

Atelier-Création de Microservice

Gestion des candidats

Objectifs

Notre objectif est de créer un Microservice qui fournit des exemples d'opérations.

Développement des micro-services

Nous visons à créer un premier microservice de gestion des candidats dans une entreprise.

Soit l'entité suivante représentant un candidat:

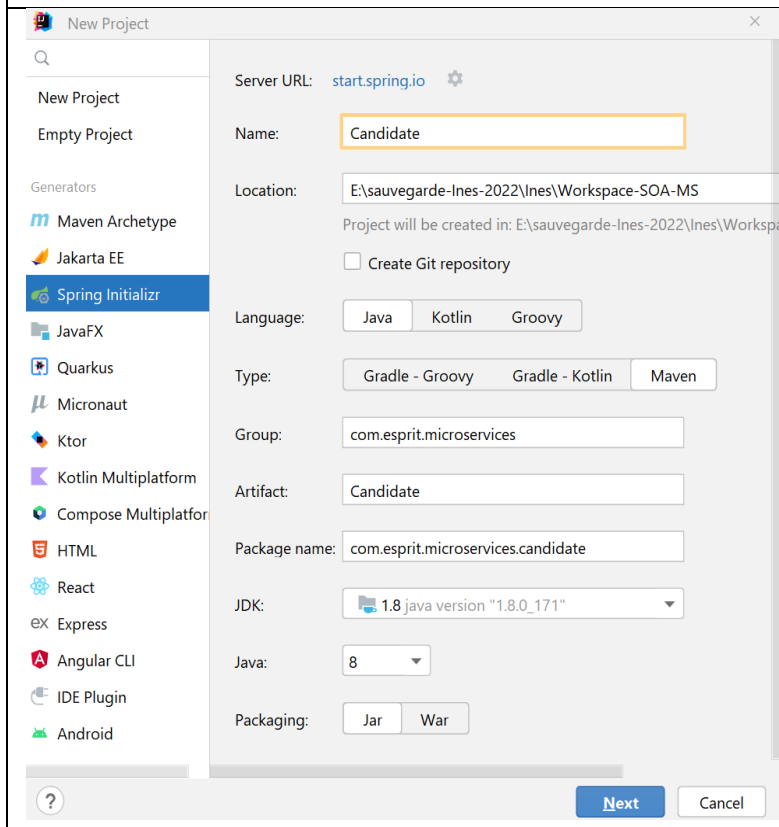
Candidat
-ID: Integer -nom:String -prenom:String - email: String

Le microservice Gestion candidats va se charger des fonctionnalités suivantes :

- Afficher tous les candidats
- Afficher un candidat par son id ou son nom
- Ajouter/modifier/supprimer un candidat

1. Créez un projet de type Spring boot selon l'IDE choisi, Eclipse ou IntelliJ :

IntelliJ



New Project

Server URL: start.spring.io

Name:

Location:

Project will be created in: E:\sauvegarde-Ines-2022\Ines\Workspace-SOA-MS

☐ Create Git repository

Language: ☐ Java ☐ Kotlin ☐ Groovy

Type: ☐ Gradle - Groovy ☐ Gradle - Kotlin ☐ Maven

Group:

Artifact:

Package name:

