Báo cáo Kế hoạch Chi tiết: Ứng dụng OCR Nhận dạng và Phân loại Nhãn Bưu kiện

Dự án: Tự động hóa Phân loại Bưu kiện bằng OCR và Streamlit
Mục tiêu: Xây dựng một kế hoạch chi tiết, toàn diện và tối ưu hóa công sức để phát triển ứng dụng Python/Streamlit nhận dạng địa chỉ trên nhãn bưu kiện và phân loại hướng giao hàng (NỘI Ô / NGOẠI Ô).
Đối tượng: Kỹ sư phát triển có kiến thức Python cơ bản, nhưng chưa có kinh nghiệm chuyên sâu về Xử lý Ảnh (CV) hoặc Nhận dạng Ký tự (OCR).

Phần 1: Lập kế hoạch và Quyết định Công nghệ Cốt lõi

Để xây dựng một ứng dụng "hoàn hảo" với công sức tối thiểu, việc lựa chọn đúng công nghệ và kiến trúc ngay từ đầu là yếu tố quyết định. Kế hoạch này ưu tiên các công cụ mạnh mẽ, dễ cài đặt và dễ tích hợp.

Kiến trúc Hệ thống End-to-End

Hệ thống sẽ được xây dựng theo kiến trúc pipeline mô-đun hóa, đảm bảo mỗi thành phần thực hiện một nhiệm vụ cụ thể và có thể được phát triển/gỡ lỗi độc lập.

1. **Giao diện Người dùng (UI - app.py):** Người dùng tải ảnh lên qua giao diện web Streamlit.
2. **Tiền xử lý Ảnh (preprocessing.py):** Ảnh thô được tự động làm sạch, xoay thẳng và tối ưu hóa bằng OpenCV.
3. **Nhận dạng Ký tự (OCR - ocr.py):** Ảnh đã xử lý được đưa vào engine EasyOCR để trích xuất toàn bộ văn bản thô.
4. **Phân tích Logic (logic.py):** Văn bản thô được phân tích bởi thư viện vietnamadminunits để trích xuất các thành phần địa chỉ có cấu trúc (Tỉnh, Quận, Phường).
5. **Phân loại & Trả về (UI - app.py):** Dựa trên địa chỉ có cấu trúc, hệ thống phân loại "NỘI

Ô" / "NGOẠI Ô" và trả kết quả về giao diện Streamlit.

Việc tách mã thành các tệp riêng biệt (thay vì viết tất cả vào app.py) là một thông lệ kỹ thuật then chốt, giúp việc bảo trì, mở rộng và gỡ lỗi trở nên đơn giản hơn đáng kể.

Quyết định về Stack Công nghệ

Các quyết định về công nghệ được đưa ra dựa trên mục tiêu "tối thiểu hóa công sức" và "tối đa hóa hiệu quả".

Bảng 1: Phân tích và Lựa chọn Stack Công nghệ

Thành phần	Công cụ được Đề xuất	Lý do lựa chọn (Dựa trên Nghiên cứu)	Lựa chọn thay thế (Và lý do không chọn)
Giao diện (UI)	**Streamlit**	Yêu cầu trực tiếp của dự án. Cho phép xây dựng UI nhanh chóng, không cần kiến thức HTML/CSS. Tích hợp hoàn hảo với các thư viện xử lý dữ liệu Python.	**Flask/Django:** Quá phức tạp cho dự án này. Yêu cầu nhiều công sức để thiết kế frontend và xử lý API, không phù hợp với mục tiêu "tối thiểu hóa công sức".
Tiền xử lý Ảnh	**OpenCV (opencv-python)**	Tiêu chuẩn ngành về xử lý ảnh.[4] Cung cấp đầy đủ các công cụ cần thiết: tự động điều chỉnh độ nghiêng (deskew) [5], nhị phân hóa thích ứng (adaptive thresholding) và khử nhiễu.[6]	**Pillow:** Tốt cho các tác vụ cơ bản (mở, lưu, thay đổi kích thước) [6], nhưng thiếu các thuật toán xử lý ảnh nâng cao (như deskew tự động) mà OpenCV cung cấp.

Nhận dạng (OCR)	**EasyOCR**	**Tối ưu nhất cho người mới:** Cài đặt đơn giản qua pip.[3] Hỗ trợ tiếng Việt tốt.[7] Nhanh hơn Tesseract.[8] Quan trọng nhất: Hỗ trợ gpu=False [7] cho phép chạy trên CPU, dễ dàng triển khai ở mọi nơi.	**Tesseract:** Chậm hơn.[8] Yêu cầu cài đặt Tesseract EXE bên ngoài Python [9], gây phức tạp lớn khi triển khai (deploy). **PaddleOCR:** Hiệu suất CPU kém.[10] Cài đặt phức tạp do phụ thuộc vào framework PaddlePaddle.[10] **LLMs (ví dụ: Gemini):** Quá phức tạp, tốn kém và không cần thiết cho nhiệm vụ trích xuất văn bản có cấu trúc này.[11]
Phân tích Địa chỉ	**vietnamadminunits**	Được thiết kế chuyên biệt để phân tích cú pháp địa chỉ Việt Nam.[12] API trả về các đối tượng có cấu trúc rõ ràng (.province, .district, .ward).[12] Hỗ trợ các thay đổi hành chính.[12]	**NER (ví dụ: underthesea):** Không đủ khả năng. Mô hình NER chung chỉ gắn thẻ "VỊ TRÍ" (LOC).[13] Nó không thể phân biệt "Quận 1" (Quận) và "TPHCM" (Tỉnh), điều này làm cho logic nghiệp vụ phân loại NỘI Ô/NGOẠI Ô thất bại.

