

### TP n°3 en Systèmes distribués

#### Objectifs : Mise en œuvre de l'architecture 3-tiers (Technologies Web Services) SpringBoot-H2 Database - JPA - Rest

Spring Boot apporte à Spring une très grande simplicité d'utilisation :

1. Il facilite notamment la création, la configuration et le déploiement d'une application complète. **On n'a plus besoin des fichiers XML à configurer** (pas besoin du fichier du descripteur de déploiement web.xml dans le cas d'une application web).
2. Spring Boot permet de **déployer très facilement une application dans plusieurs environnements sans avoir à écrire des scripts**.  
Pour ce faire, une simple indication de l'environnement (développement ou production) dans le fichier de propriétés (**.properties**) suffit à déployer l'application dans l'un ou l'autre environnement. Ceci est rendu possible grâce à la notion de profil à déclarer toujours dans le fichier de propriétés.
3. Spring Boot possède **un serveur d'application Tomcat embarqué** afin de faciliter le déploiement d'une application web.  
Il est possible d'utiliser un serveur autre ou externe, grâce à une simple déclaration dans le fichier pom.xml.
4. Spring Boot permet de mettre en place **un suivi métrique de l'application** une fois déployée sur le serveur afin de suivre en temps réel l'activité du serveur, ceci grâce à **spring-boot-starter-actuator**.

#### Les dépendances:

- La dépendance **spring-boot-starter-parent** permet de rapatrier la plupart des dépendances du projet. Sans elle, le fichier pom.xml serait plus complexe.
- La dépendance **spring-boot-starter-web** indique à Spring Boot qu'il s'agit d'une application web, ce qui permet à Spring Boot de rapatrier les dépendances comme **SpringMVC**, **SpringContext**, et même le serveur d'application **Tomcat**, etc.

#### Création de Spring Boot Project avec Eclipse:

Il existe trois options pour créer des projets Spring Boot avec Eclipse et Maven

1. Créez manuellement un projet Maven et ajoutez des dépendances de démarrage de Spring Boot.
2. **Spring Initializr** - <https://start.spring.io>
3. Utilisez STS ou **STS Eclipse Plugin** et créez un projet Spring Boot Maven directement depuis Eclipse (télécharger et installer Spring tools suite: <https://spring.io/tools>)

**Option 1:** Création d'un projet Maven de base avec aucune dépendance. Ensuite, ajoutez les démarreurs Spring Boot appropriés dans le fichier pom.xml:

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-devtools</artifactId>
  <scope>runtime</scope>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-test</artifactId>
  <scope>test</scope>
</dependency>
```

**Starter Web** est utilisé pour développer des applications Web Spring Boot ou des services RESTful.

**Starter Test** offre des capacités de test unitaire et de test d'intégration avec Spring Test, Mockito et JUnit.

Celui qui nous manque est la version de ces dépendances. Nous ajouterons un parent de démarrage Spring Boot comme pom parent dans le pom.xml:

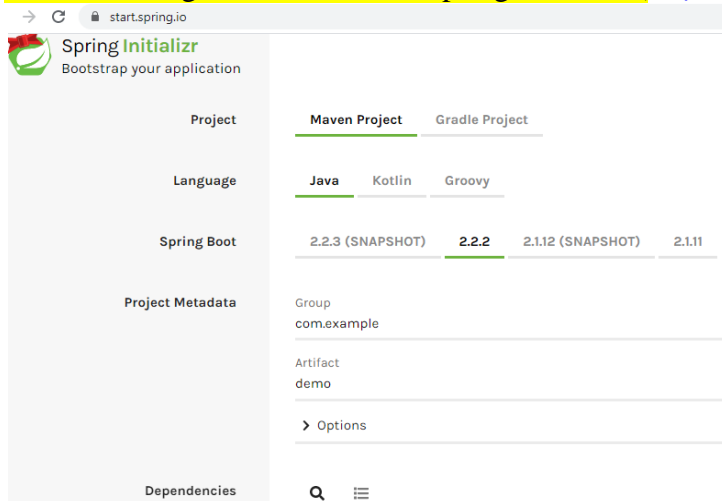
```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>2.0.0.M6</version>
  <relativePath /> <!-- lookup parent from repository -->
</parent>
```

Configurons la version Java pour utiliser 1.8:

