

SYSTÈMES DISTRIBUÉS

MASTER INTELLIGENCE
ARTIFICIELLE ET ANALYSE
DES DONNÉES

RÉALISÉE PAR :

ANOADA NOHAYLA

ENCADRÉ PAR :

M. MOHAMED YOUSSEFI



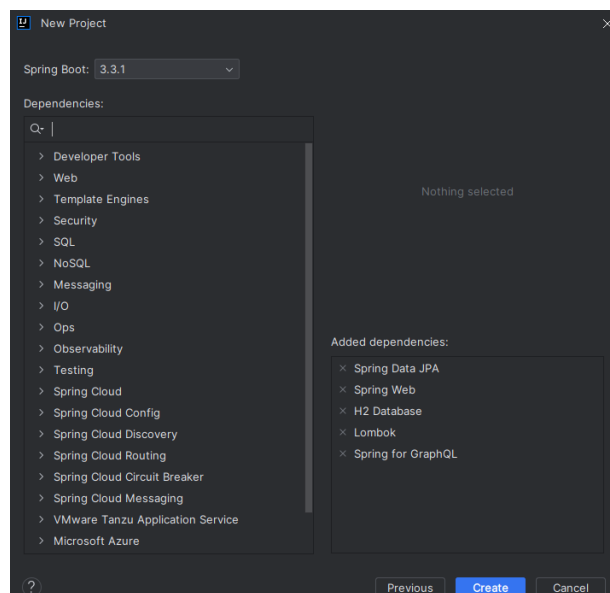
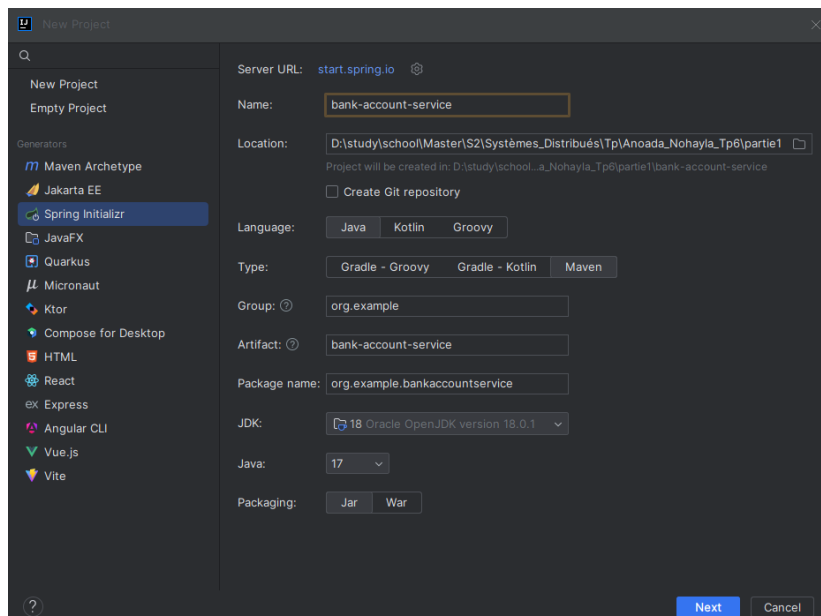
ACTIVITÉ PRATIQUE N° 6

ARCHITECTURES MICRO-SERVICES

PARTIE 1 : DÉVELOPPER UN MICRO-SERVICE

1. MICRO SERVICE AVEC WEB SERVICE RESTFUL

1. Creation de project maven



la création d'un nouveau projet Spring Boot de type Maven avec des dépendances Spring Data, Spring Web, H2, Lombok et Spring for GraphQL

2. Création d'entité

```
@Entity
@Data @NoArgsConstructor @AllArgsConstructor @Builder
public class BankAccount
{
    @Id
    private String id;
    private Date createdAt;
    private Double balance;
    private String currency;
    @Enumerated(EnumType.STRING)
    private AccountType type;
}
```

La classe BankAccount est une entité JPA avec des champs pour l'identifiant, la date de création, le solde, la devise et le type de compte. Lombok simplifie les getters, setters, constructeurs et le pattern builder. Le type de compte est une énumération stockée comme une chaîne.

3. Création de enums

```
public enum AccountType
{
    1 usage
    CURRENT_ACCOUNT, SAVING_ACCOUNT
}
```

Le type de compte est défini par l'énumération AccountType avec les valeurs CURRENT_ACCOUNT et SAVING_ACCOUNT, stockée comme une chaîne dans la base de données.

4. configuration de l'application

```
spring.application.name=bank-account-service
spring.datasource.url=jdbc:h2:mem:account-db
spring.h2.console.enabled=true
server.port=8081
```

La configuration de l'application Spring Boot inclut une base de données en mémoire H2 avec l'URL `jdbc:h2:mem:account-db`, active la console H2, et définit le port du serveur sur 8081.

5. Faire un test (insertion des comptes)

```
@SpringBootApplication
public class BankAccountServiceApplication
{
    public static void main(String[] args)
    {
        SpringApplication.run(BankAccountServiceApplication.class, args);
    }

    @Bean
    CommandLineRunner start(BankAccountRepository bankAccountRepository){
        return args -> {
            for(int i=0;i<10;i++){
                BankAccount bankAccount= BankAccount.builder()
                    .id(UUID.randomUUID().toString())
                    .type(Math.random()>0.5? AccountType.CURRENT_ACCOUNT:AccountType.SAVING_ACCOUNT)
                    .balance(10000+Math.random()*90000)
                    .createdAt(new Date())
                    .currency("MAD")
                    .build();
                bankAccountRepository.save(bankAccount);
            }
        };
    }
}
```

La classe `BankAccountServiceApplication` initialise la application Spring Boot et génère 10 comptes bancaires aléatoires à l'aide de la méthode `CommandLineRunner`. Chaque compte est créé avec un identifiant unique, un type aléatoire (compte courant ou compte épargne), un solde aléatoire, une date de création, et une devise fixée à "MAD", puis sauvegardé dans le dépôt `BankAccountRepository`.

The screenshot shows the H2 Console web interface. On the left, a tree view displays the database structure: jdbc:h2:mem:account-db, BANK_ACCOUNT (with columns BALANCE, TYPE, CREATED_AT, CURRENCY, ID), Indexes, INFORMATION_SCHE, and Users. The main area shows a SQL statement: `SELECT * FROM BANK_ACCOUNT`. Below the statement, the results are displayed as a table with 10 rows and 5 columns: BALANCE, TYPE, CREATED_AT, CURRENCY, and ID. The table contains 10 rows of data. At the bottom, it indicates "(10 rows, 7 ms)" and an "Edit" button.

BALANCE	TYPE	CREATED_AT	CURRENCY	ID
56571.06053073246	1	2024-07-08 15:13:33.993	MAD	5b93a8ae-dbab-4a45-945f-4781975df235
19194.8777814359	0	2024-07-08 15:13:34.219	MAD	f976051f-88ec-41ac-848b-15265685b979
64958.98933114584	0	2024-07-08 15:13:34.223	MAD	d5bb4b65-acb6-4a0b-a58e-d6a1268a9d47
14906.115650847854	1	2024-07-08 15:13:34.226	MAD	635108f9-a48b-4d3c-88a9-3447f9651466
23169.46726210676	1	2024-07-08 15:13:34.229	MAD	01af33a9-742c-4d6a-bd5d-086e54a082dd
74736.30389574275	1	2024-07-08 15:13:34.233	MAD	132d0861-80c8-4cac-9b78-8869e26ecaf1
67634.39253334631	1	2024-07-08 15:13:34.236	MAD	2a872a8d-a632-4037-abec-a4d31064ca2e
72006.85846944881	0	2024-07-08 15:13:34.24	MAD	518c1c8d-725d-4da8-ab08-21cfd7680f6c
74393.5428653998	1	2024-07-08 15:13:34.243	MAD	aab258a3-4c5b-41e8-94d3-4a7b4ca57524
82679.3810282873	0	2024-07-08 15:13:34.247	MAD	0dec3658-8a5d-419e-9bee-37944537a0e8

6. Création d'interface repository

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

Interface `BankAccountRepository` étend `JpaRepository<BankAccount, String>` pour interagir avec la base de données.

7. Contrôleur REST

```
@RestController
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 %s not found",id)));
}

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

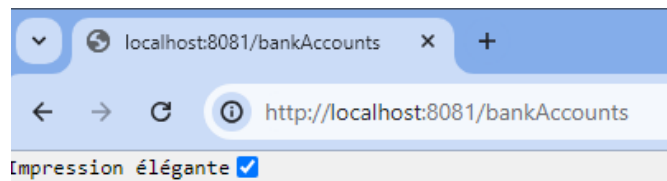
@PutMapping (Ⓜ"/bankAccounts/{id}")
public BankAccount update(@PathVariable String id,@RequestBody BankAccount bankAccount)
{
    BankAccount account=bankAccountRepository.findById(id).orElseThrow();
    if(bankAccount.getBalance()!=null) account.setBalance(bankAccount.getBalance());
    if(bankAccount.getCreatedAt()!=null)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

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

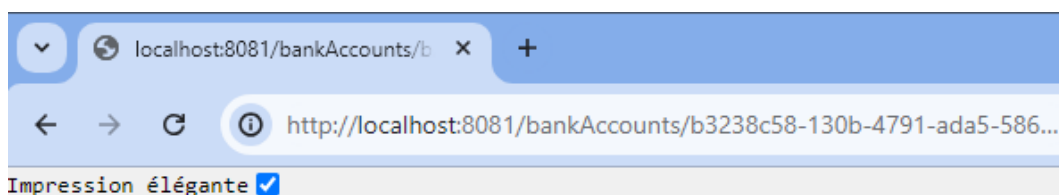
```

le contrôleur REST est conçu pour interagir avec un BankAccountRepository, qui devrait être une interface BankAccountRepository pour effectuer les opérations CRUD (Create, Read, Update, Delete) sur les entités BankAccount dans une base de données.

