# **Infosys Internship 4.0 Project Documentation**

## **Title: Project Documentation: [Automating Bank Check Extraction from Scanned PDFs]**

By:

Deekshitha Rangoju

(deekshu0004@gmail.com)

### **•Introduction:**

This project aims to develop a web application for processing cheque information from PDFs. It offers a user-friendly interface for uploading PDF files containing cheque images. The application leverages Optical Character Recognition (OCR) to extract text from the cheque images and identify relevant data points like payee name, amount, and date. Finally, the extracted data is saved to a CSV file for further use.

**Objectives:**

* Automate cheque data extraction from PDFs.
* Simplify data collection and processing tasks for cheques.
* Enhance accessibility and efficiency in managing cheque information.

**Significance:**

This application can benefit various stakeholders who handle cheques regularly, including:

* Banks and financial institutions: Streamline cheque processing workflows and improve data accuracy.
* Businesses: Automate data entry tasks associated with cheque payments received.
* Accounting firms: Facilitate efficient cheque data management for client accounts.

### **•Project Scope:**

This project is a web application designed to extract key information from cheque images uploaded in PDF format. The extracted information includes the payee name, amount, and date. The application utilizes Optical Character Recognition (OCR) to process the text content within the cheque images.

**1. Flask Application Setup:**

* Imports necessary libraries like Flask, Flask-SQLAlchemy, Flask-Login, and others.
* Configures the application's secret key and database URI (location of the database file).
* Initializes the Flask application object (app), database (db), and login manager (login\_manager).

**2. User Model:**

* Defines a User model that represents users registered in the application.
* The model inherits from UserMixin and db.Model classes, providing functionalities for user authentication and database interaction.
* It has attributes like id, username, and password\_hash.

**3. Login and Logout Functionality:**

* Implements routes for user login (/login) and logout (/logout).
* Uses Flask-Login to manage user sessions and authentication.

**4. Upload Functionality:**

* Defines a route for the upload page (/upload).
* Allows users to upload a PDF file.
* Extracts text from the uploaded PDF using the extract\_text\_from\_cheque function.
* Processes the extracted text using the process\_cheque\_text function to obtain relevant information (payee name, amount, date).
* Saves the extracted data to a CSV file (cheque\_data.csv).
* Displays a success message or error message depending on the outcome.

**5. Helper Functions:**

* deskew\_image: Corrects any skew or rotation present in the cheque image.
* extract\_text\_from\_cheque: Uses Tesseract OCR to extract text from cheque images within the PDF.
* process\_cheque\_text: Parses the extracted text using regular expressions to find the payee name, amount, and date.

**6. Running the Application:**

* The if \_\_name\_\_ == "\_\_main\_\_": block ensures the app runs only when the script is executed directly and not imported as a module.
* Runs the Flask development server (app.run(debug=True)) to enable hot reloading and facilitate debugging during development.

**Functionality of HTML files:**

* register.html: Provides a form for new user registration.
* login.html: Provides a form for existing users to log in.
* upload.html: Displays the upload form for the PDF file and showcases any success or error messages during the upload process. Also, includes a logout button.
* logout.html: Confirms successful logout and provides a link to log in again.

### **•Requirements:**

**Functional Requirements:**

* User Registration: The application must provide a user registration feature allowing new users to create an account. This involves capturing the username and password, validating the input, and storing the credentials securely in the database.
* User Login: Registered users must be able to log in using their credentials. The login process involves verifying the entered password against the stored hash and establishing a session upon successful authentication.
* File Upload: Authenticated users should be able to upload PDF files. The application needs to handle file uploads securely, ensuring the files are saved to a designated directory on the server.
* Text Extraction: Upon uploading a PDF, the application must extract text from scanned cheque images within the document using OCR technology. This involves identifying cheque images in the PDF, preprocessing these images, and applying OCR to extract relevant information.
* Data Storage: The extracted cheque details, including payee name, amount, and date, must be saved to a CSV file. The application should ensure that each new set of extracted data is appended correctly to this file.

