НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

До захисту допущено:
Завідувач кафедри
Сергій СТІРЕНКО
«___»_____20__ р.

Дипломний проект
на здобуття ступеня бакалавра
за освітньо-професійною програмою «Комп'ютерні системи та мережі»
спеціальності 123 «Комп'ютерна інженерія»
на тему: «Система розпізнавання тексту на документах»

Виконав:
студент IV курсу, групи ІО-64
Владислав Юрійович БАНДУРІН _____

Керівник:
Асистент
Владислав Ігорович ТАРАН _____

Консультант з нормоконтролю:
Професор, доктор технічних наук
Валерій Павлович СІМОНЕНКО _____

Рецензент:
Доц. каф. АУТС к.т.н.
Андрій Володимирович ПИСАРЕНКО _____

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2020 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалаврський)
Спеціальність – 123 «Комп'ютерна інженерія»
Освітньо-професійна програма «Комп'ютерні системи та мережі»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Сергій СТІРЕНКО
«___»_____2020 р.

ЗАВДАННЯ
на дипломний проект студенту
Владиславу БАНДУРІНУ

1. Тема проекту «Система для розпізнавання тексту на документах», керівник проекту асистент Таран В.І., затверджені наказом по університету від «07» травня 2020 р. №1081-с.

2. Термін подання студентом проекту 31.05.2020

3. Вихідні дані до проекту технічна документація

4. Зміст пояснювальної записки

Огляд існуючих методів оптичного розпізнавання символів. Аналіз засобів для розробки програмного забезпечення. Проектування та розробка системи оптичного розпізнавання тексту. Моделювання розробленої системи.

5. Перелік графічного матеріалу

Блок схема алгоритму, схема роботи програми, діаграма класів.

6. Консультанти розділів проекту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Валерій СІМОНЕНКО		

7. Дата видачі завдання 01.09.2019

№ з/п	Назва етапів дипломного проекту	Строк виконання етапів проекту	Примітка
1	*Затвердження теми роботи*	*01.09.2019*	
2	*Вивчення та аналіз завдання*	*02.09.2019-10.04.2020*	
3	*Огляд існуючих методів оптичного розпізнавання символів*	*10.04.2020-01.05.2020*	
4	*Аналіз засобів для розробки програмного забезпечення*	*02.05.2020-15.05.2020*	
5	*Проектування та розробка системи оптичного розпізнавання тексту*	*16.04.2020-13.05.2020*	
6	*Тестування розробленої системи*	*14.05.2020-19.05.2020*	
7	*Оформлення матеріалів роботи*	*20.05.2020-25.05.2020*	
8	*Передзахист*	*26.05.2020*	
9	*Захист*		

Студент _____ Владислав БАНДУРІН
(підпис) (прізвище та ініціали)

Керівник проекту _____ Владислав ТАРАН
(підпис) (прізвище та ініціали)

Анотація

Робота присвячена розробці системи розпізнавання тексту для автоматизації обробки документів. У століття цифрової трансформації все більше сфер бізнесу починають використовувати сучасні технології розпізнавання тексту для автоматизації бізнес процесів і підвищення якості обслуговування клієнтів. У даній роботі досліджуються існуючі технології оптичного розпізнавання тексту, такі як: Google Vision, Tesseract і інші. Досліджено методи попередньої обробки зображень і пост-обробки результатів для поліпшення якості розпізнавання тексту. Розроблена система розпізнавання тексту з фотографій закордонних паспортів. Розроблена система дозволяє автоматизувати обробку документів і в перспективі здатна заощадити безліч людських ресурсів.

Abstract

The work is devoted to the development of a text recognition system for automating the processing of documents. In the age of digital transformation, more and more business areas are starting to use modern text recognition technologies to automate business processes and improve the quality of customer service. This paper explores existing optical text recognition technologies, such as Google Vision, Tesseract, and others. The methods of image preprocessing and post-processing of the results are studied to improve the quality of text recognition. A system for recognizing text from photographs of passports has been implemented. The implemented system allows you to automate the processing of documents and in the future can save many human resources.

№ рядка	Формат	Позначення	Найменування	Кількість	Примітка
1	А4		Завдання на дипломний проект	2	
2	А4	ІАЛЦ.467100.001 ВП	Відомість дипломного проекту	1	
3	А4	ІАЛЦ.467100.002 ТЗ	Технічне завдання	3	
4	А4	ІАЛЦ.467100.003 ПЗ	Пояснювальна записка	59	
5	А4	ІАЛЦ.467100.004 Д1	Принципова схема.	1	
6	А4	ІАЛЦ.467100.005 Д2	Функціональна схема.	1	
7	А4	ІАЛЦ.467100.006 Д3	Структурна схема	1	
8	А4	ІАЛЦ.467100.007 Д4	Лістинг програми	32	

Зм.	Арк.	№ докум.	Підпис	Дата	ІАЛЦ.467100.001 ВП		
Розробив		Бандурін В.Ю.			Система розпізнавання тексту на документах Відомість дипломного проекту	Літ.	Аркуш 1 / Аркушів 1
Перевір.		Таран В.І.					
Н. контр.		Симоненко В.П				НТУУ “КПІ”, ФІОТ, каф. ОТ, гр.ІО-64	
Затверд.		Таран В. І.					

Технічне завдання до дипломного проекту

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Дане технічне завдання розповсюджується на розробку системи розпізнавання тексту на документах.

Область застосування: система може бути використана у туристичних компаніях чи державних установах для автоматизації обробки закордонних паспортів клієнтів та економії часових та людських ресурсів..

2. ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на виконання бакалаврського дипломного проекту, затверджене кафедрою обчислювальної техніки Національного технічного університету України «Київський політехнічний інститут ім. Ігоря Сікорського».

3. МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою даного проекту є розробка системи розпізнавання тексту на документах.

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелами для розробки служать науково-технічна література з комп’ютерних технологій, публікації в періодичних виданнях, публікації в Інтернеті за даним питанням.

5. ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до розроблюваного продукту

Програмний продукт, що розробляється повинен:

- Мати інтерфейс для завантаження фотографії закордонного паспорту;
- Розпізнавати текст MRZ закордонного паспорту;
- Мати велику швидкість виконання розпізнавання;

- Мати змогу оброблювати фотографії, зроблені на непрофесійну фотокамеру;
- Показувати високу точність розпізнавання тексту.

5.2. Вимоги до програмного забезпечення

- Операційна система MS Windows XP, MS Windows Vista, MS Windows 7, MS Windows 8/8.1, MS Windows 10;
- Python 3 і вище;
- Telegram 2.1. і вище

5.3. Вимоги до апаратного забезпечення

- Комп’ютер на базі процесору Intel Pentium 2 і вище
- Оперативної пам’яті не менше 128 Мбайт

Пояснювальна записка
до дипломного проекту

на тему: Система розпізнавання тексту на документах

Київ - 2020 року

ЗМІСТ

ВСТУП

Актуальність теми

Завдання розпізнавання вмісту документів, що засвідчують особу, досить затребувана при реалізації програмних систем митного контролю, правоохоронних органів, банків, з метою автоматизації процесу обробки даних за допомогою сканування і розпізнавання документів. Різні галузі бізнесу можуть використовувати такі системи розпізнавання вмісту документів для покращення якості обслуговування своїх клієнтів.

Системою розпізнавання вмісту закордонних паспортів можуть користуватися туристичні компанії, які хочуть автоматизувати обробку документів своїх клієнтів та тим самим поліпшити якість обслуговування.

Мета і задачі дослідження

Метою роботи є дослідження існуючих технологій оптичного розпізнавання тексту і розробка системи розпізнавання тексту, а саме розпізнавання MRZ коду на закордонних паспортах громадян України.

Для досягнення поставленої мети були поставлені наступні основні задачі:

- Провести аналіз існуючих систем розпізнавання тексту;
- Дослідити структуру MRZ-коду закордонного паспорту;
- Провести тестування існуючих OCR систем для розпізнавання MRZ-коду закордонного паспорту;
- Дослідити основні методи попередньої обробки зображень та методи обробки результатів роботи OCR для покращення якості розпізнавання MRZ-коду закордонних паспортів;
- Розробити комплексну систему розпізнавання MRZ-коду закордонних паспортів на основі обраної OCR технології та методів покращення результатів розпізнавання;
- Дослідити отриманні результати.

Практичне значення

Розроблена система, використовуючи передові технології розпізнавання тексту, здатна автоматизувати рутинну роботу по обробці даних з закордонних паспортів. Ця система може бути використана у туристичних компаніях чи державних установах для автоматизації обслуговування клієнтів та економії часових та людських ресурсів.

РОЗДІЛ 1
ОГЛЯД ІСНУЮЧИХ МЕТОДІВ ОПТИЧНОГО РОЗПІЗНАВАННЯ СИМВОЛІВ

Оптичне розпізнавання символів (*optical character recognition, OCR*) – це технологія яка здатна локалізувати та розпізнавати текст у цифровому зображенні [1]. Незважаючи на те, що системи оптичного розпізнавання символів вже давно з’явилися на ринку, автоматизація розпізнавання документів тільки починає набирати обертів. На сьогоднішній день не існує досить потужних та якісних продуктів, які дозволили б сканувати та розпізнавати паспортні данні зі 100% точністю.

1.1. Введення в предметну область

Паспорт громадянина України для виїзду за кордон (Закордонний паспорт) є документом, що посвідчує особу, підтверджує громадянство України особи та дає право на виїзд з України і в'їзд в Україну.

Отримати паспорт громадянина України для виїзду за кордон має право кожен громадянин України [2]. Такий паспорт може бути оформлений з народження.

На сьогоднішній день в Україні активно впроваджуються біометричні закордонні паспорти. Ці паспорти містять машинозчитуванну зону (*MRZ)*, яка містить в собі закодовану інформацію про персональні дані, зазначену в закордонному паспорті. Наявність машинозчитуваної зони значно полегшує задачу розпізнавання тексту на паспорті, адже усі необхідні данні документа знаходяться в певному місці, яке легко локалізувати за допомогою комп’ютерного зору.

MRZ закордонного паспорту має структуру, яка відповідає міжнародному стандарту ISO/IEC 7501-1.

На рис. 1.1. зображена приклад біометричного закордонного паспорту громадянина України. Дане зображення є офіційним прикладом біометричного закордонного паспорту громадянина України [3].

У прикладі можна наглядно побачити, що усі необхідні паспортні дані наявні у двох нижніх строках паспорту (MRZ).

Рис. 1.1. Приклад біометричного паспорту громадянина України для виїзду за кордон

1.2. Опис структури MRZ

Дані машинозчитуваної зони складаються з двох рядів, у кожному з них по 44 символу. Використовуються тільки букви латинського алфавіту, цифри і знак-заповнювач «<» [4]. Формат першого рядку зазначений в таблиці 1.1.

Таблиця 1.1.

Формат першого рядку MRZ

Знак	Символи	Вимоги
1	Літери	P (означає паспорт)
2	Літери	Тип паспорта (необов'язковий, при відсутності ставиться знак-заповнювач <)
3-5	Літери	Країна-емітент або організація, що випустила паспорт)
6-44	Літери	Прізвище та ім'я, поділені двома знаками-наповнювачами. Частини складних імен та прізвищ поділяються одним заповнювачем.

Пропуски, дефіси та інші знаки пунктуації представлені <, за винятком апострофа, який пропущений. Якщо імена занадто довгі, вони скорочуються до їх найбільш значущих частин. Формат другого рядку зазначений в таблиці 1.2.

Таблиця 1.2.

Формат другого рядку MRZ

Знак	Символи	Вимоги
1-9	Літери та цифри	Номер паспорту

Таблиця 1.2. (продовження)

10	Цифри	Контрольна цифра для позицій 1-9
11-13	Літери	Громадянство
14-19	Цифри	Дата народження (РРММДД)
20	Цифри	Контрольна цифра для позицій 14-19
21	Літери	Стать (M, F or < чоловіча, жіноча чи не вказано відповідно)
22-27	Цифри	Дата закінчення терміну дії паспорта (РРММДД)
28	Цифри	Контрольна цифра для позицій 22-27
29-42	Літери та цифри	Особистий номер (необов'язково)
43	Цифри	Контрольна цифра для позицій 29-42 (може бути <, якщо всі символи є <)
44	Цифри	Контрольна цифра для позицій 1-10, 14-20, та 22-43

Розрахунок контрольної цифри в обох випадках полягає в наступному: кожна позиція присвоюється значення; для чисел від 0 до 9, це значення цифр, для букв від A до Z це від 10 до 35, для знака-заповнювача <- 0. Значення кожної позиції, потім множиться на вагу; вага першої позиції 7, на другому він дорівнює 3, і третій це 1, а після цього повторити ваги 7, 3, 1, і так далі. Всі значення підсумовуються, а залишок ділення отриманої суми на 10, це і є контрольна цифра [4].

За допомогою контрольних цифр зручно перевіряти результати розпізнавання MRZ та таким чином можна робити пост-обробку результатів розпізнавання тексту.

1.3. Огляд систем оптичного розпізнавання символів

Оптичне розпізнавання символів (OCR) – переклад зображень рукописного або друкованого тексту в текст, який використовується для відображення у комп’ютері [5].

Розпізнавання тексту широко використовують для перетворення книг і документів до електронного вигляду, автоматизації систем обліку в бізнесі або публікації тексту в інтернеті. Розпізнаний текст може бути відредагований, його зручно зберігати, здійснювати пошук слів чи фраз, демонструвати або роздруковувати матеріал, без втрати якості, проводити аналіз, а також робити електронний переклад тексту. Оптичне розпізнавання зображень, що містять текст, є широко досліджуваної проблемою на стику областей розпізнавання образів, штучного інтелекту та комп'ютерного зору [5].

На етапі обробки інформації у великих підприємствах, виникає задача введення великого об'єму текстової та графічної інформації в персональний комп'ютер. Основні пристрої, за допомогою яких відбувається введення графічної інформації це: фотокамера на мобільному пристрої, сканер, і, рідше, факс. Також, за допомогою програм оптичного розпізнавання текстів, можна оцифровувати також і текстову інформацію [5]. Сучасні системи розпізнавання тексту дозволяють автоматизувати ввід великих масивів даних у цифрову форму за допомогою фотокамери або сканеру і паралельного розпізнавання тексту.

Нижче наведені найбільш популярні системи розпізнавання символів.

1.3.1. ABBYY FineReader

ABBYY FineReader – програма, яка дозволяє швидко та якісно розпізнати текст з зображення та PDF-файлів в електронні редаговані формати. Точність розпізнавання тексту становить до 98% на якісних зображеннях без шуму. Програма дає можливість конвертувати зображення та PDF-файли в формати Word, HTML, Powerpoint, PDF/A, CSV, Rich Text Format и plain text файли. Програма підтримує 190 мов та для 48 и них здатна перевіряти орфографію [6].

1.3.2. SimpleOC

SimpleOCR – це популярне безкоштовне програмне забезпечення, яке дозволяє розпізнавати текст на французькій та англійській мовах. Програмне забезпечення є безкоштовним, але для розпізнавання нестандартні шрифти та зображення в низькій якості, то доведеться завантажити одне з чотирьох платних доповнень для програми [7].

SimpleOCR – досить проста програма, але для розпізнавання зображень низької якості доведеться завантажити доповнення, які в свою чергу також не гарантуюсь високої якості розпізнавання.

1.3.3. FreeOCR

FreeOCR – безкоштовна програма для розпізнавання відсканованого тексту. Дозволяє працювати як з файлами зображень, так і з pdf-файлами так сканером. Для сканування необхідно мати підключений сканер [8].

Основною перевагою програми є відсутність зайвих налаштувань та її повна автоматизація.

Підтримуються формати JPG, GIF, TIFF BMP і PDF (тільки перша сторінка, незабаром обіцяють підтримку перших 10 сторінок). Також, існує ліміт на 10 зображень в годину. Стверджується, що система здатна розпізнавати більшість східноєвропейських мов, в тому числі російський і український. Підтримка російської мови була анонсована в жовтні 2010 року для технології Tesseract 3.0.

1.3.4. OCRFeeder

OCRFeeder – являє із себе програму, що надає графічний інтерфейс для багатьох OCR систем, наприклад таких як Tesseract, GOCR і Ocrad [9]. OCRFeeder розповсюджується вільно для операційної системи Linux.

Використовуванні компоненти:

- PIl – python бібліотека для роботи з зображеннями;
- Unpaper – здійснює обробку зображення;
- Reportlab – здійснює імпорт pdf файлів;

- PyGTK & Libgnome – для графічного інтерфейсу;
- PyeEnchant – для перевірки орфографії.

Функції:

- Формати вхідних даних: PNG,JPEG, GIF, Portable anymap (PNM, PGM, PBM, PPM) та PDF-файли;
- Для вихідних даних використовують наступні формати: текстовий файл, HTML, PDF;
- Покращення якості розпізнавання за рахунок обробки зображень – фільтри чорного кольору та відтінків сірого, фільтрація шумів та інші види обробок;
- Перевірка орфографії;
- Отримання зображення зі сканера за допомогою додаткової програми SANE;
- Пакетне розпізнавання.

1.3.5. Tesseract OCR

Tesseract - це механізм розпізнавання тексту з підтримкою Unicode і здатністю розпізнавати більше 100 мов з коробки. Його можна навчити розпізнавати інші мови. Tesseract OCR - це проект з відкритим кодом, започаткований Hewlett-Packard. Пізніше Google взяв на себе розробку. Версія 4.0.0 заснована на LSTM (довготривала короткострокова пам'ять) [10].

Офіційна версія Tesseract OCR дозволяє розробникам створювати власну програму за допомогою API C або C ++. З часом розробники створили власні версії зовнішніх інструментів, обгортки та навіть навчальні проекти.

Для Python існує обгортка під назвою tesserocr. Хоча результати є доволі хорошими, для якісного розпізнавання тексту необхідно добавити ще попередню обробку зображення, а також подальша обробка тексту.

1.3.4. Google Cloud Vision API

Google Cloud Vision API дозволяє розробникам легко інтегрувати сучасні технології комп’ютерного зору в своїх додатках, включаючи

маркування зображення, розпізнавання обличчя та оптичне розпізнавання символів (OCR) [11].

Розпізнавання тексту виконує оптичне розпізнавання символів (OCR). Технологія виявляє та витягує текст із зображення із підтримкою широкого кола мов. Vision API також має автоматичну ідентифікацію мови.