8. Tester l'application avec navigateur



```
[
  {
    "id": "698226e0-202b-4961-8903-0f421ec314d0",
    "createdAt": "2024-07-08T15:07:30.129+00:00",
    "balance": 19101.7926754692,
    "currency": "MAD",
    "type": "CURRENT_ACCOUNT"
  },
  {
    "id": "b99cacdd-faf6-4c30-9b3d-a842cb46a230",
    "createdAt": "2024-07-08T15:07:30.314+00:00",
    "balance": 76601.7293061234,
    "currency": "MAD",
    "type": "CURRENT_ACCOUNT"
  },
  {
    "id": "b3238c58-130b-4791-ada5-586b1aaa7f1f",
    "createdAt": "2024-07-08T15:07:30.319+00:00",
    "balance": 21402.4705927567,
    "currency": "MAD",
    "type": "CURRENT_ACCOUNT"
  },
  {
    "id": "1011a55b-6fcd-4582-ab18-7d32a7d73e84",
    "createdAt": "2024-07-08T15:07:30.322+00:00",
    "balance": 56747.796685566,
    "currency": "MAD",
    "type": "SAVING_ACCOUNT"
  },
  {
    "id": "0c558960-c729-4245-b31b-84c268a56e40",
    "createdAt": "2024-07-08T15:07:30.327+00:00",
    "balance": 33624.4045878484,
    "currency": "MAD",
    "type": "CURRENT_ACCOUNT"
  }
],
```



```
{
  "id": "b3238c58-130b-4791-ada5-586b1aaa7f1f",
  "createdAt": "2024-07-08T15:07:30.319+00:00",
  "balance": 21402.4705927567,
  "currency": "MAD",
  "type": "CURRENT_ACCOUNT"
}
```

9. Tester l'application avec Postman

The first screenshot shows a GET request to `http://localhost:8081/bankAccounts`. The response is a JSON array of three bank accounts. The second screenshot shows a GET request to `http://localhost:8081/bankAccounts/65512516-a893-444f-88aa-59af47dcae8c`. The response is a JSON object representing a single bank account. The third screenshot shows a PUT request to the same URL as the second screenshot. The request body is a JSON object with fields for balance, type, and currency.

GET http://localhost:8081/bankAccounts

Status: 200 OK Time: 7.11 s Size: 1.72 KB

```
1 [{"id": "92da1bab-9f73-4dda-a3e6-550b57ed8f30",
2   "createdAt": "2024-07-08T15:52:24.536+00:00",
3   "balance": 81704.31196538963,
4   "currency": "MAD",
5   "type": "SAVING_ACCOUNT"},
6   {"id": "8d4bc9cd-90fd-4fb4-825d-18c06cf51699",
7   "createdAt": "2024-07-08T15:52:24.917+00:00",
8   "balance": 22279.816289114158,
9   "currency": "MAD",
10  "type": "CURRENT_ACCOUNT"},
11  {"id": "65512516-a893-444f-88aa-59af47dcae8c",
12  "createdAt": "2024-07-08T15:52:24.920+00:00",
13  "balance": 82162.46857180166,
14  "currency": "MAD",
15  "type": "SAVING_ACCOUNT"}]
```

GET http://localhost:8081/bankAccounts/65512516-a893-444f-88aa-59af47dcae8c

Status: 200 OK Time: 8.36 s Size: 322 B

```
1 {"id": "65512516-a893-444f-88aa-59af47dcae8c",
2   "createdAt": "2024-07-08T15:52:24.920+00:00",
3   "balance": 82162.46857180166,
4   "currency": "MAD",
5   "type": "SAVING_ACCOUNT"}
```

PUT http://localhost:8081/bankAccounts/65512516-a893-444f-88aa-59af47dcae8c

Body: raw

```
1 {
2   "balance": 8000,
3   "type": "SAVING_ACCOUNT",
4   "currency": "USD"
5 }
```


10. Ajouter dependency spring boot openapi doc maven

```
<dependency>
  <groupId>org.springdoc</groupId>
  <artifactId>springdoc-openapi-ui</artifactId>
  <version>1.8.0</version>
</dependency>
```

Facilite la génération automatique de la documentation API à partir de vos contrôleurs REST

11. Ajouter dependency de service REST et Annotations dans AccountRestController et BankAccountRepository

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-rest</artifactId>
</dependency>
```

Simplifie la création de services REST complets et conformes aux principes de RESTful à partir des repositories Spring Data JPA.

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

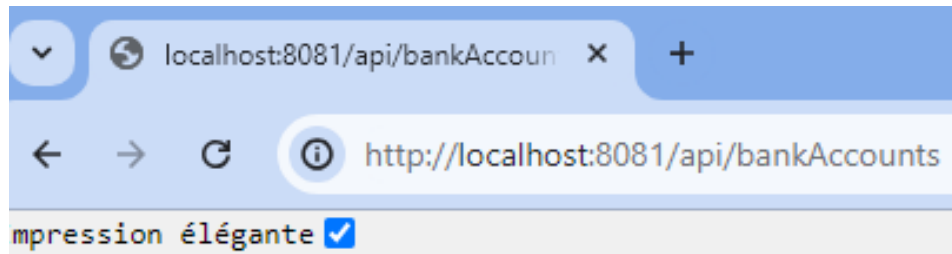
@RequestMapping("/api") définit le chemin de base pour tous les endpoints de classe de contrôleur

```
@RestController
@RequestMapping("/api")
public class AccountRestController {
}
```

@RepositoryRestResource est utilisée pour personnaliser et configurer les ressources de repository Spring Data REST

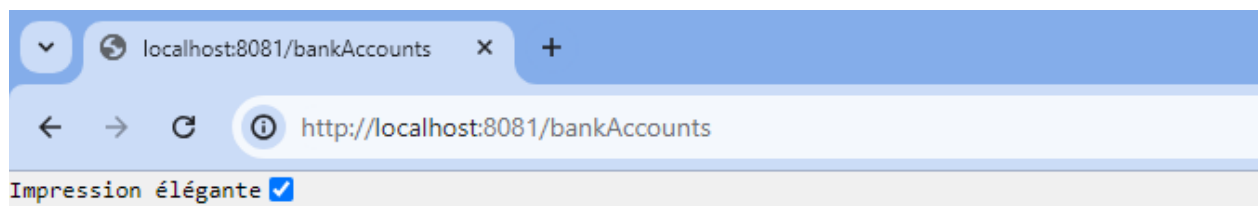
12. la gestion de services web RESTful.

a. APIs REST



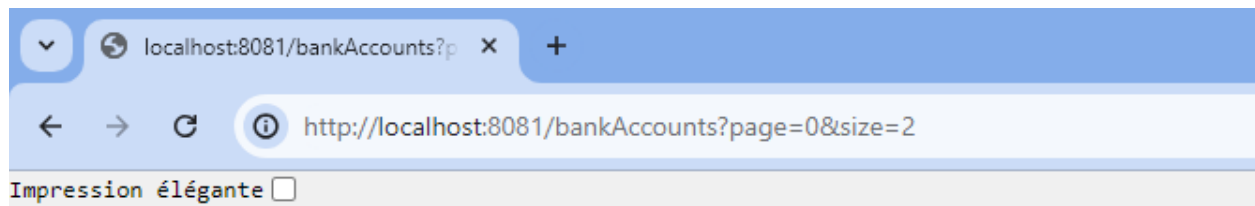
```
{
  "id": "9f761259-5c37-44f2-9d0f-5a34b7ff9ef3",
  "createdAt": "2024-07-08T17:21:32.692+00:00",
  "balance": 88565.3834768793,
  "currency": "MAD",
  "type": "SAVING_ACCOUNT"
},
{
  "id": "e135aeed-cdcd-4968-8cf5-290ff46cad1a",
  "createdAt": "2024-07-08T17:21:32.918+00:00",
  "balance": 42957.4334420347,
  "currency": "MAD",
  "type": "SAVING_ACCOUNT"
},
{
  "id": "309f500d-e2b2-4f11-aaa3-5cf92a1bd08a",
  "createdAt": "2024-07-08T17:21:32.921+00:00",
  "balance": 26695.1947922021,
  "currency": "MAD",
  "type": "SAVING_ACCOUNT"
},
{
  "id": "f8e67177-18fe-4bbf-9a7d-9cab734d0e51",
  "createdAt": "2024-07-08T17:21:32.925+00:00",
  "balance": 68591.6699147029,
  "currency": "MAD",
  "type": "CURRENT_ACCOUNT"
},
}
```

b. Spring Data REST



```
{
  "_embedded": {
    "bankAccounts": [
      {
        "createdAt": "2024-07-08T17:21:32.692+00:00",
        "balance": 88565.3834768793,
        "currency": "MAD",
        "type": "SAVING_ACCOUNT",
        "_links": {
          "self": {
            "href": "http://localhost:8081/bankAccounts/9f761259-5c37-44f2-9d0f-5a34b7ff9ef3"
          },
          "bankAccount": {
            "href": "http://localhost:8081/bankAccounts/9f761259-5c37-44f2-9d0f-5a34b7ff9ef3"
          }
        }
      },
      {
        "createdAt": "2024-07-08T17:21:32.918+00:00",
        "balance": 42957.4334420347,
        "currency": "MAD",
        "type": "SAVING_ACCOUNT",
        "_links": {
          "self": {
            "href": "http://localhost:8081/bankAccounts/e135aeed-cdcd-4968-8cf5-290ff46cad1a"
          },
          "bankAccount": {
            "href": "http://localhost:8081/bankAccounts/e135aeed-cdcd-4968-8cf5-290ff46cad1a"
          }
        }
      },
      {
        "createdAt": "2024-07-08T17:21:32.921+00:00",
        "balance": 26695.1947922021,
        "currency": "MAD",
        "type": "SAVING_ACCOUNT",
        "_links": {
          "self": {
            "href": "http://localhost:8081/bankAccounts/309f500d-e2b2-4f11-aaa3-5cf92a1bd08a"
          },
          "bankAccount": {
            "href": "http://localhost:8081/bankAccounts/309f500d-e2b2-4f11-aaa3-5cf92a1bd08a"
          }
        }
      }
    ]
  }
}
```

- **Pagination avec Spring Data REST**

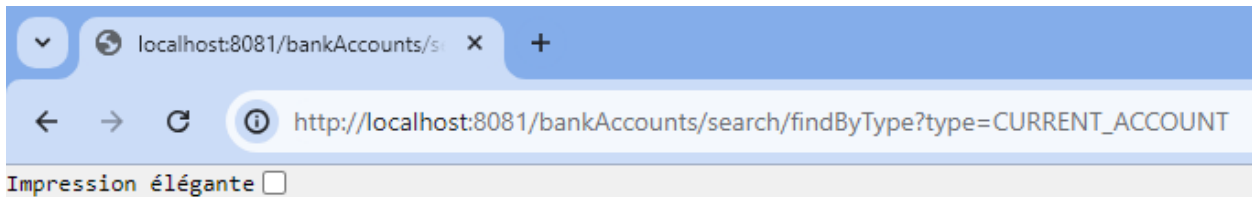


