SEG3502 – Lab 6
Services Web REST

L'objectif de ce laboratoire est de présenter la programmation des services Web RESTful avec Angular et Springboot. Nous allons (1) développer une application Angular pour consommer un service Web disponible publiquement et (2) créer une API de service Web RESTful à l'aide de Springboot.

Le code source de l'application Angular est disponible à https://github.com/stephanesome/weather-app.git et le code source de l'application Springboot à https://github.com/stephanesome/converter-api.git.

1. Weather Application (météo)

Nous allons développer une application Angular pour afficher les conditions météorologiques actuelles d'une ville. L'application présente un formulaire dans lequel l'utilisateur peut spécifier la ville d'intérêt et éventuellement sélectionner un pays où se trouve cette ville. Lors de la soumission, l'application affiche les conditions météorologiques actuelles de cette ville.

Nous utiliserons une API de service Web RESTful fournie par Open Weather Map (`http://openweathermap.org/`) pour obtenir les conditions météorologiques courantes.

Clé d'API

Vous avez besoin d'une clé API d'OpenWeatherMap que vous pouvez obtenir en vous abonnant à un compte gratuit. Accédez à `http://openweathermap.org/api`, sélectionnez *Subscribe for Current Weather Data* et suivez les instructions.

Configuration du projet

Générez une application angular (`ng new weather-application`). Le routage n'est pas nécessaire et nous utiliserons CSS comme format de feuille de style.

Ajoutez bootstrap au projet (`ng add @ng-bootstrap/ng-bootstrap`).

Nous utiliserons également un composant de Angular Material (`https://material.angular.io/`) pour sélectionner les pays.

- Ajoutez Angular Material avec la commande `ng add @angular/material`. Sélectionnez les choix par défaut.
- Ajoutez l'extension `select-country` au projet (`https://www.npmjs.com/package/@angular-material-extensions/select-country`)
    - Installez les icônes de drapeau de pays avec `npm i svg-country-flags -s`

- Mettre à jour le fichier de configuration `angular.json` en ajoutant

```
{
...
"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"
          }
        ],
....
```

- Installez l'extension angular material avec la commande
  `npm install --save @angular-material-extensions/select-country`

- Importez le MatSelectCountryModule, le HttpClientModule ainsi que ReactiveFormsModule dans le AppModule. Modifiez `src/app/app.module.ts` et ajoutez ce qui suit

```
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 { }
```

Modèle Weather

Nous créons une classe de modèle pour représenter les informations météorologiques retournées par l'appel d'API. Exécutez la commande `ng generate class model/weather --type=model` pour générer une classe de modèle. Modifiez `src/app/service/open-weather.service.ts` comme suit

```
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. }
```

La classe `Weather` définit des champs pour les informations météo qui nous intéressent. Le constructeur crée une instance à partir des informations disponibles.

Service d'accès à l'API Weather

Nous utilisons un service pour interagir avec l'API OpenWeather. Cela permet au reste du code d'être extrait des détails de l'interaction. Cela pourrait également faciliter la substitution d'un autre service pour les informations météo.

Générez un service `ng generate service service/open-weather`.
Éditez `src/app/service/open-weather.service.ts` comme suit.

```
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. }
```

Le service utilise HTTPClient (injecté à la ligne 14) pour émettre des requêtes vers l'API. Assurez-vous de spécifier votre Token API Open Weather à la ligne 7. La fonction getWeatherAtCity construit la requête en définissant les paramètres (ligne 17) avec la ville et le pays ainsi que et le jeton API, et en définissant le type de réponse attendu (ligne 18). La requête est envoyée en tant que méthode HTTP GET.

Les requêtes HTTP sont traitées de manière asynchrone. Un appel HTTPClient retourne un Observable auquel le requereur s'abonne. Nous utilisons l'opérateur pipe pour transmettre toute erreur retournée à la fonction handleError (lignes 24-27). La fonction assure simplement que nous avons un traitement d'erreur standard en retournant un Observable qui émet une notification d'erreur.

