Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп'ютерних систем та мереж
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Docker контейнер з системою розпізнавання обличчя, емоцій та рухів на AWS Cloud.

Виконав: студент *IV* курсу, групи *CIc-42*
спеціальності *123 «Комп'ютерна інженерія»*
(шифр і назва спеціальності)

(підпис) Долгушин Я.В. (прізвище та ініціали)

Керівник (підпис) Яцишин В.В. (прізвище та ініціали)

Нормоконтроль (підпис) Луцик Н.С. (прізвище та ініціали)

Завідувач кафедри (підпис) Осухівська Г.М. (прізвище та ініціали)

Рецензент (підпис) (прізвище та ініціали)

Тернопіль
2024

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет *комп'ютерно-інформаційних систем і програмної інженерії*
(повна назва факультету)

Кафедра *комп'ютерних систем та мереж*
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ *Осухівська Г.М.*
(підпис) (прізвище та ініціали)
« ____» _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня *бакалавр*
(назва освітнього ступеня)

за спеціальністю *123 «Комп'ютерна інженерія»*
(шифр і назва спеціальності)

студенту Долгушину Ярославу Валерійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи *Docker контейнер з системою розпізнавання обличчя, емоцій та Рухів на AWS Cloud*

Керівник роботи *Яцишин Василь Володимирович*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «*26*» *04* *2024* року № *4/7-468*

2. Термін подання студентом завершеної роботи

3. Вихідні дані до роботи *Технічне завдання*

4. Зміст роботи (перелік питань, які потрібно розробити)
1. Аналіз технічного завдання
2. Проєктна частина
3. Практична частина
4. Безпека життєдіяльності, основи охорони праці
Висновки

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи хорони праці	Пилипець М. І., д.т.н., проф. каф. МТ		

7. Дата видачі завдання 26.04.2024

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	*Розробка і затвердження технічного завдання*		
2.	*Аналіз технічного завдання*		
3.	*Аналіз вимог та принципів організації системи розпізнавання облич, емоцій та рухів*		
4.	*Проєктування системи розпізнавання облич, емоцій та рухів*		
5.	*Розробка схем і програмного забезпечення ситеми розпізнавання облич, емоцій та рухів*		
6.	*Розробка інструкцій з використання системи*		
7.	*Безпека життєдіяльності, основи охорони праці*		
8.	*Оформлення кваліфікаційної роботи*		
9.	*Попередній захист кваліфікаційної роботи*		
10.	*Захист кваліфікаційної роботи*		

Студент _____ Долгушин Я.В.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Яцишин В.В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Docker контейнер з системою розпізнавання обличчя, емоцій та рухів на AWS Cloud // Кваліфікаційна робота бакалавра // Долгушин Ярослав Валерійович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра комп'ютерних систем та мереж, група СІс-42 // Тернопіль, 2024 // с. – 66, рис. –11, табл. – 0, аркушів А1 – 4, додат. – 2, бібліогр. – 25.

Ключові слова: система розпізнавання, docker контейнер, хмарне середовище, програмне забезпечення.

Кваліфікаційну роботу бакалавра присвячено розгортанню Docker контейнеру з системою розпізнавання обличчя, емоцій та рухів в хмарному середовищі.

На основі результатів огляду та аналізу аналогів розроблено структурну схему системи розпізнавання рухів.

Здійснено обґрунтування вибору технологій та інтерфейсів для розгортання системи та описано процес розробки. Розроблено алгоритм тестування системи та подальшої роботи з нею.

Розглянуто основні питання безпеки життєдіяльності та основ охорони праці, стосовно проєктованої системи та її використання.

ANOTATION

Docker container with face, emotion and motion recognition system based on AWS Cloud // Bachelor's thesis // Dolhushyn Yaroslav Valeriyovych // Ternopil Ivan Puluj National Technical University, Faculty of Computer Information System and Software Engineering, Department of Computer Systems and Networks, group CIs-42 // Ternopil, 2024 // p. – 66, fig. – 11, table – 0, bibliography – 25.

Key words: recognition system, docker container, cloud environment, software.

The bachelor's thesis is devoted to the deployment of a Docker container with a face, emotion, and motion recognition system in a cloud environment. Based on the results of the review and analysis of analogues, a block diagram of the motion recognition system was developed. The choice of technologies and interfaces for deploying the system is justified and the development process is described. An algorithm for testing the system and further work with it is developed. The main issues of life safety and the basics of labor protection regarding the designed system and its use are considered.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

CI (англ. Continuous Integration) – практика перевірки та тестування програмного коду.

Docker – інструментарій для створення та управління Linux-контейнерами, які є ізольованими та переносними.

Dockerfile – текстовий файл без розширення, який містить набір інструкцій для інструмента Docker.

GitHub – сервіс для зберігання розробницького коду у системі контролю версій Git, яку розробив Лінус Торвальдс.

GitHub Actions – сервіс в GitHub, який дозволяє автоматизувати робочі процеси розробки програмного забезпечення прямо у репозиторії.

Kubernetes – інструмент для автоматичного розгортання, масштабування та керування додатками у вигляді контейнерів.

SSL (англ. Secure Sockets Layer) – протокол шифрування, який захищає дані, що передаються між сервером і браузером.

ВСТУП

Технологія розпізнавання облич, емоцій і рухів зараз використовується в різних галузях, від безпеки та охорони здоров’я до розваг і маркетингу.

Наприклад, системи розпізнавання людей активно використовуються для підвищення рівня безпеки аеропортів, банків та інших важливих об'єктів.

У галузі медицини ці технології можуть допомогти проаналізувати емоційні реакції пацієнтів і виявити симптоми стресу та тривоги.

Розваги та маркетинг можуть створювати інтерактивні інтерфейси користувача та персоналізувати рекламні кампанії на основі емоційних реакцій клієнтів.

Ключовим завданням є розробка систем, які ефективно керують цими технологіями в масштабованому та безпечному середовищі.

Розгортання таких систем у хмарі за допомогою контейнерів Docker забезпечує високий рівень продуктивності та гнучкості, забезпечуючи швидке та зручне розгортання, масштабування та оновлення програмного забезпечення.

Хмара AWS надає широкий спектр інструментів і послуг, які допоможуть ефективно розгортати системи розпізнавання та керувати ними.

Зокрема, AWS Lambda дозволяє реалізовувати функції обробки даних без керування серверами, API Gateway забезпечує надійний інтерфейс для взаємодії додатків, S3 використовується для безпечного зберігання даних, а Rekognition, Mediapipe забезпечують потужні можливості аналізу зображень і відео.

Це забезпечує не тільки точність і швидкість обробки даних, але й високу надійність і безпеку роботи системи.

Тому розробка та впровадження Docker-контейнерів із системами розпізнавання облич, емоцій та рухів у хмарі AWS є актуальним завданням, яке сприяє розвитку сучасних технологій та ефективності їх використання в різних сферах.

Ця система досліджує процес створення, налаштування та оптимізації такої системи, а також її тестування та впровадження в реальних умовах.

РОЗДІЛ 1 АНАЛІЗ ВИМОГ ДО DOCKER КОНТЕЙНЕР З СИСТЕМОЮ РОЗПІЗНАВАННЯ ОБЛИЧЧЯ, ЕМОЦІЙ ТА РУХІВ НА AWS CLOUD

1.1 Аналіз вимог до Docker контейнера

1.1.1 Загальні вимоги

Загальні вимоги до системи включають ряд основних характеристик, яким система повинна відповідати для забезпечення ефективної та надійної роботи.

Система повинна забезпечувати безперебійну роботу та мінімальний час простою. Надійність включає захист від апаратних і програмних збоїв і відновлення після них.

Програмне забезпечення повинно працювати стабільно в умовах високого навантаження та максимальної активності. Це включає в себе здатність обробляти великі обсяги даних і підтримувати високу частоту запитів без втрати продуктивності.

Система має бути легко розширюваною для підтримки більшої кількості користувачів, обробки великих обсягів даних і додавання нових функцій. Масштабованість важлива для адаптації до зростаючих потреб користувачів і обробки складніших сценаріїв виявлення.

Важливо, щоб система була енергоефективною. Це зменшує експлуатаційні витрати, особливо при роботі в хмарі протягом тривалого часу. Енергоефективність зменшує споживання ресурсів, знижує витрати на обслуговування та зменшує вплив на навколишнє середовище.

Система повинна бути сумісна з існуючою інфраструктурою та підтримувати інтеграцію з різними типами апаратного та програмного забезпечення. Це включає підтримку стандартних протоколів і API, які забезпечують взаємодію з іншими системами.

Система повинна бути простою в розгортанні та обслуговуванні, включаючи підтримку автоматизованих процесів розгортання (CI/CD) і можливість швидкого налаштування без необхідності виконання складних етапів налаштування.

Система має працювати на різних операційних системах і підтримувати різні типи апаратних платформ. Це дає можливість вибирати середовище розгортання та використовувати різні хмарні та локальні ресурси.

Система повинна забезпечувати відмовостійкість через резервування основних компонентів і здатність автоматично відновлюватися після збоїв. Це знижує ризик втрати даних і забезпечує безперервність роботи.

1.1.2 Вимоги до продуктивності

Швидкість обробки - система повинна забезпечувати високу швидкість обробки даних. Це особливо важливо для забезпечення поведінкових реакцій на події в реальному часі. Наприклад, системи розпізнавання обличчя та емоцій вимагають мінімальної затримки між захопленням та обробкою зображення, щоб забезпечити швидке та точне виявлення та ідентифікацію об’єктів.

Пропускна здатність системи має бути достатньою для обробки великих обсягів даних, включаючи зображення та відео високої роздільної здатності. Це включає в себе здатність обробляти кілька одночасних запитів без істотного впливу на продуктивність.

Наприклад, система, яка обробляє відеопотоки з кількох камер, потребує достатньої пропускної здатності для аналізу кожного зображення в реальному часі.

Мінімізація затримок у передачі та обробці даних має вирішальне значення для забезпечення інтерактивності та точності систем розпізнавання обличчя та емоцій. Це означає, що системи повинні бути оптимізовані для зменшення затримок на кожному етапі обробки даних, від прийому до остаточних рішень і дій. Щоб ефективніше використовувати хмарну інфраструктуру та зменшити витрати на обслуговування, потрібно оптимізувати

використання обчислювальних ресурсів. Це включає балансування навантаження, балансування ресурсів і ефективне керування пам’яттю та процесором для забезпечення максимальної продуктивності.

1.1.3 Вимоги до безпеки

Система повинна забезпечувати захист даних, включаючи шифрування як під час зберігання, так і під час передачі, щоб запобігти несанкціонованому доступу до конфіденційної інформації.

Щоб контролювати доступ до системи та її компонентів, повинні бути реалізовані надійні механізми автентифікації та авторизації. Це включає використання багатофакторної автентифікації та використання рольових моделей для визначення рівнів доступу для запобігання несанкціонованому доступу та зловживанням.

Щоб забезпечити безпеку конфіденційних даних і запобігти несанкціонованому доступу, контроль доступу повинен бути реалізований на різних рівнях системи. Це включає використання політик доступу для регулювання того, хто і що може робити з даними.

Система повинна вести докладні журнали активності та підтримувати засоби моніторингу для виявлення аномалій і потенційних загроз.

Це дозволяє швидко виявляти і реагувати на інциденти безпеки, аналізувати їх причини та запобігати майбутнім проблемам.

Також потрібен механізм швидкого відновлення після аварій, таких як резервне копіювання даних і автоматичне відновлення. Це включає в себе стратегії забезпечення безперервності роботи в разі серйозного збою системи.

1.1.4 Переваги CI/CD

Прискорення розробки CI/CD автоматизує інтеграцію коду та процес розгортання, значно скорочуючи час від розробки до випуску нових функцій або виправлень помилок. Це дозволяє швидше тестувати та впроваджувати нові ідеї, підвищуючи загальну продуктивність розробки.

