

Projet M2 ACSI Spring Boot

**Développement d'une API Restful avec
Spring Boot pour la recherche et la
réservation de trajets en train**

Année 2021-2022

Travail réalisé par : Manel NASRI

Proposé par : Mr. Olivier Perrin

1. Contexte et objectifs du projet :

L'objectif de ce projet est de proposer une API métier qui va permettre à un utilisateur donné de rechercher, sélectionner et payer un trajet en train entre deux villes. L'utilisateur pourra choisir un horaire, un type de siège (couloir ou fenêtre) ainsi que le type du trajet (Aller simple ou Aller Retour).

1. Choix technologiques

Pour réaliser ce projet, j'ai fait un certain nombre de choix au niveau des technologies utilisées :

1. IDE Spring Tool Suite (STS) : Il s'agit d'un IDE étendu pour Eclipse. Il se spécialise dans le développement des applications Spring.

2. Spring Initializr : C'est un outil Web qui génère la structure du projet Spring Boot. Les IDE modernes ont intégré Spring Initializr qui fournit la structure initiale du projet. Il est très facile pour les développeurs de sélectionner la configuration nécessaire pour leurs projets. L'outil Spring Initializr prend en charge la configuration suivante pour tout projet basé sur Spring.

- Outil de génération (Maven ou Gradle) pour créer l'application.
- Version Spring Boot (les dépendances sont ajoutées en fonction de la version).
- Dépendances requises pour le projet.
- Langue et sa version.
- Les métadonnées du projet telles que le nom, l'emballage (Jar ou War), le nom du package, etc.

Avec toutes les informations fournies, Spring Initializr génère la structure du projet Spring. Spring Initializr est utilisable à partir du Web, de l'IDE ou de la ligne de commande.

3. Les liens hypermédia (HATEOAS) : HATEOAS (Hypermedia As The Engine of Application State, est une contrainte de l'architecture d'application REST qui la distingue de la plupart des autres architectures d'applications réseau. Le principe de HATEOS est qu'un client interagit avec une application réseau entièrement par hypermédia fournie dynamiquement par les serveurs d'applications.

Un client REST accède à une application REST à l'aide d'une simple URL. Toutes les futures actions que le client peut entreprendre sont découvertes dans les représentations de la ressource retournée par le serveur.

4. **Base de données H2** : H2 est un système de gestion de base de données relationnelles écrit en Java qui peut être intégré à une application Java ou bien fonctionner en mode client-serveur.
5. **Consul** : Il s'agit d'une solution pour la découverte de services et la configuration, conçue pour fonctionner de manière distribuée, hautement disponible et évolutive pour des milliers de nœuds.
6. **Postman** : Il s'agit d'un outil pour créer et utiliser des API Restful.

2. Conception et modèle de données

Pour la réalisation d'une API Restful permettant la recherche et la réservation de billets de train, j'ai construit une base de données relationnelle H2 dont le schéma relationnel est présenté dans la Figure 1 :

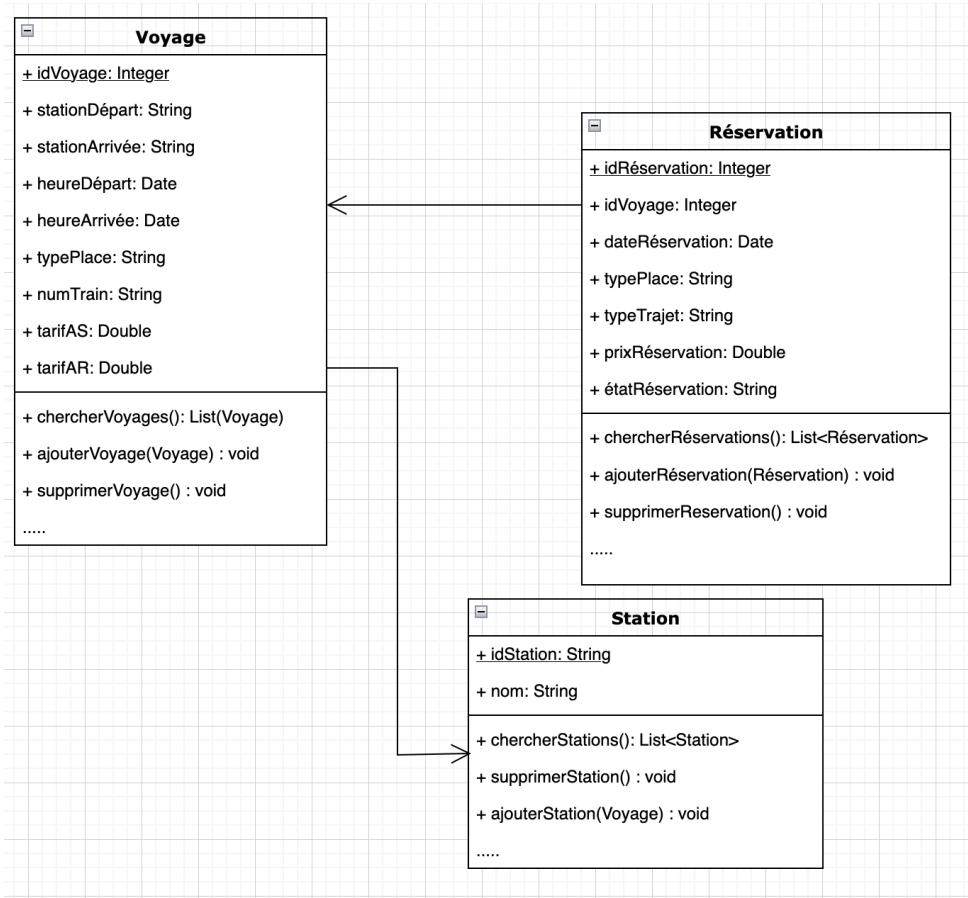
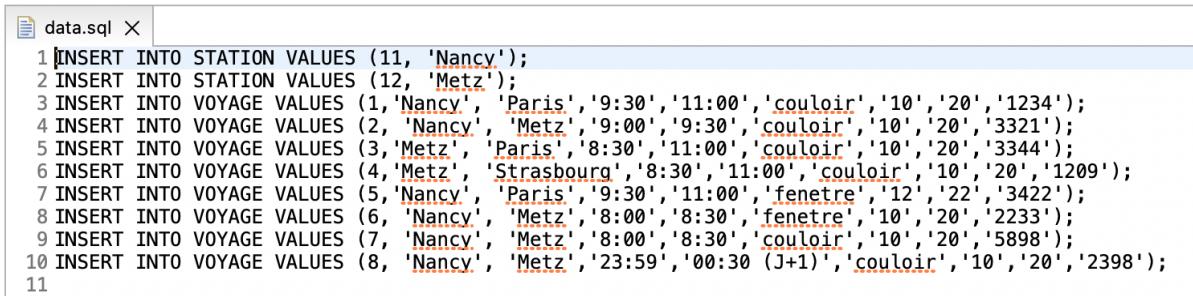


