



L'ARCHITECTURE MICROSERVICE

Comprendre et utiliser l'architecture microservice

Définition

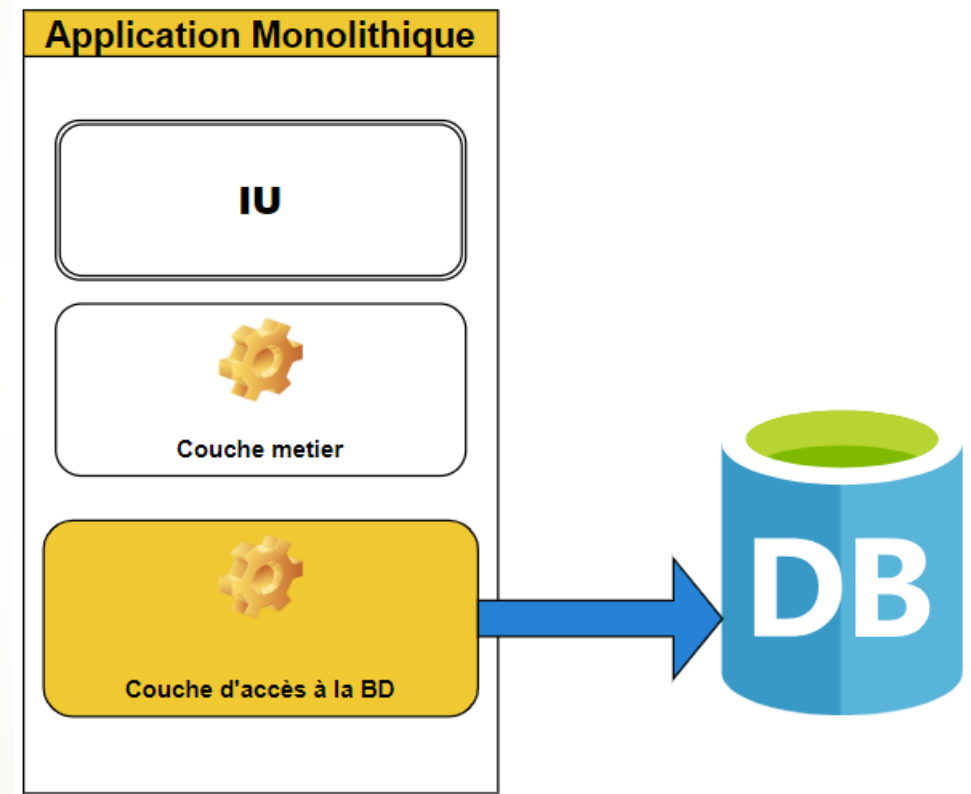
- ▀ Les microservices désignent à la fois une architecture et une approche de développement logiciel qui consiste à décomposer les applications en éléments les plus simples, indépendants les uns des autres

Service web et microservices

- Les microservices permettent de diviser l'application en plusieurs modules ou services de manière faiblement couplée afin qu'ils soient indépendants les uns des autres. Cela facilite le développement de l'application.
- Les services Web fournissent des normes ou des protocoles pour l'échange d'informations entre divers appareils ou applications.

L'architecture monolithique

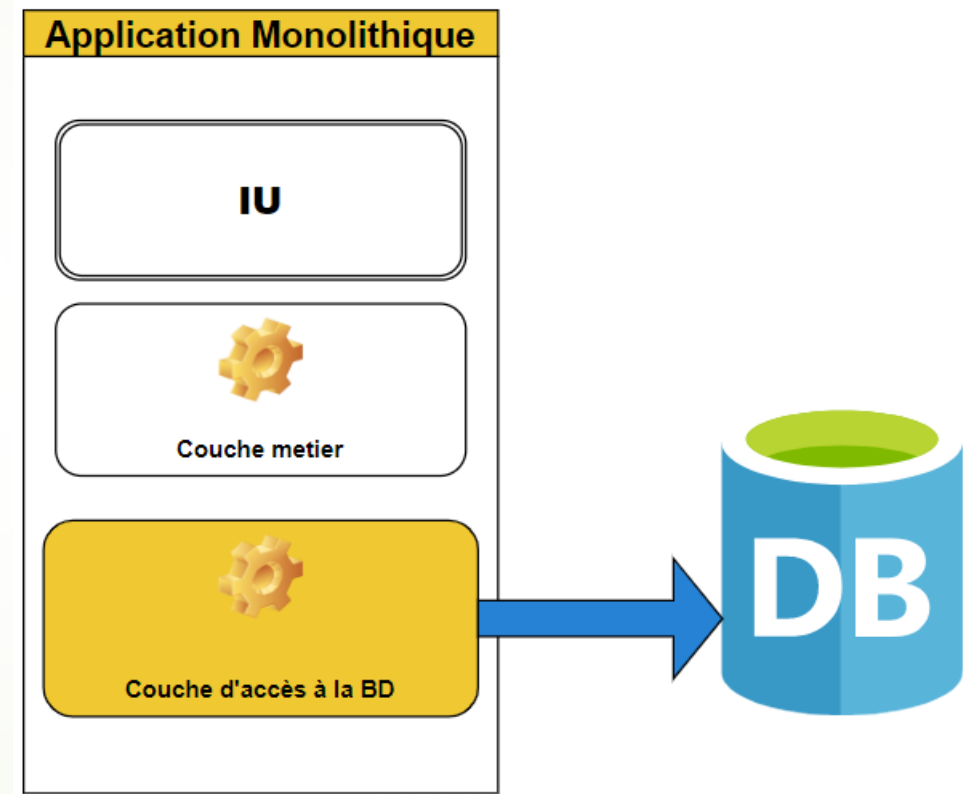
- Une manière traditionnelle de construire des applications.
- Pour l'approche monolithique une application est construite comme une unité unique et indivisible.
- Une telle solution comprend une interface utilisateur côté client, une application côté serveur et une base de données. Elle est unifiée et toutes les fonctions sont gérées et servies en un seul endroit.



L'architecture monolithique

➤ Pour

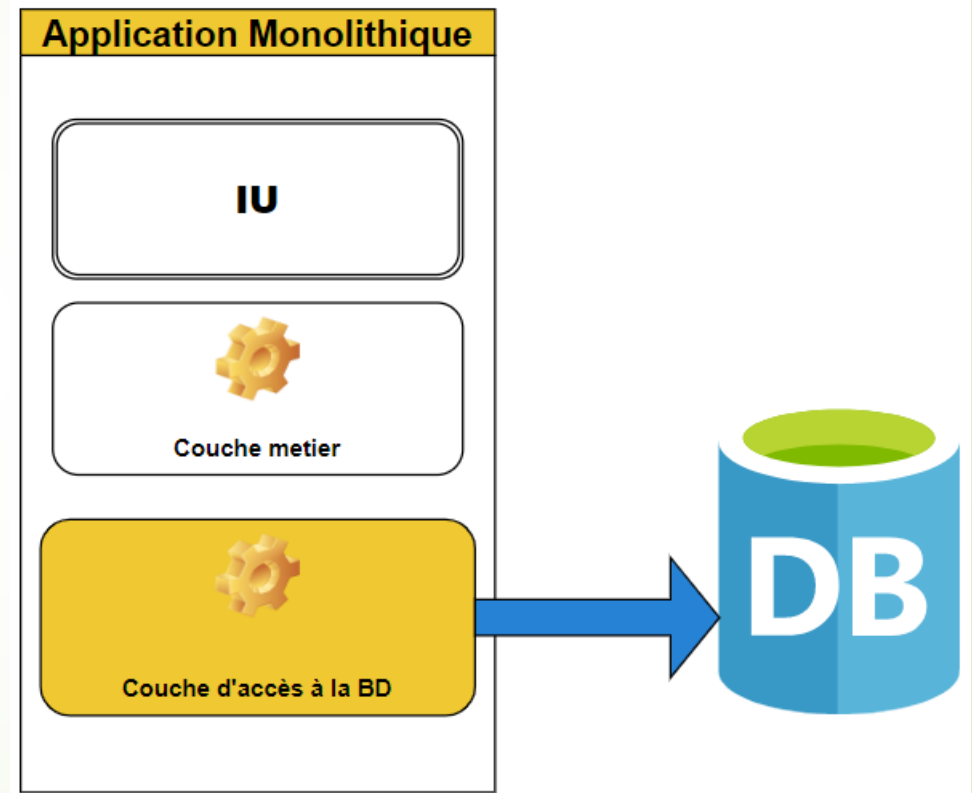
- Simplifie le développement et le déploiement d'application pour de petites équipes
- Meilleure performance due à la non latence du réseau (pas de communication nécessaire entre différentes applications)



L'architecture monolithique

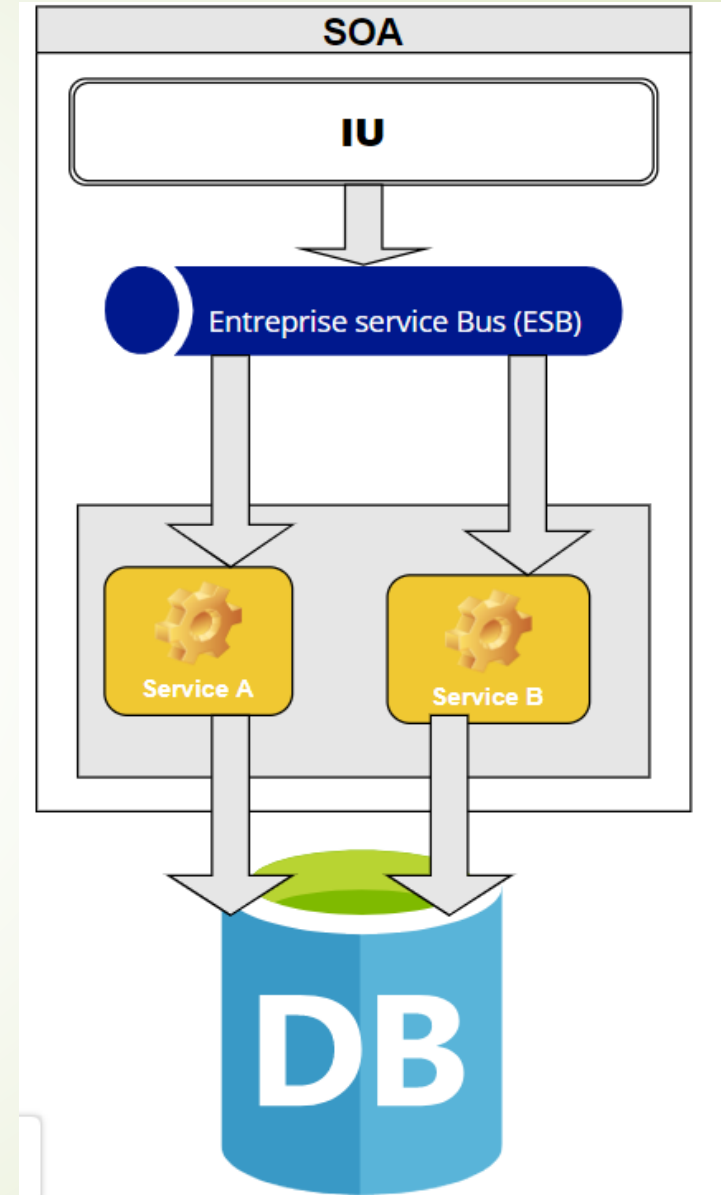
■ Contre

- L'adoption d'une nouvelle technologie est compliquée
- Ne favorise pas l'agilité dans le développement
- Difficulté de maintenir du seul code base
- Pas de tolérance aux pannes
- Toute modification nécessite le déploiement d'une nouvelle version de l'application entière



