



3/17/2022

Assignment 2 Report

Global Beers REST API

K00235626: Mario Barsoum
TUS

Table of Contents

Documentation	2
Self-Evaluation	6
Crud Repository	6
H2 Database	6
Bean Validation	6
Exception Handling	6
Method mapping and Media type specification.....	7
HATEOAS principles	8
QR Code Generation	9
PDF Brochure	9
Benchmarking and Enhancements	10
Benefits	10
Improving production	10
Cost Effective	10
Enhances the user experience	10
Enhancements	11
Search by Timestamps of modification:.....	11
Pagination:	11
JSON support / REST:	11

Documentation

For this project, I'm utilizing the swagger UI to produce the documentation and make it more user-friendly. I have done so by adding the dependency springdoc-openapi-ui to my pom.xml file. I have then set the spring documentation swagger UI path to /swagger-ui-custom.html.

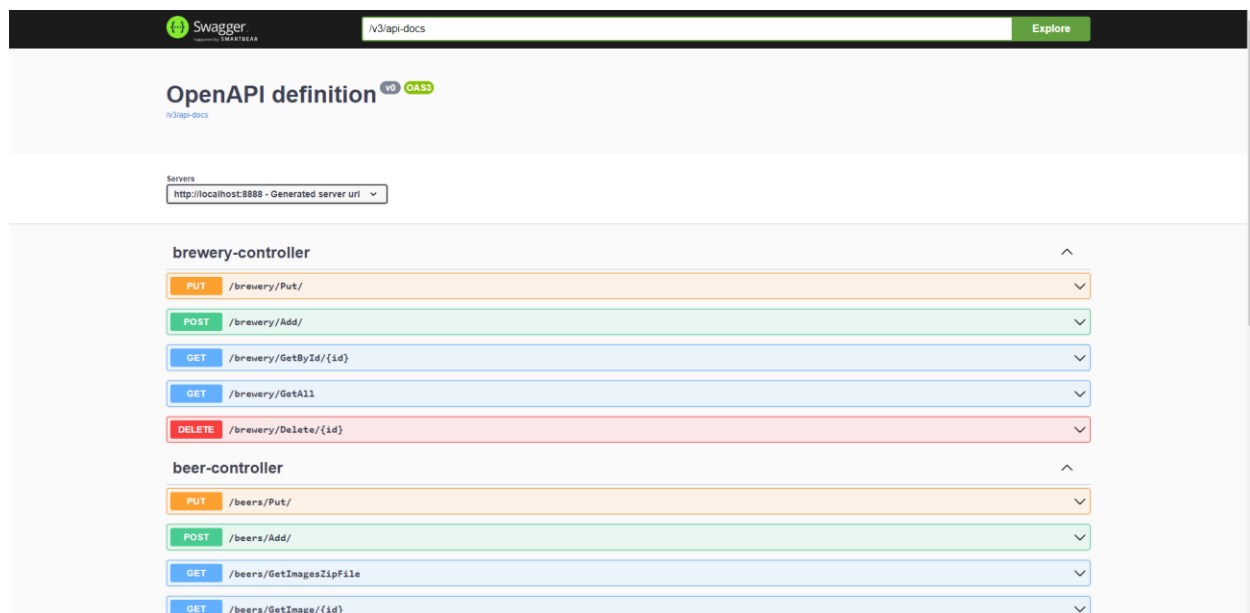
```
<dependency>
  <groupId>org.springdoc</groupId>
  <artifactId>springdoc-openapi-ui</artifactId>
  <version>1.6.5</version>
</dependency>
```

Figure 1: pom.xml

```
springdoc.swagger-ui.path=/swagger-ui-custom.html
```

Figure 2: application.properties

To launch the swagger UI the following link <http://localhost:8888/swagger-ui/index.html> can be accessed while the spring boot REST API is running. This should open the following page



There are a few advantages of using Swagger such as

- That it synchronizes the API documentation with the server and client at the same pace.
- Allows us to generate REST API documentation and interact with the REST API. The interaction with the REST API using the Swagger UI Framework gives clear insight into how the API responds to parameters.
- Provides responses in the format of JSON and XML.

