**Project Documentation: Learning Assistant video transcriber, summarizer, mcq generator and notes generator**

**Overview**

This project is an automated video processing system designed to perform the following tasks:

1. **Download YouTube videos**
2. **Transcribe video audio**
3. **Summarize transcriptions**
4. **Generate notes**
5. **Create multiple-choice questions (MCQs)**

The system utilizes various Python libraries and frameworks and is built with a FastAPI backend and a React frontend, orchestrated using Docker.

**Project Structure**

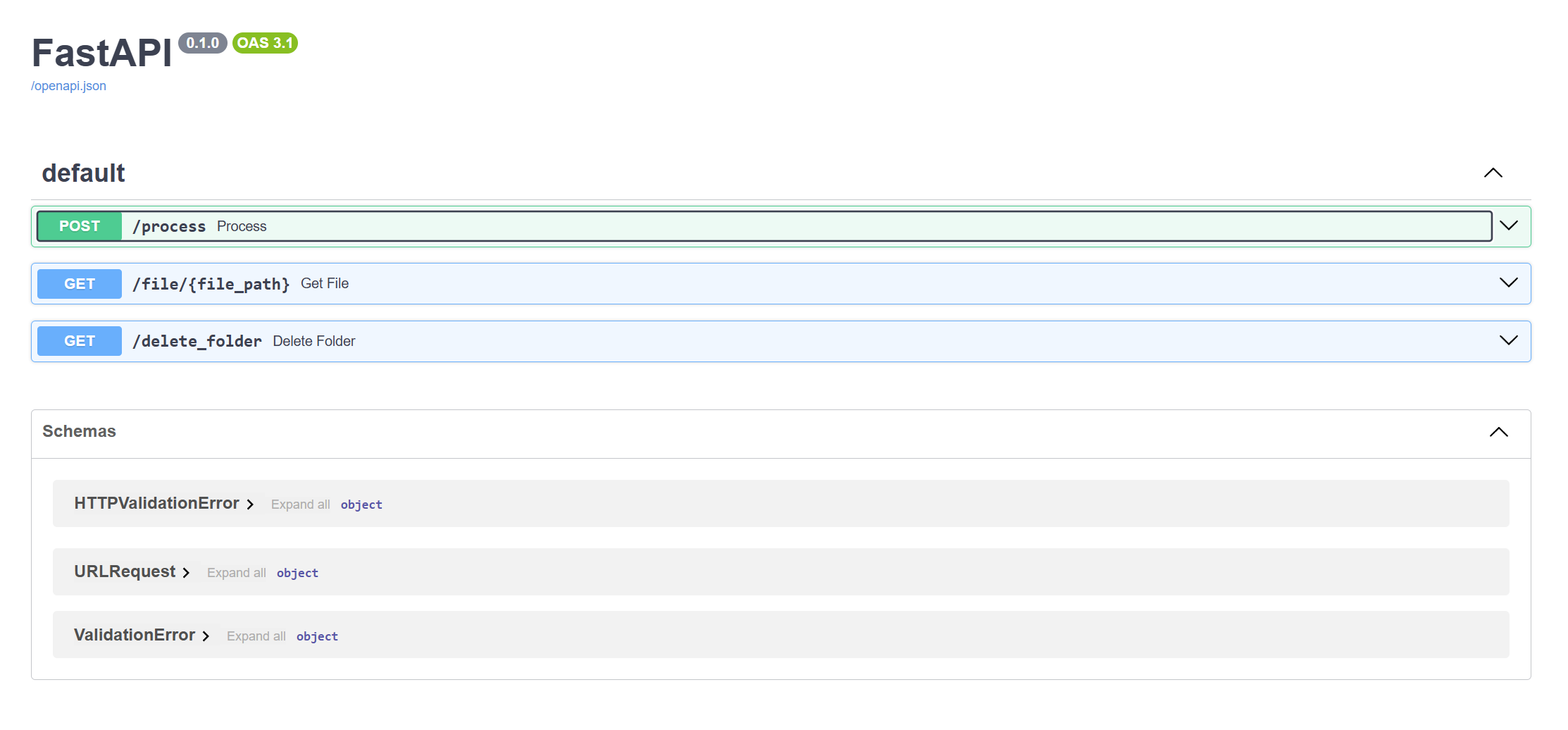
**Backend (server/)**

* main.py: The main FastAPI application.
* Dockerfile: Docker configuration for the FastAPI backend.
* requirements.txt: Python dependencies for the backend.
* component/Youtube\_reader/youtube\_reader2.py: YouTube video downloader.
* component/Video\_Transcriber2/video\_transcriber.py: Video transcriber.
* component/Summarizer2/summarize.py: Text summarizer.
* component/Note\_Generator2/note\_gen.py: Note generator.
* component/Mcq\_Generator2/mcq\_gen.py: MCQ generator.



