More Torque!

Introduction

You are the founding engineer for a taxi service company called ‘More Torque’ who provides cars for companies (interchangeably referred to as organizations) and you need to build software.
More Torque (MT), deals with numerous companies.

Key entities in your system

1. **Vehicle:** Cars that are being provided to the orgs for taxi-services.
2. **Org:** Short for companies or organizations. These organizations have signed contracts with your company MT, to provide taxi services for their orgs. You can assume that once a vehicle is associated with an org, **it will never change.** Some companies might be related to each other. They can have parent-child relations between them. For example, there can be an organization who is the parent org of another org in your company. Moreover, these companies will have some rules for their taxi-service which we need to track. For example, a company might have a speeding policy that we need to follow while driving the taxi for them.

You need to add support for the following endpoints in your system.

API Endpoints

1. **GET /vehicles/decode/:vin**
    a. Each vehicle has a unique identifier called a vin (vehicle identification number).
    b. On receiving a request with a VIN in the params, this endpoint should in-turn call NHTSA’s (https://vpic.nhtsa.dot.gov/api/) **DecodeVin** endpoint which will decode the VIN. Decoding a VIN is essentially fetching details

about the car associated with that VIN such as manufacturer, model, year of release, horsepower to name a few.

c. Your response should be the details of the vehicle which you would’ve gotten from NHTSA

Important! NHTSA is a public API so they will not allow a lot of requests continuously. Hence, we want you to implement a client side mechanism in your code which will limit the number of NHTSA calls you are making to not more than 5 a minute.

Bonus points area! Think about how you will deal with repeated calls for the same VIN?

2. **POST /vehicles**
   a. You are adding a vehicle to the system. The body of the request should only have the vin and the org to which this vehicle belongs to.
   b. This endpoint needs to decode the vin and then then add the vehicle to its system along with the org to which it belongs to
   c. **RequestBody**:
```
{
  vin: “xxxxxxxx”,
  org: “yyyyyy”
}
```
   d. **Validations**:
      i. Vin should be a valid 17 digit alpha-numeric string
      ii. The given orgId should be present in our system
   e. **Response**
      i. Success: Returns a 201 status code along with the created vehicle's details.
      ii. Error: Returns a 400 status code with an error message in case of invalid input.

3. **GET /vehicles/:vin**
   a. Fetches the vehicle corresponding to the id and returns the details of that vehicle.
   b. **Validations**:
      i. Vin should be a valid 17 digit alpha-numeric string
      ii. The given vin should be present in our system
   c. **Response**
      i. Success: Returns a 200 status code

        ii. Error: Returns a 400 status code with an error message in case of invalid input.
        iii. Error: Returns 404 if vehicle doesn't exist

4. **POST /Orgs**
    a. This endpoint allows you to create a new organization. The organization will contain information such as the name, account details, website, and specific policies related to fuel reimbursement and speed limits.
    b. Request body:

```
{

  "name": "Org1",

  "account": "acc1",

  "website": "www.org1.com",

  "fuelReimbursementPolicy": "policy1",

  "speedLimitPolicy": "policy2"

}
```

    c. **Default Values:**
        i. **fuelReimbursementPolicy**: If no `fuelReimbursementPolicy` is provided, the default value will be set to `1000`.
    d. **Response**
        i. Success: Returns a 201 status code along with the created organization's details
        ii. Error: Returns a 400 status code with an error message in case of invalid input.

5. **PATCH /orgs**
    a. This endpoint allows you to update an existing organization's details. You can update properties such as the account, website, fuel reimbursement policy, and speed limit policy.

    Request Body:

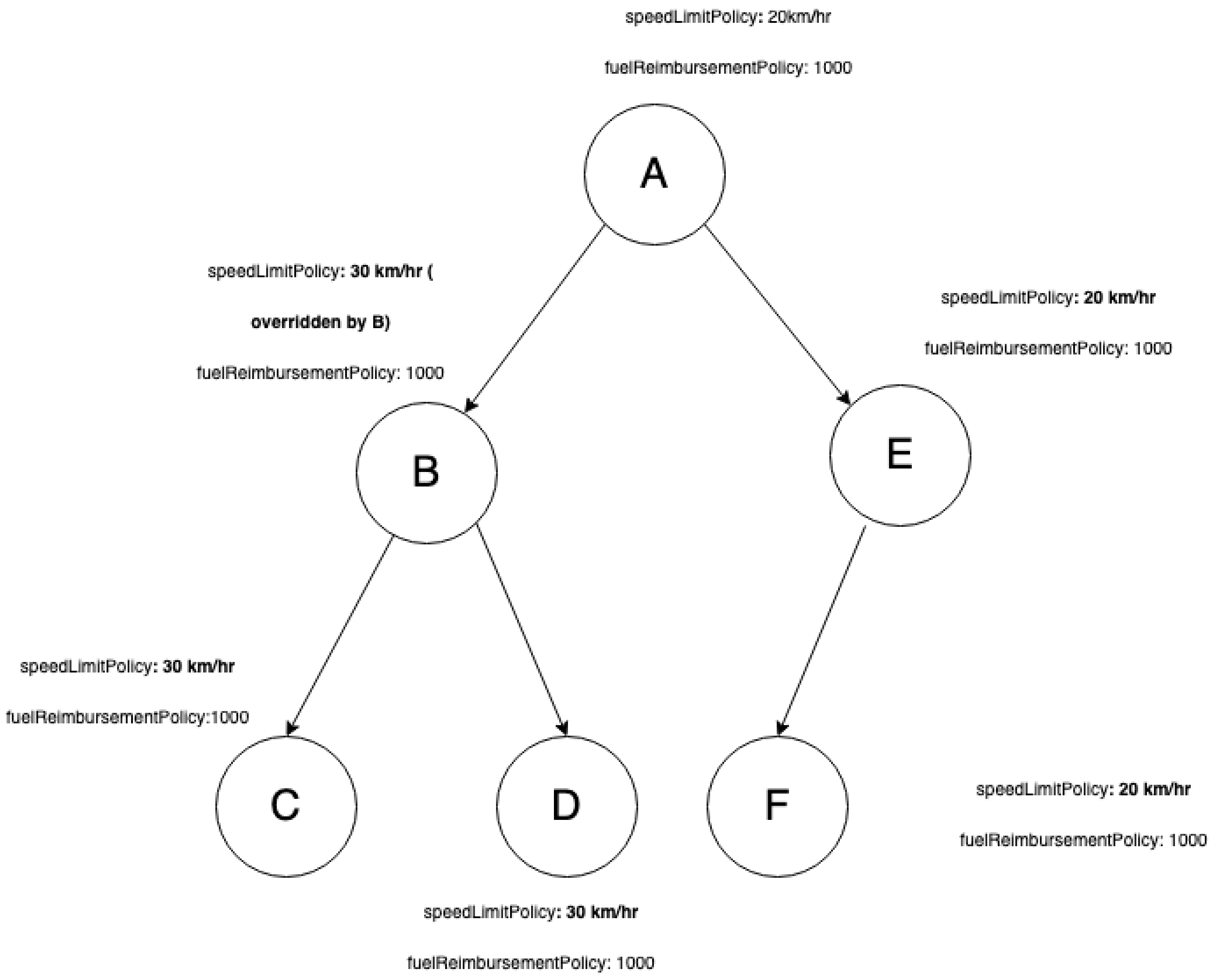
```
{

  "account": "acc1",
  "website": "www.org1.com",

  "fuelReimbursementPolicy": "policy1",

  "speedLimitPolicy": "policy2"

}
```


speedLimitPolicy: 20km/hr
fuelReimbursementPolicy: 1000
A
speedLimitPolicy: 30 km/hr (
overridden by B)
fuelReimbursementPolicy: 1000
B
speedLimitPolicy: 20 km/hr
fuelReimbursementPolicy: 1000
E
speedLimitPolicy: 30 km/hr
fuelReimbursementPolicy:1000
C
D
F
speedLimitPolicy: 20 km/hr
fuelReimbursementPolicy: 1000
speedLimitPolicy: 30 km/hr
fuelReimbursementPolicy: 1000

b. Inheritance Rules for Patching and Creating fuelReimbursementPolicy:

a. Any `fuelReimbursementPolicy` assigned to a parent organization will automatically be assigned to all its children.

b. If a child organization already has a `fuelReimbursementPolicy` and its parent is updated with a new policy, the child will also inherit the new policy.
c. If a `fuelReimbursementPolicy` already exists on a child org (**THROUGH ITS PARENT**), a child org **CANNOT** patch it. This has to be patched at the parent where this `fuelReimbursementPolicy` assignment was made.
d. given ex: The `fuelReimbursementPolicy` is set at Org A and is automatically inherited by all child entities (such as departments or sub-organizations).

speedLimitPolicy:

e. Any `speedLimitPolicy` assigned to a parent organization will automatically be assigned to all its children.
f. If a child organization already has a `speedLimitPolicy`, it will not get updated with the new policy when the parent is updated.
g. In this case, `Org B` has overridden the default `speedLimitPolicy` that is set by its parent organization. Org B has set a new `speedLimitPolicy` of 30 km/h.
h. **Propagation:** This overridden policy is then propagated to all child entities under `Org B`. Therefore, all children of `Org B` will adhere to the `speedLimitPolicy` of 30 km/h unless they override it themselves

Response:

- Success: Returns a 200 status code along with the updated organization's details.

- Error: Returns a 400 status code with an error message in case of invalid input.

Please Note: Make sure that every update leaves the whole org heirarchy is a consistent state.

6. GET /orgs

Description:

This endpoint allows you to retrieve information about all organizations, including their names, accounts, websites, and policies. This includes details of their inheritance from parent organizations.

Response:

1. Success: Returns a 200 status code along with a list of organizations and their respective details.
2. Error: Returns a 400 status code with an error message in case of invalid input.

BONUS: Due to the huge potential number of child orgs, see if a pagination can be included to ensure lightweight API responses.

Bonus areas

1. Store the entities and other similar items in a persistent storage so that your system is fault tolerant.
2. Add authentication to your APIs to ensure no one outside More Torque is using this.
3. Rate limit all API's so that they aren't being misused by a rogue script!
4. Implement caching in relevant places to save time and improve latency.