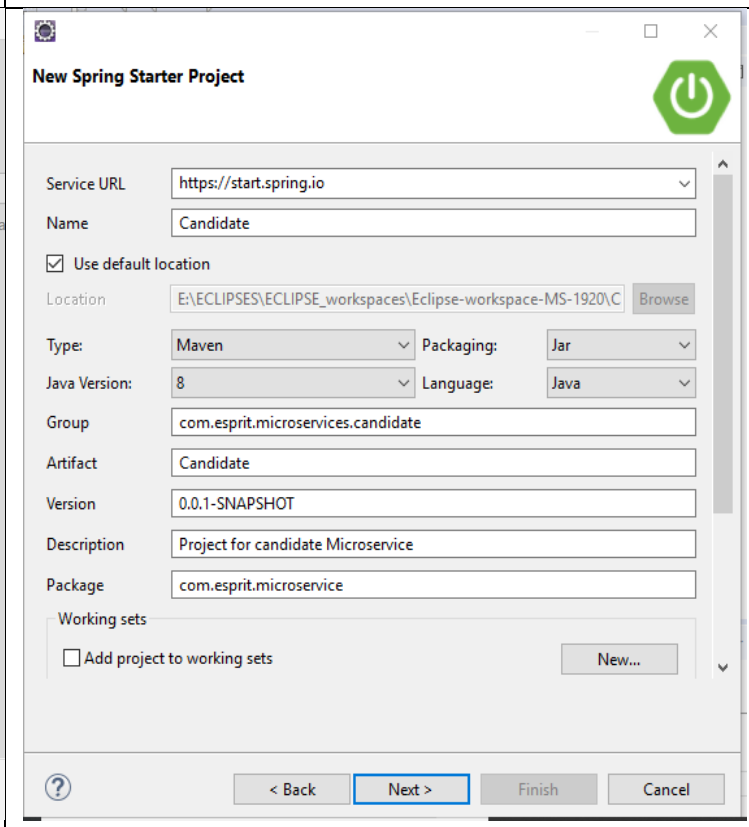
JDK:

Java:

Packaging: ☐ Jar ☐ War

[Next](#) [Cancel](#)

Eclipse



New Spring Starter Project

Service URL:

Name:

☒ Use default location

Location: [Browse](#)

Type: Packaging:

Java Version: Language:

Group:

Artifact:

Version:

Description:

Package:

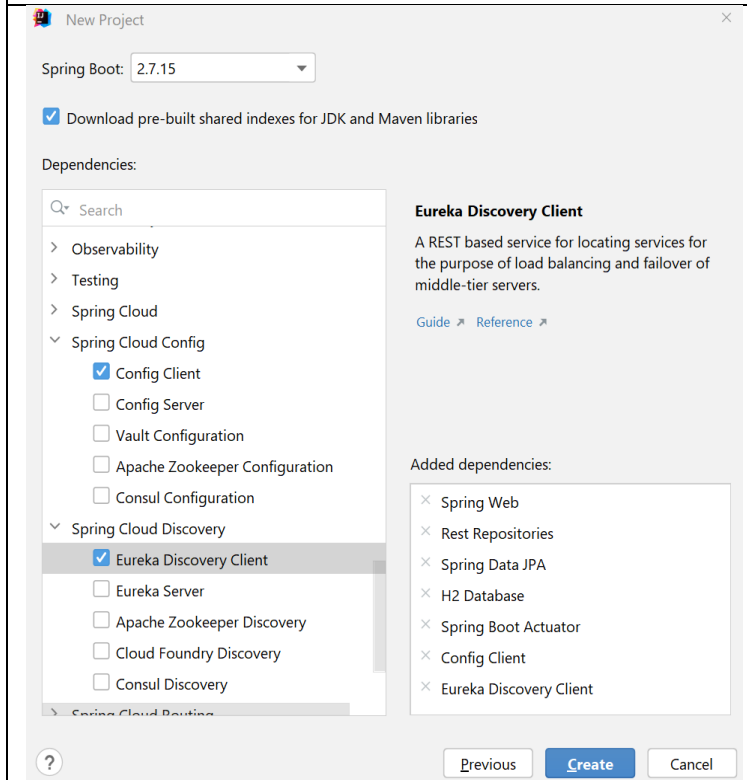
Working sets

☐ Add project to working sets [New...](#)

[Back](#) [Next >](#) [Finish](#) [Cancel](#)

2. Ajouter les dépendances nécessaires :

IntelliJ



New Project

Spring Boot:

☒ Download pre-built shared indexes for JDK and Maven libraries

Dependencies:

☒ Spring Cloud Config

☒ Config Client

☐ Config Server

☐ Vault Configuration

☐ Apache Zookeeper Configuration

☐ Consul Configuration

☒ Spring Cloud Discovery

☒ Eureka Discovery Client

☐ Eureka Server

☐ Apache Zookeeper Discovery

☐ Cloud Foundry Discovery

☐ Consul Discovery

Eureka Discovery Client

A REST based service for locating services for the purpose of load balancing and failover of middle-tier servers.