```
<properties>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
  <java.version>1.8</java.version>
</properties>
```

## Option 2: Spring Initializr est un excellent outil pour démarrer les projets Spring Boot.

a. Sur un navigateur web, lancer Spring Initializr (<https://start.spring.io/>)



The screenshot shows the Spring Initializr web application. The interface is divided into several sections: Project, Language, Spring Boot, Project Metadata, and Dependencies. The Project section has tabs for Maven Project and Gradle Project. The Language section has tabs for Java, Kotlin, and Groovy. The Spring Boot section shows version 2.2.2 (SNAPSHOT) selected. The Project Metadata section shows Group com.example and Artifact demo. The Dependencies section is empty.

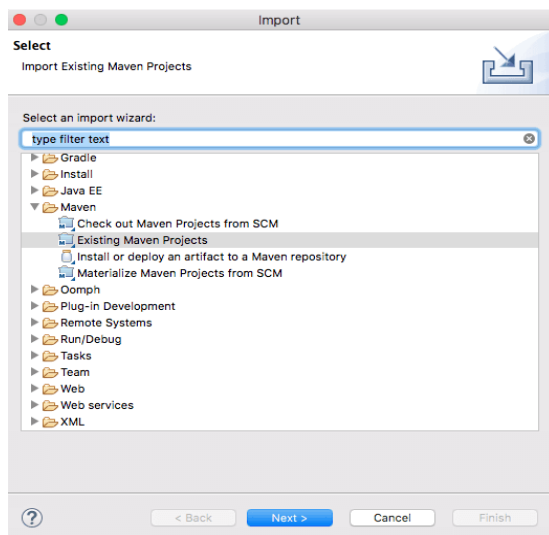
Comme indiqué dans l'image à gauche, les étapes suivantes doivent être effectuées:

1. Lancer Spring Initializr et choisir ce qui suit:
  - nom de groupe
  - le nom artefac
2. Choisir parmi les dépendances suivantes:
  - le web
  - Actionneur
  - DevTools

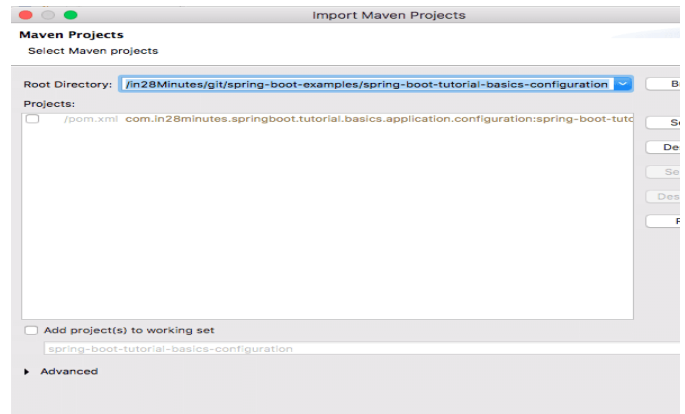
3. Cliquer sur Générer un projet

Cela téléchargerait un fichier ZIP sur votre ordinateur local. Décompressez le fichier zip et extrayez-le dans un dossier

b. Dans Eclipse, cliquez sur Fichier> Importer> Projet Maven existant, comme illustré ci-dessous.



Naviguez ou saisissez le chemin du dossier dans lequel vous avez extrait le fichier ZIP sur l'écran suivant.



### Option 3: Utiliser le plug-in STS ou STS Eclipse pour créer un projet Spring Boot Maven

**Spring Tool Suite (STS):** c'est un Plugin étendu qui soutient le programmeur Spring sur Eclipse.

Avec la suite d'outils Spring, vous pouvez créer directement un projet Spring Boot à partir d'Eclipse. Vous devez soit télécharger l'installation complète de STS, soit installer le plugin STS Eclipse.

Télécharger et installer Spring tools suite: <https://spring.io/tools>)

ou sur l'environnement Eclipse, cliquer sur menu Help/Install new software, le lien dans le tableau ci-dessous, fournit le téléchargement complet de STS ainsi que les sites de mise à jour du plug-in STS Eclipse.

ECLIPSE	ARCHIVE	SIZE
4.14.0	<a href="#">springsource-tool-suite-3.9.11.RELEASE-e4.14.0-updatesite.zip</a>	178MB

Mettre à jour les sites: Si vous souhaitez installer STS 3 dans une installation Eclipse existante, vous pouvez utiliser l'un des sites de mise à jour suivants. Veuillez choisir celle qui correspond à la version Eclipse que vous utilisez:

ECLIPSE	UPDATE SITES
2019-06 (4.12)	<a href="https://download.springsource.com/release/TOOLS/update/e4.12/">https://download.springsource.com/release/TOOLS/update/e4.12/</a>