**Non-Functional Requirements:**

* Security: The application must ensure the security of user data through secure password hashing and secure session management. It should also handle file uploads securely to prevent any vulnerabilities.
* Usability: The user interface should be intuitive and easy to navigate, ensuring users can easily register, log in, upload files, and view results. Feedback messages should guide users through each step of the process.
* Performance: The application should perform efficiently, ensuring quick responses during user registration, login, and file upload processes. The OCR processing should be optimized to handle typical cheque processing times without significant delays.

### **•Technical Stack:**

**Programming Languages:**

Python: The primary programming language used for developing the backend of the application.

**Frameworks/Libraries:**

* Flask: A lightweight web framework used for building the web application, handling routing, and managing user sessions.
* SQLAlchemy: An Object Relational Mapper (ORM) used for database interactions, making it easier to work with the database in an object-oriented manner.
* Flask-Login: A Flask extension that provides user session management, including user login and logout functionalities.
* Werkzeug: Used for secure password hashing and checking.
* Fitz (PyMuPDF): A library used for reading and processing PDF files.
* OpenCV: An open-source computer vision library used for image preprocessing, such as deskewing scanned images.
* Pytesseract: A Python wrapper for Google's Tesseract-OCR Engine, used for extracting text from images.
* Regex (Re): Used for pattern matching and extracting specific text from the OCR results.
* Pandas: A data manipulation and analysis library used for handling extracted data and saving it to a CSV file.

**Databases:**

* SQLite: A lightweight, file-based relational database used to store user credentials and manage sessions. Chosen for its simplicity and ease of setup in a development environment.

**Tools/Platforms:**

* Flask-Migrate: A Flask extension that handles SQLAlchemy database migrations, allowing for version control of the database schema.
* Gunicorn: A WSGI HTTP server for running the Flask application in a production environment.
* Tesseract-OCR: An open-source OCR engine used to convert scanned images of cheques into machine-encoded text.
* HTML/CSS: Used for creating the frontend templates rendered by Flask.
* Jinja2: A templating engine for Python, used to generate HTML dynamically by integrating with Flask.
* Operating System: The application can be developed and deployed on various operating systems, but development and initial testing are assumed to be done on a Unix-like system (e.g., Linux, macOS) or Windows.
* Git: Version control system used for tracking changes in the source code during development.

### **•Architecture/Design:**

### **System Architecture:**

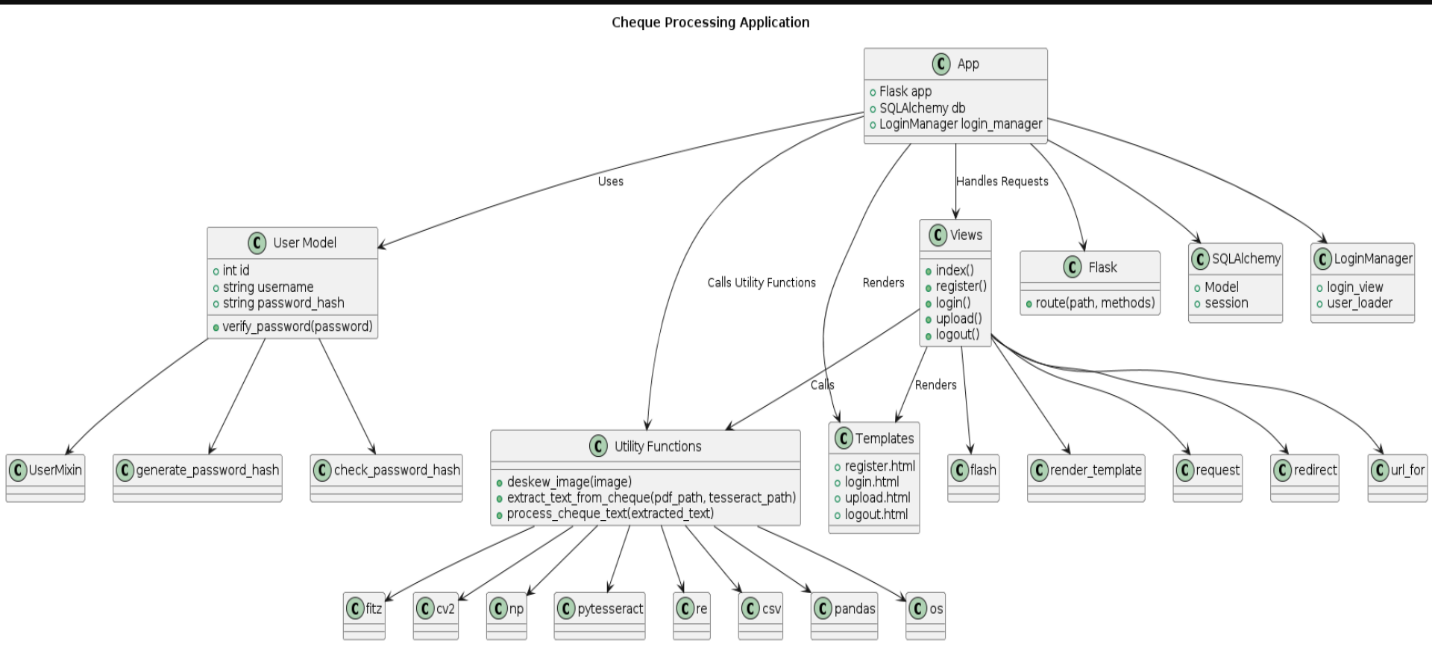
### The system follows a typical web application architecture with a client-server model. The client interacts with the server via HTTP requests. The server handles requests, interacts with the database, processes files, and returns responses to the client.