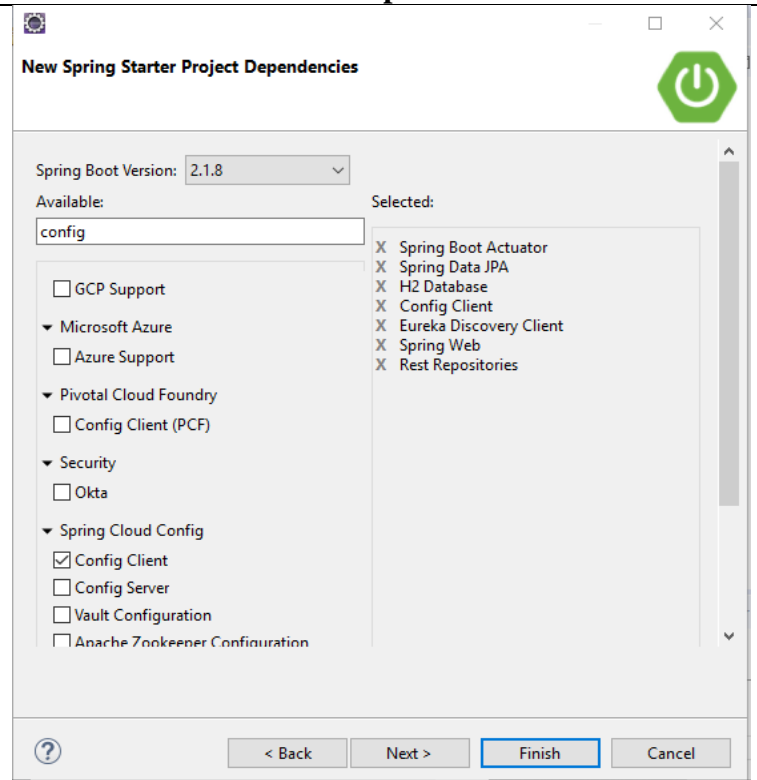
[Guide](#) [Reference](#)

Added dependencies:

- Spring Web
- Rest Repositories
- Spring Data JPA
- H2 Database
- Spring Boot Actuator
- Config Client
- Eureka Discovery Client

[Previous](#) [Create](#) [Cancel](#)

Eclipse



New Spring Starter Project Dependencies

Spring Boot Version:

Available:

☐ GCP Support

☐ Microsoft Azure

☐ Azure Support

☐ Pivotal Cloud Foundry

☐ Config Client (PCF)

☐ Security

☐ Okta

☒ Spring Cloud Config

☒ Config Client

☐ Config Server

☐ Vault Configuration

☐ Apache Zookeeper Configuration

Selected:

- Spring Boot Actuator
- Spring Data JPA
- H2 Database
- Config Client
- Eureka Discovery Client
- Spring Web
- Rest Repositories

[Back](#) [Next >](#) [Finish](#) [Cancel](#)

3. Créez le contrôleur REST de notre microservice qui est représenté par une classe nommée par exemple "CandidatRestAPI".

```
package com.esprit.ms.candidate;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class CandidatRestAPI {
    2 usages
    private String title="Hello, i'm the candidate Micro Service";

    @RequestMapping("/hello")
    public String sayHello(){
        System.out.println(title);
        return title;
    }
}
```

4. Créez l'entité "Candidat" :

```
@Entity
public class Candidat implements Serializable{
    private static final long serialVersionUID = 6

    @Id
    @GeneratedValue
    private int id;
    private String nom, prenom,email;
    public int getId() {
        return id;
    }
    public String getNom() {
        return nom;
    }
    public void setNom(String nom) {
        this.nom = nom;
    }
    public String getPrenom() {
        return prenom;
    }
    public void setPrenom(String prenom) {
        this.prenom = prenom;
    }
    public String getEmail() {
        return email;
    }
    public void setEmail(String email) {
        this.email = email;
    }
    public Candidat() {
        super();
        // TODO Auto-generated constructor stub
    }
    public Candidat(String nom) {
        super();
        this.nom = nom;
    }
}
```