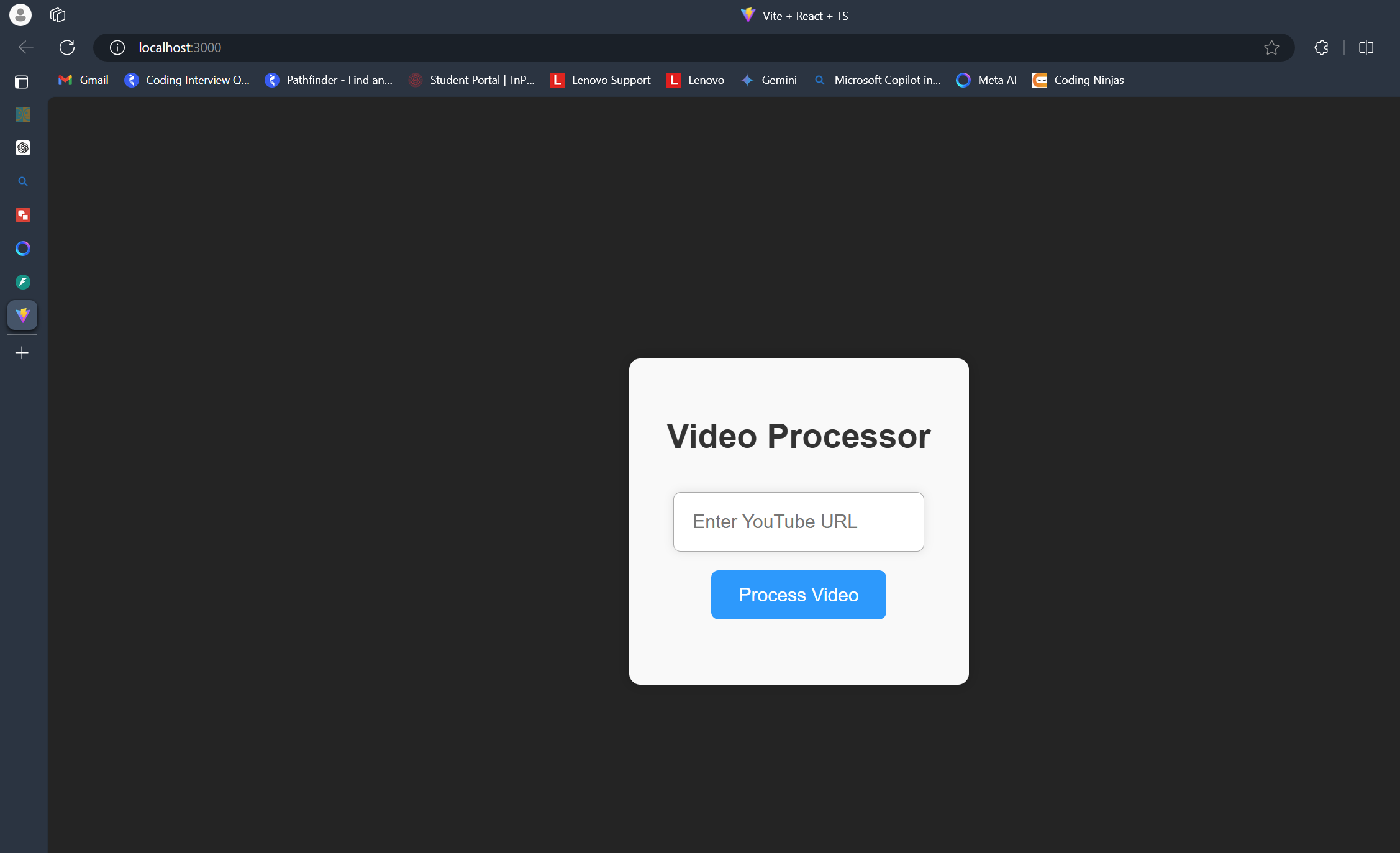
I have reduced endpoints as most of them were ‘GET’ request. Folder ‘fastapi’ will all endpoints

But Folder ‘frontend’ I have reduce endpoint because it was easier work with 3 endpoint.