```
{
  "_embedded" : {
    "bankAccounts" : [ {
      "createdAt" : "2024-07-08T17:30:15.626+00:00",
      "balance" : 45160.3253563821,
      "currency" : "MAD",
      "type" : "CURRENT_ACCOUNT",
      "_links" : {
        "self" : {
          "href" : "http://localhost:8081/bankAccounts/fa2b522c-adc6-4133-9db9-8f56b72a1a73"
        },
        "bankAccount" : {
          "href" : "http://localhost:8081/bankAccounts/fa2b522c-adc6-4133-9db9-8f56b72a1a73"
        }
      }
    }, {
      "createdAt" : "2024-07-08T17:30:15.890+00:00",
      "balance" : 13352.384527955734,
      "currency" : "MAD",
      "type" : "CURRENT_ACCOUNT",
      "_links" : {
        "self" : {
          "href" : "http://localhost:8081/bankAccounts/68c53b68-4ee3-4819-930e-ff25b02aba97"
        },
        "bankAccount" : {
          "href" : "http://localhost:8081/bankAccounts/68c53b68-4ee3-4819-930e-ff25b02aba97"
        }
      }
    }
  ]
}, {
  "_links" : {
    "first" : {
      "href" : "http://localhost:8081/bankAccounts?page=0&size=2"
    },
    "self" : {
      "href" : "http://localhost:8081/bankAccounts?page=0&size=2"
    },
    "next" : {
      "href" : "http://localhost:8081/bankAccounts?page=1&size=2"
    }
  }
}, .. .. }
```

La pagination dans Spring Data REST permet de gérer efficacement les résultats de requêtes REST en divisant les réponses en pages.

- Accéder aux méthodes de repository avec Spring Data REST

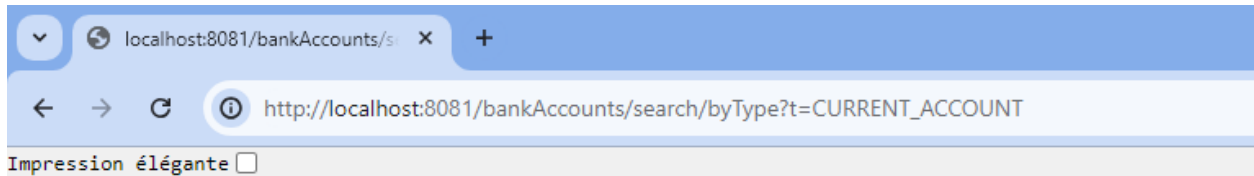
```
@RepositoryRestResource
public interface BankAccountRepository extends JpaRepository<BankAccount,String>
{
    no usages
    List<BankAccount> findByType(AccountType type);
}
```



```
{
  "_embedded" : {
    "bankAccounts" : [ {
      "createdAt" : "2024-07-08T17:30:15.626+00:00",
      "balance" : 45160.3253563821,
      "currency" : "MAD",
      "type" : "CURRENT_ACCOUNT",
      "_links" : {
        "self" : {
          "href" : "http://localhost:8081/bankAccounts/fa2b522c-adc6-4133-9db9-8f56b72a1a73"
        },
        "bankAccount" : {
          "href" : "http://localhost:8081/bankAccounts/fa2b522c-adc6-4133-9db9-8f56b72a1a73"
        }
      }
    }, {
      "createdAt" : "2024-07-08T17:30:15.890+00:00",
      "balance" : 13352.384527955734,
      "currency" : "MAD",
      "type" : "CURRENT_ACCOUNT",
      "_links" : {
        "self" : {
          "href" : "http://localhost:8081/bankAccounts/68c53b68-4ee3-4819-930e-ff25b02aba97"
        },
        "bankAccount" : {
          "href" : "http://localhost:8081/bankAccounts/68c53b68-4ee3-4819-930e-ff25b02aba97"
        }
      }
    }, {
      "createdAt" : "2024-07-08T17:30:15.908+00:00",
      "balance" : 49582.89406037885,
      "currency" : "MAD",
      "type" : "CURRENT_ACCOUNT",
      "_links" : {
        "self" : {
          "href" : "http://localhost:8081/bankAccounts/82423425-b1d0-4be3-9988-afa2012384ec"
        },
        "bankAccount" : {
          "href" : "http://localhost:8081/bankAccounts/82423425-b1d0-4be3-9988-afa2012384ec"
        }
      }
    }
  ]
}
```

Spring Data REST automatise l'exposition des méthodes de repository Spring Data JPA en tant que services RESTful. on peut modifier la notation des méthodes par default.

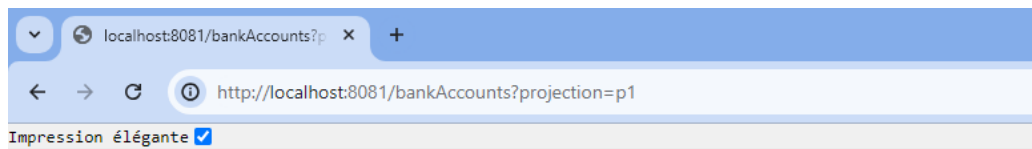
```
@RepositoryRestResource
public interface BankAccountRepository extends JpaRepository<BankAccount,String>
{
    no usages
    @RestResource(path = "/byType")
    List<BankAccount> findByType(@Param("t") AccountType type);
}
```



```
{
  "_embedded" : {
    "bankAccounts" : [ {
      "createdAt" : "2024-07-08T17:59:35.714+00:00",
      "balance" : 78923.91075207549,
      "currency" : "MAD",
      "type" : "CURRENT_ACCOUNT",
      "_links" : {
        "self" : {
          "href" : "http://localhost:8081/bankAccounts/1520da83-9a80-4e97-81cf-17118d6e5138"
        },
        "bankAccount" : {
          "href" : "http://localhost:8081/bankAccounts/1520da83-9a80-4e97-81cf-17118d6e5138{?projection}",
          "templated" : true
        }
      }
    }, {
      "createdAt" : "2024-07-08T17:59:35.720+00:00",
      "balance" : 41102.22357441115,
      "currency" : "MAD",
      "type" : "CURRENT_ACCOUNT",
      "_links" : {
        "self" : {
          "href" : "http://localhost:8081/bankAccounts/b5c61677-8bb4-4d25-9673-266d6e06e227"
        },
        "bankAccount" : {
          "href" : "http://localhost:8081/bankAccounts/b5c61677-8bb4-4d25-9673-266d6e06e227{?projection}",
          "templated" : true
        }
      }
    }, {
      "createdAt" : "2024-07-08T17:59:35.730+00:00",
      "balance" : 99057.74108493785,
      "currency" : "MAD",
      "type" : "CURRENT_ACCOUNT",
      "_links" : {
        "self" : {
          "href" : "http://localhost:8081/bankAccounts/1520da83-9a80-4e97-81cf-17118d6e5138"
        },
        "bankAccount" : {
          "href" : "http://localhost:8081/bankAccounts/1520da83-9a80-4e97-81cf-17118d6e5138{?projection}",
          "templated" : true
        }
      }
    }
  ]
}
```

- Utilisation des projections avec Spring Data REST

```
@Projection(types = BankAccount.class,name = "p1")
public interface AccountProjection
{
    no usages
    public String getId();
    no usages
    public AccountType getType();
}
```

The screenshot shows a web browser with the address bar displaying `http://localhost:8081/bankAccounts?projection=p1`. The response body is a JSON object representing a list of bank accounts with specific fields projected.

```
{
  "_embedded": {
    "bankAccounts": [
      {
        "id": "beb16f08-3f39-4836-b6c9-f6454ef56b36",
        "type": "CURRENT_ACCOUNT",
        "_links": {
          "self": {
            "href": "http://localhost:8081/bankAccounts/beb16f08-3f39-4836-b6c9-f6454ef56b36"
          }
        },
        "bankAccount": {
          "href": "http://localhost:8081/bankAccounts/beb16f08-3f39-4836-b6c9-f6454ef56b36{?projection}",
          "templated": true
        }
      },
      {
        "id": "074f6fb1-a60a-4306-95d5-0c34b390feeb",
        "type": "CURRENT_ACCOUNT",
        "_links": {
          "self": {
            "href": "http://localhost:8081/bankAccounts/074f6fb1-a60a-4306-95d5-0c34b390feeb"
          }
        },
        "bankAccount": {
          "href": "http://localhost:8081/bankAccounts/074f6fb1-a60a-4306-95d5-0c34b390feeb{?projection}",
          "templated": true
        }
      }
    ]
  },
  {
    "id": "8fdae9d9-5501-4f2c-a27a-25736c54e58f",
    "type": "SAVING_ACCOUNT",
    "_links": {
      "self": {
        "href": "http://localhost:8081/bankAccounts/8fdae9d9-5501-4f2c-a27a-25736c54e58f"
      }
    },
    "bankAccount": {
      "href": "http://localhost:8081/bankAccounts/8fdae9d9-5501-4f2c-a27a-25736c54e58f{?projection}",
      "templated": true
    }
  }
}
```

Les projections dans Spring Data REST permettent de personnaliser les réponses des requêtes REST en ne renvoyant qu'une partie spécifique des données d'une entité.

13. Définition des DTOs

```
@Data @NoArgsConstructor @AllArgsConstructor @Builder
public class BankAccountRequestDTO {

    private double balance;
    private String currency;
    private AccountType type;
}
```

Class `BankAccountRequestDTO` utilisé pour encapsuler les données d'un compte bancaire qui sont envoyées en tant que corps de requête lors de la création ou de la mise à jour d'un compte.