5. Créez le Repository lié à l'entité Candidat :

```
public interface CandidatRepository extends JpaRepository<Candidat , Integer> {
}
}
```

6. Créez une fonction dans la classe CandidateApplication (créée par défaut lors de la création du projet) qui va permettre d'insérer automatiquement des candidats dans la base

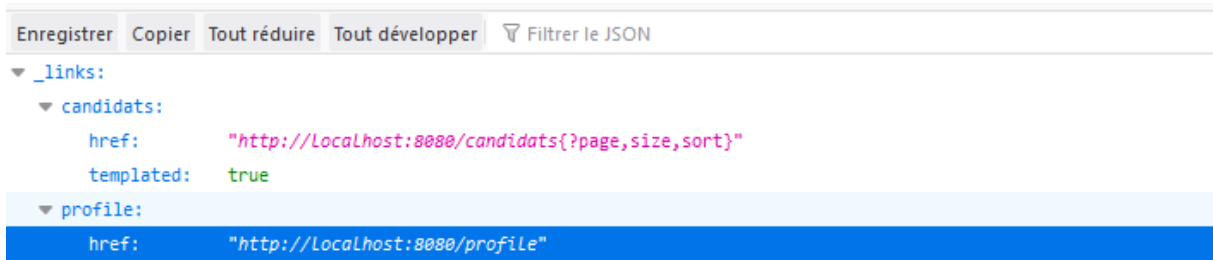
```
@SpringBootApplication
public class CandidateApplication {
    public static void main(String[] args) {
        SpringApplication.run(CandidateApplication.class, args);
    }
    4 usages
    @Autowired
    private CandidatRepository repository;

    @Bean
    ApplicationRunner init() {
        return (args) -> {
            // save
            repository.save(new Candidat( nom: "Mariem", prenom: "Ch", email: "ma@esprit.tn"));
            repository.save(new Candidat( nom: "Sarrah", prenom: "ab", email: "sa@esprit.tn"));
            repository.save(new Candidat( nom: "Mohamed", prenom: "ba", email: "mo@esprit.tn"));
            // fetch
            repository.findAll().forEach(System.out::println);
        };
    }
}
```

7. Exécuter la classe CandidateApplication :

IntelliJ	Eclipse

8. Accéder à l'adresse <http://localhost:8080/>



Vous pouvez accéder à l'adresse <http://localhost:8080/candidates> pour voir les détails de tous les candidats.

Vous pouvez également faire une recherche de candidats par id : <http://localhost:8080/candidates/3>

9. Pour ajouter par exemple une requête de recherche d'un candidat par son nom, on modifie ainsi la classe CandidatRepository:

```
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;

public interface CandidatRepository extends JpaRepository<Candidat, Integer> {

    @Query("select c from Candidat c where c.nom like :name")
    public Page<Candidat> candidatByNom(@Param("name") String n, Pageable pageable);
}
```

Lancez l'URL <http://localhost:8080/candidates/search/candidatByNom?name=Sarra> pour faire la recherche d'un candidat par nom



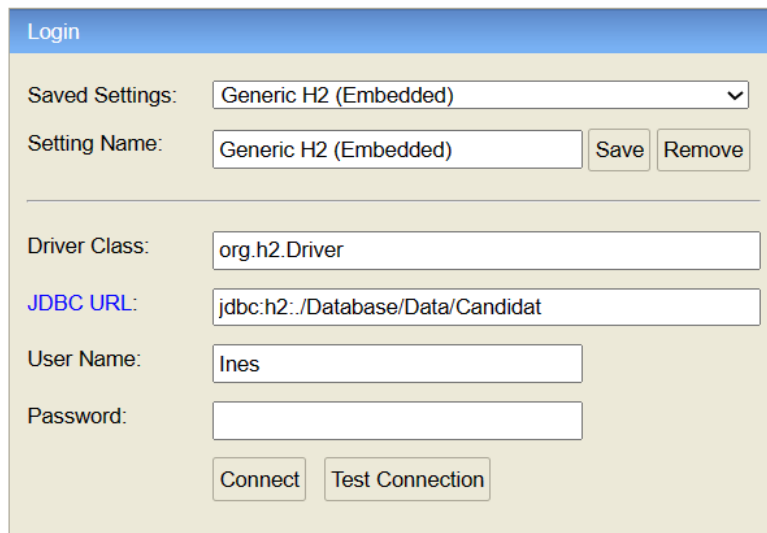
9. Pour le monitoring de la base de données H2, ajoutez la configuration suivante dans le fichier **application.properties**:

```
# H2
spring.h2.console.enabled=true
spring.h2.console.path=/h2

# Datasource
spring.datasource.username=Ines
spring.datasource.password=
spring.datasource.driver-class-name=org.h2.Driver
spring.jpa.hibernate.ddl-auto = create
```

