МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Інститут інформатики та радіоелектроніки,
Факультет радіоелектроніки та телекомунікацій
(повне найменування інституту, факультету)
Кафедра інформаційних технологій електронних засобів
(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)
бакалавр
(ступінь вищої освіти)

на тему Розробка програмного забезпечення для відтворення pdf-файлу з отриманого зображення на основі розпізнавання образів
Development of software program to restore pdf-file from received picture based on image recognation

Виконав: студент 4 курсу, групи РТ-518 Сп

Спеціальності 172 Телекомунікації та радіотехніка
(код і найменування спеціальності)

Освітня програма (спеціалізація)
Інтелектуальні технології мікросистемної радіоелектронної техніки

Михайлов Є. І.
(прізвище та ініціали)
Керівник Фарафонов О. Ю.
(прізвище та ініціали)
Рецензент Неласа Г. В.
(прізвище та ініціали)

2021

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
(повне найменування закладу вищої освіти)

Інститут, факультет Інститут інформатики та радіоелектроніки, факультет радіоелектроніки та телекомунікацій
Кафедра інформаційних технологій електронних засобів
Ступінь вищої освіти бакалавр
Спеціальність 172 Телекомунікації та радіотехніка
(код і найменування)
Освітня програма (спеціалізація) Інтелектуальні технології мікросистемної радіоелектронної техніки
(назва освітньої програми (спеціалізації)

ЗАТВЕРДЖУЮ
Завідувач кафедри ІТЕЗ
Огренич Є.В.
«31» травня 2021 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

Михайлов Євген Ігорович
(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Розробка програмного забезпечення для відтворення pdf-файлу з отриманого зображення на основі розпізнавання образів
керівник проєкту (роботи) Фарафонов О. Ю., к.т.н., доцент,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом закладу вищої освіти від «26» квітня 2021 року № 161
2. Строк подання студентом проєкту (роботи)
3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) Реферат, Вступ, 1 Розпізнавання образів за допомогою OpenCV, 2 Створення сканеру фотографії, Технічна частина, Висновки, Перелік джерел.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
Основна частина	Фарафонов О. Ю., доцент	5.04	
Нормоконтроль	Поспєєва І. Є., доцент	28.05	

7. Дата видачі завдання «26» квітня 2021 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи	1 тиждень	Завдання, ТЗ
2	Розпізнавання за допомогою OpenCV	2-3 тиждень	Розділ 1
3	Створення сканеру фотографії	4-5 тиждень	Розділ 2
4	Технічна частина	6-7 тиждень	Розділ 3
5	Оформлення пояснювальної записки та документів до неї. Нормоконтроль та рецензування.	8 тиждень	Презентація
6	Захист роботи	9 тиждень	

Студент(ка) _____ Михайлов Є. І.
(підпис) (прізвище та ініціали)

Керівник проєкту (роботи) _____ Фарафонов О. Ю.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

ПЗ: 49 листів, 37 рис., 12 джерел.

Задача дипломного проекту – програмне забезпечення для відтворення pdf-файлу з фотографії.

Дипломний проект складається з пояснювальної записки. Пояснювальна записка складається с трьох розділів – розпізнавання з допомогою opencv, сканування фотографії та технічної частини.

В першому розділі надаються загальні відомості про використовувані технології, їх можливості та принципи роботи, а саме мова програмування Python, бібліотека OpenCv, фреймворк Django та інше, а також коротко розглядаються існуючі на ринку схожі проекти. Надано літературний огляд базовим можливостям бібліотеки OpenCv.

В другому розділі надано повний процес створення сканеру фотографії.

В третьому розділі представлений шлях для створення web-сервісу для сканування фотографії також можливі варіанти розгортання даного сервісу, а саме розгортання на сервісі Amazon Web Services EC2, з допомогою Apache або Ngnix, або розгортання на сервісі Heroku.

ПРОГРАМУВАННЯ, PYTHON, OPENCV, DJANGO, РОЗПІЗНАВАННЯ ОБРАЗІВ, WEB-СЕРВІСИ, РОЗГОРТАННЯ

ЗМІСТ

ВСТУП

20 років тому, в 1999 році, компанія Kyocera випустила перший мобільний телефон з цифровою камерою - Visual Phone VP-210. З тих пір, завдяки неймовірно великим і зростаючому ринку мобільних пристроїв зв'язку, ПЗС-матриці цифрових камер зробили неймовірний стрибок за всіма параметрами. Чутливість, діапазон, розмір, енергоспоживання, але що ще важливіше - ціна.

У сучасних реаліях модуль камери, дуже технологічно складний пристрій, може коштувати всього кілька доларів. Це кардинально змінює погляд на багато процесів і завдань. Раніше складним завданням було дістати камеру яка технічно задовольняла мінімальним вимогам. Пройшовши таке випробування, вирішення питань обробки зображень здавалося лише приємними клопотами. Тепер же питання програмного забезпечення, яке буде обробляти інформацію з камери, варто більш гостро розглядати. Планка фізичного і економічного доступу до технології впала так низько, що торкнулася кордону компетентності користувача.

Безумовно, головним інструментом для роботи з зображеннями є Open Source бібліотека OpenCV. Написана на C ++ - вона також має інтерфейси для роботи з Python, Java, PHP, JavaScript і іншими, менш популярними мовами програмування.

На сьогоднішній день документооборот все більше переходить в цифровий формат. Але не кожна людина має можливість, або бажання придбати пристрій для сканування документів, або інших паперів для подальшого їх використання в мережі. В цей момент на допомогу приходять додатки сканери документів – які виконують безпосередню роботу повно-розмірних сканерів.

Мета роботи – розробка програмного забезпечення для відтворення pdf-файлу з отриманого зображення на основі розпізнавання образів з допомогою бібліотеки OpenCV.

В результаті буде створено web-сервіс з допомогою якого можна отримати скановані варіанти початкового зображення, для подальшого їх використання.

1 РОЗПІЗНАВАННЯ ЗА ДОПОМОГОЮ OpenCV

1.1 Оглядова частина

Тема дипломного проекту «Розробка програмного забезпечення для відтворення PDF-файлу з отриманого зображення на основі розпізнавання образів». З назви випливає, що мета дипломного проекту полягає в тому, щоб обробляти фотографії з документами, чеками або з іншими даними які надходять з сайту і перетворювати їх в друкований вигляд з вивантаженням в PDF-файл, для подальшого використання користувачами в своїх цілях. В дипломній роботі використовуються такі технології як: мова програмування Python, Web-фреймворк Django, також фреймфорк OpenCV2, бібліотека для математичних обчислень NumPy, сервіс хмарних обчислень Amazon Web Services, а сама EC2 та RDS, а також база даних PostgreSQL.

На даний момент на ринку існує декілька подобних рішень. А саме Scannable від розробників Evernote, CamScanner, а також Lens від Microsoft.

Scannable підходить для сканування квитанцій, візиток і друкованих документів. Додаток використовує оптичне розпізнавання букв і цифр, яке дозволяє шукати слова і фрази в відсканованому цифровому документі. Функція працює автоматично, аналізуючи слова і знаки документа, поки ви його скануєте. Також Scannable використовує цю функцію щоб виділяти інформацію на візитках (наприклад, email і телефонні номери) з можливістю її подальшого збереження в телефоні.

CamScanner використовує камеру телефону для сканування документів, рецептів, візиток, окремих слів і навіть хитромудрих каракуль з дошки. CamScanner аналізує зміст зображення, щоб системно все організувати, наприклад: скани візиток зберігаються в одній папці, а скани документів - в інший. В якості додаткового бонусу додаток дозволяє робити скани з

фотографій, знятих раніше. Якість відсканованого тексту висока, хоча може варіюватися в залежності від характеристик камери смартфона.

Lens від Microsoft підтримує усе з вище перерахованого, а також має можливість пересилки сканованих файлів в канал Microsoft. Ви можете зберігати або додавати їх на OneNote і OneDrive.

У дипломній роботі використовуються наступні технологій.

Python — інтерпретована об'єктно-орієнтована мова програмування високого рівня зі строгою динамічною типізацією. Розроблена в 1990 році Гвідо ван Россумом. Структури даних високого рівня разом із динамічною семантикою та динамічним зв'язуванням роблять її привабливою для швидкої розробки програм, а також як засіб поєднування наявних компонентів. Python підтримує модулі та пакети модулів, що сприяє модульності та повторному використанню коду. Інтерпретатор Python та стандартні бібліотеки доступні як у скомпільованій, так і у вихідній формі на всіх основних платформах. В мові програмування Python підтримується кілька парадигм програмування, зокрема: об'єктно-орієнтована, процедурна та функціональна. Серед основних її переваг можна назвати такі:

- чистий синтаксис;
- переносність програм;
- стандартний дистрибутив має велику кількість корисних модулів;
- можливість використання Python в діалоговому режимі;
- зручний для розв'язання математичних проблем;

Django - це високоякісний Web-фреймворк для Python, який стимулює швидкий розвиток та чистий, прагматичний дизайн. Побудований досвідченими розробниками, він опікується великою частиною клопоту щодо Web-розробки, тому при написанні програми можливо зосередитись на написанні програми, не вимагаючи винаходити колесо. Увесь фреймворк йде з відкритим

кодом. Django був розроблений, щоб допомогти розробникам якомога швидше приймати програми від концепції до завершення. Django серйозно ставиться до безпеки та допомагає розробникам уникнути багатьох типових помилок у безпеці. Деякі з найбільш завантажених веб-сайтів використовують здатність Django швидко та гнучко масштабуватись.

