

Міністерство освіти і науки України
Державний університет «Житомирська політехніка»
Факультет інформаційно-комп'ютерних технологій
Кафедра інженерії програмного забезпечення

ПОЯСНЮВАЛЬНА ЗАПИСКА

до випускної кваліфікаційної роботи магістра
на тему: «Мобільний додаток для розпізнавання тексту»

Виконав студент 2-го курсу, групи ІПЗм-20-1
спеціальності 121 «Інженерія програмного
забезпечення»

_____ М. І. Васишин

Керівник старший викладач кафедри ІПЗ

_____ С. М. Кравченко

Рецензент старший викладач кафедри КН

_____ Г. В. Марчук

ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
СПЕЦІАЛЬНІСТЬ 121 «ІНЖЕНЕРІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ»

ЗАТВЕРДЖУЮ

в. о. зав. кафедри інженерії
програмного забезпечення

к.т.н., доц. А.В. Морозов

« ____ » _____ 2021 р.

ЗАВДАННЯ

на магістерську кваліфікаційну роботу

Студента Василшина Максима Ігоровича

Тема роботи: Мобільний додаток для розпізнавання тексту

Затверджена Наказом університету від « ____ » _____ 2021 р. № _____

Термін здачі студентом закінченої роботи _____

Вихідні дані роботи (зазначається предмет і об'єкт дослідження)

Консультанти з магістерської кваліфікаційної роботи із зазначенням розділів,
що їх стосуються

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
1	Кравченко С. М.	14.06.2021	14.06.2021
2	Кравченко С. М.	3.09.2021	3.09.2021
3	Кравченко С. М.	25.09.2021	25.09.2021

Керівник _____
(підпис)

Календарний план

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Узгодження теми магістерської кваліфікаційної роботи	14.06.2021	Виконано
2	Формування предмету та об'єкту дослідження	14.06.2021	Виконано
3	Вивчення літературних джерел	14.06.2021-20.06.2021	Виконано
4	Аналіз існуючих аналогів	20.06.2021-15.07.2021	Виконано
5	Дослідження методів та підходів за тематикою магістерського дослідження	16.07.2021-28.08.2021	Виконано
6	Вибір інструментальних засобів	3.09.2021-13.09.2021	Виконано
7	Підготовка навчальних наборів даних	14.09.2021-6.10.2021	Виконано
8	Проектування архітектури мобільного додатку	7.10.2021-12.10.2021	Виконано
9	Налаштування серверної частини	13.10.2021-15.10.2021	Виконано
10	Написання алгоритмів	16.10.2021-17.10.2021	Виконано
11	Налаштування програмного додатку	18.10.2021-19.10.2021	Виконано
12	Тестування програмного додатку	20.10.2021-27.10.2021	Виконано
13	Написання пояснювальної записки	28.10.2021-30.11.2021	Виконано
14	Захист кваліфікаційної роботи	14.12.2021	Виконано

Студент _____

(підпис)

Керівник _____

(підпис)

РЕФЕРАТ

В структуру магістерської кваліфікаційної роботи входить мобільний додаток для розпізнавання емоцій людини та пояснювальна записка. Пояснювальна записка до випускної роботи містить 60 сторінок, 35 зображень, 1 таблицю.

Метою дипломної роботи є вивчення предметної області розпізнавання текстових символів в цифрових зображеннях з використанням допоміжного програмного забезпечення.

В кваліфікаційній роботі розглянуто предметну область, досліджено існуючі архітектури для класифікації розпізнавання символів, проаналізовано існуючі підходи та алгоритми до задачі розпізнавання символів, розібрано аналоги та зведено для них таблицю переваг та недоліків. Був розроблений мобільний додаток із серверною частиною який задовольняє всім визначеним вимогам.

Обґрунтовано вибір середовища програмування та інструментальних засобів. Проведено аналіз отриманих результатів, продемонстровано роботу мобільного додатку. Вихідний код розміщено на репозиторії Gitlab.

КЛЮЧОВІ СЛОВА: PYTHON, OCR, FLASK, OPENCV, REACT NATIVE, ДОДАТОК, КОМПОНЕНТ

ABSTRACT

The structure of the master's qualification work includes a mobile application for recognizing human emotions and an explanatory note. The explanatory note to the final work contains 60 pages, 35 images, 1 table.

The aim of the thesis is to study the subject area of text symbol recognition in digital images using auxiliary software.

In the qualification work the subject area is considered, the existing architectures for classification of character recognition are investigated, the existing approaches and algorithms to the problem of character recognition are analyzed, analogues are analyzed and the table of advantages and disadvantages is summarized for them. A mobile application with a server part has been developed that meets all the defined requirements.

The choice of programming environment and tools is substantiated. The analysis of the obtained results is carried out, the work of the web system is demonstrated. The source code is located in the Gitlab repository.

KEY WORDS: PYTHON, OCR, FLASK, OPENCV, REACT NATIVE, APPENDIX, COMPONENT

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ І ТЕРМІНІВ

OPT – оптичне розпізнавання тексту

OCR – optical character recognition

LSTM - це архітектура рекурентних нейронних мереж

OpenCV – бібліотека комп'ютерного зору з відкритим кодом;

Dataset – колекція однотипних даних, які застосовуються в задачах машинної обробки даних;

API – це набір готових класів, процедур, функцій, структур і констант, що надаються додатком для використання в зовнішніх програмних продуктах;

ЗМІСТ

РЕФЕРАТ	4
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ І ТЕРМІНІВ	6
ВСТУП	8
РОЗДІЛ 1. АНАЛІЗ ІСНУЮЧИХ ТЕХНОЛОГІЙ ДЛЯ РОЗПІЗНАВАННЯ ТЕКСТУ	10
1.1 Об'єкт та предмет дослідження	10
1.2 Аналіз аналогів програмного продукту	11
1.3 Вибір архітектури системи мобільного додатку	17
1.4. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення	20
1.5 Постановка задачі	26
Висновки до першого розділу.....	28
РОЗДІЛ 2. ПРОЕКТУВАННЯ МОБІЛЬНОГО ДОДАТКУ ДЛЯ РОЗПІЗНАВАННЯ ТЕКСТУ	29
2.1. Визначення варіантів використання системи та об'єктно-орієнтованої структури системи	29
2.2. Проблематика розпізнавання зображень.....	32
2.3. Характеристика набору даних та описи алгоритмів роботи програми	34
2.4. Реалізація функціоналу мобільного додатку	42
Висновки до другого розділу.....	51
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З СИСТЕМОЮ	52
3.1. Порядок встановлення та налаштування системи.....	52
3.2. Структура інтерфейсу клієнтського додатку	54
3.3. Тестування роботи мобільного додатку	61
Висновки до третього розділу	62
ВИСНОВКИ.....	63
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	65
ДОДАТКИ.....	66

ВСТУП

Актуальність теми. Пройшли часи всіх документів організацій, книг, бланків, довідок тощо. Зберігати тільки в паперовому або друкованому вигляді. В останньому десятилітті тому суспільство зіткнулося з проблемою перекладу, та оцифровки текстів для їх збереження у більш зручному цифровому форматі 21 століття. За останні 7-8 років за допомогою корпорацій як Google, Amazon, тощо. було досягнуто оцифровки більш ніж 15 мільйонів книг, і ця цифра зростає з кожним днем.

Зараз майже всі підприємства мають електронний документообіг, в бібліотеках не тільки можуть приймати друковані підручники, а й поступово почали видавати в електронному вигляді сертифікати державних установ. Все це допомагає дуже швидко та легко обробляти різноманітні документи, щоб вести детальну звітність у всіх сферах.

В такому разі необхідність переводити текст у електронний вид зростає з кожним днем. Але переводити всі документи в ручному порядку дуже складно та затратно, тому з'являється необхідність створювати все нові й нові методи обробки рукописного тексту в електронний, починаючи від чеків до керування складними апаратами на технологічних установах.

Метою випускної кваліфікаційної роботи є вивчення предметної області розпізнавання текстових символів в цифрових зображеннях з використанням допоміжного програмного забезпечення.

Визначимо наступні **завдання випускної роботи**:

- проаналізувати відомі підходи аналізу тексту;
- дослідити методи та підходи для символів на цифрових зображеннях;
- ознайомитися із існуючими аналогами програмних продуктів;
- спроектувати програмний додаток на основі поставленої задачі;
- досягти результатів за поставленими задачами та прослідити за їх успішним виконанням;

➤ провести тестування програмного додатку та оптимізацію у випадку невдалого тестування;

Об’єкт дослідження: Методи розпізнавання тексту, алгоритми машинного навчання, придатні для виявлення символів на зображеннях та виділення текстової інформації для подальшої її обробки користувачем.

Предметом дослідження є підходи, методи та алгоритми для реалізації програмного додатку із розпізнаванням символів на цифрових зображеннях.

Методи дослідження. Алгоритми та підходи для вирішення задач розпізнавання тексту з зображення.

Наукова новизна. Розроблена програма проста у використанні і задовольняє всі потреби користувача. Її завжди можна мати під рукою – відкривши на смартфоні. Зовнішній інтерфейс мінімалістичний, лаконічний та зрозумілий, який також може адаптуватись до різних розмірів екранів смартфонів. Таким чином ця система може бути використана для проєктів з низьким бюджетом.

Апробація результатів дослідження. Отримані результати магістерської кваліфікаційної роботи були оприлюднені на IV Всеукраїнська науково-практична інтернет-конференція здобувачів вищої освіти і молодих учених «Інформаційно-комп’ютерні технології: стан, досягнення та перспективи розвитку»

В структуру магістерської кваліфікаційної роботи входить три розділи в яких описаний повний процес створення мобільного додатку для розпізнавання тексту та пояснювальна записка. Пояснювальна записка до випускної роботи містить 60 сторінок, 35 зображень, 1 таблицю.

РОЗДІЛ 1. АНАЛІЗ ІСНУЮЧИХ ТЕХНОЛОГІЙ ДЛЯ РОЗПІЗНАВАННЯ ТЕКСТУ

1.1 Об'єкт та предмет дослідження

Оптичне розпізнавання тексту — це механічне або електронне переведення зображень рукописного, машинописного чи друкованого тексту в послідовності кодів, які пропонуються для представлення в текстовому редакторі. Розпізнавання широко використовується для конвертації книг і документів в електронний вигляд, для автоматизації систем обліку в бізнесі або для публікації тексту на веб-сторінці.

Системи оптичного розпізнавання тексту вимагають калібрування для роботи з конкретним шрифтом; у ранніх версіях, для програмування було обов'язковим для кожного символу, програма одночасно могла працювати лише з одним шрифтом. Зараз найпоширеніші так звані «інтелектуальні» системи, що розпізнають більшість шрифтів із високим ступенем точності. Деякі оптичні розпізнавання тексту можуть відновити вихідне форматування тексту, включаючи інші системи зображень, колонки й нетекстові компоненти.

Виявлення та розпізнавання тексту із зображень може мати безліч функціональних застосувань для аналізу документів, таких як допомога людям з вадами зору, розпізнавання транспортного засобу номерний знак, оцінка товару, що включає таблиці, дорожні знаки, карти, схеми, ключове слово дослідження зображень на основі, пошук документів, розпізнавання деталей у промисловому автоматизація, вилучення на основі вмісту, розпізнавання об'єктів, розташування блоку адреси, а також текст відео.

Крім того, нещодавно винайдений додаток — це додаток мобільного банкінгу, який пропонує банкам, що дозволяють клієнтам здійснювати операції навіть при відправленні зображення. Іншим подібним додатком є туристичний путівник, який дає можливість туристам розуміти табло, навіть якщо вони не знають мову місцевості, і структури перетворення тексту зображення допомагають незрячим людям, а також туристам. Кожен такий додаток залежить від структури вилучення текстової інформації, яка може грамотно виявляти,

локалізувати та отримувати текстову інформацію, наявну в природних зображеннях

Крім того, методи виявлення та розпізнавання тексту сцени можуть використовуватися для виявлення текстових орієнтирів, виявлення/ідентифікації посвідчення транспортного засобу та розпізнавання об'єктів на відміну від загального вилучення та індексація. Виявити, знайти та розпізнати текст сцени як може бути необмежену кількість можливих шрифтів, розмірів, кольорів і форм, роздільної здатності, складних фон, нерівномірне освітлення або розмиття через різне освітлення, складний рух і перетворення, незнайомий формат, затінення, а також відмінності в розмірі шрифту, стилі, вирівнювання та напрямку. Крім того, тексти можуть бути різними шрифтами. Витяг тексту із зображень або відео можна використовувати для їх анотування та індексування матеріалів.

1.2. Аналіз аналогів програмного продукту

Основною функцією розроблюваного додатку буде розпізнавання тексту, але аналогічні програми зазвичай мають певні додаткові функції, що позитивно впливає на враження від користування додатком. Також доречно звернути увагу на системні вимоги аналогічних додатків, можливість використання на різних платформах та структуру інтерфейсу. Для визначення всіх цих параметрів розглянемо його аналоги, наявні в офіційних магазинах додатків для користувачів смартфонів з різними операційними системами.

Першим додатком розглянемо Office Lens (Рисунок 1.1) – розпізнавач тексту, офіційно представлений компанією Microsoft.

За допомогою Office Lens можна легко фотографувати нотатки та інформацію на дошках та вивісках, а також у меню, записках та інших джерелах, що містять багато тексту. Вам більше не доведеться нашвидкуруч робити нотатки, які можуть загубитися, або покладатися на нечіткі зображення. У цьому

додатку також зручно фотографувати ескізи, креслення, рівняння та навіть малюнки без тексту.

В додатку є вбудована камера, а також можна завантажувати зображення з пристрою.

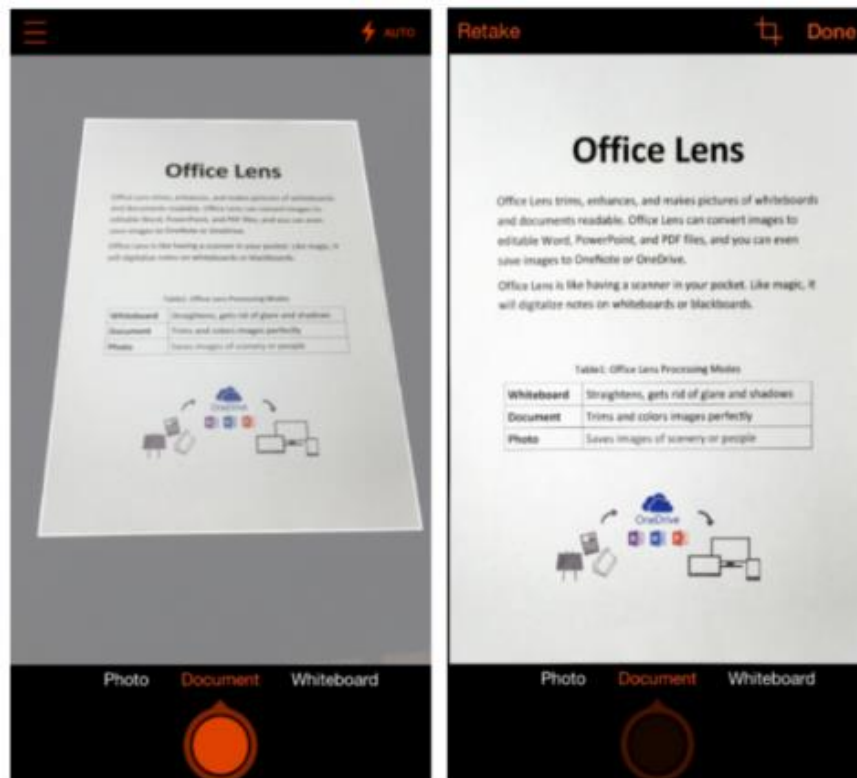


Рисунок 1.1 – вигляд додатку Office Lens

Системні вимоги:

- Android - версія 7.0 і вище;
- iOS – версія 14.0 і вище.

Основною перевагою даного додатку є можливість експорту у звичні користувачу формати для подальшого редагування, а саме .DOCX, .PDF або .PPTX. Також наявний прямий експорт в інші сервіси Microsoft, такі як OneNote та OneDrive (Рисунок 1.2). Також цей додаток можна використовувати на комп'ютерах з операційною системою Windows.

Перед збереженням фотографію можна відредагувати, щоб вона виглядала точно так, як ви хочете. Торкніться піктограми обрізки, щоб видалити непотрібні частини зображення. Також можна додати до фотографії підпис. Вона буде

використовуватися як замісний текст в OneNote і заголовок файлу в OneDrive. Коли ви завершите редагування, торкніться кнопки «Зберегти в нижній частині екрана». Якщо ви створили кілька зображень, всі вони з'являться у галереї.

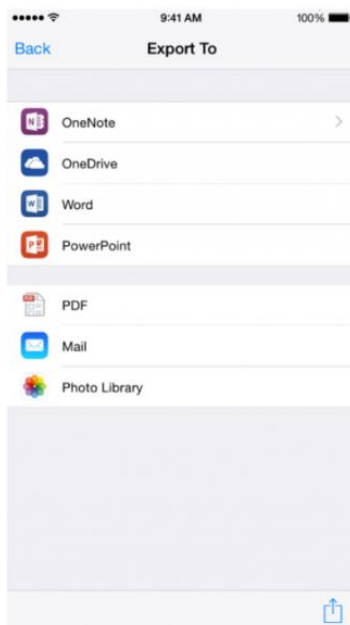


Рисунок 1.2 – експорт у різні формати в додатку Office Lens

До недоліків можна віднести неможливість виставити налаштування (наприклад, основний формат експорту) за замовчуванням, тож кожного разу після сканування тексту доводиться обирати, як саме його зберегти на пристрої або хмарних сховищах. Окрім цього, користувачі цього додатку скаржаться на проблеми в розпізнаванні кирилиці, тобто текстів українською або російською мовами. Також варто відмітити, що для користувачів iOS додаток доступний лише для найсвіжіших версій операційної системи.

Наступним розглянутим додатком є Text Fairy(Рисунок 1.3), представлена виключно для користувачів Android.