Автоматизоване тестування як частина процесу CI/CD дозволяє швидко виявляти помилки та дефекти на ранніх стадіях розробки. Це робить код більш стабільним, мінімізує ризик проблем у виробництві та покращує якість кінцевого продукту.

CI/CD процеси дозволяють впроваджувати невеликі, часті зміни, зменшуючи ризик великомасштабних помилок, які важко виправити. Крім того, якщо виникає проблема, легше скасувати зміни, зменшуючи загальний ризик і забезпечуючи безперервність роботи системи.

Continuous Delivery CI/CD забезпечує безперервне надання нових функцій і оновлень користувачам, тим самим підвищуючи задоволеність користувачів. Це дозволяє підтримувати систему в актуальному стані та отримувати нові функції, які відповідають потребам.

Процеси CI/CD покращують співпрацю між розробниками, тестувальниками та адміністраторами шляхом автоматизації та спрощення процесу розгортання. Це дозволяє ефективніше обмінюватися інформацією, швидше вирішувати проблеми та впроваджувати нові функції.

Підхід CI/CD підтримує просте масштабування розробки та тестування. Це особливо корисно для великих проектів і команд, які працюють у різних географічних місцях. Що дозволяє працювати над різними аспектами проекту паралельно, не створюючи конфліктів і затримок.

Процеси CI/CD забезпечують чітке документування змін у коді та системах та ефективний контроль версій. Це полегшує відстеження змін, аналіз історії версій і забезпечує узгодженість між різними етапами розробки та розгортання.

1.1.5 Переваги застосування контейнерів

Такі контейнери, як Docker, пропонують значні переваги для розробки, розгортання та експлуатації систем розпізнавання облич, емоцій і рухів у хмарі AWS.

Контейнерізація дозволяє створити ізольоване середовище для програм, забезпечуючи ефективне керування та використання ними. Забезпечують ізоляцію середовища виконання, дозволяючи програмам працювати незалежно від середовища хоста. Це робить додаток портативним і його можна легко переміщувати між різними середовищами (локальними комп'ютерами, тестовими серверами, хмарними платформами тощо), не змінюючи код.

Контейнери дозволяють швидко запускати і розгортати програми, скорочуючи час запуску до секунд. Це особливо корисно для додатків, які вимагають швидкого масштабування або частих оновлень, оскільки дозволяє швидко реагувати на зміни у вимогах або навантаженні.

Ефективне використання ресурсів, контейнери використовують ресурси ефективніше, ніж традиційні віртуальні машини, оскільки їм не потрібна повна емуляція операційної системи. Це дозволяє розміщувати більше програм на одному хост-сервері, оптимізуючи використання доступних ресурсів.

Containerization забезпечує узгоджене середовище виконання для програм на всіх етапах розробки та розгортання, мінімізуючи проблеми, пов'язані з несумісністю середовища. Це забезпечує стабільність і передбачуваність програми в різних середовищах.

Спрощене керування залежностями містить усі залежності, необхідні програмі для роботи, що спрощує керування та зменшує проблеми сумісності версій. Це полегшує керування складними програмами з багатьма залежностями та гарантує, що вони належним чином функціонують у контейнерах.

Підтримка мікросервісів, дозволяють створювати незалежні керовані сервіси, які можуть взаємодіяти один з одним, що робить їх ідеальними для реалізації архітектур мікросервісів. Це підвищує гнучкість і дозволяє швидко адаптувати систему до нових вимог.

Контейнери дозволяють легко масштабувати програму, додаючи або видаляючи контейнери відповідно до ваших поточних потреб. Це дозволяє системі динамічно адаптуватися до змін навантаження, забезпечуючи високу доступність і продуктивність.

Покращена безпека, контейнери забезпечують додатковий рівень безпеки, ізолюючи програми одна від одної та від операційної системи хоста. Це зменшує ризики для безпеки, оскільки вразливості в одному контейнері не впливають на інші контейнери чи хост-систему.

1.2 Аналіз можливих рішень щодо реалізації Docker-контейнера

1.2.1 Вибір програмної платформи

Docker — це потужний інструмент контейнеризації, який має значний вплив на розробку та розгортання програмних систем, особливо в області розпізнавання облич, емоцій і жестів.

Контейнерізація програми за допомогою Docker дозволяє створювати ізольовані середовища для кожного компонента системи. Кожен контейнер містить усі необхідні залежності, що забезпечує стабільність і незалежність компонентів від змін у системі. Це особливо важливо для складних систем виявлення, де компоненти можуть мати різні вимоги до часу виконання.

Docker спрощує роботу, включаючи всі необхідні бібліотеки та інструменти у контейнери. Це забезпечує однакові умови виконання в різних середовищах, від локальної розробки до виробничої хмарної інфраструктури, що зменшує проблеми, пов’язані із сумісністю залежностей, і підвищує надійність системи.

Docker має чудову інтеграцію з основними хмарними платформами, такими як AWS, Google Cloud і Azure. Це полегшує масштабування програм у хмарних середовищах. Додавати контейнери, які виконують основні функції, можна без втручання в роботу інших компонентів, особливо коли система розпізнавання обличчя стає перевантаженою додатковими вимогами.

Docker значно спрощує процес CI/CD, що має вирішальне значення для забезпечення швидких і надійних оновлень системи. Запровадження автоматизованого тестування та розгортання дає змогу швидко додавати нові функції та виправляти помилки, зберігаючи при цьому стабільність і надійність

кінцевого продукту. Наприклад, нові алгоритми розпізнавання обличчя можна перевірити в ізольованих контейнерах перед інтеграцією в основну систему.

Docker підтримує різноманітні операційні системи та архітектури, надаючи гнучкість у виборі платформи для розробки та розгортання. Це дозволяє працювати в неоднорідному середовищі та використовувати різні ресурси залежно від специфіки та вимог до продуктивності проекту.

Docker зменшує складність налаштування середовища розробки. Розробники можуть швидко розгортати контейнери, які імітують робочі середовища, щоб тестувати нові функції та запускати інтеграційні тести. Це дозволяє виявляти та вирішувати проблеми на ранніх стадіях розробки, зменшуючи ризик під час розгортання.

Docker має обширну документацію та активну спільноту, яка дозволяє швидко вирішувати проблеми та ділитися найкращими практиками. Це значно полегшує інтеграцію Docker в існуючі робочі процеси та покращує ефективність розробки.

Docker у системах розпізнавання. Система розпізнавання обличчя складається з кількох основних компонентів: сервера обробки зображень, модуля розпізнавання емоцій і модуля аналізу поведінки. Кожен із цих компонентів можна розгорнути в окремих контейнерах Docker.

– сервер обробки зображень використовує TensorFlow або OpenCV для попередньої обробки даних і може працювати всередині контейнера зі спеціальними бібліотеками машинного навчання;

– модуль розпізнавання емоцій використовує інший набір інструментів і бібліотек, таких як PyTorch і Keras, і може бути розміщений у власному контейнері;

– модуль аналізу руху може бути інтегрований з камерами та датчиками та обробляти дані в реальному часі в окремому контейнері.

Цей підхід дозволяє легко оновлювати або масштабувати кожен компонент незалежно, не впливаючи на решту системи.

Docker надає інструменти та технології, які значно спрощують і оптимізують процес розробки, тестування, розгортання та масштабування систем розпізнавання обличчя, емоцій і рухів, що робить його ідеальним вибором для таких проектів.

1.2.2 Огляд можливих інструментів для розпізнавання обличчя та емоцій

Розпізнавання обличчя та емоцій вимагає використання різноманітних інструментів, які забезпечують високу точність та надійність.

OpenCV (бібліотека комп'ютерного зору з відкритим вихідним кодом) — це бібліотека загального призначення для обробки зображень і комп'ютерного зору, яка надає широкий спектр інструментів для розпізнавання облич і облич. OpenCV підтримує як класичні методи на основі каскадів Хаара, так і сучасні підходи на основі глибокого навчання.

dlib надає високоточні алгоритми для розпізнавання обличчя та ідентифікації ключових точок (орієнтирів) за допомогою методів глибокого навчання. dlib забезпечує детальний аналіз і підходить для задач розпізнавання обличчя та відстеження.

FaceNet — це передова модель розпізнавання облич, яка використовує підхід глибокого навчання для створення векторних зображень облич. Висока точність і масштабованість FaceNet роблять його придатним для роботи з великими базами даних.

Для розпізнавання емоцій існують спеціальні бібліотеки: FER (Facial Expression Recognition), що спеціалізується на класифікації емоцій на основі зображень обличчя та може ідентифікувати різні емоційні стани, такі як радість, сум, гнів, страх тощо.

DeepFace об'єднує кілька моделей для аналізу обличчя та емоцій і підтримує різні архітектури нейронних мереж, що дає можливість вибирати підходи до розпізнавання емоцій. Використовуючи такі підходи, як Convolutional Neural Networks (CNN), і підтримуючи фреймворки TensorFlow та PyTorch, DeepFace

забезпечує високу точність у задачах розпізнавання емоцій і біометричної автентифікації.

1.2.3 Вибір можливих інструментів для розпізнавання рухів

Виявлення руху вимагає використання інструментів, які можуть точно визначити поставу та рух людини.

MediaPipe — це комплексна структура Google, яка надає модульне рішення для розпізнавання поз, жестів і рухів.

MediaPipe забезпечує розширену інтеграцію з іншими компонентами системи та підтримує роботу в режимі реального часу, що робить його ідеальним для інтерактивних програм.

OpenPose надає потужний інструмент розпізнавання поз людини, який використовує методи глибокого навчання для аналізу положення кінцівок.

OpenPose забезпечує детальний аналіз поз, підтримує одночасну обробку кількох людей у кадрі та підходить для складних сценаріїв розпізнавання дій.

1.2.4 Вибір програмного забезаечення розгортання контейнеру

Використання Docker і Kubernetes для контейнеризації та оркестровки.

Docker є важливим інструментом для контейнеризації через його здатність ізолювати середовища програми та полегшувати керування залежностями. Що дозволяє створювати контейнери, які однаково працюють на різних платформах, забезпечуючи стабільність і переносимість ваших програм.

Kubernetes, з іншого боку, доповнює Docker, автоматизуючи процес розгортання, масштабування та керування контейнерами, забезпечуючи високий рівень надійності та доступності програм.

Ізоляція середовища програми. Кожна програма працює у власному контейнері, який містить усе необхідне для роботи, включаючи бібліотеки, конфігурацію та залежності. Це ізолює програму від інших процесів і системного середовища, зменшуючи ризик конфліктів і проблем із сумісністю.

Забезпечення простим керуванням залежностей. Усі залежності додатків включено безпосередньо в контейнер, що полегшує керування та оновлення ними. Це дозволяє уникнути конфліктів версій і гарантує, що програма поводиться однаково в різних середовищах.

Забезпечення стабільності та портативності. Docker-контейнери функціонують однаково незалежно від платформи, на якій вони розгорнуті (локальна чи хмарна). Це дозволяє легко переміщувати програми між різними інфраструктурами, не вимагаючи жодних змін.

Kubernetes забезпечує ефективну оркестровку контейнерів шляхом автоматизації критичних процесів. Контейнери можна автоматично розгортати на основі визначених правил і конфігурацій. Сюди входять автоматичні оновлення та відновлення програм, які забезпечують постійну інтеграцію та безперервну доставку (CI/CD).

Динамічне масштабування. Kubernetes може автоматично масштабувати кількість контейнерів на основі поточного завантаження програми. Це дозволяє ефективно використовувати ресурси та забезпечує стабільну роботу програми навіть під час пікових навантажень.

