DOCUMENT SUMMARIZER

Table of Content

SUBMITTED BY :-

RIYA RASTOGI

DESCRIPTION

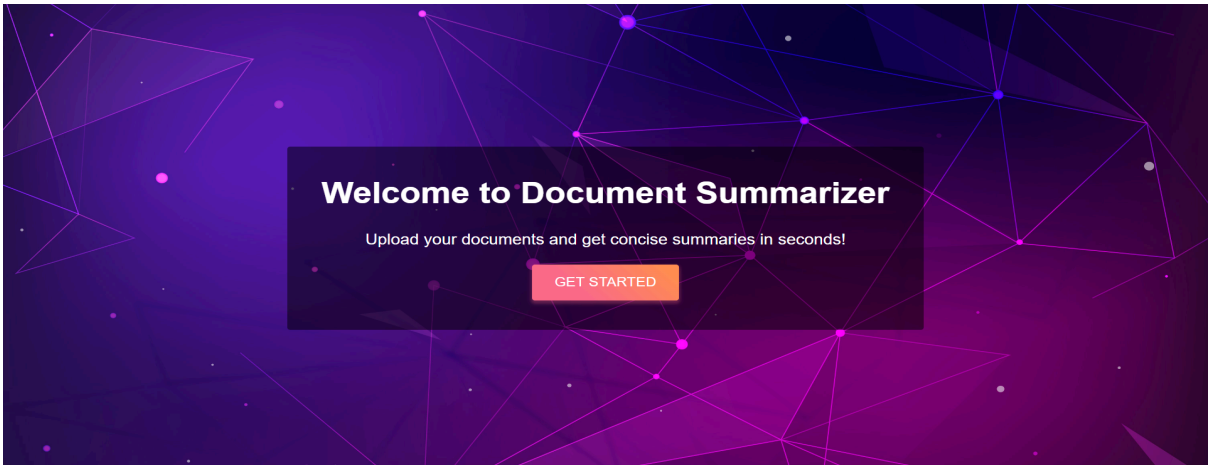
This project involves developing a React frontend and a FastAPI backend for file upload and summarization. The application starts with a Welcome Page, followed by an Upload Page for document uploads, and a Summary Page that displays the generated summaries. The backend, served using Uvicorn, leverages the Transformers library and Torch for generating summaries. An API endpoint handles file uploads and returns the summarized text. The LLM model is deployed within the FastAPI backend, and Docker ensures consistent deployment across environments. This setup provides a seamless user experience for uploading documents and receiving summaries.

DETAILED APPROACH:

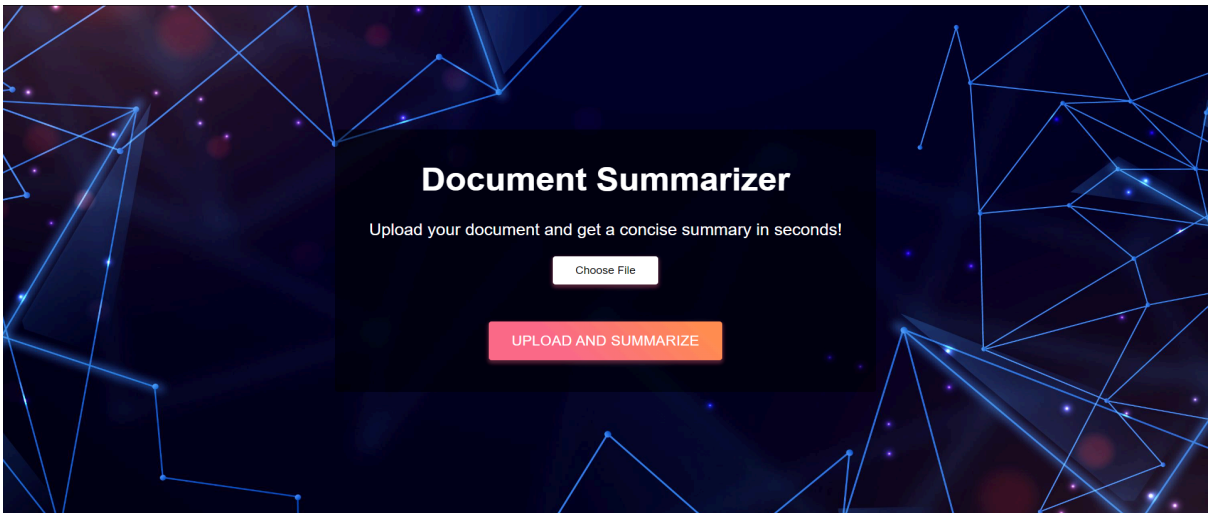
1. **Backend Setup**:
    - Utilized FastAPI to create a robust and high-performance backend and ease of use with asynchronous operations.
    - Used Hugging Face's `transformers` library to load a pre-trained BART model.
    - Uvicorn was used as the ASGI server for serving the FastAPI application.
    - Transformers library was employed for handling natural language processing tasks.
    - Torch was utilized to support the machine learning models used in summarization.
    - Ensures consistency across different environments. Separate Dockerfiles for backend and frontend, managed with Docker Compose.

2. **Frontend Development**:
    - Implemented the frontend using React.
    - Used Material-UI for responsive and modern design
    - Created a file upload component to allow users to upload documents for summarization.
    - Enhanced the user interface using Bootstrap for styling and making the components more attractive.
    - Used Axios to make API calls to the backend for file upload and summarization.

