Вінницький національний технічний університет
(повне найменування вищого навчального закладу)

Факультет інформаційних технологій та комп'ютерної інженерії
(повне найменування інституту, назва факультету (відділення))

Кафедра програмного забезпечення
(повна назва кафедри (предметної, циклової комісії))

Бакалаврська дипломна робота на тему:

«РОЗРОБКА WEB–СЕРВІСУ ДЛЯ ІДЕНТИФІКАЦІЇ ОБ'ЄКТІВ»

Виконав: студент 4 курсу групи 2ПІ–18б
спеціальності 121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Грабарчук А. В.
(прізвище та ініціали)

Керівник: к. т. н., доц. Коваль Л. Г.
(прізвище та ініціали)
«____» _____ 2022 р.

Рецензент: д. т. н., проф. каф. КН Іванчук Я. В.
(прізвище та ініціали)
«____» _____ 2022 р.

Допущено до захисту

Зав. кафедри ПЗ Романюк О. Н. _____

«____» _____ 2022 р.

Вінниця ВНТУ– 2022 рік

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший бакалаврський
Галузь знань 12 – Інформаційні технології
Спеціальність 121 – Інженерія програмного забезпечення
Освітньо-професійна програма – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О. Н.
25 березня 2022 року

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ
Грабарчуку Антону Володимировичу

1. Тема роботи: Розробка Web–сервісу для ідентифікації об'єктів, керівник роботи: Коваль Леонід Григорович, к.т.н., доцент кафедри ПЗ, затверджені наказом вищого навчального закладу від 24 березня 2022 р. № 66.

2. Термін подання студентом роботи 13 червня 2022 р.

3. Вихідні дані до роботи: засіб розпізнавання – Google Cloud Vision API; мова програмування – JavaScript; середовище розробки – WebStorm; операційна система – кросплатформний додаток.

4. Зміст текстової частини: вступ; обґрунтування вибору методу розробки та постановки задачі дослідження; розробка структури та алгоритмів Web–сервісу для ідентифікації об'єктів; розробка коду Web–сервісу; тестування Web–сервісу; висновки; список використаних джерел; додатки.

5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень): мета і задачі роботи; компоненти web- сервісу; алгоритм роботи серверної частини; алгоритм роботи мобільної частини; дизайн мобільного додатку; висновки.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Коваль Л. Г., к.т.н., доцент кафедри ПЗ		

7. Дата видачі завдання 25 березня 2022 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської дипломної Роботи	Строк виконання етапів роботи	Примітка
1	Обґрунтування методу розробки та постановки задачі дослідження	26.03.22 – 01.04.22	Вик.
2	Розробка структури та алгоритмів Web–сервісу для ідентифікації об’єктів	02.04.22 – 07.04.22	Вик.
3	Розробка інтерфейсу	08.04.22 – 11.04.22	Вик.
4	Розробка коду Web–сервісу	12.04.22 – 10.05.22	Вик.
5	Тестування Web–сервісу для ідентифікації об’єктів	11.05.22 – 28.05.22	Вик.
6	Оформлення матеріалів до захисту БДР	01.06.22– 10.06.22	Вик.

Студент _____ **Грабарчук А. В.**

(підпис) (прізвище та ініціали)

Керівник роботи _____ **Коваль Л. Г.**

(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Бакалаврська дипломна робота складається з 83 сторінок формату А4, на яких є 20 рисунків, та 2 таблиць. Список використаних джерел містить 12 найменувань. Метою роботи є покращення процесу взаємодії й ефективності при розпізнаванні об’єктів. Подальшого розвитку отримав метод ідентифікації об’єктів на зображенні, у якому, на відміну від існуючих, використано клієнт-серверну архітектуру, що зменшує технічні вимоги до робочої станції, оскільки складні обчислення виконуються на сервері.

У розділі Обґрунтування методу розробки та постановки задачі дослідження аналіз відомих додатків для ідентифікації об’єктів на зображеннях показав, що суттєвим недоліком цих додатків є відсутність можливості додавання власних об’єктів, тому розробка даного додатку є доцільною. Сформульовано основні задачі розробки, зокрема, важливою задачею є створення зручного інтерфейсу користувача та розширення функціональних можливостей додатку.

У розділі Розробка структури та алгоритмів програмного продукту Розроблено модель та загальну структуру WEB-сервісу для ідентифікації об’єктів на зображеннях. Подальшого розвитку отримав метод ідентифікації об’єктів на зображенні, у якому, на відміну від існуючих, використано клієнт-серверну архітектуру, що зменшує технічні вимоги до робочої станції, оскільки складні обчислення виконуються на сервері. Розроблено алгоритми роботи серверної частини додатку, доведена їх необхідність і важливість. Розроблено алгоритми роботи мобільного додатку, доведена їх необхідність, продумана взаємодія з серверною частиною.

У розділі розробка програмного продукту Обрано тип інтерфейсу та розроблено графічний інтерфейс мобільного додатку для ідентифікації об’єктів. Для реалізації додатку ідентифікації об’єктів обрано мову програмування JavaScript, платформу Node.js для реалізації серверної частини додатку, фреймворк ReactNative для розробки мобільного додатку та Google Cloud

Vision API для розпізнавання об’єктів. Мовою програмування JavaScript з використанням платформи Node.js розроблено серверну частину додатку для ідентифікації об’єктів.

У розділі тестування програми було проведено тестування WEB-сервісу для ідентифікації об’єктів з використанням засобу тестування Postman показало повну його працездатність та відповідність завданню на роботу. Розроблену керівництво програміста, яке забезпечує експлуатацію та подальший розвиток додатку на умовах GNU General Public License.

Ключові слова: Google bucket Vision Api, JavaScript, Node.js, Machine learning.

ANNOTATION

The bachelor's thesis consists of 83 A4 pages with 20 figures and 2 tables. The list of used sources contains 12 names. The aim of the work is to improve the process of interaction and efficiency in object recognition. Further developed is the method of identifying objects in the image, which, unlike existing ones, uses a client-server architecture, which reduces the technical requirements for the workstation, as complex calculations are performed on the server.

In the section Substantiation of the method of development and formulation of the research problem, the analysis of known applications for object identification in images showed that a significant disadvantage of these applications is the inability to add their own objects, so the development of this application is appropriate. The main development tasks are formulated, in particular, an important task is to create a user-friendly interface and expand the functionality of the application.

In the section Development of the structure and algorithms of the software product, the model and general structure of the WEB-service for identification of objects in images have been developed. Further developed is the method of identifying objects in the image, which, unlike existing ones, uses a client-server architecture, which reduces the technical requirements for the workstation, as complex calculations are performed on the server. Algorithms for the server part of the application have been developed, their necessity and importance have been proved. Algorithms of work of the mobile application are developed, their necessity is proved, interaction with a server part is thought over.

In the software development section, the interface type is selected and the graphical interface of the mobile application for object identification is developed. JavaScript implementation language, Node.js platform for server application implementation, ReactNative framework for mobile application development and Google Cloud Vision API for object recognition were chosen to implement the object

identification application. The server-side JavaScript application language Node.js has developed an server-side application for object identification.

In the testing section of the program, a test of the WEB-service for the identification of objects using the Postman testing tool was performed, which showed its full operability and compliance with the task at work. Developed a programmer's guide that ensures the operation and further development of the application under the terms of the GNU General Public License.

Keywords: Google bucket Vision Api, JavaScript, Node.js, Machine learning.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. Ще в минулому столітті, з початком розвитку комп'ютерних технологій, людство уявляло, як одного дня штучний інтелект зрівняється, або навіть перевершить людський. Утім, до 90-х років 20 століття реальних зрушень у цій темі не було, а штучним інтелектом називали звичайний блок IF в комп'ютерному коді.

Протягом останніх десятиліть сфера штучного інтелекту активно розвивається і знаходить все нове й нове застосування у різних сферах. Цей напрямок, небезпідставно, охрестили як професію майбутнього. Протягом десятиліття близько півмільярда людей втратять або змінять роботу через автоматизацію виробництва[1]. Вплив і вигоду від цієї галузі переоцінити неможливо, цей процес вже порівняли із винайденням конвеєра і переходом на масове виробництво на початку XX століття. І хоча багато людей налякані втратою роботи, так само як це було коли запроваджували масове виробництво, але спеціалісти впевнені, що люди не залишаться без роботи, просто їх робота зміниться. Але чому це насправді вигідно? Економія, штучний інтелект здатен надзвичайно швидко вчитися і розвиватися, керувати різного роду пристроями і машинами, при цьому він не просить зарплату, а отже може значно скоротити витрати на виробництво, датчики значно точніші за людське око, а отже є можливість зменшити кількість бракованих виробів, машини швидші за людей, відповідно збільшується продуктивність.

Інструментом для створення систем з штучним інтелектом є нейронні мережі. Нейронна мережа - це математична модель, а також її програмне чи апаратне втілення, побудована за принципом організації й функціонування біологічних нейронних мереж - мереж нервових клітин біологічного живого організму.

Одним із поширених способів застосування є визначення зображень, а саме, що, або хто, на них зображений. Наприклад сервіси котрі дозволяють знайти профіль людини в соціальних мережах по фото. Або ж визначити, що

саме за товар бачить людина і де його можна придбати, що значно підвищить продажі магазину.

Тому тема роботи «Розробка Web–сервісу для ідентифікації об’єктів» є вкрай актуальною.

Зв’язок роботи з науковими програмами, планами, темами. Робота виконувалася відповідно до плану виконання наукових досліджень на кафедрі програмного забезпечення

Мета та завдання дослідження. Метою роботи є покращення процесу взаємодії й ефективності при розпізнаванні об’єктів.

Основними задачами дослідження є:

- провести аналіз наявних методів і засобів розпізнавання об’єктів;
- розробка структури та алгоритмів роботи додатку для розпізнавання об’єктів;
- розробка інтерфейсу та коду програмного продукту;
- тестування розробленого програмного додатку.

Об’єкт дослідження – процес автоматичної ідентифікації об’єктів.

Предмет дослідження – методи та програмні засоби ідентифікації об’єктів на зображеннях.

Методи дослідження. У процесі досліджень використовувались: теорія алгоритмів, теорія розпізнавання образів, теорія програмування, методи теорії обробки сигналів для розробки моделей та методів ідентифікації об’єктів; комп’ютерне моделювання для аналізу та перевірки отриманих теоретичних положень.

Новизна отриманих результатів.

Подальшого розвитку отримав метод ідентифікації об’єктів на зображенні, у якому, на відміну від існуючих, використано клієнт-серверну архітектуру, що зменшує технічні вимоги до робочої станції, оскільки складні обчислення виконуються на сервері.

Практична цінність отриманих результатів.

Практична цінність одержаних результатів полягає в тому, що на основі отриманих в бакалаврській дипломній роботі теоретичних положень запропоновано алгоритми та розроблено з використанням мови програмування JavaScript, платформи Node.js та сервісу Google Vision API програмні засоби ідентифікації об’єктів на зображеннях.

Особистий внесок здобувача. Всі наукові результати, викладені у бакалаврській дипломній роботі, отримані автором особисто. У працях, опублікованих у співавторстві, автору належать такі результати: структура програмного забезпечення для ідентифікації об’єктів з використанням Google Cloud Vision API [2].

Апробація матеріалів бакалаврської дипломної роботи. . Основні положення бакалаврської дипломної роботи доповідалися та обговорювалися на науково-технічній конференції підрозділів Вінницького національного технічного університету (НТКП ВНТУ) (Вінниця, 2022 р.).

Публікації. Основні результати досліджень опубліковано в одній статті у матеріалах конференції підрозділів Вінницького національного технічного університету (НТКП ВНТУ) (Вінниця, 2022 р.) [3].

1 ОБҐРУНТУВАННЯ ВИБОРУ МЕТОДУ РОЗРОБКИ ТА ПОСТАНОВКИ ЗАДАЧІ ДОСЛІДЖЕННЯ

1.1 Аналіз стану ідентифікації об’єктів

Автоматичне розпізнавання об’єктів є важливою сферою використання штучного інтелекту. І ще важливіше для майбутнього розвитку людства, адже будь-який робот про використання яких ми мріємо повинен буде визначати, що знаходиться навколо нього. Наприклад автопілоти автомобілів зі штучним інтелектом вже використовують цю технологію, щоб чітко розрізняти навколишні об’єкти, особливо знаки дорожнього руху, для того, щоб не порушувати правила і не створювати аварійних ситуацій на дорозі. Але реальна сфера застосування, звичайно, набагато ширша, наприклад в ботаніці, добре навчений штучний інтелект може точно визначати рослини навіть за відео з коптера[4], що дозволить зекономити на підготовці коштовних експедицій і дослідити більшу площу. Ще одна сфера застосування для таких технологій, це лісові пожежі, відносно дешеві квадрокоптери можуть патрулювати всю територію і навчений штучний інтелект визначатиме місце пожеж на зародковому етапі. В торговій сфері ця технологія теж має велике майбутнє, де б ви не побачили цікаву річ, достатньо буде навести камеру на неї й нейронна мережа сама визначить, що це, і де можливо придбати. Нейронні мережі мають цікаву особливість, вони працюють так само як людський мозок, тобто чим більше виконують певне завдання, тим краще у них це виходить. Відповідно чим далі ми будемо розвивати їх, тим краще вони виконуватимуть свою роль[5]. На щастя цей процес(навчання нейронних мереж), також активно розвивається, покращується й автоматизується, що робить використання штучного інтелекту простішим, доступнішим і рентабельнішим, а також відкриває нові можливості, котрі раніше не були доступні через їх надмірну складність або дорожнечу. Фактично всі світові лідери думок стверджують, що

штучний інтелект має величезні перспективи. Тож можна з впевненістю сказати, що за штучним інтелектом майбутнє.

1.2 Порівняльний аналіз аналогів

Хоч сфера розпізнавання об’єктів є доволі новою, додатки доступні користувачу вже починають з’являтися. Розглянемо найбільш поширені з них.

Google об’єктив - це найпростіший і найпоширеніший спосіб шукати щонебудь за допомогою фото. Користуватися софтом можна не тільки на Android-пристроях (де він вбудований спочатку), але і на гаджетах з iOS. Для цього завантажте в AppStore додаток Google об’єктив, увійдіть в нього і натисніть на значок об’єктива в рядку пошуку(Рис. 1.1). Якісний додаток з хорошим інтерфейсом, але має слабку швидкодію, і не має можливості додавати свій магазин з товарами.

Рисунок 1.1 – Інтерфейс мобільного додатку Google об’єктив

Identify Anything Search By image - ще один сервіс котрий дозволяє ідентифікувати об’єкти за фото. Працює так само на основі нейронних мереж. Але має не дуже зручний інтерфейс, слабку швидкодію і погану точність, можна спостерігати, вебкамеру система ідентифікувала як сонячні окуляри(Рис. 1.2). Також в ньому забагато реклами на всіх етапах взаємодії, що заважає комфортній взаємодії з додатком. Відсутні посилання на місця де можна знайти цей товар, також система не визначає його марку і назву, а лише вид. Відповідно користувач не може отримати інформацію де можливо придбати необхідний товар.

Рисунок 1.2 – Інтерфейс мобільного додатку Identify Anything Search By image

PlantNet - дозволяє ідентифікувати вид рослини. Але підходить лише для ідентифікації рослин[6], відповідно не може бути використано для інших цілей. Має хорошу точність і швидкодію, також варто зазначити непоганий

інтерфейс(Рис. 1.3). Реклама присутня, але не надто заважає роботі з системою.