Послуга розпізнавання тексту за допомогою Google Cloud Vision API є платною, але в місяць платформа дозволяє проводити 1000 API безкоштовних запитів на розпізнавання тексту, далі – 1,5$ за кожні 1000 запитів [11].

ВИСНОВКИ ДО РОЗДІЛУ 1

1. Аналіз сучасних технологій оптичного розпізнавання тексту показав, що для обраної задачі розпізнавання паспортів найбільш доцільно використовувати наступні технології:
    - Google Cloud Vision API
    - Tesseract OCR
2. Обидві обрані технології мають свої переваги та недоліки.
   Google Cloud Vision API має зручний у використанні інтерфейс та показує високу якість розпізнавання тексту, але є платним ($1.5 за 1000 запитів) та для роботи потребує доступ до інтернету.
   Tesseract OCR є безкоштовним фреймворком з відкритим кодом, для якого вже існують навчанні моделі для багатьох мов. Tesseract OCR не потребує доступу до інтернету та здатний працювати навіть на мобільних пристроях. Також, tesseract дозволяє перенавчати існуючі моделі для своїх даних та робити детальне налаштування для покращення роботи моделі. Недоліком можна зазначити більш складний інтерфейс для взаємодії та меншу точність розпізнавання тексту в порівнянні з Google Cloud Vision API.

РОЗДІЛ 2
АНАЛІЗ ЗАСОБІВ ДЛЯ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Будь-яка програма складається із графічного інтерфейсу та бізнес логіки. У випадку нашої системи бізнес логіка – це саме розпізнавання тексту. Графічний інтерфейс програми було вирішено зробити на платформі Telegram за допомогою зручного Bot API.

2.1 Огляд Tesseract OCR

Tesseract – OCR технологія з відкритим кодом, яка набуває популярності серед розробників систем оптичного розпізнавання тексту. Незважаючи на те, що іноді може бути дуже складно інтегрувати цю технологію в проекті, ще немає достатньо потужних безкоштовних альтернатив OCR.

Tesseract OCR, доступний за ліцензією Apache 2.0. Він може використовуватися безпосередньо або (для програмістів) за допомогою API для розпізнавання друкованого тексту із зображень. Він підтримує широкий спектр мов. Tesseract не має вбудованого графічного інтерфейсу, але може інтегруватися з декількома. Обгортки tesseract існують для багатьої мов програмування. Tesseract може використовуватися з наявним детектором тексту для розпізнавання тексту у великому документі, або його можна використати із зовнішнім детектором тексту для розпізнавання лише з зображення одного рядку тексту.

2.1.1. Принцип роботи Tesseract OCR

Tesseract 4.00 включає в себе нову підсистему нейронної мережі, сконфігуровану як розпізнавач текстових рядків. Щоб розпізнати зображення, що містить один символ, ми зазвичай використовуємо конволюційну нейронну мережу (CNN). Текст довільної довжини - це послідовність символів, і такі проблеми вирішуються за допомогою RNN та LSTM (популярна форма RNN). Використовуючи LSTM для навчання та навчаючи модель на великій кількісті даних та різних шрифтів, Tesseract досягає великої якості розпізнавання тексту. Тим не менш, ще він ще недостатньо потужний

для роботи із рукописним текстом і складними шрифтами. Існує можливість вдосконалення існуючих моделей на своїх даних, перенавчаючи верхні шари нейронної мережі.

Tesseract 4.00 включає в себе новий механізм розпізнавання на основі нейронної мережі, який забезпечує значно більшу точність (на зображеннях документів), ніж попередні версії, в обмін на значне збільшення необхідної обчислювальної потужності. Однак на складних мовах це може бути швидше, ніж базовий Tesseract [14].

Нейронні мережі вимагають значно більше даних про навчання і тренуються набагато повільніше, ніж базовий Tesseract. Для мов, що базуються на латинській мові, дані моделей були підготовлені за використовуючи близько 400000 текстових ліній, що охоплюють близько 4500 шрифтів. Тренування Tesseract 4.00 займає від кількох днів до декількох тижнів. Навіть з таким великим набором даних для тренування, результат роботи Tesseract для конкретної проблеми користувача може бути недостатнім, і тому існує можливість перенавчити існуючу модель [14].

Існує декілька варіантів навчання:

- Точна настройка існуючої моделі. Починаючи з існуючої навченої мови, тренування на конкретних додаткових даних користувача. Це може спрацювати з проблемами, близькими до існуючих навчальних даних, але дещо різними в деяких випадках, як, наприклад, незвичний шрифт. Може спрацювати навіть з невеликою кількістю навчальних даних;
- Відрізання верхнього шару від мережі та перетренування нового верхнього шару за допомогою нових даних. Якщо точна настройка не працює, це, швидше за все, наступний найкращий варіант. Вирізання верхнього шару все-таки може працювати для підготовки моделі для абсолютно нової мови;
- Перенавчання з нуля. Це непросте завдання, якщо немає дуже якісного і достатньо великого навчального набору даних для конкретної проблеми.

Якщо ні, то, швидше за все, у вас з’явиться нейронна мережа перенавчиться і буде добре відповідає навчальним даним, але не тестовим [14].

2.1.2. Попередня обробка зображення для Tesseract

Щоб уникнути зниження якості розпізнавання тексту за допомогою Tesseract, потрібно переконатися, що зображення належним чином попередньо оброблено.

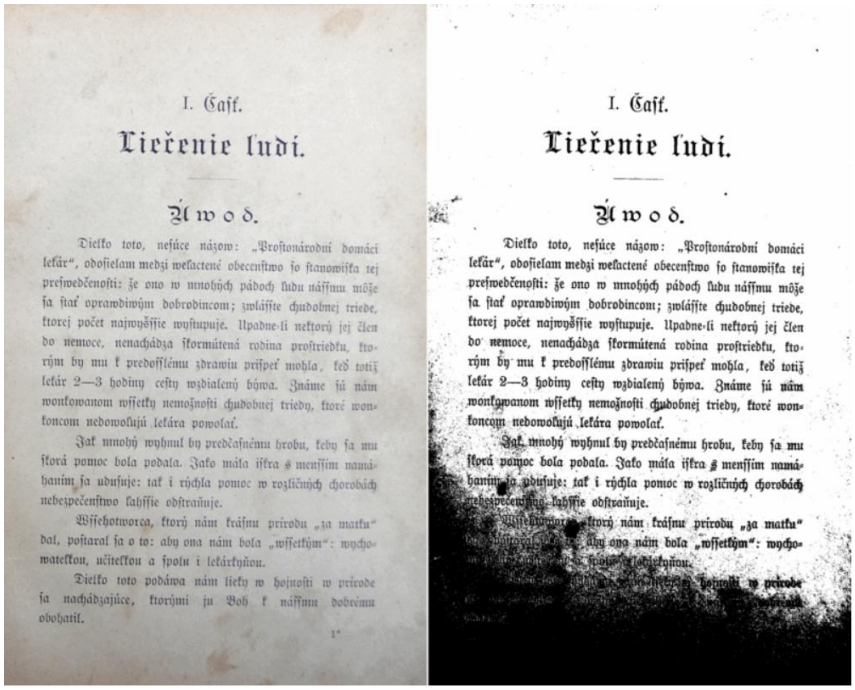
До попередньої обробки зображення входить інвертування, бінаризація, масштабування, видалення шуму та інші [15].

Далі будуть описані деякі з них:

- Інвертування зображень. У той час як tesseract версії 3.05 (і старшої) без проблем обробляє інвертоване зображення (темний фон та світлий текст), для версії 4.x слід використовувати темний текст на світлому фоні;
- Масштабування. Tesseract найкраще працює на зображеннях, які мають DPI не менше 300 dpi, тому може бути корисним масштабування зображень;
- Бінаризація. Це перетворення зображення в чорно-біле (Рис.2.1.). Tesseract робить це внутрішньо (алгоритм Оцу), але результат може бути неоптимальним, особливо якщо фон сторінки нерівномірний. Якщо немає можливості подати на вхід більш якісне зображення, то для вдосконалення розпізнавання можна використати інший алгоритм, наприклад, OpenCV Image Thresholding;
- Видалення шуму. Шум - випадкова зміна яскравості або кольору в зображенні, що може ускладнити читання тексту зображення. Певні типи шуму для Tesseract неможливо усунути на етапі бінарізації, що може призвести до зниження точності розпізнавання. Тому для покращення якості розпізнавання слід

впроваджувати більш складні методи видалення шуму, виходячи із специфіки датасету;

- Обертання. Якість сегментації рядків Tesseract значно знижується, якщо сторінка занадто перекошена, що сильно впливає на якість OCR (Рис.2.2.). Щоб вирішити цю проблему, потрібно повернути зображення сторінки так, щоб рядки тексту були горизонтальними [15].


I. Časť.

Liečenie ľudí.

Úvod.

Dielko toto, nesúce názov: „Prostonárodní domáci lekár“, obdarelam medzi welactené obecenstwo so stanowiska tej preswedčenosti: že ono w mnohých pádoch ľudu náššmu môže sa stať oprawdivým dobrodincom; zwlášste chudobnej triede, ktorej počet najwyššie wystupuje. Upadne-li niektorý jej člen do nemoce, nenachádza sformútená rodina prostriedku, ktorým by mu k predošlému zdraviu prispeť mohla, keď totiž lekár 2—3 hodiny cesty wzdialený býva. Známe sú nám wonkajšnom wšetky nemožnosti chudobnej triedy, ktoré wonkoncom nedowolujú lekára powolať.

Jak mnohý wyhnul by predčasnému hrobu, keby sa mu skorá pomoc bola podala. Jako mála iskra s menšším namáhaním sa udusuje: tak i rýchla pomoc w rozličných chorobách nebezpečenstwo ľahšie odstraňuje.

Wšemohúceho, ktorý nám krásnu prirodu „za matku“ dal, postaral sa o to: aby ona nám bola „wšetkým“: wychowateľkou, učiteľkou a spolu i lekárkyňou.

Dielko toto podáva nám lieky w hojnosti w prirode sa nachádzajúce, ktorými ju Boh k náššmu dobrému obohatil.

1*

I. Časť.

Liečenie ľudí.

Úvod.

Dielko toto, nesúce názov: „Prostonárodní domáci lekár“, obdarelam medzi welactené obecenstwo so stanowiska tej preswedčenosti: že ono w mnohých pádoch ľudu náššmu môže sa stať oprawdivým dobrodincom; zwlášste chudobnej triede, ktorej počet najwyššie wystupuje. Upadne-li niektorý jej člen do nemoce, nenachádza sformútená rodina prostriedku, ktorým by mu k predošlému zdraviu prispeť mohla, keď totiž lekár 2—3 hodiny cesty wzdialený býva. Známe sú nám wonkajšnom wšetky nemožnosti chudobnej triedy, ktoré wonkoncom nedowolujú lekára powolať.

Jak mnohý wyhnul by predčasnému hrobu, keby sa mu skorá pomoc bola podala. Jako mála iskra s menšším namáhaním sa udusuje: tak i rýchla pomoc w rozličných chorobách nebezpečenstwo ľahšie odstraňuje.

Рис.2.1. Приклад бінарізації

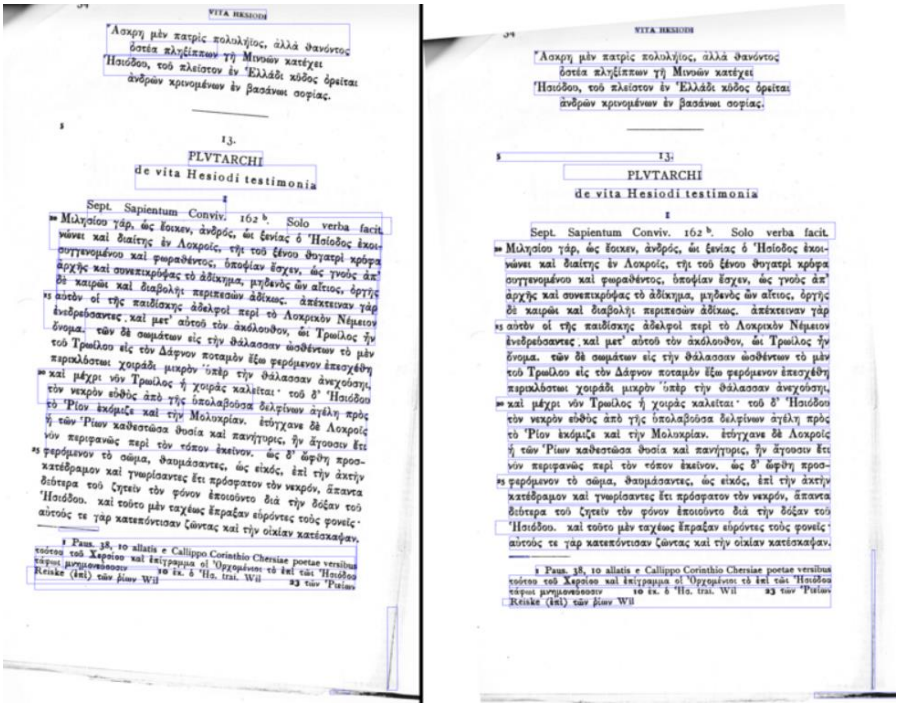
Також, існує кілька способів аналізу сторінки тексту. Tesseract api надає кілька режимів сегментації сторінок, якщо користувач хочете запустити OCR лише в невеликій області або в різних орієнтаціях тощо.

Далі наведений список із підтримуваних режимів сегментації сторінки за допомогою Tesseract [15]:

1. Тільки виявлення орієнтації (OSD);
2. Автоматична сегментація сторінки разом з OSD;
3. Автоматична сегментація сторінки без OSD;
4. Повністю автоматична сегментація сторінки без OSD (за замовченням);
5. Передбачається одна колонка тексту різного розміру;
6. Передбачається один єдиний блок вертикально вирівняного тексту;

7. Передбачається один однорідний блок тексту;
8. Трактування зображення як єдиний текстовий рядок;
9. Трактування зображення як єдине слово;
10. Трактування зображення як єдине слово по колу;
11. Трактування зображення як один символ;
12. Розріджений текст. Пошук якомога більше тексту у конкретній послідовності;
13. Розріджений текст з OSD;
14. Проста лінія. Трактування зображення як однієї текстової лінії.

За допомогою поданих режимів сегментації можна значно покращити якість розпізнавання, вибравши найбільш підходящий режим для конкретного набору даних.


VITA HESIODI

Ἄσκρη μὲν πατρὶς πολυλήϊος, ἀλλὰ θανόντος
ὅστέα πλησίπτων τῇ Μινῴων κατέχει
Ἡσιόδου, τοῦ πλείστον ἐν Ἕλλάδι κῦδος ὀρεῖται
ἀνδρῶν κρινομένων ἐν βασάνοι σοφίας.

13.

PLVTARCHI
de vita Hesiodi testimonia

Sept. Sapientum Conviv. 162 b. Solo verba facit.

Μιλησίου γάρ, ὡς ἔοικεν, ἀνδρός, ὡι ξενίας ὁ Ἡσίοδος ἐκοινώνει καὶ διαίτης ἐν Λοκροίς, τῆι τοῦ ξένου θυγατρὶ κρόφα συγγενομένου καὶ φωραθέντος, ὑποφίαν ἔσχεν, ὡς γνοὺς ἀπ' ἀρχῆς καὶ συνεπικρύφας τὸ ἀδίκημα, μηδενὸς ὢν αἴτιος, ὀργῆς δὲ καιρῶι καὶ διαβολῆι περιπεσὼν ἀδίκως. ἀπέκτειναν γάρ αὐτὸν οἱ τῆς παιδίσκης ἀδελφοὶ περὶ τὸ Λοκρικὸν Νέμειον ἐνεδρεύσαντες, καὶ μετ' αὐτοῦ τὸν ἀκόλουθον, ὠι Τρωίλος ἦν ὄνομα. τῶν δὲ σωμάτων εἰς τὴν θάλασσαν ὠσθέντων τὸ μὲν τοῦ Τρωίλου εἰς τὸν Δάφνον ποταμὸν ἔξω φερόμενον ἐπεσχέθη περικλύστωι χοιράδι μικρὸν ὑπὲρ τὴν θάλασσαν ἀνεχούσηι, καὶ μέχρι νῦν Τρωίλος ἡ χοιρὰς καλεῖται· τοῦ δ' Ἡσιόδου τὸν νεκρὸν εὐθὺς ἀπὸ τῆς ὑπολαβούσης δελφίνων ἀγέλη πρὸς τὸ Ῥίον ἐκόμιζε καὶ τὴν Μολυκρίαν. ἐτύγχανε δὲ Λοκροὶς ἡ τῶν Ῥίων καθεστῶσα θυσία καὶ πανήγυρις, ἣν ἄγουσιν ἔτι νῦν περιφανῶς περὶ τὸν τόπον ἐκεῖνον. ὡς δ' ὤφθη προσφερόμενον τὸ σῶμα, θαυμάσαντες, ὡς εἰκός, ἐπὶ τὴν ἀκτὴν κατέδραμον καὶ γνωρίσαντες ἔτι πρόσφατον τὸν νεκρόν, ἅπαντα διὼτερα τοῦ ζητεῖν τὸν φόνον ἐποιοῦντο διὰ τὴν δόξαν τοῦ Ἡσιόδου. καὶ τοῦτο μὲν ταχέως ἔπραξαν εὑρόντες τοὺς φονεῖς· αὐτοὺς τε γὰρ κατεπόντισαν ζῶντας καὶ τὴν οἰκίαν κατέσκαψαν.

1 Paus. 38, 10 allatis e Callippo Corinthio Chersiae poetae versibus τούτου τοῦ Χερσίου καὶ ἐπίγραμμα οἱ Ὀρχομένιοι τὸ ἐπὶ τῶι Ἡσιόδου τάφωι μνημονεύουσιν 10 ex. ὁ Ἡσ. trai. Wil 23 τῶν Ῥίων Reiske (ἐπὶ) τῶν βίων Wil

Рис.2.2. Приклад перекошеного зображення

2.2. Google Vision OCR API

Vision API може виявляти та витягувати текст із зображень. Є дві функції анотацій, які підтримують оптичне розпізнавання символів (OCR):

- TEXT_DETECTION виявляє та витягує текст із будь-якого зображення. Наприклад, фотографія може містити дорожній знак або дорожній знак.

JSON включає весь витягнутий рядок, а також окремі слова та їх обмежувальні поля. Може виявляти та витягувати текст із зображень. Є дві функції анотацій, які підтримують оптичне розпізнавання символів *(OCR*)[16].

