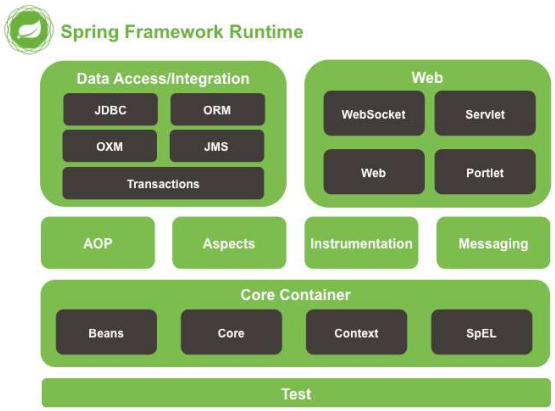
Spring 6

Spring Boot 3.2.1

Spring Framework

- **Spring Framework** est un framework mature, puissant et très flexible axé sur la création d'applications Web en Java.
- Il prend en charge la plupart des aspects de bas niveau de la construction de l'application
  - » concentrer réellement sur les fonctionnalités et la logique métier.
- Il est très activement maintenu et dispose d'une grande communauté de développeurs.
  - » Régulièrement mis à jour et aligné sur l'écosystème Java.

Spring Framework



Spring Framework

- Spring Framework se compose de fonctionnalités organisées en une vingtaine de modules.
- Ces modules sont regroupés en Core Container, Data Access/Integration, Web, AOP (Aspect Oriented Programming), Instrumentation, Messaging et Test ...
  - **Core** : Fournit des fonctionnalités de base telles que DI (injection de dépendance), internationalisation, validation et AOP (programmation orientée aspect)
  - **Accès aux données** : prend en charge l'accès aux données via JTA (Java Transaction API), JPA (Java Persistence API) et JDBC (Java Database Connectivity)
  - **Web** : prend en charge à la fois l'API Servlet (Spring MVC) et l'API récemment réactive (Spring WebFlux), et prend également en charge WebSockets, STOMP et WebClient
  - **Intégration** : Prend en charge l'intégration à Enterprise Java via JMS (Java Message Service), JMX (Java Management Extension) et RMI (Remote Method Invocation)
  - **Tests** : prise en charge étendue des tests unitaires et d'intégration via des objets fictifs (Mock), des montages de test, la gestion de contexte et la mise en cache

Module Core

- Module **Core** se compose des modules **spring-core, spring-beans, spring-context, spring-context-support et spring-expression** (Spring Expression Language).
- Les modules **spring-core** et **spring-beans** fournissent les éléments fondamentaux du framework, y compris les fonctionnalités *IoC* et *Dependency Injection*.
- **BeanFactory**, Il s'agit d'une implémentation sophistiquée du pattern Factory.
  - » vous permet de découpler la configuration et la spécification des dépendances de votre logique de programme.

Module Core

- Le module Context (**spring-context**) : c'est un moyen d'accéder aux objets d'une manière similaire à un registre JNDI (Annuaire).
- Le module **Context** hérite ses fonctionnalités du module **Beans.**
- Le module **Context** ajoute la prise en charge de l'internationalisation, la propagation d'événements, le chargement de ressources et la création transparente de contextes par un *conteneur Servlet*, par exemple.
- L' interface **ApplicationContext** est le point central du module Contexte.

Module Core

- Le module **spring-expression** fournit un langage d'expression puissant pour interroger et manipuler un graphe d'objets lors de l'exécution.
- Il s'agit d'une extension du langage d'expression unifié (unified EL) tel que spécifié dans la spécification JSP 2.1.
- Le langage prend en charge la définition et l'obtention des valeurs de propriété, l'attribution de propriété, l'invocation de méthode, l'accès au contenu des tableaux, des collections et des indexeurs, les opérateurs logiques et arithmétiques, les variables nommées et la récupération d'objets par nom à partir du conteneur IoC de Spring.
- Il prend également en charge la projection et la sélection de listes ainsi que les agrégations de listes courantes.

AOP

- Le module **spring-aop** fournit une implémentation de programmation orientée aspect conforme à Alliance AOP.
- Il permet de définir, par exemple, des intercepteurs de méthode et des points d'accrochage pour découpler proprement le code qui implémente des fonctionnalités qui doivent être séparées.
- Le module séparé **spring-aspects** fournit une intégration avec **AspectJ**.

