

MINI PROJET : GESTION DES RENDEZ-VOUS ET CONSULTATIONS MÉDICALES



RÉALISÉ PAR :
YOUSSEF AIT OUCHAOUR

ANNÉE UNIVERSITAIRE : 2023-2024

Introduction

Ce rapport présente un mini-projet de gestion des rendez-vous et consultations médicales, développé avec une architecture microservices utilisant Spring Boot. L'objectif est d'optimiser la gestion des rendez-vous médicaux en adoptant une approche modulaire et scalable. Nous explorons le processus de conception, d'implémentation et de déploiement, mettant en avant les avantages de l'architecture microservices dans ce contexte spécifique. Ce rapport offre un aperçu des choix technologiques, des défis rencontrés et des perspectives d'amélioration continue. L'approche choisie vise à moderniser et à rendre plus efficace la gestion des rendez-vous médicaux.

Objectifs

L'objectif principal de ce projet est de concevoir et de mettre en œuvre une application modulaire et évolutive pour la gestion des rendez-vous et consultations médicales.

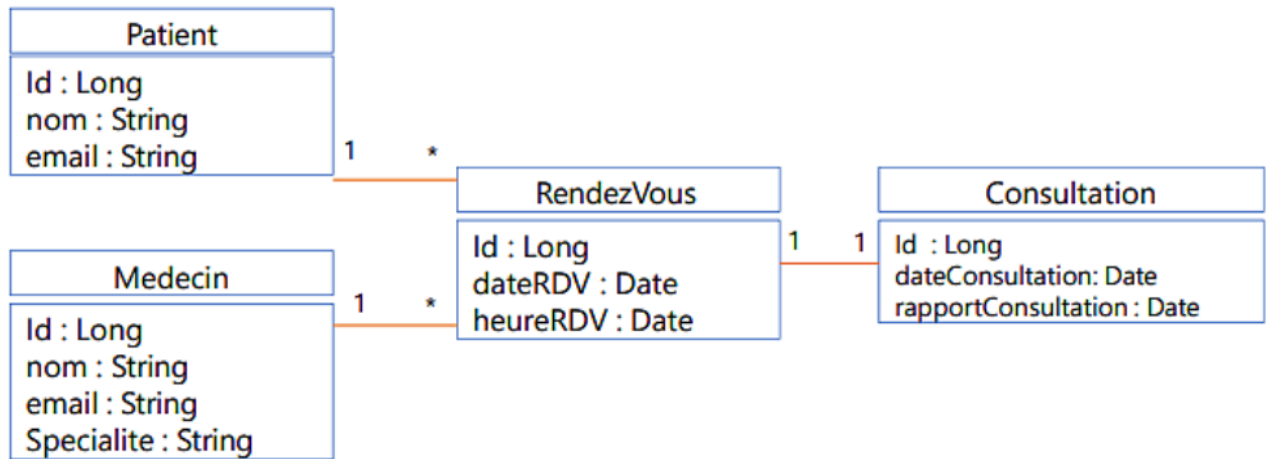
- Développer deux services distincts, le premier pour gérer les informations des patients et des médecins, et le deuxième pour gérer les rendez-vous et les consultations.
- Assurer l'intégration et la communication efficace entre les services à l'aide de l'API Gateway et des clients REST.
- Utiliser un registre de services pour la découverte et l'enregistrement des services.
- Mettre en place un serveur de configuration pour la gestion centralisée des configurations de services.
- Garantir la sécurité, la fiabilité et la disponibilité de l'application.

Table of Content

- [Introduction](#)
- [Objectifs](#)
- [Table of Content](#)
- [Conception & Architecture](#)
- [Contenu application](#)
 - [First-Service \(Medecin-Patient\)](#)
 - [Second-Service \(Rendezvous-Consultation\)](#)
 - [Gateway-Service](#)
 - [Discovery-Service](#)
 - [Config-Service](#)
- [Documentation des Tests avec Swagger UI](#)
- [Conclusion](#)

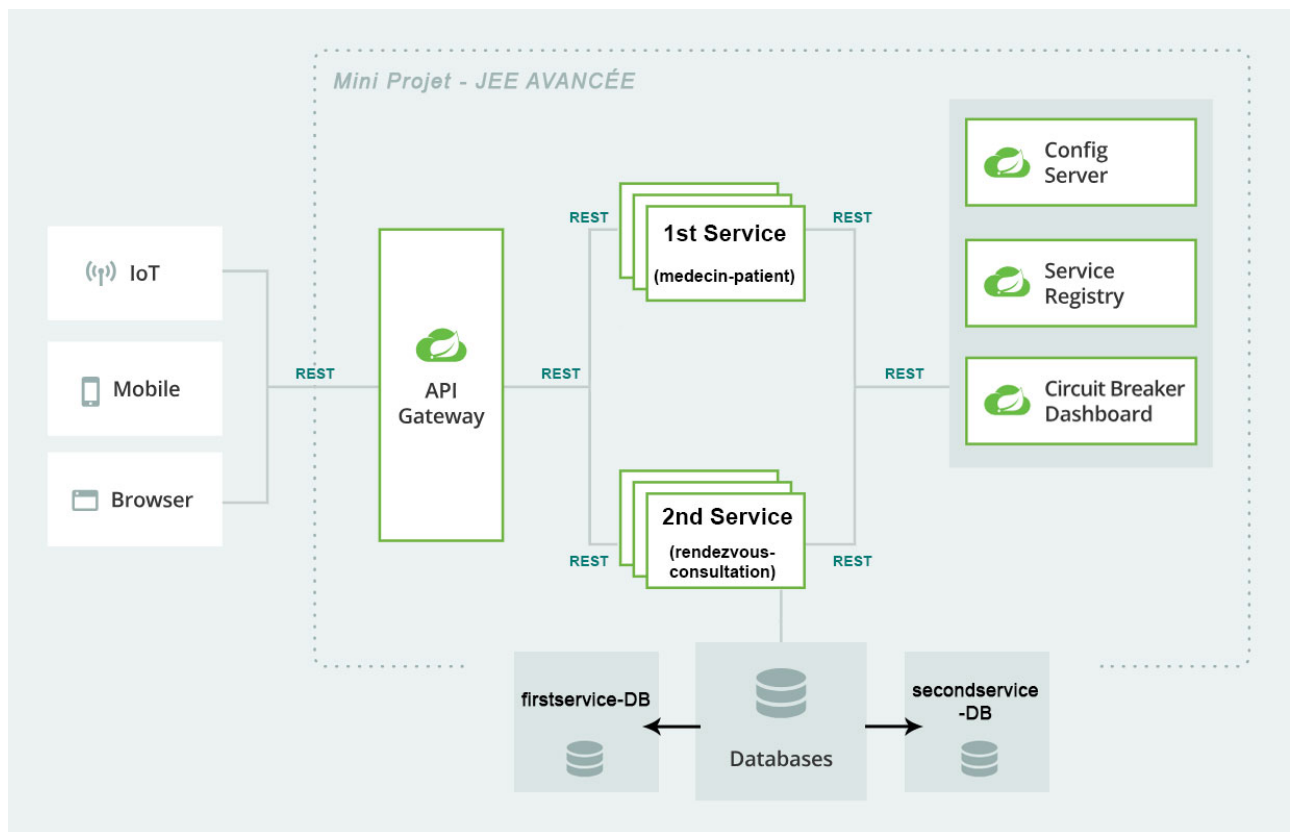
Conception & Architecture

- **Diagramme de classes :**



Le diagramme montre quatre entités principales : **Patient**, **Medecin**, **RendezVous**, et **Consultation**. Les Patients peuvent avoir plusieurs RendezVous, chaque RendezVous est pour un Patient et un Medecin spécifique, et à chaque RendezVous correspond une Consultation unique.