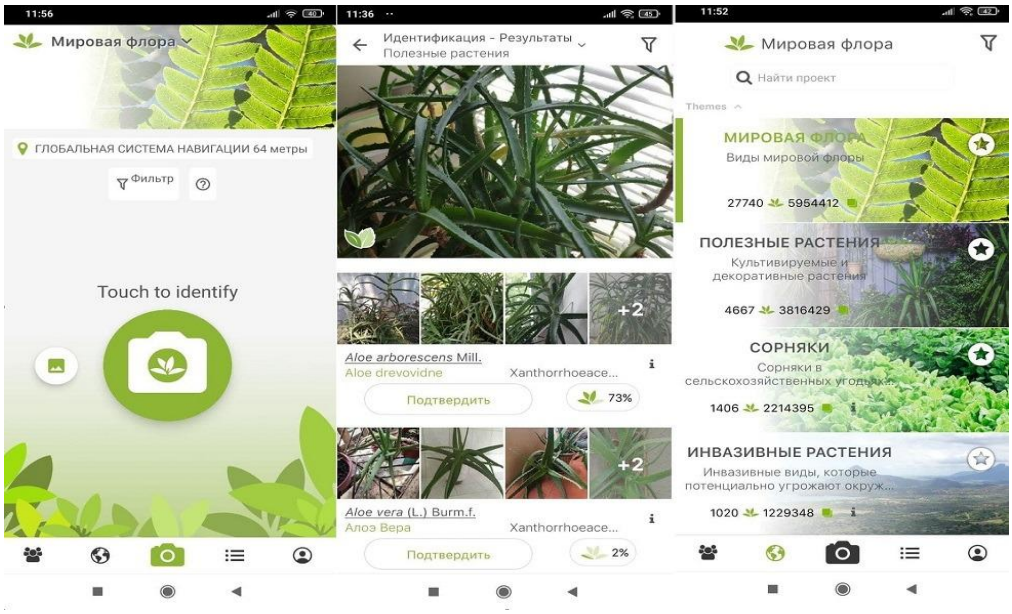

Рисунок 1.3 – Інтерфейс мобільного додатку PlantNet

Проаналізувавши аналоги визначено їх функціонал, переваги та недоліки, ці фактори були враховані при створенні додатку «Goods scan» (табл. 1.1).

Таблиця 1.1 – Порівняльні характеристики програмних продуктів

Критерій	Google об’єктив	Identify Anything Search By image	PlantNet	Goods scan
Універсальність	+	+	-	+
Точність	+	-	+	+
Можливість додавати свої компанії	-	-	-	+
Зручність	+	-	+	+
Швидкодія	+	-	+	+
Результат	4	1	3	5

Порівняння характеристик продуктів показало, що розробка додатка є доцільною. Результатом розробки буде продукт позбавлений недоліків його аналогів. Розроблений програмний додаток буде точним, універсальним, матиме можливість додавати свої компанії, з хорошим дизайном і, найголовніше, точним. Що матиме позитивні наслідки для розвитку індустрії, оскільки нове рішення краще виконуватиме необхідні функції, котрі очікуються від додатків подібного типу.

1.3 Постановка задачі розробки

Перш ніж перейти до розробки необхідно точно поставити задачу, оскільки неможливо розробляти продукт не розуміючи і не маючи перед очима, що саме ми повинні розробити.

Після детального аналізу і порівняння з аналогами були визначенні завдання, котрі необхідно виконати для розробки програмного продукту:

- розробити гарний і зручний дизайн;
- розробити можливість додавати продукти, по яких в майбутньому буде проводитись пошук;
- розробити можливість додавати компанії, котрі пропонують продукт;
- додаток повинен бути універсальним і не обмежуватись одним продуктом;
- провести тестування і перевірити надійність розробленого сервісу.

1.4 Висновки

1. Аналіз відомих додатків для ідентифікації об’єктів на зображеннях показав, що суттєвим недоліком цих додатків є відсутність можливості додавання власних об’єктів, тому розробка даного додатку є доцільною.

2. Сформульовано основні задачі розробки, зокрема, важливою задачею є створення зручного інтерфейсу користувача та розширення функціональних можливостей додатку.

2 РОЗРОБКА СТРУКТУРИ ТА АЛГОРИТМІВ WEB–СЕРВІСУ ДЛЯ ІДЕНТИФІКАЦІЇ ОБ'ЄКТІВ

2.1 Аналіз інформаційного забезпечення системи

Інформаційне забезпечення – це сукупність форм документів, нормативної бази і реалізованих рішень щодо обсягу, розміщення і форм організації інформації, яка циркулює в системі автоматизованого оброблення економічної інформації чи в інформаційній системі.

Розроблення інформаційного забезпечення – це одна із найважливіших складових розроблення інформаційної системи, вона має забезпечити:

- єдність та зберігання інформації, необхідної для розв'язання задач;
- єдність інформаційних масивів для всіх задач інформаційних систем;
- однократність уведення інформації та її багатоцільове використання;
- різні методи доступу до даних;
- низьку вартість витрат на зберігання та використання даних, а також на внесення змін.

Додаток «Goods scan», отримує інформацію від користувача один раз через мобільний додаток, і не вимагає жодних додаткових введень від користувача, вся інформація підтягується і обробляється автоматично, система зберігає її в єдиній базі даних.

2.2 Розробка загальної моделі системи

Надзвичайно важливою частиною коректної роботи будь-якої сучасної системи є взаємодія її компонентів. Оскільки всі сучасні сервіси є результатом взаємодії кількох додатків, часто використовують стороннє API, розробники ПЗ приділяють все більше уваги взаємодії окремих компонентів. Перш ніж почати розробку програмної компоненти, і архітектури окремих частин проєкту необхідно розробити діаграму взаємодії всіх компонентів(Рис. 2.1)

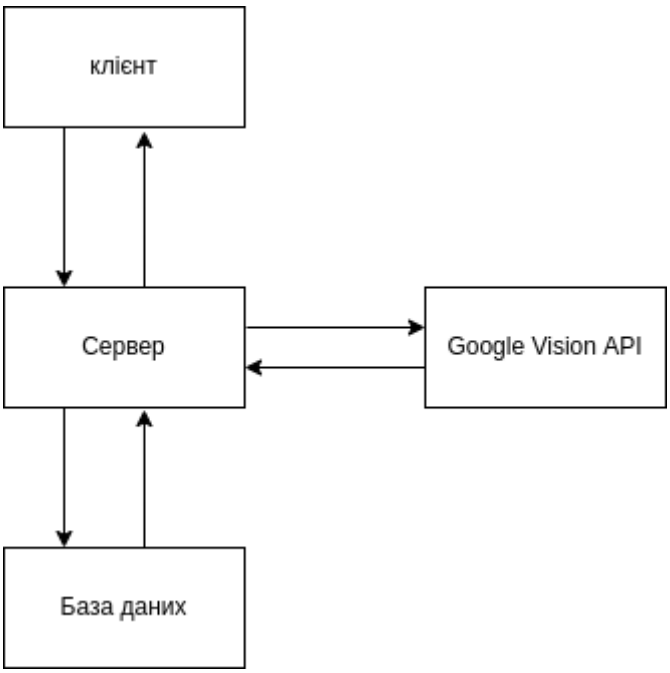

Рисунок 2.1 - взаємодія компонентів сервісу

Сервіс “Goods scan” представляє собою складну систему, котра виконує функцію розпізнавання. Як ми можемо спостерігати на схемі, сервіс має три основні компоненти:

- клієнтська частина(мобайл);
- Серверна частина;
- сервіс Google VisionAPI
- База даних

Розглянемо кожну з них.

Клієнтська частина - це мобільний додаток, котрий користувач встановлює на свій смартфон, саме користуючись нею він робить фото і відправляє запит на наш сервер.

Серверна частина - наш сервер котрий приймає і обробляє основні запити від користувача, а також відправляє запити в сервіс Google Vision API.

Google Vision API - сервіс від компанії Arphabet, котрий дає широкий функціонал для розпізнавання зображень. Дає можливість створити конкретний

продукт, надіслати його фото, і в подальшому, коли користувач, надішле фото продукту нейронна мережа його впізнає.

База даних - база даних котра зберігає всю інформацію про користувача, товари й компанії.

2.3 Розробка методу ідентифікації об’єктів та алгоритмів серверної частини

Головним аспектом роботи додатку є серверна частина, саме вона контролює взаємодію всіх частин системи, зокрема мобільної частини, веб частини (адмін), та сервісу Google Vision. Саме тому її якісне проєктування є життєво важливим для всієї системи. Для проєктування і уникнення помилок при безпосередньому написанні коду використовуються безпосередньо UML-діаграми.

UML-діаграма - це спеціалізована мова графічного опису, призначена для об 'єктного моделювання в сфері розробки різного програмного забезпечення. Ця мова має широкий профіль і є відкритим стандартом, в якому використовуються різні графічні позначення, щоб створити абстрактну модель системи. UML створювався для того, щоб забезпечити визначення, візуалізацію, документування, а також проектування всіляких програмних систем. Варто зазначити, що сама по собі UML-діаграма не являє собою мову програмування, але при цьому передбачається можливість генерації на її основі окремого коду[7].

Оскільки вони дозволяють візуалізувати алгоритм роботи, дослідити його, відправити на фідбек своїм колегам і керівникам. Загально прийняті правила створення діаграм дають змогу всім розуміти, що вони зображують, уникаючи двоякості трактування, що, своєю чергою, може призвести до великих помилок в роботі системи, котрі проявляться на пізньому етапі розробки, або навіть використанні, але будучи допущеними в фундаментальних механізмах стануть невиправними і призведуть до закриття проєкту, що в свою

чергу призведе до колосальних матеріальних втрат. Принцип роботи зображено на Рисунку 2.2.

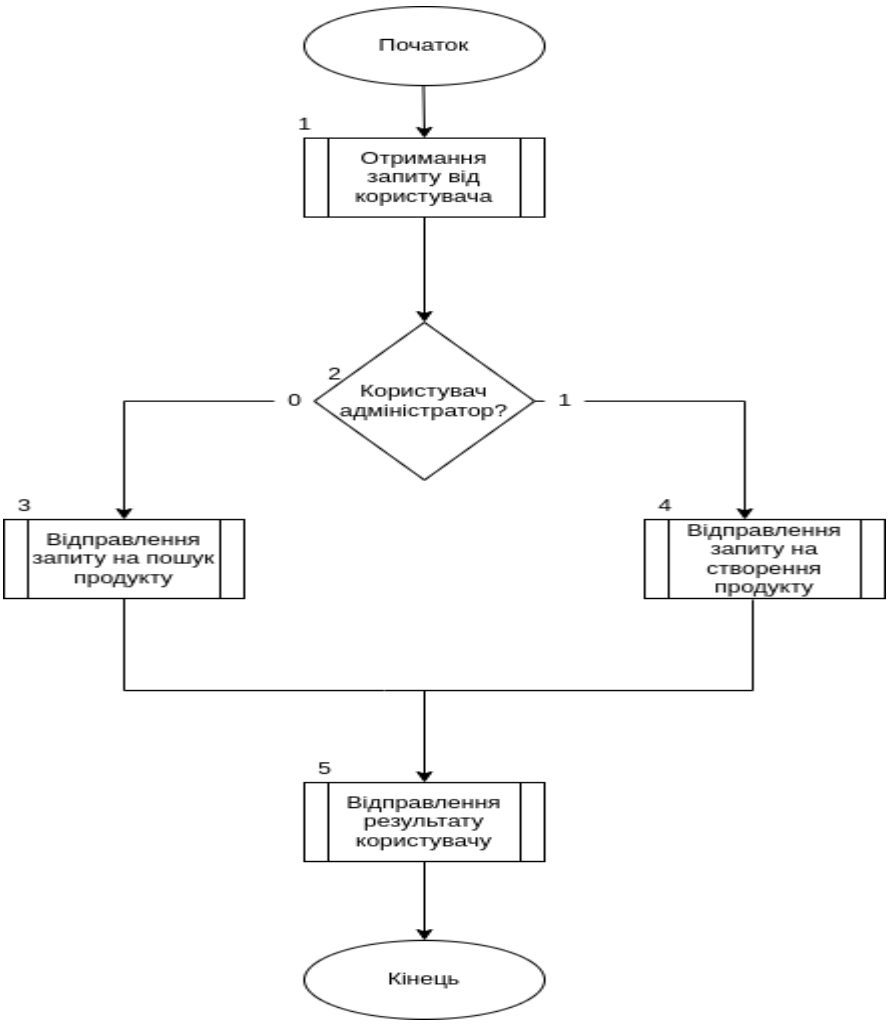

Рисунок 2.2 - Принцип роботи серверної частини

Для ідентифікації об'єкта, ми повинні створити його зразок на платформі Google Vision Api, і завантажити туди зображення зразки. Для підвищення якості й точності роботи нейронної мережі, необхідно робити зображення на білому, непрозорому фоні, в максимальній якості, вони не повинні бути розмиті, та містити шуми. Після виконання попередніх дій необхідно зачекати близько 30 хвилин, поки система проведе індексацію. Після цього наша нейронна мережа працюватиме максимально ефективним чином.

2.4 Розробка алгоритму роботи мобільного додатку

Сучасні смартфони вже давно стали невід’ємною частиною нашого життя, поява сучасних девайсів на операційних системах IOS та Android, повністю змінила ринок, завдяки великій обчислювальній здатності пристроїв, а також вдалій файловій системі, це дало змогу майже будь-якому розробнику розробляти свої додатки. А оскільки телефон люди завжди носять з собою, ринок отримав величезний попит, і почав рости шаленими темпами. Правила складання діаграм для мобільних додатків такі самі, як і для серверних, оскільки мова UML є універсальною. Для розробки діаграми були прийняті до уваги всі аспекти роботи програми, як логічні, так і графічні. Також були запровадженні всі необхідні вимоги. Внаслідок дослідження було створено діаграму роботи мобільного додатку, результат зображено на рисунку 2.3.

На діаграмі зображено всі кроки роботи алгоритму мобільного додатку:

1. Користувач робить фотографію.
2. Відправка запиту на сервер.
3. Перевірка успішності відповіді.
4. Якщо виникла помилка то вона виводиться на екран.
5. Якщо результат успішний виводимо отримані від сервера дані на екран.

2.5 Розробка бази даних

2.5.1 Обґрунтування вибору типу бази даних

База даних – сукупність даних, організованих відповідно до концепції, яка описує характеристику цих даних і взаємозв'язки між їх елементами; ця сукупність підтримує щонайменше одну з областей застосування. В загальному випадку база даних містить схеми, таблиці, подання, збережені процедури та інші об'єкти. Дані у базі організовують відповідно до моделі організації даних. Таким чином, сучасна база даних, крім самих даних, містить їх опис та може містити засоби для їх обробки.[8]

Бази даних діляться на два основних типи:

- реляційні;
- не реляційні.

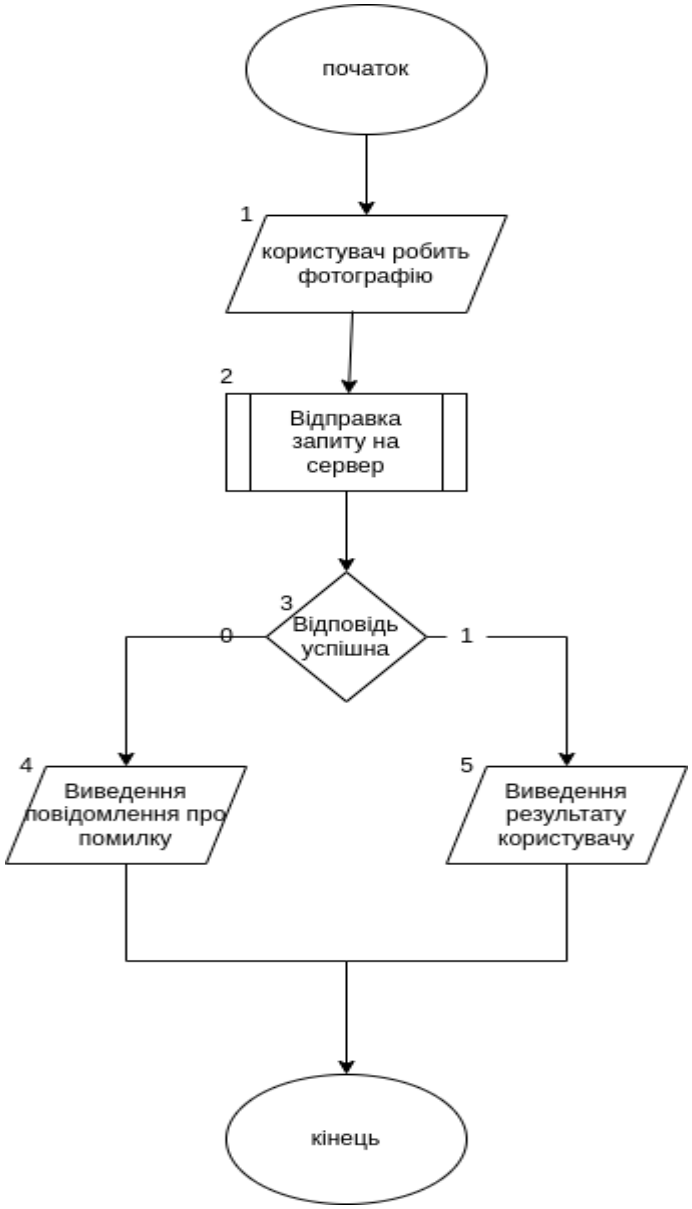

Рисунок 2.3 - Принцип роботи мобільної частини

Реляційна база даних — база даних, заснована на реляційній моделі даних. Для роботи з реляційними БД застосовують реляційні Системи Керування Базами Даних(СКБД). Інакше кажучи, реляційна база даних — це база даних, яка сприймається користувачем як набір нормалізованих відношень різного ступеня, для виконання запитів використовується синтаксис SQL.

