

SUJET DE RATRAPAGE WEBSERVICES

1. CREATION DU PROJET ET MIS EN PLACE DE L'ENVIRONNEMENT

1.1 Outils nécessaires

- **Spring Boot** pour la création de l'API REST.
- **JAX-WS** pour l'intégration de SOAP dans l'application Spring Boot.
- **H2 Database** pour la simulation de la base de données de l'Agence de la CMU.
- **Maven** ou **Gradle** pour gérer les dépendances du projet.

• **1.2 Structure du projet**

Voici la structure du projet :

cmu-bridge-rest/

```
|
|
├── src/main/java/
|   ├── com/cmu/bridge/
|   |   ├── controller/      # Pour les endpoints REST
|   |   ├── service/        # Pour la logique de communication SOAP
|   |   ├── model/          # Pour la transformation des données
|   |   └── application/     # Classe principale Spring Boot
|   |
|   └── src/main/resources/
|       ├── application.properties # Configuration de Spring Boot
|       └── soap/               # WSDL, XSD pour le service SOAP
|
|
└── pom.xml                     # Dépendances Maven
```

2. Configuration du projet Spring Boot

Créez un projet Spring Boot avec les dépendances suivantes :

2.1 Dépendances Maven (pom.xml)

<dependencies>

<!-- Spring Web (REST API) -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<!-- Spring Boot DevTools (pour rechargement rapide) -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-devtools</artifactId>

<scope>runtime</scope>

</dependency>

<!-- JAX-WS pour la gestion des appels SOAP -->

<dependency>

<groupId>org.springframework.ws</groupId>

<artifactId>spring-webservices</artifactId>

<version>4.0.0</version>

</dependency>

<!-- H2 Database (pour la simulation de l'API CMU) -->

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<scope>runtime</scope>

</dependency>

<!-- Spring Data JPA pour l'accès à la base de données -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<!-- Pour la gestion des logs -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-logging</artifactId>

</dependency>

<!-- Spring Boot Starter Test (pour les tests) -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

3. Création d'un Service SOAP

Pour simuler le service SOAP de l'Agence de la CMU, nous allons mettre en place un simple service SOAP qui interagira avec une base de données H2.

3.1 Création de la classe SOAPService

La classe SOAPService sera responsable de la simulation des vérifications d'éligibilité dans l'API SOAP.

```
package com.cmu.bridge.service;
```

```
import org.springframework.stereotype.Service;
```

```
import javax.xml.soap.*;
```

```
import java.io.ByteArrayInputStream;
```

```
import java.io.InputStream;
```

```
@Service
```

```
public class SOAPService {
```

```
    public String checkEligibility(String phoneNumber) throws Exception {
```

```
        // Crée la requête SOAP pour appeler l'API CMU (simulation)
```

```
        String soapRequest = createSOAPRequest(phoneNumber);
```

```
        // Call SOAP service (ici, nous simulerons la réponse pour l'exemple)
```

```
        String soapResponse = simulateSOAPResponse(phoneNumber);
```

```

    return soapResponse; // Retourne la réponse SOAP simulée
}

private String createSOAPRequest(String phoneNumber) {

    return "<soapenv:Envelope
xmlns:soapenv=\"http://schemas.xmlsoap.org/soap/envelope/\"
xmlns:web=\"http://webservice.cmu.com/\">\n" +

        "    <soapenv:Header/>\n" +

        "    <soapenv:Body>\n" +

        "        <web:CheckEligibility>\n" +

        "            <phoneNumber>" + phoneNumber + "</phoneNumber>\n" +

        "        </web:CheckEligibility>\n" +

        "    </soapenv:Body>\n" +

        "</soapenv:Envelope>";

}

private String simulateSOAPResponse(String phoneNumber) {

    // Simule une réponse SOAP en fonction du numéro de téléphone

    return "<soapenv:Envelope
xmlns:soapenv=\"http://schemas.xmlsoap.org/soap/envelope/\"
xmlns:web=\"http://webservice.cmu.com/\">\n" +

        "    <soapenv:Header/>\n" +

        "    <soapenv:Body>\n" +

        "        <web:CheckEligibilityResponse>\n" +

        "            <status>Eligible</status>\n" +

