Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
(повна назва)

Кафедра програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Дослідження та розробка стратегії оптимізації та підвищення продуктивності веб-додатків на базі NodeJS та React з використанням штучного інтелекту
(тема)

Виконав:
студент 2 курсу, групи ІПЗм-22-2

Дробицький Д.С.
(прізвище, ініціали)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)
Тип програми освітньо-наукова

Керівник к.т.н., доц. каф. ПІ Вечур О. В.
(посада, прізвище, ініціали)

Допускається до захисту
Зав. кафедри _____ З.В.Дудар
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерних наук
Кафедра програмної інженерії
Рівень вищої освіти другий (магістерський)
Спеціальність 121 – Інженерія програмного забезпечення
Тип програми освітньо-наукова програма
Освітня програма Інженерія програмного забезпечення
(шифр і назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«___» _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Дробицькому Даниїлу Сергійовичу
(прізвище, ім’я, по батькові)

1. Тема роботи «Дослідження та розробка стратегії оптимізації та підвищення продуктивності веб-додатків на базі NodeJS та React з використанням штучного інтелекту»
Затверджена наказом по університету від 29.03.2024р. № 250Ст

2. Термін подання студентом роботи до екзаменаційної комісії 20.06.2024

3. Вихідні дані до роботи електронні ресурси за обраною тематикою, технології AI, патерни оптимізації коду, електронні ресурси за обраною тематикою, пояснювальна записка, React, NodeJS, середовище розробки VS CODE

4. Перелік питань, що потрібно опрацювати в роботі
аналіз проблемної області, етапи проектування планування експерименту, дослідження технологій штучного інтелекту, вивчення можливостей його використання, архітектура програмного забезпечення, проведення експериментів та аналіз отриманих результатів

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Отримання завдання	01.04.2024	*виконано*
2	Аналіз предметної галузі та постановка задачі	01.04 – 10.04.24	*виконано*
3	Аналіз та вибір алгоритмів для дослідження	10.04 – 14.04.24	*виконано*
4	Аналіз та моделювання предметної області	14.04 – 20.04.24	*виконано*
5	Планування експериментів	20.04 – 29.04.24	*виконано*
6	Програмна реалізація кожного з обраних для дослідження API	29.04 – 10.05.24	*виконано*
7	Експериментальні дослідження	10.05 – 14.05.24	*виконано*
8	Аналіз результатів експериментальних досліджень	14.05 – 16.05.24	*виконано*
10	Підготовка пояснювальної записки	16.05 – 25.05.24	*виконано*
11	Підготовка презентації та доповіді	25.05 – 30.05.24	*виконано*
12	Перевірка на плагіат	30.05 – 08.06.24	*виконано*
13	Нормоконтроль	03.06 – 08.06.24	*виконано*
14	Рецензування	08.06 – 12.06.24	*виконано*
15	Попередній захист	12.06.2024	*виконано*
16	Занесення диплома в електронний архів	15.06.2023	*виконано*
17	Допуск до захисту у зав. кафедри	18.06.2024	*виконано*

Дата видачі завдання 01 квітня 2024р.

Студент (ка) _____ Дробицький Д.С.
(підпис)

Керівник роботи _____ к.т.н., доц. каф. ПІ Вечур О. В.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Пояснювальна записка містить: 53 с., 11 рис., 2 табл., 6 джерел.

ТЕХНОЛОГІЇ ОПТИМІЗАЦІЇ ТА ПІДВИЩЕННЯ ПРОДУКТИВНОСТІ ВЕБ-ДОДАТКІВ, NODEJS, REACT, ШТУЧНИЙ ІНТЕЛЕКТ, AWS, СЕРВЕРНІ ТЕХНОЛОГІЇ, ML.

Об’єктом дослідження є технології оптимізації та підвищення продуктивності веб-додатків на базі NodeJS та React з використанням штучного інтелекту.

Метою роботи є дослідження та аналіз ефективності та оптимізації існуючих технологій для веб-додатків, виділення критеріїв та методів для проведення порівняльного аналізу.

У результаті роботи розглянуто існуючі технології та протоколи для оптимізації та підвищення продуктивності веб-додатків, досліджено їх особливості, переваги та недоліки та принципи роботи, описано та продемонстровано методи порівняння та запропоновано формули для обчислення числових показників.

TECHNOLOGIES FOR OPTIMIZING AND IMPROVING THE PERFORMANCE OF WEB APPLICATIONS, NODEJS, REACT, ARTIFICIAL INTELLIGENCE, AWS, SERVER TECHNOLOGIES, ML.

The object of study is technologies for optimizing and improving the performance of web applications based on NodeJS and React using artificial intelligence.

The purpose of the study is to investigate and analyze the efficiency and optimization of existing technologies for web applications, to identify criteria and methods for comparative analysis.

As a result of the work, the existing technologies and protocols for optimizing and improving the performance of web applications are considered, their features, advantages

and disadvantages, and principles of operation are investigated, comparison methods are described and demonstrated, and formulas for calculating numerical indicators are proposed.

Заява щодо самостійного виконання кваліфікаційної роботи та можливості її публікації в електронному архіві відкритого доступу ElArKhNURE.

Я, Дробицький Даниїл Сергійович, студент гр. ІПЗм-22-2, здобувач вищої освіти на другому (магістерському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Дослідження та розробка стратегії оптимізації та підвищення продуктивності веб-додатків на базі NodeJS та React з використанням штучного інтелекту», що буде представлена в екзаменаційну комісію для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElArKhNURE. Всі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений(на) з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

ЗМІСТ

Вступ 8

1 Аналіз предметної галузі 10

1.1 Основні відомості 11

1.2 Види оптимізацій за допомогою ШІ 12

1.3 ARM сервіси 13

1.4 GitHub Copilot 14

1.5 Огляд RedisAI 15

1.5 Постановка задачі 16

2 Опис прийнятих проєктних рішень 21

2.1 Вибір технологій 21

2.2 Бібліотеки та інструменти 21

2.3 Шарова структура ML 22

3 Опис програмної реалізації 24

3.1 Архітектура застосунку 24

3.2 Реалізація ML моделі та предиктів 26

3.3 Реалізація функції передбачення оптимального розміру ресурсів 30

4 Опис експериментальних дослІджень 32

4.1 Підготовка даних та налаштування 32

4.2 Тренування ML 35

4.3 Перевірка передбачень навантаження 38

4.4 Аналіз отриманих результатів 42

Висновки 45

Перелік джерел посилання 46

ВСТУП

У сучасному світі розробка веб-додатків на основі технологій NodeJS та React є невід'ємною частиною цифрової парадигми, а їх продуктивність та оптимізація стають стратегічно важливими завданнями. Додавання штучного інтелекту (ШІ) до цього екосистеми може взаємодіяти з існуючими технологіями, сприяючи їхньому поліпшенню та розвитку[1].

Однією з ключових проблем, яку вирішує ця стратегія, є підвищення продуктивності веб-додатків. Використання ШІ дозволяє автоматизувати та оптимізувати процеси, що забезпечує більш ефективне використання ресурсів та зменшення часу відгуку системи.

Одним із напрямків дослідження є використання ШІ для аналізу та передбачення патернів використання додатків, що дозволяє оптимізувати архітектуру та ресурсне споживання. Застосування алгоритмів машинного навчання, зокрема на базі NodeJS та React, може поліпшити реакцію системи на змінні умови експлуатації.

Додатково, стратегія оптимізації може включати в себе розробку інтелектуальних алгоритмів для управління ресурсами, такими як пам'ять та обчислювальна потужність. Це дозволить системам адаптуватися до змінного навантаження та забезпечувати стабільну продуктивність.

Використання ШІ також може покращити інтерфейс користувача веб-додатків, забезпечуючи персоналізовані та контекстно-залежні взаємодії. Автоматизована аналітика та передбачення можуть допомогти в підвищенні якості обслуговування та задоволення потреб користувачів.

В цілому, дана стратегія поєднує сучасні технології веб-розробки з передовими методами штучного інтелекту для досягнення максимальної продуктивності та ефективності веб-додатків. Дослідження цього підходу дозволить розширити можливості веб-розробки та підняти якість веб-додатків на новий рівень.

Метою дослідження є пошук оптимального алгоритму ШІ, який забезпечить

високу продуктивність та ефективність при мінімальному споживанні ресурсів.

Дослідження методів та інструментів ШІ, спрямованих на оптимізацію продуктивності веб-додатків, є важливим кроком у напрямку підвищення їх якості та надійності. Результати цього дослідження мають значний потенціал для вдосконалення процесів розробки та успішного впровадження веб-додатків на ринку.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

У цьому розділі проаналізуємо використання штучного інтелекту (ШІ) у сфері веб-розробки, зосереджуючись на платформах Node.js та React. Ми розглянемо ключові завдання, які можна вирішити за допомогою ШІ, та переваги, які він може принести у контексті оптимізації продуктивності та адаптивного управління ресурсами.

Одним з основних напрямків використання ШІ у веб-розробці є оптимізація продуктивності. Завдяки здатності ШІ аналізувати великі обсяги даних та виявляти приховані закономірності, ми можемо використовувати його для виявлення вузьких місць у роботі веб-додатків, прогнозування майбутнього навантаження та автоматичного налаштування параметрів системи для досягнення максимальної ефективності.

Іншим важливим аспектом є адаптивне управління ресурсами. ШІ може допомогти нам розподіляти ресурси сервера (процесорний час, пам'ять, мережева пропускна здатність) більш ефективно, враховуючи поточний рівень навантаження та прогнози на майбутнє. Це дозволяє уникнути як перевантаження сервера, так і його недовикористання, що позитивно впливає на продуктивність та надійність веб-додатків.

Для досягнення цих цілей, ми розглянемо різні алгоритми машинного навчання, такі як регресія, дерева рішень, нейронні мережі та інші. Кожен з цих алгоритмів має свої сильні та слабкі сторони, і вибір оптимального алгоритму залежить від конкретних завдань та особливостей веб-додатку.

Ми проаналізуємо, як різні алгоритми впливають на ефективність та надійність веб-додатків, які метрики продуктивності можна покращити за їх допомогою, та які ризики та обмеження слід враховувати при їх застосуванні.

Крім того, ми розглянемо питання інтеграції ШІ у існуючі веб-додатки, розроблені на Node.js та React. Це включає в себе вибір відповідних бібліотек та фреймворків, налаштування процесу навчання моделі та її розгортання на сервері.

Особливу увагу ми приділимо пошуку оптимального алгоритму ШІ, який забезпечить високу продуктивність та ефективність при мінімальному споживанні

ресурсів. Це особливо важливо у контексті хмарних обчислень, де вартість ресурсів може бути значною.