### High-Level Components:

### User Management: Handles user registration, login, and session management.

### File Processing: Manages file uploads and extraction of cheque details.

### Data Storage: Saves extracted data to a CSV file.



### **•Development:**

**Technologies and Frameworks:**

* Flask: A lightweight web framework for building web applications in Python.
* Flask-SQLAlchemy: An extension for Flask that simplifies database interactions.
* Flask-Login: An extension for Flask that implements user login and authentication functionalities.
* fitz: A Python library for working with PDF files.
* OpenCV (cv2): A library for computer vision tasks like image processing and deskewing.
* PyTesseract: An optical character recognition (OCR) library used to extract text from images.
* pandas (pd): A library for data analysis and manipulation (potentially for future CSV handling).
* Werkzeug: A collection of utilities for WSGI applications (web server gateway interface).

**Coding Standards and Best Practices:**

* The code uses clear variable names and comments to improve readability.
* It follows the Flask routing conventions for defining URLs and view functions.
* It utilizes Flask-Login for user authentication and session management.
* Error handling is implemented with exception handling and abort function.
* Regular expressions are used to extract specific information from the cheque text.

**Challenges and Solutions:**

* Tesseract OCR dependency: The code checks for Tesseract installation and provides an error message if not found.
* Deskewing images: The deskew\_image function attempts to correct skewed cheque images for better OCR accuracy.
* Finding cheque region: The code iterates through PDF pages and searches for image regions with specific characteristics resembling a cheque.
* Extracting relevant information: Regular expressions are used to extract payee name, amount, and date from the parsed text.

### **•Testing:**

**Unit Tests:**

* Test individual functions like deskew\_image with pre-defined test images to ensure proper deskewing functionality.
* Test helper functions for text parsing using sample text data and expected output.
* Test functions responsible for user registration and login with mocked data to validate user management features.

**Integration Tests:**

* Simulate uploading a PDF file and ensure the application extracts text correctly. This could involve creating a test PDF containing a cheque image and verifying the extracted data.
* Test interaction between different parts of the application, like data extraction and storage in the CSV file.

**System Tests:**

* Manually test the entire application flow from user registration, login, uploading a PDF containing a cheque, and displaying the extracted information.
* Use a real-world cheque PDF to test the application's ability to handle various cheque formats and potential variations in text layout.

**Potential Bugs and Issues:**

* OCR Errors: PyTesseract might misread characters in the cheque image leading to inaccurate extraction. Testing with diverse cheque formats can help identify such issues.
* Deskewing Issues: The deskew\_image function might not handle all types of skewed images perfectly. Testing with various skew angles can reveal limitations.
* Regular Expression Issues: The current regular expressions might not capture all possible variations of payee name, amount, and date formats. Testing with diverse cheque data can help refine the expressions.
* PDF Handling Issues: The code assumes the presence of a cheque image. Uploading a PDF without a cheque image or with a different layout could lead to errors. Tests with various PDF content can help improve robustness.