```

@Data @NoArgsConstructor @AllArgsConstructor @Builder
public class BankAccountResponseDTO {

    private String id;
    private Date createdAt;
    private double balance;
    private String currency;
    private AccountType type;
}

```

Class BankAccountResponseDTO utilisé pour encapsuler les données d'un compte bancaire qui sont renvoyées en réponse à une requête.

14. Définition de service

```

public interface AccountService
{
    1 usage 1 implementation
    public BankAccountResponseDTO addAccount(BankAccountRequestDTO bankAccountRequestDTO);
}

```

```

@Controller
public class AccountMapper
{
    no usages
    public BankAccountResponseDTO fromBankAccount(BankAccount bankAccount)
    {
        BankAccountResponseDTO bankAccountResponseDTO=new BankAccountResponseDTO();
        BeanUtils.copyProperties(bankAccount, bankAccountResponseDTO);
        return bankAccountResponseDTO;
    }
}

```

```

@RestController
@RequestMapping("/api")
public class AccountRestController {
    6 usages
    private BankAccountRepository bankAccountRepository;
    2 usages
    private AccountService accountService;
    no usages
    private AccountMapper accountMapper;
}

```

```

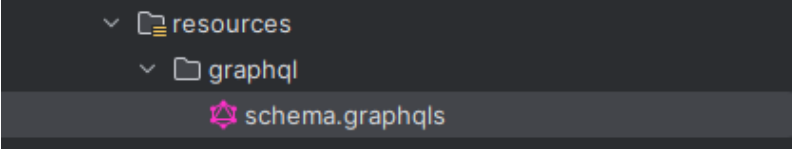
@Service
@Transactional
public class AccountServiceImpl implements AccountService
{
    @Autowired
    private BankAccountRepository bankAccountRepository;
    1 usage
    private AccountMapper accountMapper;
    1 usage
    @Override
    public BankAccountResponseDTO addAccount(BankAccountRequestDTO bankAccountDTO)
    {
        BankAccount bankAccount= BankAccount.builder()
            .id(UUID.randomUUID().toString())
            .createdAt(new Date())
            .balance(bankAccountDTO.getBalance())
            .type(bankAccountDTO.getType())
            .currency(bankAccountDTO.getCurrency())
            .build();
        BankAccount saveBankAccount = bankAccountRepository.save(bankAccount);
        BankAccountResponseDTO bankAccountResponseDTO = accountMapper.fromBankAccount(saveBankAccount);
        return bankAccountResponseDTO;
    }
}

```

implémente le service AccountService pour l'ajout de comptes bancaires. Le service utilise BankAccountRequestDTO pour recevoir les données de création, telles que le solde et la devise. Après sauvegarde via BankAccountRepository, les détails du compte sont convertis en BankAccountResponseDTO à l'aide de AccountMapper, assurant une réponse structurée et efficace dans API.

2. MICRO SERVICE AVEC WEB SERVICE GRAPHQL

1. Schéma GraphQL pour la Gestion des Comptes Bancaires et des Clients



```
type Query {
  accountList: [BankAccount] ,
  bankAccountById(id:String) : BankAccount ,
  customers : [Customer]
}

type Mutation {
  addAccount(bankAccount: BankAccountDTO): BankAccount,
  updateAccount(id: String , bankAccount: BankAccountDTO): BankAccount,
  deleteAccount(id: String): Boolean
}

input BankAccountDTO {
  balance: Float,
  currency: String,
  type: String
}

type Customer{
  id :ID ,
  name :String ,
  bankAccounts : [BankAccount]
}

type BankAccount {
  id: String,
  createdAt: Float,
  balance: Float,
  currency: String,
  type: String ,
  customer : Customer
}
```

schéma GraphQL définit les types et opérations nécessaires pour gérer les comptes bancaires et les clients. Il inclut des requêtes (Query) pour récupérer les listes de comptes et de clients, ainsi que des mutations (Mutation) pour

ajouter, mettre à jour et supprimer des comptes bancaires. Les types `BankAccount` et `Customer` sont définis pour représenter les entités principales, et un type d'entrée (`BankAccountDTO`) est utilisé pour les mutations.

2. Contrôleur GraphQL

```
@Controller
public class BankAccountGraphQLController
{
    @Autowired
    private BankAccountRepository bankAccountRepository;

    @Autowired
    private AccountService accountService;
    @Autowired
    private CustomerRepository customerRepository;

    @QueryMapping
    public List<Customer> customers() { return customerRepository.findAll(); }

    @MutationMapping
    public BankAccountResponseDTO addAccount(@Argument BankAccountRequestDTO bankAccount) {
        return accountService.addAccount(bankAccount);
    }

    @MutationMapping
    public BankAccountResponseDTO updateAccount(@Argument String id, @Argument BankAccountRequestDTO bankAccount) {
        return accountService.updateAccount(id, bankAccount);
    }

    @MutationMapping
    public Boolean deleteAccount(@Argument String id) {
        bankAccountRepository.deleteById(id);
        return true;
    }

    @QueryMapping
    public List<BankAccount> accountList() { return bankAccountRepository.findAll(); }

    @QueryMapping
    public BankAccount bankAccountById(@Argument String id) {
        return bankAccountRepository.findById(id)
            .orElseThrow(() -> new RuntimeException(String.format("Account %s not found", id)));
    }
}
```

`BankAccountGraphQLController` implémente les opérations GraphQL définies dans le schéma. Il utilise les annotations `@QueryMapping` et `@MutationMapping` pour définir les requêtes et mutations. Les méthodes permettent de récupérer la liste des clients, ajouter, mettre à jour et supprimer des comptes bancaires, et récupérer des comptes spécifiques.

3. Entité Customer

```
@Entity
@NoArgsConstructor @AllArgsConstructor @Data @Builder
public class Customer {

    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    @OneToMany(mappedBy = "customer")
    @JsonProperty(access = JsonProperty.Access.WRITE_ONLY)
    private List<BankAccount> bankAccounts;
}
```

Entité Customer représente un client dans la base de données. Chaque client peut avoir plusieurs comptes bancaires, définis par la relation @OneToMany. L'annotation @JsonProperty(access = JsonProperty.Access.WRITE_ONLY) est utilisée pour éviter que la liste des comptes bancaires ne soit incluse dans les réponses JSON, tout en permettant sa modification lors des opérations d'écriture.

4. Gestion des Exceptions dans GraphQL

```
@Component
public class CustomDataFetcherExceptionHandler extends DataFetcherExceptionHandlerAdapter
{
    no usages
    @Override
    protected GraphQLError resolveToSingleError(Throwable ex, DataFetchingEnvironment env)
    {
        return new GraphQLError() {
            @Override
            public String getMessage() {
                return ex.getMessage();
            }
            no usages
            @Override
            public List<SourceLocation> getLocations() {
                return null;
            }
            no usages
            @Override
            public ErrorClassification getErrorType() {
                return null;
            }
        };
    }
}
```


CustomDataFetcherExceptionHandlerResolver est une classe personnalisée pour gérer les exceptions dans les requêtes et mutations GraphQL. Elle étend DataFetcherExceptionHandlerResolverAdapter et redéfinit la méthode resolveToSingleError pour retourner un objet GraphQLError avec le message d'erreur approprié.

5. configuration de GraphQL

```
spring.datasource.url=jdbc:h2:mem:account-db
spring.h2.console.enabled=true
server.port=8081
spring.graphql.graphiql.enabled=true
```

6. Test de h2-console

The screenshot shows the H2 Console web interface in a browser. The address bar displays the URL: `http://localhost:8081/h2-console/login.do?jsessionid=057578a2ca7849a96839b01833e1d693`. The interface includes a sidebar with a tree view of the database schema, a central area for SQL queries, and a bottom section displaying the query results.

Database Schema:

- jdbc:h2:mem:account-db
 - BANK_ACCOUNT
 - BALANCE
 - CREATED_AT
 - CUSTOMER_ID
 - CURRENCY
 - ID
 - TYPE
 - Indexes
 - CUSTOMER
 - ID
 - NAME
 - Indexes
 - INFORMATION_SCHEMA
 - Users

SQL Statement:

```
SELECT * FROM BANK_ACCOUNT;
```

Query Results:

BALANCE	CREATED_AT	CUSTOMER_ID	CURRENCY	ID	TYPE
78346.07117765634	2024-07-10 00:25:36.173	1	MAD	1194d817-4811-4e4e-af23-dba57e4662d4	CURRENT_ACCOUNT
45221.0711338064	2024-07-10 00:25:36.249	1	MAD	18c943d0-4076-4b16-8b72-4ef6b9081e25	CURRENT_ACCOUNT
58843.60411657975	2024-07-10 00:25:36.253	1	MAD	f66292da-359b-4017-9a6c-467177f450dd	SAVING_ACCOUNT
92604.41948985054	2024-07-10 00:25:36.257	1	MAD	e59f35cf-3005-402a-b546-02512e9c3856	SAVING_ACCOUNT
40708.1732968347	2024-07-10 00:25:36.261	1	MAD	3043f614-f1c8-4923-b542-5be0da57123d	CURRENT_ACCOUNT
64658.908739761566	2024-07-10 00:25:36.265	1	MAD	8d0ba0fa-e043-4f05-b537-3bd1a44e8940	SAVING_ACCOUNT
22570.985690747857	2024-07-10 00:25:36.27	1	MAD	af496946-3a12-47da-89f3-21ca78f9f25d	SAVING_ACCOUNT
46679.71127360097	2024-07-10 00:25:36.274	1	MAD	974a7a59-48bc-4786-a799-2687921cabe9	CURRENT_ACCOUNT
49349.078577091735	2024-07-10 00:25:36.278	1	MAD	d12d698c-236f-4546-8ae7-8919f0d1e6ed	CURRENT_ACCOUNT
42182.49499196303	2024-07-10 00:25:36.282	1	MAD	3ffb85e8-702b-4928-8bfd-54d94c91f141	SAVING_ACCOUNT
59046.091760926785	2024-07-10 00:25:36.286	2	MAD	06e153ae-8589-434b-9dee-65685d9c3c9b	CURRENT_ACCOUNT
12078.950654360386	2024-07-10 00:25:36.29	2	MAD	7022f08f-a2a0-44f2-9a4e-3162664a3dca	CURRENT_ACCOUNT
70412.86553475657	2024-07-10 00:25:36.294	2	MAD	44a0881e-2450-4775-981e-3d53b54e6c32	SAVING_ACCOUNT
97334.39383170563	2024-07-10 00:25:36.298	2	MAD	0cafd3d-7e91-462a-9c75-155f6e255d39	SAVING_ACCOUNT
41128.845525934266	2024-07-10 00:25:36.302	2	MAD	d0cd9b50-37b7-49ca-a959-93f7fe63298	CURRENT_ACCOUNT
23640.44694362436	2024-07-10 00:25:36.307	2	MAD	6871cc1c-4d86-4340-960f-293e668e109a	SAVING_ACCOUNT

H2 Console

http://localhost:8081/h2-console/login.do?jsessionId=057578a2...

Auto Max 1000 Auto complete Auto select

jdbc:h2:mem:account-db

- BANK_ACCOUNT
 - BALANCE
 - CREATED_AT
 - CUSTOMER_ID
 - CURRENCY
 - ID
 - TYPE
 - Indexes
- CUSTOMER
 - ID
 - NAME
 - Indexes
- INFORMATION_SCHEMA
- Users

H2 2.2.224 (2023-09-17)

Run Run Selected Auto complete Clear SQL statement:

SELECT * FROM CUSTOMER

SELECT * FROM CUSTOMER;

ID	NAME
1	Mohamed
2	Yassine
3	Hanae
4	Imane

(4 rows, 1 ms)

Edit

7. Test de GraphQL

a. Suppression du client