Écosystème Spring

- Bien que la liste des projets de Spring soit longue et elle change constamment, il y en a quelques-uns qui méritent d'être mentionnés :
- **Boot** : nous fournit un ensemble de modèles très optionnels mais extensibles pour créer divers projets basés sur Spring en un rien de temps. Il est très facile de créer des applications Spring autonomes avec Tomcat intégré ou un conteneur similaire.
- **Cloud** : Fournit une assistance pour développer facilement certains des modèles de système distribué courants tels que service de discovery, le circuit breaker et le gateway API. Cela nous aide à réduire les efforts de déploiement de ces modèles passe-partout sur des plates-formes locales, distantes ou même gérées.

Écosystème Spring

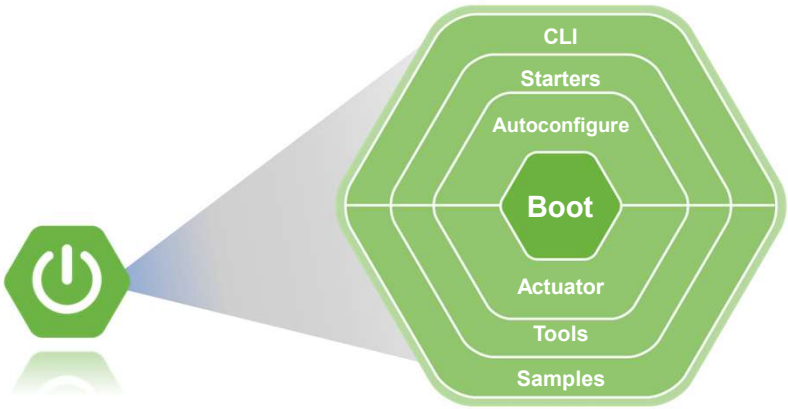
- **Sécurité** : Fournit un mécanisme robuste pour développer l'authentification et l'autorisation pour les projets basés sur Spring de manière hautement personnalisable. Avec un support déclaratif minimal, nous obtenons une protection contre les attaques courantes telles que la fixation de session, le détournement de clics et la falsification de requêtes intersites.
- **Mobile** : Fournit des capacités pour détecter l'appareil et adapter le comportement de l'application en conséquence. De plus, prend en charge la gestion des vues adaptée aux appareils pour une expérience utilisateur optimale, la gestion des préférences de site et le changement de site.
- **Batch** : Fournit un cadre léger pour le développement d'applications de traitements par lots pour les systèmes d'entreprise tels que l'archivage de données. A un support intuitif pour la planification, le redémarrage, le saut, la collecte de métriques et la journalisation. En outre, prend en charge la mise à l'échelle pour les travaux à volume élevé grâce à l'optimisation et au partitionnement.

Spring Boot

Introduction à Spring Boot

- Un point de focus unique
- Démarrage rapide avec Spring
- Ensemble des besoins non fonctionnelles une application
- Expose plusieurs fonctionnalités par défaut
- Issue rapide si vous voulez changer les valeurs par défaut
- Une façon pour Spring d'avoir une opinion