OpenCV2 - це фреймворк для Python, призначена для вирішення проблем із комп'ютерним зором. Зараз OpenCV підтримує безліч алгоритмів, пов’язаних із комп’ютерним зором та машинним навчанням, і розширюється з кожним днем. OpenCV2 використовує Numpy, що є високо оптимізованою бібліотекою для числових операцій із синтаксисом у стилі MATLAB. Усі структури масивів OpenCV2 перетворюються в масиви Numpy і навпаки. Це також полегшує інтеграцію з іншими бібліотеками, які використовують Numpy, такими як SciPy та Matplotlib.

NumPy - це основний пакет для наукових обчислень у Python. Це бібліотека Python, яка забезпечує багатовимірні об’єкти масиву, різні похідні об’єкти (наприклад, масковані масиви та матриці) та асортимент підпрограм для швидких операцій над масивами, включаючи математичні, логічні, маніпуляції з фігурами, сортування, виділення, введення/виведення, дискретні перетворення Фур'є, основна лінійна алгебра, основні статистичні операції, випадкове моделювання та багато іншого. В основі пакета NumPy лежить об'єкт ndarray. Це інкапсулює n-вимірні масиви однорідних типів даних, причому багато операцій виконуються в скомпільованому коді для продуктивності.

Amazon Web Services (AWS) - це сама розповсюджена в світі хмарова платформа з широкими можливостями, що забезпечує більш ніж 200 повнофункціональних сервісів для центрів обробки даних на всій планеті. Мільйони клієнтів, у тому числі стартапів, ставших лідерами за швидкістю роста,

найбільші корпорації та передові державні установи, використовують AWS для зниження затрат, підвищення гнучкості та прискореного впровадження інновацій.

У дипломній роботи будуть використовуватися два сервіси, а саме EC2, та RDS. Amazon Elastic Compute Cloud (Amazon EC2) - це веб-сервіс, що надає безпечні масштабовані обчислювані ресурси в хмарах. Він допомагає розробникам, проведенням розрахунків у хмарах у масштабі всього Інтернету. Простий веб-інтерфейс сервісу Amazon EC2 дозволяє отримувати доступ до обчислювальних ресурсів та налаштовувати їх з мінімальними зусиллями. Він пропонує користувачеві повний контроль над обчислювальними ресурсами, а також перевірену обчислювальну середу Amazon для роботи. Relational Database Service (Amazon RDS) дозволяє налаштовувати, використовувати і масштабувати реляційні бази даних в хмарі. Сервіс забезпечує економічне і масштабується використання ресурсів при одночасній автоматизації трудомістких завдань адміністрування, таких як виділення апаратного забезпечення, налаштування бази даних, установка виправлень і резервне копіювання. Це дозволяє зосередити увагу на додатках, щоб забезпечити для них високу продуктивність, високу доступність, безпеку і сумісність.

PostgreSQL - це потужна об'єктно-реляційна система баз даних з відкритим кодом, яка використовує та розширює мову SQL у поєднанні з багатьма функціями, які безпечно зберігають та масштабують найскладніші робочі навантаження даних. Походження PostgreSQL бере свій початок у 1986 році в рамках проекту POSTGRES в Університеті Каліфорнії в Берклі і має понад 30 років активного розвитку на базовій платформі. PostgreSQL заслужив сильну репутацію завдяки своїй перевіреній архітектурі, надійності, цілісності даних, надійному набору функцій, розширюваності та відданості спільноти з відкритим кодом, що стоїть за програмним забезпеченням, для постійного забезпечення продуктивних та інноваційних рішень.

По частинах що і для чого буде використовуватися в дипломній роботі. Web-фреймфорк Django буде використаний для створення інтерфейсу для отримання даних, а саме фотографій, від користувачів і подальшої передачі їх в обробку, а також отримання відповіді з результатом роботи функції для обробки фотографії, а так само для відображення результату її роботи. OpenCV2 в зв'язці з NumPy будуть використовуватися для обробки зображення, для отримання в подальшому сканованою версії фотографії. EC2 буде використаний в якості сервера на якому буде розгорнуто весь проект і до нього буде доступ з мережі інтернет. RDS буде використаний для створення бази даних, а саме PostgreSQL, і зберігання даних в ній, що б користувачі могли отримати доступ до вже «відсканованих» файлів.

1.2 Робота з OpenCv

Перше, що необхідно зробити - це імпортувати бібліотеку. Є кілька шляхів імпорту, найпоширеніший - це використовувати вираз:

import cv2

Також можна зустріти наступну конструкцію імпорту даної бібліотеки:

from cv2 import cv2

Так як OpenCV це open source бібліотека комп'ютерного зору, яка призначена для аналізу, класифікації та обробки зображень, перш за все нам потрібно відкрити зображення для подальшої його обробки. Для цього в OpenCV є функція *cv2.imread()*, де першим аргументом вказується шлях до зображення, а другим аргументом, який є необов'язковим, вказується, в якому колірному просторі користувач хоче використовувати зображення. Щоб вважати зображення в RGB - *cv2.IMREAD_COLOR*, в відтінках сірого - *cv2.IMREAD_GRAYSCALE*. За замовчуванням цей аргумент приймає значення *cv2.IMREAD_COLOR*. Ця функція повертає 2D (для зображення у відтінках сірого) або 3D (для кольорового зображення) масив NumPy. Форма масиву для

кольорового зображення: висота х ширина х 3, де 3 - це байти, по одному байту на кожну з компонент. У зображеннях у відтінках сірого все трохи простіше: висота х ширина.

За допомогою функції *cv2.imshow()* відображається зображення на екрані. Як перший аргумент передається функції назву майбутнього вікна, а другим аргументом зображення, яке завантажилось з диска, проте, якщо далі не буде використовувати функцію *cv2.waitKey()*, то зображення моментально закриється. Ця функція зупиняє виконання програми до натискання клавіші, яку потрібно передати першим аргументом. Для того, щоб будь-яка клавіша була зарахована передається 0.

І, нарешті, за допомогою функції *cv2.imwrite()* записується зображення в файл в форматі jpg (дана бібліотека підтримує всі популярні формати зображень: png, tiff, jpeg, bmp і т.д., Тому можна зберегти зображення в будь-якому з цих форматів), де першим аргументом передається безпосередньо сама назва і розширення, а наступним параметром зображення, яке користувач хоче зберегти.

Так же в даній бібліотеці доступні базові перетворенням зображень: зміна розміру, зміщення вздовж осей, кадрування (обрізка), поворот. Розглянемо їх детальніше.

1.2.1 Зміна розміру зображень

Перший метод, який буде розглядатися - це як змінити висоту і ширину у зображення. Для цього в OpenCV є така функція як *resize()*:

cv2.resize (image, (500, 900), cv2.INTER_NEAREST)

Ця функція першим аргументом приймає зображення, розмір якого користувач хоче змінити, другим - кортеж, який повинен містити в собі ширину і висоту для нового зображення, третім - метод інтерполяції (необов'язковий). Інтерполяція - це алгоритм, який знаходить невідомі проміжні значення по

наявного набору відомих значень. Фактично, це те, як будуть заповнюватися нові пікселі при модифікації розміру зображення. Наприклад, інтерполяція методом найближчого сусіда (*cv2.INTER_NEAREST*) просто бере для кожного пікселя підсумкового зображення один піксель вихідного, який найбільш близький до його положенню - це самий простий і швидкий спосіб. Крім цього методу в OpenCV існують такі: *cv2.INTER_AREA*, *cv2.INTER_LINEAR* (використовується за умовчанням), *cv2.INTER_CUBIC* і *cv2.INTER_LANCZOS4*. Найкращим методом інтерполяції для стиснення зображення є *cv2.INTER_AREA*, для збільшення - *cv2.INTER_LINEAR*. Від даного методу залежить якість кінцевого зображення, але як показує практика, якщо користувач зменшує/збільшує зображення менше, ніж в 1.5 рази, то не має значення яким методом інтерполяції він скористався - якість буде схожою.

1.2.2 Зсув зображення уздовж осей

Для демонстрацій зміни зображення за основу береться рис.1.1

Рисунок 1.1 – Зображення для тесту

За допомогою функції *warpAffine()* користувач може переміщати вліво і вправо, вниз та вгору, а також будь-яку комбінацію з перерахованого:

```
h, w = img.shape[: 2]
translation_matrix = np.float32([[1, 0, 200], [0, 1, 300]])
dst = cv2.warpAffine (img, translation_matrix, (w, h))
```

Для того, щоб дізнатися висоту, ширину і кількість каналів у зображення можна використовувати атрибут *shape*, де:

- *img.shape[0]* – це висота;
- *img.shape[1]* – це ширина;
- *img.shape[2]* – це кількість каналів.

Важливо пам'ятати, що у зображень у відтінках сірого *img.shape[2]* буде недоступно, так як дані зображення представлені у вигляді 2D масиву.

Спочатку, в змінної *translation_matrix* створюється матриця перетворень як показано на рис.1.2

$$R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

Рисунок 1.2 – Матриця перетворень

Перший рядок матриці - [1, 0, tx], де tx - кількість пікселів, на які користувач буде зрушувати вліво або вправо. Негативне значення tx буде зрушувати вліво, позитивне - вправо.

Другий рядок матриці - [0, 1, ty], де ty - кількість пікселів, на які користувач буде зрушувати зображення вгору або вниз. Негативне значення ty буде зрушувати зображення вгору, позитивне - вниз. Важливо пам'ятати, що дана матриця визначається як масив з плаваючою точкою.

На наступному рядку і відбувається зсув зображення уздовж осей, за допомогою, як вказувалося вище, функції *cv2.warpAffine()*, яка першим

аргументом приймає - зображення, другим - матрицю, третім - розміри зображення. Результат роботи даного фрагменту наведений на рис.1.3.