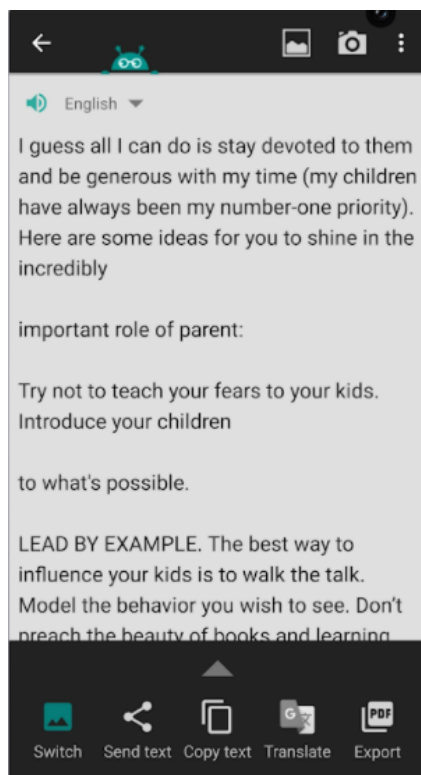


Рисунок 1.3 – вигляд додатку Text Fairy

Системні вимоги додатку: Android - версія 5.0 і вище.

Серед переваг даного додатку є його сумісність з пристроями на більш старих версіях Android. До додаткових функцій порівняно з аналогами можна віднести переклад тексту за допомогою Google Translate. Варто відмітити, що цей додаток має всі необхідні функції, а саме: швидке копіювання тексту, експорт в форматі PDF та велику кількість мов, що підтримуються для розпізнавання. Додаток має високу продуктивність і не перевантажений зайвими функціями, що позитивно впливає на враження від його використання користувачем.

До недоліків можна віднести те, що додаток створений лише для користування на смартфонах з операційною системою Android. Також в цьому додатку теж не надто якісно зчитується текст на кирилиці.

Третім додатком для аналізу є Adobe Scan, створений компанією Adobe. Особливістю даного додатку є комбінація функцій розпізнавання тексту та деяких функцій Adobe Acrobat, тобто цей програмний продукт зручно використовувати саме у випадках, якщо вихідний текст потрібно зберегти у не

просто у форматі PDF, а створити гарно відредагований якісний документ цього формату.

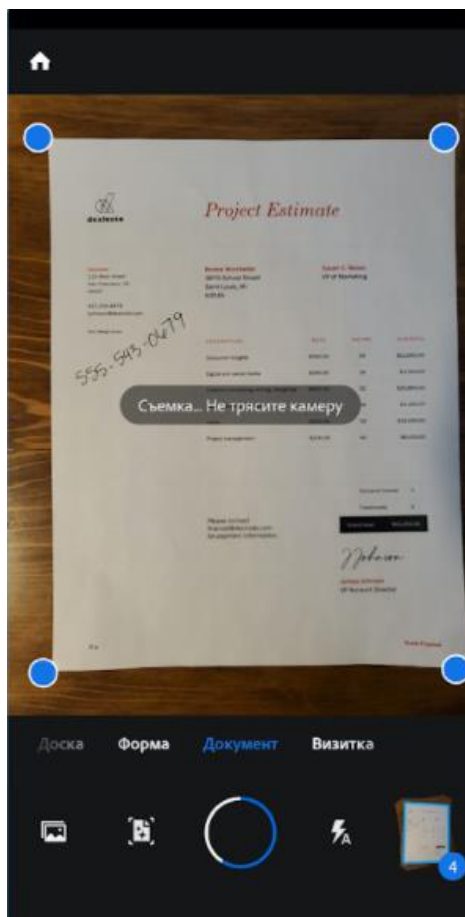


Рисунок 1.4 – вигляд додатку Adobe Scan

Системні вимоги:

- Android - версія 7.0 і вище;
- iOS – версія 13.0 і вище.

Перевагами використання цього додатку є широкий спектр можливостей редагування вихідного тексту, що перетворюється у PDF, наприклад виділення тексту кольорами, обрізка сторінок, накладення фільтрів, об'єднання створеного файлу з іншими PDF і так далі (Рисунок 1.4).

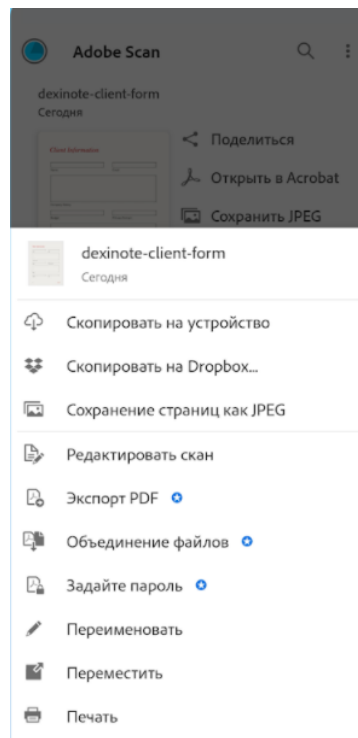


Рисунок 1.5 – можливості редагування в програмі Adobe Scan

До недоліків можна віднести те, що для більш широкого функціоналу додатку необхідна підписка. Вихідний формат тексту – тільки документ PDF або зображення, що може бути незручним, якщо користувачу потрібно просто швидко відсканувати і отримати текст. Даний додаток також погано розпізнає кирилицю.

Таблиця 1.1

Висновки аналізу аналогічних продуктів

	Office Lens	Text Fairy	Adobe Scan
Кроссплатформенність	Так	Ні	Частково
Можливість експорту тексту	Так	Так	Так
Збереження відсканованих текстів в додатку	Так	Так	Так

Швидке копіювання тексту після сканування	Так	Так	Ні
Якісне розпізнавання кирилиці	Ні	Ні	Ні
Розпізнавання рукописного тексту	Так	Ні	Частково

Проаналізувавши аналогічні програми, можна зробити висновок, що не варто перезавантажувати створюваний програмний продукт зайвими функціями, щоб він був універсальним і не обмежував користувача у можливостях збереження відсканованого тексту.

Доречно буде створити додаток сумісний як з смартфонами на iOS, так і на Android. Для кращого досвіду користування краще поєднати як швидке копіювання тексту, так і його експорт у зручному та популярному серед користувачів форматі. Також доречним буде попрацювати над розпізнаванням тексту саме українською і російською мовами, оскільки саме цієї функції не вистачає у вже створених розпізнавачах тексту.

1.3. Вибір архітектури системи мобільного додатку

Мобільний додаток передбачає одночасну участь декількох користувачів і потребує централізоване зберігання даних та їх поширення між різними користувачами. Саме тому, оптимальним рішенням для розробки становить клієнт-серверна архітектура.

Архітектура клієнт-сервер - архітектура комп'ютерної мережі, в якій багато клієнтів (віддалені процесори) запитують і отримують послугу від централізованого сервера (хост комп'ютера). Клієнтські комп'ютери забезпечують інтерфейс, що дозволяє користувачеві комп'ютера запитувати послуги сервера та відображати результати, які повертає сервер. Сервери чекають надходження запитів від клієнтів, а потім відповідають на них. В ідеалі сервер надає клієнтам стандартизований прозорий інтерфейс, щоб клієнтам не

потрібно було знати про особливості системи (тобто апаратного та програмного забезпечення), яка надає послугу.

Клієнти часто знаходяться на робочих станціях або на персональних комп'ютерах, тоді як сервери розташовані в інших місцях мережі, як правило, на більш потужних машинах. Ця обчислювальна модель особливо ефективна, коли клієнти і сервер мають різні завдання, які вони регулярно виконують. При обробці лікарняних даних, наприклад, на клієнтському комп'ютері може бути запущена прикладна програма для введення інформації про пацієнта, тоді як на серверному комп'ютері запущена інша програма, яка керує базою даних, в якій постійно зберігається інформація. Багато клієнтів можуть отримувати доступ до інформації сервера одночасно, і в той же час клієнтський комп'ютер може виконувати інші завдання, наприклад надсилати електронну пошту.

Оскільки і клієнтський, і серверний комп'ютери вважаються незалежними пристроями, модель клієнт-сервер повністю відрізняється від старої моделі мейнфрейма, в якій централізований мейнфрейм виконував усі завдання для пов'язаних з ним «німих» терміналів, які лише спілкувалися з центральним мейнфреймом.

На рисунку зображена схема архітектури клієнт-сервер та її основні компоненти.

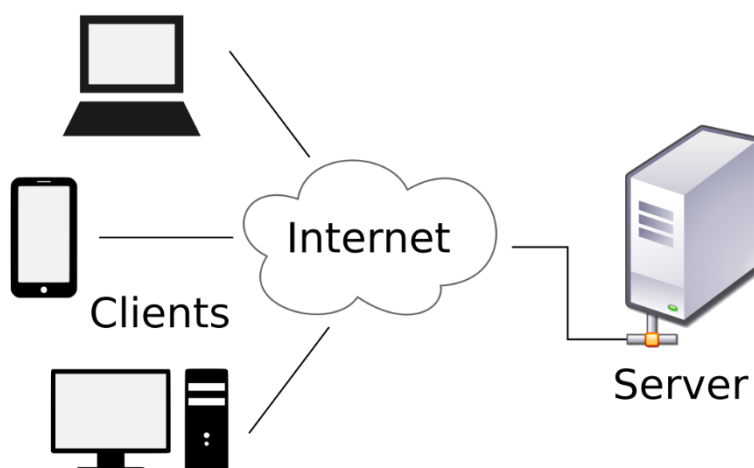


Рисунок 1.6 – Клієнт-серверна архітектура

У системі передбачено один клієнтський додаток для користувачів з різних пристроїв. Він є елементом системи, що надає користувачам здійснювати доступ до серверу на якому зберігається програмне забезпечення для розпізнання тексту.

В якості мережі використовується мережа інтернет, що забезпечує комунікацію між усіма архітектурними елементами.

Для створення універсального додатку було обрано інструмент React Native. На рисунку (номер), зображена структура компонентів з якої складається React Native.

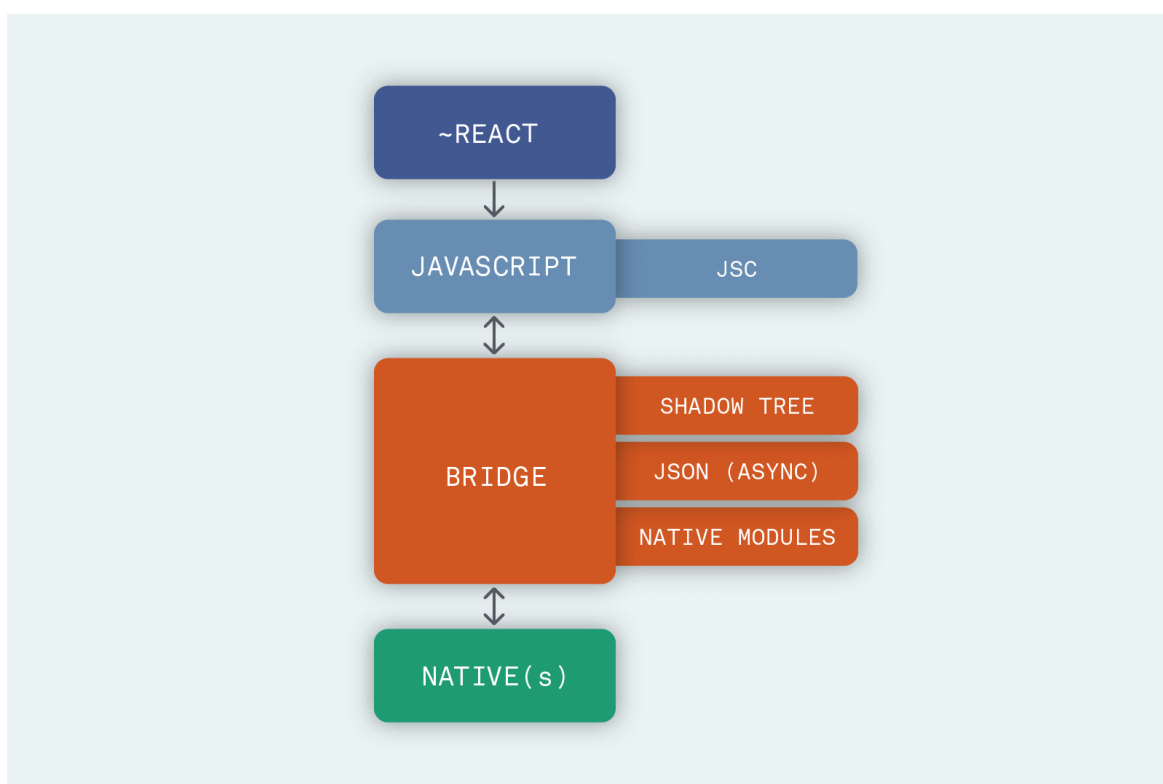


Рисунок 1.7 – Структура роботи React Native додатків

В системі є чотири основні компоненти:

- Кодова база, створена розробниками фреймворку, для розробки конкретних функцій додатку;
- Код Javascript – інтерпретований код React;
- Система «міст» - інтерфейс або ж зв'язок між Javascript кодом і API середовища виконання додатку на пристрої;

- Нативне середовище виконання пристрою (Android, IOS або ж веб-браузер)

Ключовим аспектом цієї архітектури є те, що компоненти не мають жодної інформації один про одного. Це значить, що вся комунікація буде відбуватися через асинхронні повідомлення, передані через «міст».

1.4. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення

Розглянемо вимоги до апаратного забезпечення для різних компонентів системи:

- Клієнтський додаток має працювати на мобільних системах, а саме OS Android та IOS, отже мінімальні вимоги будуть такими:

- Мобільні пристрої на базі операційної системи Android 8.0 Oreo та вище [1]. Мінімальний об'єм оперативної пам'яті пристрою 1Гб, від 2Гб внутрішньої пам'яті пристрою чи карти пам'яті, доступ до системи Інтернет;

- Мобільні пристрої на базі операційної системи Apple IOS 12 та вище. Мінімальний об'єм оперативної пам'яті пристрою 1Гб, від 2Гб внутрішньої пам'яті пристрою, доступ до системи Інтернет;

- Сервіс розпізнання зображення та серверна частина додатку в загальному має наступні вимоги до апаратного забезпечення:

- Встановлена операційна система Windows;
 - Підтримка мови Python версії 3.7 та вище;
 - Стабільний швидкісний доступ до Інтернету;
 - Оперативна пам'ять – не менше 8 гб;
 - Об'єм дискової пам'яті – не менше 128Гбайт;

Так як клієнтський додаток має бути доступним для декількох операційних систем, то на сьогоднішній день існує дуже велика кількість для створення універсальних додатків для різних систем. Розглянемо можливі варіанти серед існуючих інструментів та проведемо вибір потрібного нам інструментарію.

На сьогоднішній день існуює велика кількість різноманітних технологій для розробки веб-додатків, та мобільних додатків. Ці технології можна поділити на 2 велика категорії:

- Нативні додатки (native apps) – додатки, що розроблюються під конкретну систему (Windows, Android, IOS тощо).

Для розробки нативних додатків можна використовувати такі мови та технології:

- Мови програмування Java та Kotlin для розробки під операційну систему Android.

- Мови програмування Objective-C та Swift для розробки додатків під Apple IOS.

- Flutter. Інструментів для розробки нативних додатків під мобільні пристрої на базі систем Android та IOS. Для написання програм використовується мова Dart.

- React Native. Інструмент для розробки нативних додатків під мобільні пристрої, а також веб-додатків для різних браузерів. Розробка відбувається за допомогою мови програмування JavaScript та бібліотеки React JS.

- Гібридні додатки – додатки, що розроблюються за допомогою інструментів веб програмування HTML, JavaScript, CSS та додаткових бібліотек.

Для розробки гібридних додатків є ряд технологій:

- Electron;

- PhoneGap;

- Ionic;

Гібридні додатки працюють за принципом пакування додатку під звичайний мобільний додаток, а відображення всієї інформації відбувається як веб-сторінка – WebView в контейнері для операційної системи.

Перевагою гібридних технологій є простота розробки, так як програмний код не сильно відрізняється від звичайного написання додатку на мові JavaScript. Але великим недоліком є те, що така простота впливає на швидкодію системи, та її якість роботи.

Серед вищеперерахованих технологій найбільш оптимальною є технологія React Native. Кодова база універсальна для Android та IOS, тому швидкість розробленого додатку буде більш високою ніж розробка трьох окремих додатків під різні операційні системи. Мова програмування JavaScript та бібліотека ReactJS є більш популярною ніж мова Dart та технологія Flutter.

React Native — це кросплатформовий фреймворк з відкритим вихідним кодом для розробки нативних мобільних та настільних програм на JavaScript та TypeScript, створений Facebook, Inc. React Native підтримує такі платформи як Android, Android TV, iOS, macOS, Apple tvOS, Web, Windows та UWP, дозволяючи розробникам використовувати можливості бібліотеки React поза браузером для створення нативних програм, що мають повний доступ до системних API платформ.

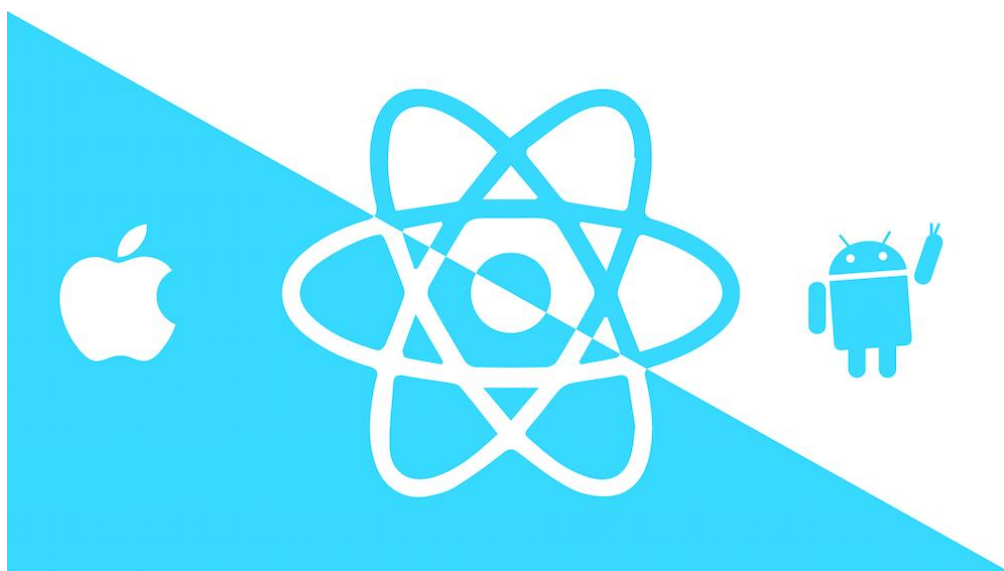


Рисунок 1.8 – React Native

Основні принципи роботи React Native практично ідентичні принципам роботи React, крім того, що React Native управляє не браузерної DOM, а платформними інтерфейсними компонентами. JavaScript-код, написаний розробником, виконується у фоновому потоці, та взаємодіє з платформними API через асинхронну систему обміну даними, звану Bridge. У 2021 році очікується заміна Bridge на продуктивнішу синхронну модель обміну даними, що підтримує парадигму zero-copy.