```
1 query{
2   accountList{
3     id , balance , customer{ name }
4   }
5 }
```

```
{
  "data": {
    "accountList": [
      {
        "id": "018ad91f-d3fe-4d5d-85e5-6859b6e5dfc4",
        "balance": 71298.51476665796,
        "customer": {
          "name": "Mohamed"
        }
      },
      {
        "id": "018ad91f-d3fe-4d5d-85e5-6859b6e5dfc4",
        "balance": 71298.51476665796,
        "customer": {
          "name": "Mohamed"
        }
      }
    ]
  }
}
```

b. Afficher les clients

```
1 query{
2   customers{
3     id , name
4   }
5 }
```

```
{
  "data": {
    "customers": [
      {
        "id": "1",
        "name": "Mohamed"
      },
      {
        "id": "2",
        "name": "Yassine"
      }
    ]
  }
}
```

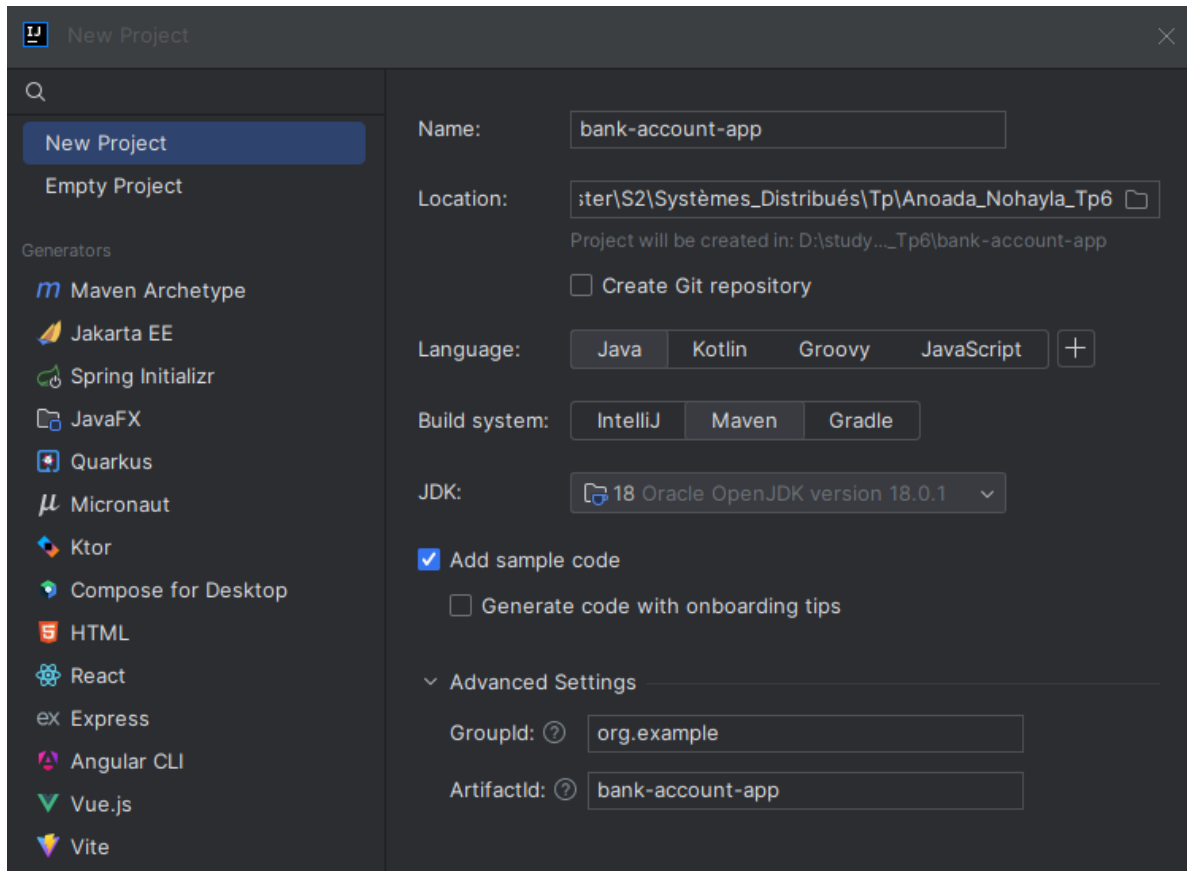
```
1 query{
2   customers{
3     id , name , bankAccounts{balance}
4   }
5 }
```



```
{
  "data": {
    "customers": [
      {
        "id": "1",
        "name": "Mohamed",
        "bankAccounts": [
          {
            "balance": 25703.861486974933
          },
          {
            "balance": 58618.83209629581
          }
        ]
      }
    ]
  }
}
```

PARTIE 2 : DÉVELOPPER UNE ARCHITECTURE MICRO-SERVICE

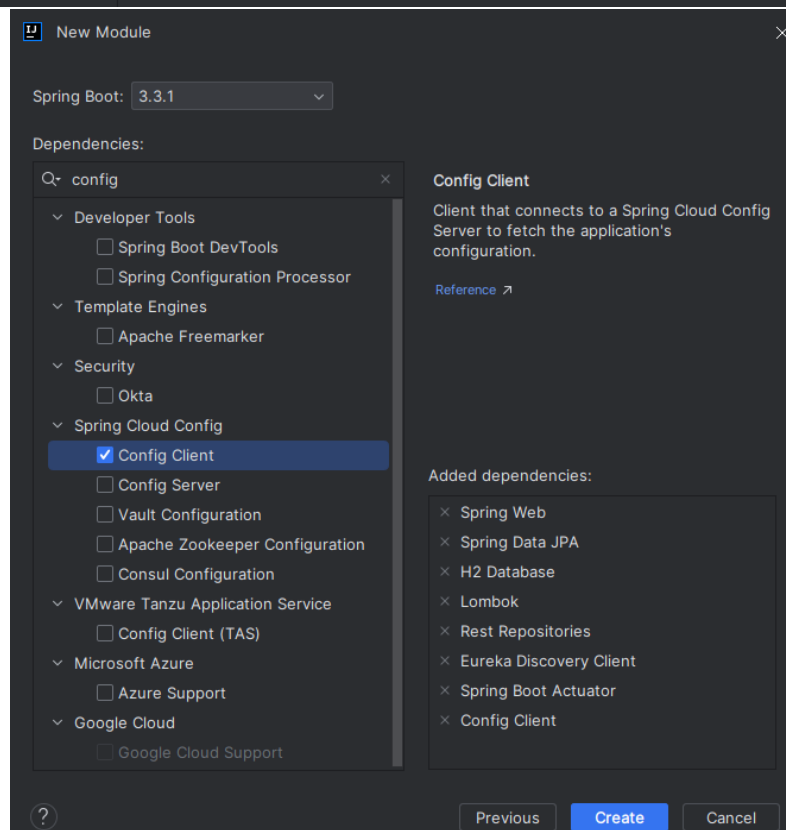
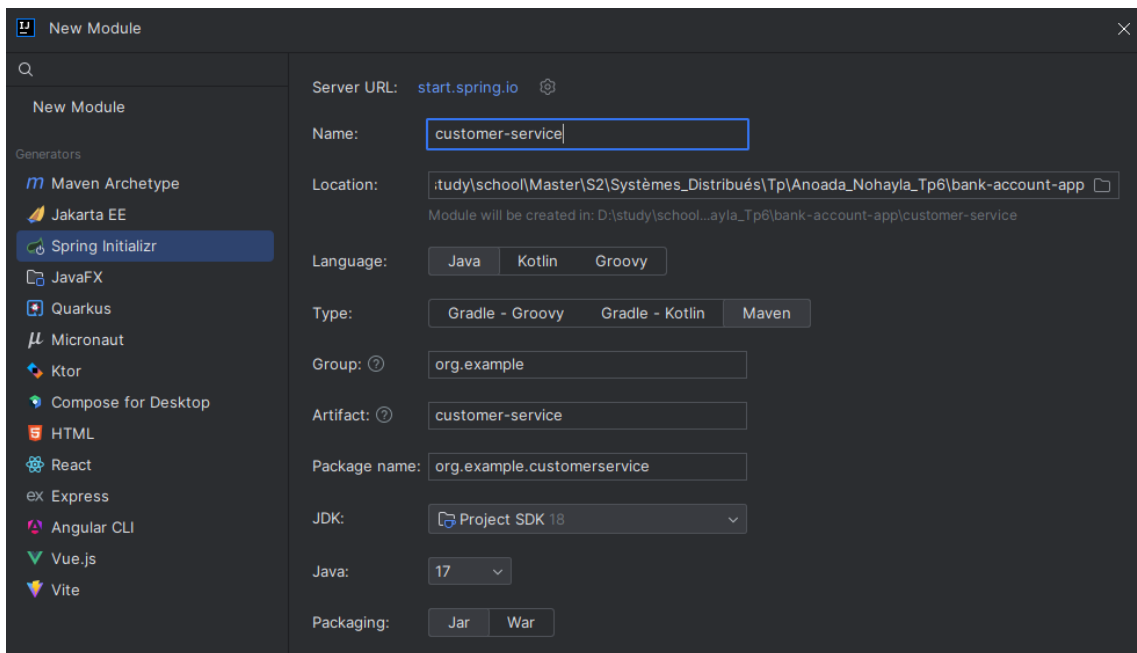
1. CRÉER DE PROJET MAVEN



la création d'un nouveau projet Spring Boot de type Maven

2. CRÉER LE MICRO-SERVICE CUSTOMER SERVICE

1. Creation de project maven



la création d'un nouveau projet Spring Boot de type Maven représente un micro-service de customer avec des dépendances Spring Data, Spring Web, H2, Lombok, Rest, discovery Client, Spring Boot et Config Client

2. Création d'entité