**Frontend (frontend/)**

* src/VideoProcessor.tsx: React component for video processing.
* Dockerfile: Docker configuration for the React frontend.
* package.json: Node.js dependencies for the frontend.



**Methodology**

**Architecture**

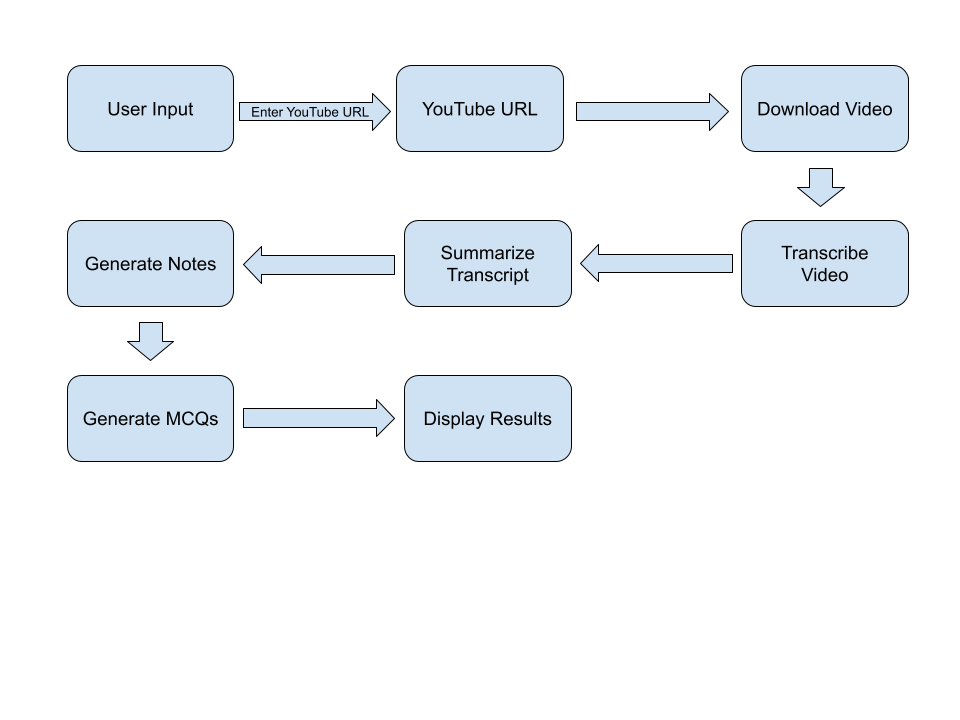
The system follows a **microservices architecture**, with distinct components for each processing step. Each component has a specific responsibility, promoting separation of concerns and scalability.

**Backend Workflow**

1. **Download Video**: Utilizes pytube to download the video and requests with BeautifulSoup to fetch video details.
2. **Transcribe Video**: Uses moviepy to extract audio and speech\_recognition to transcribe it.
3. **Summarize Transcript**: Utilizes the transformers library with the BART model to summarize the text.
4. **Generate Notes**: Summarizes the text in smaller chunks using the BART model.
5. **Generate MCQs**: Uses NLTK to create MCQs from the transcript text.