Хоча система стилів (спосіб конфігурації візуальних властивостей елементів інтерфейсу) React Native має синтаксис, схожий на CSS, фреймворк не використовує технології HTML чи CSS як такі. Натомість для кожної з операційних систем, що підтримуються фреймворком, реалізовані програмні адаптери, що застосовують заданий розробником стиль до платформного інтерфейсного елементу.

React Native також дозволяє розробникам використовувати вже існуючий код, написаний іншими мовами програмування - наприклад, Java або Kotlin для Android та Objective-C або Swift для iOS. Також React Native підтримує інтеграцію в вже існуючі програми - наприклад, частина інтерфейсу мобільного додатка може бути реалізована на React Native, а частина - за допомогою суто платформних засобів.

Існує кілька причин глобального успіху React Native.

По-перше, за допомогою React Native компанії можуть створювати код лише один раз і використовувати його для роботи своїх програм для iOS і Android. Це означає величезну економію часу та ресурсів.

По-друге, React Native був побудований на основі React – бібліотеки JavaScript, яка вже була надзвичайно популярна, коли вийшов мобільний фреймворк. У цьому розділі ми детально обговорюємо відмінності між React і React Native.

По-третє, фреймворк давав змогу розробникам інтерфейсу, які раніше могли працювати лише з веб-технологіями, створювати надійні, готові до виробництва програми для мобільних платформ.

Окрім додатку клієнта, потрібно розробити сервіс для обробки зображення. Для цього були обрані мова програмування Python, та додаток Tesseract.

Tesseract - безкоштовна комп'ютерна програма для розпізнавання тексту, розроблена Hewlett-Packard з середини 1980-х до середини 1990-х років і потім «пролежала на полиці» десять років. У серпні 2006 року Google купив його і випустив під ліцензією Apache 2.0, щоб продовжити розробку. На даний момент

програма вже працює з UTF-8, мовна підтримка (в тому числі російський з версії 3.0) здійснюється за допомогою додаткових модулів.

Ядро програми Tesseract було розроблено в лабораторіях Hewlett Packard Bristol і Hewlett Packard Co., Greeley Colorado, 1985-1994 pp. У 1996 році були внесені істотні зміни і підготовлений порт для Windows. Потім, з 1998 року, частковий перехід з C на C++. Велика частина коду спочатку була написана на C, але були внесені поліпшення в сумісність з компіляторами C++.

```
@geo /tmp $ tesseract
Usage:
  tesseract imagename|stdin outputbase|stdout [options...] [configfile...]

OCR options:
  --tessdata-dir /path    specify location of tessdata path
  -l lang[+lang]          specify language(s) used for OCR
  -c configvar=value       set value for control parameter.
                           Multiple -c arguments are allowed.
  -psm pagesegmode         specify page segmentation mode.
These options must occur before any configfile.

pagesegmode values are:
  0 = Orientation and script detection (OSD) only.
  1 = Automatic page segmentation with OSD.
  2 = Automatic page segmentation, but no OSD, or OCR
  3 = Fully automatic page segmentation, but no OSD. (Default)
  4 = Assume a single column of text of variable sizes.
  5 = Assume a single uniform block of vertically aligned text.
  6 = Assume a single uniform block of text.
  7 = Treat the image as a single text line.
  8 = Treat the image as a single word.
  9 = Treat the image as a single word in a circle.
  10 = Treat the image as a single character.

Single options:
  -v --version: version info
  --list-langs: list available languages for tesseract engine. Can be used with
  --tessdata-dir.
  --print-parameters: print tesseract parameters to the stdout.
@geo /tmp $ tesseract -v
tesseract 3.03
leptonica-1.71
  libgif 5.1.0 : libjpeg 8d : libpng 1.6.13 : libtiff 4.0.3 : zlib 1.2.8 : libwe
bp 0.4.1
```

Рисунок 1.9 – Головне вікно програми Tesseract

Як і всі інші ліцензії на безкоштовне програмне забезпечення, ліцензія Apache дає користувачеві право використовувати програмне забезпечення для будь-яких цілей, а також вільно змінювати і поширювати змінені копії в доповнення до назви.

Ця ліцензія не наказує незмінність ліцензії на поширення програмного забезпечення і навіть не вимагає збереження її безкоштовного і відкритого статусу. Єдина умова, що накладається ліцензією Apache, - інформувати одержувача про використання вихідного коду. На відміну від ліцензій з

авторським лівому, одержувач зміненої версії не обов'язково має всі права, в першу чергу, з ліцензією Apache

Python - інтерпретована об'єктно-орієнтована мова програмування високого рівня зі строгою динамічною типізацією. Розроблена у 1990 році Гвідо ван Россумом. Структури даних високого рівня разом із динамічною семантикою та динамічним зв'язуванням роблять її привабливою для швидкої розробки програм, а також як засіб поєднування наявних компонентів. 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. У мові програмування Python підтримується кілька парадигм програмування, зокрема: об'єктно-орієнтована, процедурна, функціональна та аспектно-орієнтована.

Основні бібліотеки що були використані для роботи алгоритму обробки зображення:

- OpenCV – бібліотека для розпізнавання зображень.
- Numpy – бібліотека для роботи з масивами.
- PyTesseract – бібліотека обгортки для доступу до Tesseract.

Алгоритм обробки зображення має знаходитись на сервері Python, тому потрібно налаштувати комунікацію між клієнтським додатком та сервером. Для цього використаємо веб фреймворк Flask.

Flask — мікрофреймворк для веб-додатків, створених за допомогою Python. Його основу складає інструментарій Werkzeug та рушій шаблонів Jinja2. Поширюється відповідно до умов ліцензії BSD.

Flask називається мікрофреймворком, оскільки він не вимагає спеціальних засобів чи бібліотек. У ньому відсутній рівень абстракції для роботи з базою даних, перевірки форм або інші компоненти, які надають широковживані функції за допомогою сторонніх бібліотек. Однак, Flask має підтримку розширень, які надають додаткові властивості таким чином, як вони були доступні у Flask із самого початку. Існують розширення для встановлення об'єктно-реляційних зв'язків, перевірки форм, контролю процесу завантаження, підтримки

різноманітних відкритих технологій аутентифікації та декількох поширених засобів для фреймворку. Розширення оновлюються частіше ніж базовий код.

Для самої програми в якості розпізнавання зображень використаємо програму Tesseract та бібліотеку Pytesseract в якості обгортки над нею.

Tesseract — вільна програма для розпізнавання текстів, що розроблялася Hewlett-Packard з 1985 до 1994 року, а в наступне десятиріччя залишалася практично без змін. Нещодавно Google купив її та відкрив початковий код під ліцензією Apache 2.0 у 2006 році. для продовження розробки. Зараз програма вже працює з UTF-8, розпізнає багато мов, серед яких і українська.

Для збереження перекладених текстів та інформації про користувачів використаємо додаток Firebase.

Firebase надає в режимі реального часу базу даних та бекенд як службу. Ця служба надає розробникам застосунків API, який дозволяє синхронізувати дані застосунків між клієнтами та зберігати їх у хмарі Firebase. Компанія також надає клієнтські бібліотеки, які дозволяють інтеграцію із застосуванням Android, iOS, JavaScript / Node.js, Java, Objective-C, Swift. База даних також доступна через REST API та прив'язки до декількох сценаріїв JavaScript, таких як AngularJS, React, Ember.js та Backbone.js. REST API використовує протокол подій із сервером, який є інтерфейсом для створення HTTP-з'єднань для отримання push-повідомлень від сервера. Розробники, які використовують Realtime Database, можуть захищати свої дані за допомогою правил безпеки, що застосовуються на сервері.

Cloud Firestore, яка є наступною генерацією Firebase Realtime Database, була випущена у бета-версії.

1.4 Постановка задачі

Виявлення та розпізнавання тексту включає в себе дуже делікатну процедуру, яка визначає число різноманітні шрифти, контрасти, розміри шрифтів, вирівнювання та кольори. Кожен з цих елементів робить важко встановити єдине рішення. Додатковим викликом є той факт, що люди є

розглядається як еталон для визначення того, чи функціонує алгоритм подібним чином для наших очей і мозку, оскільки іноді виникають проблеми, пов'язані з виявленням і розпізнаванням тексту у відео та зображеннях.

Основною метою є розробка мобільного додатку за допомогою якого користувач зможе перетворити текст із цифрового зображення, у розпізнаний текст, який потім зможе скопіювати та відправити в тій чи інший ресурс для подальшої роботи з ним.

В випускній роботі необхідно розробити мобільний додаток для смартфона із серверним програмним забезпеченням для отримання та розпізнавання тексту, та відправки запитів із додатку та на сервер.

Для реалізації поставленої задачі основними етапами розробки є:

1. Проаналізувати більшість існуючих додатків зі схожою тематикою
2. Спроекувати загальну концепцію класів та методів розробки для створення програмного продукту
3. Розробити дизайн користувацького інтерфейсу
 - 3.1 Створити загальну концепцію користувацького інтерфейсу.
 - 3.2 Створити елементи керування у вигляді графічних компонентів (кнопки,панелі, іконки)
 - 3.3 Розробити графічний інтерфейс
 - 3.3.1 Головний екран входу
 - 3.3.2 Екран із функціоналом розпізнання зображення
 - 3.3.3 Екран із розпізнаними запитами
4. Проектування та розробка основних програмних модулів
 - 4.1 Модуль інтерфейсу мобільного додатку
 - 4.2 Модуль управління камерою
 - 4.3 Модуль управління запитами з мобільного пристрою
 - 4.4 Модуль для розпізнання тексту
5. Тестування створеного програмного продукту

Результатом реалізації поставленого завдання є мобільний додаток який завдяки запитам на сервер буде розпізнавати текст із цифрового зображення.

Висновки до першого розділу

У першому розділі було розглянуто існуючі технології для виконання кваліфікаційної роботи. Було визначено основні цілі, проблеми, мету, а також проведено аналіз існуючого програмного забезпечення з порівняльними характеристиками.

Аналіз показав, що існує велика кількість схожого програмного забезпечення, що дозволить більш точніше потсавити онсонві цілі для нашої задачі, та вдосконалити недоліки існуючих аналогів.

Відповідно до встановленого завдання, була обрана оптимальна архітктура системи, яка допоможе реалізувати необхідний програмний продукт. На основі визначених вимог додатку, були проаналізовані можливі засоби реалізації даного програмного продукту. За результатами порівняння були обрані необхідні інструменти та технології які в тій чи інішій мірі задовольняють цілям програмного продукту.

РОЗДІЛ 2. ПРОЕКТУВАННЯ МОБІЛЬНОГО ДОДАТКУ ДЛЯ РОЗПІЗНАВАННЯ ТЕКСТУ

2.1. Визначення варіантів використання системи та об'єктно-орієнтованої структури системи

Для проведення проектування системи спершу необхідно визначити варіанти використання користувачами та скласти переліг певних вимог до системи.

Нефункціональні вимоги:

1. *Сприйняття:*

- Час, потрібний для навчання інструментами роботи з інформаційною системою для звичайних користувачів – 30 хвилин, а для досвідчених – 15 хвилин;

- Час відповіді системи для звичайних запитів не повинен перевищувати 5 секунд, а для більш складних запитів – 10 секунд;

- Інтерфейс представлення додатку повинен бути інтуїтивно зручним для користувача та не вимагати від нього додаткової підготовки;

2. *Надійність:*

- Доступність – час, потрібний для обслуговування системи не повинен перевищувати 10% від загального часу роботи;

3. *Продуктивність:*

- Система повинна підтримувати мінімум 100 одночасно працюючих користувачів, пов'язаних з спільною базою даних;

4. *Можливість експлуатації:*

- Масштабування – система повинна мати можливість збільшувати потужності (продуктивність), зі збільшенням користувачів таким чином, щоб це аж ніяк негативно не відобразилося на її роботі;

- Оновлення версій – оновлення версій повинно здійснюватися в ручному режимі залежно від вподобань користувачів та розширення списку запропонованих продуктів;

5. Системні вимоги:

- Вимоги до середовища виконання: телефон на операційній системі Android версії не нижче 7.0, та IOS не нижче версії 10.0;
- Наявність підключення до інтернету за різними вимогами (3G, WI-FI, LTE);

6. Бізнес правила:

- Система повинна відповідати всім стандартам інтерфейса користувача Android смартфону;

Також в цій системі присутній клієнт у якого є наступні можливості.

1. Створення зображення для розпізнання
2. Вибір готового зображення для розпізнання
3. Вибір мови для зображення
4. Відправка зображення на сервер
5. Копіювання отриманого тексту із розпізнаного зображення
6. Створення pdf файлу із отриманого тексту
7. Видалення розпізнаного тексту

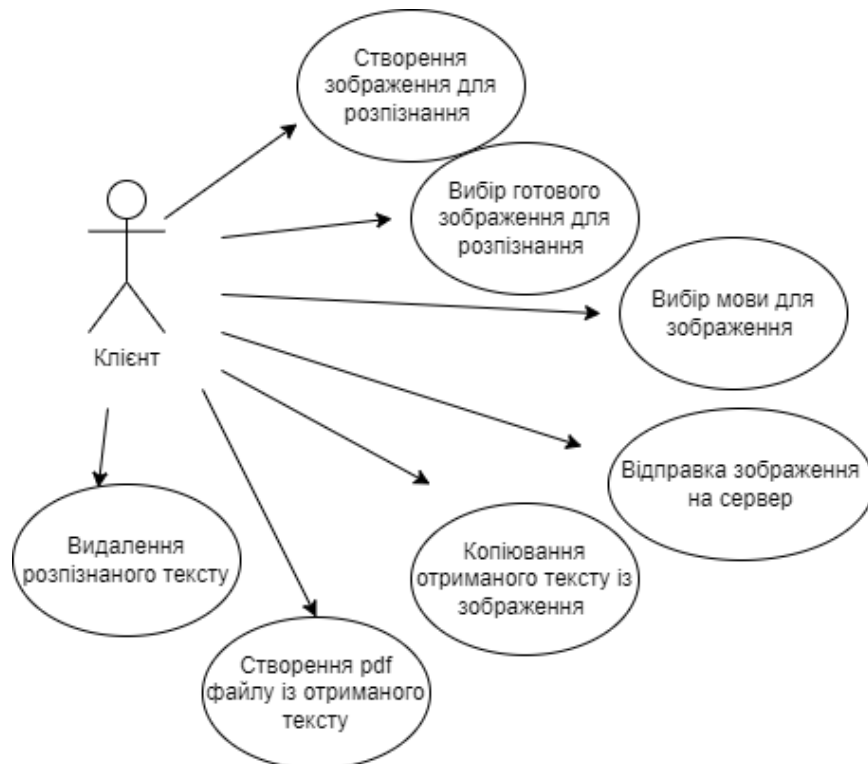


Рисунок 2.1 – Діаграма варіантів використання

В системі присутній лише один актор – клієнт, роль якого полягає у використанні всіх програмних функцій.

Клієнт – зареєстрований та автентифікований користувач, що має доступ до перегляду даних системи, основних та додаткових функцій додатку;

На рисунку 2.1 зображена діаграма варіантів використання систем, на якій схематично показані основні дії та актор.

Наступна діаграма показує нам як взагалі буде працювати додаток під виглядом звичайних дій користувача.