Керування ресурсами. Kubernetes контролює розподіл ресурсів між контейнерами, щоб забезпечити оптимальне використання ЦП і пам’яті. Це включає балансування навантаження та відстеження працездатності контейнера, щоб швидко реагувати на проблеми та гарантувати високу доступність ваших програм.

Самовідновлення. Kubernetes автоматично відновлюється після збоїв контейнера, перезапускаючи контейнер або переміщуючи його на інші вузли в кластері. Це забезпечує безперервність роботи програми та мінімізує час простою.

Розгортання в різних середовищах. Kubernetes підтримує різноманітні хмарні платформи та локальну інфраструктуру, що дозволяє створювати гібридні середовища та легко переміщувати програми між ними.

При використанні системи розпізнавання обличчя Docker включає окремі служби, такі як служби обробки зображень, служби розпізнавання емоцій і служби аналізу поведінки.

Kubernetes керує цими контейнерами та надає такі можливості:

– балансування навантаження: за допомогою Kubernetes навантаження служби розпізнавання облич розподіляється між кількома контейнерами, що дозволяє обробляти більше запитів паралельно;

– масштабування: коли навантаження на службу розпізнавання емоцій зростає, Kubernetes автоматично створює додаткові екземпляри контейнерів для обробки запитів;

– безперервні оновлення: автоматичні оновлення контейнерів дозволяють розгортати нові версії служб, не порушуючи роботу системи.

Переваги інтеграції Docker і Kubernetes:

– швидке розгортання: налаштування та налаштування середовища тепер займає менше часу;

– гнучкість: легке масштабування програми на основі попиту та доступних ресурсів;

– висока доступність: зменшення часу простою та забезпечення безперервної роботи;

– стабільність: отримання однакового часу роботи на всіх платформах, зменшуючи ризик несумісності.

Інтеграція Docker і Kubernetes створює потужну екосистему для ефективного керування та розгортання програм.

1.2.5 Планування етапів розробки структури системи

Розробка системи розпізнавання обличчя, емоцій і дій включає наступні етапи:

– підготовка середовища: інфраструктура для розробки та тестування налаштована, включаючи створення контейнерів Docker та встановлення необхідних бібліотек і залежностей;

– розробка моделі: модель навчена розпізнавати обличчя, емоції та рухи. Вибирається архітектура нейронної мережі, вибираються набори даних для навчання та тестування, а модель оптимізується для забезпечення високої точності;

– інтеграція модуля: різні компоненти системи поєднуються, і їх взаємодія забезпечує обробку даних у реальному часі.Для зв'язку між модулями використовується API, який забезпечує передачу даних між модулями;

– розробка інтерфейсу: користувацькі інтерфейси, такі як веб-інтерфейси для візуалізації результатів системи та API для інтеграції з іншими службами.

Хмарне розгортання: для розгортання контейнера використовується хмарна платформа, а автоматичне масштабування ресурсів налаштовано для забезпечення безпеки та надійності системи в хмарному середовищі.

1.2.6 Програмне забезпечення та інструменти розробки

Для розробки систем розпізнавання облич, емоцій і рухів використовується комбінація сучасних технологій і інструментів, які забезпечують ефективність, масштабованість і інтеграцію додатків.

Основними компонентами є технологічний стек MERN для веб-розробки, MediaPipe для аналізу руху та Docker для контейнеризації.

Стек MERN – це набір технологій, які використовуються для створення повнофункціональних веб-додатків. Це включає MongoDB, Express.js, React.js і Node.js.

MongoDB – документно-орієнтована база даних NoSQL для зберігання даних програми. Дозволяє зберігати дані у форматах документів, таких як JSON, що дає можливість зберігати складні дані користувача, результати розпізнавання тощо.

Express.js – легкий і гнучкий веб-фреймворк для Node.js, який використовується для створення серверних програм і API. Дозволяє швидко

розробляти серверні компоненти, обробляти запити та інтегруватися з базами даних та іншими службами.

React.js - бібліотека JavaScript для створення інтерфейсів користувача. React.js дозволяє створювати динамічні інтерактивні веб-додатки, які надають зручні інтерфейси для користувачів системи. Підтримка керування станом компонентів і програм полегшує розробку складних інтерфейсів.

Node.js - серверне середовище для виконання коду JavaScript. Дозволяє створювати високопродуктивні серверні програми з неблокуючою архітектурою введення-виведення, яка забезпечує масштабованість і ефективність обробки запитів.

Стек MERN надає повний набір інструментів для створення сучасних веб-додатків, підтримуючи логіку на стороні клієнта та на стороні сервера, а також інтегруючи інструменти розпізнавання облич, емоцій і руху в єдиний веб-інтерфейс.

MediaPipe - це платформа Google, яка надає модульне рішення для обробки мультимедійних даних у реальному часі. Використовує найновіші алгоритми комп'ютерного зору та машинного навчання для підтримки поз, жестів і аналізу рухів.

Виявлення пози - MediaPipe надає модуль для визначення пози людського тіла та скелетних точок. Це дозволяє з високою точністю відстежувати поставу та рухи, що важливо для додатків, зосереджених на аналізі фізичних рухів.

Розпізнавання жестів - платформа підтримує модулі для розпізнавання та класифікації жестів. Це дозволяє створювати інтерактивні програми, які реагують на жести користувача та надають нові способи взаємодії.

Розпізнавання обличчя та виразу - MediaPipe можна використовувати для розпізнавання обличчя та аналізу виразу. Надає інструменти для визначення ключових точок на обличчі та оцінки виразу обличчя, а також дозволяє інтегрувати можливості розпізнавання емоцій.

Docker - це інструмент контейнеризації, який дозволяє створювати ізольовані середовища для запущених програм. Забезпечує керування

залежностями, контроль версій та ізоляцію середовища для спрощення процесу розробки, тестування та розгортання.

Docker використовується для контейнеризації системних компонентів, розроблених на основі стеку MERN і MediaPipe, що забезпечує зручність розгортання та керування системами як у локальних, так і в хмарних середовищах.

Поєднання MERN Stack, MediaPipe і Docker забезпечує гнучкість, ефективність і масштабованість для розробки систем розпізнавання.

Веб-інтерфейс на основі стеку MERN забезпечує зручний доступ до функціональних можливостей системи, а MediaPipe реалізує аналіз мультимедійних даних, таких як рухи та жести. Docker забезпечує контейнеризацію та просте розгортання програми.

Це спрощує інтеграцію різних системних компонентів і забезпечує розгортання в різних середовищах, включаючи хмарні платформи.

Разом ці інструменти створюють ефективну інтегровану систему, яку можна розгортати та масштабувати відповідно до потреб користувачів, зберігаючи при цьому високу продуктивність і простоту використання.

РОЗДІЛ 2 ПРОЕКТУВАННЯ DOCKER КОНТЕЙНЕРА З СИСТЕМОЮ РОЗПІЗНАВАННЯ ОБЛИЧЧЯ, ЕМОЦІЙ ТА РУХІВ НА AWS CLOUD

2.1 Вибір апаратної платформи

2.1.1 Обґрунтування вибору хмарного середовища

Хмарне середовище AWS (Amazon Web Services) обрано для забезпечення системи розпізнавання обличчя, емоцій і рухів. Основними причинами такого вибору є:

- широкий спектр послуг;
- масштабованість;
- надійність;
- безпека;
- глобальна доступність;
- економічні показники;
- інтеграція з CI/CD.

AWS пропонує різноманітні інструменти та послуги, включаючи обробку даних, зберігання, керування мережею та безпеку. Такі сервіси, як AWS Lambda, Amazon S3, підтримують створення складних інтеграційних рішень.

Середовище надає гнучкі можливості масштабування ресурсів, які дозволяють легко адаптувати інфраструктуру до мінливих навантажень і забезпечують високу продуктивність і доступність системи.

Хмара відома своєю SLA (Service Level Agreement), яка гарантує високу надійність і доступність своїх послуг, а також можливість резервного копіювання та відновлення даних.

Сервіс надає розширені функції безпеки, такі як контроль доступу, шифрування даних, інструменти моніторингу та керування безпекою.

AWS має розгалужену мережу центрів обробки даних по всьому світу, що дозволяє розгортати системи ближче до кінцевих користувачів, зменшуючи затримку та покращуючи продуктивність. Також пропонує різноманітні моделі оплати, включаючи оплату за використання, щоб допомогти оптимізувати витрати на інфраструктуру.

Інтегрується з популярними інструментами CI/CD, такими як Jenkins і GitLab CI/CD, щоб полегшити автоматизацію процесів розробки та розгортання.

Основні сервіси AWS зображено на рис. 2.1.

Рисунок 2.1 – Основні сервіси AWS для роботи з хмарами

2.1.2 Вибір способу розгортання Docker контейнеру

Для розгортання контейнерів Docker у хмарі AWS розглядалися такі варіанти.

Amazon ECS (Elastic Container Service): Платформа керування контейнерами, яка забезпечує високу масштабованість та інтеграцію з іншими службами AWS. ECS забезпечує автоматичне масштабування контейнерів і забезпечує безпеку через IAM (Identity and Access Management).

Amazon EKS (Elastic Kubernetes Service): Керована служба Kubernetes, яка дозволяє розгортати, керувати та масштабувати програми-контейнери за допомогою стандартного API Kubernetes. Підходить для програм, які вже використовують Kubernetes.

AWS Fargate: безсерверний варіант для ECS і EKS, який дозволяє запускати контейнери без керування серверами. Fargate автоматично масштабує ресурси та полегшує керування інфраструктурою.

AWS Lambda: функція як послуга (FaaS), яка може виконувати короткострокові функції у відповідь на події, але не підходить для тривалих контейнерних програм.

Heroku — це платформа як послуга (PaaS), яка спрощує розгортання, керування та масштабування програм. Система пропонує практичний підхід до розгортання контейнерів Docker та інтеграції з різними мовами програмування та фреймворками. Основні переваги:

Інтуїтивно зрозуміла платформа з простим процесом розгортання за допомогою Heroku CLI, керує інфраструктурою, підтримка контейнерів Docker через Heroku Container Registry, легко розширюється функціональність програми за допомогою доступу до великої кількості програм та інтеграцій.

Типи розгортання додатків за видом середовища зображено на рис. 2.2.

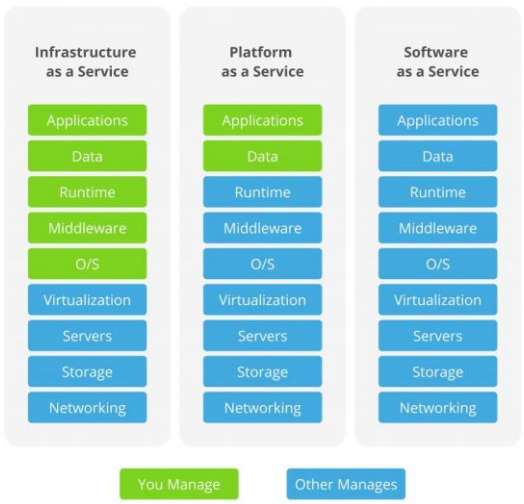

Рисунок 2.2 – Типи розгортання додатків за видом середовища

Обрано Heroku для розгортання контейнерів Docker з наступних причин:

- просте та швидке розгортання;
- інтеграція Docker;
- автоматизоване керування інфраструктурою;
- гнучкість і масштабованість;
- екосистема додатків.

2.1.3 Налаштування доменного імені та SSL сертифікату

Щоб забезпечити безпечний доступ до системи, потрібно налаштувати ім’я домену та сертифікат SSL. Це використовується для забезпечення зручного доступу до системи.

AWS надає можливість зареєструвати своє доменне ім’я через Amazon Route 53, який також дозволяє керувати своїми записами DNS та інтегруватися з іншими службами AWS.