**Frontend Workflow**

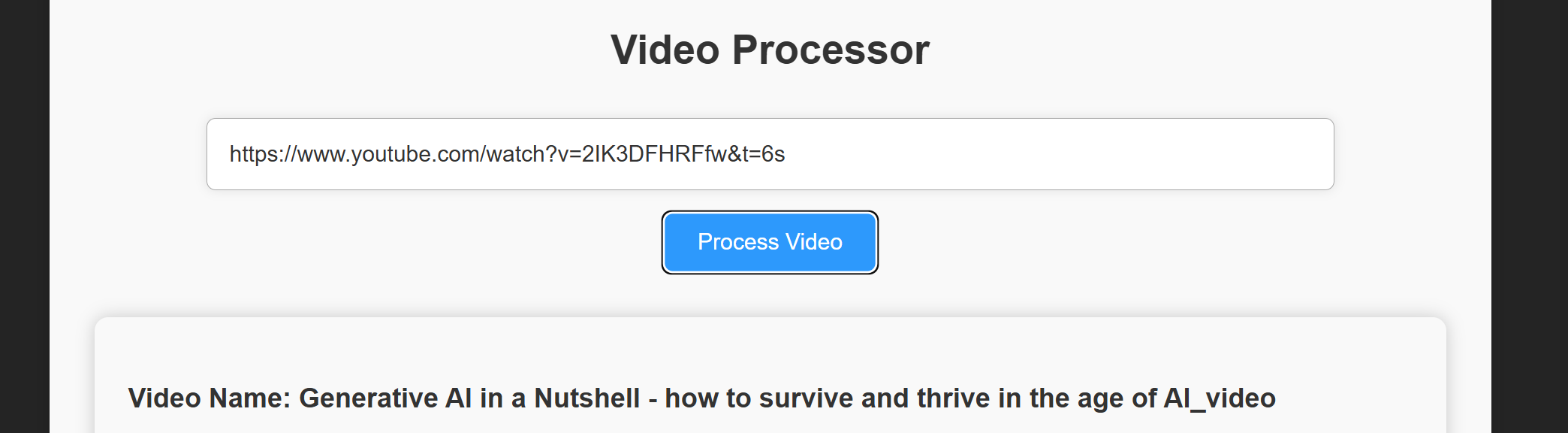
1. **User Input**: Users enter a YouTube URL.
2. **Trigger Processing**: The URL is sent to the backend via a POST request.
3. **Display Results**: The frontend fetches and displays the processed files (transcript, summary, notes, and MCQs).



**Detailed Explanation of Components**

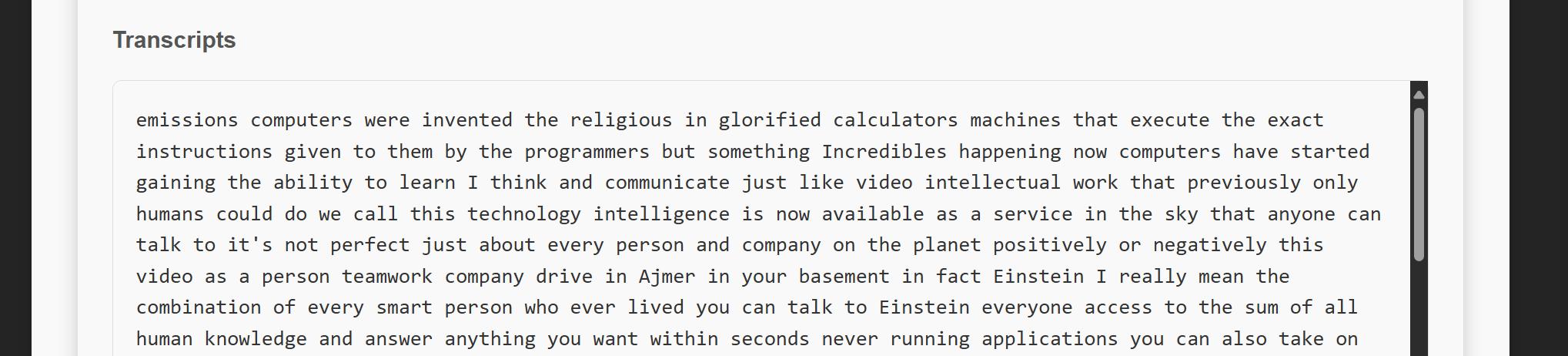
**YouTubeDownloader**

* The YouTubeDownloader component is responsible for handling the downloading of YouTube videos and extracting relevant video details. It uses the pytube library to download videos and requests with BeautifulSoup to fetch additional metadata such as title, description, views, and likes. This component ensures that videos are fetched securely and efficiently for further processing.



