Task 1: Beyblade Battle Arena API

You've been hired by a game development studio to create the initial API for their upcoming Beyblade mobile game (a game centered around spinning-top toys that battle in arenas). They need a system to manage Beyblades, simulate battles, and handle basic upgrades. This API will be used by the front-end developers to create an exciting and interactive Beyblade experience for players. They have well defined routes they want you to implement, which are as follows.

Routes:

1. GET `/beyblades`
    - Return a list of all available Beyblades.
2. GET `/beyblades/:name`
    - Return details of a specific Beyblade according to its name.
3. POST `/battle`
    - Accept two Beyblade names in the request body.
    - Simulate a battle based on their stats.
    - Return the winner and battle details.
4. PUT `/beyblades/:name/upgrade`
    - Upgrade a Beyblade's stats (increase attack or defense).
    - Return the updated Beyblade information.

API Setup:

1. Initialize a new Express.js application.
2. Create a data structure to store Beyblade information including name, attack power, defense power, and special moves (some names you can use are *“Starblast Attack”* and *“Blazing Gig Tempest”*).

Additional Requirements:

- Implement basic error handling for invalid requests using appropriate error codes (e.g; use 404 for “not found”).
- If an entry is missing, handle this scenario with the appropriate message to the user.

Task 2: Power Rangers Team Assemble!

A new Power Rangers series is in development, and the producers want a digital platform to help them experiment with different team combinations. Therefore they approached you to work on this project. Your API will allow them to create and manage various Ranger teams, helping them visualize how different combinations might work in the show. They have given you a set of requirements that you must follow, and defined all the different functionalities they expect from

your system. After reviewing these you have come to the conclusion that you need to implement the following routes.

Routes:

1. GET `/rangers`
    - Return a list of all available Power Rangers.
    - Include name, color, and special ability for each Ranger.
2. POST `/teams`
    - Create a new Power Rangers team.
    - Accept team name and list of Rangers in the request body.
    - Validate that selected Rangers exist and are not already in a team.
    - Return the created team with its details.
3. GET `/teams/:teamName`
    - Retrieve details of a specific team according to their name.
    - Include team name, members, and combined team power level.
4. DELETE `/teams/:teamName`
    - Disband a team, making its Rangers available again.
    - Return a confirmation message indicating that the team has been disbanded.

Additional Requirements:

- Implement basic error handling for invalid requests using appropriate error codes (e.g; use 404 for “not found”).
- If an entry is missing, handle this scenario with the appropriate message to the user.

Task 3: Galactic Space Station Management API

You've been recruited by the Intergalactic Space Agency (ISA) to develop an API for managing their newest space station, "Starbase Alpha". This cutting-edge facility hosts various alien species, advanced research projects, and interstellar trade operations. Your API will be crucial for maintaining the station's daily operations, managing resources, and ensuring the safety and satisfaction of all inhabitants. After discussing with ISA, they have decided they need a system that allows them to frequently update their information. Therefore, you have inferred that they will need the following routes:

Routes:

1. GET /station-info
    - Return general information about the space station (name, capacity, current population).
2. GET /inhabitants
    - Return a list of all current inhabitants on the station.
    - Include name, species, and purpose of stay for each inhabitant.

3. PUT /station-info
    - Update general information about the space station.
    - Allow updates to name, description, and maximum capacity.
    - If maximum capacity is reduced, ensure it's not below the current population.
4. PUT /inhabitants/:name
    - Update information for a specific inhabitant.
    - Allow updates to purpose of stay, assigned quarters, and access level.
    - If updating access level, implement a verification system to ensure the requester has appropriate permissions.
5. PUT /resources/:resourceName
    - Update the quantity and properties of a specific resource.
    - Allow updates to current quantity, maximum capacity, and criticality level.
6. PUT /research-projects/:projectName
    - Update the status, details, or resource allocation of a specific research project.
    - Allow updates to project status, lead researcher, allocated resources, and priority level.
7. PUT /security-protocols
    - Update the station's security protocols.
    - Allow updates to access levels, restricted areas, and emergency procedures.
8. PUT /maintenance-schedule
    - Update the station's maintenance schedule.
    - Allow rescheduling of tasks, assignment of new maintenance crews, and adjustment of task priorities.

API Setup:

1. Initialize a new Express.js application.
2. Create data structures to store information about:
    - The space station itself
    - Inhabitants
    - Resources
    - Research projects
    - Security protocols
    - Maintenance schedules

Additional Requirements:

- Implement basic error handling for invalid requests using appropriate error codes (e.g; use 404 for “not found”).
- If an entry is missing, handle this scenario with the appropriate message to the user.