Нереляційна база даних — це база даних, що надає спосіб керування даними, що знаходяться в нереляційній формі, тобто яка не структурована в табличному вигляді й не має табличні стосунки.

Правильно підібрана і спроєктована база даних це основа всієї системи. Оскільки, на ній базується робота серверного коду, і будь-які зміни після виходу програми в користування є надзвичайно дорогими, оскільки написання міграцій саме по собі займає багато часу, ще й доводиться переписувати бек-енд на новий лад, що збільшує час виконання завдання у 2-3 рази, а в, окремих випадках, технічно неможливо.

Кожен з цих видів має свої переваги та недоліки, зокрема нереляційні бази даних швидкі та зручні у випадках, коли необхідно діставати дані за ідентифікатором, тобто не роблячи вибірку окремих полів і/або за певною ознакою окрім унікального ідентифікатора, також великий плюс, що легко проводити їх горизонтальне масштабування, тобто зберігати інформацію на кількох серверах. Реляційні бази даних навпаки, завдяки мові SQL зручні для витягів певних стовпців і їх групування за певною ознакою.

SQL (*Structured query language* — мова структурованих запитів) — декларативна мова програмування яка використовується для взаємодії користувача з реляційними БД, застосовується для формування запитів, оновлення і керування базою даних, створення схеми бази даних та її модифікації, системи контролю за доступом до БД. SQL не є ані системою керування базами даних, ані окремим програмним продуктом. Саме в цій мові і криється основна перевага реляційних баз даних, вона дає нам можливість створювати таблиці, вказувати складні зв’язки між ними. Також ми можемо створювати складні запити, котрі дають нам змогу діставати різну інформацію з бази даних, за певною ознакою, отримувати лише певні поля, замість всіх стовпців таблиці.

2.5.2 Характеристика зв'язків бази даних

Основа реляційної бази даних - це коректні зв’язки між таблицями. За відношенням є 3 типи зв’язків:

- 1:n один до багатьох, тобто багато записів мають колонку котра вказує на унікальний ідентифікатор одного;
- 1:1 один до одного, це означає, що записи мають унікальний ідентифікатор один одного;
- n:n найскладніший в реалізації зв’язок, багато до багатьох. Для його реалізації створюється окрема таблиця котра має два стовпці, кожен з яких зберігає унікальні ідентифікатори записів.

Розроблена база даних складається з п’яти таблиць:

- images;
- categories;
- products;
- productSets.

Характеристики зв’язків бази даних наведені у таблиці 2.1.

Таблиця 2.1 - Характеристики зв’язків бази даних

Ім’я суті 1	Ім’я суті 2	Тип зв’язку	Ім’я зв’язку	Клас належності
ProductSets	products	1:n	Складається	не обов’язков./ обов’язков.
products	images	1:n	Містить	обов’язков./ не обов’язков.
products	categories	n:1	Містить	не обов’язков./ обов’язков.

Користуючись таблицею 2.1, побудуємо результуючу ER-модель предметної області, яка представлена на рисунку 2.4.

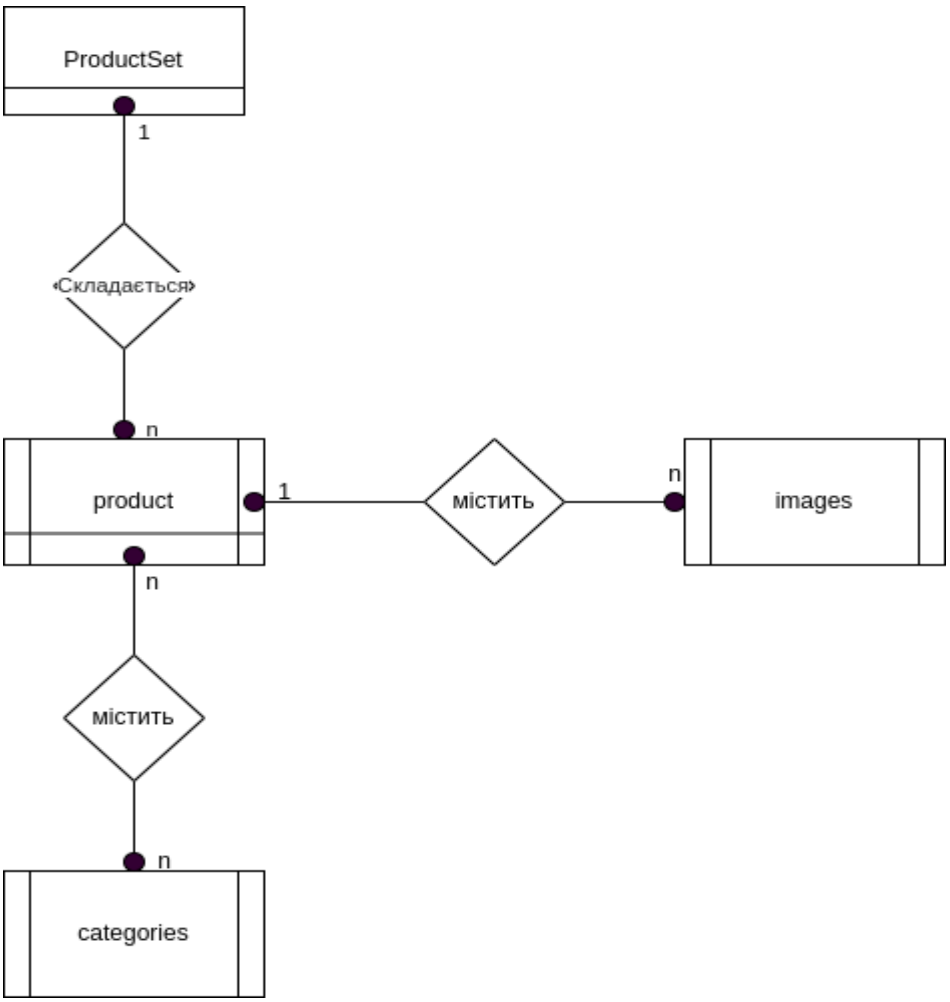

Рисунок 2.4 - Результуюча модель предметної області

2.6 Висновки

Розроблено модель та загальну структуру WEB-сервісу для ідентифікації об’єктів на зображеннях.

1. Подальшого розвитку отримав метод ідентифікації об’єктів на зображенні, у якому, на відміну від існуючих, використано клієнт-серверну архітектуру, що зменшує технічні вимоги до робочої станції, оскільки складні обчислення виконуються на сервері.
2. Розроблено алгоритми роботи серверної частини додатку, доведена їх необхідність і важливість.
3. Розроблено алгоритми роботи мобільного додатку, доведена їх необхідність, продумана взаємодія з серверною частиною.

3 РОЗРОБКА КОДУ WEB–СЕРВІСУ

3.1 Обґрунтування вибору інтерфейсу

Інтерфейс – це сукупність засобів, методів і правил взаємодії елементів системи. Найпоширенішим на цей час є графічний інтерфейс користувача (ГІК) і вважається стандартною складовою частиною більшості доступних на ринку операційних систем і додатків.

Графічний інтерфейс користувача - це одна з найважливіших частин будь-якого додатка, хороший інтерфейс повинен бути інтуїтивно зрозумілий, або вимагати мінімальних пояснень. Практика показує, що якщо користувач не може розібратись з інтерфейсом протягом 15 секунд, то імовірність видалення додатка підвищується на 90%. Ця проблема навіть дала поштовх розвитку такого напряму, як UX дизайн, навідміну від звичайного його суть в зручному розміщенні елементів керування, і перетворенні звичайного інтерфейсу на інтуїтивно зрозумілий. Основним ідеологом цього напряму став Tim Cook, на даний момент все більше й більше компаній шукають до себе в команду спеціалістів в сфері UX-дизайн

Існує 3 види графічного інтерфейсу:

- графічний інтерфейс користувача;
- інтерфейс командного рядка;
- Веб-інтерфейс.

Розглянемо переваги та недоліки кожного з них.

Графічний інтерфейс користувача(Рис 3.1) є найуніверсальнішим і найбільш розповсюдженим для десктопних і мобільних додатків. До переваг можна віднести зручність, універсальність і можливість створювати складні додатки до недоліків велику вагу і складність розробки у порівнянні з іншими варіантами. Варто зазначити, що це один з найбільш пізніх й сучасних варіантів взаємодії програми з користувачем.

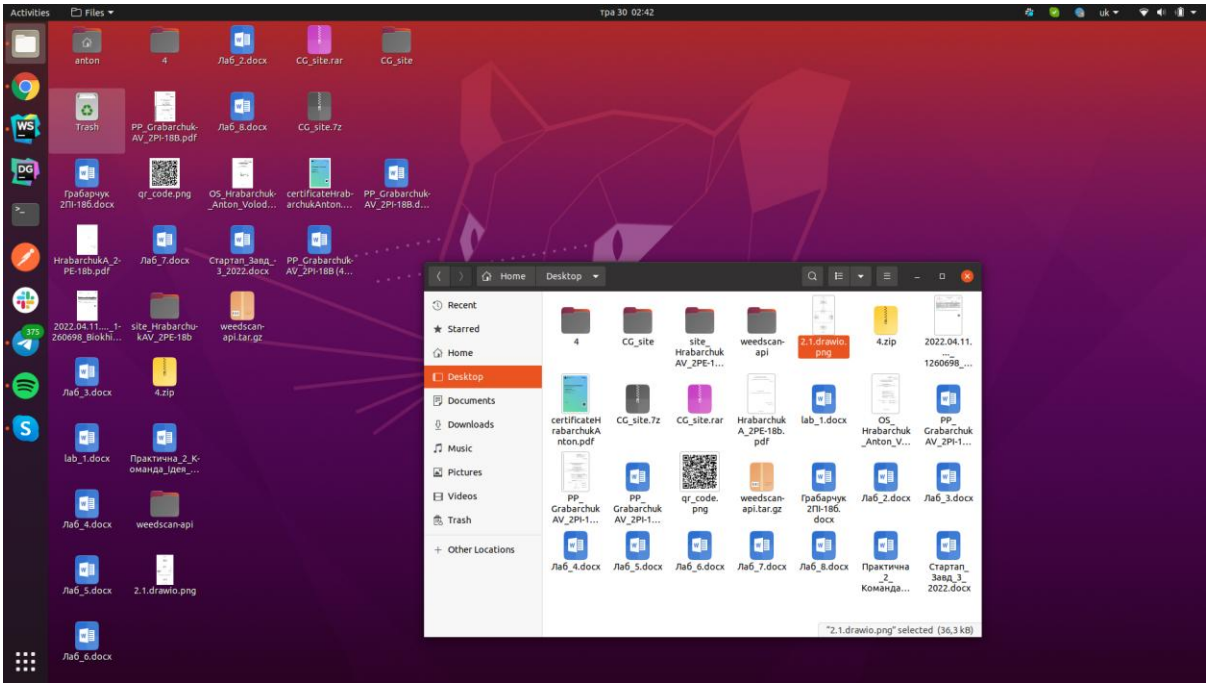

Рисунок 3.1 - Графічний інтерфейс

Наступний варіант - це інтерфейс командного рядка(Рис. 3.2), є не таким важким для системи, але набагато менш зрозумілим для користувача, і не має можливості створювати складні додатки, але при цьому швидкість розробки такого типу інтерфейсу для простих додатків є надзвичайно високою. Фактично був першим саме графічним інтерфейсом котрий створила людина для взаємодії з комп'ютером. Всупереч своєму поважному віку досі активно використовується в програмних додатках, але переважно серед комп'ютерних спеціалістів та спеціалістів з розробки програмного забезпечення, оскільки може бути застосованих навіть на найслабкіших обчислювальних машинах. Особливо популярний при взаємодії з операційною системою Linux, й серверах котрі на ній працюють, оскільки дає можливість під'єднатися через ssh-ключ до сервера і за допомогою термінала вводити команди керуючи роботою сервера. Завдяки своєму невеликому навантаженню на систему дозволяє працювати без жодних перешкод навіть при поганому Інтернеті. Що в багатьох випадках є критично важливим, і дозволяє економити час.

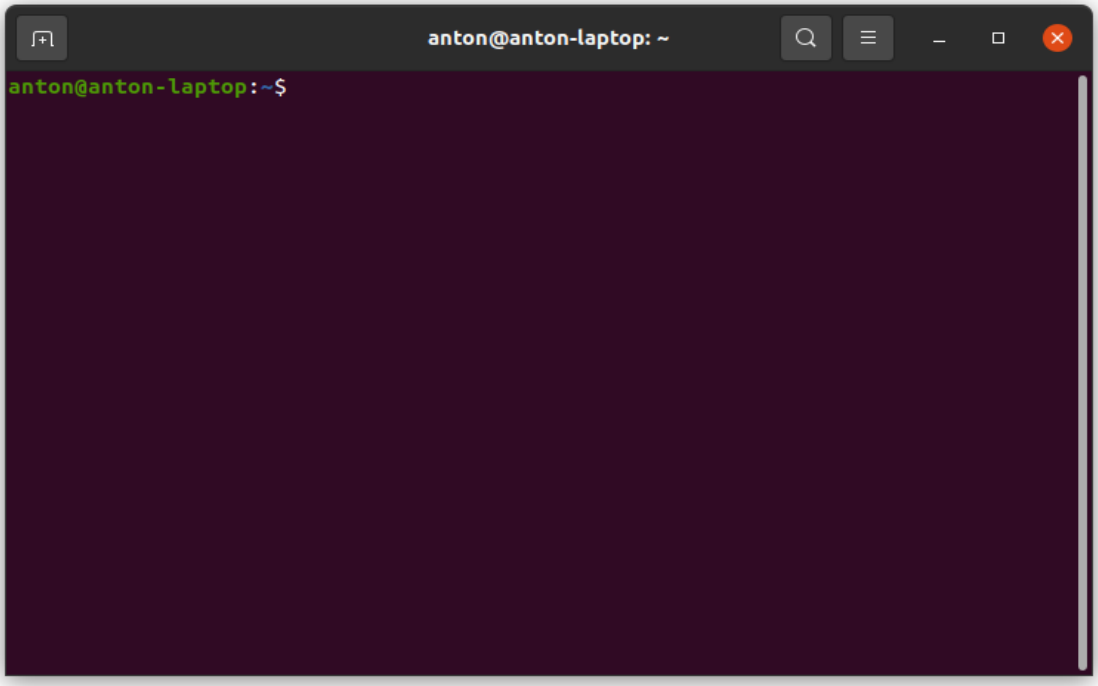

Рисунок 3.2 - Інтерфейс командного рядка

Вебінтерфейс(Рис. 3.3) - схожий на графічний інтерфейс, але може бути застосований лише у веббраузері.