Рисунок 1.3 – Зміщене зображення

1.2.3 Виріз фрагмента зображення

Для того, щоб вирізати фрагмент з зображення, досить скористатися наступним кодом:

crop_img = img [10: 450, 300: 750]

У даному кроці користувач надає масив NumPy для вилучення прямокутної області зображення, починаючи з (300, 10) і закінчуючи (750, 450), де 10 - це початкова координата по y, 300 - початкова координата по x, 450 - кінцева координата по y і 750 - кінцева координата по x. Виконавши даний код результатом буде рис.1.4.

Рисунок 1.4 – Вирізаний фрагмент зображення

1.2.4 Поворот зображення

І, останнє, що розглянеться - це як повернути зображення на деякий кут.

(h, w) = img.shape[: 2]

center = (int(w / 2), int(h / 2))

rotation_matrix = cv2.getRotationMatrix2D(center, -45, 0.6)

rotated = cv2.warpAffine(img, rotation_matrix, (w, h))

Коли користувач повертає зображення, потрібно вказати, навколо якої точки буде обертатися зображення, саме це приймає першим аргументом функція *cv2.getRotationMatrix2D()*. В даному випадку вказується центр зображення, проте OpenCV дозволяє вказати будь-яку довільну точку, навколо якої користувач захоче обертатися. Наступним аргументом дана функція приймає кут, на який користувач хоче повернути зображення, а останнім аргументом - коефіцієнт масштабування. В даному випадку використовується 0.6, тобто зменшується зображення на 40%, для того, щоб воно помістилося в кадр. Ця функція повертає масив NumPy, який передається другим аргументом в функцію *cv2.warpAffine()*. У підсумку отримується рис.1.5.

Рисунок 1.5 – Повернуте зображення

2 СТВОРЕННЯ СКАНЕРУ ФОТОГРАФІЇ

2.1 Розпізнавання об’єктів

Розпізнавання об'єктів проводиться за допомогою колірної сегментації зображення. Для цього є дві функції: *cv2.findContours()* і *cv2.drawContours()*.

Спочатку треба об’явити деякі важливі терміни: контури та пороги.

Контури можна пояснити просто як криву, яка з'єднує всі безперервні точки (разом з кордоном), що мають однаковий колір або інтенсивність. Контури - корисний інструмент для аналізу форми і виявлення і розпізнавання об'єктів.

Застосування порогових значень до зображення у відтінках сірого робить його бінарним зображенням. Користувач встановлює порогове значення, при якому всі значення нижче цього порога стають чорними, а всі значення вище стають білими.

Спочатку буде розглянута функція *cv2.findContours():*

contours, hierarchy = cv.findContours(image, mode, method[, contours[, hierarchy[, offset]]])

Функція знаходить контури на двійковому зображенні.

Функція приймає наступні параметри:

- *image* - 8-бітове одно канальне зображення. Ненульові пікселі обробляються як одиниці. Нульові пікселі залишаються нульовими, тому зображення обробляється як двійкове. Ви можете використовувати compare, inRange, threshold, adaptiveThreshold, Canny і інші, щоб створити двійкове зображення з полу тонового або кольорового зображення. Якщо режим дорівнює RETR_CCOMP або RETR_FLOODFILL, вхід також може бути 32-бітовим цілим зображенням міток;
- *contours* - виявлені контури. Кожен контур зберігається як вектор точок;

- *hierarchy* - необов'язковий вихідний вектор, що містить інформацію про топології зображення. У ньому стільки елементів, скільки контурів. Для кожного і-го контуру *contours*[i], ієрархія елементів [i] [0], *hierarchy*[i] [1], *hierarchy*[i] [2] і *hierarchy*[i] [3]] встановлюються на 0- індекси на основі контурів наступного і попереднього контурів на тому ж ієрархічному рівні, перший дочірній контур і батьківський контур відповідно. Якщо для контуру і відсутні такі, попередні, батьківські або вкладені контури, відповідні елементи *hierarchy*[i] будуть негативними;
- *mode* - режим вилучення контуру;
- method - метод апроксимації контуру;
- offset - додаткове зміщення, на яке зміщується кожна точка контуру. Це корисно, якщо контури витягуються з області інтересу зображення, а потім їх слід аналізувати в контексті всього зображення.

Наступна функція *cv2.drawContours():*

image=cv.drawContours(image, contours, contourIdx, color[, thickness[, lineType[, hierarchy[, maxLevel[, offset]]]]])

Функція малює контури або контури з заливкою.

Функція малює контури на зображенні, якщо товщина ≥0 або заповнює область, обмежену контурами, якщо товщина < 0.

Функція приймає наступні параметри:

- image - цільове зображення;
- contours - всі вхідні контури. Кожен контур зберігається як вектор точки;
- contourIdx - параметр, який вказує контур для малювання. Якщо він негативний, малюються всі контури;
- color - колір контурів;
- thickness - товщина ліній, за якими намальовані контури. Якщо він негативний внутрішня частина контуру промальовується;
- lineType - лінія зв'язку;

- hierarchy - необов'язкова інформація про ієрархію. Це потрібно тільки в тому випадку, якщо ви хочете намалювати тільки частину контурів;
- maxLevel - максимальний рівень намальованих контурів. Якщо він дорівнює 0, малюється тільки зазначений контур. Якщо він дорівнює 1, функція малює контур і всі вкладені контури. Якщо він дорівнює 2, функція малює контури, всі вкладені контури, всі контури, вкладені у вкладені, і так далі. Цей параметр враховується тільки при наявності ієрархії.
- offset Необов'язковий параметр зміщення контуру. Зрушити всі намальовані контури на вказане offset = (dx, dy).

2.2 Розробка програмного забезпечення для відтворення сканованого зображення

Створення сканованого зображення за допомогою OpenCV буде виконатися за три основних кроки:

1) Виявлення країв;

2) Використання країв зображення, щоб знайти контур, що буде представляти собою сканований лист паперу;

3) Застосування перетворень перспективи, щоб отримати вид зверху вниз.

Основний код знаходиться в файлі *scan_utils.py*.

Спочатку потрібно імпортувати усі необхідні для роботи модулі, представлені на рис 2.1.

```
1  from skimage.filters import threshold_local
2  import numpy as np
3  import cv2
4  import imutils
```

Рисунок 2.1 – Імпорт необхідних модулів

Функція *threshold_local()* допоможе в отриманні відчуття сканованого зображення.

NumPy використовується для числової обробки.

Модуль *imutils* буде використовуватися для зміни розмірів зображення, повертання, а також обрізки зображення.

Коли всі модулі імпортовано можна починати роботу з виявлення країв. Для цього використовується основна функцію *scan_file()* яка приймає своїм аргументом ім’я зображення яке в подальшому буде використовуватися для збереження зображення. Строки 44-48 відповідають за отримання зображення з бази даних, для подальшої його обробки, та перетворення в потрібний для OpenCV формат даних, а саме масив байтів (рис.2.2).

```
42
43  def scan_file(name):
44      image_obj = ScanFile.objects.last()
45      image = image_obj.photo
46
47      image = np.asarray(bytearray(image.read()), dtype="uint8")
48      image = cv2.imdecode(image, cv2.IMREAD_COLOR)
```

Рисунок 2.2 – Отримання зображення

Далі у строках 50-52 змінюється розмір зображення до висоти 500 пікселів, це робиться для більш швидкої обробки зображення, а також щоб виявлення країв було більш точним. Також особливо важливо стежити за тим щоб відстежувати відношення вихідної висоти зображення до нової висоти (рядок 51) - це дозволить виконувати сканування на оригінальному зображенні, а не на зміненому зображенні (рис.2.3).

```
50      ratio = image.shape[0] / 500.0
51      orig = image.copy()
52      image = imutils.resize(image, height=500)
```

Рисунок 2.3 – Змінення розміру зображення

Далі перетворюється зображення з RGB у відтінки сірого (рядок 54), виконується розмиття Гауса (рядок 55), щоб усунути високочастотні шуми (це має допомогти у виявленні контуру на наступному етапі), і виконується виявлення краю Canny на рядку 56 (рис.2.4).

```
54      gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
55      gray = cv2.GaussianBlur(gray, (5, 5), 0)
56      edged = cv2.Canny(gray, 75, 200)
```

Рисунок 2.4 – Маніпуляція з зображенням

Після цих кроків отримую результат на рис.2.5

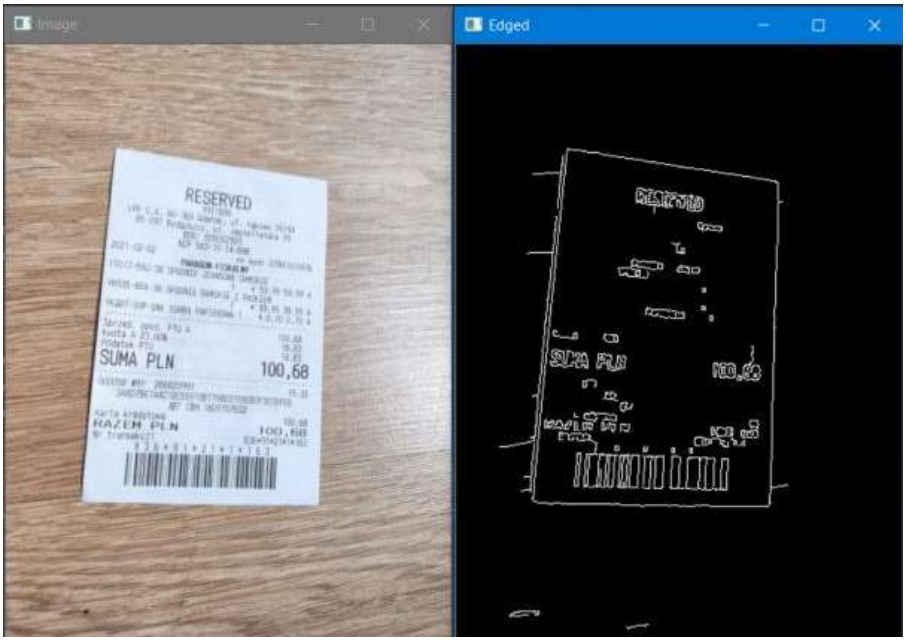


Рисунок 2.5 – Результат роботи першої частини

По зображенню добре видно контур чеку, отже можна переходити до наступного етапу пошук контурів.

