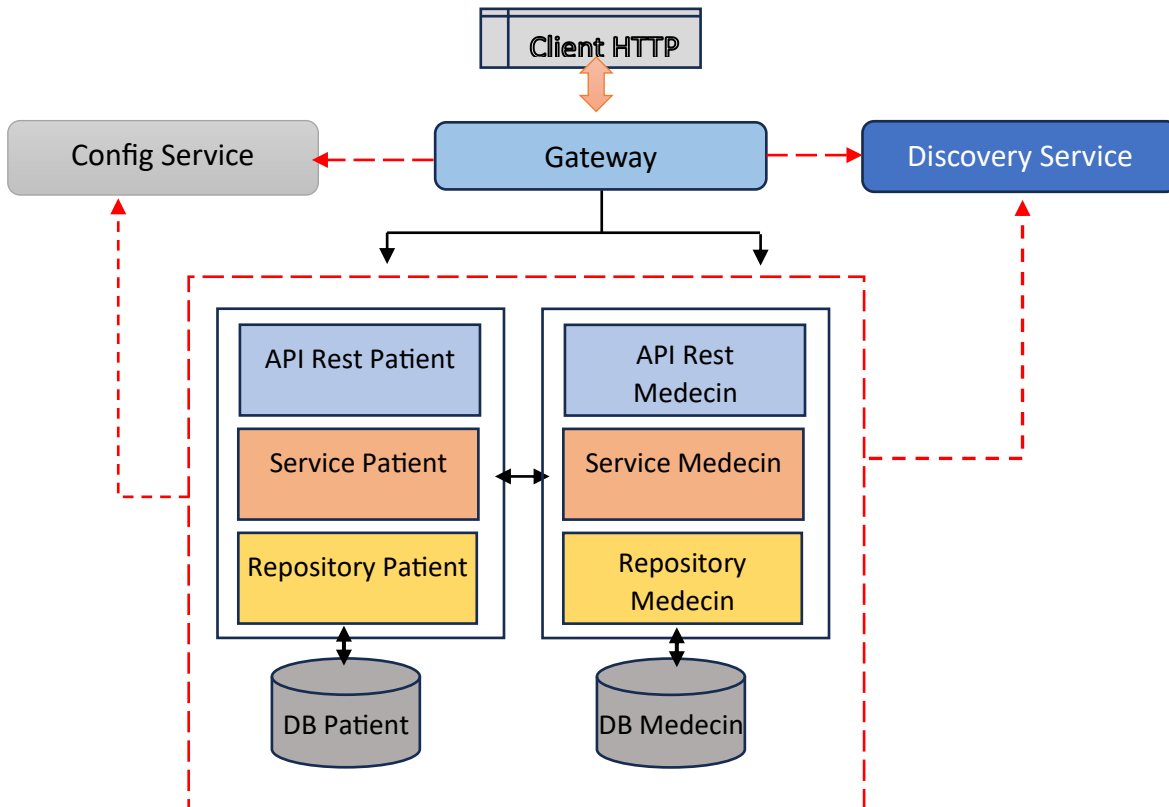


TP5 Microservices

Architecture de la solution



On va procéder à la création des services suivants :

Patient-Service : un microservice qui gère l'entité Patient par des API Rest

Medecin-Service : un microservice qui gère l'entité Medecin par des API Rest

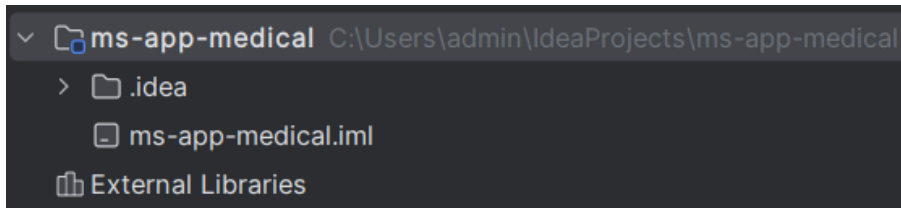
Gateway : Passerelle qui s'occupe d'acheminer une requête HTTP vers l'un des services en utilisant Spring Cloud Gateway

Discovery-Service : service qui permet d'enregistrer des instances de microservices à des fins de découverte par d'autres services. Ce service utilise Eureka Server

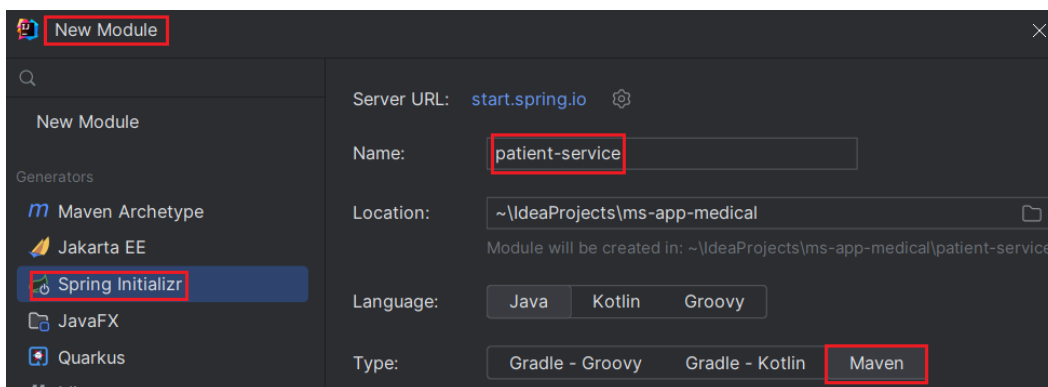
Config-Service : Service de configuration, dont le rôle est de centraliser les fichiers de configuration des différents microservices en un seul endroit.

Création des microservices :

Dans IntelliJ commencer par créer un projet vide `ms-app-medical` qui contiendra tous les microservices et services techniques chacun comme un module.



Ajouter un premier module de type Spring Initializr dans ce projet pour le premier microservice `patient-service` :



Ce microservice contiendra les dépendances suivantes :

Spring Web : pour exposer des API Rest

Spring Data JPA : pour utiliser une BD relationnelle

Mysql Driver : Le pilote de la BD

Lombok : pour réduire le code des entités en générant les getters et setters ...

Eureka Discovery Client : pour que le microservice puisse s'enregistrer dans Discovery Service à son démarrage.

Spring Boot Actuator : permet le monitoring des microservices

Config Client : afin de récupérer la configuration du microservice à partir du Config Service au démarrage du microservice.

Valider la création. Faites de même pour le microservice `medecin-service`.

➤ Créer un 3^{ème} module pour le microservice `gateway-service` contenant les dépendances suivantes :

Gateway : pour définir ce service comme Gateway

Eureka Discovery Client : pour que le Gateway puisse interroger le Discovery Service.

Spring Boot Actuator : permet le monitoring des microservices

➤ Créer un 4^{ème} module pour le microservice `discovery-service` contenant les dépendances suivantes :

Eureka Server : est une implémentation de Netflix pour le serveur Discovery

Spring Boot Actuator : permet le monitoring des microservices

➤ Créer un 5^{ème} module pour le microservice `config-service` contenant les dépendances suivantes :

Config Server : est une implémentation pour le serveur de configuration

Spring Boot Actuator : permet le monitoring des microservices

Finalement le projet contiendra les 5 modules : 2 microservices fonctionnels et 3 microservices techniques gateway, discovery et config :



Développement des microservices fonctionnels :

A partir des implémentations faites dans le projet monolithique `MedicalApp`, récupérer les différents contenu des packages suivants: `entity`, `repository`, `service` et `controller` sans oublier de définir dans `application.properties` comme datasource la BD `bd-patient` pour le microservice `patient-service` et comme datasource la BD `bd-medecin` pour le microservice `medecin-service`.

Etant donné que nous n'avons pas encore développé le services techniques `Discovery-service` et `Config-service`, **IL FAUT LES DESACTIVER** à partir des propriétés. Ajouter dans le fichier `application.properties` de `patient-service` les propriétés suivantes :

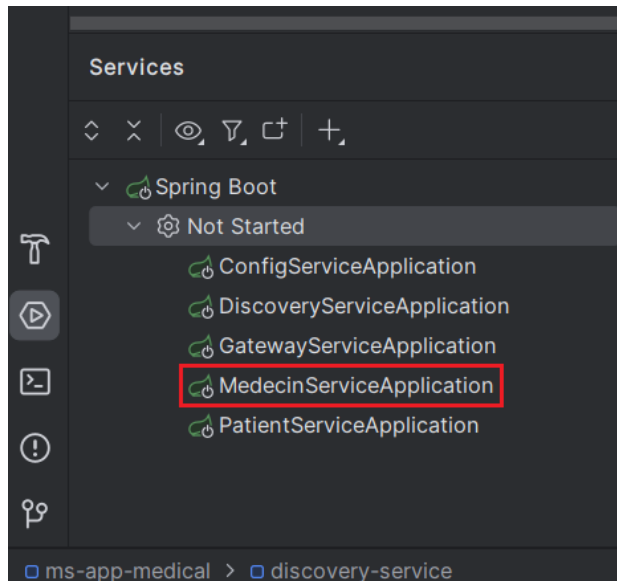
```
#le nom du microservice
spring.application.name=patient-service
#le num du port du microservice
server.port=8081
#ne pas s'enregistrer dans le Discovery-service au démarrage
spring.cloud.discovery.enabled=false
#ne pas récupérer la configuration de Config-service au démarrage
spring.cloud.config.enabled=false
```

Faites de même pour le microservice `medecin-service` en changeant le nom de l'application par `medecin-service` et le numéro du port par 8082.

Ajouter la dépendance Swagger (OpenAPI) dans chacun des microservices `patient-service` et `medecin-service` pour faciliter la consommation des Endpoints.

Pour l'instant les 2 microservices fonctionnels `patient-service` et `medecin-service` sont indépendants, on va démarrer le microservice `medecin-service` pour le tester.

L'onglet Services permet de démarrer et arrêter les différents microservices :



Ajouter quelques médecins et consulter les en utilisant Swagger.

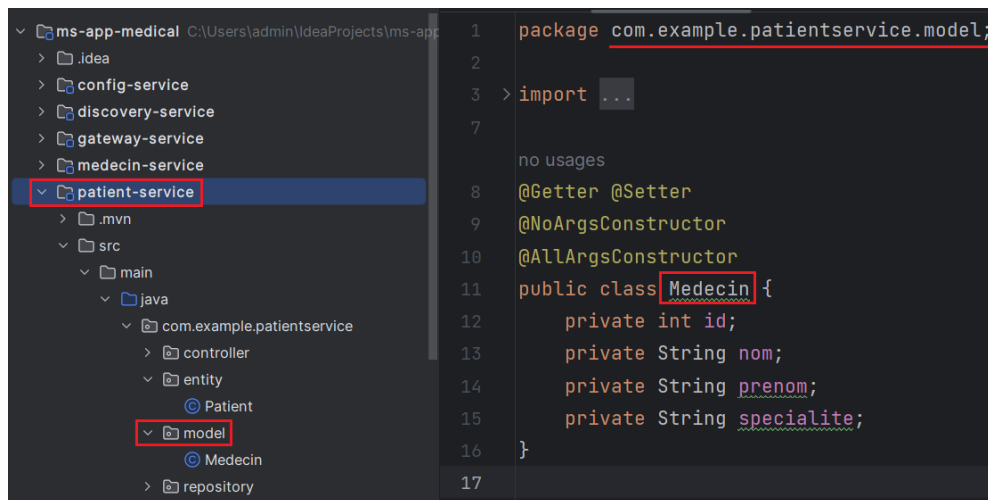
Relation entre les entités Patient et Medecin

On va maintenant relier les 2 entités (bien qu'elles ne soient pas implémenter dans une même BD). La relation permettra de relier chaque patient par son médecin traitant de manière unidirectionnelle. Pour cela on va ajouter dans l'entité Patient du microservice `patient-service` un attribut qui représente le id du médecin :

```
@Entity
public class Patient {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    private String nom;
    private String prenom;
    private String ville;
    private int tel;
    private int medecinId;
}
```

Ce `medecinId` permettra de récupérer les informations sur le médecin de ce patient en envoyant une requête vers le microservice `medecin-service`. Les informations

récupérées seront placées dans une instance d'une classe (non entité) nommée `Medecin` définies dans un package nommé `model` de `patient-service`. Cette classe contiendra les attributs à récupérer sur le médecin :



Puis on va ajouter un autre attribut dans l'entité `Patient` qui contiendra les informations sur le médecin mais cet attribut ne sera pas représenté dans la table `patient` (en l'annotant par `@Transient`). Cette annotation indique à Spring Data JPA qu'il faut ignorer la représentation de cet attribut dans la BD :

```
@Entity
public class Patient {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    private String nom;
    private String prenom;
    private String ville;
    private int tel;
    @Transient
    private Medecin medecin;
    private int medecinId;
}
```

Démarre le microservice `patient-service` et ajouter quelques patients et consulter les.

Spring Cloud Gateway

Sa tâche principale c'est le routage des requêtes HTTP entrantes des clients. Une route c'est la destination vers laquelle une requête particulière doit être acheminée. Cette route contient :

- L'URI de destination,
- Predicate : Une condition qui doit satisfaire l'URI
- Filters : Un ou plusieurs filtres qui peuvent intervenir pour apporter des traitements et des modifications des requêtes et des réponses HTTP.

Predicate peut être une condition sur :

- Host (terminaison de l'URI .com .tn)
- Path : le chemin dans l'URI
- Method : méthode de la requête.

Il y aussi d'autres Predicate sur le temps : Before, After, Between..

Filters : AddRequestHeader, AddRequestParamter, ...

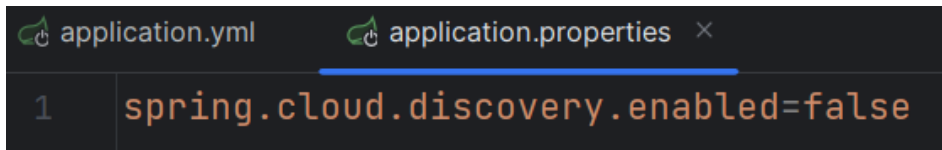
La configuration des routes dans le gateway peut se faire d'une manière statique ou bien d'une manière dynamique.

Pour configurer d'une manière statique (car, pour l'instant, on n'a pas encore configuré le Discovery service) il faut créer un fichier `application.yml` sous `resources` dans lequel on spécifie le routage statique :

```
application.yml x
1  spring:
2    cloud:
3      gateway:
4        routes:
5          - id: r1
6            uri: http://localhost:8081/
7            predicates:
8              - Path=/api/patient/**
9          - id: r2
10             uri: http://localhost:8082/
11             predicates:
12               - Path=/api/medecin/**
13         application:
14           name: gateway-service
15       server:
16         port: 8888
```

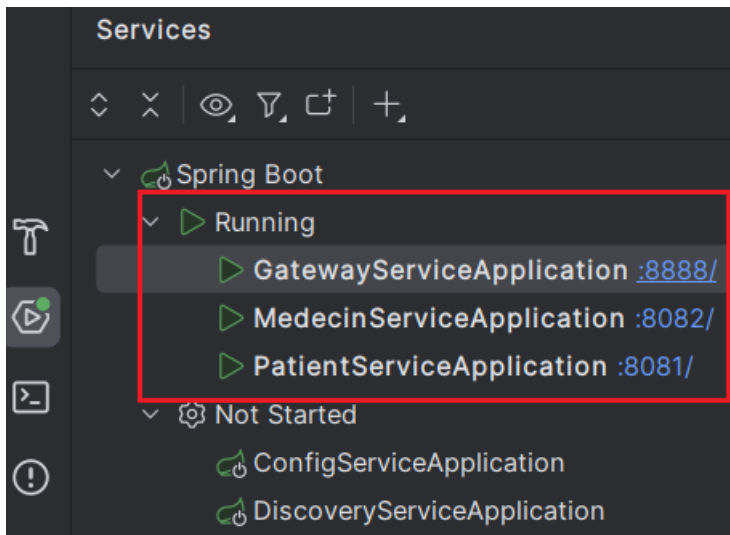
Dans ce fichier yml on définit 2 routes r1 et r2 qui portent chacune une condition (predicate) sur le path. Par exemple si le Gateway reçoit un chemin `/api/patient/**` il va router cette requête vers `http://localhost:8081/api/patient/**` (qui correspond à l'URL qui pointe vers `patient-service`)

Sans oublier de désactiver la découverte vers le Discovery service dans le fichier application.properties :



```
1 spring.cloud.discovery.enabled=false
```

Maintenant on démarre les 3 services suivants :



Lancer dans un navigateur l'URL suivant : <http://localhost:8888/api/patient/list> vous devez obtenir la liste des patient à travers le gateway :