Figure 1 : schéma relationnel de la base de données

L'ensemble des relations présentées dans ce schéma sont en 3ème forme normale (3NF) car : (1) tous les attributs sont atomiques et monovalués (1NF), (2) il n'y a pas de dépendances entre les attributs hors clé et une partie de la clé (2NF), et (3) il n'y a pas de dépendances entre les attributs hors clé (3NF).

Les requêtes de peuplement de la base de données H2 sont présentées dans la figure suivante:



```

data.sql ×
1 INSERT INTO STATION VALUES (11, 'Nancy');
2 INSERT INTO STATION VALUES (12, 'Metz');
3 INSERT INTO VOYAGE VALUES (1, 'Nancy', 'Paris', '9:30', '11:00', 'couloir', '10', '20', '1234');
4 INSERT INTO VOYAGE VALUES (2, 'Nancy', 'Metz', '9:00', '9:30', 'couloir', '10', '20', '3321');
5 INSERT INTO VOYAGE VALUES (3, 'Metz', 'Paris', '8:30', '11:00', 'couloir', '10', '20', '3344');
6 INSERT INTO VOYAGE VALUES (4, 'Metz', 'Strasbourg', '8:30', '11:00', 'couloir', '10', '20', '1209');
7 INSERT INTO VOYAGE VALUES (5, 'Nancy', 'Paris', '9:30', '11:00', 'fenetre', '12', '22', '3422');
8 INSERT INTO VOYAGE VALUES (6, 'Nancy', 'Metz', '8:00', '8:30', 'fenetre', '10', '20', '2233');
9 INSERT INTO VOYAGE VALUES (7, 'Nancy', 'Metz', '8:00', '8:30', 'couloir', '10', '20', '5898');
10 INSERT INTO VOYAGE VALUES (8, 'Nancy', 'Metz', '23:59', '00:30 (J+1)', 'couloir', '10', '20', '2398');
11

```

Figure 2 : requêtes de peuplement de la base de données H2 de l'API

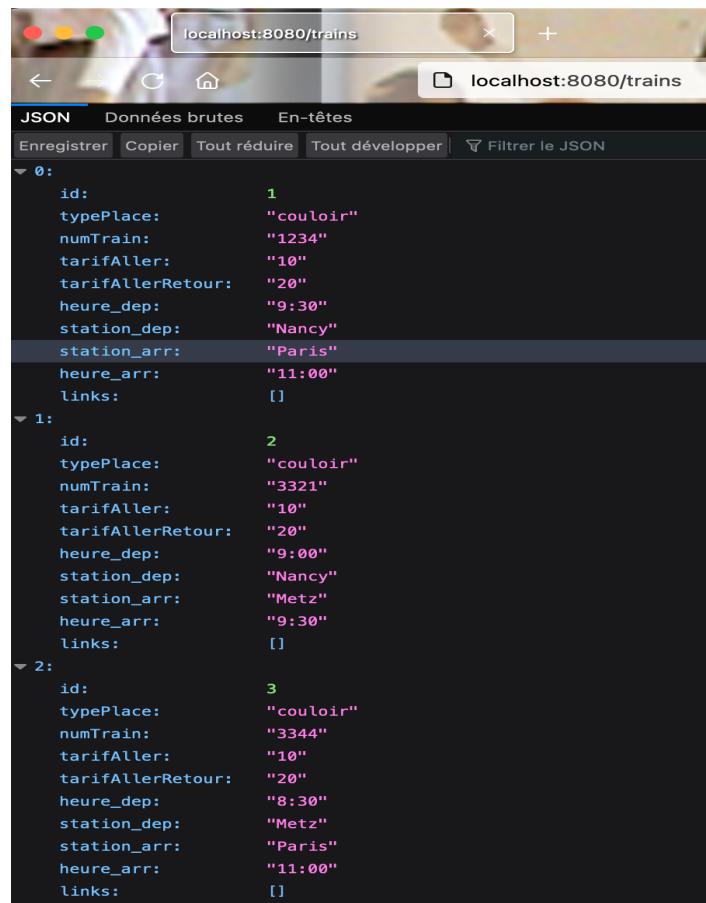
3. Réalisation

L'API développée propose des services de recherche, de réservation et de paiement de trajets en train. Voici quelques exemples de requêtes possibles :

