lnote

User-friendly note management

Spécifications techniques de l’application

Révision	Rédacteurs	Date	Objet
1	A. Mochizuki	01/02/2024	Création du document
2	A.Mochizuki	03/02/2024	Ajout du script de migration base de données
3	A.Mochizuki	05/02/2024	Modification du MCD
4	A.Mochizuki	21/02/2024	MAJ MCD + ajout vue couches

1 INTRODUCTION

1.1 Objet du document

Ce document a pour objectif de présenter l'essentiel des questions techniques liées à la mise en place de l'application Inote

Ce document présente :

- Le Modèle conceptuel de données
- Le modèle physique de données
- L'architecture de l'application
- Les outils choisis
- La stratégie de développement
- La stratégie de test

2 SOMMAIRE

Table des matières

Table des matières

3 ARCHITECTURE LOGICIELLE

3.1 Produits et versions

- **Langages**

Nom	Version	Commentaire
Java	17	Langage de programmation côté BackEnd
TypeScript 5.3	5.3	Langages de programmation côté FRONT-END pour l'implémentation du comportement de l'application
HTML (HyperText Markup Langage)	5	Langage de programmation côté FRONT-END pour l'implémentation de la structure des vues de l'interface
CSS (Cascading Style Sheets)	4	Langage de programmation côté FRONT-END pour l'implémentation de la présentation des vues de l'interface

- **Framework et dépendances côté backend**

Nom	Version	Commentaires
Spring Boot	3.2	Framework de gestion du projet, facilitateur pour l'utilisation de l'écosystème Spring.
Spring Security		Starter Spring Boot dédié à l'authentification et la gestion des accès
Spring Validation		Starter Spring Boot dédié à la validation des entrées utilisateur
Spring Web		Starter Spring Boot utilisé ici pour l'implémentation d'une API de type REST (permet d'embarquer également un serveur http embarqué)
Spring-boot-starter-test		Test de l'application. Inclus nottament JUnit(tests unitaires) et Mockito(doublures).
Lombock	1.18	Bibliothèque dédiée à la simplification de l'écriture des classes Java via annotations.
springdoc-openapi-starter-webmvc-ui	2.1	Interface WEB de présentation et de test de l'API
mysql-connector-java	8.0.33	Driver MySQL

H2	2.2	Génération de base de données de type H2 (in memory) pour le test
spring-boot-starter-data-redis		Starter pour la gestion de base de données NoSQL (Key-Value) Redis (utilisé pour la sécurité)

- **Framework et dépendances côté Front-end**

Nom	Version	Commentaires
Angular	12	Framework côté Front dédié à l'implémentation de l'interface utilisateur.
Ionic ? Flutter ? autre ?		Framework pour l'application mobile
BootStrap	5.2	Présentation de l'interface