Сертифікат SSL забезпечує безпечне з’єднання між клієнтом і сервером. Менеджер сертифікатів AWS (ACM) спрощує керування сертифікатами SSL/TLS, включаючи автоматичне оновлення та інтеграцію з іншими службами AWS, такими як Elastic Load Balancer (ELB).

2.2 Проектування програмного забезпечення

2.2.1 Налаштування Docker контейнеру

Процес налаштування контейнера Docker передбачає створення образу Docker, який містить усі компоненти, необхідні для роботи системи. Створення файлу, який визначає кроки для створення образу, включаючи базовий образ, встановлення залежностей, копіювання файлів, налаштування середовища та команди для запуску програми.

Створення образу Docker – створення зображення з Dockerfile за допомогою Docker CLI. Це зображення містить усі компоненти, необхідні для

запуску програми, зокрема: сервери, бази даних, інтерпретатори мови, бібліотеки та інші залежності.

Тестування зображень Docker – перевірка функціональності образу у локальному середовищі, щоб переконатися, що програма працює належним чином перед розгортанням у хмарі.

Архітектуру Docker зображено на рис 2.3.

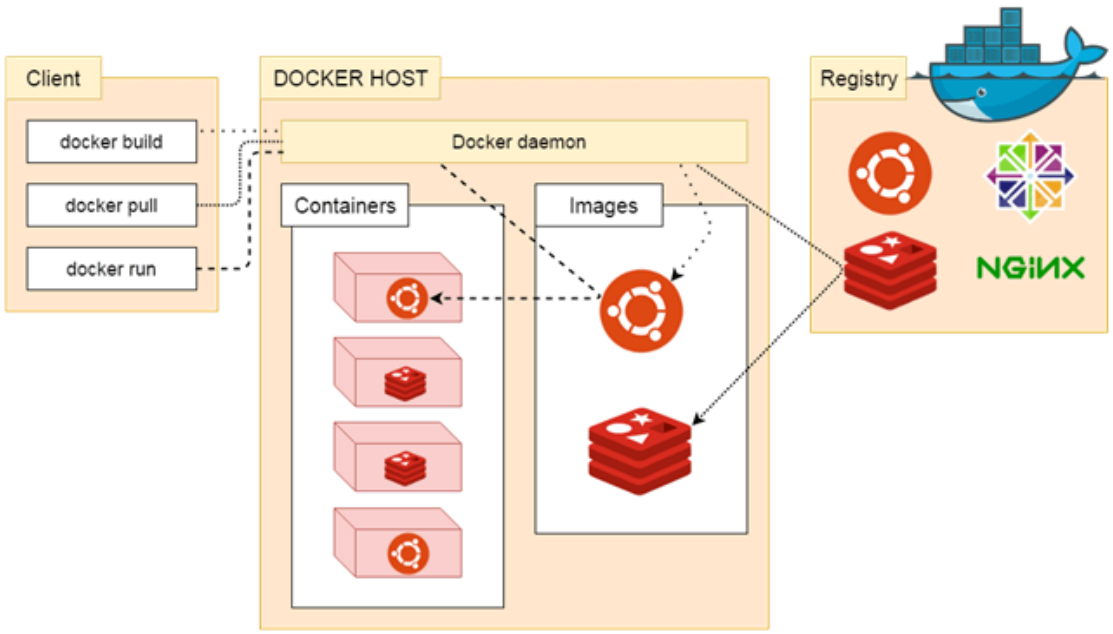

Рисунок 2.3 – Архітектура Docker

Зберігання зображення в Amazon ECR (Elastic Container Registry) – можливість зберігати скомпільовані зображення в приватному сховищі AWS ECR для доступу до зображень.

2.2.2 Використання MERN стеку для розробки системи

Для розробки системи обрано стек MERN, що складається з MongoDB, Express.js, React і Node.js.

Основними причинами вибору стека MERN є:

MongoDB: Орієнтована на документи база даних NoSQL, яка забезпечує високу продуктивність і гнучкість для зберігання даних про обличчя, емоції та рухи.

Express.js: легкий і швидкий серверний фреймворк для Node.js, який дозволяє легко створювати API і обслуговувати запити HTTP.

React: Потужна бібліотека для створення інтерфейсу користувача, яка дозволяє створювати динамічні, високопродуктивні веб-додатки.

Node.js: серверна платформа, яка забезпечує асинхронну обробку подій і високу продуктивність для керування запитами та обробки даних.

Розробка на основі стеку MERN включає розробку Backend API для обробки запитів, керування даними та інтеграції з базою даних MongoDB за допомогою Express.js і Node.js.

2.2.3 Використання MediaPipe

MediaPipe — це платформа обробки мультимедіа в реальному часі, розроблена Google. Система надає потужні бібліотеки обробки зображень і відео та важливі інструменти для систем розпізнавання обличчя, емоцій і рухів.

Основними перевагами використання MediaPipe є:

Реалізація в режимі реального часу, MediaPipe забезпечує обробку відео та зображень у реальному часі, створюючи інтерактивну систему, яка може миттєво реагувати на зміни.

Модульність, MediaPipe забезпечує модульну архітектуру, яка дозволяє легко інтегрувати різні алгоритми розпізнавання облич, емоцій і рухів, забезпечуючи гнучкість у налаштуванні та розширенні системи.

Підтримка різних платформ, MediaPipe підтримує різноманітні платформи, включаючи мобільні пристрої, що дозволяє створювати рішення, які працюють на різних пристроях і операційних системах.

Висока продуктивність, MediaPipe оптимізовано для високої продуктивності під час обробки мультимедійних даних, що є критичним для програм, які вимагають обробки в реальному часі.

Інтеграція MediaPipe включає використання бібліотеки MediaPipe для обробки зображень і відео.

РОЗДІЛ 3 РОЗРОБКА СИСТЕМИ РОЗПІЗНАВАННЯ ОБЛИЧЧЯ, ЕМОЦІЙ ТА РУХІВ НА AWS CLOUD

3.1 Реалізація програмного коду

3.1.1 Написання клієнтської частини системи

Клієнтська частина відповідає за надання зручного інтерфейсу для взаємодії з користувачами, відображення даних і маніпулювання системою.

React використовується для розробки, що дозволяє динамічно та швидко відображати контент.

Клієнтська архітектура використовує компонентний підхід.

Реалізація основних компонентів React для зовнішнього керування, таких як розпізнавання обличчя, аналіз настроїв, відстеження рухів, інформаційні панелі та налаштування.

Routing, керування маршрутизацією та навігацією між сторінками програми за допомогою React Router.

Керування станом за допомогою Redux дозволяє централізовано зберігати та керувати стани програми, включаючи дані користувача та налаштування.

Реалізація візуалізації, відображення результатів розпізнавання обличчя, емоцій і руху у формі графіки, відеоанотацій та інших візуальних елементів. Взаємодія з сервером, за допомогою бібліотеки Axios, щоб надсилати запити на сервер, обробляти медіадані та отримувати результати аналізу.

Тестування клієнтської частини, написані тести для компонентів за допомогою Jest та React Testing Library для забезпечення їх правильної роботи, використання Cypress для автоматичного тестування основних сценаріїв використання системи, таких як відображення результатів аналізу.

Приклад коду клієнтської частини зображено на рис. 3.1.

```
if (videoElement.current) {
      camera.current = new Camera(videoElement.current, {
        onFrame: async () => {
    await faceMesh.current.send({ image: videoElement.current
 });
          await holistic.current.send({ image:
videoElement.current });
        },
        width: maxVideoWidth,
        height: maxVideoHeight,
      });
      camera.current.start();
    }
  }, []);
```

Рисунок 3.1 – Лістинг коду клієтської частини для відображення камери

3.1.2 Написання серверної частини системи

Серверна частина відповідає за обробку запитів від клієнтів, виконання аналізу медіаданих і управління базою даних.

Архітектура серверної частини складається з: маршрути, для обробки таких запитів, як /api/users, /api/wishlist, /api/order.

Контролери, які обробляють логіку запиту, наприклад отримання медіаданих, виклик відповідного алгоритму обробки та надсилання результатів назад клієнту.

Сервіси, інкапсулює логіку обробки медіафайлів та інтегрується з MediaPipe.

Функції серверної частини, полягають в обробці мультимедійних даних.

Взаємодія з базою даних, виконання операції CRUD для збереження результатів аналізу та керування параметрами системи, а також забезпечення надійного і швидкого доступу до даних для подальшого використання в процесі прийняття рішень.

Приклад серверної частини коду зображено на рис. 3.2.

```
const { analyzeFace } = require('../services/faceService');
const MediaPipe = require('mediapipe');
exports.analyzeFace = async (image) => {
  // Logic to analyze face using MediaPipe
  // Return analysis results
};
```

Рисунок 3.2 – Лістинг коду серверної частини для обробки запиту

Тестування серверної частини полягає в написанні тестів для контролерів та сервісів за допомогою Jest для перевірки правильної роботи логіки обробки, тестування взаємодії між різними компонентами серверної частини, зокрема обробка запитів та інтеграція з базами даних.

3.1.3 Підключення баз даних

База даних MongoDB використовується для зберігання даних, забезпечуючи гнучкість і продуктивність для зберігання структурованих і неструктурованих даних. Налаштування бази даних полягає в створенні моделі даних, визначенні схеми та моделей для зберігання облич, емоцій, рухів і налаштувань системи за допомогою Mongoose.

Приклад підключення mongoose до баз даних, зображено на рисунку 3.3.

```
const mongoose = require('mongoose');
require('dotenv').config();
mongoos.connect(process.env.MONGO_Vikki)
     .then(()=>console.log("DBConnection S  uccessfull!"))
     .catch((err)=>{console.log(err)});
```

Рисунок 3.3 – Лістинг коду підключення бази даних з mongoose

Конфігурація підключення до MongoDB через Atlas або локальну установку здійснюється за допомогою URI, що зберігається у файлі конфігурації, де вказуються параметри підключення, такі як ім'я користувача, пароль, адреса хоста та ім'я бази даних. Це забезпечує централізоване та зручне управління

налаштуваннями підключення, спрощуючи процес зміни параметрів у випадку переносу або оновлення серверів.

3.2 Налаштування та оптимізація системи

3.2.1 Налаштування взаємодії користувача з системою

Успішна взаємодія користувача з системами розпізнавання обличчя, емоцій і рухів вимагає надання інтуїтивно зрозумілого і зручного інтерфейсу, який дозволяє користувачам легко отримувати результати аналізу та налаштовувати параметри системи.

Найважливіші аспекти налаштування взаємодії включають дизайн інтерфейсу, інтеграцію сервера та швидкість відтворення.

Дизайн інтерфейсу, полягає в простоті та інтуїтивності, інтерфейс простий у використанні та містить мінімальну кількість кроків для виконання основних завдань, таких відео для аналізу. Приклад дизайну зображено на рис. 3.4.

Візуалізація, забезпечує чітку візуалізацію результатів аналізу. Наприклад, результати розпізнавання відображатються в рамці, що оточує виявлене обличчя.

Інтеграція з сервером, оптимізація механізму аналізу медіафайлів і відправлення запитів на сервер для обробки відповідей. Це включає використання асинхронних запитів для безперебійної роботи програми. Обробка помилок, щоб користувачі отримували чіткі повідомлення про помилки, якщо виникає проблема з підключенням до сервера.

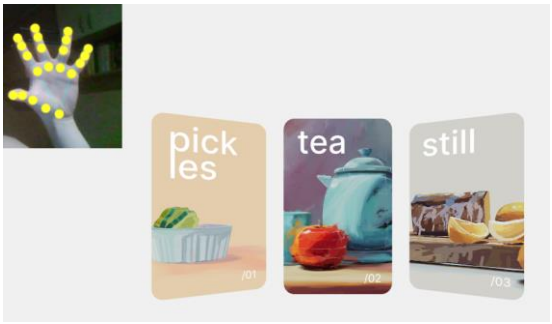

Рисунок 3.4 – Приклад інтерфейсу користувача

Аналіз використання, розуміння, як користувачі взаємодіють із системою, і визначення функції та проблемних областей, які найчастіше використовуються.