a. Recherche de trajets

La fonctionnalité “Recherche de trajets” permet à un utilisateur de faire des recherches sur des trajets en train en spécifiant la ville de départ, la ville d'arrivée et éventuellement le type de place (couloir ou fenêtre) et l'horaire. Ci-dessous, des exemples de requêtes utilisateurs pour la recherche de trajets en train.

- Liste de tous les trajets disponibles



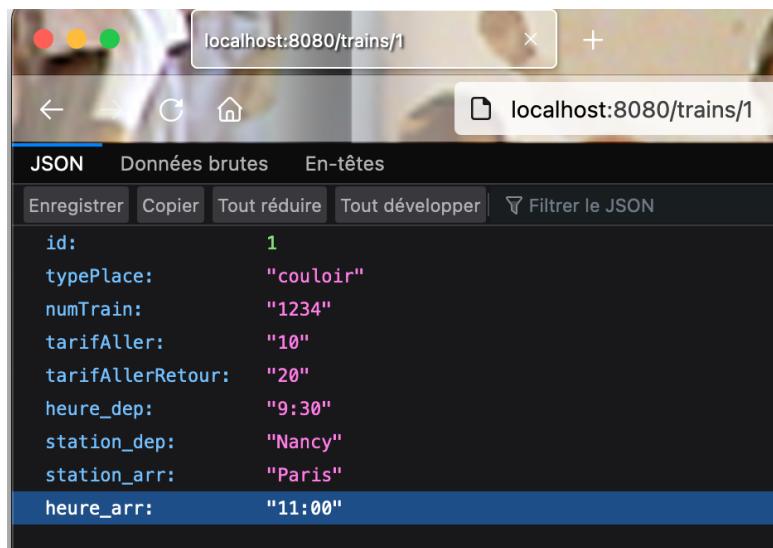
The screenshot shows a browser window with the URL `localhost:8080/trains`. The page displays a JSON array of three train journey objects. Each object has properties: id, typePlace, numTrain, tarifAller, tarifAllerRetour, heure_dep, station_dep, station_arr, heure_arr, and links.

```

[{"id": 1, "typePlace": "couloir", "numTrain": "1234", "tarifAller": "10", "tarifAllerRetour": "20", "heure_dep": "9:30", "station_dep": "Nancy", "station_arr": "Paris", "heure_arr": "11:00", "links": []}, {"id": 2, "typePlace": "couloir", "numTrain": "3321", "tarifAller": "10", "tarifAllerRetour": "20", "heure_dep": "9:00", "station_dep": "Nancy", "station_arr": "Metz", "heure_arr": "9:30", "links": []}, {"id": 3, "typePlace": "couloir", "numTrain": "3344", "tarifAller": "10", "tarifAllerRetour": "20", "heure_dep": "8:30", "station_dep": "Metz", "station_arr": "Paris", "heure_arr": "11:00", "links": []}]
  
```

Figure 3 : Liste de tous les trajets disponibles

- **Recherche d'un voyage en utilisant l'identifiant du voyage**



The screenshot shows a browser window with the URL `localhost:8080/trains/1`. The page displays a single JSON object representing a train journey. The object has properties: id, typePlace, numTrain, tarifAller, tarifAllerRetour, heure_dep, station_dep, station_arr, and heure_arr.

```

{id: 1, "typePlace": "couloir", "numTrain": "1234", "tarifAller": "10", "tarifAllerRetour": "20", "heure_dep": "9:30", "station_dep": "Nancy", "station_arr": "Paris", "heure_arr": "11:00"}
  
```

Figure 4 : Recherche d'un voyage en utilisant l'identifiant du voyage

- **Recherche d'un trajet en Aller Simple d'une ville A à une ville B**

```

localhost:8080/trainsAS/Nancy/Metz
localhost:8080/trainsAS/Nancy/Metz

JSON  Données brutes  En-têtes
Enregistrer Copier Tout réduire Tout développer Filtrer le JSON

0:
  id: 2
  typePlace: "couloir"
  numTrain: "3321"
  tarifAller: "10"
  tarifAllerRetour: "20"
  heure_dep: "9:00"
  station_dep: "Nancy"
  station_arr: "Metz"
  heure_arr: "9:30"
  links:
    0:
      rel: "Réservation en Aller Simple"
      href: "http://localhost:8080/reservation/2/couloir/ALLER%20SIMPLE/10"
1:
  id: 6
  typePlace: "fenetre"
  numTrain: "2233"
  tarifAller: "10"
  tarifAllerRetour: "20"
  heure_dep: "8:00"
  station_dep: "Nancy"
  station_arr: "Metz"
  heure_arr: "8:30"
  links:
    0:
      rel: "Réservation en Aller Simple"
      href: "http://localhost:8080/reservation/6/fenetre/ALLER%20SIMPLE/10"

```

Figure 5 : Recherche d'un voyage en Aller Retour de nancy à Metz

Comme illustré dans la Figure 5, des liens HATEOS sont créés avec chaque voyage pour permettre à l'utilisateur de réserver le trajet pour un Aller Simple.

