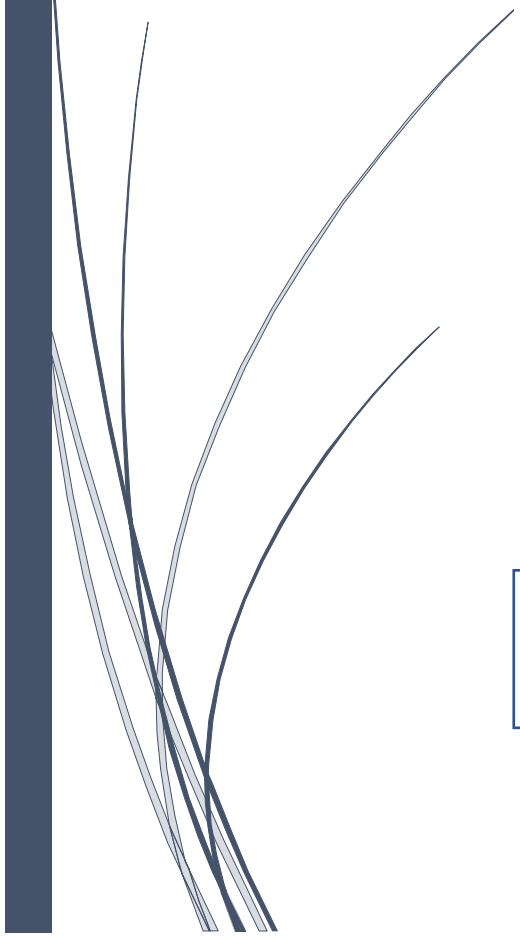
Rapport
Project Spring Boot

JAVA Q5

Ahmad
Abu Nassar



HELB Haute Ecole Libre de Bruxelles

Table des matières

1. Introduction

Dans le cadre du cours de Java durant le quadrimestre 5, il nous à été demandé de développer deux api Web multicouche avec Spring boot. Avec cela, s'accompagne évidemment un rapport/documentation des différentes fonctionnalité ainsi que l'utilisation de celle-ci. C'est pour cela que dans les points qui suivront j'aborderai les différents points nécessaires à la compréhension du projet.

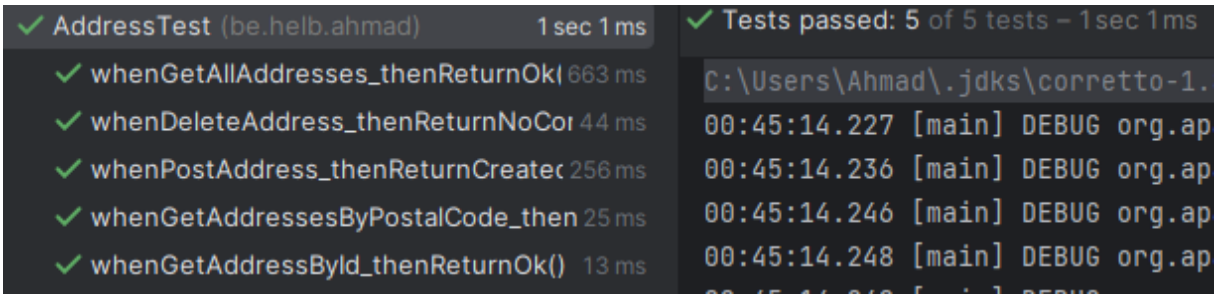
2. Description de l'application

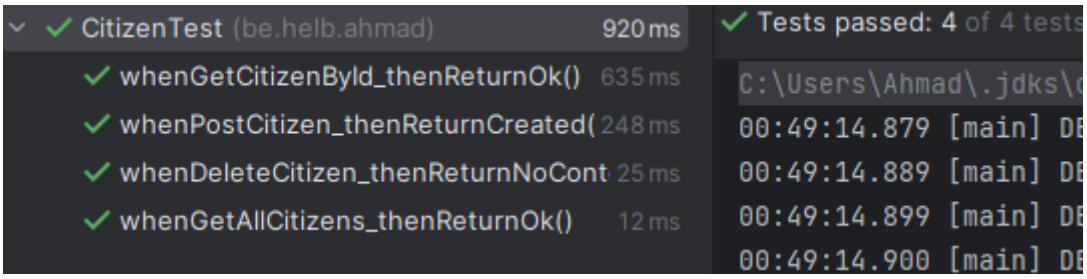
Tout d'abord, tel que rapporté durant l'introduction, il s'agit-là d'une application Spring boot, qu'est ce que c'est Spring Boot ? il s'agit d'un Framework Java qui simplifie le développement d'applications par des conventions et certaines configurations qui vont être prête à l'emploi, de plus elle intègre un serveur web embarquer et est adapté au déploiement de micro-service.