2019-03 (4.11)	<a href="https://download.springsource.com/release/TOOLS/update/e4.11/">https://download.springsource.com/release/TOOLS/update/e4.11/</a>
2018-12 (4.10)	<a href="https://download.springsource.com/release/TOOLS/update/e4.10/">https://download.springsource.com/release/TOOLS/update/e4.10/</a>

## Exercice 1. TP Application avec l'option 2:

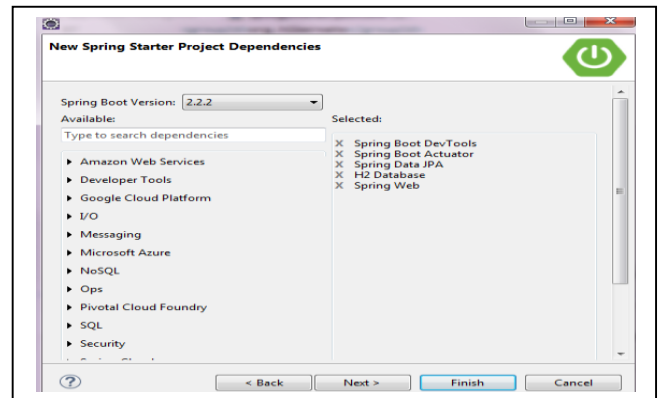
a. Sur un navigateur web, lancer Spring Initializr (<https://start.spring.io/>)

1. Lancer Spring Initializr et choisir ce qui suit:

- nom de groupe: com.springboot.biblio
- le nom artefact : spring-boot-biblio

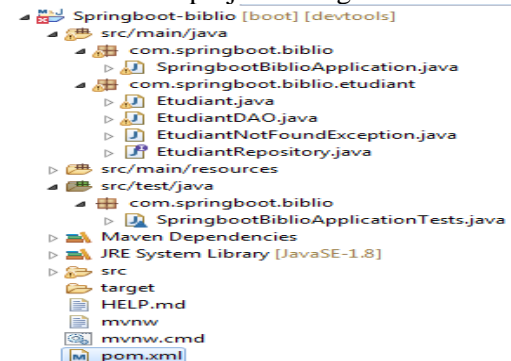
2. Choisir parmi les dépendances suivantes:

- le web
- Actionneur
- DevTools



3. Cliquer sur Générer un projet: Cela téléchargerait un fichier ZIP sur votre ordinateur local. Décompressez le fichier zip et extrayez-le dans un dossier

Les classes du projet : Vue globale des classes



### Les classes du projet :

**La classe Etudiant.java:** est une classe 'POJO' qui contient l'ensemble des attributs qui représentent les champs de la table etudiant dans la base de données. Chaque champ à ses getters et setters; Deux méthodes constructeur avec et sans paramètres, et une méthode ToString pour affichage.

**@entity:** Déclare que cette classe ne s'agit pas d'une classe ordinaire mais d'une table à la base de donne qui sera persiste.

**@id:** le champ id est un identifiant de la table etudiant.

**@GeneratedValue:** génération d'une clé auto incrémente.

**package** com.springboot.biblio.etudiant;

**import** javax.persistence.Entity;

**import** javax.persistence.GeneratedValue;

**import** javax.persistence.Id;

@Entity

**public class** Etudiant {

    @Id

    @GeneratedValue(strategy=GenerationType.AUTO)

**private** long id;

**private** String nom;

**private** String prenom;

**private** String reffil;

    // ajouter le constructeur sans et avec des paramètres

    //ajouter les méthodes getters et setters et la méthode ToString

}

**La classe EtudiantDAO.java:** Cette classe contient les méthodes CRUD implémentées à partir de l'interface JpaRepository via l'attribut **etudiantDao**

**@RestController:** Dit à Spring que cette classe est un contrôleur qui retourne les résultats des méthodes comme JSON à l'aide des annotations suivantes :

**@GetMapping("/etudiants/{id}") :** Retourne l'enregistrement ayant id=id

**@DeleteMapping("/etudiants/{id}") :** Supprime l'étudiant par id via JSON

**@PutMapping("/etudiants/{id}") :** Modifie un étudiant via Id

**@PostMapping("/etudiants") :** Ajoute un étudiant via la Methode http POST