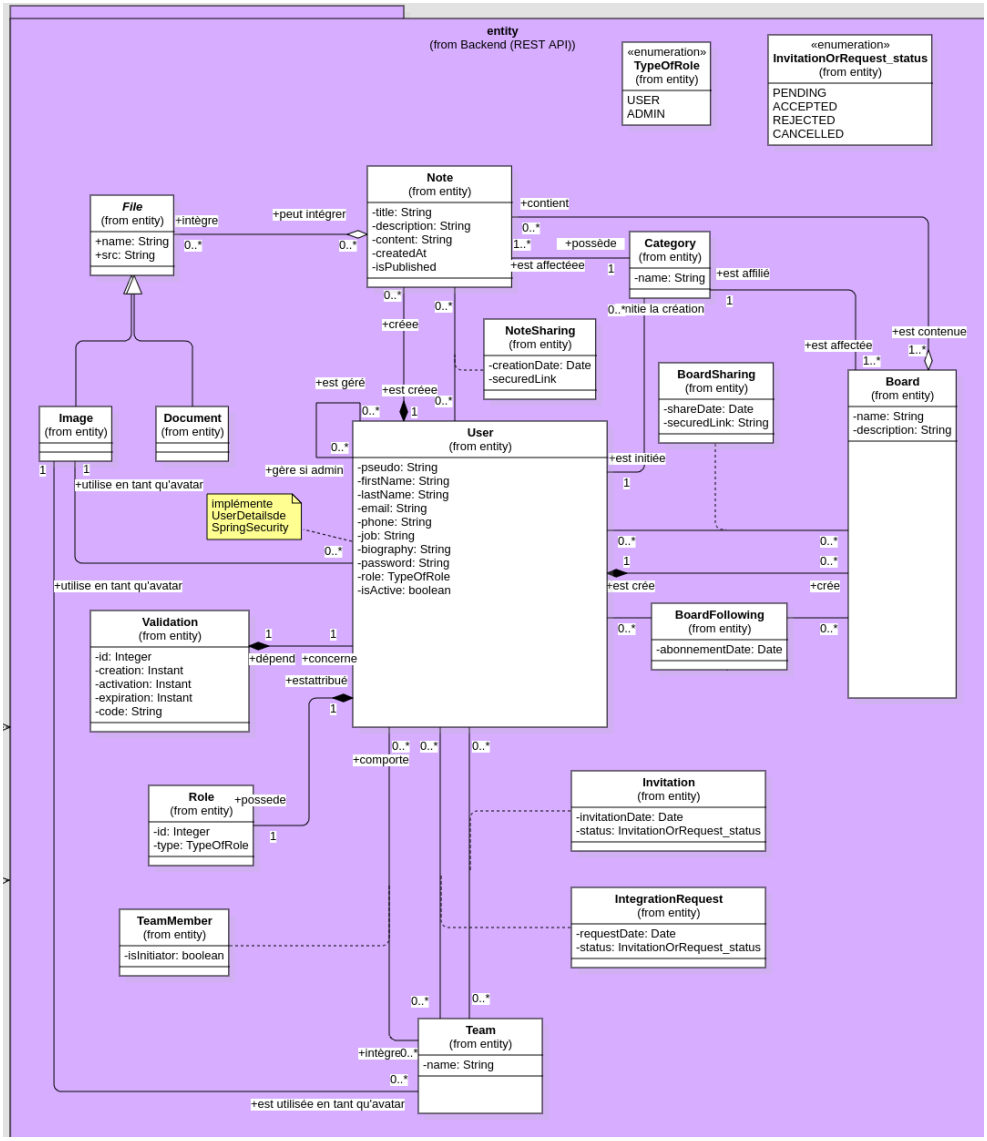
- **Serveur de base de données**

Nom	Version	Commentaires
MySQL	8.2	SGBDR

4 FOCUS TECHNIQUE

4.1 Organisation des données

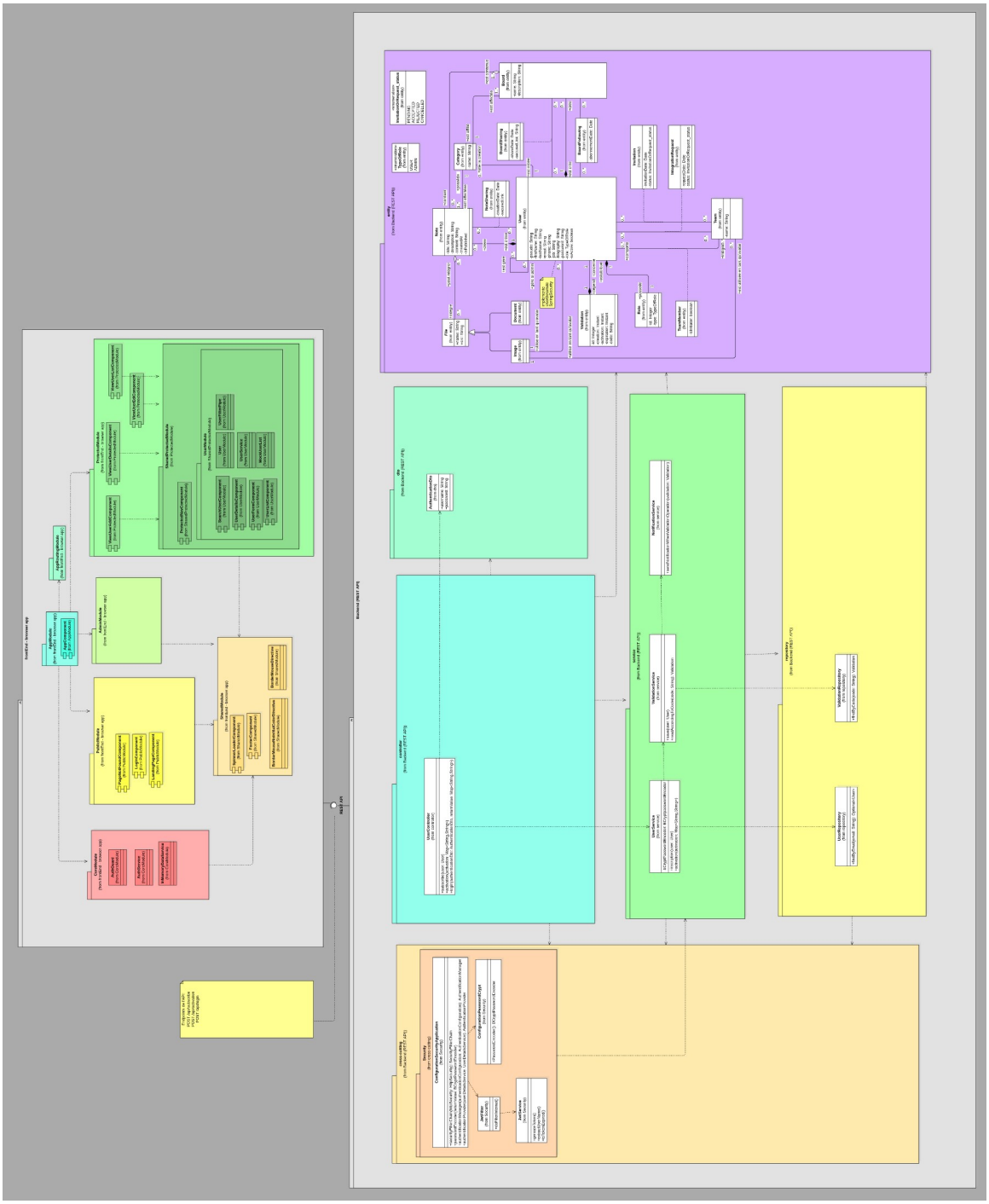
4.2 Diagramme de classes métier (Modèle Conceptuel de données)



4.3 Modèle Logique de données (mapping MCD)

4.4 Script SQL de création de la base de données

4.5 Architecture 3-Tiers de application (couches)



Règles de développement coté back

- Respect des conventions de nommage
- Présence de la Javadoc
- Formatage du code