```

```

        "        <message>Le patient avec le numéro " + phoneNumber + " est
éligible à la prise en charge.</message>\n" +

        "    </web:CheckEligibilityResponse>\n" +

        " </soapenv:Body>\n" +

        "</soapenv:Envelope>";

    }

}

```

4. Création de l'Endpoint REST

L'API REST sera responsable de la gestion des requêtes des applications tierces et de l'interaction avec le service SOAP.

4.1 Création du contrôleur REST PatientController

java

CopierModifier

```
package com.cmu.bridge.controller;
```

```
import com.cmu.bridge.service.SOAPService;
```

```
import org.springframework.beans.factory.annotation.Autowired;
```

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

```
@RestController
```

```
@RequestMapping("/api/patient")
```

```
public class PatientController {
```

```
    @Autowired
```

```
    private SOAPService soapService;
```

```

@GetMapping("/eligibility")

public String checkEligibility(@RequestParam String phoneNumber) {

    try {

        // Appel du service SOAP pour vérifier l'éligibilité

        return soapService.checkEligibility(phoneNumber);

    } catch (Exception e) {

        return "Erreur lors de la vérification de l'éligibilité : " + e.getMessage();

    }

}
}

```

5. Transformation de la réponse SOAP en JSON

Nous allons transformer la réponse SOAP (XML) en une réponse JSON REST pour les applications frontend. Cela sera réalisé dans le service SOAP avec une conversion appropriée de l'XML en JSON.

5.1 Exemple de réponse JSON

Lorsque le service SOAP renverra une réponse, vous devrez la transformer en un format JSON. Par exemple :

```

{

    "status": "Eligible",

    "message": "Le patient avec le numéro 1234567890 est éligible à la prise en charge."

}

```

Configuration de la base de données H2

Dans **application.properties**, ajoutez les configurations suivantes :

properties

CopierModifier

```
spring.datasource.url=jdbc:h2:mem:testdb
```

```
spring.datasource.driverClassName=org.h2.Driver
```

```
spring.datasource.username=sa
```

```
spring.datasource.password=password
```

```
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
```

```
spring.jpa.hibernate.ddl-auto=update
```

```
spring.h2.console.enabled=true
```

6.2 Création de l'entité Patient

java

CopierModifier

```
package com.cmu.bridge.model;
```

```
import javax.persistence.Entity;
```

```
import javax.persistence.Id;
```

```
@Entity
```

```
public class Patient {
```

```
    @Id
```

```
    private String phoneNumber;
```

```
    private boolean eligible;
```

```
    // Getters et Setters
```



```
}
```

6.3 Création du repository JPA

```
java
```

```
CopierModifier
```

```
package com.cmu.bridge.repository;
```

```
import com.cmu.bridge.model.Patient;
```

```
import org.springframework.data.jpa.repository.JpaRepository;
```

```
public interface PatientRepository extends JpaRepository<Patient, String> {
```

```
    Patient findByPhoneNumber(String phoneNumber);
```

```
}
```

7. Conclusion

Avec ce code, vous avez un connecteur **REST** qui permet à des applications tierces de vérifier l'éligibilité d'un patient pour la prise en charge en interrogeant un service **SOAP** et en renvoyant une réponse **JSON**.

Les points importants du connecteur :

- **Endpoint REST** pour l'interaction avec des applications tierces.
- **Service SOAP** pour consulter l'API SOAP de l'agence de la CMU.
- **Transformation SOAP vers JSON** pour le frontend.
- **Base de données H2** pour simuler l'API CMU et vérifier l'éligibilité.