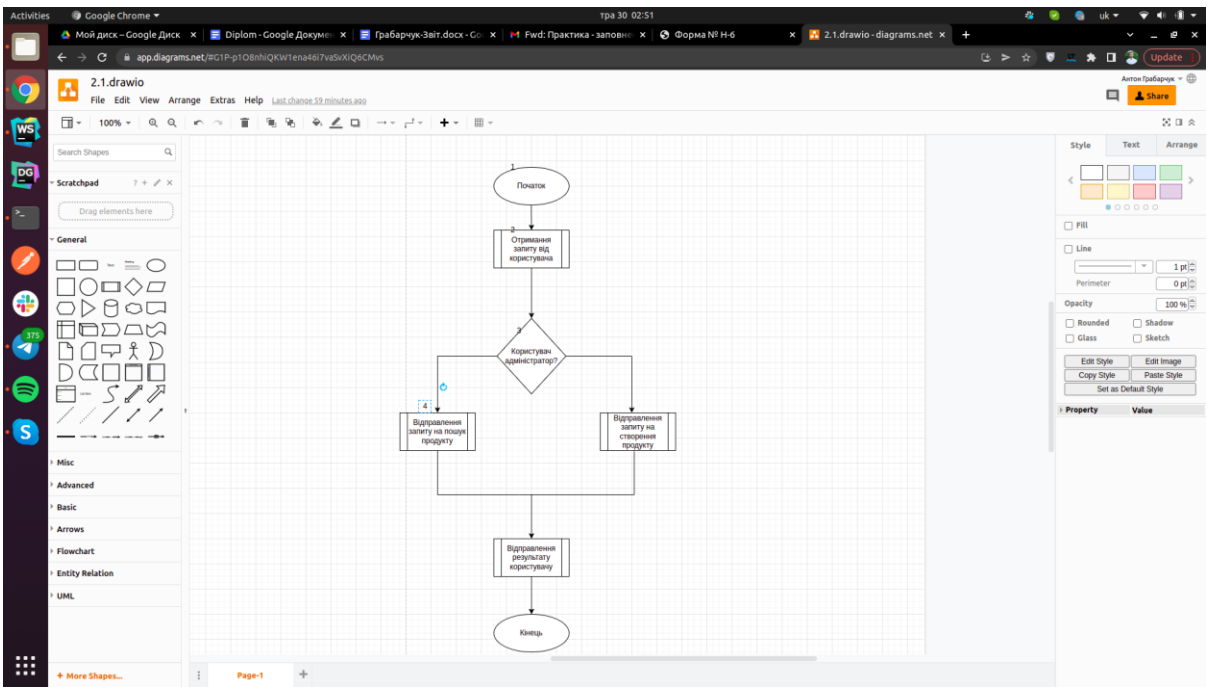

Рисунок 3.4 – Веб-інтерфейс

3.2 Розробка інтерфейсу

Враховуючи всі переваги та недоліки видів графічних інтерфейсів було обрано графічний інтерфейс користувача, оскільки він є універсальним для всіх операційних систем, простий для користувача і надає широкі можливості з розробки.

При проєктуванні графічного дизайну було обрано обрати за стиль мінімалізм(Рис. 3.5), не додаючи жодних зайвих компонентів. Також ставка робилась на простоту, оскільки користувач повинен бути здатен працювати з додатком без жодної додаткової підготовки.

Рисунок 3.5 - графічний дизайн розробленого додатка

3.3 Розробка програмної компоненти

3.3.1 Обґрунтування вибору мови програмування

Програмна компонента вже по суті є самим додатком, до її розробки приступають після розробки дизайну й основних діаграм, оскільки ці пункти вже виконані в попередніх розділах, можна перейти до, безпосередньо, розробки коду.

Перш за все ми повинні обрати мову програмування для серверної частини.

З моменту винайдення першого комп’ютера постала проблема його програмування, спочатку використовувався двійковий(бінарний) код, але складність в його читанні, низька швидкість розробки програмного забезпечення, змусило розробників шукати нові шляхи розробки ПЗ. Наступним щаблем еволюції стала мова Ассемблер, вона значно пришвидшила розробку, спростила її, але все одно залишалася надто складною й громіздкою. Тому була розроблена мова програмування С котра стала справжнім проривом, адже вона стала врази простішою, але при цьому майже не втратила швидкості.

Варто зазначити, що всі попередньо описані мови є морально застарілими й не могли бути обрані, як оптимальні для розробки даного технічнго завдання.

Найпопулярнішими рішеннями для серверної розробки на цей час є такі мови програмування:

- Java;
- C#;
- Python;
- JavaScript.

Розглянемо кожну мову програмування, щоб оцінити її переваги та недоліки.

Java — об'єктно-орієнтована мова програмування, створена компанією Sun Microsystems в 1995 році. Синтаксис мови багато в чому походить від C та C++. Java програми виконуються у середовищі Java Virtual Machine. Java

програми компілюються у байткод, який при виконанні інтерпретується віртуальною машиною для конкретної платформи. Що створює основний принцип цієї мови “Write once run everywhere” - пиши(код) один раз запускай всюди. В 2009 році Sun Microsystems придбала компанія Oracle, яка продовжує розвивати цю мову. Найстарша із представлених в цьому розділі мов програмування. Має перевагу завдяки гарній структурованості, що є важливим для великих проектів, котрі необхідно довго підтримувати. Має хороше комьюніті котре вже довго існує, відповідно легко знайти рішення на майже будь-яку помилку. Але до недоліків можна віднести повільність, що може бути критичним для малобюджетних проєктів, оскільки навантаження на сервер буде велике. Також недоліком є низька швидкість розробки. Ці недоліки є критичними для розроблюваного проєкту, оскільки він не має великих інвестицій і команди розробки.

C# — це також об'єктно-орієнтована мова програмування, створена в 2002 році, фактично є клоном Java, має всі ті ж самі переваги та недоліки.

Python — інтерпретована об'єктно-орієнтована мова програмування високого рівня зі строгою динамічною типізацією. Створена в 1990 році. Структури даних високого рівня у зв’язці з динамічною семантикою, а також динамічним зв'язуванням роблять її привабливою для швидкої розробки програм, і як засіб поєднування готових компонентів(бібліотек). Python підтримує модулі та пакети модулів, що сприяє модульності та повторному використанню коду. Інтерпретатор Python та стандартні бібліотеки доступні як у скомпільованій, так і у вихідній формі на всіх основних платформах. В мові програмування Python підтримується кілька парадигм програмування, зокрема: об'єктно-орієнтована, процедурна, функціональна та аспектно-орієнтована. Зручна мова, але залишається проблема зі швидкодією й немає можливості писати код для мобайл додатків.

JavaScript — динамічна, прототипна, об'єктно-орієнтована мова програмування. Реалізує стандарт ECMAScript. Найчастіше використовується для створення логіки роботи вебсторінок, надає можливість на боці клієнта

змінювати структуру та зовнішній вигляд вебсторінки, взаємодіяти з користувачем, асинхронно обмінюватися даними з сервером керувати браузером. Є простою мовою в освоєнні. Також її використовують такі платформи, як Node.js версія JavaScript для серверної розробки, і React native фреймворк для розробки під мобайл. Універсальність, простота розробки, хороша швидкодія, робить цю мову програмування ідеальним вибором для виконання необхідної роботи.

Отже, для виконання роботи обрано мову програмування JavaScript, для серверної частини буде використана платформа Node.js, для мобільної фреймворк ReactNative.

3.3.2 Обґрунтування вибору середовища розробки

Середовище розробки не менш важливий інструмент ніж мова програмування, саме він здатен запускати наш код, відображати його, керувати ним. Просунуті версії середовищ розробки здатні знаходити помилки в коді, надають доступ до термінала через свій інтерфейс. Без сучасних середовищ розробки була б неможлива продуктивна розробка програмного забезпечення. Ринок дає широкий вибір інструментів, велика кількість компаній будують свій бізнес саме на розробці даних програмних продуктів, але беззаперечним лідером є чеська компанія JetBrains. Як правило для кожної мови програмування існує своє середовище розробки IDE.

Втім трапляються і винятки з правила, наприклад Notepad++(Рис. 3.6) запущений 23 листопада 2003 року програмістом Доном Хо й розповсюджується за ліцензією GPL 3.0, але активно розвивається і використовується по сьогоднішній день, написаний мовою програмування C++, що добре вплинуло на його швидкодію, є універсальним засобом для розробки ПЗ, однаково гарно працює з Java, C++ та навіть Assembler, утім цей засіб має величезний недолік, а саме свою низько рівневість, і відсутність власних механізмів взаємодії з файловою системою. Це робить неможливою комфортну і продуктивну розробку будь-якого середнього або великого проєкту. Також

серйозним недоліком є те що він працює винятково в операційній системі Windows, отже його неможливо використати на ОС Linux та MacOS.

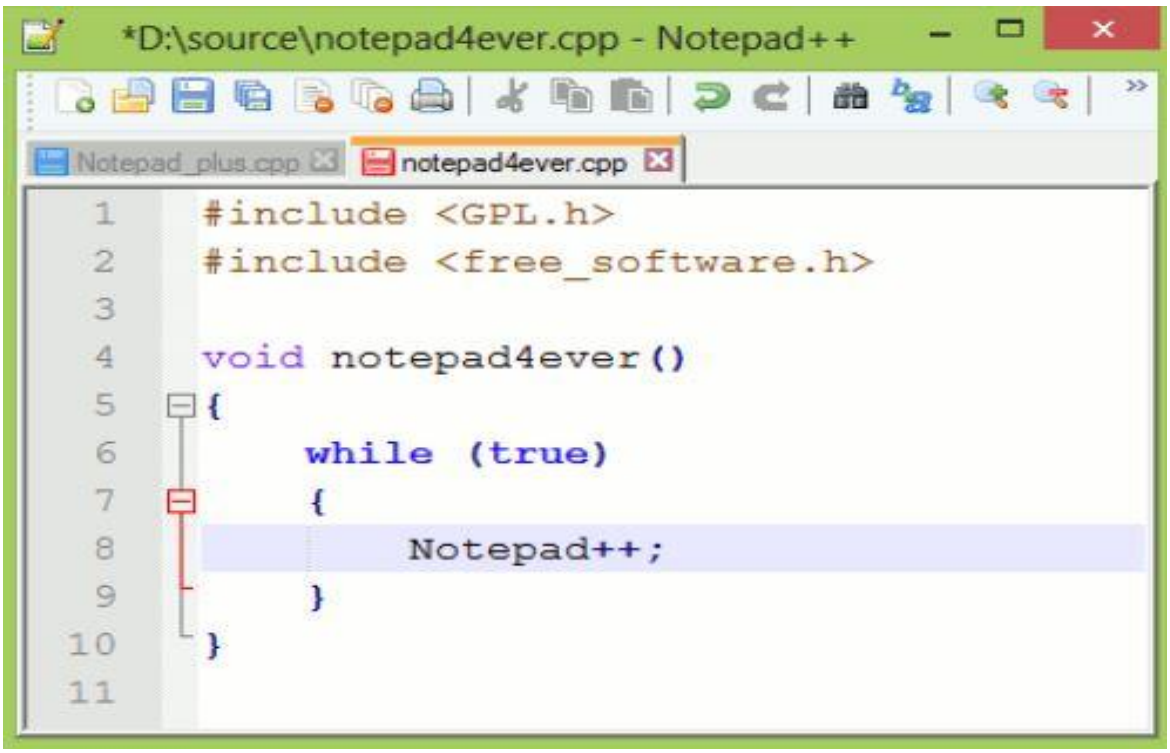

Рисунок 3.6 - інтерфейс додатка Notepad++

Оскільки була обрана мова програмування JavaScript, то зважаючи на низьку функціональність універсальних середовищ розробки варто підібрати IDE котрі використовуються спеціально для цієї мови, основних конкурентів на ринку є два:

- Visual studio code;
- WebStorm.

Visual studio code - середовище розробки створене компанією Майкрософт(не плутати з Visual studio) 18 листопада 2015 року, активно розвивається по сьогоднішній день та має широке комьюніті, створена за допомогою мови програмування TypeScript, дає ширший функціонал у порівнянні з Notepad++, також до переваг відносяться широкі можливості з кастомізації, за допомогою плагінів, наприклад можливість додати компілятор

або підсвічення синтаксису для необхідної мови, таким чином кожен користувач може підігнати систему під себе і для своїх потреб, є можливість самостійної розробки плагінів. Але в цьому і криється недолік, даний програмний додаток вимагає багато часу і зусиль для його налаштування. З переваг варто зазначити, що цей програмний продукт є повністю безкоштовним.

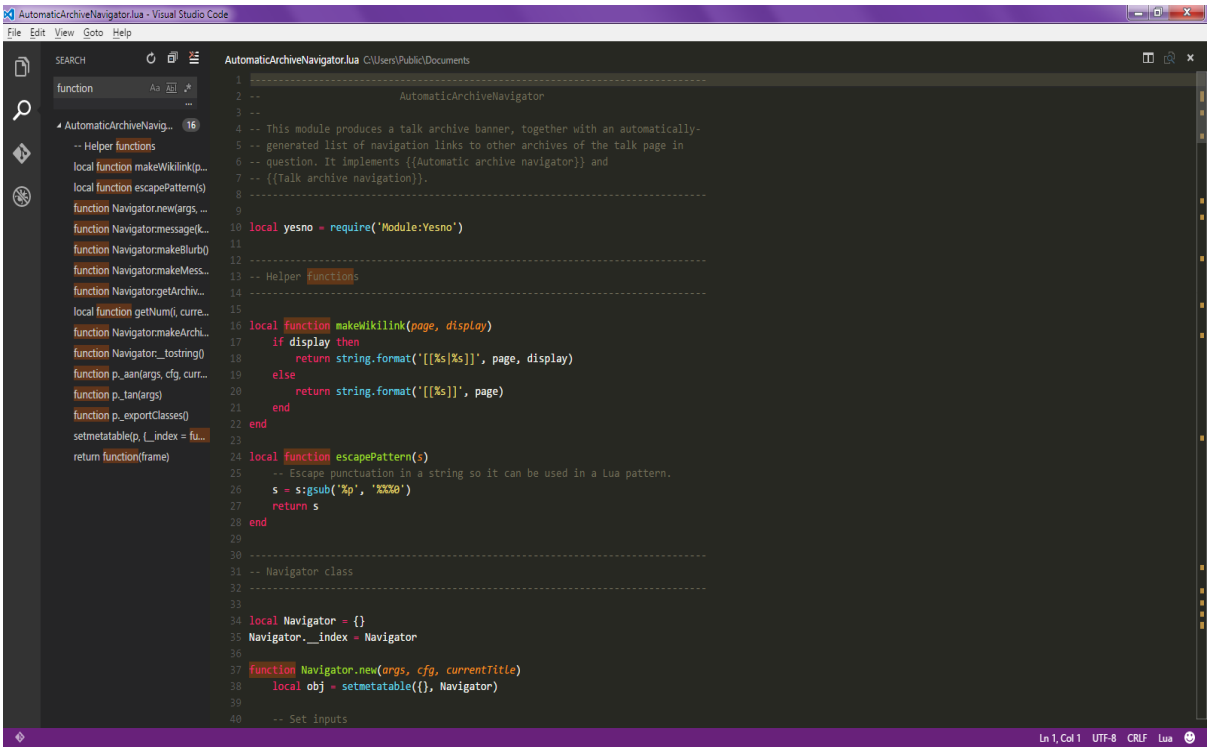

Рисунок 3.7 - Інтерфейс додатка Visual studio code

WebStorm - продукт чеської компанії JetBrains, створений у 2010 році, на основі програми для розробки Java додатків Intelij Idea, активно розвивається і сьогодні. Написаний мовою програмування Java. Є найбільш досконалим із всіх існуючих на ринку додатків, має вбудовані засоби для роботи з базою даних, вбудований термінал, кнопки запуску додатка, дебагер. Має найкращий UX дизайн, тобто система продумана до дрібниць, всі кнопки керування знаходяться у найзручніших місцях, а також присутня велика кількість функціонала котра просто робить розробку приємнішою, наприклад є кнопка котра показує місце знаходження у файловій структурі проєкту конкретного

відкритого на екрані файлу. Варто зазначити, що програма є платною, але вона має безкоштовну підписку за студентським квитком. З недоліків можна зазначити велике навантаження котре вона створює на систему, тому для її використання необхідно мати потужний комп'ютер.

