# **Assignment 2 - OpenAI Evaluation App**

Application link: <http://3.145.73.49:8501/>

Video Link: https://drive.google.com/file/d/1FgN-lEHbVfYQ6hQLEsMZnDbP6LPa-EFJ/view?usp=drive\_link

## **Team member 1: Dharun Karthick Ramaraj**

## **Team member 2: Nikhil Godalla**

## **Team member 3: Linata Deshmukh**

# **Introduction**

This project is a comprehensive tool built to evaluate OpenAI's ChatGPT capabilities on the GAIA dataset. The application combines data engineering with machine learning evaluation by preprocessing PDF documents, generating and comparing ChatGPT responses, and visualizing evaluation results. The goal of this assignment is to create an automated pipeline for PDF text extraction and analysis using Airflow, integrate it with FastAPI and Streamlit, and evaluate OpenAI's performance in answering specific questions.

Technologies involved:

* **Airflow**: For automating the text extraction process from PDFs using PyMuPDF and Amazon Textract.
* **FastAPI**: To serve as the backend for authentication, data management, and API integrations.
* **Streamlit**: To build a user-friendly interface for the tool, allowing both users and administrators to interact with the app.
* **AWS S3**: To store the processed files and ensure scalability.
* **Azure SQL**: To manage and store evaluation results and user data.

**Goal**: By the end of this project, we will have an operational pipeline that extracts text from PDF files, processes it for ChatGPT evaluation, and allows users to view and analyze the results within a secure web application.

# **Problem Statement**

The goal of this project is to create an efficient tool to evaluate ChatGPT's responses to questions associated with text documents in PDF format from the GAIA dataset. ChatGPT cannot directly process PDF files, so the tool needs to automate the ingestion, preprocessing, and evaluation of these documents, allowing the Model Evaluation Team to assess ChatGPT's performance accurately.

### **Project Requirements**

#### **Part 1: Automating Text Extraction and Database Population**

* **Airflow Pipelines**:
  + Set up an Airflow pipeline to automate PDF data acquisition and text extraction.
  + Process PDFs from the GAIA Benchmarking Validation & Testing Dataset.
  + Integrate text extraction options, including an open-source (e.g., PyPDF) and an API-based solution (e.g., AWS Textract), for robust and scalable processing.
  + Store extracted text and metadata in AWS S3 for secure, cloud-based storage and access.

#### **Part 2: Client-Facing Application with Streamlit and FastAPI**

* **FastAPI Backend**:
  + Implement user authentication, including registration and login, secured with JWT tokens.
  + Secure all endpoints with JWT, showing protected routes in the Swagger UI with a padlock icon.
  + Store user credentials in an Azure SQL database with hashed passwords.
  + Move core business logic to the FastAPI backend, allowing for streamlined processing and improved security.
  + Define services for Streamlit integration, including PDF selection, question answering, and metadata evaluation.
* **Streamlit Frontend**:
  + Develop a user-friendly interface that includes:
    - Secure login and registration.
    - A Question Answering interface for querying ChatGPT.
    - Options for selecting and processing specific PDF documents.
  + Implement session management for user roles (admin and user), allowing admins to upload files and regular users to interact with the evaluation tool.

#### **Deployment**

* **Amazon EC2 Deployment**:
  + Containerize the FastAPI and Streamlit components with Docker and host them on an Amazon EC2 instance.
  + Push Docker images to a repository, pull them onto the EC2 instance, and configure them to be publicly accessible.
  + Use a public URL for user access, providing seamless interaction for clients.

### **Development Details**

* **Database**: Use Azure SQL for storing user data and evaluation results, leveraging its scalability and integration with FastAPI.
* **File Storage**: Use AWS S3 to store and retrieve PDF documents securely.
* **Evaluation**: Send questions from selected PDF files to ChatGPT, compare its response to the expected answer, and allow users to modify Annotator instructions for re-evaluation.
* **Visualization**: Integrate Matplotlib within Streamlit to generate real-time visualizations of evaluation metrics, providing insights into ChatGPT's performance.

### **Testing**

* Create test cases for PDF processing, text extraction, and ChatGPT evaluations, including:
  + Direct response comparisons.
  + User modification of evaluation parameters.
  + Robust error handling to manage incomplete or erroneous responses.

### **Documentation & Compliance**

* Include a contribution statement on GitHub with the attestation clause.
* Document your process in a Code Lab document and create a detailed readme.md file.
* Keep your repository private until the deadline to comply with the anti-plagiarism policy.

# 

# 

# 

# **Proof of Concept**

To demonstrate that our solution is viable, we have combined several technologies. Airflow automates the text extraction process, enabling scheduled and on-demand processing of PDFs using either PyMuPDF or Amazon Textract. FastAPI handles user management and API endpoints, while Streamlit provides a frontend for user interactions.

**Key Technologies**:

* **Airflow** for task automation and scheduling.
* **PyMuPDF** for text extraction from PDFs as an open-source solution.
* **Amazon Textract** as an alternative for enterprise-grade text extraction, providing OCR capabilities.
* **FastAPI** for a secure backend API with JWT-based authentication.
* **Streamlit** for a responsive web interface that can interact with FastAPI endpoints.

**Initial tests involved:**

* Configuring Airflow to execute basic text extraction jobs using PyMuPDF and uploading results to AWS S3.
* Setting up FastAPI to handle basic user authentication and database connections with Azure SQL.
* Building a simple Streamlit interface that could display extracted text and allow for user input.

**Findings**:

* The OpenAI model performs well with most questions when provided with the correct context.
* However, handling complex files remains a challenge, as OpenAI doesn’t accept direct file inputs.
* All files need to be preprocessed before being sent to the model, ensuring the elevant information is extracted and presented in a way that the model can interpret.