- Les classes entités seront implémentés sous forme de POJO, selon la convention Javabean.

  Exemple :

- Les controllers ont interdiction d'envoyer les entités sur le réseau : toute donnée envoyée appartenant à une entité doit passer par un DTO si elle sort de l'API. Les DTO ne seront implémentés que par la couche controller.

- Le type record de java sera utilisé pour les dto.

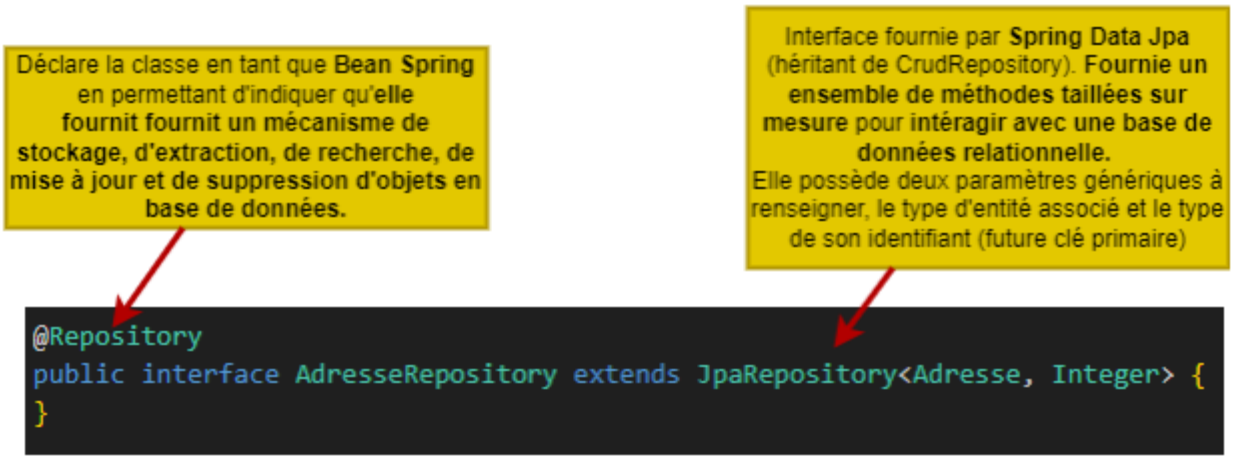
Exemple :

```
public record Person(
     Integer id,
     String lastName,
     String firstName,
     int age
) {
}
```

