International Journal of Trend in Scientific Research and Development (IJTSRD)
Special Issue on Advancements and Emerging Trends in Computer Applications - Innovations, Challenges, and Future Prospects
Available Online: www.ijtsrd.com e-ISSN: 2456 – 6470

Image Extraction Tool

Akanksha Padmakar Khandare

PG Student, Department of Computer Application, G. H. Raisoni University, Amravati, Maharashtra, India

ABSTRACT

This project is a web-based application developed using the Flask framework, aimed at streamlining document processing and user management. It provides a secure and intuitive platform where users can register, log in, and efficiently manage their documents. The application features a user-friendly interface that simplifies document uploads, organization, and access. With an emphasis on usability and scalability, this system is ideal for environments requiring structured document handling and user authentication.

KEYWORDS: *User Authentication, Document Upload and Processing, Dashboard, Error Handling and Logging, Database Integration.*

I. INTRODUCTION

This project is a web application built using Flask, designed to facilitate document processing and user management. The application allows users to register, log in, and manage their documents through a user-friendly interface.

Key Features:

1. User Authentication:
   - Users can register with a unique username, email, and phone number. Passwords are securely hashed for protection.
   - The application supports user login and logout functionalities, ensuring that only authenticated users can access certain features.
2. Document Upload and Processing:
   - Users can upload various document types, which are then processed to extract images or relevant data.
   - The processed files are zipped and made available for download, enhancing the user experience by allowing easy access to extracted content.
3. Dashboard:
   - After logging in, users are directed to a dashboard where they can manage their documents and access other features of the application.
4. Error Handling and Logging:
   - The application includes robust error handling and logging mechanisms to track user actions and any issues that may arise during document processing.
5. Database Integration:
   - User data is stored in a SQLite database, ensuring persistence and easy management of user information.

II. Scope of System

The scope of the document processing and user management system encompasses the following key functionalities and features:

1. User Management
   - User Registration: Allows users to create accounts with unique usernames, emails, and passwords.
   - User Authentication: Implements secure login and logout functionalities to protect user accounts.
   - Profile Management: Enables users to update their profile information, including contact details.
2. Document Processing
   - File Upload: Supports uploading various document types (e.g., PDF, DOCX, images) for processing.
   - Document Processing: Automatically extracts relevant data or images from uploaded documents.
   - Download Processed Files: Provides users with the ability to download processed files in a convenient format (e.g., ZIP).
3. Error Handling and Logging
   - Error Management: Implements error handling mechanisms to provide feedback to users in case of issues during document processing.
   - Logging: Maintains logs of user actions and system events for monitoring and troubleshooting purposes. Target Users

Exclusuion

- Advanced Document Editing: The system will not include features for editing documents directly within the application.
- Integration with Third-Party Services: The initial scope does not cover integration with external APIs or services for additional functionalities.
- Multi-Language Support: The application will initially support only one language (English) for user interfaces and messages Constraints.

III. Hardware and Software Requirements

A. Hardware Requirements

1. Server-Side

- Processor: Minimum dual-core processor (Intel i3 or equivalent) for basic operations; quad-core recommended for better performance.
- RAM: At least 4 GB of RAM; 8 GB or more is recommended for handling multiple concurrent users and document processing tasks.
- Storage: Minimum 20 GB of available disk space for application files, database storage, and uploaded documents. SSD storage is preferred for faster access times.
- Network: Reliable internet connection for cloud-hosted solutions; local network access for on-premise deployments.

2. Client-Side

- Processor: Any modern processor capable of running a web browser.
- RAM: Minimum 2 GB of RAM; 4 GB or more is

recommended for optimal performance.

- Storage: Sufficient storage for browser cache and temporary files.

B. Software Requirements

1. Server-Side

- Operating System:
- Linux (Ubuntu, CentOS) or Windows Server for hosting the application.
- Web Server:
- Flask development server for local testing; Nginx or Apache for production deployment.
- Database:
- SQLite for local development; PostgreSQL or MySQL for production environments.
- Programming Language:
- Python 3.6 or higher.
- Frameworks and Libraries: o Flask, SQL Alchemy, Flask-Login, and any other necessary libraries for document processing (e.g., PyPDF2, python-docx).