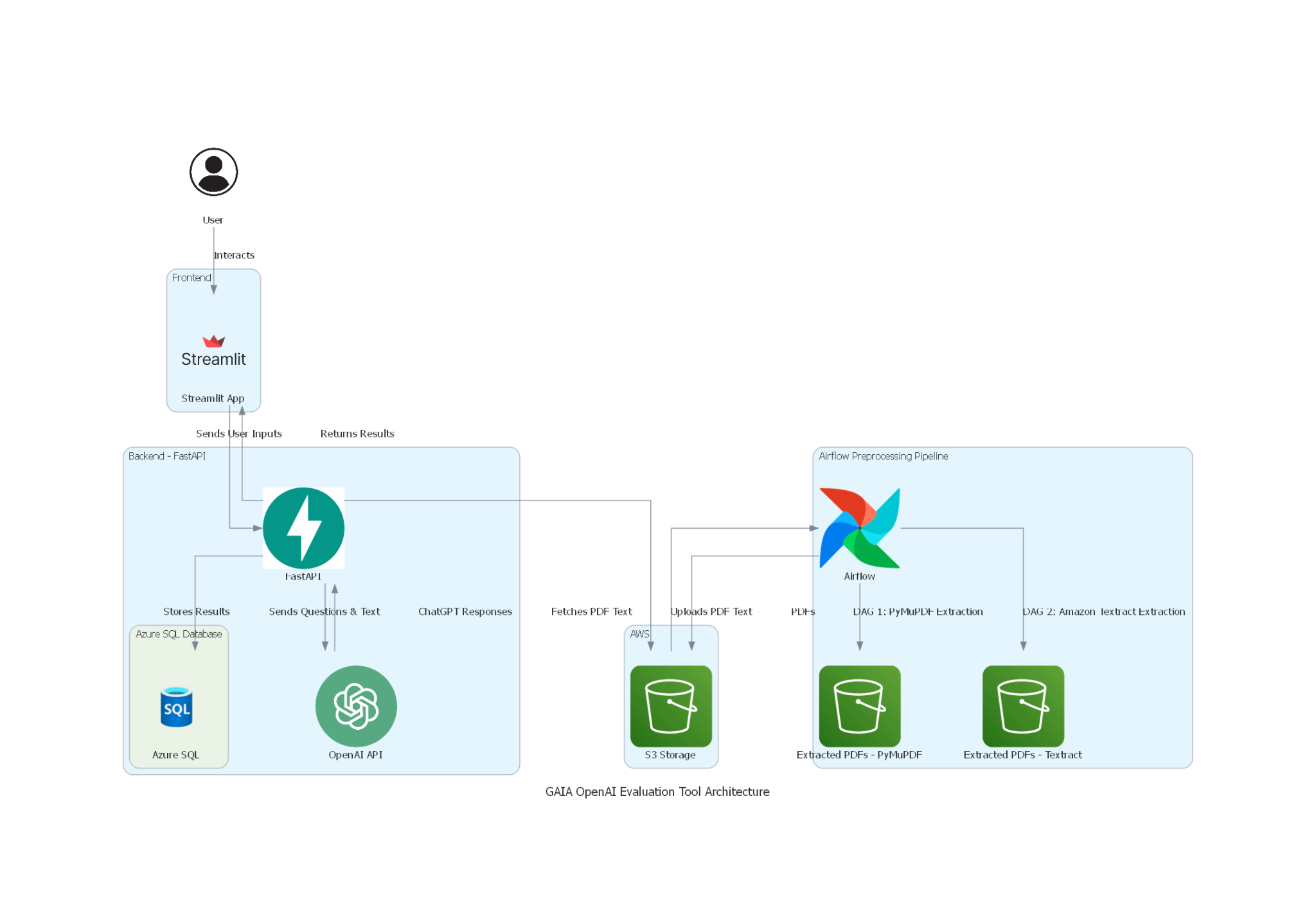
# 

# 

# 

# 

# **Architecture Diagram**



**Components**:

1. **User Interface**: Represents the end user interacting with the Streamlit frontend.
2. **Frontend - Streamlit**: User interacts with the Streamlit app to access evaluation functionalities.
3. **Airflow Preprocessing Pipeline**: Automates PDF preprocessing using PyMuPDF and Amazon Textract. Text data is then stored in AWS S3.
4. **AWS S3 Storage**: Holds both the uploaded PDFs and the extracted text.
5. **FastAPI Backend**: Manages API requests, interactions with the OpenAI API, and database storage.
6. **Azure SQL Database**: Stores the results and metadata from the evaluations.
7. **OpenAI API**: Processes questions along with the preprocessed PDF text and returns answers back to the backend for further handling.