Отже сканер просто сканує лист бумаги > Лист бумаги в свою чергу представляє собою прямокутник > Прямокутник має чотири края. Слідкуючи з цього отримується простий алгоритм, який має допомогти у пошуку контурів.

Він виглядає наступним чином: робиться догадка, що найбільший контур зображення рівно з чотирма точками – це лист бумаги, який потрібно сканувати. Додатки сканери допускають, що документ, який людина хоче відсканувати, являється головним об’єктом зображення. А також можна сміло припустити, або принаймні, так повинно бути, що лист бумаги має чотири края. Саме це і робить наступний код. Все починається з пошуку контурів на рядку 58 (рис.2.6). Далі робиться сортування отриманих контурів, залишаючи тільки 5 найбільших контурів. Це допомагає швидше знайти найбільший замкнутий контур відкидаючи інші.

```
58      cnt = cv2.findContours(edged.copy(), cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)
59      cnt = imutils.grab_contours(cnt)
60      cnt = sorted(cnt, key=cv2.contourArea, reverse=True)[:5]
```

Рисунок 2.6 – Пошук контурів

Наступним кроком починається обход в циклі отримані контури і приблизно отримувати кількість точок на рядках 63-64. І якщо приблизно отриманий контур має 4 точки, то робиться висновок що скоріш за все це і є потрібний елемент зображення (рис.2.7).

```
62      for c in cnt:
63          peri = cv2.arcLength(c, True)
64          approx = cv2.approxPolyDP(c, 0.02 * peri, True)
65          if len(approx) == 4:
66              screen_cnt = approx
67              break
```

Риунок 2.7 – Пошук основного контуру

І частіше за все такий висновок безпечний, так як сканер робить припущення що документ який потрібно сканувати являється основним фокусом зображення і сам документ прямокутний, отже, буде мати чотири чітких края. На рис.2.8 показано результат роботи цієї частини.

Рисунок 2.8 – Результат роботи по пошуку контурів

Дивлячись на зображення можна зрозуміти, що етап пройшов добре і програма виявила основний контур чеку показаний на зображенні зеленим контуром.

І нарешті останній етап отримання перспективи виду згори. Потрібно взяти чотири точки, які представляють собою контур документа, і використати на них перспективне перетворення для отримання зображення згори до низу.

Код представлений на рис.2.6.

```
69      warped = four_point_transform(orig, screen_cnt.reshape(4, 2) * ratio)
70
71      warped = cv2.cvtColor(warped, cv2.COLOR_BGR2GRAY)
72      t = threshold_local(warped, 11, offset=10, method="gaussian")
73      warped = (warped > t).astype("uint8") * 255
74
75      ret, buf = cv2.imencode('.jpg', warped)
76      content = ContentFile(buf.tobytes())
77
78      image_obj.scanned_photo.save(f'scanned_{name}.jpg', content)
```

Рисунок 2.6 – Перетворення для виду згори

Основну роботу виконує ще одна функція *four_point_transform*, в яку передається два аргументи, перший – це початкове зображення (не змінене у розмірі), а другий аргумент – це контур, представляючий собою документ, помножений на коефіцієнт зміни розміру.

Множення робиться тому що раніше робився пошук країв, а також контурів на зображенні розміром 500 пікселів, але потрібно сканувати початкове зображення, а не зображення з зміненим розміром, тому тут використовується множення на коефіцієнт зміни розміру.

Принцип роботи функції *four_point_transform()* буде розглянутий після ще однієї важливої функції – order_points(), яка приймає своїм аргументом, список з чотирьох точок, що задають координати (x, y) кожної точки прямокутника. Функція представлена на рис.2.7.

```
14  def order_points(pts):
15      rect = np.zeros((4, 2), dtype="float32")
16
17      s = pts.sum(axis=1)
18      rect[0] = pts[np.argmin(s)]
19      rect[2] = pts[np.argmax(s)]
20
21      diff = np.diff(pts, axis=1)
22      rect[1] = pts[np.argmin(diff)]
23      rect[3] = pts[np.argmax(diff)]
24
25      return rect
```

Рисунок 2.7 – Функція order_points

Найважливіший момент, щоб напротязі усієї програми була послідовність точок прямокутника. Сам порядок може бути будь-яким якщо на протязі всієї програми він однаковий. В даному випадку використовується наступна послідовність – верхній лівий, верхній правий, нижній правий і нижній лівий.

У рядку 15 виділяється пам’ять для чотирьох впорядкованих точок. Далі йде пошук верхньої лівої точки, яка буде мати найменшу суму x + y, і нижню праву точку яка буде мати найбільшу суму x + y, данні розрахунки показані на рядку 18-19.

Тепер залишилося знайти верхню праву точку, а також нижню ліву точку. Тут буде використовуватися різниця (тобто x – y) між точками (рядок 21). Координати, з найменшою різницею, будуть верхньою правою точкою, тоді як координати з найбільшою різницею будуть нижньою лівою точкою (рядок 22-23).

Нарешті повертаються упорядковані значення для подальшого їх використання.

Тепер можна розглядати функцію four_point_transform(). Функція приймає два аргументи, саме зображення, а також послідовність з чотирьох точок.

У рядку 29 викликається раніше розглянута функція order_points(). Яка розміщує змінну pts у послідовному порядку. Далі ця послідовність розпаковується на рядку 30, для зручності використання(рис.2.8).

```
28  def four_point_transform(img, pts):
29      rect = order_points(pts)
30      (tl, tr, br, bl) = rect
```

Рисунок 2.8 – Отримання результатів функції order_points

Тепер потрібно отримати розмір нового деформованого зображення. Отримується ширина зображення наступним чином у рядках 31-33, де ширина це найбільша відстань між правою нижньою і лівою нижньою координатами або координатами справа зверху і зліва зверху.

Аналогічним чином отримується висота нового зображення у рядках 34-36, де висота – це максимальна відстань між координатами зверху з права і внизу з права, або між координатами зліва внизу та зліва зверху (рис.2.9).

```
31      width_a = np.sqrt(((br[0] - bl[0]) ** 2) + ((br[1] - bl[1]) ** 2))
32      width_b = np.sqrt(((tr[0] - tl[0]) ** 2) + ((tr[1] - tl[1]) ** 2))
33      max_width = max(int(width_a), int(width_b))
34      height_a = np.sqrt(((tr[0] - br[0]) ** 2) + ((tr[1] - br[1]) ** 2))
35      height_b = np.sqrt(((tl[0] - bl[0]) ** 2) + ((tl[1] - bl[1]) ** 2))
36      max_height = max(int(height_a), int(height_b))
```

Рисунок 2.9 – Отримання ширини та висоти

Ось наступив момент де важливу роль відіграє послідовність точок. У рядку 37 отримуються 4 точки, які представляють собою вид згори. Перший запис (0, 0), - це верхній лівий кут, другий запис (max_width – 1, 0) – що відповідає верхньому правому куту, наступним йде правий нижній кут – (max_width – 1, max_height – 1), і нарешті (0, max_height – 1) який знаходиться в нижньому лівому куту. Висновок з цього наступний, що ці точки задані в послідовному порядку – і саме це дозволить отримати зображення згори до низу.

І щоб отримати такий ефект використовується функція getPerspective-Tranform() (рядок 38). Ця функція приймає два аргументи, прямокутник який представляє собою список чотирьох точок в початковому зображенні, и dst, що є списком перетворених точок. І ця функція повертає матрицю яка є фактичною матрицею перетворення.

Далі ця матриця використовується для функції warpPerspective в яку передається сама матриця перетворення, разом із шириною та висотою нового вихідного зображення.

На виході цієї функції буде перетворене зображення, яке і буде являтися видом згори до низу, яке і повертається на наступному кроці рис.2.10.

```
37      dst = np.array([[0, 0], [max_width - 1, 0], [max_width - 1, max_height - 1], [0, max_height - 1]], dtype="float32")
38      m = cv2.getPerspectiveTransform(rect, dst)
39      warped = cv2.warpPerspective(img, m, (max_width, max_height))
40      return warped
```

Рисунок 2.10 – Перетворення зображення

Щоб отримати відчуття чорно-білого зображення, береться змінене зображення і перетворюється у відтінки сірого та застосовується адаптивне порогове значення на рядках 71-73 (рис.2.11).

```
71      warped = cv2.cvtColor(warped, cv2.COLOR_BGR2GRAY)
72      t = threshold_local(warped, 11, offset=10, method="gaussian")
73      warped = (warped > t).astype("uint8") * 255
```

Рисунок 2.11 – Зміна кольору зображення

І нарешті у рядках 75-78 виконуються потрібні перетворення для збереження зображення до бази даних (рис.2.12). Спочатку за допомогою бібліотеки OpenCV кодується зображення до розширення jpg. Наступним кроком за допомогою стандартного класу Django – ContentFile отримується байтове відображення сканованого зображення. І нарешті отриманий в самому початку останній запис в базі даних зберігається з додаванням у нього сканованого зображення з допомогою метода save(), у який передається нове ім’я зображення, для усунення можливих конфліктів у подальшому, а також саме зображення.

```
75      ret, buf = cv2.imencode('.jpg', warped)
76      content = ContentFile(buf.tobytes())
77
78      image_obj.scanned_photo.save(f'scanned_{name}.jpg', content)
```

Рисунок 2.12 – Збереження зображення

Результат роботи цієї частини показано на рис.2.13.