Composant App

Le composant App est le seul composant utilisé dans cette application. Il présente un formulaire pour obtenir la ville et le pays d'intérêt d'un utilisateur, utilise le service Weather pour obtenir des données météorologiques et les affiche à l'utilisateur.

Éditez la classe de composant App (`src/app/app.component.ts`) comme suit.

```
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. }
```

La fonction parseResponse (lignes 8 à 18) prend un objet JSON, extrait les informations pertinentes et crée un objet de modèle Weather à partir de ces informations. La classe définit un ReactiveForm à utiliser par le modèle HTML et gère la soumission du formulaire. Les réponses de l'observable du service Weather (injecté à la ligne 34) sont analysées et mises à la disposition du modèle HTML (lignes 49 à 52) tandis que les erreurs retournées définissent un message d'erreur à afficher (ligne 53).

Éditez le modèle HTML du composant 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>
```

Nous pouvons voir la composante de sélection de pays (select-country) utilisée aux lignes 7-9.

Éditez le fichier de style CSS du composant App comme suit.

```
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;
```

```
    font-weight: bold;
}

#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. Service de convertisseur de température (Temperature Converter)

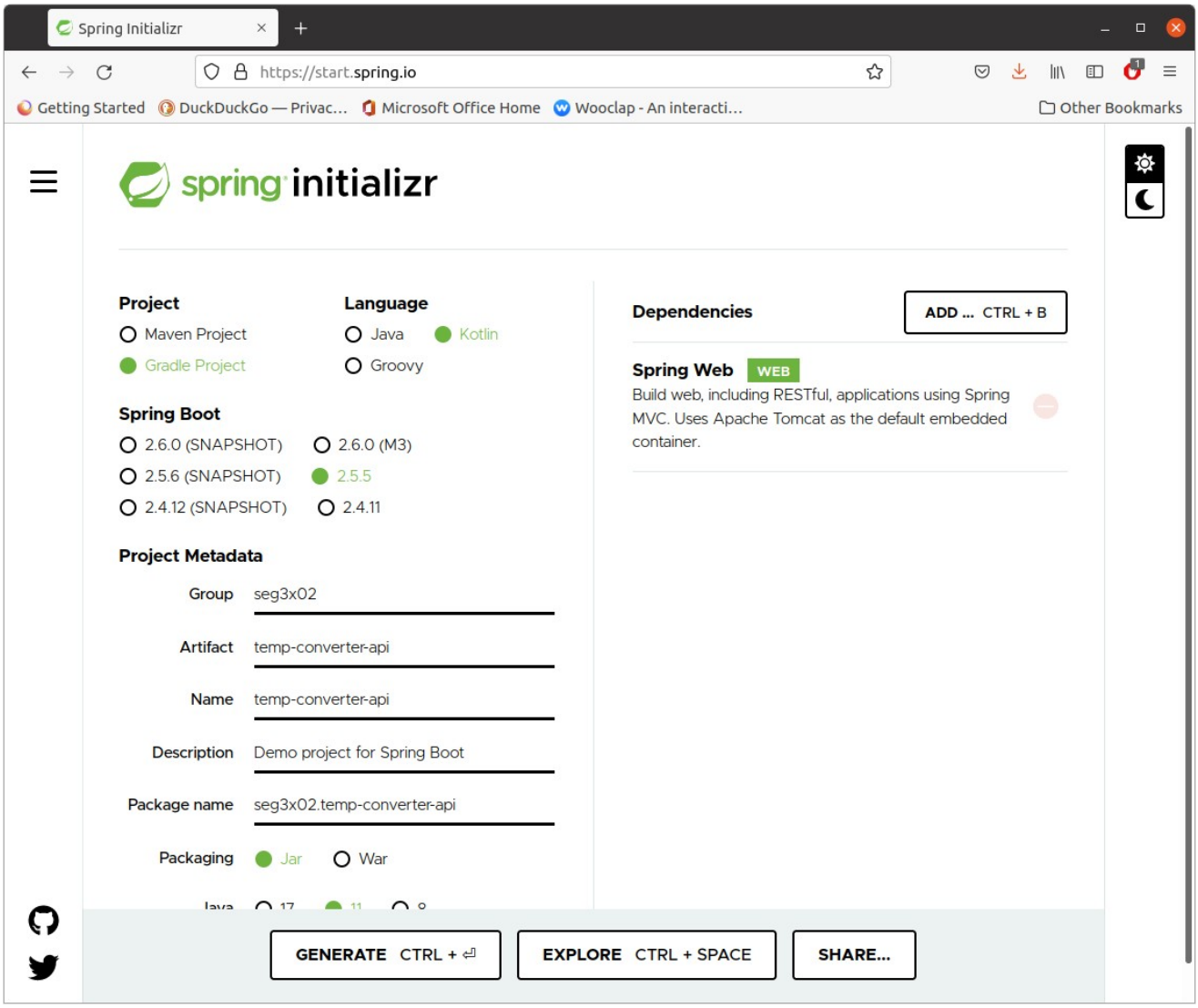
Nous implémentons un service Web RESTful pour la conversion de température avec Springboot. Le service expose les opérations suivantes:

- ***GET temperature-converter/celsius-fahrenheit/{celsius}*** pour retourner la valeur en Fahrenheit de la valeur *celsius*, et
- ***GET temperature-converter/fahrenheit-celsius/{fahrenheit}*** pour retourner la valeur en Celsius de la valeur *fahrenheit.*

Setup du Projet

Créer un projet avec ***Spring Intializr***.

- Sélectionnez *Gradle Project* comme Project type et *Kotlin* comme Language
- Laissez la version Spring Boot sélectionnée par défaut
- Entrez les métadonnées du projet (Group: *seg3x02*, Artifact: *temp-converter-api*)
- Ajoutez la dépendance *Spring Web*.



Générez, téléchargez, décompressez et ouvrez le projet généré dans votre IDE.

OpenAPI

Nous ajoutons une dépendance à la bibliothèque springdoc-openapi, pour générer automatiquement de la documentation de la spécification OpenAPI 3 pour l'API.

Éditez le fichier build.gradle.kts (dans le dossier racine du projet) et ajouter une dépendance org.springdoc:springdoc-openapi-ui:1.5.11. La déclaration des dépendances doit ressembler à ceci:

```
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.  }
```

La dépendance ajoutée est en ligne 3. Elle configurera la génération automatisée de la documentation OpenAPI basée sur le code.

Contrôleur REST

Créez une classe Kotlin ConverterController dans le package seg3x02.tempconverterapi.controller et modifiez-la comme suit.

```
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. }
```

La classe est annotée avec @RestController pour l'enregistrer comme contrôleur pour les requêtes REST. L'annotation @RequestMapping au niveau de la classe spécifie *temperature-converter* comme URI racine pour tous les points de terminaison de service de la classe. Nous définissons deux points de terminaison correspondant aux opérations de service. Chacun est défini comme une fonction annotée avec @GetMapping, une annotation de mappage de requêtes pour la méthode HTTP GET.

Advice de Contrôleur REST

Créez une classe Kotlin ControllerExceptionHandler dans le package seg3x02.tempconverterapi.controller et modifiez-la comme suit.

```
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. }
```

L'annotation @RestControllerAdvice enregistre la classe comme Controller Advice pour les contrôleurs REST. Nous définissons une fonction de gestion des exceptions (qui gère toute exception ici). La fonction retourne une entité de réponse avec le code d'état BAD_REQUEST (400) et un message d'erreur dans le corps.

