YouTube Video Uploader - Project Documentation

Project Title: YouTube Video Uploader

This is a full-stack web application that allows users to upload videos to YouTube through a user-friendly interface. The project leverages Google OAuth for secure login and the YouTube Data API for uploading videos with metadata. It uses React for the frontend and Spring Boot for the backend.

Technologies Used

Frontend (React):

- React JS, Axios, React Router (if used), Vite

- HTML/CSS/JSX for UI

- Google OAuth2 login

Backend (Spring Boot):

- Spring Web, Spring MVC, Spring Boot Devtools

- MultipartResolver for file uploads

- Java HTTP client for REST communication

- Jackson for JSON processing

External APIs:

- YouTube Data API v3 with OAuth 2.0 authorization

- Resumable Upload for large file support

Backend Flow (Spring Boot)

1. File Upload Handling (YoutubeUploadController.java):
   - POST endpoint /api/youtube/upload accepts MultipartFile and metadata.
   - Passes data to the upload service.

2. Upload Service (YoutubeVideoUpload.java):
   - Builds an HTTP POST request to get a resumable upload URL.
   - Uses PUT request to upload the video file to that URL.
   - Adds necessary headers: Authorization, Content-Type, X-Upload-Content-Length.
   - Handles HTTP response and status.

Frontend Flow (React)

1. Upload Form (Upload.jsx):
   - Input fields for video title, description, and file.
   - Retrieves accessToken from Google login.
   - Sends FormData using Axios to backend.

2. Axios Request:
   - POST to http://localhost:8989/api/youtube/upload.
   - Handles success and failure messages.

Google OAuth 2.0 Flow

- Users authenticate via Google.
- Returns an access_token to the frontend.
- Token is included in the upload request for authorization.

Full Flow Summary

1. User logs in via Google.
2. Selects video file and enters metadata.
3. React frontend sends this data to the Spring backend.
4. Backend requests a resumable upload URL from YouTube.
5. Video is uploaded to YouTube via PUT request.
6. Success or failure response sent back to user.

Key Concepts Used

- OAuth 2.0: Secure authorization protocol.
- Resumable Upload: Googles protocol for large files.
- REST API: Backend exposes clean endpoints.
- CORS: Handled to allow frontend-backend communication.
- Spring MVC: Clean separation using controller/service layer.
- Axios + FormData: Uploads from React.

Security Notes

- Do not hardcode secrets in code.

- Use .env files and environment variables.

- Add application.properties to .gitignore.

- GitHub Push Protection blocked secret push.

Interview-Worthy Highlights

- Complete full-stack integration with Google services.

- Secure OAuth 2.0 login flow.

- Resumable upload handling with correct headers.

- Spring MVC best practices.

- Axios file upload with FormData.

- Robust error and exception handling.

- Clean architecture following separation of concerns.