SEG3102 – Lab 6
REST Web Services

The objective of this lab is to introduce to RESTful Web Services programming with Angular and Springboot. We will (1) develop an Angular application to consume a publicly available Web Service and (2) create a RESTful Web Service API using Springboot.

The source code for the Angular application is available at https://github.com/stephanesome/weather-app.git and the source code for the Springboot application at https://github.com/stephanesome/converter-api.git.

1. Weather Application

We are going to develop an Angular application to display the current weather conditions of a city. The application presents a form in which the user can specify the city of interest and optionally select a country where this city is located. Upon submission, the application displays the current weather conditions of that city.

We will use a RESTful Web Service API provided by Open Weather Map (`http://openweathermap.org/`) to obtain current weather conditions.

API Key

You need an API Key from OpenWeatherMap that you can get that by subscribing for a free account. Go to `http://openweathermap.org/api` select to *Subscribe for Current Weather Data* and follow the instructions.

Project Setup

Generate an angular application (`ng new weather-application`). Routing is not needed and we will use CSS as Stylesheet format.

Add bootstrap to the project (`ng add @ng-bootstrap/ng-bootstrap`).

We will also use a component from Angular Material (`https://material.angular.io/`) to select countries.

- Add Angular Material with command `ng add @angular/material`. Select the default choices.
- Add the `select-country` extension to the project (`https://www.npmjs.com/package/@angular-material-extensions/select-country`)
  - Install country flag icons with `npm i svg-country-flags -s`
  - Update the configuration file `angular.json` by adding

```
{
...
"projects": {
   "weather-application": {
   ...
"architect": {
   "build": {
    ...
   "options": {
       ....
      "assets": [
          "src/favicon.ico",
          "src/assets",
          {
            "glob": "**/*",
            "input": "./node_modules/svg-country-flags/svg",
            "output": "/assets/svg-country-flags/svg"
          }
        ],
....
```

- Install the angular material extension with command
  `npm install --save @angular-material-extensions/select-country`

- Import the MatSelectCountryModule,the HttpClientModule and the ReactiveFormsModule to the AppModule. Edit `src/app/app.module.ts` and add the following

```
import {MatSelectCountryModule} from '@angular-material-extensions/select-country';
import {HttpClientModule} from '@angular/common/http';
import {ReactiveFormsModule} from "@angular/forms";

@NgModule({
  ...
  imports: [
    ...
    MatSelectCountryModule.forRoot('en'),
    HttpClientModule,
    ReactiveFormsModule
  ],
   ...
})
export class AppModule { }
```

Weather Model

We create a model class to represent weather information returned by the API call. Run command `ng generate class model/weather --type=model` to generate a model class. Edit `src/app/model/weather.model.ts` as follow

```
1.  export class Weather {
2.    mainCondition: string;
3.    temperature: number;
4.    pressure: number;
5.    humidity: number;
6.    windspeed: number;
7.    city: string;
8.    country: string;
9.
10.   constructor(obj?: any) {
11.     this.mainCondition = obj && obj.mainCondition || null;
12.     this.temperature = obj && obj.temperature || null;
13.     this.pressure = obj && obj.pressure || null;
14.     this.humidity = obj && obj.humidity || null;
15.     this.windspeed = obj && obj.windspeed || null;
16.     this.city = obj && obj.city || null;
17.     this.country = obj && obj.country || null;
18.   }
19. }
```

Class `Weather` defines fields for the information about weather that we're interested in. The constructor builds an instance from the available information.

Weather API access Service

We use a service to interact with the OpenWeather API. This allows the rest of the code to be abstracted from the interaction details. It could also make it easier to switch to another weather service.

Generate a service `ng generate service service/open-weather`.
Edit `src/app/service/open-weather.service.ts` as follow.

```
1.  import { Injectable } from '@angular/core';
2.  import {HttpClient, HttpErrorResponse, HttpParams} from '@angular/common/http';
3.  import {Observable, throwError} from 'rxjs';
4.  import {catchError} from 'rxjs/operators';
5.
6.  const baseUrl = 'http://api.openweathermap.org/data/2.5/';
7.  const APPID_HEADER = 'YOUR_API_ID';
8.  const resource = 'weather';
9.
10. @Injectable({
```

```
11.   providedIn: 'root'
12. })
13. export class OpenWeatherService {
14.   constructor(private httpClient: HttpClient) { }
15.
16.   public getWeatherAtCity(city: string, country: string): Observable<unknown> {
17.     const params = new HttpParams().set('q', city + ',' + country).set('appid', APPID_HEADER);
18.     const options = {params, responseType: 'json' as const};
19.     return this.httpClient.get(baseUrl + resource, options).pipe(
20.       catchError(this.handleError)
21.     );
22.   }
23.
24.   private handleError(error: HttpErrorResponse): Observable<never> {
25.     return throwError(() =>
26.       'Error - Unable to retrieve Weather Condition for Specified City.');
27.   }
28. }
```