```
package com.springboot.biblio.etudiant;
```

```
import java.net.URI;
```

```
import java.util.List;
```

```
import java.util.Optional;
```

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

```
import org.springframework.http.ResponseEntity;
```

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

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

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

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

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

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

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

```
import org.springframework.web.servlet.support.ServletUriComponentsBuilder;
```

```
@RestController
```

```
public class EtudiantDAO {
```

```
// Annotation de la variable etudiantDao par @Autowired afin que Spring se charge d'en fabriquer une instance. etudiantDao a désormais //accès à toutes les méthodes que nous allons définir.
```

```
@Autowired
```

```
private EtudiantRepository etudiantDao;
```

```
@GetMapping("/etudiants") // recherche tous les étudiants
```

```
public List<Etudiant> retrieveAllEtudiants() {
```

```
    return etudiantDao.findAll();
```

```
}
```

```
@GetMapping("/etudiants/{id}") // recherche via id
```

```
public Etudiant retrieveEtudiant(@PathVariable long id) {
```

```
    Optional<Etudiant> etudiant = etudiantDao.findById(id);
```

```
    if (!etudiant.isPresent())
```

```
        throw new EtudiantNotFoundException("id-" + id);
```

```
    return etudiant.get();
```

```
}
```

```
@GetMapping("/etudiants/filiere/{refil}") // recherche par reffil
```

```
public List<Etudiant> retrieveEtudiantByFiliere(@PathVariable String reffil) {
```

```
    return etudiantDao.findByReffil(reffil);
```

```
}
```

```
@DeleteMapping("/etudiants/{id}") // Suppression
```

```
public void deleteEtudiant(@PathVariable long id) {
```

```
    etudiantDao.deleteById(id);
```

```
}
```

```
@PostMapping("/etudiants") // Ajout
```

```
public ResponseEntity<Object> createEtudiant(@RequestBody Etudiant etudiant) {
```

```
    Etudiant savedEtudiant = etudiantDao.save(etudiant);
```

```
    URI location = ServletUriComponentsBuilder.fromCurrentRequest().path("/{id}")  
        .buildAndExpand(savedEtudiant.getId()).toUri();
```

