**Problem Statement:**

**Build a REST based JSON mock server to easily add, update, delete and access data from a JSON file.**

- **Every data set should have a parent identifier (entity type), which will be used in the GET APIs.**
- **Every data set should have an ID (Primary key)**
- **ID should be immutable, error needs to be thrown if ID is tried to be mutated.**
- **If you make POST, PUT, PATCH or DELETE requests, changes have to be automatically saved to store.json.**
- **The store.json file should support multiple entity types.**

```
{
  posts: [{
    id: 0
    title: "title1",
    author: "CIQ",
    views: 100,
    reviews: 31
  }, {
    id: 1
    title: "title2",
    author: "CommerceIQ",
    views: 10,
    reviews: 3
  }],
  authors: [{
    id: 0,
    first_name: "Commerce",
    last_name: "IQ",
    posts: 45
  }]
}
```

- **Sample APIs to be supported by the mock server on store.json file:**
  - **GET /posts**
  - **GET /posts/0**
  - **POST /posts**
  - **PUT /authors/1**
  - **PATCH /posts/1**
  - **DELETE /posts/1**
- **Enable filtering at entity level :**
  - **GET /posts?title=title1&author=CIQ**
- **Enable sorting at entity level :**
  - **GET /posts?_sort=views&_order=asc**
- **Enable basic search at entity level:**
  - **GET /posts?q=IQ**
- **Support for nested structures will yield a bonus point.**
- **Treat store.json as an empty slate where you can add and retrieve any data.**

**Rules of the game!**

- **There is no restriction on the programming language.**
- **Candidates are expected to deploy the application on AWS / equivalent cloud provider.**
- **Candidates are expected to submit github repo details along with the public endpoint [Need not be behind a R53 domain] for us to access the server.**
- **Do submit sample API requests used for dev testing along with the list of functionalities you have implemented.**
- **Candidates will be given 3 days to submit the solution.**
- **Feel free to reach out Chaithra for any clarifications.**
- **Candidates will be judged based on completeness of functional and nonfunctional requirements, code style and quality.**
- **Additional functionalities identified and implemented will yield bonus points.**
- **Get coding! All the best.**

## Solution:

**Application Language:** Developed using **Java-Spring Boot**

**JSON Data:**

- Data stored in store.json that is initialized on the server in the following format:


```
{
 "authors": [
  {
   "firstName": "Elon",
   "lastName": "Smuck",
   "post": 2,
   "id": 1
  },
  {
   "firstName": "Nas",
   "lastName": "Medium",
   "post": 1,
   "id": 2
  }
 ],
 "posts": [
  {
   "reviews": 30,
   "author": "Elon",
   "id": 1,
   "title": "Post1",
   "views": 600
  },
  {
   "reviews": 50,
   "author": "Elon",
   "id": 2,
   "title": "Post2",
   "views": 500
  },
```

```
 {
    "reviews": 60,
    "author": "Nas",
    "id": 3,
    "title": "Post3",
    "views": 100
  }
 ]
}
```


## Features Implemented:

- ❏ Every data set should have a parent identifier (entity type), which will be used in the GET APIs.
- ❏ Every data set should have an ID (Primary key)
- ❏ ID should be immutable, error needs to be thrown if ID is tried to be mutated.
- ❏ If you make POST, PUT, PATCH or DELETE requests, changes have to be automatically saved to store.json.
- ❏ The store.json file should support multiple entity types: Code changes required to GET/PUT new entities
- ❏ Sample APIs to be supported by the mock server on store.json file:
  GET /posts
  GET /posts/0
  POST /posts
  PUT /authors/1
  PATCH /posts/1
  DELETE /posts/1
- ❏ Enable filtering at entity level :
  GET /posts?title=title1&author=CIQ (single entity filter and sort functionality code changes are not implemented yet)
- ❏ Treat store.json as an empty slate where you can add and retrieve any data.

**Application Deployment:**

- ❏ Application hosted on AWS EC2 instance for accessing API Requests
  **API Access on:** http://3.135.234.185:8080/

**GitHub Repository:**

- ❏ **https://github.com/sanjana3011/json-api/tree/master**

## Dev Testing and Endpoints Supported:

- **Postman Collection API Link:**
  https://github.com/sanjana3011/json-api/blob/master/postman_collection.json

- **Postman Documentation Link:**
  https://documenter.getpostman.com/view/11826258/TzCQaRU1