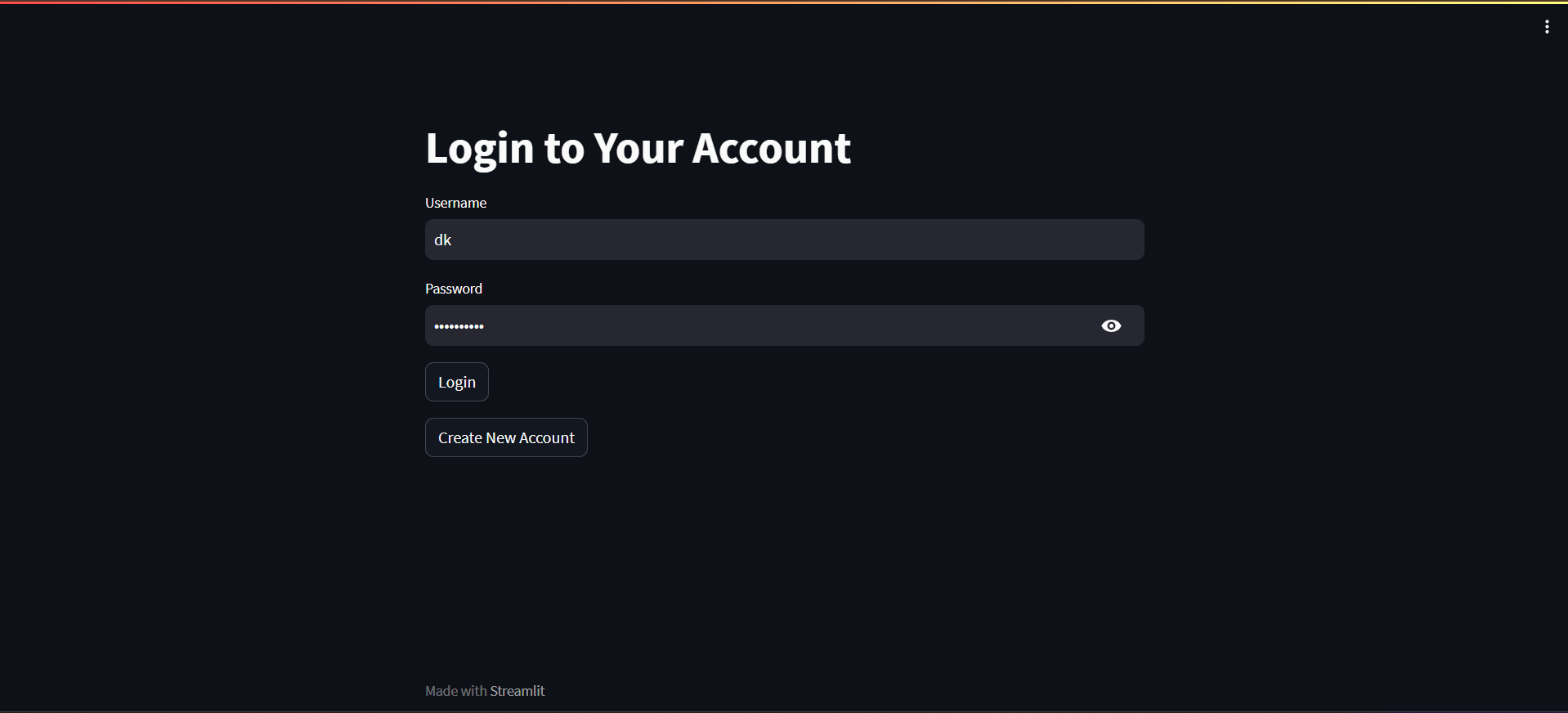
# 

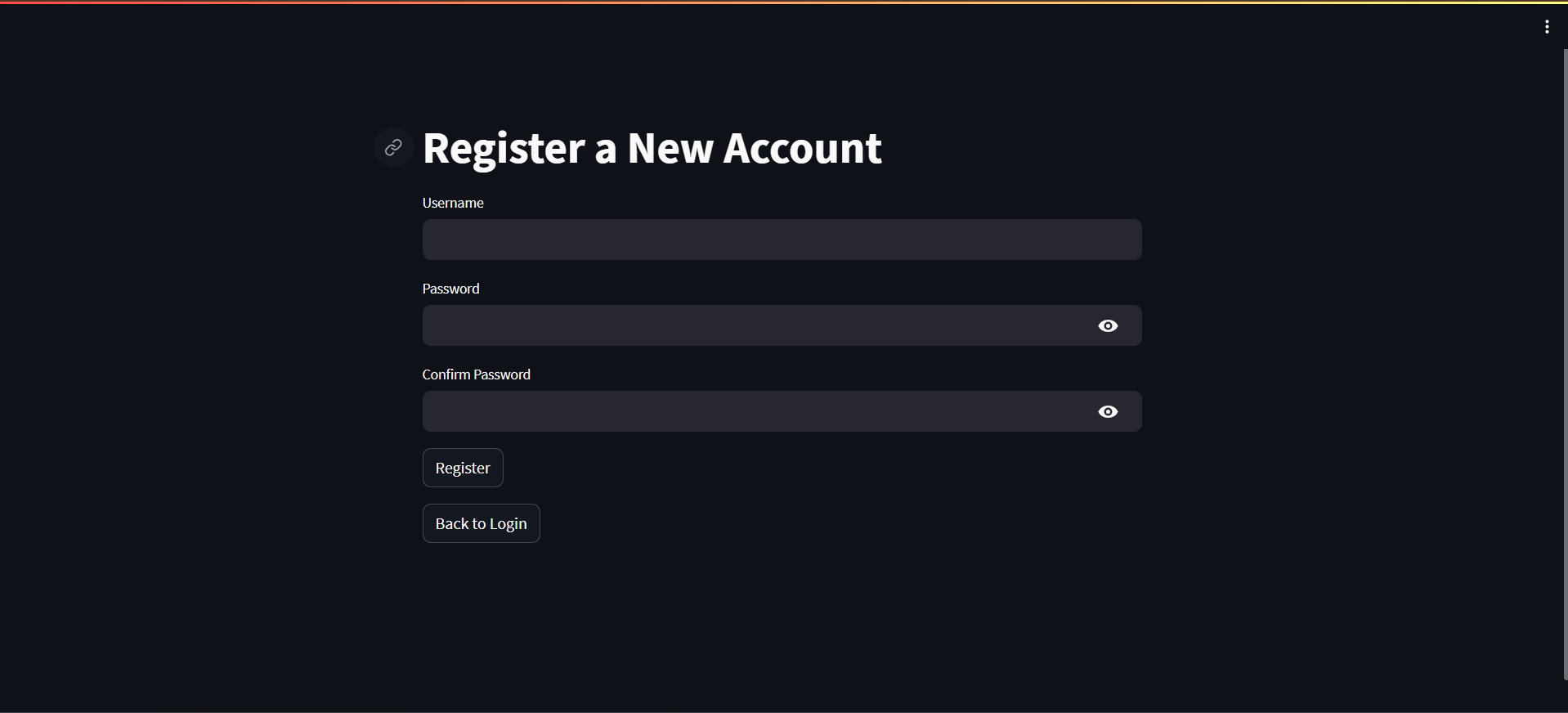
# 

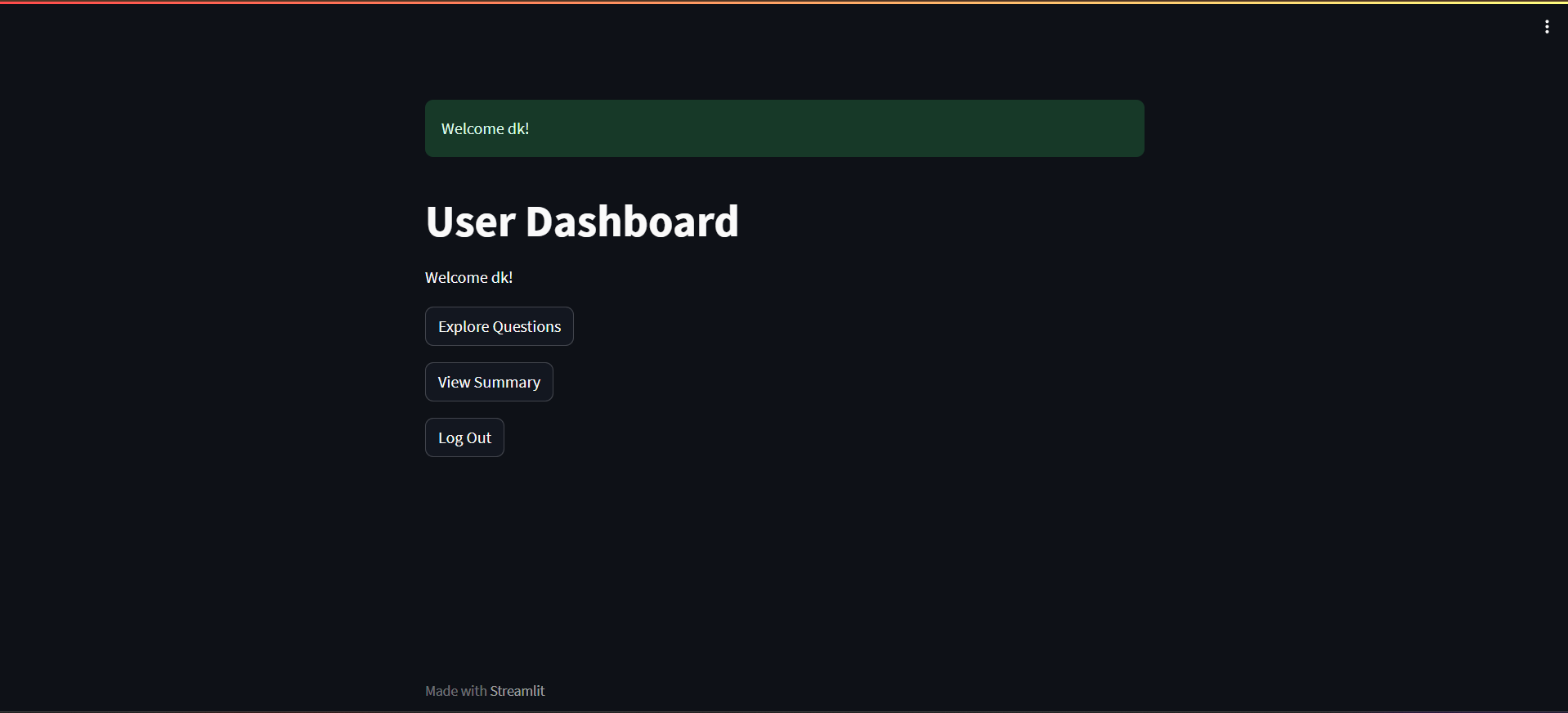
# **Walkthrough of the Application**

## **1. User Registration and Login**

The user registers through the Streamlit interface, which sends a request to the FastAPI backend. After successful registration or login, the user is authenticated with a JWT token.

