INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

# Rapport Architectures Orientées Services

LEMMERS
Nathan
OSORNIO
Patrick
5-SDBD
2023-2024

# Rapport Architectures Orientées Services

Code source disponible sur : https://github.com/posornio/SOA-5SDBD

# 1) SOA vs Rest vs MS Spring Boot

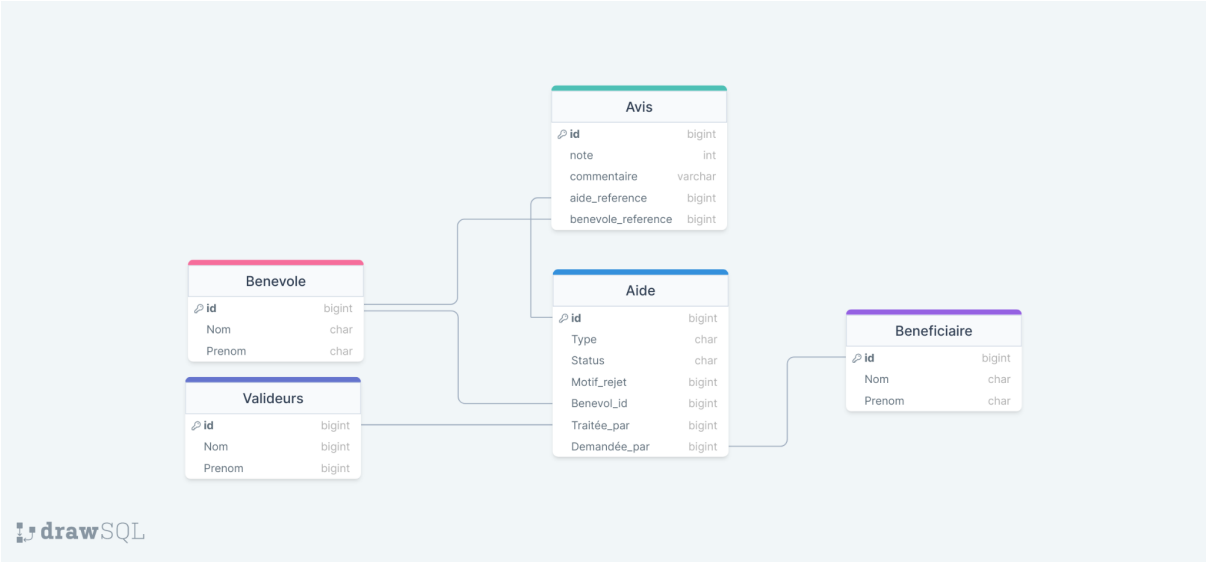
Pour nous familiariser avec le monde des services web, nous avons suivi les tutoriels de SOAP et de Rest.
Cependant, il ne nous est pas paru pertinent de déployer des endpoint de notre application en utilisant ces technologies car on savait dès le début que nous utiliserions le framework Spring Boot et nous avons décidé de rester homogènes dans notre infrastructure.

Par contre, nous avons beaucoup appris en suivant ces tutoriels.
Nous avons choisi d'utiliser la technologie Rest car elle est plus récente. L'utilisation de Spring Boot rend le processus beaucoup plus facile et permet de créer plusieurs micro services rapidement. On a donc décidé d'utiliser une architecture de type micro services pour travailler sur eux sans avoir besoin de redéployer toute l'application dans le cas de mise à jour d'un composant. Chaque micro service peut gérer une table de la base de données et nous pouvons les déployer indépendamment.
Pour finir, nous avons décidé d'utiliser le format JSON au lieu de XML pour tester plus efficacement et utiliser JAVA.

# 2) Présentation de la base de données

La première étape de la conception de cette application est de définir la base de données.
Pour cela, nous distinguons les bénévoles, les bénéficiaires et les valideurs, pour les placer sur 3 tables différentes. Ces tables ont 3 colonnes : Nom, Prénom et ID. La clé primaire sera toujours l'ID. Cette séparation nous sera utile lors de l'utilisation des clés étrangères.
La table Aide est un peu plus complexe, la clé primaire est toujours un *id*, et chaque aide à un *type*, un *bénéficiaire* qui demande l'aide et un *statut* : en attente, validée ou refusée. Si elle est refusée, le validateur devra donner un motif. De plus, une aide ne peut être attribuée qu'à un bénévole.
Les colonnes *benevol_id*, *traite_par* et *demandee_par* sont donc des clés étrangères qui font référence à leur table d'origine.
La table Avis a 5 colonnes, *id, note, commentaire, aide_reference* et *benevol_reference*. *aide_reference* et *benevol_reference* sont des clés étrangères qui font référence à leur table d'origine.
Ceci est la structure de notre base de données :

