Project Assignment
Paynet REST API workshop Jul - Aug 2021

1 Project outline

You are required to create and deploy a local REST service based on an existing public REST API service. Your REST service will implement and expose a local REST API that is based on the functionality of the public REST API. The data retrieved from the public REST APIs are persisted to a backend RDBMS, which are then retrievable from the local REST service through the local REST API by any suitable REST clients.

2 Functional specification

You will basically replicate all (or a subset) of the data set available from a public REST service of your choice and persist it to a local database. This data set is then made available for consumption by clients through a REST API implemented by your REST service. This REST API should incorporate at **the minimum** the following HTTP methods for the following sample endpoints. For e.g. assuming you have a basic collection of resources that are termed `shops`:

Method	Sample endpoints	Description
GET	`/api/shops`	Retrieve the entire collection of shops
GET	`/api/shops/{id}`	Retrieve the shop with the specified id
POST	`/api/shops`	Add a new shop to the list of shops
PUT	`/api/shops/{id}`	Make a modification to a shop with the specified id
DELETE	`/api/shops/{id}`	Delete a shop with the specified id
GET	`/api/shops?key1=val1`	Filter the shops returned based on a criteria specified by one or more query parameters
GET	`/api/shops?limit=X&page=Y`	Pagination functionality to retrieve page no Y containing a total of X shops

You can choose to expose more endpoints if you wish. For example, you might have another internal collection of resources (lets say `products`) that you may wish to link with the main set of resources (`shops`) and expose through appropriate end points, such as:
`GET /api/shops/3/products`

You can choose to provide more than one set of query parameters to filter on and also allow them to be used in combination for e.g:
`GET /api/shops?key1=val1&key2=val2&……`

Below is a simple scenario of how this can be achieved. The URL endpoints shown in the examples require an active Rapid API account.

2.1 Example public REST API

There are numerous public REST APIs that give information related to countries, for e.g. https://rapidapi.com/apilayernet/api/rest-countries-v1

If we make a GET request to this URL endpoint:
https://restcountries-v1.p.rapidapi.com/all

we would obtain a list of detailed information for all the countries in the world in JSON format:


```
[
    {
        "name": "Brunei",
        "topLevelDomain": [
            ".bn"
        ],
        "alpha2Code": "BN",
        "alpha3Code": "BRN",
        "callingCodes": [
            "673"
        ],
        "capital": "Bandar Seri Begawan",
        "altSpellings": [
            "BN",
            "Nation of Brunei",
            " the Abode of Peace"
        ],
….
….
….
        "languages": [
            "vi"
        ],
        "translations": {
            "de": "Vietnam",
            "es": "Vietnam",
            "fr": "Viêt Nam",
            "ja": "ベトナム",
            "it": "Vietnam"
        },
        "relevance": "1.5"
    }
]
```


We can create a schema for a database table to store all or a subset of this returned information.

For e.g. you could have a table with columns to store values for all the top level JSON keys (`name`, `topLevelDomain`, `alpha2Code`, `alpha3Code`, etc). Alternatively, you may choose to store a subset of this returned information (for e.g. simplifying the data for the nested JSON key/value pairs) or just storing data for the first 3 - 5 top level JSON keys. The choice is entirely up to you. The end goal is that the database table you design will hold the data that you will eventually retrieve from this public REST API for subsequent access by a client through you're the local REST API that you will design for your service.

Next, design a REST API interface which incorporates the minimum HTTP methods for the sample endpoints illustrated earlier. The end points for your API should differ from the endpoints of the original API in order to be able to accommodate all these methods. To illustrate:

Original API endpoint	**Local API endpoint**	**Purpose**
`https://restcountries-v1.p.rapidapi.com/all`	`localhost:8080/api/countries`	To retrieve all countries
`https://restcountries-v1.p.rapidapi.com/name/malaysia`	`localhost:8080/api/countries/malaysia`	To retrieve a specific country by its name
`https://restcountries-v1.p.rapidapi.com/currency/MYR`	`localhost:8080/api/countries?currency=MYR`	To retrieve a specific country by its currency
`https://restcountries-v1.p.rapidapi.com/subregion/southern%20asia`	`localhost:8080/api/countries?subregion=southern%20asia`	To retrieve a list of countries in a specific subregion

You can also invent other API endpoints of your own that were not present in the original API for customized functionality of your own. For e.g.

GET `localhost:8080/api/countries/Malaysia?details=area`

might return the contents of the keys: `region`, `subregion` and `population` for the country Malaysia while

GET `localhost:8080/api/countries/Malaysia?details=phone`

might return the contents of the keys: `alpha2Code`, `alpha3Code` and `callingCodes` for the same country.

You will also need to incorporate suitable endpoints for the PUT, POST and DELETE methods (which are not provided in the original public REST API).

Then, either:

- manually using tools like Postman or Curl, or
- writing your own client REST code

retrieve all (or a subset) of the data from the public REST API and populate your local database table with this content based on whatever schema you have chosen for it.

Implement the functionality of your local REST API endpoints in a REST service so that it can retrieve data correctly from the local database table in correspondence with the purpose of each particular endpoint. Test the functionality with tools like Postman or Curl.

Finally, write client code that will consume all these API endpoints and validate their functionality appropriately. Any data to be sent from the client to the REST service (for e.g. in a POST request) can be hardcoded directly into the client app or read from a properties file.

2.2 Functionality specification outline

You will all use Java 8 for your implementation to simplify the process of assessment.