```
@Entity
@Data
@NoArgsConstructor @AllArgsConstructor @Builder
@Getter @Setter
@ToString
public class Customer
{
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String firstName;
    private String lastName;
    private String email;
}
```

La classe Customer représente un client avec un identifiant, un prénom, un nom de famille, et une adresse e-mail, et utilise diverses annotations pour simplifier le code et la gestion de l'entité dans une application utilisant JPA et Lombok.

3. configuration de l'application

```
spring.application.name=customer-service
spring.cloud.discovery.enabled=false
server.port=8081
spring.datasource.url=jdbc:h2:mem:customer-db
spring.h2.console.enabled=true
spring.cloud.config.enabled=false
```

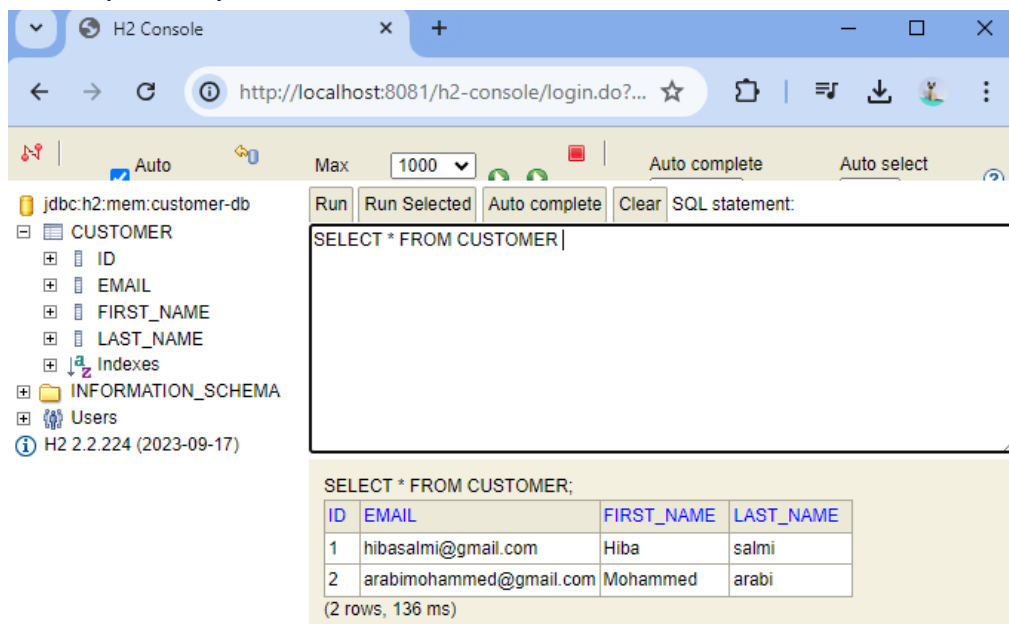
La configuration de l'application Spring Boot inclut une base de données en mémoire H2 avec l'URL jdbc:h2:mem:customer-db, active la console H2, et définit le port du serveur sur 8081.

4. Faire un test (insertion des clients)

```
@SpringBootApplication
public class CustomerServiceApplication
{
    public static void main(String[] args)
    {
        SpringApplication.run(CustomerServiceApplication.class, args);
    }

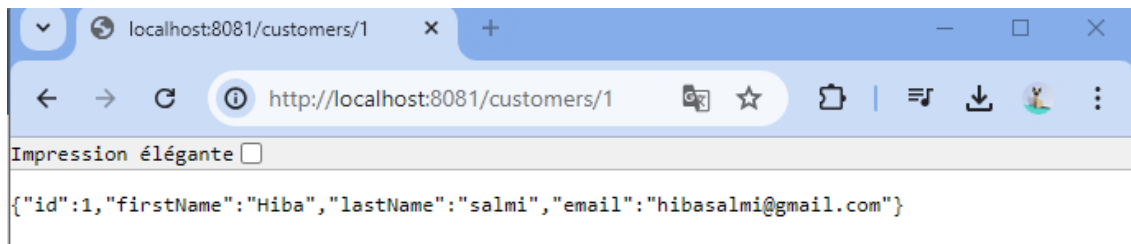
    @Bean
    CommandLineRunner start(CustomerRepository customerRepository)
    {
        return args -> {
            List<Customer> customerList=List.of(
                Customer.builder()
                    .firstName("Hiba")
                    .lastName("salmi")
                    .email("hibasalmi@gmail.com")
                    .build(),
                Customer.builder()
                    .firstName("Mohammed")
                    .lastName("arabi")
                    .email("arabimohammed@gmail.com")
                    .build()
            );
            customerRepository.saveAll(customerList);
        };
    }
}
```

La classe CustomerServiceApplication initialise la application Spring Boot et génère 2 clients à l'aide de la méthode CommandLineRunner. Chaque client est créé avec un nom, prenom et email puis sauvegardé dans le dépôt CustomerRepository.



The screenshot shows the H2 Console interface in a web browser. The URL is `http://localhost:8081/h2-console/login.do?...`. The left sidebar shows the database structure for `jdbc:h2:mem:customer-db`, including a `CUSTOMER` table with columns `ID`, `EMAIL`, `FIRST_NAME`, and `LAST_NAME`. The main area displays the SQL statement `SELECT * FROM CUSTOMER` and its results. The results are shown in a table with 2 rows and 4 columns: `ID`, `EMAIL`, `FIRST_NAME`, and `LAST_NAME`. The first row contains `1`, `hibasalmi@gmail.com`, `Hiba`, and `salmi`. The second row contains `2`, `arabimohammed@gmail.com`, `Mohammed`, and `arabi`. The status bar indicates `(2 rows, 136 ms)`.

ID	EMAIL	FIRST_NAME	LAST_NAME
1	hibasalmi@gmail.com	Hiba	salmi
2	arabimohammed@gmail.com	Mohammed	arabi



5. Création d'interface repository

```
@RepositoryRestResource
public interface CustomerRepository extends JpaRepository<Customer, Long> {
}
```

Interface CustomerRepository étend JpaRepository<Customer, Long> pour interagir avec la base de données.

6. Contrôleur

```
@RestController
@RefreshScope
public class CustomerRestController {

    3 usages
    private CustomerRepository customerRepository;

    public CustomerRestController(CustomerRepository customerRepository) {
        this.customerRepository = customerRepository;
    }

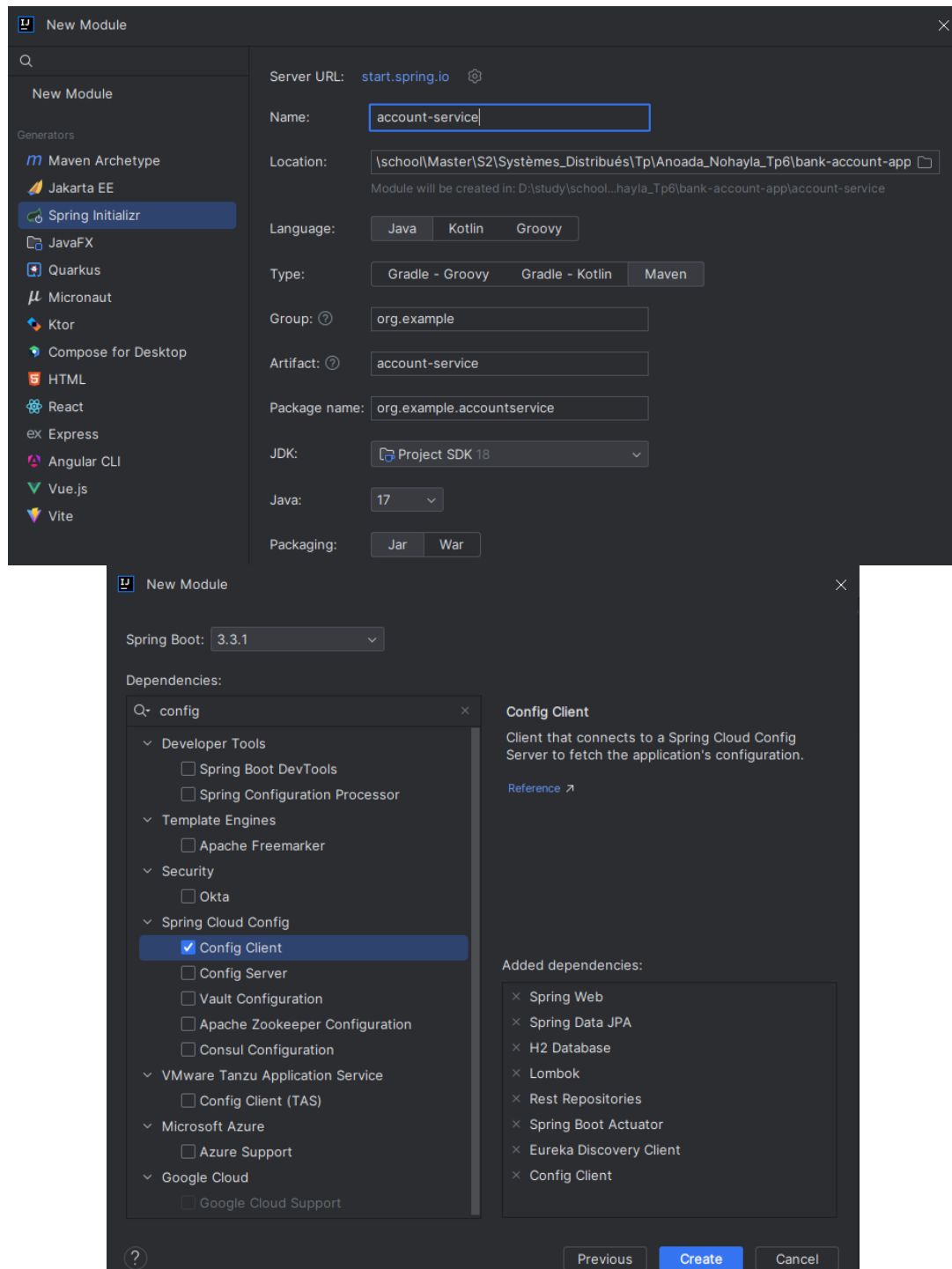
    @GetMapping("/customers")
    public List<Customer> customerList(){
        return customerRepository.findAll();
    }