```

        return ResponseEntity.created(location).build();
    }
    @PutMapping("/etudiants/{id}") // Modification
    public ResponseEntity<Object> updateEtudiant(@RequestBody Etudiant etudiant,
    @PathVariable long id) {
        Optional<Etudiant> etudiantOptional = etudiantDao.findById(id);
        if (!etudiantOptional.isPresent())
            return ResponseEntity.notFound().build();
        etudiant.setId(id);
        etudiantDao.save(etudiant);
        return ResponseEntity.noContent().build();
    }
}

```

**L'interface EtudiantRepository.java:** Dans SpringBoot il n'est pas nécessaire de créer les méthodes CRUD puisque ils sont tous prédéfinies et déclarés via l'implémentation de l'interface JpaRepository, il suffit juste de les appeler dans la classe EtudiantDAO.java.

Note: Pour rechercher un ou plusieurs enregistrements par un attribut différent de celui de l'Id il faut créer une méthode en respectant la syntaxe suivante :

findBy+le nom du champ dont on veut chercher par exemple : findByreffil.

**@Repository:** Dit à Spring de rendre la classe actuelle comme bean afin d'être instancié par l'attribut EtudiantRepository dans la classe EtudiantDAO.java.

```
package com.springboot.biblio.etudiant;
```

```
import java.util.List;
```

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

```
import org.springframework.stereotype.Repository;
```

```
@Repository
```

```
public interface EtudiantRepository extends JpaRepository<Etudiant, Long>{
    List<Etudiant> findByReffil(String reffil);
}

```

**La classe SpringBootBibliothequeApplication.java**

```
package com.springboot.biblio;
```

```
import org.springframework.boot.SpringApplication;
```

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

```
@SpringBootApplication
```

```
public class SpringBootBibliothequeApplication {
    public static void main(String[] args) {
        SpringApplication.run(SpringBootBibliothequeApplication.class, args);
    }
}

```

La méthode statique [SpringApplication.run] a pour rôle de créer le contexte Spring, ç-à-d créer les différents beans trouvés soit dans les classes de configuration soit dans les dossiers explorés par l'annotation [@ComponentScan].

**La classe EtudiantNotFoundException.java** pour la gestion des exception

```
package com.springboot.biblio.etudiant;
```

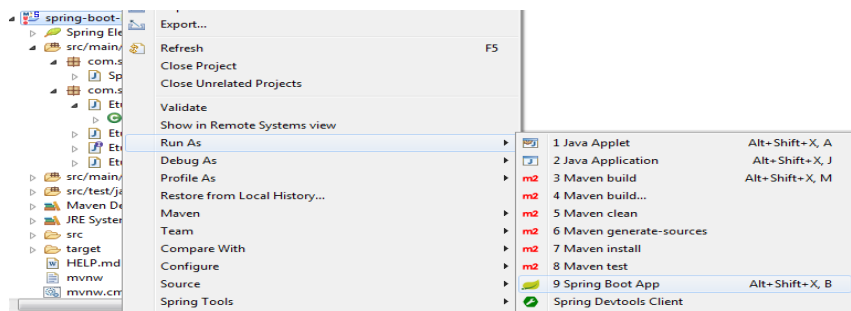
```
public class EtudiantNotFoundException extends RuntimeException {
```

```
    private static final long serialVersionUID = 1L;
```

```
    public EtudiantNotFoundException(String exception) {
        super(exception);
    }
}