У результаті, це дослідження дозволить нам отримати глибоке розуміння потенціалу ШІ у сфері веб-розробки та розробити рекомендації щодо його ефективного застосування для оптимізації продуктивності та адаптивного управління ресурсами веб-додатків на Node.js та React.

1.1 Основні відомості

Дослідження та розробка стратегії оптимізації та підвищення продуктивності веб-додатків на базі NodeJS та React з використанням штучного інтелекту є важливим і актуальним напрямом у сучасній веб-розробці. Ця галузь охоплює ряд дій та методологій, спрямованих на підвищення ефективності роботи веб-додатків шляхом використання сучасних технологій та методів машинного навчання.

Основною метою даної галузі є створення ефективних, швидкодіючих та надійних веб-додатків, які можуть обробляти великі обсяги даних та забезпечувати високий рівень користувацького досвіду. Оптимізація продуктивності передбачає не тільки збільшення швидкості завантаження сторінок і зменшення часу відповіді сервера, але й забезпечення стабільної роботи додатка під навантаженням та зменшення ресурсомісткості.

NodeJS є популярною платформою для серверної частини веб-додатків завдяки своїй асинхронній архітектурі та можливості обробляти велику кількість одночасних з'єднань без значних витрат на ресурси. React, у свою чергу, є однією з провідних бібліотек для розробки користувацьких інтерфейсів, що дозволяє створювати динамічні та інтерактивні веб-додатки з високою продуктивністю.

Штучний інтелект (ШІ) та машинне навчання (ML) відіграють ключову роль у динамічній оптимізації продуктивності серверів NodeJS та React. Вони дозволяють автоматизувати процеси аналізу та налаштування продуктивності, знаходити вузькі місця в коді, прогнозувати навантаження та адаптуватися до змін у реальному часі.

Застосування ШІ та ML у цій галузі вимагає глибокого розуміння як

технологій NodeJS та React, так і методів машинного навчання. Розробники повинні володіти знаннями в області програмування, алгоритмів оптимізації, аналізу даних та моделей машинного навчання, щоб ефективно використовувати ці інструменти для підвищення продуктивності веб-додатків.

Таким чином, дослідження та розробка стратегій оптимізації та підвищення продуктивності веб-додатків на базі NodeJS та React з використанням штучного інтелекту є складним та багатогранним процесом, який вимагає поєднання сучасних технологій, методів аналізу та інноваційних підходів для досягнення високої ефективності та надійності веб-додатків.

1.2 Види оптимізацій за допомогою ШІ

Оптимізація та підвищення продуктивності веб-додатків, або оптимізація продуктивності - це процес внесення змін до ваших веб-додатків, щоб вони могли швидше завантажуватися та відображатися. Він поєднує в собі кілька дисциплін: веб-розробку, програмування, оптимізацію продуктивності та штучний інтелект.

Актуальність цієї теми полягає в широкому спектрі її застосування - від покращення продуктивності веб-додатків до автоматизації процесів розробки та тестування. З огляду на значну актуальність оптимізації веб-додатків у різних сферах бізнесу та практичного застосування, розуміння архітектури моделей, які лежать в основі цього процесу, стає ключовим.

Архітектура оптимізації веб-додатків зазвичай складається з двох основних компонентів: модуля аналізу коду, також відомого як кодер, і модуля генерації оптимізованого коду, який працює на основі інформації, закодованої кодером.

Кодер використовує нейронні мережі для обробки вхідного коду. Його основне завдання - виокремити ознаки з коду. Цей процес може включати ідентифікацію функцій, визначення їхнього положення, розміру, типу та взаємозв'язків між різними елементами коду.

Після вилучення ознак ці дані передаються модулю для генерації оптимізованого коду. Цей модуль зазвичай базується на рекурентних нейронних мережах (RNN), LSTM (Long Short-Term Memory) або трансформаторах. Він

відповідає за перетворення функцій в оптимізований код з урахуванням структури та контексту.

Важливою частиною цього процесу є створення когерентного та релевантного коду, який відповідає змісту вихідного коду. Спрощена схема взаємодії кодера і декодера для оптимізації веб-додатків виглядає так:

Підхід, використовується завдяки своїй ефективності у вилученні ознак з коду (за що відповідає CNN) та здатності зберігати контекстну інформацію в послідовності даних (за що відповідає рекурентна нейронна мережа з архітектурою LSTM).

Недоліком цього підходу для великих веб-додатків є продуктивність. Продуктивність архітектури CNN-LSTM зменшується в геометричній прогресії через відсутність семантичного розуміння коду. Цей недолік можна частково компенсувати за допомогою механізму уваги. Це техніка, яка дозволяє моделям динамічно фокусуватися на важливих частинах вхідних даних. Це імітує людську здатність зосереджуватися та обробляти великі обсяги інформації.

З появою трансформаторів та архітектури, таких як BERT (Bidirectional Encoder Representations from Transformers) для обробки тексту та більш сучасних версій CNN для аналізу коду, вчені та інженери почали переходити до більш ефективних методів. Активно розробляються моделі, які можуть краще зрозуміти та оптимізувати код веб-додатків таким чином, щоб він був одночасно точним і контекстно релевантним. Однією з таких моделей є CNN - GRU (Gated Recurrent Unit). При цьому CNN витягує просторові особливості, а модель GRU ефективно вирішує проблему вибуху градієнта при довгостроковому прогнозуванні. Її структура простіша і швидша за LSTM-моделі при вищій точності прогнозування.

1.3 ARM сервіси

В останні роки епохи цифрових технологій, коли веб-додатки на базі NodeJS та React стають не просто інтерфейсами, але і складними системами, одним із ключових напрямків їх розвитку стає інтеграція Штучного Інтелекту (ШІ) та машинного навчання (МН). Ця інтеграція не просто підвищує продуктивність, але

й надає додаткам інтелектуальних можливостей, властивих ері штучного інтелекту.

Коли веб застосунки тільки зароджувались, це були звичайні сайти написані на html, з базовою логікою, та працювали на звичайних серверах, зараз же складні веб додатки, можуть складатись з сотень сервісів, розподілених на окремі незалежні бекенд, чи фронтенд сервіси, працюючи на сотнях серверів паралельно.

За допомогою AI та ML, вже зараз такі хостинг та сервіс провайдери, як AWS, Azure, Google Cloud, розраховують можливі навантаження на сервери та сервіси, та оптимізують використання ресурсів. Вони натреновані на даних користувачів, по часу пікових навантажень, на той чи інший сервіс застосунку, та автоматично надають чи забирають певні ресурсу хостингу, які були надані під ці сервіси.

Такі технології як ARM від AWS уже успішно з цим справляються. Такі технології як: Технологія прецизійного масштабування дозволяє сервісам динамічно регулювати ресурси в реальному часі, уникнувши надмірного резервування.

Щодо проактивного управління інцидентами, використання штучного інтелекту дозволяє виявляти аномалії та забезпечує швидке реагування на можливі загрози безпеці в хмарних сервісах.

У контексті підвищення економічної ефективності, технологія управління хмарними технологіями на основі штучного інтелекту допомагає організаціям ефективно використовувати дані для прийняття рішень, контролювати витрати та раціонально розподіляти ресурси.

1.4 GitHub Copilot

Існує ще безліч рішень для покращення розробки, та оптимізації веб застосунків, одним з них є чат боти. Такі чат боти, як ChatGPT, та Github Copilot, почали активно використовуватись у розробці програмного забезпечення[2].

GitHub Copilot - це штучний інтелект, розроблений спільно GitHub та OpenAI, який допомагає у розробці програмного забезпечення. Він базується на технології GPT (Generative Pre-trained Transformer), яка дозволяє йому розуміти природні мовні описи та генерувати код на основі контексту. GitHub Copilot пропонує

автозаповнення коду в стилі AI, надаючи користувачам пропозиції для цілих рядків або цілих функцій прямо в їхньому редакторі. Він може бути встановлений як розширення в різних середовищах, включаючи Visual Studio Code. GitHub Copilot також може бути використаний в командному рядку, надаючи інтерфейс у стилі чату, який дозволяє користувачам ставити питання про командний рядок. Це робить GitHub Copilot корисним інструментом для розробників, які шукають швидкі та ефективні рішення для своїх програмних завдань. GitHub Copilot не замінює потребу в розумінні коду та програмування, але він може бути корисним інструментом для підвищення продуктивності, оптимізації коду та навчання нових концепцій.

1.5 Огляд RedisAI

RedisAI - це модуль Redis для виконання моделей глибокого навчання/машинного навчання та керування їхніми даними. Його мета полягає в тому, щоб бути “робочим конем” для обслуговування моделей, надаючи підтримку популярних фреймворків DL/ML та неперевершену продуктивність.

RedisAI максимізує пропускну здатність обчислень і зменшує затримку, дотримуючись принципу локальності даних. Він також спрощує розгортання та обслуговування графів, використовуючи перевірену виробничу інфраструктуру Redis.

RedisAI надає підтримку для популярних фреймворків DL/ML. Це означає, що ви можете тренувати свої моделі за допомогою будь-якого зручного для вас інструменту, а потім використовувати RedisAI для обслуговування цих моделей в режимі реального часу.

Використання RedisAI для обслуговування моделей глибокого навчання має наступні переваги:

Максимізація пропускної здатності обчислень і зменшення затримки: RedisAI дотримується принципу локальності даних, що дозволяє максимізувати пропускну здатність обчислень і зменшити затримку.

Спрощення розгортання та обслуговування графів: RedisAI спрощує

розгортання та обслуговування графів, використовуючи перевірену виробничу інфраструктуру Redis.

Використання можливостей сервера Redis: Оскільки RedisAI реалізований як модуль Redis, він автоматично отримує переваги від можливостей сервера, таких як власні типи даних Redis, надійний екосистему клієнтів, висока доступність, стійкість, кластеризація та підтримка Enterprise.

Управління запитами для виконання моделей: RedisAI керує всіма запитами на виконання моделей у чергах обробки та виконує їх на окремих потоках, в той час як Redis залишається чутливим до всіх інших команд.

Підтримка різних фреймворків DL/ML: RedisAI підтримує популярні фреймворки DL/ML, що дозволяє тренувати моделі за допомогою будь-якого зручного інструменту, а потім використовувати RedisAI для обслуговування цих моделей в режимі реального часу.

1.5 Постановка задачі

В магістерському дослідженні ми виділили наступні задачі, які будуть розглядатись:

Проаналізувати існуючу літературу про методи машинного навчання для оптимізації та підвищення продуктивності веб-додатків.

Визначити ключові характеристики та обмеження сучасних моделей машинного навчання, що використовуються для оптимізації та підвищення продуктивності веб-додатків.