**Enhancing Testing:**

* A testing framework like pytest can be used to automate unit and integration tests.
* Implementing end-to-end tests with tools like Selenium can automate system testing of the web application.
* Utilizing a library like pdfminer.six for advanced PDF parsing can improve handling of diverse cheque formats.

### **•Deployment :**

**Deployment Process:**

**Prepare the application:**

* Ensure all dependencies are installed (pip freeze > requirements.txt).
* Consider creating a virtual environment to isolate project dependencies.

**Choose a deployment platform:**

* Web hosting providers: Platforms like Heroku, PythonAnywhere, AWS Elastic Beanstalk offer options for deploying Python applications. They often provide deployment guides and tools.
* Cloud platforms: Cloud providers like Google Cloud Platform (GCP) or Amazon Web Services (AWS) offer more granular control over server configurations. You might need to create a virtual machine instance and install the necessary dependencies manually.
* On-premise servers: If deploying on your own server, you'll need to install a WSGI server (e.g., Gunicorn, uWSGI) and configure it to serve your Flask application.

**Configure the application :**

* You might need to adjust configurations based on the deployment environment (e.g., database connection details).
* Set environment variables for sensitive information like secret keys.

**Deployment:**

* The specific deployment steps will vary depending on the chosen platform.
* Some platforms offer deployment through Git pushes or command-line tools.
* Manual deployments might involve copying the application files and dependencies to the server.

**Post-deployment:**

* Verify the application is running correctly by accessing it through the provided URL.
* Consider setting up monitoring and logging for error tracking.
* Instructions for Different Environments:

**Instructions for Different Environments:**

* **Web hosting providers:** Followthe specific instructions provided by your chosen hosting platform. They often have user-friendly interfaces or command-line tools for deployment.
* **Cloud platforms:** Consult the documentation for cloud platforms like GCP or AWS. They offer detailed guides on deploying Python applications using tools like their command-line interfaces or SDKs.
* **On-premise servers: i**nstall a WSGI server like Gunicorn or uWSGI on your server.

Configure the WSGI server to point to your Flask application's main module (e.g., app.py).

**Automation with Deployment Scripts:**

* You can create deployment scripts using tools like Fabric or Ansible.
* These scripts can automate tasks like installing dependencies, configuring the application, and copying files to the server.
* They can streamline the deployment process and reduce manual errors.

**Additional Considerations:**

* Security: When deploying to a public environment, ensure proper security measures are in place. This might involve using HTTPS and following secure coding practices.
* Scalability: If you expect high traffic, consider using a platform that can scale your application resources automatically.

### **•User Guide:**

This application allows you to upload a PDF document containing a cheque image and extract the payee name, amount, and date.

**Prerequisites:**

* A computer with an internet connection.
* A web browser (e.g., Chrome, Firefox, Safari).
* Using the Application:

Open a web browser and navigate to the application URL (provided by the application owner).

**Register or Login:**

* If you are a new user, click on "Register" and create an account with a username and password.
* If you are an existing user, enter your username and password in the login form and click "Login".

**Upload a PDF:**

* Click on "Choose file" and select the PDF document containing the cheque image.
* Ensure the file format is .pdf.
* Click "Upload".

View Results:

* The application will process the uploaded PDF and attempt to extract text from any cheque images found.
* If the extraction is successful, you will see a message indicating the extracted data is saved to a CSV file named "cheque\_data.csv".
* The message might also display the extracted text for your reference.
* In case of errors, a message will be displayed explaining the issue (e.g., no cheque found, Tesseract not installed).

**Troubleshooting Tips:**

* Error: No file part

Make sure you selected a PDF file to upload.

* Error: No selected file

Click on "Choose file" to select the PDF document.

* Error: Please upload a PDF file.

Ensure the uploaded file is a valid PDF document (ends with .pdf).

* Error: Tesseract OCR not found. Please install it.