3.2.2 Оптимізація продуктивності системи

Оптимізація продуктивності є ключовим аспектом для досягнення швидких і ефективних систем розпізнавання обличчя, емоцій і дій. Це включає покращення обробки медіаданих, оптимізацію взаємодії з хмарними службами та скорочення часу, необхідного для виконання основних операцій.

Оптимізація обробки медіаданих, а саме паралельна обробка, використання паралельної обробки даних для прискорення аналізу зображень і відео. Наприклад, обробка різних частин зображення або відеозображення одночасно.

Використання кешування, щоб уникнути повторної обробки тих самих медіаданих. Це збільшує швидкість аналізу без істотної шкоди для якості.

Оптимізація взаємодії з хмарними службами, використовуються мережі доставки вмісту (CDN) для доставки вмісту, що прискорює завантаження медіафайлів. Приклад налаштування оптимізації зображено на рисунку 3.5.

```
const analyzeFace = async (image) => {
  const analysisResults = await Promise.all([
    detectFaces(image),
    analyzeEmotions(image),
    trackMovements(image),
  ]);
  return {
    faces: analysisResults[0],
    emotions: analysisResults[1],
    movements: analysisResults[2],
  };
};
```

Рисунок 3.5 – Лістинг прикладу оптимізації коду.

Покращення взаємодії API, оптимізація викликів API, що зменшує затримку та підвищує ефективність використання ресурсів.

Скорочення часу виконання основних операцій, аналіз вузьких місць, інструменти профілювання та моніторингу, для виявлення та усунення вразливих місць продуктивності системи.

Оптимізація коду, рефакторинг коду для підвищення ефективності та швидкості виконання. Це включає використання ефективніших алгоритмів і структур даних.

3.2.3 Налаштування GitHub Actions

Щоб забезпечити постійну інтеграцію та доставку (CI/CD), налаштовано GitHub Actions, які можуть автоматизувати процес тестування, створення та розгортання систем. Це знижує ризик помилок і забезпечує високу якість програмного забезпечення. Приклад конфігурації зображено на рис. 3.6.

Базове налаштування робочого процесу. Перевірка коду, автоматичне запускання модульних тестів для кожного запиту на фіксацію або витягування, щоб переконатися, що код працює правильно.

```
name: CI/CD Pipeline
  deploy:
    runs-on: ubuntu-latest
    needs: build
    steps:
    - name: Checkout code
      uses: actions/checkout@v2
    - name: Configure AWS credentials
      uses: aws-actions/configure-aws-credentials@v2
      with:
        aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
        aws-secret-access-key: ${{
secrets.AWS_SECRET_ACCESS_KEY }}
        aws-region: ${{ secrets.AWS_REGION }}

    - name: Deploy to AWS ECS
      run: |
        aws ecs update-service --cluster my-cluster --service
my-service --force-new-deployment
```

Рисунок 3.6 – Лістинг коду конфігурації YAML файлу

Створення програми. Налаштуйтування процесу збирання для створення контейнерів Docker з останніми версіями клієнта та серверної частини.

Розгортання. Після успішного тестування та збірки система автоматично розгортається у хмарному середовищі AWS. Налаштування робочого процесу на GitHub. Створено файл YAML, який описує робочий процес, включаючи тригери, кроки виконання та умови для успішного завершення.

Безперервна інтеграція та розгортання, AWS CLI для автоматизації розгортання контейнерів для таких служб, як ECS (Elastic Container Service) і EKS (Elastic Kubernetes Service). Налаштування моніторингу робочого процесу і сповіщення про статус збірки та розгортання, щоб швидко реагувати на проблеми та помилки у процесі CI/CD.

3.3 Тестування системи в реальних умовах

3.3.1 Проведення тестових сценаріїв

Тестування систем розпізнавання обличчя, емоцій і дій у реальних ситуаціях є важливим кроком для підтвердження їх ефективності, продуктивності та стабільності.

Тестовий сценарій — це набір конкретних умов і операцій, призначених для оцінки різних аспектів функціональності системи.

Розроблено кілька важливих сценаріїв для перевірки продуктивності систем розпізнавання облич, емоцій і дій, індивідуальний аналіз — за різних умов освітлення та під різними кутами.

Груповий аналіз — перевірка здатності системи розпізнавати обличчя на групових фотографіях або відео кількох людей.

Динамічні умови — перевірка здатності системи виявляти рухомі обличчя, такі як обертання голови або зміна виразів.

Сценарії аналізу емоцій. Базове розпізнавання емоцій, перевірка здатності системи точно ідентифікувати основні емоції (радість, сум, гнів, страх, здивування).

Комплексний аналіз емоцій — тест на здатність розпізнавати більш складні або змішані емоції. Перевірка здатності системи відстежувати зміни емоцій з часом, наприклад, під час перегляду відео.

Сценарії аналізу руху. Виявлення руху, перевірка здатності системи розпізнавати основні рухи тіла, такі як жести рук і поза.

Рух за наявності перешкод — Перевірка, чи можна виявити рух, навіть якщо частини тіла тимчасово приховані або заблоковані.

Сценарії продуктивності. Тестування навантаження, перевірка здатності системи обробляти велику кількість запитів одночасно без втрати продуктивності.

Тестування за різних мережевих умов — аналіз продуктивності системи за різних мережевих пропускних спроможностей і затримок.

Сценарії безпеки. Тестування вразливості, перевірка стійкості системи до різних типів атак, включаючи ін’єкції SQL, атаки XSS і атаки типу «людина посередині».

Тестування автентифікації та авторизації, перевірка роботи механізмів контролю доступу, включаючи автентифікацію користувача та обмеження доступу.

3.3.2 Оцінка ефективності використання хмарних середовищ

Щоб оцінити ефективність хмарного середовища, потрібно враховувати кілька ключових показників, таких як продуктивність, масштабованість, стабільність, вартість і безпека.

Продуктивність. Час відповіді — показник середнього часу відповіді системи для різних типів запитів.

Пропускна здатність. Оцінка максимальної кількості запитів, які система може обробити за одиницю часу без втрати продуктивності.

Масштабованість. Автомасштабування — перевірка здатності системи автоматично масштабувати ресурси у відповідь на зміни навантаження.

Гнучкість розгортання. Легке розширення системи, щоб перевірити здатність підтримувати нові функції або підключати додаткові служби.

Стабільність. Час безвідмовної роботи — оцінка середньої доступності системи за фактичних умов експлуатації.

Відмовостійкість. Перевірка здатності системи відновлюватися після збою або переривання обслуговування, наприклад, коли один із її компонентів виходить з ладу.

Вартість. Операційна вартість — аналіз вартості використання хмарних ресурсів, включаючи обчислювальну потужність, зберігання та передачу даних.

Оптимізація витрат. Оцінка можливості зниження витрат.

Приклад оцінки ефективності зображено на рис. 3.7.

```
dashboards:
  - name: System Performance
    widgets:
      - type: metric
        x: 0
        y: 0
        width: 24
        height: 6
        properties:
          metrics:
            - [ "AWS/EC2", "CPUUtilization", "InstanceId", "i-
1234567890abcdef0" ]
            - [ ".", "NetworkIn", ".", "." ]
            - [ ".", "NetworkOut", ".", "." ]
          period: 300
          stat: Average
          region: us-west-2
          title: System Performance Metrics
```

Рисунок 3.7 – Лістинг прикладу інформаційної панелі показників.

Використання надлишкових екземплярів або оптимізація використання мережевих ресурсів.

Безпека. Конфіденційність — оцінка ефективності заходів безпеки для захисту даних користувача, включно з шифруванням у спокої та під час передачі.

Контроль доступу, перевірка ефективності механізмів, які контролюють доступ до системних ресурсів.

3.3.3 Аналіз результатів тестування

Після тестування результати були детально проаналізовані для визначення рівня відповідності системи встановленим вимогам і виявлення можливих проблем і недоліків.

Проаналізовано середній час відгуку(250ms) та частоту відмов(0.02%). Оцінка того, наскільки результати тесту відповідають системним вимогам, щодо продуктивності чи стабільності склала 89%. Ідентифіковано критичні проблеми, виявлені під час тестування, і їх вплив на загальну функціональність системи.

На основі отриманих результатів складено вимоги щодо покращення інтерфейсу користувача та продуктивності системи.

Тестування системи в реальних умовах є критичним для забезпечення її готовності до використання в практичних сценаріях. Ефективне проведення тестових сценаріїв, оцінка використання хмарних середовищ та детальний аналіз результатів тестування дозволили гарантувати, що система розпізнавання облич, емоцій та рухів відповідає високим стандартам якості та надійності.

3.4 Оцінка результатів та перспективи розвитку

3.4.1 Аналіз досягнутих результатів

Під час розробки та тестування системи було отримано ряд результатів, які можна підсумувати наступним чином.

Точність розпізнавання – підтримка використовуваних алгоритмів і технологій, таких як MediaPipe є доволі високою.

Продуктивність – система забезпечує незмінно високу продуктивність під час обробки великих обсягів запитів завдяки ефективному використанню хмарних ресурсів AWS.

Стабільність – система надійно працює в різних робочих умовах з мінімальними порушеннями функціональності.

Масштабованість – система вільно маштабується у міру зміни навантаження, підтримуючи стабільну роботу системи навіть під час пікових навантажень.

Безпека та відповідність – заходи, реалізовані для захисту даних користувача, такі як шифрування та контроль доступу, відповідають сучасним стандартам безпеки.

Відповідність – система повністю відповідає функціональним вимогам, продуктивності та вимогам безпеки.

3.4.2 Виявлені проблеми та способи їх вирішення

Незважаючи на досягнуті результати, під час розробки та тестування системи було виявлено декілька проблем, які потребували вирішення:

Висока вартість хмарних ресурсів.

Проблема хмарного середовища AWS з високою вартістю використання обчислювальних ресурсів та зберігання даних.

Рішення полягає в оптимізовації використання ресурсів шляхом реалізації надлишкових екземплярів, використання автомасштабування та оптимізації алгоритмів обробки даних для зменшення споживання ресурсів.

Межі точності розпізнавання.

Проблема точністі розпізнавання обличчя та емоцій знижується за складних умов, таких як погане освітлення, дощ, часткове перекриття поверхонь.

Рішення полягає у використанні додаткових методів попередньої обробки зображень, такі як, корекція освітлення та розпізнавання обличчя.

Проблеми з інтеграцією.

Проблема – систему важко інтегрувати з деякими зовнішніми службами та інструментами.

Рішення полягає у додаткових модулях для інтеграції з певними службами та використання API для забезпечення сумісності.

Проблеми з конфігурацією кінцевого користувача.

У деяких кінцевих користувачів виникають проблеми з налаштуванням і використанням системи.

Рішення полягає у розробці більш детальної документації та створенні інтерфейсу користувача для легкого налаштування та використання.

3.4.3 Перспективи подальшого розвитку та удосконалення системи

Враховуючи досягнуті результати та виявлені проблеми, можна виділити кілька напрямів подальшого розвитку та вдосконалення системи.

Покращення функцій. Додаткові алгоритми. Представлення нових алгоритмів для розпізнавання більш складних емоцій і специфічних рухів, покращення функціональності рекламованої системи.

Нові послуги. Інтеграція з новими хмарними службами та інструментами для розширення функціональності системи.

Оптимізація продуктивності. Покращення алгоритму. Оптимізація існуючих алгоритмів для зменшення часу обробки та споживання ресурсів.