Fig 1: Structure de notre base de données




# 3)Structure de nos micro services

Voici nos 5 micro-services essentiels : ***BenevolManager***, ***BeneficiaireManager***, ***AideManager, AvisManager*** et ***ValideurManager***.
Chaque micro service a son contrôleur qui contient toutes les méthodes pour modifier sa table respective dans la base de données.
Chaque micro service à 3 paquets, le paquet contenant l'application, le paquet du contrôleur et un paquet Model qui contient les classes de référence telles que Aide, Bénéficiaire, etc.
Le contrôleur communique avec l'application en utilisant la notation @Autowired pour partager le RestTemplate et l'objet de Connection à la base de données. Ces valeurs sont initialisées sur la classe Application et sont annotées en tant que @Bean.

Par la suite, nous avons implémenté un micro service de découverte : *Eureka*, qui permet au micro services d'être visibles entre eux et de s'appeler si nécessaire.

**Instances currently registered with Eureka**

| Application | AMIs | Availability Zones | Status |
|---|---|---|---|
| AIDEMANAGERSERVICE | n/a (1) | (1) | UP (1) - 10.32.43.203:AideManagerService:8092 |
| AVISMANAGERSERVICE | n/a (1) | (1) | UP (1) - 10.32.43.203:AvisManagerService:8095 |
| BENEFICIAIREMANAGERSERVICE | n/a (1) | (1) | UP (1) - 10.32.43.203:BeneficiaireManagerService:8093 |
| BENEVOLMANAGERSERVICE | n/a (1) | (1) | UP (1) - 10.32.43.203:BenevolManagerService:8090 |
| VALIDEURMANAGERSERVICE | n/a (1) | (1) | UP (1) - 10.32.43.203:ValideurManagerService:8094 |

Fig 2. Nos micro services visibles sur Eureka

Maintenant, les requêtes sont redirigées par *Eureka* car nous avons ajouté l'annotation @LoadBalanced dans nos fichiers Applications. Si jamais nos micro services changent de ports ou sont présents dans plusieurs zones de disponibilités, nous pouvons les joindre en utilisant le nom affiché sur *Eureka*, sans avoir besoin de savoir leurs adresses ou leurs

ports, et sans risquer de casser le micro service si jamais un autre change d'adresse ou de port.

Grâce à cette connexion via Eureka, les micro-services Avis et Aides peuvent échanger des informations même si le host ou le port change.

# 4) Connexion à la base de données

Pour éviter de mettre les identifiants de la base de données dans le code, et pour permettre de changer l'utilisateur ou le mot de passe sans avoir besoin de modifier le code, nous avons implémenté un micro service de *Config Server* qui se connecte au fichier client-service (ou client-service-dev si jamais on a besoin d'utiliser ce profil). Ce fichier se situe sur Github.
Ensuite, nous avons implémenté un micro service *Config Client* qui récupère les informations téléchargées par le *Config Server* et publie les informations dont on a besoin.
Nous avons décidé de créer une classe *DbInfo* contenant l'url de la base de données, l'utilisateur et le mot de passe.
Chaque micro service de type manager se connecte à ce client et récupère un objet de type DbInfo, pour se connecter ensuite à la base de données.
Nous avons donc mis une classe *DbInfo* sur chacun des paquets modèles de nos micro services.
Nous avons choisi de ne pas montrer le micro service *Client Config* ou *Config Server* sur *Eureka* pour limiter leur visibilité et restreindre l'exposition de ces données sensibles.
Nos fichiers d'applications de type manager sont très similaires :

- Définition du *RestTemplate*.
- Mise en place de l'objet de connexion à la base de données.

```
@SpringBootApplication
public class BenevolManagerApplication {
    posornio +1 *
    @Bean
    public Connection dbinit() throws Exception {
        DbInfo dbInfo = this.restTemplate().getForObject( url: "http://localhost:8087/dbInfo", DbInfo.class);
        Class.forName( className: "com.mysql.jdbc.Driver");
        System.out.println(dbInfo);
        return DriverManager.getConnection(dbInfo.getUrl(), dbInfo.getUser(), dbInfo.getPassword());
    }
    1 usage   osorniog +1
    @Bean
    @LoadBalanced
    public RestTemplate restTemplate() {
        return new RestTemplate();
    }
    osorniog +1 *
    public static void main(String[] args) {
        SpringApplication.run(BenevolManagerApplication.class, args);
    }
}
```

