Breweries Rest API Report

Table of Contents

Introduction

Development of an API is a cycle of that never-ending choices whether those choices are to create a specific architecture and design of the API because development lifecycle of API holds a various possible option for the aspiring developer to develop API. While developing the Breweries Rest API I have considered more about the API development lifecycle. Also, using HTTP to obtain data and generate operations on those data to all possible formats, for instance XML and JSON. This is more popular alternative to create standard data capacity but are also very complex to developed.

Configuration API

Before looking at the controller themselves, first we need to setup the web project and servlet configuration.

Web.xml

Spring MVC provides a despatcher servlet that receives incoming requests and routes them to suitable controllers. So, it requires declaring tis dispatcher servlet into web.xml file and configure the URL for mapping the servlet.

Swagger UI

I am using the swagger UI for this project to generate the documentation and make it more user interactive. Before started the swagger UI we need to do some configuration which are the flowing.

Java configuration

The configuration of the swagger is all about the Docker bean. Here is the code snipped for the configuration and the enabling the swagger 2 through the @EnableSwagger2 annotation.

```
 */
@EnableSwagger2
@Configuration
public class SwaggerConfig {

    @Bean
    public Docket productAPI() {
        return new Docket(DocumentationType.SWAGGER_2)
                .select()
                .apis(RequestHandlerSelectors.basePackage("com.ruchi.breweriesrestapi"))
                .paths(regex("/rest.*"))
                .build();

    }
}
```

To use the swagger UI we need to add some dependency on pom.xml file, here are the dependency that I use in pom.xml file.

```
    <type>jar</type>
</dependency>
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger2</artifactId>
    <version>2.9.2</version>
</dependency>
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger-ui</artifactId>
    <version>2.9.2</version>
</dependency>
<dependency>
```

Beer API

To build a Beer API I have given the Crud action for all methods, which are working correctly and retuning the JSON format when selecting the beer Id, I was getting JAXBContext errors. After adding the JAXBcontext dependency on the pom.xml file I was able to fix this problem. Also, I have given the pagination to get maximum numbers of row and page on the browser. Breweries API is same as the Beer API only difference is, they have different service class and the methods.

```
        <artifactId>jaxb-api</artifactId>
        <version>2.2.3</version>
    </dependency>
    <dependency>
        <groupId>com.sun.xml.bind</groupId>
        <artifactId>jaxb-impl</artifactId>
        <version>2.3.0.1</version>
    </dependency>
    <dependency>
        <groupId>javax.activation</groupId>
        <artifactId>activation</artifactId>
        <version>1.1.1</version>
    </dependency>
    <dependency>
        <groupId>com.sun.xml.bind</groupId>
        <artifactId>jaxb-core</artifactId>
        <version>2.3.0.1</version>
    </dependency>
```

Beer API Client/Server protocol

Individually HTTP contains all the necessary information to run it, which mean client or server side both no need to remember the previous state of completion .There are four most important data transitions in this API system and HTTPs specification: POST, GET, PUT, and DELETE operation that use in the data transfer. Also, the benefit of the web and its core protocol, HTTP provide a stack of features, that provide the caching, Redirection and forwarding.

Identification of Resources:

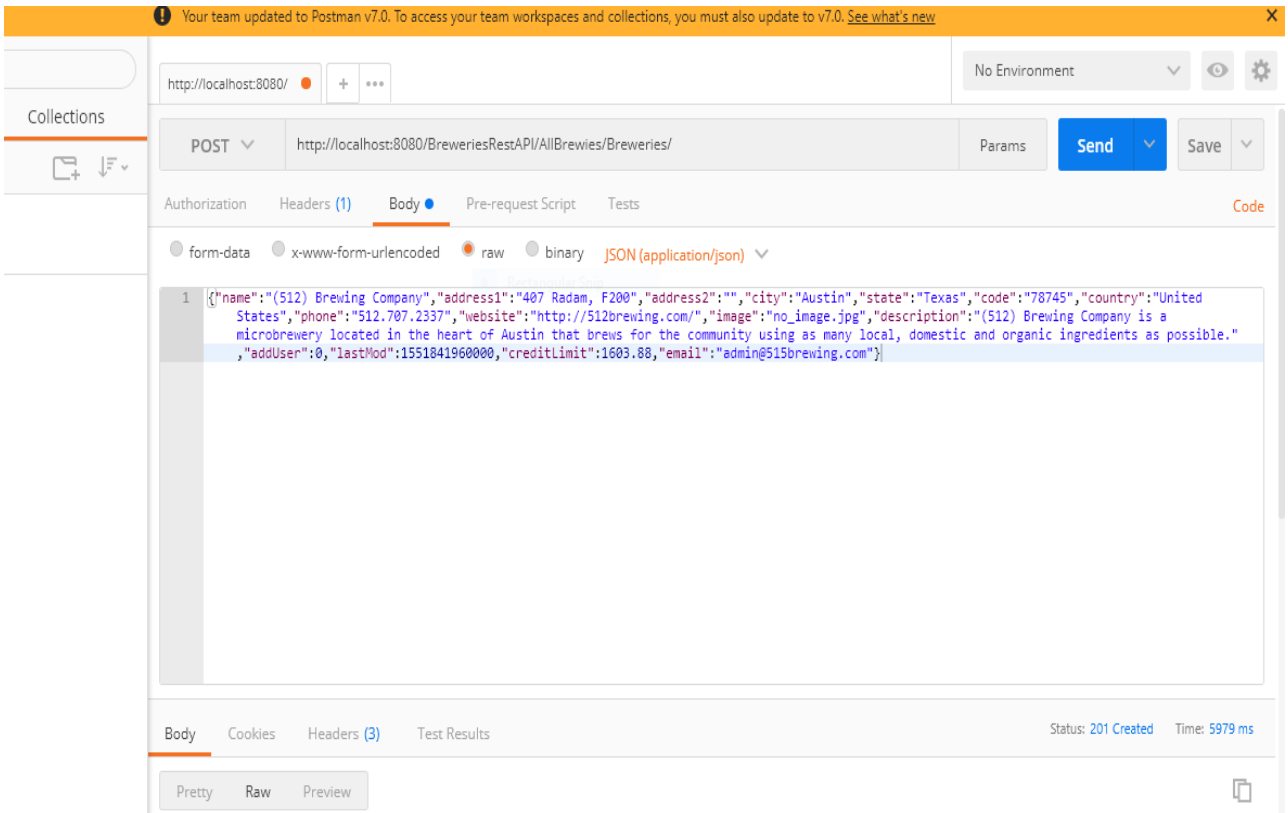
This basically states that a request will need to identify the resources which it is looking for URL. Furthermore, the resources themselves may not have any relationship with how they are returned to the client (for example in this API has a request in JSON format). @Request Body binds the parameters of the method to the body of the HTTP request.

