Infosys Internship 5.0

Title: Project Documentation: Smart Recipe Generator

• Introduction:

The **Smart Recipe Generator** is an innovative application designed to transform the way users plan and prepare meals. By utilizing cutting-edge technologies such as machine learning, optical character recognition (OCR), the system aims to provide personalized recipe recommendations based on the ingredients users already have at home. Through this application, users can upload images of raw ingredients or packaged items, and the system will accurately detect the ingredients, suggest recipes, and generate step-by-step cooking instructions.

The primary objectives of the project are:

- To minimize food waste by suggesting recipes tailored to available ingredients.
- To enhance meal planning by providing users with diverse, customizable recipe options.
- To create an intuitive and efficient tool that supports dietary preferences and culinary creativity.

• Project Scope:

Inclusions

The **Smart Recipe Generator** focuses on delivering an intuitive and user-friendly application for meal planning by incorporating the following functionalities:

1. **Ingredient Recognition**
    - Integration of machine learning models to identify raw ingredients from uploaded images.
    - Optical Character Recognition (OCR) for extracting text from packaged or labeled items.

2. **Recipe Recommendations**
    - Algorithms to match recognized ingredients with recipes using collaborative or content-based filtering techniques.
3. **Instruction Generation**
    - Use of GPT-3 model to generate detailed, step-by-step cooking instructions tailored to the selected recipe.
4. **User Interface (UI)**
    - A platform for users to upload images of ingredients.
    - A clean, interactive interface to display personalized recipe recommendations and instructions.

Exclusions

The following functionalities are outside the scope of this project:

- Real-time tracking of ingredients in the user's pantry or fridge.
- Integration with external APIs for grocery delivery or nutritional analysis.
- Advanced dietary customization such as strict allergy filtering, which may be implemented in future versions.
- Voice-enabled recipe navigation or live cooking assistance.

Limitations and Constraints

1. **Technical Constraints**
    - Low-resolution or poorly lit images may result in errors.
    - OCR might misread text on labels if the text is curved, distorted, or written in uncommon fonts.
2. **Performance Constraints**
    - The processing time for image recognition and recipe generation could be affected by the size of the dataset or server limitations.
3. **Generated Recipe Constraints**
    - The recipe database may not encompass all possible cuisines or ingredients, limiting the diversity of recommendations.
4. **User Constraints**
    - Users need to manually upload images of ingredients; no real-time scanning or automated input is available.

• **Requirements:**

Functional Requirements

1. **Ingredient Recognition**
    - The system must allow users to upload images of raw ingredients or packaged food items.
    - OCR (Tesseract and EasyOCR) should extract and recognize text from packaged items to identify ingredient labels.
2. **Cooking Instruction Generation**
    - The system should integrate with GPT-3 model to generate step-by-step cooking instructions for selected recipes.
3. **Database Interaction**
    - The system should store recognized ingredients in a database for querying and the database must include recipe details such as ingredients, instructions, and nutritional values.
4. **Error Handling**
    - If an ingredient cannot be identified, the system should notify the user and allow them to manually correct or add it.

Non-Functional Requirements

1. **Performance**
    - The system must process image uploads and provide recipe recommendations within a minute.
2. **Scalability**
    - The application should support a growing user base and an expanding recipe database without performance degradation.
3. **Security**
    - User data, including uploaded images and preferences, must be securely stored and processed.

•**Technical Stack:**

Programming Languages: Python
Frameworks/Libraries: Streamlit, OpenCV, Transformers, PIL, EasyOCR, Tesseract
Databases: PostgreSQL
Tools/Platforms: Visual Studio Code, Machine Learning (Pretrained model), OCR

•Architecture/Design:

System Architecture

- **High-Level Components:**
    - **User Interface (UI):** Built using Streamlit for user interactions.
    - **Image Processing & OCR Module:** Uses OpenCV, Tesseract, and EasyOCR to process images and extract text.
    - **Ingredient Recognition Model:** A machine learning model implemented with Transformers to identify ingredients from images.
    - **Database:** PostgreSQL for storing recognized ingredients, recipes, and user data.
    - **Recipe Recommendation Engine:** Matches identified ingredients to recipes using collaborative or content-based filtering.
    - **Instruction Generation:** Utilizes OpenAI's GPT-3 for creating step-by-step cooking instructions.

Design Decisions

- **Trade-offs:**
    - Used Python for its ML and OCR libraries but limited native multithreading capabilities. Addressed with optimized library usage.
    - Selected Streamlit for rapid prototyping, sacrificing some UI customization.