Ensuite, que propose l'application que j'ai développé ? il s'agit-là de deux apis pour un système dit « Communale ». La première API va permettre de POST, GET et DELETE des données concernant des citoyens, c'est-à-dire, l'identité, l'adresse, le ou les documents leur appartenant et les différents évènements auquel l'individu a pu participer. Mais ce n'est pas tout, effectivement la première api permet aussi de récupérer des données de la deuxième api, mais quels sont ces données ? La deuxième api ne s'occupe que

de la composition ménage d'individu, donc depuis la première api on peut GET toutes les compositions ménages existantes dans la base de données. Et comme vous l'auriez compris, la deuxième api permet de GET, POST et DELETE des informations de composition ménage.

Enfin, j'y ai intégré des tests en « RestAssured » qui est une library Java qui simplifie le test d'api RESTful qui se sont avéré tous concluent.





3. Architecture

PREMIERE API

Toutes les méthodes du packages Controller retourne une ResponseEntity

Package Controller

AddressController
- getAllAddresses()
- getAddressById()
- saveAddress()
- getAddressByPostalCode
- deleteAddress

CitizenController
- getAllCitizens()
- getCitizenById()
- saveCitizen()
- deleteCitizen

DocumentController
- getAllDocuments()
- getDocumentById()
- saveDocument()
- deleteDocument

EventController
- getAllEvents()
- getEventById()
- saveEvent()
- deleteEvent()

CommunicationSecondApiController
- getHouseHoldById()

Package Services

AddressService
- List <Address> getAllAddresses()
- Optional <Address> getAddressById()
- void saveAddress()
- List <Address> getAddressesByPostalCode
- void deleteAddress()

CitizenService
- List <Citizen> getAllCitizen()
- Optional <Citizen> getCitizenById()
- void saveCitizen()
- void deleteCitizen()

DocumentService
- List <Document> getAllDocument()
- Optional <Document> getCitizenById()
- void saveDocument()
- void deleteDocument()

EventService
- List <Event> getAllEvent()
- Optional <Event> getEventById()
- void saveEvent()
- void deleteEvent()

Package Models

Address
- Constructor
- @ManyToOne Address -> Citizen

Citizen
- Constructor
- @OneToMany Citizen -> Address
- @OneToMany Citizen -> Document
- @OneToMany Citizen -> Events

Document
- @JsonFormat Date ("yyyy-MM-dd")
- @ManyToOne Document -> Citizen

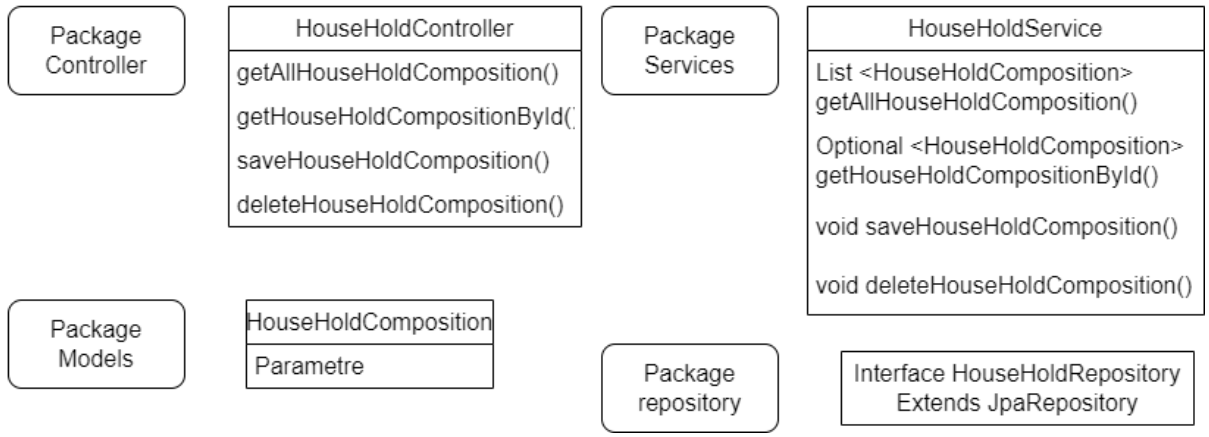
Event
- @JsonFormat Date ("yyyy-MM-dd")
- @ManyToOne Event -> Citizen