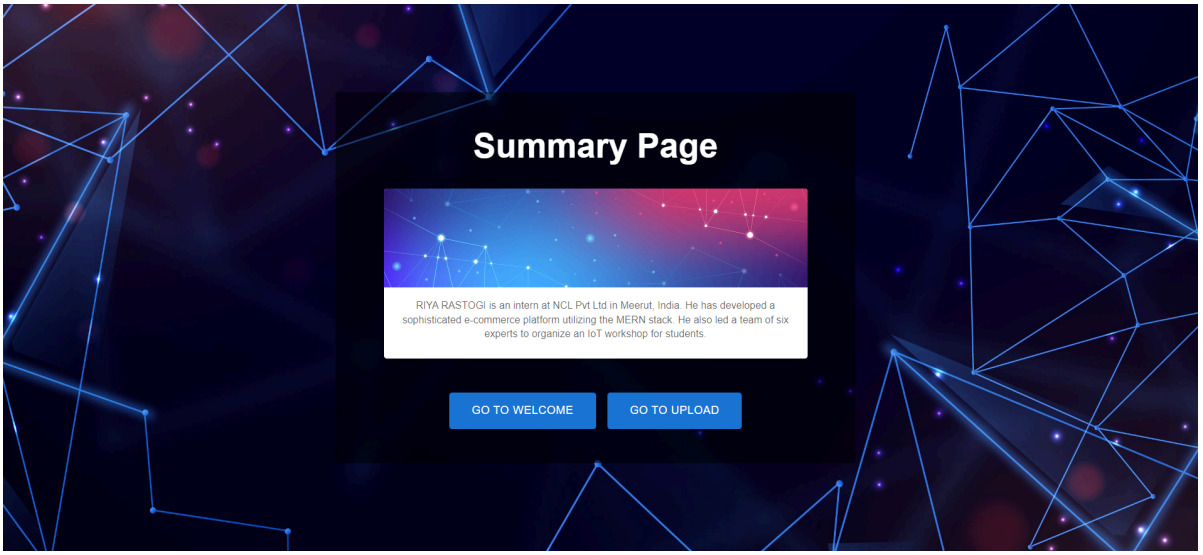
WELCOME PAGE



UPLOAD PAGE



SUMMARY PAGE



3. **Integration**:
    - Connected the React frontend with the FastAPI backend to enable seamless file uploads and retrieval of summaries.
    - Implemented necessary API endpoints in FastAPI to handle file uploads and return summaries.

Challenges and Solutions

During the development of this project, we encountered several challenges and devised effective solutions to overcome them.

Challenge 1: File Upload Handling Handling various file types and sizes was a significant issue. To address this, we utilized FastAPI's UploadFile class, implementing checks for file extensions and sizes. We also incorporated error handling to manage unsupported file types and excessively large files, ensuring robust file upload functionality.

Challenge 2: Summarization Accuracy Generating accurate and concise summaries posed another challenge. To enhance the quality of the summaries, we fine-tuned the parameters of the BART model, specifically adjusting the max_length and min_length settings. We tested the model with various document types to ensure its robustness and ability to produce high-quality summaries consistently.

Challenge 3: Docker Configuration Ensuring seamless communication between the backend and frontend containers was crucial for smooth operation. We addressed this by creating detailed Dockerfiles and a Docker Compose file. Proper networking and volume mapping were configured to guarantee that the containers could interact without issues, facilitating efficient data flow and service integration.

Challenge 4: User Interface and Experience Creating an intuitive and visually appealing user interface was essential for user satisfaction. We adopted Material-UI to achieve a modern design and implemented feedback mechanisms to inform users about the progress of file uploads and the status of summary generation. This approach significantly improved the overall user experience, making the application more user-friendly and engaging.

Challenge 5: Backend Deployment Issue

While we successfully deployed the frontend with Docker, we encountered issues with the backend deployment. Specifically, the packages were not installed due to read-only permissions. Despite our efforts, we were unable to resolve this issue, which impacted the deployment process.

Bibliography

1. **FastAPI**
    - FastAPI is a modern, fast (high-performance), web framework for building APIs with Python 3.6+ based on standard Python type hints.
2. **Transformers**
    - Transformers provides general-purpose architectures for natural language understanding and generation with pretrained models.
3. **PyPDF2**
    - PyPDF2 is a pure-python PDF library capable of splitting, merging, cropping, and transforming the pages of PDF files.
4. **Uvicorn**
    - Uvicorn is a lightning-fast ASGI server implementation, using `uvloop` and `httptools`.
5. **React**
    - A declarative, efficient, and flexible JavaScript library for building user interfaces.
6. **React Router**
    - Declarative routing for React.
7. **Axios**
    - Promise-based HTTP client for the browser and Node.js.
8. **Material-UI**
    - React components for faster and easier web development. Build your own design system, or start with Material Design.
9. **Docker**

    Docker is an open platform for developing, shipping, and running applications.

10. **Docker Compose**
    - Define and run multi-container applications with Docker.

How They Were Used

- **FastAPI**: Used to create the backend server that handles file uploads and document summarization.
- **Transformers**: Used to load and run the GPT-2 model for summarizing text.
- **PyPDF2**: Used to extract text from PDF documents for summarization.
- **Uvicorn**: Used as the ASGI server to run the FastAPI application.
- **React**: Used to create the frontend user interface.
- **React Router**: Used for routing in the React application.
- **Axios**: Used to make HTTP requests from the frontend to the backend.
- **Material-UI**: Used to design and style the frontend components.
- **Docker**: Used to containerize the backend and frontend applications.
- **Docker Compose**: Used to manage and run the multi-container Docker applications.