Documentation for Branchly

Branchly's goal is to help automate the workflow of creating digital assets from a youtube video. I am a long believer that long form content like youtube is at the core the most authentic content available.

Specification for MVP

The user is to be able to input a link to his youtube video. The link validity is checked. If the link is valid the system is to download the video, extract the audio from it and transcribe it. The output of this transcription is to be input for an LLM. The LLM is to digest the transcript and understand the key takeaways, the lessons learnt, the message that was portrayed, the hardships experienced, the overall tone of the video and other important intricacies of the video. Once it has understood the message of the video the LLM is to output a prompt. This prompt output should something like this:

{

"timestamps for Youtube video" :

"Description for Youtube video" :

"Hashtags for Youtube video" :

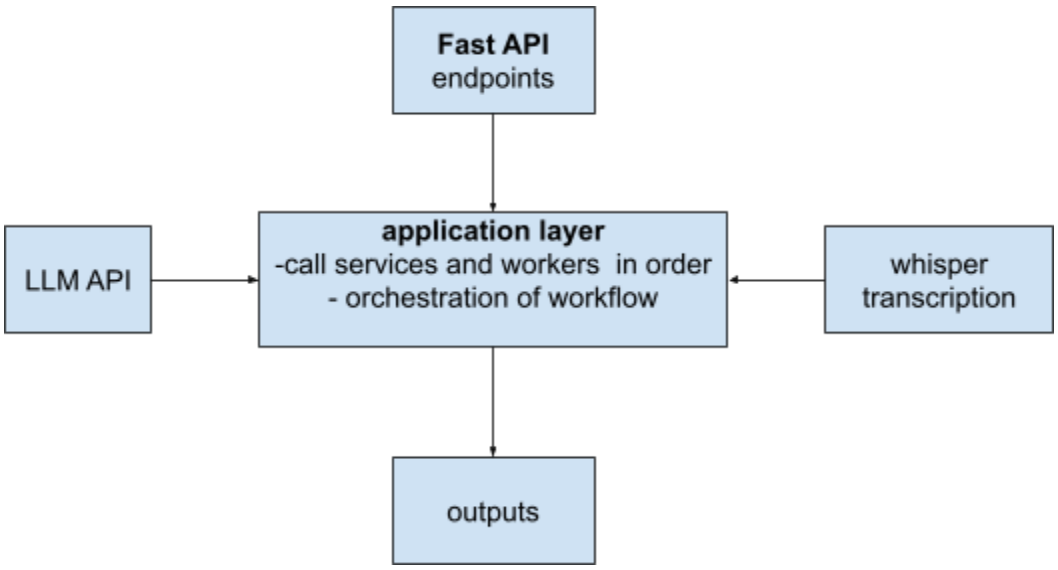
"Linkedin Post" :

"Twitter Post" :

}

The user of the MVP will be me. I am a youtuber and I am first building this product for myself. I don't need a front end for this. For this MVP we also don't need a database. We will be doing a dry run for v1. However the system is to be built in a scalable manner and should allow easy integration of databases for v2. The system is to be built for me but with the mindset that is to be given to other youtubers like me somewhere later when the product has matured. For this reason me and any LLM helping me prompt is to keep that under consideration.

Lets identify the basic MVP architecture for this project



Now lets identify the MVP endpoints needed

```
1. POST /upload_video
  - Input: YouTube URL
  - Task: Download (using yt-dlp if URL), extract audio, store in data/uploads/ folder
(later DB like Supabase)
  - Output: video_id, status
```

```
2. POST /transcribe/{video_id}
  - Input: video_id
  - Task: Send audio to Whisper API, save transcript to
/data/transcripts/{video_id}.txt
  - Output: transcript_text, status
```

```
3. POST /generate_assets/{video_id}
  - Input: video_id
  - Task: Fetch transcript → Send to LLM like GPT → Generate JSON of assets
  - Output: JSON with generated assets (title ideas, tweets, LinkedIn post,
timestamps, etc.) and save to /data/outputs/{video_id}.json
```

```
4. GET /assets/{video_id}
  - Input: video_id
  - Output: All generated assets related to that video (read JSON)
```

Scalability of the product

Design Choice	MVP Behavior	Scalable Future Behavior
Storage	Uses flat files in `/data/`	Swap with a database or S3
Services	Functions in `/services/`	Replace with microservices or async tasks
Workflows	Sequential, synchronous	Wrap in background queue (Celery, Redis)
Auth	None	Add `User` model + JWT auth later
Config	`.env` file for keys	Replace with Docker secrets or AWS envs
Hosting	Local FastAPI + uvicorn	Deploy to Render / Railway / AWS Lambda

Here is the starting file structure to start off with

```
Branchly/
├── app/
│   ├── main.py               # FastAPI entry point
│   ├── routes/
│   │   ├── video_routes.py     # /upload, /transcribe, /generate
│   ├── services/
│   │   ├── youtube_service.py  # Downloads video/audio
│   │   ├── whisper_service.py  # Transcribes
│   │   ├── llm_service.py      # Generates content
│   │   └── storage_service.py  # File-based (later: swap for Supabase)
│   ├── core/
│   │   ├── config.py          # API keys, environment vars
│   │   └── utils.py           # Helper functions
│   ├── models/
│   │   ├── schemas.py         # Pydantic models (Video, Asset)
│   └── workflows/
│       └── video_pipeline.py   # Orchestrates the process
├── data/
│   ├── uploads/               # raw .mp4 or .mp3
│   ├── transcripts/           # text files
│   ├── outputs/               # generated content JSON
│   └── logs/
├── .env
└── requirements.txt
```