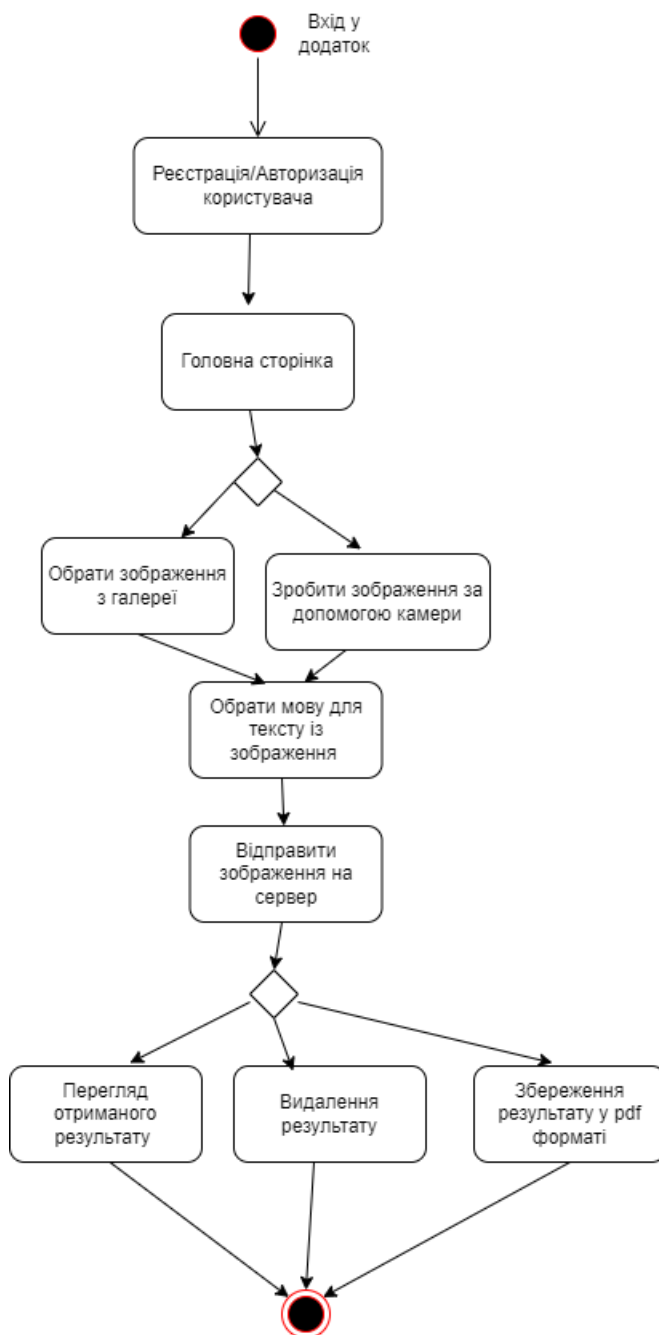


Рисунок 2.2 - Діаграма діяльності користувача

Розглянемо послідовні кроки користувача за діаграмою:

- Вхід у додаток – користувач запускає додаток на вибраній ОС;
- Реєстрація/Авторизація – користувач лише чекає пару секунд для того щоб увійти в систему;
- Головна сторінка – користувач отримає доступ до головної сторінки додатку
- Обрати зображення з галереї – користувачеві пропонується обрати зображення з галереї.
- Зробити зображення за допомогою камери – користувачеві надходить запит від додатку до доступу до камери пристрою
- Обрати мову для тексту із зображення – користувачеві пропонується обрати одну із наявних мов для відображення
- Відправити зображення на сервер – користувач відправляє зображення із текстом для розпізнання на сервер
- Перегляд отриманого результату – після того як прийде відповідь на запит клієнта він хомже переглянути отриманий результат
- Видалення результату – після того як користувач перегляне текст він може вирішити, лишити результат, чи видалити його
- Збереження у форматі pdf – користувач зможе зберегти тей чи інший текст у форматі pdf для подальшого його використання.

2.2 Проблематика розпізнавання зображень

Виявлення тексту – це проблема пошуку тексту в зображенні. Надано будь-яке зображення, текст детектор повинен повертати всі обмежувальні рамки всього тексту в зображенні. Жодних припущень щодо розміру, напрямку чи положення тексту.

У зображеннях природних сцен виявлення та розпізнавання тексту є складним завданням з тих же причин, що й ідентифікація тексту, яка включає такі фактори, як текстурований фон, різницю в тексті через умови освітлення, деформацію через перспективу та зображення зниженої якості.

Тому в природних сценах виявлення та розпізнавання текстів є досить складним завданням.

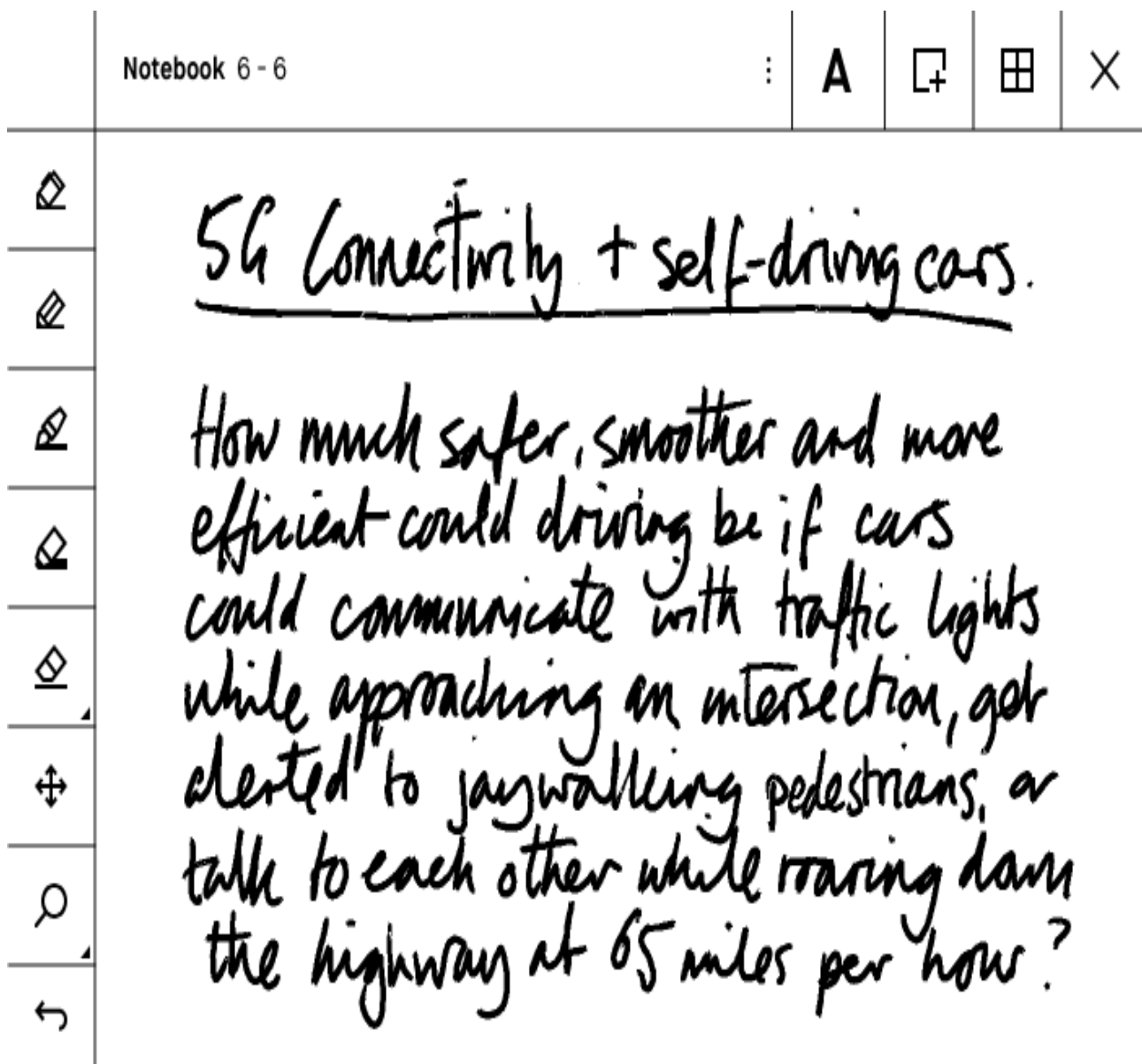


Рисунок 2.3 – Приклад важко розпізнаного тексту

Ці складні завдання пов'язані з властивостями тексту на зображенні. Основні проблеми для виявлення та розпізнавання тексту сцени можна грубо розділити на наступні типи:

- Різноманітність тексту сцени: текст у зображеннях документів зазвичай відображається звичайним шрифтом, одноколірний і однорідний макет. Навпаки, тексти з природних сцен можуть мати абсолютно різні шрифти, кольори, орієнтації та масштаби навіть в одній сцені. Символи з різних шрифтів

можуть включати великі відмінності всередині класу і можуть створювати численні підпростіри послідовності, що ускладнює точне розпізнавання.

- Співвідношення сторін: текст має різні співвідношення сторін у природному зображенні. Наприклад, дорожні знаки можуть бути короткими, але інший текст, наприклад підписи до відео, може бути значно довшим. Виявлення таких типів текстів вимагає процедури з високою обчислювальною складністю, оскільки в процесі пошуку необхідно враховувати розташування, масштаб і довжину.

- Багатомовне середовище: незважаючи на те, що більшість латинських мов мають десятки символів, інші мови включають тисячі категорій символів, таких як китайська, японська та корейська (CJK). Арабські символи змінюють форму відповідно до контексту, оскільки вони мають пов'язані символи. Хінді об'єднує букви алфавіту в тисячі фігур, які означають склади. У багатомовних налаштуваннях OCR у відсканованих документах все ще є проблемою для дослідження, хоча ідентифікація тексту в складних зображеннях є додатковою проблемою.

Що стосується текстових зображень сцени, то підвищення рівня розпізнавання тексту більш ніж на 50% може бути досягнуто завдяки покращенню визначення тексту таким чином, виявлення тексту є важливим для пошуку текстової інформації

2.3. Характеристика набору даних та описи алгоритмів роботи програми

В загальному в даному програмному продукті буде викристовуватися перелік готових даних які вже мають своє розширення, та натреновані під ті чи інші шрифти, та мови.

Набір даних — це набір даних на обрану тематику. У разі табличних даних набір даних відповідає одній або кільком таблицям бази даних, де кожен стовпець таблиці представляє певну змінну, а кожен рядок відповідає заданому запису відповідного набору даних. Набір даних перераховує значення для кожної зі змінних, таких як висота і вага об'єкта, для кожного члена набору даних. Набори даних також можуть складатися з колекції документів або файлів.

У дисципліні відкритих даних набір даних — це одиниця вимірювання інформації, опублікованої в загальнодоступному сховищі відкритих даних.

Але для найбільш якісного результату в даному випадку ми можемо натренувати свої дані, та отримати результат у вигляді файлу з розширенням .traineddata, який в свою чергу є зрозумілим форматом для програми Tesseract за допомогою якої і буде виконуватися основний функціонал програми, а саме розпізнавання тексту.





















 Abstracting.traineddata	04.12.2021 0:29	Файл "TRAINEDD...	493 КБ
 AC.traineddata	04.12.2021 0:29	Файл "TRAINEDD...	365 КБ
 Acki.traineddata	04.12.2021 0:29	Файл "TRAINEDD...	360 КБ
 Ackident.traineddata	04.12.2021 0:29	Файл "TRAINEDD...	498 КБ
 adam.traineddata	04.12.2021 0:29	Файл "TRAINEDD...	372 КБ
 ADAMSFONT.traineddata	04.12.2021 0:29	Файл "TRAINEDD...	367 КБ
 Adelyne.traineddata	04.12.2021 0:29	Файл "TRAINEDD...	365 КБ
 Adenote.traineddata	04.12.2021 0:29	Файл "TRAINEDD...	358 КБ
 adhie.traineddata	04.12.2021 0:29	Файл "TRAINEDD...	365 КБ
 Adoulliss.traineddata	04.12.2021 0:29	Файл "TRAINEDD...	372 КБ
 advertisingphilfont.traineddata	04.12.2021 0:29	Файл "TRAINEDD...	367 КБ
 AFE.traineddata	04.12.2021 0:29	Файл "TRAINEDD...	360 КБ
 AFE-2.traineddata	30.11.2020 23:03	Файл "TRAINEDD...	12 501 КБ
 afr.traineddata	04.12.2021 0:29	Файл "TRAINEDD...	370 КБ
 Afromatic.traineddata	04.12.2021 0:29	Файл "TRAINEDD...	372 КБ
 ah.traineddata	04.12.2021 0:29	Файл "TRAINEDD...	365 КБ
 AHintofSass.traineddata	04.12.2021 0:29	Файл "TRAINEDD...	367 КБ
 Al.traineddata	04.12.2021 0:29	Файл "TRAINEDD...	372 КБ
 Aida.traineddata	04.12.2021 0:29	Файл "TRAINEDD...	353 КБ
 Aiden.traineddata			

Рисунок 2.4 – Набір тренуваних даних

Для цього нам потрібно зробити власний шрифт, будемо використовувати онлайн сервіси для цього. Також необхідно буде встановити програму під назвою jTessBoxEditor яка є допоміжною у створенні тестових даних для нашої програми.

Після того, як ви підготували всі вищенаведені кроки встановлення, ви готові до навчання свого Tesseract. Tesseract використовує «мову» як свою модель для OCR. Існує багато мов за замовчуванням, як-от eng (англійська), ind (індонезійська) тощо. Ми намагаємося створити нову мову для Tesseract, щоб мати можливість передбачати наш шрифт, створюючи деякі навчальні дані, що складаються з випадкових чисел, використовуючи наш шрифт. Є 2 способи зробити це. По-перше, якщо у вас є колекція зображень, що складається лише з ваших шрифтів, ви можете використовувати це або, другий спосіб, тобто ввести

будь-яке число (або символ), яке ви бажаєте, у слові за допомогою шрифту та використовувати інструменти для вирізання (вікна).)

З мого досвіду, 10–15 даних було достатньо, щоб створити точну (суб'єктивну) модель, яка є достатньо точною як для чистих зображень, так і для деяких із шумами. Зауважте, що ви повинні намагатися створити якомога збалансовані дані та якомога ближчі до реальних.

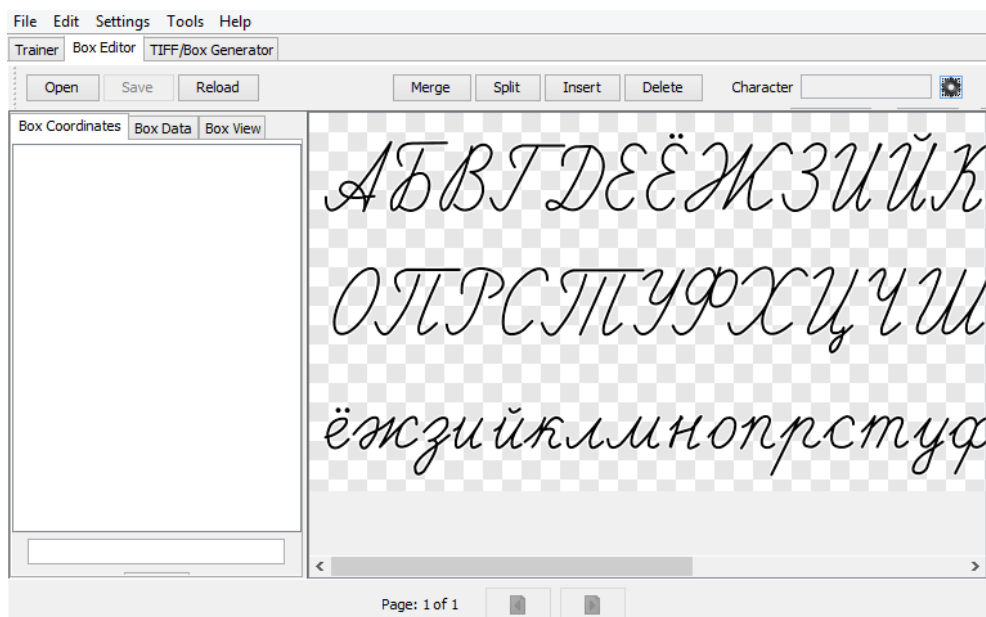


Рисунок 2.5 – Інтерфейс програми jTessBoxEditor

Після того як ми зберегли дані в потрібному форматі, нам потрібно створити тренований ярлик, для нашої програми Tesseract. Для цього необхідно ввести деякі команди у терміналі

```
C:\Program Files\Tesseract-OCR>tesseract --psm 6 --oem 3 myfont.font.exp0.tif myfont.font.exp0 makebox
```

Рисунок 2.6 – Команди для створення «боксів»

Параметри psm та параметри oem здаються дуже дивними. Psm означає режими сегментації сторінки, тобто те, як Tesseract обробляє зображення. Якщо потрібно, щоб Tesseract розглядав кожне зображення, яке він бачить, як одне слово, ви можете вибрати psm 8. У нашому випадку, оскільки наші зображення у файлі .tiff є колекцією однорядкового тексту, ми вибираємо psm 6. Щодо OEM, це означає режими Ocr Engine, оскільки для tesseract є застарілий движок, який працює шляхом розпізнавання шаблонів символів або використанням нейронних мереж і механізмів LTSM

Використовуючи наведену вище команду, ми хочемо, щоб Tesseract створював обмежувальні рамки та передбачення кожного зображення у файлі .tiff і зберігав його у текстовому файлі myfont.font.exp0.box. Файл .tiff, який ми створили раніше, містить ваші навчальні зображення, сегментовані за «сторінкою». Використовуючи наведену вище команду, вона створить файл .box, що містить передбачення та обмежувальну рамку для кожного слова у файлі .tiff. з назвою myfont.font.exp0.box

Тепер необхідно відкрити jTessBoxEditor, перейти на вкладку редактора коробок, натиснути кнопку відкрити та вибрати .tiff. Ви повинні бачити, що кожне зображення на кожній сторінці має свої обмежувальні рамки та передбачення. Ваше завдання тепер полягає в тому, щоб виправити кожну обмежувальну рамку та її передбачення символів у файлі .box.

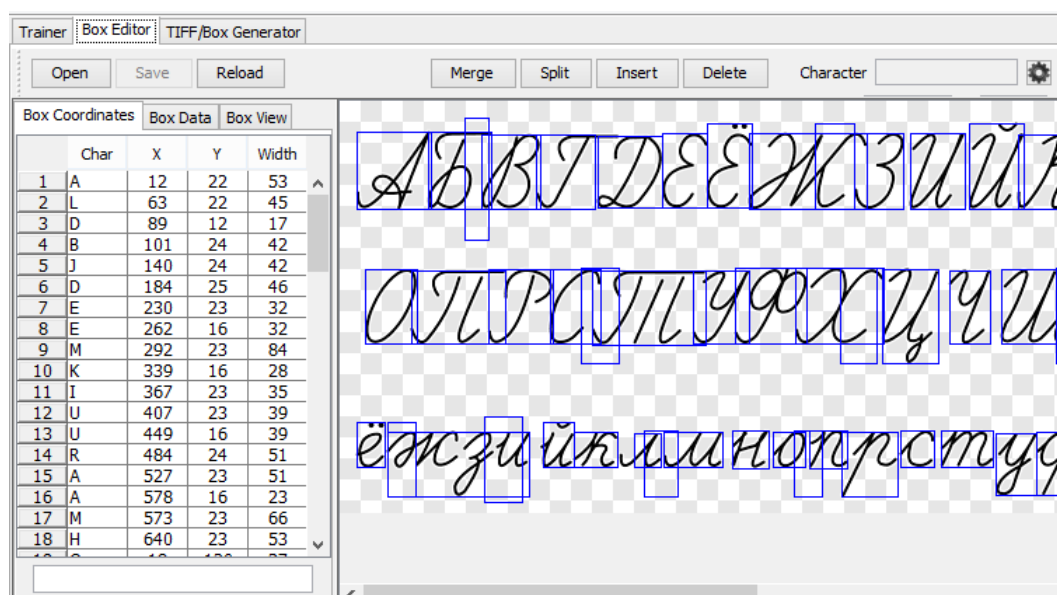


Рисунок 2.7 – Розпізнаний шрифт

Що ж настав час натренувати нашу модель. Команди які необхідно буде вводити будуть повністю пояснені. Але спершу необхідно трішки погратися з розмірами, тому що як видно з рисунку програма розпізнала наш шрифт трішки невдало

```
tesseract num.font.exp0.tif font_name.font.exp0 nobatch box.train
```

Лістинг 1.1 – Команда для створення тренованого файлу

```
unicharset_extractor font_name.font.exp0.box
```

Лістинг 1.2 – Команда для створення формувальної таблиці

Після введення команд які будуть у Додатку (), ми побачимо деякі результати виконання команд своєму терміналі, і, головне, у частині кластеризації форм. Наші навчальні зображення містять усі необхідні символи, ви побачите, що кількість фігур. Наприклад, якщо ми хочемо навчити Tesseract правильно читати літери, тоді кількість фігур дорівнює 32 (що дорівнює $0,1,2,3,\dots, 32$).