Рисунок 2.13 – Результат роботи перетворень перспективи для виду згори

3 ТЕХНІЧНА ЧАСТИНА

3.1 Алгоритм настройки проекту

Для створення web-сервісу буде використовуватися фреймворк – Django, версії 3.0. Як вказувалось раніше Django – це високоякісний web-фреймворк Python, для швидкої розробки сайтів. Також Django містить в собі вбудований сервер для розробки. Сервер розробки Django (також званий «runserver», по імені команди, яка його запускає) - це вбудований легкий веб сервер, який користувач можете використовувати в процесі розробки свого сайту. Він включений в Django для того, щоб користувачі могли швидко приступити до розробки власного сайту без витрачання часу на конфігурацію свого основного web-сервера (тобто, Apache, або Nginx) завчасно. Цей сервер розробки відстежує зміни в коді розробника і автоматично перезавантажує його, допомагаючи бачити внесені користувачем зміни без перезавантаження веб сервера.

Спочатку знадобиться створити віртуальне середовище. У корені своєму, головне завдання віртуального середовища Python - створення ізольованого середовища для проектів Python. Це означає що – кожен проект може мати свої власні залежності, незалежно від того, які залежності у іншого проекту. Спочатку встановлюється сам пакет для створення віртуального середовища, а саме virtualenv, відкривши термінал або консоль вводиться команда *pip install virtualenv*.

Наступним кроком потрібно створите віртуальне середовище. Перейти в терміналі в директорію в якій буде знаходитися проект, та вписати команду *virtualenv venv*, де virtualenv – команда для створення середовища, venv – ім’я директорії в якій будуть розташовуватися усі основні файли віртуального середовища. Тепер щоб запустити середовище потрібно активувати скрипт

ввівши в термінал env\Scripts\activate для Windows, або source env/bin/activate для дистрибутивів Linux.

Тепер усі встановлюванні модулі будуть доступні лише для цього віртуального середовища. Встановлюється потрібна версію Django ввівши в термінал команду *pip install django==3.0.** , після цього встановиться сам Django, а також інші необхідні для роботи цього фреймворка модулі.

Наступним кроком потрібно стартувати сам проект для того щоб усі файли проекта були в одному місці потрібно створити нову директорію *src*, тепер в термінали треба перейти в цю директорію, та ввести команду *django-admin startproject config .*, де config це головна папка проекту в якій будуть знаходитися основні файли конфігурації, а саме *settings.py* та *urls.py*. Далі потрібно застосувати всі початкові міграції для створення бази даних, локально буде використовуватися sqlite, виконавши команду *python manage.py migrate*. Також необхідно виконати команду *python manage.py createsuperuser*, для того щоб мати доступ до адмін панелі Django.

І нарешті останнім кроком підготовки буде створення нового додатку, виконавши команду *python manage.py startapp scan*, де scan – це назва нового додатку.

Тепер можна переходити до конфігурації проекту, відкривши файл settings.py. Додати в змінну INSTALLED_APPS, нещодавно створений додаток, результат надано на рис. 3.1.

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',

    'scan.apps.ScanConfig',
]
```

Рисунок 3.1 – Змінна INSTALLED_APPS

Також потрібно в змінну TEMPLATES додати ключ DIRS для того щоб в подальшому Django знав де йому шукати темплейти для сайту. Змінена змінна представлена на рис.3.2.

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'templates')],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

Рисунок 3.2 – Змінна TEMPLATES

І нарешті останній крок для конфігурації додати змінні STATIC_ROOT та STATICFILES_DIRS щоб Django розумів де йому шукати статичні файли для сайту. Змінні показані на рис. 3.3.

```
123
124     STATIC_ROOT = os.path.join(BASE_DIR, 'staticfiles')
125
126     STATICFILES_DIRS = [
127         os.path.join(BASE_DIR, "static"),
128     ]
```

Рисунок 3.3 – Змінні STATIC_ROOT та STATICFILES_DIRS

Тепер потрібно створити модель для подальшого доступ через неї до бази даних. Відкривши файл *scan/models.py*, та прописавши модель ScanFile (рис.3.4). Модель ScanFile має наступні поля photo – відповідає за початкове зображення, і визначає директорію для збереження, поле scanned_photo – відповідає за скановане зображення, та місце його збереження, і нарешті поле pdf_file – відповідає для конвертоване скановане зображення в pdf-файл, та

```
class ScanFile(models.Model):
    photo = models.ImageField(upload_to='static/image')
    scanned_photo = models.ImageField(upload_to='static/scanned_image', default=None, null=True)
    pdf_file = models.FileField(upload_to='static/pdf_files', default=None, null=True)

    class Meta:
        verbose_name = 'Файл'
        verbose_name_plural = 'Файли'
```

Рисунок 3.4 – Модель ScanFile

місце його збереження.

Щоб дана модель була створена в консоль потрібно вписати наступну команду *python manage.py makemigrations*, після чого виконати *python manage.py migrate*.

Тепер потрібно створити класи відображення. Для цього проекту потрібно створити 3 класи, а саме HomeView (рис.3.5), ScannedView(рис.3.6), ScanDetailView(рис.3.7).

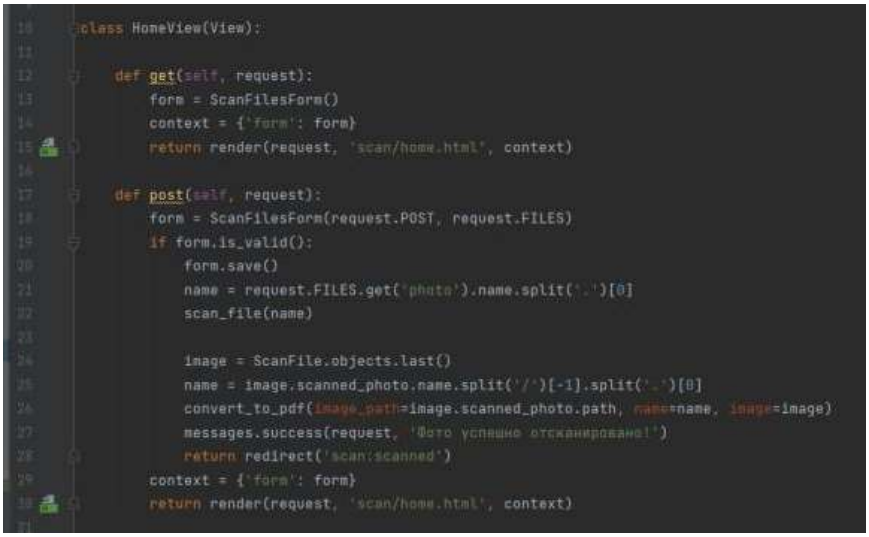


Рисунок 3.5 – HomeView

Функція get відповідає за показ домашньої сторінки при запиті GET, вона використовує форму ScanFilesForm яка представлена в *scan/forms.py*(рис.3.8), а також формує контекст який в подальшому буде використовуватися в html файлі, в даному випадку *scan/home.html*.

Функція post відповідає за запит POST. А саме вона перевіряє надіслану форму на валідність, зберігає його в базу даних, сканує отримане фото і також зберігає його, і нарешті конвертує отримане скановане зображення в pdf-файл. І якщо усі попередні кроки були виконані то виконує редірект на ScannedView. В іншому випадку повертає на головну сторінку, а саме *scan/home.html*.

Рисунок 3.6 – ScannedView

Метод get який виконується при запиті GET, отримує усі записи в базі даних та повертає їх в зворотному напрямку, тобто спочатку йдуть останні додані записи. Та повертає *scan/scanned.html*.

```
class ScanDetailView(View):

    def get(self, request, pk):
        image = ScanFile.objects.get(id=pk)
        context = {'photo': image}
        return render(request, 'scan/detail.html', context=context)
```

Рисунок 3.7 – ScanDetailView

Функція get визивається при запиті GET, та повертає *scan/detail.html*. Цей клас відображує один конкретний файл в базі даних.

```
class ScanFilesForm(forms.ModelForm):

    photo = forms.ImageField(widget=forms.ClearableFileInput(attrs={
        'class': 'form-control'
    }))

    class Meta:
        model = ScanFile
        fields = ('photo', )
```

Рисунок 3.8 – ScanFilesForm

Після того як усі view класи готові потрібно додати роутінг до проекту, на данний момент існує один файл в додатку config/urls.py (рис.3.9). Потрібно створити ще один такий же файл для додатку scan(рис.3.10).

```python
from django.contrib import admin
from django.urls import path, include
from django.conf import settings
from django.conf.urls.static import static
from django.views.generic import RedirectView

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', RedirectView.as_view(url='files/')),
    path('files/', include('scan.urls', 'scan')),
] + static(settings.STATIC_URL, document_root=settings.STATIC_ROOT)
```

Рисунок 3.9 – Config/urls.py

Файлу config/urls.py потрібно модифікувати, додавши path(‘’), який при запиті на цю сторінку буде робити редірект на головну сторінку сайту, а також додати path(‘files/’), який відповідає за роутінг додатку scan.

