Project Idea: Task Management System API

Project Overview

Develop a Task Management System API using FastAPI that allows users to manage their tasks efficiently. The API will support user authentication, task creation, updating, deletion, and categorization. Additionally, it will provide features such as task prioritization, deadlines, and status tracking.

Key Features

1. User Authentication and Authorization

- Registration: Allow new users to create an account.

- Login: Authenticate existing users using JWT (JSON Web Tokens).

- Role-Based Access Control: Differentiate permissions between regular users and administrators.

2. Task Management

- Create Task: Users can create new tasks with details like title, description, priority, and deadline.

- Read Tasks: Retrieve a list of all tasks or filter them based on status, priority, or deadline.

- Update Task: Modify task details or change its status (e.g., from "Pending" to "Completed").

- Delete Task: Remove tasks that are no longer needed.

3. Categorization and Tagging

- Categories: Organize tasks into categories (e.g., Work, Personal, Urgent).

- Tags: Add tags to tasks for better filtering and searching.

4. Notifications

- Email Reminders: Send email notifications for upcoming deadlines.

5. Reporting and Analytics

- Task Statistics: Provide insights like the number of completed tasks, pending tasks, and tasks per category.

Technologies and Tools

- Backend Framework: FastAPI

- Programming Language: Python

- Database: PostgreSQL (using SQLAlchemy ORM)

- Authentication: JWT (using fastapi-jwt-auth or similar libraries)

- Email Service: SMTP (using libraries like smtplib) or third-party services like SendGrid

- Testing: PyTest for unit and integration tests

- Documentation: Automatic API docs generated by FastAPI (Swagger UI)

- Version Control: Git (hosted on GitHub)

- Deployment: Docker for containerization, deployed on platforms like Heroku, AWS, or DigitalOcean

Implementation Steps

1. Project Setup

- Initialize a new FastAPI project.

- Set up a virtual environment and install necessary dependencies (fastapi, uvicorn, SQLAlchemy, Pydantic, etc.).

- Configure Git for version control and create a GitHub repository.

2. Database Design

- Design the database schema with tables for Users, Tasks, Categories, and Tags.

- Implement SQLAlchemy models corresponding to each table.

- Set up Alembic for database migrations.

3. Authentication System

- Implement user registration and login endpoints.

- Secure endpoints using JWT authentication.

- Set up role-based access control for different user types.

4. CRUD Operations for Tasks

- Create endpoints to handle creating, reading, updating, and deleting tasks.

- Implement filtering and searching capabilities based on task attributes.

5. Categorization and Tagging

- Develop endpoints to manage categories and tags.

- Allow users to assign categories and tags to tasks.

6. Email Notifications

- Integrate an email service to send reminders for upcoming deadlines.

- Schedule background tasks using FastAPI's BackgroundTasks or Celery for sending emails.

7. Reporting and Analytics

- Create endpoints to fetch task statistics.

- Implement aggregation queries to provide meaningful insights.

8. Testing

- Write unit tests for individual components using PyTest.

- Perform integration testing to ensure different parts of the application work seamlessly together.

9. Documentation

- Utilize FastAPI's automatic documentation generation to create interactive API docs.

- Optionally, add additional documentation or a README file explaining project setup and usage.

10. Deployment

- Containerize the application using Docker.

- Deploy the Docker container to a cloud platform like Heroku, AWS, or DigitalOcean.

- Set up environment variables for configuration and secure sensitive information.

Sample Project Structure

task-management-api/

├── app/

│ ├── main.py

│ ├── models/

│ │ ├── user.py

│ │ ├── task.py

│ │ ├── category.py

│ │ └── tag.py

│ ├── schemas/

│ │ ├── user.py

│ │ ├── task.py

│ │ ├── category.py

│ │ └── tag.py

│ ├── crud/

│ │ ├── user.py

│ │ ├── task.py

│ │ ├── category.py

│ │ └── tag.py

│ ├── routers/

│ │ ├── auth.py

│ │ ├── tasks.py

│ │ ├── categories.py

│ │ └── tags.py

│ ├── core/

│ │ ├── config.py

│ │ └── security.py

│ └── utils/

│ └── email.py

├── tests/

│ ├── test\_auth.py

│ ├── test\_tasks.py

│ ├── test\_categories.py

│ └── test\_tags.py

├── alembic/

│ └── ...

├── requirements.txt

├── Dockerfile

├── docker-compose.yml

└── README.md

Benefits of This Project

- Comprehensive Skill Demonstration: Showcases your ability to work with FastAPI, databases, authentication mechanisms, and deployment processes.

- Scalability and Best Practices: Emphasizes structuring projects for scalability, using ORM for database interactions, and securing APIs.

- Real-World Application: Reflects a real-world scenario where task management is essential, making your project relatable and practical.

- Extensibility: Offers numerous opportunities for adding advanced features like real-time updates with WebSockets, integration with frontend frameworks, or implementing microservices architecture.

Additional Enhancements

- Frontend Integration: Develop a simple frontend using React.js or another framework to interact with your FastAPI backend.

- Advanced Authentication: Implement OAuth2 with social logins (e.g., Google, GitHub).

- Caching: Use Redis or similar tools to cache frequent database queries and improve performance.

- Logging and Monitoring: Integrate logging (using Python's logging module) and monitoring tools to track application performance and errors.

- CI/CD Pipeline: Set up continuous integration and deployment pipelines using tools like GitHub Actions or Jenkins to automate testing and deployment.

Summary

Creating a Task Management System API with FastAPI not only aligns well with the requirements of a Junior Backend Engineer role but also provides a platform to demonstrate a wide range of backend development skills. By implementing user authentication, CRUD operations, database interactions, and deploying the application, you'll showcase your ability to handle end-to-end backend projects effectively. This project can be a valuable addition to your resume and GitHub portfolio, making you a strong candidate for backend development positions.