Після всіх виконаних команд ми отримаємо готові дані у форматі .tessdata який є звичайним форматом для Tesseract

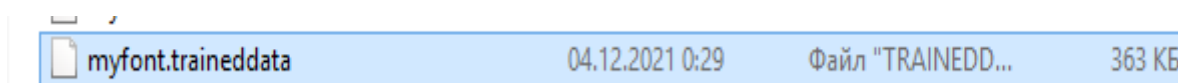


Рисунок 2.8 – готові дані для обробки тексту.

Тепер потрібно розглянути ще деякі аспекти роботи програми для повного її розуміння

Частина процесу розпізнавання будь-якого символу механізму розпізнавання полягає в тому, щоб визначити, яким має бути слово розділений на символи. Початкова сегментація вихід із пошуку лінії класифікується першим. Решта кроків розпізнавання слів застосовується лише до тексту без фіксованого кроку.

Тоді як результат від слова є незадовільним, Tesseract намагається покращити результат відрізавши крапку з найгіршою впевненістю класифікатору символів.

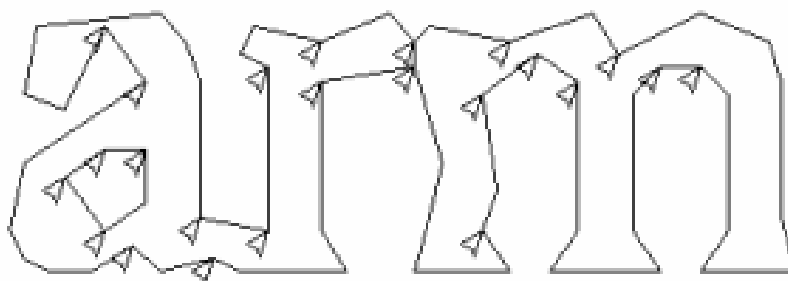


Рисунок 2.9 – Виріз непотрібних частин букв для розпізнання

На рис. 2.9 показаний набір точок-кандидатів стрілки та вибраний розріз у вигляді лінії поперек, де «r» торкається «m». Прорізи виконуються в

пріоритетному порядку. Будь-який проріз, що не може підвищити впевненість результату скасовується, але не повністю викинутий, щоб можна було використати його повторно, якщо це необхідно.

Коли потенційні прорізи були вичерпані, якщо слово все ще недостатньо добре розпізнано, воно передається асоціатору. Асоціатор виконує пошук A^* (найкращий перший) за можливим графіком сегментації комбінації максимально порізаних крапель на символи-кандидати. Він робить це, фактично не будуючи графік сегментації, а натомість підтримує хеш-таблицю відвідуваних станів.

Пошук A^* продовжується шляхом вилучення нових станів-кандидатів із черги пріоритетів та їх оцінки шляхом класифікації некласифікованих комбінацій фрагментів. Можна стверджувати, що цей підхід «повністю відрізати, а потім асоціювати» в кращому випадку неефективний, у гіршому може пропустити важливі відбивні, і це цілком може бути так. Перевага полягає в тому, що схема «відрізати для асоціації» спрощує структури даних, які будуть потрібні для підтримки повного графіка сегментації.

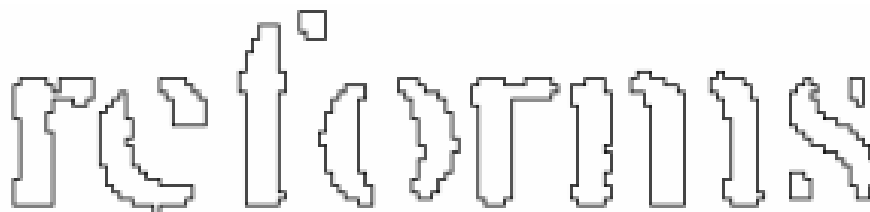


Рисунок 2.10– Розпізнане слово

Класифікація відбувається як двоетапний процес. На першому кроці обрізувач класів створює короткий список класів символів, яким може відповідати невідомий. Кожна функція вибирає з грубо квантованої тривимірної таблиці пошуку біт-вектор класів, яким вона може відповідати, і біт-вектори підсумовуються за всіма функціями.

Класи з найбільшою кількістю (після виправлення очікуваної кількості ознак) стають коротким списком для наступного кроку. Кожна ознака невідомого шукає бітовий вектор прототипів даного класу, яким вона може відповідати, а потім обчислюється фактична подібність між ними. Кожен клас символів прототипу представлений виразом логічної суми добутку з кожним

терміном, який називається конфігурацією, тому процес обчислення відстані веде запис про загальні докази подібності кожної функції в кожній конфігурації, а також кожного прототипу. Найкраща комбінована відстань, яка розраховується з узагальнених доказів функцій і прототипів, є найкращим серед усіх збережених конфігурацій класу.

Слова з різних сегментів можуть мати різну кількість символів. Важко порівнювати ці слова безпосередньо, навіть якщо класифікатор стверджує, що створює ймовірності, чого Tesseract не робить.

Ця проблема вирішується в Tesseract шляхом створення двох чисел для кожної класифікації символів. Перший, який називається впевненістю, - мінус нормована відстань від прототипу. Це дає змогу бути «впевненим» у тому сенсі, що більше чисел краще, але все одно відстань, оскільки чим далі від нуля, тим більше відстань. Другий вихід, який називається рейтингом, множить нормовану відстань від прототипу на загальну довжину контуру в невідомому символі. Оцінки для символів у слові можна осмислено підсумувати, оскільки загальна довжина контуру для всіх символів у слові завжди однакова.

Було висловлено припущення та продемонстровано, що механізми OCR можуть отримати користь від використання адаптивного класифікатора. Оскільки статичний класифікатор повинен добре узагальнювати будь-який тип шрифту, його здатність розрізняти різні символи або символи та не символи послаблюється. Тому адаптивний класифікатор з більшою чутливістю до шрифтів, який навчається на виході статичного класифікатора, зазвичай використовується для отримання більшої дискримінації в кожному документі, де кількість шрифтів обмежена.

Tesseract не використовує шаблонний класифікатор, але використовує ті ж функції, що й статичний класифікатор. Єдина істотна відмінність між статичним класифікатором і адаптивним класифікатором, окрім навчальних даних, полягає в тому, що адаптивний класифікатор використовує ізотропну нормалізацію базової лінії/х-висоти, тоді як статичний класифікатор нормалізує символи за центром (перші моменти) для позиції та другого моменти для нормалізації

анізотропного розміру. Нормалізація базової лінії/висоти x полегшує розрізнення символів верхнього та нижнього регістру, а також покращує стійкість до шумових плям.



Рисунок 2.11 – Подрібнене слово

Основною перевагою нормалізації моменту символу є видалення співвідношення сторін шрифту та певного ступеня ширини обведення шрифту. Це також спрощує розпізнавання підрядкових і верхніх індексів, але вимагає додаткової функції класифікатора, щоб розрізнити деякі символи верхнього та нижнього регістру.

**of 9.5% annually while the Fed-
erated junk fund returned 11.9%
*fear of financial collapse,***

Рисунок 2.12 – Важкість розпізнання слова із-за висоти лінії

Tesseract містить відносно мало лінгвістичного аналізу. Щоразу, коли модуль розпізнавання слів розглядає нову сегментацію, лінгвістичний модуль (неправильно названий *permuter*) вибирає найкращий доступний рядок слів у кожній з наступних категорій: найпоширеніші слова, головне словникове слово, верхнє числове слово, верхнє слово верхнього регістру. Верхнє слово у нижньому регістрі (з необов'язковим початковим верхнім), Вибір верхнього класифікатора слово. Остаточне рішення для даної сегментації є просто слово з найнижчим рейтингом загальної відстані, де кожна з наведених вище категорій помножених на різну константу.

Але чи Tesseract це єдиний шлях, якщо ви хочете отримати готовий і швидкий механізм OCR?. На мою чесну думку, Tesseract добре підходить, якщо ваші зображення дійсно чисті (наприклад, текстовий документ, касовий рахунок

тощо. Якщо дані зображення містять багато шумів, ви можете використовувати порогове значення, щоб розрізнити фон і шум від самого шрифту. З мого досвіду, використовуючи лише 10–20 даних, Tesseract зміг конкурувати навіть з найсучаснішою моделлю виявлення об’єктів, як-от Faster R-CNN, яку навчали, використовуючи набагато більше даних (з також багато збільшення). АЛЕ якщо дані ваших зображень мають деякі шуми (випадкові точки, брудні позначки) з тим самим кольором вашого шрифту, Tesseract не зможе правильно передбачити ваші зображення. Рекомендується повинні використовувати Tesseract, якщо хочете щоб побудувати модель OCR якомога швидше, або у вас є обмежена кількість навчальних даних.

Однією з головних слабкостей Tesseract є те, що він досить нестабільний. Ми могли б отримати інший результат, просто використавши більший обріз того самого зображення. Також чомусь, якщо я використовую більше 50 даних, Tesseract працює гірше. Але наша задача покладає в тому щоб збалансувати програму та отримати реальний результат, що приведе до більш менш задовільних результатів.

2.4. Реалізація функціоналу мобільного додатку

Реалізацію функціоналу програмного продукту потрібно розбити на дві частини: створення клієнтського додатку і створення системи для розпізнавання тексту.

Звернення до серверу відбуваються методами створення HTTP-запитів. HTTP припускає, що клієнтська програма — веб-браузер — здатна відображати гіпертекстові веб-сторінки та файли інших типів у зручному для користувача вигляді. Для правильного відображення HTTP дозволяє клієнтові знати мову та кодування символів веб-сторінок й/або запитати сторінки в потрібних мовах/кодуванні, використовуючи значення зі стандарту MIME.

Для обробки цих запитів було ініціалізовано сервер на основі мікро-фреймворку FLASK.

Програмний код ініціалізації серверного додатку, та прикладом обробки основного HTTP- запиту.

```
from flask import Flask, request, jsonify
import pytesseract
from utils.recognize import recs
pytesseract.pytesseract.tesseract_cmd=r'C:\Program Files\Tesseract-OCR\tesseract.exe'
app = Flask(__name__)

@app.route("/<userID>", methods=['GET', 'POST'])
def ocr(userID):
    if request.method == 'POST':
        data = request.get_json();
        result = recs(data)
        return jsonify(recognizedText = result)
    if request.method == 'GET':
        print(request)
        return jsonify(content)
if __name__ == '__main__':
    app.run(debug=True)
```

В прикладі вище використовується так званий сніпет для того щоб фізичного запустити виконуваний файл програми Tesseract, що свідчить про те, що програма має бути встановлена на сервері.

Також в прикладі вище використовується бібліотека jsonify яка дає можливість отримувати необхідну інформацію із запиту який приходить. Також у коді можна помітити, що цей запит є запитом із параметрами, а отже кожен унікальний користувач який відправлятиме цей запит буде мати свій унікальний id.

За все відповідає основна функція яка викликається при прийнятті запиту. Але для того щоб отримати потрібний результат необхідно виконати ряд перетворень із нашими даними у запиті, тобто привести їх до зручного виду, а вже потім віддавати результат.

```
def data_uri_to_cv2_img(uri):
    image = Image.open(BytesIO(base64.b64decode(uri.split(',')[1])))
    return cv2.cvtColor(np.array(image), cv2.IMREAD_COLOR)
```

Функція data_uri_to_cv2_img() приймає в себе так званий base64 рядок і повертає зображення для роботи з ним.

Base64 — позиційна система обчислення з основної 64. Система широкого доступу до електронної пошти передає бінарні листи в текст (транспортне кодування).

Усі широко поширені варіанти, відомі під назвою Base64, відомі символи A...Z,...z і...9, що становлять 62 знаки, для інших двох знаків у різних системах є різні символи. Основа 64 (26) — це найбільша частина двійки, яка може бути представлена лише друкованими символами ASCII. Ця система використовується в основному, щоб передавати зображення на сервер, або зберігати саму їх сутність в такому вигляді.

```
def resizeImage(image):  
    scale = 50  
    width = int(image.shape[0] * scale/100)  
    height = int(image.shape[1] * scale/100)  
    dim = (width,height)  
    return cv2.resize(image, dim, interpolation = cv2.INTER_AREA)
```

Наступна функція змінює розмір зображення, для зручної обробки.

Після цього відбуваються корегуючі функції, які приводять зображення до потрібного виду, а саме перетворюють його у чорнобіле, прибирають шум, вирівнюють літери, прибирають розмиття і т.д.

```
def gray(img):  
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)  
    return img  
def blur(img) :  
    img_blur = cv2.GaussianBlur(img,(5,5),0)  
    return img_blur  
def threshold(img):  
    img = cv2.threshold(img, 100, 255, cv2.THRESH_OTSU | cv2.THRESH_BINARY)[1]  
    return img  
def remove_noise(image):  
    return cv2.medianBlur(image,5)  
def thresholding(image):  
    return cv2.threshold(image, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)[1]  
def deskew(image):  
    coords = np.column_stack(np.where(image > 0))  
    angle = cv2.minAreaRect(coords)[-1]  
    if angle < -45:  
        angle = -(90 + angle)  
    else:  
        angle = -angle
```

```

(h, w) = image.shape[:2]
center = (w // 2, h // 2)
M = cv2.getRotationMatrix2D(center, angle, 1.0)
rotated = cv2.warpAffine(image, M, (w, h), flags=cv2.INTER_CUBIC,
borderMode=cv2.BORDER_REPLICATE)
return rotated

```

Ці всі функції призначені для приведення зображення до зручного виду. Основна частина відбувається у функції `find_contours()`.

```

def find_contours(img, contours, language):
    config = ('--oem 3 --psm 3')
    result=""
    for cnt in contours:
        x, y, w, h = cv2.boundingRect(cnt)
        rect = cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 255), 2)
        cropped = img[y:y + h, x:x + w]
        if language == 'ukr':
            result = result + pytesseract.image_to_string(cropped,
config=config, lang='ukr')
            result = result.strip()
        elif language == 'rus':
            result = result + pytesseract.image_to_string(cropped,
config=config, lang='rus')
            result = result.strip()
        else:
            result = result + pytesseract.image_to_string(cropped,
config=config, lang='eng')
            result = result.strip()
    return result

```

Функція приймає в себе декілька параметрів, а саме: зображення, тип зображення ((png/jpeg), є також функція яка визначає тип зображення) та мову – параметр що прийшов з клієнтського додатку у вигляді параметру JSON об'єкту.

Отриманий результат прибирається від лишніх символів та повертається у вигляді JSON-об'єкту відповіді, який потім оброблюється на клієнті.

Іншою важливою частиною системи є клієнтський додаток. Додаток створюється за допомогою системи React Native, що має свої недоліки та переваги які були зазначені у минулому розділі, а отже дещо відрізняється від звичного формату CSS, хоча й дуже схожий, але зі своїми відмінностями та недоліками.

Розглянемо основний алгоритм роботи клієнтського додатку.

Як тільки користувач заходить у додаток відбувається запит на Firebase Cloud Database, що реєструє або авторизує користувача.

```
const registerUser = async()=>{
  try{
    user.get().then((snapShot)=>{
      if (snapShot.exists) {
        console.log("user exists!")
        return true;
      }else{
        user.set(newUser);
        return true;}
    })
  }catch(e){
    console.warn(e)}
}
```

Слід зазначити що всі методи які будуть використовуватися у програмі будуть асинхронними. Для позначення того, що методи будуть асинхронними вказано модифікатор `async` перед описом методу. Обов'язковою умовою такого методу є те, що він має повертати результат у вигляді об'єкту `Promise`.

Проміс — це заглушка для асинхронної операції. Кожен проміс проходить через короткий життєвий цикл, що починається зі стану *очікування* (*pending*), який вказує на те, що асинхронна операція ще не завершена. Проміс, що перебуває у стані очікування, вважається *невстановленим* (*unsettled*). Проміс з останнього прикладу перебуває у стані очікування, як тільки функція `readFile()` повертає його. Як тільки асинхронна операція завершується, проміс вважається *встановленим* (*settled*) і переходить один з двох можливих станів:

- завершений — коли асинхронна операція у промісі завершена успішно;
- відхилений — коли асинхронна операція не завершена успішно через помилку, або з іншої причини.

```
const sendPicture = async(bodyObject) =>{
  try{
    let response = await fetch(`http://192.168.1.6:5000/${uniqueID}`, {
      method: 'POST',
      headers: {
        Accept: 'application/json',
        'Content-Type': 'application/json'
      }
    })
  }catch(e){
    console.warn(e)}
}
```

```

    },
    body:JSON.stringify(bodyObject)
  })
  let json = await response.json();
  return json;
}
catch(e){
  console.warn(e)
}
}

```

У коді вище зазначений POST запит на сервер який формується при натисненні клавiшi «Вiдправити» яка з'являється пiсля вибору зображення.