Огляд існуючих критеріїв оцінки ефективності моделей для оптимізації та підвищення продуктивності веб-додатків.

Дослідити існуючі набори даних і вибрати той, на якому будуть навчатися моделі.

Реалізувати та порівняти декілька обраних моделей для виконання задачі оптимізації та підвищення продуктивності веб-додатків.

Протестувати розроблені моделі на сформованому наборі даних з реальними веб-додатками для визначення їхньої точності та здатності генерувати змістовну

оптимізацію.

Проаналізувати вплив різних факторів, таких як розмір набору даних та якість веб-додатків, на продуктивність моделей.

Розробити рекомендації щодо вдосконалення моделей для підвищення точності та скорочення часу обробки.

Основні критерії оцінки включають швидкодію, масштабованість, ресурсоємність, стійкість до навантажень та адаптивність.

Швидкодія визначає, наскільки ефективно система реагує на користувацькі запити. Це важливо для задоволення користувачів та поліпшення загального враження від використання додатку. Масштабованість оцінює здатність системи працювати при збільшенні обсягу навантаження, що є ключовим аспектом для забезпечення стабільної роботи при зростанні кількості користувачів.

Ресурсоємність визначає ефективність використання обчислювальних та мережевих ресурсів системи. Це важливо для оптимізації вартості та забезпечення стабільності. Стійкість до навантажень оцінює здатність системи підтримувати продуктивність під час стресових ситуацій або при збільшенні навантаження.

Адаптивність системи є важливою для того, щоб ефективно реагувати на зміни в вимогах чи навантаженні. Застосування штучного інтелекту для оптимізації алгоритмів, використання кешування та оптимізація запитів до сервера є можливими альтернативами для досягнення цих цілей. Дослідження також розглядає автоматичне масштабування ресурсів та використання штучного інтелекту для передбачення об’єму навантаження.

Загальний фокус дослідження полягає в розробці стратегій оптимізації, які покращують якість та ефективність веб-додатків, сприяючи задоволенню користувачів та забезпечуючи стабільну роботу системи в умовах змінного користувацького навантаження та технологічного середовища.

Швидкодія (Час відгуку системи): Використано шкалу порядку з трьома значеннями, де 3 позначає дуже швидку відповідь, 2 - прийнятний час відгуку, а 1 - повільний час відгуку.

Масштабованість: Використана шкала порядку з трьома значеннями.

Оцінюється здатність додатку працювати при великому користувацькому навантаженні, де 1 позначає неефективність, 2 - певну масштабованість з покращеннями, а 3 - ефективність масштабування при великому навантаженні.

Ресурсоємність: Використана шкала порядку з трьома значеннями. Оцінює використання ресурсів системи, де 1 вказує на високе використання ресурсів, 2 - на модератне використання з можливою потребою в оптимізації, а 3 - на низьке використання ресурсів.

Стійкість до навантажень: Використана шкала порядку з трьома значеннями. Оцінюється здатність системи утримувати продуктивність при навантаженні, де 1 вказує на втрату продуктивності, 2 - на збереження продуктивності до певного рівня, а 3 - на ефективну роботу під великими навантаженнями.

Адаптивність: Використана шкала порядку з трьома значеннями. Оцінює швидкість та ефективність адаптації системи до змін, де 1 позначає неефективність адаптації, 2 - адаптацію, але не так ефективно, як бажалося б, а 3 - швидку та ефективну адаптацію.

Занесемо оцінки до таблиці (див. табл. 1.1)

Таблиця 1.1 – Оцінка варіантів оптимізації (таблиця виконана самостійно)

Альтернатива	Швидкодія	Масштабованість	Ресурсоємність	Стійкість	Адаптивність
Оптимізація алгоритмів	3	2	2	3	3
Кешування та попереднє завантаження	2	2	1	2	2
Оптимізація запитів до сервера	2	3	2	2	2
Автоматичне масштабування	3	2	2	3	3

Штучний інтелект для передбачення навантаження	3	2	3	3	3
Оптимізація фронтенду за допомогою AI	2	2	2	2	2

Враховуючи результати проведеного аналізу, отриманих даних та спостереженнях за поведінкою системи в різних умовах, можна зробити обґрунтований висновок про те, що оптимізація серверних ресурсів у поєднанні з проактивним передбаченням навантаження є найбільш ефективним та перспективним підходом до вирішення проблеми оптимізації продуктивності веб-додатків, розроблених на основі технологій Node.js та React[9].

Цей висновок підтверджується низкою факторів. По-перше, динамічне масштабування ресурсів сервера дозволяє забезпечити оптимальний баланс між продуктивністю та витратами. Завдяки здатності системи автоматично адаптувати кількість виділених ресурсів до поточного рівня навантаження, можна уникнути як надмірного використання ресурсів у періоди низької активності, так і їх нестачі у моменти пікового навантаження[3].

По-друге, використання методів машинного навчання для прогнозування майбутнього навантаження дозволяє проактивно реагувати на зміни в поведінці користувачів та запобігати виникненню проблем з продуктивністю ще до їх появи[4-5]. Завдяки здатності моделі враховувати різноманітні фактори, такі як час доби, день тижня, сезонність та інші, можна досягти високої точності прогнозів та забезпечити стабільну роботу веб-додатків навіть за умов непередбачуваних змін у навантаженні.

Звичайно, даний підхід не є універсальним рішенням для всіх можливих проблем з продуктивністю веб-додатків. У деяких випадках, можуть знадобитися додаткові заходи, такі як оптимізація коду додатку, налаштування бази даних,

використання кешування та інші. Проте, оптимізація серверних ресурсів та передбачення навантаження є фундаментальними інструментами, які дозволяють значно покращити продуктивність веб-додатків на Node.js та React та забезпечити їх стабільну роботу в умовах зростаючого навантаження.

2 ОПИС ПРИЙНЯТИХ ПРОЄКТНИХ РІШЕНЬ

2.1 Вибір технологій

Для побудови сервісу та проведення дослідження було обрано фреймворк NestJS на базі TypeScript, як основної мови програмування. [3]

Базова архітектура цього фреймворку, дозволяє будувати на ньому сучасні, високонавантажені бекенд додатки. Цей фреймворк підтримує більшість NodeJS [7] бібліотек, а також має свої бібліотеки. Має підтримку TypeScript, що дозволяє створювати моделі та DTO для більшого контролю коду. NestJS є одним з найпопулярніших фреймворків побудованих на NodeJS.

2.2 Бібліотеки та інструменти

Для реалізації логіки хаб серверу та серверу ML було використано декілька бібліотек, як для побудови ML, так і для підключення з популярним хостинг та клауд провайдерами такими як AWS. Перелік бібліотек:

- @nestjs/core (версія 9.2.0): ядро фреймворку NestJS, яке забезпечує модульну архітектуру для створення серверних додатків на Node.js. Підтримує використання TypeScript і надає потужні можливості для організації структури проекту, управління залежностями, створення модулів, контролерів та сервісів. NestJS побудований на основі Express або Fastify, що дозволяє розробляти масштабовані і високопродуктивні додатки. rxjs (версія 7.8.1): бібліотека для роботи з реактивним програмуванням в JavaScript. Забезпечує потужні можливості для роботи з асинхронними даними, подіями і потоками, що робить її незамінною при розробці складних додатків.
- @tensorflow/tfjs-node (версія 3.14.0): бібліотека для машинного навчання, яка дозволяє запускати TensorFlow моделі на Node.js. Підтримує апаратне прискорення і надає зручний інтерфейс для роботи з нейронними мережами.
- @swc/core (версія 1.2.143): швидкий компілятор і мінімізатор для

JavaScript та TypeScript. Забезпечує високу продуктивність завдяки написанню на Rust, що робить процес компіляції значно швидшим у порівнянні з аналогами.

- @nestjs/axios (версія 7.1.5): модуль для NestJS, який інтегрує популярну бібліотеку Axios для здійснення HTTP-запитів. Дозволяє легко здійснювати асинхронні HTTP-запити з мінімальною кількістю налаштувань.
- @nestjs/schedule (версія 9.0.0): модуль для NestJS, який надає зручний спосіб планування завдань у Node.js застосунках. Дозволяє запускати функції за розкладом (cron-подібний синтаксис), з певною періодичністю (інтервали) або одноразово через певний час (таймаути). Спрощує створення завдань, пов'язаних з оновленням даних, надсиланням повідомлень, очищенням кешу тощо.
- @aws-sdk/client-cloudwatch (версія 3.333.0): клієнтська бібліотека для роботи з Amazon CloudWatch, яка забезпечує зручний інтерфейс для моніторингу та управління ресурсами AWS. Підтримує всі функції CloudWatch для збору та аналізу метрик.
- @aws-sdk/client-ec2 (версія 3.333.0): клієнтська бібліотека для взаємодії з Amazon EC2, що надає можливість керування віртуальними серверами в хмарі AWS. Забезпечує повний доступ до функціоналу EC2, включаючи створення, налаштування та моніторинг інстансів.

2.3 Шарова структура ML

Для розробки ML для аналітики та керування ресурсами серверу було обрану наступну структуру:

- Перший шар - повністю зв'язаний (dense) шар з 10 нейронами, активаційна функція якого - 'relu'. Цей шар має вхідну форму [8], оскільки вхідні дані мають розмірність 2.
- Другий шар - також повністю зв'язаний, але з двома нейронами і активаційною функцією 'softmax'. Це зазвичай використовується для задач

класифікації, де модель має вибрати один з двох класів.

Після створення моделі, вона компілюється з оптимізатором 'adam', втратною функцією 'categoricalCrossentropy' (оскільки, імовірно, модель призначена для багатокласової класифікації) та метрикою 'accuracy' для оцінки точності моделі.

Отже такий підхід допомогає створити модель нейронної мережі, розділену на два шари, та з розмірностю вхідних даних 2 на два класи.

3 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

В даному блоці описані загальні технічні характеристики, будова та структура коду основних елементів, основні методи та функції, що були використані під час дослідження, а також застосування описаних бібліотек на практиці.

3.1 Архітектура застосунку

Для проектування архітектури системи було розроблено діаграму архітектури системи та усіх її сервісів (див рис 3.1).