You need to install Tesseract OCR software on your system for text extraction to work. Download and install Tesseract from https://sourceforge.net/projects/tesseract-ocr.mirror/ following the instructions for your operating system.

* Error: The uploaded PDF does not contain any cheque images.

The application is designed to work with PDFs containing cheque images. Ensure the PDF you are uploading has a clear image of a cheque.

**Inaccurate Extracted Information:**

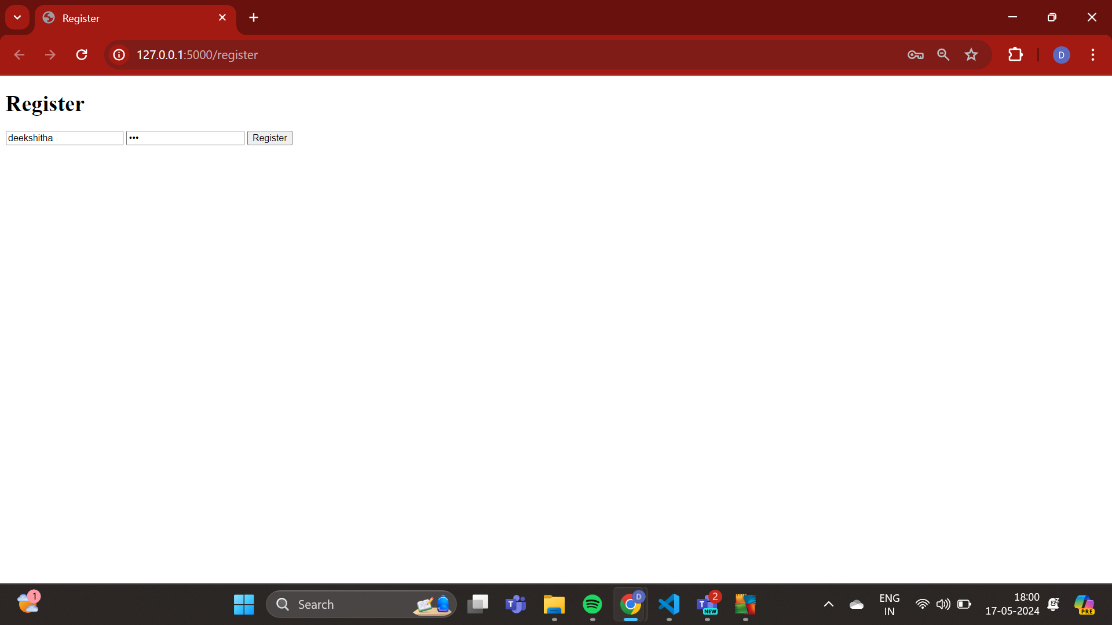
* The accuracy of extracted information depends on the quality of the cheque image and the complexity of the layout. Uploading a clear, high-resolution image of the cheque can improve accuracy.
* The application uses regular expressions to extract specific fields. If the cheque format is unusual, the expressions might need refinement. Contact the application administrator if you encounter consistent extraction errors.