Cấu trúc Dự án và Cài đặt Môi trường

Để bắt đầu, dự án sẽ được tổ chức theo cấu trúc thư mục rõ ràng và một môi trường ảo Python chuyên dụng.

1. **Thiết lập Môi trường Ảo (Virtual Environment):**
   Bash
   python -m venv venv
   source venv/bin/activate # (Trên Linux/macOS)

.\venv\Scripts\activate # (Trên Windows)
```

2. Tệp requirements.txt (Danh sách thư viện cần cài đặt):
   Tệp này chứa tất cả các thư viện cần thiết. Chạy pip install -r requirements.txt.
   streamlit
   opencv-python-headless
   easyocr
   numpy
   vietnamadminunits
   Pillow

   *Ghi chú: opencv-python-headless được sử dụng thay vì opencv-python đầy đủ. Phiên bản headless không bao gồm các thành phần UI của OpenCV, giúp nó nhẹ hơn và dễ dàng hơn cho việc triển khai máy chủ.*
3. **Cấu trúc Thư mục Dự án:**
   Parcel_OCR_App/
   ├──.streamlit/
   │  └── config.toml       # (Tệp cấu hình Streamlit, tùy chọn)
   ├── venv/                # (Thư mục môi trường ảo)
   ├── app.py               # (File UI Streamlit chính)
   ├── logic.py             # (File logic phân tích địa chỉ & phân loại)
   ├── ocr.py               # (File chứa hàm gọi EasyOCR)
   ├── preprocessing.py     # (File chứa các hàm xử lý ảnh OpenCV)
   ├── requirements.txt     # (Tệp thư viện)
   └── test_images/
      └── label_sample.jpg  # (Ảnh nhãn mẫu để thử nghiệm)

---

# Phần 2: Giai đoạn 1 - Xây dựng Module Tiền xử lý Ảnh (preprocessing.py)
```

Đây là giai đoạn quan trọng nhất để đảm bảo độ chính xác của OCR. Ảnh chụp nhãn bưu kiện trong thực tế thường bị nghiêng, mờ, và chụp trong điều kiện ánh sáng không đồng đều.[4] Mục tiêu của mô-đun này là chuẩn hóa mọi ảnh đầu vào trước khi đưa cho OCR.

Pipeline Tiền xử lý Tự động

Chúng ta sẽ xây dựng một pipeline gồm các hàm của OpenCV.[6]

1. **Tải ảnh:** Chuyển đổi đối tượng UploadedFile của Streamlit [16] sang định dạng mảng NumPy mà OpenCV có thể đọc.
2. **Chuyển đổi sang Ảnh Xám (Grayscale):** Loại bỏ thông tin màu sắc không cần thiết, đây là bước bắt buộc cho nhiều thuật toán xử lý ảnh tiếp theo.[6]
3. **Tự động Điều chỉnh Độ nghiêng (Automatic Deskewing):** Đây là "vũ khí bí mật" của pipeline. Ảnh chụp bằng điện thoại gần như luôn bị nghiêng. Việc bỏ qua bước này có thể làm giảm đáng kể độ chính xác của OCR. Chúng ta sẽ triển khai thuật toán dựa trên việc tìm hình chữ nhật bao quanh nhỏ nhất (cv2.minAreaRect) để xác định góc nghiêng và xoay ngược lại.[5]
4. **Tối ưu hóa cho OCR (Optimization):** Sau khi ảnh đã thẳng, chúng ta cần làm cho văn bản trở nên rõ nét nhất.
    - **Nhị phân hóa Thích ứng (Adaptive Thresholding):** Thay vì một ngưỡng cố định cho toàn bộ ảnh, phương pháp này sẽ tính toán ngưỡng cho các vùng nhỏ. Điều này cực kỳ hiệu quả với các nhãn có điều kiện ánh sáng không đồng đều (ví dụ: một góc bị bóng mờ).[6]
    - **Khử nhiễu (Denoising):** Sử dụng các thuật toán như cv2.fastNlMeansDenoising để loại bỏ các đốm nhiễu nhỏ li ti trên ảnh, giúp OCR tập trung vào ký tự.[6]