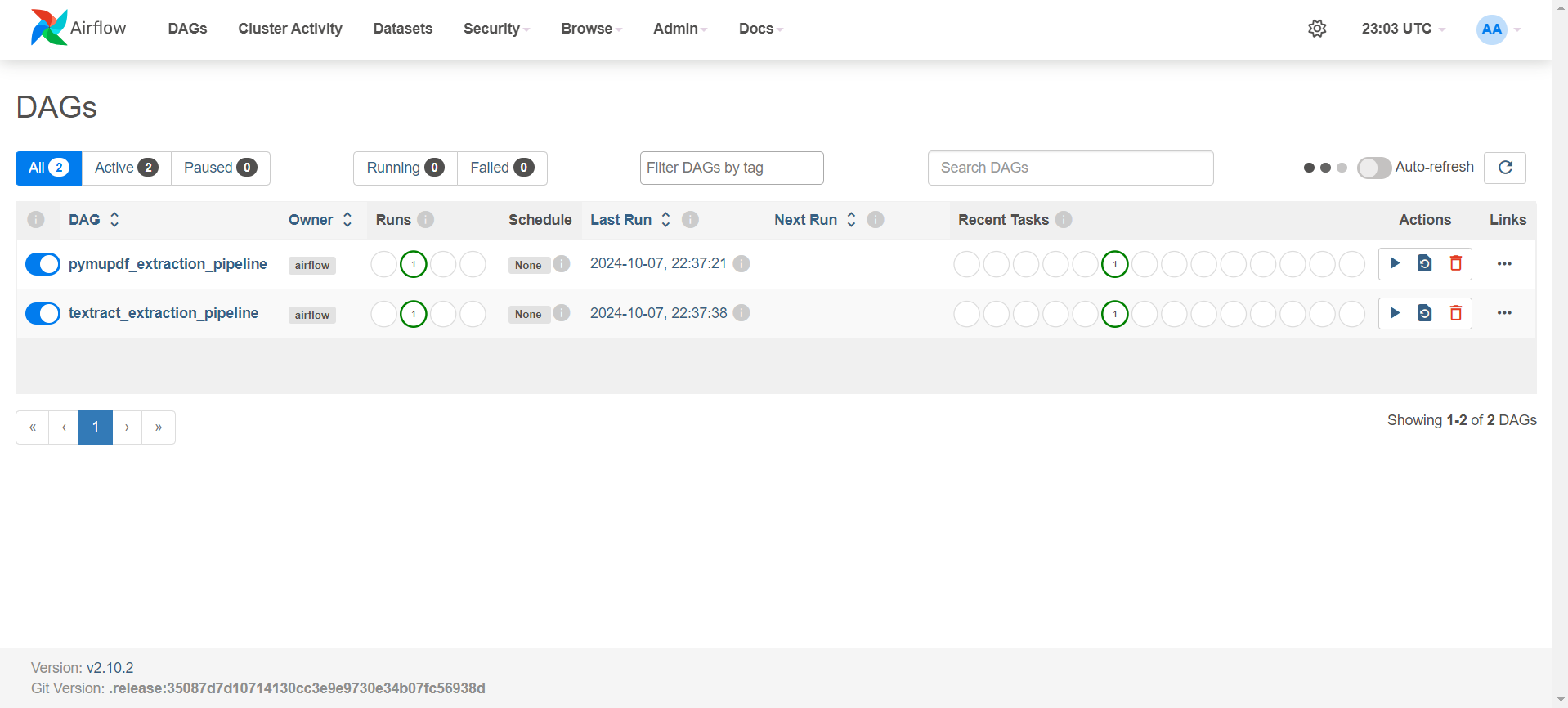






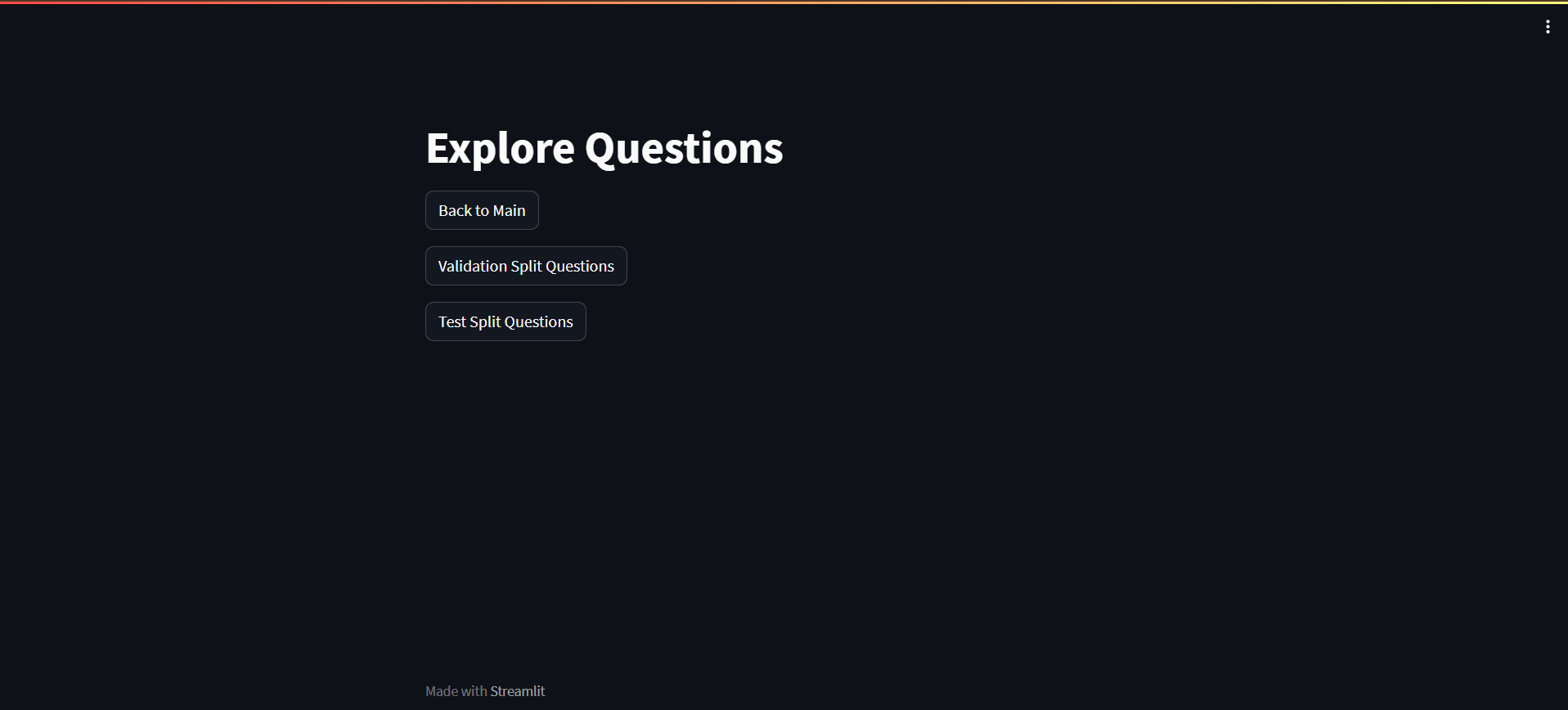
## **2. Dataset Loading and Processing**

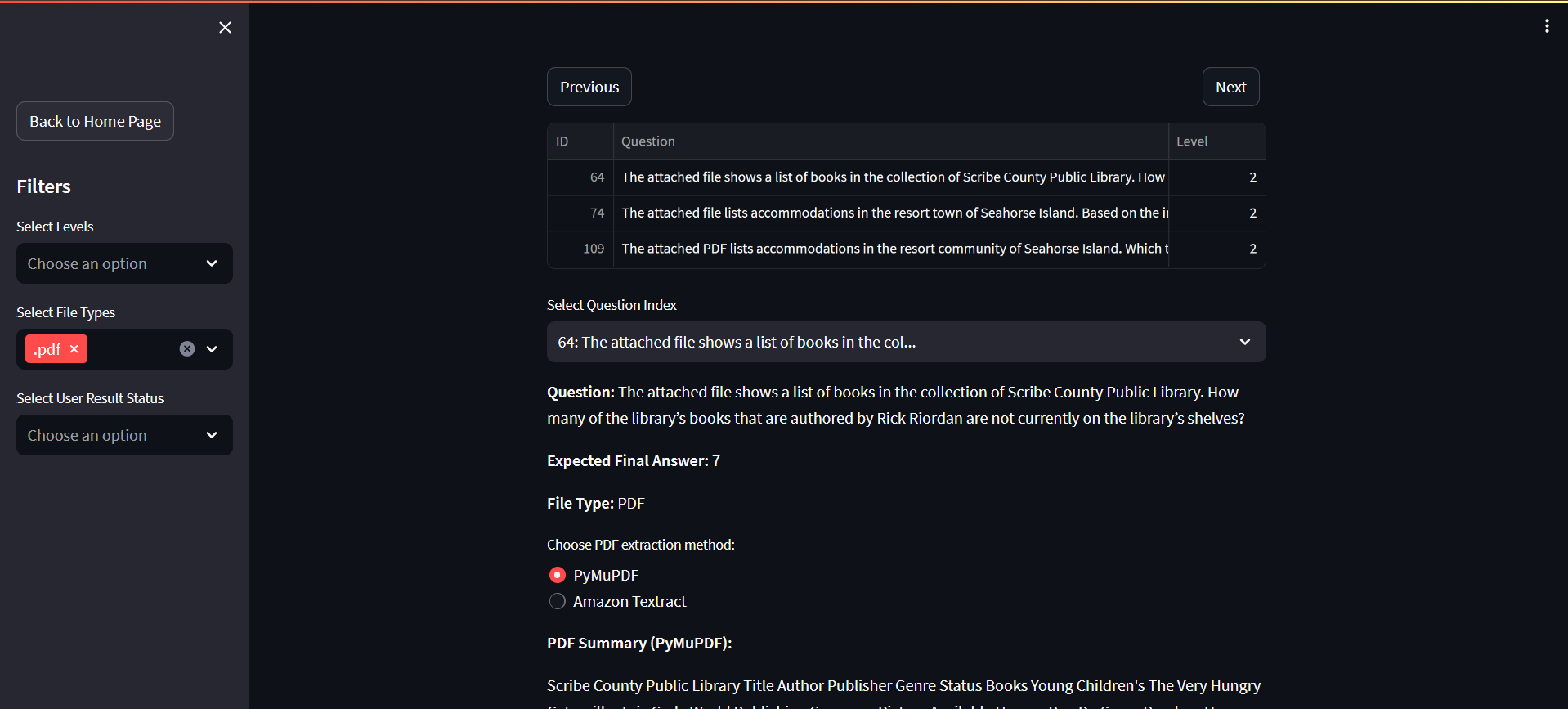
Admin users can rely on the scheduled Airflow pipeline running in Docker to automatically process new PDFs in AWS S3. The PDFs are processed using either PyMuPDF or Amazon Textract, based on the admin's configured preferences.

