Equevu – Senior Software Engineer Task: Implement a Tiny HR System API

You are required to develop an API for a minimal HR system, allowing job applicants to register as candidates and upload their resumes, while HR managers can log in, view the list of candidates, and download their resumes.

Functional Requirements

Candidate Functionality

- **Registration Endpoint:** Candidates should be able to register with the following fields:
  - **Full Name** (Required)
  - **Date of Birth** (Required)
  - **Years of Experience** (Required, Integer)
  - **Department ID** (Required, Enum: `IT`, `HR`, `Finance`)
  - **Resume Upload** (Required, must be PDF or DOCX)
- **Resume Handling:**
  - Validate file type and size (Max: **5MB**).
  - Store files in a structured manner (e.g., by user ID).
  - Design the system to allow future migration to cloud storage (S3, Azure, etc.).

Admin Functionality

- **List Candidates Endpoint (Admin Only):**
  - Returns a **paginated** list of candidates ordered by **registration date (descending order)**.
  - Data returned:
    - Full Name
    - Date of Birth
    - Years of Experience
    - Department
  - Allow filtering by department.
- **Resume Download Endpoint (Admin Only):**
  - Allows downloading a candidate's resume by their ID.
  - Implement proper error handling (e.g., invalid ID, file not found).

Authentication/Authorization

- No full authentication system required. Instead, an admin request must contain the header:
  - `X-ADMIN=1` (Indicates the request is from an admin user).
- Candidates can register without authentication.

Technical Requirements

- **Backend:** Python 3 + Django (or Django Rest Framework or any Python Framework).
- **Database:** PostgreSQL or MySQL.
- **File Storage:**
    - Files must be stored in a way that allows future storage system changes.
    - Implement an abstraction layer for storage, making it easy to switch between local and cloud storage (AWS S3, Google Cloud Storage, etc.).
- **Security & Validation:**
    - Validate input fields (e.g., proper date format, non-negative years of experience).
    - Prevent duplicate registrations using a unique constraint (e.g., unique email or phone number).
    - Implement basic logging for key events (e.g., user registration, file upload/download).
- **Performance Considerations:**
    - Optimize queries (e.g., use indexes where appropriate).
    - Ensure the system can handle **at least 100,000 candidate records** efficiently.
    - Paginate the list API for better performance.

Evaluation Criteria

- **Code Quality:** Clean, modular, and maintainable code following best practices.
- **Functional Completeness:** Does the API meet all the requirements?
- **Security Considerations:** Proper validation, file handling, and error responses.
- **Scalability & Extensibility:** How well does the system handle future growth and modifications?

Bonus Points

- **Frontend:** Implement a simple UI using React (or the JS Library/Framework of your choosing) to:
    - Register candidates
    - List applicants (admin view)
    - Download resumes (admin only)
- **Testing:**
    - Unit tests (Pytest/Django test framework).
    - Integration tests (API-level).
- **Cloud File Storage:** Implement Amazon S3 or another cloud storage option.

- **Docker:** Provide a `Dockerfile` and `docker-compose.yml` to simplify setup.

What to Deliver

- A **public GitHub repository** with:
  - The full implementation.
  - A **README** file containing:
    - Setup instructions (how to install dependencies and run the application).
    - Database setup steps (SQL script or Django migration command).
    - API documentation (can be Swagger/OpenAPI or simple markdown).
  - A `requirements.txt` or `pyproject.toml` file for dependency management.