Package repository

- Interface AddressRepository Extends JpaRepository
- Interface CitizenRepository Extends JpaRepository
- Interface DocumentRepository Extends JpaRepository
- Interface EventRepository Extends JpaRepository

DEUXIEME API

Toutes les méthodes du packages
Controller retourne une ResponseEntity



4. Mise en marche et fonctionnalités

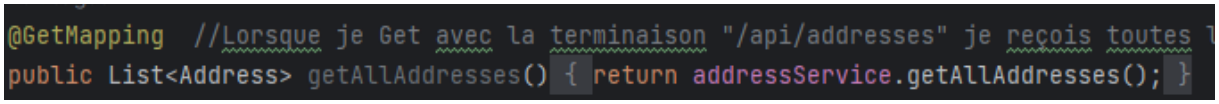
Pour la mise en marche de l'application, rien de plus simple ! Il suffit d'installer PostGreSql, une fois cela fait, mettre la base de données sur un serveur en spécifiant le port : « 5555 » et crée deux bases de données pour les deux API. La première qui s'appelle « commune » et la seconde qui s'appelle « citizen_information ». Le plus gros du travail est fait désormais, il manque plus qu'a créé un utilisateur en lui mettant tous les droits : ahmad et comme mot de passe : ahmad00. Maintenant spécifié que le owner du serveur est l'utilisateur ahmad et le tour est joué.

Ensuite, l'explication rapide des différentes fonctionnalités se retrouve déjà dans la description du projet, cependant des exemples de celles-ci est toujours la bienvenue. Tout d'abord, voyons comment récupéré les données des adresses stockés dans la base de données (nous allons prendre l'exemple des adresses car tous les modèles reposent sur le même principe).


```
[
  {
    "id": 1,
    "street": "Rue de la Liberté 1",
    "city": "Bruxelles",
    "postalCode": 1000,
    "citizen": {
      "id": 1,
      "firstName": "Alice",
      "lastName": "Doe",
      "email": "alice.doe@example.com"
    }
  },
```

```
  {
    "id": 3,
    "street": "Chaussée de Louvain 3",
    "city": "Schaerbeek",
    "postalCode": 1030,
    "citizen": {
      "id": 3,
      "firstName": "Charlie",
      "lastName": "Brown",
      "email": "charlie.brown@example.com"
    }
  },
  {
    "id": 4,
    "street": "Chaussée de Boitsfort 46",
    "city": "Bruxelles",
    "postalCode": 1050,
    "citizen": null
  }
]
```


Aussi simplement que cela, envoyer un GET à la terminaison spécifier dans le code va permettre de retourner les données voulues



Maintenant, imaginons qu'un utilisateur veut récupérer les adresses selon ses préférences de code postal tel que 1050, nous allons profiter de la méthode getAddressesByCodePostal qui possède une certaine terminaison pour arriver à l'objectif voulu

```java
@GetMapping("postalCode/{postalCode}")  //Get L'adresse mais de manière dynamique selon le co
public ResponseEntity<List<Address>> getAddressesByPostalCode(@PathVariable int postalCode) {
    List<Address> addresses = addressService.getAddressesByPostalCode(postalCode);

    if (!addresses.isEmpty()) {
        return new ResponseEntity<>(addresses, HttpStatus.OK);
    } else {
        return new ResponseEntity<>(HttpStatus.NOT_FOUND);
    }
}
```


```
[
    {
        "id": 4,
        "street": "Chaussée de Boitsfort 46",
        "city": "Bruxelles",
        "postalCode": 1050,
        "citizen": null
    }
]
```


Supposons que j'aimerai upload une nouvelle donnée dans la base de données, alors répétons le processus mais cette fois-ci avec une méthode POST :