Test Tool

For the testing, the rest API or any other API, postman is the best tool when trying cut up restful APIs that could be made by others or self-made APIs.

This gives polished user interface with HTML request, without writing a bunch of code just test an API functionality. Postman can run PUT, PATCH,DELETE, and various other request and methods which help to develop APIs.

Third party tool: While developing API I 'am using the Postman for test all request. Blow is the screen shot for the post request. Where I have given the URL pattern and the object to insert into the breweries table, that return the status 201 which shows its created successfully.

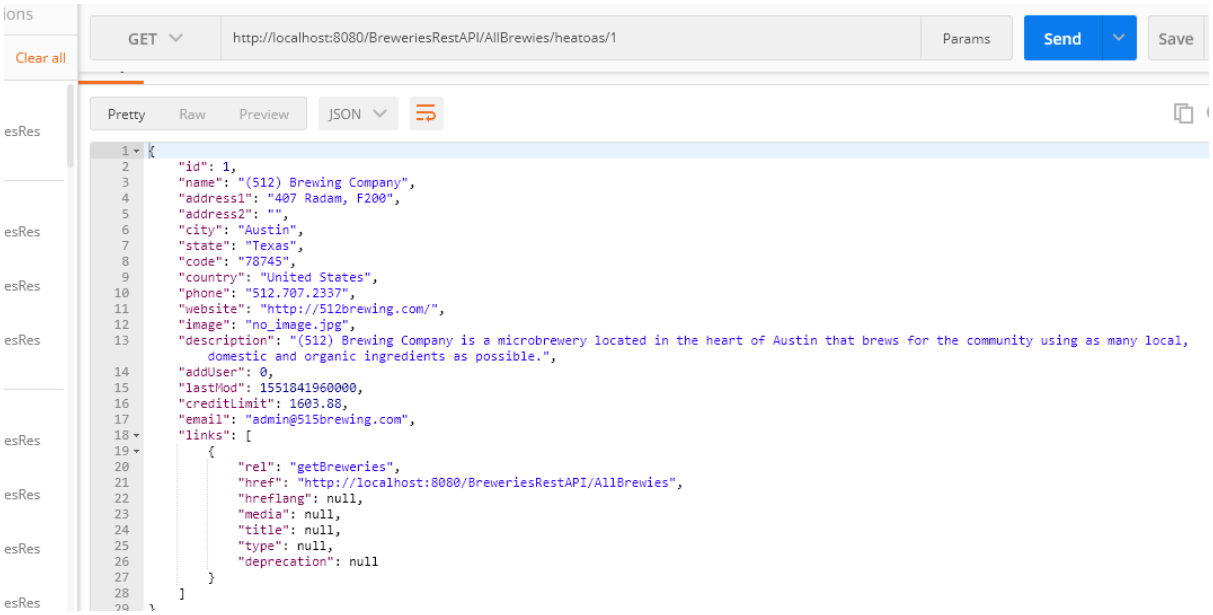


HATEOAS

HATEOAS is an important aspect of REST, because it allows you to build services that decouple client and server to a large extent and let them

evolve separately. The representations returned for REST resources contain not only data but also links to related resource.

Applying Spring HATEOAS in API to provides some of creating the REST API HATEOAS is an extra level upon REST and is used to present information about a REST API to a client, allowing for a better understanding of the API without the need to bring up the specification or documentation. This is done by including links in returned response and using only these links to further communicate with the server. Because the HATEOAS reduce the link to the other page. Below is the example of Breweries Rest API that using the HATEOAS.



Jackson data bind:

This library provides implicit conversion between JSON and POJO classes. When this library is imported into pom.xml you do not have to worry about converting JSON requests into POJO response. Because this is fully handled by library.

Benchmarking

Overall project is working well I can see the result by each endpoint also see the individual's API controller. It involves considering high level aspects such as core competences, developing API and its services has improving capabilities for dealing with changes in the environment. Mainly improving specific request send by the user.