- **Recherche d'un trajet en Aller Retour d'une ville A à une ville B**

```

{
  "id": 2,
  "typePlace": "couloir",
  "numTrain": "3321",
  "tarifAller": "10",
  "tarifAllerRetour": "20",
  "heure_dep": "9:00",
  "station_dep": "Nancy",
  "station_arr": "Metz",
  "heure_arr": "9:30",
  "links": [
    {
      "rel": "Réservation en Aller Simple",
      "href": "http://localhost:8080/reservation/2/couloir/ALLER%20SIMPLE/10"
    },
    {
      "rel": "Réservation en Aller Retour",
      "href": "http://localhost:8080/reservation/2/couloir/ALLER%20RETOUR/20"
    }
  ]
},
{
  "id": 6,
  "typePlace": "fenetre",
  "numTrain": "2233",
  "tarifAller": "10",
  "tarifAllerRetour": "20",
  "heure_dep": "8:00",
  "station_dep": "Nancy",
  "station_arr": "Metz",
  "heure_arr": "8:30",
  "links": [
    {
      "rel": "Réservation en Aller Simple",
      "href": "http://localhost:8080/reservation/6/fenetre/ALLER%20SIMPLE/10"
    },
    {
      "rel": "Réservation en Aller Retour",
      "href": "http://localhost:8080/reservation/6/fenetre/ALLER%20RETOUR/20"
    }
  ]
}

```

Figure 6 : Recherche d'un voyage en Aller Retour de nancy à Metz

Comme illustré dans la Figure 6, des liens HATEOS sont créés avec chaque voyage pour permettre à l'utilisateur de réserver le trajet pour un Aller Simple ou un Aller Retour.

- **Recherche d'un trajet en Aller Retour d'une ville A à une ville B en précisant le type de place**

```

localhost:8080/trainsAR/Nancy/Metz
localhost:8080/trainsAR/Nancy/Metz/couloir

JSON  Données brutes  En-têtes
Enregistrer Copier Tout réduire Tout développer  Filtrer le JSON

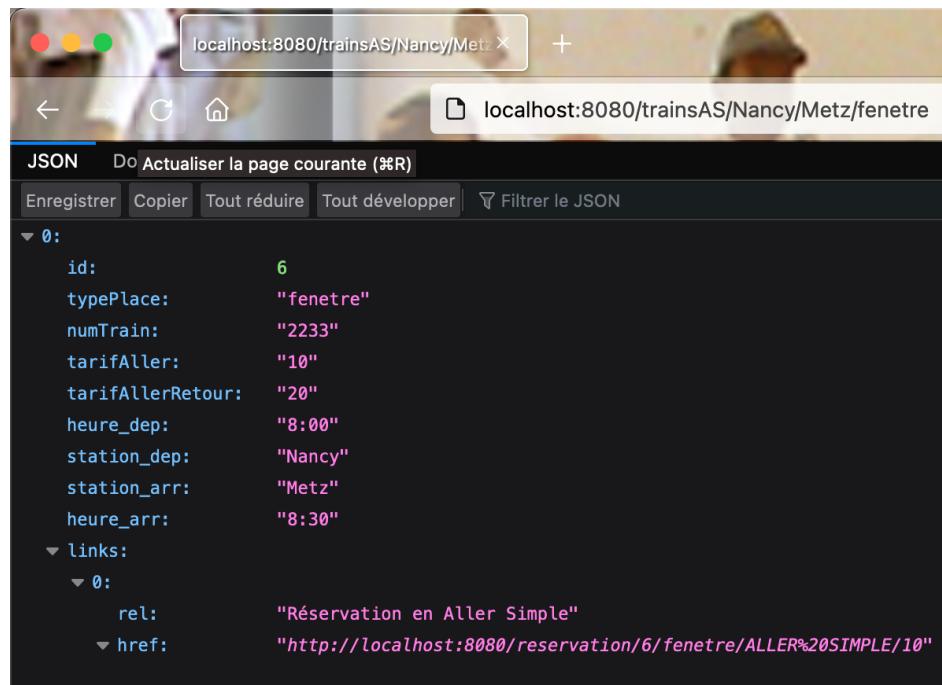
0:
  id: 2
  typePlace: "couloir"
  numTrain: "3321"
  tarifAller: "10"
  tarifAllerRetour: "20"
  heure_dep: "9:00"
  station_dep: "Nancy"
  station_arr: "Metz"
  heure_arr: "9:30"
  links:
    0:
      rel: "Réservation en Aller Retour"
      href: "http://localhost:8080/reservation/2/couloir/ALLER%20RETOUR/20"
1:
  id: 7
  typePlace: "couloir"
  numTrain: "5898"
  tarifAller: "10"
  tarifAllerRetour: "20"
  heure_dep: "8:00"
  station_dep: "Nancy"
  station_arr: "Metz"
  heure_arr: "8:30"
  links:
    0:
      rel: "Réservation en Aller Retour"
      href: "http://localhost:8080/reservation/7/couloir/ALLER%20RETOUR/20"

```

Figure 7 : Recherche d'un voyage en Aller Retour de nancy à Metz avec un type de place “couloir”

Comme illustré dans la Figure 7, des liens HATEOS sont créés avec chaque voyage pour permettre à l'utilisateur de réserver le trajet pour un Aller Retour.

