

**MICROSERVICES**  
**AMÉLIORATION DE L'OBSERVABILITÉ DANS LES**  
**MICROSERVICES**  
**“Pattern d’Observability”**  
**Application de Gestion des Patients avec**  
**Microservices**

**Préparé par :**

- EL AYADI Fatima Ezzahra
- BOUYARMANE Chaimae
- EZZAGRANI Habiba

**Sous la direction de :**

**Mme.LAMHARHAR Hind**

**Filière : Master M2SI**

**Année universitaire : 2023 - 2024**

# Table des matières:

<b>Table des matières:</b> .....	<b>2</b>
<b>I. Introduction :</b> .....	<b>4</b>
<b>II. Cahier des charges : Application de Gestion des Patients avec Microservices.....</b>	<b>5</b>
A. Introduction :.....	5
B. Fonctionnalités Principales :.....	5
C. Intégration Eureka :.....	5
D. Implémentation des Patterns :.....	5
<b>III. Modélisation des Microservices : Processus, Responsabilités et Workflow.....</b>	<b>7</b>
A. Modélisation du Processus :.....	7
B. Définition des Microservices :.....	7
➤ Gestion des Médecins :.....	7
➤ Gestion des Patients:.....	7
➤ Gestion des Rendez-vous:.....	7
C. Responsabilités et Rôles :.....	8
D. Workflow Explicatif :.....	8
<b>IV. Architecture des Microservices et Patterns d'Observabilité Utilisés :.....</b>	<b>10</b>
A. Architecture Globale :.....	10
1. Langage de Programmation et Environnement de Développement:.....	10
2. Microservices et Communication :.....	11
3. Gestion des Données :.....	11
4. Conteneurisation :.....	11
5. Outils d'Observabilité :.....	11
B. Patterns d'Observabilité :.....	12
<b>V. Implémentation des microservices fonctionnels :.....</b>	<b>14</b>
A. Microservices des patients :.....	15
B. Microservices des médecins :.....	15
C. Microservices des rendez-vous :.....	16
D. Eureka Server :.....	17
1. Implémentation de Eureka server - côté serveur :.....	17
2. Implémentation de Eureka server - côté client :.....	17
E. Implémentation du pattern :.....	18
<b>VI. Test :.....</b>	<b>19</b>
A. Déploiement sur Eureka server :.....	20
B. Test des APIs :.....	20
1. Gestion des medecins :.....	20
2. Gestion des rendez-vous et des patients :.....	21
C. Visualisation du Pattern :.....	23
1. Grafana :.....	23
2. Loki :.....	23
3. Prometheus :.....	23
<b>VII. Conclusion:.....</b>	<b>25</b>

# I. Introduction :

L'architecture des microservices, émergeant comme une réponse aux besoins évolutifs de la technologie moderne, représente une approche révolutionnaire dans la conception des systèmes informatiques. Fondée sur le principe de la décomposition d'une application en composants autonomes et interopérables, cette structure favorise l'agilité, la scalabilité et la résilience. Théoriquement, chaque microservice, agissant comme une entité autonome, exerce une fonction spécifique au sein du système global. Cette modularité permet une gestion indépendante, le déploiement continu, et une adaptabilité dynamique aux changements. Au cœur de cette architecture résident des défis complexes tels que la gestion des communications inter-microservices, la surveillance des performances, et la coordination des mises à jour. Cette introduction théorique vise à jeter les bases conceptuelles nécessaires pour comprendre la dynamique et les principes sous-jacents à l'architecture générale des microservices. Ce rapport vise à explorer en profondeur la mise en œuvre d'un projet dédié au "Pattern Observability" dans le contexte des microservices. En analysant les fondements de l'observabilité, les liens étroits entre cette dernière et la résilience des microservices, ainsi que les objectifs clairs pour l'amélioration, nous cherchons à fournir des insights significatifs pour optimiser la performance, la stabilité, et la réactivité de notre système. En nous appuyant sur une approche systématique, notre objectif est d'offrir des perspectives pratiques pour naviguer avec succès dans ce paysage technologique dynamique.

## **II. Cahier des charges : Application de Gestion des Patients avec Microservices**

### **A. Introduction :**

- **Objectif** : Développer une application de gestion des patients permettant l'ajout, la suppression et la modification des médecins, patients, et rendez-vous.
- **Architecture** : Utiliser des microservices pour une scalabilité et une flexibilité optimales.
- **Intégration Eureka** : Intégrer le service de découverte Eureka pour la gestion dynamique des microservices.

### **B. Fonctionnalités Principales :**

#### **Microservice Médecins** :

- Ajout de médecins avec leurs détails (nom, spécialité, etc.).
- Modification des informations des médecins existants.
- Suppression des médecins du système.

#### **Microservice Patients** :

- Ajout de nouveaux patients avec informations médicales.
- Modification des détails des patients existants.
- Suppression de patients de la base de données.

#### **Microservice Rendez-vous** :

- Ajout de rendez-vous entre médecins et patients.
- Modification des détails des rendez-vous existants.
- Suppression des rendez-vous programmés.

### **C. Intégration Eureka :**

- Configuration d'Eureka comme service de découverte pour la gestion dynamique des microservices.
- Enregistrement automatique des microservices auprès du serveur Eureka.
- Possibilité de localiser et d'accéder aux microservices de manière transparente.

### **D. Implémentation des Patterns :**

#### **Logs** :

- Intégration du pattern logs pour enregistrer des événements significatifs dans chaque microservice.
- Utilisation de logs pour le débogage, la sécurité, et la traçabilité des actions.

#### **Métriques** :

- Implémentation du pattern métriques pour mesurer quantitativement les performances de chaque microservice.
- Surveillance de la charge, de la latence, et de l'utilisation des ressources.

### **Traces :**

- Intégration du pattern traces pour suivre le cheminement des transactions à travers les microservices.
- Facilitation de la visualisation des interactions entre services et identification des points de latence.

## **III. Modélisation des Microservices : Processus, Responsabilités et Workflow**

L'étape cruciale de modélisation du processus et de définition des microservices constitue le fondement de notre projet. Chaque microservice, soigneusement conçu pour répondre à des besoins spécifiques, assume des responsabilités claires au sein de notre système de gestion des patients. La détermination précise des rôles de chaque microservice façonne la dynamique globale du projet, tandis que l'explication détaillée du workflow offre une vision cohérente de la manière dont ces entités interagissent pour atteindre nos objectifs.

### **A. Modélisation du Processus :**

Commencant par une analyse approfondie des fonctionnalités attendues, la modélisation du processus détaille le flux d'activités nécessaires à la gestion complète des patients. Chaque étape, de l'ajout initial des médecins et patients à la planification et la modification des rendez-vous, est soigneusement cartographiée pour garantir une compréhension exhaustive des interactions au sein du système.

### **B. Définition des Microservices :**

Chaque microservice a été défini avec précision en fonction de ses responsabilités spécifiques dans le projet de gestion des patients avec l'utilisation du pattern observabilité. Voici les définitions de chaque microservice :

#### **> Gestion des Médecins :**

Ce microservice est chargé de la gestion complète des informations relatives aux médecins du système. Il assure la création, la mise à jour et la suppression des profils des médecins, offrant ainsi une vue centralisée et actualisée de ces professionnels de la santé.

#### **> Gestion des Patients:**

Ce microservice se concentre sur la gestion des données liées aux patients. Il permet l'ajout, la modification et la suppression des informations médicales des patients, favorisant ainsi une prise en charge efficace et personnalisée au sein du système.