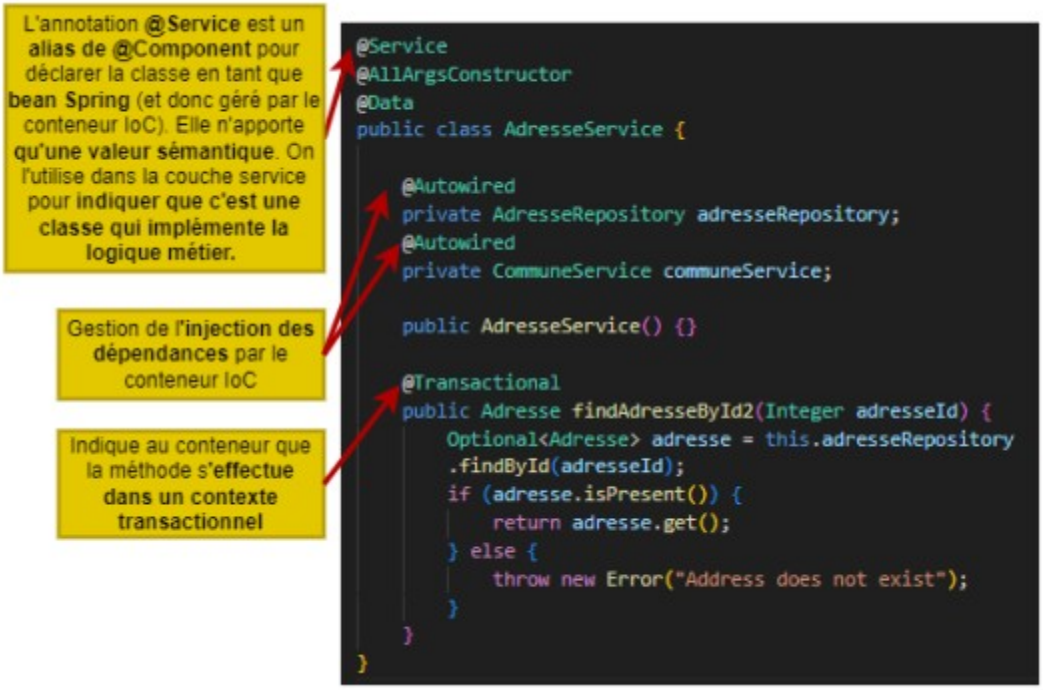
- Les DTO ne seront utilisés que par la couche controller, à l'intérieur de l'API seront manipulées les entités.

- Les dépendances seront injectées dans les classes et gérées par le conteneur IOC de Spring (utilisation de @Autowired)

