

Wealth Rating Exercise

Purpose

The purpose of this exercise is to create a **Java microservice** that exposes **REST API** according to **industry standards**. This MS is called '**wealth-rating**' and its functionality is to manage the *wealth rating* in the state.

The following MS with its API is given (you are not supposed to implement it):

central-bank MS

- GET `central-bank/regional-info/evaluate?city={city}`
- GET `central-bank/wealth-threshold`

The Flow

- The **client** of the *wealth-rating* MS sends person's data to the **wealth-rating** service:

```
{
  "id": 123456789,
  "personalInfo": {
    "firstName": "Bill",
    "lastName": "Gates",
    "city": "Washington"
  },
  "financialInfo": {
    "cash": 16000000000,
    "numberOfAssets": 50
  }
}
```

- The **wealth-rating** analyses the person's financial status (evaluates his/her **Fortune**) using the **central-bank** API:
 - the service will call the *central-bank* API:
`GET central-bank/regional-info/evaluate?city={city}`
 - the *central-bank* will return the asset evaluation per city (value of a **single** asset)
 - Then it will call another API:
`GET central-bank/wealth-threshold`
to get the threshold value to be considered as a rich.
 - The person's fortune is calculated as follows:
`Fortune = cash + numberOfAssets * evaluateResponse`
- If the person's fortune is greater than the threshold, meaning the person is considered as a **Rich**, it will be persisted to the DB by the **wealth-rating** service with the following fields:
 - ID
 - firstName
 - lastName
 - fortune

Requirements

1. Implement the **wealth-rating** microservice in Java (use Java 8) with Spring Boot, with the following end points:
 - a. Endpoint to handle the wealth-rating's client request with the person's information (as described above).
This endpoint will return a proper response according to the person's financial status.
 - b. Get all Rich persisted in the DB
 - c. Get Rich by ID
2. Add unit tests (JUnit, Mockito).
3. Submit your solution within 24 hours of receiving it.

Tips

1. **Spring Boot** makes it easy to create stand-alone, production-grade Spring based Applications that you can "just run". Many commonly used capabilities are implemented by Spring and can be used with minimal configuration. Such as
 - a. Creating a microservice from scratch (Spring initializer will be very helpful)
 - b. Exposing REST API easily
 - c. Accessing data in DB
2. An **in-memory database** can be used for this exercise (consider using h2 in memory DB)