MediaProject

Media ingestion and management platform with CDN, PostgreSQL, MongoDB, Redis, JWT and async workers, Docker.

Overview

MediaProject is a Django REST Framework *(last version only)* backend that allows authenticated users to ingest videos and miniature (BLOB format), store them on a CDN, manage technical and editorial metadata, and expose them through a REST API.

The system must:

- **Provide full CRUD on media resources.**
- **Use PostgreSQL for core entities (users, roles, media, jobs, URL HTTP, miniature) with ORM.**
- **Use MongoDB for detailed technical metadata with ODM.**
- **Use Redis for caching and deduplication (avoid duplicate uploads).**
- **Use background tasks and Redis for async processing (CDN upload (imagekit.io) annexe 2-3-4 with metadata retrieve).**
- **Protect the API with a JWT middleware.**
- **Offer an admin mode to manage users and roles.**
- **Be containerized with Docker and covered by unit tests ~80%.**

Roles

- **Admin – manage users and roles, full access to all media and jobs.**
- **Media Operator – upload new media, manage own media (update/delete), view job status.**
- **Viewer – read-only access to allowed media.**

Data stores

- PostgreSQL: `User`, `Media`, `MediaJob`, `HTTP URL`.
- MongoDB: `media_metadata` collection linked to `Media` by UUID.
- Redis: dedup keys `media:title:{slug}` → `media_uuid`, optional title search.

Core endpoints

- `GET ping/` → return pong.
- `GET version/` →return version.
- `POST auth/signup/` → create user.
- `POST auth/signin/` →JWT.
- `POST auth/signup/` → create user.
- `POST auth/signin/` →JWT.
- `POST media/` → create media (video + miniature), Redis dedup, enqueue job, write meta data.
- `GET media/` → list media with pagination.
- `GET media/{uuid}/` → media details with pagination.
- `PATCH media/{uuid}/` → update (role-based).
- `DELETE media/{uuid}/` → delete (role-based).
- `GET media/{uuid}/jobs/` → job history with pagination.
- `GET search?title=...` → title search (Redis + DB) with pagination.

Async processing

- `upload_to_cdn(media_id)` → upload to CDN, update `cdn_url` and status.
- `extract_metadata(media_id)` → extract video metadata, store in MongoDB.

Swagger

- Implement Swagger

Security

- JWT middleware + DRF authentication & permission classes.

Validation data

- **Use pydantic.**

Plan Architecture (cf page 1)

- **`web` (Django) port 8000, `db` (Postgres) port 5432, `mongo` (MongoDB) port 27017, `redis`(Redis) port 6379, `background tasks`, via docker-compose.**

Testing

- **JWT auth, role-based permissions, Redis dedup logic, async status updates, Postgres/Mongo consistency.**

Postman

- **Add a Postman collection to automate API calls.**

README

- **Add a README to explain your architecture and your logic.**

Bonus 🤑

- **Front end for the Application**

⚠️ WARNING

- **English names for all functions, classes, variables, files, etc.**
- **Clean naming & conventions (PEP8 (use black), clear responsibility).**
- **Short functions (no god-functions doing everything).**

- **OOP only: classes, services, repositories, etc.**
- **Clean architecture: clear layers (API / services / repositories / infrastructure).**
- **Clean commits and repo too (atomic, readable messages, no junk).**

- **Do not depend on ChatGPT for critical work — otherwise, you will retake the exam.**

HELPFUL RESSOURCES:

docs.djangoproject.com/en/5.2/

https://django-background-tasks.readthedocs.io/en/latest/

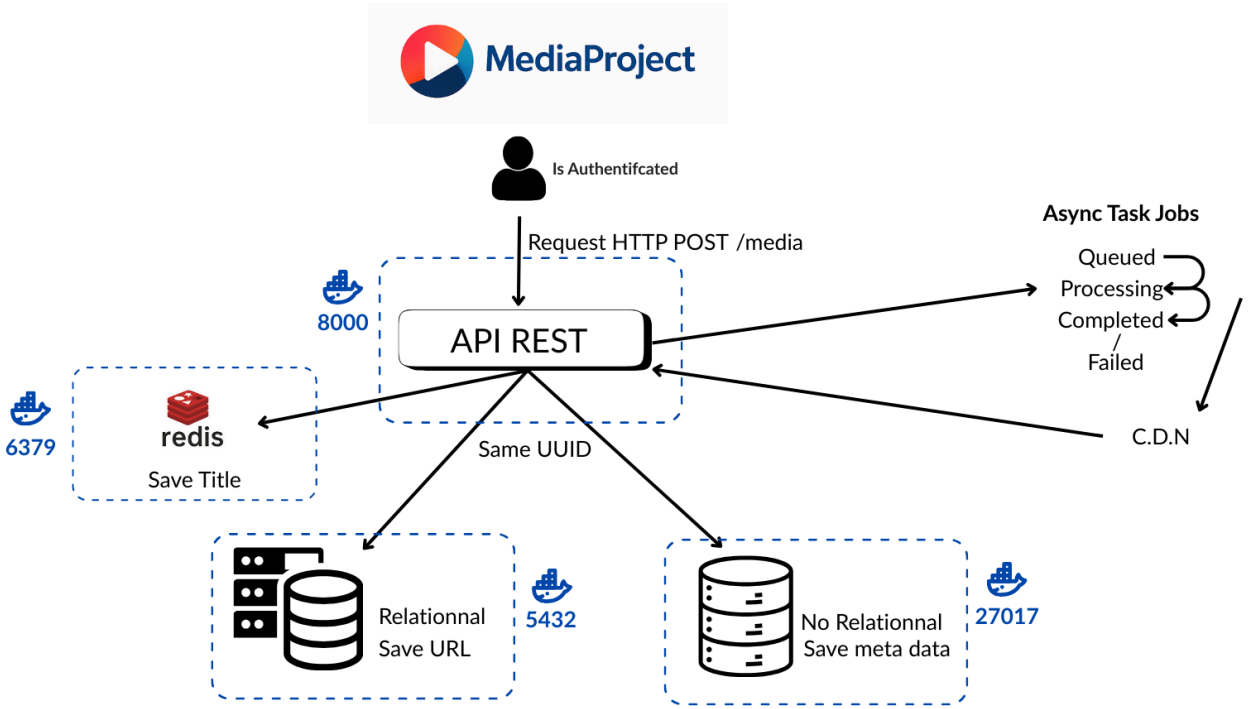
https://imagekit.io/

https://docs.pydantic.dev/latest/