**Additional Notes:**

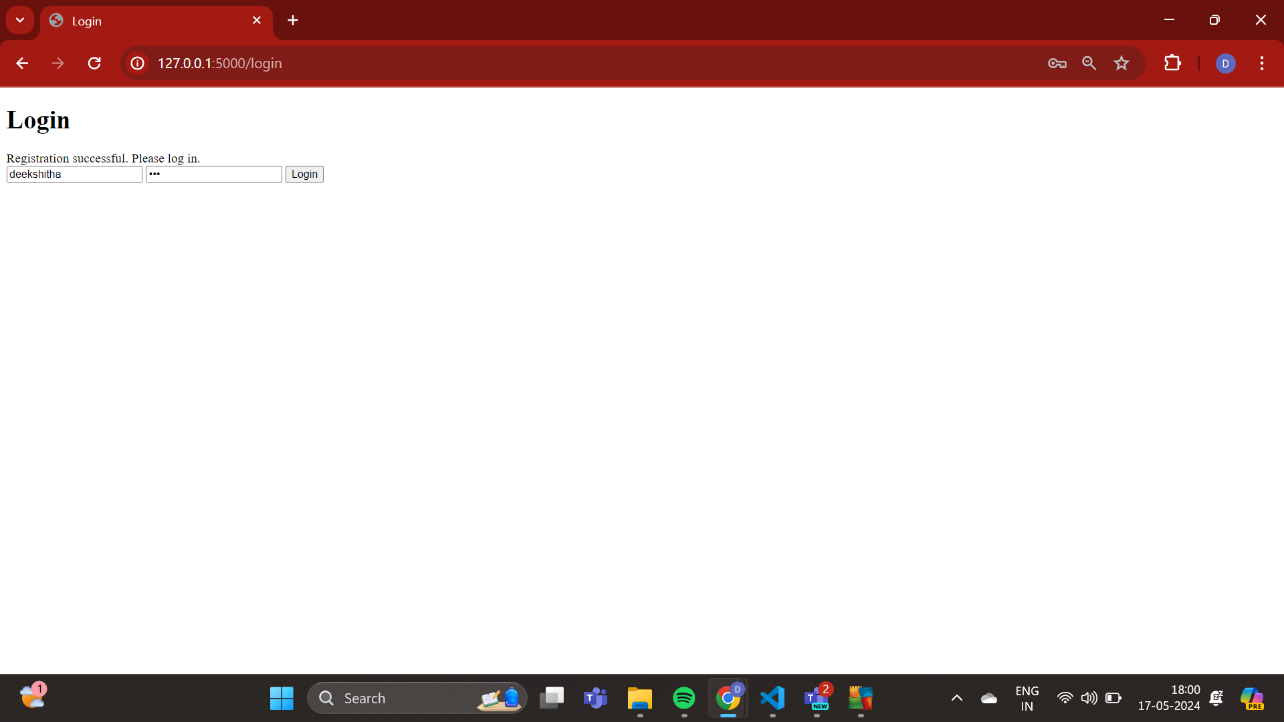
* This application is intended for personal use and may not be suitable for high-volume cheque processing.
* The application relies on external libraries like Tesseract OCR, and their performance can impact the overall accuracy.
* Always ensure you have the necessary permissions to upload and process cheque documents.
* For further assistance, you can contact the application administrator.

