TubeIQ – YouTube Knowledge Companion

Project Manual Documentation

---

1. Project Overview

TubeIQ is an AI-powered web application that allows users to summarize YouTube videos and interact with their content via chat. It uses **LangChain, RAG (Retrieval-Augmented Generation), FAISS vector storage, and OpenAI APIs** to provide intelligent responses to user queries about a video.

Main Features: - Paste a YouTube link → get a structured summary of the video. - Chat with the video's content using AI-driven RAG pipeline. - User authentication via **Clerk**. - Rate-limiting to control OpenAI API usage. - Live deployment on **AWS** with frontend (Next.js + Amplify) and backend (FastAPI + Docker + EC2).

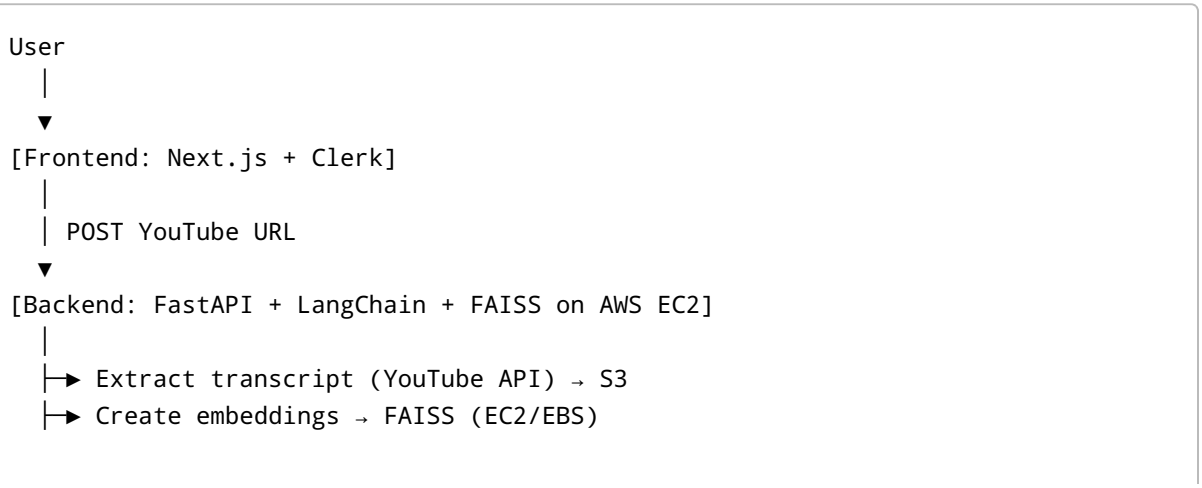
Tech Stack: - Frontend: React (Next.js) + Clerk authentication - Backend: FastAPI + Python + LangChain + FAISS + OpenAI API - Storage: AWS S3 (transcripts & summaries), EBS (FAISS index) - Deployment: AWS EC2 (backend), AWS Amplify (frontend) - Containerization: Docker (backend) - Monitoring: AWS CloudWatch

---

2. System Architecture

Components: 1. **Frontend:** Next.js UI for inputting YouTube links, displaying summaries, and chatting. 2. **Backend:** FastAPI server managing: - Transcript extraction (YouTube API) - Summary generation (OpenAI GPT via LangChain) - RAG chat queries (FAISS + LangChain) 3. **Storage:** - S3 for transcripts and summaries - Persistent FAISS vector store on EC2/EBS 4. **Security:** - Clerk for authentication - AWS Secrets Manager for OpenAI keys - Rate limiting (slowapi)

Workflow Diagram:



```
├─► Summarize video → S3 → Return to frontend
└─► Chat RAG query → Retrieve from FAISS → OpenAI → Return to frontend
```

---

3. Installation & Setup

3.1 Prerequisites

- Python 3.10+
- Node.js 18+
- Docker
- AWS Account (EC2, S3, Secrets Manager, Amplify)
- OpenAI API Key

3.2 Backend Setup (FastAPI + LangChain + FAISS)

1. **Clone repository**

```
git clone https://github.com/your-repo/TubeIQ.git
cd TubeIQ/backend
```

2. **Create virtual environment**

```
python -m venv venv
source venv/bin/activate  # Linux/Mac
venv\Scripts\activate     # Windows
```