- DOCUMENT_TEXT_DETECTION також витягує текст із зображення, але відповідь оптимізована для документів. JSON включає інформацію про сторінку, блок, абзац, слово та розрив[17].

Приклади роботи двох функцій можна побачити на Рис. 2.7. та Рис. 2.8.

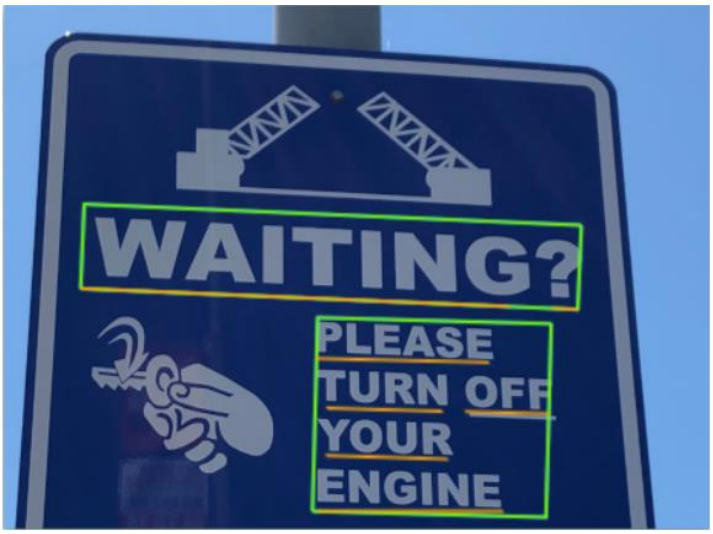


Рис. 2.7. Приклад роботи функції *TEXT_DETECTION*.

Рис. 2.8. Приклад роботи функції *DOCUMENT_TEXT_DETECTION*.

Ми маємо задачу розпізнавання тексту на документі, отже найбільш правильно буде використовувати функцію *DOCUMENT_TEXT_DETECTION*, яка дасть більше інформації про текст документу.

2.2.1. Анотування зображення за допомогою OCR Text Document.

У цій роботі було розроблено додаток, який робить DOCUMENT_TEXT_DETECTION запит, а потім обробляє об'єкт fullTextAnnotation відповідь.

Об'єкт **fullTextAnnotation**- структурована ієрархічна відповідь на текст UTF-8, витягнутий із зображення, організований як Сторінки → Блоки → Абзаци → Слова → Символи:

- **Page** – це сукупність блоків, а також метаінформація про сторінку: розміри, роздільна здатність (роздільна здатність X та роздільна здатність Y можуть відрізнятися);
- **Block** являє собою один "логічний" елемент сторінки - наприклад, область, охоплена текстом, або зображення або роздільник між стовпцями. Блоки тексту та таблиці містять основну інформацію, необхідну для вилучення тексту;
- **Paragraph** є структурною одиницею тексту, що представляє впорядковану послідовність слів. За замовчуванням слова вважаються відокремленими розривами слів;
- **Word** – найменша одиниця тексту. Він представлений у вигляді масиву Symbols;
- **Symbol** представляє символ або розділовий знак.

Також, fullTextAnnotation може надавати URL - адреси для веб - зображень, які частково або повністю відповідають зображення в запиті [17].

Ця ієрархічна структура відповіді дуже корисна для обробки результатів розпізнавання тексту для виявлення необхідних текстових зон (у нашому випадку, MRZ зони).

2.2.2. Використання даних та безпека.

Важливою вимогою в розробці системи залишається конфіденційність даних. Користувач повинен бути впевненим, що його дані не будуть ніде зберігатися та використовуватись не за прямим призначенням.

Саме тому, Google гарантує, що компанія не використовує будь-який вміст запиту (наприклад, зображення та мітки) ні для яких цілей, крім того, щоб надати послугу Vision API [18].

Спираючись на офіційну документацію, можна виділити наступні тези у використанні даних Google:

- Google не будемо робити зображення, яке ви надсилаєте, доступним для публіки, або надавати доступ до зображень нікому іншому;
- Для операцій в режимі он-лайн (негайне реагування) дані зображення обробляються в пам'яті і не зберігаються на диску. Google також тимчасово реєструє деякі метадані щодо ваших запитів на API Vision (наприклад, час отримання запиту та розмір запиту), щоб покращити наш сервіс та боротися зі зловживаннями;
- Наразі Google не використовує контент, який ви надсилаєте для тренувань та вдосконалення наших функцій Cloud Vision, таких як його модель сприйняття машини;
- Google не претендує на право власності на будь-який вміст (включаючи зображення та мітки), який ви передаєте в API Vision [18].

Враховуючи усі зазначені пункти, можна зробити висновок, що в процесі роботи нашої платформи можна не хвилюватися за безпеку паспортних даних, використовуючи Google Vision API.

2.3. Telegram Bot API.

Telegram Боти - це спеціальні додатки, які працюють на серверах Telegram. Користувачі можуть взаємодіяти з Telegram ботами, за допомогою повідомлення, команди та вбудованих запитів . Керування ботами через код відбувається за допомогою HTTPS-запитів до API Bot [19].

Користувачі можуть взаємодіяти з Telegram ботами у наступні два способи:

- Надсилаючи повідомлення або команди Telegram ботам, через чат з ними та додавши їх до груп.
- Надсилати безпосередньо запити з поля введення. Для цього потрібно ввести @username бота та сам запит. Це дозволяє надсилати повідомлення від вбудованих ботів до будь-якого чату, групи чи каналу.

Коли користувач надсилає повідомлення, запити чи команди данні передаються до серверів розробників, де програмне забезпечення їх оброблює. Telegram виступає як сервер-посередник бере на себе обробку шифрування та робить зв'язок з Telegram API, призначеного для розробників.

Взаємодія з сервером відбувається за допомогою простого HTTPS-інтерфейсу (API Bot). Цей інтерфейс являє собою спрощену версію самого API Telegram [19].

2.3.1. Авторизація бота

Кожному боту при створенні присвоюється унікальний токен виду *123456:ABa-DEaa1434ghIal-zyr57q2v1u123ew11*. У документації для простоти замість нього буде використовуватися <token>. Токен для нового бота можна отримати за допомогою боту BotFather [20].

2.3.2. Відправлення запитів

Здійснення запитів до Telegram Bot API повинно бути здійснено за допомогою HTTPS у наступному форматі:

Всі запити до Telegram Bot API повинні здійснюватися через HTTPS в наступному вигляді:

https://api.telegram.org/bot<telegram-bot-token>/назва_методу.

наприклад:

https://api.telegram.org/bot320356:asdf0987ghIasdf7W2v1uasdfw11/sendMessage

Можна робити GET та POST запити. Передача параметрів у запит здійснюється у наступні способи:

- *Запит через URL;*
- *Запит application\x-www-form-urlencoded;*
- *Запит application\json (не використовується при завантаженні файлів);*
- *Запит multipart\form-data (використовується при завантаженні файлів).*

На виході отримуємо JSON-об'єкт відповіді, у якому завжди буде логічне поле «ОК» та строкове поле description, що містить опис результату виконання запиту. У випадку, коли поле «ОК», то результат виконання запиту є успішним і його можна побачити в полі «Результат». У випадку, коли ми отримуємо помилку, поле «ОК» буде дорівнювати false, а про саму помилку можна буде прочитати у полі description. Також, у відповіді буде міститись error_code, який дорівнює цілому числу, яке відповідає конкретній помилці [20].

2.3.3. Відправлення запитів при отриманні поновлення

Щоб бот працював через вебхукі, треба здійснити то ви можете зробити HTTPS запит до Bot API, відправивши відповіді на вебхук. Передача параметрів у запит здійснюється у наступні способи [20]:

- *application/json*
- *application/x-www-form-urlencoded*
- *multipart/form-data*

Метод, що викликається має бути прописаний в запиті в полі *«Метод»*. Але треба мати на увазі, що дізнатися про статус та результат запиту у такий спосіб неможливо.

2.3.4. Отримання оновлень

Для отримання оновленнь існує дава методи: через getUpdates та вебхукі. Дані на вході зберігаються на серверах телеграму до їх кінцевої обробки, або по закінченню 24 годин.

Далі описані основні методи Telegram Bot API, які були використані у процесі розробки програми.

Отримана відповідь приходить і вигляді об'єктів *Update* в JSON. Ця відповідь не залежить від способу отримання оновлень.

Update представляє із себе вхідне оновлення. Під оновленням мається на увазі дію, вчинене з ботом - наприклад, отримання повідомлення від користувача.

GetUpdates використовують як спосіб отримання оновлень за допомогою long polling. У відповіді приходить масив із *Update*.

SetWebhook необхідний для завдання URL вебхука, на який бот буде відправляти оновлення. Кожен раз при отриманні поновлення на цю адресу буде відправлений HTTPS POST з серіалізовані в JSON об'єктом *Update* . При невдалий запит до вашого сервера спроба буде повторена помірне число раз [20].

Для більшої безпеки рекомендується включити токен в URL вебхука, наприклад, так: https://yourwebhookserver.com/<token>. Так як ніхто сторонній не знає вашого токена, ви можете бути впевнені, що запити до вашого вебхуку шле саме Telegram [20].

Для роботи з Telegram Bot API існують багато бібліотек під різні мови програмування, у тому числі і для Python.

Об'єкт *InputFile* представляє вміст файлу, який потрібно завантажити. Повідомлення повинні бути розміщені за допомогою даних із кількох частин у звичайний спосіб завантаження файлів через браузер.

Існує два способи надсилання файлу. Якщо це новий файл, ви можете завантажити його, використовуючи дані форми. Якщо файл вже є на серверах телеграму, вам не потрібно його повторно завантажувати: у кожного об’єкта файлу є поле file_id, можна просто передати цей file_id як параметр.

Метод *sendMessage* дозволяє відправити текстове повідомлення.

Метод *getFile*, використовується щоб отримати основну інформацію про файл та підготувати його до завантаження. На даний момент боти можуть завантажувати файли розміром до 20 МБ. Після успіху повертається об’єкт *File*. Потім файл можна завантажити за посиланням

https://api.telegram.org/file/bot<telegram_bot_token>/<path_to_file>, де з відповіді взято < path_to_file>. Гарантується, що посилання буде дійсною не менше 1 години. Коли посилання закінчується, нове можна отримати, надіславши запит *getFile* ще раз.

2.4. Python-telegram-bot

Для більш зручної та ефективної розробки телеграм ботів існують багато бібліотек для роботи з Telegram bot API. У цій роботі ми будемо використовувати найвідомішу з них - python-telegram-bot [21].

Бібліотека python-telegram-bot забезпечує чистий інтерфейс Python для API Telegram Bot . Він сумісний з версіями Python 3.5+ та PyPy . Всі типи та методи API Telegram Bot API 4.8 підтримуються [22].

Окрім чистої реалізації API, у цій бібліотеці є ряд класів високого рівня, щоб зробити розробку ботів легкою та зрозумілою. Ці класи містяться в telegram.ext модулі [22].

ВИСНОВКИ ДО РОЗДІЛУ 2

1. Аналіз засобів для розробки системи оптичного розпізнавання тексту на фотографіях закордонних паспортів показав, що найбільш доцільно використовувати Google Vision API у якості технології для розпізнавання тексту та Telegram Bot API у якості зручного графічного інтерфейсу користувача;
2. Обрані технології гарантують надійну та безперебійну роботу програми та безпеку персональних даних;
3. В якості мови програмування була обрана мова Python, оскільки вона має підтримку бібліотек для роботи з Google Cloud Platform та Telegram Bot API.

РОЗДІЛ 3
ПРОЕКТУВАННЯ ТА РОЗРОБКА СИСТЕМИ ОПТИЧНОГО РОЗПІЗНАВАННЯ ТЕКСТУ

В даній роботі розроблена система оптичного розпізнавання тексту на зображеннях закордонних паспортів базі технології Google Cloud Vision API. У якості графічного інтерфейсу системи було прийнято рішення використовувати Telegram bot, оскільки Telegram дозволяє швидко та зручно ділитися медіа файлами та автоматизувати взаємодію з користувачем за допомогою безкоштовного Bot API [20].

3.1 Загальний опис системи та принципу її роботи

Реалізована в рамках дипломного проекту система, написана на мові програмування Python, працює на віддаленому сервері та інтегрована з платформою Telegram для легкої взаємодії з користувачем.

На вхід користувач подає фотографію закордонного паспорту.

На виході користувач отримує наступні данні з паспорту:

- Прізвище та ім’я;
- Номер паспорту;
- Код країни;
- Дата народження;
- Стать;
- Дата закінчення дії;
- Дата закінчення терміну дії паспорта;
- Особистий номер (не обов’язково).

Принцип роботи системи:

1. Користувач відсилає фотографію закордонного паспорта у telegram bot;
2. Запит від telegram потрапляє на сервер, який його оброблює та отримує зображення;
3. Отримане зображення проходить попередню обробку;

4. Після попередньої обробки, зображення потрапляє до Google Vision. API для виконання розпізнавання тексту;
5. Отриманні від Google Vision API данні про текст на зображенні оброблюються спеціальним образом для виділення тексту машино зчитувальної зони (MRZ);
6. Текст MRZ проходить перевірку спеціальним модулем перевірки правильності розпізнавання всіх символів на основі контрольних символів;
7. Після перевірки бот відправляє користувачу текст MRZ.

На Рис. 3.1 можна побачити приклад роботи Telegram bot.

Рис. 3.1. Ілюстрація роботи платформи в telegram bot.

Структурна схема роботи системи зображена на Рис 3.2.

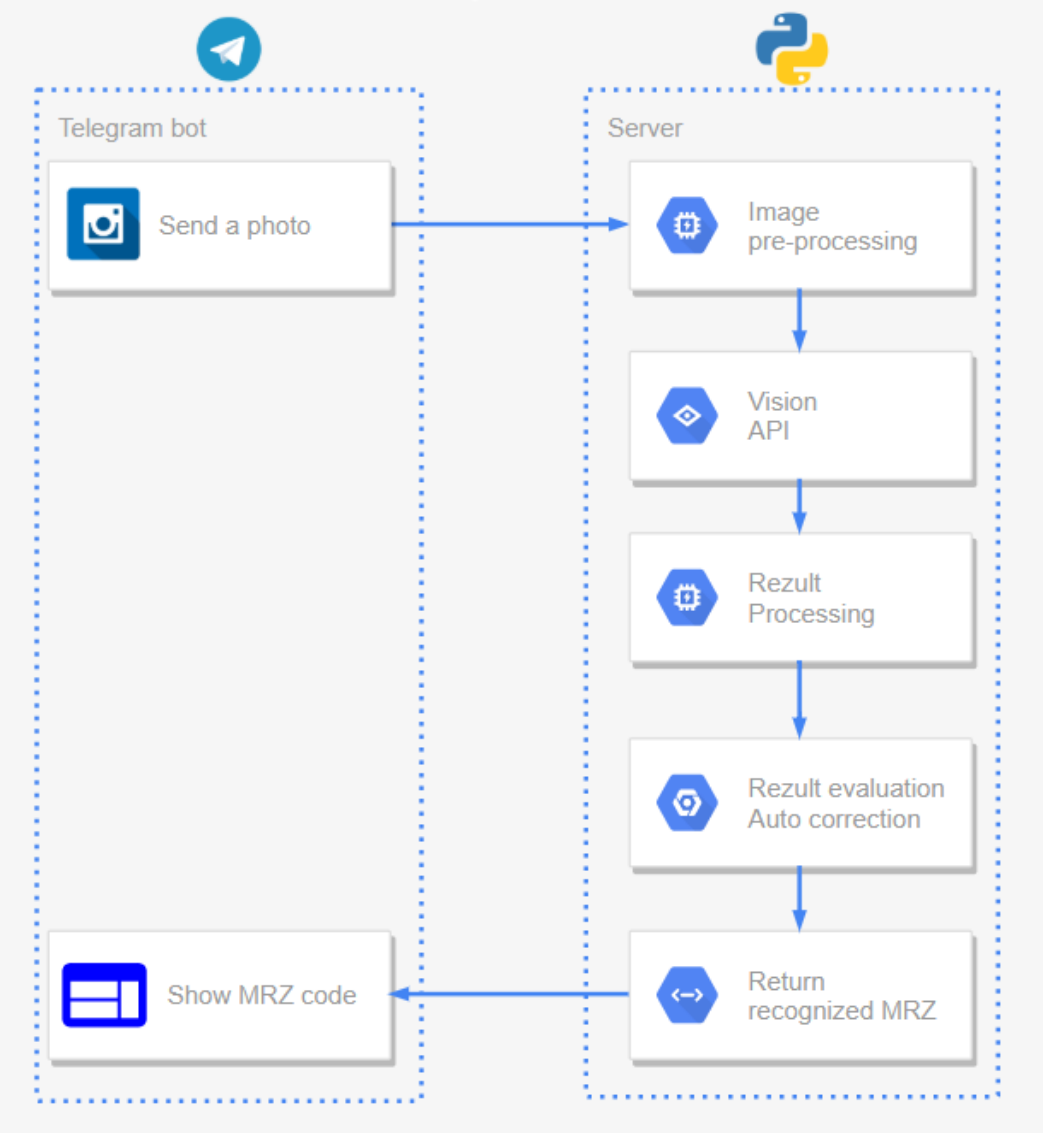


Рис 3.2. Структурна схема роботи системи.

3.2. Налаштування Google Vision API.

У цьому підрозділі описані етапи налаштування для початку використання Google Vision API.

Взаємодія зі всіма API Google відбувається за допомогою Google Cloud Console (GCP). GCP являє собою веб-інтерфейс та використовується для забезпечення, налаштування, управління та моніторингу систем, які використовують GCP продукти [23]. У цій роботі ми використовуємо GCP для отримання доступу та управління ресурсами Vision.

3.2.1. Створення проекту

Для того, щоб почати користування послугами GCP потрібно створити проект.

Проект потрібен для організації всіх ресурсів GCP. Проект складається з наступних компонентів:

- Набір співробітників;
- Підключені API;
- Інструменти моніторингу;
- Платіжна інформація;
- Контроль автентифікації та доступу [23].

Можна створити один або декілька проектів. Проекти можна використовувати для організації ресурсів GCP.

Для того, щоб створити проект на Cloud Console на сторінці вибору проекту потрібно на сторінці вибору проекту натиснути «Create a Cloud project» та вказати унікальне ім’я для проекту. Після успішного створення проекту можна побачити меню проекту GCP Рис.3.3.