•Development:

Technologies and Frameworks

- **Programming Language:** Python
- **Libraries:** Streamlit, OpenCV, Transformers, PIL, EasyOCR, and Tesseract
- **Database:** PostgreSQL

Coding Standards

- Followed PEP 8 for Python coding standards.
- Used meaningful variable names, modular functions, and error handling practices.
- Included Comments in the code for easy understanding.

Challenges and Solutions

- **Challenge:** OCR misread text in noisy images.
  Solution: Applied image preprocessing techniques like resizing, grayscale conversion, and noise removal using OpenCV.

•Testing:

Testing Approach

- **Unit Tests:** Tested individual functions like image preprocessing, ingredient recognition, and database queries.
- **Integration Tests:** Verified seamless interaction between the OCR module, ML model, and database.
- **System Tests:** Tested end-to-end workflows, from image upload to recipe generation.

Results

- Resolved bugs in text extraction, database queries, and recipe ranking.
- Achieved better accuracy in ingredient recognition tests on a sample images

•Deployment :

Deployment Process

1. **Environment Setup:** Install dependencies from `requirements.txt`.

```
# Requirements

python==3.9
torch==2.0.1
transformers==4.32.0
Pillow==9.5.0
pytesseract==0.3.10
opencv-python==4.8.0
numpy==1.24.3
easyocr==1.7.0
openai==0.27.0
psycopg2-binary==2.9.3
```

```
# Front end

Streamlit

# Back end

PostgreSQL

# Operating System

Windows (any suitable OS can be used)

# IDE

Visual Studio Code (as per your preference)
```

2. **Database Initialization:** Load PostgreSQL database schema and seed recipe data.
3. **Application Launch:** Run the Streamlit app locally.

Automation

- Deployment scripts using Python and Bash automate the setup and launch processes.

Deployment Instructions

- Local Deployment: Use `streamlit run app.py` to start the application.

•User Guide:

Application Setup

1. Install Visual Studio Code and PostgreSQL.
2. Install dependencies.
3. Run the application using `streamlit run app.py`.

Using the Application

1. Upload images or input ingredient names.
2. View recognized ingredients and recommended recipes.
3. View the saved recipes that were generated previously.

Troubleshooting Tips

- If OCR fails, ensure the uploaded image is clear and in good lighting and make sure grouped images are not uploaded.
- Verify database connectivity if recipes are not displayed.

• Conclusion :

Outcomes and Achievements

- Successfully integrated ML and OCR for accurate ingredient recognition.
- Generated personalized recipe recommendations and detailed cooking instructions.

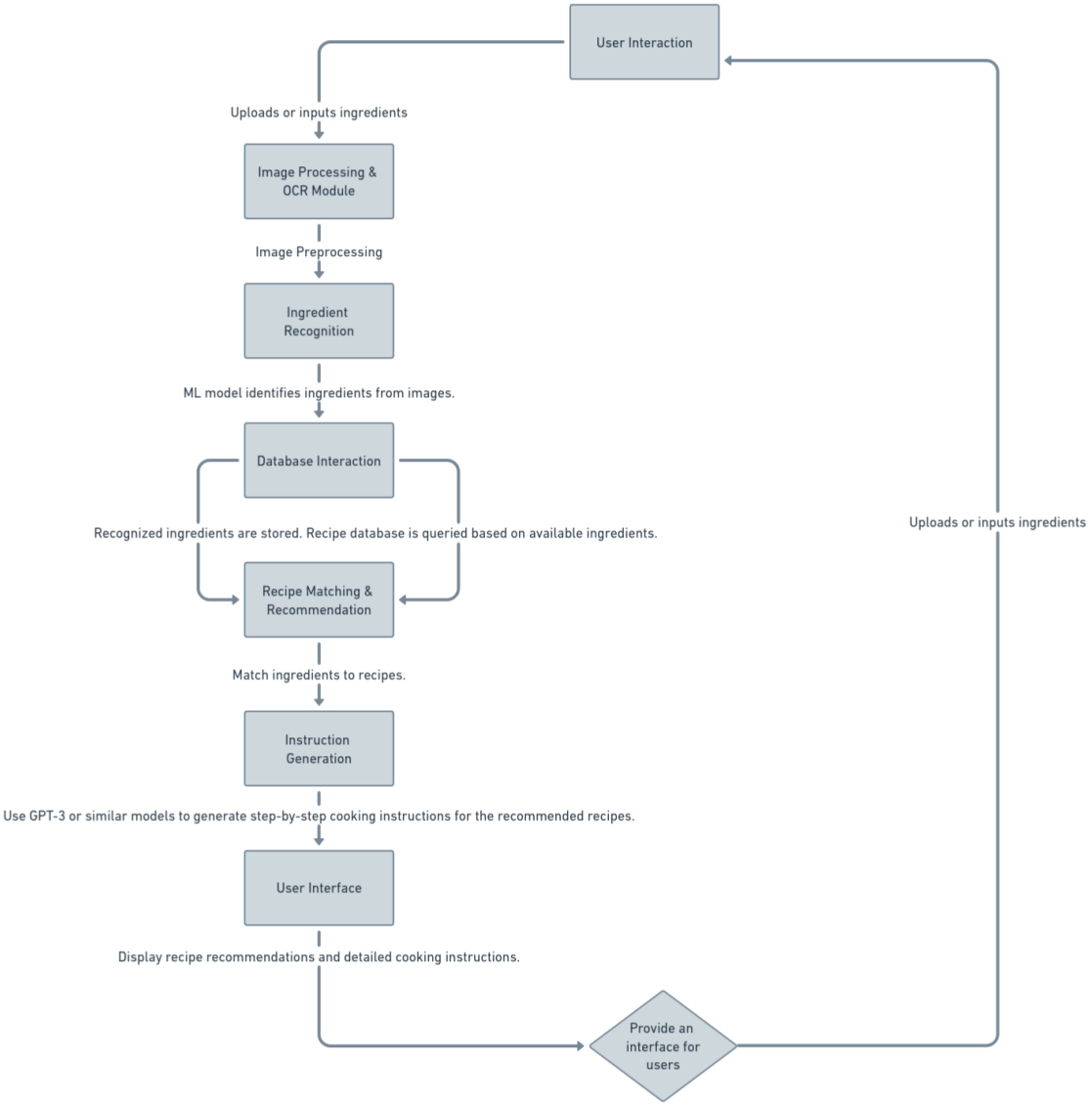
- Delivered a user-friendly interface for an enhanced user experience.