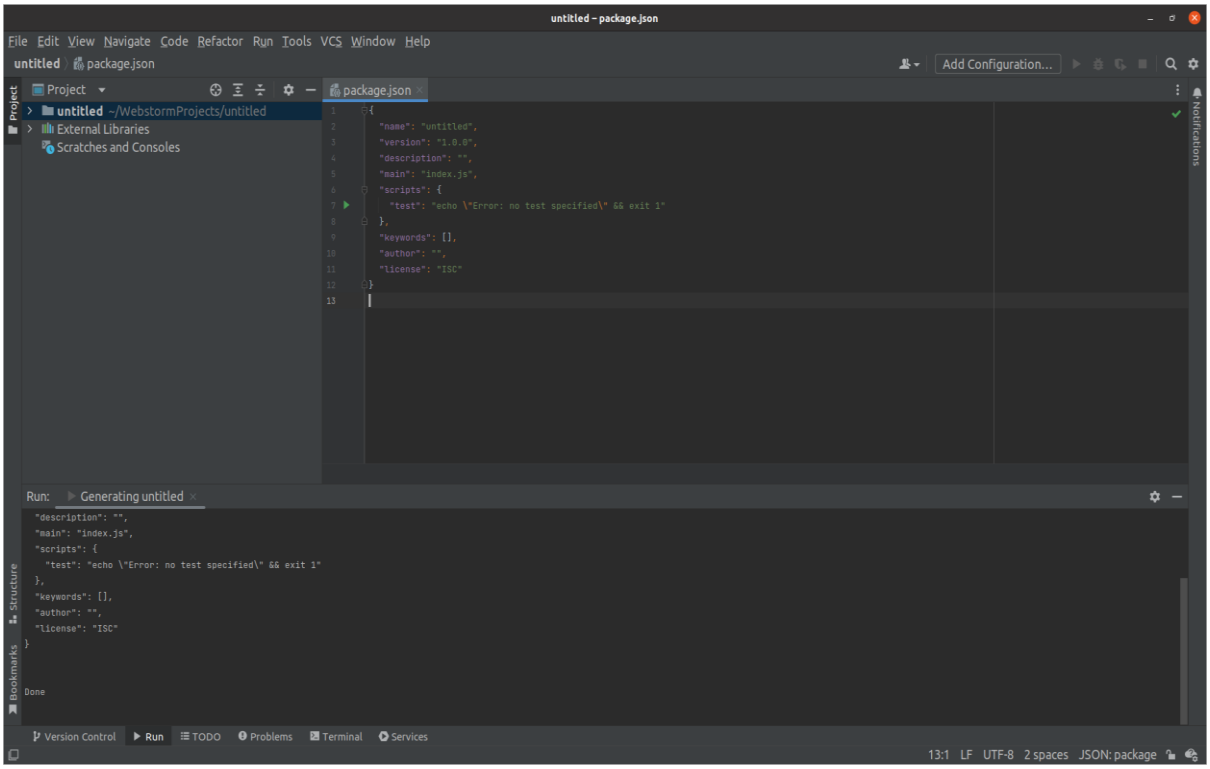

Рисунок 3.8 - Інтерфейс додатка WebStorm

Отже, проаналізувавши переваги та недоліки всіх представлених додатків було обрано для розробки представленої бакалаврської роботи додаток від компанії JetBrains під назвою WebStorm. Оскільки його широкий функціонал буде найкращим варіантом для розробки необхідного програмного продукту.

3.3.3 Обґрунтування вибору СУБД

В попередніх розділах було вирішено використовувати саме реляційну базу даних і створені схеми котрі описують архітектуру бази даних розроблюваного додатка, оскільки дана концепція найкраще підходить до поставлених вимог. Тепер необхідно обрати Систему Управління Базами Даних. Цей вибір не менш важливий, ніж вибір типу бази даних. Оскільки

кожна СУБД це окрема технічна й програмна реалізація певного концепту, у нашому випадку концепту реляційної бази даних

Система управління базами даних (СУБД, СКБД) — набір взаємопов'язаних даних (база даних) і програм для доступу до цих даних[9]. Надає можливості створення, збереження, оновлення та пошуку інформації в базах даних з контролем доступу до даних.

Ринок надає широкий вибір субд для реляційних баз даних. Нами буде розглянуто дві найпопулярніші та найдосконаліші з тих котрі існують на цей час, а саме:

- MySQL;
- PostgreSQL.

MySQL — вільна система для керування реляційними базами даних, котра була розроблена компанією «TcX» для підвищення швидкодії обробки великих баз даних. Ця система управління базами даних (СУБД) з відкритим кодом була створена як альтернатива комерційним системам. MySQL з самого початку була дуже схожою на mSQL, проте з часом вона все розширювалася і зараз MySQL — одна з найпоширеніших систем керування базами даних. Вона використовується, перш за все, для створення динамічних вебсторінок, оскільки має чудову підтримку з боку різноманітних мов програмування. У 2008 році викуплена компанією Sun Microsystems, а після поглинання цієї компанії компанією Oracle стала власністю останньої. Це рішення з великою історією, втім воно активно підтримується і по сьогоднішній день, тому є актуальним. Має непоганий графічний інтерфейс для керування базами даних(Рис. 3.9). Але з недоліків можна зазначити низьку швидкодію, також частина функцій є платними, що може негативно вплинути на якість додатка та вартість розробки. Також варто зазначити, що всі права належать одній компанії, що негативно впливає на комьюніті даного програмного рішення, і розвиток може бути повільнішим ніж у додатків котрі підтримують ентузіасти.

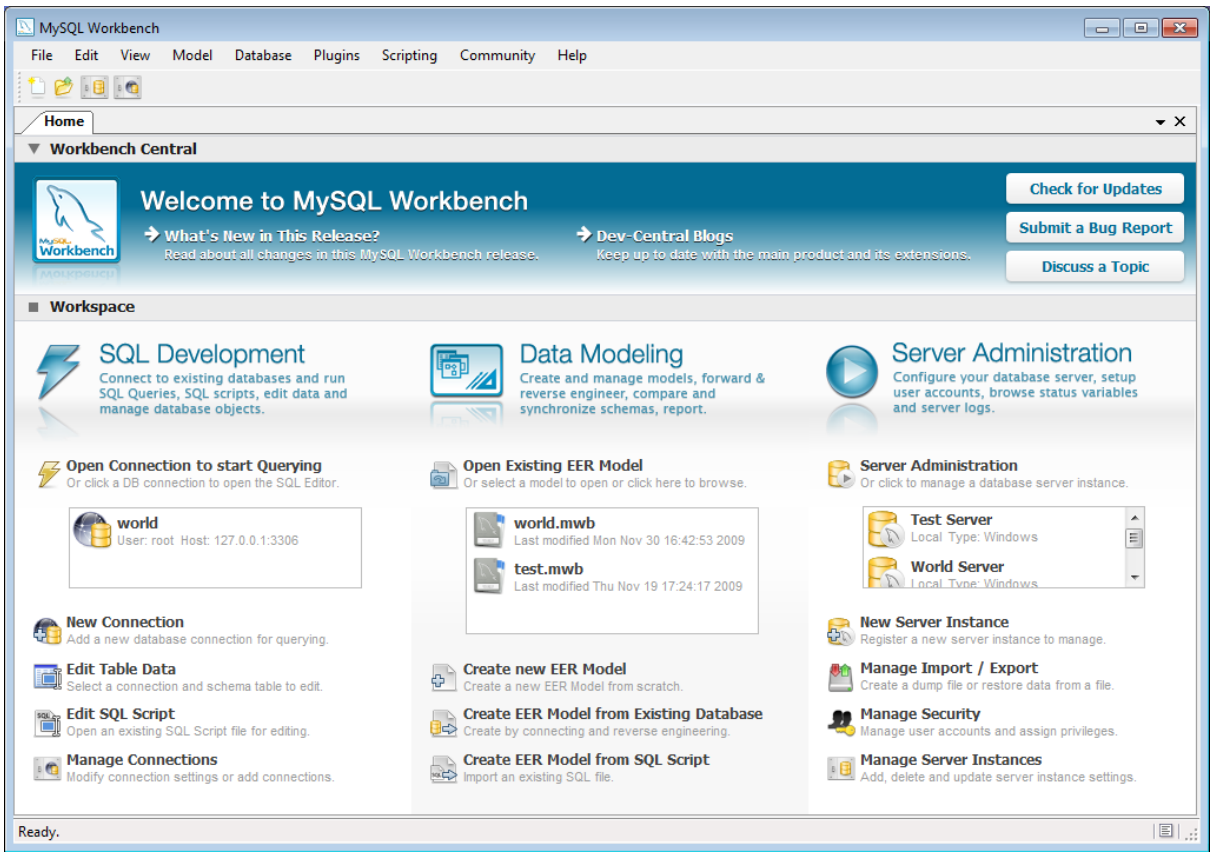

Рисунок 3.9 - Інтерфейс додатка MySql Workbench

PostgreSQL — це об'єктно-реляційна система управління базами даних (СУБД). Альтернатива як комерційним СУБД (Oracle Database, Microsoft SQL Server, IBM DB2 та інші), так і СКБД з відкритим кодом (MySQL, Firebird, SQLite).

Порівняно з іншими проєктами з відкритим кодом, такими як MySQL, PostgreSQL не контролюється однією компанією, її розробка можлива завдяки співпраці багатьох людей та компаній, які бажають використовувати вказану СУБД, а також впроваджувати у неї найновіші досягнення. Перший прототип був створений в університеті Берклі(Каліфорнія), у 1987 році. В 1990 та 1991 роках вийшли друга та третя версії відповідно. У 1996 році отримав свою актуальну назву та версію 6.0. Розробляється групою ентузіастів-спеціалістів в галузі баз даних, а також тими хто бажає внести свій внесок в розвиток проєкту. Є програмним продуктом з відкритим кодом, що дає змогу будь-кому дослідити даний продукт, а також покращити його, чи підігнати під свої потреби, набагато

легше налаштовується у порівнянні з MySQL, а також є набагато швидшим і надійнішим. Має доволі непоганий графічний додаток PgAdmin(Рис. 3.10) котрий є доволі функціональним і зручним.

Рисунок 3.10 - Інтерфейс додатка PgAdmin

Отже, у зв'язку з перевагами додатка PostgreSQL було обрано використовувати саме його. Як більш швидку, надійну та просту реалізацію.

3.3.4 Обґрунтування вибору сервісу для розпізнавання образів

Основа розроблюваного додатка полягає саме в сервісі для розпізнавання образів, оскільки саме він і виконуватиме функції з розпізнавання зображень, останнім часом ринок активно розвивається, і з'являються все нові й нові рішення. Найбільш досконалими на сьогодні є два з них, це:

- Google cloud Vision Api;
- Amazon Rekognition.

Google Vision API - це послуга аналізу зображень Google Cloud Platform, заснована на попередньо навченому та постійно розвиваючомуся машинному

навчанні з застосуванням систем глибокого навчання. Є одним з провідних стандартів точності розуміння зображень штучного інтелекту. Навчальна програма EITC/AI/GVAPI Google Vision API концентрується на роботі з AI Vision у Python через API Vision Google Cloud, котрий є сильним хмарним сервісом AI, й пропонує попередньо навчені та постійно вдосконалювані моделі машинного навчання. Для взаємодії з ними є хороша консоль розробника(Рис. 3.11). А також хороше API, для взаємодії. Окрім того, присутні бібліотеки для всіх найпопулярніших мов програмування, зокрема для платформи Node.js на основі JavaScript. Використовуючи Vision AI, можна виконувати завдання для аналізу візуальних даних, наприклад присвоєння міток зображенням для організації великих баз даних зображень, отримання рекомендованих вершин обрізання, виявлення відомих пейзажів або місць, витягування текстів та багато іншого. Також додаток є безкоштовним до певної кількості запитів, що також є критично важливим для розробки нашої системи.

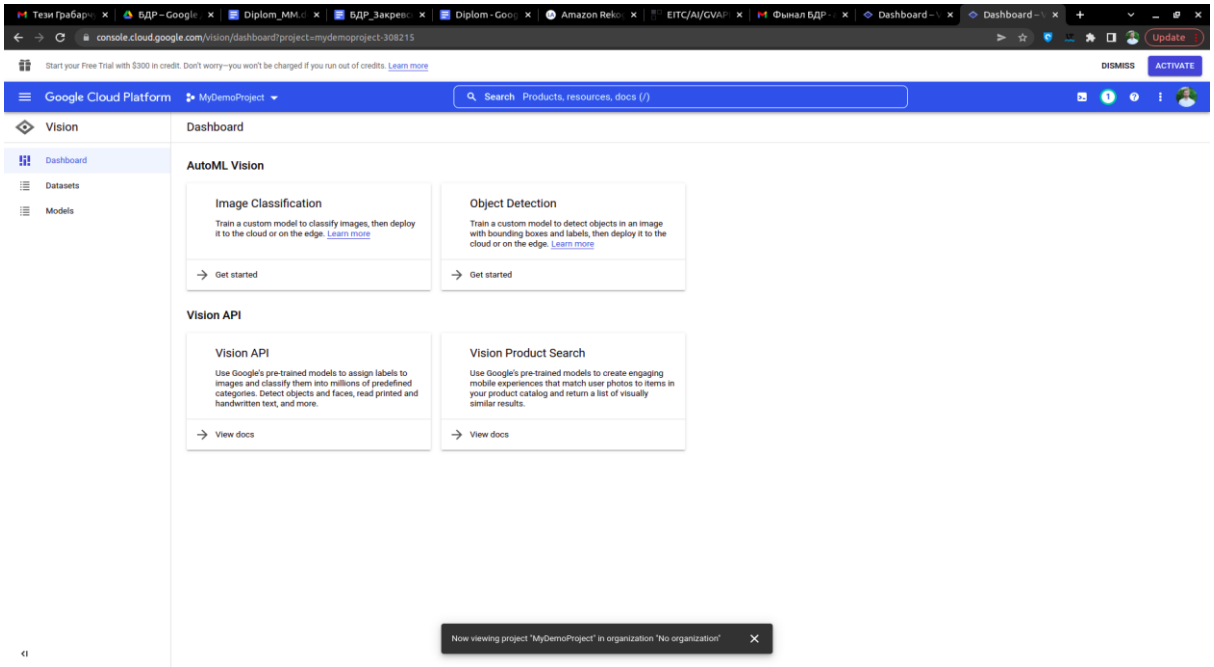

Рисунок 3.11 - інтерфейс додатка Google cloud Vision API

Amazon Rekognition - система розпізнавання образів від компанії Amazon. Має ширший функціонал у порівнянні з попереднім рішенням, тобто може

розпізнавати об’єкти не лише на фото, а й на відео. Додаток теж має непогану консоль(Рис. 3.11) З недоліків можна відзначити відсутність безкоштовного тарифу, а також відсутність готових бібліотек. Це все сильно ускладнює розробку нашого проєкту.