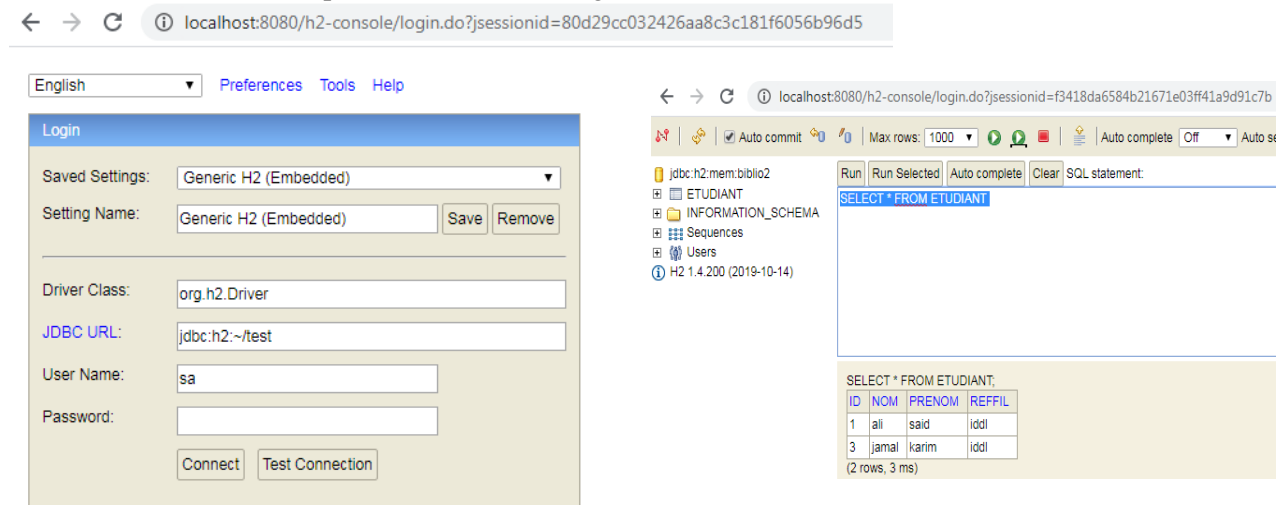
```

## Test du projet :

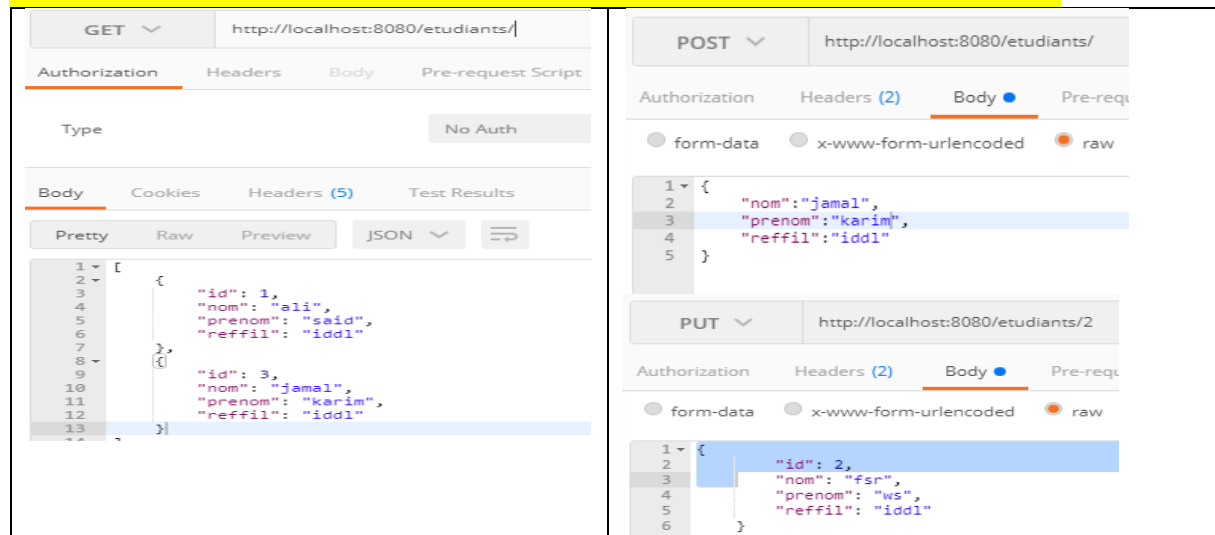


## L'accès à la Base de données H2:

Dans l'écran suivant, vous pouvez utiliser un navigateur web : localhost:8080/h2-console



## Test les web services avec l'outil PostMan sans démarrer le serveur tomcat:



## Exercice 2:

En suivant le modèle précédent, Créer le package com.springboot.biblio.livre, Créer les classes permettant de gérer les informations des livres dans la bibliothèque. Vous allez définir :

La classe "Livre.java" qui contient quatre attributs d'instance id (long), titre (String), auteur(String) et prix (double), L'interface LivreRepository.java, La classe LivreDAO.java qui contient les méthodes CRUD implémentées à partir de l'interface JpaRepository, etc...