The Service uses the HTTPClient (injected in line 14) to issue requests to the API. Make sure to specify your Open Weather API Token in line 7. Function getWeatherAtCity constructs the query by setting parameters (line 17) with the requested city, country and API Token, and by setting the expected response type (line 18). The request is send as HTTP GET.

HTTP Requests are processed asynchronously. An HTTPClient call returns an Observable to which the requestor subscribes. We use the pipe operator to pass any error returned to function handleError (lines 24-27). The function simply ensure we have a standard error treatment by returning an Observable that emits an error notification.

App Component

The App Component is the only component used in this application. It presents a form to get the city and country of interest from a user, uses the Weather Service to obtain weather data and displays to the user.

Edit the App Component Class (`src/app/app.component.ts`) as follow.

```
1.  import {Component, OnInit} from '@angular/core';
2.  import {FormBuilder, FormControl, FormGroup} from '@angular/forms';
3.  import {Country} from '@angular-material-extensions/select-country';
4.  import {Weather} from './model/weather.model';
5.  import {OpenWeatherService} from './service/open-weather.service';
6.  import {noop} from 'rxjs';
7.
8.  function parseResponse(response: any): Weather {
9.    return new Weather({mainCondition: response.weather[0].main,
10.       temperature: response.main.temp - 273.15,
```

```
11.       pressure: response.main.pressure,
12.       humidity: response.main.humidity,
13.       windspeed: response.wind.speed,
14.       city: response.name,
15.       country: response.sys.country
16.     }
17.   );
18. }
19.
20. @Component({
21.   selector: 'app-root',
22.   templateUrl: './app.component.html',
23.   styleUrls: ['./app.component.css']
24. })
25. export class AppComponent implements OnInit {
26.   title = 'weather-application';
27.   weatherForm: FormGroup;
28.   condition: Weather;
29.   message: string;
30.   currentDate: number;
31.
32.
33.   constructor(private formBuilder: FormBuilder,
34.               private weatherService: OpenWeatherService) {
35.   }
36.
37.   ngOnInit(): void {
38.
39.     this.weatherForm = this.formBuilder.group({
40.       country: [],
41.       city: []
42.     });
43.   }
44.
45.   submit(): void {
46.     const selectedCountry: Country = this.weatherForm.get('country').value;
47.     const selectedCity: string = this.weatherForm.get('city').value;
48.     this.weatherService.getWeatherAtCity(selectedCity, selectedCountry.alpha2Code).subscribe(
49.       (response: any) => {
50.         this.message = null;
51.         this.currentDate = Date.now();
52.         this.condition = parseResponse(response); },
53.       (error: any) => {this.condition = null; this.message = error; }
54.     );
55.   }
56. }
```

Function parseResponse (lines 8-18) takes a JSON object, extracts relevant information and creates a Weather model object from that information. The class sets a ReactiveForm to be used by the

template and handles the form submission. Responses from the Weather Service Observable (injected in line 34) are parsed and made available to the template (lines 49-52) while, returned errors set an error message to be displayed (line 53).

