Question - 1
SpringBoot(Debugging) : Home API

In this project, virtual data related to homes is provided for many areas. (Note that all the data is artificial.) Your task is to implement several REST endpoints to handle this data.

Here is an example of a home data JSON object:

```
{
  "id": 1,
  "area": 36.1189,
  "price": 86.6892,
  "city": "Noida"
}
```

The application should adhere to the following API format and response codes:

`POST /api/home`:

- Accepts a home object without an ID and returns status code 201 on creation.
- If the home object is provided with an ID, status code 400 is returned.
- It is secured by HTTP basic authentification, where the username and password are "admin" and "admin".

`GET /api/home/{id}`:

- Returns the home entry with the given ID and status code 200.
- Returns status code 404 if the requested home does not exist.
- Returns status code 400 if the requested home ID is invalid.
- This should be publicly accessible and should not require HTTP basic authentication.

`GET /api/home`:

- Returns all the home entries with status code 200.
- It is secured by HTTP basic auth, where the username and password are "admin" and "admin".

There are 4 tests already written, but they don't pass due to bugs in the implementation of the endpoints. Debug the code so that all tests pass.

Question - 2
SpringBoot (Debugging): Location API

In this project, data is provided for many countries with API endpoints for fetching specific information. (Note that all the data is artificial.) Your task is to implement several REST endpoints to handle this data.

Here is an example of a location data JSON object:

```
{
    "id": 1,
    "lat": 36.1189,
    "lon": -86.6892,
    "temperature": [17.3, 16.8, 16.4, 16.0, 15.6, 15.3, 15.0, 14.9, 15.8, 18.0, 20.2, 22.3, 23.8, 24.9, 25.5,
```

```
25.7, 24.9, 23.0, 21.7, 20.8, 29.9, 29.2, 28.6, 28.1]
}
```

The application should adhere to the following API format and response codes:

`POST /api/location`:

- Accepts a location object without an ID and returns status code 201 on creation.
- If the location object is provided with an ID, status code 400 is returned.

`GET/api/location/{id}`:

- Returns the location entry with the given ID and status code 200.
- Returns status code 404 if the requested location doesn't exist.
- Returns status code 400 if the requested location ID is invalid.

`GET /api/location`:

- Returns all the location entries with status code 200.

There are 5 tests already written, but some don't work due to bugs in the implementation of the endpoints. Find the bugs and fix them so that all the tests pass.

Question - 3

SpringBoot(Debugging): Vehicle API

In this project, virtual data related to vehicles is provided for many countries. (Note that all the data is artificial.) Your task is to implement several REST endpoints to handle this data.

Here is an example of a vehicle data JSON object:

```
{
    "id": 1,
    "brand": "suzuki",
    "type": "hatchback",
    "cc": 1200,
    "color": "red"
}
```

The application should adhere to the following API format and response codes:

`POST /api/vehicle`:

- Accepts a vehicle object without an ID and returns status code 201 on creation.
- If the vehicle object is provided with an ID, status code 400 is returned.

`GET /api/vehicle/{id}`:

- Returns the vehicle entry with the given ID and status code 200.
- Returns status code 404 if the requested vehicle does not exist.
- Returns status code 400 if the requested vehicle ID is invalid.

`GET /api/vehicle`:

- Returns all the vehicle entries with status code 200.

There are 4 tests already written, but they don't pass due to bugs in the implementation of the endpoints. Debug the code so that all tests pass.

Question - 4

SpringBoot (Debugging): Sales API

In this project, data is given regarding various sales. (Note that all data is artificial.) Your task is to implement several REST endpoints to handle this data.

Here is an example of a sale data JSON object:

```
{
  "id": 1,
  "productName": "product1",
  "customerEmail": "cust@gmail.com",
  "sellingPrice": 200,
  "buyingPrice": 100
}
```

The application should adhere to the following API format and response codes:

`POST /sale`:

- Accepts a sale object and returns status code 201 if the object is valid.
- All the properties are validated using the following rules:
  - *productName* cannot be empty. If empty, return status code 400 and the message "Product name is mandatory".
  - *customerEmail* should be a valid email. If invalid, return status code 400 and the message "Invalid customer email".
  - *sellingPrice* should be a positive integer. If invalid, return status code 400 and the message "Value should not be negative".
  - *buyingPrice* should be a positive integer. If invalid, return status code 400 and the message "Value should not be negative".

`GET /sale/{id}`:

- Returns the sale entry with the given ID and status code 200.

`GET /sale`:

- Returns all the sale entries with status code 200.

There are 4 tests already written, but some don't work due to bugs in the implementation of the endpoints. Find the bugs and fix them so that all the tests pass.

Question - 5

SpringBoot(Debugging): Patient Health Records API

In this project, virtual data related to patients is provided for many countries. (Note that all the data is artificial.) Your task is to implement several REST endpoints to handle this data.

Here is an example of a patient data JSON object:

```
{
  "id": 1,
  "country": "MyCountry",
  "active": 300000,
  "death": 100000,
  "recovered": 7000
}
```

The application should adhere to the following API format and response codes:

`POST /patient:`

- Accepts a patient record object (without an ID) and returns status code 201.
- If the patient record object is provided with an ID, return status code 400.

`GET /patient/{id}:`

- Returns the patient record with the given ID and status code 200.
- If no patient record is found for the requested ID, return status code 404.
- If the ID is invalid, return status code 400.

`GET /patient:`

- Returns all patient records with status code 200.

There are 6 tests already written, but some are not working due to a bug in the implementation of the endpoints. Find the bugs and fix them so that all the tests pass.