Elevator System API

Table of Contents:-

Introduction

The Elevator System API is a Django REST framework-based application that provides endpoints to manage elevators and user requests for an elevator system. It enables users to interact with elevators, request an elevator, mark elevators as not working, and more. The API is designed to be user-friendly and easy to integrate with an existing elevator control system.

Design Decisions

- **Elevator and User Request Models:** The Elevator model represents the properties and actions of an elevator, such as availability, operational status, current floor, and direction. The User Request model is used to capture user requests for an elevator, including the requested floor and direction.

- **Elevator System Class:** The ElevatorSystem class is used to manage and assign elevators to user requests. It uses a distance-based algorithm to find the closest available elevator to the requested floor and in the same direction.

- **API Endpoints:** The API endpoints are designed to be RESTful and follow best practices for resource management. The API uses viewsets to handle CRUD operations for elevators and user requests.

API Contracts

The API provides the following endpoints:

Elevator Endpoints

- `**GET /api/elevators/**`: Get a list of all available elevators.
- `**GET /api/elevators/{elevator_id}/**`: Get details of a specific elevator.
- `**PUT /api/elevators/{elevator_id}/**`: Update the details of a specific elevator.
- `**DELETE /api/elevators/{elevator_id}/**`: Delete a specific elevator.
- `**GET /api/elevators/{elevator_id}/next_destination_floor/**`: Get the next destination floor of a specific elevator.
- `**GET /api/elevators/{elevator_id}/moving_direction/**`: Get the moving direction of a specific elevator.

User Request Endpoints

- `**POST /api/user-requests/**`: Create a new user request for an elevator.
- `**POST /api/user-requests/{elevator_id}/mark_maintenance/**`: Mark an elevator as not working or in maintenance.
- `**POST /api/user-requests/{elevator_id}/open_door/**`: Open the door of a specific elevator.
- `**POST /api/user-requests/{elevator_id}/close_door/**`: Close the door of a specific elevator.

Architecture

The Elevator System API is built using Django and Django REST framework. It follows the Model-View-Controller (MVC) architectural pattern, where models represent the data structure, views handle HTTP requests and responses, and controllers manage the business logic.

The architecture is designed to be scalable and maintainable. The Elevator and User Request models encapsulate the data and actions related to elevators and user requests. The views and viewsets handle the API endpoints and interact with the models to perform CRUD operations.

Repository File Structure

The repository file structure is organized as follows:

```
Elevator-system-Jumping-minds/
  ├── elevator_app/              Django app directory
  │    ├── migrations/           Database migrations
  │    ├── models.py             Models for Elevator and User Request
  │    ├── serializers.py        Serializers for models
  │    ├── views.py              API viewsets and actions
  │    └── urls.py               URL configurations for the app
  ├── Elevator/                  Django project directory
  ├── db.sqlite3                 SQLite database file (auto created)
  ├── README.md                  Project documentation (you are here)
  ├── manage.py                  Django management script
  └── requirements.txt           List of project dependencies
```

Database Modeling

The database schema is designed to store information about elevators and user requests. The Elevator model has fields for availability, operational status, current floor, and direction. The User Request model captures the requested floor and direction for an elevator.

Plugins or Libraries Used

The Elevator System API uses the following key libraries and frameworks:

- Django: The web framework for building the API.
- Django REST framework: An extension of Django that adds powerful features to create RESTful APIs.

Note: The requirements.txt file contains the necessary dependencies to run the program.

Steps to Setup, Deploy, and Test

To set up, deploy, and test the Elevator System API, follow these steps:

1. Clone the repository to your local machine.

2. Install Python and create a virtual environment.

3. Activate the virtual environment and install project dependencies:

    “pip install -r requirements.txt”

4. Perform database migrations:

“python manage.py migrate”

5. Create a superuser to access the Django admin panel (optional):

python manage.py createsuperuser

6. Start the development server:

“python manage.py runserver”

7. Access the API at **`http://127.0.0.1:8000/api/`.**

8. Use tools like cURL or Postman to test the API endpoints as documented. (I had used postman to test the api (screenshots attached)).

Conclusion

The Elevator System API is a robust and scalable solution for managing elevators and user requests in an elevator control system. By following the provided documentation and guidelines, anyone can set up, deploy, and test the API effectively, providing a seamless experience for users interacting with elevators in the system.