Building and Running

Exécutez la commande `./gradlew bootRun` à partir du dossier principal du projet pour générer et exécuter le projet. Vous pouvez également utiliser la commande run de votre IDE.

Une fois l'application démarrée (notez que pour une raison quelconque, l'indicateur de progression avec ./gradlew bootRun reste autour de 80%), vous pouvez vérifier l'API de différentes manières:

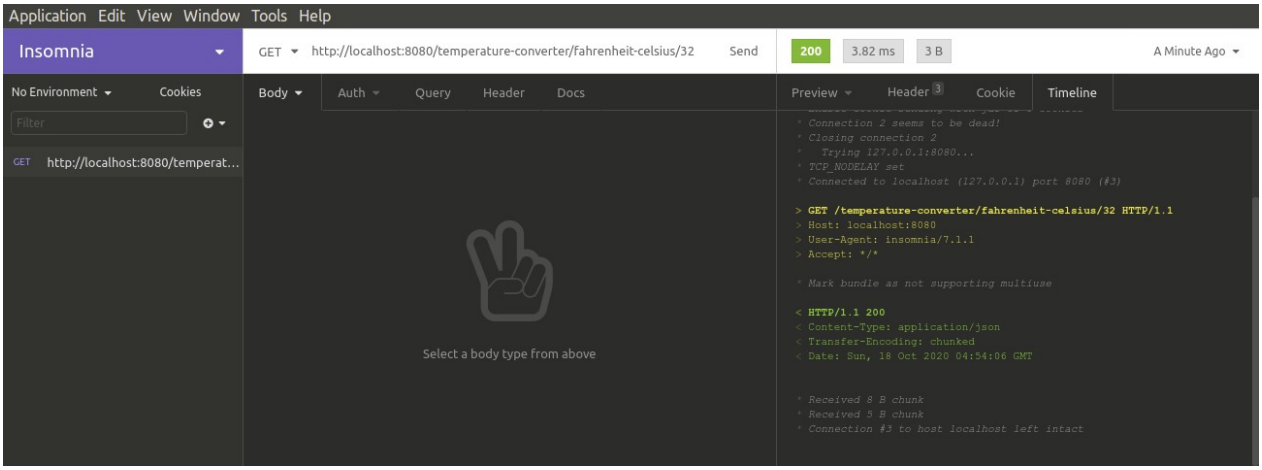
- Avec un navigateur internet, en naviguant à des URL tels que

  http://localhost:8080/temperature-converter/celsius-fahrenheit/100 pour convertir 100 Celsius en Fahrenheit ou http://localhost:8080/temperature-converter/fahrenheit-celsius/-20 pour convertir -20 Fahrenheit en Celsius.

- En utilisant l'outil de ligne de commande curl (https://curl.haxx.se/). Par exemple

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

- À l'aide d'un outil graphique tel que Insomnia or Postman. Insomnia est illustrée ci-dessous.