2. Client-Side

- Web Browser:
- Latest versions of Chrome, Firefox, Safari, or Edge for optimal compatibility and performance.
- JavaScript:
- Enabled in the browser for interactive features of the web application.

C. Development Environment

- IDE/Text Editor:
- Visual Studio Code, PyCharm, or any preferred text editor for Python development.
- Version Control:
- Git for source code management and collaboration.

Fig.1 ER Diagram

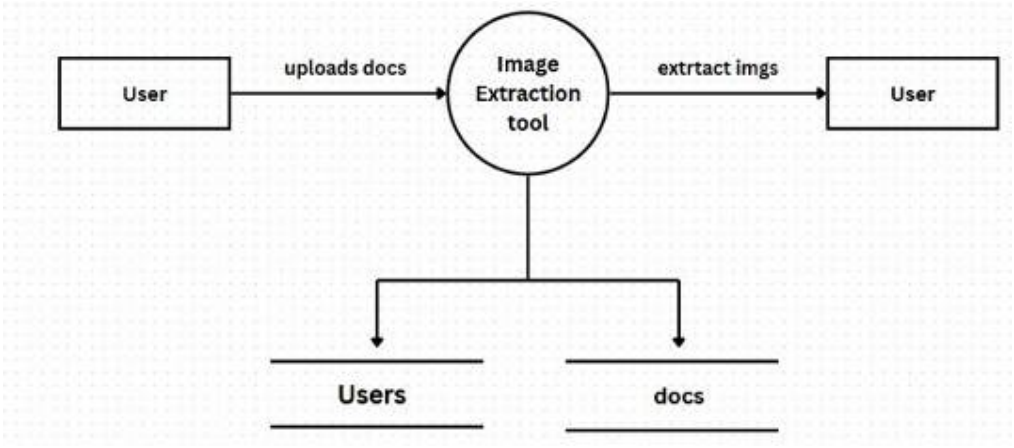
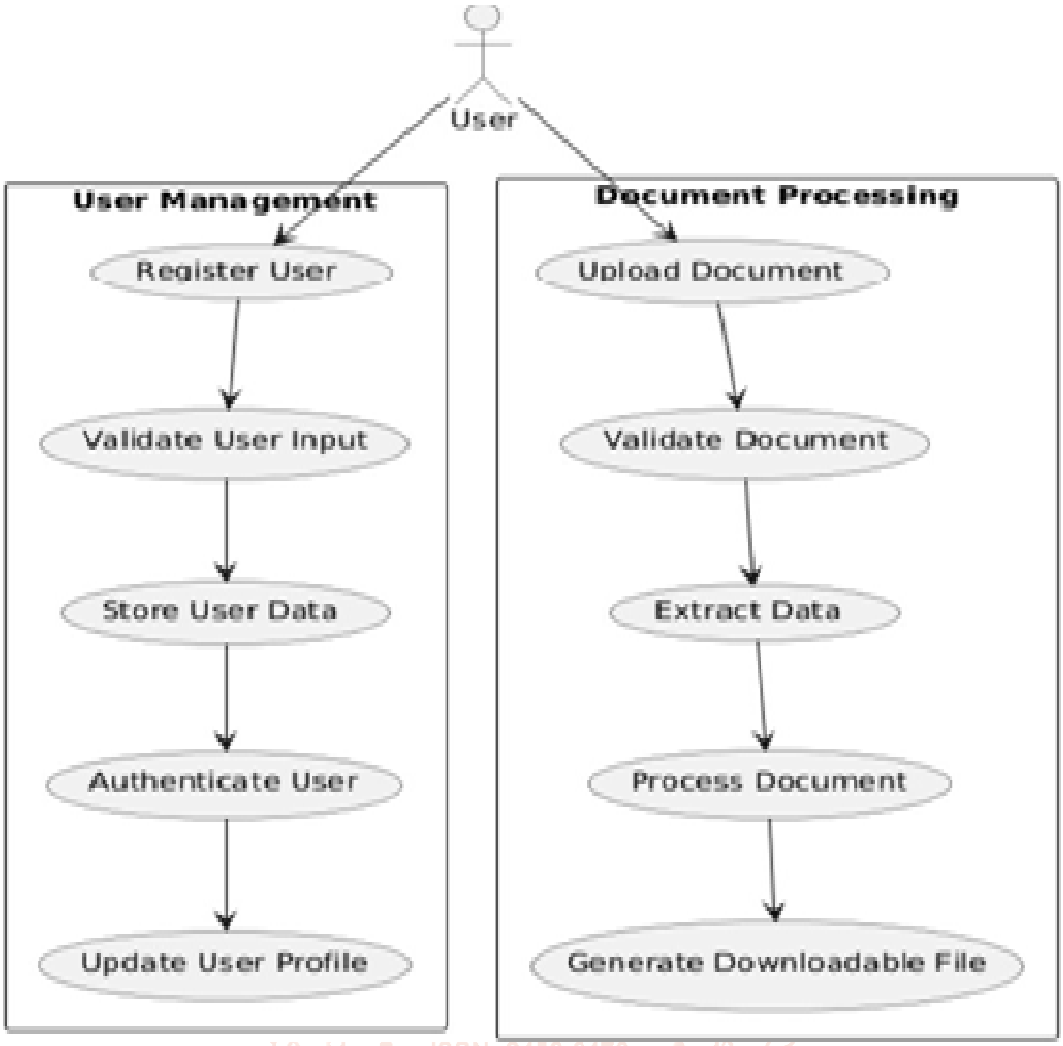