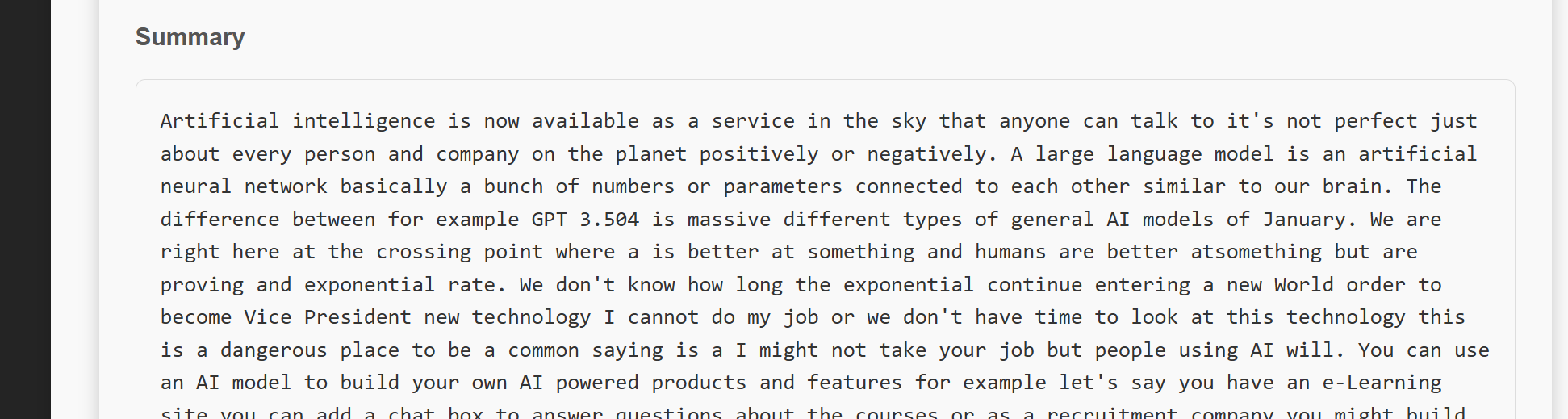
**VideoTranscriber**

* The VideoTranscriber component extracts audio from downloaded videos and transcribes this audio into text format. It utilizes the moviepy library to extract the audio stream from video files and speech\_recognition to perform the transcription. This component is crucial for converting spoken content from videos into textual data, which is essential for subsequent analysis and processing.