Here is an example of the produced documentation for one of the GET Requests that returns a QR code that once scanned on a smartphone will attempt to add the brewery to a contact. By expanding the GET request tab it outlines the parameters needed to run this request and as shown in the screenshot below it also allows you to execute the request.

qr-code-rest-controller

GET /brewery/QRCode/{id}

Parameters

Name	Description
id * required integer(\$int64) (path)	<input type="text" value="2"/>

Execute Clear

Responses

Curl

```
curl -X 'GET' \
'http://localhost:8888/brewery/QRCode/2' \
-H 'accept: image/jpeg'
```

Request URL

http://localhost:8888/brewery/QRCode/2

Server response


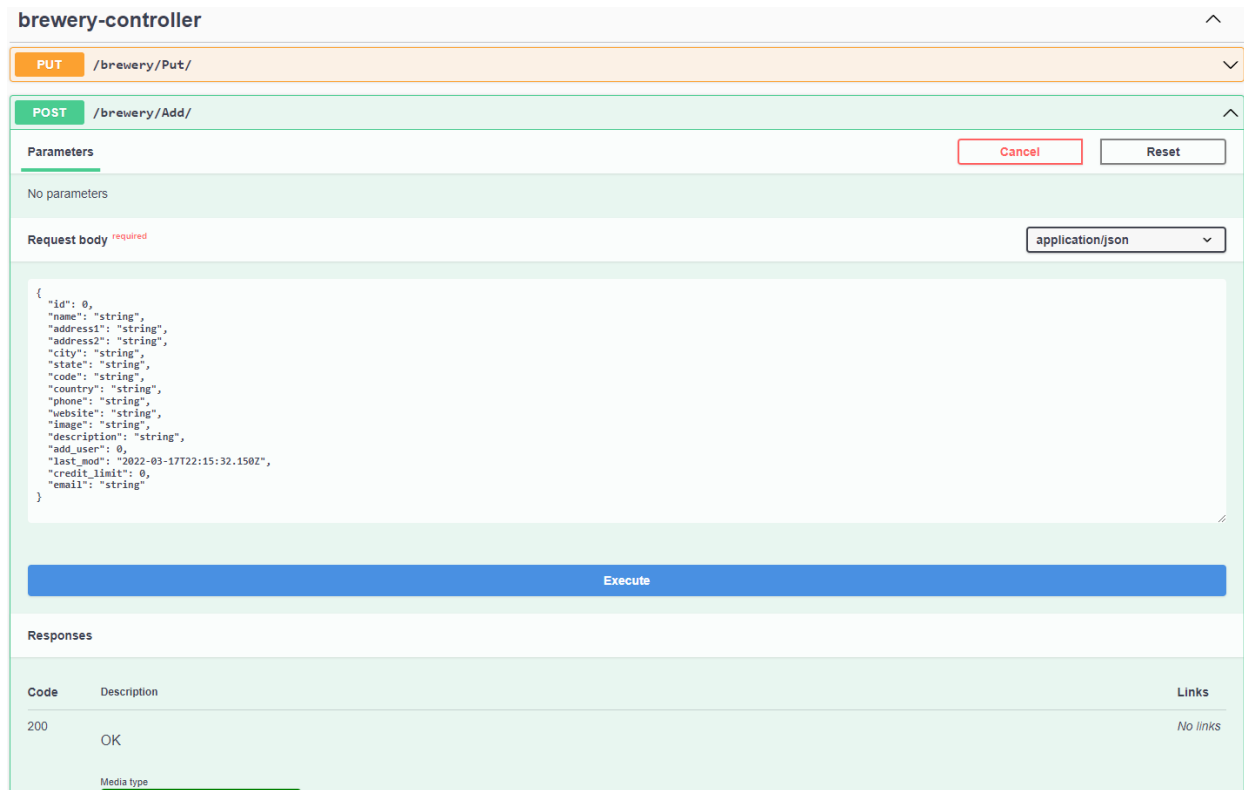
Code	Details
200	<p>Response body</p> 

Figure 3: Swagger UI used to execute a GET Request

Another example shown in the screenshot below is that it also provides the request body needed in the media type that has been provided in API, in this case JSON and XML.



The screenshot displays the Swagger UI for a REST API endpoint. At the top, a header bar shows the API name 'brewery-controller' and a dropdown menu. Below this, a navigation bar highlights the 'POST /brewery/Add/' endpoint. The main interface is divided into several sections: 'Parameters' (showing 'No parameters'), 'Request body' (with a dropdown set to 'application/json' and a required label), and 'Responses' (showing a 200 OK response). The 'Request body' section contains a JSON schema for a brewery object, including fields like id, name, address, city, state, code, country, phone, website, image, description, add_user, last_mod, credit_limit, and email. A large blue 'Execute' button is positioned below the request body. The 'Responses' section is a table with columns for Code, Description, and Links.

brewery-controller

PUT /brewery/Put/

POST /brewery/Add/

Parameters Cancel Reset

No parameters

Request body required application/json

```
{
  "id": 0,
  "name": "string",
  "address1": "string",
  "address2": "string",
  "city": "string",
  "state": "string",
  "code": "string",
  "country": "string",
  "phone": "string",
  "website": "string",
  "image": "string",
  "description": "string",
  "add_user": 0,
  "last_mod": "2022-03-17T22:15:32.150Z",
  "credit_limit": 0,
  "email": "string"
}
```

Execute

Responses

Code	Description	Links
200	OK	No links

Media type

Figure 4: Swagger UI for brewery POST method

Swagger UI also provides documentation for the data types of the used entities in the schemas tab at the bottom of the page. Figure 5 shows a screenshot of this.