The functionality specification outline based on the example described previously:

a) Identify any publicly available REST API service that returns data in JSON format. Experiment with it to understand that type and format of data that you can retrieve from it using the various API end points available.

b) Create a schema for a database table (either in MySQL or H2) to store all (or a subset) of this retrieved data. The structure of this schema is up to you: you can choose to replicate exactly the format of the data retrieved from the public REST API or you can choose to only hold several key columns of data from it.

c) Retrieve all (or a subset) of the data from the public REST API and populate your local database table with this content based on whatever schema you have chosen for it

d) Design and implement a REST service using Spring REST which supports the minimum of HTTP REST methods specified earlier through suitable API endpoints. Your API endpoints should differ from the original API endpoints for the public REST API in order to support this minimum list of HTTP REST methods appropriately

e) Implement functionality using Spring Data JPA to allow your REST service to access your local database table in order to store and retrieve data via the API endpoints it exposes. Create some sample data (via *.sql tables) to initialize your table when your REST service boots up.

f) Implement a REST client using Spring RestTemplate to consume all these API endpoints and validate their functionality appropriately

g) Ensure that your source code is commented where appropriate (e.g. complex code segments or functionality).

3 Guidelines and tips

You can use any public REST API of your choice for this assignment. In addition to the APIs accessible via RapidAPI, a comprehensive list of other public REST APIs that are available for access can be obtained at:

https://any-api.com/
https://public-apis.xyz/

Keep in mind that many of these API services require you to register an account on their website in order to obtain an API key that you need to include with your HTTP request to their services, similar to RapidAPI.

Design your database table schema alongside your API endpoints at the same time as the latter relies on the former. For e.g. if you have decided you wish to support the following API endpoints:

```
localhost:8080/api/countries?name=malaysia
localhost:8080/api/countries?currency=MYR
localhost:8080/api/countries?subregion=southern%20asia
```

then at the bare minimum, your table must have at least 3 columns (`name`, `currency` and `subregion`).

Instead of using explicit CREATE TABLE statements in mySQL or H2 to create your initial table, use the `save` method of the CrudRepository instead. This will create the table automatically when the first record is saved based on the structure of your @Entity domain class. Then, when you have finished populating the table with data, export / dump it to a *.sql file that you can use to initialize your REST service the next it starts up. This ensures that this initialization *.sql file will have the correct CREATE TABLE and INSERT INTO statements for Hibernate to use since it was generated by Hibernate itself (as opposed to you manually creating it yourself - which may conflict with the way Hibernate would do it).

For e.g. in MySQL:

```
mysqldump -u root -p db_name tb_name > mytable.sql
```

For e.g. in H2:

```
SCRIPT TO 'fileName'
```

See also:
https://stackoverflow.com/questions/3256694/how-in-h2db-get-sql-dump-like-in-mysql

Implement your REST client as a separate application (Spring Boot project) from your REST service. This makes it easier to run them both together for testing. Run your REST client from the command line (set `spring.main.web-application-type=none` in your `application.properties`) to avoid conflicting with the embedded Tomcat server of your REST service.

Feel free to discuss tackling this project at a high level with your colleagues, but DO NOT copy and paste code from their projects directly (this would constitute plagiarism). As there are probably a few hundred good quality public REST APIs available for use on the Internet, it is very unlikely that any two project submissions are going to be using exactly the same public REST APIs in exactly the same manner, or implement a public REST API with exactly the same URL endpoints for the same HTTP requests.

4 Further improvements

If you have adequate time left after completing your assignment, and wish to challenge yourself further, consider the following features:

4.1 Improvement on REST Service

Instead of supplying a *.sql file in `src/main/resources` to initialize the database table, have your REST service directly perform some GET requests to the public REST API when your REST service starts up and use the content returned to initialize the database table.

Add a PATCH method implementation to your REST API based on the use of JSON Patch and add additional code to your client to test this.
Some sample tutorial guides on JSON PATCH:
http://jsonpatch.com/
https://www.baeldung.com/spring-rest-json-patch
https://www.javacodegeeks.com/2021/06/spring-endpoint-to-handle-json-patch-and-json-merge-patch.html

Use Swagger Hub tools to document your REST API based on Open API 3.0 and generate documentation from it.
https://swagger.io/docs/specification/about/

4.2 Improvement on REST client

Implement a simple command line based app (using either a menu interface or command line option driven) which allows the user to interact with the client and specify the HTTP method to use, the API endpoint to invoke and the JSON content to send (if any). A widely used library to consider is: Picocli (https://picocli.info/)

Implement a Spring MVC based web app which allows the user to interact with the service by specifying the HTTP method to use, the API endpoint to invoke and the JSON content to send (if any). The web app will run on a separate server from the REST service, so ensure that there are no port conflicts by appropriate settings.

5 Submission requirements

You can submit this project either by:

- Uploading a zip of your Spring Boot project folders (in the event you are using different projects for your client and server) to a file-sharing cloud service like Google Drive or DropBox and email me me the link to download

- Putting your code up on a GitHub code repo and emailing me the link to it

You should provide basic documentation on your project with information on:

- The various endpoints exposed by your REST API and the HTTP methods used to test them, as well as the format of the JSON to be submitted (for e.g. in the case of a PUT or POST request).

- Any additional info required for me to start your REST service and your REST client properly (for e.g. API keys that may need to be placed in the `application.properties` for the app to run correctly). Otherwise, I will just assume I can import your project as a Maven project into STS and then run it directly from the Boot Dashboard without any further modifications.

The documentation can be included as a Microsoft Word document or text file in the root of your Spring Boot project folder, or emailed to me separately.