N'hésitez pas à me demander si vous avez besoin d'explications supplémentaires ou si vous

Demander à ChatGPT

Vous avez dit :

code sources

ChatGPT a dit :

Voici l'ensemble du code source pour le projet de connecteur **REST SOAP Bridge** que vous pouvez utiliser pour interagir avec l'API SOAP de l'agence de la CMU. Ce code comprend :

- L'**API REST** pour interroger les droits de prise en charge d'un patient.
- La **simulation d'une API SOAP** pour vérifier l'éligibilité du patient.
- La transformation de la **réponse SOAP en JSON** pour le frontend.
- Une **base de données H2** pour simuler un enregistrement des patients.

1. pom.xml - Dépendances Maven

xml

CopierModifier

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">

    <modelVersion>4.0.0</modelVersion>

    <groupId>com.cmu.bridge</groupId>
    <artifactId>cmu-bridge-rest</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>jar</packaging>

    <properties>
        <java.version>11</java.version>
    </properties>

    <dependencies>
```

<!-- Spring Boot Web for REST API -->

<dependency>

 <groupId>org.springframework.boot</groupId>

 <artifactId>spring-boot-starter-web</artifactId>

</dependency>

<!-- Spring Boot DevTools for easier development -->

<dependency>

 <groupId>org.springframework.boot</groupId>

 <artifactId>spring-boot-devtools</artifactId>

 <scope>runtime</scope>

</dependency>

<!-- Spring Web Services for SOAP -->

<dependency>

 <groupId>org.springframework.ws</groupId>

 <artifactId>spring-webservices</artifactId>

 <version>4.0.0</version>

</dependency>

<!-- H2 Database (in-memory) for simulating database -->

<dependency>

 <groupId>com.h2database</groupId>

 <artifactId>h2</artifactId>

```
        <scope>runtime</scope>
    </dependency>

    <!-- Spring Data JPA for database access -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>

    <!-- Spring Boot Starter Test (for unit testing) -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>

<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
```

</build>

</project>

2. application.properties - Configuration Spring Boot

properties

CopierModifier

Configuration de la base de données H2

spring.datasource.url=jdbc:h2:mem:testdb

spring.datasource.driverClassName=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=password

spring.jpa.database-platform=org.hibernate.dialect.H2Dialect

spring.jpa.hibernate.ddl-auto=update

spring.h2.console.enabled=true

Port de l'application REST

server.port=8080

3. Classe principale de l'application Spring Boot - Application.java

java

CopierModifier

package com.cmu.bridge.application;

import org.springframework.boot.SpringApplication;

```
import org.springframework.boot.autoconfigure.SpringBootApplication;
```

```
@SpringBootApplication
```

```
public class Application {
```

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

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

```
    }
```

```
}
```

4. Entité Patient - Patient.java

```
java
```

```
CopierModifier
```

```
package com.cmu.bridge.model;
```

```
import javax.persistence.Entity;
```

```
import javax.persistence.Id;
```

```
@Entity
```

```
public class Patient {
```

```
    @Id
```

```
    private String phoneNumber;
```

```
    private boolean eligible;
```

```
public String getPhoneNumber() {  
    return phoneNumber;  
}
```

```
public void setPhoneNumber(String phoneNumber) {  
    this.phoneNumber = phoneNumber;  
}
```

```
public boolean isEligible() {  
    return eligible;  
}
```

```
public void setEligible(boolean eligible) {  
    this.eligible = eligible;  
}  
}
```

5. Repository Patient - PatientRepository.java

java

CopierModifier

```
package com.cmu.bridge.repository;
```

```
import com.cmu.bridge.model.Patient;
```

```
import org.springframework.data.jpa.repository.JpaRepository;
```

```
public interface PatientRepository extends JpaRepository<Patient, String> {  
    Patient findByPhoneNumber(String phoneNumber);  
}
```

6. Service SOAP - SOAPService.java

java

CopierModifier

```
package com.cmu.bridge.service;
```

```
import org.springframework.beans.factory.annotation.Autowired;
```

```
import org.springframework.stereotype.Service;
```

```
import com.cmu.bridge.repository.PatientRepository;
```

```
@Service
```

```
public class SOAPService {
```

```
    @Autowired
```

```
    private PatientRepository patientRepository;
```

```
    public String checkEligibility(String phoneNumber) {
```

```
        // Vérifier dans la base de données (H2)
```

```
        Patient patient = patientRepository.findByPhoneNumber(phoneNumber);
```



