Backend: (Django)

Project Description:

Build a backend system using Django with Django ORM to manage the data layer. The application should be fully functional, handling data through a relational database. You can use SQLite (default) or configure another database like PostgreSQL or MySQL if you prefer.

Requirements:

1. Backend Implementation (Django):

User Authentication: Implement user authentication using Django's built-in authentication system, allowing users to register, log in, and log out using email and password.

Database Migrations (Django ORM):

- Employees Table:
  - Create a migration for an `Employee` model with the following fields: `id`, `name`, `email`, `phone`, `address`, and `department_id`.
- Departments Table:
  - Create a migration for a `Department` model with the following fields: `id`, `name`.
- Achievements Table:
  - Create a migration for an `Achievement` model with the following fields: `id`, `name`.
- Pivot Table:
  - Create a pivot table (many-to-many relationship) for associating employees with achievements (`AchievementEmployee`) with the following fields: `id`, `achievement_id`, `employee_id`, and `achievement_date`.

Model Relationships:

- Define a one-to-many relationship between `Department` and `Employee` models.
- Define a many-to-many relationship between `Employee` and `Achievement` models through the pivot table.

Views/Controllers: Implement the following features in Django views:

- List Employees: Retrieve and display a list of all employees along with their department and achievements.
- Create Employee: Implement functionality to add a new employee, assigning them to a department and adding achievements.
- Update Employee: Allow updating an employee's details, department, and associated achievements.
- Delete Employee: Implement functionality to remove an employee from the system.

2. Database:

- Use SQLite (default with Django) as the relational database.

- Alternatively, you may configure and use PostgreSQL or MySQL if you're familiar with them.

3. Django ORM:

- Use Django's built-in ORM for all database interactions. This includes creating models, performing CRUD operations, and managing database relationships.

Deliverables:

1. Project Code:

- A fully functional Django backend with proper models, views, migrations, and database relationships.

- Use Django ORM to manage all database interactions.

2. Documentation:

- Provide a **README.md** file with:

  - Clear instructions for setting up and running the project locally.

  - Information on any dependencies or configurations needed for the database.

  - Details on any additional features implemented.

3. Source Code:

- Follow the best practices. (Like commenting on code where necessary)

- Host your project on a public repository (e.g., GitHub, GitLab).

- Ensure a clean and structured commit history with meaningful commit messages.

Submission:

- Submit a GitHub repository link containing your Django project.

- Ensure that all deliverables are included, and the project is well-organized with proper documentation.

“” Note: **

- You are required to use Django ORM for database interactions.

- SQLite is the default database, but you are free to use PostgreSQL or MySQL if preferred.