Mã nguồn: preprocessing.py

Python

```
import cv2
```

```python
import numpy as np
from PIL import Image
import io

def load_image_from_streamlit(uploaded_file):
    """
    Chuyển đổi file tải lên từ Streamlit (UploadedFile) sang định dạng ảnh
    OpenCV (mảng NumPy) mà vẫn giữ nguyên 3 kênh màu.
    """
    # Đọc dữ liệu file vào bộ nhớ
    image_data = uploaded_file.getvalue()

    # Sử dụng PIL để mở ảnh từ bytes
    pil_image = Image.open(io.BytesIO(image_data))

    # Chuyển đổi ảnh PIL sang mảng NumPy (định dạng OpenCV)
    # Đảm bảo chuyển đổi sang RGB (OpenCV mặc định là BGR khi đọc file)
    cv_image_rgb = np.array(pil_image)

    # Chuyển đổi từ RGB (PIL) sang BGR (OpenCV)
    cv_image_bgr = cv2.cvtColor(cv_image_rgb, cv2.COLOR_RGB2BGR)

    return cv_image_bgr

def deskew_image(image):
    """
    Tự động phát hiện và điều chỉnh độ nghiêng (deskew) của văn bản trong ảnh.
    Sử dụng logic từ  và.
    """
    # 1. Chuyển sang ảnh xám
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # 2. Đảo ngược màu (để văn bản trở thành màu trắng trên nền đen)
    # Điều này quan trọng cho việc tìm contour
    gray = cv2.bitwise_not(gray)

    # 3. Nhị phân hóa (Thresholding)
    # Sử dụng THRESH_OTSU để tự động tìm ngưỡng tốt nhất
    thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY | cv2.THRESH_OTSU)

    # 4. Tìm tọa độ của tất cả các pixel văn bản (màu trắng)
    coords = np.column_stack(np.where(thresh > 0))

    # 5. Tính toán hình chữ nhật xoay có diện tích nhỏ nhất (minAreaRect)
```

```
    # bao quanh tất cả các điểm ảnh
    angle = cv2.minAreaRect(coords)[-1]

    # 6. Chuẩn hóa góc: cv2.minAreaRect trả về góc trong [-90, 0)
    if angle < -45:
        angle = -(90 + angle)
    else:
        angle = -angle

    # Nếu góc quá nhỏ, có thể bỏ qua để tiết kiệm thời gian
    if abs(angle) < 0.1:
        return image

    # 7. Xoay ảnh để "deskew"
    (h, w) = image.shape[:2]
    center = (w // 2, h // 2)
    M = cv2.getRotationMatrix2D(center, angle, 1.0)

    # Sử dụng cờ INTER_CUBIC để chất lượng ảnh tốt hơn sau khi xoay
    rotated = cv2.warpAffine(image, M, (w, h),
                             flags=cv2.INTER_CUBIC,
                             borderMode=cv2.BORDER_REPLICATE)

    return rotated

def optimize_for_ocr(image):
    """
    Tối ưu hóa ảnh đã được deskew để OCR hoạt động tốt nhất.
    """
    # 1. Chuyển sang ảnh xám
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # 2. Khử nhiễu
    # fastNlMeansDenoising hiệu quả cho nhiễu Gaussian
    denoised = cv2.fastNlMeansDenoising(gray, None, h=10, templateWindowSize=7,
searchWindowSize=21)

    # 3. Nhị phân hóa Thích ứng (Adaptive Thresholding)
    # Đây là bước quan trọng nhất cho OCR, tốt hơn Otsu/thresholding
    # thông thường khi ánh sáng không đồng đều.
    optimized = cv2.adaptiveThreshold(denoised, 255,
                                      cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
                                      cv2.THRESH_BINARY,
```

```
                                blockSize=11,
                                C=2)

    return optimized

def full_preprocessing_pipeline(uploaded_file):
    """
    Hàm tổng hợp chạy toàn bộ pipeline.
    """
    original_cv_image = load_image_from_streamlit(uploaded_file)
    deskewed_image = deskew_image(original_cv_image)
    final_image_for_ocr = optimize_for_ocr(deskewed_image)

    # Trả về cả ảnh gốc (đã deskew) để hiển thị và ảnh đã tối ưu cho OCR
    return deskewed_image, final_image_for_ocr
```

Phần 3: Giai đoạn 2 - Xây dựng Module Nhận dạng (OCR) (ocr.py)

Sau khi có ảnh đã được làm sạch, bước này thực hiện trích xuất văn bản. Chúng ta sử dụng EasyOCR vì tính đơn giản và hiệu quả của nó.

Cấu hình EasyOCR

Quyết định cấu hình quan trọng nhất là:

1. **Ngôn ngữ (lang_list):** Chúng ta sẽ sử dụng ['vi', 'en'].[7] Lý do là nhãn bưu kiện thường chứa cả tiếng Việt (tên đường, tỉnh) và tiếng Anh (tên không dấu, mã đơn hàng, "Tel:", "Address:").
2. **Thiết bị (gpu):** Đặt gpu=False.[7] Đây là quyết định then chốt để "tối thiểu hóa công sức". Nó đảm bảo ứng dụng có thể chạy trên mọi máy tính (bao gồm cả môi trường miễn phí của Streamlit Cloud) mà không cần cài đặt CUDA phức tạp. EasyOCR chạy trên CPU đủ nhanh cho ứng dụng này.[8]

Xử lý Kết quả của EasyOCR

EasyOCR trả về một danh sách (list) các kết quả, mỗi kết quả bao gồm (bounding_box, text_string, confidence_score). Đối với bài toán này, chúng ta không cần bounding_box. Thay vào đó, chúng ta cần:

1. Nối (join) tất cả các text_string lại thành một khối văn bản duy nhất để đưa vào trình phân tích cú pháp.
2. Tính toán điểm tin cậy (confidence) trung bình. Đây là một chỉ số quan trọng để Kiểm soát Chất lượng (QC) ở Giai đoạn 5.

