**Tables and Relationships**

**DEPARTMENT**

PK ID

name

created\_at

updated\_at

**POSITION**

PK ID

name

salary

created\_at

updated\_at

FK department\_id

**EMPLOYEE**

PK ID

name

surname

email

FK department\_id

FK position\_id

status

created\_at

updated\_at

**EMPLOYEE management app**

**USER**

1. User register

2. User login

**DEPARTMENT**

1. Department save

2. Department get one

3. Department get all

4. Department edit

5. Department delete

**POSITION**

1. Position save

2. Position get one

3. Position get all

4. Position edit

5. Position delete

**EMPLOYEE**

1. Employee save

2. Employee get one

3. Employee get all

4. Employee edit

5. Employee delete

**Technologies to be used**

1. Django

2. Django Rest Framework

3. Celery

4. JSON Web Tokens - JWT

5. PostgreSql

**Technical structure**

1. Users must register (Token authentication). Each user must have at least one role (admin; user).
2. The /register and /login endpoints must be open to everyone. That is, there should be no restrictions on permissions. All other endpoints can only be accessed by logged-in users.
3. DELETE endpoints can only be accessed by users with the "ADMIN" role.
4. Multilanguage support is required (az; en is sufficient). Data such as position name and department name should return according to the selected language in the response.
5. API documentation must be provided (Swagger).
6. All fields of the model should be visible in the admin interface.
7. A middleware should be written to prevent requests from blocked IP addresses.
8. A periodic task should be written to send notifications every day to employees who have not been registered for more than 2 days.

**Some Comments**

The project will not utilize any custom templates, as the automatic views provided by the Django Rest Framework are sufficient. When the API page is accessed, it displays directories such as "employees," "departments," and others in text format. Clicking on these directories will open their respective data pages in the same API view. However, a graphical interface would provide a better user experience.

Steps

1. **Project Initialization and Dependencies**
   * Set up a Django project and install dependencies: Django, Django Rest Framework, PostgreSQL.
   * Create a virtual environment to manage dependencies.
2. **Database Configuration**
   * Connect the project to PostgreSQL by updating the database settings.
3. **Model Definition**
   * Define models for Department, Position, and Employee, with specified fields and relationships.
   * Set up foreign keys for department\_id and position\_id in the Employee model.
4. **User Model and Role Management**
   * Use Django's built-in user model, adding roles (admin, user) via groups.
5. **JWT Authentication**
   * Set up JSON Web Tokens for secure user registration and login using djangorestframework-simplejwt.
6. **Authentication Endpoints**
   * Create /register and /login endpoints, accessible to all users without restrictions.
7. **CRUD Functionality for Department, Position, and Employee**
   * Implement serializers and views for CRUD operations in Department, Position, and Employee.
8. **Access Permissions**
   * Restrict DELETE operations to users with the ADMIN role.
9. **Multilanguage Support**
   * Enable support for az and en languages.
   * Localize fields such as name in Position and Department.
10. **API Documentation with Swagger**

* Integrate drf-yasg for auto-generating Swagger API documentation.

1. **Blocked IP Middleware**

* Implement middleware to block requests from specified IP addresses.

1. **Admin Interface Configuration**

* Ensure all model fields are displayed in the Django admin panel.

1. **Periodic Task Setup**

* Configure Celery for background tasks, connecting it to Redis in Docker.

1. **Employee Notification Task**

* Create a daily task to notify employees registered over two days ago.

1. **Testing**

* Test all endpoints, CRUD operations, permissions, and middleware functionality in Postman or similar tools.