Duration: 8 hours

Objective: Build a **backend** for a **Job Listing Platform API** where recruiters can post jobs and job seekers can apply to jobs. The task should focus on backend development using **Django, Flask, or Node.js** and include the use of a **queue system** to handle background tasks. Front-end development, Docker integration, and deployment will be treated as bonuses.

Task Requirements:

1. **Backend Development:**
   - Implement **user authentication** using **JWT** or **session-based authentication**.
   - Design and develop APIs for:
     - **Job Postings**: Recruiters can create, update, and delete job postings.
     - **Job Listings**: Job seekers can view available jobs with the option to filter by title, location, or company.
     - **Job Applications**: Job seekers can apply to a job, submitting their details (name, email) and uploading an optional resume (file upload).
   - Implement **role-based access control**:
     - **Recruiters** should only be able to manage their job postings.
     - **Job seekers** should be able to view and apply to job listings.

2. **Queue System Integration**:
   - Set up a **queue system** to handle background tasks:
     - **Email Notification:** When a job seeker applies for a job, send an email to the recruiter notifying them of the new application. This should be processed in the background using the queue system.

3. **Database:**
   - Use **MySQL** or **PostgreSQL** or **MongoDB** to store:
     - **Users** (Recruiters and Job Seekers with distinct roles).
     - **Jobs** (job title, description, company, location, etc.).
     - **Applications** (job ID, applicant details, and resume).
   - Ensure proper relationships between users, jobs, and applications.

4. **Security:**
   - Implement **password hashing** using bcrypt or another secure method.
   - Use **JWT** tokens for protected routes and role-based access control.
   - Ensure input validation and security measures against common attacks (SQL Injection, XSS).

5. **Error Handling:**
   - Implement appropriate error handling for all endpoints, returning relevant HTTP status codes (e.g., 401 Unauthorized, 404 Not Found).
   - Ensure clear and useful error messages for invalid requests.

6. **Testing:**
   - Write **unit tests** for at least three critical API endpoints (e.g., user registration, job posting creation, and job application submission).

- Demonstrate API functionality using a tool like **Postman**. (By sharing the collection file in the repo)

Bonus Tasks:

1. **Frontend Development** (Optional):
   - Create a simple **frontend** using **HTML/CSS/JavaScript** or **React** to interact with the API.
   - Features:
     - Job seekers can browse and filter job listings.
     - Recruiters can log in, post jobs, and view applicants.
     - Job seekers can submit job applications through the front-end.

2. **Docker Integration** (Optional):
   - Containerize the application using **Docker**.
   - Provide a ***Dockerfile*** and use **Docker Compose** to set up the application, database, and queue system.
   - Include a ***docker-compose.yml*** file that can set up and run the entire system with one command.

3. **Deployment** (Optional):
   - Deploy the application on a cloud platform (e.g., **AWS**, **Heroku**, or **DigitalOcean**).
   - Provide a live URL where the application can be tested.

API Endpoints to Implement:
- *POST /register* – Register a new user (recruiter or job seeker).
- *POST /login* – User login and JWT token generation.
- *POST /jobs* – Create a new job posting (recruiters only).
- *GET /jobs* – Retrieve all job listings.
- *GET /jobs/:id* – Retrieve details of a specific job posting.
- *POST /jobs/:id/apply* – Apply to a specific job (job seekers only).
- **Queue task**: Trigger an email notification to the recruiter when a job application is submitted.

Deliverables:
- A fully functional **backend API** that meets the requirements.
- Source code stored in a **Git repository**.
- A brief **README** explaining:
  - How to set up and run the project locally.
  - How to set up the queue system.
- **Bonus**: Include Dockerfile and docker-compose.yml for Docker setup.
- **Bonus**: Deployed version of the app (provide the link if deployed).