## Project Documentation: Automated Video Processing System

### Overview

This project is an automated video processing system designed to:

1. Download YouTube videos.

2. Transcribe their audio.

3. Summarize the transcriptions.

4. Generate notes.

5. Create multiple-choice questions (MCQs).

The system uses various Python libraries and frameworks and is built with a FastAPI backend and a React frontend.

### Project Structure

- `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.

- `Video\_Transcriber2/video\_transcriber.py`: Video transcriber.

- `Summarizer2/summarize.py`: Text summarizer.

- `Note\_Generator2/note\_gen.py`: Note generator.

- `Mcq\_Generator2/mcq\_gen.py`: MCQ generator.

- `frontend/`

- `src/VideoProcessor.tsx`: React component for video processing.

- `Dockerfile`: Docker configuration for the React frontend.

- `package.json`: Node.js dependencies for the frontend.

- `docker-compose.yml`: Docker Compose configuration to set up both backend and frontend services.

### FastAPI Backend

The FastAPI backend handles video processing through various endpoints.

#### Endpoints

- `/process`: POST endpoint to initiate the video processing workflow.

- `/file/{file\_path:path}`: GET endpoint to retrieve processed files.

- `/delete\_folder`: GET endpoint to delete temporary folders and files.

#### Example Code

Here's a simplified example of how the `/process` endpoint works:

```python

from fastapi import FastAPI, HTTPException

from pydantic import BaseModel

app = FastAPI()

class URLRequest(BaseModel):

url\_P: str

@app.post("/process")

async def process(request: URLRequest):

try:

# Download video

downloader = YouTubeDownloader()

video\_filename = downloader.download\_video\_and\_audio(request.url\_P)

# Transcribe video

transcriber = VideoTranscriber()

transcriber.transcribe\_video\_in\_chunks(video\_filename)

# Summarize transcript

summarizer = TextSummarizer()

summarizer.summarize\_text\_file(transcriber.transcripts\_path)

# Generate notes

note\_generator = NoteGenerator()

note\_generator.generate\_notes\_from\_file(transcriber.transcripts\_path)

# Generate MCQs

mcq\_generator = MCQGenerator(transcriber.transcripts\_path)

mcq\_generator.generate\_and\_save\_mcqs('mcqs.csv', 'mcqs.txt')

return {"message": "Processing completed successfully"}

except Exception as e:

raise HTTPException(status\_code=500, detail=str(e))

```

### React Frontend

The React frontend allows users to input a YouTube URL and view the processed files.

#### `VideoProcessor.tsx`

Here's a simplified example of the React component:

```tsx

import React, { useState } from "react";

import axios from "axios";

import "./VideoProcessor.css";

const VideoProcessor: React.FC = () => {

const [url, setUrl] = useState("");

const [response, setResponse] = useState(null);

const [error, setError] = useState("");

const handleProcess = async () => {

try {

const res = await axios.post("http://localhost:8000/process", { url\_P: url });

setResponse(res.data);

setError("");

} catch (err) {

setError("An error occurred during processing.");

}

};

return (

<div className="container">

<h1>Video Processor</h1>

<input type="text" value={url} onChange={(e) => setUrl(e.target.value)} placeholder="Enter YouTube URL" />

<button onClick={handleProcess}>Process Video</button>

{error && <p className="error">{error}</p>}

{response && <div>{JSON.stringify(response)}</div>}

</div>

);

};

export default VideoProcessor;

```

### Docker Configuration

#### Backend Dockerfile

```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

```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

```yaml

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

```

### How to Run

1. \*\*Backend\*\*: Navigate to the `server` directory and build the Docker image.

```sh

docker build -t fastapi-backend .

docker run -p 8000:8000 fastapi-backend

```

2. \*\*Frontend\*\*: Navigate to the `frontend` directory and build the Docker image.

```sh

docker build -t react-frontend .

docker run -p 3000:3000 react-frontend

```

3. \*\*Using Docker Compose\*\*: From the root directory, run:

```sh

docker-compose up --build

```

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