bootiful tests avec Springboot, Serenity, Travis CI et SonarQube

Jauffrey Flach

2020-2021

A rendre

Ce TP se compose de 3 parties. Il faudra rendre uniquement la 3ème partie, mais vous aurez besoin des 2 autres parties pour faire fonctionner la troisième. Faites bien attention d'avoir des éléments fonctionnels au début de chaque séance!

Pré-requis

- un compte Github
- Internet
- JDK 8+
- Maven 3+
- NodeJS et Angular CLI
- Git
- Un IDE (STS ou IntelliJ IDEA)

Partie 1: Construction des API avec Spring boot

Nous allons tout d'abord construire un simple projet avec spring boot, et écrire une application fort utile (pour le TP): Un additioneur REST.

L'objectif de ce projet est de pouvoir récupérer un nombre au hasard, et d'y ajouter un autre nombre en utilisant les API REST. Nous y ajouterons par la suite un front Angular 7.

Construction du projet

- Initialisez votre projet avec **http://start.spring.io** en utilisant les dépendances:
  - Web

Structure des packages

Spring utilise la notion de services pour effectuer des opérations sur les POJO (entités). Lorsque vous concevez des packages, il est important de bien structurer vos packages et de mettre les services dans un package à part, et les classes qui utilisent ces services dans d'autres (les controlleurs ou le modèle par exemple). Pour illustrer, voici un exemple de service d'addition:

```java
@Service
public class AdderService {

    private int num;

    public void baseNum(int base) {
        this.num = base;
    }

    public int currentBase() {
        return num;
    }

    public int add(int adder) {
        return this.num + adder;
    }

    public int accumulate(int adder) {
        return this.num += adder;
    }
}
```

Cest un service très simple, et il utilise une annotation @Service. C'est cette annotation qui va permettre à Spring de:

- Reconnaître que lors du parcours des classes, cette classe fait partie du contexte Spring
- Être capable de l'injecter dans d'autres classes

Voici par exemple une classe Controller, qui va utiliser le service d'addition et exposer l'API REST:

```java
@RequestMapping(value = "/adder", produces = MediaType.APPLICATION_JSON_UTF8_VALUE)
@RestController
public class AdderController {

    private AdderService adderService;

    public AdderController(AdderService adderService) {
        this.adderService = adderService;
    }

    @GetMapping("/current")
    public int currentNum() {
        return adderService.currentBase();
    }

    @PostMapping
    public int add(@RequestParam int num) {
        return adderService.add(num);
    }
}
```

Vous noterez que le constructeur prend en paramètre une implémentation d'AdderService: elle est injectée par Spring depuis le contexte. Le reste des annotations est toutefois classique, il décrit le point d'entrée du service (/adder) et les opérations GET et POST avec leur mapping. La sortie de l'appel fournit par défaut du JSON en UTF-8.

A vous de jouer! Créez un API qui a les fonctionnalités similaires à l'exemple ci-dessus. Vous écrirez une classe de test avec JUnit, qui teste tous les controlleurs REST.

Vous pouvez utiliser RESTAssured, ou n'importe quel framework de test d'API.