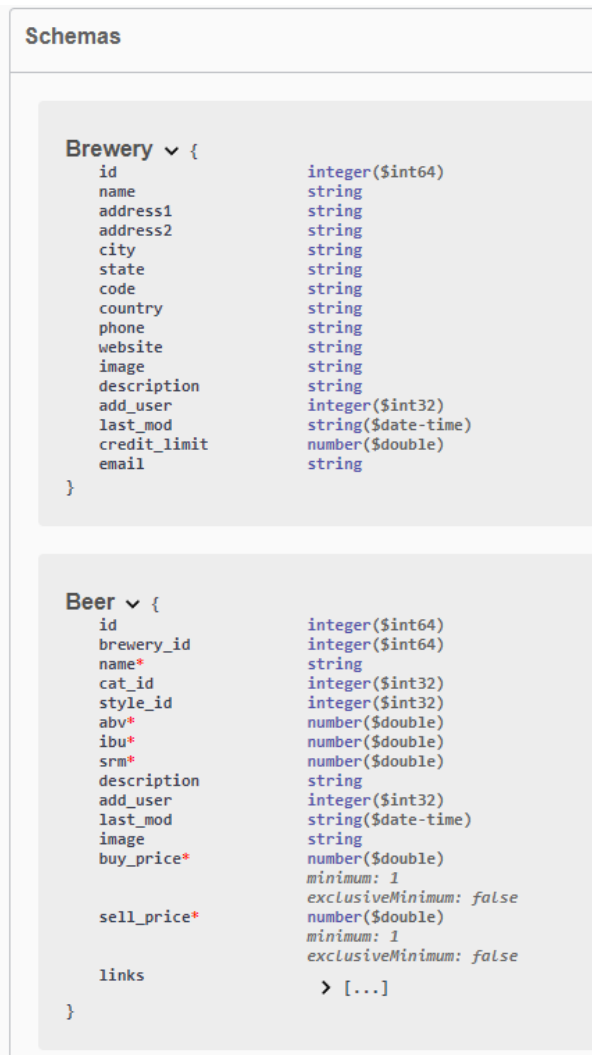


Figure 5: Swagger UI schemas

Self-Evaluation

The benefits of this API is that Web services allow different applications from different sources to communicate with each other without time-consuming custom coding. They are not tied to any one operating system or programming language

Crud Repository

This API is using the Crud Repository interface which provides methods for CRUD operations, so it allows you to create, read, update, and delete records without having to define my own methods.

H2 Database

A H2 database is also utilized in this Application. H2 is a disk-based or in-memory databases and tables, read-only database support, temporary tables. This comes with many benefits mostly being that it is an extremely fast database engine and open source written in java.

Bean Validation

One of the strengths of this API is that I am using Bean validation for the Beer entity. This is on the server side as client-side validation can be circumvented and if relying on JavaScript this may be switched off on the browser and that validated data on the client can be changed after leaving the browser. I have copied some of the validation I have had in place in the previous assignment, and I have set it to validate the input data on each POST and PUT method. If any parameters do not comply with the required conditions an error message will be sent back to the client stating what the error is. Figure 6 shows an example of this.

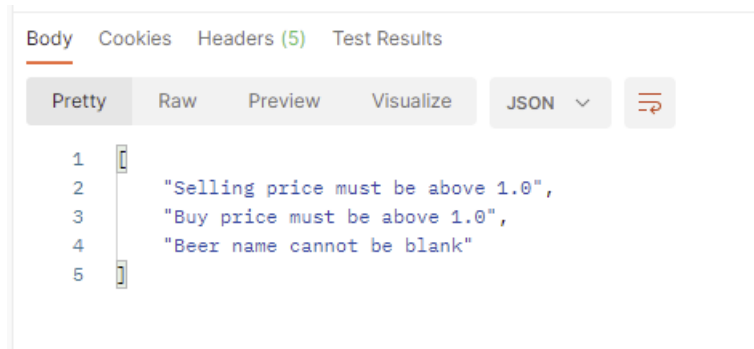


Figure 6: Returned body from violating the entity validation

Exception Handling

The exception handling in this API was not hugely focused on but there is some exception handling in this API. For the helpers that generate the PDF brochure for the beers and the ZXing QR code generator these methods have the throws {exception type} Exception for example the PDF helper class throws MalformedURLException and IOException. This ensures that I create higher quality code where errors are checked at compile time instead of runtime and create custom exceptions that make debugging and recovery easier.