- Recherche d'un trajet en Aller Simple d'une ville A à une ville B en précisant le type de place



```

localhost:8080/trainsAS/Nancy/Metz
localhost:8080/trainsAS/Nancy/Metz/fenetree

JSON Do Actualiser la page courante (⌘R)
Enregistrer Copier Tout réduire Tout développer Filtrer le JSON

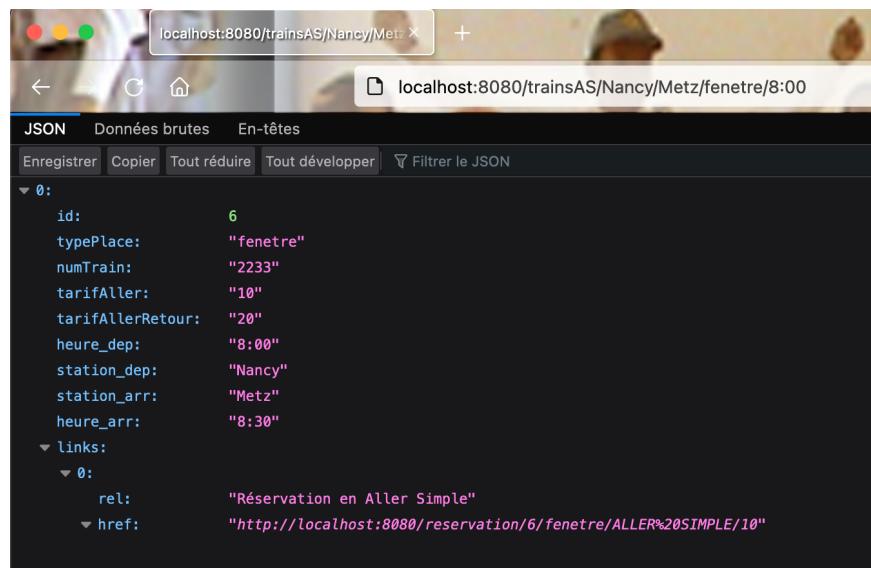
▼ 0:
  id: 6
  typePlace: "fenetre"
  numTrain: "2233"
  tarifAller: "10"
  tarifAllerRetour: "20"
  heure_dep: "8:00"
  station_dep: "Nancy"
  station_arr: "Metz"
  heure_arr: "8:30"
  ▼ links:
    ▼ 0:
      rel: "Réservation en Aller Simple"
      ▼ href: "http://localhost:8080/reservation/6/fenetree/ALLER%20SIMPLE/10"

```

Figure 8 : Recherche d'un voyage en Aller Simple de Nancy à Metz dans un siège de type "fenêtre"

Comme illustré dans la Figure 8, un lien HATEOS est créé avec chaque voyage pour permettre à l'utilisateur de réserver le trajet pour un Aller Simple.

- **Recherche d'un trajet en Aller Simple d'une ville A à une ville B en précisant le type de place ainsi que l'horaire**



```

localhost:8080/trainsAS/Nancy/Metz
localhost:8080/trainsAS/Nancy/Metz/8:00

JSON Données brutes En-têtes
Enregistrer Copier Tout réduire Tout développer Filtrer le JSON

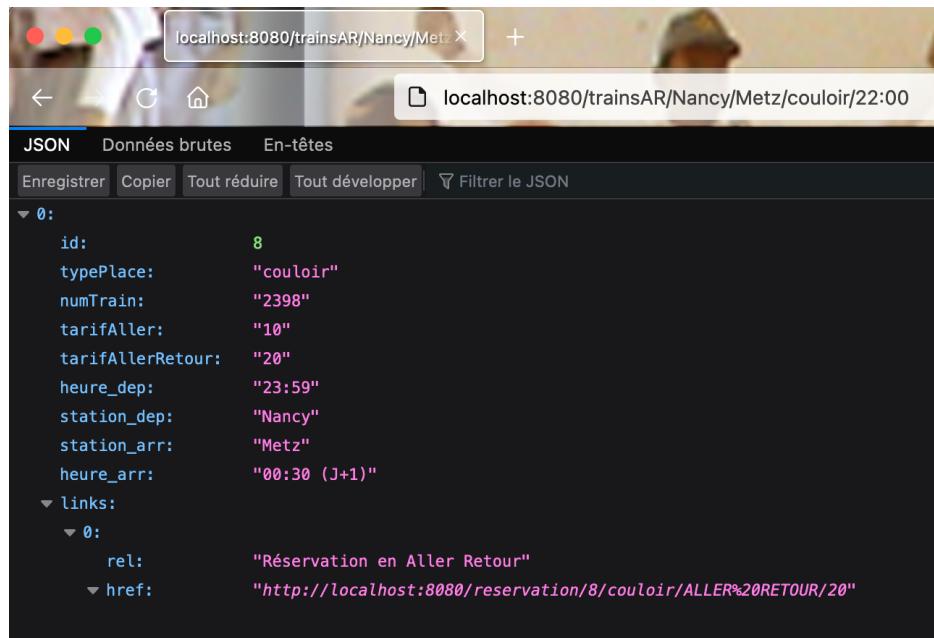
▼ 0:
  id: 6
  typePlace: "fenetre"
  numTrain: "2233"
  tarifAller: "10"
  tarifAllerRetour: "20"
  heure_dep: "8:00"
  station_dep: "Nancy"
  station_arr: "Metz"
  heure_arr: "8:30"
  ▼ links:
    ▼ 0:
      rel: "Réservation en Aller Simple"
      ▼ href: "http://localhost:8080/reservation/6/fenetree/ALLER%20SIMPLE/10"

```

Figure 9 : Recherche d'un voyage en Aller Simple de Nancy à Metz dans un siège de type "fenêtre" à partir de 8h:00

Comme illustré dans la figure ci-dessus, un lien HATEOS est créé avec chaque voyage pour permettre à l'utilisateur de réserver le trajet pour un Aller Simple.