```

    if (patient != null) {

        // Simulation de la réponse SOAP basée sur les données du patient

        return simulateSOAPResponse(patient);

    } else {

        return "<soapenv:Envelope
xmlns:soapenv=\"http://schemas.xmlsoap.org/soap/envelope/\"
xmlns:web=\"http://webservice.cmu.com/\">\n" +

            "  <soapenv:Header/>\n" +

            "  <soapenv:Body>\n" +

            "    <web:CheckEligibilityResponse>\n" +

            "      <status>NonEligible</status>\n" +

            "      <message>Patient non trouvé.</message>\n" +

            "    </web:CheckEligibilityResponse>\n" +

            "  </soapenv:Body>\n" +

            "</soapenv:Envelope>";

    }

}

```

```

private String simulateSOAPResponse(Patient patient) {

    String status = patient.isEligible() ? "Eligible" : "NonEligible";

    String message = patient.isEligible() ?

        "Le patient avec le numéro " + patient.getPhoneNumber() + " est
éligible." :

        "Le patient avec le numéro " + patient.getPhoneNumber() + " n'est
pas éligible.";
}

```

```

        return "<soapenv:Envelope
xmlns:soapenv=\"http://schemas.xmlsoap.org/soap/envelope/\"
xmlns:web=\"http://webservice.cmu.com/\">\n" +

        "    <soapenv:Header/>\n" +

        "    <soapenv:Body>\n" +

        "        <web:CheckEligibilityResponse>\n" +

        "            <status>" + status + "</status>\n" +

        "            <message>" + message + "</message>\n" +

        "        </web:CheckEligibilityResponse>\n" +

        "    </soapenv:Body>\n" +

        "</soapenv:Envelope>";

    }

}

```

7. Contrôleur REST - PatientController.java

java

CopierModifier

```
package com.cmu.bridge.controller;
```

```
import com.cmu.bridge.service.SOAPService;
```

```
import org.springframework.beans.factory.annotation.Autowired;
```

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

```
@RestController
```

```

@RequestMapping("/api/patient")

public class PatientController {

    @Autowired

    private SOAPService soapService;

    @GetMapping("/eligibility")

    public String checkEligibility(@RequestParam String phoneNumber) {

        try {

            // Appel du service SOAP pour vérifier l'éligibilité

            return soapService.checkEligibility(phoneNumber);

        } catch (Exception e) {

            return "Erreur lors de la vérification de l'éligibilité : " + e.getMessage();

        }

    }

}

```

8. Base de données H2 - Initialisation avec des données

J'ai initialiser la base de données H2 en créant un fichier de script dans src/main/resources/data.sql pour peupler la base avec quelques données de test.

sql

CopierModifier

```
INSERT INTO patient (phone_number, eligible) VALUES ('0123456789', true);
```

```
INSERT INTO patient (phone_number, eligible) VALUES ('0987654321', false);
```

9. Démarrer l'application

- **Lancer l'application** : Dans votre terminal, à la racine du projet, exécutez la commande suivante pour démarrer l'application Spring Boot :

```
bash
```

```
mvn spring-boot:run
```

- **Accéder à l'API REST** : Une fois l'application lancée, vous pouvez interroger l'API à l'endpoint `http://localhost:8080/api/patient/eligibility?phoneNumber=0123456789`.

10. Exemple de réponse JSON

Si un patient est éligible, la réponse sera :

```
json
```

```
{  
  "status": "Eligible",  
  "message": "Le patient avec le numéro 0123456789 est éligible."  
}
```

Si le patient n'est pas trouvé ou non éligible :

```
json
```

```
{  
  "status": "NonEligible",  
  "message": "Le patient avec le numéro 0987654321 n'est pas éligible."  
}
```