L'architecture SOA

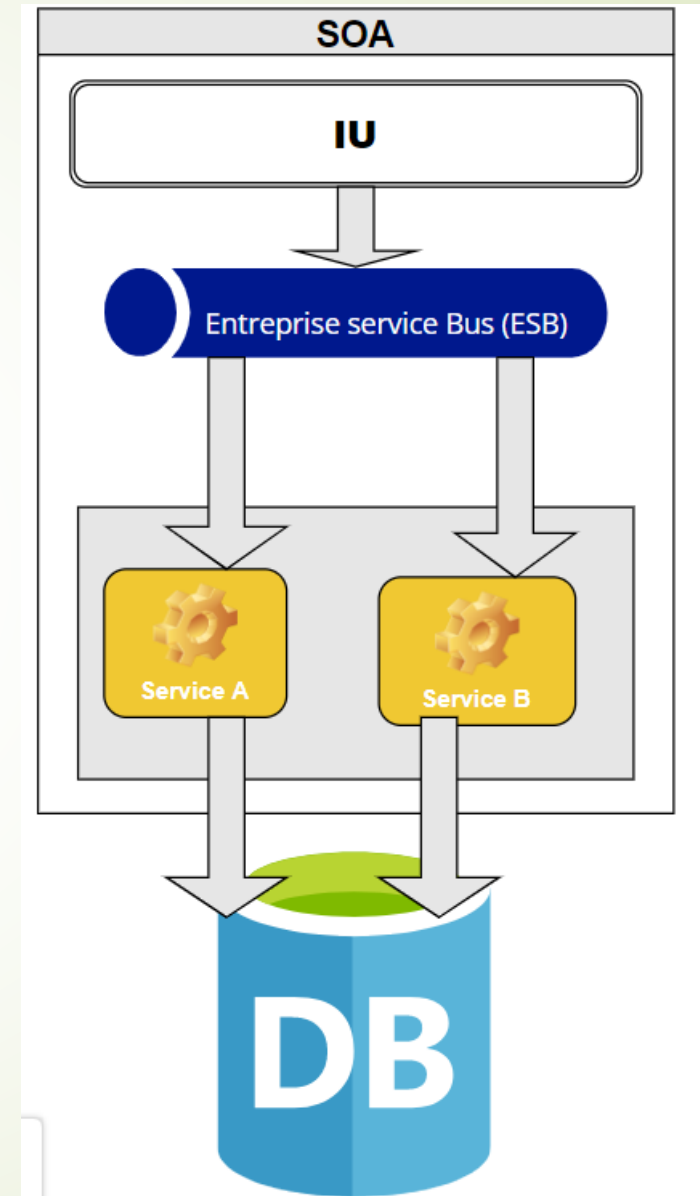
- L'architecture orientée services (ou SOA, Service-Oriented Architecture) est un modèle de conception qui rend des composants logiciels réutilisables, grâce à des interfaces de services qui utilisent un langage commun pour communiquer via un réseau.
- L'architecture SOA permet à des composants logiciels déployés et gérés séparément de communiquer et de fonctionner ensemble sous la forme d'applications logicielles communes à différents systèmes



L'architecture SOA

► Pour

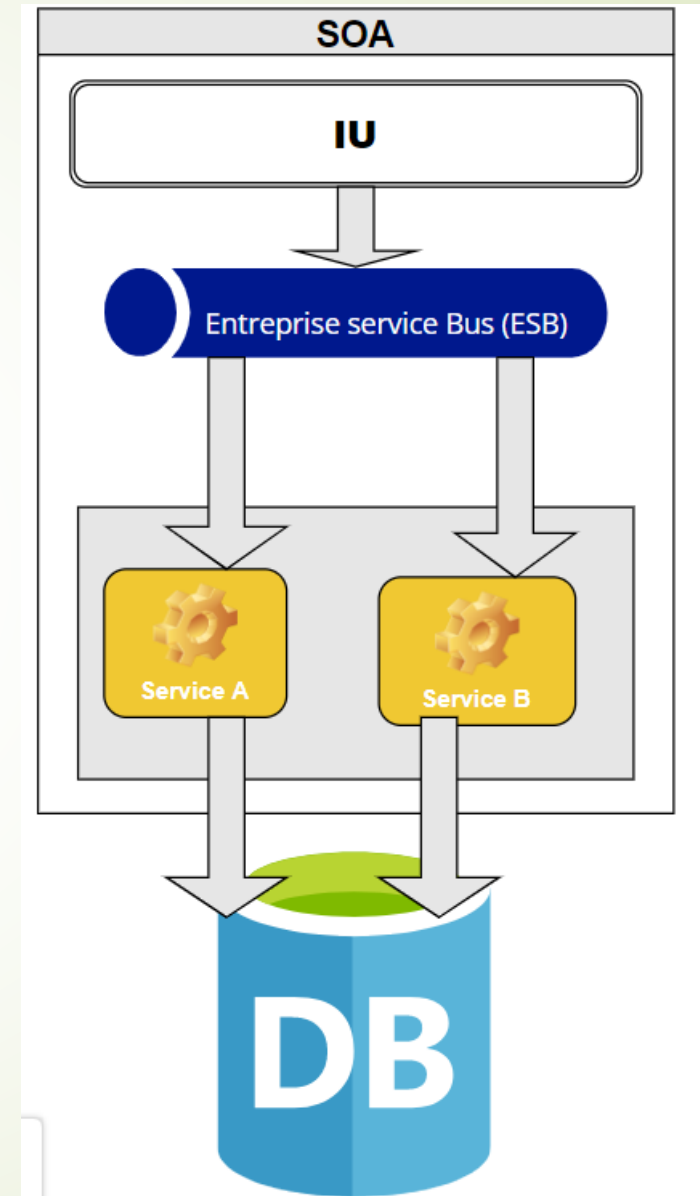
- Réutilisabilité des service ou composants
- Facilité de maintenance
- Développement parallèle
- Tolérance aux panes



L'architecture SOA

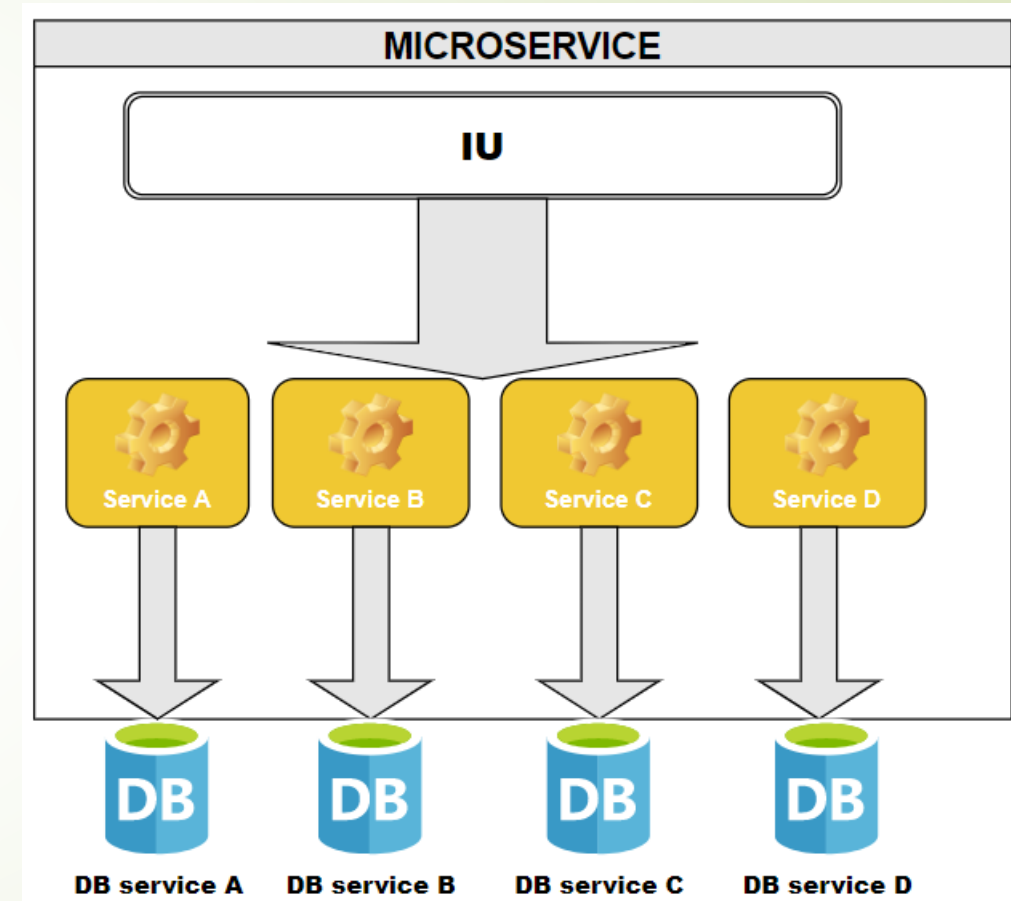
➤ Contre

- Management complexe
- Coût d'investissement élevé
- surcharge supplémentaire



L'architecture Microservice

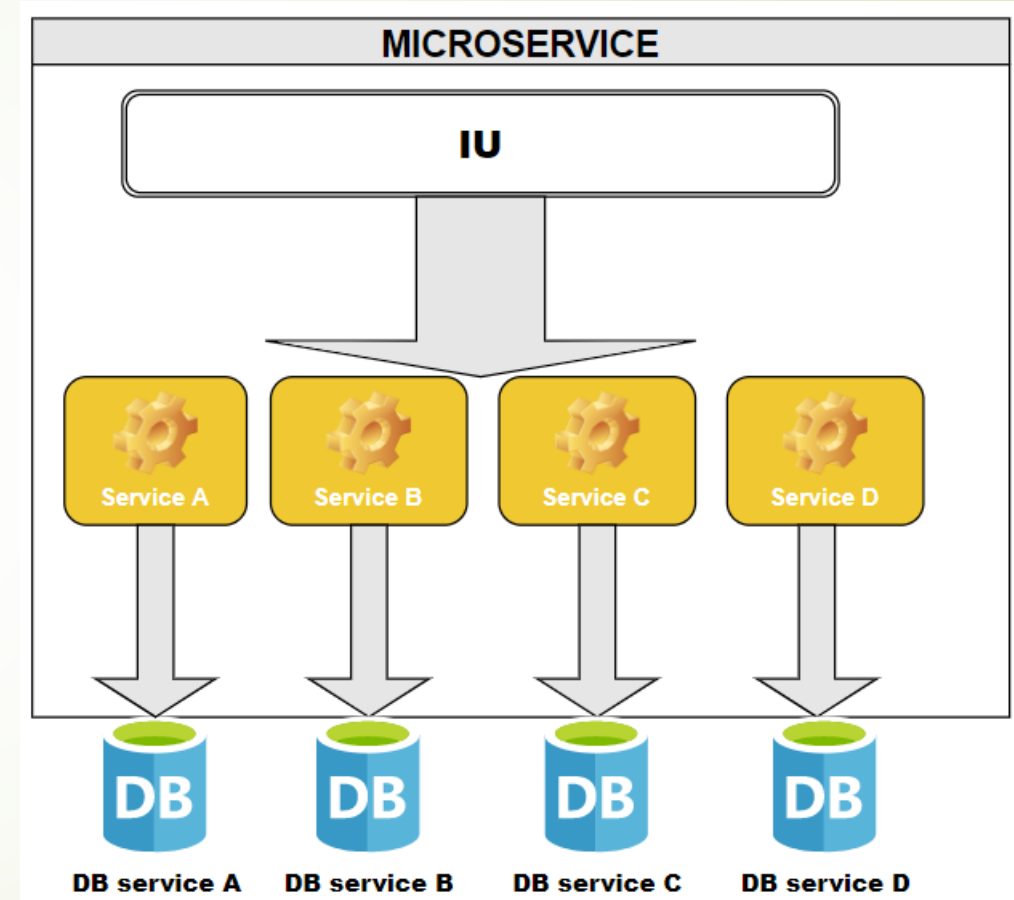
- Les applications sont décomposées en plusieurs petits services.
- Chaque service s'exécute dans son propre processus.
- Les services sont alignés autour des domaines métier.
- Les services communiquent via des API légères, généralement à l'aide des requêtes HTTP.
- Les services peuvent être déployés et mis à niveau indépendamment des autres.
- Les services ne dépendent pas d'une base de données unique.
- En cas d'échec d'un service les autres peuvent toujours s'exécuter t.



