🌐 Objective

Build a **Movie Management Platform** that allows users to perform CRUD (Create, Read, Update, Delete) operations on movies. The system should consist of a **Python Django backend** and a **frontend** (choose one: **(Preferred)Angular**, React, or Vue). The frontend and backend must communicate over a **REST API**.

🚀 Time Limit: 3 days

This test is designed to evaluate your ability to build, integrate, and document a basic full-stack application under time constraints.

🧠 Backend (Python Django)

Requirements:

- Create a Django project with a single app: `movies`
- Implement a Movie model with the following fields:
  - `title`: string
  - `description`: string
  - `date_added`: date
  - `video_file`: file (uploaded video file)
- Configure media settings in Django to support file uploads:
  - Add `MEDIA_URL` and `MEDIA_ROOT`
  - Update `urls.py` to serve media files in development
- Implement the following API endpoints using Django REST Framework (DRF):
  - `GET /movies/`: list all movies
  - `GET /movies/<id>/`: get details of a specific movie
  - `POST /movies/`: create a movie with video upload
  - `PUT /movies/<id>/`: update a movie (support file replacement)
  - `DELETE /movies/<id>/`: delete a movie
- Ensure proper use of serializers, models, views, and file handling.

🖼️ Frontend (Angular / React / Vue)

Requirements:

- Choose one framework: **(Preferred)Angular**, React, or Vue
- Use Axios (or HttpClient) to interact with the backend
- Implement the following features:
  - List movies (title, date added)
  - View movie details
  - Create a new movie with file upload
  - Edit a movie (including replacing the video)
  - Delete a movie
  - Play movie using the uploaded video file via `<video>` tag
- UI must include basic validations and feedback (e.g. loading indicators, success/error messages)
- **TIP:** Netflix style UI

🎓 Bonus (Optional)

- Use of Docker for deployment is a bonus!
- Use of Angular is preferred!
- Use Django background tasks (e.g., Celery + Redis) to process video or file handling or thumbnail generation or HLS generation for streaming
- User Authentication (JWT)

📄 Documentation (README)

Include a `README.md` that contains:

- Tech stack used
- Prerequisites (e.g. Python, Node.js, package managers)
- Setup instructions for both backend and frontend
- Known Issues or Limitations: Mention any known bugs, missing features, or limitations encountered during development.
- Demo Instructions: Include instructions or links for accessing the demo video, and steps on how to test file upload and video playback manually.

📬 Submission

- Commit all your code to a **GitHub repository**.
- Ensure that the repository is well-structured and contains all necessary files.
- Share the GitHub repository link for evaluation.
- Record and submit a **short demo video** showing how the application works.
- **Deadline**: 3 days

📊 Evaluation Criteria

Area	Points	Criteria
Backend (Django)	30	RESTful API correctness, file handling, model/schema design, code quality
Frontend (Vue/React/Angular)	20	Proper REST API integration, file upload implementation, video playback
Frontend UI/UX	10	Clean UI UX
Integration	15	Successful end-to-end connection between frontend and backend
Code Organization	5	Folder structure, naming, separation of concerns
Documentation	5	Clear README, setup steps, helpful notes
Bonus	15	Dockerized deployment, Angular Frontend, etc.

Total: 100 Points

Good luck, and have fun building!