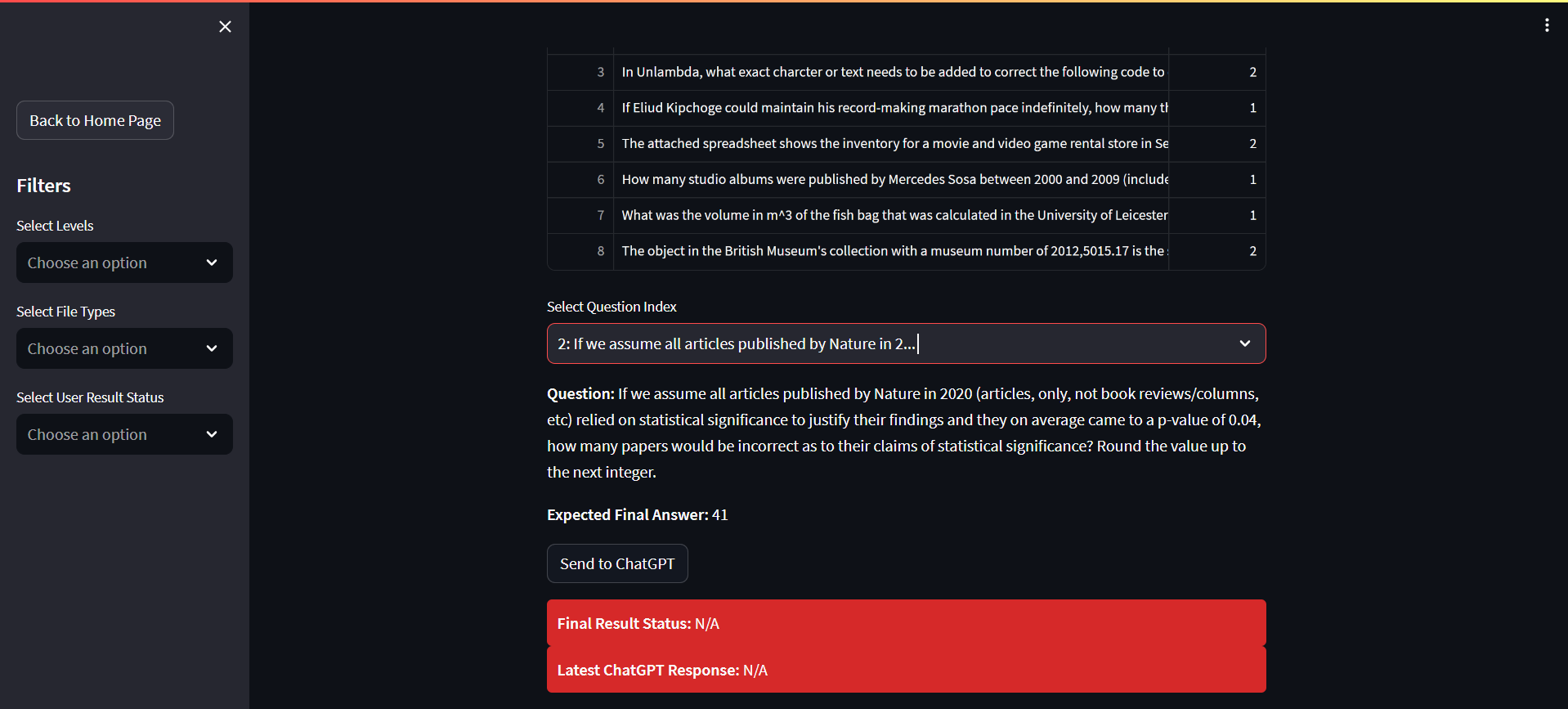


## **3. Question Evaluation**

Users can select a question associated with a preprocessed PDF. The extracted text is then sent to ChatGPT for response generation. The app compares ChatGPT's response to a predefined "final answer."