Покращення масштабованості. Використання нових метод автомасштабування та балансування навантаження для покращення продуктивності системи.

Покращення безпеки. Додаткові заходи безпеки. Впровадження нових методів захисту даних, зокрема – багатофакторна автентифікація та розширені механізми шифрування.

Відповідність, адаптація системи до нових стандартів безпеки та захисту даних.

Розробка більш інтуїтивно зрозумілого і зручного інтерфейсу для кінцевих користувачів. Детальна документація. Оновлення та розширення документації.

РОЗДІЛ 4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

4.1 Характеристика життєдіяльності людини у системі, “людина – машина – середовище існування”.

Розглядаючи людину в безперервній взаємодії з довкіллям, система “людина – машина – середовище існування” включає три складові елементи: “людина”, “машина” та “середовище”.

Основні характеристики системи “людина – машина – середовище існування”:

а) структура системи. Людина представлена трьома аспектами;
   1) людина 1 – людина, яка виконує певні дії;
   2) людина 2 – людина, як біологічний об’єкт, що впливає на середовище;
   3) людина 3 – людина, що відбиває свій психофізіологічний стан;
   4) машина: технічні засоби, які взаємодіють із людиною та середовищем;
   5) середовище: навколишнє середовище, яке може включати устаткування, сировину, предмети побуту;

б) взаємодія елементів;
   1) вплив “людини” на “середовище”;
   2) вплив “середовища” на психофізіологічний стан “людини”;
   3) вплив “середовища” на дії “людини”;
   4) вплив “середовища” на роботу “машини”;
   5) вплив “машини” на психофізіологічний стан “людини”;

в) цілісність системи;

1) усі елементи системи служать для досягнення єдиної мети;

г) взаємозалежність елементів;

1) зміна стану одного елемента або підсистеми призводить до змін у інших елементах і підсистемах;

ґ) роль людини;

1) у побутовій і техносфері головну роль відіграє людина;

2) у біосфері всі елементи системи мають рівнозначну роль;

д) характер поведінки;

1) поведінка системи та її елементів може бути як детермінованою, так і ймовірнісною, залежно від динаміки довкілля та психофізіологічних особливостей людини;

е) ігрова ситуація;

1) людина постійно приймає рішення, обираючи певну стратегію.

Взаємодія людей у системі:

– система “людина – машина – середовище існування” може охоплювати індивіда, колектив, спільноту або суспільство загалом, враховуючи зв'язки між людьми, що залежать від типу колективу, спорідненості, освіти, соціального статусу, політичних і релігійних переконань;

Визначення рівня системи:

– залежно від поставленої мети, система “людина – машина – середовище існування” може розглядатися на різних рівнях: робочого місця, приміщення, об'єкта, населеного пункту, регіону, країни чи людства загалом. Занадто вузький рівень може призвести до пропуску важливих джерел небезпек, тоді як занадто широкий ускладнює аналіз;

Аналіз взаємодії людини з іншими елементами:

– необхідно розглянути модель сприйняття та обробки інформації в системі “людина – машина – середовище існування”. Важливо сформулювати умови безпеки життєдіяльності людини, враховуючи

такі види сумісності: інформаційну, біофізичну, енергетичну, просторово-антропометричну, техніко-естетичну та соціальну. Необхідно також запропонувати заходи і засоби для забезпечення оптимальних умов життєдіяльності людини, колективу, соціуму та людства в цілому.

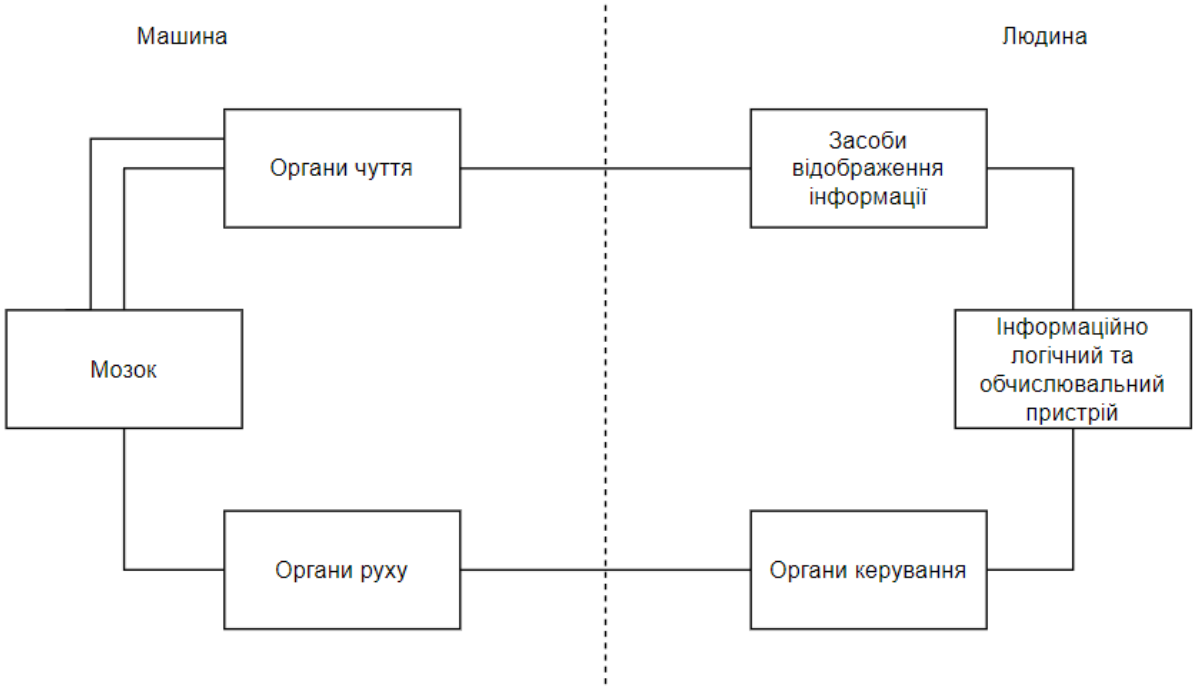

Рисунок 4.1 – Схема системи “Людина-машина”

Необхідно розглянути модель сприйняття та обробки інформації в системі “людина – машина – середовище існування”. Важливо сформулювати умови безпеки життєдіяльності людини, враховуючи такі види сумісності: інформаційну, біофізичну, енергетичну, просторово-антропометричну, техніко-естетичну та соціальну. Необхідно також запропонувати заходи і засоби для забезпечення оптимальних умов життєдіяльності людини, колективу, соціуму та людства в цілому.

4.2 Заходи щодо забезпеченню безпечної роботи при ремонті технологічного обладнання.

Техніка безпеки при ремонті технологічного обладнання:

а) загальні положення;

1) забезпечення безпечних умов праці при ремонті технологічного обладнання є пріоритетним завданням. Всі ремонтні роботи повинні проводитися відповідно до встановлених норм і правил техніки безпеки;
2) перед початком ремонтних робіт персонал повинен бути ознайомлений з інструкціями з охорони праці, пройти відповідний інструктаж і мати необхідну кваліфікацію для виконання робіт;
3) всі працівники повинні використовувати засоби індивідуального захисту (ЗІЗ), відповідні до характеру виконуваних робіт;

б) підготовка до ремонту;

1) перед початком ремонтних робіт необхідно провести детальний огляд обладнання для визначення його технічного стану та виявлення можливих небезпек;
2) знеструмити обладнання, переконатися у відсутності залишкової напруги та заблокувати його від випадкового вмикання. Використовувати замки та попереджувальні таблички “Не вмикати! Працюють люди”;
3) забезпечити належне освітлення робочої зони, усунути зайві предмети, які можуть заважати роботі або створювати небезпеку;
4) підготувати необхідні інструменти та обладнання, переконатися в їх справності та безпечності використання;

в) виконання ремонтних робіт;

1) під час виконання ремонтних робіт необхідно дотримуватися послідовності операцій, викладених у технічній документації та інструкціях;
2) усі рухомі частини обладнання повинні бути закріплені або заблоковані, щоб уникнути їх неконтрольованого переміщення;
3) використовувати інструменти та обладнання тільки за призначенням, дотримуючись правил їх експлуатації;
4) з особливою обережністю працювати з гострими, ріжучими інструментами, високотемпературними або високовольтними елементами обладнання;
5) забороняється виконувати ремонтні роботи самостійно, без нагляду або допомоги інших працівників, якщо це передбачено технічними умовами безпеки;

г) завершення ремонтних робіт;

1) після завершення ремонтних робіт необхідно провести перевірку всіх вузлів та механізмів на правильність складання та відсутність дефектів;
2) видалити з робочої зони всі інструменти, матеріали та інші предмети, що залишилися після ремонту;
3) після проведення ремонтних робіт обладнання повинно бути введено в експлуатацію згідно з вимогами технічної документації та з дотриманням всіх правил безпеки;
4) переконатися у відсутності сторонніх осіб у робочій зоні перед вмиканням обладнання. Попередньо повідомити всіх задіяних осіб про закінчення ремонтних робіт і введення обладнання в експлуатацію;

ґ) надзвичайні ситуаці;

1) у випадку виникнення надзвичайної ситуації (пожежа, ураження електричним струмом, механічні травми) негайно

припинити роботу та повідомити відповідальних осіб;

2) надати першу медичну допомогу постраждалим, за необхідності викликати медичну службу;
3) дотримуватися плану евакуації та інструкцій щодо дій у надзвичайних ситуаціях, які затверджені на підприємстві;

д) контроль та перевірка;

1) регулярно проводити інструктажі з охорони праці та техніки безпеки для всіх працівників, залучених до ремонтних робіт;
2) вести журнал обліку проведених ремонтних робіт та інструктажів з техніки безпеки;
3) здійснювати періодичний контроль стану обладнання та робочих місць на предмет відповідності вимогам безпеки.

Дотримання цих правил і заходів допоможе зменшити ризик виникнення нещасних випадків та забезпечити безпечні умови праці при ремонті технологічного обладнання.

ВИСНОВКИ

Технології розпізнавання облич, емоцій і рухів застосовуються в різних сферах, таких як безпека, медицина, розваги та маркетинг.

Ці програми можуть підвищити ефективність і точність аналітики та створити нові можливості для покращення послуг, взаємодії з користувачами та забезпечення безпеки.

Водночас розробка та впровадження такої технології вимагає ретельного планування та використання сучасних інструментів для забезпечення продуктивності, масштабованості та безпеки.

Аналізуючи різні можливі рішення, тепер можна визначити найкращі інструменти для розпізнавання облич, емоцій і рухів, а також платформу для використання контейнерів.

Розробка програмного забезпечення передбачала використання конфігурації контейнера Docker і стека MERN для розробки системи, яка забезпечує високу продуктивність і гнучкість.

Налаштування продуктивності, налаштування GitHub Actions і тестування в реальному часі показали, що система може виконувати завдання з високою ефективністю та надійністю.

Впровадження систем розпізнавання облич, емоцій і рухів у хмарному середовищі AWS за допомогою контейнерів Docker демонструє великий потенціал для вдосконалення різних сфер діяльності.

Він забезпечує ефективність, гнучкість, безпеку та здатність легко адаптуватися до мінливих потреб бізнесу та користувачів.