```python
from django.urls import path
from django.conf import settings
from django.conf.urls.static import static

from .views import ScannedView, HomeView, ScanDetailView

app_name = 'scan'

urlpatterns = [
    path('', HomeView.as_view(), name='home'),
    path('scanned/', ScannedView.as_view(), name='scanned'),
    path('detail/<int:pk>', ScanDetailView.as_view(), name='detail'),
] + static(settings.STATIC_URL, document_root=settings.STATIC_ROOT)

```

Рисунок 3.10 – scan/urls.py

Файл scan/urls.py містить наступні роути, path(‘’) – відповідає за головну сторінку сайту, і до нього підв’язанний клас HomeView, path(‘scanned’) – відповідає за сторінку з списком сканованих фотографій до нього підв’язанний клас ScannedView, path(‘detail/<int>:pk’) – відповідає за відображення конкретної відсканованої фотографії, а також до нього підв’язанний клас ScanDetailView.

На даний момент структура усього проекту виглядає наступним чином рис.3.11.

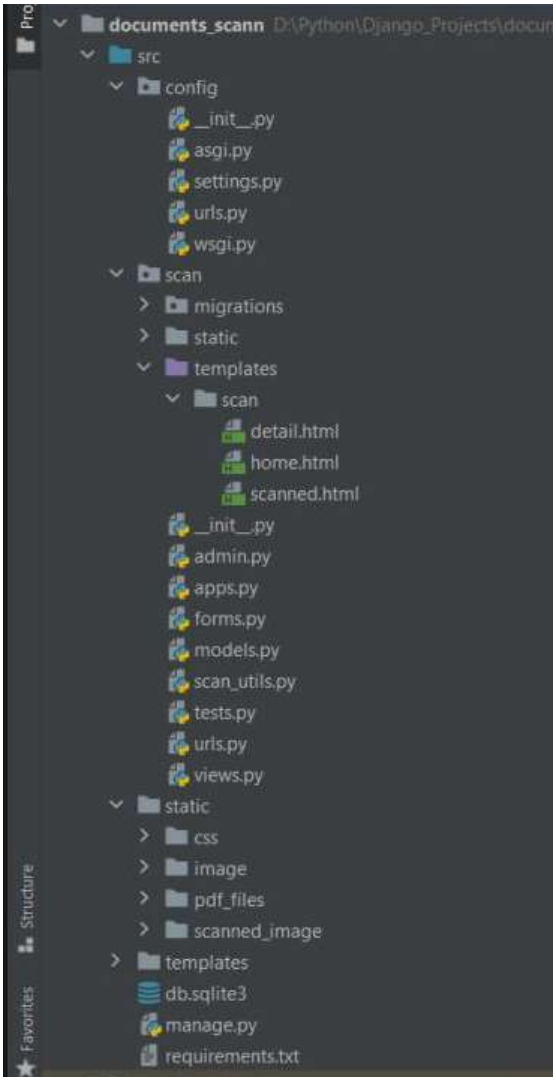


Рисунок 3.11 – Структура проекту

Тепер щоб перевірити чи все працює потрібно запустити сервер розробки. Для цього потрібно виконати команду *python manage.py runserver* і Django запустить локальний сервер, до якого можна буде отримати доступ за посиланням 127.0.0.1:8000, по стандарту Django використовує 8000 порт.

3.2 Розгортання сайту на AWS EC2

3.2.1 Розгортання з допомогою Apache

Кінцевою метою даного етапу є підключення інтерфейсу шлюзу веб-сервера Django (WSGI) до сервера Apache для виконання коду Python у фреймфорку Django у виробничому середовищі з використанням AWS EC2 як хмарної служби. Apache використовується як веб-сервер для виробництва має свої плюси, такі як безпека та ефективна обробка трафіку, а також підтримка роботи сервера.

Для розгортання треба виконати наступні кроки:

- запуск екземпляра EC2;

- підключення екземпляра EC2 на локальній машині;

- налаштування екземпляру EC2 (встановлення Apache2, pip, модуль apache для wsgi та віртуального середовища);

- скопіювати проект;

- створити віртуальне середовище;

- встановити необхідні модулі;

- налаштувати Apache на запити на проект django.

Інтерфейс шлюзу веб-сервера (WSGI) - це лише специфікація інтерфейсу, з якою взаємодіють сервер та програма.

Сервер WSGI (мається на увазі, сумісний з WSGI) лише отримує запит від клієнта, передає його додатку, а потім надсилає клієнту відповідь, повернуту додатком. Він використовується лише для мови програмування на Python.

Основним завданням веб-сервера HTTP є отримання запиту та відповідь на обслуговування файлів HTTP. Apache - приклад веб-сервера HTTP.

Проблема тут полягає в тому, що Apache по своїй суті призначений для обслуговування лише запитів HTTP. Тут з’являються різні модулі для Apache. Ці модулі допомагають Apache виконувати різні функції.

Один із таких модулів – це mod-wsgi він буде використаний, щоб допомогти Apache взаємодіяти з інтерфейсом WSGI у додатку Django.

Запуск екземпляру EC2. Для цього потрібно ввійти в свій акаунт на сайті aws.awazon.com, після завершення процесу входу в систему потрібно перейти на сторінку послуг і вибрати EC2 у підменю «Обчислення». На інформаційній панелі EC2 портал Launch Instance встановити новий екземпляр та обрати потрібний екземпляр машини Amazon (AMI). Налаштування типів екземплярів та груп безпеки. Обов'язково треба додати принаймні IP локальної машини для доступу до загальнодоступного DNS машини EC2. Пара ключів створюється після запуску екземпляра. Цей файл пари ключів слід завантажити в локальну систему, оскільки він допомагає підключатися до EC2 за допомогою SSH.

Підключення до екземпляру EC2. Спочатку треба скопіювати команду SSH, яка буде використана для доступу до екземпляра EC2, за допомогою терміналу. На локальній машині треба перейти до каталогу в терміналі, куди був завантажений файл пари ключів і який зараз там знаходиться. У термінал вставляється скопійована раніше команду SSH і виконується, якщо буде запропоновано певний дозвіл на доступ (що в основному полягає в додаванні локальної машини як надійного хоста) натиснути Так.

Налаштування екземпляра EC2. Екземпляр EC2 потрібно налаштувати для досягнення мети. Спочатку потрібно ввести наступні команді *sudo apt-get update* і *sudo apt-get install python3-pip apache2 libapache2-mod-wsgi-py3*. Перша команда – оновить менеджер пакунків apt в Ubuntu. Друга команда встановлює apache2, pip3 та mod-wsgi (модуль, що використовується в Apache, для запуску сервера Django і буде автоматично ввімкнений після встановлення). Встановлюється virtualenv, використовуючи наступну команду sudo pip3 install virtualenv. Створюється каталог під назвою django, використовуючи команду: *mkdir django*. Тепер треба перейти в створену директорію командою *cd django*.

В цій директорії буде налаштоване віртуальне середовище, а також скопійовано проект Django.

Конфігурація сервера Apache. Потрібно відредагувати файл, де визначається віртуальний хост. Щоб це зробити потрібно ввести таку команду *sudo vim /etc/apache2/sites-available/000-default.conf*, щоб відкрити файл у редакторі vim. Натиснути I, щоб розпочати редагування файлу, та замінити файл наступною конфігурацією рис.3.12.

```
<VirtualHost *:80>

ServerAdmin webmaster@example.com

DocumentRoot /home/ubuntu/django/myproject

ErrorLog ${APACHE_LOG_DIR}/error.log

CustomLog ${APACHE_LOG_DIR}/access.log combined

Alias /static /home/ubuntu/django/myproject/static

<Directory /home/ubuntu/django/myproject/static>

Require all granted

</Directory>

<Directory /home/ubuntu/django/myproject/myproject>

<Files wsgi.py>

Require all granted

</Files>

</Directory>

WSGIDaemonProcess myproject python-path=/home/ubuntu/django/myproject
python-home=/home/ubuntu/django/myprojectenv

WSGIProcessGroup myproject

WSGIScriptAlias / /home/ubuntu/django/myproject/myproject/wsgi.py

</VirtualHost>
```

Рисунок 3.12 – Конфігурація Apache

Перший рядок визначає, що Apache буде слухати порт 80 (HTTP за замовчуванням) і обслуговувати вказані файли. Корінь документа вказує розташування файлів програми. Теги каталогів використовуються для надання серверу Apache дозволу на доступ до відповідного каталогу та його файлів. Для

роботи сервера Django потрібні налаштування WSGI. Потрібно переконатися, що шляхи правильні та відповідно до імен каталогів та файлів у проекті.

Віртуальні хости Apache використовуються для запуску декількох веб-сайтів (доменів) з використанням однієї IP-адреси. Іншими словами, кілька веб-сайтів (доменів), але один сервер. Потрібно перейти до каталогу проекту django за допомогою команди *cd /home/ubuntu/django/myproject*. Наступні команди, потрібні щоб надати необхідні дозволи: *chmod 664 db.sqlite3*, *sudo chown: www-data db.sqlite3*, *sudo chown: www-data ~/myproject*. Після всього цього потрібно перезапустити сервер apache2, використовуючи таку команду *sudo service apache2 restart*.

3.2.2 Розгортання з допомогою nginx та gunicorn

Nginx - це веб-сервер з відкритим вихідним кодом, який з часу свого успіху як веб-сервер тепер також використовується як зворотний проксі-сервер, кеш-пам'ять HTTP та балансування навантаження. В даному випадку буде використовуватися Nginx для серверування веб-сторінок за потреби.

Gunicorn - це серверна реалізація шлюзового інтерфейсу веб-сервера (WSGI), яка зазвичай використовується для запуску веб-додатків Python.

Весь принцип розгортання з допомогою nginx такий же як із допомогою Apache, до моменту конфігурації. Для початку після створення екземпляру EC2, та копіювання до нього проекту потрібно встановити nginx за допомогою команди *sudo apt install nginx*. Після цього треба відредагувати файл конфігурації виконавши команду *sudo vim /etc/nginx/sites-enabled/default*. Отриманий файл буде виглядати наступним чином рис.3.13.