#### **> Gestion des Rendez-vous:**

Le microservice de gestion des rendez-vous facilite la planification et la gestion des rendez-vous entre médecins et patients. Il offre des fonctionnalités d'ajout, de modification et de suppression des rendez-vous, garantissant une coordination optimale dans le cadre des consultations médicales.

Chaque microservice, tout en opérant de manière indépendante, contribue de manière synergique à l'ensemble du système, permettant une gestion fluide et efficiente des différentes entités liées à la santé et aux soins des patients.

## C. Responsabilités et Rôles :

Ce tableau offre une vue concise des rôles spécifiques et des responsabilités principales de chaque membre de l'équipe dans le contexte du projet de gestion des patients avec l'utilisation du pattern observabilité.

Membre de l'Équipe	Microservice Assigné	Rôles et Responsabilités
Fatima Ezzahra EL AYADI	Gestion des Médecins	- Conception et développement du microservice de gestion des médecins.
Habiba Ezzagrani	Gestion des Patients	- Conception et développement du microservice de gestion des patients.
Chaimae BOUYAR MANE	Gestion des Rendez-vous	- Conception et développement du microservice de gestion des rendez-vous.

## D. Workflow Explicatif :

L'explication détaillée du workflow dévoile la séquence d'actions coordonnées entre les microservices, offrant une vue holistique sur la manière dont le système de gestion des patients s'articule.

### ➤ Ajout d'un Médecin :

- L'utilisateur initie une demande d'ajout d'un nouveau médecin via l'interface utilisateur.
- Le microservice de gestion des médecins traite la requête, vérifie les données, et ajoute le médecin à la base de données.

### ➤ Ajout d'un Patient :

- Une demande d'ajout d'un nouveau patient est émise par l'utilisateur.
- Le microservice de gestion des patients prend en charge la requête, valide les données, et intègre le nouveau patient dans le système.

### ➤ Planification d'un Rendez-vous :

- L'utilisateur programme un rendez-vous en spécifiant le médecin et le patient concernés.
- Le microservice de gestion des rendez-vous coordonne l'opération, vérifie la disponibilité, et enregistre le rendez-vous dans la base de données.

➤ **Modification des Informations d'un Patient :**

- Une demande de modification des informations d'un patient est générée.
- Le microservice de gestion des patients prend en compte la requête, effectue les mises à jour nécessaires, et assure la cohérence des données.

➤ **Suppression d'un Rendez-vous :**

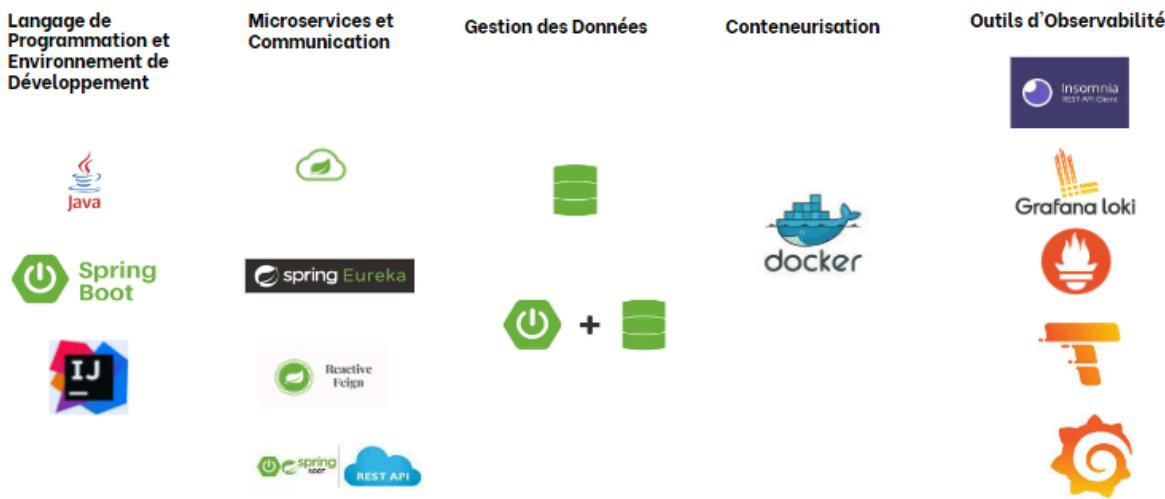
- L'utilisateur demande la suppression d'un rendez-vous existant.
- Le microservice de gestion des rendez-vous gère la requête, vérifie les autorisations, et retire le rendez-vous de la base de données.

Ce workflow assure une coordination fluide entre les microservices, reflétant les interactions essentielles du système de gestion des patients. Chaque étape est conçue pour garantir la cohérence des données, l'intégrité du système, et une expérience utilisateur optimale.

# IV. Architecture des Microservices et Patterns d'Observabilité Utilisés :

## A. Architecture Globale :

Notre architecture microservices repose sur les frameworks Spring Boot et Spring Cloud, offrant une approche modulaire et distribuée. Voici une vue d'ensemble de notre architecture :



### 1. Langage de Programmation et Environnement de Développement:

- **Java et Maven** : Le développement est réalisé en utilisant le langage Java, avec la gestion efficace des dépendances orchestrée par Maven. Maven assure la cohérence du développement et facilite le cycle de vie du projet.
- **Spring Boot et Spring Cloud** : Notre architecture microservices est construite sur les frameworks Spring Boot et Spring Cloud, fournissant une approche modulaire et distribuée pour le développement de microservices.
- **IntelliJ** : L'environnement de développement intégré (IDE) principal pour la création et la gestion des microservices en raison de sa robustesse et de ses fonctionnalités avancées.

### 2. Microservices et Communication :

- **Spring Cloud et Eureka** : Utilisation de Spring Cloud pour la création d'une architecture microservices. Eureka est intégré en tant que service de découverte pour simplifier la localisation dynamique des microservices.
- **Feign Client** : Facilitation de la communication déclarative entre microservices, renforçant l'interopérabilité et simplifiant les échanges d'informations.
- **RESTful API** : Au cœur de notre architecture microservices se trouve une RESTful API. Cette API expose les fonctionnalités nécessaires pour la gestion des patients à travers des points d'accès REST.

### 3. Gestion des Données :

- **Spring Data** : Utilisation de Spring Data comme couche d'abstraction pour simplifier l'accès et la manipulation des données dans nos microservices.
- **H2** : La base de données H2 est utilisée pendant le développement pour faciliter la gestion des données de manière efficace.

### 4. Conteneurisation :

- **Docker** : Choix de Docker pour la conteneurisation, simplifiant ainsi le déploiement et la gestion des microservices dans des environnements variés.

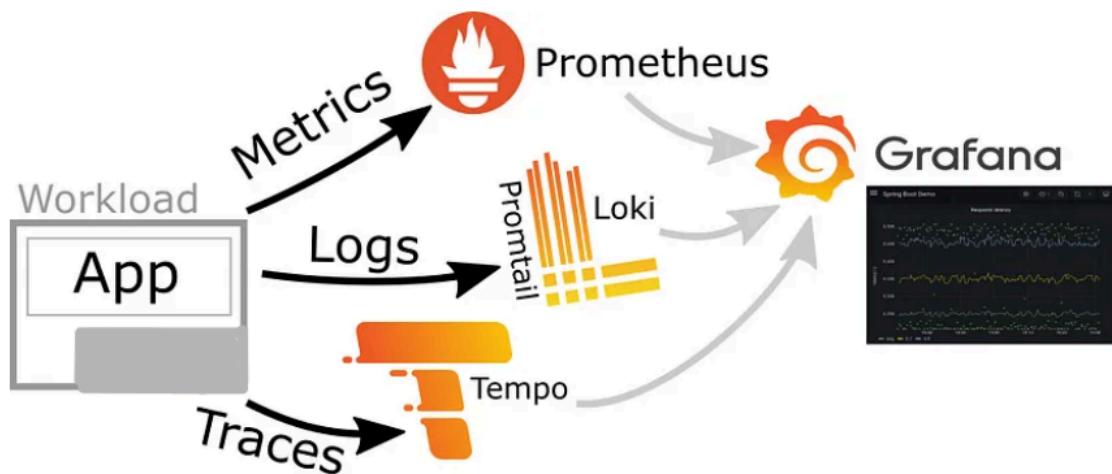
### 5. Outils d'Observabilité :

- **Insomnia** : Incorporé dans le processus de développement pour tester et valider les API de manière efficace, assurant la qualité des interactions entre les services.
- **Grafana** : Tableau de bord essentiel offrant une visualisation personnalisée des métriques et des journaux collectés. Facilite la surveillance proactive et l'intervention rapide en cas de besoin.
- **Loki** : Système de gestion de journaux offrant une solution efficace pour stocker et rechercher des journaux distribués, crucial pour le suivi et la résolution des problèmes.
- **Prometheus** : Outil robuste central dans notre approche d'observabilité, collectant et stockant des métriques essentielles pour une analyse approfondie des performances du système.

- **Tempo** : Intégré pour le traçage distribué, offrant une visibilité approfondie dans le comportement des microservices, facilitant ainsi le diagnostic des problèmes et l'optimisation des performances.

## B. Patterns d'Observabilité :

Notre architecture d'observabilité est soigneusement conçue pour fournir une visibilité approfondie dans le comportement, les performances et les logs de notre application de gestion des patients basée sur microservices. Voici une description détaillée de chaque composant de cette architecture :



### Workload (Application) :

- Au cœur de notre système, le "Workload" représente l'ensemble de nos microservices responsables de la gestion des patients. Ces microservices fonctionnent en tandem pour fournir des fonctionnalités spécifiques, créant ainsi une application distribuée et modulaire.

### Metrics (Prometheus) :

- Prometheus agit comme notre système central de collecte et de stockage de métriques. Il analyse en profondeur les performances du système en recueillant des données essentielles telles que le temps de réponse, le débit, l'utilisation des ressources, etc. L'architecture multi-dimensionnelle de

Prometheus s'aligne parfaitement sur notre approche microservices, fournissant une vue holistique des métriques système.

#### **Logs (Loki) :**

- Loki a joué un rôle critique dans la gestion des logs distribués de notre application. Il offre une solution efficace pour stocker et rechercher des journaux, permettant ainsi un suivi précis des activités du système. Les logs provenant de chaque microservice sont centralisés dans Loki, facilitant la recherche et la résolution des problèmes.

#### **Traces (Tempo) :**

- Tempo est notre composant dédié au traçage distribué. Il offre une visibilité approfondie dans le comportement des microservices en permettant la capture et la visualisation des traces d'exécution. Cela facilite le diagnostic des problèmes, l'optimisation des performances et une compréhension claire des interactions entre les microservices.

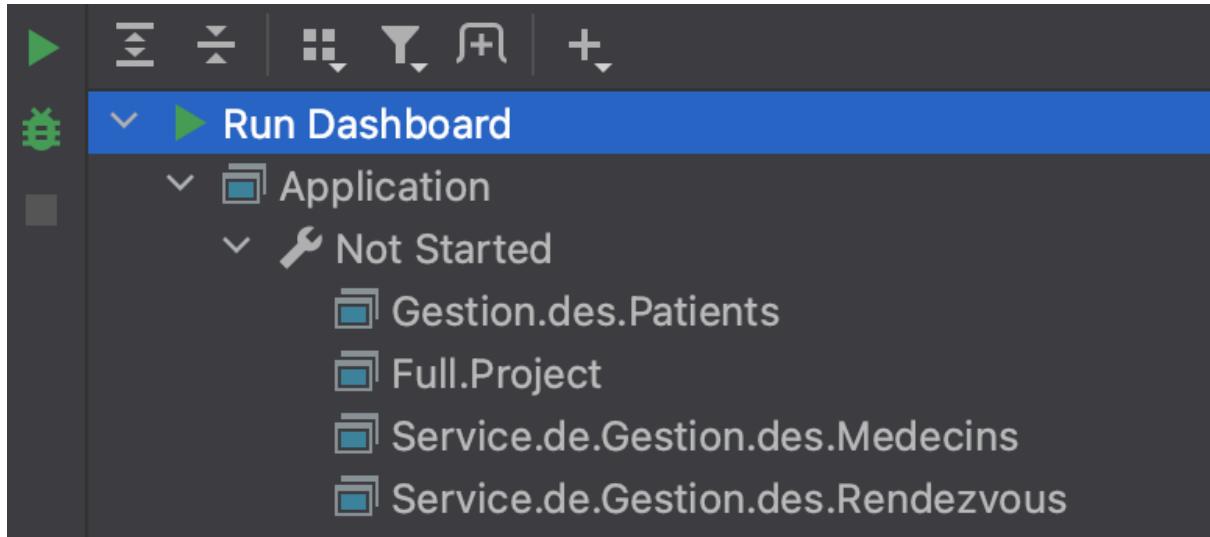
#### **Visualisation (Grafana) :**

- Grafana constitue la pièce maîtresse de notre infrastructure d'observabilité. Il sert de tableau de bord personnalisé, agrégeant les métriques, logs et traces collectés par Prometheus, Loki et Tempo. Les tableaux de bord de Grafana offrent une visualisation claire et personnalisée de la santé du système, permettant une surveillance proactive et une intervention rapide en cas de besoin. Ils peuvent être ajustés finement pour mettre en évidence les métriques clés et fournir une vue complète de la performance et de la fiabilité du système.

## **V. Implémentation des microservices fonctionnels :**

Ce rapport explore en profondeur une application novatrice conçue autour d'une architecture de microservices. Cette application révolutionnaire se concentre sur trois piliers essentiels : la gestion des patients, la planification efficace des rendez-vous et la centralisation des données médicales des médecins. En fusionnant ces aspects

dans un écosystème intégré, cette application redéfinit la manière dont les établissements de santé abordent la gestion administrative et offre une expérience utilisateur améliorée. Ce rapport détaillera l'architecture, les fonctionnalités et les avantages de cette application, soulignant son impact dans le domaine médical actuel.



## A. Microservices des patients :

Notre premier microservice gère les patients en implémentant des méthodes de crud simple .

```

package MicroServices.ObservabilityPattern.Gestion.des.Patients.Controllers;
import ...;

no usages ▲ Habiba Ezzagragi *
@RestController @RequestMapping("patient")
public class PatientController {
    @Autowired
    private PatientService patientService;
    @Autowired
    private RendezvousFeignClient rendezvousFeignClient;

    @GetMapping("/getAll") @ResponseStatus(HttpStatus.OK)
    public List<Patient> findAll() { return patientService.findAll(); }

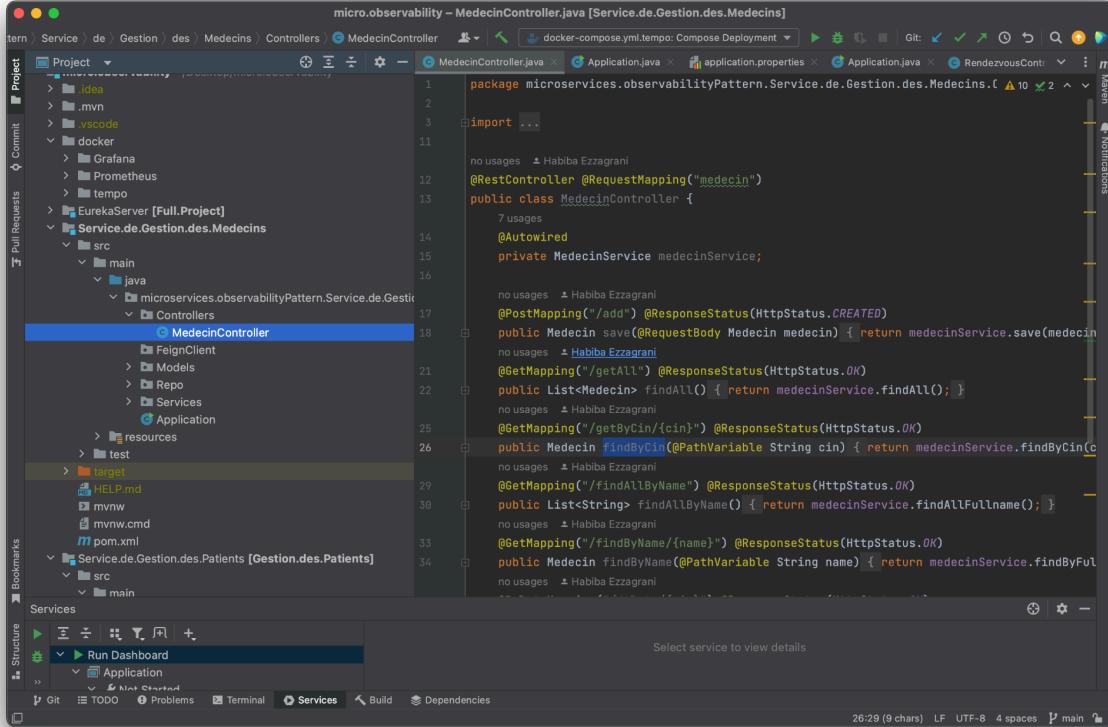
    @GetMapping("/byCIN/{cin}") @ResponseStatus(HttpStatus.OK)
    public Patient getByCin(@PathVariable String cin) { return patientService.findByCin(cin); }

    @PostMapping("/add")
    @ResponseStatus(HttpStatus.CREATED)
    public Patient save(@RequestBody Patient patient) {
        try {
            Patient existingPatient = patientService.findByCin(patient.getCin());
        }
    }
}

```

## B. Microservices des médecins :

Même implémentation concernant le microservice qui gère les médecins .



The screenshot shows the IntelliJ IDEA interface with the project navigation bar at the top. The main area displays the code for `MedecinController.java`. The code implements a REST controller for managing doctors. It includes methods for saving a doctor, getting all doctors, finding a doctor by CIN, and finding a doctor by name. The code uses annotations like `@RestController`, `@RequestMapping`, and `@Autowired`. The code editor shows several comments from a developer named Habiba Ezzagragi. Below the code editor, the Services panel is visible, showing a list of services including "Run Dashboard" and "Application". The bottom status bar indicates the current time as 26:29 and file encoding as UTF-8.

```
micro.observability - MedecinController.java [Service.de.Gestion.des.Medecins]
1 package microservices.observabilityPattern.Service.de.Gestion.des.Medecins;
2 import ...
3
4 no usages ▲ Habiba Ezzagragi
5 @RestController @RequestMapping("medecin")
6 public class MedecinController {
7
8     @Autowired
9     private MedecinService medecinService;
10
11     no usages ▲ Habiba Ezzagragi
12     @PostMapping("/add") @ResponseStatus(HttpStatus.CREATED)
13     public Medecin save(@RequestBody Medecin medecin) { return medecinService.save(medecin);
14     no usages ▲ Habiba Ezzagragi
15     @GetMapping("/") @ResponseStatus(HttpStatus.OK)
16     public List<Medecin> findAll() { return medecinService.findAll(); }
17     no usages ▲ Habiba Ezzagragi
18     @GetMapping("/getByCin/{cin}") @ResponseStatus(HttpStatus.OK)
19     public Medecin findByCin(@PathVariable String cin) { return medecinService.findByCin(c
20     no usages ▲ Habiba Ezzagragi
21     @GetMapping("/findAllByName") @ResponseStatus(HttpStatus.OK)
22     public List<String> findAllByName() { return medecinService.findAllfullname(); }
23     no usages ▲ Habiba Ezzagragi
24     @GetMapping("/findByName/{name}") @ResponseStatus(HttpStatus.OK)
25     public Medecin findByName(@PathVariable String name) { return medecinService.findByName(
26     no usages ▲ Habiba Ezzagragi
27
28
29
30
31
32
33
34
```

## C. Microservices des rendez-vous :

Même implémentation concernant le microservice qui gère les rendez-vous .

The screenshot shows an IDE interface with the following details:

- Project View:** Shows the project structure under "micro.observability". The "Service.de.Gestion.des.Rendez-vous" module is selected, revealing its internal structure: src/main/java/microservices/observabilityPattern/Service/de/Gestion/des/Rendez-vous/Controllers/RendezvousController.java.
- Code Editor:** Displays the content of `RendezvousController.java`. The code implements a REST controller for managing rendezvous. It includes imports for `java.util.List`, `java.util.Optional`, and various annotations like `@RestController`, `@RequestMapping`, `@Autowired`, and `@PostMapping`. The code handles saving a rendezvous by calling a patient Feign client and a doctor Feign client.
- Services Panel:** Shows a "Run Dashboard" entry under the "Application" section.
- Bottom Status Bar:** Displays the time as 19:15, file encoding as LF, character set as UTF-8, and code style as 4 spaces.

```
import ...  
no usages ▲ Habiba Ezzagragni  
@RestController @RequestMapping("rendezvous")  
public class RendezvousController {  
    6 usages  
    @Autowired  
    private RendezvousService rendezvousService;  
    1 usage  
    @Autowired  
    private MedecinFeignClient medecinFeignClient;  
    3 usages  
    @Autowired  
    private PatientFeignClient patientFeignClient;  
  
    no usages ▲ Habiba Ezzagragni  
    @PostMapping("/add")  
    public Rendezvous save(@RequestBody RequestBodyData data) {  
        try {  
            PatientDTO patientDTO = data.getPatient();  
            Rendezvous rendezvousDTO = data.getRendezvous();  
            MedecinDTO medecinName = data.getMedecin();  
  
            PatientDTO savedPatient = patientFeignClient.save(patientDTO);  
  
            if (savedPatient == null) {  
                return new Rendezvous();  
            } else {  
                rendezvousDTO.setPatient(savedPatient);  
                rendezvousDTO.setMedecin(medecinName);  
                rendezvousDTO.setFeignClient(feignClient);  
                rendezvousService.save(rendezvousDTO);  
                return rendezvousDTO;  
            }  
        } catch (Exception e) {  
            e.printStackTrace();  
            return new Rendezvous();  
        }  
    }  
}
```

## D. Eureka Server :

### 1. Implémentation de Eureka server - côté serveur :

The screenshot shows a code editor with several tabs at the top: MedecinController.java, Project/Application.java (which is currently selected), and application.properties. The Application.java file contains the following code:

```
1 package Parent.MicroServices.Observability.Full.Project;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5 import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;
6
7 @SpringBootApplication    @EnableEurekaServer
8 public class Application {
9
10    public static void main(String[] args) {
11        SpringApplication.run(Application.class, args);
12    }
13
14 }
15
```

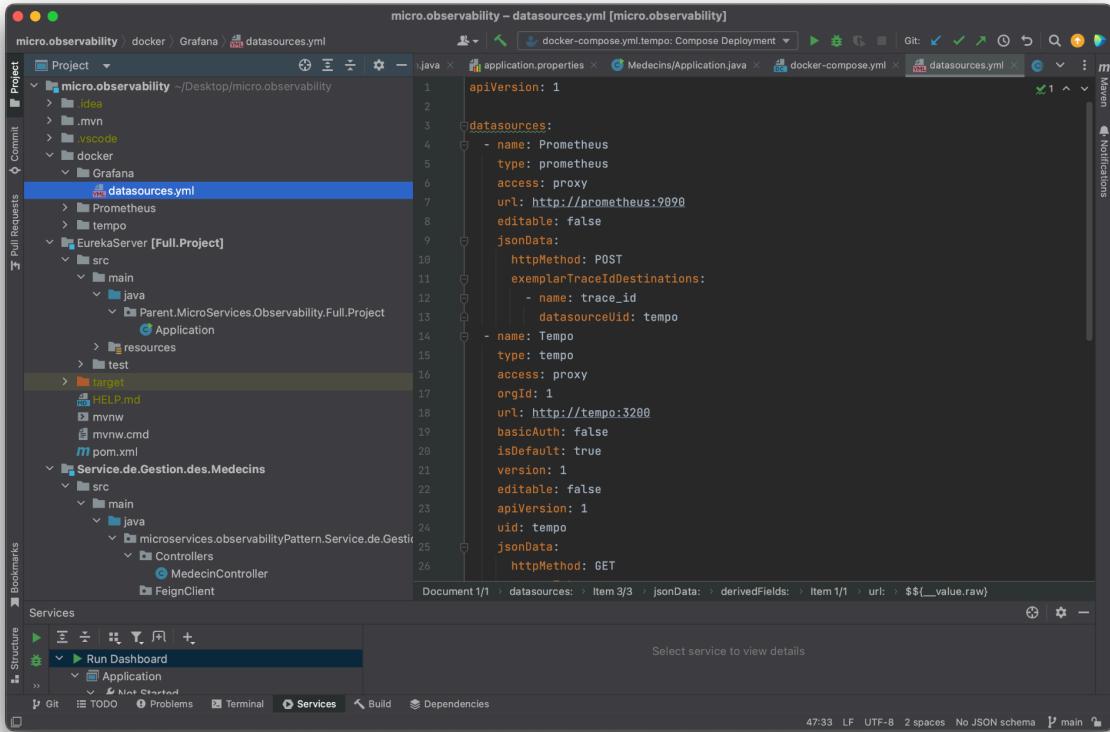
### 2. Implémentation de Eureka server - côté client :

The screenshot shows a code editor with several tabs at the top: MedecinController.java, Project/Application.java, application.properties, and Medecins/ (which is currently selected). The Application.java file contains the following code:

```
1 package microservices.observabilityPattern.Service.de.Gestion.des.
2
3 import ...
4
5 @SpringBootApplication @EnableDiscoveryClient @EnableFeignClients
6 public class Application {
7
8
9    public static void main(String[] args) {
10        SpringApplication.run(Application.class, args);
11    }
12
13 }
14
15 }
```

## E. Implémentation du pattern :

### 1. Grafana :



The screenshot shows a code editor with the file `datasources.yml` open. The file defines two data sources:

```
apiVersion: 1
datasources:
  - name: Prometheus
    type: prometheus
    access: proxy
    url: http://prometheus:9090
    editable: false
    jsonData:
      httpMethod: POST
      exemplarTraceIdDestinations:
        - name: trace_id
          dataSourceUid: tempo
  - name: Tempo
    type: tempo
    access: proxy
    orgId: 1
    url: http://tempo:3280
    basicAuth: false
    isDefault: true
    version: 1
    editable: false
    apiVersion: 1
    uid: tempo
    jsonData:
      httpMethod: GET
```

The project structure on the left includes `micro.observability`, `EurekaServer`, and `Service.de.Gestion.des.Medecins` projects, each with its own `src` and `java` directories.

### 2. Prometheus :

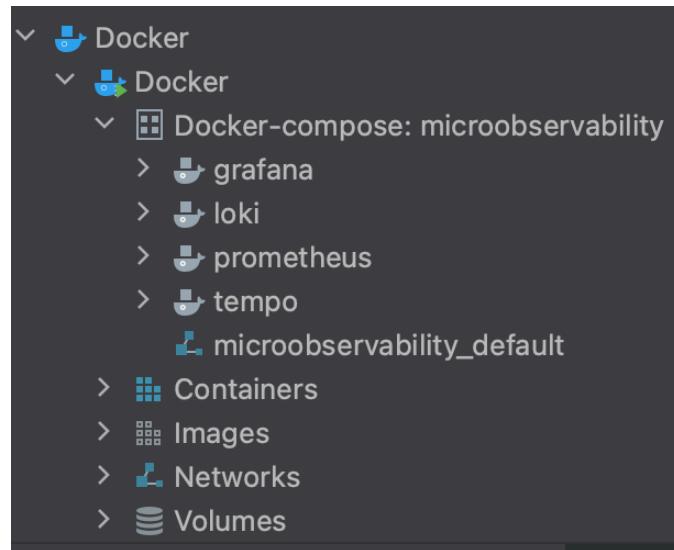
The screenshot shows the IntelliJ IDEA interface with the following details:

- Project View:** The left sidebar displays the project structure under "micro.observability". It includes modules like "micro.observability", "docker", "Prometheus", and "Service.de.Gestion.des.Medecins".
- Code Editor:** The main window shows a code editor with the file "prometheus.yml" open. The content is as follows:

```
global:  
  scrape_interval: 2s  
  evaluation_interval: 2s  
  
scrape_configs:  
  - job_name: 'prometheus'  
    static_configs:  
      - targets: ['prometheus:9090']  
  - job_name: 'Service.de.Gestion.des.Medecins'  
    metrics_path: '/actuator/prometheus'  
    static_configs:  
      - targets: ['host.docker.internal:8080'] ## only for demo purposes don't use host.docker.internal  
  - job_name: 'Gestion.des.Patients'  
    metrics_path: '/actuator/prometheus'  
    static_configs:  
      - targets: ['host.docker.internal:8081'] ## only for demo purposes don't use host.docker.internal  
  - job_name: 'Service.de.Gestion.des.Rendez-vous'  
    metrics_path: '/actuator/prometheus'  
    static_configs:  
      - targets: ['host.docker.internal:8081'] ## only for demo purposes don't use host.docker.internal
```

- Services Tab:** The bottom navigation bar has a "Services" tab, which is currently selected.
- Bottom Status Bar:** The status bar at the bottom shows the time as 17:50, and the schema as "prometheus.json".

### **3. Environnement docker :**



## **VI. Test :**

## A. Déploiement sur Eureka server :

On peut remarquer que tous les microservices ont été déployés sur Eureka server .

The screenshot shows the Eureka Server dashboard at [localhost:8761](http://localhost:8761). The top section displays system configuration:

Environment	test	Current time	2023-12-18T08:51:14 +0100
Data center	default	Uptime	00:01
		Lease expiration enabled	false
		Renews threshold	6
		Renews (last min)	0

The "DS Replicas" section shows registered instances:

Application	AMIs	Availability Zones	Status
GESTION.DES.PATIENTS	n/a (1)	(1)	UP (1) - <a href="http://172.16.60.108:Gestion_des_Patients:8080">172.16.60.108:Gestion_des_Patients:8080</a>
SERVICE.DE.GESTION.DES.MEDECINS	n/a (1)	(1)	UP (1) - <a href="http://172.16.60.108:Service.de.Gestion.des.Medecins:8082">172.16.60.108:Service.de.Gestion.des.Medecins:8082</a>
SERVICE.DE.GESTION.DES.RENDEZ-VOUS	n/a (1)	(1)	UP (1) - <a href="http://172.16.60.108:Service.de.Gestion.des.Rendez-vous:8081">172.16.60.108:Service.de.Gestion.des.Rendez-vous:8081</a>

General Info

## B. Test des APIs :

### 1. Gestion des medecins :

a) Ajout d'un medecin :

POST <http://localhost:8082/medecin/add> Send 201 Created 173 ms 99 B

JSON Auth Query Headers 2 Docs Preview Headers 3 Cool

```
1 {  
2   "cin": "HH679",  
3   "fullname": "aziz",  
4   "mobile": "98765654",  
5   "gender": "male",  
6   "specialite": "dentist"  
7 }
```

```
1 {  
2   "id": 1,  
3   "cin": "HH679",  
4   "fullname": "aziz",  
5   "mobile": "98765654",  
6   "gender": "male",  
7   "specialite": "dentist"  
8 }
```

b) Modification d'un medecin :

PUT <http://localhost:8082/medecin/update> Send 200 OK 596 ms 102 B

JSON Auth Query Headers 2 Docs Preview Headers 3 Cool

```
1 {  
2   "cin": "HH679",  
3   "fullname": "aziza",  
4   "mobile": "98765654",  
5   "gender": "female",  
6   "specialite": "dentist"  
7 }
```

```
1 {  
2   "id": 1,  
3   "cin": "HH679",  
4   "fullname": "aziza",  
5   "mobile": "98765654",  
6   "gender": "female",  
7   "specialite": "dentist"  
8 }
```

c) Rechercher tous les médecins ou par CIN :

GET ▼ http://localhost:8082/medecin/getByCin/HH679

Send ▾ 200 OK 187 ms 99 B

Body ▾	Auth ▾	Query	Headers 1	Docs	Preview ▾	Headers 3	Cookies
--------	--------	-------	-----------	------	-----------	-----------	---------

```

1 ▶ {
2   "id": 1,
3   "cin": "HH679",
4   "fullname": "aziz",
5   "mobile": "98765654",
6   "gender": "male",
7   "specialite": "dentist"
8 }

```

GET ▼ http://localhost:8082/medecin/getAll

Send ▾ 200 OK 15.8 ms 212 B

Body ▾	Auth ▾	Query	Headers 1	Docs	Preview ▾	Headers 3	Cookies
--------	--------	-------	-----------	------	-----------	-----------	---------

```

1 ▶ [
2 ▶   {
3     "id": 1,
4     "cin": "HH679",
5     "fullname": "aziz",
6     "mobile": "98765654",
7     "gender": "male",
8     "specialite": "dentist"
9   },
10  {
11    "id": 2,
12    "cin": "HH67909",
13    "fullname": "aziz bensouda",
14    "mobile": "98765654",
15    "gender": "male",
16    "specialite": "dentist"
17  }
18 ]

```

d) Suppression d'un médecin :

DELETE ▼ http://localhost:8082/medecin/delete/HH679

Send ▾ 200 OK

Body ▾	Auth ▾	Query	Headers 1	Docs	Preview ▾
--------	--------	-------	-----------	------	-----------

1 HH679

## 2. Gestion des rendez-vous et des patients :

a) Réservation d'un rendez-vous par un patient :

POST ▼ http://localhost:8081/rendezvous/add

Send ▾ 200 OK 2.02 s 120 B

JSON ▾	Auth ▾	Query	Headers 2	Docs	Preview ▾	Headers 3	Cookies	Timeline
--------	--------	-------	-----------	------	-----------	-----------	---------	----------

```

1 ▶ {
2   "patient": {
3     "cin": "xyz",
4     "fullname": "chaimae",
5     "mobile": "0654637893",
6     "age": 22,
7     "gender": "female"
8   },
9   "rendezvous": {
10    "idrendezvous": 1,
11    "date": "2023-12-20T00:00:00.000+00:00",
12    "cause": "les bagues",
13  },
14  "medecin": {
15    "fullname": "aziz bensouda"
16  }
17 }

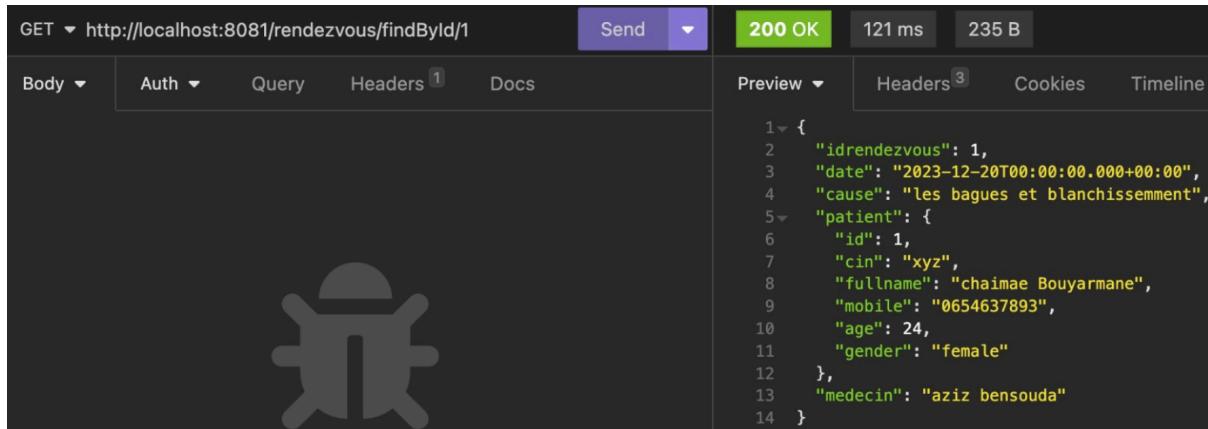
```

```

1 ▶ {
2   "idrendezvous": 1,
3   "date": "2023-12-20T00:00:00.000+00:00",
4   "cause": "les bagues",
5   "patient": "xyz",
6   "medecin": "aziz bensouda"
7 }

```

b) Rechercher un rendez-vous :



```

1 {  

2   "idrendezvous": 1,  

3   "date": "2023-12-20T00:00:00.000+00:00",  

4   "cause": "les bagues et blanchissement",  

5   "patient": {  

6     "id": 1,  

7     "cin": "xyz",  

8     "fullname": "chaimae Bouyarmane",  

9     "mobile": "0654637893",  

10    "age": 24,  

11    "gender": "female"  

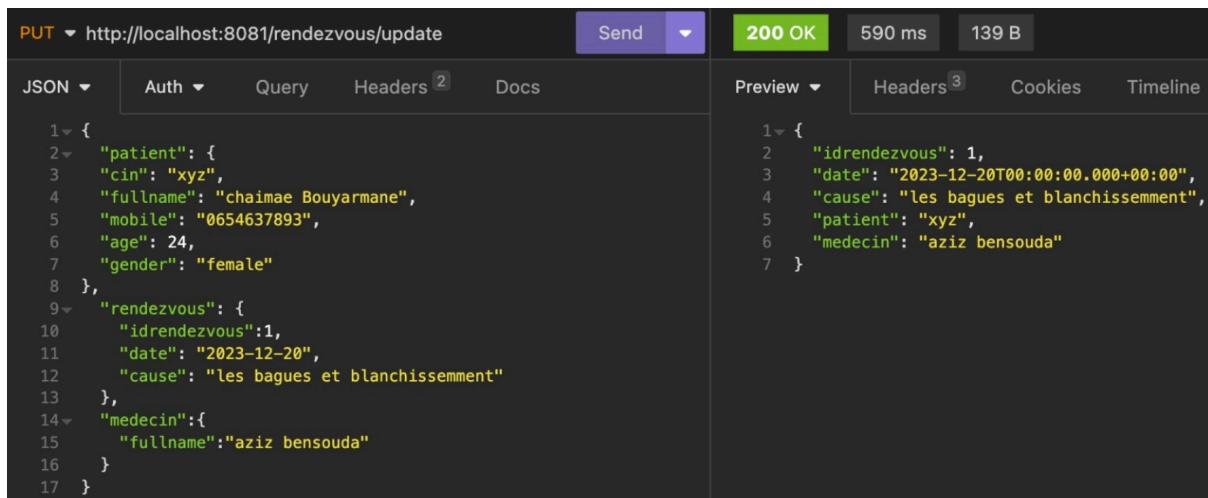
12  },  

13  "medecin": "aziz bensouda"  

14 }

```

c) Modification des informations d'un rendez-vous :



```

1 {  

2   "idrendezvous": 1,  

3   "date": "2023-12-20T00:00:00.000+00:00",  

4   "cause": "les bagues et blanchissement",  

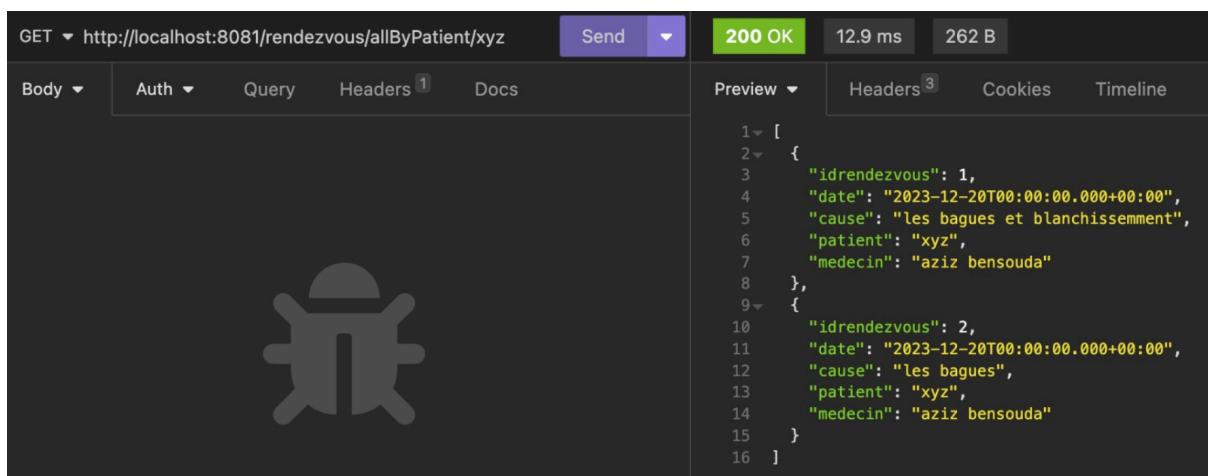
5   "patient": "xyz",  

6   "medecin": "aziz bensouda"  

7 }

```

d) Accéder à la liste des rendez-vous pris par un patient :



```

1 [  

2 {  

3   "idrendezvous": 1,  

4   "date": "2023-12-20T00:00:00.000+00:00",  

5   "cause": "les bagues et blanchissement",  

6   "patient": "xyz",  

7   "medecin": "aziz bensouda"  

8 },  

9 {  

10  "idrendezvous": 2,  

11  "date": "2023-12-20T00:00:00.000+00:00",  

12  "cause": "les bagues",  

13  "patient": "xyz",  

14  "medecin": "aziz bensouda"  

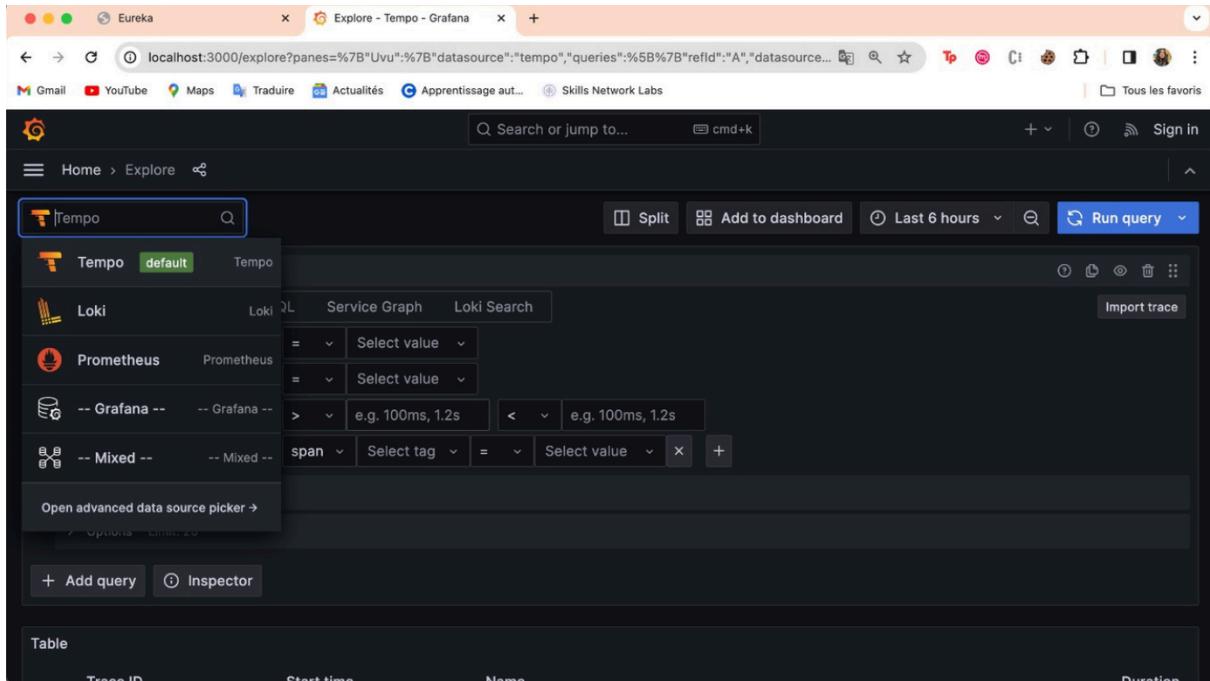
15 }  

16 ]

```

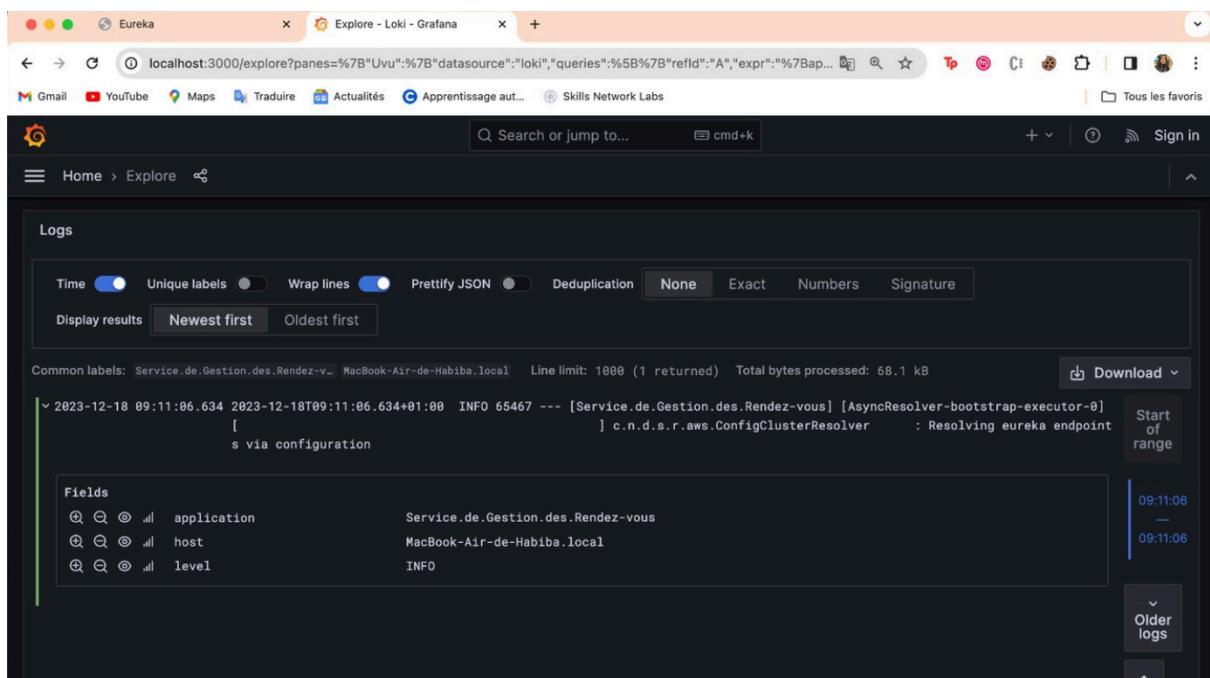
## C. Visualisation du Pattern :

### 1. Grafana :



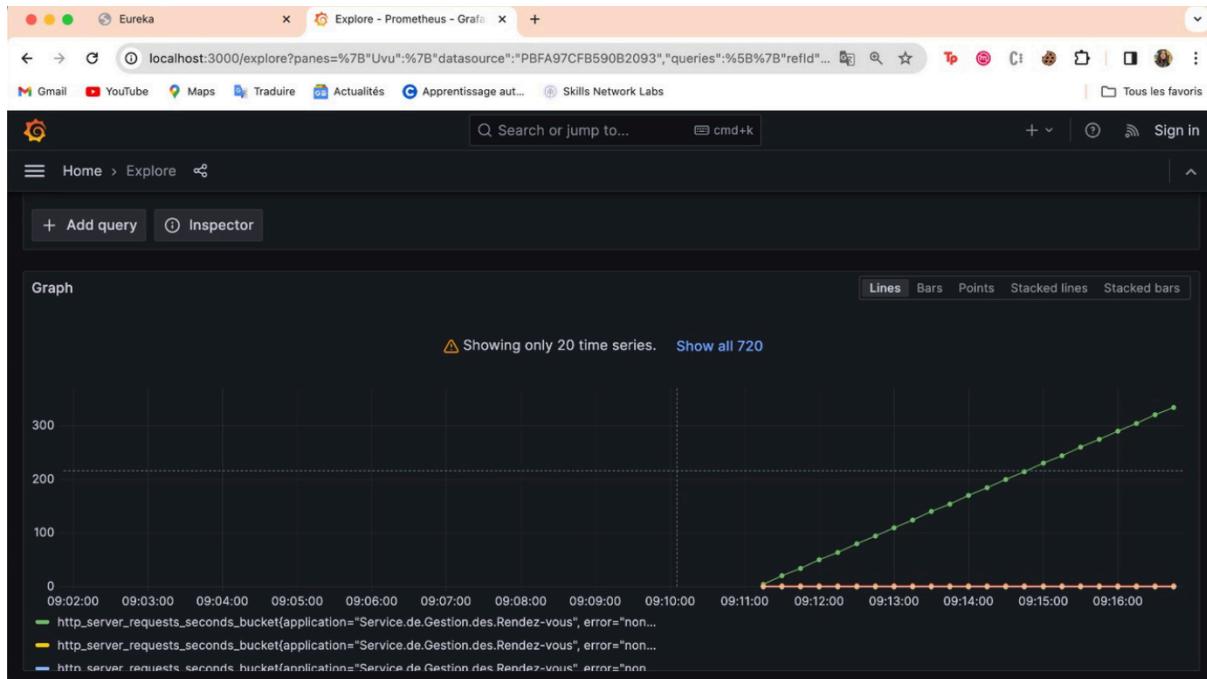
The screenshot shows the Grafana Explore interface. At the top, there's a search bar with the placeholder "Search or jump to...". Below it, a toolbar includes "cmd+k", "Split", "Add to dashboard", "Last 6 hours", "Run query", and "Import trace". The main area has a sidebar with data sources: "Tempo" (selected), "Loki", "Prometheus", "-- Grafana --", and "-- Mixed --". A "Service Graph" and "Loki Search" tab are also present. The "Tempo" section contains dropdowns for "Select value" and "Select tag", and input fields for "e.g. 100ms, 1.2s" and "e.g. 100ms, 1.2s". Below this is an "Open advanced data source picker" button. At the bottom, there are "Add query" and "Inspector" buttons. The main table view is titled "Table" and includes columns for "Trace ID", "Start time", "Name", and "Duration".

### 2. Loki :



The screenshot shows the Grafana Explore interface with the "Logs" panel selected. The top navigation bar and toolbar are identical to the previous screenshot. The main area is titled "Logs" and includes settings for "Time", "Unique labels", "Wrap lines", "Pretty JSON", "Deduplication", and "None", "Exact", "Numbers", "Signature" options. It also has "Display results" buttons for "Newest first" and "Oldest first". Below this, a log entry is shown: "2023-12-18 09:11:06.634 2023-12-18T09:11:06.634+01:00 INFO 65467 --- [Service.de.Gestion.des.Rendez-vous] [AsyncResolver-bootstrap-executor-0] [c.n.d.s.r.aws.ConfigClusterResolver : Resolving eureka endpoint s via configuration". On the right, there are "Download" and "Start of range" buttons, and a vertical timeline showing "09:11:06" and "Older logs". At the bottom left, there's a "Fields" section with checkboxes for "application", "host", and "level".

### 3. Prometheus :



## **VII. Conclusion:**

Ce projet a marqué notre première expérience avec Spring, et cela a été une expérience très instructive. En explorant les tenants et aboutissants de cette technologie tout en mettant en place une architecture microservices pour la gestion des patients, notre équipe a acquis des connaissances significatives qui ont grandement enrichi notre compréhension du développement avec des frameworks tels que Spring Boot et Spring Cloud.

L'intégration de ces nouvelles technologies a été source de nombreux apprentissages. La modularité et la scalabilité offertes par l'approche microservices se sont avérées essentielles, tandis que la gestion des données avec Spring Data a simplifié considérablement nos interactions avec la base de données. L'utilisation de Feign pour la communication entre microservices a été une découverte marquante, offrant une approche déclarative qui a renforcé l'unité de notre architecture.

Par ailleurs, l'intégration approfondie d'outils d'observabilité tels que Prometheus, Loki, Tempo et Grafana a été un point fort. Ces outils ont considérablement amélioré notre visibilité sur le fonctionnement de l'application, facilitant la détection précoce des problèmes et soutenant une amélioration continue des performances.

Cependant, ces découvertes n'ont pas été exemptes de défis, surtout compte tenu de notre expérience avec Spring. La complexité de la migration depuis une architecture monolithique a présenté des défis supplémentaires, en particulier en ce qui concerne la réorganisation des fonctionnalités existantes et la gestion des données. La coordination entre les microservices a continué d'être un défi, demandant une attention particulière pour assurer une synchronisation efficace des données et des opérations.

En résumé, cette première expérience avec Spring a été pleine d'apprentissages et de défis. Les connaissances acquises et les obstacles surmontés serviront de guide pour nos futures approches, renforçant ainsi notre capacité à aborder avec succès des projets similaires dans le domaine complexe des architectures distribuées.