This API also uses `ResponseStatusExceptions`. I can pass three arguments to the `ResponseStatusException` constructor. The Status - a HTTP status. A reason – a message explaining the exception and a cause – which is a throwable cause of the `ResponseStatusException`.

```
beerService.deleteByID(id);
return new ResponseEntity(HttpStatus.OK);
}

@PostMapping(value = "/beers/Add/", consumes = {MediaType.APPLICATION_XML_VALUE, MediaType.APPLICATION_JSON_VALUE})
public ResponseEntity add(@Valid @RequestBody Beer b, BindingResult result) {
    if (result.hasErrors()) {
        List<String> errors = result.getAllErrors().stream().map(e -> e.getDefaultMessage()).collect(Collectors.toList());
        return new ResponseEntity<>(errors, HttpStatus.NOT_ACCEPTABLE);
    }

    b.setId(0);
    b.setAdd_user(0);
    b.setLast_mod(new Date());
    beerService.saveBeer(b);
    return new ResponseEntity(HttpStatus.CREATED);
}

@PutMapping(value = "/beers/Put/", consumes = {MediaType.APPLICATION_XML_VALUE, MediaType.APPLICATION_JSON_VALUE})
public ResponseEntity edit(@Valid @RequestBody Beer b, BindingResult result) {
    if (result.hasErrors()) {
        List<String> errors = result.getAllErrors().stream().map(e -> e.getDefaultMessage()).collect(Collectors.toList());
        return new ResponseEntity<>(errors, HttpStatus.NOT_ACCEPTABLE);
    }
}
```

Figure 7: Response Status Exceptions

Advantages of using these for exception handling are that they are easy to use and exceptions of the same type can be processed separately and different status codes can be set on the response. It also avoids the creation of additional exception classes and gives more control over exception handling.

Method mapping and Media type specification

The actual value of the mapping, as well as the HTTP method, determine the target method for the request. Using the `produces` attribute on a request allows me to specify the MIME media types or representations a resource can produce and send back to the client. While using the `consumes` attribute on a request allows me to specify the MIME media types or representations a resource can consume from a client.

They also guarantee that the resource is marshalled and unmarshalled correctly using the appropriate HTTP converter. Here is an example of some method mapping and media type specification used in this API.

```
@GetMapping(value = "/beers/GetImagesZipFile", produces = "application/zip")
public void zipDownload(HttpServletRequest response) throws IOException {
    ZipOutputStream zipOut = new ZipOutputStream(response.getOutputStream());
    Resource resource = new ClassPathResource("static/assets/images/");

@GetMapping(value = "beers/GetBeerPDF/{id}", produces = {"application/json", "text/xml"})
public ResponseEntity<?> BeersPDF(@PathVariable long id) throws FileNotFoundException, DocumentException, IOException {
    Optional<Beer> beer = beerService.findOne(id);
}
```

Figure 8 Method mapping and Media type specification

HATEOAS principles

The GET requests in this API adhere to the HATEOAS principles (“Hypermedia as the engine of application state”). By providing hypermedia links with the responses, a hypermedia-driven site enables information to navigate the site's REST APIs dynamically. The basic concept of hypermedia is to use hypermedia components to enhance the depiction of a resource. Figure 9 illustrates the hyperlink returned in the JSON that can further enhance the navigation in this case drilldown to the specific Beer.



Figure 9: HATEOS Example from the API

This is also another example from the API that allows the navigation back to view all or to further drilldown to get more details.



QR Code Generation

I used ZXing (Zebra Crossing) to generate the QR codes. It is an open-source, multi-format 1D/2D barcode image processing library implemented in Java, with ports to other languages. The whole idea was to create a QR code that once it is scanned by a smartphone it will attempt to add the brewery that is generated for to your contacts. This has been achieved by creating a vCard with the required details and encoding it to a QR code. A vCard is both compatible with Android and IOS and over all this method is a success. Figure 10 will show an example QR code generated from this API.