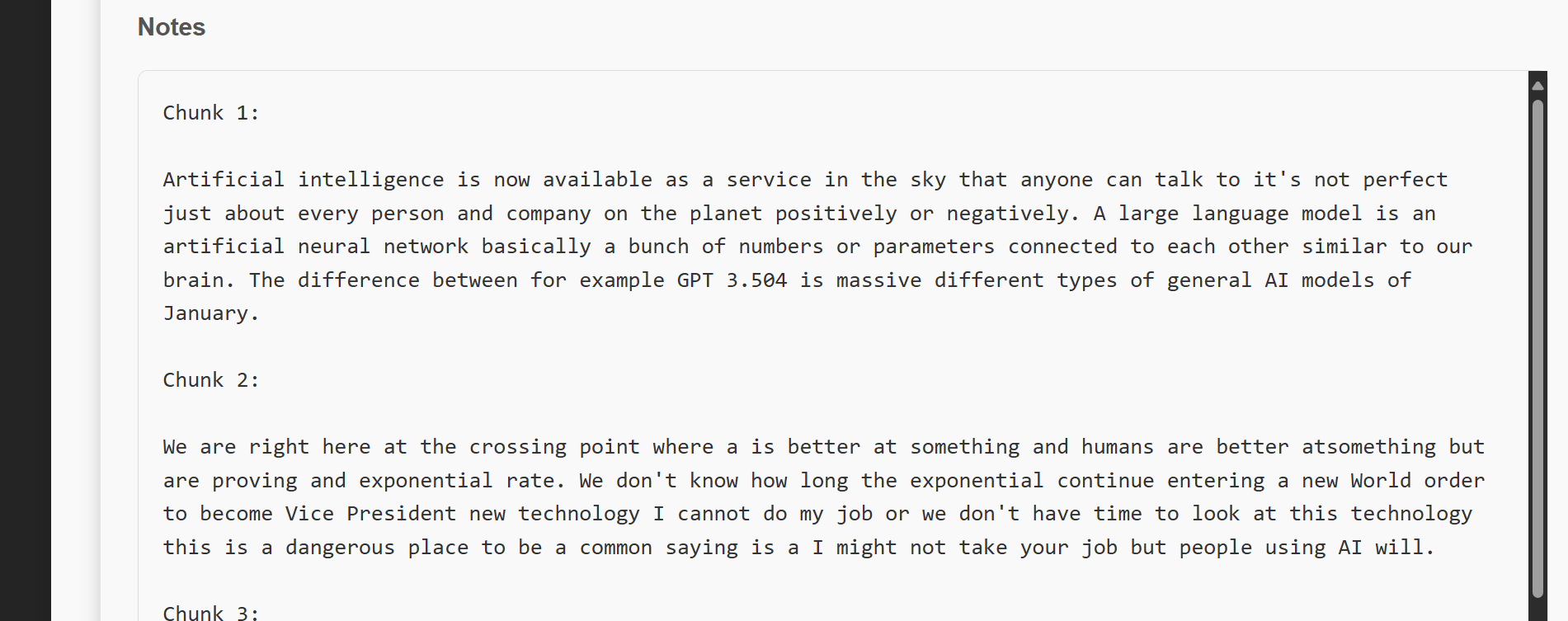
**TextSummarizer**

* The TextSummarizer component employs transformer-based models, such as BART from the transformers library, to summarize the transcribed text. It condenses lengthy transcripts into concise summaries, enhancing readability and digestibility of the content. This summarization process helps in generating high-level overviews of the video content, facilitating quicker understanding and review.



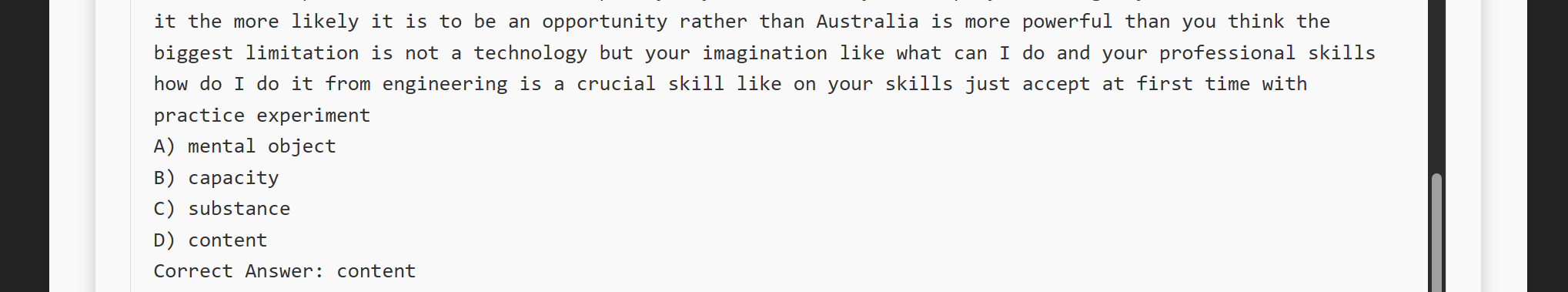
**NoteGenerator**

* The NoteGenerator component builds upon the text summarization process by generating detailed notes from the summarized text. It segments the summarized content into smaller chunks or key points, making it easier for users to review and retain important information. This component is particularly useful for educational purposes, aiding in the creation of structured study materials based on video content.



**MCQGenerator**

* The MCQGenerator component creates multiple-choice questions (MCQs) based on the transcribed and summarized text. It uses natural language processing (NLP) techniques, including part-of-speech tagging and synonym extraction from the NLTK library, to formulate questions and identify potential answer options. This component enhances interactivity by generating quizzes or assessments from video content, promoting active learning and engagement.



**React Frontend**

* Allows users to input a YouTube URL and view the processed files.

**VideoProcessor.tsx**

* The React frontend component, VideoProcessor.tsx, serves as the user interface for interacting with the automated video processing system. It allows users to input a YouTube URL, triggers the backend processing through API requests, and displays the processed results. This frontend ensures a seamless user experience by presenting the generated transcripts, summaries, notes, and MCQs in a structured and accessible format.

**Integration and Workflow**

Each component in the system operates independently but collaboratively to streamline the video processing workflow:

* **Input**: Users provide a YouTube URL via the React frontend.
* **Processing**: The backend components (YouTubeDownloader, VideoTranscriber, TextSummarizer, NoteGenerator, MCQGenerator) sequentially process the video content, from downloading and transcribing to summarizing and generating educational resources.
* **Output**: Processed files (transcript, summary, notes, MCQs) are then presented back to the user through the React frontend, enabling easy access and utilization of the generated content.