- **Recherche d'un trajet en Aller Retour d'une ville A à une ville B en précisant le type de place ainsi que l'horaire**



```

{
  "id": 8,
  "typePlace": "couloir",
  "numTrain": "2398",
  "tarifAller": "10",
  "tarifAllerRetour": "20",
  "heure_dep": "23:59",
  "station_dep": "Nancy",
  "station_arr": "Metz",
  "heure_arr": "00:30 (J+1)",
  "links": [
    {
      "rel": "Réservation en Aller Retour",
      "href": "http://localhost:8080/reservation/8/couloir/ALLER%20RETOUR/20"
    }
  ]
}

```

Figure 10 : Recherche d'un voyage en Aller Retour de Nancy à Metz dans un siège de type “couloir” à partir de 22h:00

Comme illustré dans la Figure 10, un lien HATEOS est créé avec chaque voyage pour permettre à l'utilisateur de réserver le trajet pour un Aller Retour.

b. Réservation de train

La fonctionnalité “Réservation de train” permet à un utilisateur de :

- Faire une réservation de trajet après avoir fait la recherche de celui-ci.
- Afficher une réservation ou la liste de l'ensemble des réservations effectuées (confirmées ou non).
- Confirmer une réservation

Dans la partie suivante, je vais vous présenter quelques exemples de requêtes de réservation.

- **Réervation d'un trajet**

Après avoir effectué une recherche sur des trajets en Aller Simple “Nancy” vers “Metz”, l’utilisateur décide de faire la réservation en cliquant sur le lien HATEOS afficher avec les informations du trajet (voir figures suivantes).

```

localhost:8080/trainsAS/Nancy/Metz X
localhost:8080/trainsAS/Nancy/Metz/couloir

JSON Données brutes En-têtes
Enregistrer Copier Tout réduire Tout développer Filtrer le JSON

▼ 0:
  id: 2
  typePlace: "couloir"
  numTrain: "3321"
  tarifAller: "10"
  tarifAllerRetour: "20"
  heure_dep: "9:00"
  station_dep: "Nancy"
  station_arr: "Metz"
  heure_arr: "9:30"
  ▼ links:
    ▼ 0:
      rel: "Réservation en Aller Simple"
      ▼ href: "http://localhost:8080/reservation/2/couloir/ALLER%20SIMPLE/10"

```

Figure 11 : résultat de la recherche d'un trajet Nancy Metz en Aller Simple et une place de type “couloir”

Quand l’utilisateur clique sur le lien HATEOS, une réservation avec un statut “NON CONFIRMÉE” va être créée (voir figure suivante).

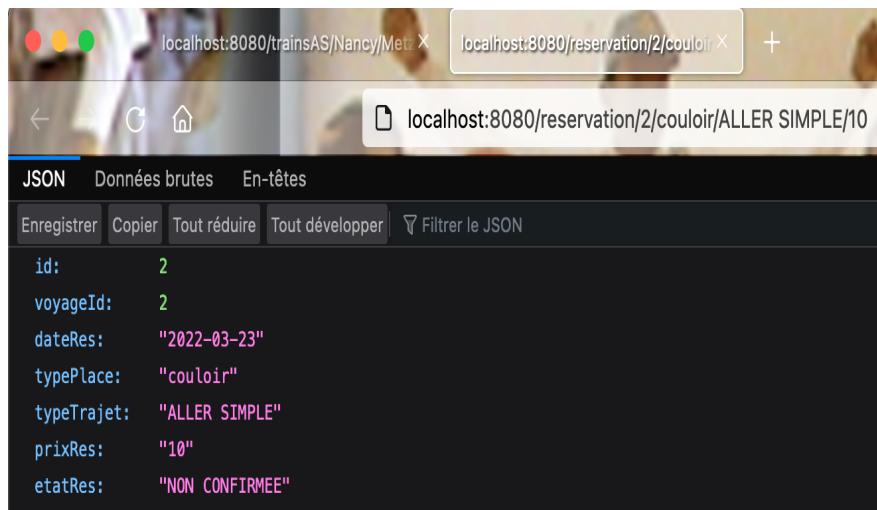


Figure 12 : création d'une réservation

- **Gestion des réservations**

L'API Restful proposée permet à un utilisateur d'afficher la liste de ses réservations (voir figure suivante) :

```

localhost:8080/reservations/
localhost:8080/reservations/

JSON Données brutes En-têtes
Enregistrer Copier Tout réduire Tout développer Filtrer le JSON

0:
  id: 1
  voyageId: 8
  dateRes: "2022-03-23"
  typePlace: "couloir"
  typeTrajet: "ALLER RETOUR"
  prixRes: "20"
  etatRes: "NON CONFIRMEE"
  links:
    0:
      rel: "confirmer-et-payer-reservation"
      href: "http://localhost:8080/confirmer-reservation/1"
1:
  id: 2
  voyageId: 2
  dateRes: "2022-03-23"
  typePlace: "couloir"
  typeTrajet: "ALLER SIMPLE"
  prixRes: "10"
  etatRes: "NON CONFIRMEE"
  links:
    0:
      rel: "confirmer-et-payer-reservation"
      href: "http://localhost:8080/confirmer-reservation/2"

```

Figure 13 : Liste des réservation d'un utilisateur

Comme illustré dans la Figure 13, un lien HATEOS est affiché avec chaque réservation qui permet de confirmer et payer la réservation effectuée. Quand l'utilisateur clique sur le lien de confirmation et de paiement, une requête est envoyée au serveur qui permet de valider/confirmer la réservation et de changer l'état de la réservation vers “CONFIRMEE” (voir Figure 14).