L'architecture Microservice

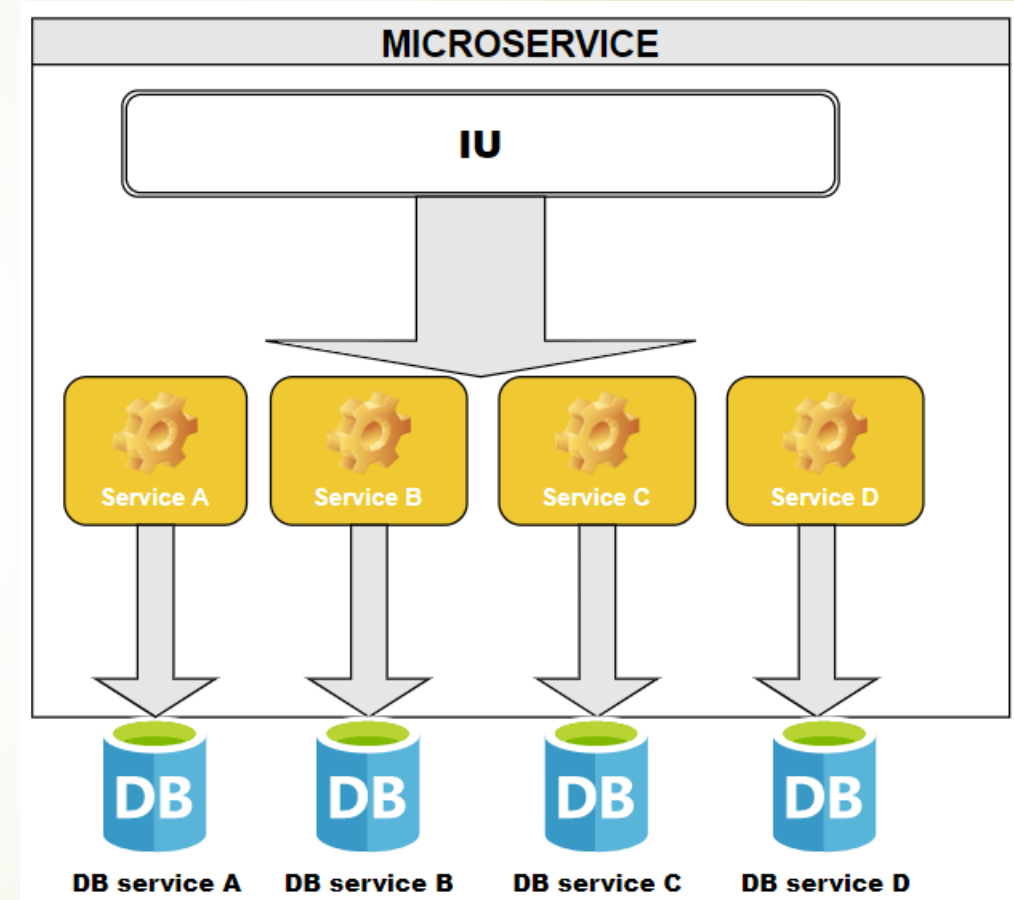
► Pour

- Facilite le développement, les tests et le déploiement
- Adapté à l'agilité
- Facilite la scalabilité horizontale
- Développement parallèle
- Les services peuvent être mis à niveau indépendamment les uns des autres



L'architecture Microservice

- ▀ Contre (beaucoup plus que des défis)
 - Complexité de mise en place
 - Problème de sécurité



L'architecture microservice avec Spring cloud

- Spring cloud est un projet du framework Spring de Java. Il fournit des outils aux développeurs pour créer rapidement certains des modèles courants dans les systèmes distribués notamment : **configuration management, service discovery, circuit breakers, intelligent routing ...**
- Spring Cloud se concentre sur la fourniture d'une bonne expérience prête à l'emploi pour les cas d'utilisation typiques et le mécanisme d'extensibilité pour couvrir les autres.

Projet d'exemple: Gestion d'une entreprise commerciale

- Gestion des approvisionnements
- Gestion des ventes
- Gestion de la facturation

Diagramme de classe: gestion des approvisionnements

En élaboration

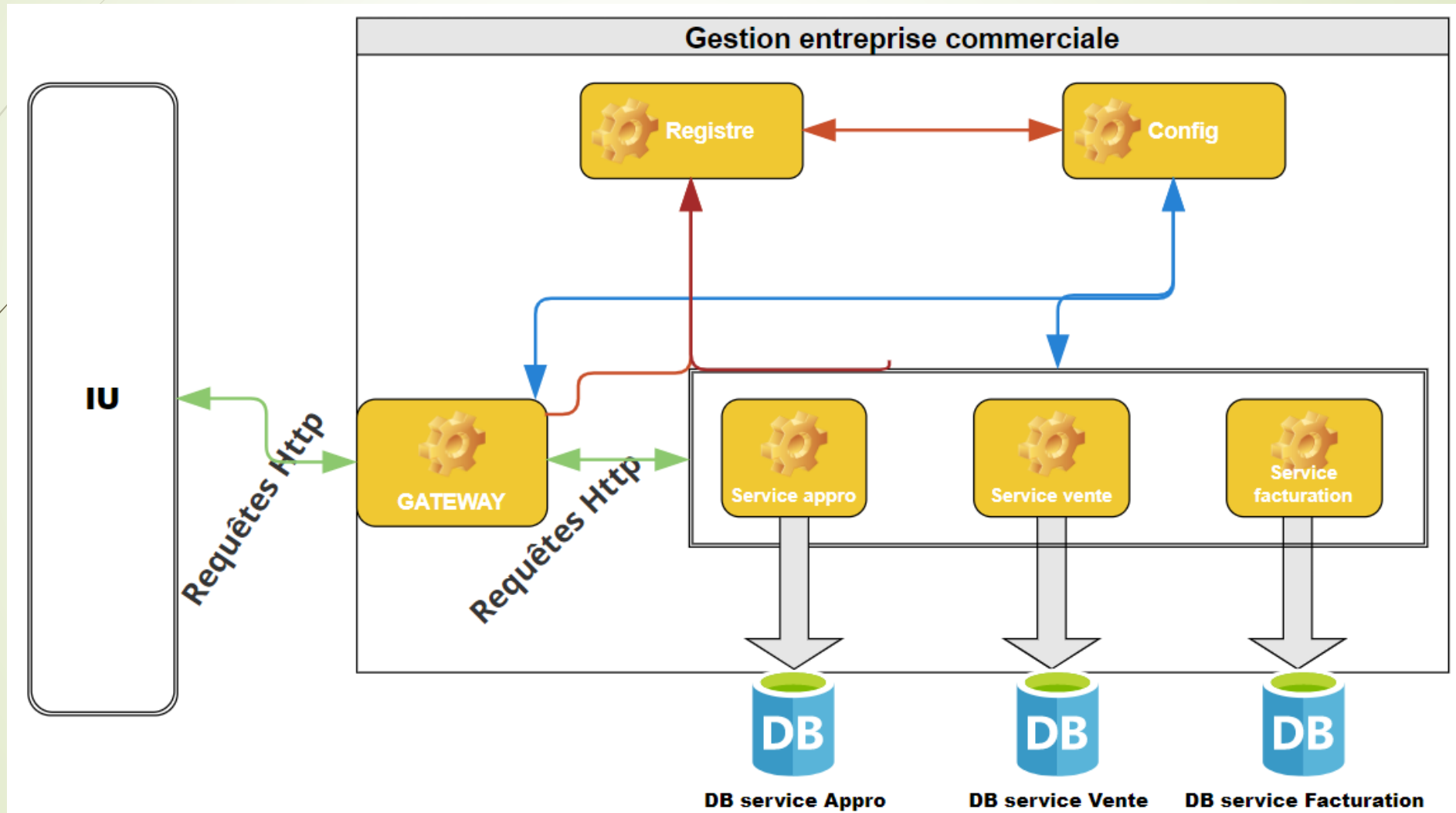
Diagramme de classe: Gestion de la vente

En élaboration

Diagramme de classe: Gestion de la facturation

En élaboration

L'architecture de l'application



Service : config avec Spring cloud config

- Spring Cloud Config permet une configuration externalisée dans un système distribué.
- Avec le serveur de configuration, vous disposez d'un emplacement central pour gérer les configurations externes des applications.

Dépendance

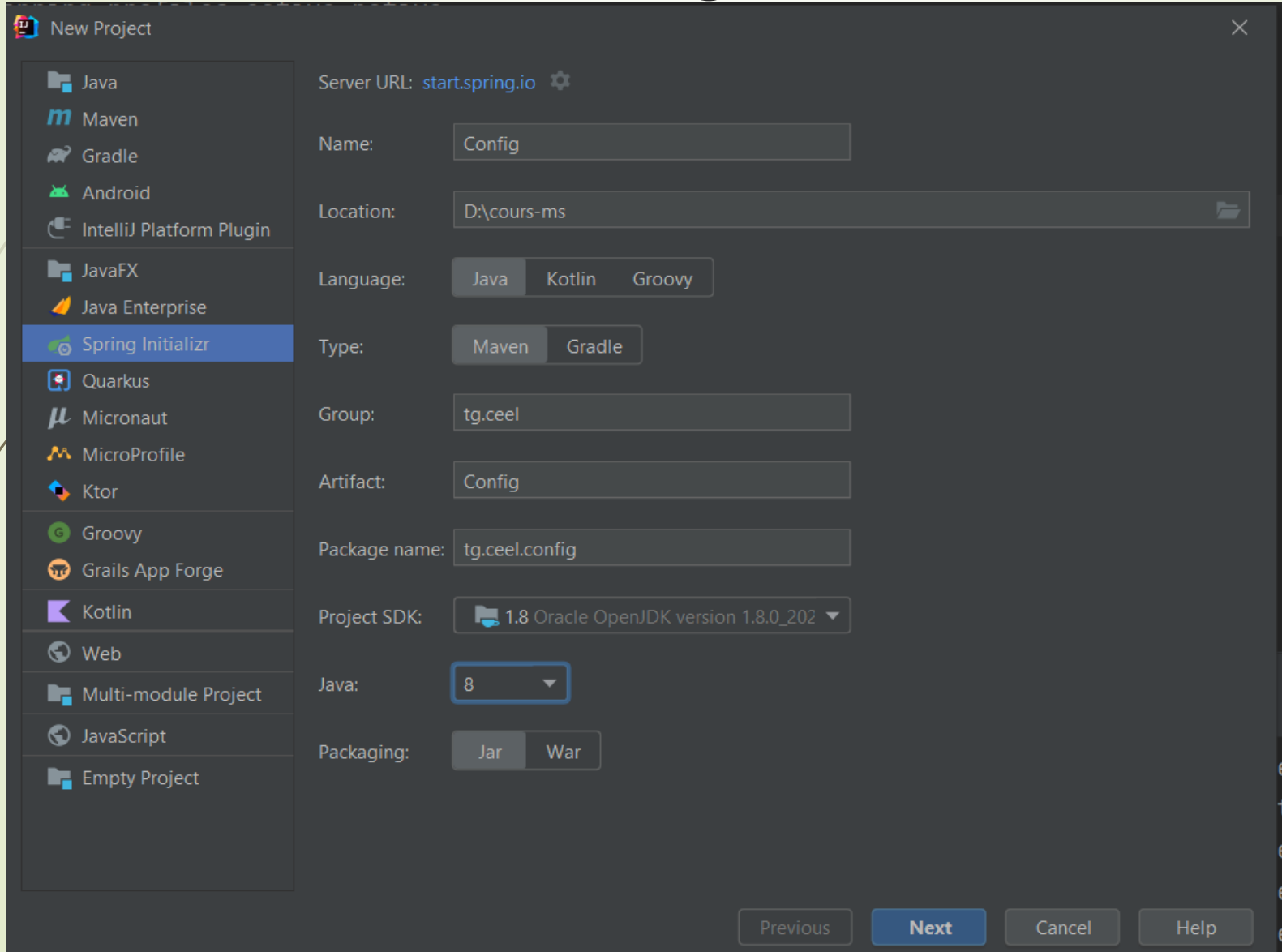
<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-config</artifactId>