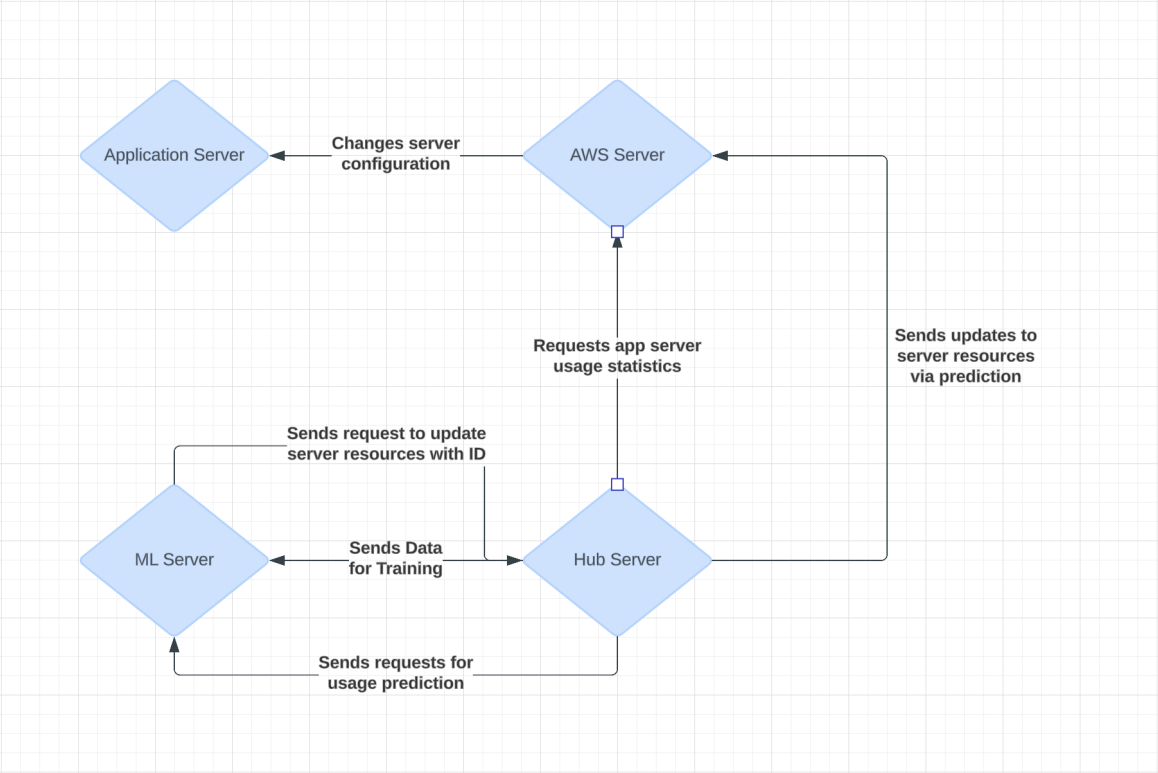


Рисунок 3.1 –Архітектура системи (рисунок створено самостійно)

Для створення архітектури сервісу ML для проведення дослідження було створено рекомендовану розробниками NestJS архітектуру. Для ініціації NestJS було використано NestJS CLI [6-7], середовищем для розробки був VS CODE.

Архітектура проекту представлена на рис.3.2.

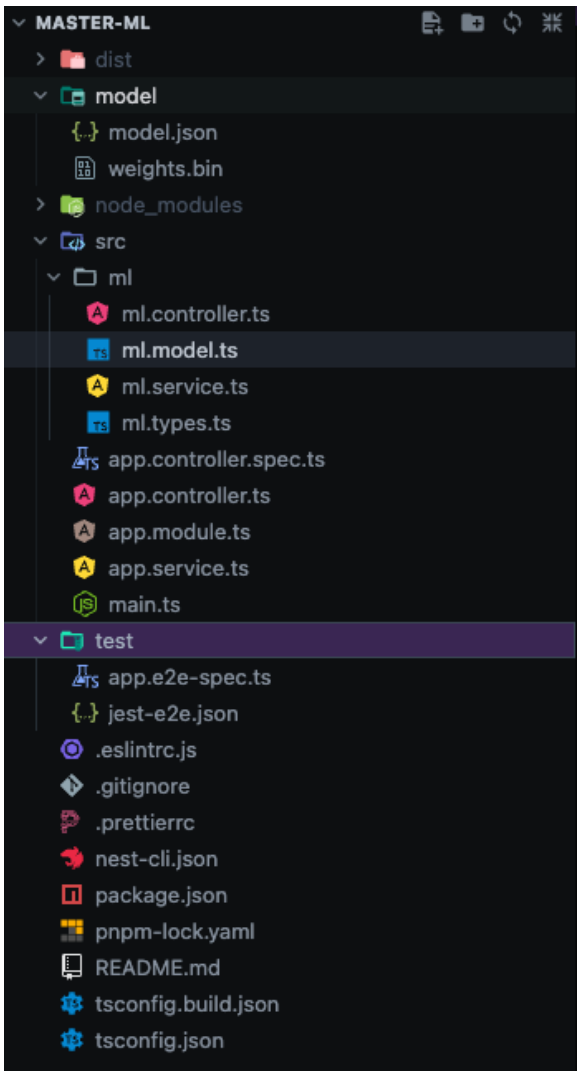


Рисунок 3.2 –Архітектура сервісу ML (рисунок створено самостійно)

Коренева папка проекту включає в себе теки бібліотек, які знаходяться у папці "node_modules", базову структуру додатку у папці src, яка містить основну логіку, теку model з даними тренованої моделі, та папку test з авто тестами проекту

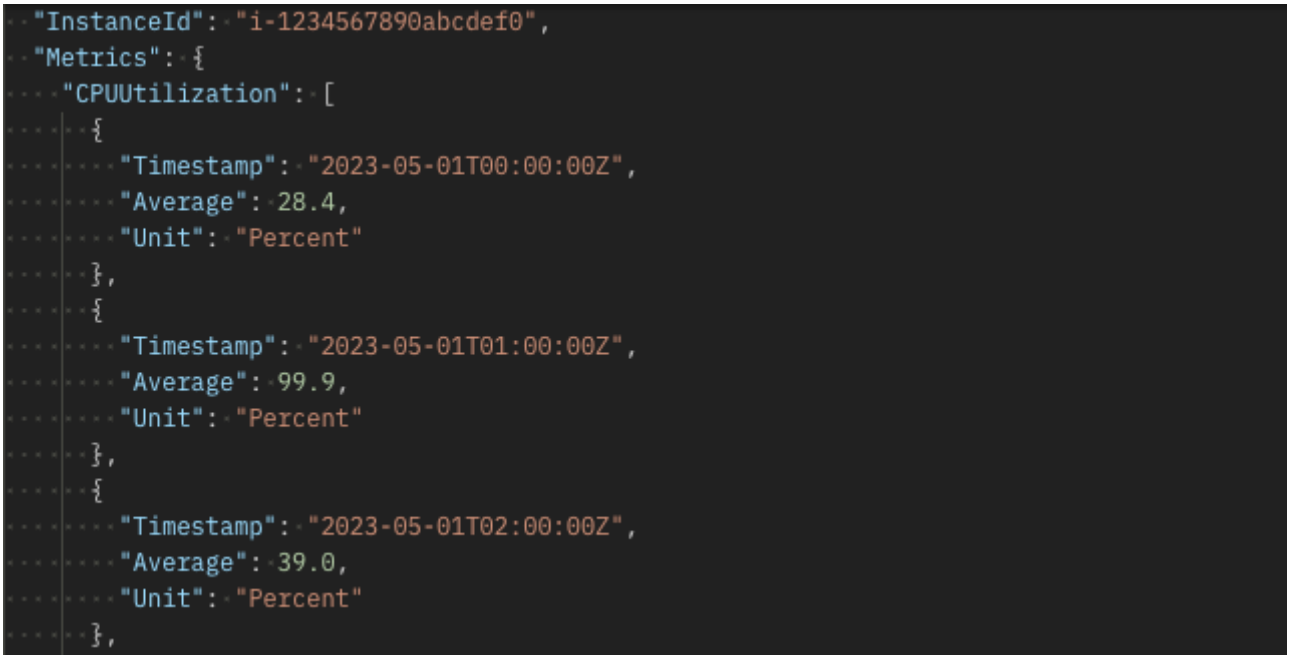
Файли «.gitignore», «package.json», «package-lock.json», «README.md» – це файли конфігурації проекту, які містять інформацію про використані бібліотеки, загальний опис проекту та іншу технічну документацію. (див. рис. 3.1).

Вся основна логіка, що використовується лежить у папці src. Тут знаходиться основні модулі та сервіси додатку, котрі відповідають за підключення інших сервісів та модулів, також тут є тека ML – в котрій знаходиться ml.controller.ts – файл контролеру сервісу ML. В ньому прописані усі ендпоінти. Також тут знаходиться файл ML моделі – ml.model.ts, в котрому знаходиться уся логіка

навчання моделі, ml.service.ts з основноми функціями предикту та логікою ендпоінтів, та файл типів – ml.types.ts.

Основними файлами є main.ts в котрому прописана логіка запуску серверу NestJS та app.module.ts, в котрому відбуваються підключення модулів до серверу.

Для проведення дослідження було зібрані дані використання VPS інстансу на AWS, для навчання моделі та генерації предиктів наватанження, ці дані передаються в форматі JSON (див. рис. 3.2) на один з ендпоінтів для актуалізації моделі.



```
"InstanceId": "i-1234567890abcdef0",
"Metrics": {
  "CPUUtilization": [
    {
      "Timestamp": "2023-05-01T00:00:00Z",
      "Average": 28.4,
      "Unit": "Percent"
    },
    {
      "Timestamp": "2023-05-01T01:00:00Z",
      "Average": 99.9,
      "Unit": "Percent"
    },
    {
      "Timestamp": "2023-05-01T02:00:00Z",
      "Average": 39.0,
      "Unit": "Percent"
    },
```


Рисунок 3.2 – Приклад даних для навчання моделі (рисунок створено самостійно)

Формат даних є скомбінованим форматом даних використання серверу, що повертає AWS CloudWatch за запитами на отримання даних з навантаження CPU та Memory.