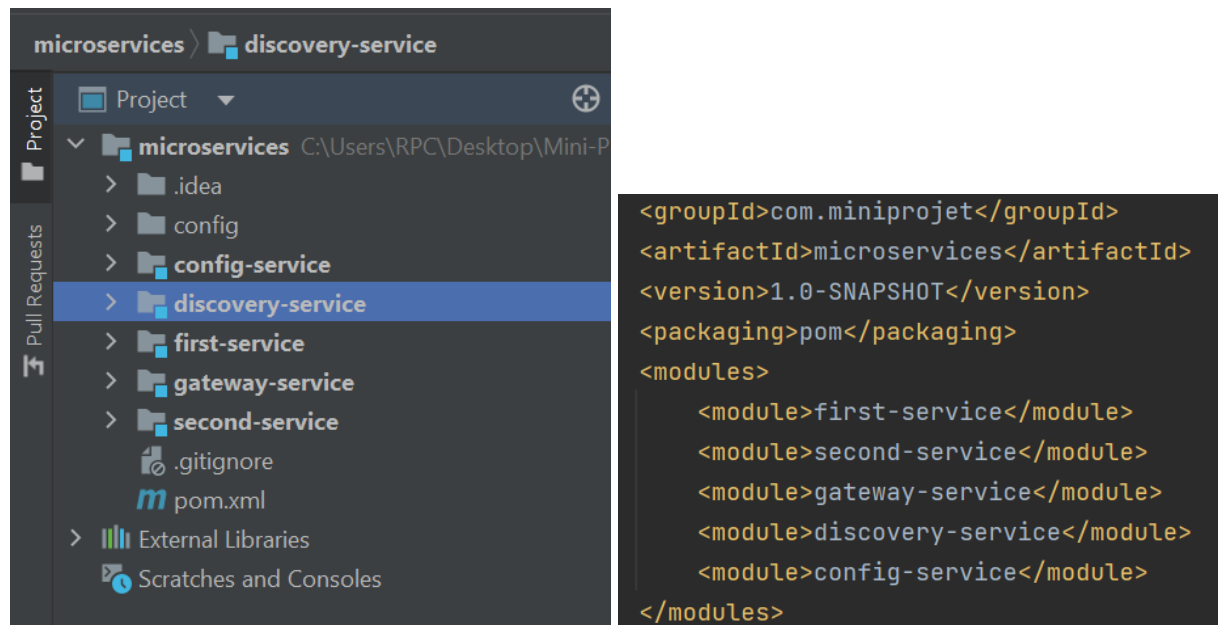
- **Architecture Microservices :**



L'application reçoit des requêtes HTTP d'une partie cliente qui interagit avec le système via une API Gateway. Le système est divisé en deux services principaux : le premier service gère les médecins et les

patients, et le deuxième service gère les rendez-vous et les consultations. Chaque service interagit avec sa propre base de données.

Il existe également des composants de support tels que la configuration centralisée (Config Server) et l'enregistrement des services (Service Registry ou Discovery Service). Cela indique une structure conçue pour la résilience, la scalabilité et la gestion efficace des configurations et des services dans un environnement de microservices.



voici les services de projet, sachant que:

first service : Service de Gestion des Patients et Médecins.

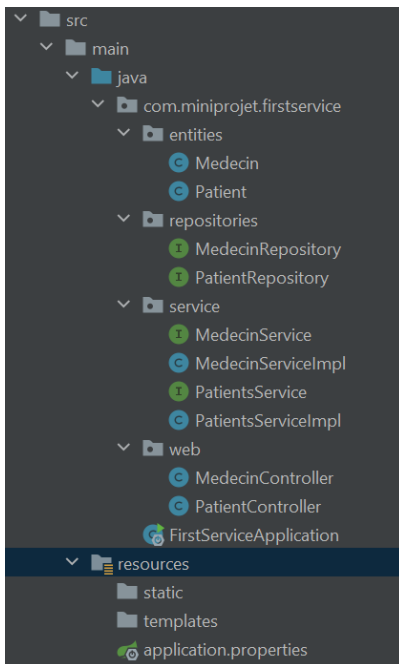
second service : Service de Gestion des Rendez-vous et Consultations.

- **Technologies Utilisées :**

- **Spring Boot**: Pour le développement des microservices.
- **Spring Data JPA**: Pour l'intégration avec la base de données MySQL.
- **Spring Cloud Config**: Pour la gestion centralisée de la configuration des services.
- **Spring Cloud Gateway**: Pour la gestion des routes.
- **Open Feign Rest Client**: Pour la communication entre les microservices.
- **Circuit Breaker Resilience4j**: Pour la gestion de la tolérance aux pannes.
- **Eureka Discovery Service**: Pour la découverte et l'enregistrement des services.

Contenu application

I.First Service :



```
localhost:8081/patients

1 [
2   {
3     "id": 4,
4     "nom": "Youssef",
5     "dateNaissance": "2024-01-11",
6     "malade": true
7   },
8   {
9     "id": 5,
10    "nom": "Mohammed",
11    "dateNaissance": "2024-01-11",
12    "malade": true
13  },
14  {
15    "id": 6,
16    "nom": "najat",
17    "dateNaissance": "2024-01-11",
18    "malade": true
19  },
20  {
21    "id": 7,
22    "nom": "hassan",
23    "dateNaissance": "2024-01-11",
24    "malade": true
25  }
26 ]
```

- Entities:

```
import ...
@Entity
@Builder
@Data @NoArgsConstructor @AllArgsConstructor
public class Medecin {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nom;
    private String email;
    private String specialite;
}

import ...
@Entity
@Builder
@Data @AllArgsConstructor @NoArgsConstructor
public class Patient {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nom;
    @Temporal(TemporalType.DATE)
    private Date dateNaissance;
    private boolean malade;
}
```

- Repository:

```
package com.miniprojet.firstservice.repositories;
//MedecinRepository
@RepositoryRestResource
public interface MedecinRepository extends JpaRepository<Medecin, Long> {
    Medecin findByNom(String nom);
}
```

```
package com.miniprojet.firstservice.repositories;
//PatientRepository
@RepositoryRestResource
public interface PatientRepository extends JpaRepository<Patient, Long> {
    Patient findByNom(String nom);
}
```