У програмуванні POST - один з багатьох методiв запиту, що пiдтримуються HTTP протоколом, що використовується у Всесвiтнiй мережi. Метод запиту POST призначений для надсилання запиту, при якому веб-сервер приймає данi, що мiстяться в тiло повiдомлення, для зберiгання. Вiн часто використовується для завантаження файлу або представлення заповненої веб-форми. Саме POST запит використовується для передачі будь-яких файлiв на сервер.

Наш клiєнтський додаток має декiлька вкладок, тому важливо органiзувати зручну навігацію між ними. Нижче представлений код який допомагає у навігації між вкладками.

```

const Tab = createBottomTabNavigator();
const TabNavigator = () => {
  const screenOptions = (route, color) => {
    let iconName;
    switch (route.name) {
      case 'Тексти':
        iconName = 'file-text';
        break;
      case 'Розпізнати':
        iconName = 'camera';
        break;
      default:
        break;
    }
    return <Icon name={iconName} color='#fff' size={24} />;
  };
  return (
    <Tab.Navigator
      screenOptions={({route}) => ({
        tabBarIcon: ({color}) => screenOptions(route, color),
        headerShown:false,

```

```

        tabBarStyle:{backgroundColor:"#0C3F48"},
      }}}>
      <Tab.Screen name="Позпізнати" component={ImagePickerScreen}
tabBarItemStyle={{color:'#fff'}}/>
      <Tab.Screen name="Тексти" component={Home} />
    </Tab.Navigator>
  );
};
export default TabNavigator;

```

Основний функціонал додатку – це сторінки які взаємодіють з клієнтом. Кожна така сторінка представлена у вигляді React компоненту з власним станом та функціональними можливостями. Нижче наведений приклад запиту для створення зображення для відправки:

```

const ImagePickerComponent = () =>{
  const [imageSource, setImageSource] = useState(null);
  const [selectedLanguage,setSelectedLanguage] = useState(pickerLanguages[0].value);
  const [bodyObject,setBodyObject] = useState({"language":selectedLanguage});
  const [isSend,setIsSend] = useState(false);
  const navigation = useNavigation();
  const imageOptions = {
    mediaType: 'photo',
    includeBase64: true,
    saveToPhotos:true,
  };

  const selectImageFromGallery = async () => {
    let result = await ImagePicker.launchImageLibrary(imageOptions);
    if(!result.didCancel){
      setImageSource(result.assets[0].uri);
      setBodyObject({...bodyObject,"type":result.assets[0].type,"base64":result.ass
ets[0].base64})
    }
  };
  return (
... // об'єкт розмітки компоненту
);
};

```

Важливою частиною цього процесу є надання прав доступу до камери та до файлової системи системи на якій працює додаток. У даному випадку додаток працює під операційною системою Android, і для того, щоб викликати вікно для надання прав, необхідно прописати певні правила у спеціальному конфігураційному файлі Android додатку (AndroidManifest.xml). У коді нижче показаний вигляд цих правил.


```
<uses-permission android:name="android.permission.CAMERA"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
```

Створення компонентів React – це процес створення функціональних або класових компонентів, що відповідають вимогам системи React. У компоненті містяться описи його ініціалізації, обробки подій користувачів, та зовнішній вид компонента. У наведеному програмному коді нижче використовується так званий хук `useEffect`. Цей хук або ж функція є подією загального життєвого циклу React компоненту яка спрацьовує при ініціалізації компонента, або ж зміни його стану.

Для компонента із набором всіх розпізнаних текстів користувача `useEffect` використовується одразу ж як користувач зайшов на цей компонент. Створюється запит до бази даних який повертає масив всіх текстів зв'язаних з цим користувачем.

Також React надає можливість створювати гнучкі компоненти із використанням логічних операторів та фрагментів коду. Ця можливість є дуже зручною тому, що завдяки їй ми можемо створювати гнучкі інтерфейси які будуть різними в залежності від стану компоненту.

Розглянемо код для створення розмітки шаблону компоненту з користувацькими текстами

```
<ScrollView
  indicatorStyle="white"
  contentContainerStyle={[
    styles.scrollContentContainer,
    { paddingBottom: tabBarHeight },
  ]}>{
  myData && !loading ? (<
    {
//mapping through array
    }</>) : (
    <ActivityIndicator
      color="rgba(255,255,255,1)"
      size="large"
      style={styles.activityIndicator}
    ></ActivityIndicator>)
  }
</ScrollView>
```

В цьому фрагменті коду використовуються тернарні оператори які надають можливість гнучко оперувати React компонентом в залежності від його стану, або наприклад вивести дані, зі списку, запиту, файлу, тощо.

Ще одним важливим плюсом є те, що React суспільство має розвивається з величезною швидкістю і створює багато модулів, для створення того чи іншого функціоналу у додатку.

Наприклад для збереження файлу до файлової системи використовується модуль RNHTML to PDF, що дозволяє зберегти HTML розмітку у pdf форматі.

Нижче наведена функція яка відповідає за це.

```
const createPDF = async (value) => {
  if (await isPermitted()) {
    let options = {
      html: value,
      fileName: new Date().toLocaleString(),
      directory: 'Documents',
    };
    let file = await RNHTMLtoPDF.convert(options)

    Alert.alert("Файл збережено!", `Файл знаходиться у: ${file.filePath}`);
  }
}
```

Додавання стилів до компоненту відбувається в шаблоні самого компоненту. Для цих цілей існує спеціальний атрибут style, який приймає об'єкт з правилами стилізації. Ці правила можна створювати безпосередньо в самому компоненті або виносити їх в окремий файл, для зручного читання коду.

Опція стилю може бути простим старим об'єктом JavaScript. Це те, що ми зазвичай використовуємо для прикладу коду. Ми також можемо передати масив стилів - останній стиль у масиві має пріоритет, тому ви можете використовувати його для успадкування стилів.

Приклад стилізації наведений нижче та створюється наступним чином:

```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: "#1fb2cc"
  },
  contentContainer: {
    marginTop: 20,
    alignItems: 'center',
    paddingHorizontal: 20,
```

```

paddingBottom: 2,
},
title: {
  fontFamily: "roboto-700",
  color: "#121212",
  fontWeight: 'bold',
  fontSize: 15,
  textAlign: "center"
},
scrollContainer: {
  backgroundColor: "#1fb2cc",
  flex: 1,
},
})

```

На відміну від класичного CSS в правилах стилізації відсутні одиниці вимірювання, але все ж деякі властивості співпадають з таблицями CSS, проте є ряд специфічних властивостей як `resizeMode` наприклад.

Інші компоненти створені схожим чином та змінюються в залежності від потребностей дизайну та функціоналу компоненту.

Висновки до другого розділу

Мобільний додаток реалізує функціонал, який був визначений раніше з усіма перевагами та недоліками порівняно з аналогами програм, також було налаштовано веб-сервер, та реалізовано всі функції для коректної роботи з ним.

Встановлені бізнес вимоги, вимоги користувачів, основні функціональні та нефункціональні вимоги які стали основою для проектування даної системи.

Було розглянуто основні алгоритми на яких будується додаток та вебсервер а також проаналізовано проблеми та способи їх вирішення під час процесу розробки.

Також, було розроблено діаграми діяльності користувача, визначено та описано алгоритми роботи основних функцій системи. На основі описаних алгоритмів, було розроблено основний функціонал як серверного, так і клієнтського додатку.

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З СИСТЕМОЮ

3.1. Порядок встановлення та налаштування системи

Система мобільного додатку складається з клієнтського та серверного додатку. Кожен з цих компонентів має свій процес встановлення та налаштування. Клієнтський додаток розроблений з використанням технології React Native, тому він може бути доступним на різних платформах: ОС Android, IOS, веб-браузер. Серверний додаток включає в себе систему Firebase, REST API та процес бази даних.

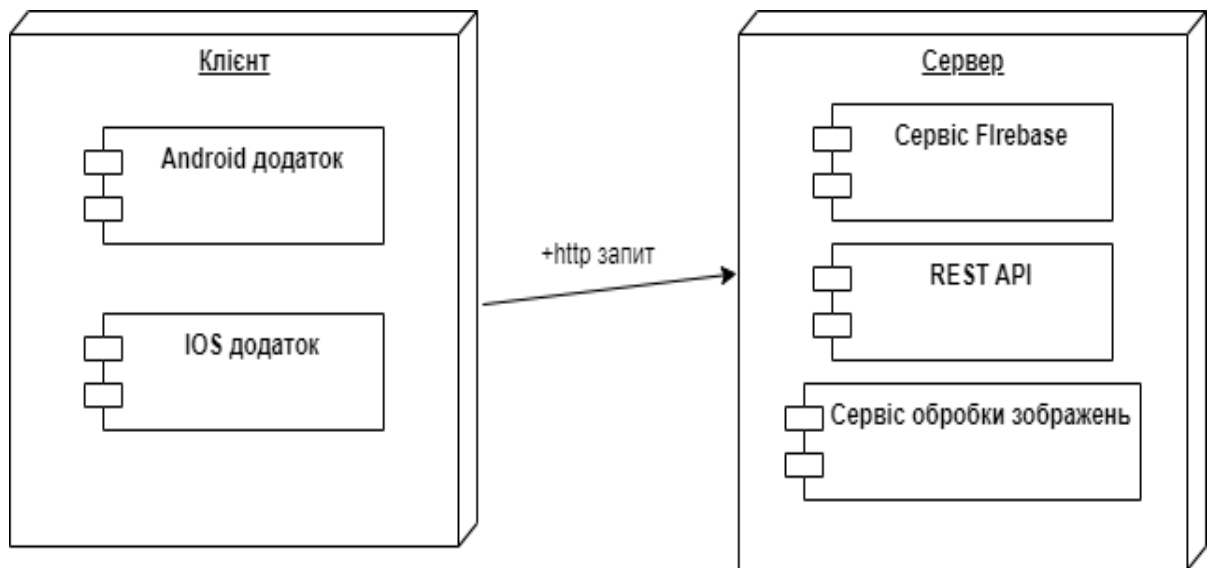


Рисунок 3.1 – Діаграма розгортання системи

На рисунку 3.1 зображена діаграма розгортання системи. На цій діаграмі зображено основні компоненти клієнтської та серверної частин додатку та їх взаємозв'язок. Розглянемо процеси встановлення та налаштування кожного з компонентів системи.

Для налаштування серверної частини додатку необхідно створити проект в системі Firebase та налаштувати параметри клієнтських додатків. В консолі управління проектами Firebase <https://console.firebase.google.com/> необхідно створити клієнтські додатки. Після створення клієнтського додатку ми матимемо змогу налаштувати його напряму в проєкті використовуючи налаштування, які Firebase створить для нас, просто імпортуючи їх в окремий файл нашої системи.

На рисунку 3.2 зображено інтерфейс налаштування проєкту Firebase.

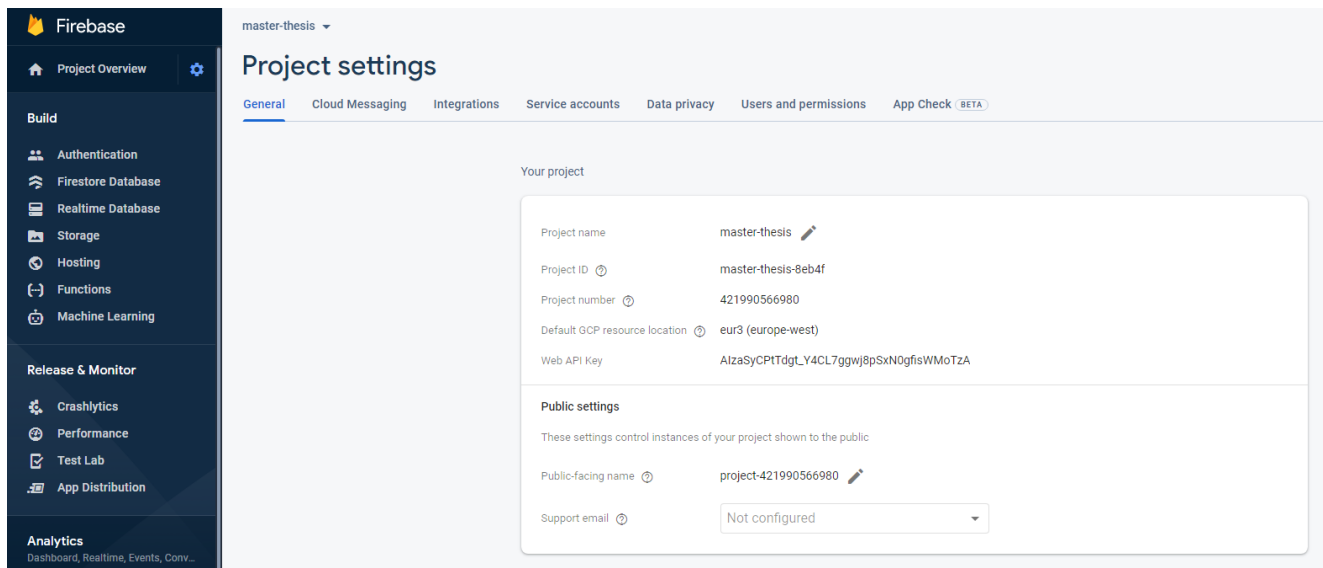


Рисунок 3.2 – Інтерфейс консолі налаштування проекту у Firebase

Як видно на рисунку, налаштування в основному складається з таких параметрів:

- Ім'я проекту
- ID проекту
- Номер проекту
- Розташування віддаленого серверу
- Ключ АПІ (для запитів до додатку)

Для створення бази даних необхідно перейти в пункт `firestore database` та створити її самостійно.

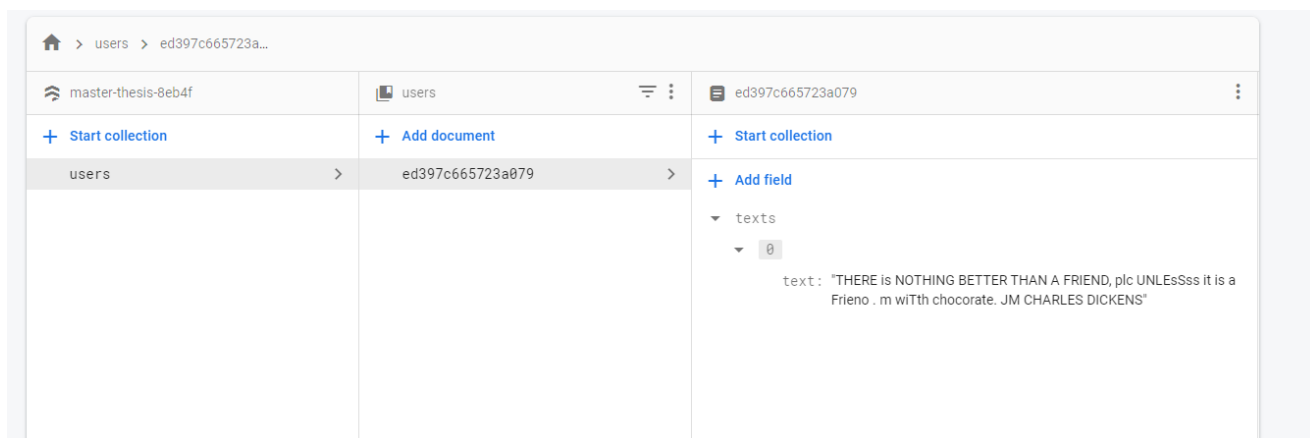


Рисунок 3.3 - Структура бази даних

В нашому випадку всі розпізнані тексти будуть зберігатися у вигляді масиву об'єктів із полем text. На жаль таку структур даних неможливо створити у консолі на даний час, тож довелось написати запит який при реєстрації користувача, додає таку структуру.

Для встановлення Android додатку на пристрій, необхідно встановити програмний файл із розширенням .apk у систему та надати необхідні доступи до функцій.

Для встановлення додатку на операційну систему IOS необхідно встановити програмний файл з розширенням .app у систему. Так як і у випадку з Android системою необхідно надати дозвіл на встановлення файлів з невідомих систем.

3.2. Структура інтерфейсу клієнтського додатку

Розглянемо основні сторінки (екрани) мобільного додатку на прикладі Android смартфона із встановленою операційною системою версії Android 10.

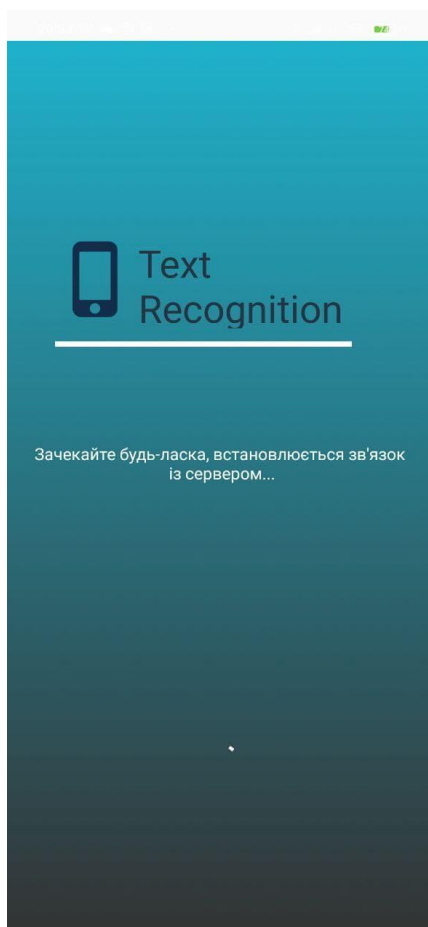


Рисунок 3.4 – Інтерфейс вікна «Логін»

На головному екрані ми бачимо назву нашої програми і одразу ж попередження «зачекати». Саме в цей час відбувається підключення до бази даних й додавання користувача у систему у випадку якщо його немає. Якщо ж користувач присутній у системі, то результат викликаної функції повертає істинний результат і система знає, що користувач вже зареєстрований.



Рисунок 3.5 – Головне вікно програми

На головному екрані видно, що наш додаток має 2 іконки що зазначають, 2 основні функції нашої програми, а саме «Сфотографувати текст» і «Обрати зображення».