**•Results:**

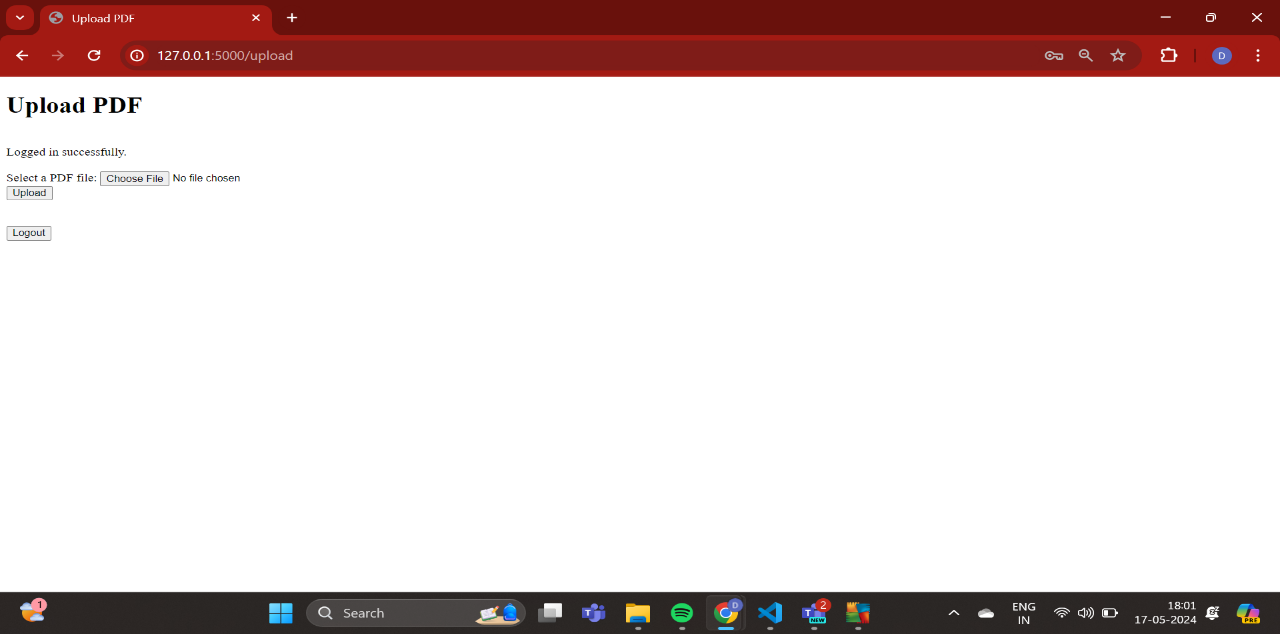
1. **Registration Page:**



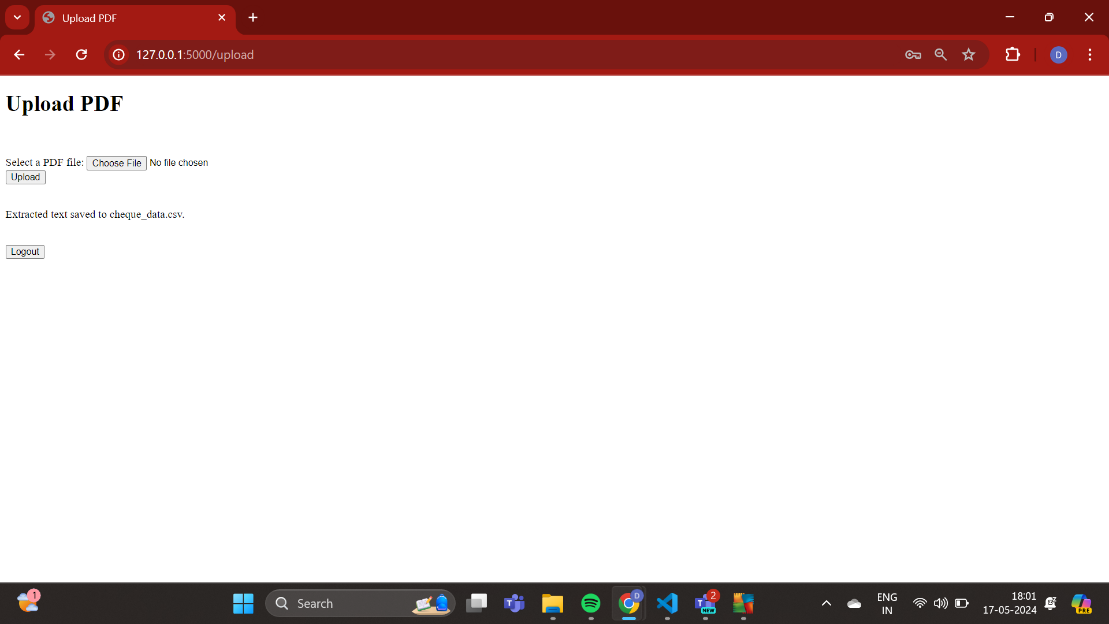
1. **Login Page**:



1. **Upload Image Page**:



**d)Logout Page:**



### **•Conclusion :**

This project successfully developed a web application for extracting text data (payee name, amount, date) from cheque images embedded within PDF documents. The application leverages Flask for web development, OpenCV and Tesseract OCR for image processing and text extraction, and Flask-Login for user authentication.