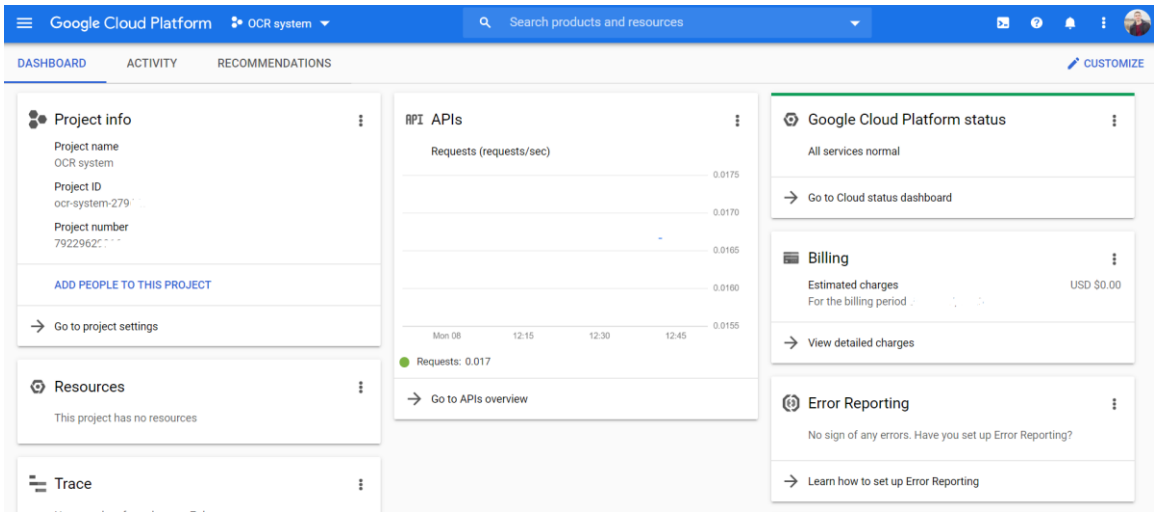


Рис.3.3. Dashboard проекту GCP

Після створення нового проекту необхідно підключити платіжний рахунок. Він визначає хто платить за певний набор ресурсів. Ці рахунки можуть бути пов’язані з одним або з декількома проектами. Налаштування виставлення рахунків відбувається під час створення проекту.

Для налаштування платіжної інформації, у розділі «Billing» потрібно вибрати пункт меню «Payment method» та прив’язати платіжну картку.

3.2.2. Підключення API

Після успішного підключення платіжної інформації, потрібно Vision API для свого проекту. Для цього у панелі приладів проекту потрібно вибрати «Go to APIs overview», знайти Vision API та включити його для проекту. Після цього з’явиться вікно управління API Рис.3.4.

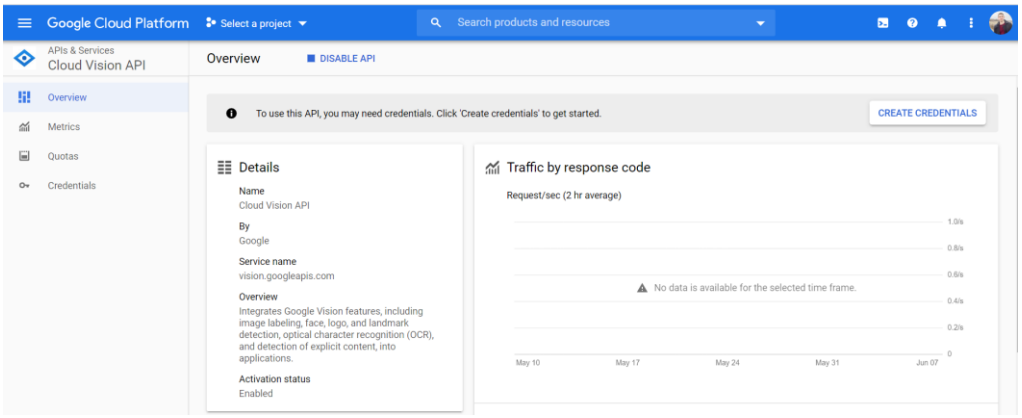


Рис.3.4. Вікно управління Cloud Vision API

3.2.3. Налаштування аутентифікації

Будь-яка клієнтська програма, яка використовує API, повинна бути аутентифікована та наданий доступ до запитуваних ресурсів (Рис.3.5.).

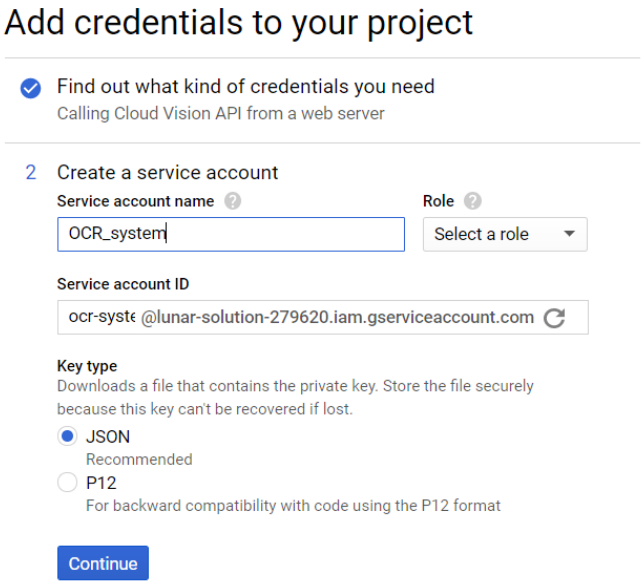


Рис.3.5. Налаштування аутентифікації для Cloud Vision API

Щоб створити API-token, вікні управління Cloud Vision API потрібно:

1. Натиснути «Create credentials»;
2. Вибрати Vision API у списку;
3. Вказати ім’я акаунту та його роль;
4. Завантажити json з ключем для доступу до API.

3.2.4. Використання Cloud Vision API

Для того, щоб надати облікові дані автентифікації коду програми, необхідно вказати шлях до файлу з ключем доступу встановивши змінну середовища GOOGLE_APPLICATION_CREDENTIALS.

Далі потрібно встановити та ініціалізувати хмарний SDK. Cloud SDK - це набір інструментів, які можна використовувати для управління ресурсами та програмами, розміщеними в GCP. Сюди входить інструмент gcloud командного рядку [23].

Для встановлення клієнтської бібліотеки для python для роботи з GCP потрібно у командному рядку виконати наступну команду:

- *pip install --upgrade google-cloud-vision.*

Після успішного виконання усіх наведених інструкцій, можна використовувати Vision API за допомогою python бібліотеки google-cloud-vision. Приклад використання пожна побачити на Рис.3.6.

```
import io
import os

# Imports the Google Cloud client library
from google.cloud import vision
from google.cloud.vision import types

# Instantiates a client
client = vision.ImageAnnotatorClient()

# The name of the image file to annotate
file_name = os.path.abspath('resources/wakeupcat.jpg')

# Loads the image into memory
with io.open(file_name, 'rb') as image_file:
    content = image_file.read()

image = types.Image(content=content)

# Performs document_text_detection on the image file
response = client.document_text_detection(image=image)
text = response.full_text_annotation
```

Рис.3.6. Приклад розпізнавання тексту за допомогою python та Vision API

3.3. Отримання зображення від Telegram bot.

Для отримання зображення була реалізована функція *get_photo*, за допомогою методу *get_file* бібліотеки python-telegram-bot, який дозволяє отримати файл зображення з об'єкта повідомлення.

Отримане зображення надсилається до методу *get_text_recognition* для розпізнавання тексту.

3.4. Розпізнавання тексту

Розпізнавання тексту відбувається за допомогою запиту DOCUMENT_TEXT_DETECTION до Vision API.

Метод *get_text_recognition* (Рис.3.7.) приймає на вхід зображення та повертає json з результатом розпізнавання тексту з зображення.

```
def get_text_recognition(img=None, img_path=None):
    """Detects document features in an image."""
    client = vision.ImageAnnotatorClient()

    if img_path:
        with open(img_path, 'rb') as image_file:
            content = image_file.read()
    else:
        content = img
    image = vision.types.Image(content=content)
    response = client.document_text_detection(image=image)
    return response
```

Рис.3.7. Реалізований метод *get_text_recognition*

На вході ми подаємо зображення з текстом, на виході – отримуємо json з наступною ієрархією:

Сторінки → Блоки → Абзаци → Слова → Символи

На кожному ступені ієрархії Google vision віддає наступні параметри:

- **Bounding box** – координати прямокутної зони тексту;
- **Confidence** – впевненість у результаті розпізнавання;
- **Text** – текст зони.

Приклад вихідних даних розпізнавання закордонного паспорту за допомогою Google vision можна побачити на Рис.3.8.

Рис.3.8. Приклад вихідних даних розпізнавання закордонного паспорту за допомогою Google vision.

3.5. Обробка результатів розпізнавання тексту.

Для обробки результатів розпізнавання тексту був створений метод *get_ocr_rezult_processing*, який знаходить текст, що відповідає MRZ, у структурному json, який був отриманий від *get_text_recognition* методу.

Метод *get_ocr_rezult_processing* приймає json з результатом розпізнавання тексту та повертає словник, який містить інформацію про текст верхньої та нижньої строки MRZ, а також інформацію про координати та ймовірність кожного символу.

Пошук текстової зони, що відповідає MRZ відбувається на підставі самої структури MRZ. Визначним фактором для виявлення потрібної зони є наявність символу «<» у тексті цієї зони, оскільки цей символ завжди присутній у тексті MRZ.

Далі, у методі відбувається поділ MRZ зони на верхню та нижню строки та видача результату.

3.6. Оцінка результатів розпізнавання.

Дуже важливо, щоб користувач був впевнений, що всі символи було розпізнано правильно та без помилок. Тому етап перевірки результатів розпізнавання є необхідним.

На цьому етапі ми хочемо перевірити чи правильно було розпізнано кожен символ MRZ за допомогою контрольних цифр (усі контрольних цифри описаних у підрозділі 1.2. Опис структури MRZ).

Для другої строки MRZ використовуються 5 контрольних цифр:

- Контрольна цифра для позицій 1-9 (розташована на позиції 10);
- Контрольна цифра для позицій 14-19 (розташована на позиції 20);
- Контрольна цифра для позицій 22-27 (розташована на позиції 28);
- Контрольна цифра для позицій 29-42 (розташована на позиції 43);
- Контрольна цифра для позицій 1-10, 14-20 та 22-43 (розташована на позиції 44).

Для обрахунку контрольної цифри була створена функція *get_check_number* (Рис.3.9.), яка приймає на вхід строку із символів та повертає значення контрольної цифри для цієї строки.

```
def get_check_number(text):
    sum = 0
    for i in range(len(text)):
        x_dict = {0: 7,
                  1: 3,
                  2: 1}
        ind = i % 3
        x = x_dict[ind]
        n = text[i]
        if n.isdigit():
            sum += int(n) * x
        else:
            sum += common.later_dict[n] * x
    return sum
```

Рис.3.9. Реалізація методу *get_check_number*

Функція get_check_number реалізує наступну логіку:

1. Кожній позиції присвоюється значення:
   - Для чисел від 0 до 9, це значення цифр;
   - Для букв від A до Z це значення від 10 до 35;
   - Для знаку-заповнювача <- 0;
2. Значення кожної позиції потім множиться на вагу. Вага першої позиції 7, на другому вона дорівнює 3, і на третій це 1, а після цього повторити ваги 7, 3, 1, і так далі;
3. Всі значення підсумовуються, а залишок ділення отриманої суми на 10, це і є контрольна цифра.

Для перевірки результатів розпізнавання MRZ було створено метод *check_mrz,* який приймає на вхід текст MRZ та повертає True – якщо текст проходить усі перевірки, або False в іншому випадку.

3.7. Автоматична корекція.

Основною причиною, по якій результат розпізнавання не проходить перевірку за контрольними символами є те, що OCR модель іноді плутає схожі символи.

Нижче наведені деякі пари із символів, які модель найчастіше плутає в процесі розпізнавання:

- O та 0;
- M та H;
- K та <;
- S та 5;
- A та R;
- D та O;
- U та O;
- G та O;
- Q та O;
- 7 та I;

- J та I.

Це пояснюється тим, що на фотографії, зробленій на деякій відстані або при поганому освітлені перелічені пари символів можуть бути дуже схожими за написанням.

Знаючи пари символів, які модель часто плутає та знаючи структуру MRZ, описану у розділі 1, коду, можна покращити якість моделі за допомогою автокорекції результату.

За допомогою впровадження додаткових логічних правил перевірки та корекції результатів в розробленій системі вдалося значно покращити якість розпізнавання даних, навіть коли OCR робить помилки.

Наприклад, знаючи що у полі «Дата народження» повинні бути лише цифри, а модель розпізнала там символ «S», то можна спробувати на етапі автокорекції замінити символ «S» на символ «5» та перевірити отриману послідовність символів на правильність за допомогою контрольних символів.

Також, знаючи що у першому рядку є лише літери, то у випадку наявності у першому рядку цифри, ми можемо замінити цю цифру на найбільш схожу букву.

Наприклад, цифру «0» можна замінити на букву «O».

Описаний приклад автоматичної корекції результату для поля «Дата народження» зображений на Рис.3.3.

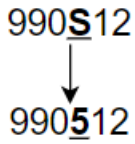


Рис.3.3. Приклад автоматичної корекції для поля «Дата народження» у MRZ

Для оцінки та автокорекції ще можна використовувати той факт, що коди країн, які розташовані на позиціях 3-5 у першому рядку та на позиціях 11-13 у другому рядку, вказані згідно зі стандартом ISO-3166 та завжди складаються із 3-х літер. Цей факт дуже корисний для автокорекції, оскільки

за допомогою нього можна із повною впевненістю робити автокорекції помилок для цього поля.

Для отримання бази усіх кодів країн за стандартом ISO-3166 була використана бібліотека *pycountry* [23]. Ця бібліотека містить коди 249 країн за стандартом ISO-3166. Приклад роботи бібліотеки зображено на Рис.3.10.

```
>>> import pycountry
>>> len(pycountry.countries)
249
>>> Ukraine = pycountry.countries.get(alpha_3='UKR')
>>> Ukraine
Country(alpha_2='UA', alpha_3='UKR', name='Ukraine', numeric='804')
```

Рис.3.10. Приклад роботи бібліотеки pycountry

Якщо на етапі перевірки було виявлено, що коду країни, який був отриманий у результаті розпізнавання тексту, не існує в базі, спрацьовує алгоритм автокорекції.

Наприклад, в результаті розпізнавання тексту було отримано код країни «OKR», якого немає в базі та припустимо, що модель видала найменшу ймовірність для символу «O». Враховуючи, що символ «O» модель часто плутає з символами «D», «U», «G» , «Q», ми можемо підставити ці символи замість «O» та перевірити чи є отримані коди країн в базі. Якщо ми таким чином отримали код, який існує в базі, ми підставимо цей код у MRZ та ще раз перевіримо за допомогою відповідної контрольної цифри. І тільки якщо результат автокорекції пройшов перевірку, ми зможемо зробити заміну. Цей алгоритм перевірки дасть змогу забезпечити правильність автокорекції.

3.7. Надсилання результату розпізнавання MRZ користувачу.

Після проходження усіх перевірок, результат розпізнавання надсилається користувачу.

Для цього реалізовано метод *send_mrz_recognition,* який за допомогою методу *send_message* бібліотеки python-telegram-bot відправляє повідомлення у чат телеграм боту з користувачем.

Повідомлення про результат розпізнавання містить у собі текст MRZ, якщо вдалося коректно розпізнати текст, або повідомлення: «Не вдалося точно

розпізнати текст MRZ», якщо результат розпізнавання не пройшов усіх перевірок.

3.8. Створення Flask API для розпізнавання MRZ

В процесі реалізації системи оптичного розпізнавання тексту на документах, було використано шаблон проектування програмного забезпечення model–view–controller. Базуючить на цьому шаблоні проектування, ми відокремили розпізнавання даних MRZ з закордонного паспорту у окремий API, який здатен працювати незалежно. Цей API приймає на вхід зображення та повертає результат розпізнавання MRZ.

Для створення API був використаний python Flask (мікро веб-фреймворк). Реалізацію описаного API можна побачити на Рис.3.11.

```python
from flask import Flask, jsonify, send_file
from flask import request
import base64
import json
import os
import shutil
import time
import recognizer
import rezult_processing
from settings import host, database, user, password
import rotate_img


app = Flask(__name__)
DEBUG = False

@app.route('/api', methods=['POST'])
def get():...

if __name__ == '__main__':

    if DEBUG:
        try:
            app.run()
        except Exception as e:
            print('No text error')
    else:
        # Bind to PORT if defined, otherwise default to 5000.
        port = int(os.environ.get('PORT', 5000))
        app.run(host='0.0.0.0', port=port, debug=False)
```

Рис.3.11. Реалізація Flask API для розпізнавання MRZ

Таким чином, за допомогою відокремлення в окремий API модуля для розпізнавання MRZ, до розробленої технології можна легко інтегрувати будь-

який графічний інтерфейс користувача, який буде взаємодіяти з API за допомогою https запитів.

Окрім Telegram bot, у якості графічного інтерфейсу користувача можна використати Facebook чи Viber ботів, мобільний додаток або веб-сайт.

3.8. Деплой системи на heroku

Платформою для розгортання було вибрано heroku.com.

Heroku - це платформа як сервіс, заснований на керованій контейнерній системі, з інтегрованими службами передачі даних та потужною екосистемою для розгортання та роботи сучасних додатків [25].

Ця платформа була вибрана з наступних причин:

1. Існує можливість безкоштовно для розгортання невеликих сервісів;
2. Підтримка розгортання за допомогою Docker;
3. Підтримка python додатків;
4. Зручне управління додатками та стабільна робота сервісу.

Перш за все, потрібно створити акаунт на heroku.com. Після успішного створення акаунту, потрібно створити новий додаток, давши йому унікальну назву [26].

Після створення нового додатку, можна робити деплой на сервер.Heroku дозволяє робити деплой у три методи:

- За допомогою Heroku Git;
- За допомогою GitHub;
- За допомогою Container Registry.

Ми будемо робити деплой нашого додатку за допомогою Docker та Container Registry.