    @GetMapping("/{id}")
    public Customer customerById(@PathVariable Long id){
        return customerRepository.findById(id).get();
    }
}
```

le contrôleur est conçu pour interagir avec un CustomerRestController, qui devrait être une interface CustomerRepository pour effectuer les opérations CRUD (Create, Read, Update, Delete) sur les entités Customer dans une base de données.

3. CRÉER LE MICRO-SERVICE ACCOUNT SERVICE

1. Creation de project maven



la création d'un nouveau projet Spring Boot de type Maven représente un micro-service de aount avec des dépendances Spring Data, Spring Web, H2, Lombok, Rest, discovery Client, Spring Boot et Config Client

2. Création d'entité BankAccount

```
@Entity
@Getter @Setter @ToString @NoArgsConstructor @AllArgsConstructor @Builder
public class BankAccount
{
    @Id
    private String accountId;
    private double balance;
    private LocalDate createdAt;
    private String currency;
    @Enumerated(EnumType.STRING)
    private AccountType type;
    @Transient
    private Customer customer;
    private Long customerId;
}
```

La classe BankAccount représente un compte bancaire avec un identifiant, un solde, une date de création, une devise, un type de compte, un client associé (non persistant) et l'identifiant du client. Elle utilise diverses annotations pour simplifier le code et la gestion de l'entité dans une application utilisant JPA et Lombok.

3. Création de enums

```
public enum AccountType
{
    1 usage
    CURRENT_ACCOUNT, SAVING_ACCOUNT
}
```

Le type de compte est défini par l'énumération AccountType avec les valeurs CURRENT_ACCOUNT et SAVING_ACCOUNT, stockée comme une chaîne dans la base de données.

4. Création d'entité Customer

```
2 usages
@Getter
@Setter
@ToString

public class Customer {
    private Long id;
    private String firstName;
    private String lastName;
    private String email;
}
```

La classe Customer représente un client avec un identifiant, un prénom, un nom de famille, et une adresse e-mail, et utilise diverses annotations pour simplifier le code et la gestion de l'entité dans une application utilisant JPA et Lombok.

5. configuration de l'application

```
spring.application.name=account-service
spring.cloud.discovery.enabled=false
spring.datasource.url=jdbc:h2:mem:account-db
spring.h2.console.enabled=true
server.port=8082
spring.cloud.config.enabled=false
```

La configuration de l'application Spring Boot inclut une base de données en mémoire H2 avec l'URL jdbc:h2:mem:account-db, active la console H2, et définit le port du serveur sur 8081.

6. Ajouter dependency spring boot openapi doc maven

```
<dependency>
  <groupId>org.springdoc</groupId>
  <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
  <version>2.6.0</version>
</dependency>
```

7. Faire un test (insertion des comptes)

```
@SpringBootApplication
public class AccountServiceApplication {

    public static void main(String[] args) {

        SpringApplication.run(AccountServiceApplication.class, args);
    }

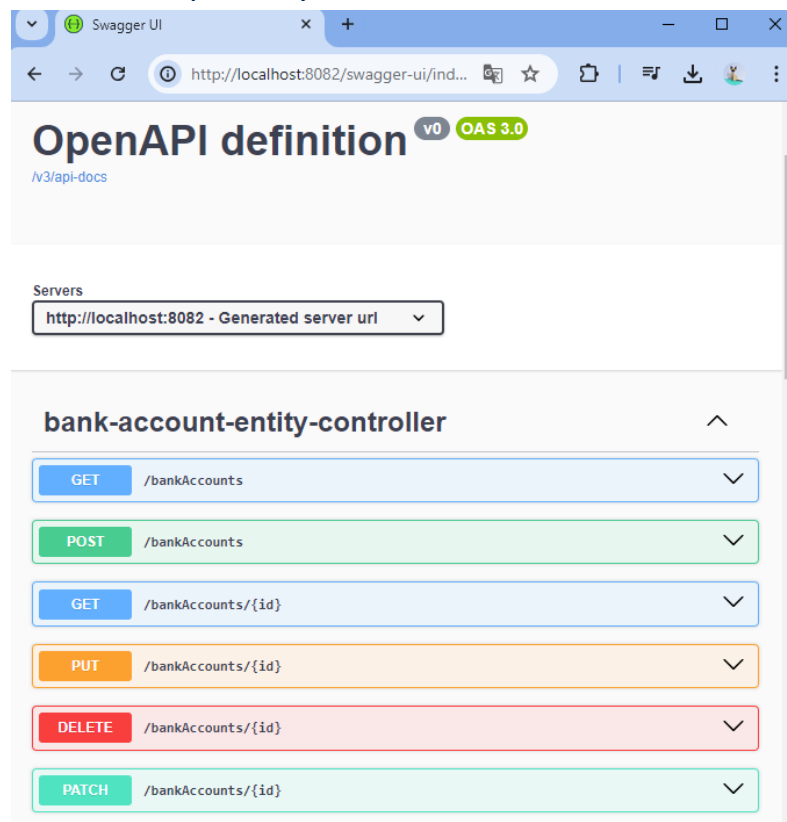
    @Bean
    CommandLineRunner commandLineRunner(BankAccountRepository accountRepository) {
        return args -> {
            BankAccount bankAccount1 = BankAccount.builder()
                .accountId(UUID.randomUUID().toString())
                .currency("MAD")
                .balance(98080)
                .createAt(LocalDate.now())
                .type(AccountType.CURRENT_ACCOUNT)
                .customerId(Long.valueOf(1))
                .build();
        }
    }
}
```

```

        BankAccount bankAccount2 = BankAccount.builder()
            .accountId(UUID.randomUUID().toString())
            .currency("MAD")
            .balance(100000)
            .createAt(LocalDate.now())
            .type(AccountType.SAVING_ACCOUNT)
            .customerId(Long.valueOf(12))
            .build();
        accountRepository.save(bankAccount1);
        accountRepository.save(bankAccount2);
    };
}

```

La classe AccountServiceApplication initialise la application Spring Boot et génère 2 comptes bancaires à l'aide de la méthode CommandLineRunner. Chaque compte est créé avec un identifiant unique, un type aléatoire (compte courant ou compte épargne), un solde aléatoire, une date de création, et une devise fixée à "MAD" et associer chaque compte a un client, puis sauvegardé dans le dépôt AccountRepository.



8. Création d'interface repository

```

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

```

Interface BankAccountRepository étend JpaRepository<BankAccount, String> pour interagir avec la base de données.

9. Contrôleur REST

```
@RestController
public class AccountRestController {
    3 usages
    private BankAccountRepository accountRepository;

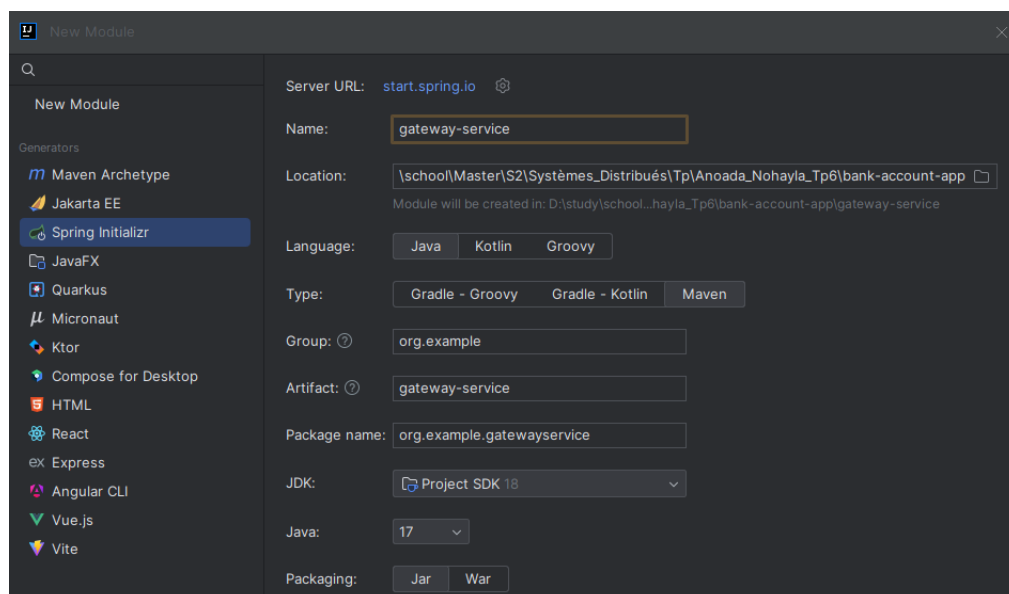
    public AccountRestController(BankAccountRepository accountRepository) {
        this.accountRepository = accountRepository;
    }