Edit the App Component Template as follow.

```
1.  <div class="container-sm">
2.    <form [formGroup]="weatherForm" (ngSubmit)="submit()" >
3.      <div class="row" >
4.        <div class="col-sm">
5.          <label>
6.            Country
7.            <mat-select-country appearance="outline"
8.                                formControlName="country">
9.            </mat-select-country>
10.         </label>
11.       </div>
12.       <div class="col-sm">
13.         <label>
14.           City
15.           <input type="text" class="form-control" name="city" formControlName="city">
16.         </label>
17.       </div>
18.       <label>
19.         <br>
20.         <div class="col-sm">
21.           <button type="submit" class="btn btn-success">Check Weather</button>
22.         </div>
23.       </label>
24.     </div>
25.   </form>
26. </div>
27. <div class="container" *ngIf="message">
28.   <div id="message">{{message}}</div>
29. </div>
30. <div class="container" *ngIf="condition">
31.   <div id="city">Current Conditions in {{condition.city}} {{condition.country}} - {{currentDate | date:
    'medium' }}</div>
32.   <br>
33.   <div class="container-sm">
34.     <div class="row">
35.       <div class="col-sm">
36.         <p>Main Condition:</p>
37.       </div>
38.       <div class="col-sm">
39.         <p>{{condition.mainCondition}}</p>
40.       </div>
41.       <div class="col-sm">
```

```
42.         <p>Temperature:</p>
43.       </div>
44.       <div class="col-sm">
45.         <p>{{condition.temperature | number}} Celsius</p>
46.       </div>
47.     </div>
48.     <div class="row">
49.       <div class="col-sm">
50.         <p>Pressure:</p>
51.       </div>
52.       <div class="col-sm">
53.         <p>{{condition.pressure | number}} hPa</p>
54.       </div>
55.       <div class="col-sm">
56.         <p>Humidity:</p>
57.       </div>
58.       <div class="col-sm">
59.         <p>{{condition.humidity | number}} %</p>
60.       </div>
61.     </div>
62.     <div class="row">
63.       <div class="col-sm">
64.         <p>Wind Speed:</p>
65.       </div>
66.       <div class="col-sm">
67.         <p>{{condition.windspeed | number}} meter/sec</p>
68.       </div>
69.       <div class="col-sm">
70.         <p><br></p>
71.       </div>
72.       <div class="col-sm">
73.         <p><br></p>
74.       </div>
75.     </div>
76.   </div>
77. </div>
```

We can see the select-country component in use at lines 7-9.

Edit the App Component CSS style file as follow.

```
1.  #city {
2.    -webkit-box-shadow: 5px 5px 15px 5px #000000;
3.    box-shadow: 5px 5px 15px 5px #000000;
4.    background-color: darkgrey;
5.    color: blue;
6.    font-weight: bold;
7.  }
8.
9.  #message {
```

```
    -webkit-box-shadow: 5px 5px 15px 5px #000000;
    box-shadow: 5px 5px 15px 5px #000000;
    background-color: beige ;
    color: red;
    font-weight: bold;
    font-size: large;
}
```

2. Temperature Converter Service

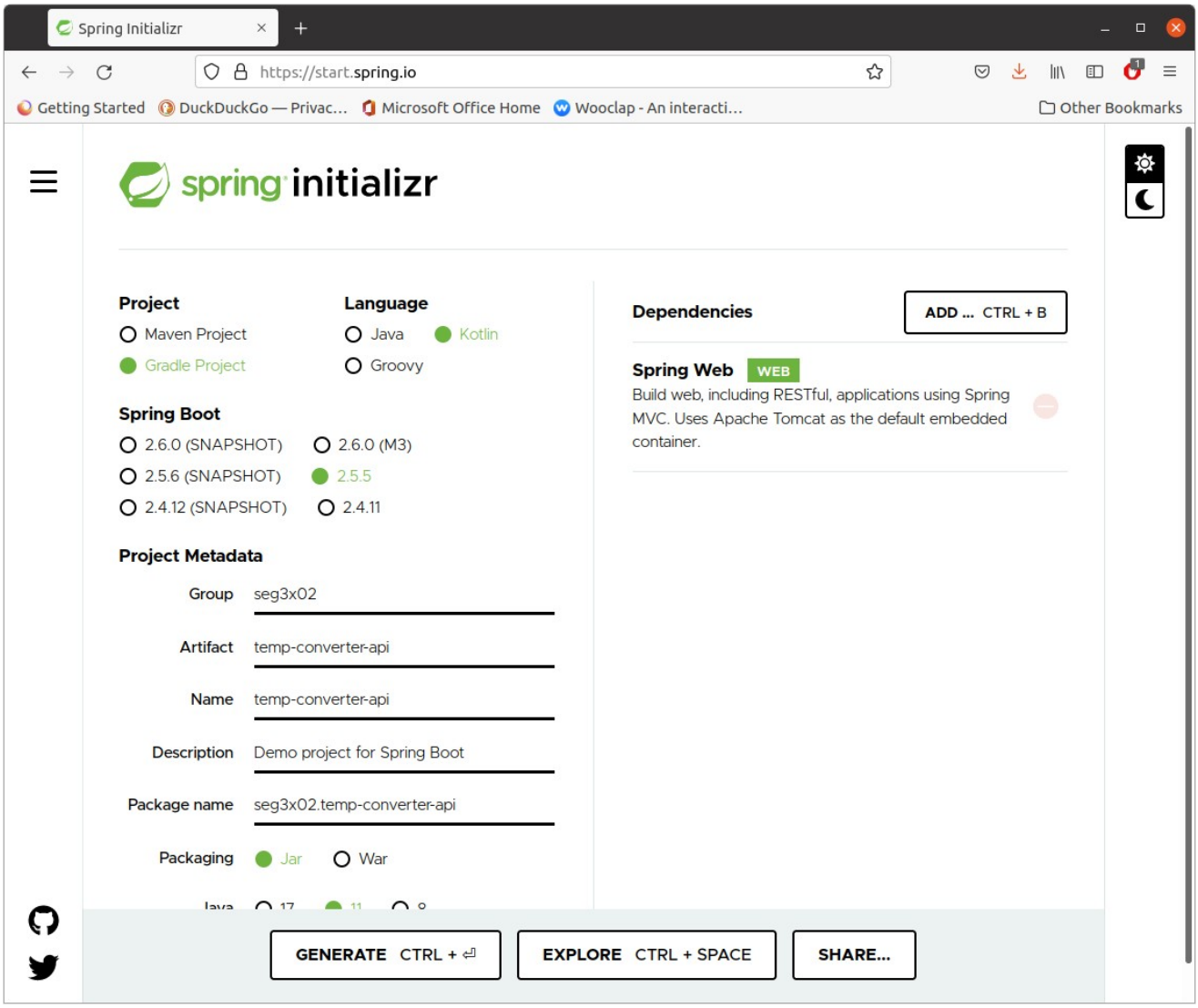
We implement a RESTful Web Service for temperature conversion with Springboot. The service exposes the following operations:

- ***GET temperature-converter/celsius-fahrenheit/{celsius}*** to return the Fahrenheit value of value *celsius*, and
- ***GET temperature-converter/fahrenheit-celsius/{fahrenheit}*** to return the Celsius value of value *fahrenheit*.

Project Setup

Create project with ***Spring Intializr***.

- Select *Gradle Project* as Project type and *Kotlin* as Language
- Leave the selected Spring Boot version to the default
- Enter the Project Metadata (Group: *seg3x02*, Artifact: *temp-converter-api*)
- Add the *Spring Web* dependency.



Generate, download, unzip and open the generated project in your IDE.

OpenAPI

We add a dependency to the springdoc-openapi library, to automatically generate OpenAPI 3 specification documentation for the API.

Edit file build.gradle.kts (in the project root folder) and add dependency org.springdoc:springdoc-openapi-ui:1.5.11. The dependencies declaration should look as follow:

```
1.  dependencies {
2.      implementation("org.springframework.boot:spring-boot-starter-web")
3.      implementation("org.springdoc:springdoc-openapi-ui:1.5.11")
4.      implementation("com.fasterxml.jackson.module:jackson-module-kotlin")
5.      implementation("org.jetbrains.kotlin:kotlin-reflect")
6.      implementation("org.jetbrains.kotlin:kotlin-stdlib-jdk8")
```

```
7.     ...
8.   }
```

The added dependency is in line 3. It will setup the automated generation of OpenAPI documentation based on the code.

REST Controller

Create a Kotlin class ConverterController in package seg3x02.tempconverterapi.controller and edit as follow.

```
1.  package seg3x02.tempconverterapi.controller
2.
3.  import org.springframework.web.bind.annotation.GetMapping
4.  import org.springframework.web.bind.annotation.PathVariable
5.  import org.springframework.web.bind.annotation.RequestMapping
6.  import org.springframework.web.bind.annotation.RestController
7.
8.  @RestController
9.  @RequestMapping("temperature-converter")
10. class ConverterController {
11.     @GetMapping("/celsius-fahrenheit/{celsius}")
12.     fun getFahrenheit(@PathVariable celsius: Double) = ((celsius * 9) / 5 + 32)
13.
14.     @GetMapping("/fahrenheit-celsius/{fahrenheit}")
15.     fun getCelsius(@PathVariable fahrenheit: Double) = ((fahrenheit - 32) * 5) / 9
16. }
```

The class is annotated with @RestController to register it as a controller for REST requests. The class-level @RequestMapping specifies *temperature-converter* as root URI for all the service end-points in the class. There are two end-points corresponding to the service operations. Each is defined as a function annotated with @GetMapping, a request mapping annotation for the HTTP Method GET.