- Services:

```

PatientsServiceImpl.java
@Service
@AllArgsConstructor
public class PatientsServiceImpl implements PatientsService {
    private PatientRepository patientRepository;

    public Patient savePatient(Patient patient) {
        return patientRepository.save(patient);
    }

    public List<Patient> getAllPatients() {
        return patientRepository.findAll();
    }

    public Patient getPatient(Long auteurId) {
        return patientRepository.findById(auteurId).orElse(null);
    }

    public void DeletePatient(Long id) {
        patientRepository.deleteById(id);
    }

    public Patient updatePatient(Patient patient) {
        return patientRepository.save(patient);
    }
}

MedecinServiceImpl.java
@Service
@AllArgsConstructor
public class MedecinServiceImpl implements MedecinService {
    private MedecinRepository medecinRepository;

    public Medecin saveMedecin(Medecin medecin) {
        return medecinRepository.save(medecin);
    }

    public List<Medecin> getAllMedecins() {
        return medecinRepository.findAll();
    }

    public Medecin getMedecin(Long medecinId) {
        return medecinRepository.findById(medecinId).orElse(null);
    }

    public void DeleteMedecin(Long id) {
        medecinRepository.deleteById(id);
    }

    public Medecin updateMedecin(Medecin medecin) {
        return medecinRepository.save(medecin);
    }
}

```

```

public interface PatientsService {
    Patient savePatient(Patient patient);
    public List<Patient> getAllPatients();
    public Patient getPatient(Long auteurId);
    public void DeletePatient(Long id);
    public Patient updatePatient(Patient patient);
}

```

```

public interface MedecinService {
    Medecin saveMedecin(Medecin medecin);
    public List<Medecin> getAllMedecins();
    public Medecin getMedecin(Long auteurId);
    public void DeleteMedecin(Long id);
    public Medecin updateMedecin(Medecin medecin);
}

```

- Web:

```

//Medecin Controller
@RestController
public class MedecinController {
    @Autowired
    private MedecinService medecinService;

    @GetMapping(path = "medecins")
    public List<Medecin> getMedecins() {
        return medecinService.getAllMedecins();
    }

    @GetMapping("medecins/{medecinId}")
    public Medecin getMedecinById(@PathVariable Long medecinId) {
        return medecinService.getMedecin(medecinId);
    }
}

```

```

    }
    @DeleteMapping("medecins/{medecinId}")
    public void deleteMedecin(@PathVariable Long medecinId){
        medecinService.DeleteMedecin(medecinId);
    }
    @PostMapping("/medecins")
    public Medecin saveMedecin(@RequestBody Medecin medecin){
        return medecinService.saveMedecin(medecin);
    }
    @PutMapping("medecins/{medecinId}")
    public Medecin updateMedecin(@RequestBody Medecin medecin) {
        return medecinService.updateMedecin(medecin);
    }
}

```

```

//Patient Controller
@RestController
public class PatientController {
    @Autowired
    private PatientsService patientsService;
    @GetMapping(path = "patients")
    public List<Patient> getPatients(){
        return patientsService.getAllPatients();
    }
    @GetMapping("patients/{patientId}")
    public Patient getPatientById(@PathVariable Long patientId){
        return patientsService.getPatient(patientId);
    }
    @DeleteMapping("patients/{patientId}")
    public void deletePatient(@PathVariable Long patientId){
        patientsService.DeletePatient(patientId);
    }
    @PostMapping("/patients")
    public Patient savePatient(@RequestBody Patient patient){
        return patientsService.savePatient(patient);
    }
    @PutMapping("patients/{patientId}")
    public Patient updatePatient( @PathVariable Long patientId, @RequestBody
    Patient patient) {
        return patientsService.updatePatient(patient);
    }
}

```

- application.properties:

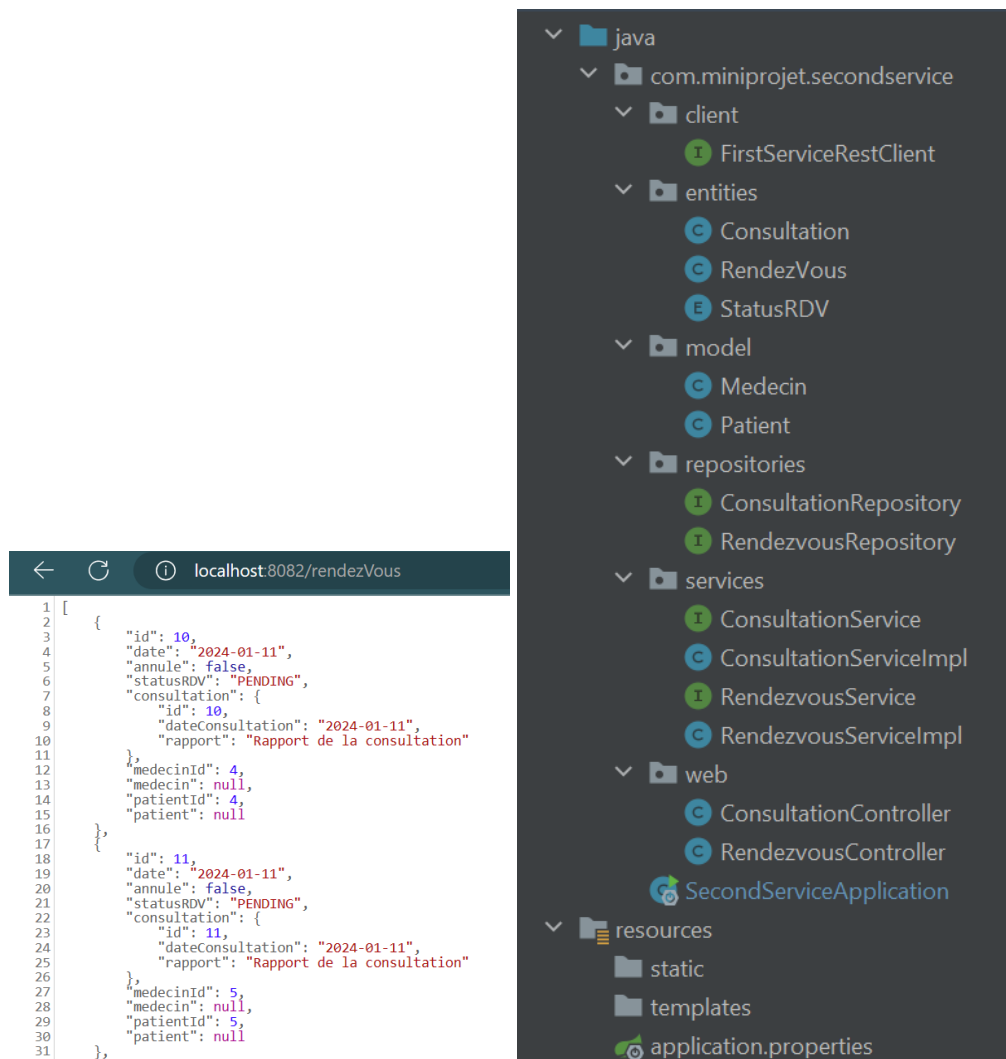
```

spring.application.name=first-service
server.port=8081
spring.config.import=optional:configserver:http://localhost:9999/}

```

Pour importer la configuration à partir du serveur de configuration situé à <http://localhost:9999/>. Le si le serveur de configuration n'est pas disponible ou ne peut pas être contacté, l'application ne s'arrêtera pas ou ne renverra pas d'erreur.

II.Second Service :



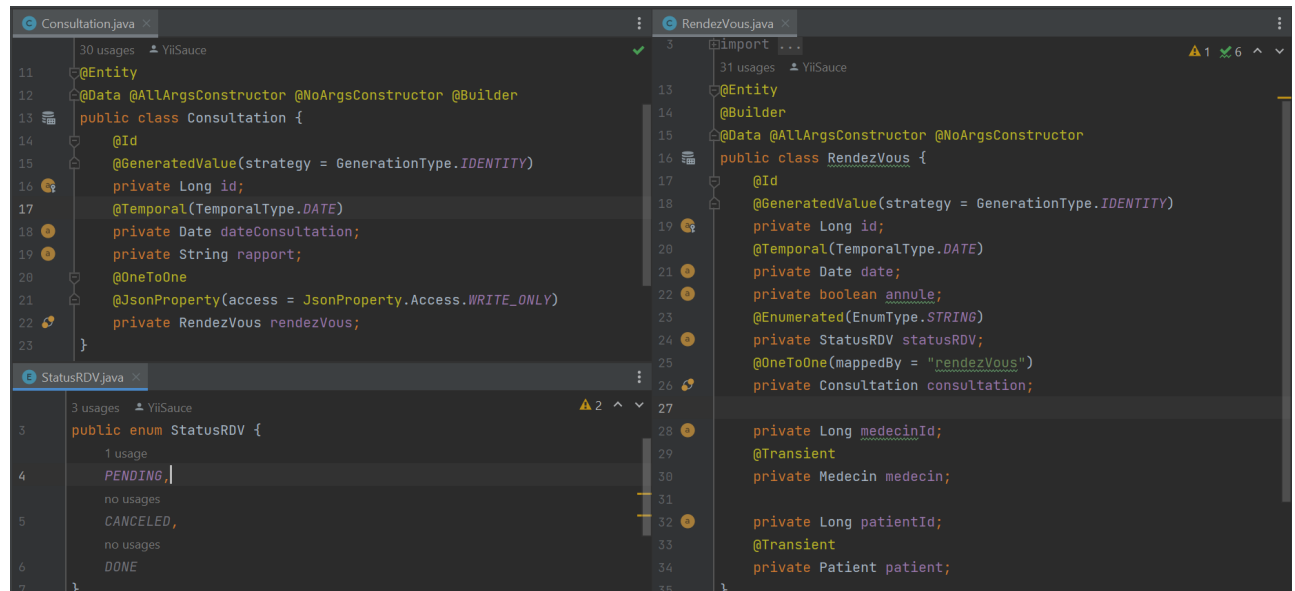
The image shows a project structure in an IDE for the package `com.miniprojet.secondservice`. The structure includes:

- `client`: `FirstServiceRestClient`
- `entities`: `Consultation`, `RendezVous`, `StatusRDV`
- `model`: `Medecin`, `Patient`
- `repositories`: `ConsultationRepository`, `RendezvousRepository`
- `services`: `ConsultationService`, `ConsultationServiceImpl`, `RendezvousService`, `RendezvousServiceImpl`
- `web`: `ConsultationController`, `RendezvousController`, `SecondServiceApplication`
- `resources`: `static`, `templates`, `application.properties`

Below the structure, a REST client response is shown for `localhost:8082/rendezVous`. The response is a JSON array of two objects:

```
1 [
2   {
3     "id": 10,
4     "date": "2024-01-11",
5     "annule": false,
6     "statusRDV": "PENDING",
7     "consultation": {
8       "id": 10,
9       "dateConsultation": "2024-01-11",
10      "rapport": "Rapport de la consultation"
11    },
12    "medecinId": 4,
13    "medecin": null,
14    "patientId": 4,
15    "patient": null
16  },
17  {
18    "id": 11,
19    "date": "2024-01-11",
20    "annule": false,
21    "statusRDV": "PENDING",
22    "consultation": {
23      "id": 11,
24      "dateConsultation": "2024-01-11",
25      "rapport": "Rapport de la consultation"
26    },
27    "medecinId": 5,
28    "medecin": null,
29    "patientId": 5,
30    "patient": null
31  }
32 ]
```

- Entities:



The image shows the code for three entities in an IDE:

- `Consultation.java`:

```
11 @Entity
12 @Data @AllArgsConstructor @NoArgsConstructor @Builder
13 public class Consultation {
14     @Id
15     @GeneratedValue(strategy = GenerationType.IDENTITY)
16     private Long id;
17     @Temporal(TemporalType.DATE)
18     private Date dateConsultation;
19     private String rapport;
20     @OneToOne
21     @JsonProperty(access = JsonProperty.Access.WRITE_ONLY)
22     private RendezVous rendezVous;
23 }
```
- `StatusRDV.java`:

```
3 public enum StatusRDV {
4     PENDING,
5     CANCELED,
6     DONE
7 }
```
- `RendezVous.java`:

```
13 @Entity
14 @Builder
15 @Data @AllArgsConstructor @NoArgsConstructor
16 public class RendezVous {
17     @Id
18     @GeneratedValue(strategy = GenerationType.IDENTITY)
19     private Long id;
20     @Temporal(TemporalType.DATE)
21     private Date date;
22     private boolean annule;
23     @Enumerated(EnumType.STRING)
24     private StatusRDV statusRDV;
25     @OneToOne(mappedBy = "rendezVous")
26     private Consultation consultation;
27
28     private Long medecinId;
29     @Transient
30     private Medecin medecin;
31
32     private Long patientId;
33     @Transient
34     private Patient patient;
35 }
```

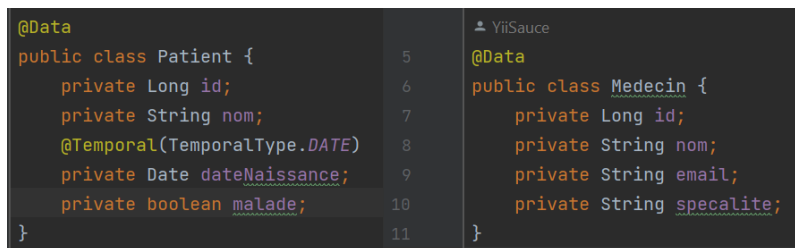

- Repository:

```
//Consultation Repository
@RepositoryRestResource
public interface ConsultationRepository extends JpaRepository<Consultation, Long> {
}
```

```
//Rendez-vous Repository
@RepositoryRestResource
public interface RendezvousRepository extends JpaRepository<RendezVous, Long>
{
}
```

- Model:

le package model contient toutes les classes dont nous aurons besoin dans le 2ème service à partir du 1er service (**ce ne sont pas des entités JPA**).



```
@Data
public class Patient {
    private Long id;
    private String nom;
    @Temporal(TemporalType.DATE)
    private Date dateNaissance;
    private boolean malade;
}

@Data
public class Medecin {
    private Long id;
    private String nom;
    private String email;
    private String specialite;
}
```

- Client:

```
@FeignClient(name = "FIRST-SERVICE")
public interface FirstServiceRestClient {
    @GetMapping("/patients/{id}")
    public Patient getPatientById(@PathVariable Long id);
    @GetMapping("/patients")
    public List<Patient> getPatients();
    @GetMapping("/medecins/{id}")
    public Medecin getMedecinById(@PathVariable Long id);
    @GetMapping("/medecins")
    public List<Medecin> getMedecins();
}
```

FirstServiceRestClient : est un client **Feign** utilisé pour interagir avec un service distant. Elle appartient au deuxième service (SECOND-SERVICE) et communique avec le premier service (FIRST-SERVICE). Les méthodes définies dans cette interface correspondent aux points de terminaison du premier service pour récupérer des informations sur les patients et les médecins.

`@FeignClient(name = "FIRST-SERVICE")` : Spécifie que cette interface est un client Feign pour le service nommé "FIRST-SERVICE".

- Services:

on retrouve ici toutes les fonctionnalités **CRUD** pour créer, supprimer mettre à jour et manipuler toutes les entités de notre service.

```
@Service
public class RendezvousServiceImpl implements RendezvousService{
    @Autowired
    private RendezvousRepository rendezvousRepository;
    @Autowired
    private ConsultationRepository consultationRepository;
    @Autowired
    private FirstServiceRestClient firstServiceRestClient;
    @Override
    public Rendezvous saveRendezvous(Rendezvous rendezvous){
        Medecin medecin =
firstServiceRestClient.getMedecinById(rendezvous.getMedecinId());
        Patient patient =
firstServiceRestClient.getPatientById(rendezvous.getPatientId());
        if(medecin!=null && patient!=null){
            rendezvous.setMedecin(medecin);
            rendezvous.setPatient(patient);
            return rendezvousRepository.save(rendezvous);
        }
        return null;
    }
    @Override
    public List<Rendezvous> getAllRendezvous(){return
rendezvousRepository.findAll();}
    @Override
    public Rendezvous getRendezvous(Long rendezvousId) {
        Rendezvous rendezvous =
rendezvousRepository.findById(rendezvousId).orElse(null);
        Medecin medecin =
firstServiceRestClient.getMedecinById(rendezvous.getMedecinId());
        Patient patient =
firstServiceRestClient.getPatientById(rendezvous.getPatientId());
        rendezvous.setMedecin(medecin);
        rendezvous.setPatient(patient);
        return rendezvous;
    };
    @Override
    public void deleteRendezvous(Long id){
        Rendezvous rendezvous =
rendezvousRepository.findById(id).orElse(null);
        if (rendezvous != null) {
            Consultation consultation = rendezvous.getConsultation();
            if (consultation != null) {
                consultationRepository.deleteById(consultation.getId());
            }
        }
    }
}
```

```

        }
        rendezvousRepository.deleteById(id);
    }
};
@Override
public RendezVous updateRendezVous(RendezVous rendezVous){
    RendezVous rv = getRendezVous(rendezVous.getId());
    rv.setStatusRDV(rendezVous.getStatusRDV());
    rv.setDate(rendezVous.getDate());
    rv.setAnnule(rendezVous.isAnnule());
    rv.setConsultation(rendezVous.getConsultation());
    rv.setPatientId(rendezVous.getPatientId());
    rv.setMedecinId(rendezVous.getMedecinId());
    Medecin medecin =
firstServiceRestClient.getMedecinById(rendezVous.getMedecinId());
    Patient patient =
firstServiceRestClient.getPatientById(rendezVous.getPatientId());
    rv.setMedecin(medecin);
    rv.setPatient(patient);
    return rendezvousRepository.save(rv);
};
}

```

pour casser la relation entre FIRST-SERVICE et SECOND-SERVICE (OneToMany), nous obtenons les models dont nous avons besoin en utilisant le firstservicerestclient et les intégrons dans le notre service

```

@Service
public class ConsultationServiceImpl implements ConsultationService{
    @Autowired
    private ConsultationRepository consultationRepository;
    public Consultation saveConsultation(Consultation consultation){return
consultationRepository.save(consultation);};
    public List<Consultation> getAllConsultations(){
        return consultationRepository.findAll();
    };
    public Consultation getConsultation(Long consultationId){
        return
consultationRepository.findById(consultationId).orElse(null);
    };
    public void DeleteConsultation(Long id){
        consultationRepository.deleteById(id);
    };
    public Consultation updateConsultation(Consultation consultation){
        Consultation consultation1 = getConsultation(consultation.getId());

consultation1.setDateConsultation(consultation.getDateConsultation());
        consultation1.setRapport(consultation.getRapport());
        return consultationRepository.save(consultation1);
    };
}

```

- Web:

```
@RestController
public class RendezvousController {
    @Autowired
    private RendezvousService rendezvousService;
    //READ
    @GetMapping(path = "rendezVous")
    public List<RendezVous> getRendezVous(){
        return rendezvousService.getAllRendezVous();
    }
    //READ BY ID
    @GetMapping("rendezVous/{rendezVousId}")
    public RendezVous getRendezvousById(@PathVariable Long rendezVousId)
    {return rendezvousService.getRendezVous(rendezVousId);}
    //DELETE
    @DeleteMapping("rendezVous/{rendezVousId}")
    public void deleteRendezVous(@PathVariable Long rendezVousId)
    {rendezvousService.deleteRendezVous(rendezVousId);}
    //CREATE
    @PostMapping("rendezVous")
    public RendezVous saveRendezVous(@RequestBody RendezVous rendezVous)
    {return rendezvousService.saveRendezVous(rendezVous);}
    //UPDATE
    @PutMapping("rendezVous/{rendezVousId}")
    public RendezVous updateRendezvous(@RequestBody RendezVous rendezVous)
    {return rendezvousService.updateRendezVous(rendezVous);}
}
```

```
@RestController
public class ConsultationController {
    @Autowired
    private ConsultationService consultationService;
    @GetMapping(path = "consultations")
    public List<Consultation> getConsultations(){return
consultationService.getAllConsultations();}
    @GetMapping("consultations/{consultationId}")
    public Consultation getConsultationById(@PathVariable Long
consultationId){
        return consultationService.getConsultation(consultationId);
    }
    @DeleteMapping("consultations/{consultationId}")
    public void deleteConsultation(@PathVariable Long consultationId){
        consultationService.DeleteConsultation(consultationId);
    }
    @PostMapping("consultations")
    public Consultation saveConsultation(@RequestBody Consultation
consultation){
        return consultationService.saveConsultation(consultation);
    }
}
```

```

    @PostMapping("consultations/{consultationId}")
    public Consultation updateConsultation(@RequestBody Consultation
consultation) {
        return consultationService.updateConsultation(consultation);
    }
}

```

- application.properties:

```

spring.application.name=second-service
server.port=8082
spring.config.import=optional:configserver:http://localhost:9999/
feign.client.config.default.loggerLevel=full

```

Note: nous devons ajouter ces annotations à la classe SecondServiceApplication:

@EnableDiscoveryClient @EnableFeignClients

III. Gateway Service :

Le service Gateway agit comme un intermédiaire qui simplifie la complexité des interactions entre le client et les microservices.

- GatewayServiceApplication:

```

@SpringBootApplication
public class GatewayServiceApplication {
    public static void main(String[] args) {
        SpringApplication.run(GatewayServiceApplication.class, args);
    }
    @Bean
    DiscoveryClientRouteDefinitionLocator
dynamicRoutes(ReactiveDiscoveryClient rdc,DiscoveryLocatorProperties dlp){
        return new DiscoveryClientRouteDefinitionLocator(rdc,dlp);
    }
}

```

- application.properties:

```

spring.application.name=gateway-service
server.port=8888
spring.config.import=optional:configserver:http://localhost:9999/

```

- application.yml:

```


```

```

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

```

Ce fichier .yml est une configuration pour le Gateway Service Il définit des règles de routage pour le service passerelle (API Gateway).

IV.Discovery Service :

Les microservices interrogent le Discovery Service pour trouver les instances disponibles d'autres services auxquels ils souhaitent faire appel. Cela facilite la communication dynamique et l'équilibrage de charge entre les services, et pour une meilleure tolérance aux pannes. Les outils populaires pour le service de découverte incluent Eureka de Netflix que nous utilisons dans notre application.

- DiscoveryServiceApplication:

```

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

```

- application.properties:

```

spring.application.name=discovery-service
server.port=8761
#dont register server itself as client
eureka.client.fetch-registry=false
# does not register itself in the service registry
eureka.client.register-with-eureka=false

```

voici le server d'Eureka:

The screenshot shows the Spring Eureka dashboard. At the top, there's a header with the Spring Eureka logo and navigation links for HOME and LAST 1000 SINCE STARTUP. Below the header, the 'System Status' section displays two tables. The first table shows Environment: test and Data center: default. The second table shows Current time: 2024-01-11T13:33:11 +0100, Uptime: 00:12, Lease expiration enabled: true, Renew threshold: 6, and Renew (last min): 10. Below this, the 'DS Replicas' section has a search bar containing 'localhost'. The 'Instances currently registered with Eureka' section contains a table with the following data:

Application	AMIs	Availability Zones	Status
FIRST-SERVICE	n/a (1)	(1)	UP (1) - localhost:first-service:8081
GATEWAY-SERVICE	n/a (1)	(1)	UP (1) - 192.168.56.1:gateway-service:8888
SECOND-SERVICE	n/a (1)	(1)	UP (1) - localhost:second-service:8082

@EnableEurekaServer : pour activer la fonctionnalité du serveur Eureka dans une application Spring Boot. Lorsqu'elle est appliquée à la classe principale d'un projet Spring Boot, elle indique que l'application agira en tant que serveur Eureka.

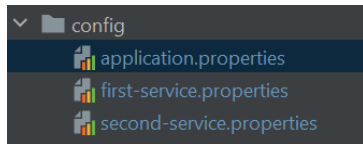
V.Config Service :

- ConfigServiceApplication:

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

- application.properties:

```
spring.application.name=config-service
server.port=9999
#spring.cloud.config.server.git.uri=file://${user.home}/config
spring.cloud.discovery.enabled=true
#spring.cloud.config.server.git.default-label=master
spring.cloud.config.server.git.uri=file:///C:/Users/RPC/Desktop/Mini-Projet/microservices/config
```



- config(dossier):

pour fournir un support de configuration centralisé et externe aux applications, nous mettons le fichier de configuration de chaque service(configurations de base de données...) dans le répertoire de **config**.

service	properties
first-service	<pre>spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver spring.datasource.url=jdbc:mysql://localhost:3306/firstservice-db?createDatabaseIfNotExist=true spring.datasource.username=root spring.datasource.password= spring.jpa.hibernate.ddl-auto=update spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MariaDBDialect</pre>
second-service	<pre>spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver spring.datasource.url=jdbc:mysql://localhost:3306/secondservice-db?createDatabaseIfNotExist=true spring.datasource.username=root spring.datasource.password= spring.jpa.hibernate.ddl-auto=update spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MariaDBDialect</pre>
application.properties	<pre>spring.cloud.config.enabled=true spring.cloud.discovery.enabled=true eureka.instance.prefer-ip-address=true eureka.client.service-url.defaultZone=\${DISCOVERY_SERVICE_URL:http://localhost:8761/eureka} management.endpoints.web.exposure.include=*</pre>

Documentation des Tests avec Swagger UI

avant d'exécuter tous les services pour tester les API et leurs méthodes, nous ajoutons le package d'openAPI au pom.xml du chaque service (first-service & second-service) afin d'obtenir une interface UI:

```
<dependency>
  <groupId>org.springdoc</groupId>
  <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
  <version>2.3.0</version>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-validation</artifactId>
  <version>3.2.1</version>
</dependency>
```

Note: nous avons ajouté quelques données dans les deux services pour tester et visualiser.

id	annule	date	medecin_id	patient_id	statusrdv
10	0	2024-01-11	4	4	PENDING
11	0	2024-01-11	5	5	PENDING
12	1	2024-01-11	6	6	PENDING
13	1	2024-01-11	7	7	PENDING
14	0	2024-01-11	8	8	PENDING

1. Service de Gestion des Patients et Médecins:

Nous pouvons accéder à l'API en utilisant cette URL (depuis le Gateway) :

```
< > ↻ 🌐 localhost:8888/FIRST-SERVICE/swagger-ui/index.html
```

- **Patient :**

patient-entity-controller	
GET	/patients
POST	/patients
GET	/patients/{id}
PUT	/patients/{id}
DELETE	/patients/{id}

Afficher tous les patients

Curl

```
curl -X 'GET' \
'http://localhost:8081/patients?page=0&size=20' \
-H 'accept: application/hal+json'
```

Request URL

```
http://localhost:8081/patients?page=0&size=20
```

Server response

Code	Details
200	<div><div>Response body</div><pre>[{ "id": 4, "nom": "Youssef", "dateNaissance": "2024-01-11", "malade": true }, { "id": 5, "nom": "Mohammed", "dateNaissance": "2024-01-11", "malade": true }, { "id": 6, "nom": "najat", "dateNaissance": "2024-01-11", "malade": true }, { "id": 7, "nom": "hassan", "dateNaissance": "2024-01-11", "malade": true }, { "id": 8, "nom": "saida", </pre></div> <div><div>Response headers</div><pre>connection: keep-alive content-type: application/hal+json date: Thu,11 Jan 2024 14:34:48 GMT keep-alive: timeout=60 transfer-encoding: chunked</pre></div>

Ajouter un patient

Curl

```
curl -X 'POST' \
  'http://localhost:8081/patients' \
  -H 'accept: application/hal+json' \
  -H 'Content-Type: application/json' \
  -d '{
    "id": 9,
    "nom": "Patient Test",
    "dateNaissance": "2024-01-11T14:36:19.288Z",
    "malade": true
  }'
```

Request URL

http://localhost:8081/patients

Server response

Code	Details
200	<p>Response body</p> <pre>{ "id": 9, "nom": "Patient Test", "dateNaissance": "2024-01-11", "malade": true }</pre> <p>Response headers</p> <pre>connection: keep-alive content-type: application/hal+json date: Thu,11 Jan 2024 14:36:43 GMT keep-alive: timeout=60 transfer-encoding: chunked</pre>

Afficher un patient par son identifiant

Curl

```
curl -X 'GET' \
  'http://localhost:8081/patients/9' \
  -H 'accept: application/hal+json'
```

Request URL

http://localhost:8081/patients/9

Server response

Code	Details
200	<p>Response body</p> <pre>{ "id": 9, "nom": "Patient Test", "dateNaissance": "2024-01-11", "malade": true }</pre> <p>Response headers</p> <pre>connection: keep-alive content-type: application/hal+json date: Thu,11 Jan 2024 14:37:17 GMT keep-alive: timeout=60 transfer-encoding: chunked</pre>

Mettre à jour un patient

Curl

```
curl -X 'PUT' \
  'http://localhost:8081/patients/9' \
  -H 'accept: application/hal+json' \
  -H 'Content-Type: application/json' \
  -d '{
    "id": 9,
    "nom": "Another Patient",
    "dateNaissance": "2024-01-11T14:38:03.791Z",
    "malade": true
  }'
```

Request URL

http://localhost:8081/patients/9

Server response

Code	Details
------	---------

200

Response body

```
{
  "id": 9,
  "nom": "Another Patient",
  "dateNaissance": "2024-01-11T14:38:03.791+00:00",
  "malade": true
}
```

Response headers

```
connection: keep-alive
content-type: application/hal+json
date: Thu, 11 Jan 2024 14:38:26 GMT
keep-alive: timeout=60
transfer-encoding: chunked
```

Supprimer un patient

Curl

```
curl -X 'DELETE' \
  'http://localhost:8081/patients/9' \
  -H 'accept: */*'
```

Request URL

http://localhost:8081/patients/9

Server response

Code	Details
------	---------

200

Undocumented

Response headers

```
connection: keep-alive
content-length: 0
date: Thu, 11 Jan 2024 14:39:00 GMT
keep-alive: timeout=60
```

- **Medecin :**

medecin-entity-controller		^
GET	/medecins	▼
POST	/medecins	▼
GET	/medecins/{id}	▼
PUT	/medecins/{id}	📄 ▼
DELETE	/medecins/{id}	▼
PATCH	/medecins/{id}	▼
medecin-search-controller		^
GET	/medecins/search/findByNom	▼

Afficher tous les medecins

Curl

curl -X 'GET' \
'http://localhost:8081/medecins?page=0&size=20' \
-H 'accept: application/hal+json'

Request URL

http://localhost:8081/medecins?page=0&size=20

Server response

Code	Details
200	<div>Response body</div> <div>[{ "id": 4, "nom": "Bennani", "email": "Bennani@gamil.com", "specalite": "Dentiste" }, { "id": 5, "nom": "Adnani", "email": "Adnani@gamil.com", "specalite": "Cardio" }, { "id": 6, "nom": "Youssef", "email": "Youssef@gamil.com", "specalite": "Dentiste" }, { "id": 7, "nom": "Zineb", "email": "Zineb@gamil.com", "specalite": "Cardio" }],</div>

Ajouter un medecin

Responses

Curl

curl -X 'POST' \
'http://localhost:8081/medecins' \
-H 'accept: application/hal+json' \
-H 'content-type: application/json' \
-d '{
 "id": 10,
 "nom": "Test Medecin",
 "email": "string",
 "specalite": "Ophtalmologie"
}'

Request URL

http://localhost:8081/medecins

Server response

Code	Details
200	<div>Response body</div> <div>{ "id": 9, "nom": "Test Medecin", "email": "string", "specalite": "Ophtalmologie" }</div> <div>Response headers</div> <div>connection: keep-alive content-type: application/hal+json date: Thu, 11 Jan 2024 14:27:50 GMT keep-alive: timeout=60 transfer-encoding: chunked</div>

Afficher un medecin par son identifiant

Curl

```
curl -X 'GET' \
'http://localhost:8081/medecins/9' \
-H 'accept: application/hal+json'
```

Request URL

`http://localhost:8081/medecins/9`

Server response

Code	Details
200	<p>Response body</p> <pre>{ "id": 9, "nom": "Test Medecin", "email": "string", "specalite": "Ophtalmologie" }</pre> <p>Response headers</p> <pre>connection: keep-alive content-type: application/hal+json date: Thu,11 Jan 2024 14:30:09 GMT keep-alive: timeout=60 transfer-encoding: chunked</pre>

Mettre à jour un medecin

Curl

```
curl -X 'PUT' \
'http://localhost:8081/medecins/9' \
-H 'accept: application/hal+json' \
-H 'Content-Type: application/json' \
-d '{
  "id": 9,
  "nom": "Another Test"
}'
```

Request URL

`http://localhost:8081/medecins/9`

Server response

Code	Details
200	<p>Response body</p> <pre>{ "id": 9, "nom": "Another Test", "email": null, "specalite": null }</pre> <p>Response headers</p> <pre>connection: keep-alive content-type: application/hal+json date: Thu,11 Jan 2024 14:32:30 GMT keep-alive: timeout=60 transfer-encoding: chunked</pre>

Supprimer un medecin

Curl

```
curl -X 'DELETE' \
'http://localhost:8081/medecins/9' \
-H 'accept: */*'
```

Request URL

```
http://localhost:8081/medecins/9
```

Server response

Code	Details
200	Response headers
Undocumented	<pre>connection: keep-alive content-length: 0 date: Thu,11 Jan 2024 14:33:23 GMT keep-alive: timeout=60</pre>

Afficher un Medecin par son nom

Curl

```
curl -X 'DELETE' \
'http://localhost:8081/medecins/9' \
-H 'accept: */*'
```

Request URL

```
http://localhost:8081/medecins/9
```

Server response

Code	Details
200	Response headers
Undocumented	<pre>connection: keep-alive content-length: 0 date: Thu,11 Jan 2024 14:33:23 GMT keep-alive: timeout=60</pre>

2. Service de Gestion des Rendez-vous et Consultations:

Nous pouvons accéder à l'API en utilisant cette URL (depuis le Gateway) :

< > ↻ 🌐 localhost:8888/SECOND-SERVICE/swagger-ui/index.html

- **RendezVous :**

rendez-vous-entity-controller

GET	/rendezVouses	✓ 📄
POST	/rendezVouses	✓
GET	/rendezVouses/{id}	✓
PUT	/rendezVouses/{id}	✓
DELETE	/rendezVouses/{id}	✓

Afficher tous les rendezVous

Curl

curl -X 'GET' \
'http://localhost:8082/rendezVous' \
-H 'accept: application/hal+json'

Request URL

http://localhost:8082/rendezVous

Server response

Code

Details

200

Response body

[
{
 "id": 10,
 "date": "2024-01-11",
 "annule": true,
 "statusRDV": "PENDING",
 "consultation": {
 "id": 10,
 "dateConsultation": "2024-01-11",
 "rapport": "Rapport de la consultation"
 },
 "medecinId": 4,
 "medecin": null,
 "patientId": 4,
 "patient": null
},
{
 "id": 11,
 "date": "2024-01-11",
 "annule": false,
 "statusRDV": "PENDING",
 "consultation": {
 "id": 11,
 "dateConsultation": "2024-01-11",
 "rapport": "Rapport de la consultation"
 },
 "medecinId": 5,
 "medecin": null,

Response headers

connection: keep-alive
content-type: application/hal+json
date: Thu,11 Jan 2024 15:40:04 GMT
keep-alive: timeout=60
transfer-encoding: chunked

Ajouter un rendezVous

Curl

```
curl -X 'POST' \
  'http://localhost:8082/rendezVous' \
  -H 'accept: application/hal+json' \
  -H 'Content-Type: application/json' \
  -d '{
    "id": 19,
    "date": "2024-01-11T14:58:18.939Z",
    "annule": true,
    "statusRDV": "PENDING",
    "medecinId": 6,
    "patientId": 5
  }'
```

Request URL

http://localhost:8082/rendezVous

Server response

Code

Details

200

Response body

```
{
  "id": 19,
  "date": "2024-01-11",
  "annule": true,
  "statusRDV": "PENDING",
  "consultation": null,
  "medecinId": 6,
  "medecin": null,
  "patientId": 5,
  "patient": null
}
```

Response headers

```
connection: keep-alive
content-type: application/hal+json
date: Thu, 11 Jan 2024 15:38:29 GMT
keep-alive: timeout=60
transfer-encoding: chunked
```

Afficher un rendezVous par son identifiant

Curl

```
curl -X 'GET' \
  'http://localhost:8082/rendezVous/19' \
  -H 'accept: application/hal+json'
```

Request URL

http://localhost:8082/rendezVous/19

Server response

Code

Details

200

Response body

```
{
  "id": 19,
  "date": "2024-01-11",
  "annule": true,
  "statusRDV": "PENDING",
  "consultation": null,
  "medecinId": 6,
  "medecin": {
    "id": 6,
    "nom": "Youssef",
    "email": "Youssef@gamil.com",
    "specialite": "Dentiste"
  },
  "patientId": 5,
  "patient": {
    "id": 5,
    "nom": "Mohammed",
    "dateNaissance": "2024-01-11T00:00:00.000+00:00",
    "malade": true
  }
}
```

Response headers

```
connection: keep-alive
content-type: application/hal+json
date: Thu, 11 Jan 2024 15:40:50 GMT
keep-alive: timeout=60
transfer-encoding: chunked
```

Mettre à jour un rendezVous

Curl

```
curl -X 'PUT' \
  'http://localhost:8082/rendezVous/19' \
  -H 'accept: application/hal+json' \
  -H 'Content-Type: application/json' \
  -d '{
    "id": 19,
    "date": "2024-01-11T15:41:38.199Z",
    "annule": true,
    "statusRDV": "PENDING",
    "medecinId": 8,
    "patientId": 8
  }'
```

Request URL

http://localhost:8082/rendezVous/19

Server response

Code

Details

200

Response body

```
{
  "id": 19,
  "date": "2024-01-11T15:41:38.199+00:00",
  "annule": true,
  "statusRDV": "PENDING",
  "consultation": null,
  "medecinId": 8,
  "medecin": {
    "id": 8,
    "nom": "Benzabour",
    "email": "Benzabour@gamil.com",
    "specialite": "Dentiste"
  },
  "patientId": 8,
  "patient": {
    "id": 8,
    "nom": "saïda",
    "dateNaissance": "2024-01-11T00:00:00.000+00:00",
    "malade": true
  }
}
```

Response headers

```
connection: keep-alive
content-type: application/hal+json
```

Supprimer un rendezVous

Curl

```
curl -X 'DELETE' \
'http://localhost:8082/rendezVous/19' \
-H 'accept: */*'
```

Request URL

```
http://localhost:8082/rendezVous/19
```

Server response

Code	Details
200	<div>Response headers<div>connection: keep-alive content-length: 0 date: Thu,11 Jan 2024 16:04:26 GMT keep-alive: timeout=60</div></div>

Responses

- **Consultation :**

consultation-entity-controller		^
GET	/consultations	▼
POST	/consultations	▼
GET	/consultations/{id}	▼
PUT	/consultations/{id}	▼
DELETE	/consultations/{id}	▼

Afficher tous les consultations

Curl

```
curl -X 'GET' \
  'http://localhost:8082/consultations?page=0&size=20' \
  -H 'accept: application/hal+json'
```

Request URL

```
http://localhost:8082/consultations?page=0&size=20
```

Server response

Code	Details
------	---------

200

Response body

```
[
  {
    "id": 10,
    "dateConsultation": "2024-01-11",
    "rapport": "Rapport de la consultation"
  },
  {
    "id": 11,
    "dateConsultation": "2024-01-11",
    "rapport": "Rapport de la consultation"
  },
  {
    "id": 12,
    "dateConsultation": "2024-01-11",
    "rapport": "Rapport de la consultation"
  },
  {
    "id": 13,
    "dateConsultation": "2024-01-11",
    "rapport": "Rapport de la consultation"
  },
  {
    "id": 14,
    "dateConsultation": "2024-01-11",
    "rapport": "Rapport de la consultation"
  }
]
```

Ajouter une consultation

Curl

```
curl -X 'POST' \
  'http://localhost:8082/consultations' \
  -H 'accept: application/hal+json' \
  -H 'Content-Type: application/json' \
  -d '{
    "id": 16,
    "dateConsultation": "2024-01-11T16:07:34.278Z",
    "rapport": "string",
    "rendezVous": {
      "id": 19
    }
  }'
```

Request URL

http://localhost:8082/consultations

Server response

Code

Details

200

Undocumented

Response body

```
{
  "id": 15,
  "dateConsultation": "2024-01-11",
  "rapport": "string"
}
```

Response headers

```
connection: keep-alive
content-type: application/hal+json
date: Thu, 11 Jan 2024 16:11:52 GMT
keep-alive: timeout=60
transfer-encoding: chunked
```

Afficher une consultation par son identifiant

Curl

```
curl -X 'GET' \
  'http://localhost:8082/consultations/15' \
  -H 'accept: application/hal+json'
```

Request URL

http://localhost:8082/consultations/15

Server response

Code	Details
200	<p>Response body</p> <pre>{ "id": 15, "dateConsultation": "2024-01-11", "rapport": "string" }</pre> <p>Response headers</p> <pre>connection: keep-alive content-type: application/hal+json date: Thu,11 Jan 2024 16:14:01 GMT keep-alive: timeout=60 transfer-encoding: chunked</pre>

Mettre à jour une consultation

Curl

```
curl -X 'PUT' \
  'http://localhost:8082/consultations/15' \
  -H 'accept: application/hal+json' \
  -H 'Content-Type: application/json' \
  -d '{
    "id": 15,
    "dateConsultation": "2024-01-11T16:14:31.734Z",
    "rapport": "Some Changes!!"
  }'
```

Request URL

http://localhost:8082/consultations/15

Server response

Code	Details
200	<p>Response body</p> <pre>{ "id": 15, "dateConsultation": "2024-01-11T16:14:31.734+00:00", "rapport": "Some Changes!!" }</pre> <p>Response headers</p> <pre>connection: keep-alive content-type: application/hal+json date: Thu,11 Jan 2024 16:15:04 GMT keep-alive: timeout=60 transfer-encoding: chunked</pre>

Supprimer une consultation

Curl

curl -X 'DELETE' \
'http://localhost:8082/consultations/15' \
-H 'accept: */*'

Request URL

http://localhost:8082/consultations/15

Server response

Code

Details

200

Undocumented

Response headers

connection: keep-alive
content-length: 0
date: Thu,11 Jan 2024 16:15:33 GMT
keep-alive: timeout=60

Responses

Conclusion

L'approche microservices adoptée pour la conception de l'application de gestion des rendez-vous et consultations médicales a démontré une amélioration significative de la modularité et de la scalabilité du système. Grâce à la séparation des préoccupations, chaque service peut être développé, déployé et mis à jour indépendamment, permettant une maintenance et une évolution plus aisées de l'application. L'utilisation d'une API Gateway a simplifié l'interaction entre les clients et les services, tandis que l'intégration du registre de services et du serveur de configuration a renforcé la cohérence et la fiabilité de la configuration.

vous pouvez trouver le code source du projet dans mon github : [Microservices-Mini-Projet](#)