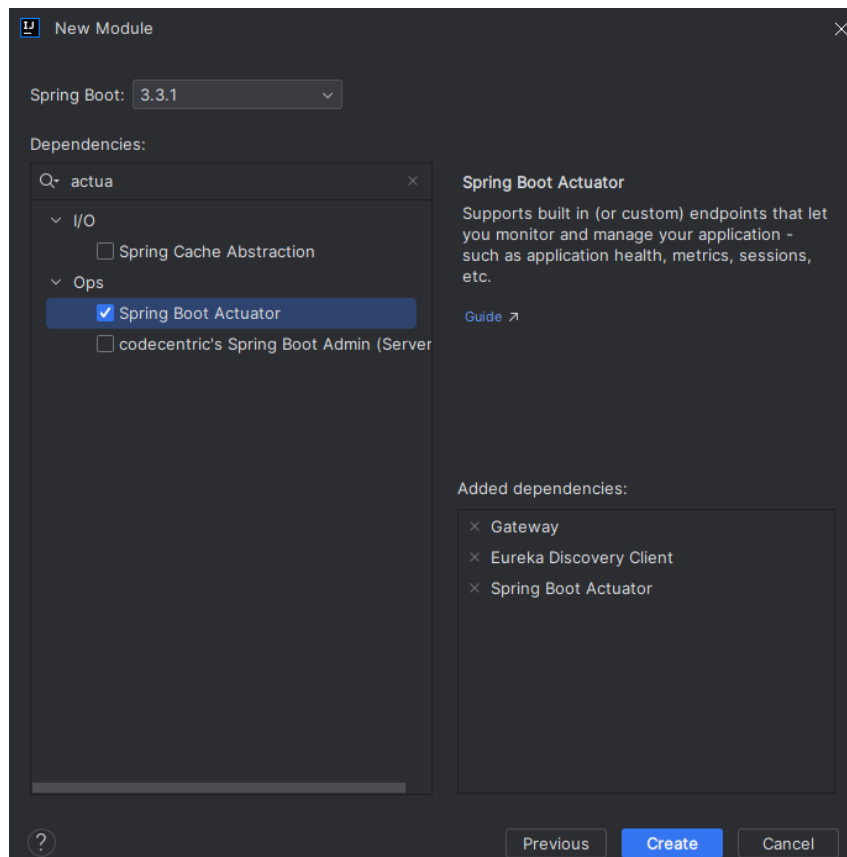
    @GetMapping("/accounts")
    public List<BankAccount> accountList() {
        return accountRepository.findAll();
    }

    @GetMapping("/accounts/{id}")
    public BankAccount bankAccountById(@PathVariable String id) {
        return accountRepository.findById(id).get();
    }
}
```

4. CRÉER LE MICRO-SERVICE GATEWAY SERVICE

1. Creation de project maven





2. configuration de l'application

```
spring:
  cloud:
    gateway:
      routes:
        - id: r1
          uri: http://localhost:8081/
          predicates:
            - Path=/customers/**
        - id: r2
          uri: http://localhost:8082/
          predicates:
            - Path=/accounts/**
      application:
        name: gateway-service
server:
  port: 8888
```

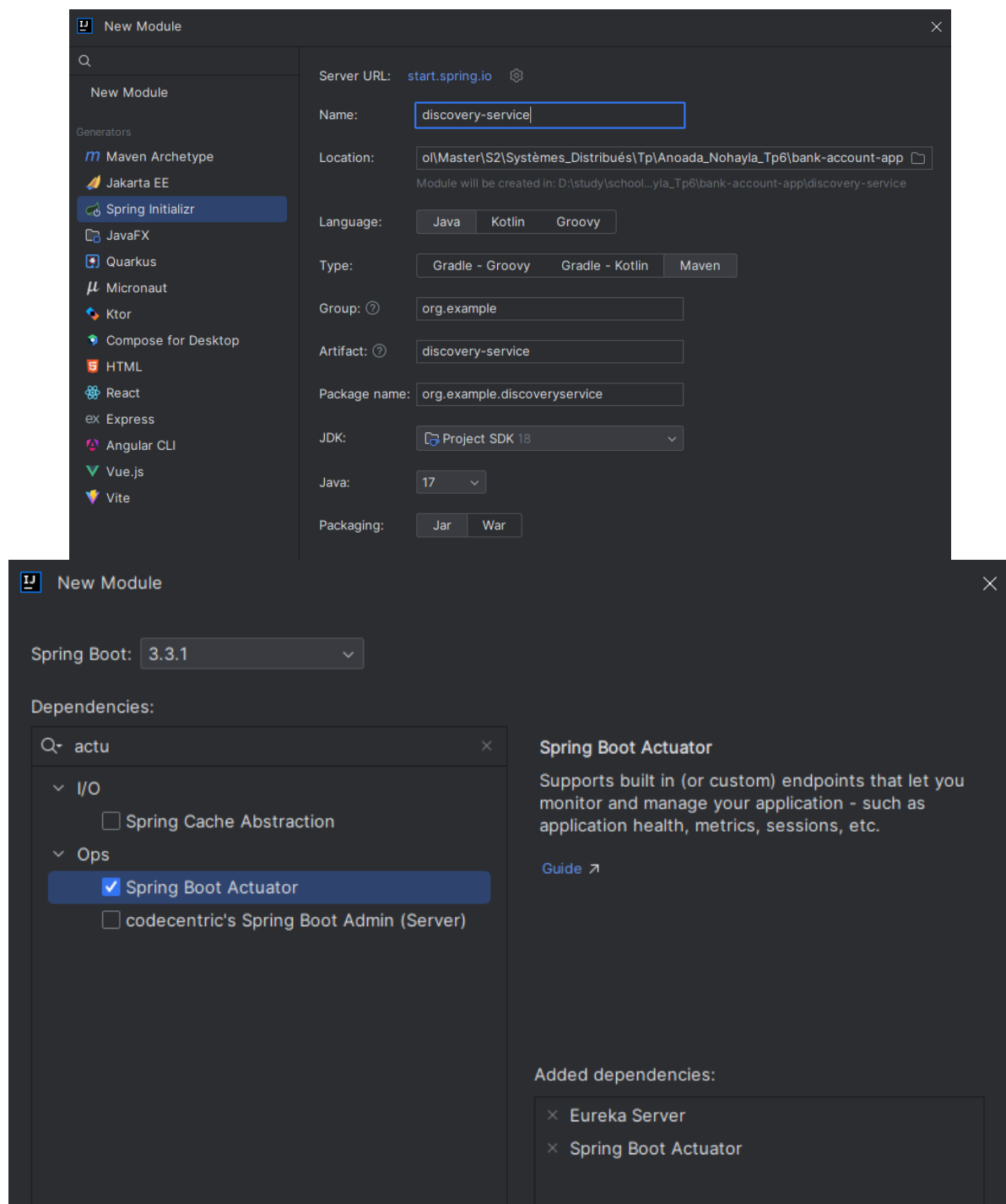
Le fichier de configuration application.yml configure un service Spring Cloud Gateway avec deux routes:

- Les requêtes vers les chemins commençant par /customers/ seront redirigées vers le service exécuté sur http://localhost:8081/.

- Les requêtes vers les chemins commençant par /accounts/ seront redirigées vers le service exécuté sur <http://localhost:8082/>.

5. CRÉER LE MICRO-SERVICE DISCOVERY SERVICE

1. Creation de project maven



2. classe DiscoveryServiceApplication

```
import ...

@SpringBootApplication
@EnableEurekaServer
public class DiscoveryServiceApplication {

    public static void main(String[] args) {

        SpringApplication.run(DiscoveryServiceApplication.class, args);
    }

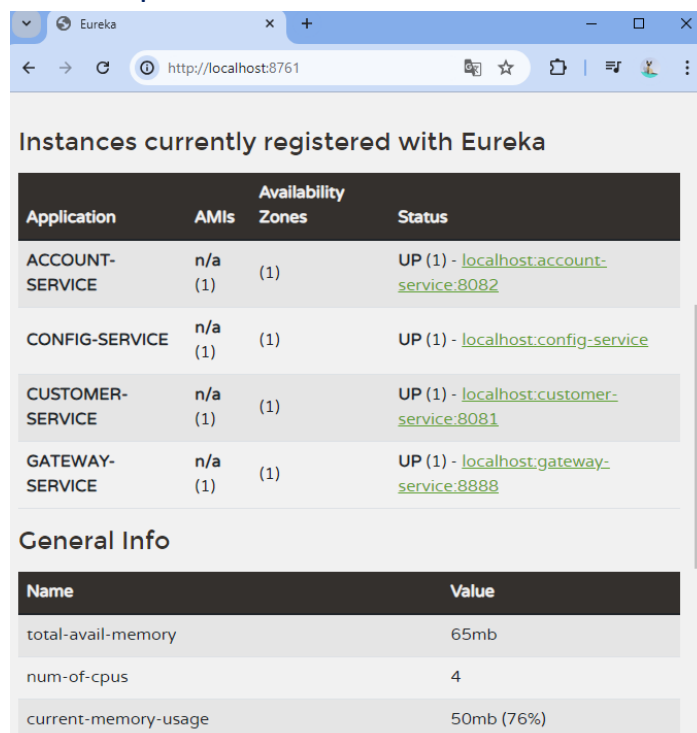
}
```

La classe `DiscoveryServiceApplication` configure un service de découverte Eureka avec Spring Boot. Lorsque l'application est démarrée, elle lance un serveur Eureka qui permet aux autres microservices de s'enregistrer et de découvrir d'autres services.

3. configuration de l'application

```
spring.application.name=discovery-service
server.port=8761
eureka.client.fetch-registry=false
eureka.client.register-with-eureka=false
```

Configuration fait en sorte que l'application Spring Boot démarre en tant que serveur Eureka sur le port 8761



The screenshot shows a web browser window with the URL `http://localhost:8761`. The page title is "Eureka". The main content area is titled "Instances currently registered with Eureka". It contains a table with the following data:

Application	AMIs	Availability Zones	Status
ACCOUNT-SERVICE	n/a (1)	(1)	UP (1) - localhost:account-service:8082
CONFIG-SERVICE	n/a (1)	(1)	UP (1) - localhost:config-service
CUSTOMER-SERVICE	n/a (1)	(1)	UP (1) - localhost:customer-service:8081
GATEWAY-SERVICE	n/a (1)	(1)	UP (1) - localhost:gateway-service:8888

Below the table is a section titled "General Info" with a table showing system metrics:

Name	Value
total-avail-memory	65mb
num-of-cpus	4
current-memory-usage	50mb (76%)

G. CRÉER LE MICRO-SERVICE CONFIG SERVICE

1. Creation de project maven