REST Controller Advice

A Controller Advice defines exception handlers that apply globally to all the controllers in an application. Create a Kotlin class ControllerExceptionHandler in package seg3x02.tempconverterapi.controller and edit as follow.

```
1.  package seg3x02.tempconverterapi.controller
2.
3.  import org.springframework.http.HttpStatus
4.  import org.springframework.http.ResponseEntity
5.  import org.springframework.web.bind.annotation.ExceptionHandler
6.  import org.springframework.web.bind.annotation.ResponseStatus
7.  import org.springframework.web.bind.annotation.RestControllerAdvice
8.
9.  @RestControllerAdvice
10. class ControllerExceptionHandler {
```

```
11.     @ExceptionHandler
12.     @ResponseStatus(HttpStatus.BAD_REQUEST)
13.     fun handleException(ex: Exception): ResponseEntity<String> {
14.         return ResponseEntity.badRequest().body("Unable to process request")
15.     }
16. }
```

Annotation @RestControllerAdvice registers the class as a Controller Advice for REST Controllers. We define an exception handling function (that handles any exception here). The function returns a Response Entity with status code BAD_REQUEST (400) and an error message in the body.

Building and Running

Execute command `./gradlew bootRun` from the project main folder to build and run the project. Alternatively, you may use the run command of your IDE.

Once the application has started (note that for some reason, the progress indicator with ./gradlew bootRun remains at around 80%), you can check the API different ways:

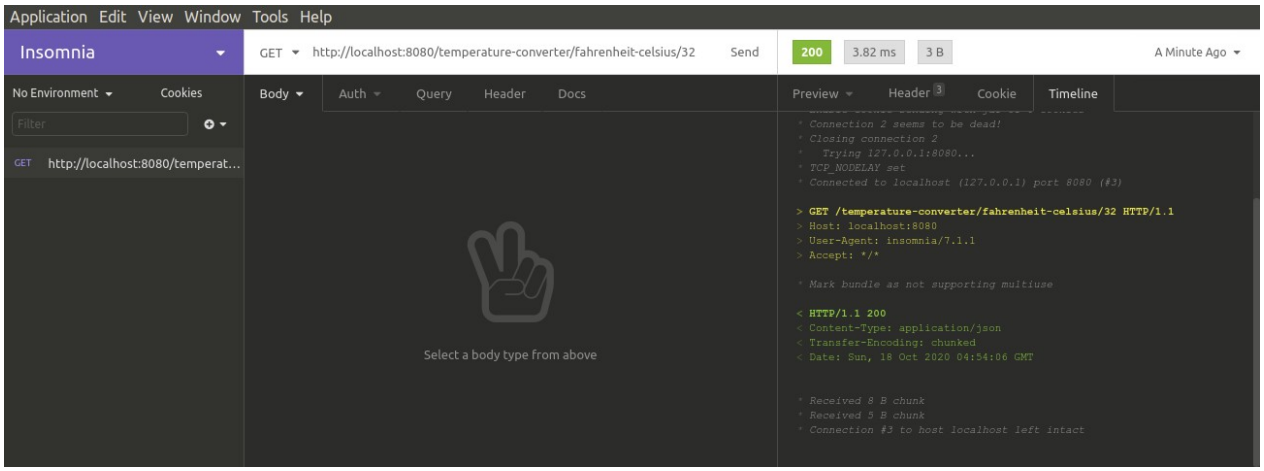
- On a browser, by entering URLs such as

  http://localhost:8080/temperature-converter/celsius-fahrenheit/100 to convert 100 Celsius in Fahrenheit or http://localhost:8080/temperature-converter/fahrenheit-celsius/-20 to convert -20 Fahrenheit to Celsius.

- Using the command line tool (https://curl.haxx.se/). For instance

  `curl http://localhost:8080/temperature-converter/fahrenheit-celsius/32`

- Using a graphical tool such as Insomnia or Postman. Insomnia is shown below.