3.2 Реалізація ML моделі та предиктів

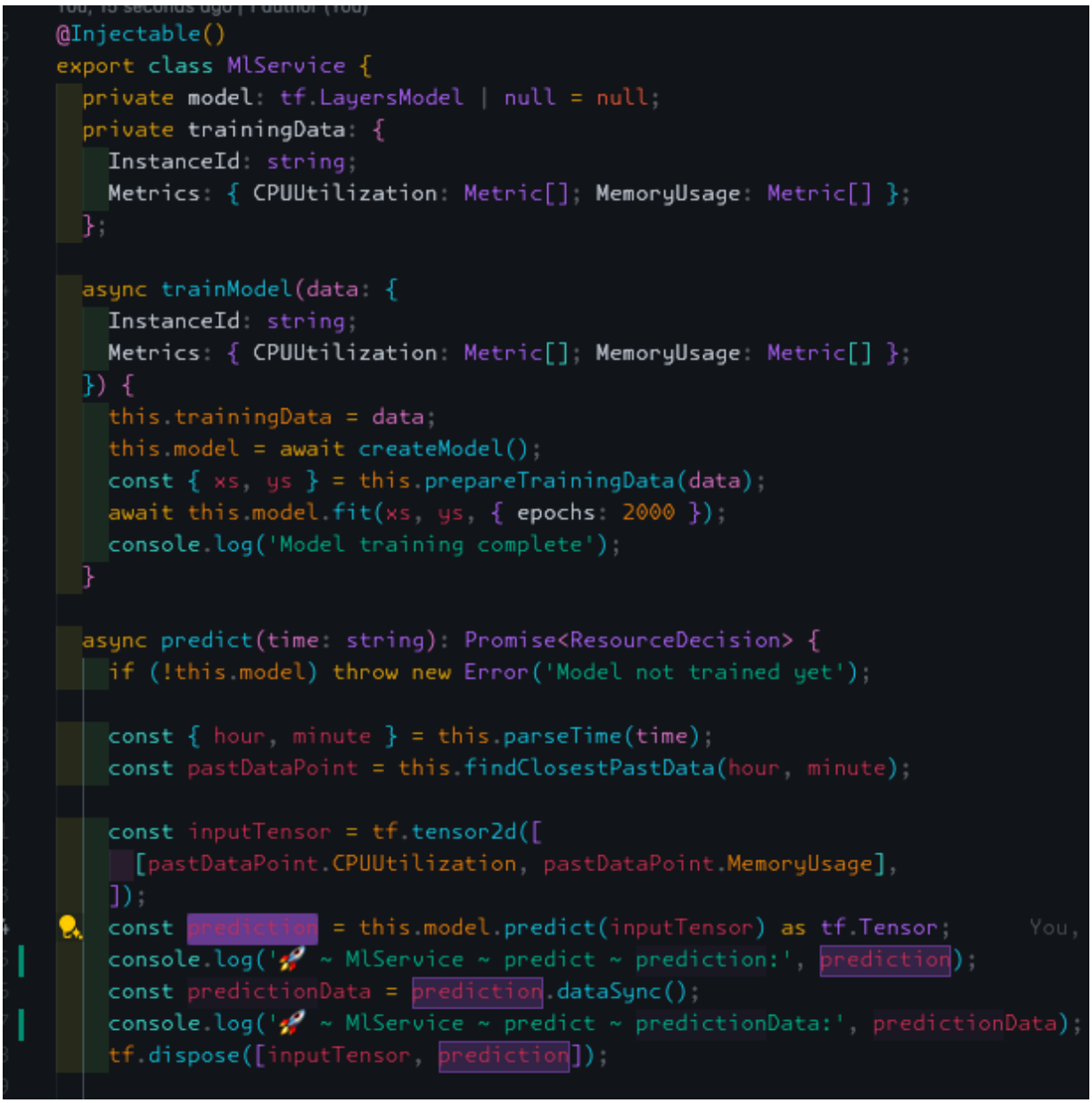
Виконавчим файлом є «main.ts», що потрібен для запуску додатку, налаштовування CORS, та портів додатку.

Директорія «ml» містить усі функції, що використовуються сервісом. У файлі ml.service.ts описані основні функції цього сервісу, на які посилаються запити з ml.controller.ts, та відповіді котрих приходить у відповіді.

У сервіс файлі є основні функції, а саме:

- функція тренування моделі: функція що запускає навчання ML, тут можна конфігурувати кількість поколінь навчання;
- функція предикту: функція що на основі натренованої моделі видає предикти на потрібні ресурси серверу на обраний час та дату;
- функція нормалізації дат: функція що задіяна під час навчання для нормалізації дат у даних для навчання для легкої типізації;
- функція стандартизації дат для предиктів: функція що відповідає за преведння дат для.

Усі функції даного сервісу поєднані в один клас (див. рис. 3.3). В рамках класу мають “private” перемінні, та технічні функції, що дозволяє зв’язати логіку в одному місці, для зручності підтримки коду, та ефективного керування. Налаштування базових параметрів відбувається завдяки env та “config” сервісу, що типізує ці env.



```
@Injectable()
export class MlService {
  private model: tf.LayersModel | null = null;
  private trainingData: {
    InstanceId: string;
    Metrics: { CPUUtilization: Metric[]; MemoryUsage: Metric[] };
  };

  async trainModel(data: {
    InstanceId: string;
    Metrics: { CPUUtilization: Metric[]; MemoryUsage: Metric[] };
  }) {
    this.trainingData = data;
    this.model = await createModel();
    const { xs, ys } = this.prepareTrainingData(data);
    await this.model.fit(xs, ys, { epochs: 2000 });
    console.log('Model training complete');
  }

  async predict(time: string): Promise<ResourceDecision> {
    if (!this.model) throw new Error('Model not trained yet');

    const { hour, minute } = this.parseTime(time);
    const pastDataPoint = this.findClosestPastData(hour, minute);

    const inputTensor = tf.tensor2d([
      [pastDataPoint.CPUUtilization, pastDataPoint.MemoryUsage],
    ]);
    const prediction = this.model.predict(inputTensor) as tf.Tensor;       You,
    console.log('🚀 ~ MlService ~ predict ~ prediction:', prediction);
    const predictionData = prediction.dataSync();
    console.log('🚀 ~ MlService ~ predict ~ predictionData:', predictionData);
    tf.dispose([inputTensor, prediction]);
```

Рисунок 3.3 –Сервіс основної логіки (рисунок створено самостійно)

Інші файли відповідають за саму ML. У файлі ml.model.ts прописан конфіг моделі тренування, кількість шарів, параметри оптимізатору, втратної функції та метрик оцінки точності даних.

Натренована модель зберігається в файлі model.json та weight.bin (див. рис. 3.4).

Моделі даних що використовуються у функціях зберігаються у файлі ml.types.ts. Тут описані основні інтерфейси, а саме:

– інтерфейс відповіді на запит тренування даних
– інтерфейс відповіді на запит предикту навантаження серверу

– інтерфейс даних для тренування

– інтерфейси метрик

```
"modelTopology": {
  "class_name": "Sequential",
  "config": {
    "name": "sequential_1",
    "layers": [
      {
        "class_name": "LSTM",
        "config": {
          "name": "lstm_LSTM1",
          "trainable": true,
          "batch_input_shape": [null, 24, 2],
          "dtype": "float32",
          "units": 50,
          "activation": "tanh",
          "recurrent_activation": "hard_sigmoid",
          "use_bias": true,
          "kernel_initializer": {
            "class_name": "VarianceScaling",
            "config": {
              "scale": 1,
              "mode": "fan_avg",
              "distribution": "normal",
              "seed": null
            }
          },
          "recurrent_initializer": {
            "class_name": "Orthogonal",
            "config": {
              "gain": 1,
              "seed": null
            }
          },
          "bias_initializer": {
            "class_name": "Zeros",
            "config": {}
          },
          "unit_forget_bias": null,
          "kernel_regularizer": null,
          "recurrent_regularizer": null,
          "bias_regularizer": null,
          "activity_regularizer": null,
          "kernel_constraint": null,
          "recurrent_constraint": null,
```


Рисунок 3.4 – Файл тренованої моделі (рисунок створено самостійно)

Модуль цього сервісу підключен до основної частини додатку за допомогою app.module.ts, де прописан конфіг на підключення.

3.3 Реалізація функції передбачення оптимального розміру ресурсів

Основна функція передбачення оптимального розміру ресурсів знаїодить у файлі “ml.service.ts” та має назву “getResourcePrediction”. Вона приймає в себе параметр часу – “time”, за запитом котрого вираховує навантаження серверу у цей час (див. рис. 3.5).

```
async getResourcePrediction(time: string): Promise<ResourceDecision> {
   if (!this.model) throw new Error('Model not trained yet');

   const { hour, minute } = this.parseTime(time);
   const pastDataPoint = this.findClosestPastData(hour, minute);

   const inputTensor = tf.tensor2d([
     [pastDataPoint.CPUUtilization, pastDataPoint.MemoryUsage],
   ]);
   const prediction = this.model.predict(inputTensor) as tf.Tensor;
   console.log('🚀 ~ MlService ~ predict ~ prediction:', prediction);
   const predictionData = prediction.dataSync();
   console.log('🚀 ~ MlService ~ predict ~ predictionData:', predictionData);
   tf.dispose([inputTensor, prediction]);

   const scaleUpThreshold = 0.7;
   const scaleDownThreshold = 0.3;

   let action: 'Scale Up' | 'Scale Down' | 'No Action' = 'No Action';
   let reason;

   if (predictionData[0] > scaleUpThreshold) {
     action = 'Scale Up';
     reason =
       pastDataPoint.CPUUtilization > pastDataPoint.MemoryUsage
         ? 'CPU'
         : 'Memory';
   } else if (predictionData[1] > scaleDownThreshold) {
     action = 'Scale Down';
   }

   return {
     action,
     resources:
       action !== 'No Action'
         ? { instanceType: action === 'Scale Up' ? 't2.large' : 't2.small' }
         : undefined,
     time,
     reason,
   };
 }
```

Рисунок 3.5 – Функція передбачення оптимального розміру ресурсів (рисунок створено самостійно)

Основні кроки виконання даної функції:

- перевірка наявності навченої моделі. Якщо модель ще не навчена, генерується помилка;
- часові дані, передані у функцію, розбиваються на години та хвилини за допомогою функції parseTime();
- знаходиться найближча минула точка даних, що використовується для прогнозування. Ймовірно, це дані про використання центрального процесора (CPU) та пам'яті на попередній момент часу;
- створюється вхідний тензор TensorFlow, що містить дані про використання CPU та пам'яті;
- модель машинного навчання використовується для прогнозування оптимального розміру ресурсів. Результат прогнозу зберігається у змінній prediction;
- результат прогнозу перетворюється в масив даних за допомогою dataSync();
- Визначаються порогові значення для масштабування вгору та вниз.
- визначається дія на основі результату прогнозу. Якщо прогноз вказує на потребу у збільшенні обсягу ресурсів (вище порогового значення масштабування вгору), алгоритм вирішує масштабувати ресурси вгору. Якщо прогноз вказує на потребу у зменшенні обсягу ресурсів (вище порогового значення масштабування вниз), алгоритм вирішує масштабувати ресурси вниз;
- повертаються результати прогнозу: дія (масштабування вгору, масштабування вниз або бездіяльність), ресурси (якщо дія не є "бездіяльністю", вказується розмір ресурсу, який рекомендується масштабувати), час та причина дії;

Таким чином у результаті виконання функції «getResourcePrediction» отримаємо дані по теоретичному навантаженні серверу у заданий нами час.