```
# Add index.php to the list if you are using PHP
index index.html index.htm index.nginx-debian.html;

server_name _;

location / {
        # First attempt to serve request as file, then
        # as directory, then fall back to displaying a 404.
        proxy_pass http://0.0.0.0:9000;
}

location /static/ {

    alias  /home/ubuntu/Project/ProjectName/static/;
}
```

Рисунок 3.13 – Конфігурація nginx

Треба додати proxy_pass http://0.0.0.0:9000 і надати шлях до статичної папки, додавши шлях всередину location/static/, як зазначено вище. Переконатися, що зібрані всі статичні файли до загальної папки, запустивши команду *manage.py collectstatic*. І запустити сервер nginx за допомогою команд: *sudo service nginx start*, *sudo service nginx stop*, *sudo service nginx restart*.

Останнім кроком буде установка gunicorn *pip install gunicorn*. Треба переконатися, що зараз користувач знаходиться в папці проекту, наприклад: /home/ubuntu/Project, і виконати наступну команду, щоб запустити gunicorn *gunicorn config.wsgi: application- -bind 0.0.0.0:9000*

Тепер, коли встановлено та налаштовано nginx та gunicorn, до сайту можна отримати доступ через DNS екземпляру EC2.

3.3 Розгортання проекту на сервісі Heroku

Heroku - це хмарна платформа, яка дозволяє компаніям створювати, доставляти, відстежувати і масштабувати додатки. Платформа для даних, а також додатків - забезпечує безпечну, масштабовану базу даних як послугу з безліччю інструментів розробників, таких як послідовники бази даних, форкінгу, кліки даних та автоматизовані перевірки стану.

Отже для розгортання на Heroku потрібно зробити початкову підготовку проекту:

- конфігурація settings.py у production.py;

- створення файлу .gitignore;
- створення файлу Procfile;
- перенесення проекту на GitHub.

Оскільки проект буде використовувати GitHub, потрібно створити файли конфігурації для локального запуску, та для продакшену. У директорії config потрібно створити python package з назвою settings, і перенести туди файл settings.py та перейменувати його в production.py, а також створити копію цього файлу та назвати її local_settings.py. Файл __init__.py має виглядати наступним чином рис.3.14.

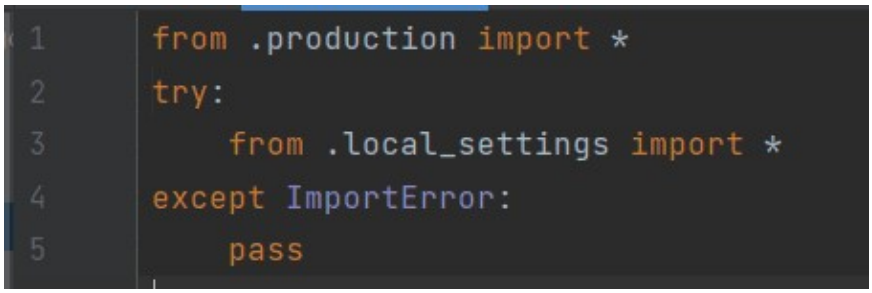


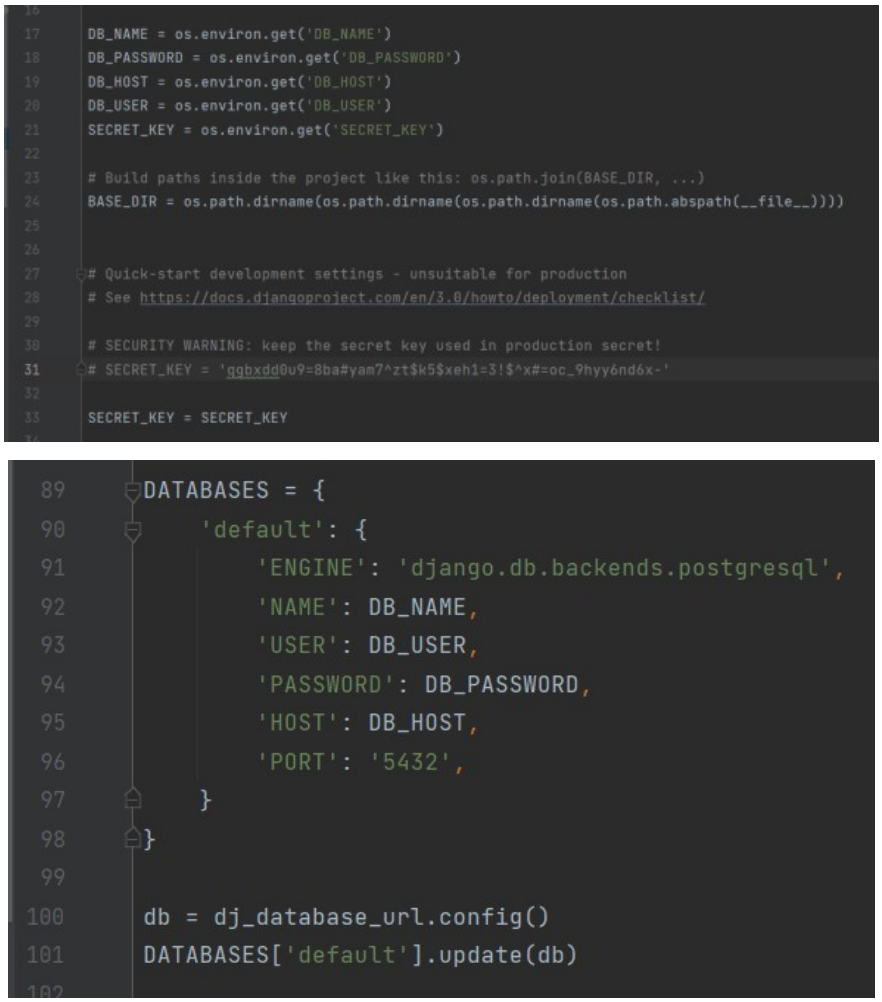
Рисунок 3.14 – Файл __init__.py

Суть цього файлу наступна – оскільки Django робить пошук не по файлам, а по простору імен то Django знайде python package з назвою settings та прийме його за конфігураційний і перейде до файлу init, де спочатку імпортується конфігурація для запуску на сервері, а вже потім імпортується конфігурація для локального запуску. Якщо локальний імпорт пройшов вдало то він перезапише усю конфігурацію продакшену, і навпаки якщо не вийде імпортувати файл local_settings (а саме так і буде оскільки цього файлу не буде на сервері і виникне помилка ImportError яке буде перехвачено конструкцією try except) залишаться усі настройки продакшену.

Тепер потрібно змінити конфігурацію файлу production.py. Усі зміни показані на рис.3.15. Основні зміни це додані змінні віртуального оточення, а саме DB_NAME, DB_PASSWORD, DB_HOST, DB_USER, SECRET_KEY ці змінні знадобляться на сервісі Heroku для конфігурації бази даних PostgreSQL.

Які будуть братися з змінних оточення. А також змінена конфігурація бази даних, а саме змінна DATABASES, саме тут і використовуються змінні оточення які отримувались раніше, також замість драйверу sqlite використовується драйвер для PostgreSQL, і стандартний порт для цієї бази даних 5432.

І за допомогою модулю dj_database_url оновити стандарту конфігурацію для сервісу Heroku.



```
DB_NAME = os.environ.get('DB_NAME')
DB_PASSWORD = os.environ.get('DB_PASSWORD')
DB_HOST = os.environ.get('DB_HOST')
DB_USER = os.environ.get('DB_USER')
SECRET_KEY = os.environ.get('SECRET_KEY')

# Build paths inside the project like this: os.path.join(BASE_DIR, ...)
BASE_DIR = os.path.dirname(os.path.dirname(os.path.dirname(os.path.abspath(__file__))))


# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/3.0/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
# SECRET_KEY = 'ggbxdd0u9=8ba#yam7^zt$k5$xeh1=3!$^x#=oc_9hyy6nd6x-'

SECRET_KEY = SECRET_KEY
```

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': DB_NAME,
        'USER': DB_USER,
        'PASSWORD': DB_PASSWORD,
        'HOST': DB_HOST,
        'PORT': '5432',
    }
}

db = dj_database_url.config()
DATABASES['default'].update(db)
```

Рисунок 3.15 – Змінена конфігурація production.py

Також потрібно створити файл .gitignore, а також створити Procfile та додати до нього наступну строку: web: gunicorn config.wsgi --log-file . Тепер можна створювати репозиторій на GitHub, та за допомогою git переносити туди проект. І нарешті можна переходити до створення проекту на Heroku.

Спочатку треба обрати назву проекту (рис.3.16), на цьому сервісі усі імена зберігаються у одному оточенні імен тому потрібно вигадати дуже унікальну назву, або потратити деякий час на створення вільного імені, а також потрібно обрати регіон, на вибір дається європейський та американський регіони, для даного проекту обирається Європа оскільки у цьому випадку це більш підходящий варіант.

Наступним кроком потрібно підключити створений GitHub репозиторій та почати розгортання, натиснувши на кнопку Deploy.