Fig.2 DFD Diagram

Level 1



IV. Input Output Screens

Input

```python
app.py
import os
from flask import Flask, request, send_file, render_template, redirect, url_for, flash
from flask_sqlalchemy import SQLAlchemy
from flask_login import LoginManager, UserMixin, login_user, login_required, logout_user, current_user
from werkzeug.security import generate_password_hash, check_password_hash
from document_processing import process_document
import zipfile
from io import BytesIO
import logging
from concurrent.futures import ThreadPoolExecutor, as_completed

app = Flask(__name__)
app.config['SECRET_KEY'] = 'your_secret_key'
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///users.db'
db = SQLAlchemy(app)
login_manager = LoginManager()
login_manager.init_app(app)
login_manager.login_view = 'login'

# User model
class User(UserMixin, db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(150), unique=True, nullable=False)
    email = db.Column(db.String(150), unique=True, nullable=False)
    phone = db.Column(db.String(15), nullable=False)
    password = db.Column(db.String(150), nullable=False)


@login_manager.user_loader
def load_user(user_id):
    return User.query.get(int(user_id))

# Routes
@app.route('/')
def index():
    return render_template('index.html')
```

app.py

```python
@app.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        username = request.form['username']
        email = request.form['email']
        phone = request.form['phone']
        password = request.form['password']
        hashed_password = generate_password_hash(password, method='pbkdf2:sha256')

        # Check if username or email already exists
        if User.query.filter_by(username=username).first():
            flash('Username already exists!', 'danger')
            return redirect(url_for('register'))

        if User.query.filter_by(email=email).first():
            flash('Email already registered!', 'danger')
            return redirect(url_for('register'))

        # Create a new user and save to database
        new_user = User(username=username, email=email, phone=phone, password=hashed_password)
        db.session.add(new_user)
        db.session.commit()

        flash('Registration successful! Please log in.', 'success')
        return redirect(url_for('login'))

    return render_template('register.html')

@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']
        user = User.query.filter_by(username=username).first()

        if user and check_password_hash(user.password, password):
            login_user(user)
```