4 ОПИС ЕКСПЕРИМЕНТАЛЬНИХ ДОСЛІДЖЕНЬ

4.1 Підготовка даних та налаштування

Метою даного експерименту є всебічне дослідження потенціалу штучного інтелекту (ШІ) у сфері оптимізації та підвищення продуктивності веб-додатків. Зокрема, ми прагнемо перевірити гіпотезу про те, що інтеграція ШІ у процес прогнозування навантаження на сервери та автоматизація прийняття рішень щодо розподілу ресурсів може призвести до значного покращення ефективності роботи веб-додатків.

Для досягнення цієї мети, ми розробили комплексний підхід, що включає в себе збір, аналіз та обробку великих обсягів даних, навчання моделі машинного навчання та її інтеграцію у реальне середовище.

На першому етапі, ми отримали доступ до детальної статистики навантаження EC2 інстансів (віртуальних серверів) Amazon Web Services (AWS) за два місяці. Ця статистика включала в себе дані про використання оперативної пам'яті (Memory) та центрального процесора (CPU) з погодинною деталізацією. Загалом, було зібрано близько 12 тисяч рядків даних у форматі JSON, що представляють собою цінне джерело інформації для подальшого аналізу та навчання моделі.

Для отримання цих даних, ми скористалися потужним інструментом AWS CloudWatch, який надає широкий спектр можливостей для моніторингу та збору метрик з різних компонентів хмарної інфраструктури. Зокрема, ми використовували API CloudWatch (див. рис. 4.1), що дозволило нам автоматизувати процес збору даних та отримати їх у зручному для подальшої обробки форматі.

Отримані дані були ретельно проаналізовані та підготовлені для навчання моделі машинного навчання.

```
async modifyInstanceResources(
    instanceId: string,
    resources: any,
  ): Promise<void> {
    const command = new ModifyInstanceAttributeCommand({
      InstanceId: instanceId,
      InstanceType: {
        Value: resources.instanceType,
      },
    });
    await this.ec2Client.send(command);
  }

  async getMetricStatistics(metricName: string) {
    const command = new GetMetricStatisticsCommand({
      Namespace: 'AWS/EC2',
      MetricName: metricName,
      Dimensions: [{ Name: name, Value: id }],
      StartTime: startDate
      EndTime: new Date(),
      Period: 3600, // 60 minutes
      Statistics: ['Average'],
    });

    return this.cloudWatchClient.send(command);
  }
}
```

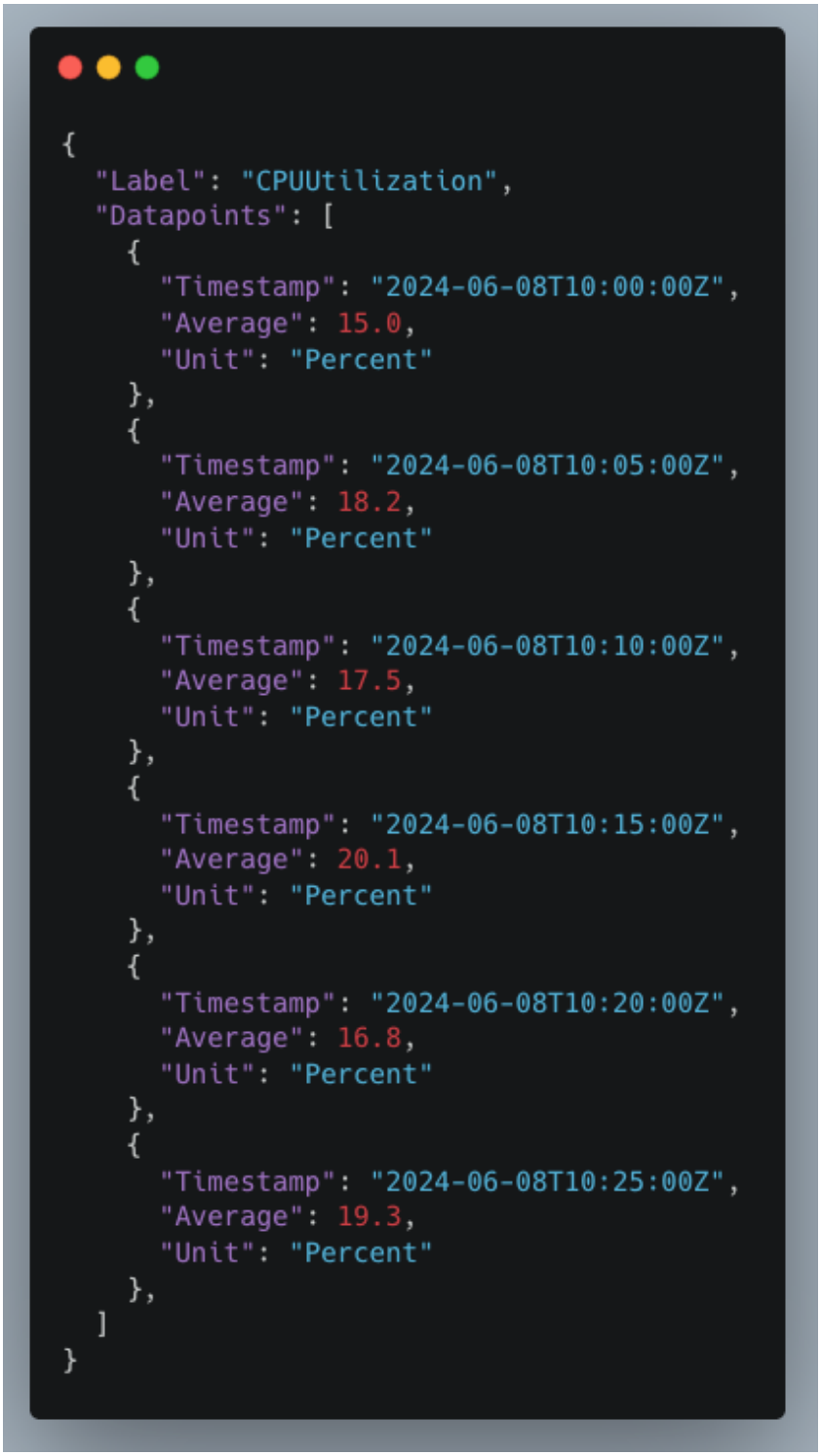
Рисунок 4.1 – Запит отримання даних (рисунок створено самостійно)

Після ініціації та виконання функції запиту до CloudWatch, отримано відповідь, що містить цінні дані про стан та продуктивність наших систем (див. рис. 4.2). CloudWatch, потужний сервіс моніторингу та спостереження від Amazon Web Services, надає нам детальну інформацію про різноманітні метрики, що характеризують роботу наших ресурсів.

Проте, важливо зазначити, що відповідь від CloudWatch зазвичай містить дані лише за однією метрикою. Це може бути, наприклад, інформація про завантаження центрального процесора (CPU), використання оперативної пам'яті (RAM), мережевий трафік або інші важливі показники. Однак, для отримання повної картини та глибокого розуміння стану системи, нам необхідно зібрати дані

за декількома метриками одночасно.

Для вирішення цієї задачі, було використано підхід, що полягає у компонуванні декількох запитів до CloudWatch в один. Це дозволяє нам отримати всі необхідні дані за один раз, що значно спрощує процес збору та аналізу інформації.



```
{
  "Label": "CPUUtilization",
  "Datapoints": [
    {
      "Timestamp": "2024-06-08T10:00:00Z",
      "Average": 15.0,
      "Unit": "Percent"
    },
    {
      "Timestamp": "2024-06-08T10:05:00Z",
      "Average": 18.2,
      "Unit": "Percent"
    },
    {
      "Timestamp": "2024-06-08T10:10:00Z",
      "Average": 17.5,
      "Unit": "Percent"
    },
    {
      "Timestamp": "2024-06-08T10:15:00Z",
      "Average": 20.1,
      "Unit": "Percent"
    },
    {
      "Timestamp": "2024-06-08T10:20:00Z",
      "Average": 16.8,
      "Unit": "Percent"
    },
    {
      "Timestamp": "2024-06-08T10:25:00Z",
      "Average": 19.3,
      "Unit": "Percent"
    },
  ]
}
```


Рисунок 4.2 – Відповідь CloudWatch (рисунок створено самостійно)

Загалом для отримання усіх потрібних даних треба надіслати 2 запити на CloudWatch, та потім лог цих даних зберігається у хаб сервері. Для відслідкування переходів данних по системі було створено Data Flow Diagram

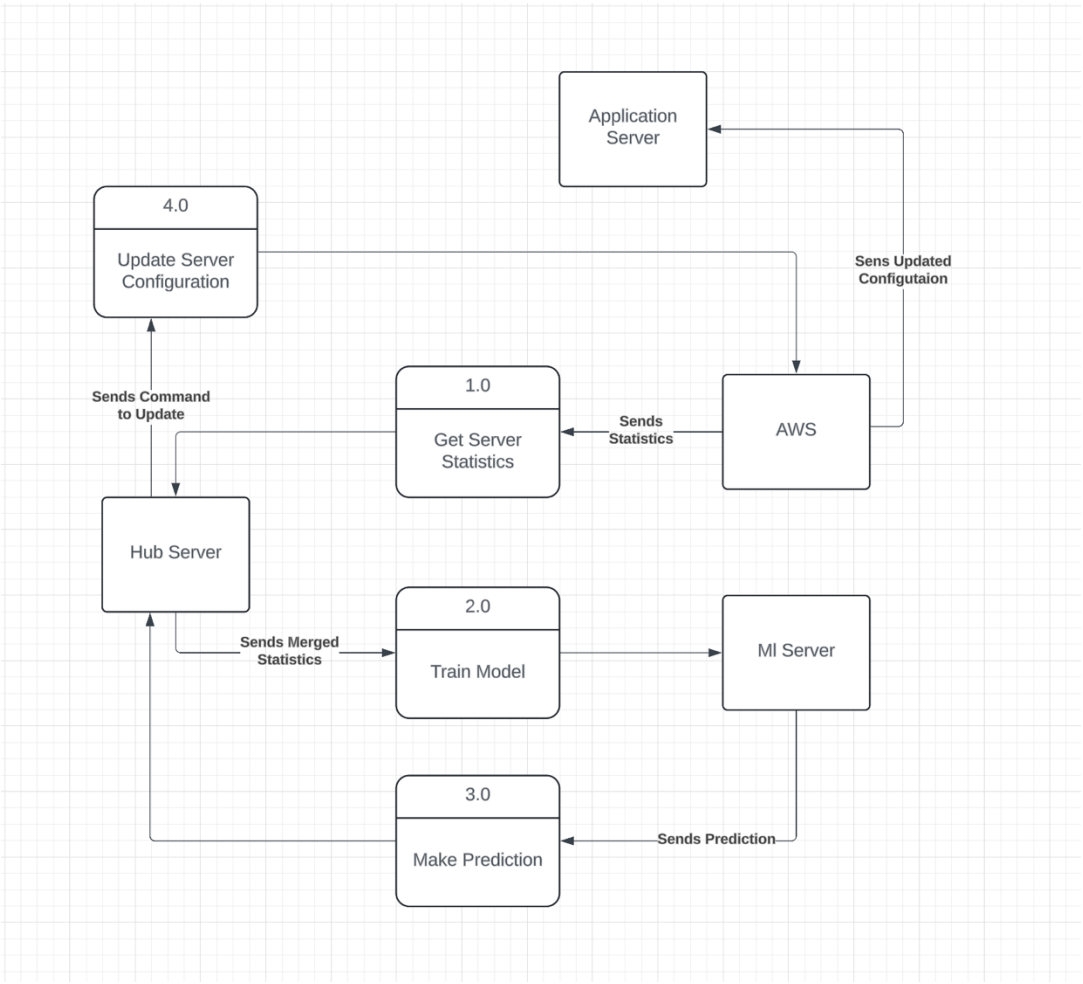


Рисунок 4.3 – Data Flow діаграма (рисунок створено самостійно)