Також знизу є 2 вкладки які відповідають за «головний екран» та «тексти», тобто всі тексти, що є у користувача будуть на тій вкладці.

Що ж саме час продемонструвати одну із основних функцій.

Після натиснення на іконку фотоапарату користувач повинен надати дозвіл до камери девайсу, для того, щоб зробити зображення.

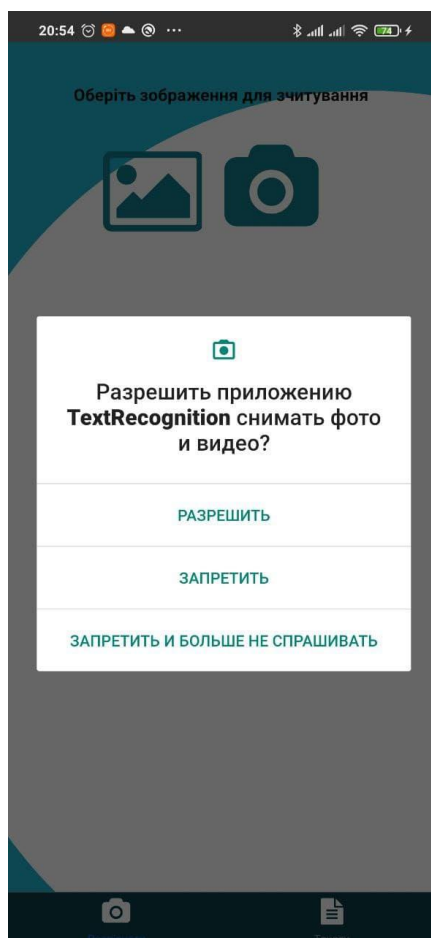


Рисунок 3.6 – Надання дозволу

На наступному рисунку зображенні всі встановлені програми у користувача за допомогою яких він може зробити зображення.



Рисунок 3.7 – Встановлені додатки для фото

На наступному екрані зображено основне меню, нашого додатку. Тобто вже видно саме зображення яке обрав користувач для аналізу, вибір мов, та кнопку відправити зображення.

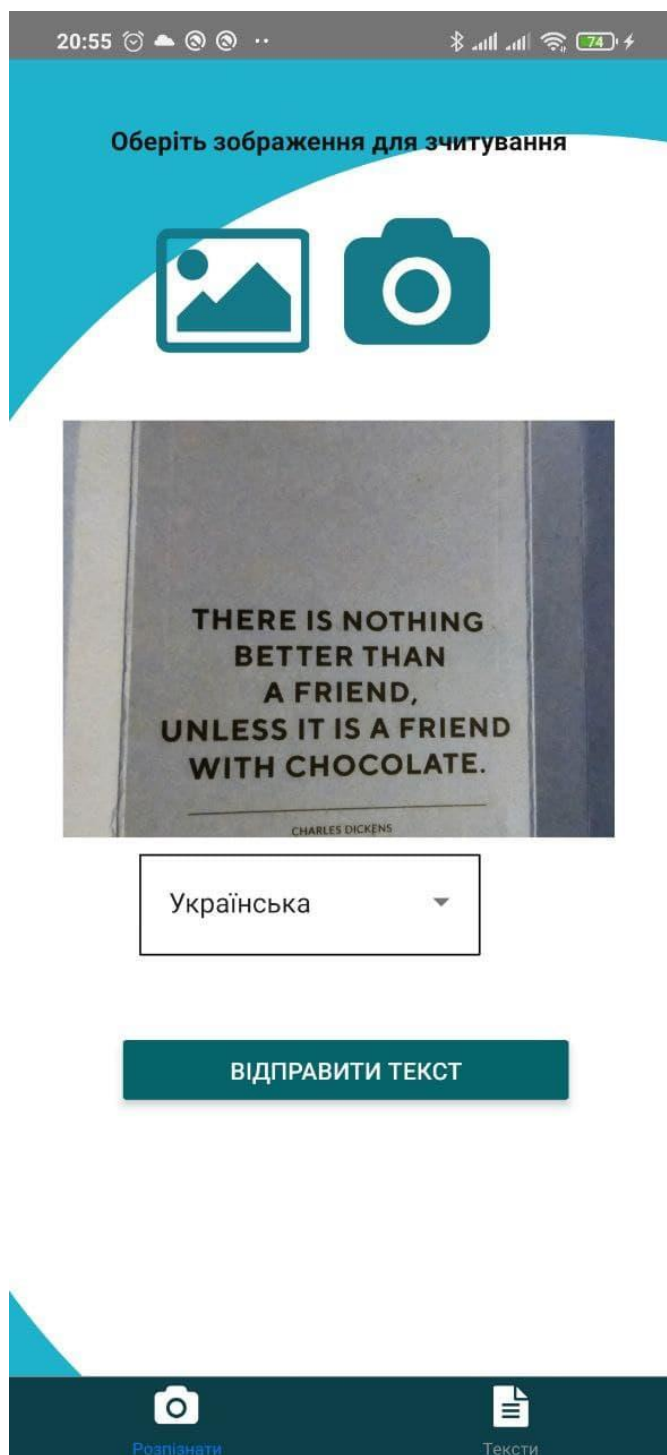


Рисунок 3.8 – Головне вікно програми із обраним зображенням

Немало важливим компонентом головного меню є вибір мови для зображення яке було обрано. Цей компонент був придуманий для покращення розпізнаного тексту у майбутньому. Тобто користувач має обрати саме той текст

який у нього присутній на зображенні. За замовчуванням стоїть «Українська мова».

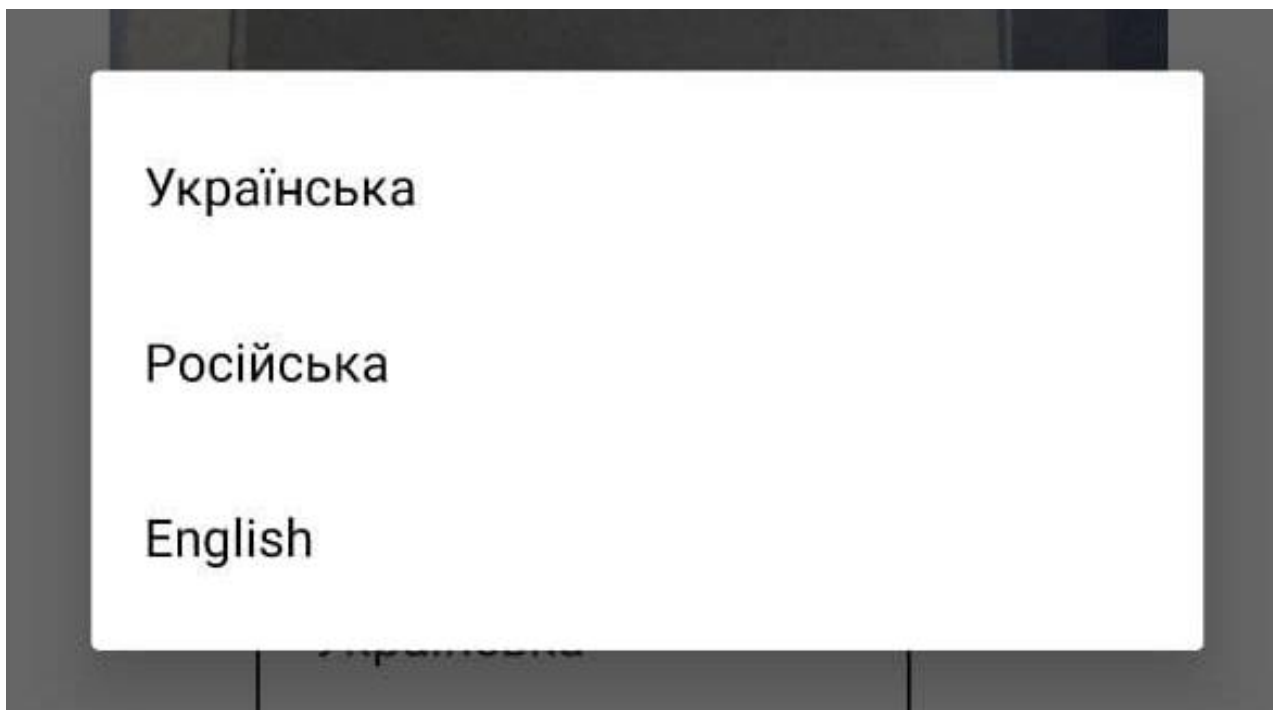


Рисунок 3.9 – Інтерфейс вибору мови

Після налаштування всіх параметрів користувач має можливість відправити обране зображення на сервіс. Під час цього процесу, у користувача з'являється індикатор який показує тому, що необхідно зачекати певний час.

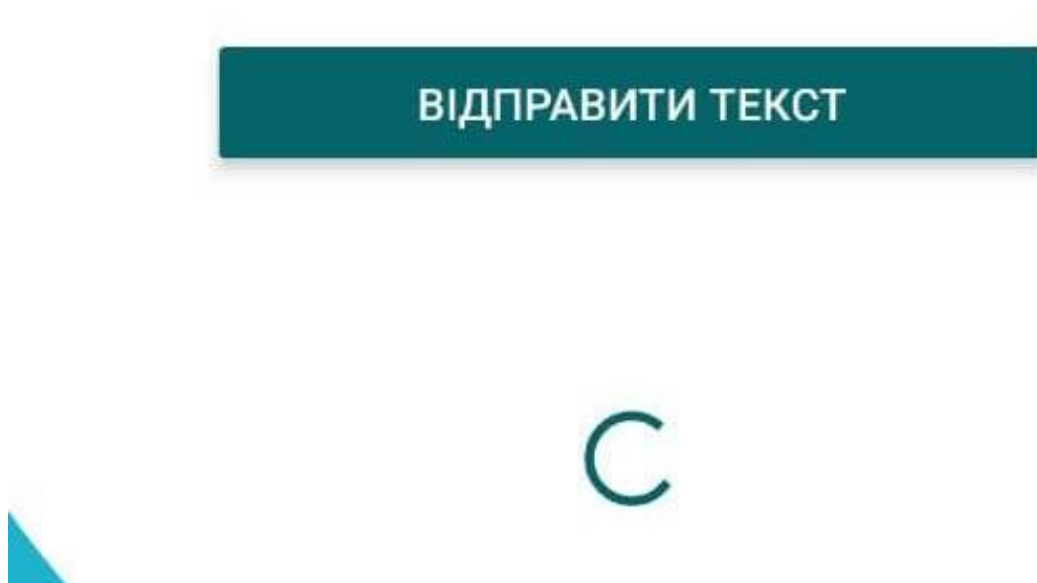


Рисунок 3.10 – Індикатор очікування

Після успішного перекладу користувача автоматично перекине на екран з розпізнаними текстами які належать йому.

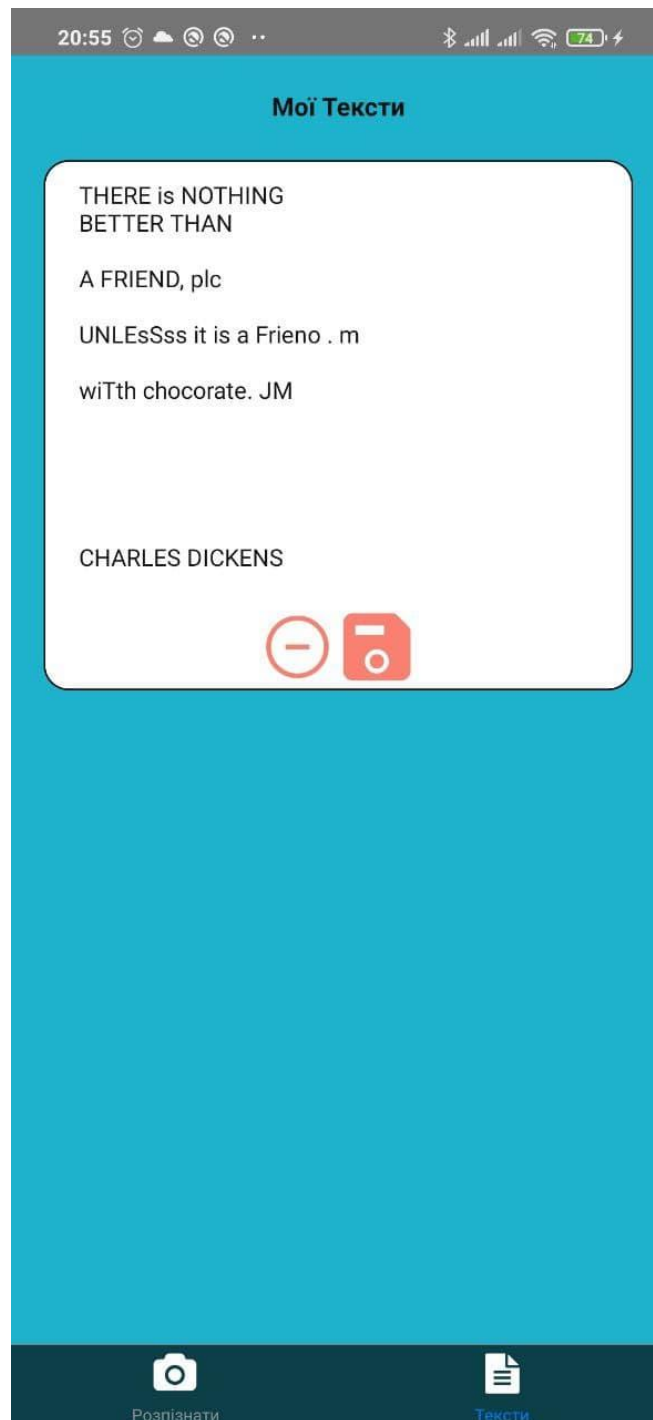


Рисунок 3.11 – Вкладка тексти

На екрані у користувача з'являється перекладені ним тексти із зображень. Вони оформлені у компоненті який дозволяє прогортувати їх неначе веб-сторінку, тобто якщо кількість перекладених текстів займе весь екран то користувач матиме можливість робити свайп вниз для пролистування.

Також ми бачимо дві кнопки які відповідають за створення pdf файлу, та видалення обраного тексту. Ще є функціонал який було важко відобразити, а саме копіювання тексту у буфер обміну. Воно відбувається тоді коли користувач натисне на сам текст у блоці.

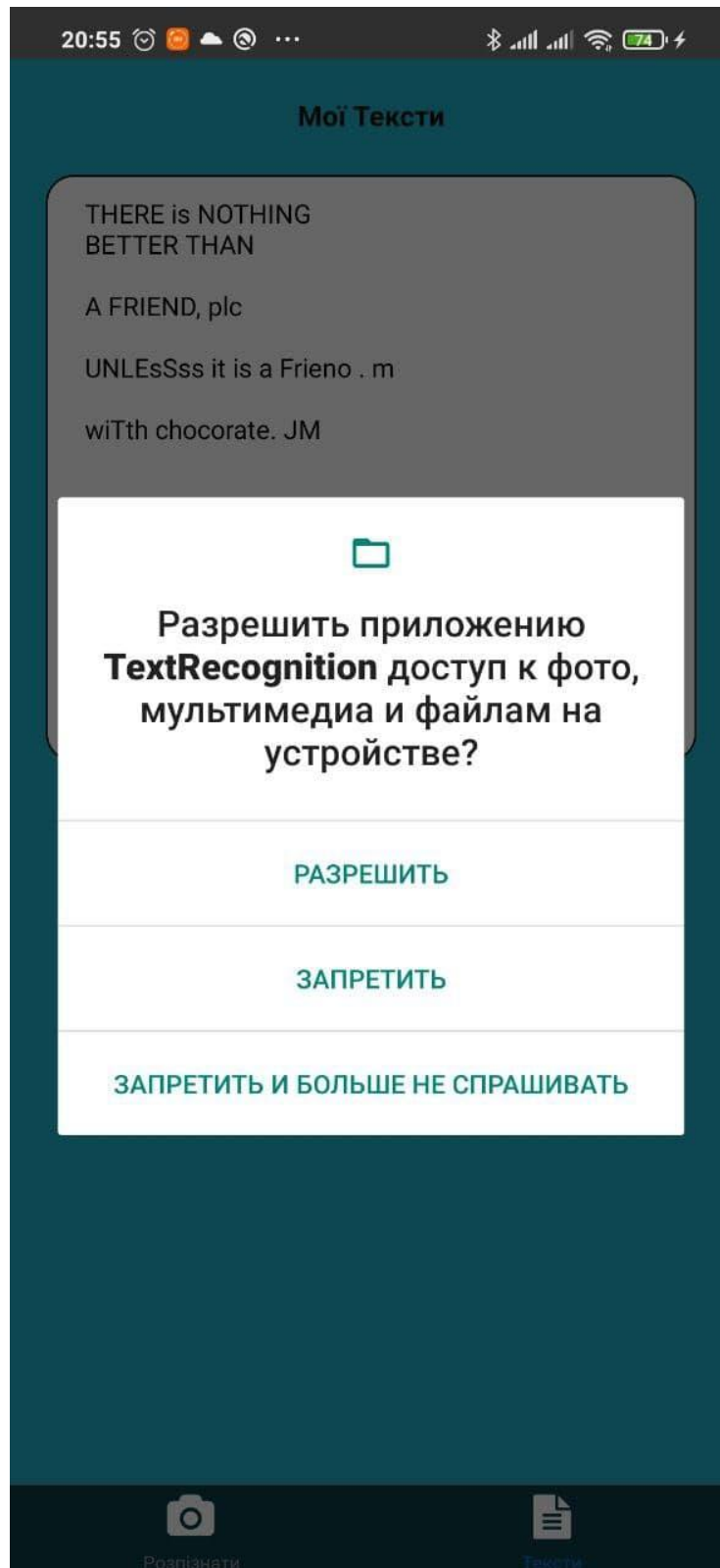


Рисунок 3.12 – Доступ до файлової системи

Якщо користувач спробує зберегти файл у форматі pdf, додаток знову ж таки запросить права на доступ до файлової системи. Після цього виведеться результат у вигляді вікна на якому буде зображено шлях до збереженого файлу.