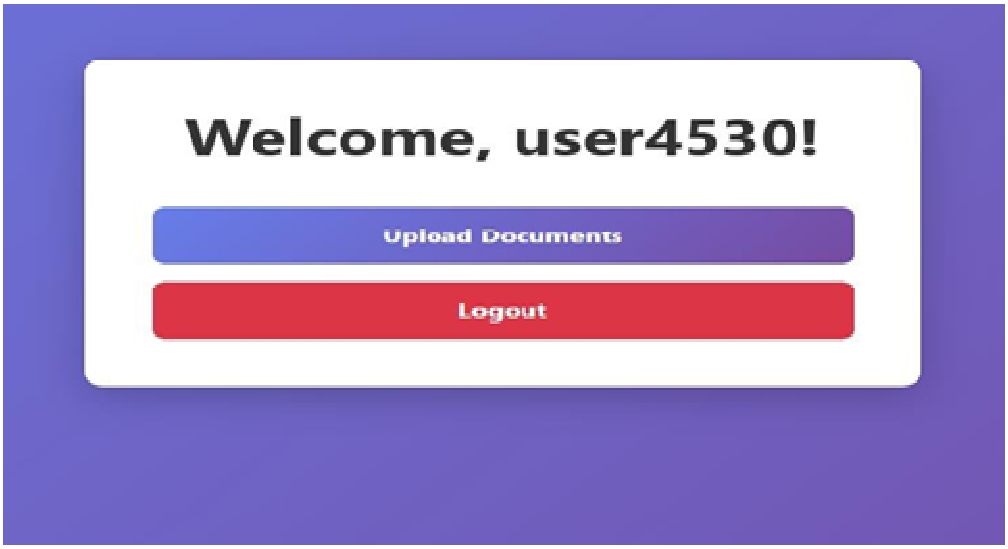
app.py

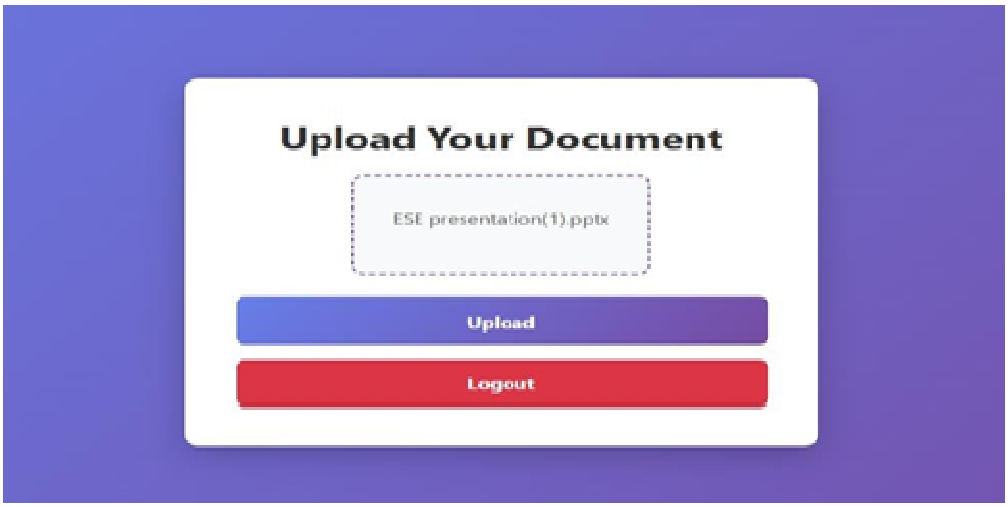
```python
def login():
            return redirect(url_for('dashboard'))
        else:
            flash('Invalid credentials', 'danger')

    return render_template('login.html')

@app.route('/dashboard')
@login_required
def dashboard():
    return render_template('dashboard.html', username=current_user.username)

@app.route('/logout')
@login_required
def logout():
    logout_user()
    flash('Logged out successfully!', 'success')
    return redirect(url_for('login'))

@app.route('/upload', methods=['GET', 'POST'])
@login_required
def upload():
    if request.method == 'GET':
        return render_template('upload.html')  # Ensure 'upload.html' exists

    files = request.files.getlist('files')

    if not files or files[0].filename == '':
        return "No files uploaded", 400

    logging.info(f"Uploaded files: {[file.filename for file in files]}")
    zip_stream = BytesIO()
    zip_filename = 'extracted_images.zip'

    with zipfile.ZipFile(zip_stream, 'w') as zipf:
        if len(files) == 1:
```

```python
def upload():
            file = files[0]
            try:
                img_data, folder_name, base_name = process_document(file, single_file=True)
                logging.info(f"Base name for single file ZIP: {base_name}")
                for img_filename, img_bytes in img_data:
                    img_filename = f"{folder_name}{img_filename.split('/')[-1]}"
                    zipf.writestr(img_filename, img_bytes)
                zip_filename = f"{base_name}.zip"
                logging.info(f"Zip filename set to: {zip_filename}")
            except Exception as e:
                logging.error(f"Error processing {file.filename}: {e}")
                zip_filename = 'extracted_images.zip'
        else:
            with ThreadPoolExecutor() as executor:
                futures = {executor.submit(process_document, file): file for file in files}
                for future in as_completed(futures):
                    file = futures[future]
                    try:
                        img_data, folder_name, _ = future.result()
                        for img_filename, img_bytes in img_data:
                            img_filename = f"{folder_name}{img_filename.split('/')[-1]}"
                            zipf.writestr(img_filename, img_bytes)
                    except Exception as e:
                        logging.error(f"Error processing {file.filename}: {e}")

    zip_stream.seek(0)
    logging.info(f"Final zip filename before sending: '{zip_filename}'")