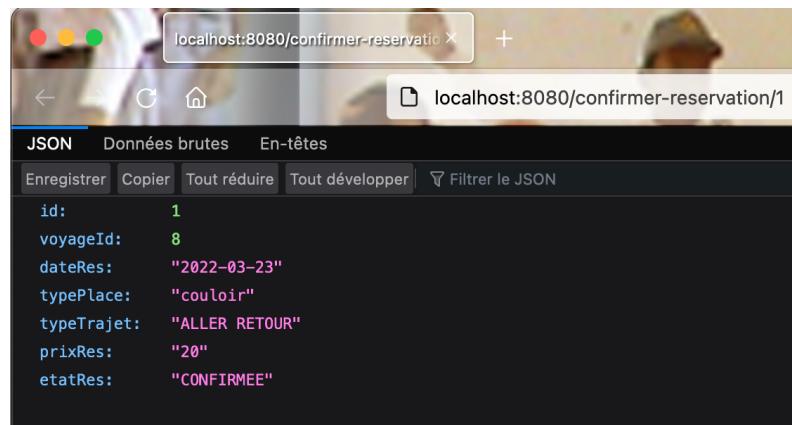


Figure 14 : Confirmation d'une réservation

c. Découverte et enregistrement des services avec Consul

Comme expliqué dans la section 2, Consul est un outil utilisé pour la découverte, l'enregistrement de services ainsi que la vérification du bon fonctionnement des services (*Health Checking*).

L'API développée permet d'utiliser Consul afin d'enregistrer et de découvrir l'emplacement des services en cours d'exécution via des requêtes REST.

Après avoir exécuté, dans un premier lieu, l'API avec la commande : java -jar "-Dserver.port=8080" app-0.0.1-SNAPSHOT.jar, Consul détecte automatiquement l'existence du service (voir Figure 15).

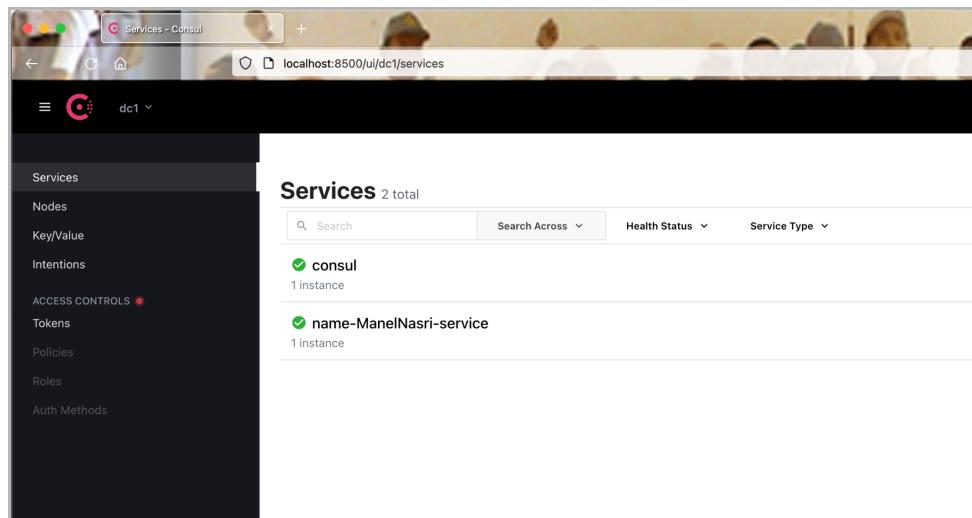


Figure 15 : Découverte d'une instance du service par le Consul

Dans un second lieu, j'ai lancé une deuxième instance de l'API avec la commande `java -jar "-Dserver.port=8081" app-0.0.1-SNAPSHOT.jar`, le Consul détecte les deux instances du service (voir figures 16 et 17).

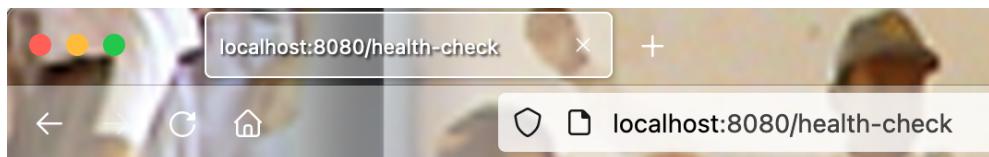
The screenshot shows the Consul UI with the URL `localhost:8500/ui/dc1/services`. The sidebar on the left includes options like Services, Nodes, Key/Value, Intentions, ACCESS CONTROLS, Tokens, Policies, Roles, and Auth Methods. The main panel displays the 'Services' section with a total of 2 instances. It lists 'consul' with 1 instance and 'name-ManelNasri-service' with 2 instances. Both entries have a green checkmark indicating they are healthy.