Для розгортання додатка на основі Docker необхідно виконати наступні кроки:

1. Встановити Heroku CLI;
2. Виконати вхід до свого облікового запису Heroku;

3. Перевірити налаштування Docker.
   Для продовження роботи повинен бути локально налаштований Docker. Перевірити коректність налаштування Docker можна за допомогою наступної команди:
   $ docker ps;
4. Виконати вхід до Реєстру контейнерів.
   Увійти до Реєстру контейнерів можна за допомогою наступної команди:
   $ heroku container:login;
5. Створити Dockerfile (*Рис.3.12.)*у поточному каталозі та завантажити зображення Docker за допомогою наступної команди:
   $ heroku container:push web;
6. Дочекатися повного завантаження Docker зображення на Heroku;
7. Розгорнути завантажене Docker зображення за допомогою команди:
   $ heroku container:release web [26].

```
FROM ubuntu:latest
RUN apt-get update -y
RUN apt-get install -y python3-pip python3-dev build-essential
COPY . /app
WORKDIR /app
RUN pip3 install -r requirements.txt
ENTRYPOINT ["python3"]
CMD ["app.py"]
```

Рис.3.12. Зміст Dockerfile

Після виконання усіх зазначених кроків, потрібно увімкнути додаток, як показано на Рис.3.13. обравши Dyno Type – Free, щоб використовувати безкоштовний сервер. Heroku надає 1000 годин безперервної роботи додатків, отже цього досить, щоб наша система безперебійно працювала.

Розгорнувши Flask API для розпізнавання MRZ Heroku, та запустивши додаток під назвою *pass-bot* ми можемо надсилати запити до API на наступний url: https://pass-bot.herokuapp.com/api.

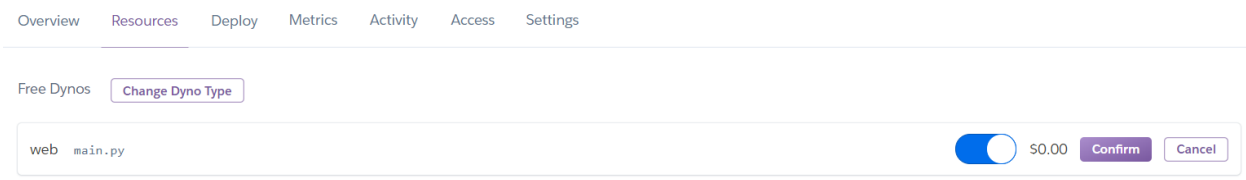


Рис.3.13. Запуск додатку на Heroku

ВИСНОВКИ ДО РОЗДІЛУ 3

1. В даному розділі було розроблено систему розпізнавання тексту на документах, а саме розпізнавання MRZ коду з закордонних паспортів;
2. Був розроблений інтерфейс користувача для зручної взаємодії з системою;
3. Був розроблений алгоритм розпізнавання тексту з зображення та алгоритм автокорекції помилок;
4. Було розроблено сценарій взаємодії користувача із програмою за допомогою телеграм боту;
5. Було розроблено структурну взаємодії графічного інтерфейсу з серверною частиною програми.

РОЗДІЛ 4
ТЕСТУВАННЯ РОЗРОБЛЕНОЇ СИСТЕМИ

Для тестування та аналізу результатів роботи системи було використано фотографії біометричних закордонних паспортів різних країн, які знаходяться у відкритому доступі. Приклад закордонного паспорту громадянина України, який було використано для тестування системи, можна побачити на Рис.4.1.

Рис. 4.1. Приклад закордонного паспорту.

При моделюванні роботи системи були зроблені наступні експерименти:

1. Перевірка роботи системи для паспортів різних країн;
2. Перевірка якості роботи системи для фотографій, зроблених під кутом;
3. Перевірка якості роботи системи для засвічених фотографій та фотографій, зроблених при поганому освітленні;

4.1. Оцінка якості роботи системи

Час на обробку зображення паспорту складає у середньому 1.5 секунди. Треба зауважити, що час залежить від розміру зображення, оскільки витрачається більше часу на пересилку зображення.

5.1.1. Оцінка роботи системи для паспортів різних країн

Для оцінки роботи розробленої системи було проведено тестування на закордонних паспортах наступних країн:

- України (Рис. 4.2.);
- Австралії (Рис. 4.3.);
- Республіки Сінгапур (Рис. 4.4.);
- Китайської Народної Республіки (Рис. 4.5.);
- Сполучених Штатів Америки (Рис. 4.6.).

Рис. 4.2. Приклад закордонного паспорту України.

Рис. 4.3. Приклад закордонного паспорту Австралії.

В результаті тестування на перелічиних фотографіях ми отримали 100% точності розпізнавання тексту. З цього можна зробити висновок, що система,

розроблена в рамках даної дипломної роботи, здатна коректно працювати з паспортами різних країн.

Рис. 4.4. Приклад закордонного паспорту Республіки Сінгапур.

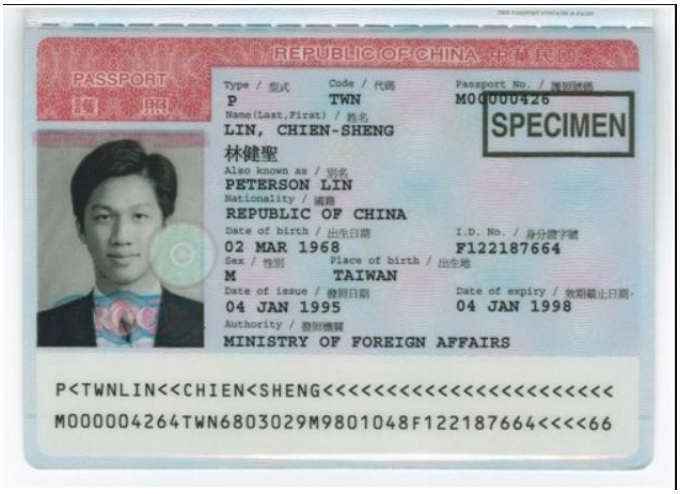


Рис. 4.5. Приклад паспорту Китайської Народної Республіки.

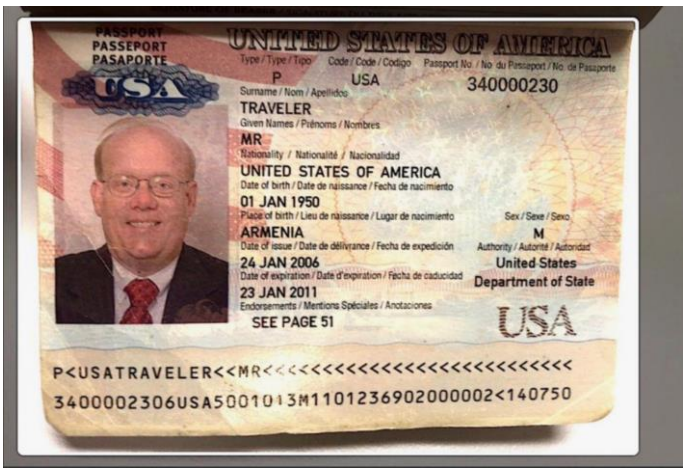


Рис. 4.6. Приклад закордонного паспорту Сполучених Штатів Америки.

Приклад результату розпізнавання іноземного паспорту зображено на Рис.4.7.

Рис.4.7. Приклад результату розпізнавання паспорту громадянина Австралії.

5.1.2. Оцінка роботи системи для фотографій, зроблених під кутом

Для цього тестування був використаний закордонний паспорт громадянина України, сфотографований під різним кутом повороту.

Були проведені наступні експерименти з кутом повороту паспорта:

- Поворот на 30° (Рис. 4.9.);
- Поворот на 60° (Рис. 4.11.) ;
- Поворот на 90° (Рис. 4.8.);
- Поворот на 180° (Рис. 4.10.).

В аналізуючи результати експериментів, було виявлено, що при повороті на 30°, 90° або 180° модель без зайвих зусиль здатна розпізнати текст. Експериментально встановлено, що при цих кутах нахилу Google Vision API легше розпізнати орієнтацію тексту та зробити автокорекцію куту нахилу тексту.

Отже, проведені дослідження дають зрозуміти, що система здатна опрацьовувати не лише якісно відскановані зображення, а й фотознімки, зроблені користувачем власноруч на свою фотокамеру.

Рис. 4.8. Результат розпізнавання паспорту під кутом повороту ~90°.

Рис. 4.9. Результат розпізнавання паспорту під кутом повороту ~30°.

Також, в результаті тестування системи на фотознімку з кутом повороту рівним близько 60°, була виявлена помилка розпізнавання тексту. Помилку було зроблено у символі MRZ, який відповідає за стать (замість символу M

було розпізнано символ Н). Ця помилка була виявлена на етапі перевірки результатів та користувач отримав повідомлення, що MRZ не вдалося розпізнати правильно.

Рис. 4.10. Результат розпізнавання паспорту під кутом повороту ~180°.

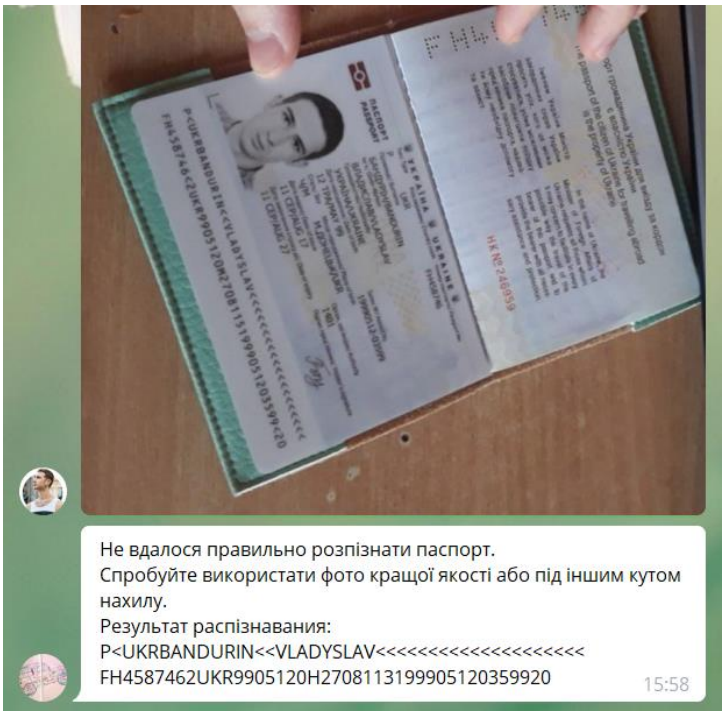


Рис. 4.11. Помилка при розпізнаванні паспорту під кутом повороту ~60°.

4.1.3. Оцінка роботи системи для фотографій, зроблених при поганому освітленні

За допомогою додавання шуму до зображення можна зробити імітацію поганого освітлення. Задля чистоти експерименту були використані ті самі

зображення, що і у попередніх експериментах, але з додавання ефекту поганого освітлення.

В результаті проведеного експерименту було виявлено, що при додавання шуму до зображення без повороту система змогла правильно зробити розпізнавання MRZ (Рис. 4.12.).

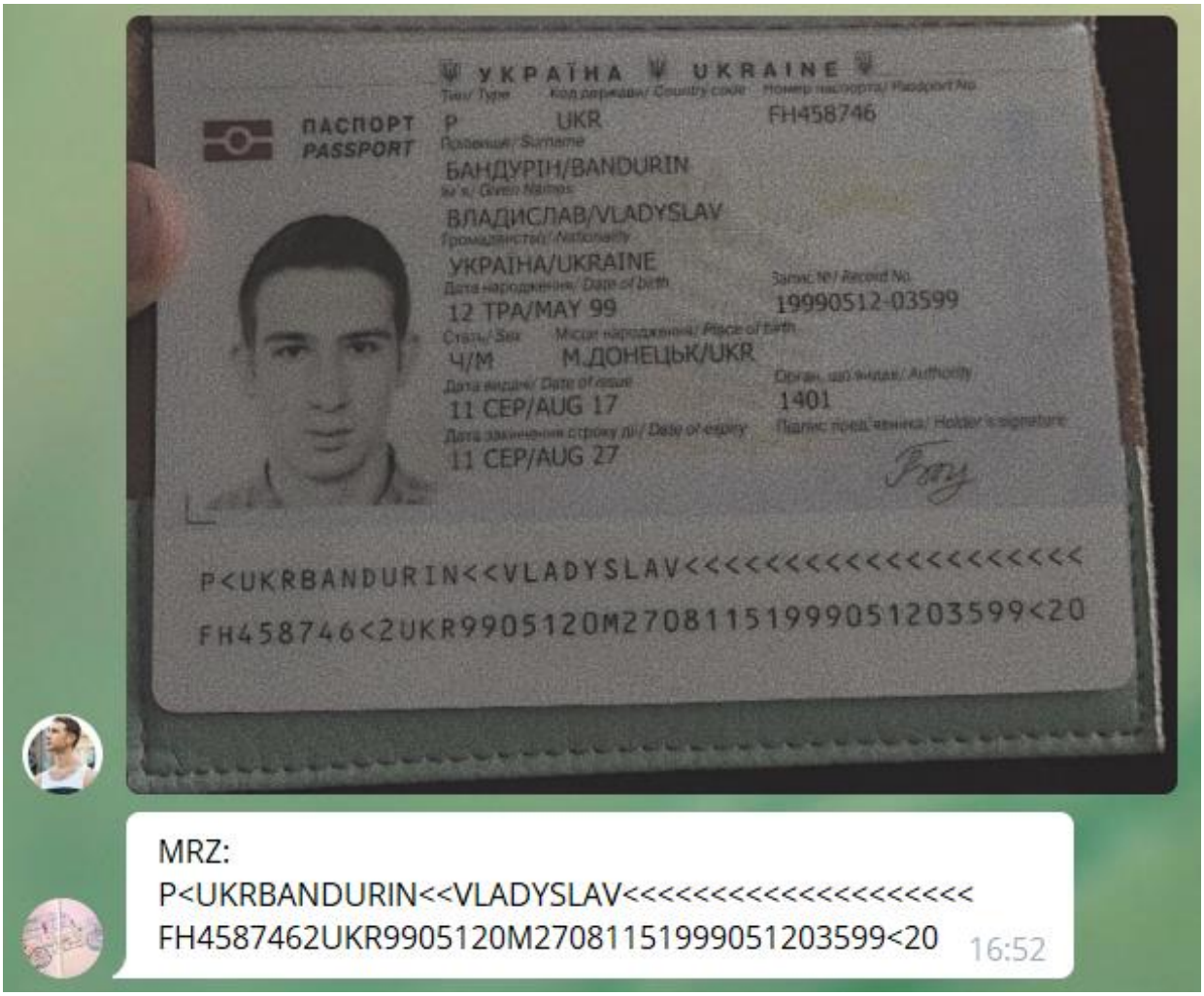


Рис. 4.12. Результат розпізнавання паспорту при поганому освітленні.

Однак, при додаванні ефекту поганого освітлення до фотознімку паспорту при куті повороту ~30°, система не змогла правильно розпізнати MRZ та зробила помилки одразу у кількох символах.

Треба зауважити, що теж саме зображення без шуму система змогла розпізнати без помилок, та додавання шуму значно вплинуло на читабельність тексту.

На Рис. 4.13. можна побачити результат експерименту фотознімком при поганому освітленні.

Рис. 4.13. Результат розпізнавання паспорту з поганим освітленням та поворотом на кут ~30°.

5.2. Аналіз результатів експериментів

Після проведення чисельних експериментів з використанням різних прикладів закордонних паспортів різних країн, сфотографованих в різних умовах, можна зробити декілька висновків щодо якості роботи розробленої системи:

- Розроблена система здатна опрацьовувати паспорти різних країн, якщо код MRZ відповідає структурі, описаній у цій роботі у розділі 1;

- Система здатна оброблювати не лише якісні зображення, зроблені за допомогою сканера, а й фотознімки низької якості, зроблені за допомогою фотокамери користувача;
- Система здатна розпізнавати текст з зображень, зроблених під кутом та при поганому освітленні, однак в деяких випадках це може вплинути на якість розпізнавання. Тому, слід явно вказувати користувачу у рекомендації як правильно зробити фото паспорту, задля досягнення максимальної точності розпізнавання.

Підсумовуючи усі перераховані висновки, можна сказати, що розроблена система розпізнавання тексту вже на даному етапі здатна автоматизувати та значно скоротити час на обробку фотознімків закордонних паспортів, а це в свою чергу надає велику цінність продукту для бізнесу.

ВИСНОВКИ ДО РОЗДІЛУ 4

1. В даному розділі було представлено тестування роботи системи оптичного розпізнавання тексту з закордонних паспортів;
2. Були проведені експерименти для оцінки якості розпізнавання тексту, а саме: тестування на паспортах різних країн, тестування на фотознімках паспортів під різним кутом нахилу та при поганому освітленні;
3. Експериментально було виявлено чинники, які впливають на якість розпізнавання тексту, а саме: чіткість зображення, кут нахилу тексту, освітлення та наявність бліків на зображенні;
4. Також, було доведено, що система здатна працювати не тільки із якісними зображеннями, а й з реальними фотознімками, зробленими користувачем на камеру телефону при поганому освітленні.

ВИСНОВКИ

1. У представленій роботі було розглянуто методи оптичного розпізнавання тексту. Були розглянуті передові технології в області розпізнавання тексту. Було аргументовано вибір технології Google Vision API для задачі розпізнавання тексту на закордонних паспортах.
2. Буда розроблена система оптичного розпізнавання тексту з закордонних паспортів та графічний інтерфейс до неї у вигляді Telegram Bot.
3. Було проведено тестування роботи системи та аналіз результатів за допомогою ряду експериментів. Оцінка якості роботи показала, що система здатна опрацьовувати фотознімки паспортів різної якості, у деяких випадках, навіть при поганому освітленні та наявному тексті під кутом нахилу.