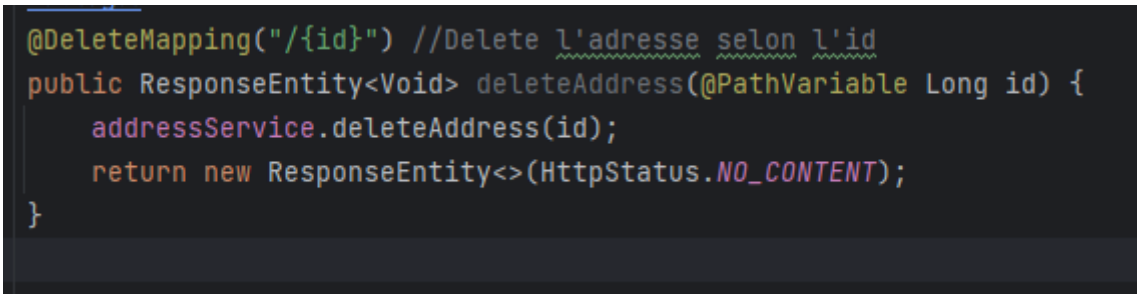
```java
@PostMapping   //Post l'adresse à l'endpoint "/api/addresses/"
public ResponseEntity<Void> saveAddress(@RequestBody Address address) {
    addressService.saveAddress(address);
    return new ResponseEntity<>(HttpStatus.CREATED);
}
```


```
{
    "street": "Boulevard Louise mettewie 7",
    "city": "Koekelberg",
    "postalCode": 1081,
    "citizen": {"id": 3}
}
```


Et maintenant, vérifions si les données se sont post avec une méthode GET :

```
[
  {
    "id": 5,
    "street": "Boulevard Louise mettewie 7",
    "city": "Koekelberg",
    "postalCode": 1081,
    "citizen": {
      "id": 3,
      "firstName": "Charlie",
      "lastName": "Brown",
      "email": "charlie.brown@example.com"
    }
  }
]
```


Enfin, essayons de supprimer le nouvel ajout grâce à la méthode DELETE :



```
@DeleteMapping("/{id}") //Delete l'adresse selon l'id
public ResponseEntity<Void> deleteAddress(@PathVariable Long id) {
    addressService.deleteAddress(id);
    return new ResponseEntity<>(HttpStatus.NO_CONTENT);
}
```

Vérifions de nouveau avec la même méthode GET :

Et comme on le voit, rien n'est retourné car il n'y a plus d'adresses avec comme code postal 1081.

Il reste tout de même la communication avec la seconde API et cela reste le même principe :

```java
@RequestMapping("/api/communication")
public class CommunicationSecondApiController {

    no usages
    @GetMapping(value = "/households", produces = MediaType.APPLICATION_JSON_VALUE)
    public ResponseEntity<String> getHouseHoldById() {
        OkHttpClient client = new OkHttpClient();

        // Utilisez une constante pour l'URL de l'API externe (évite les fautes de frappe)
        String externalApiUrl = "http://localhost:8181/api/households";

        Request request = new Request.Builder()
                .url(externalApiUrl)
                .build();

        try {
            Response response = client.newCall(request).execute();

            // Vérifiez si la réponse est réussie (code 2xx)
            if (response.isSuccessful()) {
                final String responseData = response.body().string();
                return new ResponseEntity<>(responseData, HttpStatus.OK);
            } else {
                // Gérez les codes d'erreur spécifiques si nécessaire
                return new ResponseEntity<>( body: "La demande à l'API externe a échoué", HttpStatus.INTERNAL_SERVER_ERROR);
            }
        } catch (Exception e) {
            // Loggez l'exception ou gérez-la selon les besoins
            e.printStackTrace();
            return new ResponseEntity<>( body: "Une erreur s'est produite lors de la communication avec l'API externe", HttpStatus.INTERNAL_SERVER_ERROR);
        }
    }
}
```

GET http://localhost:8081/api/communication/households Send


```json
[
    {
        "id": 1,
        "name": "Alice",
        "numberOfMembers": 15,
        "address": "Rue de la Liberté 1",
        "city": "Bruxelles",
        "postalCode": "1000"
    }
]
```