Reflections

- **Lessons Learned:** Importance of preprocessing for OCR accuracy and database optimization for faster queries.
- **Areas for Improvement:** Add multi-language support and expand the recipe database for global cuisines.

*** Appendices:**

Diagrams :



Code Snippets :

Image preprocessing and OCR

```
# Preprocessing function for image enhancement (OCR)
def preprocess_image(image):
    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    blurred_image = cv2.GaussianBlur(gray_image, (5, 5), 0)
    thresh_image = cv2.adaptiveThreshold(blurred_image, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
                                         cv2.THRESH_BINARY, 11, 2)
    kernel = np.ones((3, 3), np.uint8)
    processed_image = cv2.morphologyEx(thresh_image, cv2.MORPH_CLOSE, kernel)
    return processed_image

# Tesseract OCR Function
def ocr_with_tesseract(image):
    text = pytesseract.image_to_string(image)
    return text

# EasyOCR Function
def ocr_with_easyocr(image_path):
    reader = easyocr.Reader(['en'])
    result = reader.readtext(image_path, detail=0)
    return result
```

GPT

```
# GPT API call
def ask_gpt(prompt):
    gpt_key = database_connection.get_gpt_key()
    openai.api_key = gpt_key

    response = openai.ChatCompletion.create(
        model="gpt-3.5-turbo",
       messages=[
    {"role": "system", "content": "You are an assistant that identifies ingredients,
    {"role": "user", "content": prompt}
],
        max_tokens=150
    )

    return response.choices[0].message['content']
```