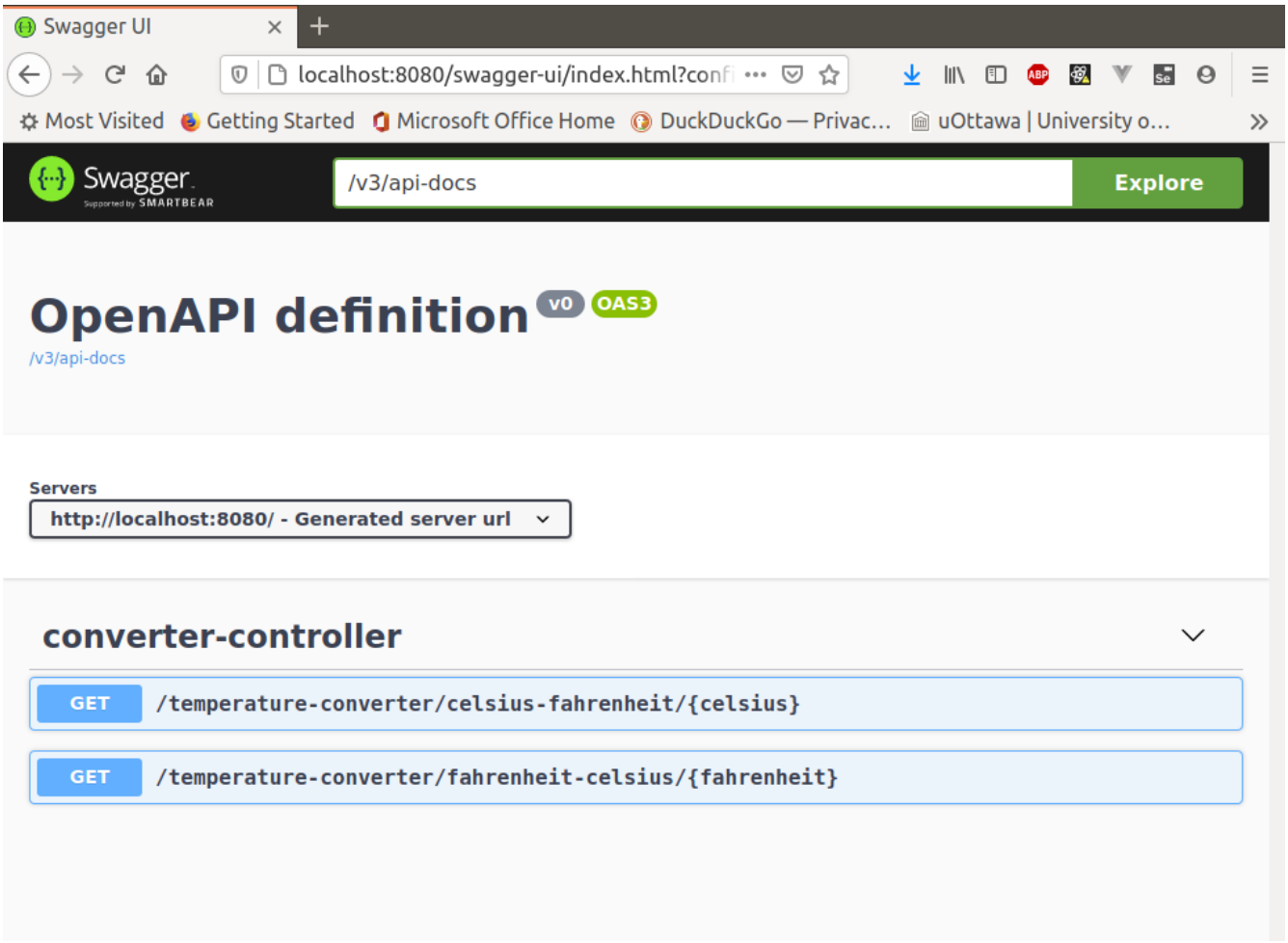


Documentation OpenAPI

Après une exécution de l'application, nous pouvons accéder à la documentation générée par springdoc-openapi dans le format JSON à l'adresse URL http://localhost:8080/v3/api-docs/ .

La documentation dans le format YAML est générée à l'adresse URL `http://localhost:8080/v3/api-docs.yaml`.

Nous pouvons également accéder à la documentation avec Swagger UI à `http://localhost:8080/swagger-ui.html`.



Exercise

L'exercice suivant est le livrable pour le laboratoire. Complétez et enregistrez le code dans Github Classroom avant la date limite. Seul le travail de ce exercice sera évalué.

Développez un service Web Springboot REST Calculator. Le service doit fournir les opérations suivantes:

- GET calculator/add/{nombre1}/{nombre2} pour l'addition
- GET calculator/subtract/{nombre1}/{nombre2} pour la soustraction
- GET calculator/multiply/{nombre1}/{nombre2} pour la multiplication
- GET calculator/divide/{nombre1/{nombre2} pour la division