4.2 Тренування ML

Для початку тренування моделі ML, необіхдно надіслати запит з даними для тренування (див. рис. 4.3)

Рисунок 4.3 – Ендпоінт запуску тренування моделі (рисунок створено самостійно)

Після того, як запит, що містить необхідні параметри та часовий інтервал, буде успішно відправлений на вказаний ендпоінт, система ініціює ряд складних та взаємопов'язаних процесів, спрямованих на створення, навчання та налаштування моделі машинного навчання, яка буде відповідати за прогнозування навантаження на сервер.

Першим кроком у цьому ланцюжку є генерація ключових файлів моделі, а саме model.json та weights.bin. Файл model.json містить архітектуру моделі, тобто її структуру, шари, зв'язки між ними та інші параметри, що визначають її функціональність. Файл weights.bin, у свою чергу, зберігає ваги моделі, які є числовими коефіцієнтами, що визначають силу зв'язків між нейронами та впливають на результат прогнозування.

Після успішного створення цих файлів, система переходить до наступного етапу - навчання моделі. Цей процес є ітеративним та включає в себе багаторазове пред'явлення моделі навчальних даних, аналіз її помилок та коригування ваг з метою мінімізації цих помилок. У нашому випадку, для візуалізації процесу навчання та відстеження його прогресу, використовуються виводи console.log() у коді (див. рис. 4.2). Ці виводи дозволяють нам спостерігати за зміною втрат моделі, точністю її прогнозів та іншими важливими метриками в режимі реального часу.

Важливим параметром, що впливає на якість та точність моделі, є кількість поколінь (epochs) навчання. У рамках нашого дослідження, було прийнято рішення обрати 2000 поколінь. Це означає, що модель буде 2000 разів "проглядати"

навчальні дані, кожного разу покращуючи свої ваги та наближаючись до оптимального стану. Звичайно, вибір кількості поколінь є компромісом між часом навчання та точністю моделі. Більша кількість поколінь може призвести до більш точних прогнозів, але також вимагатиме більше часу та обчислювальних ресурсів.

Рисунок 4.4 – Результат тренування моделі (рисунок створено самостійно)

Після навчання отримуємо готову модель, за допомогою котрої можна продовжити експеримент.

4.3 Перевірка передбачень навантаження

Для того, щоб перевірити ефективність нашої моделі прогнозування та оцінити її здатність передбачати майбутнє навантаження на сервер, необхідно провести ретельне тестування. Цей процес включає в себе кілька важливих етапів, кожен з яких відіграє ключову роль у забезпеченні точності та надійності отриманих результатів.

Для проведення тестування логіки передбачень, була розроблена UML діаграма логіки (див. рис. 4.5).

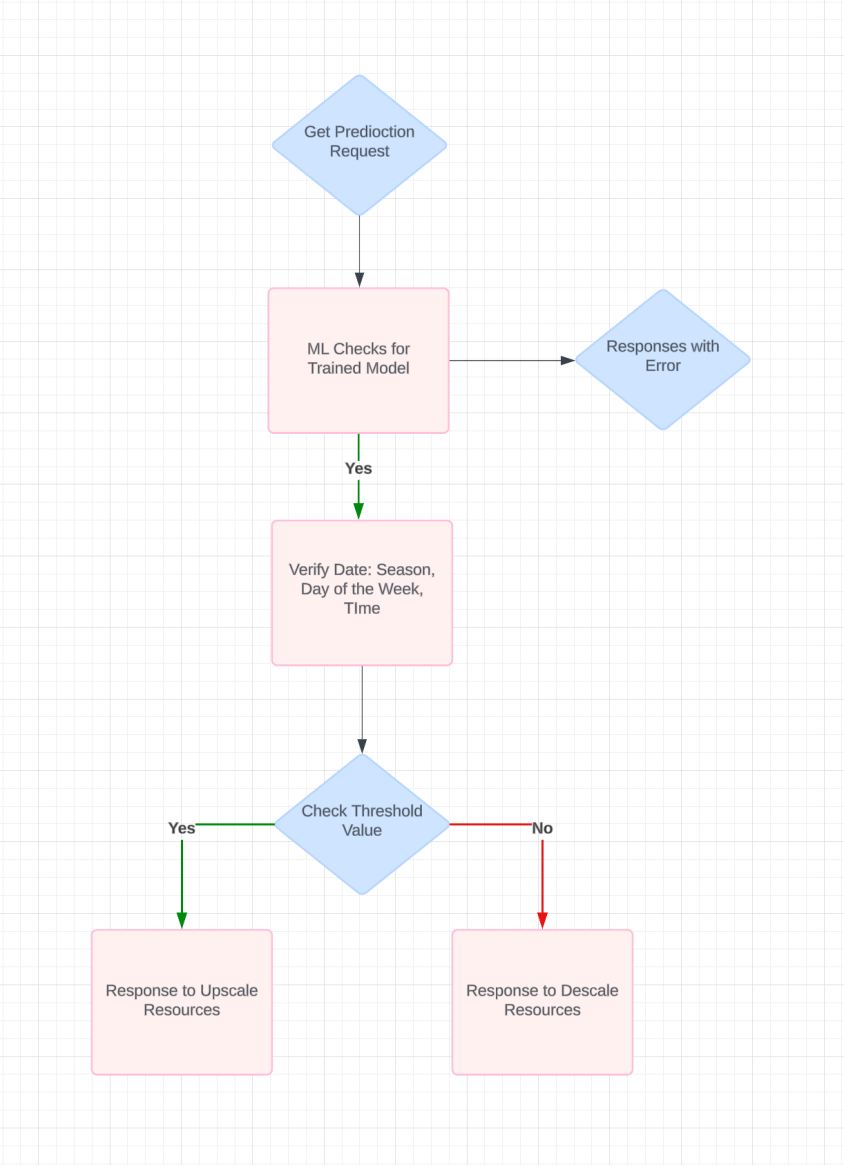


Рисунок 4.5 – UML Логіка передбачення (рисунок створено самостійно)

Першим кроком у тестуванні є формування та відправка спеціального запиту на визначений ендпоінт (рис. 4.5). Цей ендпоінт являє собою інтерфейс, через який наша система взаємодіє з зовнішнім світом, приймаючи запити та надаючи відповіді. У нашому випадку, запит повинен містити ключовий параметр - часовий інтервал, для якого ми бажаємо отримати прогноз навантаження на сервер.

Цей часовий інтервал може бути вибраний довільно, в залежності від цілей тестування та особливостей досліджуваної системи.

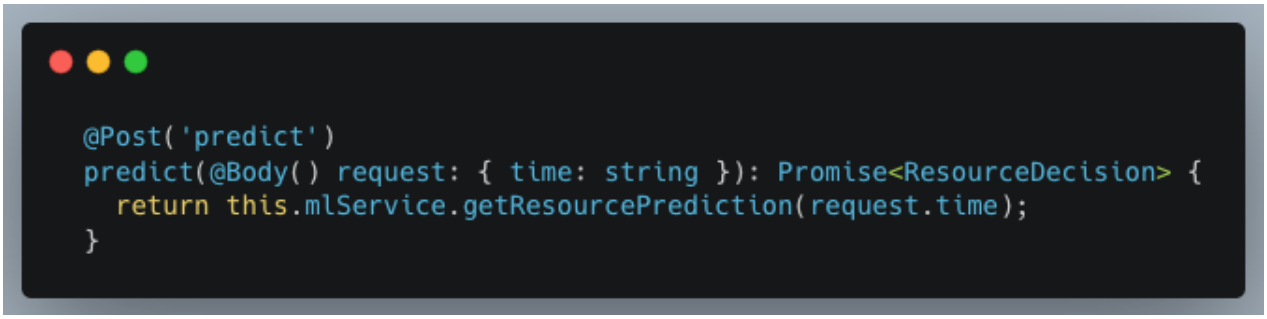


Рисунок 4.5 – Запит на передбачення ресурсів (рисунок створено самостійно)

Після ініціації запиту користувача, активується ключовий компонент системи - функція передбачення та аналізу (див. рис. 4.6). Ця функція являє собою складний алгоритм, що враховує безліч факторів, таких як історичні дані про навантаження, поточний стан сервера, прогнозований трафік та інші релевантні параметри.

На основі глибокого аналізу цих даних, функція формує оптимальне рішення для налаштування сервера на заданий проміжок часу (див. рис. 4.6). Це рішення може включати в себе рекомендації щодо збільшення або зменшення обсягу виділеної пам'яті, кількості процесорних ядер, та інших ресурсів, необхідних для забезпечення стабільної та ефективної роботи сервера.

