ENZIGMA- Assignment for AI Intern Position

ASSIGNMENT 01

Name- Akshay Bhosale

PRN- 202101040158

MIT Academy of engineering, Pune

Project Documentation: Automated Onboarding System with OCR and Database Integration

GitHub Link:- https://github.com/letsdoitbycode/OCR_detection

Demo drive links:- https://drive.google.com/drive/folders/1JQt4TOC1ZH_4Xh9YR4lZYCMIM6eo1ORG?usp=sharing

Overview of the Solution

This project will develop an onboarding automation process, during which users can upload a scanned form in the format of an image. Solution: The textual information is captured from these forms, further consolidated into structured data, placed in a MySQL database, and provided with an easy-to-navigate user interface for uploading files as well as managing records.

Key Features:

1. File Upload: Upload scanned images in .jpg, .jpeg, or .png formats.
2. OCR Processing: Extract both printed and handwritten text from images.
3. Data Consolidation: Process and clean extracted text, converting it into a structured tabular format.
4. Database Storage: Save structured data into a MySQL database for easy management and retrieval.
5. Record Management:
    - View all records in a tabular format.

- Search records by name or email.

```
//requirements.txt

groq
python-dotenv
numpy
streamlit
langchain-groq
langchain_community
pillow
mysql-connector-python
PyPDF2
PyMuPDF
```

Steps to Set Up and Run the System

Prerequisites:

1. Python 3.x installed on your machine.
2. MySQL server running locally or on a remote server.

Environment Variables:

Create a .env file in the project root directory with the following variables:

```
GROQ_API_KEY = os.getenv("GROQ_API_KEY")
DB_HOST = os.getenv("DB_HOST")
DB_USER = os.getenv("DB_USER")
DB_PASSWORD = os.getenv("DB_PASSWORD")
DB_NAME = os.getenv("DB_NAME")
```

Running the Application:

1. Start the MySQL server and ensure the database is accessible.
2. Run the Streamlit application using the command:

➔ streamlit run app.py

Database Schema

Database Table: Record

The Record table stores extracted details in a structured format. Below is the schema:

Field	Type	Description
name	VARCHAR(255)	Full name of the candidate.

Field	Type	Description
email	VARCHAR(255)	Email address of the candidate.
address	TEXT	Residential address.
dob	DATE	Date of birth.
age	INT	Age of the candidate.
gender	VARCHAR(50)	Gender of the candidate.
mobile	VARCHAR(15)	Contact number.
education	TEXT	Educational qualifications.
profile	TEXT	Professional profile or job title.

Code and Functionalities

1. Image Upload and Preprocessing

The system allows users to upload scanned forms and preprocesses them for OCR. Preprocessing includes resizing the image and splitting it into horizontal stripes for better accuracy during text extraction.

