Document Summarizer Project: Approach and Challenges

Overview

This document provides a detailed explanation of the approach taken to develop the Document Summarizer web application. It also discusses the challenges encountered during the development process and how they were overcome.

Approach

1. Project Structure

The project is divided into three main components:

Backend: Developed using FastAPI, it handles file uploads, interacts with a locally hosted Language Model (LLM), and returns summarized text.
Frontend: Developed using React, it provides a user interface for users to upload documents and view the generated summaries.
LLM Deployment: A pre-trained model from Hugging Face's transformers library (such as GPT-2 or DistilBART) is used to generate summaries.

2. Backend Development

FastAPI: Chosen for its simplicity and high performance, FastAPI serves as the backbone of the application, handling REST API requests for file uploads and summarization.
File Handling: The backend supports .txt file uploads, which are stored in a local directory. File handling is implemented to ensure that only valid files are processed.
Summarization: The summarization pipeline leverages the transformers library, where a lightweight model (e.g., sshleifer/distilbart-cnn-12-6) is loaded for generating summaries.

3. Frontend Development

React: Chosen for its component-based architecture, React is used to create a user-friendly interface where users can upload files and view summaries.
API Integration: The frontend interacts with the backend through API calls to upload files and fetch summaries.
User Experience: The UI is kept simple, focusing on ease of use, with clear instructions and feedback for the user during file uploads and summary generation.

4. Docker and Deployment

Dockerization: Both the backend and frontend are containerized using Docker to ensure that the application can be easily deployed and run on any machine.
Docker Compose: Used to orchestrate the running of both containers, making it easier to start the entire application with a single command.

Challenges Faced

1. Model Deployment

Challenge: Deploying a pre-trained Language Model locally while ensuring efficient performance and resource management.
Solution: A smaller, more efficient model like DistilBART was chosen over larger models like GPT-2. This reduced the memory footprint and improved the response time, making it suitable for local deployment without the need for specialized hardware.

2. File Handling and Upload

Challenge: Ensuring that the application can handle various file sizes and types without errors.
Solution: Implemented file validation checks on both the frontend and backend to ensure that only valid .txt files are accepted. Additionally, error handling was added to manage cases where the file size was too large or the content was unsupported.

3. Docker Build Issues

Challenge: Encountered issues during the Docker build process, particularly when copying unnecessary files like the virtual environment.

Solution: Introduced a .dockerignore file to exclude unnecessary files such as venv/ and other temporary or binary files. This resolved the Docker build errors and reduced the image size, making the build process more efficient.

4. Frontend and Backend Integration

Challenge: Ensuring seamless communication between the React frontend and FastAPI backend.
Solution: Developed and tested each API endpoint separately using Postman before integrating them with the React frontend. This approach made it easier to debug and ensure that the frontend and backend interacted correctly.

Conclusion

This project involved multiple components, each presenting its own set of challenges. By carefully selecting the appropriate technologies and implementing solutions to address the encountered issues, a robust and user-friendly Document Summarizer application was successfully developed. The use of Docker further ensured that the application could be easily deployed and run on various environments, making it versatile and ready for production use.