ПЕРЕЛІК ПОСИЛАНЬ

1. Что такое OCR [Електронний ресурс]. – ABBYY FineReader, 2019. – Режим доступу: https://www.abbyy.com/ru-ru/finereader/what-is-ocr/.
2. Закордонний паспорт громадянина України для виїзду за кордон [Електронний ресурс]. – ДП "Документ", 2020, – Режим доступу: https://rivne.pasport.org.ua/poslugi/zakordonnij-pasport.
3. Біометричні документи в Україні [Електронний ресурс]. – ДМС України, 2020, – Режим доступу: *https://dmsu.gov.ua/faq/biometrichni-dokumenti-v-ukrajni.html*.
4. Паспорт громадянина України для виїзду за кордон [Електронний ресурс]. – Cyclowiki, 2020, – Режим доступу: http://cyclowiki.org/wiki/%D0%9C%D0%B0%D1%88%D0%B8%D0%BD%D0%BE%D1%87%D0%B8%D1%82%D0%B0%D0%B5%D0%BC%D1%8B%D0%B9_%D0%BF%D0%B0%D1%81%D0%BF%D0%BE%D1%80%D1%82.
5. ABBYY. What is OCR and OCR technology [Електронний ресурс] / ABBYY – Режим доступу до ресурсу: https://www.abbyy.com/en-eu/finereader/what-is-ocr/.
6. OCR System FineReader by ABBY [Електронний ресурс] – Режим доступу: https://ru.wikipedia.org/wiki/ABBYY_FineReader.
7. SimpleOCR system [Електронний ресурс] – Режим доступа: http://freeanalogs.ru/SimpleOCR.
8. Free-OCR system [Електронний ресурс] – Режим доступу: http://www.free-ocr.com/.
9. OCRFeeder system source code [Електронний ресурс] – Режим доступу: https://github.com/GNOME/ocrfeeder.
10. Tesseract-OCR source code [Електронний `ресурс] – Режим доступу: https://github.com/tesseract-ocr.
11. Cloud Vision documentation [Електронний ресурс] – Режим доступу: https://cloud.google.com/vision/docs.

12. Афонасенко А. В. Обзор методов распознавания структурированных символов [Електронний ресурс] / А. В. Афонасенко, А. И. Елизаров. – 2008. – Режим доступу до ресурсу: http://old.tusur.ru/filearchive/reports-magazine/2008-2-1/83-88.pdf.
13. Котович Н. В. Распознавание скелетных образов [Електронний ресурс] / Н. В. Котович, О. А. Славин. – 2003. – Режим доступу до ресурсу: http://ocrai.narod.ru/skeletrecognize.html.
14. How to use the tools provided to train Tesseract 4.00 [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/tesseract-ocr/tessdoc/blob/master/TrainingTesseract-4.00.md.
15. Improving the quality of the output [Електронний ресурс] – Режим доступу до ресурсу: https://tesseract-ocr.github.io/tessdoc/Technical-Documentation.
16. Detect text in images [Електронний ресурс]. – Google, 2020. – Режим доступу: https://cloud.google.com/vision/docs/ocr#vision_text_detection-python.
17. Document Text Tutorial [Електронний ресурс]. – Google, 2020. – Режим доступу: https://cloud.google.com/vision/docs/fulltext-annotations.
18. Data Usage FAQ [Електронний ресурс]. – Google, 2020. – Режим доступу: https://cloud.google.com/vision/docs/data-usage
19. Bots: An introduction for developers [Електронний ресурс]. – Telegram, 2020. – Режим доступу: https://core.telegram.org/bots.
20. Справочник по Bot API [Електронний ресурс]. – Telegram, 2020. – Режим доступу: https://tlgrm.ru/docs/bots/api#authorizing-your-bot
21. Python Telegram Bot's documentation [Електронний ресурс]. – Leandro Toledo, 2020. – Режим доступу: https://python-telegram-bot.readthedocs.io/en/stable/
22. Python Telegram Bot's Github [Електронний ресурс]. – Leandro Toledo, 2020. – Режим доступу: https://github.com/python-telegram-bot/python-telegram-bot

23. Google. Setup the Vision API [Електронний ресурс] / Google – Режим доступу до ресурсу: https://cloud.google.com/vision/docs/setup.
24. Pycountry [Електронний ресурс]. – Christian Theune, 2019. – Режим доступу: https://pypi.org/project/pycountry/
25. The Heroku Platform [Електронний ресурс] – Режим доступу до ресурсу: https://www.heroku.com/platform.
26. Container Registry & Runtime (Docker Deploys) [Електронний ресурс] – Режим доступу до ресурсу:https://devcenter.heroku.com/articles/container-registry-and-runtime.

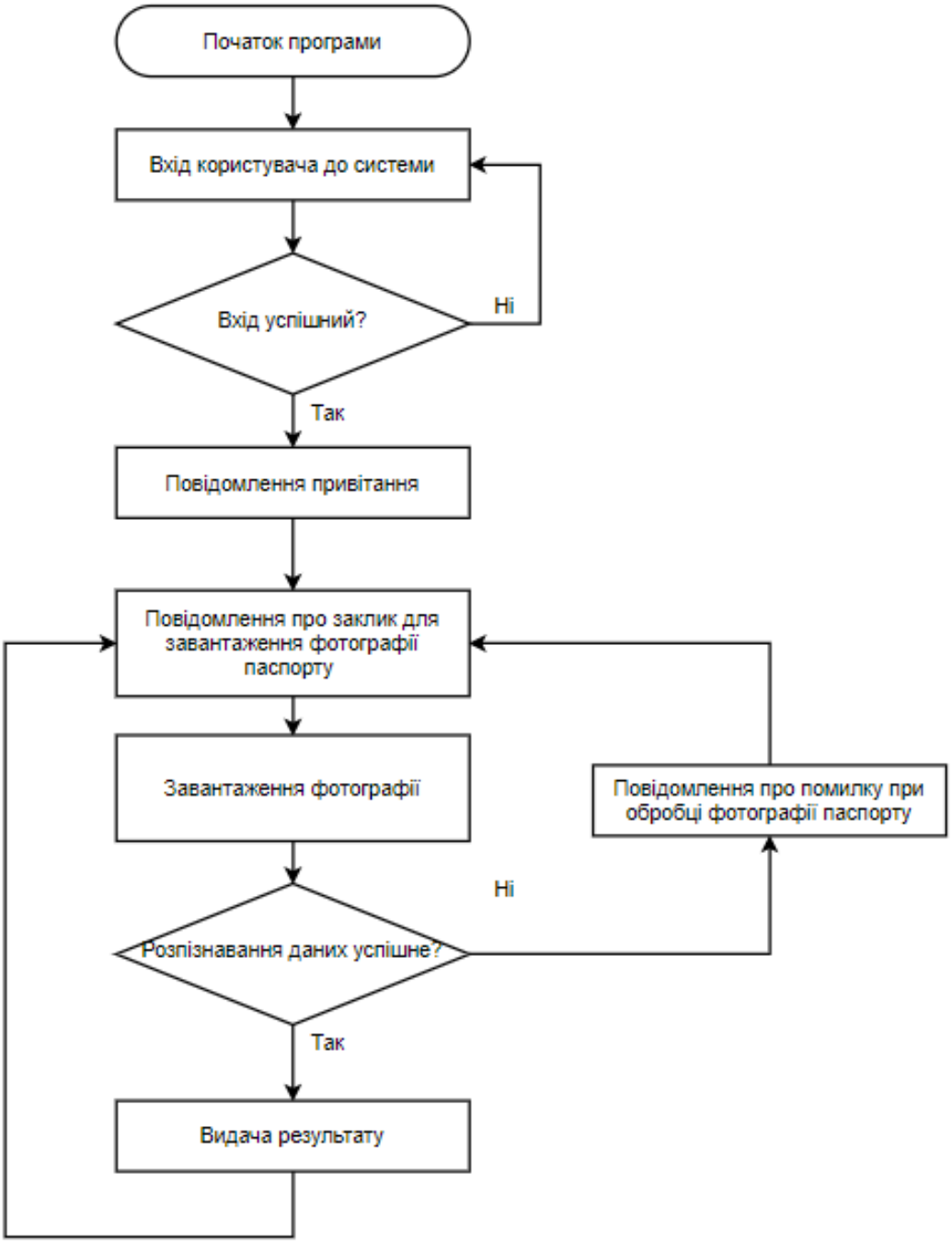
ДОДАТОК А

«Система оптичного розпізнавання тексту на документах»

Блок схема алгоритму

Аркушів 1

Київ – 2020р.



					ІАЛЦ.467800.004 Д1	
Зм.	*Арк.*	*№ докум.*	*Підпис*	*Дата*		
Розробив		*Бандурін В.Ю.*			*Система розпізнавання тексту на документах*	*Літ.* / *Аркуш* / *Аркушів*
Перевірив		*Таран В.І.*				*1* / *1*
Реценз.					*Блок схема алгоритму*	*НТУУ «КПІ», ФІОТ, ІО-64*
Н. Контр.		*Сімоненко В.П.*				
Затв.						

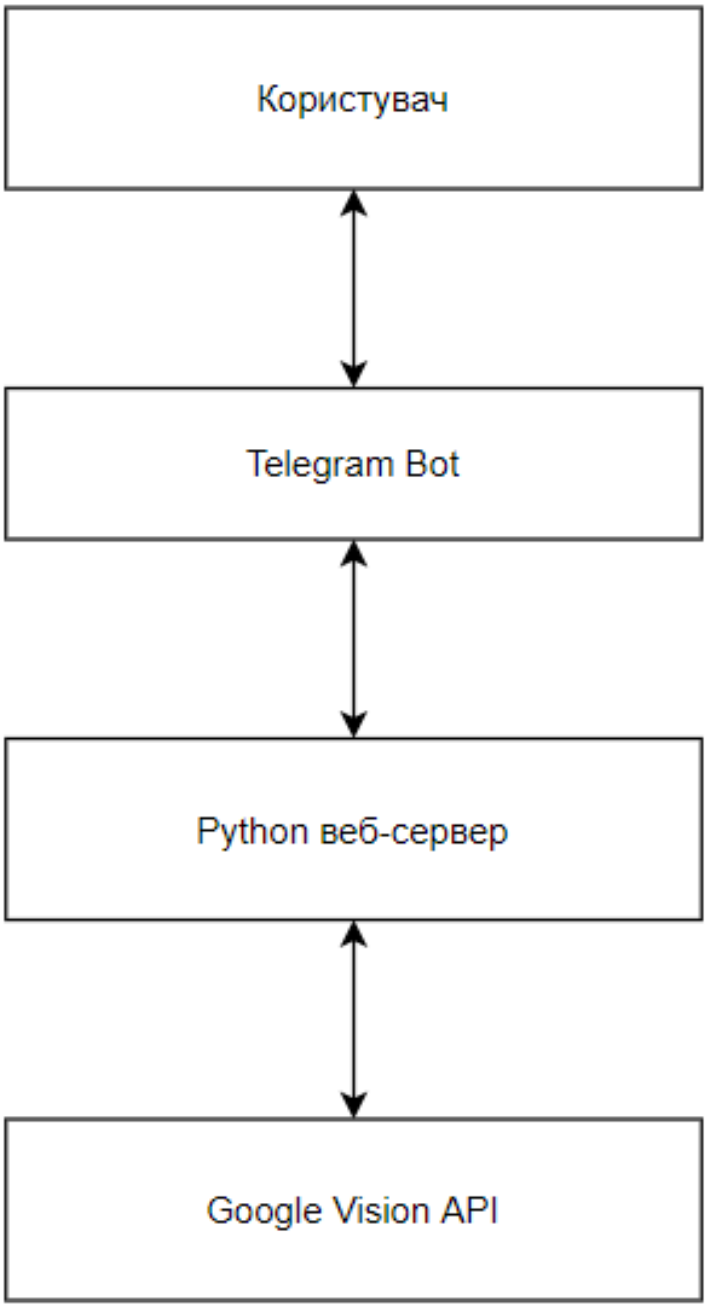
ДОДАТОК Б

«Система оптичного розпізнавання тексту на документах»

Схема роботи програми

Аркушів 1

Київ – 2020р.



					ІАЛЦ.467800.004 Д2	
Зм.	*Арк.*	*№ докум.*	*Підпис*	*Дата*		
Розробив		*Бандурін В.Ю.*			*Система розпізнавання тексту на документах* *Схема роботи програми*	*Літ.* / *Аркуш* / *Аркушів*: 1 / 1
Перевірив		*Таран В.І.*				
Реценз.						*НТУУ «КПІ», ФІОТ, ІО-64*
Н. Контр.		*Сімоненко В.П.*				
Затв.						

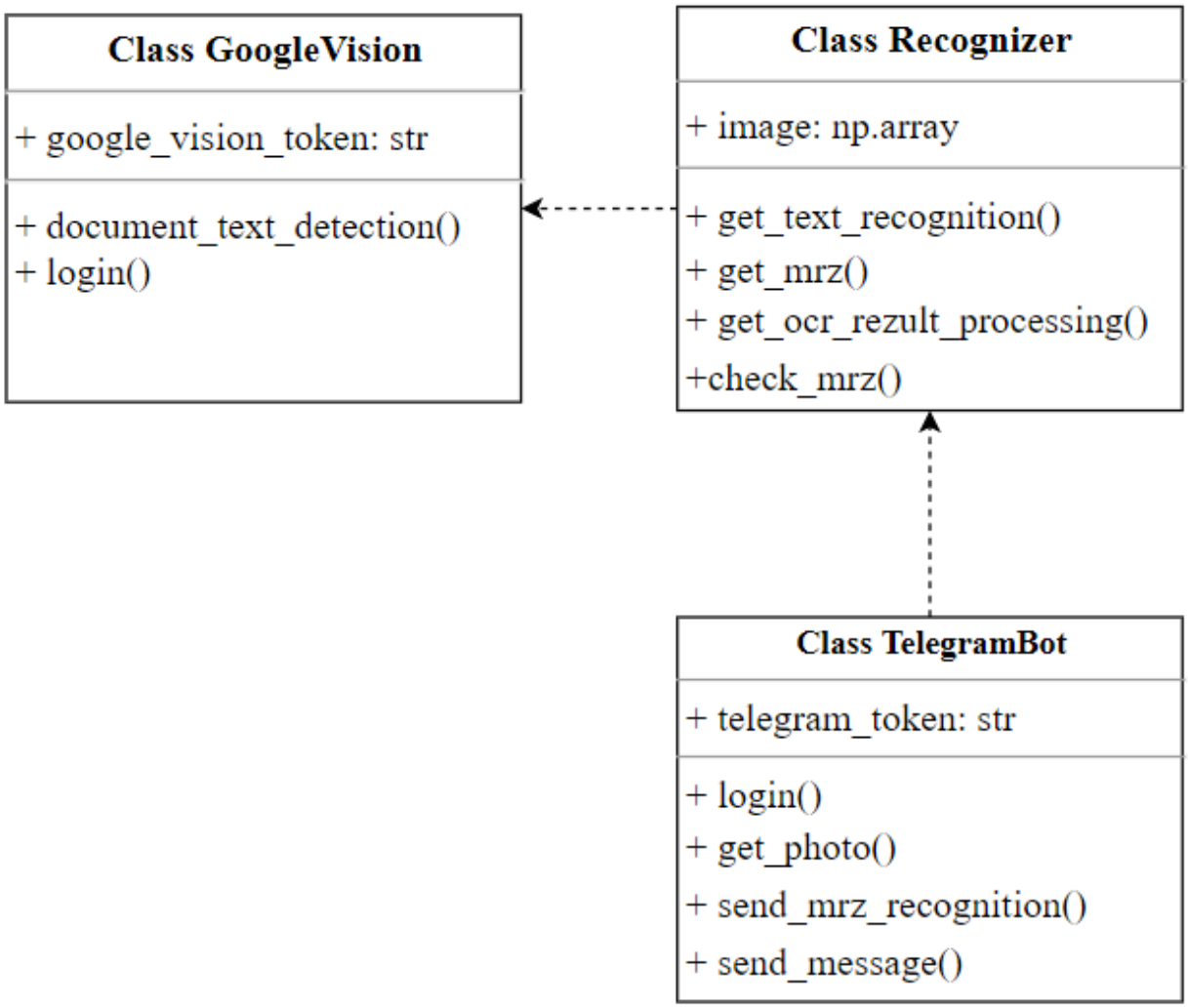
ДОДАТОК В

«Система оптичного розпізнавання тексту на документах»

Діаграма класів

Аркушів 1

Київ – 2020р.



					ІАЛЦ.467800.004 ДЗ	
Зм.	Арк.	№ докум.	Підпис	Дата		
Розробив		Бандурін В.Ю.			Система розпізнавання тексту на документах Діаграма класів	Літ. / Аркуш: 1 / Аркушів: 1
Перевірив		Таран В.І.				
Реценз.						НТУУ «КПІ», ФІОТ, ІО-64
Н. Контр.		Сімоненко В.П.				
Затв.						