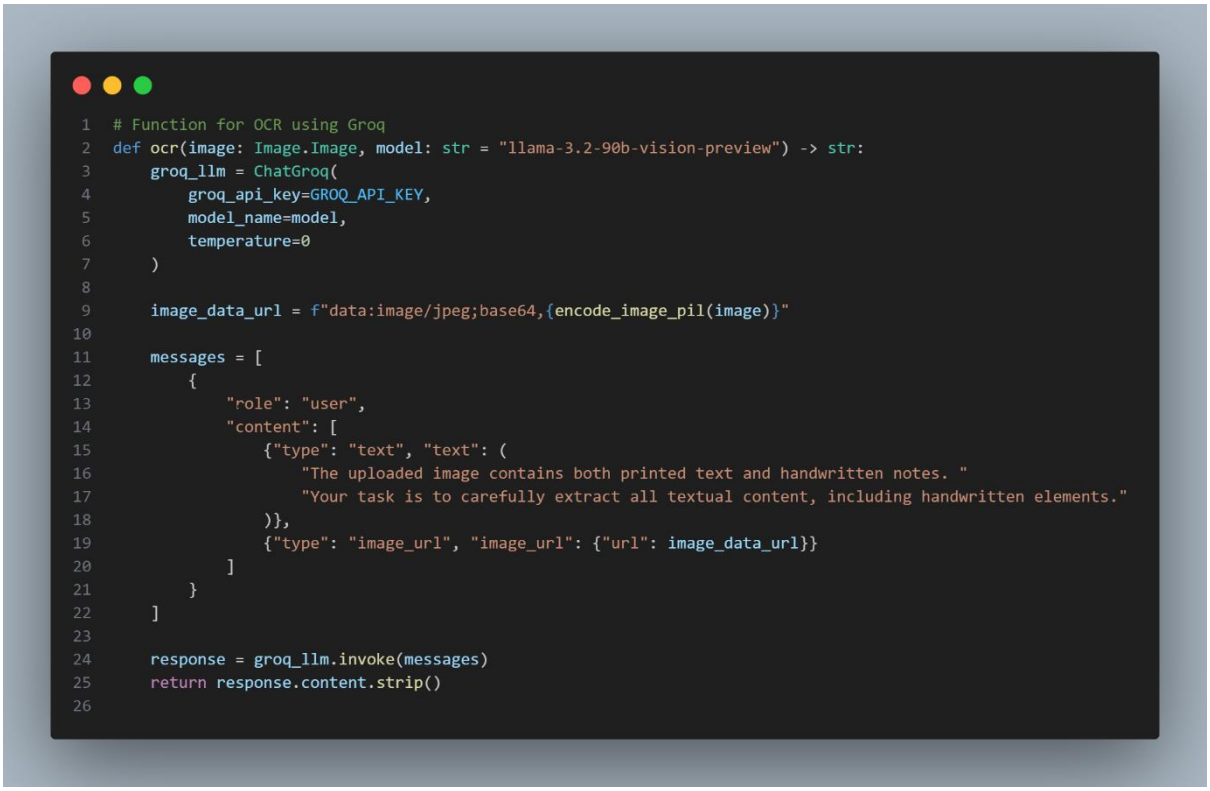
```python
# Function to encode an image to base64
def encode_image_pil(image: Image.Image) -> str:
    buffered = io.BytesIO()
    image = image.convert("RGB")
    image.save(buffered, format="JPEG", quality=85)
    return base64.b64encode(buffered.getvalue()).decode("utf-8")

# Function to split an image into horizontal stripes
def split_image_into_horizontal_stripes(image: Image.Image, stripe_count: int = 5, overlap: float = 0.1):
    width, height = image.size
    stripe_height = height // stripe_count
    overlap_height = int(stripe_height * overlap)

    stripes = []
    for i in range(stripe_count):
        upper = max(i * stripe_height - overlap_height, 0)
        lower = min((i + 1) * stripe_height + overlap_height, height)
        stripe = image.crop((0, upper, width, lower))
        stripes.append(stripe)
    return stripes
```

2. OCR Functionality

Extracts textual content from uploaded images. Both printed and handwritten elements are captured and processed.



```python
# Function for OCR using Groq
def ocr(image: Image.Image, model: str = "llama-3.2-90b-vision-preview") -> str:
    groq_llm = ChatGroq(
        groq_api_key=GROQ_API_KEY,
        model_name=model,
        temperature=0
    )

    image_data_url = f"{encode_image_pil(image)}"

    messages = [
        {
            "role": "user",
            "content": [
                {"type": "text", "text": (
                    "The uploaded image contains both printed text and handwritten notes. "
                    "Your task is to carefully extract all textual content, including handwritten elements."
                )},
                {"type": "image_url", "image_url": {"url": image_data_url}}
            ]
        }
    ]

    response = groq_llm.invoke(messages)
    return response.content.strip()

```

3. Data Consolidation

The extracted text from image sections is combined into a structured format, eliminating duplicates and resolving conflicts between overlapping sections.

```python
# Function to consolidate markdown into tabular format
def format_to_table(markdown_runs: list, model: str = "llama-3.3-70b-versatile") -> str:
    groq_llm = ChatGroq(
        groq_api_key=GROQ_API_KEY,
        model_name=model,
        temperature=0
    )

    combined_markdown = "\n\n".join(markdown_runs)

    messages = [
        {
            "role": "user",
            "content": (
                "You are provided with multiple markdown outputs extracted from overlapping sections of an image."
                "Some sections may contain duplicate or conflicting information due to overlaps. "
                "Your task is to:"
                "\n\n1. Identify and consolidate rows of data that are related, ensuring that the most complete version of the information is retained."
                "\n2. For rows with conflicting information (e.g., different values for a field), prioritize the more detailed entry."
                "\n3. If a field is missing in one row but present in another, combine the information into a single row."
                "\n4. Output the consolidated data in a clean tabular format using Markdown syntax, suitable for direct rendering."
                "\n5. Output Only Markdown: Return solely the Markdown content without any additional explanations or comments."
                "\n\nHere is the data to process:\n\n"
                + combined_markdown
            )
        }
    ]

    response = groq_llm.invoke(messages)
    return response.content.strip()