Accédez ensuite à l'URL <http://localhost:8080/h2> pour consulter l'interface de gestion de la base H2.

Ajouter l'URL d'accès au JDBC **./Database/Data/Candidat** pour voir et traiter les données de votre BD h2. Donner votre userName et password :



10. Nous allons maintenant préparer les services “ajouter”, “modifier” et “supprimer” candidat.

Ajoutez la classe CandidatService qui va se charger d'implémenter ces méthodes:

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

@Service
public class CandidatService {

    @Autowired
    private CandidatRepository candidateRepository;

    public Candidat addCandidat(Candidat candidate) {
        return candidateRepository.save(candidate);
    }

    public Candidat updateCandidat(int id, Candidat newCandidat) {
```

```

        if (candidateRepository.findById(id).isPresent()) {

            Candidat existingCandidat = candidateRepository.findById(id).get();
            existingCandidat.setNom(newCandidat.getNom());
            existingCandidat.setPrenom(newCandidat.getPrenom());
            existingCandidat.setEmail(newCandidat.getEmail());

            return candidateRepository.save(existingCandidat);
        } else
            return null;
    }

    public String deleteCandidat(int id) {
        if (candidateRepository.findById(id).isPresent()) {
            candidateRepository.deleteById(id);
            return "candidat supprimé";
        } else
            return "candidat non supprimé";
    }
}

```

Nous préparons par la suite notre API REST. Configurer les requêtes REST dans le controleur REST en ajoutant les méthodes qui suivent:

```

@Autowired
private CandidatService candidatService;

@PostMapping(consumes = MediaType.APPLICATION_XML_VALUE)
@ResponseStatus(HttpStatus.CREATED)
public ResponseEntity<Candidat> createCandidat(@RequestBody Candidat candidat) {
    return new ResponseEntity<>(candidatService.addCandidat(candidat), HttpStatus.OK);
}

@PutMapping(value =("/{id}", produces = MediaType.APPLICATION_JSON_VALUE)
@ResponseStatus(HttpStatus.OK)
public ResponseEntity<Candidat> updateCandidat(@PathVariable(value = "id") int id,
                                                @RequestBody Candidat candidat){
    return new ResponseEntity<>(candidatService.updateCandidat(id, candidat),
HttpStatus.OK);
}