ДОДАТОК Г

«Система оптичного розпізнавання тексту на документах»

Лістинг програми

Аркушів 32

Київ – 2020р.

```
import os
from google.cloud import vision
import settings
import re
os.environ["GOOGLE_APPLICATIO
N_CREDENTIALS"] = settings.key

def recognize_document(img=None,
img_path=None):

    """
    Recognize document features in an
image.
     :input img - numpy img or img_path
- str path to img
     :return rez - json of recognized page
    """

    client =
vision.ImageAnnotatorClient()

    if img_path:
        with open(img_path, 'rb') as
image_file:
content = image_file.read()
    else:
        content = img
    image =
vision.types.Image(content=content)

    response =
client.document_text_detection(image=
image)

    try:
        rez = response
        return rez

    except Exception as e:
        return {}


def get_country_code(text):
    country_code = re.findall('[0-9]+[A-
Z]{3}[0-9]+', text)
    if country_code:
        country_code = country_code[0]
        country_code = re.search('[A-
```

					ІАЛЦ.467800.004 Д4		
Зм.	Арк.	№ докум.	Підпис	Дата			
Розробив		Бандурін В.Ю.			Система розпізнавання тексту на документах *Лістинг програми*	Літ.	Аркуш 1 / Аркушів 32
Перевірив		Таран В.І.					
Реценз.						НТУУ «КПІ», ФІОТ, ІО-64	
Н. Контр.		Сімоненко В.П.					
Затв.							

```
Z]{3}', country_code).group(0)
    else:
        country_code = None
    return country_code
import common
import os
import json
import re
import pickle
import numpy as np
from enum import Enum
import time

import pycountry



class FeatureType(Enum):
    PAGE = 1
    BLOCK = 2
    PARA = 3
    WORD = 4
    SYMBOL = 5

class Lines():

    def __init__(self, mrz_block):
        bounding_boxes = []
        centres = []
        # print(mrz_block.bounding_box)
        #
print(mrz_block.bounding_box.max(ax
is=0))
        block_box =
mrz_block.bounding_box

        max_x, max_y =
mrz_block.bounding_box.max(axis=0)
        min_x, min_y =
mrz_block.bounding_box.min(axis=0)

        center_block =
block_box.mean(axis=0)

        for d in block_box:
            d = np.array(d)
            if d[0] == min_x:
                self.left_dot = d
            elif d[0] == max_x:
                self.right_dot = d
            elif d[1] == min_y:
                self.down_dot = d
            elif d[1] == max_y:
                self.top_dot = d
        # print('dist',
self.get_dist(self.top_dot, self.left_dot))

        for p in mrz_block.block:
            for w in p.paragraph:
                for s in w.word:
                    box = s.bounding_box
```

```
                center = box.mean(axis=0)
                centres.append(center)
                #
bounding_boxes.append(box)



    def get_dist(self, d1, d2):
        return np.sqrt(((d1 - d2) ^
2).sum())

    def get_rectangles(self):

        rectangle_1 = []
        rectangle_2 = []

class Symbol():

    def __init__(self, text, confidence,
bounding_box):
        self.text = text.upper()
        self.text =
common.translit(self.text)
        self.confidence = confidence
        self.bounding_box =
np.array([[bounding_box.vertices[0].x,
bounding_box.vertices[0].y],

[bounding_box.vertices[1].x,
bounding_box.vertices[1].y],

[bounding_box.vertices[2].x,
bounding_box.vertices[2].y],

[bounding_box.vertices[3].x,
bounding_box.vertices[3].y]])


class Word():

    def __init__(self, symbols_list,
bounding_box):
        self.word = symbols_list
        self.text = ''.join([i.text for i in
self.word])
        self.confidence =
np.array([i.confidence for i in
self.word]).sum() / len(self.word)
        self.bounding_box =
np.array([[bounding_box.vertices[0].x,
bounding_box.vertices[0].y],

[bounding_box.vertices[1].x,
bounding_box.vertices[1].y],

[bounding_box.vertices[2].x,
bounding_box.vertices[2].y],

[bounding_box.vertices[3].x,
bounding_box.vertices[3].y]])

class Paragraph():
```

```
    def __init__(self, word_list, bounding_box):
        self.paragraph = word_list
        self.text = ''.join([i.text for i in self.paragraph])
        self.confidence = np.array([i.confidence for i in self.paragraph]).sum() / len(self.paragraph)
        self.bounding_box = np.array([[bounding_box.vertices[0].x, bounding_box.vertices[0].y],

[bounding_box.vertices[1].x, bounding_box.vertices[1].y],

[bounding_box.vertices[2].x, bounding_box.vertices[2].y],

[bounding_box.vertices[3].x, bounding_box.vertices[3].y]])

class Block():

    def __init__(self, paragraph_list, bounding_box):
        self.block = paragraph_list
        self.text = ''.join([i.text for i in self.block])
        self.confidence = np.array([i.confidence for i in self.block]).sum() / len(self.block)
        try:
            self.bounding_box = np.array([[bounding_box.vertices[0].x, bounding_box.vertices[0].y],

[bounding_box.vertices[1].x, bounding_box.vertices[1].y],

[bounding_box.vertices[2].x, bounding_box.vertices[2].y],

[bounding_box.vertices[3].x, bounding_box.vertices[3].y]])
        except:
            self.bounding_box = bounding_box

    def get_words(self):
        words = []
        for p in self.block:
            for w in p.paragraph:
                words.append(w)
        return words

    def get_symbols(self):
        symbols = []
        for p in self.block:
            for w in p.paragraph:
```

```
            for s in w.word:
                symbols.append(s)
        return symbols


def get_word_list(block_list,
mrz_blocks):
    word_list = []
    for b in block_list:
        # # print(len(b.get_words()),
len(mrz_block.get_words()))
        if b not in mrz_blocks:
            words = [w.text for w in
b.get_words()]
            word_list += words
        # else:
            #
print('========================
==========')
    return word_list


def save_json(j, file_name):
    dir_name = './recognized_json/'
    if file_name not in
os.listdir(dir_name):
        file_name = dir_name + file_name
        pickle.dump(j, file_name)
        # with open(file_name, 'w') as f:
        #     y = json.dumps(str(j))
        #     f.write(y)
        return True
```

```
def get_block_list(document):
    bounds = []

    block_list = []
    for page in
document.full_text_annotation.pages:
        for block in page.blocks:
            paragraph_list = []

            for paragraph in
block.paragraphs:
                word_list = []

                for word in paragraph.words:
                    symbols_list = []

                    for symbol in
word.symbols:

symbols_list.append(Symbol(symbol.te
xt, symbol.confidence,
symbol.bounding_box))

bounds.append(symbol.bounding_box)

bounds.append(word.bounding_box)
                w = Word(symbols_list,
word.bounding_box)
                word_list.append(w)
            p = Paragraph(word_list,
paragraph.bounding_box)
```

```
            paragraph_list.append(p)
        b = Block(paragraph_list,
block.bounding_box)


bounds.append(block.bounding_box)
        block_list.append(b)



    return block_list

def get_mrz(block_list):
    blocks = []
    paragraph_list = []
    for b in block_list:
        text = b.text
        if '<' in text:
            blocks.append(b)
    if blocks:
        paragraph_list = []
        for b in blocks:


            paragraph_list += b.block


        b = Block(paragraph_list,
b.bounding_box)
        return b, blocks
    else:
        return []
```

```
class MRZ_block():


    def __init__(self, block_list,
word_list=None,
img='../images_24_06/2019-6-24
08_01_02.jpg'):


        mrz_block, blocks =
get_mrz(block_list)
        self.word_list =
get_word_list(block_list, blocks)
        self.mrz_block = mrz_block
        self.text = mrz_block.text


        for b in block_list:
            if b != mrz_block:
                self.not_mrz = b
                break


        self.get_lines(img=img)
        self.change_symbols()
        self.check_country_numb()
        self.check_passport_number()
        self.check_birthday()
        self.check_sex()
        self.check_end_date()



    def _find_similar(self, s1, s2):
        n = 0
```

```
        if len(s1) <= len(s2):
            for i in range(len(s1)):
                if s1[i] == s2[i]:
                    n += 1
        return n


    def get_lines(self, plot=False,
img='../images_24_06/2019-6-24
08_01_02.jpg'):
        """
        Находит центры каждого
символа. Записывает координаты
каждого центра в centres.
        Словарь symbols_dict: key -
center, value - Symbol
        Обучает линейный бинарный
классификатор для
классифицирования точек для
каждой линии.
        (Линейная регрессия для всех
центров, выше линии - один класс,
ниже - другой)

        :return:
        np.array([first_line, second_line])
        """



        text = self.mrz_block.text
        text_cleaned = text.lstrip('<')
        # print('text', text)
        try:
            sep = re.findall('<{4,}',
text_cleaned)[0]
        except:
            sep = '<<<'


        split_text = text.split(sep)
        text_line_1, text_line_2 =
split_text[0], sep.join(split_text[1:])


        symbols_list = []


        for p in self.mrz_block.block:
            for w in p.paragraph:
                for s in w.word:
                    symbols_list.append(s)
                    # center =
s.bounding_box.mean(axis=0)
        # # print(text)
        ind_1 = text.find(text_line_1),
text.find(text_line_1) + len(sep) +
len(text_line_1)
        ind_2 = text.find(text_line_2),
text.find(text_line_2) +
len(text_line_2)
        self.first_line =
symbols_list[ind_1[0]: ind_1[1]]
        self.second_line =
symbols_list[ind_2[0]: ind_2[1]]
```

```
        return np.array([self.first_line,
self.second_line])

    def change_symbols(self,
conf_1=0.3, conf_2=0.3):



        for s in self.first_line:
            if s.confidence <= 1:
                # print(s.text, s.confidence)
                if s.text in
common.always_change_1.keys():
                    s.text =
common.always_change_1[s.text]

        for s in self.first_line:
            if s.confidence < conf_1:
                if s.text in
common.similar_letters_1.keys():
                    s.text =
common.similar_letters_1[s.text]


            n = self.first_line.index(s)
            if n != len(self.first_line) - 1:
                if s.text == 'P' and
self.first_line[n+1].text == 'C' and
self.first_line[n-1].text != 'P' and n < 4:
                    #
print('==========C==========
=')
                    self.first_line[n+1].text =
'<'


        for s in self.second_line:
            if s.confidence < conf_2:
                # print(s.text, s.confidence)
                if s.text in
common.similar_letters_2.keys():
                    s.text =
common.similar_letters_2[s.text]
            n = self.second_line.index(s)
            ls = '0'
            rs = '0'
            kr = ''
            ge = ''
            ur = ''
            if n != len(self.second_line)-1
and n != 0:
                ls = self.second_line[n-
1].text
                rs =
self.second_line[n+1].text
                ur = self.second_line[n-
1].text + self.second_line[n + 1].text
            if n < len(self.second_line) - 3:
                kr =
self.second_line[n+1].text +
self.second_line[n+2].text

            if n > 2:
```

```
            ge = self.second_line[n-
2].text + self.second_line[n-1].text
            if s.text == 'S' and
self.second_line[n - 1].text == '<':
                s.text = '5'

        if s.text == 'O':
            s.text = '0'

        if s.text == '0' and (not
ls.isdigit() and not rs.isdigit()) and ls
not in '<O' and rs not in '<O':
            s.text = 'O'

        if kr == 'KR' and (s.text == '0'
or s.text == 'O'):
            s.text = 'U'
        elif ge == 'GE' and (s.text ==
'0'):
            s.text = 'O'
        elif ur == 'UR' and (s.text ==
'<'):
            s.text = 'K'

    # print(''.join([s.text for s in
self.first_line]))
    # print(''.join([s.text for s in
self.second_line]))

  def check_sex(self):
      '''
      Проверяет наличие M/F
      Если нет, то ставит M/F в
нужную позицию, проверяя перед
этим на проверке check_end_date,
      куда входит M/F
      :return:
      {'is_true': is_true,
      'sex': sex_section}
      '''
      self.ch_sex = False
      self.sex = None
      second_line = ''.join([s.text for s in
self.second_line])

      def is_true_sex(second_line):
          sex_section = re.findall('[0-
9]{6,}[A-Z][0-9]{3,}', second_line)
          # # print('sex_section',
sex_section)
          if len(sex_section) > 0:
              sex_section = sex_section[0]
          else:
              return {'is_true': False,
                      'sex': ''}
          sex_section = [i for i in
sex_section if not i.isdecimal()][0]
          is_true = sex_section in
'MMMF'
          return {'is_true': is_true,
                  'sex': sex_section}
```

```
        d = is_true_sex(second_line)
        is_true = d['is_true']
        sex_section = d['sex']

        if not is_true:
            n =
second_line.find(self.country_code) +
10
            second_line = second_line[:n] +
'M' + second_line[n+1:]
            ch =
self.check_end_date(second_line=seco
nd_line)

            if ch['is_true']:
                d = is_true_sex(second_line)
                is_true = d['is_true']
                sex_section = d['sex']
                self.second_line[n].text = 'M'

            else:
                second_line =
second_line[:n] + 'F' + second_line[n +
1:]
                ch =
self.check_end_date(second_line=seco
nd_line)
                if ch['is_true']:
                    d =
is_true_sex(second_line)
                    is_true = d['is_true']
                    sex_section = d['sex']
                    self.second_line[n].text =
'F'

        self.ch_sex = is_true
        self.sex = sex_section
        return {'is_true': is_true,
                'sex': sex_section}

    def get_check_number(self, text):
        sum = 0
        for i in range(len(text)):
            x_dict = {0: 7,
                      1: 3,
                      2: 1}
            ind = i % 3
            x = x_dict[ind]
            n = text[i]
            if n.isdigit():
                sum += int(n) * x
            else:
                sum +=
common.later_dict[n] * x
        return sum

    def check_passport_number(self):
        self.ch_number = False
        self.pass_number = None

        second_line = ''.join([s.text for s in
self.second_line])
```

```
        country_code = self.country_code
        if not country_code:
            country_code = '[A-Z]{3}'

        reg_ex = '^\w+<{0,}[0-9]{0,1}' +
country_code + '[0-9]{7}'
        number_section =
re.findall(reg_ex, second_line)

        if len(number_section) > 0:
            number_section[0] =
''.join(number_section[0].split('<'))
            number_section =
number_section[0][:-10]

            if None:
                ind_check_number =
number_section.find(country_code) - 1
                check_number =
number_section[ind_check_number]
                if check_number.isdecimal():
                    check_number =
int(check_number)
                elif check_number in
common.leter2digit.keys():
                    check_number =
int(common.leter2digit[check_number]
)

            else:
                if number_section[-
1].isdecimal():
                    check_number =
int(number_section[-1])

                elif number_section[-1] in
common.leter2digit.keys():
                    check_number =
int(common.leter2digit[number_sectio
n[-1]])


            else:
                return {'is_true': False,
                        'text': None,
                        'check_number':
None}

        else:
            return {'is_true': False,
                    'text': None,
                    'check_number': None}
        self.pass_number =
number_section[:-1]
        if self.country_code == 'UKR':
            if len(self.pass_number) == 9:
                self.pass_number =
self.pass_number[:-1]
```

```
            n =
second_line.find(self.pass_number) +
len(self.pass_number)
            self.second_line[n].text = '<'

        sum =
self.get_check_number(self.pass_numb
er)

        is_true = sum % 10 ==
check_number

        if not is_true:
            for i in
range(len(self.word_list)):
                w = self.word_list[i]
                w_2 = ''
                if i != len(self.word_list) - 1:
                    w_2 = w + self.word_list[i
+ 1]
                n =
self._find_similar(self.pass_number, w)
                n_2 =
self._find_similar(self.pass_number,
w_2)

                if len(self.pass_number) ==
n:
                    new_number =
str(self.pass_number) +
str(check_number)
                    m =
self._find_similar(new_number, w)
                    if m == len(new_number):
                        # print('equal', w)
                        self.pass_number =
new_number
                        sum =
self.get_check_number(self.pass_numb
er)
                        check_number = sum %
10
                        is_true = True
                        # print(sum)

                elif len(self.pass_number) *
0.6 <= n:
                    sum =
self.get_check_number(w)
                    if sum % 10 ==
check_number:
                        self.pass_number = w
                        is_true = True
                    # print('similar', w)
                elif len(self.pass_number) *
0.6 <= n_2:
                    sum =
self.get_check_number(w_2)
                    if sum % 10 ==
check_number:
                        self.pass_number = w_2
                        is_true = True
```

```
                else:
                    w_2 = w_2.replace('O',
'0')
                    sum =
self.get_check_number(w_2)
                    if sum % 10 ==
check_number:
                        self.pass_number =
w_2
                        is_true = True

            sum =
self.get_check_number(self.pass_numb
er)

        if check_number or
str(check_number) == '0':
            sn = self.pass_number +
str(check_number)
        else:
            sn = self.pass_number

        self.ch_number = is_true
        return {'is_true': is_true,
                'text': self.pass_number,
                'check_number':
check_number}

    def check_country_numb(self):
        self.ch_country = False
        self.country_code = None
        # country_code = re.findall('[0-
9]+[A-Z]{3}[0-9]+', text)
        text = ''.join([s.text for s in
self.second_line])

        country_code = re.findall('[0-
9<]+[A-Z]{3}[0-9]{7}', text)
        # # print(country_code)
        if country_code:

            country_code =
country_code[0]
            country_code = re.search('[A-
Z]{3}', country_code).group(0)
            country =
pycountry.countries.get(alpha_3=count
ry_code)
            self.country_code =
country_code

            if country:


                self.ch_country = True
                return {'is_true': True,
                        'country': country.name,
                        'text': text}
            else:

                return {'is_true': False,
```

```
                'country': None,
                'text': text}
        else:

            return {'is_true': False,
                    'country': None,
                    'text': text}

    def check_birthday(self):
        self.ch_birthday = False

        second_line = ''.join([s.text for s in
self.second_line])
        # second_line = ''.join(zones[3:])
        country_code = self.country_code
        if not country_code:
            country_code = '[A-Z]{3}'

        reg_ex = country_code + '[0-
9]{7}'
        birthday_section =
re.findall(reg_ex, second_line)
        if birthday_section:
            birthday_section =
birthday_section[0]
        else:
            return {'is_true': False,
                    'text': None,
                    'check_number': None}
        check_number =
int(birthday_section[-1])
        text = birthday_section[3:-1]
        sum =
self.get_check_number(text)
        is_true = sum % 10 ==
check_number

        self.ch_birthday = is_true
        self.birthday = text
        return {'is_true': is_true,
                'text': text,
                'check_number':
check_number}

    def check_end_date(self,
second_line=None):
        self.ch_end_date = False

        if not second_line:
            second_line = ''.join([s.text for s
in self.second_line])

        end_date_section = re.findall('[0-
9]{7}[A-Z]{1}[0-9]{7}', second_line)
        if end_date_section:
            end_date_section =
end_date_section[0]
        else:
            return {'is_true': False,
                    'text': None,
                    'check_number': None}
```

```
        check_number =
int(end_date_section[-1])
        text = end_date_section[8:-1]
        sum =
self.get_check_number(text)
        is_true = sum % 10 ==
check_number
        self.ch_end_date = is_true
        self.end_date = text
        return {'is_true': is_true,
                'text': text,
                'check_number':
check_number}

    def change_name(self):
        try:
            if self.country_code == 'UKR':
                names = ''.join([s.text for s in
self.first_line]).split('<')
                names = [i for i in names if i]
                surname = ''
                surname_1 = ''
                for s in names:
                    if 'UKR' in s:
                        surname_1 =
s.split('UKR')[0] + 'UKR'
                        surname =
s.split('UKR')[1]

                        n = names.index(s)
                if len(names) > n + 1:
                    given_names = names[n +
1:]
                else:
                    return

                tr_words = []
                for w in self.word_list:

tr_words.append(common.name_transli
t(w))

                for name in given_names:
                    for tr_w in tr_words:
                        m =
self._find_similar(name, tr_w)
                        j = tr_words.index(tr_w)

                        try:
                            if name ==
self.word_list[j][:-1]:

given_names[given_names.index(name
)] = self.word_list[j]
                        except:
                            pass

                        try:
                            if len(name) - 2 < m:
                                if tr_w in
self.word_list and tr_w !=
self.word_list[j]:
```

```
given_names[given_names.index(name
)] = tr_w
                        # print(tr_w)
                except Exception as e:
                    print('error', e)
                try:
                    t = tr_w.split('II')[0] +
'II'
                    tw =
self.word_list[j].split('II')[0] + 'II'

                    if name == t[:-1]:
                        # print(t)

given_names[given_names.index(name
)] = t
                    elif name == tw[:-1]:

given_names[given_names.index(name
)] = tw
                    elif name[:-1] ==
tw[:-1]:

given_names[given_names.index(name
)] = tw

                except Exception as e:
                    print('error', e)
            # print(given_names)
        for tr_w in tr_words:
            m =
self._find_similar(surname, tr_w)
            j = tr_words.index(tr_w)
            try:
                if surname ==
self.word_list[j][:-1]:
                    surname =
self.word_list[j]
            except:
                pass

            try:
                t = tr_w.split('II')[0] +
'II'
                if surname == t[:-1]:
                    # print(t)
                    surname = t
            except:
                print('error')

            if len(surname) - 2 < m:
                j = tr_words.index(tr_w)

                if tr_w in self.word_list
and tr_w != self.word_list[j]:
                    surname = tr_w
                    # print(tr_w)
```

```
            surname = surname_1 +
surname
            if len(names[n-1]) < 2:
                p =
'{:<<2s}'.format(names[n-1])
            else:
                p = ''
            first_line = p + surname +
'<<' + '<'.join(given_names)
            first_line =
'{:<<44s}'.format(first_line)

            for i in
range(len(self.first_line)):
                self.first_line[i].text =
first_line[i]
            # print(first_line)
        except Exception as e:
            pass
            # print('error', e)

    def get_check(self):
        check_1 = self.ch_number
        check_2 = self.ch_country
        check_3 = self.ch_end_date
        check_4 = self.ch_birthday
        check_5 = self.ch_end_date
        check_6 = self.ch_sex

        ch_list = [check_1, check_2,
check_3, check_4, check_5, check_6]
        is_right = all(ch_list)
        # # print(check_1, check_2,
check_3, check_4, check_5, check_6)
        pass_ch =
self.get_check_number(self.pass_numb
er) % 10

        # clear_first_line
        self.clear_first_line()

        # self.split_names()
        self.change_name()

        self.format_names()

        second_line = ''.join([s.text for s in
self.second_line])
        second_line = self.pass_number +
str(pass_ch) + self.country_code +
''.join(second_line.split(self.country_co
de)[1:])
        first_line = ''.join([s.text for s in
self.first_line])
        mrz_text = first_line +
second_line
        country_code = self.country_code

        return {'is_right': is_right,
                'mrz_text': mrz_text,
                'country_code':
country_code,
```

```
            'pass_numb':
self.pass_number,
            'sex': self.sex,
            'birthday': self.birthday,
            'end_date': self.end_date}

    def get_mrz(self):
        s = ''.join([s.text for s in
self.first_line])
        s += ''.join([s.text for s in
self.second_line])
        return s

    def split_names(self):
        try:
            first_line = ''.join([s.text for s in
self.first_line])
            names = ''.join([s.text for s in
self.first_line]).split(self.country_code)
            # print(self.word_list)
            # print(names)
            if len(names) > 1:
                names = names[1]
            names = names.split('<')
            names = [i for i in names if i
and i != '<']
            # print(names)

            if len(names) == 1:

                k_split = names[0].split('K')
                comb = []
                comb_2 = []
                for i in range(len(k_split)):
                    c = 'K'.join(k_split[0:1+i])
                    comb.append(c)
                    c = 'K'.join(k_split[-1 - i:])
                    comb_2.append(c)
                # print(comb)
                change_ind = -1
                for s in comb:
                    if s in self.word_list:
                        change_ind =
first_line.find(s) + len(s)

                for s in comb_2:
                    if s in self.word_list:
                        change_ind =
first_line.find(s) - 1

                if change_ind != -1:

self.first_line[change_ind].text = '<'

                # else:
                #     k_split =
names[0].split('C')
                #     comb = []
                #     comb_2 = []
                #     for i in
range(len(k_split)):
```

```
            #       c = 'C'.join(k_split[0:1
+ i])
            #       comb.append(c)
            #       c = 'C'.join(k_split[-1-
i:])
            #       comb_2.append(c)
            #
            #   print(comb)
            #
            #   for s in comb:
            #       if s in self.word_list:
            #           change_ind =
first_line.find(s) + len(s)
            #   if change_ind != -1:
            #
self.first_line[change_ind].text = '<'
            #   else:
            #       k_split =
names[0].split('S')
            #       comb = []
            #       comb_2 = []
            #       for i in
range(len(k_split)):
            #           c =
'S'.join(k_split[0:1 + i])
            #           comb.append(c)
            #           c = 'S'.join(k_split[-
1 - i:])
            #           comb_2.append(c)
            #       print(comb)
            #
            #       for s in comb:
            #           if s in
self.word_list:
            #               change_ind =
first_line.find(s) + len(s)
            #       if change_ind != -1:
            #
self.first_line[change_ind].text = '<'


        except Exception as e:
            print('split_names')
            print(e)

    def format_names(self):
        try:
            first_line = ''.join([s.text for s in
self.first_line])
            names = ''.join([s.text for s in
self.first_line]).split('<<')
            names = [i for i in names if i
and i != '<']
            if len(names) >= 2:
                print(names)
            else:
                names = ''.join([s.text for s in
self.first_line]).split('<')
                names = [i for i in names if i
and len(i) > 2]
```

```
            print(names)
            if len(names) > 1 and
len(names) == 2:
                n =
first_line.find(names[1])
                self.first_line =
self.first_line[:n] + [self.first_line[n-1]]
+ self.first_line[n:]
                # first_line = [s.text for s
in self.first_line]
                # print(first_line)
        except Exception as e:
            print('format_names')
            print(e)

    def clear_first_line(self):
        try:
            if (len(self.first_line) +
len(self.second_line)) > 40:
                first_line = self.first_line

                text = ''.join([s.text for s in
first_line])
                country_code =
self.country_code
                s = "[PV]+[A-Z0-9<`']{1}" +
country_code
                text_split = re.split(s, text)
                if len(text_split) > 1:
                    n = text.find(text_split[0])
+ len(text_split[0])
                    if n > 0:
                        self.first_line =
self.first_line[n:]

        except Exception as e:
            print('clear_first_line')

            print(e)

    def __str__(self):
        return self.get_mrz()



import os
from google.cloud import vision
import settings
import re
os.environ["GOOGLE_APPLICATIO
N_CREDENTIALS"] = settings.key

def recognize_document(img=None,
img_path=None):

    """
    Recognize document features in an
image.
     :input img - numpy img or img_path
- str path to img
     :return rez - json of recognized page
    """
```

```
    client = vision.ImageAnnotatorClient()

    if img_path:
        with open(img_path, 'rb') as image_file:
            content = image_file.read()
    else:
        content = img
    image = vision.types.Image(content=content)

    response = client.document_text_detection(image=image)

    try:
        rez = response
        return rez

    except Exception as e:
        return {}


def get_country_code(text):
    country_code = re.findall('[0-9]+[A-Z]{3}[0-9]+', text)
    if country_code:
        country_code = country_code[0]
        country_code = re.search('[A-Z]{3}', country_code).group(0)
    else:
        country_code = None
    return country_code


import json


later_dict = {  '<': 0,
                'A': 10,
                'B':11,
                'C':12,
                'D':13,
                'E':14,
                'F':15,
                'G':16,
                'H':17,
                'I':18,
                'J':19,
                'K':20,
                'L':21,
                'M':22,
                'N':23,
                'O':24,
                'P':25,
                'Q':26,
                'R':27,
                'S':28,
                'T':29,
```

```
        'U':30,
        'V':31,
        'W':32,
        'X':33,
        'Y':34,
        'Z':35}

similar_letters_1 = {
    u'K': u'<',
    u'4': u'<',
    u'C': u'<',
    u'E': u'F',
    u'O': u'D',

}

always_change_1 = {
    u'5': u'S',
    u'1': u'I',
    u'6': u'G',
    u'/': u'T',
    u'7': u'T',
    u'2': u'Z',
    u'0': u'O',
    u'4': u'<',
    u'$': u'S',
    u'-': u'<',
    u'Ý': u'Y'
}

similar_letters_2 = {
    u'K': u'<',
    u'4': u'<',
    u'B': u'8',
    u'E': u'F',
    u'S': u'5',
    u'5': u'S',
    u'I': u'1',
    u'1': u'I',
    u'/': u'7',
    u'T': u'7',
    u'7': u'T',
    u'O': u'0',
    u'6': u'G',
    u'-': u'<',
    u'%': u'4',
    u'Ý': u'Y'

}


leter2digit = {u'O': u'0', u'T': u'7', u'I':
u'1', u'B': u'8', u'S': u'5'}

def name_translit(text):
    text = text.upper()
    capital_letters = {
        u'A': u'A',
        u'A': u'A',
        u'Б': u'B',
        u'B': u'V',
        u'B': u'V',
```

```
    u'Г': u'H',
    u'Ґ': u'G',
    u'Д': u'D',
    u'Е': u'E',
    u'Е': u'E',
    u'Ё': u'E',
    u'^Є': u'YE',
    u'Є': u'IE',
    u'Ж': u'ZH',
    u'З': u'Z',
    u'З': u'Z',
    u'И': u'Y',
    u'І': u'I',
    u'І': u'I',
    u'^Ї': u'YI',
    u'Ї': u'I',
    u'^Й': u'Y',
    u'Й': u'I',
    u'Л': u'L',
    u'Н': u'N',
    u'Н': u'N',
    u'К': u'K',
    u'К': u'K',
    u'М': u'M',
    u'М': u'M',
    u'О': u'O',
    u'О': u'O',
    u'П': u'P',
    u'Р': u'R',
    u'Р': u'R',
    u'R': u'R',
    u'С': u'S',
    u'С': u'S',
    u'Т': u'T',
    u'Т': u'T',
    u'У': u'U',
    u'Y': u'U',
    u'Ф': u'F',
    u'Х': u'KH',
    u'Х': u'KH',
    u'Ц': u'TS',
    u'Ч': u'CH',
    u'Ш': u'SH',
    u'Щ': u'SHCH',
    u'^Ю': u'YU',
    u'Ю': u'IU',
    u'^Я': u'YA',
    u'Я': u'IA',
    # u'>': u'<',
    # u'«': u'<',
    # u'=': u'F',

}
translit_string = ""

s = '^' + text[0]
if s in capital_letters.keys():
    translit_string += capital_letters[s]
elif text[0] in capital_letters.keys():
    translit_string +=
capital_letters[text[0]]
```

```
    for i in text[1:]:

        if i in capital_letters.keys():
            translit_string +=
capital_letters[i]
        # else:
        #     translit_string += i

    return translit_string

def translit(string):
    """ This function works just fine """
    capital_letters = {
        u'Н': u'H',
        u'З': u'3',
        u'А': u'A',
        u'Б': u'B',
        u'В': u'B',
        u'Е': u'E',
        u'Ё': u'E',
        u'И': u'I',
        u'Ї': u'I',
        u'І': u'I',
        u'К': u'K',
        u'М': u'M',
        u'О': u'O',
        u'Р': u'P',
        u'Т': u'T',
        u'Х': u'X',
        u'С': u'C',
        u'Ч': u'4',
        u'У': u'Y',
        u'Г': u'F',
        u'Ш': u'M',
        u'>': u'<',
        u'«': u'<',
        u'=': u'F',

    }
    translit_string = ""

    for i in string:
        if i in capital_letters.keys():
            translit_string +=
capital_letters[i]
        else:
            translit_string += i

    return translit_string


def update_rezult(file_name, rezult):
    with open('rezult.json', 'r') as f:
        j = f.read()
        print(j)
    update = json.loads(j)
    update[file_name] = rezult
    update = json.dumps(update)
    with open('rezult.json', 'w') as f:
        f.write(update)
    return True
```