    response = send_file(zip_stream, as_attachment=True, download_name=zip_filename, mimetype='application/zip')
    response.headers["Content-Disposition"] = f"attachment; filename={zip_filename}"
    return response


    # (Your existing file processing logic here...)
```

```python
    # (Your existing file processing logic here...)


if __name__ == "__main__":
    with app.app_context():
        db.create_all()  # Ensure database tables exist
    logging.basicConfig(level=logging.INFO)
    app.run(debug=True)
```

V. OUTPUT





VI. Conclusion

The Document Processing and User Management System has been designed to streamline user account management and document processing. By leveraging modern web technologies and best practices, the system provides a user-friendly interface that enhances the experience for individuals and organizations a like.

Key Achievements

- User Management: Successfully implements user registration, authentication, and profile management, ensuring secure access to system features.
- Document Processing: Enables users to upload, process, and download documents efficiently, supporting multiple file types and essential functionalities.
- Error Handling and Logging: Integrates robust error handling and logging mechanisms to ensure transparency and facilitate troubleshooting.

Future Directions

While the current implementation meets the initial project goals, there are opportunities for further enhancements, including:

- Expanding support for additional file formats.
- Integrating advanced processing features such as Optical Character Recognition (OCR).
- Optimizing the user interface for better accessibility and ease of use.
- Developing a mobile application to enhance user engagement and accessibility. Final Thoughts

The **Document Processing and User Management System** marks a significant step in automating and improving document handling processes. With continuous development and enhancements, the system is well-positioned to adapt to evolving user needs and technological , ensuring its relevance and effectiveness in the future.

References

[1] Flask Web Development – Miguel Grinberg, O'Reilly, 2018.

[2] Learning SQLAlchemy – Mark Ramm, Jason Myers, O'Reilly, 2015.

[3] Python for Data Analysis – Wes McKinney, O'Reilly, 2018.

[4] Web Development with Python and Flask – Daniel Gasienica, Packt, 2019.

[5] Flask Docs: flask.palletsprojects.com

[6] SQLAlchemy Docs: docs.sqlalchemy.org

[7] Python Docs: docs.python.org

[8] W3Schools (File Handling): w3schools.com

[9] MDN Web Docs: developer.mozilla.org