**Docker Configuration**

**Backend Dockerfile**

FROM python:3.10-slim

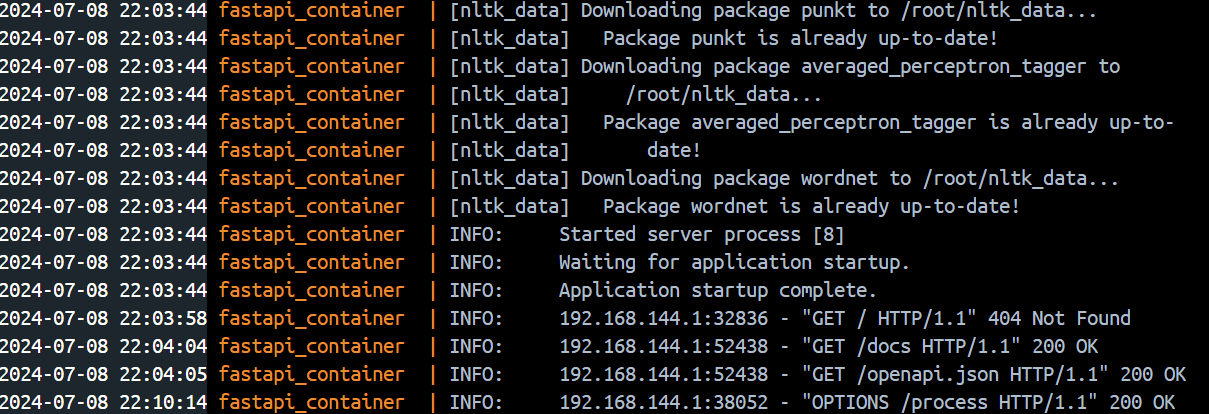
WORKDIR /usr/app

COPY . /usr/app/

RUN pip install fastapi uvicorn -r /usr/app/requirements.txt

EXPOSE 8000

CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000", "--reload"]



**Frontend Dockerfile**

FROM node:18

WORKDIR /app

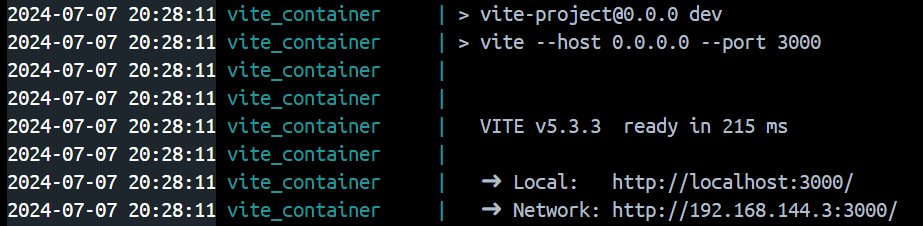
COPY package.json .

RUN npm install

COPY . .

EXPOSE 3000

CMD ["npm", "run", "dev", "--", "--host", "0.0.0.0", "--port", "3000"]



**Docker Compose**

services:

frontend:

build: ./

container\_name: vite\_container

expose:

- 3000

ports:

- "3000:3000"

volumes:

- /app/node\_modules

- ./:/app

depends\_on:

- backend\_service

backend\_service:

build: ./server

container\_name: fastapi\_container

expose:

- 8000

ports:

- "8000:8000"

volumes:

- /usr/app/\_\_pycache\_\_

- ./server:/usr/app

**Running the Project**

1. **Backend:**
   * Navigate to the server directory and build the Docker image.
   * Run the following commands:
   * docker build -t fastapi-backend .
   * docker run -p 8000:8000 fastapi-backend
2. **Frontend:**
   * Navigate to the frontend directory and build the Docker image.
   * Run the following commands:
   * docker build -t react-frontend .
   * docker run -p 3000:3000 react-frontend
3. **Using Docker Compose:**
   * From the root directory, run:
   * docker-compose up --build

**Key Points and Best Practices:**

1. **Separation of Concerns:**
   * Each component (downloader, transcriber, summarizer, note generator, MCQ generator) has a single responsibility, making the codebase more modular and maintainable.
2. **Error Handling:**
   * Comprehensive error handling is implemented to ensure the system’s robustness.
3. **Scalability:**
   * The microservices architecture allows individual components to be scaled independently.
4. **Asynchronous Processing:**
   * The FastAPI backend is designed to handle asynchronous processing, improving performance and responsiveness.
5. **User Experience:**
   * The React frontend provides a simple and intuitive interface for users to interact with the system.

This setup provides a fully automated workflow for processing YouTube videos, from downloading and transcribing to summarizing and generating educational content.