Fig 3. Squelette de nos fichier Application (ex: BenevolManagerApplication)

# 5) Méthodes dans nos micro-services

Nos micro services de gestion de bénévoles, bénéficiaires et valideurs se ressemblent beaucoup.
Ils nous permettent de récupérer les informations d'un utilisateur en utilisant leur id, de récupérer tous les utilisateurs et de récupérer le nombre d'utilisateurs sur une table.
Il est aussi possible d'ajouter un utilisateur ou de le supprimer via son id.
Les requêtes où on demande une information sont annotées avec @GetMapping pour indiquer une requête de type GET. Les paramètres de la méthode sont attribués dans l'url de la requête grâce à l'annotation @PathVariable.
Pour ajouter un utilisateur, on utilise l'annotation @PostMapping, cela indique une requête de type POST. Ces requêtes utilisent l'annotation @RequestBody et on passe un objet en paramètre. Cet objet peut être un bénévole, bénéficiaire, valideur où une aide, mais le constructeur associé doit exister dans le paquet Model. Pour envoyer l'objet on doit le créer avant dans le code qui fait appel à notre endpoint.

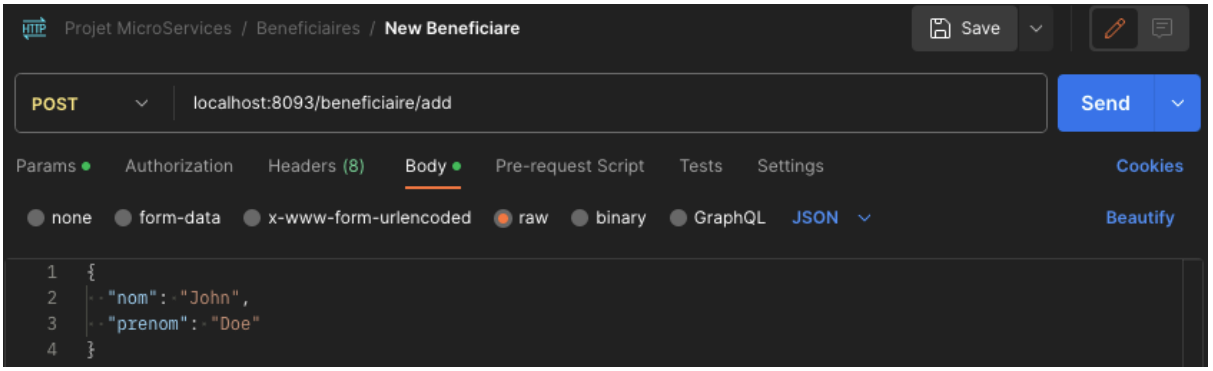



Fig 4. Exemple de création d'un bénéficiaire avec PostMan.

Le micro service gestionnaire des aides est plus complexe. Il a les mêmes fonctionnalités que les autres mais il nous permet aussi de valider une aide, de rejeter une aide et d'attribuer un bénévole à une aide. Ces méthodes sont aussi des requêtes de type POST.
Pour valider, rejeter ou attribuer une aide, on doit spécifier qui la valide. Pour la rejeter il faut aussi donner un motif, et lorsqu'on attribue l'aide, il est nécessaire de dire à qui l'aide est attribuée
Par la suite, nous avons implémenté un micro service pour '*update*', une rangée de la table Aide. Celui-ci utilise l'annotation @PatchMapping, et prend deux paramètres, une @PathVariable(value = 'id') qui correspond à la rangée à modifier et un @RequestBody Aide qui correspond au nouvel objet à mettre sur la base de données.
Il nous est également paru pertinent de publier les endpoint *getAidesByBenevol*, *getAidesByBeneficiare*, *getAideByValideur*, *getAideByStatus* et *getAideByType*. Ces endpoints sont de type get et nous seront très utiles pour la prochaine partie.
Nous avons décidé d'ajouter un objet Avis à chaque objet Aide vu que chaque Aide peut avoir un seul avis. Ce choix nous a facilité le développement de notre application front-end. Ceci a été facilité par la présence du server Eureka.

```
Avis avis = restTemplate.getForObject(avisUrl, Avis.class);
```

De plus, lorsqu'une aide est supprimé, son avis sera supprimé en cascade.

```
restTemplate.delete("http://AvisManagerService/avis/delete/"+id);
```

# 6) Testing de nos micro-services

On a testé la totalité de nos micro-services en utilisant Postman, cet outil est très intuitif et puissant.
Vous pouvez trouver la totalité de nos tests sous le fichier Projet MicroServices.postman_collection.json présent sur notre dépôt GitHub.
Cet outil nous à permis de vérifier la présence de nos micro-services, puisque nous avons eu souvent des erreurs 404, et de vérifier que nos données s'enregistrent correctement, tout en validant les performances temporelles.

