RESTful Notes API Guide

Depanker Sharma

Table of Contents

Overview

These are the API's to facailtate currency exchange.

Chapter 1. General

1.1. HTTP verbs

Verb	Usage
`GET`	Used to retrieve a resource (idempotent)
`POST`	Used to create a new resource
`PUT`	Used to update an existing resource (idempotent)
`DELETE`	Used to delete an existing resource

Chapter 2. HTTP status codes

RESTful notes tries to adhere as closely as possible to standard HTTP and REST conventions in its use of HTTP status codes.

Status code	Usage
`200 OK`	The request completed successfully
`202 Accepted`	The request has been accepted for processing, but the processing has not been completed. The request might or might not be eventually acted upon, and may be disallowed when processing occurs.
`204 No Content`	An update to an existing resource has been applied successfully
`400 Bad Request`	The request was malformed. The response body will include an error providing further information
`404 Not Found`	The requested resource did not exist
`409 Not Found`	Attempt to create a duplicate resource
`500 Internal server error`	Any Unhandled
`502 Bad Gateway`	When and external service fails

Chapter 3. Errors

Whenever an error response (status code >= 400) is returned, the body will contain a JSON object that describes the problem. The error object has the following structure:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json;charset=UTF-8
Content-Length: 62

[ {
  "field" : "fieldName",
  "message" : "Error message"
} ]
```

How to run the software

To run the software, use the terminal, `cd` to `currencyconverter` type `./gradlew clean build`. After the process gets successfully competed a jar `currencyconverter-0.0.1-SNAPSHOT.jar` will get created under build/libs directory Now there are two ways to run the jar

1. Execute `java -jar build/libs/currencyconverter-0.0.1-SNAPSHOT.jar`. Once the jar is up, below mentioned curl request's could be used as test cases.

2. Alternatively if docker is installed on the system following command can start the application as Docker container

2.1 Build the docker image `docker build -t depanker/demo-currency-converter .` or pull the image from dockerhub using `docker pull depanker/demo-currency-converter`

2.2 Once the above command has been successfully installed, execute the following `docker run -p 8080:8080 -t depanker/demo-currency-converter`

Resources

Request

```
$ curl 'http://localhost:8080/currency/convert' -i -X POST \
    -H 'Accept: application/json' \
    -H 'Content-Type: application/json' \
    -d '{
  "from" : "EUR",
  "to" : "USD",
  "amount" : 3.14
}'
```

Response

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 77

{
  "from" : "EUR",
  "to" : "USD",
  "amount" : 3.14,
  "converted" : 3.49
}
```

.3. Error

Request when not data is present

```
$ curl 'http://localhost:8080/currency/convert' -i -X POST \
    -H 'Accept: application/json' \
    -H 'Content-Type: application/json' \
    -d '{
  "from" : null,
  "to" : null,
  "amount" : null
}'
```

Response

```
HTTP/1.1 400 Bad Request
Content-Type: application/json
Content-Length: 178

[ {
  "field" : "to",
  "message" : "must not be blank"
}, {
  "field" : "from",
  "message" : "must not be blank"
}, {
  "field" : "amount",
  "message" : "must not be null"
} ]
```

Request with invalid values

```
$ curl 'http://localhost:8080/currency/convert' -i -X POST \
    -H 'Accept: application/json' \
    -H 'Content-Type: application/json' \
    -d '{
  "from" : "EURO",
  "to" : "USD ",
  "amount" : 0.0
}'
```

Response

```
HTTP/1.1 400 Bad Request
Content-Type: application/json
Content-Length: 211

[ {
  "field" : "to",
  "message" : "Size must be equal to 3"
}, {
  "field" : "amount",
  "message" : "must be greater than or equal to 0.01"
}, {
  "field" : "from",
  "message" : "Size must be equal to 3"
} ]
```

When both the external services fail

Response

```
HTTP/1.1 502 Bad Gateway
Content-Type: application/json
Content-Length: 94

[ {
  "message" : "No provider available at the moment, please try again after some time."
} ]
```

Bonus Tasks

.1. How to secure API endpoint

.1.1. JWT is the most common authencation menchanism in API's. If required by a vendor

we can secure a API using JWT and also set an expery to the token. It would also mean that we would need an api where the client can regenerate the tokens.

We would also need to make sure that we use SSL as JWT will only mitigate the Attack on data integrity but not on confidentiality.

When communicating server to server, security could be managed by whitelisting IP's

.2. How to improve API's performance:

.2.1. *Internal caching vs External caching*

Internal caching:

Pros:

1. Can be easily implemented using a spring scheduler.
2. Do not require any object Serialization or Deserialization.
3. Not dependent on any network connection as the data resides in JVM heap.
4. A secure API endpoint could be exposed, which when invoked reloads the cache.

Cons:

1. Data is tightly bound to the application and not abstracted to an external data source.

2.a In a production setup where multiple servers are running, there is no clean way to refresh the cache.

Internal caching:

Pros:

1. Central cache ensures that update one location will reflect everywhere.
2. Provides more control over the change, example if we have stored the data in Redis cache as a Hash data structure we can directly update one single value, without needing to update the entire cache.

Cons:

1. Adds a network overhead, can be very little but still an overhead.
2. Prone to network failures.
3. Added cost of object Serialization and Deserialization.
4. Additional infrastructure cost.

.2.2. *This functionality could be exposed to a severless architecture.*

Pros:

1. Lower costing is achieved as there is no managing of servers.
2. Elasticity is managed by the vendor.

Cons:

1. Cold start creates an overhead but can be managed by repeatedly hitting the service within a given interval, it will still cost less.