@DeleteMapping(value =("/{id}", produces = MediaType.APPLICATION_JSON_VALUE)
@ResponseStatus(HttpStatus.OK)
public ResponseEntity<String> deleteCandidat(@PathVariable(value = "id") int id){
    return new ResponseEntity<>(candidatService.deleteCandidat(id), HttpStatus.OK);
}

```

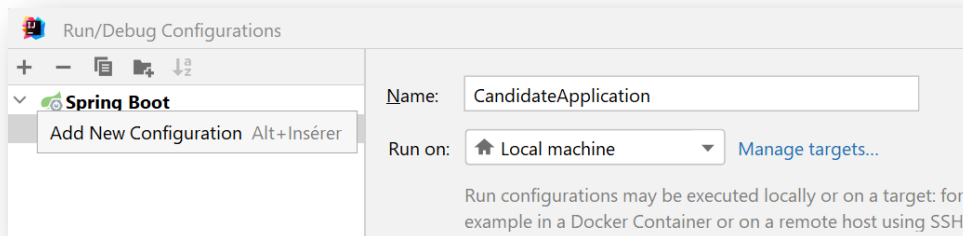
11. Afin de lancer une deuxième instance du même service sur un port différent :

❖ **IDE IntelliJ :**

a. accédez à **Edit Configuration** :

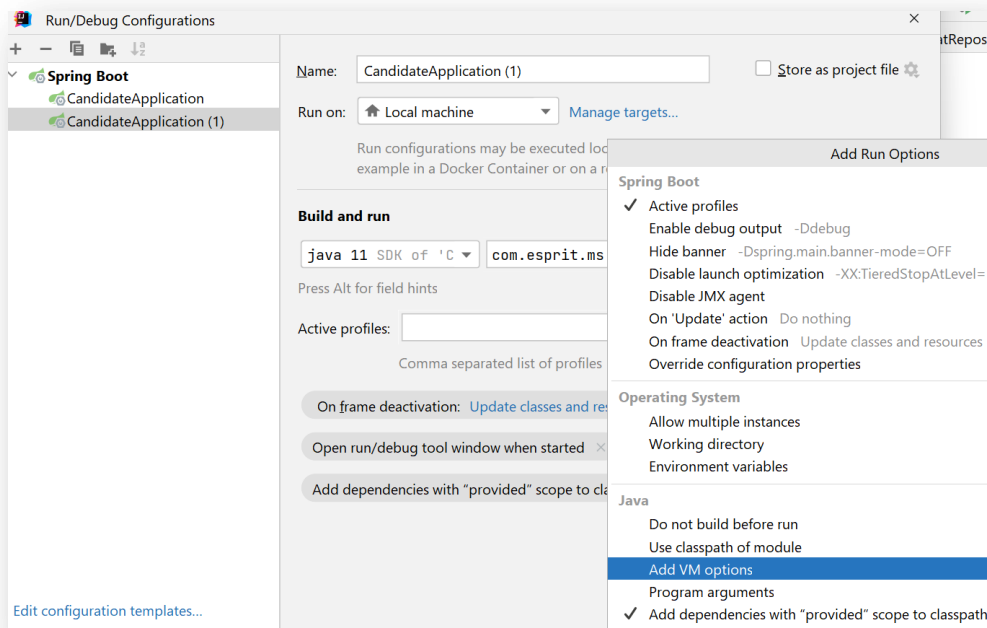


b. Cliquez sur « Add New Configuration » :



c. Choisissez « **Spring Boot** » puis ajouter la configuration suivante en choisissant la classe **CandidateApplication** comme « **Main class** ».

d. Cliquez sur « **Modify options** » --> « **Add VM options** » :



- e. Fixez le port de la nouvelle instance, du même service, en ajoutant l'argument :
`-Dserver.port=8082` :

Name: CandidateApplication (1) ☐ Store as project file ⚙️

Run on: 🏠 Local machine [Manage targets...](#)

Run configurations may be executed locally or on a target: for example in a Docker Container or on a remote host using SSH.

Build and run [Modify options](#) Alt+M

java 11 SDK of 'Canc' -Dserver.port=8082 📄 ↗

com.esprit.ms.candidate.CandidateApplication 📄

Press Alt for field hints

Active profiles:

❖ IDE Eclipse :

- ❖ faites le Run (**run->edit configuration->duplicate configuration**) en ajoutant l'argument `--server.port=8082` dans la partie Arguments/*Program Arguments*.

12. Exécutez la nouvelle instance et y accéder à travers l'URL « <http://localhost:8082/>

****** Ne pas arrêter l'ancienne instance du service s'exécutant sur le port 8080.

13. Travail à faire :

Implémentez un deuxième microservice (dans un nouveau projet spring boot) qui se charge de la gestion des jobs et utilisant **MySQL** comme Base de données. L'entité Job est représentée comme suit:

Job
-ID: Integer -Service: String -Etat: Boolean

Ce deuxième microservice prendra en charge les fonctionnalités suivantes:

- Afficher tous les jobs
- Afficher un job par son id ou son nom
- La modification de l'état de poste :
 - Etat= oui (si poste disponible)
 - Etat= non (si poste est occupé).