Elevator Problem Statement

In this challenge, you are asked to implement the business logic for a simplified elevator model in Python. We'll ignore a lot of what goes into a real world elevator, like physics, maintenance overrides, and optimizations for traffic patterns. All you are asked to do is to decide whether the elevator should go up, go down, or stop.

An elevator system, which can be initialised with N elevators and maintains the elevator states as well.

Table of Contents:

A. Each elevator has below capabilities:

- Move Up and Down
- Open and Close Door
- Start and Stop Running
- Display Current Status
- Decide whether to move up or down, based on a list of requests from users.

B. Elevator System takes care of:

- Decides which lift to associate which floor.
- Marks which elevator is available or busy.
- Can mark which elevator is operational and which is not.

C. Assumptions:

- Number of elevators in the system will be defined by the API to intialise the elevator system
- Elevator System has got only one button per floor.
- So if there are a total of 5 floors, there will be 5 buttons per floor.
- Note that, this doesn't not mimic real world, when you would have a total of 10 buttons for 5 floors (one for up and one for down)
- Once the elevator reaches its called point, then based on what floor is requested, it moves either up or down.

- Assume the API calls which make the elevator go up/down or stop will reflect immediately. When the API to go up is called, you can assume that the elevator has already reached the above floor.
- The system has to assign the most optimal elevator to the user according to their request.

D. Asks:

- Create Django Project to maintain elevator system and its operations
- You can use Postgres for DB if needed
- You can use redis for caching if needed
- Come up with a API response enabling easy integrations & makes sense by just looking at it
- Use good naming conventions
- Add comments and documentations where ever necessary

E. API's required:

1. Initialise the elevator system to create 'n' elevators in the system
2. Fetch all requests for a given elevator
3. Fetch the next destination floor for a given elevator
4. Fetch if the elevator is moving up or down currently
5. Saves user request to the list of requests for a elevator
6. Mark a elevator as not working or in maintenance
7. Open/close the door.

Note: Add any other API which may be helpful according to you, you can assume a few scenarios in a way it makes sense.

F. Submission Process and Judging Criteria:

For the submission, please do the following:

1. Create a Public Github Repository before you start working on the problem statement.
2. Keep committing code through the development process to the github repository to **avoid pushing the entire codebase in one single commit.**
3. Once the submission is complete, please add a well structured and descriptive README.md detailing out your thought process; your design decisions; API contracts and Steps to setup, deploy and test the submission. *(including the choice of Architecture, Repository file structure, Database modelling, Plugins or Libraries used)*
4. Once done, Email us the Github Repository link along with a short video depicting the use of the project and the APIs as expected which will help us with the functioning of the submission.

We are going to judge you on

1. Code quality (follow all good practices and code should be production ready)
2. Thoughtfulness (Cover all edge cases and think of anything that we might have missed in the problem statement)
3. Comments and Instructions (Readme file)