Ця логіка прийняття рішень працює за наступною формулою 4.1:

$$Prediction_t = Model.predict([\, CPU_{t-1}\, Memory_{t-1} \,]) \qquad 4.1$$

Наступним критерієм є Action, який буде відбуватися після визначення предикту. Розраховується за формулою 4.2:

$$Action_t = \begin{cases} \text{"Scale Up"} \quad if\ Prediction_t[0] > 0.8 \\ \text{"Scale Down"}\ if\ Prediction_t[1] > 0.4 \\ \text{"No Action"}\ otherwise \end{cases} \quad 4.2$$

Після визначення Action, відбувається визначення причини (Reason) через яку буде відбуватися збільшення ресурсів серверу. Розраховується за формулою 4.3:

$$Reason_t = \begin{cases} \text{"CPU"} \quad if Prediction_t = Scale\ Up\ and\ CPU_{t-1} > Memory_{t-1} \\ \text{"Memory"}\ if\ Action_t = Scale\ Up\ and\ CPU_{t-1} \leq Memory_{t-1} \\ \text{undefined}\ if\ Action_t = \text{"Scale Down"}\ or\ \text{"}No\ Action\text{"} \end{cases} \quad 4.3$$

Останньою дією є визначення результату (Result), він отримую всі попередні розрахунки та акумулює їх в фінальний результат. Розраховується за формулою 4.4:

$$Result_t = \{action: Action_t,\ resources:$$

$$\begin{cases} \{instanceType\text{: t2.large}\}\ if\ Action_t = \text{Scale Up} \\ \{instanceType\text{: t2.small}\}\ if Action_t = Scale\ Down\ \text{, time: t, reason: Reason}\} \\ undefined\ if\ Action = No\ Action \end{cases} \quad 4.4$$

де CPU – використання CPU в момент часу t;
Memory – використання пам’яті в момент часу t;
Prediction – прогнозоване значення моделі на момент часу t;
Action – дія в момент часу t;
Reason – причина масштабування в момент часу t.

Після формування рішення, система автоматично надсилає запит на центральний сервер управління (хаб), який відповідає за розподіл ресурсів між серверами. Цей запит містить детальну інформацію про необхідні зміни в конфігурації сервера, а також часовий інтервал, протягом якого ці зміни повинні бути застосовані.

Важливою перевагою даної системи є її повна автоматизація. Завдяки використанню CRON - спеціального інструменту для планування завдань у Unix-

подібних системах, всі етапи роботи системи, включаючи навчання моделі прогнозування, аналіз даних, прийняття рішень та внесення змін до конфігурації сервера, відбуваються без необхідності втручання людини.

```
async getResourcePrediction(time: string): Promise<ResourceDecision> {
    if (!this.model) throw new Error('Model not trained yet');

    const { hour, minute } = this.parseTime(time);
    const pastDataPoint = this.findClosestPastData(hour, minute);

    const inputTensor = tf.tensor2d([
      [pastDataPoint.CPUUtilization, pastDataPoint.MemoryUsage],
    ]);
    const prediction = this.model.predict(inputTensor) as tf.Tensor;
    console.log('🚀 ~ MlService ~ predict ~ prediction:', prediction);
    const predictionData = prediction.dataSync();
    console.log('🚀 ~ MlService ~ predict ~ predictionData:', predictionData);
    tf.dispose([inputTensor, prediction]);

    const scaleUpThreshold = 0.7;
    const scaleDownThreshold = 0.3;

    let action: 'Scale Up' | 'Scale Down' | 'No Action' = 'No Action';
    let reason;

    if (predictionData[0] > scaleUpThreshold) {
      action = 'Scale Up';
      reason =
        pastDataPoint.CPUUtilization > pastDataPoint.MemoryUsage
          ? 'CPU'
          : 'Memory';
    } else if (predictionData[1] > scaleDownThreshold) {
      action = 'Scale Down';
    }

    return {
      action,
      resources:
        action !== 'No Action'
          ? { instanceType: action === 'Scale Up' ? 't2.large' : 't2.small' }
          : undefined,
      time,
      reason,
    };
  }
```

Рисунок 4.5 – Функція аналізу ресурсів та передбачення на обраний час (рисунок

створено самостійно)

4.4 Аналіз отриманих результатів

Для обчислення використання серверних ресурсів для застосунку React включає врахування декількох факторів, таких як CPU, пам'ять. Формула може бути розбита на компоненти, що враховують кожну з цих областей.

Для обчислення використання CPU на запит скористаймось формулою 4.5:

$$\text{Вик. } CPU \text{ на запит} = \text{середній час } CPU \text{ на запит} * \text{кількість запитів} \quad (4.5)$$

де середній час CPU на запит – вимірюється час, необхідний для обробки одного запиту;

кількість запитів на секунду – кількість запитів, які ваш сервер обробляє за секунду.

Для обчислення використання Memory на запит скористаймось формулою 4.6:

$$\text{Вик. } Memory \text{ на запит} = \text{середній } Memory \text{ на запит} * \text{кількість запитів} \quad (4.6)$$

де середній Memory на запит – Вимірюється обсяг пам'яті, необхідний для обробки одного запиту;

кількість запитів на секунду – кількість запитів, які ваш сервер обробляє за секунду.

Для оцінки середнього часу виконання візьмемо формулу 4.7:

$$\text{середній час} = \frac{\sum_{\text{запитів}}}{\sum_{\text{часу виконання}}} \quad (4.7)$$

де $\sum_{запитів}$– сума кількості запитів, що були надісланні під експерименту;

$\sum_{часу\ виконання}$ – сумарна кількість часу необхідна на виконання запитів.

Після розрахунків середнього часу виконання та аналізу отриманих даних – створюємо таблицю з результати, та заносимо туди дані (див. табл. 4.1)

Таблиця 4.1 – Результат серднього часу

Рішення	Кількість запитів	Час виконання (мс)	Середня швидкість
З підключення системи аналізу навантаження	500	175000	350
Без підключення системи	500	300000	600

Згідно з проведеним дослідженням, можна зробити висновок, що при збільшенні кількості паралельних запитів до React-додатку з Server-Side Rendering (SSR) спостерігається значне зростання часу відповіді сервера. Це пов'язано з тим, що при обробці великої кількості запитів одночасно ресурси сервера виявляються недостатніми, що призводить до зниження його продуктивності.

Для вирішення цієї проблеми було впроваджено систему прогнозування навантаження сервера, яка дозволила своєчасно виявляти потенційні проблеми з продуктивністю. У результаті аналізу даних, отриманих від системи прогнозування, було прийнято рішення про збільшення обсягу оперативної пам'яті (Memory), доступної серверу. Це дозволило суттєво покращити процес рендерингу сторінок на стороні сервера та скоротити час надання відповідей користувачам.

Завдяки впровадженню системи прогнозування навантаження та збільшення обсягу оперативної пам'яті сервера, вдалося досягти значного покращення

продуктивності React-додатку з SSR. Середній час виконання запитів скоротився на 42%, що свідчить про ефективність застосованих заходів.

На підставі отриманих результатів можна стверджувати, що система динамічного управління серверами є ефективним інструментом для оптимізації ресурсів React та NodeJS додатків. Завдяки можливості своєчасно реагувати на зміни навантаження та адаптувати ресурси сервера до поточних потреб, можна забезпечити високу продуктивність додатків навіть за умов пікових навантажень.

ВИСНОВКИ

Під час виконання даного дослідження та розробки стратегії оптимізації та підвищення продуктивності веб-додатків на базі NodeJS та React з використанням штучного інтелекту. У звіті дослідження було докладно проаналізовано принципи роботи обраних технологій, виявлені їх переваги та недоліки, і сформульовані ключові відмінності між ними.

Результатом проведеного дослідження є звіт, що включає в себе сформульовані вимірні критерії для порівняння технологій. В роботі детально описано та аргументовано використання кожної технології, а також визначено випадки, коли краще відмовитися від їх використання. Ключовими метриками для порівняння технологій були визначені такі аспекти, як безпечність у використанні, розмір накладних витрат пам'яті, швидкість обміну даними та легкість інтеграції.

В рамках цього дослідження були запропоновані актуальні методи вимірювання кожної з визначених метрик. Також були сформульовані критерії та математичні формули, що використовувалися для обчислення числових значень цих метрик. Ці результати стануть основою для подальшої розробки стратегії оптимізації та підвищення продуктивності веб-додатків з використанням NodeJS, React та штучного інтелекту.

Під час проведення експерименту було порівняно дві моделі навантаження серверу з використанням штучного інтелекту (ШІ) для аналізу навантаження серверу, та без.

За результатами експерименту, можна сказати, що за допомогою ШІ, продуктивність додатку зросла на 42%, з кількість запитів 500. При збільшення кількості запитів, розрив між двома моделями використання буде збільшуватись, через нестачу ресурсів.

Отже, впровадження системи інтелектуального використання та аналізу ресурсів серверів стане вагомим кроком в шляху оптимізації роботи додатків на React та NodeJS, та призведе до вагомого підвищення ефективності цих додатків.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Salunkhe, G. ., Nagpurkar, S. ., & Kengale, J. A. . (2024). Boosting Productivity through Deep Learning: Strategies for Enhanced Efficiency. International Journal of Intelligent Systems and Applications in Engineering, 12(13s), 396–406.

2. GitHub Copilot under the hood. How it works and getting the best out of it. | by Julio Almeida | Medium. URL: https://medium.com/@enoch3712/github-copilot-is-under-the-hood-how-it-works-and-getting-the-best-out-of-it-4699d4dc3cd8 (дата звернення 07.04.2024)

3. AI for Smart Resource Optimization: Revolutionizing Compute, Storage, and Network Management | by Veer Bahadur | LinkedIn. URL: AI for Smart Resource Optimization: Revolutionizing Compute, Storage, and Network Management (дата звернення: 08.05.2024

4. Renyu Yang, Paul Townend, Xue Ouyang, YaofengChen (2018). Intelligent Resource Scheduling at Scale: A Machine Learning Perspective. IEEE International Symposium on Service Oriented System Engineering, 3-10. (дата звернення 08.05.20224)

5. Scaling Deep Learning Models in Production for millions of users | by Lucas de Lima Nogueira | Medium. URL: https://medium.com/@lucasdelimanogueira/scaling-deep-learning-models-in-production-for-millions-of-users-779baff25dde(дата звернення 08.05.20224)

6. Introduction – NestJS. URL: https://docs.nestjs.com/ (дата звернення: 30.04.2024).

7. First Steps – NestJS. URL: https://docs.nestjs.com/first-steps (дата звернення: 30.04.2024).

8. For JavaScript- TensorFlow. URL - https://www.tensorflow.org/js (дата звернення: 30.04.2024)

9. Дробицький Д.С., Вечур О.В. Дослідження та розробка стратегії оптимізації та підвищення продуктивності веб-додатків на базі NodeJS та React з використанням штучного інтелекту: VI Міжнародна студентська наукова конференція “ДІДЖИТАЛІЗАЦІЯ НАУКИ ЯК ВИКЛИК СЬОГОДЕННЯ” Суми

03.05.2024 P.117-119 URL: https://doi.org/10.62732/liga-inter-03.05.2024 (дата звернення: 25.05.2024).