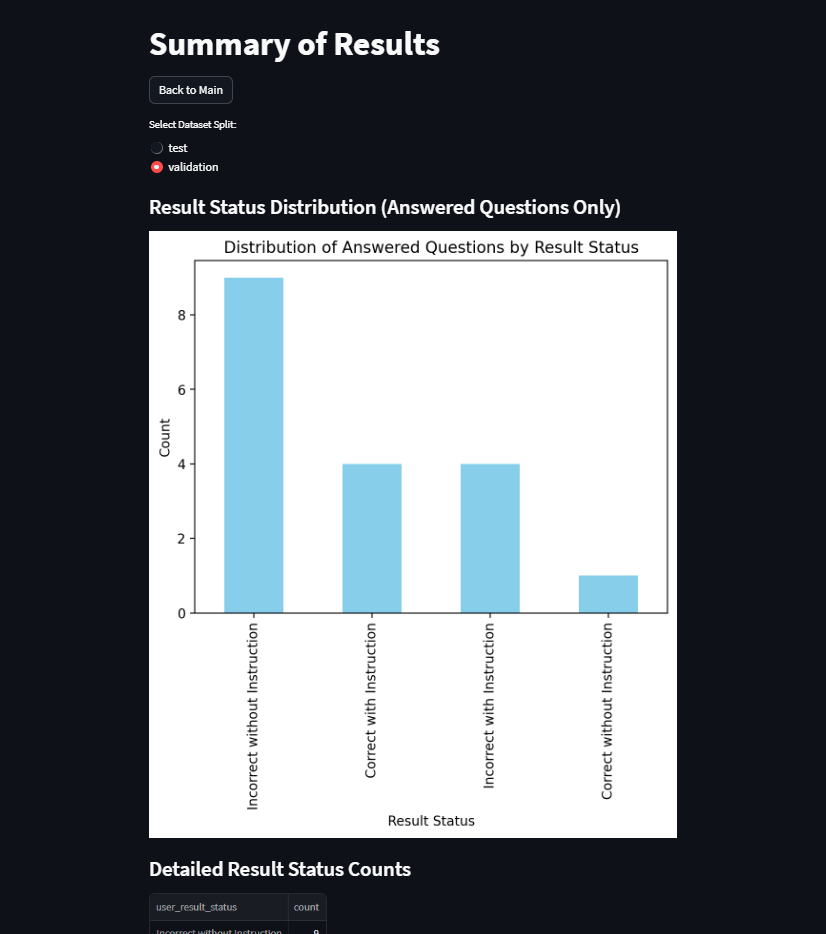


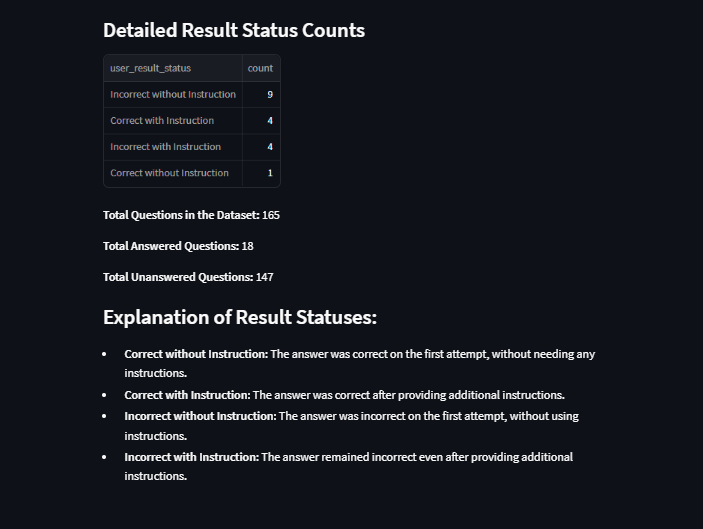


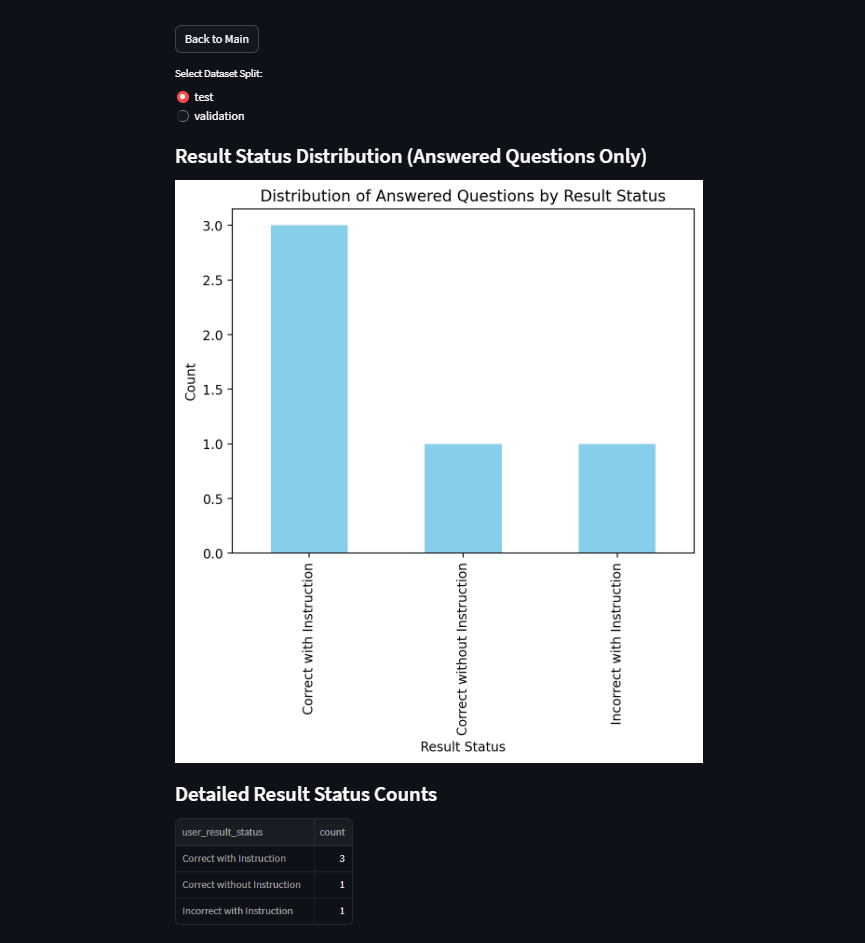


## **4. Visualization**

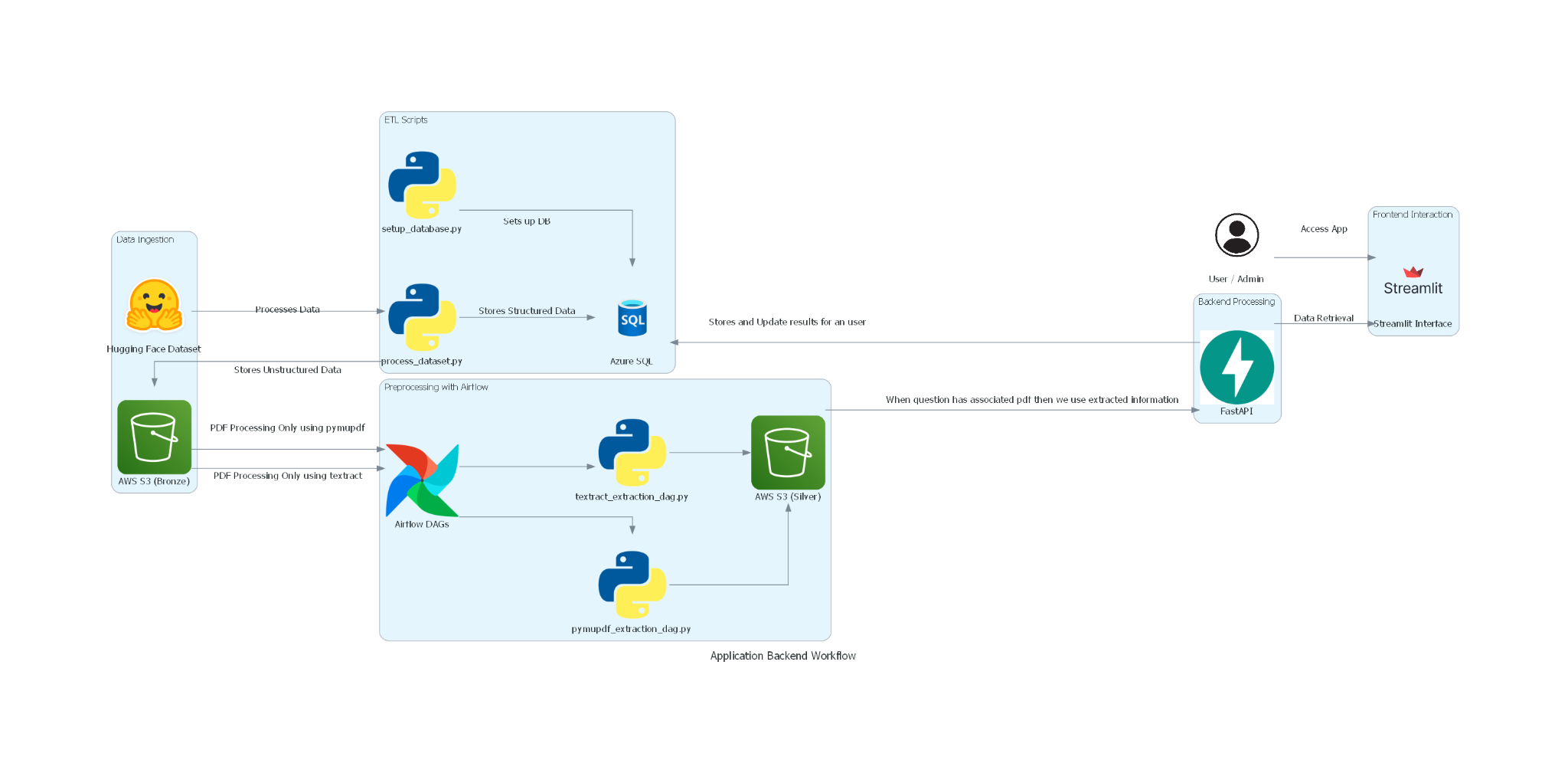
Users can view a summary of the evaluation results, including metrics on ChatGPT's accuracy.







## **Application Workflow**



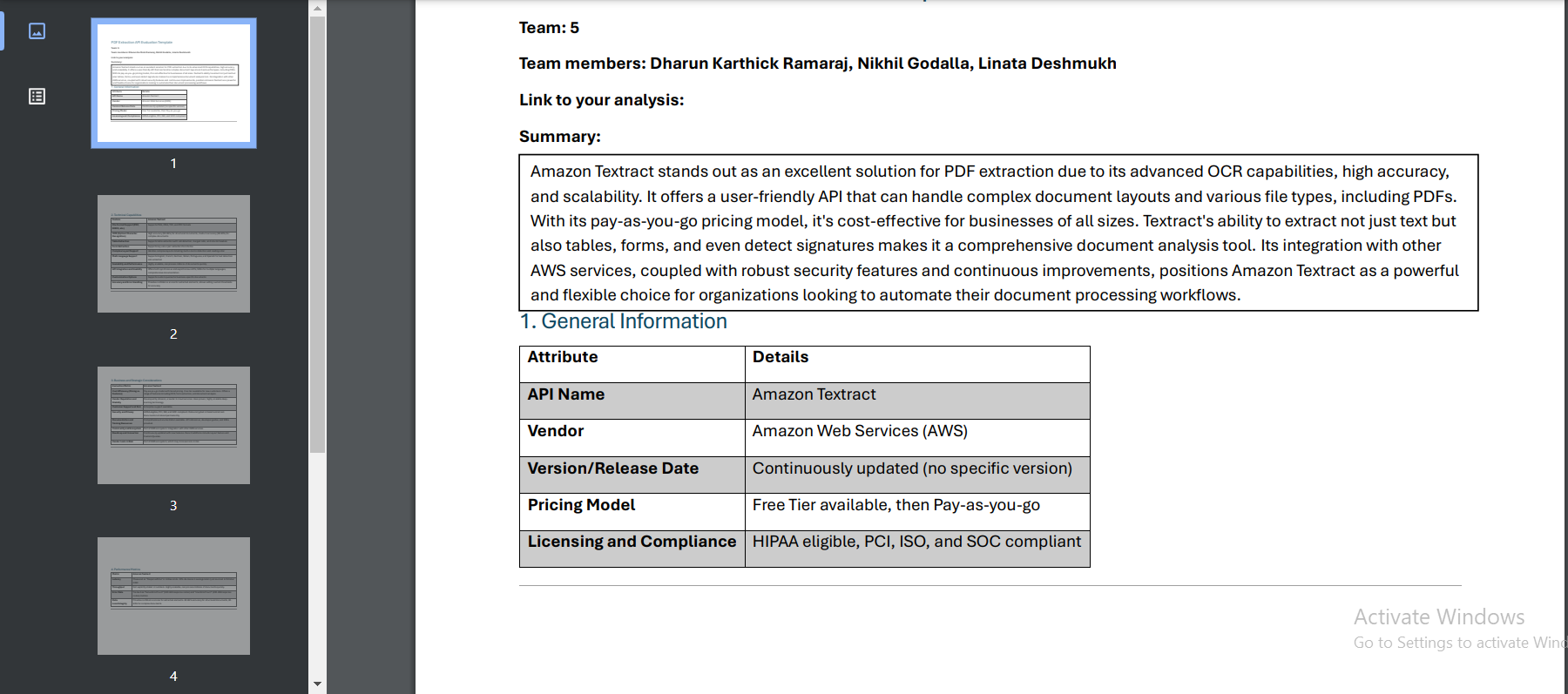
### **Data Flow and Processing**

**Data Ingestion**:

* PDFs are uploaded to AWS S3 under specific folders based on data types (e.g., "test" and "validation" sets).