Mã nguồn: ocr.py

Python

```python
import easyocr
import streamlit as st
import numpy as np

@st.cache_resource
def initialize_ocr_reader():
    """
    Khởi tạo EasyOCR Reader và cache lại bằng Streamlit.
    Hàm này sẽ chỉ chạy một lần duy nhất khi ứng dụng khởi động.
    """
    # Quyết định quan trọng: ['vi', 'en'] và gpu=False
    reader = easyocr.Reader(['vi', 'en'], gpu=False)
    return reader

def extract_text_from_image(reader, image_np):
    """
    Sử dụng EasyOCR reader đã khởi tạo để đọc văn bản từ ảnh (mảng NumPy).

    Trả về:
```

```
    - (str) Một chuỗi văn bản thô duy nhất.
    - (float) Điểm tin cậy trung bình của OCR.
    """
    # image_np phải là ảnh đã được tiền xử lý (ảnh xám/nhị phân)

    # Chạy OCR
    # detail=1 để lấy cả text và confidence
    results = reader.readtext(image_np, detail=1)

    if not results:
        return "", 0.0  # Không tìm thấy văn bản nào

    all_text =
    all_confidence =

    for (bbox, text, conf) in results:
        all_text.append(text)
        all_confidence.append(conf)

    # 1. Tạo khối văn bản thô
    # Nối bằng dấu cách " " để tách các từ
    raw_text_block = " ".join(all_text)

    # 2. Tính điểm tin cậy trung bình
    avg_confidence = np.mean(all_confidence) if all_confidence else 0.0

    return raw_text_block, avg_confidence
```

Phần 4: Giai đoạn 3 - Xây dựng Module Logic Nghiệp vụ (logic.py)

Đây là "bộ não" của ứng dụng, chuyển đổi khối văn bản thô vô nghĩa từ OCR thành thông tin nghiệp vụ có giá trị: địa chỉ có cấu trúc và phân loại khu vực.

Thất bại của NER và Giải pháp Chuyên dụng

Một sai lầm phổ biến của người mới là cố gắng sử dụng các thư viện Nhận dạng Thực thể (NER) chung chung (như underthesea [13]). Mặc dù hữu ích, NER chỉ có thể gắn thẻ các từ như "Quận 1" hay "TPHCM" là "VỊ TRÍ" (LOC).[14] Nó không hiểu được hệ thống phân cấp hành chính (đâu là Tỉnh, đâu là Quận).

Do đó, logic nghiệp vụ "NỘI Ô" (dựa trên Quận/Huyện của TPHCM) sẽ thất bại.

Giải pháp Tối ưu (Tối thiểu hóa Công sức): Sử dụng thư viện vietnamadminunits.[12] Thư viện này được thiết kế chuyên biệt để phân tích địa chỉ Việt Nam. Hàm parse_address [12] của nó sẽ quét toàn bộ khối văn bản thô, tìm các từ khóa khớp với cơ sở dữ liệu hành chính và trả về một đối tượng có cấu trúc.

Quyết định Cấu hình: mode='LEGACY'

Khi gọi hàm parse_address, chúng ta sẽ sử dụng mode='LEGACY'.[12] Lý do là:

1. Nhãn bưu kiện hiện tại vẫn tuân theo cấu trúc 63 tỉnh/thành phố cũ.
2. Tài liệu API cho thấy mode='LEGACY' trả về đầy đủ các cấp province (tỉnh), district (quận/huyện), và ward (phường/xã) [12], đây là điều bắt buộc cho logic phân loại của chúng ta.

Xây dựng Cơ sở dữ liệu Phân loại (NỘI Ô)

Theo yêu cầu, "NỘI Ô" được định nghĩa là các địa chỉ thuộc Thành phố Hồ Chí Minh. Dựa trên dữ liệu hành chính chính thức [17], TP. Hồ Chí Minh bao gồm: 1 thành phố, 16 quận, và 5 huyện. Chúng ta sẽ định nghĩa một set trong Python chứa tất cả các đơn vị này để tra cứu.

Bảng 2: Cơ sở dữ liệu Phân loại NỘI Ô (TP. Hồ Chí Minh)

Loại	Tên Đơn vị Hành chính (Chuẩn hóa)
Thành phố	Thành phố Thủ Đức

Quận	Quận 1, Quận 3, Quận 4, Quận 5, Quận 6, Quận 7, Quận 8, Quận 10, Quận 11, Quận 12
Quận	Quận Bình Tân, Quận Bình Thạnh, Quận Gò Vấp, Quận Phú Nhuận, Quận Tân Bình, Quận Tân Phú
Huyện	Huyện Bình Chánh, Huyện Cần Giờ, Huyện Củ Chi, Huyện Hóc Môn, Huyện Nhà Bè