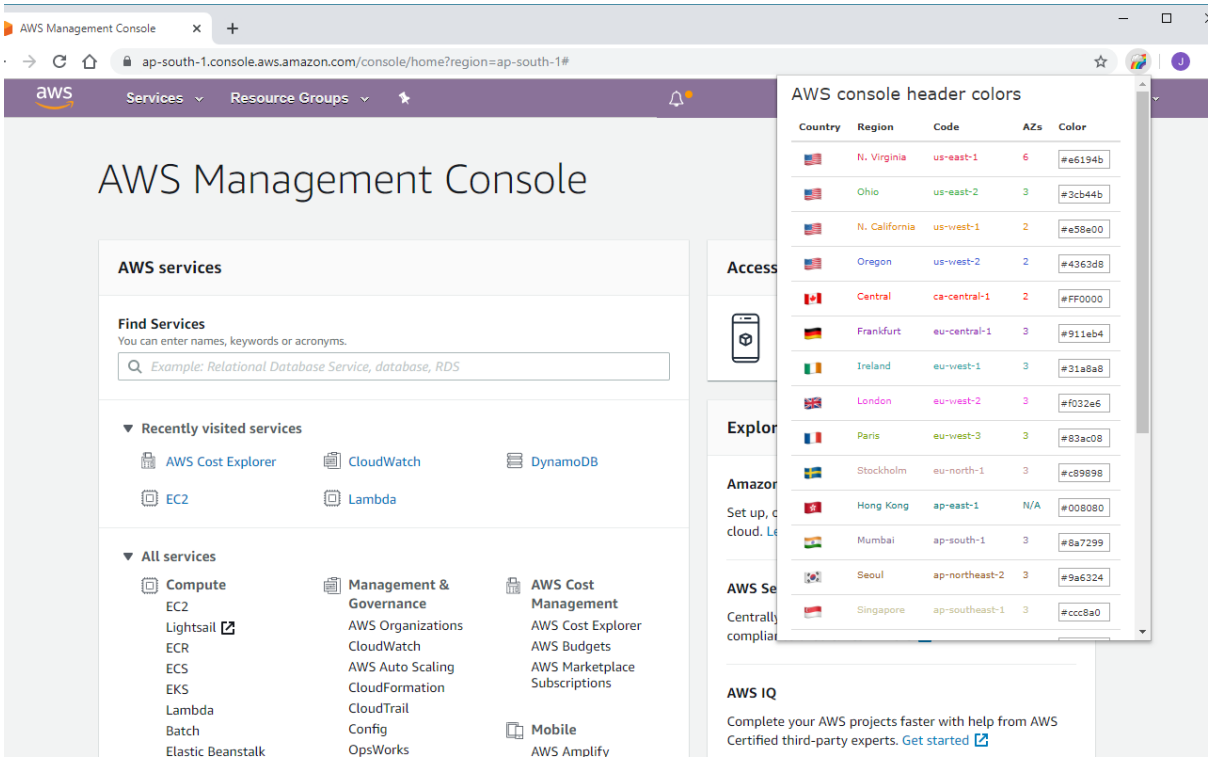

Рисунок 3.11 - інтерфейс додатка AWS Managment console

3.3.5 Розробка програмного коду серверної частини

Отже, на основі обраних засобів, і створених алгоритмів можна приступити до розробки програмного коду.

Першим важливим фактором котрий ми розглянемо буде взаємодія нашої програмної компоненти з базою даних. Оскільки це завжди є найбільш вузькою частиною розробки ПЗ, оскільки неправильно побудований процес може призвести до великих проблем зі швидкодією.

Для платформи Node.js є два основних рішення:

- TypeORM;
- Sequelize.

Хоча обидва варіанти можна використовувати для JavaScript, все ж TypeORM краще підходить для TypeScript, на Js є лише порт котрий працює не найкращим чином. Тому для розробки програмного продукту буде використано Sequelize котрий безпосередньо створений для цих цілей.

Отже, для встановлення зв’язку з БД необхідно створити наступний об’єкт:

```
const Sequelize = require("sequelize");
const sequelize = new Sequelize(DB_NAME, USER_NAME, PASSWORD, {
  host: config.host,
  dialect: config.DIALECT,
});
```

В нього передати наступні змінні:

- DB_NAME - назва використовуваної бази даних;
- USER_NAME - ім’я користувача;
- PASSWORD - пароль від бази даних;
- host - адреса БД;
- dialect - тип використаної СКБД.

Принцип роботи Sequelize заключається в створені моделі для кожної таблиці і подальшої взаємодії через неї.

Для прикладу розглянемо модель таблиці Products. Спочатку потрібно підключити бібліотеку sequelize для зв’язку з базою даних та скофігурувати таблицю Products (файл database.js):

```
const Sequelize = require("sequelize");
const sequelize = require("../database").
```

Ініціалізує модель метод sequelize.define(), який приймає два аргументи - назву таблиці та об’єкт із метадатою для створення моделі:

```
const products = sequelize.define('products', { … });
```

В об’єкті метадати створюємо поля відповідно до назви полів таблиці. Опис поля, яке є первинним ключом такий:

```
product_id: {
    primaryKey: true,параметрів таке:
    autoIncrement: true,
    allowNull: false,
    type: Sequelize.INTEGER
},
```

Де призначення параметрів наступне:

- primaryKey - вказує, що поле є первинним ключем;
- autoIncrement - при кожному записі числове поле збільшується на один;
- allowNull - вказує чи може значення поля бути Null;
- type - вказує на тип запису;
- unique - вказує чи повинне значення бути унікальним.

Крім того, в таблиці Products створюються інші поля:

- поле product_guid – непередбачуваний унікальний ідентифікатор;
- поле name – ім’я продукту
- поле description – поле опису продукту;
- поле category_id – посилання на запис в таблиці Gategories.

Повний лістинг коду методу sequelize.define() наведено в додатку В.

Наступним необхідним аспектом розробки є API. На сьогодні існує два основних способи створення API, це використання:

- Rest API;
- GraphQL API.

REST — підхід до архітектури мережевих протоколів, котрі надають доступ до інформаційних ресурсів. Був описаний і популяризований 2000 року Роєм Філдінгом, одним із творців протоколу HTTP. В основі REST закладено принципи функціонування Всесвітньої павутини й можливості HTTP. Філдінг розробив REST паралельно з HTTP 1.1 базуючись на попередньому протоколі HTTP 1.0. Для передачі даних використовує формат Json.

JSON — текстовий формат обміну даними. Основується на тексті, й може бути прочитаним людиною. Формат надає змогу описувати об'єкти, та інші структури даних. Даний формат використовується для передавання структурованої інформації через мережу.

GraphQL — мова запитів й маніпуляції даними з відкритим кодом для API й середовище виконання для обслуговування запитів з наявних даних. GraphQL розробила компанія Facebook в 2012 році, а публічний реліз відбувся 2015 року. 7 листопада 2018 року, GraphQL переведено від Facebook до новоутвореної GraphQL фундації, котру прийняла неприбуткова Linux фундація. Є більш зручним в порівнянні з REST для великих проєктів оскільки клієнт може вказувати які саме поля йому потрібні.

Отже, було обрано використовувати стандарт Rest API, в зв’язку з тим, що його структура легша, зручніша, й простіша для читання, а також більш зручна для розробки невеликих й середніх проєктів.

Для розробки Rest API додатка, використаємо фреймворк Express, це легкий фреймворк котрий дозволяє у зручному форматі описати ендпоінти.

Функціонал серверного додатка передбачає ендпоінти для мобільної та адмінівської частини.

Для реалізації endpoint мобільної мобільної частини необхідно підключити фреймворк express та викликати метод Router():

```
const {Router} = require("express");
const router = Router();
```

Далі потрібно потрібно підключити mobileService, де виконуються основна логіка додатку та сконфігурувати проєкт:

```
const mobileService = require('../services/mobileService');
const ResponseError = require("../utils/ResponseError");
const PROTOCOL_VAR =
require("../config/appConfig").weedServer.protocol;
```

Перший enspoint це categories, його суть надати перелік можливих категорій товарів:

```
router.get( '/categories', async (req) => {
  return {
     categories: await mobileService.getCategories()
  };
});
```

Другий - це /search/image, по суті він і є основним функціоналом додатку, тобто приймає від користувача фото для пошуку і надсилає отриманий результат. При цьому виконується перевірка формату зображення на відповідність внутрішнім форматам бази даних, тобто перевіряється чи є зображення даними типу String, якщо формат не відповідає вимогам виводиться повідомлення про полку:

```
if (!_.isString(image) || !image.length)
     throw new ResponseError("image field is missing", 400);
```

Крім того, перевіряється чи є категорія типом String і наявність такої категорії в базі даних і у випадку невідповідності однієї з умов виводиться повідомлення про помилку:

```
  if (_.isString(category) && !await mobileService.getCategory(category))
     throw new ResponseError("Invalid category", 400);
```

Далі виконується ідентифікація зображення з використанням методу searchImage () з поверненням результатів користувачу (див. додаток В):

```
const products = await
mobileService.searchImage (req.headers[PROTOCOL_VAR],req.headers.host,
image, category);
```

Endpoint для адмінської частини виконує функцію валідації даних, тобто перевіряє дані на відповідність очікуваним. Зокрема, виконує такі перевірки:

- чи є назва продукту типу String і чи не рівна вона «ноль». Невиконання умов призводить до виедення помилки «name field is missing»;

- чи є опис продукту типу String і чи не рівний він «ноль». Невиконання умов призводить до виедення помилки «description field is missing»;
- чи є категорія типу String і чи не рівна він «ноль». Невиконання умов призводить до виедення помилки «category field is missing»;
- чи існує дана категорія в базі даних. Невиконання умови призводить до виедення помилки «Invalid category».

У випадку успішного проходження перевірок дані повертаються користувачу:

```
return {
   product_guid: id
};
```

Для взаємодії з Google Vision Api потрібно створити продукт на стороні сервісу з використаннм методу createProduct():

```
async createProduct(name, product_guid, labels) {
  const product = {
     displayName: name,
     productCategory: googleConfig.productCategory,
     productLabels: labels || null
  };
…
}
```

Вхідні параметри методу такі:

- name - назва продукту;
- product_guid – унікальний непередбачуваний ідентифікатор;
- labels – мітка продукту, тобто це його категорія.

На основі цих вхідних даних створюється продукт, котрий записується в запит до API Google Vision:

const request = { parent: this._locationPath, product: product, productId: product_guid }; const [createdProduct] = await this._client.createProduct(request); await this.addProductToProductSet(product_guid);

Оскільки сам сервіс Google Vision Api, не здатен зберігати фото на основі, яких вивчає продукти, постає потреба в місці для їх зберігання, для цього використовується сервіс Google Bucket, тобто Google Vision Api отримує лише посилання, тому потрібно завантажити зображення в сховище.

Для завантаження зображення до сховища потрібно додати новий продукт до набору продуктів відомих Google Vision. Це виконує метод addProductToProductSet (), якому необхідно передати параметр product_guid (додаток В):

```
async addProductToProductSet(product_guid) {
  const prodPath = this._client.productPath(this._projectId, this._location, product_guid);
  const prodSetPath = this._client.productSetPath(
    this._projectId,
    this._location,
    this._productSetId
  );
  …
}
```

В кінці роботи методу запит передається в Google Vision:

```
const request = {
    name: prodSetPath,
    product: prodPath
  }
  await this._client.addProductToProductSet(request);
}
```

Завантаження зображення в сховище виконує метод uploadToGCS (), який по завершенню роботи повертає адресу файла:

return `gs://${this._savePlave}/${fileName}`;

Вхідні параметри методу uploadToGCS () такі:

- fileName - назва файлу зображення;
- contentType – тип файлу (підтримуються формати .jpg та .png);
- buffer – дані зображення

Метод downloadFromGCS() призначений для завантаження зображення з сховища. Вхідними параметрами методу є параметр name – назва файлу, який завантажуэться із сховища. По завершенню роботи методу зображення повертається в метод, який його викликав:

return (await file.download())[0];

Часто виникає необхідність в отриманні всіх продуктів. Це виконує метод getAllProductsInSet():

```
async getAllProductsInSet() {
…
  return products;
}
```

Метод getAllProductsInSet() викликає додатковий метод productSetPath (), який і формує URL для запиту.

Відправку запиту виконує метод listProductsInProductSet ():

const [products] = await this._client.listProductsInProductSet(request);

Таким чином, ми отримуємо продукт за його унікальним ідентифікатором.

І останній надважливий аспект, котрий необхідно розглянути, це додання зображення до продукту для навчання нейронної мережі. Це виконує метод addImageToProduct (). Вхідні параметри методу такі:

- productId - унікальний ідентифікатор продукту;
- gcsUri – посилання на зображення в сховищі;
- referenceImageId – унікальний ідентифікатор зображення.

Завершується робота методу додавання зображення до Google Cloud Vision для навчання:

```
await this._client.createReferenceImage(request);
```

Повий текст поогоами та методів наведено в додатку В.

3.4 Висновки

1. Обрано тип інтерфейсу та розроблено графічний інтерфейс мобільного додатку для ідентифікації об’єктів.

2. Для реалізації додатку ідентифікації об’єктів обрано мову програмування JavaScript, платформу Node.js для реалізації серверної частини додатку, фреймворк ReactNative для розробки мобільного додатку та Google Cloud Vision API для розпізнавання об’єктів.

3. Мовою програмування JavaScript з використанням платформи Node.js розроблено серверну частину додатку для ідентифікації об’єктів.

4 ТЕСТУВАННЯ WEB–СЕРВІСУ

4.1 Тестування програмного забезпечення

Баги це величезна проблема всієї IT галузі, котра завдає збитків на сотні мільйонів, якщо не мільярдів доларів кожного року. У випадку з ними діє пряма кореляція між часом знайдення багу і кількістю витрат необхідних для його виправлення. Тобто, чим раніше буде знайдена помилка, тим легше її виправити. Якщо вона буде знайдена відразу, то достатньо просто замінити кілька стрічок коду. Але якщо вона пробуде тривалий час і обросте функціоналом, який буде спиратись на цей помилковий код, то виправляти доведеться не тільки цей баг, а і всі частини які використовують зламаний код. В окремих випадках, якщо баг знайшли через кілька років, то його виправлення може стати просто неможливим, ця проблема призвела до закриття великої кількості проєктів[10].

Тестування програмного забезпечення — процес перевірки відповідності заявлених до продукту вимог і реально реалізованої функціональності, який здійснюють шляхом спостереження за його роботою в штучно створених ситуаціях і на обмеженому наборі тестів, обраних певним чином.

Тестування це надзвичайно важливий аспект розробки програмного забезпечення. Саме завдяки йому можливо знайти баги раніше ніж з ними зіткнеться користувач. Багаторівневість контролю якості властива не тільки розробці програмних додатків, а й іншим галузям, таким як машинобудування, наприклад на заводі Mercedes-Benz, в кінці конвеєра знаходиться група спеціалістів контролю якості, котра перевіряє продукцію на відсутність дефектів. Схожі принципи використовуються також у сферах будівництва.

Історія знає велику кількість випадків коли відсутність тестування призводила до фатальних наслідків, наприклад ситуація з американським космічним шатлом, котрий вибухнув через те, що при переносі коду з паперу в систему програміст переплутав знак дефіс із знаком мінус, через що система

повела себе некоректно. Інший випадок трапився у сфері медицини, а саме в рентгенівському апараті, тобто через помилку програміста, котрий писав програмне забезпечення сам, і не дотримувався правил з розробки ПЗ, через що за певних умов апарат замість того, щоб вимкнутися видавав сплеск радіації в десятки разів більше за норму, це стало причиною смерті від променевої хвороби у шістьох людей. Всіх цих помилок можна було б уникнути, якби вчасно провели якісне тестування системи

Саме тому тестуванню нашої системи була приділена велика увага, зокрема були протестовані, як серверна, так і мобайл частина, а також проведені інтеграційні тести для перевірки їх взаємодії, оскільки навіть при поєднанні двох досконалих систем можуть виникнути проблеми з їх взаємодією.

Для тестування серверної частини, або як ще її можна назвати - API, існує велика кількість способів, найбільш поширенні з них - це:

- Swagger - бібліотека, котра вбудовується в сам код і дозволяє графічно зобразити формат запитів(Рис. 4.1), а також зробити їх підставивши свої дані[11].
- Postman - окрема програма з графічним інтерфейсом(Рис. 4.2), для відправлення запитів, і отримання результату[12].