Pour lancer l'application, il est nécessaire d'importer les dossiers : *AideManager*, *BenevolManager*, *BeneficiareManager*, *ConfigClient*, *ConfigServer*, *msDiscovery* et *ValideurManager* comme des projets Maven et de les lancer dans l'ordre suivant:

- msDiscovery
- ConfigServer
- ConfigClient
- BenevolManager,BeneficiaireManager, AideManager et ValideurManager.

# 7) Interface Graphique

Nous avons décidé de créer une interface graphique pour accompagner le projet et simuler le fonctionnement de l'application. Cette interface graphique à été développée en utilisant le framework Angular et nous permet d'exploiter toute la puissance de notre application intuitivement et sans avoir besoin de maîtriser des technologies tels que Postman.
Pour pouvoir exploiter nos micro-services, nous avons ajouté une politique CORS (Cross-origin resource sharing) qui permet de définir depuis quel site nos micro-services peuvent être appelés. Puisque nous n'avons pas de domaine, nous avons whitelisté tous les sites avec l'annotation @CrossOrigin(origins = "*") sous nos contrôleurs.

```java
@RestController
@CrossOrigin(origins = "*")
@RequestMapping("/valideurs")
public class ValideurController {
    5 usages
    @Autowired
    Connection connection = null;

    @Autowired
    private RestTemplate restTemplate;

    osorniog
    @GetMapping("/nbOf")
    public int getNumberValideurs() {
        String sql = "SELECT COUNT(*) FROM Valideurs";
        try (Statement stmt = connection.createStatement()) {
            ResultSet rs = stmt.executeQuery(sql);
            if (rs.next()) {
                int nbOf = rs.getInt( columnIndex: 1);
                return nbOf;
            }
            return 0;
        } catch (SQLException e) {
            System.out.println(e.getMessage());
            return 0;
```

Fig 5. Exemple d'un de nos contrôleurs

Le code du front-end est disponible sous le dossier SOA-GUI. Pour lancer cette application il est nécessaire d'avoir installé NodeJS et npm. Puis on se place sous le dossier SOA-GUI, et sur un terminal on tape:

- npm install -g @angular/cli
- npm install
- npm start

Notre application est formée de 5 onglets, l'onglet *Bénévoles*, *Bénéficiaires*, *Valideurs*, *Aides* et *Avis*.

Dans les onglets on trouvera une table avec la totalité des personnes ou aides et les informations pertinentes. On peut ajouter des classes avec un bouton qui révèle des inputs pour donner les informations sur la classe à ajouter.

En cliquant sur une personne, un bouton apparaît pour supprimer cette personne de la base de données. Il apparaît aussi une table avec toutes les aides où cette personne à été mentionnée.

Si cette personne est un bénévole, ces avis vont apparaître pour faire un résumé de ces bénévolats.

Si c'est un bénéficiaire, ces aides demandées apparaîtront.

# Quel table analyser?

◉ Benevoles ◯ Beneficiaires ◯ Valideurs ◯ Aides ◯ Avis

Benevoles

Ajouter un Benevole

| ID | Nom | Prénom |
|---|---|---|
| 1 | Osornio | Patrick |
| 3 | Lemmers | Nathan |
| 4 | Leo | Messi |

Fig 6. Interface de bienvenue

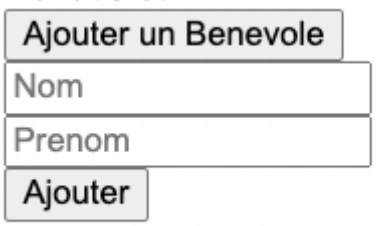



Fig. 7 Ajout d'un bénévole

## Quel table analyser?

◉ Benevoles ◯ Beneficiaires ◯ Valideurs ◯ Aides ◯ Avis

Benevoles

Ajouter un Benevole | Supprimer un benevole

| ID | Nom | Prénom |
|---|---|---|
| 1 | Osornio | Patrick |
| 3 | Lemmers | Nathan |
| 4 | Leo | Messi |

## Aides affectés à ce Benevol

Ajouter une Aide

| ID | Beneficiaire | Status | Type | Motif de rejet | Bénevole associé | Traitée par |
|---|---|---|---|---|---|---|
| 3 | Clooney George | Valide | Financiere | | Osornio Patrick | Karim Benzema |
| 8 | Brad Pitt | | Mobilité | | Osornio Patrick | Karim Benzema |