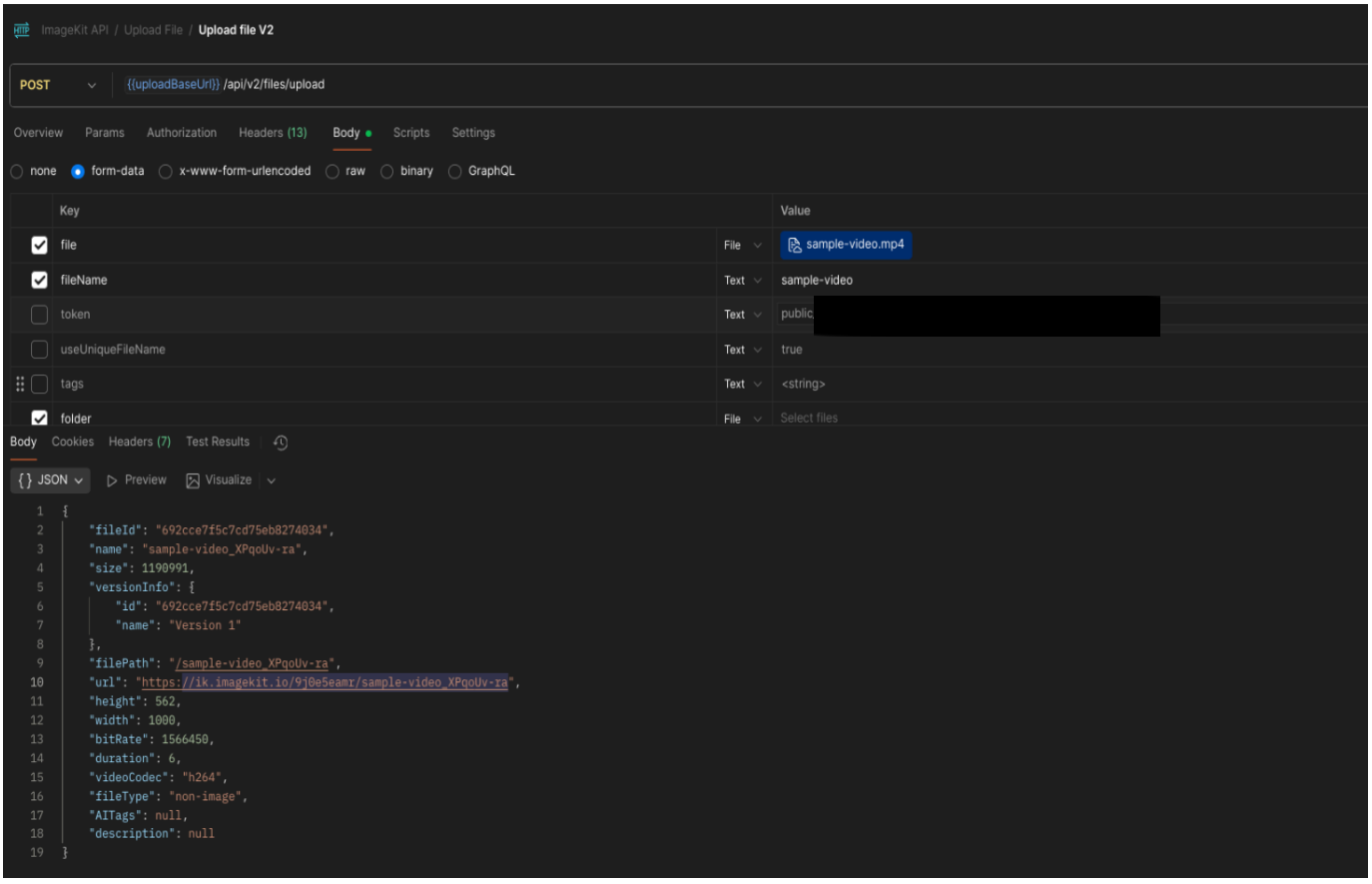
https://pypi.org/project/black/

https://pypi.org/project/python-ffmpeg/

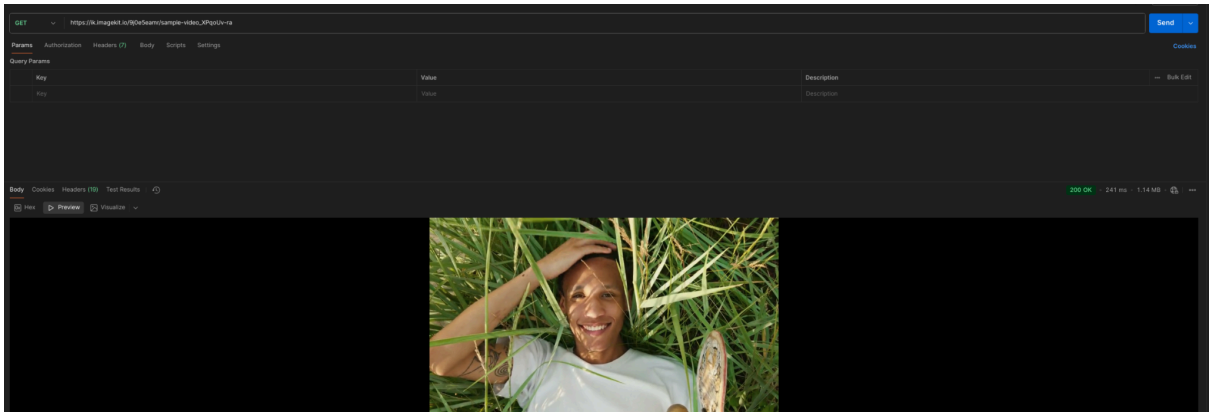
Annexe 1:



Annexe 2:



Annexe 3:



Annexe 4: