

Rapport sur les micro-services REST

ACHBAD Fatima

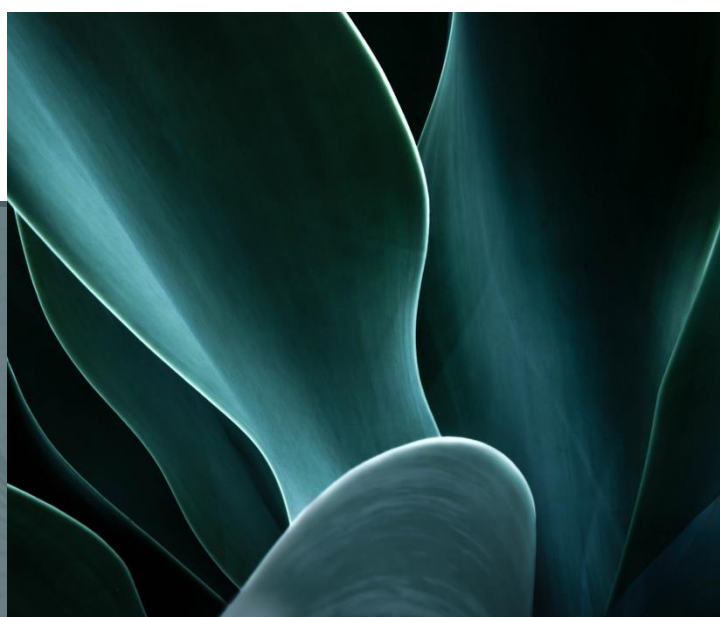
30/10/2023

—

Mohamed YOUSSEFI

Introduction

Les microservices REST, ou services web RESTful, sont une approche d'architecture logicielle qui permet la création d'applications distribuées, évolutives et interconnectées en utilisant les principes de l'architecture orientée ressources (Representational State Transfer, ou REST). Contrairement à SOAP, qui est basé sur XML et est souvent considéré comme lourd et complexe, les microservices REST se concentrent sur la simplicité, la légèreté et l'utilisation d'URL (Uniform Resource Locators) pour accéder à des ressources via des méthodes HTTP standard.





Objectifs du rapport

Le présent rapport vise à fournir une introduction aux micro services REST , permettent aux développeurs de créer et d'utiliser des micro services REST qui peuvent être utilisés par des applications de n'importe quel langage de programmation ou de plateforme. Cette interopérabilité est rendue possible grâce à l'utilisation du protocole HTTP standard et à la simplicité de la communication basée sur des URL. Les microservices REST échangent des données dans un format léger et universel tel que JSON, ce qui facilite la compréhension et l'intégration par des applications variées. De plus dans ce rapport, nous explorerons un cas d'utilisation pratique de microservices REST .

Plan du rapport

Le rapport sera organisé comme suit :

|  *Introduction aux web services*

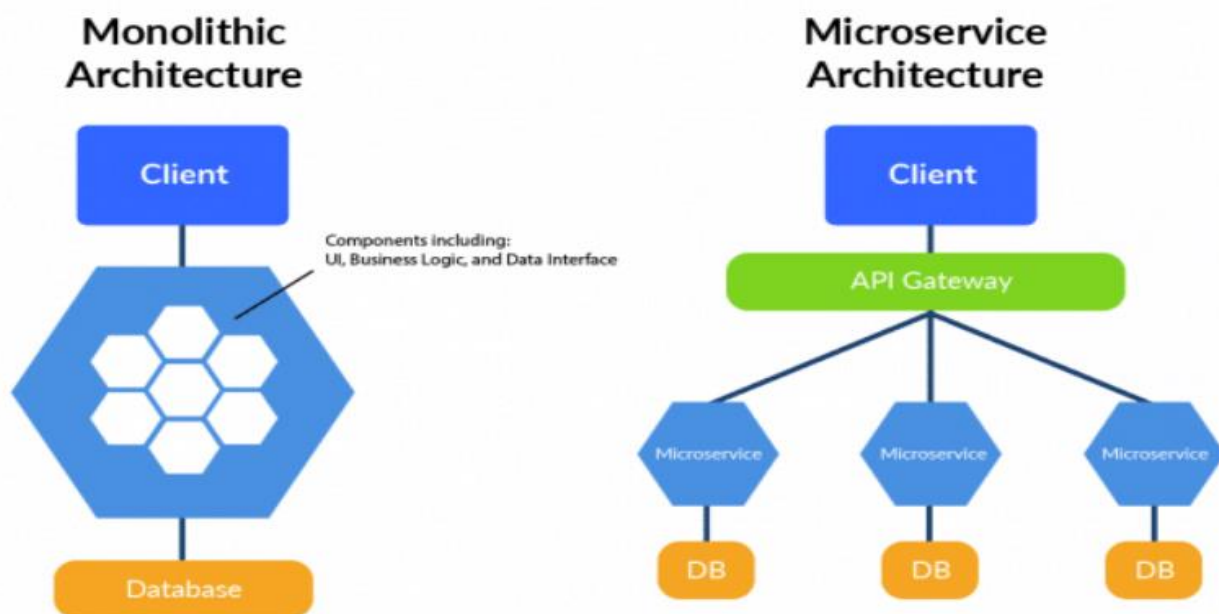
|  *Technologies : REST, SWAGGER, POSTMAN*

|  *Cas Pratique*

|  *Conclusion*

I. Introduction aux microservices REST

Les microservices REST sont une approche d'architecture logicielle qui révolutionne la manière dont les applications interagissent et évoluent. Basés sur les principes de l'architecture orientée ressources (REST), ces microservices sont conçus pour être légères, flexibles et interopérables. Ils permettent aux applications de communiquer efficacement, quel que soit leur langage de programmation ou leur plateforme. Cette approche favorise la modularité, la scalabilité et la simplicité, permettant aux développeurs de créer des services indépendants qui coopèrent pour offrir une fonctionnalité globale.



Un micro service REST :

- ✓ Composant d'une application qui offre un service en tant que ressource identifiée par une URL.
- ✓ Facilite la communication entre les applications, quels que soient le langage de programmation ou la plateforme.
- ✓ Utilise généralement JSON pour la représentation des données, bien qu'il puisse également prendre en charge d'autres formats.
- ✓ Basé sur le protocole HTTP pour la transmission des messages, utilisant les méthodes HTTP standard (GET, POST, PUT, DELETE).

II. Les technologies

1. *REST (Representational State Transfer) :*

- REST est un style d'architecture logicielle qui définit un ensemble de contraintes pour la conception des services web.
- Il repose sur des principes fondamentaux, notamment l'utilisation de ressources (identifiées par des URLs), l'utilisation des méthodes HTTP standard (GET, POST, PUT, DELETE) pour effectuer des opérations sur ces ressources, et l'absence d'état entre les requêtes.
- REST est largement utilisé pour créer des services web légères, évolutives et interopérables, idéales pour les applications modernes et l'intégration entre systèmes distribués.

2. *Swagger :*

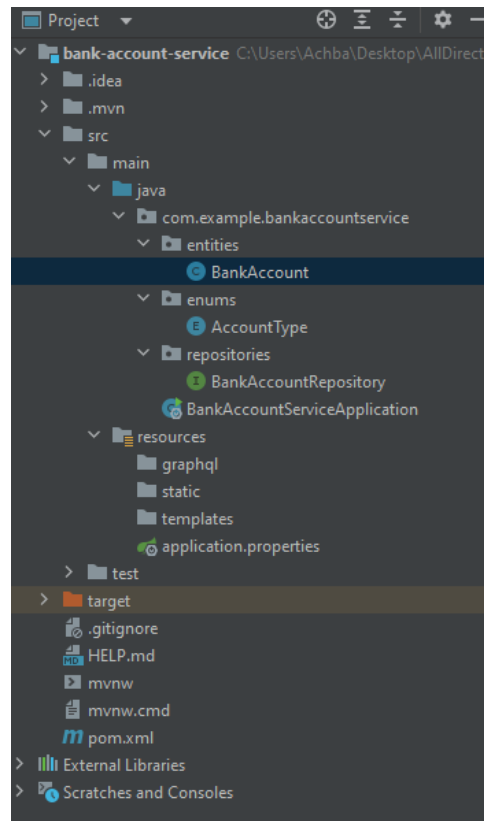
- Swagger est un ensemble d'outils open source pour la conception, la documentation et le test d'API REST.
- Il permet de créer des spécifications détaillées pour les API REST, en utilisant la spécification OpenAPI, anciennement connue sous le nom de Swagger Specification.
- Swagger fournit des outils pour générer automatiquement une documentation interactive pour vos API, ce qui facilite la compréhension et l'utilisation par les développeurs.
- Il offre également des fonctionnalités pour tester et déboguer les API directement depuis l'interface de documentation.

3. *Postman :*

- Postman est un outil populaire pour tester et développer des API, en particulier les API REST.
- Il offre une interface utilisateur conviviale pour créer, envoyer et recevoir des requêtes HTTP vers des API.
- Postman permet de définir des collections de requêtes, de créer des environnements de test, et de gérer l'automatisation des tests API.
- Il est largement utilisé par les développeurs, les testeurs et les équipes de développement d'API pour simplifier le processus de test et de débogage des services web.

Cas pratique

Pour commencer ,on crée un projet spring :



Ou l'entité est BankAccount comme suivant :

1. **@Lombok :**

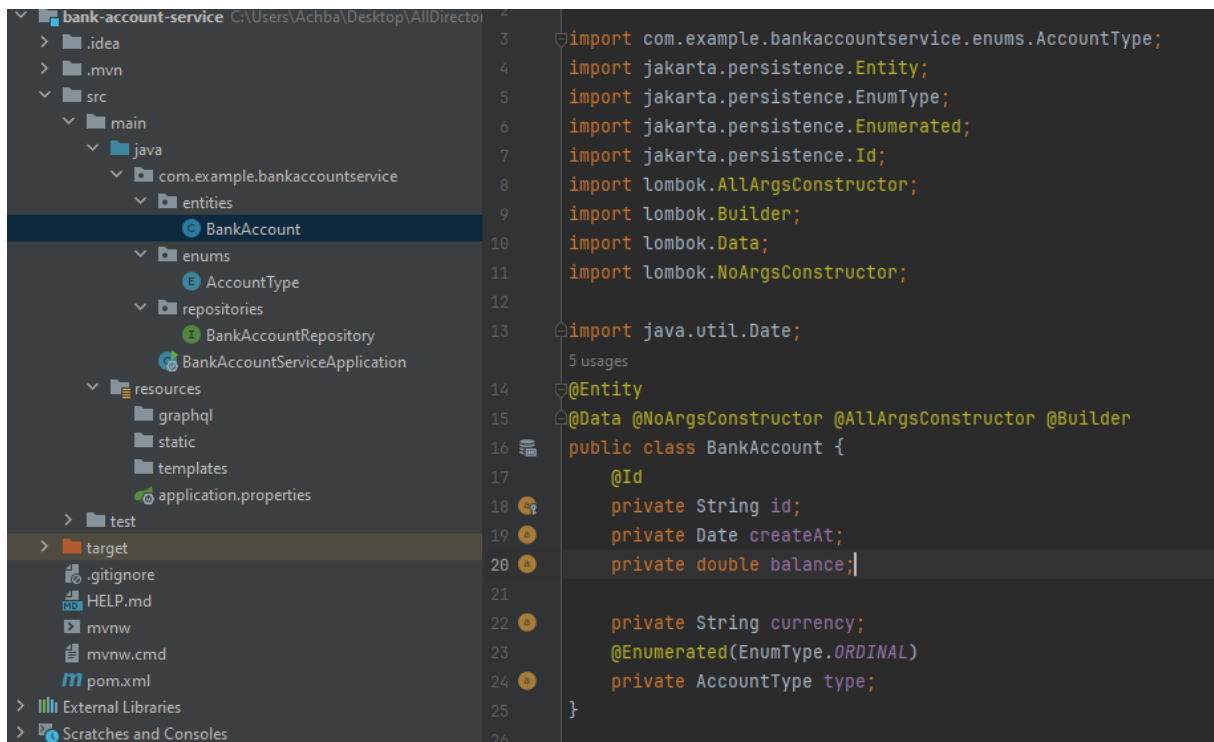
- L'annotation **@Lombok** est un marqueur qui indique que Lombok doit être activé pour la classe.
- Elle n'a pas de fonction spécifique en soi, mais elle permet d'activer les fonctionnalités de Lombok pour la classe annotée.
- En activant Lombok avec **@Lombok**, vous pouvez utiliser d'autres annotations Lombok, telles que **@Data** et **@Builder**, pour simplifier le code.

2. **@Data :**

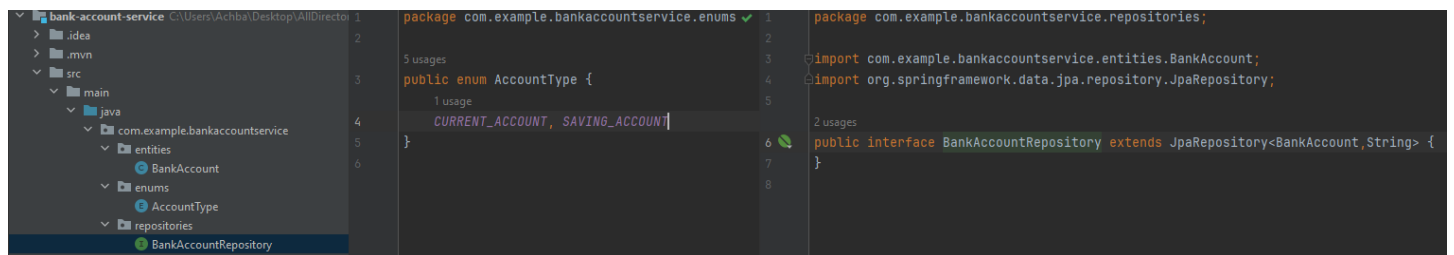
- L'annotation **@Data** est l'une des annotations les plus utilisées de Lombok.
- Elle génère automatiquement les méthodes standard telles que **toString()**, **equals()**, **hashCode()**, ainsi que les méthodes getters et setters pour tous les champs de la classe.
- En utilisant **@Data**, vous pouvez simplifier considérablement le code en évitant d'écrire ces méthodes manuellement.

3. **@Builder :**

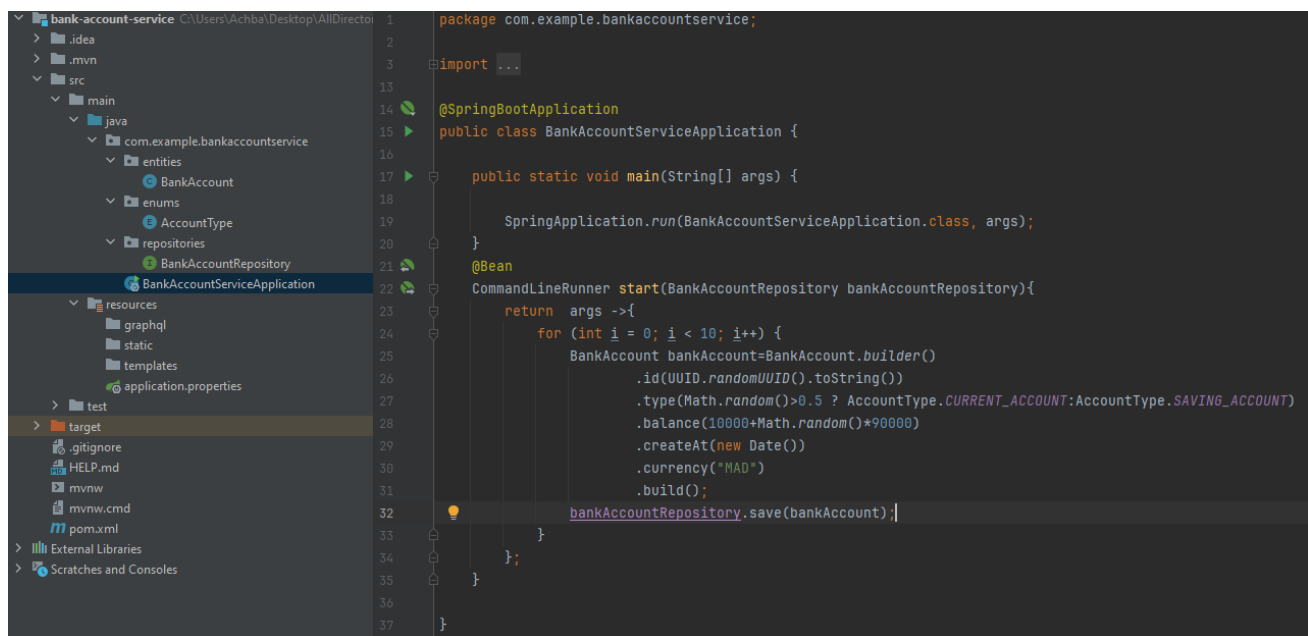
- L'annotation **@Builder** permet de générer automatiquement un constructeur de type "builder" pour une classe.
- Les constructeurs de type builder permettent de créer des instances d'objets en utilisant des méthodes de style fluide, ce qui rend le code plus lisible et plus facile à comprendre.
- Vous pouvez personnaliser les valeurs des champs en chaînant des appels de méthode, ce qui améliore la lisibilité et la maintenabilité du code.



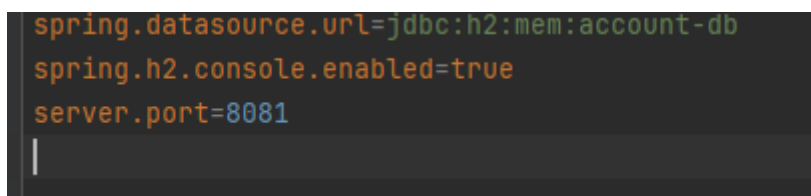
L'annotation `@Enumerated` est une annotation de la Java Persistence API (JPA) qui est utilisée pour mapper un champ d'une entité Java à une colonne d'une table de base de données qui contient des valeurs énumérées (énumérations). Elle permet de spécifier comment les valeurs énumérées doivent être stockées et récupérées dans la base de données.



La méthode annotée avec **@Bean** et retournant un **CommandLineRunner** est un point d'entrée personnalisé qui peut être utilisé pour exécuter des tâches au démarrage de votre application Spring Boot. La méthode **run** de l'interface **CommandLineRunner** sera appelée automatiquement par Spring Boot une fois que l'application Spring a été correctement initialisée et est prête à être exécutée. Cela vous permet d'effectuer des opérations spécifiques au démarrage de l'application. Dans notre cas ,on va pouvoir créer 10 éléments de type BankAccount .



Pour tester on va créer une BD h2 qui s'appelle account-db



On a donc les élément dans la table comme suivant :

Run Run Selected Auto complete Clear SQL statement:

SELECT * FROM BANK_ACCOUNT

BALANCE	CREATE_AT	CURRENCY	ID	TYPE
78038.78057502986	2023-10-27 20:38:07.62	MAD	937c6ed8-e021-4945-842f-4dce193b3910	SAVING_ACCOUNT
10869.241220346144	2023-10-27 20:38:07.729	MAD	df450c78-0c88-46bf-b3e2-e2be26165577	SAVING_ACCOUNT
44324.43817153908	2023-10-27 20:38:07.731	MAD	a364db46-9f24-4bf6-8afe-99279db38508	CURRENT_ACCOUNT
92209.64670087592	2023-10-27 20:38:07.733	MAD	c91d37e7-df02-4002-8992-ff768b789e4d	CURRENT_ACCOUNT
74668.21512088414	2023-10-27 20:38:07.735	MAD	76c50d87-4387-401f-97c6-71c5a85058ed	SAVING_ACCOUNT
26111.23853836454	2023-10-27 20:38:07.736	MAD	c3b061a8-30f9-4716-a485-0e7621341875	CURRENT_ACCOUNT
24311.389826192426	2023-10-27 20:38:07.738	MAD	475602ae-8913-47fe-8fa2-e9dedf75a34c	CURRENT_ACCOUNT
18220.45132058175	2023-10-27 20:38:07.74	MAD	b6acc18f-40e2-46b0-83b2-5980656eca90	SAVING_ACCOUNT
18149.246738576338	2023-10-27 20:38:07.743	MAD	784bc344-f27d-4231-b2ef-c393499295c7	SAVING_ACCOUNT
39971.76697589455	2023-10-27 20:38:07.745	MAD	cc46260f-792c-47a9-bc5b-12169370eabe	CURRENT_ACCOUNT

(10 rows, 6 ms)

Edit

🚦 Créer un microservice REST

Pour tester un microservice REST en créant un contrôleur REST. Les contrôleurs REST sont conçus pour exposer des points de terminaison (endpoints) qui peuvent être appelés à l'aide de requêtes HTTP, ce qui facilite les tests de microservices REST.

```
package com.example.bankaccountservice.web;
import ...
@RestController
@RequestMapping("/api")
public class AccountRestController {
    7 usages
    private BankAccountRepository bankAccountRepository;

    public AccountRestController(BankAccountRepository bankAccountRepository) {
        this.bankAccountRepository = bankAccountRepository;
    }

    @GetMapping("/bankAccounts")
    public List<BankAccount> bankAccounts() { return bankAccountRepository.findAll(); }

    @GetMapping("/bankAccounts/{id}")
    public BankAccount bankAccounts(@PathVariable String id) {
        return bankAccountRepository.findById(id)
            .orElseThrow(() -> new RuntimeException(String.format("Account not found", id)));
    }

    @PostMapping("/bankAccounts")
    public BankAccount save(@RequestBody BankAccount bankAccount) {
        if (bankAccount.getId() == null) bankAccount.setId(UUID.randomUUID().toString());
        return bankAccountRepository.save(bankAccount);
    }

    @PutMapping("/bankAccounts/{id}")
    public BankAccount update(@PathVariable String id, @RequestBody BankAccount bankAccount) {
        BankAccount account = bankAccountRepository.findById(id).orElseThrow();
        if (bankAccount.getBalance() != null) account.setBalance(bankAccount.getBalance());
        if (bankAccount.getCreatedAt() != null) account.setCreatedAt(new Date());
        if (bankAccount.getType() != null) account.setType(bankAccount.getType());
        if (bankAccount.getCurrency() != null) account.setCurrency(bankAccount.getCurrency());
        return bankAccountRepository.save(account);
    }

    @DeleteMapping("/bankAccounts/{id}")
    public void deleteAccount(@PathVariable String id) { bankAccountRepository.deleteById(id); }
```

Vous pouvez tester un microservice REST à la fois en utilisant Postman (ou un outil similaire) et directement depuis un navigateur web.

- **Postman :**

The screenshot shows the Postman interface. The top bar displays the method 'GET' and the URL 'http://localhost:8081/api/bankAccounts'. Below this, the 'Params' tab is selected, showing a table with 'Key' and 'Value' columns. The 'Body' tab is also visible, showing a JSON response in 'Pretty' format. The response is a list of bank accounts with fields like 'id', 'createAt', 'balance', 'currency', and 'type'. The right sidebar shows the 'Headers' tab with a table of headers and the 'Test Results' tab showing a status of '200 OK'.

```
GET http://localhost:8081/api/bankAccounts

Params
Query Params
Key Value
Key Value

Body
Cookies Headers (5) Test Results
Pretty Raw Preview Visualize JSON

[{"id": "cf2d1d7a-5366-4261-baa2-a23a48df6c27", "createAt": "2023-10-29T10:09:33.233+00:00", "balance": 88943.9969310985, "currency": "MAD", "type": "CURRENT_ACCOUNT"}, {"id": "ab5ef340-2de0-4a22-b5fa-2944313cdb71", "createAt": "2023-10-29T10:09:33.301+00:00", "balance": 19153.09154840054, "currency": "MAD", "type": "SAVING_ACCOUNT"}, {"id": "e3db4c25-9f3e-4fd4-b008-8d550aaf37c1", "createAt": "2023-10-29T10:09:33.303+00:00", "balance": 28051.476046871492, "currency": "MAD", "type": "SAVING_ACCOUNT"}, {"id": "04a4d216-6bce-4b96-9c0b-abe8c43c226e", "createAt": "2023-10-29T10:09:33.304+00:00", "balance": 31494.930384471438, "currency": "MAD", "type": "SAVING_ACCOUNT"}]
```

- **Navigateur web**

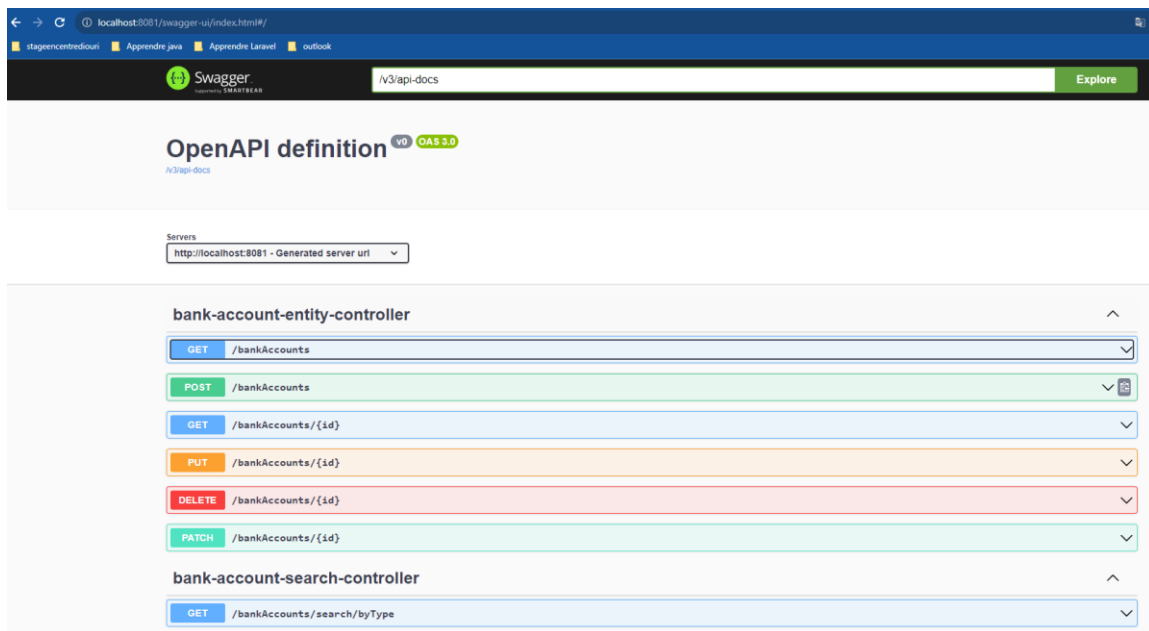
The screenshot shows a web browser with the URL 'localhost:8081/api/bankAccounts'. Below the address bar, there are tabs for 'stagecentreduri', 'Apprendre java', 'Apprendre Laravel', and 'outlook'. The main content area displays a REST client interface with a 'GET' request to 'localhost:8081/api/bankAccounts/cf2d1d7a-5366-4261-baa2-a23a48df6c27'. The response is shown in JSON format, displaying the details of the selected bank account.

```
localhost:8081/api/bankAccounts

stagecentreduri Apprendre java Apprendre Laravel outlook

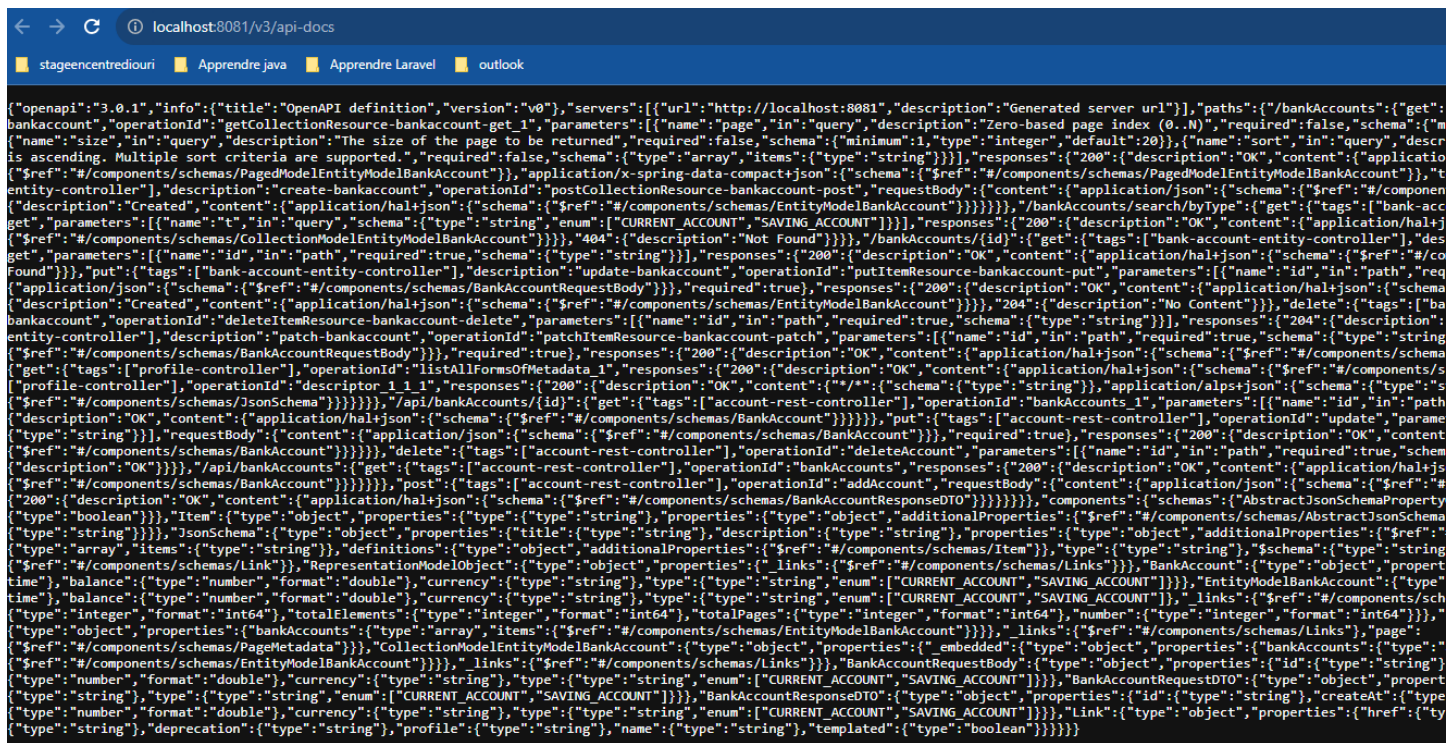
[{"id": "cf2d1d7a-5366-4261-baa2-a23a48df6c27", "createAt": "2023-10-29T10:09:33.233+00:00", "balance": 88943.9969310985, "currency": "MAD", "type": "CURRENT_ACCOUNT"}, {"id": "ab5ef340-2de0-4a22-b5fa-2944313cdb71", "createAt": "2023-10-29T10:09:33.301+00:00", "balance": 19153.09154840054, "currency": "MAD", "type": "SAVING_ACCOUNT"}, {"id": "e3db4c25-9f3e-4fd4-b008-8d550aaf37c1", "createAt": "2023-10-29T10:09:33.303+00:00", "balance": 28051.476046871492, "currency": "MAD", "type": "SAVING_ACCOUNT"}, {"id": "04a4d216-6bce-4b96-9c0b-abe8c43c226e", "createAt": "2023-10-29T10:09:33.304+00:00", "balance": 31494.930384471438, "currency": "MAD", "type": "SAVING_ACCOUNT"}]
```

- **Swagger :**



Cependant, la mention "/v3/api-docs" est une URL typique utilisée pour accéder à la documentation de l'API d'une application.

Le fichier est une définition de l'API au format OpenAPI (également connu sous le nom de Swagger). Il décrit la structure et la documentation d'une API (interface de programmation d'application) Web.



Voici quelques éléments clés de ce fichier :

1. Version d'OpenAPI : La version d'OpenAPI utilisée est "3.0.1."
2. Informations sur l'API : Il y a des informations sur le titre de l'API et sa version.
3. Serveurs : Les serveurs où l'API est déployée sont spécifiés avec des descriptions.
4. Chemins (Paths) : Les différentes routes de l'API sont définies, y compris les opérations possibles (par exemple, GET, POST, PUT, DELETE) pour chaque route.
5. Paramètres : Les paramètres qui peuvent être passés aux opérations sont détaillés, y compris leur nom, leur type, et s'ils sont requis ou non.
6. Réponses : Les réponses attendues pour chaque opération sont spécifiées, avec des descriptions et des schémas de données.
7. Composants (components) : Les schémas de données réutilisables sont définis dans la section "components/schemas."

Ce fichier est essentiellement une documentation décrivant comment interagir avec l'API, quelles routes sont disponibles, quels paramètres peuvent être utilisés, et quelles réponses vous pouvez attendre. Il sert de guide pour les développeurs qui souhaitent utiliser l'API dans leurs applications ou services.

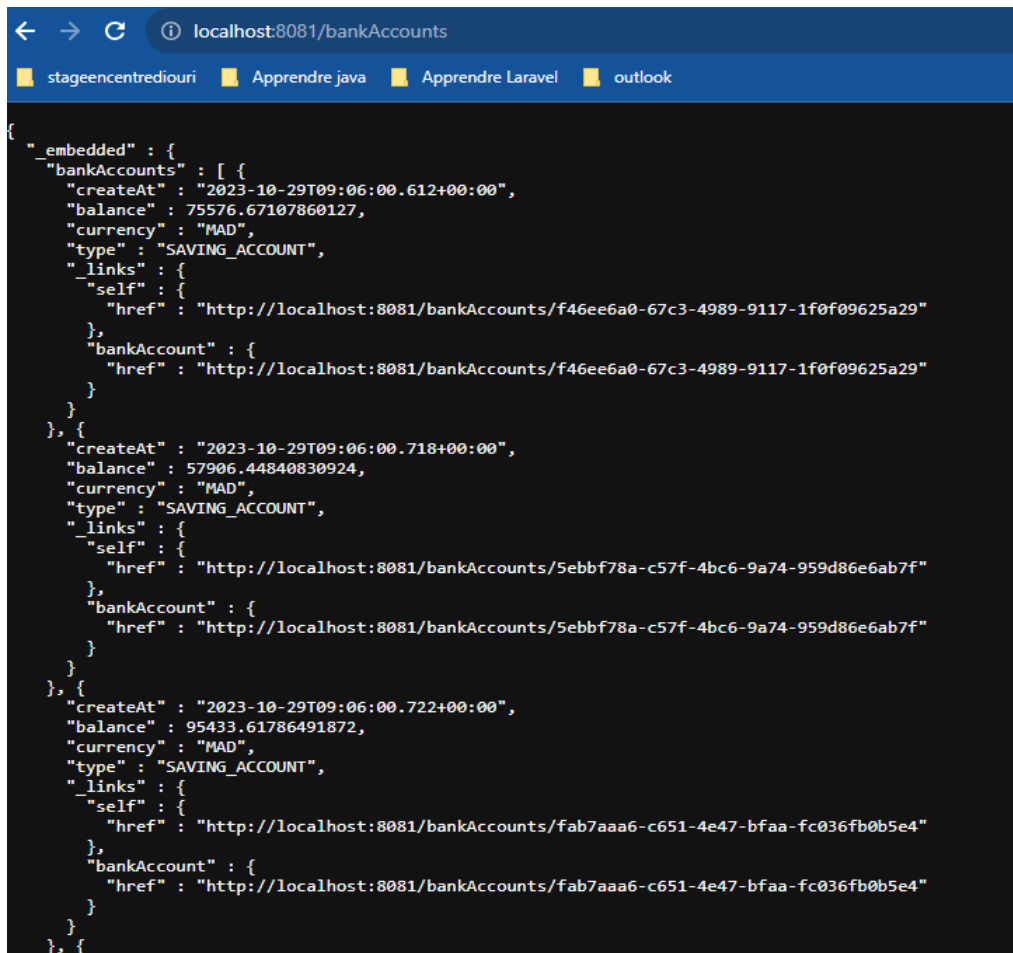
Créer un microservice REST générique

La dépendance `spring-boot-starter-data-rest` de Spring Boot permet de créer facilement des web services génériques en exposant les opérations CRUD (Create, Read, Update, Delete) sur des entités JPA (Java Persistence API) sous forme d'API REST. Voici comment vous pouvez configurer un web service générique en utilisant cette dépendance :

On doit ajouter l'annotation `@RepositoryRestResource` dans le référentiel (repository) JPA qu'on souhaite exposé , dans notre cas `bankAccount`:

```
@RepositoryRestResource
public interface BankAccountRepository extends JpaRepository<BankAccount,String> {
}
```

Au cours du démarrage ,les opérations CRUD pour notre entité sont maintenant disponibles en tant que service web REST. On peut accéder aux opérations via les méthodes HTTP standard telles que GET, POST, PUT et DELETE.

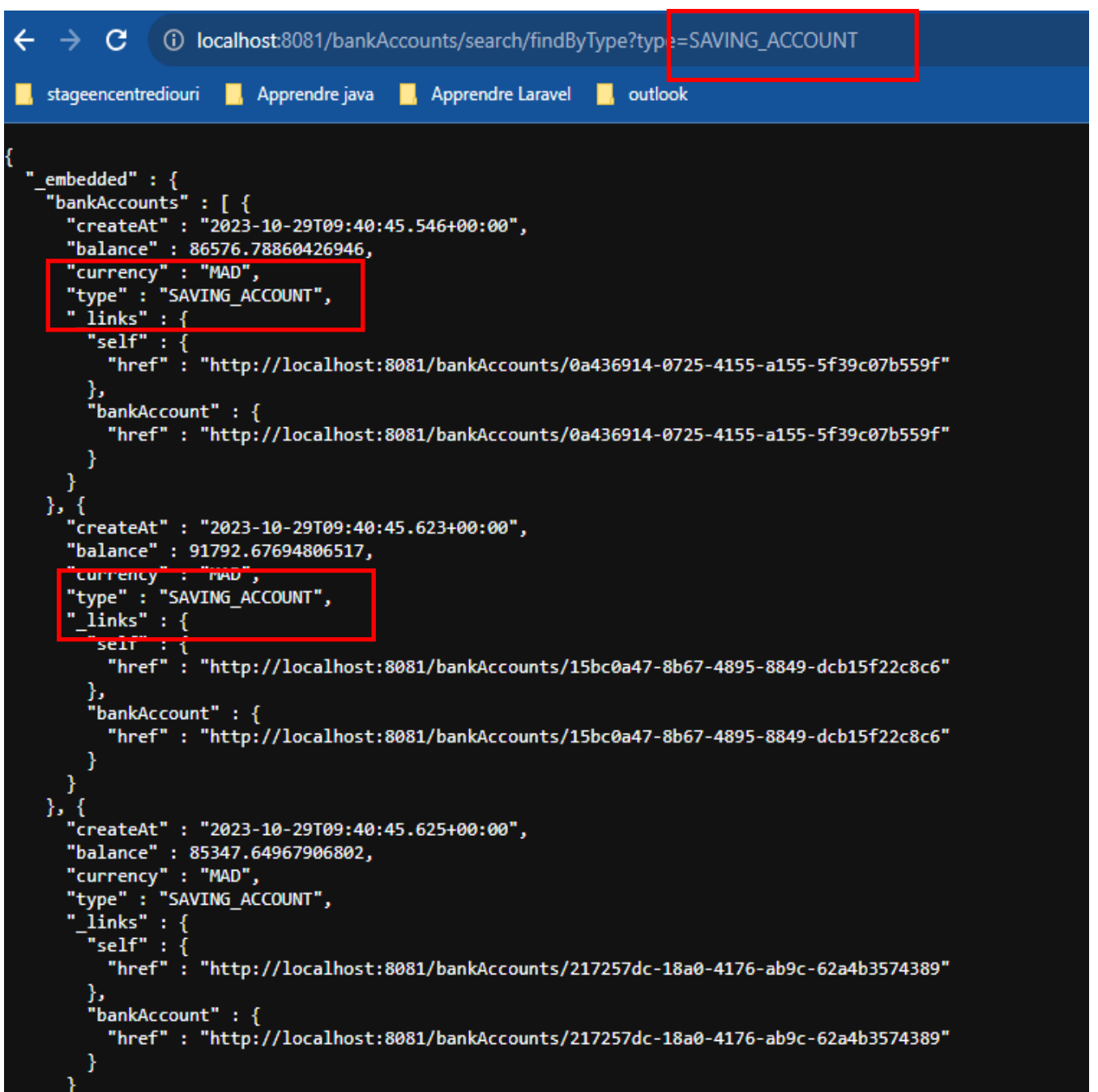


```
{
  "_embedded": {
    "bankAccounts": [ {
      "createAt": "2023-10-29T09:06:00.612+00:00",
      "balance": 75576.67107860127,
      "currency": "MAD",
      "type": "SAVING_ACCOUNT",
      "_links": {
        "self": {
          "href": "http://localhost:8081/bankAccounts/f46ee6a0-67c3-4989-9117-1f0f09625a29"
        }
      },
      "bankAccount": {
        "href": "http://localhost:8081/bankAccounts/f46ee6a0-67c3-4989-9117-1f0f09625a29"
      }
    }, {
      "createAt": "2023-10-29T09:06:00.718+00:00",
      "balance": 57906.44840830924,
      "currency": "MAD",
      "type": "SAVING_ACCOUNT",
      "_links": {
        "self": {
          "href": "http://localhost:8081/bankAccounts/5ebbf78a-c57f-4bc6-9a74-959d86e6ab7f"
        }
      },
      "bankAccount": {
        "href": "http://localhost:8081/bankAccounts/5ebbf78a-c57f-4bc6-9a74-959d86e6ab7f"
      }
    }, {
      "createAt": "2023-10-29T09:06:00.722+00:00",
      "balance": 95433.61786491872,
      "currency": "MAD",
      "type": "SAVING_ACCOUNT",
      "_links": {
        "self": {
          "href": "http://localhost:8081/bankAccounts/fab7aaa6-c651-4e47-bfaa-fc036fb0b5e4"
        }
      },
      "bankAccount": {
        "href": "http://localhost:8081/bankAccounts/fab7aaa6-c651-4e47-bfaa-fc036fb0b5e4"
      }
    }
  ]
}
```

Spring boot data rest ,nous permet de créer une méthode et de l'utiliser ;comme suivant :

```
@RepositoryRestResource
public interface BankAccountRepository extends JpaRepository<BankAccount,String> {
    no usages
    List<BankAccount> findByType(AccountType type);
}
```

Selon le type insérer au niveau de l'URL , on aura des résultats précis :



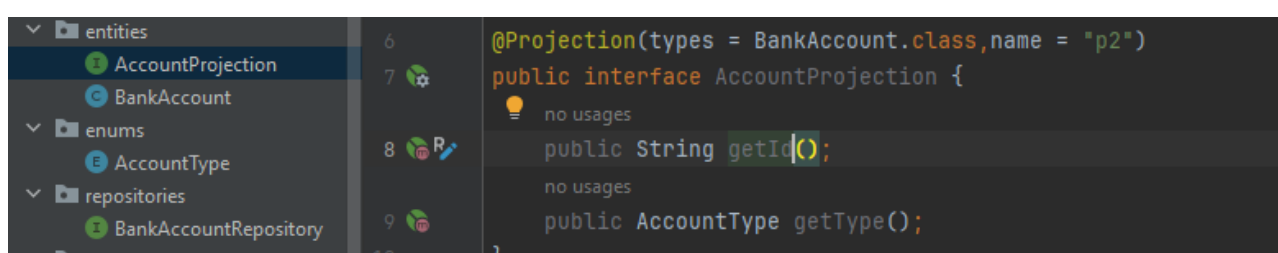
localhost:8081/bankAccounts/search/findByType?type=SAVING_ACCOUNT

```
{
  "_embedded": {
    "bankAccounts": [ {
      "createAt" : "2023-10-29T09:40:45.546+00:00",
      "balance" : 86576.78860426946,
      "currency" : "MAD",
      "type" : "SAVING_ACCOUNT",
      "_links": {
        "self": {
          "href" : "http://localhost:8081/bankAccounts/0a436914-0725-4155-a155-5f39c07b559f"
        },
        "bankAccount" : {
          "href" : "http://localhost:8081/bankAccounts/0a436914-0725-4155-a155-5f39c07b559f"
        }
      }
    }, {
      "createAt" : "2023-10-29T09:40:45.623+00:00",
      "balance" : 91792.67694806517,
      "currency" : "MAD",
      "type" : "SAVING_ACCOUNT",
      "_links": {
        "self": {
          "href" : "http://localhost:8081/bankAccounts/15bc0a47-8b67-4895-8849-dcb15f22c8c6"
        },
        "bankAccount" : {
          "href" : "http://localhost:8081/bankAccounts/15bc0a47-8b67-4895-8849-dcb15f22c8c6"
        }
      }
    }, {
      "createAt" : "2023-10-29T09:40:45.625+00:00",
      "balance" : 85347.64967906802,
      "currency" : "MAD",
      "type" : "SAVING_ACCOUNT",
      "_links": {
        "self": {
          "href" : "http://localhost:8081/bankAccounts/217257dc-18a0-4176-ab9c-62a4b3574389"
        },
        "bankAccount" : {
          "href" : "http://localhost:8081/bankAccounts/217257dc-18a0-4176-ab9c-62a4b3574389"
        }
      }
    }
  ]
}
```

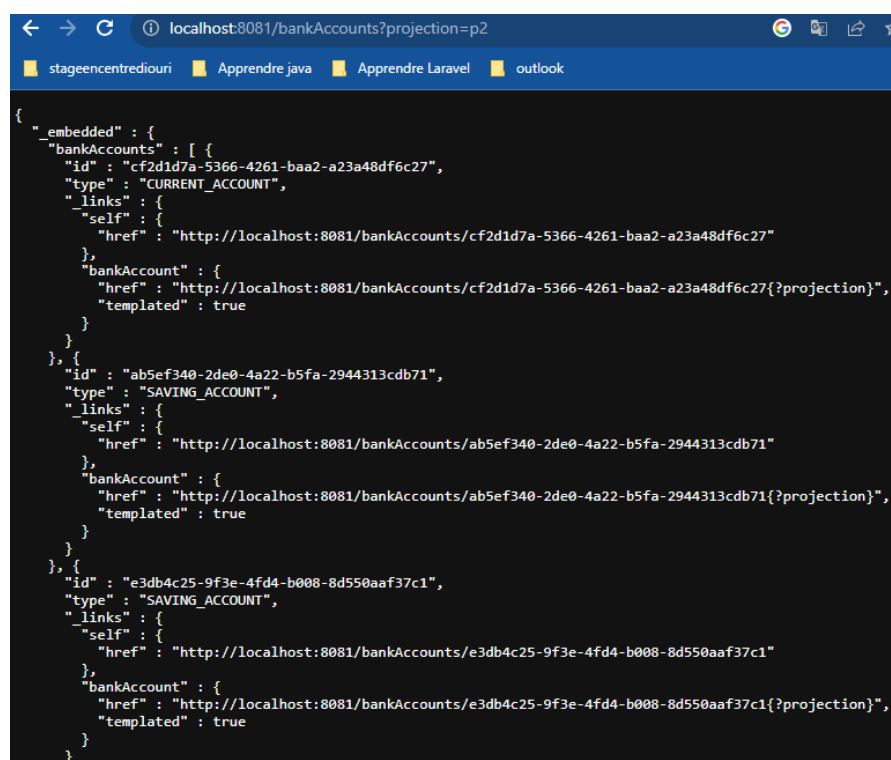
Les projections en utilisant spring-boot-starter-data-rest vous permettent de personnaliser la forme des données renvoyées par votre API REST en fonction de ce dont vous avez besoin pour une requête spécifique, au lieu de renvoyer l'ensemble complet d'attributs d'une entité. Cela peut être utile pour réduire la charge de données, améliorer les performances et rendre les réponses plus adaptées aux besoins de l'application cliente. Vous pouvez utiliser des projections pour inclure ou exclure des attributs, renommer des attributs, ou même agréger des données à partir de plusieurs entités.

Voici comment vous pouvez utiliser les projections en utilisant spring-boot-starter-data-rest :

Vous devez créer une interface pour définir la projection. Cette interface peut inclure des méthodes de getter correspondant aux attributs que vous souhaitez inclure dans la projection.



Pour avoir le résultat , vous pouvez spécifier la projection dans l'URL. Par exemple :



Conclusion :

En résumé, les microservices REST sont une approche agile et évolutive pour la construction d'applications modernes. Ils offrent une architecture modulaire, flexible et découplée, favorisant la collaboration et la réactivité aux besoins changeants du marché. Cependant, leur mise en œuvre nécessite une planification minutieuse et une attention continue à la qualité du code. Les microservices REST sont un atout précieux pour les entreprises cherchant à offrir des services interopérables et à relever les défis de la connectivité dans le monde numérique d'aujourd'hui.