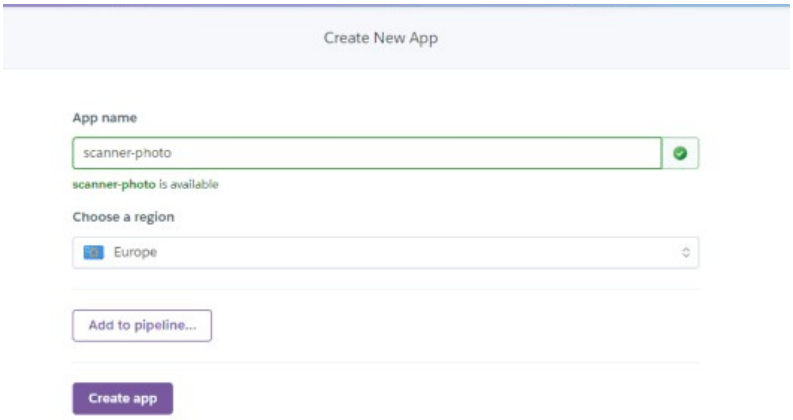


Рисунок 3.16 – Створення проекту на Heroku

Результат розгортання наданий на рис.3.17. Все пройшло успішно, і тепер проект вже доступний, але залишилось сконфігурувати базу даних.

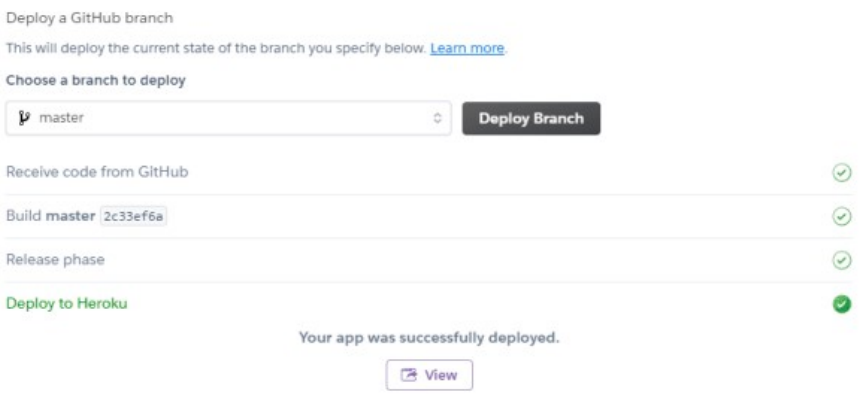


Рисунок 3.17 – Результат розгортання

Оскільки раніше в файлі production.py було додано конфігурацію бази даних при розгортанні на Heroku сервіс це розпізнав, та вже додав базу даних до проекту, залишилося тільки перенести данні для конфігурації бази даних до середовища змінних. Та створити ключі які були вказані раніше. Результат наданий на рис.3.18.

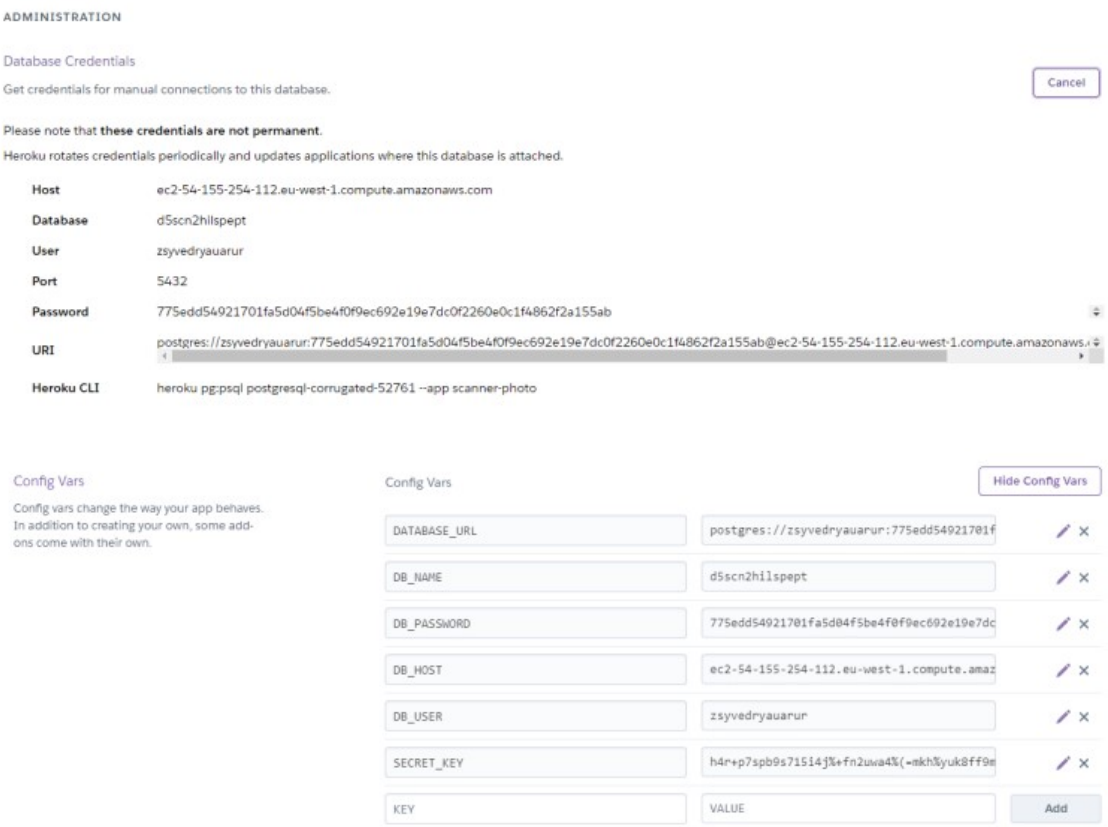


Рисунок 3.18 – Створення середовище змінних

Останнім пунктом перед повною готовністю проекту буде застосування міграцій для створення стандартних моделей Django, а також самостійно створених моделей. Для цього потрібно відкрити вбудовану для Heroku консоль, та застосувати команду *python manage.py migrate*. Також можна виконати команду *python manage.py createsuperuser* для створення користувача с доступом до адміністративної панелі сайту. Результат роботи наданий на рис.3.19.

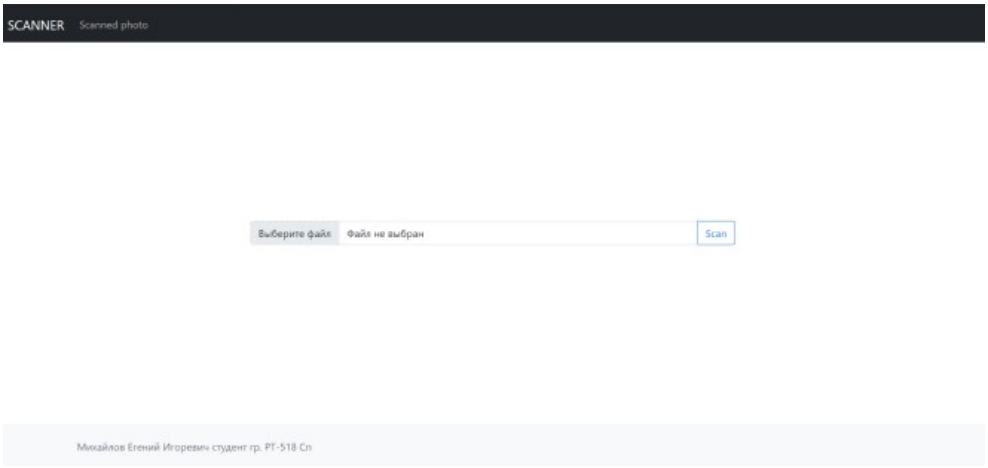


Рисунок 3.19 – Розгорнутий сайт

ВИСНОВКИ

Під час виконання дипломної роботи було виконано дослідження предметної області, розглянуто та визначено основні поняття, які пов’язані з обраною предметною областю. Також були розглянуті та проаналізовані існуючі аналоги. Було визначено і сформульовано завдання до роботи.

Метою роботи є створення програмного забезпечеqння, та web-сервісу для відтворення pdf-файлу з фотографії шляхом розпізнавання образів на фотографії.

Для виконання поставленої мети було використано наступні технології:

– мова програмування Python;

– бібліотека OpenCV;

– фреймворк Django;

– бази даних PostgreSQL та Sqlite;

– сервіс для розгортання Heroku.

Результатом дипломного проекту є робочий, готовий web-сервіс для відтворення pdf-файлу. Також була проведена теоретична робота над базовими можливостями бібліотеки OpenCv, розглянуті основні використовувані технології. Розроблений web-сервіс як видно з прикладів являється робочим, а також відповідає всім пунктам завдання.

ПЕРЕЛІК ДЖЕРЕЛ

1. Python documentation [Електронний ресурс]. – Режим доступу: https://docs.python.org/3/tutorial/index.html

2. Django documentation [Електронний ресурс]. – Режим доступу: https://docs.djangoproject.com/en/3.0/

3. OpenCV documentation [Електронний ресурс]. – Режим доступу: https://docs.opencv.org/master/d3/dc0/group__imgproc__shape.htm

4. PostgreSQL documentation [Електронний ресурс]. – Режим доступу: https://www.postgresql.org/docs/

5. Amazon Web Services documentation [Електронний ресурс]. – Режим доступу: https://docs.aws.amazon.com/ec2/?id=docs_gateway

6. TProger [Електронний ресурс]. – Режим доступу: https://tproger.ru/translations/opencv-python-guide/

7. Habr [Електронний ресурс]. – Режим доступу: https://habr.com/ru/post/519454/

8.

9. Deploying Django Apps [Електронний ресурс]. – Режим доступу: https://medium.com/analytics-vidhya/deploying-django-apps

10. Deploying Django Apps with Apache [Електронний ресурс]. – Режим доступу: https://medium.com/saarthi-ai/ec2apachedjango

11. 4 Point OpenCV [Електронний ресурс]. – Режим доступу: https://www.pyimagesearch.com/2014/08/25/4-point-opencv-getperspective-transform-example/

12. Deploying Django Apps on Heroku [Електронний ресурс]. – Режим доступу: https://www.codementor.io/@jamesezechukwu/how-to-deploy-django-app-on-heroku-dtsee04d4