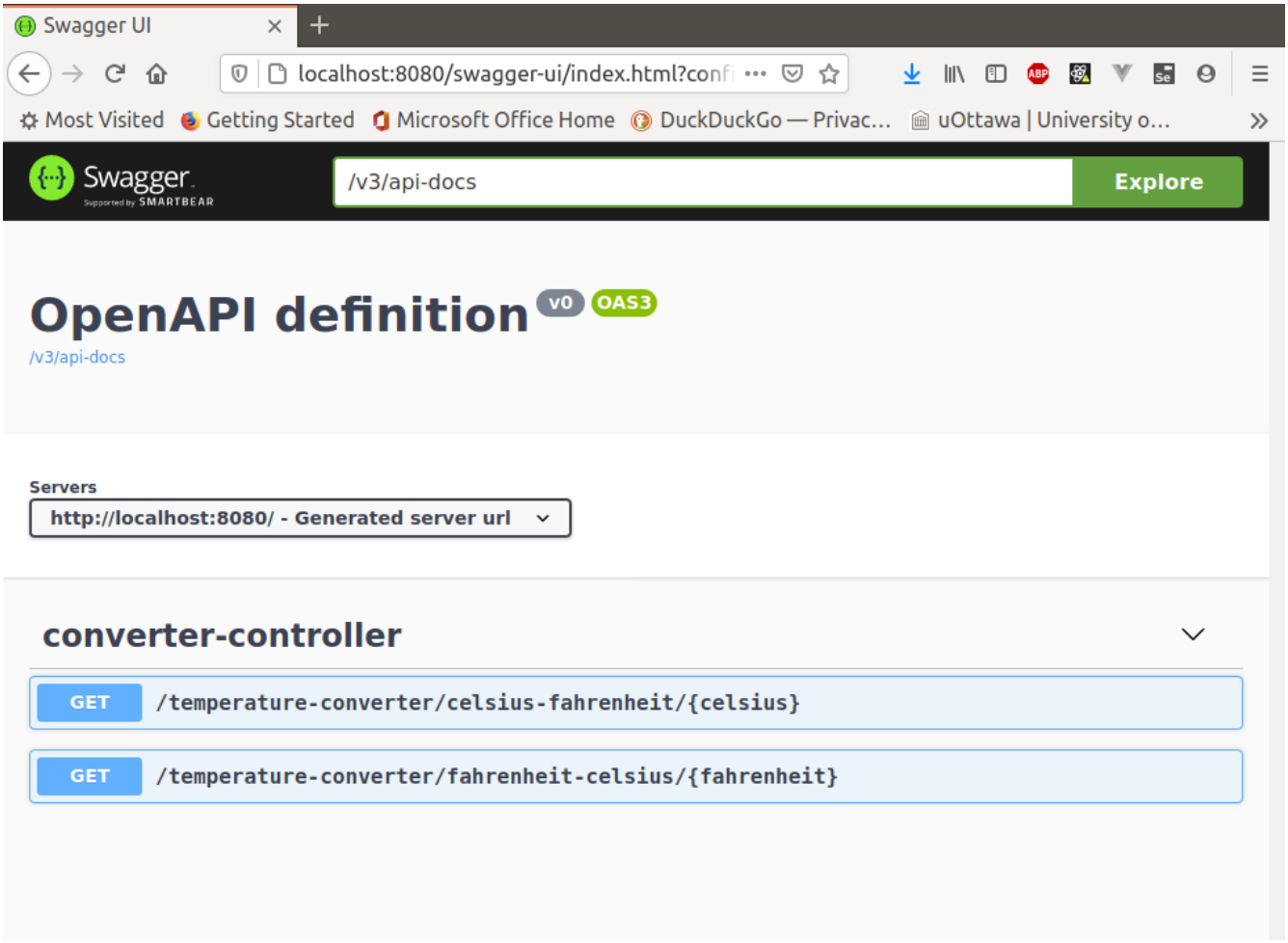


OpenAPI Documentation

After a run of the application, we can access springdoc-openapi generated documentation in JSON format at URL `http://localhost:8080/v3/api-docs/`.

Documentation in YAML format is generated at URL http://localhost:8080/v3/api-docs.yaml.

We can also access the documentation with Swagger UI at http://localhost:8080/swagger-ui.html.



Exercise

The following exercise is the deliverable for the lab. Complete and check-in the code to Github Classroom before the deadline. Only this exercise will be evaluated.

Develop a Springboot REST Calculator Web Service. The service should provide the following operations:

- GET calculator/add/{number1}/{number2} for addition
- GET calculator/subtract/{number1}/{number2} for subtraction
- GET calculator/multiply/{number1}/{number2} for multiplication
- GET calculator/divide/{number1}/{number2} for division