Figure 16 : Découverte de deux instances du service par le Consul

The screenshot shows the Consul UI with the URL `localhost:8500/ui/dc1/services/name-ManelNasri-service/instances`. The sidebar is identical to Figure 16. The main panel shows the 'name-ManelNasri-service' section. It has tabs for Instances, Intentions, Routing, and Tags, with 'Instances' selected. There are two listed instances: one with ID `id-51223fc9-8926-40f6-8c1b-c3468d33828f` and another with ID `id-b21df858-039a-4de9-a54b-f7b9473dd40a`. Both instances show green checkmarks for 'All service checks passing' and 'All node checks passing', along with their respective IP addresses (192.168.1.119) and ports (8081 and 8080).

Figure 17 : Découverte de deux instances du service par le Consul

Pour vérifier que l'API fonctionne correctement, j'ai implémenté un controller simple qui permet de faire une vérification simple du bon fonctionnement du service. La figure suivante présente le résultat de la requête REST qui permet de faire cette vérification (Health Check) :



Les service de réservation de billets de train est OK !

Figure 18 : Vérification du bon fonctionnement de l'API

d. Test de l'application

Pour tester l'API développée, j'ai implémenter les tests nécessaires pour vérifier les différentes fonctionnalités.

La figure suivante présente le code Java pour le test de la fonctionnalité “recherche de voyage en Aller Retour” du service Voyage :

```
VoyageServiceTest.java X
1 package com.miage.app.service;
2
3 import static org.junit.jupiter.api.Assertions.assertEquals;
4
5 @ExtendWith(MockitoExtension.class)
6 public class VoyageServiceTest {
7
8     @InjectMocks
9     VoyageService voyageService2;
10    @Mock
11    VoyageRepository voyageRepository;
12
13    @Test
14    public void testChercherVoyagesAllerSimple() throws Exception {
15        Voyage voyage1 = new Voyage(109, "Nancy", "Metz", "9:00", "10:00", "couloir", "10", "15", "2333");
16        Voyage voyage2 = new Voyage(109, "Metz", "Nancy", "11:00", "12:00", "couloir", "10", "15", "3211");
17        List<Voyage> voyages = new ArrayList<Voyage>();
18        voyages.add(voyage2);
19        voyages.add(voyage1);
20
21        when(voyageRepository.findByStationDepAndStationArrLike("Nancy", "Metz"))
22            .thenReturn(Arrays.asList(voyage1));
23
24        List<Voyage> liste = voyageService2.chercherVoyagesAllerSimple("Nancy", "Metz");
25
26        assertEquals(1, liste.size());
27    }
28}
```

Figure 19 : Implémentation du test du service Voyage

La Figure 20 présente l'exécution de quelques tests JUnit de l'API développée.

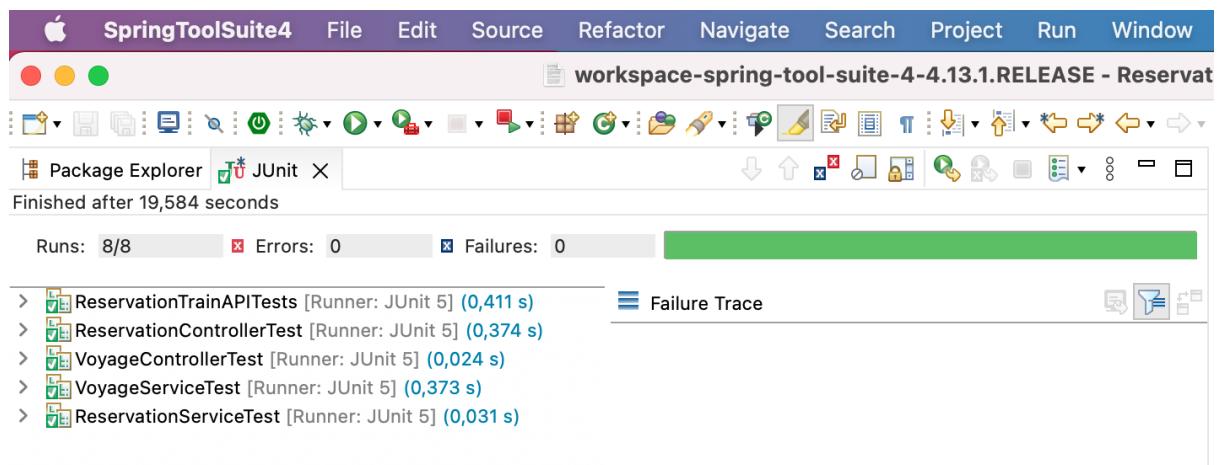


Figure 20 : Fenêtre JUnit de SpringToolSuite

4. Conclusion

Ce projet vise à développer une API qui permet de rechercher et/ou réserver des trajets en train. L'API développée répond aux spécifications du projet et propose deux services différents (service *Voyage* et service *Réservation*). L'ensemble des fonctionnalités ont été testées avec des tests unitaires pour vérifier le bon fonctionnement de l'API.

Plusieurs améliorations possibles peuvent être ajoutées comme par exemple la gestion des utilisateurs, la gestion des trains ainsi que la création d'une interface graphique pour une exploitation plus facile de l'API.