</dependency>

Service : config

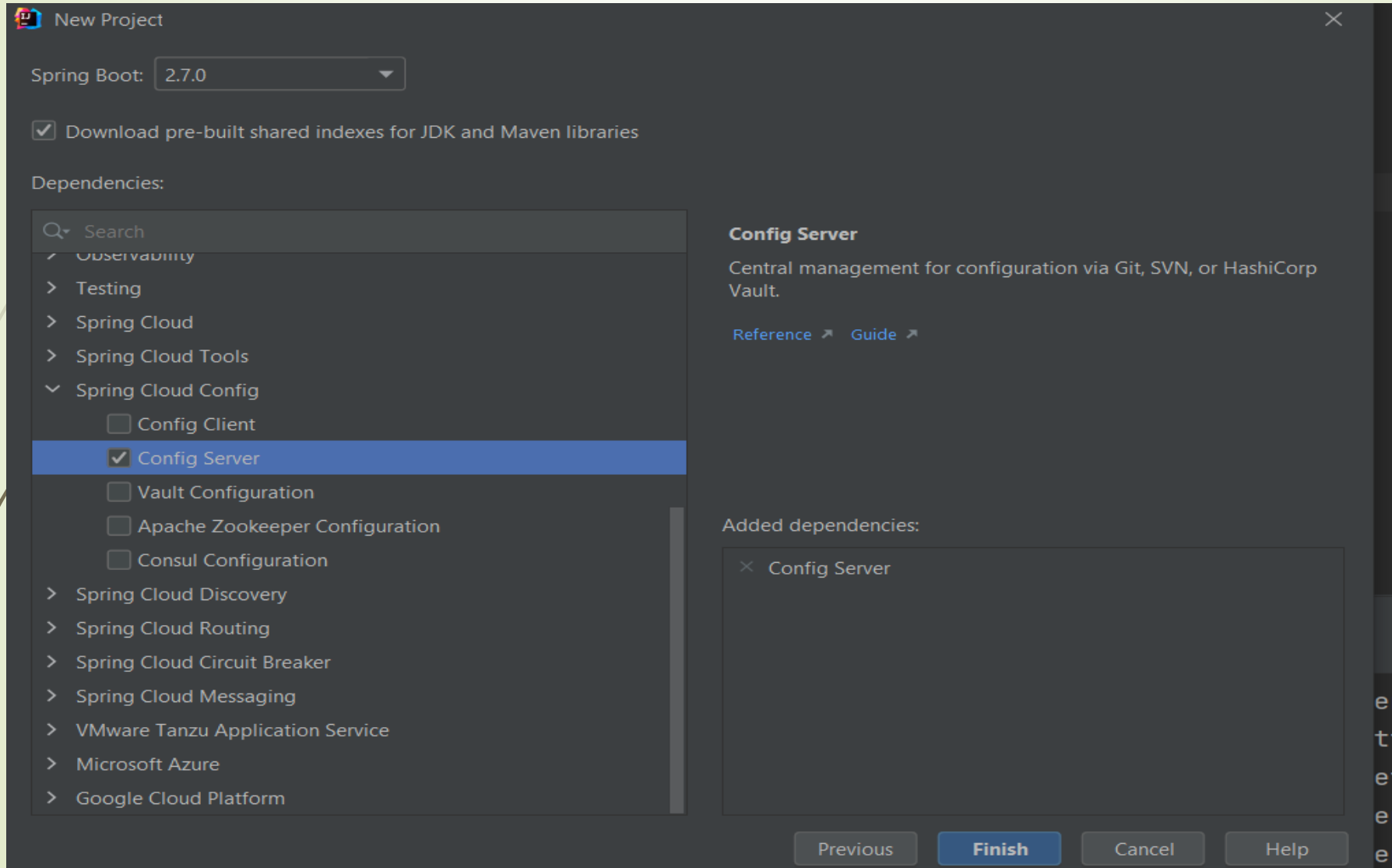


The image shows the 'New Project' dialog in IntelliJ IDEA, configured for a Spring Initializr project. The 'Spring Initializr' option is selected in the left-hand category list. The right-hand panel contains the following configuration fields:

- Server URL:** start.spring.io (with a settings gear icon)
- Name:** Config
- Location:** D:\cours-ms
- Language:** Java (selected), Kotlin, Groovy
- Type:** Maven (selected), Gradle
- Group:** tg.ceel
- Artifact:** Config
- Package name:** tg.ceel.config
- Project SDK:** 1.8 Oracle OpenJDK version 1.8.0_202
- Java:** 8
- Packaging:** Jar (selected), War

At the bottom of the dialog are four buttons: 'Previous', 'Next' (highlighted in blue), 'Cancel', and 'Help'.

Service : config



Service : Config

► Config Server:

```
package tg.ceel.configuration;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.config.server.EnableConfigServer;

@SpringBootApplication
@EnableConfigServer
public class ConfigurationApplication {

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


Service : Config

➤ **Eureka Server:**

`server.port=8889`

`spring.cloud.config.server.git.uri=file://${user.home}/ceel/config`

Service : registre

- ▀ il permet l'équilibrage de charge côté client
- ▀ dissocie les fournisseurs de services des consommateurs sans avoir besoin de DNS

- ▀ **Eureka Server:**

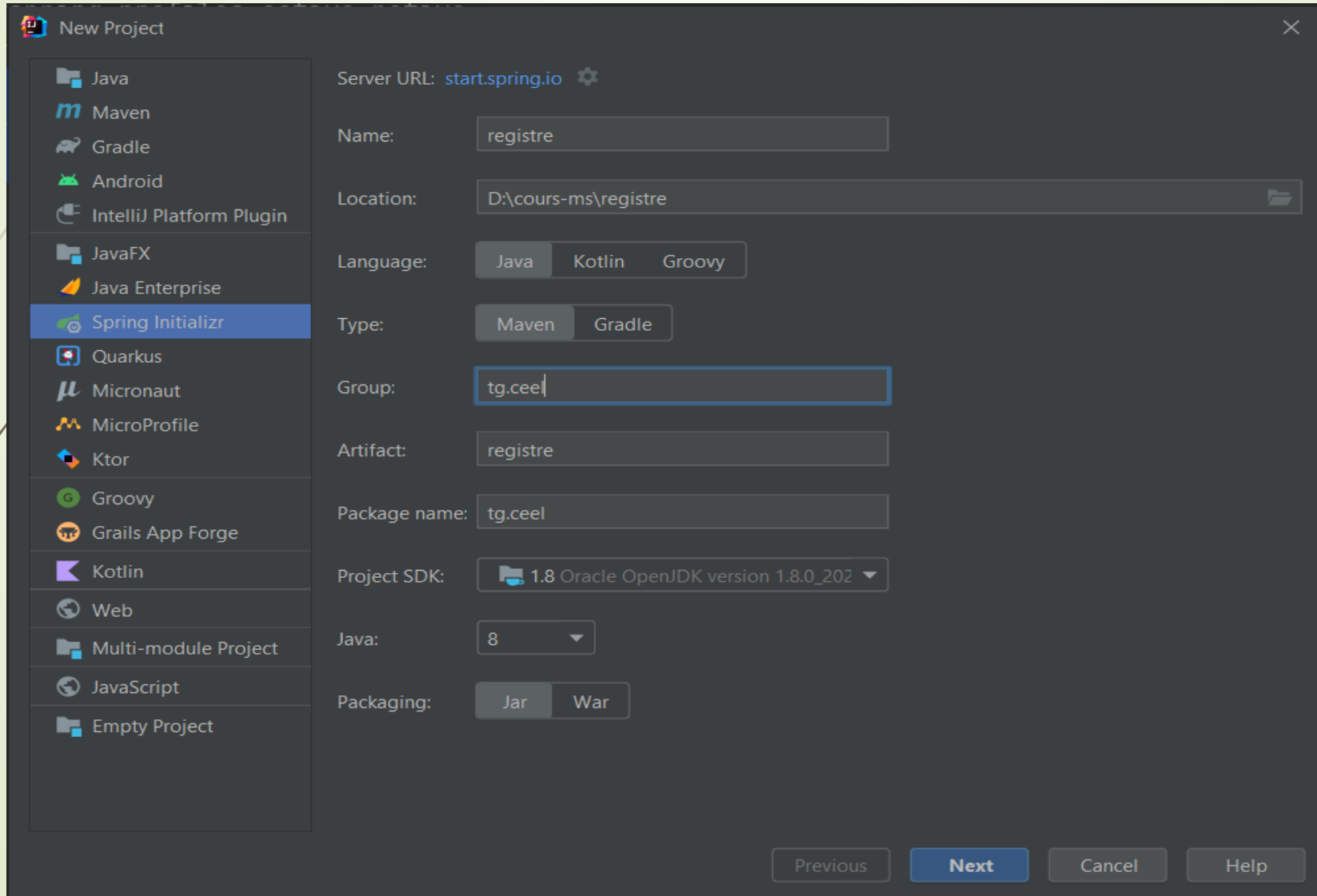
```
<dependency>
```