**Preprocessing with Airflow**:

* Airflow DAGs (pymupdf\_extraction\_dag.py and textract\_extraction\_dag.py) automate text extraction from PDFs.
* Extracted text is stored back in AWS S3 with a .txt extension.



[Google Drive Link](<https://drive.google.com/file/d/1W5_tsqub1gHULE9xUUKbs1mgiOS16O1t/view?usp=sharing> ) [Github Location](docs/PDF\_Extraction\_API\_Evaluation.pdf)\*\*

**Backend Processing (FastAPI)**:

* The extracted text and corresponding questions are sent to ChatGPT via FastAPI endpoints.
* FastAPI manages database interactions with Azure SQL, saving the results and metadata.

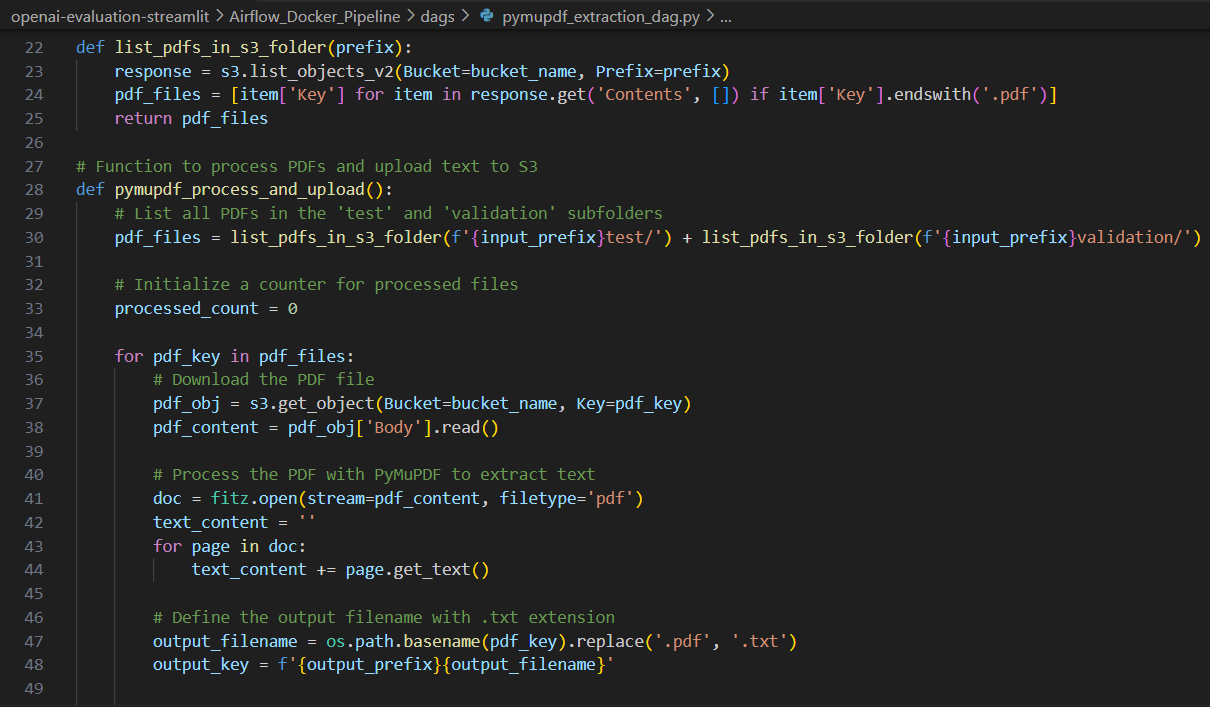
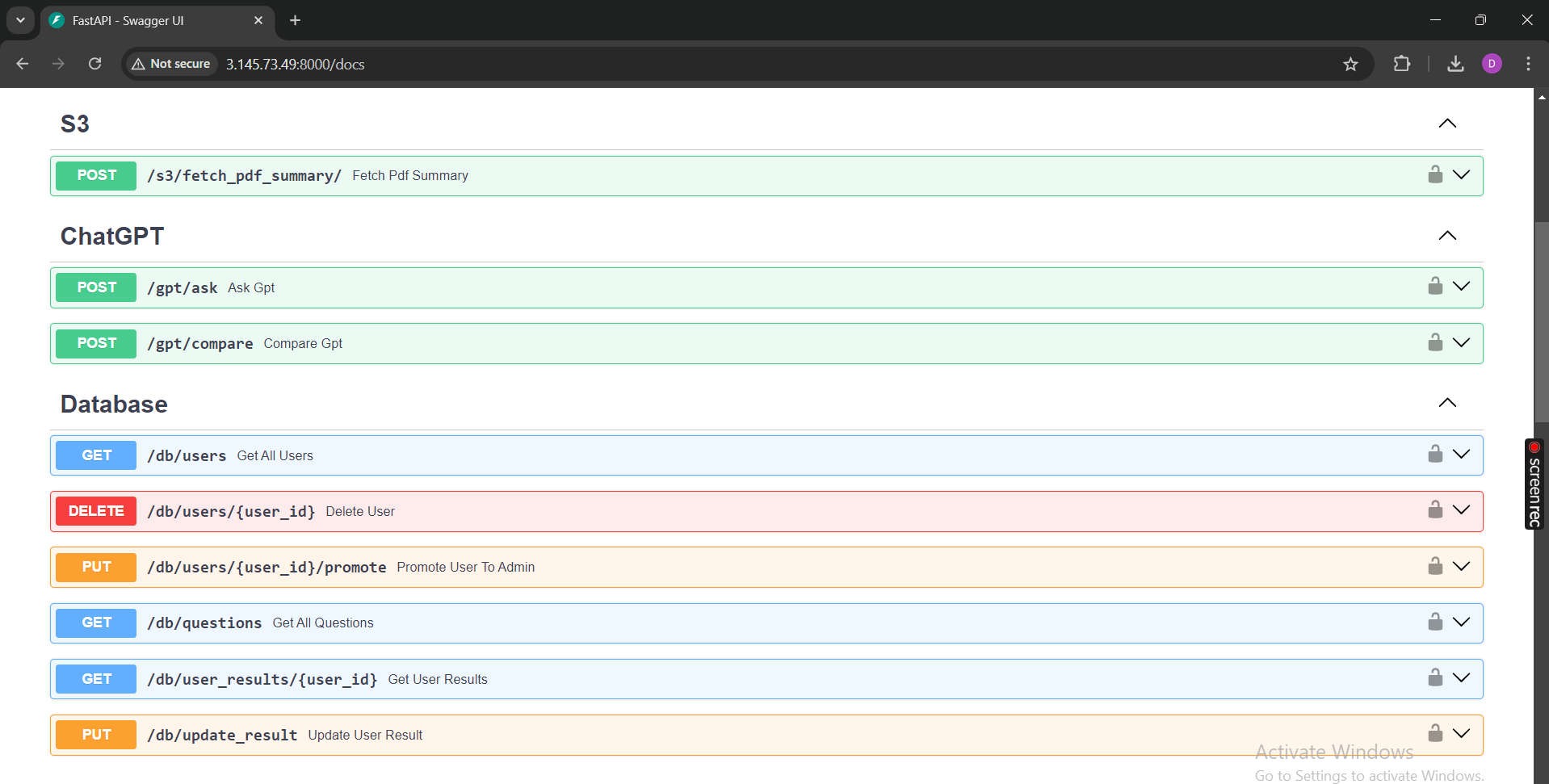
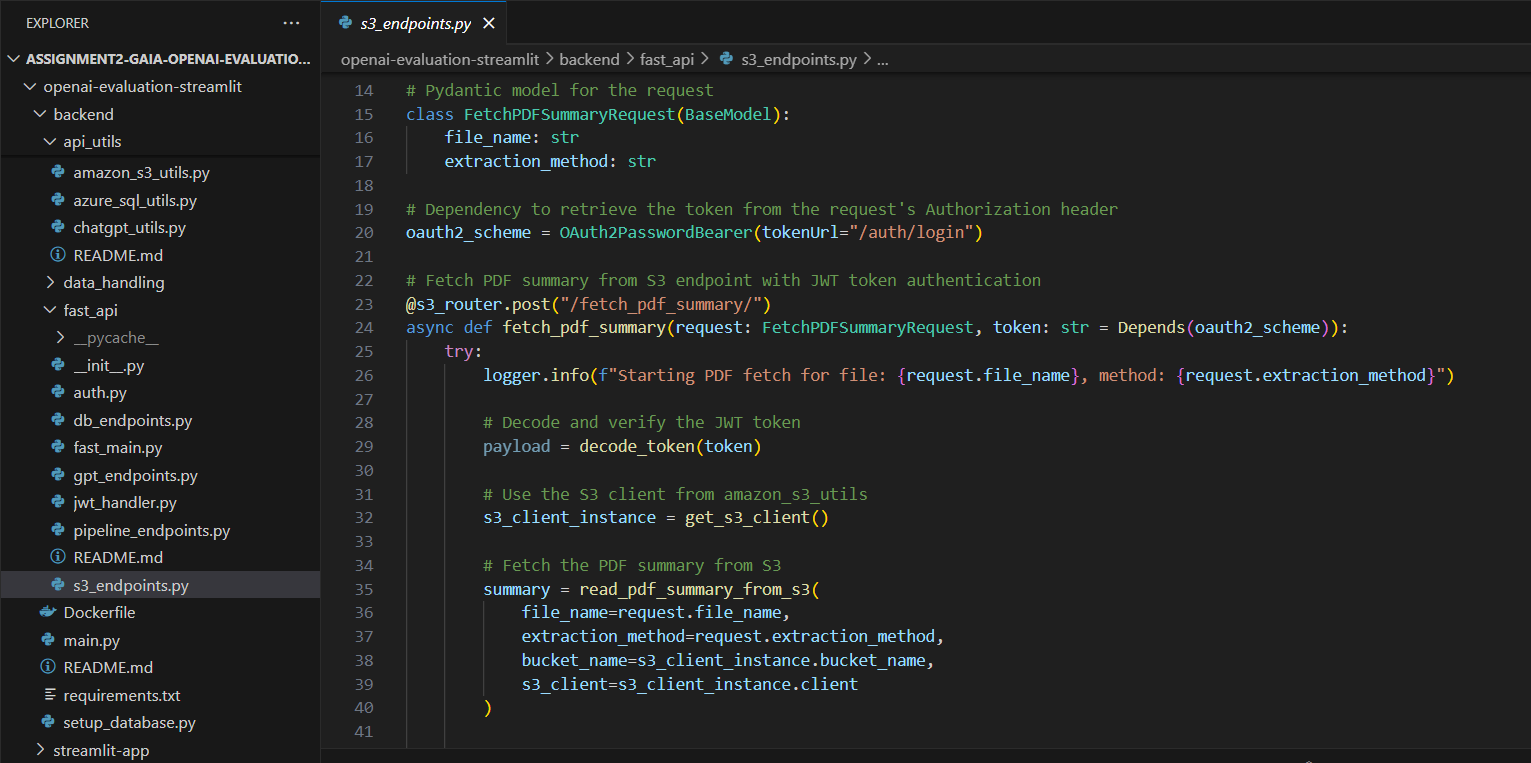
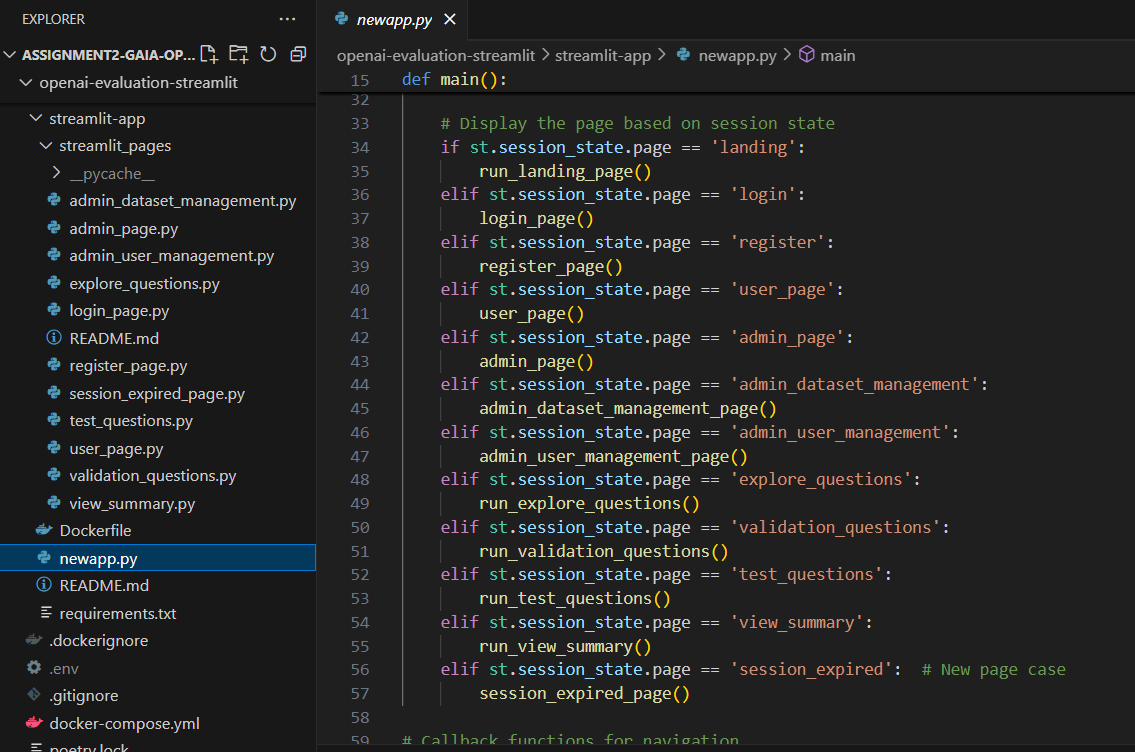
**Frontend Interaction (Streamlit)**:

* Users interact with Streamlit to view questions, initiate evaluations, and see summaries.
* Admins can manage datasets and users, controlling access to the evaluation functionalities.

### **ETL Scripts**

1. **Dataset Processing Script (process\_dataset.py)**:
   * This script authenticates with Hugging Face, clones the dataset repository, and loads the GAIA dataset for preprocessing.
   * Files are uploaded to AWS S3, and the data is inserted into Azure SQL Database.
2. **Database Setup Script (setup\_database.py)**:
   * This script configures the Azure SQL Database by creating tables for user management and results tracking.
   * It includes the setup of default admin and user accounts, with secure password storage.

### **Code Snippets**

* **Airflow DAGs**: Automate text extraction and file upload to S3.
* **FastAPI Authentication**: Provides JWT-based authentication for secure access to API endpoints.
* **Streamlit Pages**: Provide interfaces for user login, dataset exploration, and the admin dashboard.

# **References**

1. FastAPI Documentation:<https://fastapi.tiangolo.com/>
2. Streamlit Documentation:<https://docs.streamlit.io/>
3. Apache Airflow Documentation:<https://airflow.apache.org/docs/>
4. Amazon Textract Documentation:<https://docs.aws.amazon.com/textract/>
5. PyMuPDF Documentation:<https://pymupdf.readthedocs.io/en/latest/>
6. Azure SQL Documentation:<https://docs.microsoft.com/en-us/azure/azure-sql/>