Mã nguồn: logic.py

Python

```python
import re
from vietnamadminunits import parse_address, AdminUnit
from vietnamadminunits.enums import ParseMode

# Cơ sở dữ liệu các đơn vị hành chính của TPHCM (NỘI Ô)
# Dựa trên. Đã chuẩn hóa về chữ thường để so sánh.
HCM_UNITS_SET = {
    # Thành phố
    "thành phố thủ đức",
    # 16 Quận
    "quận 1", "quận 3", "quận 4", "quận 5", "quận 6", "quận 7", "quận 8",
    "quận 10", "quận 11", "quận 12", "quận bình tân", "quận bình thạnh",
    "quận gò vấp", "quận phú nhuận", "quận tân bình", "quận tân phú",
    # 5 Huyện
    "huyện bình chánh", "huyện cần giờ", "huyện củ chi", "huyện hóc môn",
    "huyện nhà bè"
}

def clean_ocr_text(raw_text):
    """
    Làm sạch văn bản OCR thô trước khi đưa vào trình phân tích cú pháp.
    Ví dụ: loại bỏ các ký tự đặc biệt, mã bưu kiện, SĐT...
```

```
    """
    # Loại bỏ SĐT (ví dụ: 09... , 84...)
    cleaned_text = re.sub(r'(\b(0|84)\d{9}\b)', '', raw_text)
    # Loại bỏ các từ khóa không liên quan (có thể thêm nếu cần)
    cleaned_text = re.sub(r'(SĐT|Tel|Phone|Mã đơn|Người nhận|Người gửi):?', '', cleaned_text,
flags=re.IGNORECASE)
    # Thay thế nhiều dấu cách bằng một dấu cách
    cleaned_text = re.sub(r'\s+', ' ', cleaned_text).strip()
    return cleaned_text

def extract_address_components(raw_text_block):
    """
    Sử dụng vietnamadminunits để phân tích khối văn bản thô thành các
    thành phần địa chỉ có cấu trúc.
    """
    if not raw_text_block:
        return None

    cleaned_text = clean_ocr_text(raw_text_block)

    try:
        # Quyết định quan trọng: Sử dụng mode='LEGACY'
        # để đảm bảo nhận dạng được cấu trúc 63 tỉnh thành hiện tại
        # và trả về đầy đủ Tỉnh, Quận, Phường.
        parsed_unit = parse_address(cleaned_text, mode=ParseMode.LEGACY)
        return parsed_unit
    except Exception as e:
        print(f"Lỗi khi phân tích địa chỉ: {e}")
        return None

def classify_location(parsed_unit: AdminUnit):
    """
    Phân loại địa chỉ là NỘI Ô hay NGOẠI Ô dựa trên đối tượng AdminUnit.
    """
    if not parsed_unit:
        return "NGOẠI Ô", "Không thể xác định"

    province_name = (parsed_unit.province or "").lower()
    district_name = (parsed_unit.district or "").lower()

    # Tạo tên quận/huyện đầy đủ (ví dụ: "quận 1", "huyện bình chánh")
    # vì đối tượng AdminUnit trả về district_type
    full_district_name = f"{parsed_unit.district_type.lower()} {district_name}".strip()
```

```
    # Logic phân loại chính
    if province_name == "thành phố hồ chí minh":
        return "NỘI Ô", f"{parsed_unit.province}"

    # Kiểm tra xem quận/huyện chuẩn hóa có nằm trong set NỘI Ô không
    if full_district_name in HCM_UNITS_SET:
        return "NỘI Ô", f"TP. Hồ Chí Minh (Phát hiện qua {full_district_name})"

    # Trường hợp còn lại là NGOẠI Ô
    if parsed_unit.province:
        return "NGOẠI Ô", f"{parsed_unit.province}"
    else:
        return "NGOẠI Ô", "Không thuộc TP. HCM"

def format_address_string(parsed_unit: AdminUnit):
    """
    Định dạng chuỗi địa chỉ đầu ra cho người dùng.
    """
    if not parsed_unit:
        return "Không trích xuất được địa chỉ."

    # Cố gắng xây dựng địa chỉ đầy đủ nhất có thể
    parts =
    # Thuộc tính 'street' cũng có sẵn
    if parsed_unit.street:
        parts.append(parsed_unit.street)
    if parsed_unit.ward:
        parts.append(parsed_unit.ward)
    if parsed_unit.district:
        parts.append(parsed_unit.district)
    if parsed_unit.province:
        parts.append(parsed_unit.province)

    if not parts:
         return "Không tìm thấy thông tin địa chỉ."

    return ", ".join(parts)
```

Phần 5: Giai đoạn 4 - Tích hợp Giao diện Người dùng

(app.py)

Đây là bước cuối cùng, liên kết tất cả các mô-đun (preprocessing, ocr, logic) lại với nhau trong một giao diện Streamlit thân thiện với người dùng.

Thiết kế Bố cục (Layout)

Để "tối đa hóa hiệu quả", một giao diện hai cột (side-by-side) là lựa chọn tối ưu. Chúng ta sẽ sử dụng st.columns(2) [19]:

- **Cột 1 (Bên trái):** Cho phép người dùng tải ảnh lên (st.file_uploader [16]) và hiển thị ảnh gốc (hoặc ảnh đã deskew).
- **Cột 2 (Bên phải):** Hiển thị kết quả: Phân loại, Địa chỉ trích xuất, và các thông tin gỡ lỗi (như văn bản OCR thô).

Tối ưu hóa Hiệu suất: Caching

Mô hình EasyOCR (ocr.py) tốn vài giây để tải vào bộ nhớ. Nếu không cache, mỗi lần người dùng tương tác với UI, mô hình sẽ bị tải lại. Chúng ta sửll dụng @st.cache_resource (đã triển khai trong ocr.py) để đảm bảo mô hình chỉ được tải một lần duy nhất khi ứng dụng khởi động.

Xử lý Lỗi và Kiểm soát Chất lượng (QC)

Ứng dụng sẽ được bao bọc trong một khối try...except [21] để xử lý các lỗi không mong muốn (ví dụ: ảnh quá mờ, không có văn bản, không phải địa chỉ).

Quan trọng hơn, chúng ta sẽ sử dụng avg_confidence (điểm tin cậy trung bình) trả về từ ocr.py. Đây là một cơ chế QC tự động.[22] Nếu điểm tin cậy thấp (ví dụ: dưới 70%), chúng ta vẫn hiển thị kết quả nhưng kèm theo một cảnh báo (st.warning) cho người dùng biết rằng kết quả có thể không chính xác.