Docker file

```
FROM ubuntu:latest
RUN apt-get update -y
RUN apt-get install -y python3-pip
python3-dev build-essential
COPY . /app
WORKDIR /app
RUN pip3 install -r requirements.txt
ENTRYPOINT ["python3"]
CMD ["app.py"]
```

```
from flask import Flask, jsonify,
send_file
from flask import request

import base64
import json
import os
import shutil
import time
import recognizer
import rezult_processing
from settings import host, database,
user, password, TOKEN



app = Flask(__name__)
DEBUG = True
```

```
try:
    import db_connector
    connector =
db_connector.DB_connector(host,
database, user, password)
except Exception as e:
    print(e)

@app.route('/api', methods=['POST'])
def get():
    data = request.get_json()
    data = json.loads(data)
    autorization =
request.headers['Authorization']
    img =
base64.b64decode(data["img"])
    n = len(os.listdir('./images')) + 1
    file_name = './images/' + str(n) +
'.jpg'
    with open(file_name, 'wb') as f:
        f.write(img)

    # img = data['img']
    if autorization == TOKEN:


        try:
            ex = str(None)
            block_list =
rezult_processing.get_block_list(docu
ment=rez)
```

```
            mrz =
rezult_processing.MRZ_block(block_li
st, img=f)
            # mrz_text = mrz.get_mrz()

            mrz_info = mrz.get_check()
            is_right =
str(int(mrz_info['is_right']))
            pass_numb =
str(mrz_info['pass_numb'])
            country_code =
str(mrz_info['country_code'])
            birthday = mrz_info['birthday']
            end_date =
mrz_info['end_date']
            mrz_text = mrz_info['mrz_text']
            sex = mrz_info['sex']

        except Exception as ex:
            print(ex)
            mrz_text = ''
            is_right = '0'
            # rezult_json = {'mrz': '',
'is_right': '0', 'exception': e}
            country_code = 'None'
            pass_numb = 'None'
            birthday = 'None'
            end_date = 'None'
            sex = 'None'
        if is_right == '1':
            error = 'false'
        else:
            error = 'true'
        # print(is_right, mrz_text,
country_code, pass_numb)
        t = str(time.time())


connector.add_passport('heroku_v3_tes
t', is_right, mrz_text, country_code,
file_name, pass_numb, t)
        # rezult_check =
is_mistake.is_mistake(text)

        try:
            rezult_json = {'mrz': mrz_text,
                    # 'is_right': is_right,
                    #
'country_code':country_code,
                    #
'pass_numb':pass_numb,
                    # 'sex':sex,
                    # 'birthday': birthday,
'end_date': end_date,
                    # 'exception': ex,
                    'error': error}
        except:

            rezult_json = {'mrz': mrz_text,
                    # 'is_right': is_right,
                    # 'country_code':
country_code,
```

```
                    # 'pass_numb':
pass_numb,
                    # 'sex': sex,
                    # 'birthday': birthday,
'end_date': end_date,
                    # 'exception': 'n',
                    'error': error}

        # file_name = str(n) + '.jpg'
        # update =
common.update_rezult(file_name,
str(rezult_json))
        # print(update)
    else:
        rezult_json = {'Autorization error':
True}

    return jsonify(rezult_json)

@app.route('/', methods=['POST'])
def test():
    data = request.get_json()
    data = json.loads(data)

    rezult_json = {'success': True}
    return jsonify(rezult_json)
from telegram.ext import Updater,
CommandHandler, MessageHandler,
Filters
import settings
import os
import requests
import base64
import json
import re

# from database import *
DEBUG = True
TEST = True


class Bot():

    def __init__(self):

        self.start_bot()

    def reply_to_start_command(self,
bot, update):
        user = update.message.from_user
        user_id = user.id

        update.message.reply_text(
            "Привіт! За допомогою мене
ти зможеш розпізнати MRZ код
закордонного паспорту ☺\n"
            "Все що треба - це відправити
фото закордонного паспорту",)

        start_photo_id =
'AgADAgADcKoxG2b_cEvmMbRDy
```

```
khuKH9HhA8ABAy2SAuHAcswxDs
BAAEC'
        m =
bot.send_photo(chat_id=user_id,
photo=start_photo_id)

    def photo_handler(self, bot, update):
        print('Photo handler')
        user = update.message.from_user
        first_name = str(user.first_name)
        last_name = str(user.last_name)
        username = str(user.username)

        admin_msg = 'Пользователь ' +
first_name + username + ' отправил
картинку'
        # m =
bot.sendMessage(362212345,
admin_msg)

        file_id = update.message.photo[-
1].file_id
        newFile = bot.getFile(file_id)
        newFile.download('test.jpg')
        print(newFile)
        file_path = newFile.file_path
        file_id = newFile.file_id
        # m = bot.sendPhoto(362212345,
file_id)

        # url = r'https://passport-
detection.herokuapp.com/api'
        # url = r'https://print-me-
bot.herokuapp.com/api'
        url = r'https://pass-
bot.herokuapp.com/api'

        path_img = 'test.jpg'

        with open(path_img, "rb") as file:
            encoded_string =
base64.b64encode(file.read())
        req_file =
encoded_string.decode('utf-8')

        headers = {
            "Authorization": 'Bearer
YyYLcNPoiLq2SZz8PQqL'
        }

        payload = {'img': req_file}

        res = requests.post(url,
json=json.dumps(payload),
headers=headers)
        print(res)
        data = res.json()


        if data['error'] == 'false':
            text = 'MRZ:\n'
```

```
        else:
            text = 'Не вдалося правильно розпізнати паспорт. \nСпробуйте використати фото кращої якості або під іншим кутом нахилу. \nРезультат распізнавания:\n'
        text = text + data['mrz']
        try:

            sep = re.search('<{3,}', text)[0]
            first = text.split(sep)[0] + sep
            second = text.split(sep)[1]
            text = first + '\n' + second
        except:
            pass


bot.sendMessage(chat_id=update.message.chat_id, text=text)
        # m = bot.sendMessage(chat_id=362212345, text=text)


    def start_bot(self):
        print('bot started')

        if DEBUG == False:
            PORT = int(os.environ.get('PORT', '8443'))
            PORT = int(os.environ.get('PORT', '5000'))

            updater = Updater(settings.TELEGRAM_API_KEY)

            dp = updater.dispatcher

dp.add_handler(CommandHandler("start", self.reply_to_start_command))

dp.add_handler(MessageHandler(Filters.photo, self.photo_handler))

updater.start_webhook(listen="0.0.0.0",
                      port=PORT,
url_path=settings.TELEGRAM_API_KEY)


updater.bot.set_webhook("https://passport-bot.herokuapp.com/" + settings.TELEGRAM_API_KEY)
            #
updater.bot.set_webhook("https://test-pass-bot.herokuapp.com/" + settings.TELEGRAM_API_KEY)
```

```
            updater.idle()

        else:

            updater =
Updater(settings.TELEGRAM_API_K
EY_test)

            dp = updater.dispatcher

dp.add_handler(CommandHandler("sta
rt", self.reply_to_start_command))

dp.add_handler(MessageHandler(Filter
s.photo, self.photo_handler))
            #
dp.add_handler(CommandHandler("hel
p", self.get_advice))



            updater.start_polling()
            updater.idle()



if __name__ == "__main__":
    bot = Bot()
```