Figure 10: QR code generated from API

PDF Brochure

The library used to generate the beer pdf brochure in this API is iText. This is a library for creating and manipulating PDF files in Java and .NET. The end result of the PDF was achieved by adding paragraphs and chunks to the document with the required data for the chosen beer. As I have taken a different approach at the beginning and that was to write up my design in HTML format and then convert it to a PDF. This was an easy method and had required way less code, but I started to face issues in adding and formatting the image in the PDF. I have then decided that I was going to take a different approach and that was to draw out the document using code which is available in the GeneratePDFReport Helper class. Figure 11 show a sample of the brochure for one of the beers.

Hocus Pocus

ABV
4.6

Description:
Our take on a classic summer ale. A toast to weeds, rays, and summer haze. A light, crisp ale for mowing lawns, hitting lazy fly balls, and communing with nature, Hocus Pocus is offered up as a summer sacrifice to cloudless days. Its malty sweetness finishes tart and crisp and is best appreciated with a wedge of orange.

Sell Price:
13.12

Brewery Name:
Magic Hat

Brewery Website:
<http://www.magicchat.net/>

Beer Category:
Other Style

Style:
Light American Wheat Ale or Lager


A photograph of a dark glass beer bottle with a yellow and orange label featuring the text 'Hocus Pocus' and 'Magic Hat'.

Figure 11: Sample PDF brochure produced by the API

Benchmarking and Enhancements

An API (application programming interface) is a piece of software that allows two or more programs to communicate with one another. Simply said, the messenger is the person who delivers your request to the supplier and then returns the result to you. It's a user interface that lets one program communicate with another using simple instructions.

Benefits

Improving production

Developers don't normally start from scratch when writing code. APIs allow users to use pre-existing frameworks to develop code and software, allowing them to focus on the unique value proposition of their apps rather than attempting to start from scratch every time they build a new application.

API integrations allow companies to optimize processes by combining their software and databases with industry-standard apps. They guarantee smooth and steady interaction across numerous applications, allowing employees to reap the benefits of several cloud-based services while also improving product development. APIs enable organizations to implement new solutions in an agile, practical, and cost-effective manner.

Cost Effective

The cost of developing an application varies based on several factors, including the project's complexity, the technology employed, and the developers' skills. One of the most significant advantages of APIs for organizations is the possibility to save money. Because APIs greatly reduce development work, employing them to construct apps is an excellent approach to cut expenses. Developers may use APIs to get the majority of the functionality they need to construct programs from elsewhere, eliminating the need to start from scratch.

Enhances the user experience

Enterprises may establish innovative and effective methods of communicating with customers by exploiting API capabilities, especially in the present digital age when consumers want top-notch experiences. Customers today are primarily interested in individualized experiences rather than one-size-fits-all corporate solutions. Organizations that expose their data and services through APIs may enable their API customers to govern their own customer experiences, opening up a world of possibilities. Developers may use APIs to design solutions that fulfill particular client requirements, which would be impossible to do without them.

Enhancements

Search by Timestamps of modification:

A decent API should let you search datasets by a variety of parameters, the most essential of which is the date. Simply because it is the modifications that we are most interested in following the first data synchronization. In other words, we require the information that has changed (updated, removed, rectified, etc.) or been added since the previous time the synchronization was triggered.

Pagination:

Naturally, there may be large volumes of data, even if this is merely altered data. To deal with it efficiently, we need a mechanism to declare that we don't require all the modified data in one sitting, but only the first "page" of it. For example, of a thousand data records in size.

This is the purpose of paging. A decent API should be able to limit the quantity of data that may be received in one go, as well as the frequency of data queries. It should also be able to tell you how many "pages" of data are remaining.

JSON support / REST:

To be fair, an API does not have to be RESTful in order to be regarded excellent. However, most new APIs are REST APIs that, by default, handle JSON, and there are several good reasons for this.

REST APIs are stateless, which makes them ideal for applications that require a significant amount of back-and-forth transmission, such as mobile apps. If an upload to a mobile app is stopped working, for example, a loss of service, REST APIs make it very simple to repeat the operation. This is also achievable using SOAP, but with a lot more work. Furthermore, because they employ basic URIs for communication, REST APIs are lightweight and more compatible with the web.

REST APIs handle a variety of formats, including JSON; additional examples include TXT, CSV, and XML. This implies that you, as a developer, have a choice between several formats (as opposed to SOAP, which only supports XML), and may select the one that best suits your needs.

However, using JSON with REST is considered best practice. Specifically, unlike XML, the syntax of JSON is quite near to those of most programming languages, making it very straightforward to understand in virtually any language. Not to add that JSON is really simple to construct.