

TP 1 - Spring Boot et REST

- Spring BOOT
- Spring REST
- Spring HATEOAS
- Swagger
- Spring REST Docs

Estimated Time : **60** *minutes*

1.0 - Objectifs

Le présent TP se veut présenter comment mettre en place les technologies à partir de Spring qui constituent les briques de bases d'un microservice à savoir :

- Spring Boot
- REST
- HATEOAS

Pour chacun des aspects nous montrerons comment :

- Tester
- Documenter
- Mettre en place les bonnes pratiques

1.1 Enoncé court

1.1.1 - Mise en place d'une application avec Spring Boot

Dans `com.zenika.microservices.resanet.starter`, il y a :

- `Application.java` : Une classe pour lancer l'application
- `ApplicationConfiguration.java` : Une classe à part de configuration qui servira à mettre en place la configuration de la documentation

1.2 Mise en place de l'application

TODO 01 :

Mettre en place l'application avec `Spring Boot` . Pour cela, il faut rajouter l'annotation

```
@SpringBootApplication
```

sur la classe qui contient le main à savoir `Application.java`.

Comme l'arborescence n'est pas au même niveau que le starter, il faut aussi penser à inclure les packages suivants dans le `component-scan` :

- `com.zenika.microservices.resanet.controller`
- `com.zenika.microservices.resanet.services`

1.3 Référencement des entités annotées

TODO 02 :

Voir `com.zenika.microservices.resanet.starter.Application.java`

Pour donner les entités du domaine, il faut préciser où elles se situent avec l'annotation de `spring-data-jpa` :

```
@EntityScan
```

Les entités du domaine sont dans le package :
`com.zenika.microservices.resanet.domain`

1.4 Ajout spring-data JPA

TODO 03 :

Il faut lui demander de considérer le package **`com.zenika.microservices.resanet.repository`** comme celui qui contient les classes de repository spring-data.

Cela se fait avec l'annotation :

```
@EnableJpaRepositories
```

1.5 Configuration de l'application

La configuration de l'application est dans `src/main/resources/application.properties`, elle contient notamment :

- le port d'écoute : **server.port**
- la configuration de la datasource JPA : `spring.datasource.*`
- le profil de l'application ici : **dev**
- la configuration automatique du schéma en jpa : **spring.jpa.hibernate.ddl-auto**
- ...

Un jeu de données dans `import.sql` et la configuration des logs dans `logback.xml` complètent la configuration.

1.6 Lancer l'application

TODO 04 :

Dans le projet faire Run As > Maven Build

Dans l'invitation faire : **spring-boot:run**

Puis ouvrir un navigateur et demander : `http://localhost:8000/resanet/application/`

Le navigateur doit afficher **0.1**

1.7 Mettre en place le controller

Gestion des Villes

Mettre en place le controller REST

TODO 05 :

Dans **com.zenika.microservices.resanet.resanet.controller.VilleController.java**

```
//Mettre ici l'annotation pour un controller REST
//Mettre ici l'annotation pour mapper l'uri /villes
public class VilleController {

}
```

Tips : Pour gérer la root URI à savoir villes on peut gérer le mapping sur tout le controller.

TODO 06 :

Dans **com.zenika.microservices.resanet.resanet.controller.VilleController.java**

Annoter la méthode **getList()** pour mapper les URL avec la méthode GET pour obtenir la

liste des villes.

TODO 07 :

Dans **com.zenika.microservices.resanet.resanet.controller.VilleController.java**

Annoter la méthode pour obtenir depuis l'URI **/villes/{villeId}** le détail sur une ville.

```
public Ville readVille(Long villeId) {  
    LOGGER.info("demande de la ville : "+villeId);  
    return villeRepository.findOne(villeId);  
}
```

Mettre en place une politique de tests d'intégration

Avec le **RestTemplate**, tester ce qui vient d'être mis en place.

//TODO mettre un schéma pour expliquer

TODO 08

Mettre en place les tests pour une application Spring-Boot dans **/src/test/java** package **com.zenika.microservices.resanet** et classe **VilleTest**

Pour cela mettre en place un **runner** dédié à Spring dans JUnit et initialiser un **ApplicationContext** de test.

TODO 09

Instancier un **RestTemplate** sur l'initialisation des tests.

TODO 10

Utiliser le **RestTemplate** pour récupérer la liste des villes et vérifier qu'il y ait bien :

- *Paris*
- *Lyon*
- *Marseille*
- *London*

et que la taille de la liste est de 4.

TODO 11

Avec le **RestTemplate** demander la ville avec l'id **1**.

Vérifier que :

- ce soit *"Paris"*
- que le pays associé soit la *"France"*

1.8 Créer des villes

Dans un premier temps, la ville sera créée 'naïvement' en retournant le contenu de la ressource créée et la création sera testée.