```
    <groupId>org.springframework.cloud</groupId>
```

```
    <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>
```

```
</dependency>
```

Service : registre



The image shows the 'New Project' dialog in IntelliJ IDEA. On the left, a list of project templates is shown, with 'Spring Initializr' selected. On the right, various project configuration fields are visible, including 'Server URL', 'Name', 'Location', 'Language', 'Type', 'Group', 'Artifact', 'Package name', 'Project SDK', 'Java', and 'Packaging'. The 'Name' field is set to 'registre', 'Location' to 'D:\cours-ms\registre', 'Language' to 'Java', 'Type' to 'Maven', 'Group' to 'tg.ceel', 'Artifact' to 'registre', 'Package name' to 'tg.ceel', 'Project SDK' to '1.8 Oracle OpenJDK version 1.8.0_202', 'Java' to '8', and 'Packaging' to 'Jar'. The 'Next' button is highlighted in blue.

New Project

Server URL: start.spring.io

Name: registre

Location: D:\cours-ms\registre

Language: Java Kotlin Groovy

Type: Maven Gradle

Group: tg.ceel

Artifact: registre

Package name: tg.ceel

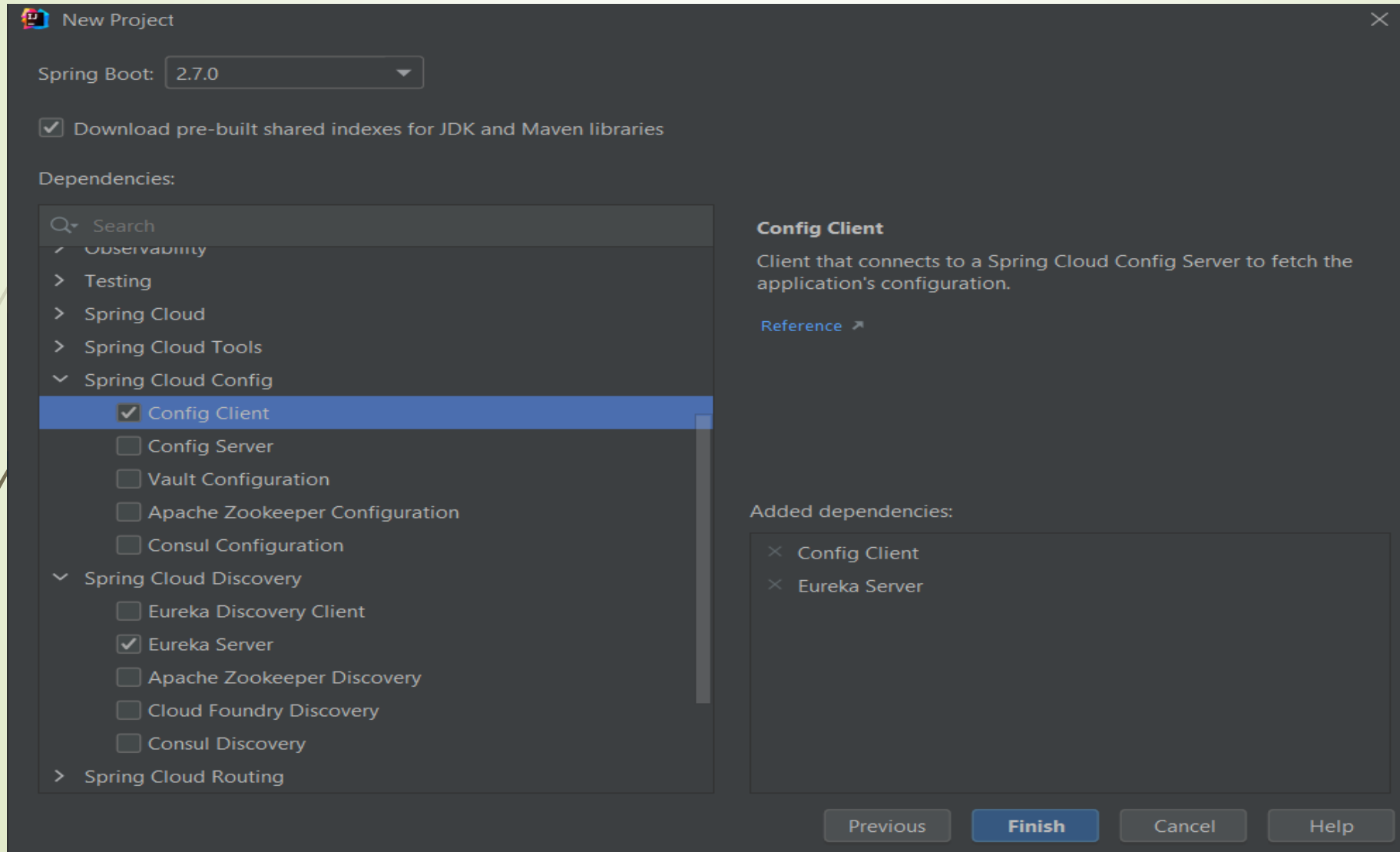
Project SDK: 1.8 Oracle OpenJDK version 1.8.0_202

Java: 8

Packaging: Jar War

Previous Next Cancel Help

Service : registre



Fichier registre.properties

➤ **server.port=8761**

Le port pour les requêtes HTTP

➤ **eureka.client.register-with-eureka=false**

Si cette propriété est définie sur true, alors pendant le démarrage du serveur, le client intégré (le serveur lui-même) essaiera de s'enregistrer auprès du serveur Eureka.

➤ **eureka.client.fetch-registry=false**

Si cette propriété est sur true, le client intégré essaiera de récupérer le registre Eureka

Registre

```
package tg.ceel;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;

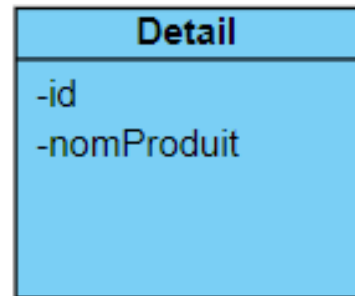
@SpringBootApplication
@EnableEurekaServer
public class RegistreApplication {
    public static void main(String[] args) {
        SpringApplication.run(RegistreApplication.class, args);
    }
}
```

Registre: contenu du fichier application.properties

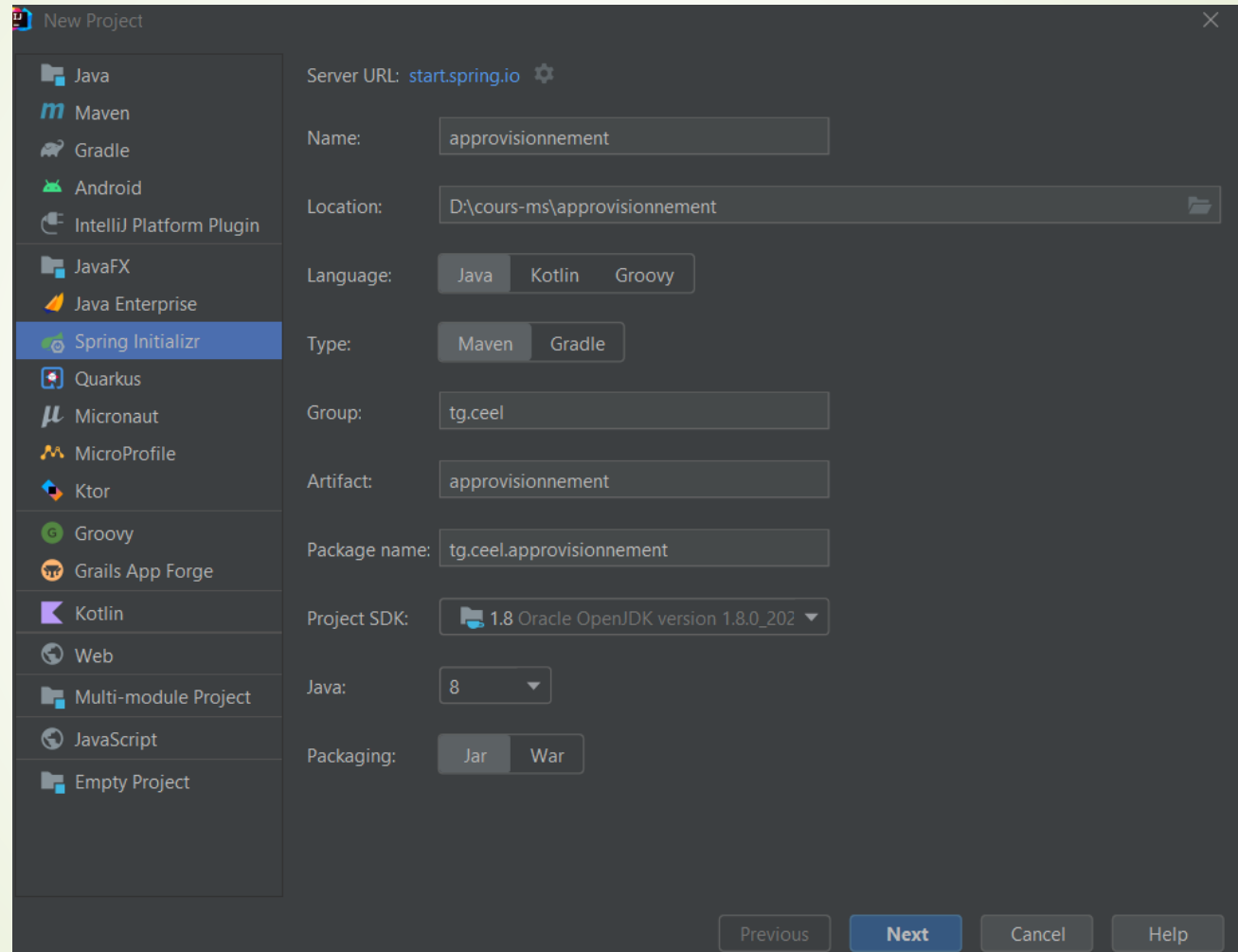
spring.application.name=register

spring.config.import=optional:configserver:http://localhost:8889/

Service Approvisionnement: Diagramme de classes



Service Approvisionnement



The image shows the 'New Project' dialog in IntelliJ IDEA. On the left, a list of project types is shown, with 'Spring Initializr' selected. On the right, various project configuration fields are visible, including 'Server URL', 'Name', 'Location', 'Language', 'Type', 'Group', 'Artifact', 'Package name', 'Project SDK', 'Java', and 'Packaging'. The 'Next' button is highlighted in blue.

Server URL: start.spring.io

Name:

Location:

Language: ☒ Java ☐ Kotlin ☐ Groovy

Type: ☒ Maven ☐ Gradle

Group:

Artifact:

Package name:

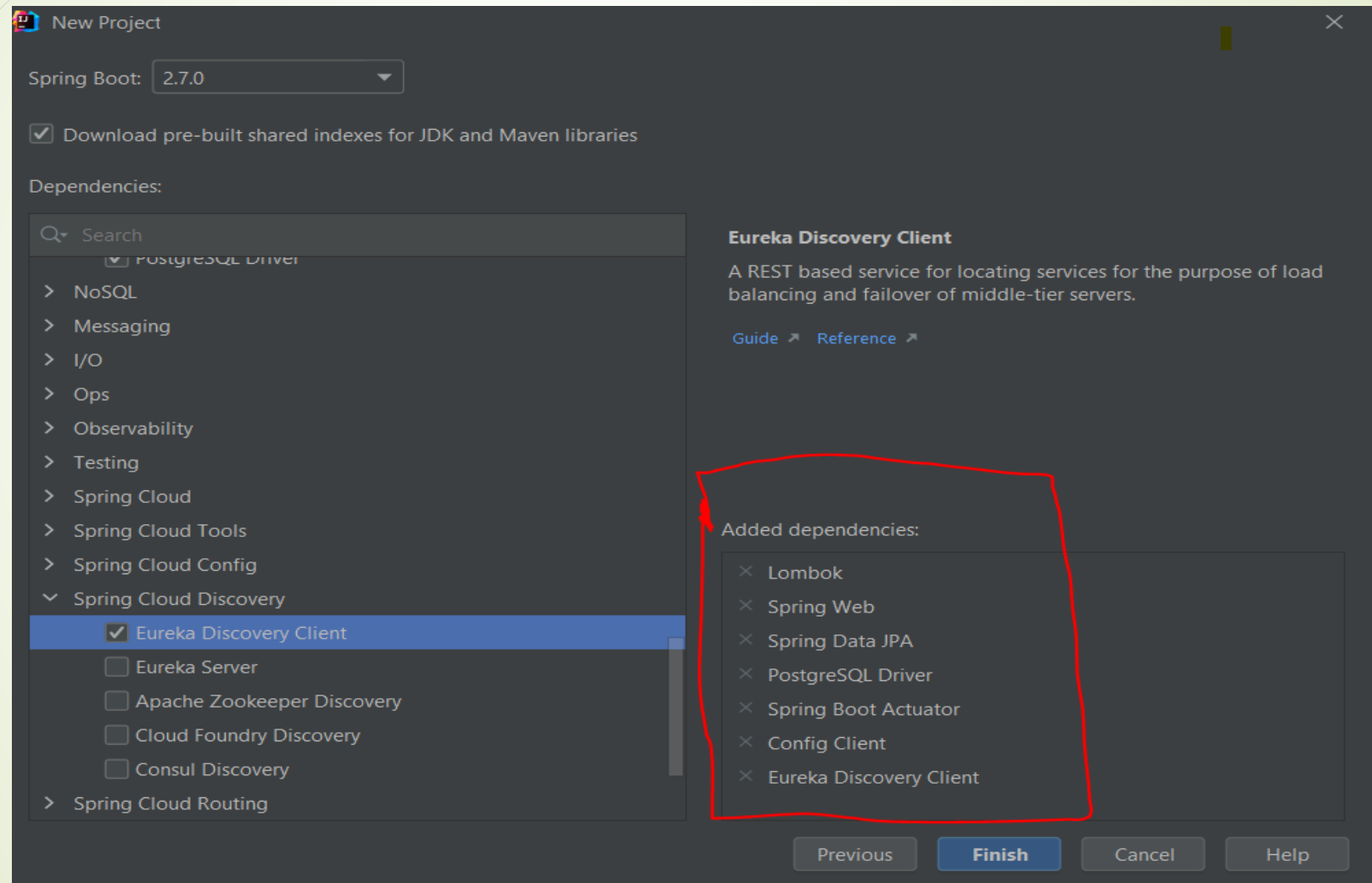
Project SDK:

Java:

Packaging: ☒ Jar ☐ War

Buttons: Previous, **Next**, Cancel, Help

Service Approvisionnement



Service Approvisionnement: Dépendances

- `spring-boot-starter-web` (Utilise Tomcat comme conteneur intégré par défaut)
- `spring-boot-starter-actuator` (utilisé essentiellement pour administrer les applications (surveillance, métriques, analyse du trafic HTTP,...))
- `spring-boot-starter-data-jpa` (une implémentation de la couche d'accès aux données embarquant l'ORM Hibernate)
- `spring-cloud-starter-config` (utilisé pour communiquer avec le serveur de configuration)
- `spring-cloud-starter-netflix-eureka-client` (Permet aux services de s'enregistrer auprès du registre)
- `postgresql` (embarque le pilote de communication avec la base de données PostgreSQL)
- `lombok`

Service Approvisionnement: config

- `server.port=8081`
- `spring.datasource.url= jdbc:postgresql://localhost:5432/approvisionnement`
- `spring.datasource.username=postgres`
- `spring.datasource.password=postgres`
- `spring.jpa.database-platform=org.hibernate.dialect.PostgreSQL9Dialect`
- `spring.jpa.hibernate.ddl-auto=update`
- `eureka.instance.hostname=localhost`

Service Approvisionnement