```

4. Data Parsing

Converts the consolidated tabular data into a Python dictionary format for database insertion.

```python
# Function to parse the consolidated markdown into a dictionary for database insertion
def parse_markdown_to_dict(markdown_table: str) -> list:
    rows = markdown_table.split("\n")[2:]  # Skip the header row
    records = []
    for row in rows:
        fields = [field.strip() for field in row.split("|")[1:-1]]
        if len(fields) >= 9:
            record = {
                "name": fields[0],
                "email": fields[1],
                "address": fields[2],
                "dob": fields[3],
                "age": int(fields[4]) if fields[4].isdigit() else None,
                "gender": fields[5],
                "mobile": fields[6],
                "education": fields[7],
                "profile": fields[8],
            }
            records.append(record)
    return records
```

5. Database Operations

- Insert Records: Saves extracted details into the MySQL database.
- Fetch All Records: Retrieves all stored records.
- Search Records: Searches for records by name or email.

```python
# Function to insert records into MySQL database
def insert_records_to_db(records: list):
    try:
        connection = mysql.connector.connect(
            host=DB_HOST,
            user=DB_USER,
            password=DB_PASSWORD,
            database=DB_NAME
        )
        cursor = connection.cursor()

        query = """
        INSERT INTO Record (name, email, address, dob, age, gender, mobile, education, profile)
        VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s)
        """
        for record in records:
            cursor.execute(query, (
                record["name"], record["email"], record["address"], record["dob"],
                record["age"], record["gender"], record["mobile"],
                record["education"], record["profile"]
            ))
        connection.commit()
        st.success("Records have been successfully inserted into the database!")
    except Error as e:
        st.error(f"Error connecting to the database: {e}")
    finally:
        if connection.is_connected():
            cursor.close()
            connection.close()

# Function to fetch all records from the database
def fetch_all_records():
    try:
        connection = mysql.connector.connect(
            host=DB_HOST,
            user=DB_USER,
            password=DB_PASSWORD,
            database=DB_NAME
        )
        cursor = connection.cursor(dictionary=True)
        cursor.execute("SELECT * FROM Record")
        records = cursor.fetchall()
        return records
    except Error as e:
        st.error(f"Error fetching data: {e}")
        return []
    finally:
        if connection.is_connected():
            cursor.close()
            connection.close()

# Function to search records by name or email
def search_records(search_term):
    try:
        connection = mysql.connector.connect(
            host=DB_HOST,
            user=DB_USER,
            password=DB_PASSWORD,
            database=DB_NAME
        )
        cursor = connection.cursor(dictionary=True)
        query = """
        SELECT * FROM Record
        WHERE name LIKE %s OR email LIKE %s
        """
        cursor.execute(query, (f"%{search_term}%", f"%{search_term}%"))
        records = cursor.fetchall()
        return records
    except Error as e:
        st.error(f"Error searching data: {e}")
        return []
    finally:
        if connection.is_connected():
            cursor.close()
            connection.close()


```

6. Streamlit Interface

The user interface built with Streamlit includes:

1. Sidebar for Uploading Files: Allows users to upload images and displays a preview.
2. Main Section: Displays OCR results and buttons to save data to the database.
3. Record Management: Provides options to view all records or search specific entries.

Instructions for Using the User Interface

1. Uploading an Image:
    - Use the Upload Image section in the sidebar.
    - Preview the uploaded image to confirm its accuracy.
2. Processing and Saving Data:
    - After uploading, the image is processed to extract text.
    - Review the consolidated tabular data in the OCR and Results section.
    - Click Insert Records into Database to save the data.
3. Managing Records:
    - Use the View All Records button to see all stored entries.
    - Search for specific records using the Search Records input field.