакалаврська робота і рецензія додаються.

Декан факультету _____ В.Є. Снитюк
(підпис)

Довідка про успішність

Бойко Д.В. за період навчання на факультеті інформаційних технологій, з 2016 року до 2020 року повністю виконав навчальний план за спеціальністю 172 «Телекомунікації та радіотехніка», з таким розподілом оцінок за:
національною шкалою: відмінно ____%, добре ____%, задовільно ____%;
шкалою ECTS: A ____%; B ____%; C ____%; D ____%; E ____%.

Методист факультету _____ _____
(підпис) (прізвище та ініціали)

Висновок керівника бакалаврської роботи

Студент **Бойко Дмитро Володимирович**
Виконання кваліфікаційної роботи продемонструвало відмінну теоретичну та практичну підготовку здобувача, різнобічне знання матеріалу, вміння самостійно вирішувати професійні задачі, знаходити відповіді на фахові питання і робити відповідні висновки. Роботу виконував старанно, сумлінно, відповідально, виважено та аргументовано підходив до вирішення поставлених завдань.
Все це дозволяє оцінити виконану бакалаврську роботу студента **Бойка Дмитра Володимировича** на оцінку «_____» та присвоїти йому кваліфікацію «бакалавр».

Керівник роботи _____ К.В. Герасименко
(підпис)
«_____» _____ 2021 року

Висновок кафедри про бакалаврську роботу

Бакалаврську роботу розглянуто. Студент **Бойко Дмитро Володимирович** допускається до захисту даної роботи в Державній екзаменаційній комісії.

Завідувач кафедри мережевих та інтернет технологій _____ Ю.В. Кравченко
(підпис)

«____»_____2021 року

РЕЦЕНЗІЯ

на бакалаврську кваліфікаційну роботу
студента факультету інформаційних технологій
Київського національного університету імені Тараса Шевченка

Бойка Дмитра Володимировича
представленої на здобуття освітнього – кваліфікаційного рівня «бакалавр»
- галузі знань 12 **Інформаційні технології**
- за спеціальністю 172 **«Телекомунікації та радіотехніка»**

Бойко Дмитро Володимирович виконав бакалаврську кваліфікаційну роботу на тему «Методика створення та розгортання програм на платформі AWS з використанням шифрування SSL».

Актуальність теми. Створення та розгортання програм грають дуже важливу роль у інформаційних, державних та багатьох різних сферах. Актуальність теми роботи полягає в тому, що стрімке впровадження цифрових технологій задля оптимізації процесу рішень задач сучасного бізнесу, створило великий попит на високопродуктивні спеціалізовані програми, які розгортають на хмарному провайдері. Під час розв'язання поставленої задачі проведено аналіз існуючих видів баз даних та можливостей їх застосування до реалізації цієї задачі, розглянуто та проаналізовано сучасні системи управління базами даних, що дозволило реалізувати прототип бази даних в системі управління базами даних з Postgres із використанням SSL шифруванням.

Напрям досліджень обумовив структуру роботи, яка складається зі вступу, трьох розділів, загальних висновків, списку використаних джерел.

Зауваження до змісту та оформлення роботи. Зауважень до роботи не виявлено. Недоліком є те, що не було формул. Роботу оформлено якісно. Зміст містить інформацію суто по актуальності теми.

Пояснювальна записка оформлена грамотно, логічно, послідовно в доступній для розуміння формі.

Висновок. Тема «Методика створення та розгортання програм на платформі AWS з використанням шифрування SSL» є актуальною і має науковий інтерес. Атестаційна робота виконана якісно, відповідно до затвердженого завдання і заслуговує оцінки «відмінно», а автору роботи може бути присвоєний освітнього-кваліфікаційного рівень «бакалавр» за спеціальністю 172 «Телекомунікації та радіотехніка».

Рецензент дипломної роботи — доктор технічних наук, доцент,
завідувач кафедри
Кравченко Ю.В.

ВІДГУК

на бакалаврську кваліфікаційну роботу
студента факультету інформаційних технологій
Київського національного університету імені Тараса Шевченка

Бойка Дмитра Володимировича
представленої на здобуття освітнього – кваліфікаційного рівня «бакалавр»
- галузі знань 12 **Інформаційні технології**
- за спеціальністю 172 **«Телекомунікації та радіотехніка»**

Бойко Дмитро Володимирович виконав бакалаврську кваліфікаційну роботу на тему «Методика створення та розгортання програм на платформі AWS з використанням шифрування SSL».

Актуальність теми. Створення та розгортання програм грають дуже важливу роль у інформаційних, державних та багатьох різних сферах. Актуальність теми роботи полягає в тому, що стрімке впровадження цифрових технологій задля оптимізації процесу рішень задач сучасного бізнесу, створило великий попит на високопродуктивні спеціалізовані програми, які розгортають на хмарному провайдері. Під час розв'язання поставленої задачі проведено аналіз існуючих видів баз даних та можливостей їх застосування до реалізації цієї задачі, розглянуто та проаналізовано сучасні системи управління базами даних, що дозволило реалізувати прототип бази даних в системі управління базами даних з Postgres із використанням SSL шифруванням.

Напрям досліджень обумовив структуру роботи, яка складається зі вступу, трьох розділів, загальних висновків, списку використаних джерел.

Зауваження до змісту та оформлення роботи. Зауважень до роботи не виявлено. Недоліком є те, що не було формул. Роботу оформлено якісно. Зміст містить інформацію суто по актуальності теми.

Пояснювальна записка оформлена грамотно, логічно, послідовно в доступній для розуміння формі.

Висновок. Тема «Методика створення та розгортання програм на платформі AWS з використанням шифрування SSL» є актуальною і має науковий інтерес. Атестаційна робота виконана якісно, відповідно до затвердженого завдання і заслуговує оцінки «відмінно», а автору роботи може бути присвоєний освітнього-кваліфікаційного рівень «бакалавр» за спеціальністю 172 «Телекомунікації та радіотехніка».

Науковий керівник асистент кафедри,
Герасименко К.В.