Подальший розвиток цих технологій забезпечить ще вищий рівень інтерактивності та персоналізації, сприяючи подальшому розвитку різних галузей.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Вивчення бахових команд системи керування додатками Docker, 2023. URL: https://freehost.com.ua (дата звернення: 15.06.2024).
2. Fowler, Martin, and Jim Highsmith. "The agile manifesto." Software development 9.8. 2001, 28-35.
3. Awad, M. A. A comparison between agile and traditional software Development methodologies. University of Western Australia, 2005, 1-69.
4. Docker Compose overview, 2024. URL: https://docs.docker.com/compose/ (дата звернення: 15.06.2024).
5. Define and run multi-container applications with Docker, 2024. URL: https://github.com/docker/compose (дата звернення: 15.06.2024).
6. Start Building on AWS Today, 2024. URL: https://aws.amazon.com (дата звернення: 15.06.2024).
7. GitHub: збереження коду, 2024. URL: https://github.com (дата звернення: 18.06.2024).
8. Leszko, Rafal. Continuous Delivery with Docker and Jenkins: Create secure applications by building complete CI/CD pipelines. Packt Publishing Ltd, 2022.
9. MongoDB: база даних, 2024. URL: https://account.mongodb.com/ (дата звернення: 18.06.2024).
10. Chen, Boyuan. Improving the software logging practices in DevOps. In: 2019 IEEE/ACM 41st International Conference on Software Engineering: Companion
11. Proceedings (ICSE-Companion). IEEE, 2019. p. 194-197.
12. Stripe: платіжна система, 2024. URL: https://stripe.com/ (дата звернення: 18.06.2024).
13. Heroku is a platform as a service, 2024. URL: https://dashboard.heroku.com/ (дата звернення: 18.06.2024).
14. PostMan: надсилання запитів, 2024. URL: https://identity.getpostman.com (дата звернення: 18.06.2024).
15. Miell, Ian; SAYERS, Aidan. Docker in practice. Simon and Schuster,

2019.

16. Firebase: збереження фото, 2024. URL: https://firebase.google.com (дата звернення: 18.06.2024).

17. Гурик О. Я., Король О. І., Сенчишин В. С. Методичні вказівки до лабораторної роботи №2 з дисципліни :”Основи охорони праці” ”Дослідження метеорологічних умов у виробничих приміщеннях” . Тернопіль, 2016. 35 с.

18. Yatsyshyn V., Pastukh O., Palamar A., Zharovskyi R. Technology of relational database management systems performance evaluation during computer systems design. Scientific Journal of TNTU.Tern.: TNTU. 2023. Vol 109. No 1. P. 54–65.

19. Yatsyshyn V., Pastukh O., Zharovskyi R., Shabliy N. Software tool for productivity metrics measure of relational database management system. Mathematical Modeling. No 1 (48). 2023. P. 7-17.

20. Yatsyshyn V., Kharchenko O., Lutskiv A. Maturity Requirements Model for Software Requirements with the Implementation of ISO/IEC 25010 Recommendations. International Journal "Information Models and Analyses" Volume 9, Number 2, 2020 p. 126-143.

21. Yatcyshyn V. Kharchenko A., Bodnarchuk I., Galay I. An Optimal Trade-off Solution of the Software Architecture Choice Problem/.// Journal of Information and Computing Science. 2016. Vol 11. No 4. P. 281-290.

22. Harchenko A., Bodnarchuk I., Halay I, Yatcyshyn V. The method for comparative evaluation of software architecture with accounting of trade-offs/ // American Journal of Information Systems. Vol. 2. No 2. 2014. P. 20-25

23. Pastukh O., Yatsyshyn V. Brain-computer interaction neurointerface based on artificial intelligence and its parallel programming using high-performance calculation on cluster mobile devices. Scientific Journal of TNTU. Tern.: TNTU. 2023. Vol 112. No 4. P. 26–31.

24. Pastukh O., Yatsyshyn V. Development of software for neuromarketing based on artificial intelligence and data science using high-performance computing and parallel programming technologies. Scientific Journal of TNTU. Vol 113. No 1. 2024.

pp. 143–149.

25. Осухівська Г.М., Тиш Є.В., Луцик Н.С., Паламар А.М. Методичні вказівки до виконання кваліфікаційних робіт здобувачів першого (бакалаврського) рівня вищої освіти спеціальності 123 «Комп'ютерна інженерія» усіх форм навчання. Тернопіль, ТНТУ. 2022. 28 с.

Додаток А

Технічне завдання

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп’ютерно-інформаційних систем і програмної інженерії

Кафедра комп’ютерних систем та мереж

“Затверджую”

Завідувач кафедри КС _____ Осухівська Г.М.

“___” _____ 2024 р.

DOCKER КОНТЕЙНЕР З СИСТЕМОЮ РОЗПІЗНАВАННЯ ОБЛИЧЧЯ, ЕМОЦІЙ ТА РУХІВ НА AWS CLOUD

ТЕХНІЧНЕ ЗАВДАННЯ на __ листках

На здобуття освітньо-кваліфікаційного рівня *бакалавр*

Напрям *123 Комп’ютерна інженерія*

Спеціальність *123 Комп’ютерна інженерія*

«УЗГОДЖЕНО»

Керівник кваліфікаційної роботи

_____ к.т.н., доцент

Яцишин В.В.

“____” _____ 2024 р.

«ВИКОНАВЕЦЬ»

Студент групи СІс-42

_____ Долгушин Я.В.

“____” _____ 2024 р.

Тернопіль 2024

1. Назва та підстава для виконання роботи.

1.1. Docker контейнер з системою розпізнавання обличчя, емоцій та рухів на AWS Cloud.

1.2. Підставою для виконання кваліфікаційної роботи бакалавра (КРБ) є Наказ по Університету (№ 4/7-468 від 26.04.2024 р.).

2. Виконавець.

2.1. Студент групи СІс-42 кафедри КС Тернопільського національного технічного університету ім. І. Пулюя, Долгушин Ярослав Валерійович.

3. Мета роботи.

3.1. Метою роботи є розгортання Docker контейнеру з системою розпізнавання обличчя, емоцій та рухів в хмарному середовищі.

4. Склад виробу.

4.1. До складу виробу повинні входити:

- Docker контейнер;
- система розпізнавання обличчя, емоцій та рухів;
- хмарне середовище AWS Cloud;
- інтерфейс для взаємодії;
- підключення та комунікація.

5. Технічні вимоги.

5.1. Вимоги по призначенню.

5.1.1. Функціональні вимоги:

– система повинна забезпечувати розпізнавання обличчя, емоцій та рухів у реальному часі з використанням вбудованих алгоритмів машинного навчання;

– інтеграція з хмарним середовищем AWS Cloud для розгортання та масштабування контейнеру;

– підтримка з'єднання з хмарними сервісами для зберігання та обробки даних;

– підтримка Docker для ізольованого виконання програми;

– використання веб-інтерфейсу або RESTful API для взаємодії з системою;

– здатність обробляти відеопотоки з камер або записів;

– здатність визначати присутність облич у кадрі та ідентифікувати їх емоційний стан (щастя, сум, гнів, нейтральність тощо);

– фільтрація та обробка даних з відеопотоків для виділення ключових рухів та жестів;

– підтримка автоматичного оновлення контейнеру та бібліотек.

5.1.2. Система повинна живитись напругою постійного струму 5 В через USB порт та/або 12 В.

5.2. Вимоги до умов експлуатації.

5.2.1. Умови експлуатації:

– система повинна бути сумісна з хмарною платформою AWS та підтримувати роботу в умовах доступності мережі Інтернет;

– забезпечення безперервного доступу до хмарних ресурсів з можливістю автоматичного масштабування;

– Підтримка оновлень та моніторингу через хмарні сервіси;

– Забезпечення відмовостійкості та відновлення системи після збоїв хмарної інфраструктури.

5.3. Конструктивні вимоги.

5.3.1. Конструкція контейнера:

– контейнер повинен бути сумісний з AWS EC2, Fargate або

іншим сервісом для розгортання Docker-контейнерів;

– система повинна підтримувати конфігурацію через файли YAML або Docker Compose для автоматизованого розгортання;

– Підтримка масштабованості для обробки змінних обсягів запиті.

5.3.2. Для побудови системи мають бути використані сучастні технології.

5.3.3. При побудові системи необхідно передбачити зручний та інтуїтивно зрозумілий веб-інтерфейс для налаштування та моніторингу роботи.

5.4. Вимоги до надійності.

5.4.1. Система повинна працювати без збоїв при обробці даних з кількох джерел одночасно.

5.4.2. Можливість автоматичного відновлення після збоїв.

5.5. Вимоги метрології.

5.5.1. Перевірка точності розпізнавання облич, емоцій та рухів за стандартними тестовими наборами даних.

5.5.2. Оцінка продуктивності системи при різних умовах навантаження та з різними параметрами конфігурації.

6. Економічні показники.

6.1. Вартість розробки та впровадження

6.1.1 Вартість комерційних бібліотек і програм, якщо використовуються: не більше 2500 грн;

6.1.2 Зарплатні витрати на команду робробки не більше 50000 грн;

7. Вимоги до документації.

7.1. Конструкторська документація повинна відповідати вимогам ЄСКД, ДСТУ, ISO ТА IEEE.

7.2. До складу документації повинно входити:

– архітектура системи;

- функціональні схеми;
- технічний опис;
- Алгоритми;
- Інструкції по розгортанню.

*Примітка: У комплект документації можуть вноситися зміни та доповнення в процесі розробки.

8. Стадії та етапи розробки КРБ

8.1 Стадії та етапи виконання КРБ наведенні в таблиці 1.

Таблиця 1

№ з/п	Назва етапів роботи	Термін виконання етапів роботи
1	*Розробка і затвердження технічного завдання*	
2	*Аналіз технічного завдання*	
3	*Аналіз вимог та принципів організації системи розпізнавання облич, емоцій та рухів*	
4	*Проєктування системи розпізнавання облич, емоцій та рухів*	
5	*Розробка схем і програмного забезпечення ситеми розпізнавання облич, емоцій та рухів*	
6	*Розробка інструкцій з використання системи*	
7	*Безпека життєдіяльності, основи охорони праці*	
8	*Оформлення кваліфікаційної роботи*	
9	*Попередній захист кваліфікаційної роботи*	
10	*Захист кваліфікаційної роботи*	

У дане ТЗ можуть вноситись зміни по узгодженню сторін.

Додаток Б

Лістинги програмного забезпечення

Лістиг файлу docker-compose.yaml

```
version: '3.8'
# Services
services:
  # Server service
  server:
    build:
      context: ./server
      dockerfile: Dockerfile

    container_name: serverPart
    ports:
      - "5000:5000"
    env_file: ./server/.env
    environment:
      - MONGO_Vikki=MONGO_Vikki
      - SECRET=SECRET
      - JWT_KEY=JWT_KEY
      - STRIPE_SKEY=STRIPE_SKEY
      - EMAIL=EMAIL
      - PASSWORD=PASSWORD
      - RESET_PASSWORD_KEY=RESET_PASSWORD_KEY

  # Client service
  client:
    build:
      context: ./client
      dockerfile: Dockerfile

    container_name: frontend
    ports:
      - "3000:3000"
```

```
    env_file: ./client/.env
    environment:
      - REACT_APP_SERVICE_ID=REACT_APP_SERVICE_ID
      - REACT_APP_TEMPLATE_ID=REACT_APP_TEMPLATE_ID
      - REACT_APP_PUBLIC_KEY=REACT_APP_PUBLIC_KEY
      - REACT_APP_BASE_URL=REACT_APP_BASE_URL
      - REACT_APP_KEY=REACT_APP_KEY
    depends_on:
      - server

  admin:
    build:
      context: ./admin
      dockerfile: Dockerfile

    container_name: admin
    ports:
      - "3001:3000"
    env_file: ./admin/.env
    environment:
      - REACT_APP_BASE_URL=REACT_APP_BASE_URL
      - REACT_APP_apiKey=REACT_APP_apiKey
      - REACT_APP_authDomain=REACT_APP_authDomain
      - REACT_APP_projectIdL=REACT_APP_projectId
      - REACT_APP_storageBucket=REACT_APP_storageBucket
      -
REACT_APP_messagingSenderId=REACT_APP_messagingSenderId
      - REACT_APP_appId=REACT_APP_appId
    depends_on:
      - server
```