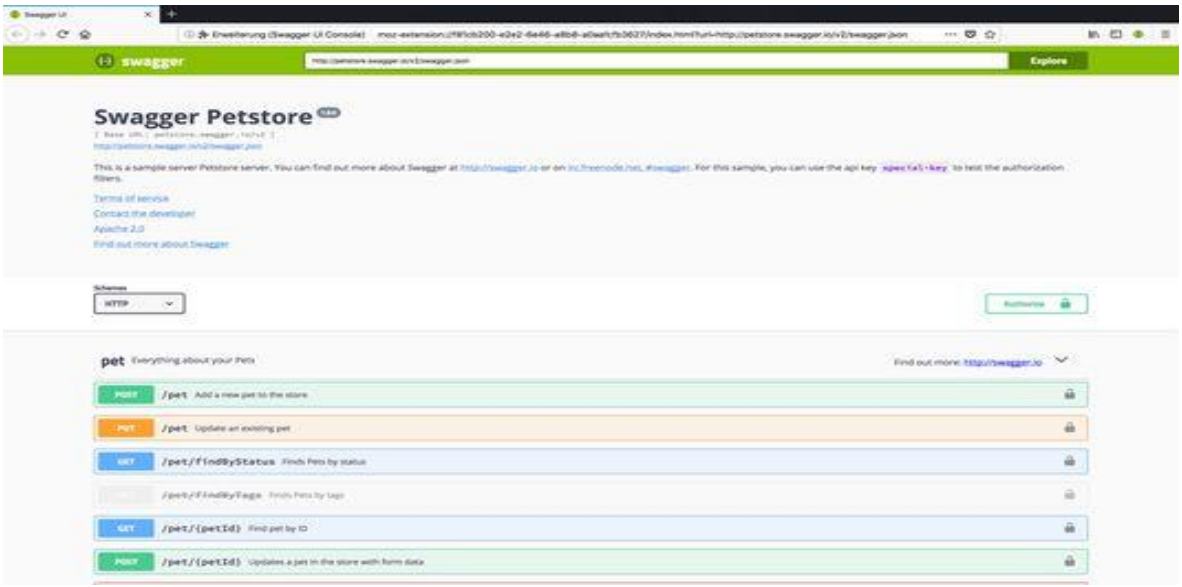

Рисунок 4.1 - Приклад тестування API за допомогою Swagger

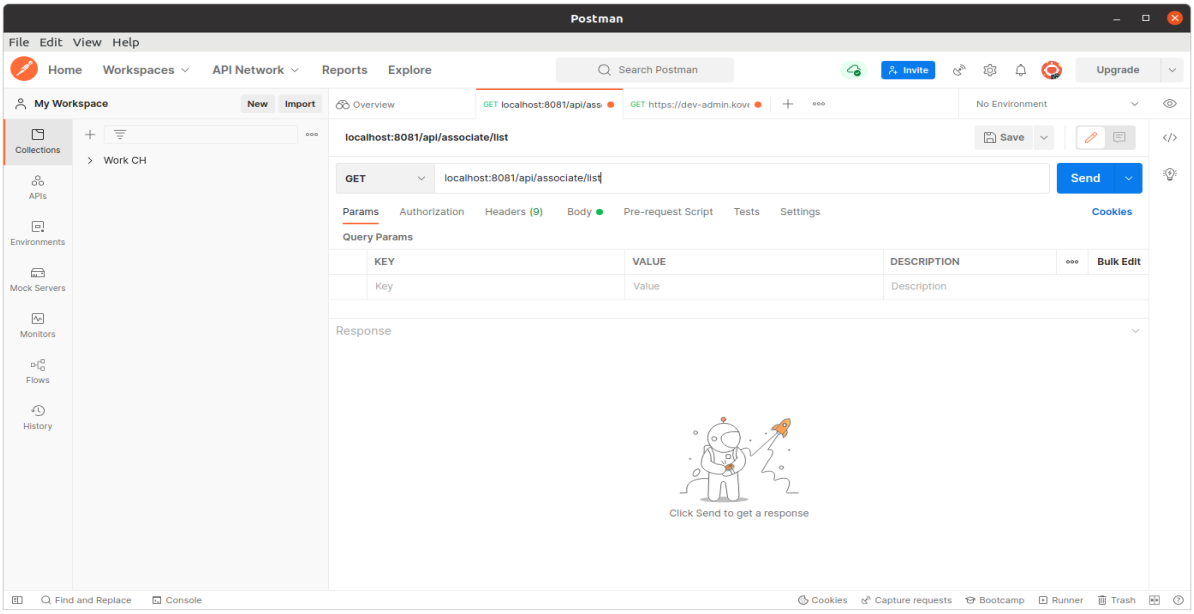

Рисунок 4.2 - Приклад тестування API за допомогою Postman

Проаналізувавши ці методи тестування було обрано використовувати для тестування Postman, оскільки він не вимагає жодних змін в коді сервера, а зручний графічний інтерфейс дозволяє швидко прописувати необхідні запити. Варто зазначити, що в подальшому Postman може бути використаний для тестування з'єднання типу socket, оскільки в останній версія програмного засобу є і така функція

Протестувавши за допомогою Postman всі ендпоінти було перевірено, що сервер працює саме так, як повинен, жодних багів знайдено не було.

4.2 Розробка інструкції користувача

Інструкція користувача з'явилась задовго до виникнення взагалі такої галузі, як інформаційні технології. Вони створювались ще для перших механізмів в часи промислової революції. І були необхідними, оскільки перші механізми були складні, і абсолютно незрозумілі для звичайного користувача. В подальшому всі механізми намагались робити максимально простими для інтуїтивного розуміння, але навіть в наш час більшість механізмів вимагають інструкції, та не є інтуїтивно зрозумілими.

Інструкція - послідовність певних кроків для використання засобу призначених для отримання необхідного результату.

Хоч додаток створювався таким чином, щоб бути інтуїтивно зрозумілим, але від цього не менш важлива детальна інструкція користувача, оскільки у разі її відсутності, користувач при виникненні проблем з додатком не матиме змоги знайти відповідь на своє питання.

При використанні додатка необхідно виконати наступні дії:

- встановити додаток з магазину PlayMarket;
- натиснути на іконку додатка на робочому столі смартфона;
- навести камеру на необхідний об’єкт, котрий ви бажаєте ідентифікувати;
- сфотографувати цей об’єкт;
- переконатись, що фото чітке і не розмите;
- якщо фото не чітке, необхідно зробити фотографію ще раз;
- якщо фото влаштовує, натисніть на кнопку відправити;
- ви побачите список результатів, для отримання детальної інформації про товар - натисніть на нього;
- для виходу з додатку натисніть кнопку “Вийти”.

Отже, детальна інструкція користувача була розроблена у цьому розділі, вона дає відповідь на абсолютну більшість базових питань з використання додатку.

4.3 Висновки

1. Тестування WEB-сервісу для ідентифікації об’єктів з використанням засобу тестування Postman показало повну його працездатність та відповідність завданню на роботу.
2. Розроблену керівництво програміста, яке забезпечує експлуатацію та подальший розвиток додатку на умовах GNU General Public License.

ВИСНОВКИ

У роботі розглянуто питання ідентифіції об’єктів на фотографічних зображеннях. Основні результати, отримані при виконанні роботи такі:

- аналіз відомих додатків для ідентифікації об’єктів на зображеннях показав, що суттєвим недоліком цих додатків є відсутність можливості додавання власних об’єктів, тому розробка даного додатку є доцільною;

- розроблено модель та загальну структуру WEB-сервісу для ідентифікації об’єктів на зображеннях;

- подальшого розвитку отримав метод ідентифікації об’єктів на зображенні, у якому, на відміну від існуючих, використано клієнт-серверну архітектуру, що зменшує технічні вимоги до робочої станції, оскільки складні обчислення виконуються на сервері;

- розроблено алгоритми роботи серверної частини додатку;

- розроблено алгоритми роботи мобільного додатку;

- обрано тип інтерфейсу та розроблено графічний інтерфейс мобільного додатку для ідентифікації об’єктів;

- для реалізації додатку ідентифікації об’єктів обрано мову програмування JavaScript, платформу Node.js для реалізації серверної частини додатку, фреймворк ReactNative для розробки мобільного додатку та Google Cloud Vision API для розпізнавання об’єктів;

- мовою програмування JavaScript з використанням платформи Node.js розроблено серверну частину додатку для ідентифікації об’єктів;

- тестування WEB-сервісу для ідентифікації об’єктів з використанням засобу тестування Postman показало повну його працездатність та відповідність завданню на роботу;

- розроблену керівництво користувача, яке забезпечує експлуатацію та подальший розвиток додатку на умовах GNU General Public License.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. На порозі майбутнього: як штучний інтелект уже незабаром змінить ринок праці [Електронний ресурс]. – URL: https://www.radiosvoboda.org/a/28669670.html.
2. Cloud Vision documentation [Електронний ресурс]. – URL: https://cloud.google.com/vision/docs.
3. Грабарчук А. В. Ідентифікація об’єктів на основі Google Cloud Vision API / Л. Г. Коваль, В. П. Майданюк. [Електронний ресурс]. URL: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2022/paper/view/15845.
4. The Future of AI: How Artificial Intelligence Will Change the World [Електронний ресурс]. – URL: https://builtin.com/artificial-intelligence/artificial-intelligence-future
5. The way we train AI is fundamentally flawed [Електронний ресурс]. – URL: https://www.technologyreview.com/2020/11/18/1012234
6. Кращі програми для розпізнавання рослин та грибів [Електронний ресурс]. – URL: https://gsminfo.com.ua/51625-nazvano-krashhi-programy-dlya-rozpiznavannya-roslyn-grybiv-i-komah.html
7. UML-діаграма. Види діаграм UML [Електронний ресурс]. – URL: https://ukrpublic.com/aktualne/uml-diagrama-vidi-diagram-uml.html
8. Організація баз даних: практичний курс: Навч. посіб. для студ. / А. Ю. Берко, О. М. Верес; Нац. ун-т «Львів. політехніка». — Л., 2003. — 149 с. — Бібліогр.: 8 назв.
9. Database system concepts Silberschatz, Abraham; Sudarshan M.: New York: McGraw-Hill. S. 2011. 95 с.
10. How Google Tests Software Уітакер Д. Каролло Д. / Уітакер Д. Каролло Д. М.: Addison-Wesley Professional, 2012. 156 с.
11. Swagger documentation [Електронний ресурс]. – URL: https://swagger.io/tools/swagger-ui/
12. Postman documentation [Електронний ресурс]. – URL: https://www.postman.com/explore

ДОДАТОК А

(обов’язковий)

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп’ютерної інженерії

ЗАТВЕРДЖУЮ

д.т.н., проф. О. Н. Романюк

"25" березня 2022 р.

Технічне завдання

на бакалаврську дипломну роботу «Розробка Web–сервісу для ідентифікації об’єктів» за спеціальністю

121 – Інженерія програмного забезпечення

Керівник бакалаврської дипломної роботи:

д.т.н., проф. Л. Г.Коваль

"_25_"_____березня_____2022 р.

Виконав:

студент гр.2ПІ-18б А.В. Грабарчук

"__25__"____березня____2022 р.

Вінниця – 2022 року

1. Найменування та галузь застосування

Бакалаврська дипломна робота: «Розробка Web–сервісу для ідентифікації об’єктів». Галузь застосування - системи розпізнавання об’єктів.

2. Підстава для розробки.

Підставою для виконання бакалаврської дипломної роботи (БДР) є індивідуальне завдання на БДР та наказ №66 від «24» березня 2022 р. ректора ВНТУ про закріплення тем БДР.

3. Мета та призначення розробки

Метою роботи є покращення процесу взаємодії й ефективності при розпізнаванні об’єктів.

Основними задачами дослідження є:

- провести аналіз наявних методів і засобів розпізнавання об’єктів;
- розробка структури та алгоритмів роботи додатку для розпізнавання об’єктів;
- розробка інтерфейсу та коду програмного продукту;
- тестування розробленого програмного додатку.

4. Вихідні дані для проведення НДР

Перелік основних літературних джерел, на основі яких буде виконуватись БДР.

1. Cloud Vision documentation [Електронний ресурс]. – Режим доступу: – https://cloud.google.com/vision/docs

2. Організація баз даних: практичний курс: Навч. посіб. для студ. / А. Ю. Берко, О. М. Верес; Нац. ун-т «Львів. політехніка». — Л., 2003. — 149 с. — Бібліогр.: 8 назв.

3. How Google Tests Software Уітакер Д. Каролло Д. / Уітакер Д. Каролло Д. – Addison-Wesley Professional, 2012. – 156с

5. Технічні вимоги

База Даних – реляційна; архітектура – клієнт-серверна;; середовище розробки – WebStorm; мова програмування – JavaScript.

6. Конструктивні вимоги.

Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред'являється по закінченню робіт:

– пояснювальна записка до МКР;

– технічне завдання;

– лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки:

№ з/п	Назва етапів бакалаврської дипломної Роботи	Строк виконання етапів роботи
1	Обґрунтування методу розробки та постановки задачі дослідження	26.03.22 – 01.04.22
2	Розробка структури та алгоритмів Web–сервісу для ідентифікації об'єктів	02.04.22 – 07.04.22
3	Розробка інтерфейсу	08.04.22 – 11.04.22
4	Розробка коду Web–сервісу	12.04.22 – 10.05.22
5	Тестування Web–сервісу для ідентифікації об'єктів	11.05.22 – 28.05.22
6	Оформлення матеріалів до захисту БДР	01.06.22– 10.06.22

10. Порядок контролю та прийняття.

Виконання етапів бакалаврської дипломної роботи контролюється керівником згідно з графіком виконання роботи. Прийняття бакалаврської дипломної роботи здійснюється ДЕК згідно з графіком.

ДОДАТОК Б

(обов’язковий)

ПРОТОКОЛ
ПЕРЕВІРКИ БАКАЛАВРСЬКОЇ ДИПЛОМНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: «Розробка Web–сервісу для ідентифікації об’єктів»

Тип роботи: БДР

Підрозділ : кафедра програмного забезпечення, ФІТКІ

Науковий керівник:

Оригінальність	86,2 %
Схожість	13,8 %

Аналіз звіту подібності

■ **Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.**

□ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.

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

Особа, відповідальна за перевірку _____ Черноволик Г. О.

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck

Автор роботи _____ Грабарчук А.В.

Керівник роботи _____ Коваль Л.В.

ДОДАТОК В
(довідниковий).