**Lessons Learned:**

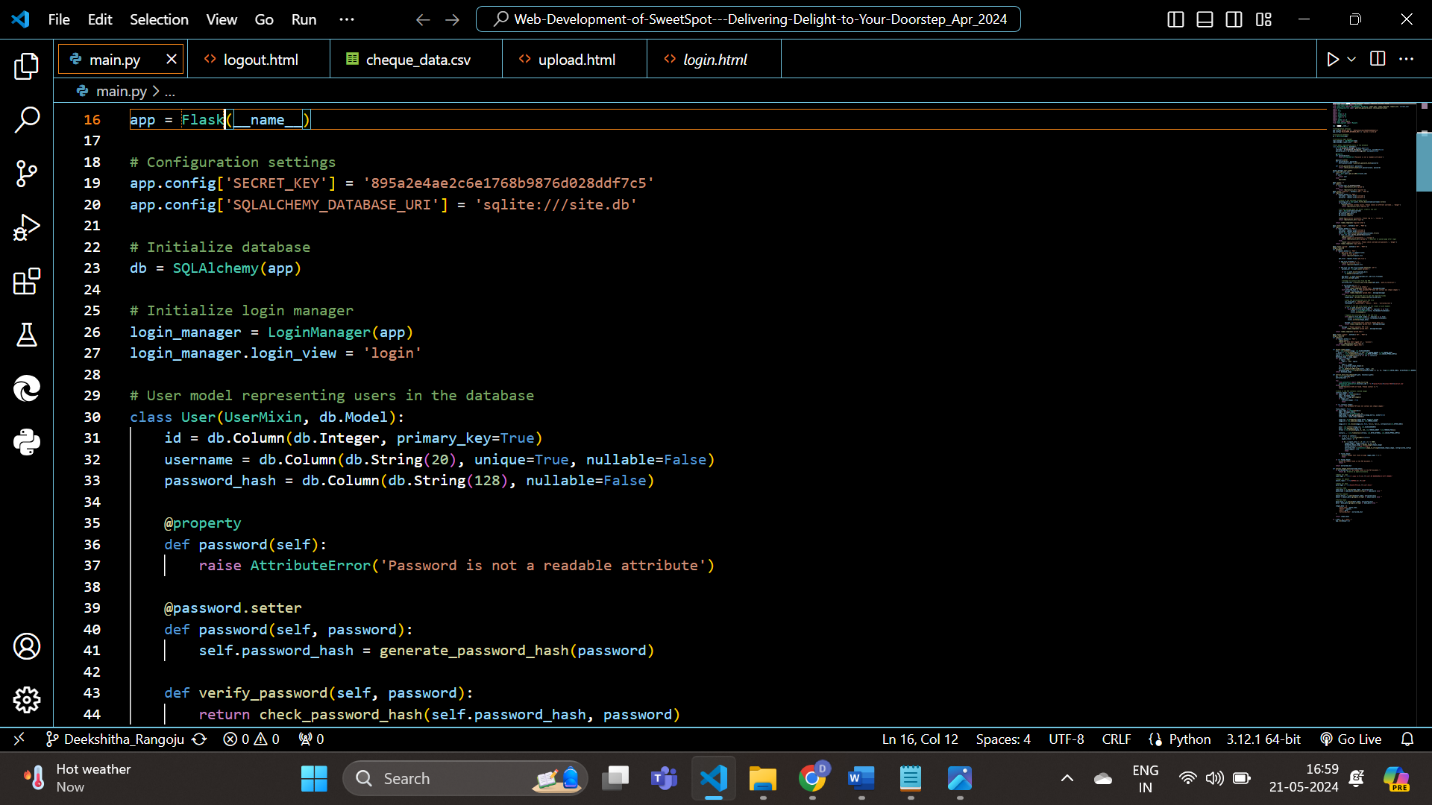
* Testing is Crucial: Thorough testing throughout the development process (unit, integration, system) can identify and fix bugs early on, leading to a more robust application.
* Informative Error Handling: Providing clear error messages and troubleshooting tips within the application improves user experience and helps them resolve issues independently.
* Manage External Dependencies: Applications that rely on external libraries (like Tesseract OCR) introduce potential points of failure. Consider alternative libraries or cloud-based solutions for better control and potentially higher accuracy.

**Areas for Improvement:**

* Accuracy Refinement: Regular expressions for text parsing can be further refined, and exploring advanced OCR techniques can improve the accuracy of extracted information from cheque images.
* Scalability for High Volume: The current application might not handle large volumes of cheques efficiently. Implementing load balancing and optimizing database interactions can address scalability needs.
* Enhanced Security: For production deployment, consider additional security measures like HTTPS and user authorization for specific functionalities to protect sensitive data.

### **\* Appendices:**

**Main.py:**



**Application Overview:**