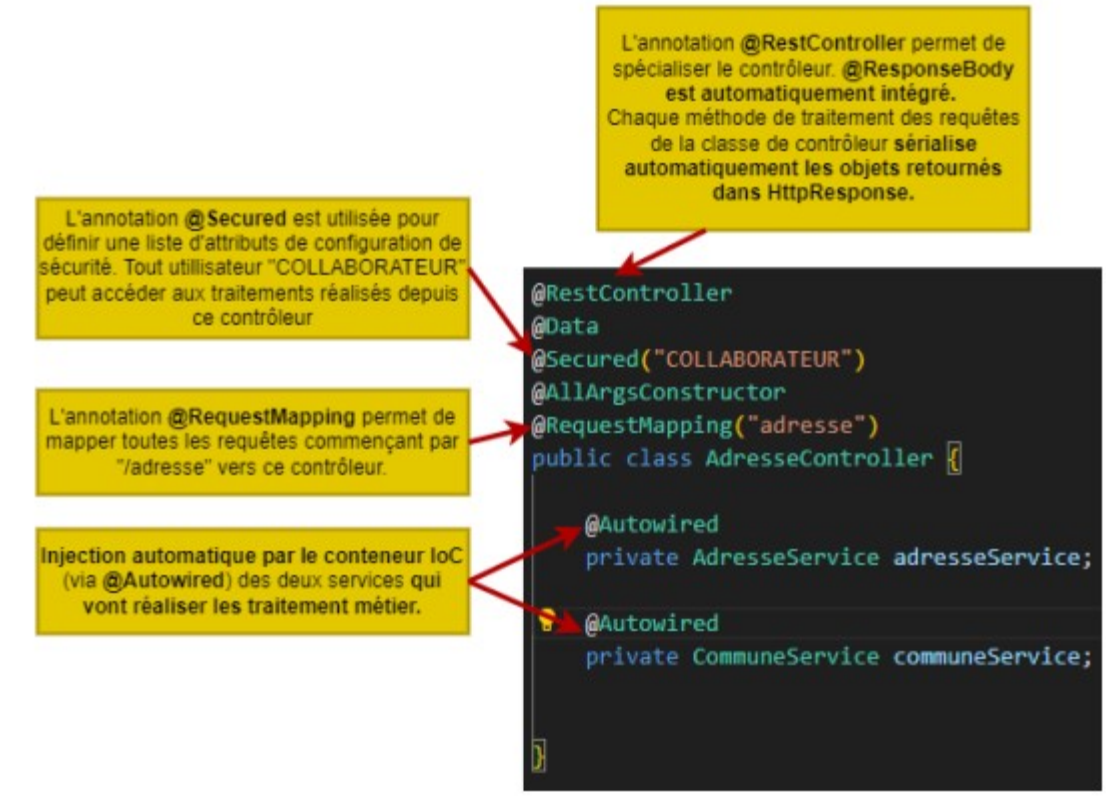
- Exemple de repository :