Лістинг серверної частини додатку

```
const vision = require('@google-cloud/vision');
const {Storage} = require('@google-cloud/storage');
const googleConfig = require('../config/appConfig').googleVision;
const tables = require("../database/models");
const uuid = require("uuid");
const _ = require("lodash");

class VisionApiService {
  constructor() {
    this._savePlave = require('../config/appConfig').googleBucket.BUCKET_NAME;//"iatm-product-search";
    this._storage = new Storage();
    this._client = new vision.ProductSearchClient();
    this._imageAnotatorClient = new vision.ImageAnnotatorClient();
    this._projectId = googleConfig.projectId;
    this._location = googleConfig.location;
    this._bucket = this._storage.bucket(this._savePlave);
    this._locationPath = this._client.locationPath(this._projectId, this._location)

    this._productSetId = googleConfig.productSetId;
  }

  async createProductSet() {
```

```
        const productSetId = uuid.v4();
        const productSetDisplayName = 'grass';

        const productSet = {
          displayName: productSetDisplayName,
        };

        const request = {
          parent: this._locationPath,
          productSet: productSet,
          productSetId: productSetId,
        };

        const [createdProductSet] = await this._client.createProductSet(request);
        await tables.productSet.create({product_set_guid: productSetId});
        return await tables.productSet.findOne({where: {product_set_guid:
productSetId}});
      }

      async createProduct(name, product_guid, labels) {
        const product = {
          displayName: name,
          productCategory: googleConfig.productCategory,
          productLabels: labels || null
        };
        const request = {
          parent: this._locationPath,
          product: product,
          productId: product_guid
        };
```

```
        const [createdProduct] = await this._client.createProduct(request);
        await this.addProductToProductSet(product_guid);
    }

    async addProductToProductSet(product_guid) {
        const prodPath = this._client.productPath(this._projectId, this._location,
product_guid);
        const prodSetPath = this._client.productSetPath(
            this._projectId,
            this._location,
            this._productSetId
        );
        const request = {
            name: prodSetPath,
            product: prodPath
        }
        await this._client.addProductToProductSet(request);

    }

    async uploadToGCS(fileName, contentType, buffer) {
        const file = await this._bucket.file(fileName);
        await file.save(buffer, {
            metadata: {
                contentType
            },
        });
        return `gs://${this._savePlave}/${fileName}`;
    }
```

```
    async downloadFromGCS(name) {

        const file = await this._bucket.file(name);
        return (await file.download())[0];
    }

    async deleteFromGCS(file_name) {
        await (await this._bucket.file(file_name)).delete();
    }

    async addImageToProduct(productId, gcsUri, referenceImageId = null) {
        const formattedParent = this._client.productPath(this._projectId,
this._location, productId);


        const referenceImage = {
            uri: gcsUri,
        };

        const request = {
            parent: formattedParent,
            referenceImage: referenceImage,
            referenceImageId: referenceImageId,
        };

        await this._client.createReferenceImage(request);
    }

    async getAllProductsInSet() {
        const productSetPath = this._client.productSetPath(
```

```
            this._projectId,
            this._location,
            this._productSetId
        );
        const request = {
            name: productSetPath,
        };
        const [products] = await this._client.listProductsInProductSet(request);

        return products;
    }

    async getProductById(guid) {
        const productPath = this._client.productPath(this._projectId, this._location,
guid);
        const [product] = await this._client.getProduct({name: productPath});
        return product;
    }


    async lookForProd(imageBase64, filter = null) {
        filter = null; //TODO: refactor it
        const productSetPath = this._client.productSetPath(this._projectId,
this._location, this._productSetId);
        const request = {
            image: {content: imageBase64},
            features: [{type: 'PRODUCT_SEARCH'}],
            imageContext: {
                productSearchParams: {
                    productSet: productSetPath,
```

```
                productCategories: [googleConfig.productCategory],
                filter: filter,
            },
        },
    };
    const [responses] = await this._imageAnotatorClient.batchAnnotateImages({
        requests: [request],
    });
    if (!responses || !_.isArray(responses.responses) || !responses.responses.length)
        throw Error("Invalid image search response format");

    const response = responses.responses[0];
    if (response.error)
        throw Error(response.error.message || "Product search failed")

    const results = response.productSearchResults;
    return  _.map(results.productGroupedResults, (product) => {
        let arr = [];
        for (let i = 0; i < product.results.length; i++) {
            const productNameParts = product.results[i].product.name.split("/");
            arr.push({
                score: product.results[i].score,
                product_id: productNameParts[productNameParts.length - 1]
            });
        }
        return arr;
    });
}
```

```
    async removeProd(product_guid, images) {
        const prodPath = this._client.productPath(this._projectId, this._location, product_guid);
        await this._client.deleteProduct({name: prodPath});
        await images.forEach(image => {
            this.removeImg(product_guid, image.image_guid, image.file_name);
        });
    }

    async removeImg(image_guid, product_guid, file_name) {
        const formattedName = this._client.referenceImagePath(this._projectId, this._location, product_guid, image_guid);
        const request = {
            name: formattedName
        };
        await this._client.deleteReferenceImage(request);
        await this.deleteFromGCS(file_name);
    }
}

module.exports = new VisionApiService();

const tables = require("../database/models");
const googleService = require("../gateway/googleGateway");
const uuid = require("uuid");
const _ = require("lodash");
const ResponseError = require("../utils/ResponseError");
const THRESHOLD = require("../config/appConfig").settings.THRESHOLD;
const sharp = require('sharp');
```

```
module.exports = {
  async getCategories() {
    return _.map(await tables.categories.findAll(), "name");
  },

  async createCategory(name) {
    return await tables.categories.create({name});
  },

  async getCategory(name) {
    return await tables.categories.findOne({where: {name: name}});
  },

  async createProductSet() {
    return await googleService.createProductSet();
  },

  async getProductSets() {
    return await tables.productSet.findAll();
  },

  async createProduct(name, description, category) {
    const product_guid = uuid.v4();

    const categoryData = await module.exports.getCategory(category);
    const labels = [];
    if (categoryData) {
      labels.push({
        key: "category",
```

```
            value: categoryData.name
        });
    }

    await googleService.createProduct(name, product_guid, labels);
    await tables.products.create({
        product_guid,
        name,
        description,
        category_id: categoryData ? categoryData.category_id : null
    });
    return product_guid;
},

async addImage(product_guid, file, directory = "") {
    if (!file.mimetype || !file.mimetype.length)
        throw new ResponseError("Expected mime type", 400);

    const mimeParts = file.mimetype.split("/");
    if (!mimeParts || mimeParts.length < 2)
        throw new ResponseError(`Invalid mime type ${file.mimetype}`, 400);

    if (mimeParts[1] !== 'jpg' && mimeParts[1] !== 'jpeg') {
        file.buffer = await sharp(file.buffer).flatten({
            background: { r: 0xFF, g: 0xFF, b: 0xFF }
        }).toFormat('jpeg', {
            quality: 100,
            chromaSubsampling: '4:4:4',
            force: true,
        }).toBuffer();
```

```
            mimeParts[1] = "jpeg";
        }

        const image_guid = uuid.v4();
        const fileName = `${directory ? directory + "/" : ""}${image_guid}.${mimeParts[1]}`;

        const prod = await tables.products.findOne({where: {product_guid: product_guid}});
        if (!prod)
            throw new ResponseError("Product with this id does not exist", 404);

        const gsURI = await googleService.uploadToGCS(fileName, file.mimetype, file.buffer); // TODO: test it

        await googleService.addImageToProduct(product_guid, gsURI, image_guid);

        return await tables.images.create({
            image_guid: image_guid,
            gs_URI: gsURI,
            file_name: fileName,
            product_id: prod.product_id
        }, {returning: true});
    },

    createImageURL(protocol, host, product_id, image_guid) {
        return `${protocol || "http"}://${host || "localhost"}/product/${product_id}/image/${image_guid}`;
```

```
    },

    async getImage(product_guid, image_guid, size = null) {
      const product = tables.products.findOne({where: {product_guid: product_guid}});
      if (!product)
        throw new Error("Product with such id does not exist");

      const image = await tables.images.findOne({
        where: {
          image_guid: image_guid
        }
      });
      if (!image)
        return null;

      let img = await googleService.downloadFromGCS(image.file_name);

      // TODO: Not CPU efficient
      if(size)
        img = await sharp(img).resize(size.width, size.height).toBuffer();

      // console.log(img)
      console.log(img);
      return img;
    },

    async updateProduct(product_guid, name, description, categoryName) {
      const category = await tables.categories.findOne({where: {name: categoryName}});
```

```
            if(!category)
                throw new ResponseError("Wrong category name", 400);

            const product = await tables.products.findOne({where: {product_guid: product_guid}});
            if(!product)
                throw new ResponseError("Product not found", 404);
            product.name = name;
            product.description = description;
            product.category_id = category.category_id;
            await product.save();
            product.categoryName = category.name;

            return product;
        },

        async getProduct(protocol, host, product_guid) {
            const product = await tables.products.findOne({where: {product_guid}});
            if(!product)
                throw new ResponseError("Product not found", 404);
            return await module.exports._getFormattedProducts(protocol, host, [product]);
        },

        async getAllProductsInSet(protocol, host) { // TODO: currently return all products should be fixed
            const products = await tables.products.findAll();

            return await module.exports._getFormattedProducts(protocol, host, products);
```

```
    },
    async _getFormattedProducts(protocol, host, products) {
        return await Promise.all(_.map(products, async (prod) => {
            const images = await tables.images.findAll({
                where: {
                    product_id: prod.product_id
                }
            });

            const formattedProd = {product_guid: prod.product_guid,
                                   name: prod.name,
                                   description: prod.description};

            const category = await tables.categories.findOne({where: {category_id: prod.category_id}});

            formattedProd.categories = [];
            if(category)
                formattedProd.categories.push({key: "category", value: category.name});

            formattedProd.images = [];
            formattedProd.images = _.map(images, (img) => {
                return {
                    id: img.image_guid,
                    url: module.exports.createImageURL(protocol, host, formattedProd.product_guid, img.image_guid)
                };
            });
            return formattedProd;
```

```
        }));
    },

    async deleteProduct(product_guid) {

        const product = await tables.products.findOne({where: {product_guid: product_guid}});
        if (!product)
            throw new ResponseError("Product not found", 404);
        const images = await tables.images.findAll({where: {product_id: product.product_id}});

        await googleService.removeProd(product_guid, images);

        await tables.images.destroy({where: {product_id: product.product_id}});
        await tables.products.destroy({where: {product_guid: product_guid}});
    },

    async deleteImage(image_guid, product_guid) {
        const product = await tables.products.findOne({where: {product_guid: product_guid}});
        if (!product)
            throw new ResponseError("product not found", 404);

        const image = await tables.images.findOne({where: {image_guid: image_guid}});
        if (!image)
            throw new ResponseError("image not found", 404);
        await googleService.removeImg(image_guid, product_guid, image.file_name);
```

```
        tables.images.destroy({where: {image_guid: image_guid}});
    },

    async getThreshold() {
        const result = await tables.settings.findOne({where: {name:
THRESHOLD}});
        if(!result)
            throw new ResponseError("Threshold does not exist", 500);
        return result.value;
    },

    async setThreshold(value) {
        await tables.settings.update({value}, {
            where: {
                name: THRESHOLD
            }
        });
        return (await tables.settings.findOne({where: {
                name: THRESHOLD
            }}));
    }

    /*async PurgeProductsInProductSet() { // TODO: test it
        const products = await tables.products.findAll();
        await googleService.purgeProductsInProductSet();
        await products.forEach((product) => {
            googleService.createProduct(
                product.name,
                product.product_guid,
```

```
            tables.categories.findOne({where: {category_id: product.category_id}}));
            const images = tables.images.findAll({where: {product_id: product.product_id}});
            images.forEach(Promise.all(async (image) => {
              await googleService.addImageToProduct(product.product_guid, image.gs_URI, image.image_guid);
            }));
          });
        }*/
      }
```

ДОДАТОК Г

(обов’язковий)

ІЛЮСТРАТИВНА ЧАСТИНА

«РОЗРОБКА WEB–СЕРВІСУ ДЛЯ ІДЕНТИФІКАЦІЇ ОБ’ЄКТІВ»

Рисунок Г.1 – Титульна сторінка

МЕТА І ЗАДАЧІ РОБОТИ

Мета роботи:

Метою роботи є покращення процесу взаємодії й ефективності при розпізнаванні об'єктів.

Завдання:

- провести аналіз наявних методів і засобів розпізнавання об'єктів;
- розробка структури та алгоритмів роботи додатку для розпізнавання об'єктів;
- розробка інтерфейсу та коду програмного продукту;
- тестування розробленого програмного додатку.

Об'єкт дослідження – процес автоматичної ідентифікації об'єктів.

Предмет дослідження – методи та програмні засоби ідентифікації об'єктів на зображеннях.

Новизна отриманих результатів.

Подальшого розвитку отримав метод ідентифікації об'єктів на зображенні, у якому, на відміну від існуючих, використано клієнт-серверну архітектуру, що зменшує технічні вимоги до робочої станції, оскільки складні обчислення виконуються на сервері.

Рисунок Г.2 – Мета і задачі роботи

Компоненти Сервісу

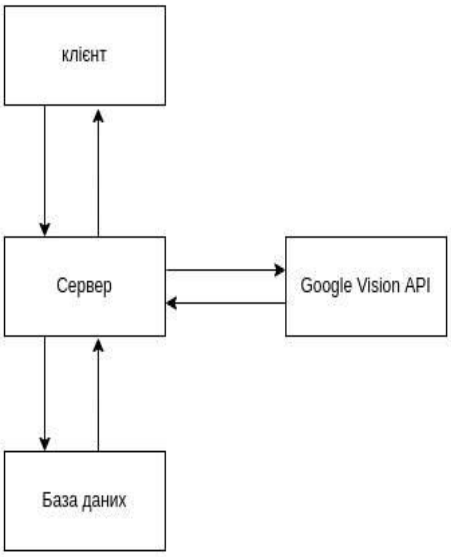

Рисунок Г.3 – Компоненти сервісу

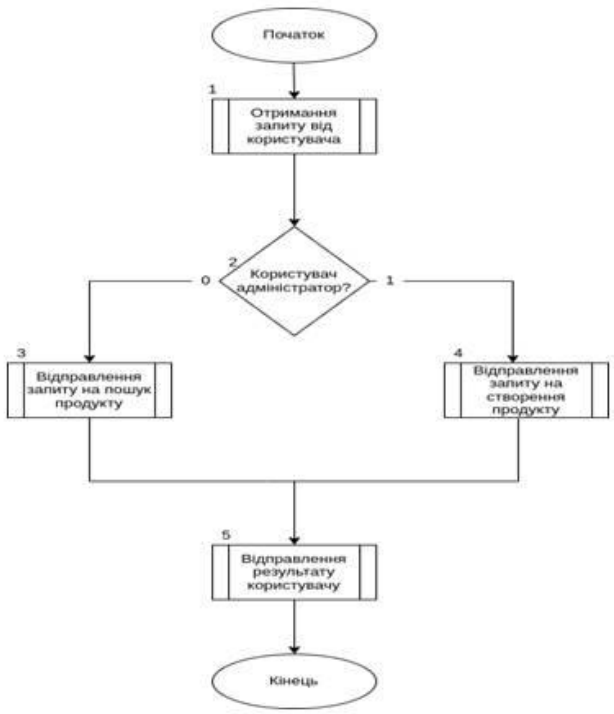

Рисунок Г.4 – Алгоритм роботи серверної частини

Рисунок Г.5 – Алгоритм роботи мобільної частини

Рисунок Г.6 – Структура бази даних

Дизайн мобільного додатку

Рисунок Г.7 – Дизайн мобільного додатку

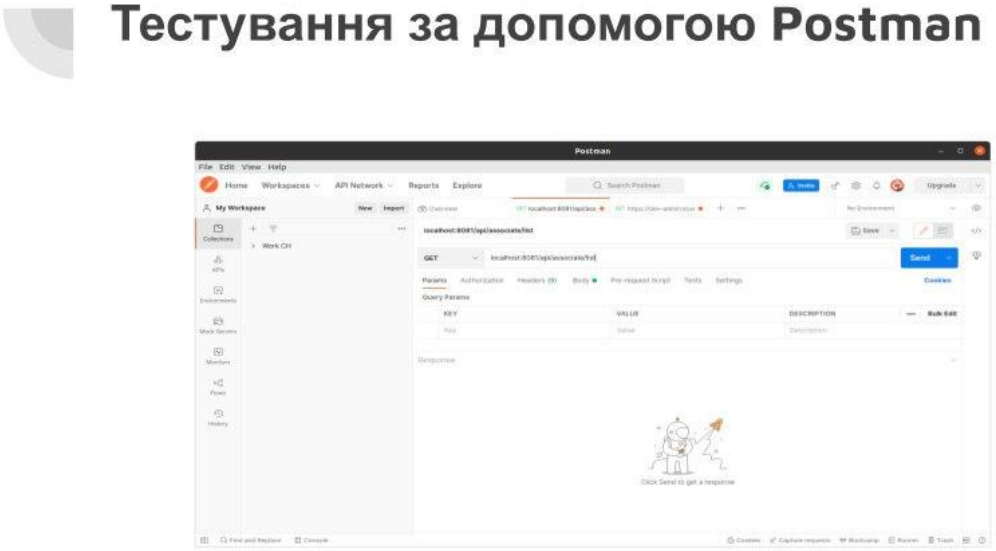

Рисунок Г.8 – Тестування за допомогою Postman

ВИСНОВКИ

- аналіз відомих додатків для ідентифікації об'єктів на зображеннях показав, що суттєвим недоліком цих додатків є відсутність можливості додавання власних об'єктів, тому розробка даного додатку є доцільною;
- розроблено модель та загальну структуру WEB-сервісу для ідентифікації об'єктів на зображеннях;
- подальшого розвитку отримав метод ідентифікації об'єктів на зображенні, у якому, на відміну від існуючих, використано клієнт-серверну архітектуру, що зменшує технічні вимоги до робочої станції, оскільки складні обчислення виконуються на сервері;
- розроблено алгоритми роботи серверної частини додатку;
- розроблено алгоритми роботи мобільного додатку;
- обрано тип інтерфейсу та розроблено графічний інтерфейс мобільного додатку для ідентифікації об'єктів;
- для реалізації додатку ідентифікації об'єктів обрано мову програмування JavaScript, платформу Node.js для реалізації серверної частини додатку, фреймворк ReactNative для розробки мобільного додатку та Google Cloud Vision API для розпізнавання об'єктів;
- мовою програмування JavaScript з використанням платформи Node.js розроблено серверну частину додатку для ідентифікації об'єктів;
- тестування WEB-сервісу для ідентифікації об'єктів з використанням засобу тестування Postman показало повну його працездатність та відповідність завданню на роботу;
- розроблену керівництво користувача, яке забезпечує експлуатацію та подальший розвиток додатку на умовах GNU General Public License.

Рисунок Г.9 – Висновки