Mã nguồn: app.py

Python

```
import streamlit as st
from PIL import Image
import numpy as np

# Import các mô-đun đã xây dựng
import preprocessing
import ocr
import logic

# === CẤU HÌNH TRANG ===
st.set_page_config(
    page_title="Trình Phân loại Nhãn Bưu kiện",
    page_icon="📦",
    layout="wide"
)

# === TẢI MÔ HÌNH (CACHE) ===
# Tải mô hình OCR một lần duy nhất bằng cache của Streamlit
# Hàm này được định nghĩa trong ocr.py
try:
    ocr_reader = ocr.initialize_ocr_reader()
    st.sidebar.success("Mô hình OCR đã sẵn sàng.")
except Exception as e:
    st.sidebar.error(f"Lỗi khởi tạo OCR: {e}")
    st.stop()

# === GIAO DIỆN CHÍNH ===
st.title("📦 Ứng dụng OCR Nhận dạng và Phân loại Nhãn Bưu kiện")
st.caption("Tải lên ảnh nhãn bưu kiện để tự động trích xuất địa chỉ và phân loại NỘI Ô/NGOẠI Ô.")

# Sử dụng st.columns để tạo layout 2 cột
col1, col2 = st.columns() # Cột 1 nhỏ hơn cột 2
```

```
# --- CỘT 1: ĐẦU VÀO ---
with col1:
    st.header("1. Đầu vào")

    # Widget tải file
    uploaded_file = st.file_uploader(
        "Tải lên ảnh nhãn bưu kiện:",
        type=["jpg", "png", "jpeg"],
        help="Tải ảnh rõ nét, đã xoay đúng chiều để có kết quả tốt nhất."
    )

    if uploaded_file is not None:
        # Hiển thị ảnh đã tải lên
        image = Image.open(uploaded_file)
        st.image(image, caption="Ảnh bưu kiện đã tải lên", use_column_width=True)

        # Nút kích hoạt xử lý
        process_button = st.button("🚀 Bắt đầu Xử lý", type="primary", use_container_width=True)
    else:
        process_button = False

# --- CỘT 2: KẾT QUẢ ---
with col2:
    st.header("2. Kết quả")

    if process_button:
        # Bắt đầu xử lý khi người dùng nhấn nút
        with st.spinner("Đang xử lý, vui lòng chờ..."):
            try:
                # BƯỚC 1: TIỀN XỬ LÝ ẢNH
                # Chạy pipeline tiền xử lý đầy đủ
                deskewed_image, final_image_for_ocr =
preprocessing.full_preprocessing_pipeline(uploaded_file)

                # BƯỚC 2: CHẠY OCR
                raw_text, avg_confidence = ocr.extract_text_from_image(
                    ocr_reader,
                    final_image_for_ocr
                )

                if not raw_text:
                    st.error("Không phát hiện được văn bản nào trong ảnh. Vui lòng thử ảnh khác.")
                    st.stop()
```

```
            # BƯỚC 3: PHÂN TÍCH LOGIC
            parsed_unit = logic.extract_address_components(raw_text)

            if parsed_unit is None:
                st.warning("Đã trích xuất được văn bản, nhưng không nhận dạng được địa chỉ Việt Nam hợp lệ.")
                # Vẫn hiển thị raw text để gỡ lỗi
                classification = "CHƯA XÁC ĐỊNH"
                address_string = "Không thể phân tích địa chỉ."
            else:
                classification, region_info = logic.classify_location(parsed_unit)
                address_string = logic.format_address_string(parsed_unit)

            # BƯỚC 4: HIỂN THỊ KẾT QUẢ
            st.subheader("Kết quả Phân loại")

            if classification == "NỘI Ô":
                st.metric("Khu vực Giao hàng", "🏙️ NỘI Ô", f"{region_info}")
            else:
                st.metric("Khu vực Giao hàng", "🚚 NGOẠI Ô", f"{region_info}")

            st.subheader("Địa chỉ Trích xuất")
            st.info(address_string)

            # Kiểm soát Chất lượng (QC) dựa trên Confidence
            st.subheader("Thông tin Kỹ thuật")
            st.progress(avg_confidence, text=f"Độ tin cậy OCR trung bình: {avg_confidence:.1%}")

            if avg_confidence < 0.7: # Ngưỡng 70%
                st.warning("Độ tin cậy OCR thấp. Kết quả trên có thể không chính xác. "
                           "Vui lòng kiểm tra lại văn bản thô bên dưới.")

            # Hiển thị ảnh đã xử lý và văn bản thô để gỡ lỗi
            with st.expander("Xem chi tiết xử lý (để gỡ lỗi)"):
                st.text_area("Văn bản OCR thô trích xuất được:", value=raw_text, height=200) # [2]
                st.image(deskewed_image, caption="Ảnh đã được tự động điều chỉnh độ nghiêng", use_column_width=True)
                st.image(final_image_for_ocr, caption="Ảnh đã tối ưu hóa cho OCR (Nhị phân hóa)", use_column_width=True)

        except Exception as e:
            st.error(f"Đã xảy ra lỗi trong quá trình xử lý: {e}")
```

```
            st.error("Vui lòng thử lại với một ảnh khác hoặc kiểm tra lại file ảnh.")
    else:
        st.info("Vui lòng tải ảnh lên và nhấn nút 'Bắt đầu Xử lý'.")
```

Phần 6: Giai đoạn 5 - Kiểm soát Chất lượng và Triển khai

Một kế hoạch "hoàn hảo" phải bao gồm cả việc xử lý các trường hợp ngoại lệ và một lộ trình rõ ràng để chia sẻ ứng dụng.