Лістинг файлу Dockerfile

```
FROM node:18-alpine

WORKDIR /app
```

```
COPY package.json .
COPY package-lock.json .

RUN npm install

COPY . .

EXPOSE 3000

CMD ["npm", "start"]
```

Лістинг файлу deploy.yaml

```
# .github/workflows/deploy.yml
name: CI/CD Pipeline

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

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
    - name: Checkout code
      uses: actions/checkout@v2

    - name: Set up Node.js
      uses: actions/setup-node@v2
      with:
        node-version: '14'

    - name: Install dependencies
      run: npm install
```

```
      - name: Run tests
        run: npm test

      - name: Build Docker image
        run: docker build -t my-app .

      - name: Push Docker image to registry
        run: |
          echo ${{ secrets.DOCKER_PASSWORD }} | docker login -u
${{ secrets.DOCKER_USERNAME }} --password-stdin
          docker tag my-app:latest my-docker-repo/my-app:latest
          docker push my-docker-repo/my-app:latest

  deploy:
    runs-on: ubuntu-latest
    needs: build

    steps:
    - name: Deploy to AWS
      run: |
        aws ecs update-service --cluster my-cluster --service
my-service --force-new-deployment
```

Лістинг файлу Scene.tsx

```
import * as THREE from "three";
import { useEffect, useRef, useState } from "react";
import { Canvas, extend, useFrame, useThree } from "@react-
three/fiber";
import {
  useCursor,
  MeshPortalMaterial,
  CameraControls,
  Gltf,
  Text,
} from "@react-three/drei";
```

```
    import { useRoute, useLocation } from "wouter";
    import { easing, geometry } from "maath";
    import { suspend } from "suspend-react";
    import { useAppSelector } from "../../store";

    extend(geometry);

    const regular =
import("@pmndrs/assets/fonts/inter_regular.woff");
    const medium =
import("@pmndrs/assets/fonts/inter_medium.woff");

    export const SceneApp = () => (
      <Canvas
        camera={{ fov: 75, position: [0, 0, 20] }}
        eventSource={document.getElementById("root")}
        eventPrefix="client"
      >
        <color attach="background" args={["#f0f0f0"]} />
        <Frame
          id="01"
          name={`pick\nles`}
          bg="#e4cdac"
          position={[-1.15, 0, 0]}
          rotation={[0, 0.5, 0]}
        >
          <Gltf
            src="pickles_3d_version_of_hyuna_lees_illustration-
transformed.glb"
            scale={8}
            position={[0, -0.7, -2]}
          />
        </Frame>
        <Frame id="02" name="tea">
          <Gltf src="fiesta_tea-transformed.glb" position={[0, -2,
-3]} />
```

```
        </Frame>
        <Frame
          id="03"
          name="still"
          bg="#d1d1ca"
          position={[1.15, 0, 0]}
          rotation={[0, -0.5, 0]}
        >
          <Gltf
            src="still_life_based_on_heathers_artwork-
transformed.glb"
            scale={2}
            position={[0, -0.8, -4]}
          />
        </Frame>
        <Rig />
      </Canvas>
    );

    function Frame({
      id,
      name,
      author,
      bg,
      width = 1,
      height = 1.61803398875,
      children,
      ...props
    }) {
      const portal = useRef();
      const [, setLocation] = useLocation();
      const [, params] = useRoute("/item/:id");
      const [hovered, hover] = useState(false);
      useCursor(hovered);

      useFrame((state, dt) =>
```

```
          easing.damp(portal.current, "blend", params?.id === id ?
1 : 0, 0.2, dt)
        );

        return (
          <group {...props}>
            <Text
              font={suspend(medium).default}
              fontSize={0.3}
              anchorY="top"
              anchorX="left"
              lineHeight={0.8}
              position={[-0.375, 0.715, 0.01]}
              material-toneMapped={false}
            >
              {name}
            </Text>
            <Text
              font={suspend(regular).default}
              fontSize={0.1}
              anchorX="right"
              position={[0.4, -0.659, 0.01]}
              material-toneMapped={false}
            >
              /{id}
            </Text>
            <mesh
              name={id}
              onDoubleClick={(e) => (
                e.stopPropagation(), setLocation("/item/" +
e.object.name)
              )}
              onPointerOver={(e) => hover(true)}
              onPointerOut={() => hover(false)}
            >
              <roundedPlaneGeometry args={[width, height, 0.1]} />
```

```
            <MeshPortalMaterial
              ref={portal}
              events={params?.id === id}
              side={THREE.DoubleSide}
            >
              <color attach="background" args={[bg]} />
              {children}
            </MeshPortalMaterial>
          </mesh>
        </group>
      );
    }

    function Rig({
      position = new THREE.Vector3(0, 0, 2),
      focus = new THREE.Vector3(0, 0, 0),
    }) {
      const [, setLocation] = useLocation();

      const { horizontalRotation, verticalRotation, number, zoom }
= useAppSelector(
        (state) => state.global
      );

      const { controls, scene } = useThree();
      const [, params] = useRoute("/item/:id");

      // apply current screen
      useEffect(() => {
        if (number === 0) {
          setLocation("/");
        } else {
          setLocation("/item/0" + number);
        }
      }, [number]);
```

```
        // apply zoom in and zoom out
        useEffect(() => {
          position.set(0, 0, zoom / 10);
          controls?.setLookAt(...position.toArray(),
...focus.toArray(), true);
        }, [zoom]);

        // apply rotation
        useEffect(() => {
          scene.rotation.y += horizontalRotation * 3;
          scene.rotation.x += verticalRotation * -1;
        }, [horizontalRotation, verticalRotation]);

        useEffect(() => {
          const active = scene.getObjectByName(params?.id);
          if (active) {
            active.parent.localToWorld(position.set(0, 0.5, 0.25));
            active.parent.localToWorld(focus.set(0, 0, -2));
          }

          controls?.setLookAt(...position.toArray(),
...focus.toArray(), true);
        });
        return (
          <CameraControls makeDefault minPolarAngle={0}
maxPolarAngle={Math.PI / 2} />
        );
      }
```

Лістинг файлу Webcam.tsx

```
      import { useEffect, useRef } from "react";
      import { useDispatch } from "react-redux";
      import { Camera } from "@mediapipe/camera_utils";
      import { drawConnectors, drawLandmarks } from
"@mediapipe/drawing_utils";
      import { Hands, HAND_CONNECTIONS, Results } from
```

```
"@mediapipe/hands";
     import { AppDispatch } from "../../../store";
     import {
       applyZoom,
       setHorizontalRotation,
       setNumber,
       setVerticalRotation,
     } from "../../../store/global";
     import { getGlobalDistance } from "../../../utils";

     const maxVideoWidth = 920 / 2;
     const maxVideoHeight = 540 / 2;

     function RecognitionHook() {
       const dispatch = useDispatch<AppDispatch>();

       const videoElement = useRef<any>(null);
       const hands = useRef<any>(null);
       const camera = useRef<any>(null);
       const canvasEl = useRef<any>(null);

       let lastPalmX: number | null = null;
       let lastPalmY: number | null = null;

       let lastNumber = 0;
       let handNumberFrames = 0;

       function applyHand(screen: number) {
         handNumberFrames = lastNumber === screen ?
handNumberFrames + 1 : 0;
         if (handNumberFrames > 2) dispatch(setNumber(screen));
         lastNumber = screen;
       }

       // get hand informations
       async function processHands(results: Results) {
```

```
          if (canvasEl.current) {
            const glms = results.multiHandWorldLandmarks[0];
            const lms = results.multiHandLandmarks[0];

            const ctx = await canvasEl.current.getContext("2d");
            ctx.save();
            ctx.clearRect(0, 0, canvasEl.current.width,
canvasEl.current.height);
            ctx.drawImage(results.image, 0, 0, maxVideoWidth,
maxVideoHeight);

            if (lms && glms) {
              const openFingers = [
                getGlobalDistance(glms[8], glms[5]) > 0.055 ? 1 : 0,
                getGlobalDistance(glms[12], glms[9]) > 0.07 ? 1 : 0,
                getGlobalDistance(glms[16], glms[13]) > 0.065 ? 1 :
0,
                getGlobalDistance(glms[20], glms[17]) > 0.055 ? 1 :
0,
                getGlobalDistance(glms[4], glms[5]) > 0.05 ? 1 : 0,
              ];

              // Apply zoom in and zoom out
              if (
                getGlobalDistance(glms[8], glms[5]) > 0.045 &&
                !openFingers[1] &&
                !openFingers[2] &&
                !openFingers[3] &&
                getGlobalDistance(glms[4], glms[5]) > 0.04
              ) {
                const distance = getGlobalDistance(glms[4],
glms[8]);
                const percent = (distance / 0.01) * 10;
                dispatch(applyZoom(percent));
              }
```

```
// Verify finger number
if (
  !openFingers[0] &&
  !openFingers[1] &&
  !openFingers[2] &&
  !openFingers[3] &&
  openFingers[4]
)
  applyHand(0);

if (
  openFingers[0] &&
  !openFingers[1] &&
  !openFingers[2] &&
  !openFingers[3] &&
  !openFingers[4]
)
  applyHand(1);

if (
  openFingers[0] &&
  openFingers[1] &&
  !openFingers[2] &&
  !openFingers[3] &&
  !openFingers[4]
)
  applyHand(2);

if (
  openFingers[0] &&
  openFingers[1] &&
  openFingers[2] &&
  !openFingers[3] &&
  !openFingers[4]
)
  applyHand(3);
```

```
            // verify palm rotation
            if (lastPalmX === null || lastPalmY === null) {
              lastPalmX = lms[0].x;
              lastPalmY = lms[0].y;
            } else {
              if (
                openFingers[0] &&
                openFingers[1] &&
                openFingers[2] &&
                openFingers[3]
              ) {
                // set horizontal rotation
                if (lms[0].x - lastPalmX < 0.1 && lms[0].x -
lastPalmX > -0.1) {
                  dispatch(setHorizontalRotation(lastPalmX -
lms[0].x));
                }

                // set vertical rotation
                if (lms[0].y - lastPalmY < 0.1 && lms[0].y -
lastPalmY > -0.1) {
                  dispatch(setVerticalRotation(lastPalmY -
lms[0].y));
                }

                lastPalmX = lms[0].x;
                lastPalmY = lms[0].y;
              }
            }

            // draw hand landmarks and lines
            drawConnectors(ctx, lms, HAND_CONNECTIONS, {
              color: "#00ffff",
              lineWidth: 0.5,
            });
```

```
            drawLandmarks(ctx, lms, {
              color: "#ffff29",
              lineWidth: 0.1,
            });
          }

          ctx.restore();
        }
      }

      // load hand medidapipe model
      const loadHands = () => {
        if (!hands.current) {
          hands.current = new Hands({
            locateFile: (file: String) => {
              return
`https://cdn.jsdelivr.net/npm/@mediapipe/hands/${file}`;
            },
          });
          hands.current.setOptions({
            maxNumHands: 2,
            modelComplexity: 1,
            minDetectionConfidence: 0.5,
            minTrackingConfidence: 0.5,
          });
          hands.current.onResults(processHands);
        }
      };

      useEffect(() => {
        // start camera and capture frames
        async function initCamara() {
          camera.current = new Camera(videoElement.current, {
            onFrame: async () => {
              await hands.current.send({ image:
```

```
videoElement.current });
              },
              width: maxVideoWidth,
              height: maxVideoHeight,
            });
            camera.current.start();
          }

          initCamara();
          loadHands();
        });

        return { maxVideoHeight, maxVideoWidth, canvasEl,
videoElement };
      }
      export default RecognitionHook;
```