

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/>

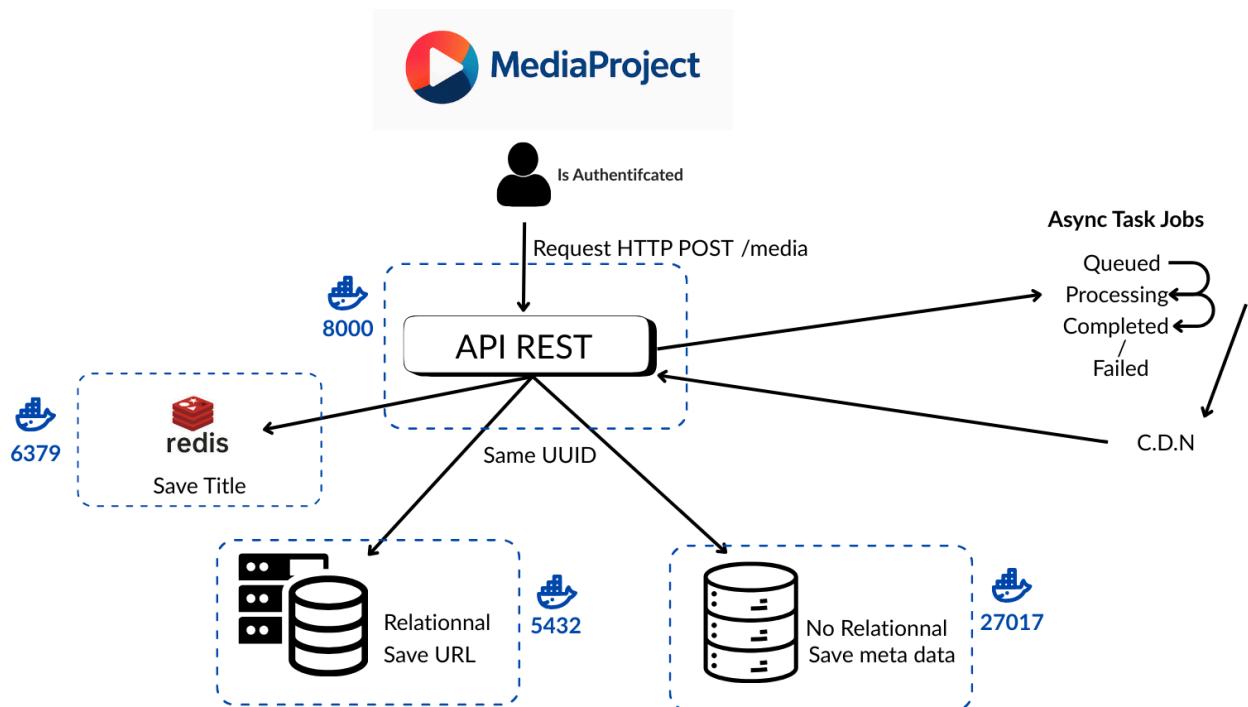
<https://imagekit.io/>

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

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

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

Annexe 1:



Annexe 2:

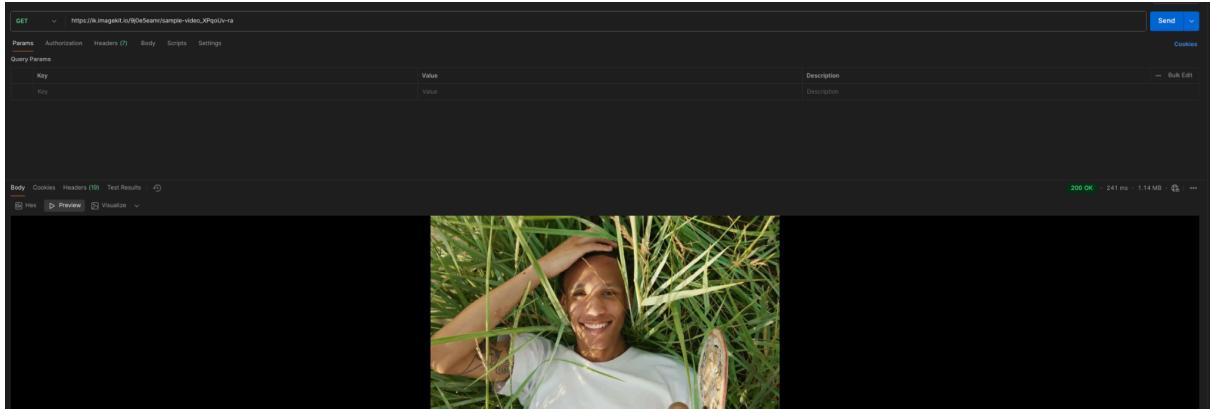
The screenshot shows a Postman collection for the ImageKit API's "Upload File V2" endpoint. The request method is "POST" and the URL is `{{uploadBaseUrl}}/api/v2/files/upload`. The "Body" tab is selected, showing a "form-data" payload with the following fields:

Key	Value
<input checked="" type="checkbox"/> file	File <input type="file" value="sample-video.mp4"/>
<input checked="" type="checkbox"/> fileName	Text <input type="text" value="sample-video"/>
<input type="checkbox"/> token	Text <input type="text" value="public"/>
<input type="checkbox"/> useUniqueFileName	Text <input type="text" value="true"/>
<input type="checkbox"/> tags	Text <input type="text" value="<string>"/>
<input checked="" type="checkbox"/> folder	File <input type="file"/>

Below the body, the "Headers" tab shows 13 headers. The "Body" tab also includes "JSON" and "Preview" sections.

```
1 {  
2   "fileId": "692cce7f5c7cd75eb8274034",  
3   "name": "sample-video_XPqoUv-ra",  
4   "size": 1190991,  
5   "versionInfo": {  
6     "id": "692cce7f5c7cd75eb8274034",  
7     "name": "Version 1"  
8   },  
9   "filePath": "/sample-video_XPqoUv-ra",  
10  "url": "https://ik.imagekit.io/9j@e5eamr/sample-video_XPqoUv-ra",  
11  "height": 562,  
12  "width": 1000,  
13  "bitRate": 1566450,  
14  "duration": 6,  
15  "videoCodec": "h264",  
16  "fileType": "non-image",  
17  "AITags": null,  
18  "description": null  
19 }
```

Annexe 3:



Annexe 4:

A screenshot of the `imagekit.io` dashboard under the `media-library` section. The top navigation bar includes links for Most Visited, Fedora Docs, Fedora Magazine, Non, merci, Fedora Project, and User. On the left, there's a sidebar with options like Get started, Usage analytics, Purge cache, Media library (which is selected and highlighted in blue), Media collections, Saved searches, Bookmarks, Path policy, External storage, URL endpoints, and Settings. The main content area shows a search bar with the placeholder "Start typing to search". Below it, there's a "Folders" section with a single folder named "home". In the "Files" section, there is a large preview of a video file. The video thumbnail shows a white document with a play button icon. The file details below the thumbnail are: `sample-video_XPqoUv-ra`, MP4, 1.14 MB, 1000x562.