Xử lý Lỗi và các Trường hợp Ngoại lệ (Edge Cases)

Hệ thống đã được thiết kế để xử lý các lỗi phổ biến:

1. **Ảnh không có văn bản:** ocr.py sẽ trả về chuỗi rỗng. app.py sẽ phát hiện điều này và báo lỗi st.error cho người dùng.
2. **Văn bản không phải địa chỉ (ví dụ: ảnh một cuốn sách):** ocr.py trích xuất văn bản, nhưng logic.py (hàm parse_address) sẽ trả về None. app.py sẽ phát hiện điều này và báo st.warning, yêu cầu người dùng kiểm tra.
3. **Lỗi chính tả OCR (ví dụ: "Hò Chí Minh"):** Các thư viện như vietnamadminunits và vn_address_standardizer [23] được thiết kế để có khả năng chịu lỗi (fuzzy matching) với các lỗi chính tả và không dấu cơ bản, đây chính là lý do chúng ta chọn chúng thay vì tự viết logic so khớp chuỗi.

Lộ trình Triển khai (Deployment)

Sau khi ứng dụng chạy ổn định trên máy cục bộ (streamlit run app.py), có hai phương án chính để triển khai.

Lựa chọn 1: Streamlit Community Cloud (Đề xuất cho người mới)
Đây là cách đơn giản nhất, miễn phí và được quản lý hoàn toàn bởi Streamlit.24

1. **Yêu cầu:** Đẩy toàn bộ thư mục dự án (bao gồm requirements.txt) lên một kho lưu trữ

(repository) GitHub công khai.

2. **Tệp packages.txt:** Cả OpenCV và EasyOCR đều yêu cầu một số thư viện hệ thống của Linux để chạy. Cần **bắt buộc** tạo một tệp mới tên là packages.txt trong thư mục gốc của repo với nội dung sau:
   ```
   freeglut3-dev
   libgtk2.0-dev
   libgl1-mesa-glx
   ```

3. **Triển khai:** Truy cập share.streamlit.io, đăng nhập bằng tài khoản GitHub, chọn repository, và nhấn "Deploy".

Lựa chọn 2: Docker (Lựa chọn Chuyên nghiệp & Linh hoạt)
Phương án này tạo ra một "image" chứa toàn bộ ứng dụng, có thể chạy ở bất cứ đâu (Heroku, AWS, Google Cloud, VPS).24

1. **Tạo Dockerfile:** Tạo một tệp tên Dockerfile (không có phần mở rộng) trong thư mục gốc.
2. Nội dung Dockerfile [25]:
   Dockerfile
   ```
   # Sử dụng base image Python 3.9-slim
   FROM python:3.9-slim

   # Cài đặt các dependencies hệ thống cho OpenCV
   RUN apt-get update && apt-get install -y \
       libgl1-mesa-glx \
       libglib2.0-0 \
       libgtk2.0-dev \
       && rm -rf /var/lib/apt/lists/*

   # Đặt thư mục làm việc trong container
   WORKDIR /app

   # Sao chép tệp requirements và cài đặt thư viện Python
   COPY requirements.txt./
   RUN pip install --no-cache-dir -r requirements.txt

   # Sao chép toàn bộ mã nguồn ứng dụng vào container
   COPY..

   # Mở cổng 8501 (cổng mặc định của Streamlit)
   EXPOSE 8501

   # Thêm health check để Docker biết ứng dụng còn chạy hay không
   HEALTHCHECK CMD curl --fail http://localhost:8501/_stcore/health

   # Lệnh để chạy ứng dụng Streamlit
   ENTRYPOINT ["streamlit", "run", "app.py", "--server.port=8501", "--server.address=0.0.0.0"]
   ```

3. **Build và Run (Trên máy chủ/local):**
   Bash
   ```
   # Build image
   docker build -t ocr-parcel-app.

   # Run container
   docker run -p 8501:8501 ocr-parcel-app
   ```

   Sau đó, ứng dụng sẽ có thể truy cập tại http://localhost:8501.

Phần 7: Tổng kết Kế hoạch và Lộ trình Tối ưu

Bằng cách tuân theo kế hoạch 5 giai đoạn này, một nhà phát triển có thể xây dựng một ứng dụng OCR mạnh mẽ, chính xác và dễ bảo trì mà không cần kinh nghiệm AI/ML chuyên sâu.

Lộ trình Tóm tắt:

1. **Giai đoạn 1 (Nền tảng):** Sao chép cấu trúc dự án và cài đặt requirements.txt.
2. **Giai đoạn 2 (Tiền xử lý):** Tạo tệp preprocessing.py với mã nguồn được cung cấp.
3. **Giai đoạn 3 (OCR):** Tạo tệp ocr.py với mã nguồn được cung cấp.
4. **Giai đoạn 4 (Logic):** Tạo tệp logic.py với mã nguồn được cung cấp (bao gồm cả cơ sở dữ liệu HCM_UNITS_SET).
5. **Giai đoạn 5 (Tích hợp):** Tạo tệp app.py, liên kết 3 mô-đun trên, và chạy thử nghiệm.

Thành công của dự án này không nằm ở sự phức tạp của mô hình AI, mà nằm ở sự vững chắc của *pipeline* (Đường ống xử lý): từ **Tiền xử lý** ảnh (loại bỏ nhiễu và độ nghiêng) đến **Phân tích cú pháp** chuyên dụng (sử dụng đúng thư viện để hiểu địa chỉ Việt Nam). Kế hoạch này được thiết kế để tối ưu hóa chính xác các điểm mấu chốt đó, đảm bảo hiệu quả và giảm thiểu công sức.

Nguồn trích dẫn

1. Python Tutorial to build Image to Text App using EasyOCR & Streamlit - YouTube, truy cập vào tháng 11 13, 2025, https://www.youtube.com/watch?v=j7TH0MRlnGs
2. Xử lý hình ảnh trong Python: từ thuật toán đến công cụ - VinBigdata, truy cập vào tháng 11 13, 2025, https://vinbigdata.com/camera-ai/xu-ly-hinh-anh-trong-python-tu-thuat-toan-den-cong-cu.html
3. Text skew correction with OpenCV and Python - PyImageSearch, truy cập vào tháng 11 13, 2025, https://pyimagesearch.com/2017/02/20/text-skew-correction-opencv-python/

4. 7 steps of image pre-processing to improve OCR using Python, truy cập vào tháng 11 13, 2025, https://nextgeninvent.com/blogs/7-steps-of-image-pre-processing-to-improve-ocr-using-python-2/
5. JaidedAI/EasyOCR: Ready-to-use OCR with 80+ supported ... - GitHub, truy cập vào tháng 11 13, 2025, https://github.com/JaidedAI/EasyOCR
6. Tesseract vs EasyOCR - ArivElm, truy cập vào tháng 11 13, 2025, https://arivelm.com/tesseract-vs-easyocr/
7. Convert Image to Editable Text using Python | Streamlit + Tesseract OCR Full Tutorial (Web + Mobile) - YouTube, truy cập vào tháng 11 13, 2025, https://www.youtube.com/watch?v=-dcyPlwe8uI
8. PaddleOCR vs Tesseract: Which is the best open source OCR? - Koncile, truy cập vào tháng 11 13, 2025, https://www.koncile.ai/en/ressources/paddleocr-analyse-avantages-alternatives-open-source
9. Gemini beats everyone is OCR benchmarking tasks in videos. Full Paper : https://arxiv.org/abs/2502.06445 : r/LocalLLaMA - Reddit, truy cập vào tháng 11 13, 2025, https://www.reddit.com/r/LocalLLaMA/comments/1ioikl0/gemini_beats_everyone_is_ocr_benchmarking_tasks/
10. tranngocminhhieu/vietnamadminunits: Helps businesses ... - GitHub, truy cập vào tháng 11 13, 2025, https://github.com/tranngocminhhieu/vietnamadminunits
11. undertheseanlp/underthesea: Underthesea - Vietnamese NLP Toolkit - GitHub, truy cập vào tháng 11 13, 2025, https://github.com/undertheseanlp/underthesea
12. Under The Sea Documentation, truy cập vào tháng 11 13, 2025, https://underthesea.readthedocs.io/_/downloads/en/v6.4.0/pdf/
13. How to pre-process images for OCR ? edit - OpenCV Q&A Forum, truy cập vào tháng 11 13, 2025, https://answers.opencv.org/question/235432/how-to-pre-process-images-for-ocr/
14. st.file_uploader - Streamlit Docs, truy cập vào tháng 11 13, 2025, https://docs.streamlit.io/develop/api-reference/widgets/st.file_uploader
15. truy cập vào tháng 11 13, 2025, https://vi.wikipedia.org/wiki/Th%C3%A0nh_ph%E1%BB%91_H%E1%BB%93_Ch%C3%AD_Minh#:~:text=Hi%E1%BB%87n%20nay%2C%20Th%C3%A0nh%20ph%E1%BB%91%20H%E1%BB%93,ph%C6%B0%E1%BB%9Dng%2C%20x%C3%A3%2C%20th%E1%BB%8B%20tr%E1%BA%A5n.
16. TPHCM Có Bao Nhiêu Quận Huyện? Cập Nhật Mới Nhất, truy cập vào tháng 11 13, 2025, https://dichvudongduong.com/tphcm-co-bao-nhieu-quan-huyen/
17. st.columns - Streamlit Docs, truy cập vào tháng 11 13, 2025, https://docs.streamlit.io/develop/api-reference/layout/st.columns
18. Streamlit Columns Explained: Grid Layout, Data Display, and Interaction - Kanaries Docs, truy cập vào tháng 11 13, 2025, https://docs.kanaries.net/topics/Streamlit/streamlit-columns
19. 8. Errors and Exceptions — Python 3.14.0 documentation, truy cập vào tháng 11

13, 2025, https://docs.python.org/3/tutorial/errors.html
20. Python OCR Tutorial: Tesseract, Pytesseract, and OpenCV - Nanonets, truy cập vào tháng 11 13, 2025, https://nanonets.com/blog/ocr-with-tesseract/
21. vantrong291/vn_address_standardizer: A package for ... - GitHub, truy cập vào tháng 11 13, 2025, https://github.com/vantrong291/vn_address_standardizer
22. Deployment tutorials - Streamlit Docs, truy cập vào tháng 11 13, 2025, https://docs.streamlit.io/deploy/tutorials
23. Deploy Streamlit using Docker - Streamlit Docs, truy cập vào tháng 11 13, 2025, https://docs.streamlit.io/deploy/tutorials/docker
24. Deploying Streamlit Apps with Docker on Heroku, truy cập vào tháng 11 13, 2025, https://discuss.streamlit.io/t/deploying-streamlit-apps-with-docker-on-heroku/5136