## Avis associés à ce Benevol

| ID | Note / 5 | Commentaire | Aide | Bénévole associé |
|---|---|---|---|---|
| 0 | 4 | Tres bien mais peut faire mieux | 3 | Osornio Patrick |

Fig. 8 Infos sur un bénévol

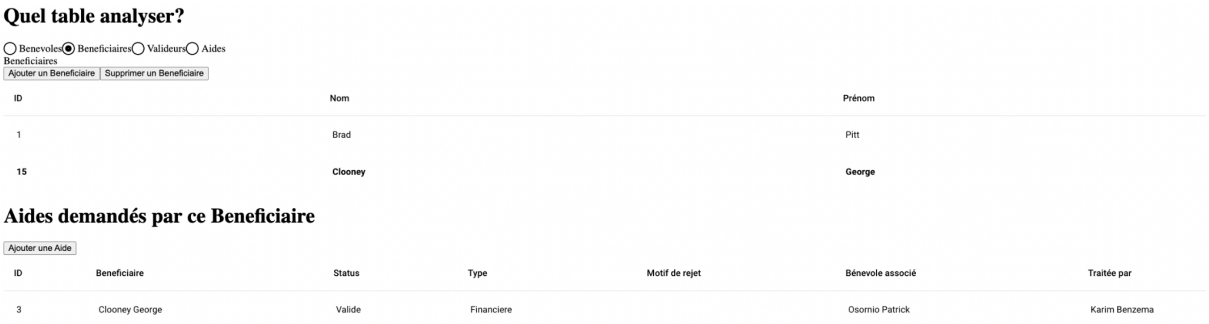



Fig. 9 Aides demandés par George Clooney

La table Aide est légèrement différente, on peut ajouter ou supprimer une aide facilement, mais on peut aussi la valider ou la rejeter en donnant le id du valideur et un motif de rejet, ou encore attribuer une aide à un bénévole. On peut aussi noter l'aide directement sur cet onglet en cliquant sur l'aide à noter.

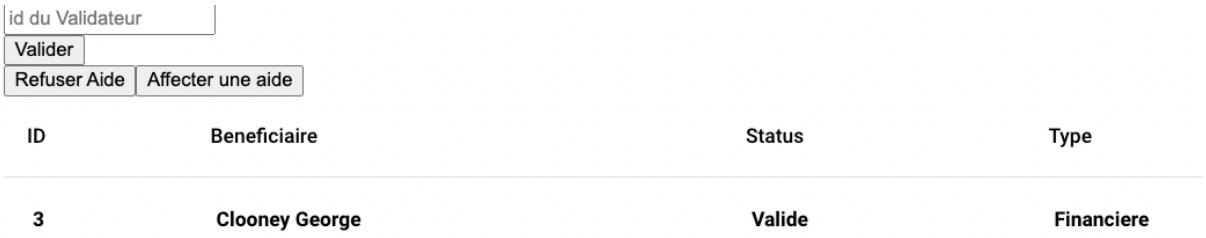



Fig. 10 Menu pour valider une aide


Fig. 11 Menu pour refuser une aide


Fig. 12 Menu pour affecter une aide à un bénévole


Fig. 13 Menu pour noter une aide

## Quel table analyser?

◯ Benevoles ◯ Beneficiaires ◯ Valideurs ◉ Aides ◯ Avis

Aides

Ajouter une Aide | Supprimer une aide | Valider Aide | Refuser Aide | Affecter une aide | Noter Aide

| ID | Beneficiaire | Status | Type | Motif de rejet | Bénevole associé | Traitée par |
|---|---|---|---|---|---|---|
| **3** | **Clooney George** | **Valide** | **Financiere** | | **Osornio Patrick** | **Karim Benzema** |
| 8 | Brad Pitt | | Mobilité | | Osornio Patrick | Karim Benzema |

## Avis associés à cet aide

**Note : 4**

**Commentaire : Tres bien mais peut faire mieux**

Fig. 14 Infos sur une aide

## Quel table analyser?

◯ Benevoles ◯ Beneficiaires ◯ Valideurs ◯ Aides ◉ Avis

Avis

Supprimer un avis

| ID | Note / 5 | Commentaire | Aide | Bénévole associé |
|---|---|---|---|---|
| **0** | **4** | **Tres bien mais peut faire mieux** | **3** | **Osornio Patrick** |

## Aide associés à cet avis

**Type : Valide**

**Statut : Financiere**

**Benevole associé : Osornio Patrick**

**Beneficiaire associé : Clooney George**

Fig. 15 Infos sur un avis