3. **Install dependencies**

```
pip install -r requirements.txt
```

4. **Set environment variables (.env)**

```
OPENAI_API_KEY=your_openai_key
AWS_ACCESS_KEY_ID=your_aws_key
AWS_SECRET_ACCESS_KEY=your_aws_secret
S3_BUCKET_NAME=your_bucket
```

5. **Run backend locally**

```
uvicorn app.main:app --reload --host 0.0.0.0 --port 8080
```

6. **Optional: Dockerize backend**

```
docker build -t tubeiq-backend .
docker run -p 8080:8080 tubeiq-backend
```

3.3 Frontend Setup (Next.js + Clerk)

1. **Navigate to frontend**

```
cd ../frontend
```

2. **Install dependencies**

```
npm install
```

3. **Set environment variables (.env.local)**

```
NEXT_PUBLIC_BACKEND_URL=http://<EC2-PUBLIC-IP>:8080
CLERK_FRONTEND_API=<clerk_frontend_api>
CLERK_API_KEY=<clerk_api_key>
```

4. **Run locally**

```
npm run dev
```

---

4. Deployment Instructions

4.1 Backend (AWS EC2 + Docker)

1. Launch EC2 instance (Ubuntu 22.04 recommended).
2. SSH into EC2, install Docker & Docker Compose.
3. Copy backend Docker image or clone repo on EC2.
4. Build and run container:

```
docker build -t tubeiq-backend .
docker run -d -p 8080:8080 --name tubeiq tubeiq-backend
```

5. Configure security group: open **port 8080** for frontend access.

4.2 Frontend (AWS Amplify)

1. Connect GitHub repo to AWS Amplify.
2. Configure **build settings**: `npm install` → `npm run build` → `npm run start`.
3. Set environment variables in Amplify console.
4. Deploy → Amplify provides a public URL.

---

5. User Workflow

1. User signs in via **Clerk**.
2. User pastes a **YouTube link**.
3. Backend extracts transcript → stores in **S3**.
4. Backend generates **embeddings** → stores in **FAISS**.
5. Backend generates **summary** → returns to frontend.
6. User can **chat with video content**; backend fetches relevant chunks from FAISS and queries OpenAI.
7. Rate limiter ensures **limited usage per user**.

---

6. Backend Endpoints

Endpoint	Method	Input	Output	Description
`/extract`	POST	`{ "url": "<YouTube Link>" }`	`{ "video_id": "abc123", "transcript": "..." }`	Extract transcript from video
`/summarize`	POST	`{ "video_id": "abc123" }`	`{ "summary": {...} }`	Generate summary from transcript
`/chat`	POST	`{ "video_id": "abc123", "query": "..." }`	`{ "answer": "..." }`	Answer user query using RAG

---

7. Rate Limiting

- Implemented with **slowapi** in FastAPI: 5 requests per minute per user/IP.
- Protects OpenAI credit ($5).
- Optional upgrade: Redis or API Gateway throttling for production scaling.

---

8. FAISS & RAG Notes

- **FAISS vector store** keeps embeddings for each video.
- Stored **persistently on EC2 EBS volume**.
- Backup to S3 for safety.
- Retrieval workflow: top-N relevant chunks → OpenAI → response.

---

9. Security & Secrets

- **Clerk:** user authentication & session management.
- **AWS Secrets Manager:** secure OpenAI keys.
- **Rate limiting:** prevent abuse.
- **HTTPS:** use Amplify built-in SSL or CloudFront.

---

10. Monitoring & Logging

- **AWS CloudWatch** tracks backend logs and uptime.
- Errors and API usage can be monitored.
- Alerts for high error rates can be configured.

---

11. Future Enhancements

- Support **multiple videos in one session**.
- Integrate **Whisper API** for videos without transcripts.
- Store **chat history per user** in DynamoDB.
- Deploy backend to **ECS** for auto-scaling.
- Add **analytics dashboard** for most asked questions.

---

12. Demo Video Outline

1. Show **login via Clerk**.
2. Paste a YouTube link → show transcript + summary.
3. Ask questions in chat → AI answers appear instantly.

4. Explain **backend on EC2**, **frontend on Amplify**, **Docker container**, **FAISS embeddings**, **rate limiting**, and **OpenAI integration**.