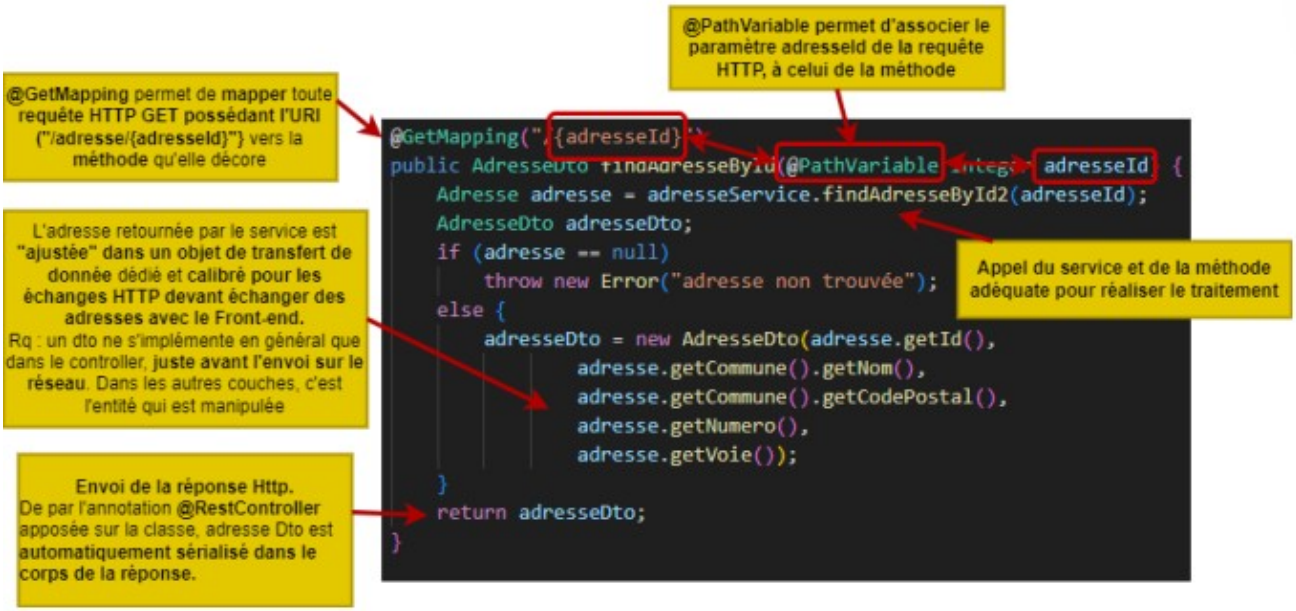


- Exemple de service



- Exemple de controller





5 TESTS ET INTÉGRATION

5.1 Stratégie de tests unitaires

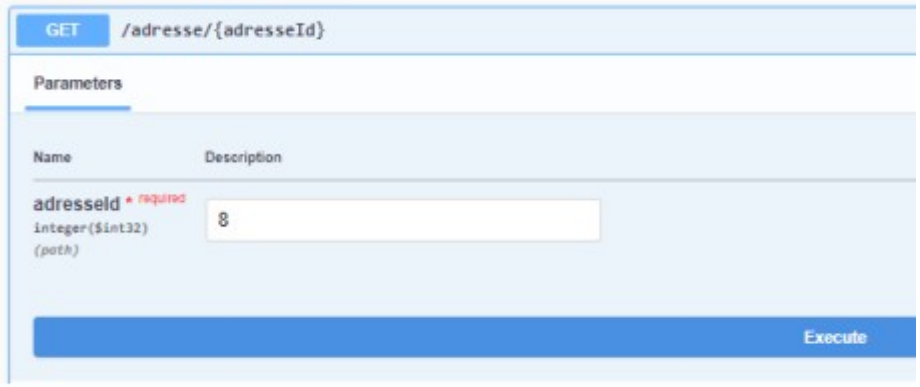
- De façon générale, toutes les méthodes codées devraient comporter un test unitaire.
- Les repositories n'auront pas à être testés (code Hibernate)
- Tous les services devront comporter un test unitaire
    - Toutes les dépendances utilisées par la classe seront simulées à l'aide de doublure Mokito
    - Exemple de test d'un service :

- Tous les controllers devront comporter un test unitaire
    - Exemple de test d'un controller


@Test indique à JUnit que la méthode à laquelle elle est attachée peut être exécutée en tant que cas de test
Comportement à simuler par la doublure de AdresseService à l'aide des méthode when() et thenReturn() de Mockito
Envoi de la requête GET
Contrôle du status de la réponse http retournée
Contrôle du format de données reçu
Vérification de l'intégrité des données retournées
Envoi du résultat de la requête HTTP sur la sortie standard
On s'assure que la doublure du service a bien été appelé, une unique fois. S'utilise en général en fin de test
@Test
public void testGetShouldReturn200OK() throws Exception {
Mockito.when(this.adresseService.findAdresseById2(adresseRef.getId()))
.thenReturn(adresseRef);
mockMvc.perform(get("/adresse/{id}", adresseRef.getId()))
.andExpect(status().isOk())
.andExpect(content().contentType(MediaType.APPLICATION_JSON))
.andExpect(jsonPath("$['id']").value(adresseRef.getId()))
.andExpect(jsonPath("$['commune']").value(adresseRef.getCommune().getNom()))
.andExpect(jsonPath("$['codePostal']").value(adresseRef.getCommune().getCodePostal()))
.andExpect(jsonPath("$['numero']").value(adresseRef.getNumero()))
.andExpect(jsonPath("$['voie']").value(adresseRef.getVoie()))
.andDo(print());
Mockito.verify(this.adresseService,
times(1)).findAdresseById2(adresseRef.getId());
}

5.2 Stratégie de tests d'intégration

- Tous les Endpoints de l'application devront à minima être testé manuellement via l'interface open-API (Swagger)



- Voir pour l'automatisation du test des Endpoint