Dans un second temps, la ville sera créée en retournant l'URI de la ressource.

TODO 12 :

Dans **VilleController** faire une méthode pour mapper POST sur l'uri **`"/villes/"`**. La méthode doit retourner :

- le code HTTP 201
- le retour doit être l'instance de la ville créée en *JSON* ou *XML* selon le **"Content-Type"** précisé dans le header **"Accept"**.

TODO 13 :

Dans **VilleTest** utiliser le **RestTemplate** pour tester la création d'une ville.

TODO 14 :

Dans le controller changer le return type pour pouvoir faire un `postForLocation` par la suite. Pour cela, utiliser les **HTTPEntity**.

Le retour doit respecter les éléments suivants :

- le code HTTP 201
- l'uri de la ville créée doit être du type `"/villes/{villeId}"`
- plusieurs appels successifs doivent incrémenter l'identifiant `"{villeId}"`

TODO 15 :

Dans la classe de test **VilleTest.testCreateVille** changer la méthode du **RestTemplate** en `postForLocation` puis demander la ressource associée à l'URI dans un 2e temps et conserver les assertions sur le contenu de la ville créée.

1.9 Mettre à jour une ville

TODO 16 :

Dans le controller annoter avec la méthode adéquate (celle pour la mise à jour)

TODO 17 :

Dans le testUnitaire appeler la méthode depuis le restTemplate

1.14 Documentation avec Swagger (SpringFox)

1.2 Aide

1.2.1 - Mise en place d'une application avec Spring Boot

1.2.1.1 - L'annotation @SpringBootApplication

Si l'on regarde de plus près, l'annotation @SpringBootApplication en regroupe en fait plusieurs :

```
@Documented
@Inherited
@Configuration
@EnableAutoConfiguration
@ComponentScan
```

En détails :

- @Documented :

Norme de java pour indiquer que l'application fait partie des classe documentées avec les outils habituels et donc être considérée comme faisant partie de l'API publique du framework.

- @Inherited :

Norme java pour indiquer que le comportement s'applique aux sous-classe de la dite annotation.

- @Configuration :
- @EnableAutoConfiguration :
- @ComponentScan :

1.2.1.2 - Les paramètres de l'annotation

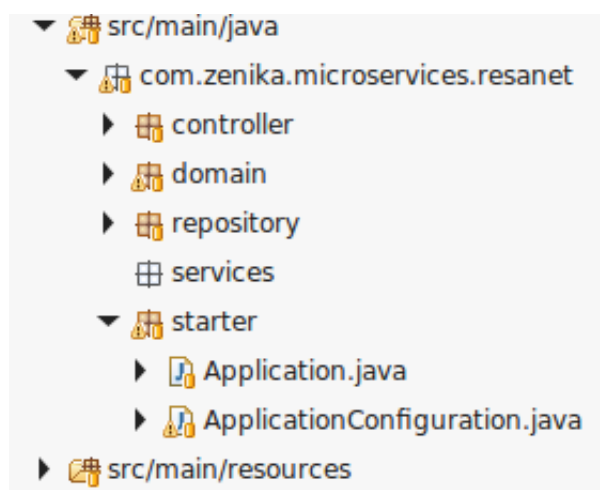
- exclude() / default {}

- `excludeName()` / default `{}`
- `String[] scanBasePackages()` default `{}` -> `@AliasFor(annotation = ComponentScan.class, attribute = "basePackages")`

1.2.1.3 - L'exercice

Le démarrage et la configuration de base se font dans la classe **`com.zenika.microservices.resanet.starter.Application`**

```
//Mettre les annotations spring ici dont @SpringBootApplication
public class Application {
    public static void main(final String[] args) {
        //Classe de démarrage d'une application Spring Boot
        SpringApplication.run(Application.class, args);
    }
}
```













Il faudra ainsi demander à spring de "scanner", c'est à dire ajouter à l'`ApplicationContext`, les classes annotées des packages suivants :

- `com.zenika.microservices.resanet.controller`
- `com.zenika.microservices.resanet.services`

Pour ce faire, il faut préciser la liste ci-dessus dans l'attribut **`scanBasePackages`** de l'annotation **`@SpringBootApplication`** comme suit :

```
@SpringBootApplication(scanBasePackages={"com.zenika.microservices.resanet.controller",
                                         "com.zenika.microservices.resanet.services"})
```

Pour ne pas avoir à spécifier les éléments à scanner, il faudrait que ces derniers soient dans des sous-packages un peu comme ci-dessous.

- ▼  > src/main/java
 - ▼  > com.zenika.microservices.resanet
 - ▶  > controller
 - ▶  domain
 - ▶  repository
 -  services
 - ▶  Application.java
 - ▶  ApplicationConfiguration.java
 - ▶  src/test/java
 - ▶  src/main/resources