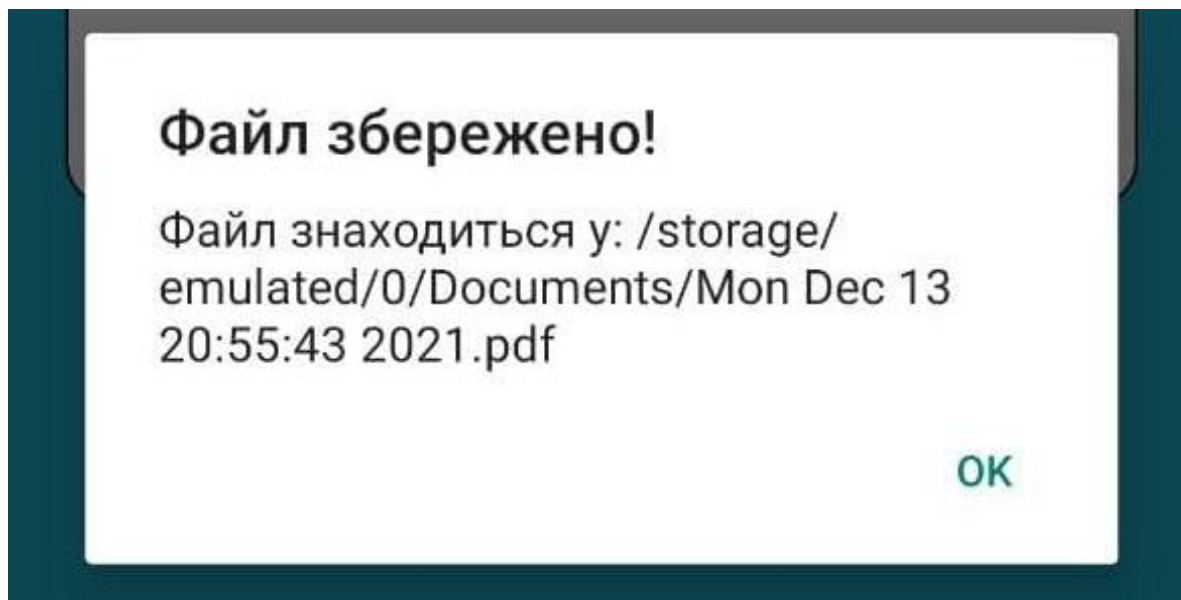


Рисунок 3.13 – Результат збереженого файлу

3.3. Тестування роботи мобільного додатку

Тестування – невід’ємний процес розробки програмного забезпечення. Завдяки різним видам тестування можна знайти та виправити безліч помилок та несправностей у програмі. У цій роботі було проведено декілька видів тестування:

- тестування сумісності;
- тестування функціональності;
- користувацьке тестування;

Інструмент Postman використовується для тестування компонентів REST API сервера.

Postman — інструмент для створення запитів API, який дозволяє тестувати API та усувати помилки, які можуть виникнути при створенні запиту до клієнта або обробці результату цього запиту.

Дана утиліта допомагає виявити всі помилки при відправці запитів на сервер та отримання результатів з нього.

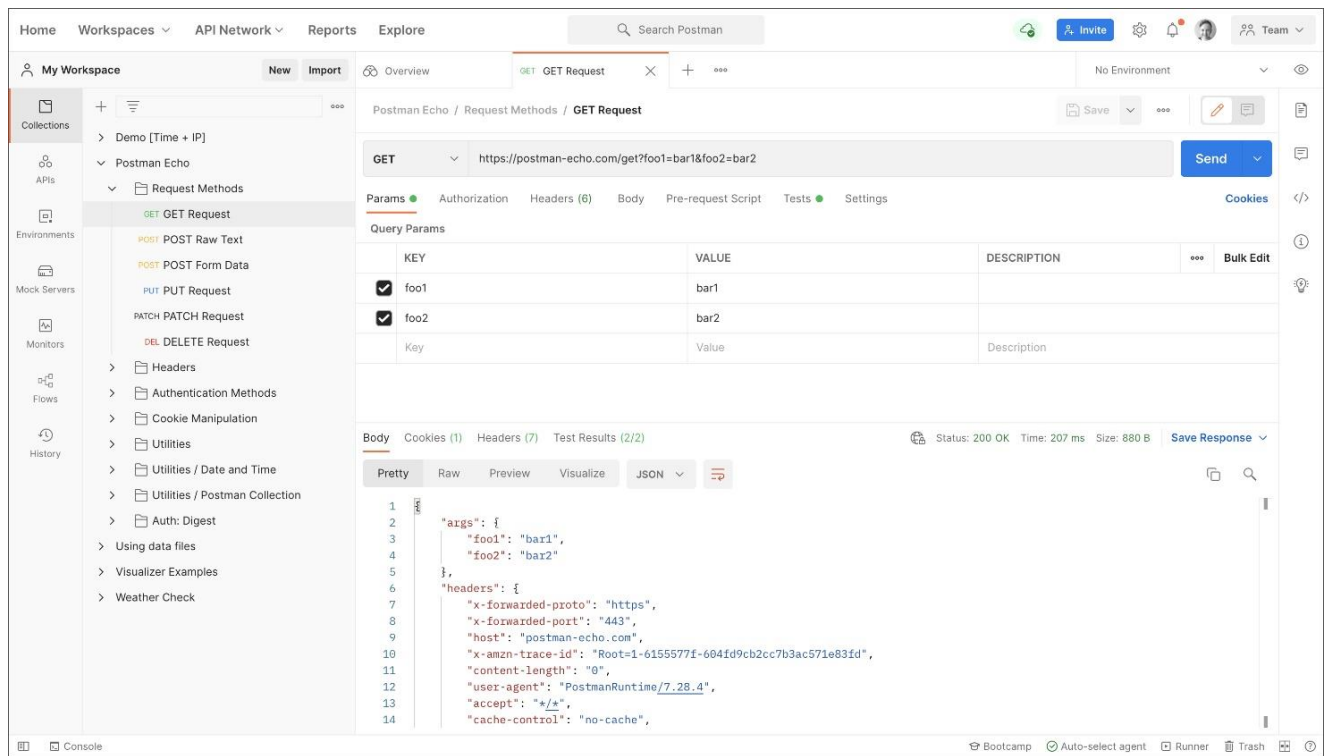


Рисунок 3.14 – додаток Postman

Клієнтську програму тестували в ручному режимі. Під час тестування було виявлено та виправлено багато помилок, розроблено операції з помилками користувача.

Висновки до третього розділу

У третьому розділі магістерської кваліфікаційної роботи було проведено було розглянуто порядок та налаштування для коректної роботи системи на тих чи інших пристроях. Описано передумови необхідні для налаштування системи.

Важливою частиною є користувацький додаток, інтерфейс якого був детально описаний в даному розділі. Наведено опис сторінок клієнтського додатку та реалізації його інтерфейсу. До всіх перерахованих інтерфейсів були додані зображення з детальним поясненням для повного розуміння зовнішнього вигляду системи.

Окрім цього було проведено тестування клієнтської та серверної частини додатку. За результати тестування виявлено та виправлено ряд певних помилок в роботі системи, а також сформовані підходи для обробки інформації користувача та зміни інтерфейсу системи в залежності від середовища додатку.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи магістра було спроектовано та розроблено клієнтський мобільний додаток та сервіс для розпізнавання тексту.

У першому розділі було розглянуто існуючі технології для виконання кваліфікаційної роботи. Було визначено основні цілі, проблеми, мету, а також проведено аналіз існуючого програмного забезпечення з порівняльними характеристиками.

Аналіз показав, що існує велика кількість схожого програмного забезпечення, що дозволить більш точніше потсавити онсонві цілі для нашої задачі, та вдосконалити недоліки існуючих аналогів.

Відповідно до встановленого завдання, була обрана оптимальна архітктура системи, яка допоможе реалізувати необхідний програмний продукт. На основі визначених вимог додатку, були проаналізовані можливі засоби реалізації даного програмного продукту. За результатами порівняння були обрані необхідні інструменти та технології які в тій чи інішій мірі задовольняють цілям програмного продукту.

Мобільний додаток реалізує функціонал, який був визначений раніше з усіма перевагами та недоліками порівняно з аналогами програм, також було налаштовано веб-сервер, та реалізовано всі функції для коректної роботи з ним.

Встановлені бізнес вимоги, вимоги користувачів, основні функціональні та нефункціональні вимоги які стали основою для проектування даної системи.

Було розглянуто основні алгоритми на яких будується додаток та вебсервер а також проаналізовано проблеми та способи їх вирішення під час процесу розробки.

Також, було розроблено діаграми діяльності користувача, визначено та описано алгоритми роботи основних функцій системи. На основі описаних алгоритмів, було розроблено основний функціонал як серверного, так і клієнтського додатку.

У третьому розділі магістерської кваліфікаційної роботи було проведено було розглянуто порядок та налаштування для коректної роботи системи на тих чи інших пристроях. Описано передумови необхідні для налаштування системи.

Важливою частиною є користувацький додаток, інтерфейс якого був детально описаний в даному розділі. Наведено опис сторінок клієнтського додатку та реалізації його інтерфейсу. До всіх перерахованих інтерфейсів були додані зображення з детальним поясненням для повного розуміння зовнішнього вигляду системи.

Окрім цього було проведено тестування клієнтської та серверної частини додатку. За результати тестування виявлено та виправлено ряд певних помилок в роботі системи, а також сформовані підходи для обробки інформації користувача та зміни інтерфейсу системи в залежності від середовища додатку.

В майбутньому заплановано ряд нових функціональних можливостей та способів покращення даної системи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. R.J. Shillman. Character Recognition Based on Phenomenological Attributes: Theory and Methods, PhD. Thesis, : дис. докт. / R.J. Shillman. – Massachusetts Institute of Technology., 1974.
2. Терміни OCR [Електронний ресурс] - 2018. - Режим доступу до ресурсів: https://en.wikipedia.org/wiki/Timeline_of_optical_character_recognition
3. Герберт Ф. OCR: історія, розпізнавання символів optisk / Герберт Ф. Шнц - Манчестер: Манчестерський центр, Вт., 1982 - 156 с.
4. Android version history [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Android_version_history
5. Fetch [Електронний ресурс] – Режим доступу до ресурсу: <https://javascript.info/fetch>
6. Flask [Електронний ресурс] – Режим доступу до ресурсу: <https://palletsprojects.com/p/flask/>
7. Firebase CLI reference [Електронний ресурс] – Режим доступу до ресурсу: <https://firebase.google.com/docs/cli>.
8. Комп'ютерний зір, технології і тренди [Електронний ресурс] – Режим доступу до ресурсу: <https://rdc.grfc.ru/2021/04/analytics-computer-vision/>
9. Learn OpenCV. Understanding Activation Functions in Deep Learning. URL: <https://www.learnopencv.com/understanding-activation-functions-in-deep-learning/>
10. Introduction – React Native [Електронний ресурс] – Режим доступу до ресурсу: <https://reactnative.dev/docs/getting-started>
11. Promise [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/enUS/docs/Web/JavaScript/Reference/Global_Objects/Promise
12. Python [Електронний ресурс] – Режим доступу до ресурсу: <https://uk.wikipedia.org/wiki/Python>
13. The New React Native Architecture Explained [Електронний ресурс] – Режим доступу до ресурсу: <https://formidable.com/blog/2019/react-codegen-part-1/>

ДОДАТКИ

Лістинг обраних скриптів клієнтського додатку

RootNavigator.js

```

import React, { useEffect, useState, Component } from 'react';
import { NavigationContainer } from '@react-navigation/native';
import { createNativeStackNavigator } from '@react-navigation/native-stack';
import { useNavigation } from '@react-navigation/core';
import TabNavigator from './TabNavigator';
import Login from '../screens/LoginScreen';
import { registerUser } from '../network/userRequests';

const Stack = createNativeStackNavigator();

const RootNavigator = () => {
  const [isLoggedIn, setLoggedIn] = useState(false);

  useEffect(() => {
    setTimeout(() => {
      setLoggedIn(registerUser)
    }, 5000)
  }, [])

  return (
    <NavigationContainer>
      {
        !isLoggedIn ? (<Login />) : (<TabNavigator />)
      }
    </NavigationContainer>
  );
};

export default RootNavigator;

```

UserRequests.js

```

import { getUniqueId, getManufacturer } from 'react-native-device-info';
import uuid from 'react-native-uuid';
import { db } from '../config/firebase';
import firebase from 'firebase/app';
const uniqueID = getUniqueId();

const user = db.collection('users').doc(uniqueID);
const newUser = {
  texts:[
    {
      text:""
    }
  ]
}

```

```

}

const registerUser = async()=>{
  try{
    user.get().then((snapShot)=>{
      if (snapShot.exists) {
        console.log("user exists!")
        return true;
      }
      else{
        user.set(newUser);
        return true;
      }
    })
  }
  catch(e){
    console.warn(e)
  }
}

const sendPicture = async(bodyObject) =>{
  try{
    let response = await fetch(`http://192.168.1.6:5000/${uniqueID}`, {
      method: 'POST',
      headers: {
        Accept: 'application/json',
        'Content-Type': 'application/json'
      },
      body:JSON.stringify(bodyObject)
    })
    let json = await response.json();
    return json;
  }
  catch(e){
    console.warn(e)
  }
}

const addText = async(text)=>{
  try{
    user.update({
      texts:firebase.firestore.FieldValue.arrayUnion({"text":text})
    })
  }
  catch(e){
    console.warn(e)
  }
}

const deleteText = async(text)=>{
  try{
    user.update({
      texts:firebase.firestore.FieldValue.arrayRemove({"text":text})
    })
  }
  catch(e){
    console.warn(e)
  }
}

```

```
        })
      }
    catch(e){
      console.warn(e)
    }
  }
  const getData = async()=>{
    let response = user.get();
    if((await response).exists){
      return (await response).data().texts
    }
  }
  export{registerUser,uniqueID,sendPicture,addText,user,getData,deleteText}
```

Лістинг обраних серверних скриптів

Recognize.py

```

import pytesseract
import numpy as np
import cv2
import sys
import re
import base64
import io
import pandas as pd
from matplotlib import pyplot as plt
from io import StringIO
from io import BytesIO
from pytesseract import Output
from PIL import Image
pytesseract.pytesseract.tesseract_cmd=r'C:\Program Files\Tesseract-OCR\tesseract.exe'

def gray(img):
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    return img
# blur
def blur(img) :
    img_blur = cv2.GaussianBlur(img,(5,5),0)
    return img_blur
# threshold
def threshold(img):
    #pixels with value below 100 are turned black (0) and those with higher value are
turned white (255)
    img = cv2.threshold(img, 100, 255, cv2.THRESH_OTSU | cv2.THRESH_BINARY)[1]
    return img
def sharpen_image(im):
    kernel = np.ones((3,3),np.float32)/90
    im = cv2.filter2D(im,-1,kernel)
    return im
# get grayscale image
def get_grayscale(image):
    return cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
# noise removal
def remove_noise(image):
    return cv2.medianBlur(image,5)
#thresholding
def thresholding(image):
    return cv2.threshold(image, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)[1]
#dilation
def dilate(image):
    kernel = np.ones((5,5),np.uint8)
    return cv2.dilate(image, kernel, iterations = 1)
#erosion
def erode(image):

```

```

        kernel = np.ones((5,5),np.uint8)
        return cv2.erode(image, kernel, iterations = 1)
#opening - erosion followed by dilation
def opening(image):
    kernel = np.ones((5,5),np.uint8)
    return cv2.morphologyEx(image, cv2.MORPH_OPEN, kernel)
#canny edge detection
def canny(image):
    return cv2.Canny(image, 100, 200)
#skew correction
def deskew(image):
    coords = np.column_stack(np.where(image > 0))
    angle = cv2.minAreaRect(coords)[-1]
    if angle < -45:
        angle = -(90 + angle)
    else:
        angle = -angle
    (h, w) = image.shape[:2]
    center = (w // 2, h // 2)
    M = cv2.getRotationMatrix2D(center, angle, 1.0)
    rotated = cv2.warpAffine(image, M, (w, h), flags=cv2.INTER_CUBIC,
borderMode=cv2.BORDER_REPLICATE)
    return rotated

#type of image
def get_type(type):
    print(type[6:])
    return type[6:]

#image resizing
def resizeImage(image):
    scale = 50
    width = int(image.shape[0] * scale/100)
    height = int(image.shape[1] * scale/100)
    dim = (width,height)
    return cv2.resize(image, dim, interpolation = cv2.INTER_AREA)

#conveting base to image
def data_uri_to_cv2_img(uri):
    image = Image.open(BytesIO(base64.b64decode(uri.split(',')[1])))
    return cv2.cvtColor(np.array(image), cv2.IMREAD_COLOR)
#main function
def recs(obj):
    image_string = "data:"+str(obj['type'])+";base64,"+str(obj['base64'])
    language = str(obj['language'])

    img = data_uri_to_cv2_img(image_string)
    # resized = resizeImage(img)
    # img = cv2.imread('image.jpg',cv2.IMREAD_COLOR)
    cv2.imshow('cnt',img)

```

```

cv2.waitKey(0)

im_gray = gray(img)
im_blur = blur(im_gray)
im_open = opening(im_blur)
im_thresh = threshold(im_open)
# noise_remove = remove_noise(im_open)

contours, _ = cv2.findContours(im_thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)
recognized_text = find_contours(im_thresh, contours, language)
beautified_text = beautify_text(recognized_text)
print(recognized_text)
return recognized_text

def find_contours(img, contours, language):
    config = ('--oem 3 --psm 3')
    result=""
    for cnt in contours:
        x, y, w, h = cv2.boundingRect(cnt)

        rect = cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 255), 2)
        cropped = img[y:y + h, x:x + w]

        if language == 'ukr':
            result = result + pytesseract.image_to_string(cropped,
config=config, lang='ukr')
            result = result.strip()
        elif language == 'rus':
            result = result + pytesseract.image_to_string(cropped,
config=config, lang='rus')
            result = result.strip()
        else:
            result = result + pytesseract.image_to_string(cropped,
config=config, lang='eng')
            result = result.strip()
    return result
#used for debugging
def beautify_text(string):
    return re.sub('\n', '', string)

```