```
package tg.ceel.approvisionnement;  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;
```

```
@SpringBootApplication
```

```
@EnableDiscoveryClient
```

```
public class ApprovisionnementApplication {
```

```
    public static void main(String[] args) {
```

```
        SpringApplication.run(ApprovisionnementApplication.class, args);
```

```
    }
```

```
}
```

Service Approvisionnement: swagger

➤ Dépendances

<dependency>

<groupId>org.springdoc</groupId>

<artifactId>springdoc-openapi-ui</artifactId>

<version>1.6.8</version>

</dependency>

➤ Propriétés

springdoc.api-docs.enabled=true

springdoc.swagger-ui.enabled=true

springdoc.swagger-ui.path=/swagger-ui.html

Service Approvisionnement: swagger

➤ Informations

```
@OpenAPIDefinition(info =
```

```
@Info(title = "Service d'approvisionnement", version = "1.0", description =  
"Approvisionnement API v1.0")
```

```
)
```

Service Approvisionnement: class Produit

```
package tg.ceel.approvisionnement.entites;  
import lombok.*;  
  
@Data  
@NoArgsConstructor  
@AllArgsConstructor  
@Getter  
@Setter  
@ToString  
public class Produit {  
    private Long id;  
    private String code;  
    private String nomProduit;  
}
```

Service Approvisionnement: ProduitRepository

```
import org.springframework.data.jpa.repository.JpaRepository;  
import tg.ceel.approvisionnement.entites.Produit;
```

```
public interface ProduitRepository extends JpaRepository<Produit, Long> {  
  
}
```

Service Approvisionnement: API de création de produit

➤ Contrôleur

```
package tg.ceel.approvisionnement.ws;
```

```
import org.springframework.web.bind.annotation.RestController;
```

```
import tg.ceel.approvisionnement.entites.Produit;
```

```
import tg.ceel.approvisionnement.reporitories.ProduitRepository;
```

```
@RestController
```

```
@RequestMapping("produits")
```

```
public class ProduitController {
```

```
    private final ProduitRepository produitRepository;
```

```
    public ProduitController(ProduitRepository produitRepository) {
```

```
        this.produitRepository = produitRepository;
```

```
    }}
```

Service Approvisionnement: API de création de produit

- **API de création de produit**

```
@Operation(summary = "Permet l'enregistrement des produits")
@ApiResponses(value = {
    @ApiResponse(responseCode = "201", description = "Le produit enregistrée avec succès",
        content = {@Content(mediaType = "application/json",
            schema = @Schema(implementation = Produit.class))}),
    @ApiResponse(responseCode = "400", description = "L'objet produit envoyé ou le code null ou le nom est null",
        content = @Content),
    @ApiResponse(responseCode = "500", description = "Erreur interne au serveur",
        content = @Content)})
@PostMapping()
public ResponseEntity<?> createProduct(@RequestBody Produit produit) {
    try {
        if (produit == null) {
            return new ResponseEntity<>(body: "L'objet produit à créer est null", HttpStatus.BAD_REQUEST);
        }
        if (produit.getCode() == null) {
            return new ResponseEntity<>(new String(original: "Le code du produit n'est pas indiqué"), HttpStatus.BAD_REQUEST);
        }
        if (produit.getNomProduit() == null) {
            return new ResponseEntity<>(body: "Le nom du produit n'est pas indiqué", HttpStatus.BAD_REQUEST);
        }
        return new ResponseEntity<>(produitRepository.save(produit), HttpStatus.CREATED);
    } catch (Exception e) {
        e.printStackTrace();
        return new ResponseEntity<>(e.getMessage(), HttpStatus.INTERNAL_SERVER_ERROR);
    }
}
```

Service Approvisionnement: API liste des produits

- **API de listing de produit**

```
@Operation(summary = "Permet de récupérer la liste des produits")
@ApiResponses(value = {
    @ApiResponse(responseCode = "200", description = "Liste chargée avec succès",
        content = {@Content(mediaType = "application/json",
            array = @ArraySchema(schema = @Schema(implementation = Produit.class)))}),
    @ApiResponse(responseCode = "500", description = "Erreur interne au serveur",
        content = @Content)})
```



```
@GetMapping
```

```
public ResponseEntity<?> getListProduct(){
    try {
        return new ResponseEntity<>(produitRepository.findAll(), HttpStatus.OK);
    } catch (Exception e) {
        e.printStackTrace();
        return new ResponseEntity<>(e.getMessage(), HttpStatus.INTERNAL_SERVER_ERROR);
    }
}
```


Service Approvisionnement: API récupération d'un produit à partir de son ID

- **API de récupération d'un produit à partir de son ID**

```

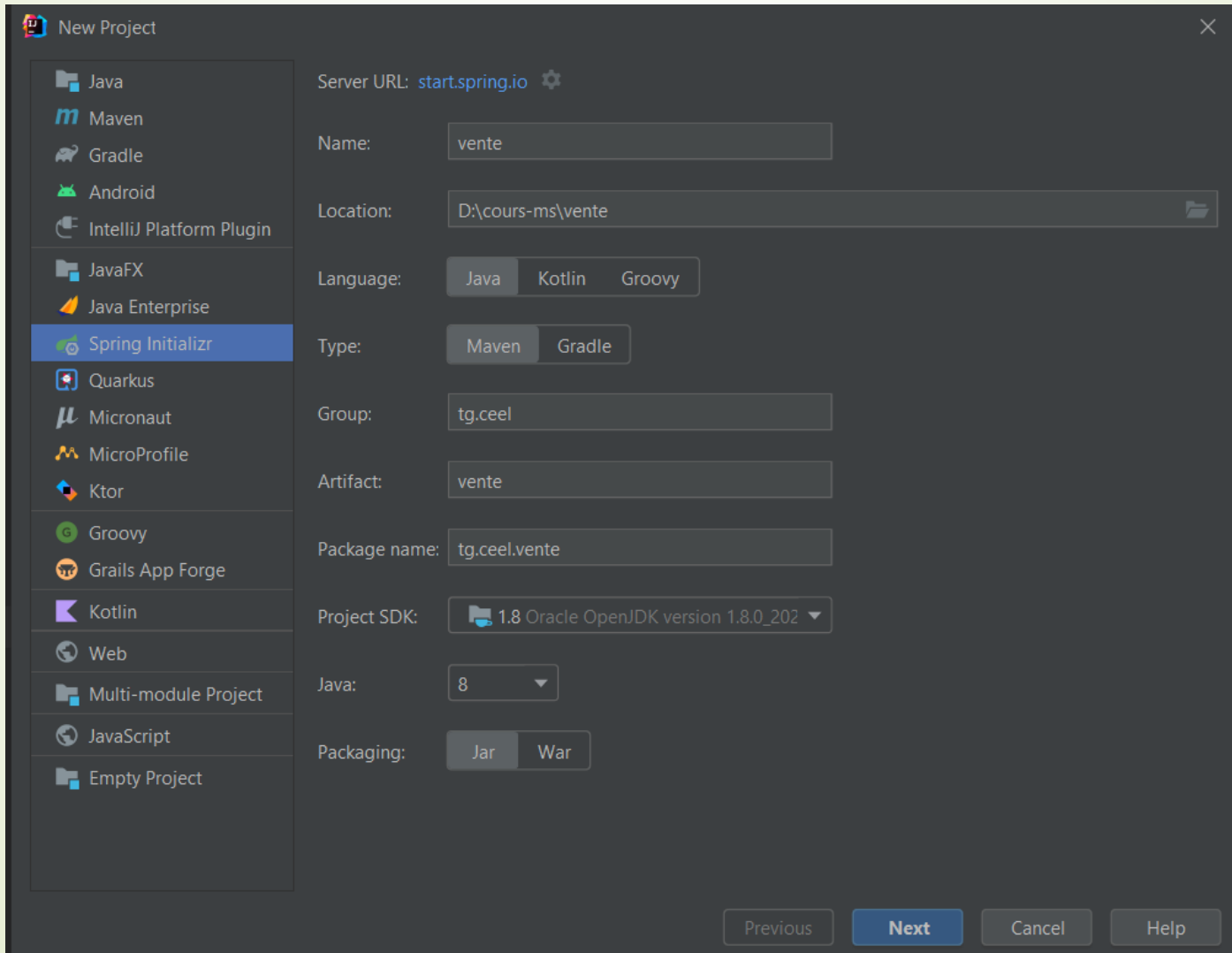
@Operation(summary = "Permet de récupérer un produit par son identifiant")
@ApiResponses(value = {
    @ApiResponse(responseCode = "200", description = "Liste chargée avec succès",
        content = {@Content(mediaType = "application/json",
            array = @ArraySchema(schema = @Schema(implementation = Produit.class)))}),
    @ApiResponse(responseCode = "404", description = "Aucun produit trouvé avec l'identifiant fourni ",
        content = @Content),
    @ApiResponse(responseCode = "400", description = "Aucun id de produit fourni",
        content = @Content),
    @ApiResponse(responseCode = "500", description = "Erreur interne au serveur",
        content = @Content)})
@GetMapping("/{id}")
public ResponseEntity<?> getProductById(@PathVariable("id") Long id){
    if (id == null) {
        return new ResponseEntity<>( body: "Aucun produit trouvé avec l'identifiant fourni", HttpStatus.BAD_REQUEST);
    }
    try {
        return new ResponseEntity<>(produitRepository.findById(id).orElse( other: null), HttpStatus.OK);
    } catch (Exception e) {
        e.printStackTrace();
        return new ResponseEntity<>(e.getMessage(), HttpStatus.INTERNAL_SERVER_ERROR);
    }
}

```

Service vente: Diagramme de classes



Service vente



The image shows the 'New Project' dialog in IntelliJ IDEA. On the left, a list of project templates is shown, with 'Spring Initializr' selected. On the right, various project configuration fields are visible, including 'Server URL', 'Name', 'Location', 'Language', 'Type', 'Group', 'Artifact', 'Package name', 'Project SDK', 'Java', and 'Packaging'. The 'Next' button is highlighted in blue.

New Project

Server URL: start.spring.io ⚙️

Name:

Location:

Language: ☒ Java ☐ Kotlin ☐ Groovy

Type: ☒ Maven ☐ Gradle

Group:

Artifact:

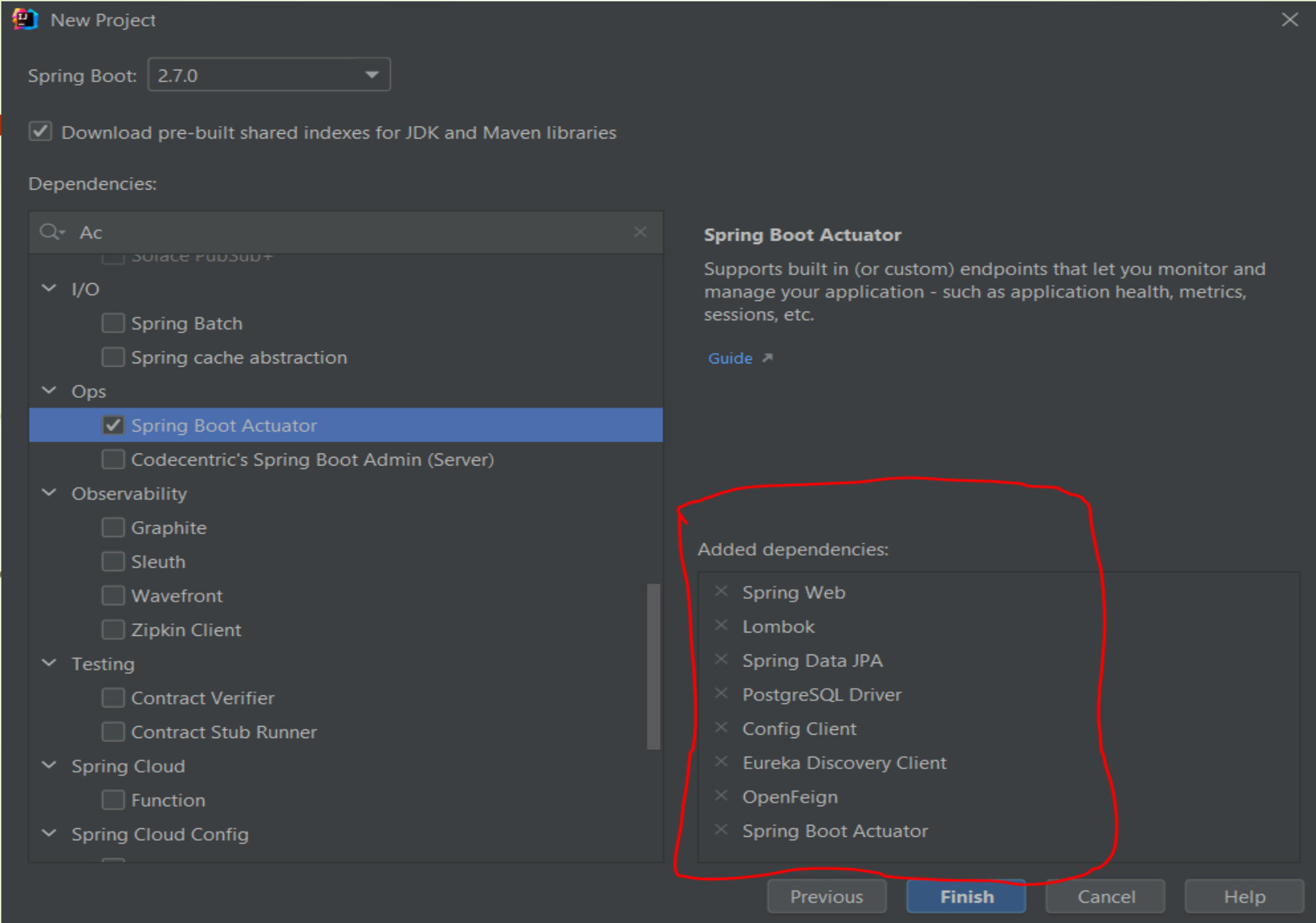
Package name:

Project SDK:

Java:

Packaging: ☒ Jar ☐ War

Previous **Next** Cancel Help



Service vente: Dépendances

- `spring-boot-starter-web`
- `spring-boot-starter-actuator`
- `spring-boot-starter-data-jpa`
- `spring-cloud-starter-config`
- `spring-cloud-starter-netflix-eureka-client`
- `postgresql`
- `lombok`
- `spring-cloud-starter-openfeign`

Service vente: configuration

- `server.port=8082`
- `spring.datasource.url= jdbc:postgresql://localhost:5432/vente`
- `spring.datasource.username=postgres`
- `spring.datasource.password=postgres`
- `spring.jpa.database-platform=org.hibernate.dialect.PostgreSQL9Dialect`
- `spring.jpa.hibernate.ddl-auto=update`
- `eureka.instance.hostname=localhost`

Service vente

```
package tg.ceel.vente;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;

@SpringBootApplication
@EnableDiscoveryClient
public class VenteApplication {

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


Service vente: swagger

➤ Dépendance

<dependency>

<groupId>org.springdoc</groupId>

<artifactId>springdoc-openapi-ui</artifactId>

<version>1.6.8</version>

</dependency>

Service vente: swagger

■ Titre de la documentation

@SpringBootApplication

@EnableDiscoveryClient

@OpenAPIDefinition(info =

@Info(title = "Service de vente", version = "1.0", description = "Vente API v1.0")

)

public class VenteApplication {

 public static void main(String[] args) {

 SpringApplication.run(VenteApplication.class, args);

 }

}

Service vente: classe Vente

```
package tg.ceel.vente.entities;

import lombok.*;

import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;
import java.util.Date;

@Data
@NoArgsConstructor
@AllArgsConstructor
@Getter
@Setter
@ToString
@Entity
@Table(name = "ventes")
public class Vente {
    @Id
    private String numero;
    private Date dateVente;
}
```

Service vente: classe VenteRepository

```
package tg.ceel.vente.repositories;

import org.springframework.data.jpa.repository.JpaRepository;
import tg.ceel.vente.entities.Vente;

public interface VenteRepository extends JpaRepository <Vente, String>{
}
```

Service vente: classe Detail

```
import lombok.*;

import javax.persistence.*;

@Data
@NoArgsConstructor
@AllArgsConstructor
@Getter
@Setter
@ToString
@Entity
@Table(name = "details")
public class Detail {
    @Id
    private Long id;
    private Long idProduit;
    private Integer quantite;
    private Long prixVente;
    @ManyToOne
    @JoinColumn(name = "vente_numero")
    private Vente vente;
}
```

Service vente: classe DetailRepository

```
package tg.ceel.vente.repositories;

import org.springframework.data.jpa.repository.JpaRepository;
import tg.ceel.vente.entities.Detail;

public interface DetailRepository extends JpaRepository<Detail, Long> {
}
```

Les Data Transfer Object (DTOs) et mappers

- DTO est un design pattern (patron de conception)
- Le but des DTO est de simplifier le transfert de données entre les différentes couches d'une application.
- Les mappers nous permettent d'appliquer les DTO en transformant les DTO en entité et vice versa

Service vente: classe VenteDTO

```
package tg.cceel.vente.dto;

import lombok.*;

import java.util.Date;
import java.util.List;

@Data
@NoArgsConstructor
@AllArgsConstructor
@Getter
@Setter
@ToString
public class VenteDTO {
    private String numero;
    private Date dateVente;
    List<DetatilDTO> detatils;
}
```


Service vente: classe DetailDTO

```
package tg.ceel.vente.dto;

import lombok.*;

@Data
@NoArgsConstructor
@AllArgsConstructor
@Getter
@Setter
@ToString
public class DetailDTO {
    private Long id;
    private Long idProduit;
    private Integer quantite;
    private Long prixVente;
}
```

Service vente: l'interface VenteMapper

```
package tg.ceel.vente.mappers;

import tg.ceel.vente.dto.VenteDTO;
import tg.ceel.vente.entities.Vente;

public interface VenteMapper {
    Vente venteDtoToVente(VenteDTO dto);
}
```

Service vente: la classe VenteMapperImpl

```
import java.util.List;
import java.util.stream.Collectors;

@Service
public class VenteMapperImp implements VenteMapper{
    @Override
    public Vente venteDtoToVente(VenteDTO dto) {
        if (dto==null){ return null; }
        Vente vente = new Vente();
        vente.setDateVente(dto.getDateVente());
        vente.setNumero(dto.getNumero());
        List<Detail> details = dto.getDetatils().stream().map(d -> {
            Detail detail = new Detail();
            detail.setVente(vente);
            detail.setIdProduit(detail.getIdProduit());
            detail.setQuantite(d.getQuantite());
            detail.setPrixVente(d.getPrixVente());
            return detail;
        }).collect(Collectors.toList());
        vente.setDetails(details);
        return null;
    }
}
```

Service vente: la classe DetailDTO

```
package tg.ceel.vente.dto;

import lombok.*;

@Data
@NoArgsConstructor
@AllArgsConstructor
@Getter
@Setter
@ToString
public class DetailDTO {
    private Long id;
    private Long idProduit;
    private Integer quantite;
    private Long prixVente;
}
```

Service vente: l'interface DetailMapper

```
package tg.ceel.vente.mappers;

import tg.ceel.vente.dto.DetailDTO;
import tg.ceel.vente.entities.Detail;

public interface DetailMapper {
    Detail detailDtoToDetail(DetailDTO dto);
    DetailDTO detailToDetailDto(Detail detail);
}
```

Service vente: la classe DetailMapperImpl

```
@Service
public class DetailMapperImp implements DetailMapper {
    @Override
    public Detail detailDtoToDetail(DetailDTO dto) {
        if (dto.equals(null)) return null;
        Detail detail = new Detail();
        detail.setPrixVente(dto.getPrixVente());
        detail.setQuantite(dto.getQuantite());
        detail.setIdProduit(dto.getIdProduit());
        return detail;
    }

    @Override
    public DetailDTO detailToDetailDTO(Detail detail) {
        if(detail.equals(null)) return null;
        DetailDTO dto = new DetailDTO();
        dto.setQuantite(detail.getQuantite());
        dto.setPrixVente(detail.getPrixVente());
        dto.setId(detail.getId());
        dto.setIdProduit(detail.getIdProduit());
        return dto;
    }
}
```

Service vente: l'interface VenteService

```
package tg.ceel.vente.service;

import tg.ceel.vente.dto.VenteDTO;

import java.util.List;

public interface VenteService {
    VenteDTO createVente(VenteDTO dto);
    List<VenteDTO> getListeVente();
    VenteDTO getByNumeroVente(String numero);
}
```

Service vente: la class VenteServiceImp

```
import tg.ceel.vente.entities.Vente;
import tg.ceel.vente.mappers.DetailMapper;
import tg.ceel.vente.mappers.VenteMapper;
import tg.ceel.vente.repositories.DetailRepository;
import tg.ceel.vente.repositories.VenteRepository;

import java.util.List;
import java.util.stream.Collectors;
```

```
@Service
public class VenteServiceImp implements VenteService {
    private final DetailMapper detailMapper;
    private final VenteMapper venteMapper;
    private final DetailRepository detailRepository;
    private final VenteRepository venteRepository;

    public VenteServiceImp(DetailMapper detailMapper, VenteMapper venteMapper,
        DetailRepository detailRepository, VenteRepository venteRepository) {
        this.detailMapper = detailMapper;
        this.venteMapper = venteMapper;
        this.detailRepository = detailRepository;
        this.venteRepository = venteRepository;
    }
}
```

```
@Override
public VenteDTO createVente(VenteDTO dto) {
    Vente vente = venteMapper.venteDtoToVente(dto);
    List<Detail> details = vente.getDetails();
    vente = venteRepository.save(vente);
    for (Detail d : details) {
        d.setVente(vente);
    }
    details = detailRepository.saveAll(details);
    vente.setDetails(details);
    return venteMapper.venteToVenteDto(vente);
}
```

```
@Override
public List<VenteDTO> getListeVente() {
    return venteRepository.findAll()
        .stream()
        .map(d -> venteMapper.venteToVenteDto(d))
        .collect(Collectors.toList());
}
```

```
@Override
public VenteDTO getByNumeroVente(String numero) {
    return venteMapper.venteToVenteDto(venteRepository.findById(numero).orElse(null));
}
```


Spring cloud OpenFeign

- ▀ permet aux développeurs de créer des requêtes HTTP en se basant sur des interfaces annotées sans avoir à recourir à des clients comme OkHttp et RestTemplate pour communiquer entre services

- ▀ Dépendance

<dependency>

<groupId>org.springframework.cloud**</groupId>**

<artifactId>spring-cloud-starter-openfeign**</artifactId>**

</dependency>

Spring cloud OpenFeign

- ➡ Configuration dans le service vente
 - Annoté la classe main avec

```
@SpringBootApplication
@EnableDiscoveryClient
@EnableFeignClients
@OpenAPIDefinition(info =
    @Info(title = "Service de vente", version = "1.0",
        description = "Vente API v1.0")
)
public class VenteApplication {

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

Spring cloud OpenFeign

- Créer une interface et l'annoter avec **@FeignClient**

```
package tg.ceel.vente.service.feigns;

import org.springframework.cloud.openfeign.FeignClient;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;

import java.util.List;

@FeignClient(name = "approvisionnement")
public interface ApprovisionnementFeign {
    @GetMapping("produits/{id}")
    Produit getProductById(@PathVariable("id") Long id);
    @GetMapping("produits")
    List<Produit> getListProduct();
}
```

Premier API

1

```

@RestController
@RequestMapping("/ventes")
public class VenteController {

    private final VenteService venteService;
    private final ApprovisionnementFeign approvisionnementFeign;

    public VenteController(VenteService venteService,
                           ApprovisionnementFeign approvisionnementFeign) {
        this.venteService = venteService;
        this.approvisionnementFeign = approvisionnementFeign;
    }

```

2

```

@PostMapping
public ResponseEntity<> createVente(@RequestBody VenteDTO venteDTO) {
    try {
        if (venteDTO == null) {
            return new ResponseEntity<>(body: "L'objet vente soumis est null",
                                       HttpStatus.BAD_REQUEST);
        }

        Long id = null;
        for (DetailDTO dto : venteDTO.getDetails()) {
            Produit produit = approvisionnementFeign.getProductById(dto.getIdProduit());
            if (produit == null) {
                id = dto.getIdProduit();
                continue;
            }
        }

        if (id != null) {
            return new ResponseEntity<>(body: "Le produit dont l'identifiant est " + id
                                         + " n'existe pas", HttpStatus.NOT_FOUND);
        }

        return new ResponseEntity<>(venteService.createVente(venteDTO), HttpStatus.CREATED);
    } catch (Exception e) {
        return new ResponseEntity<>(e.getMessage(), HttpStatus.INTERNAL_SERVER_ERROR);
    }
}

```

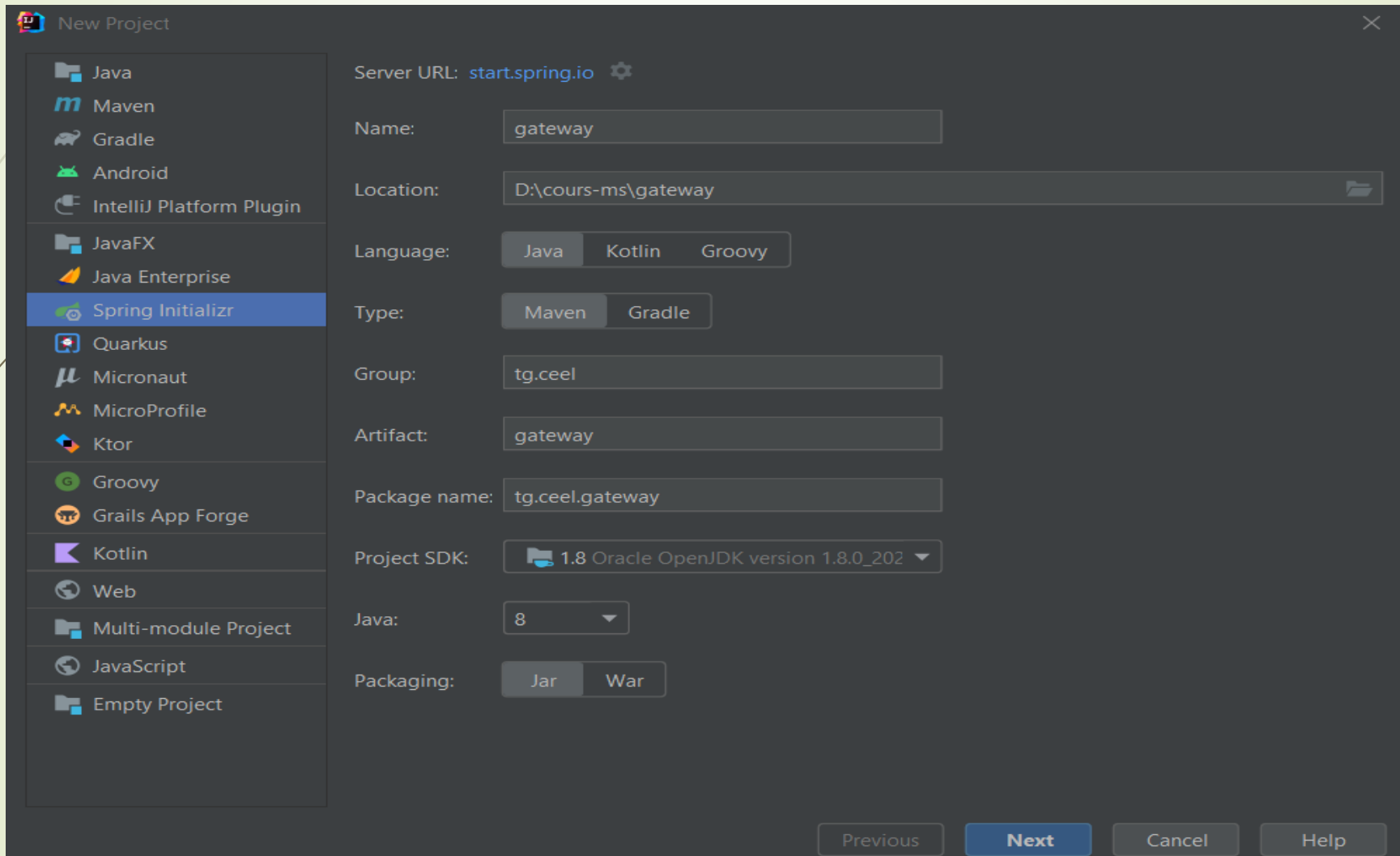
Le Gateway

- Fourni les routes vers les APIs ou microservices
- Permet la prise en charge de la sécurité, la surveillance de la santé de microservice

Dépendance

- `spring-cloud-starter-gateway`

Le Gateway



The image shows the 'New Project' dialog in IntelliJ IDEA. On the left, a list of project types is shown, with 'Spring Initializr' selected. On the right, various project configuration fields are visible, including 'Server URL', 'Name', 'Location', 'Language', 'Type', 'Group', 'Artifact', 'Package name', 'Project SDK', 'Java', and 'Packaging'. The 'Next' button is highlighted in blue.

New Project

Server URL: start.spring.io ⚙️

Name:

Location:

Language: ☒ Java ☐ Kotlin ☐ Groovy

Type: ☒ Maven ☐ Gradle

Group:

Artifact:

Package name:

Project SDK:

Java:

Packaging: ☒ Jar ☐ War

[Previous](#) **Next** [Cancel](#) [Help](#)

Spring Boot: 2.7.0 ▼

☒ Download pre-built shared indexes for JDK and Maven libraries

Dependencies:

Search

- SQL
- > NoSQL
- > Messaging
- > I/O
- > Ops
- > Observability
- > Testing
- > Spring Cloud
- > Spring Cloud Tools
- > Spring Cloud Config
- > Spring Cloud Discovery
- ▼ Spring Cloud Routing
 - ☒ Gateway
 - ☐ OpenFeign
 - ☐ Cloud LoadBalancer
- > Spring Cloud Circuit Breaker
- > Spring Cloud Messaging

Gateway

Provides a simple, yet effective way to route to APIs and provide cross cutting concerns to them such as security, monitoring/metrics, and resiliency.

[Reference](#) ➤ [Guide](#) ➤

Added dependencies:

- ✕ Gateway
- ✕ Eureka Discovery Client
- ✕ Config Client
- ✕ Spring Boot Actuator

Previous

Finish

Cancel

Help

Le Gateway

#port

server.port=8080

Application name

spring.application.name=gateway

#Pour le service approvisionnement

spring.cloud.gateway.routes[0].id=approvisionnement

spring.cloud.gateway.routes[0].uri=http://localhost:8081/

spring.cloud.gateway.routes[0].predicates[0]=Path=/produits/**

Pour le service vente

spring.cloud.gateway.routes[1].id=vente

spring.cloud.gateway.routes[1].uri=http://localhost:8082/

spring.cloud.gateway.routes[1].predicates[0]=Path=/ventes/**

Eureka

eureka.instance.hostname=localhost

Le Gateway

```
package tg.ceel.gateway;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;

@SpringBootApplication
@EnableDiscoveryClient
public class GatewayApplication {

    public static void main(String[] args) { SpringApplication.run(GatewayApplication.class, args); }

}
```

Le Gateway

```
- spring:
-   application:
-       name: gateway
-   config:
-       import: optional:configserver:http://localhost:8889/

- management:
-   endpoints:
-       web:
-           exposure:
-               include: '*'
```