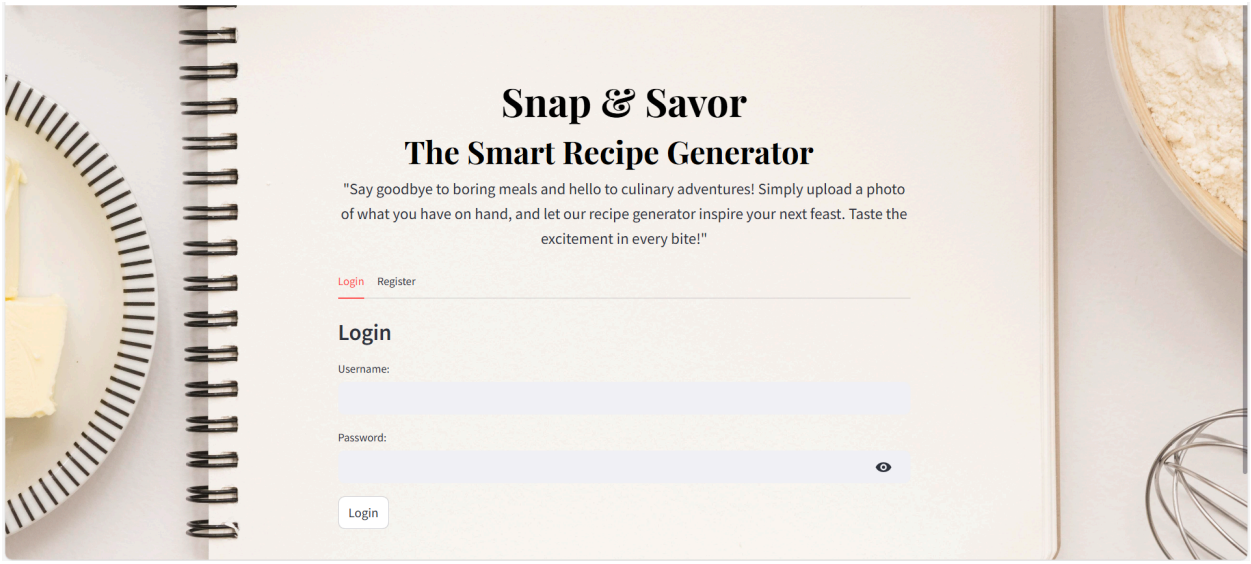
Pretrained model

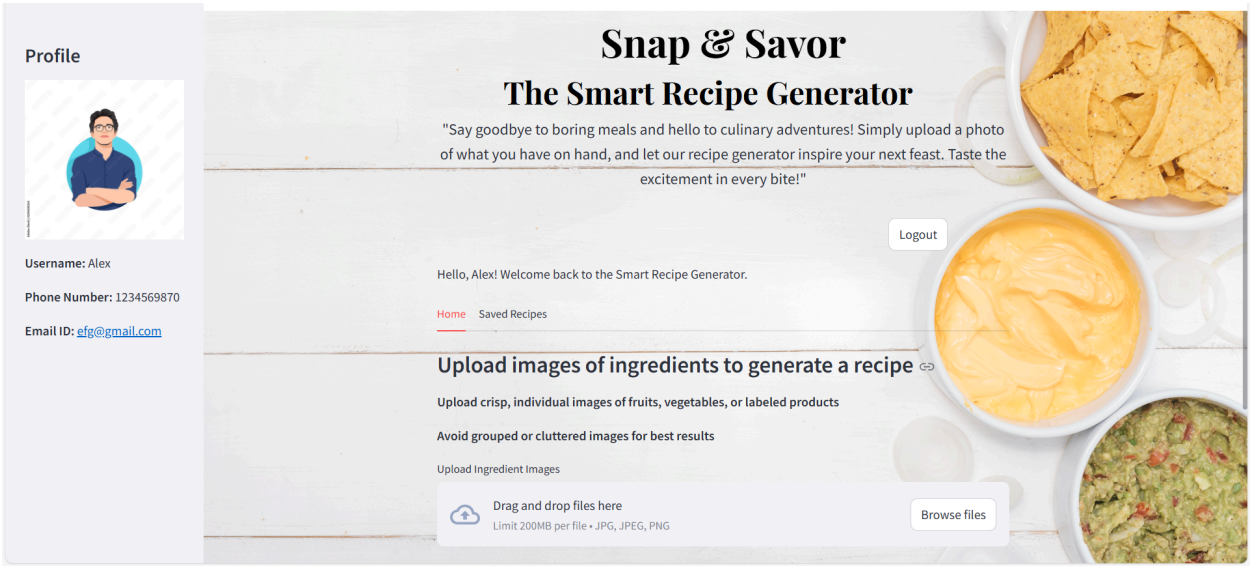
```
# Fruits/Vegetables detection using pre-trained model
def classify_fruit_vegetable(image):
    model = AutoModelForImageClassification.from_pretrained("jazzmacedo/fruits-and-vegetables-detector-36")
    labels = list(model.config.id2label.values())

    preprocess = transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
    ])

    pil_image = Image.fromarray(image)
    input_tensor = preprocess(pil_image).unsqueeze(0)
    outputs = model(input_tensor)
    predicted_idx = torch.argmax(outputs.logits, dim=1).item()
    predicted_label = labels[predicted_idx]
    return predicted_label
```

Output Screenshots :





References :

Tesseract OCR Documentation – Official documentation for Tesseract OCR: https://github.com/tesseract-ocr/tesseract

EasyOCR Documentation – EasyOCR library usage guide: https://github.com/JaidedAI/EasyOCR

OpenAI GPT-3 Documentation – For natural language generation: https://platform.openai.com/docs/

Streamlit Documentation – Framework documentation for building interactive user interfaces: https://docs.streamlit.io/