Spring Boot Modules



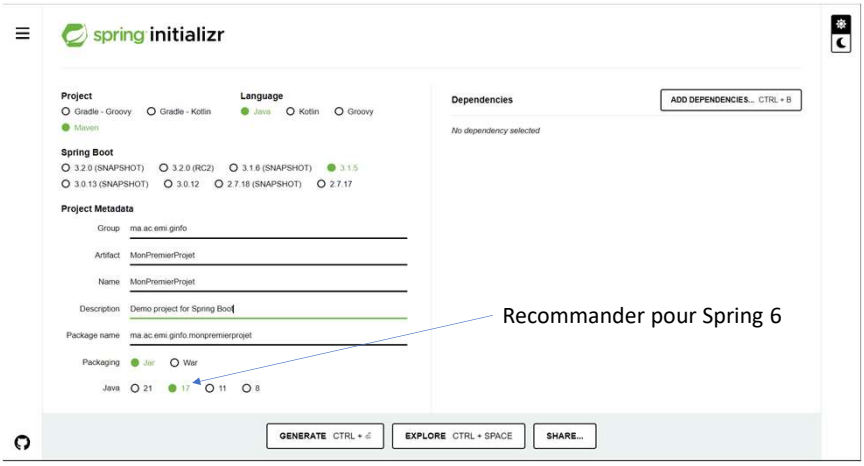
Spring Boot Modules

- Spring Boot - bibliothèque principale supportant les autres parties de Spring Boot
- Configuration automatique de démarrage de spring - une seule annotation @EnableAutoConfiguration crée un contexte de spring complet
- Spring Boot Starters - Un ensemble de descripteurs de dépendance pratiques que vous pouvez inclure dans votre application.
- Spring Boot CLI - compile et exécute un projet en tant qu'application Spring
- Spring Boot Actuator - fonctionnalités communes non fonctionnelles permettant de déployer une application instantanément et de la prendre en charge en production
- Spring Boot Tools - pour créer et exécuter des archives JAR et WAR autonomes
- Spring Boot Samples - un large éventail d'applications d'exemple

Idées fortes

- Accélérer le développement d'applications Spring
- Convention over Configuration
- Pas de code généré
- Déploiement facilité
- Prêt pour la production

Création d'une application Spring



Projet Spring : POM.xml

```
11      <groupId>ma.ac.emi.ginfo</groupId>
12      <artifactId>MonPremierProjet</artifactId>
13      <version>0.0.1-SNAPSHOT</version>
14      <name>MonPremierProjet</name>
15      <description>Demo project for Spring Boot</description>
16      <properties>
17          <java.version>17</java.version>
18      </properties>
```


Projet Spring : configuration initiale

```
5       <parent>
6           <groupId>org.springframework.boot</groupId>
7           <artifactId>spring-boot-starter-parent</artifactId>
8           <version>3.1.5</version>
9           <relativePath/> <!-- lookup parent from repository -->
10      </parent>
```

Gestion optimale des dépendances entre les versions des Frameworks nécessaires et utilisées au sien du projet

Projet Spring : ses dépendances

```
19      <dependencies>
20          <dependency>
21              <groupId>org.springframework.boot</groupId>
22              <artifactId>spring-boot-starter</artifactId>
23          </dependency>
24
25          <dependency>
26              <groupId>org.springframework.boot</groupId>
27              <artifactId>spring-boot-starter-test</artifactId>
28              <scope>test</scope>
29          </dependency>
30      </dependencies>
```

Pas de version spécifique
Tous est hérités du projet parent

Dépendances 1 niveau

pom.xml (SpringBoot1) | spring-boot-starter-3.1.5.pom

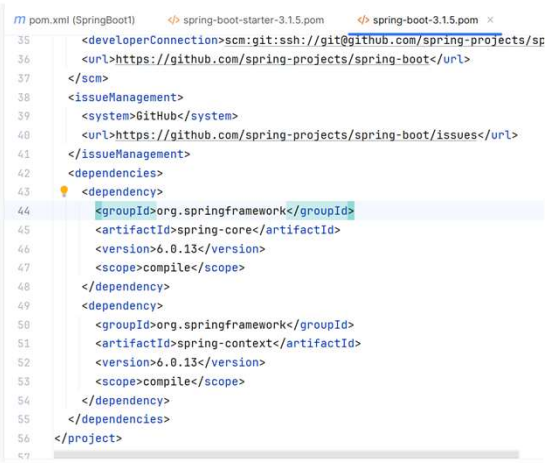
```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot</artifactId>
    <version>3.1.5</version>
    <scope>compile</scope>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-autoconfigure</artifactId>
    <version>3.1.5</version>
    <scope>compile</scope>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-logging</artifactId>
    <version>3.1.5</version>
    <scope>compile</scope>
  </dependency>
```

pom.xml (SpringBoot1) | spring-boot-starter-3.1.5.pom

```
  </dependency>
  <dependency>
    <groupId>jakarta.annotation</groupId>
    <artifactId>jakarta.annotation-api</artifactId>
    <version>2.1.1</version>
    <scope>compile</scope>
  </dependency>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-core</artifactId>
    <version>6.0.13</version>
    <scope>compile</scope>
  </dependency>
  <dependency>
    <groupId>org.yaml</groupId>
    <artifactId>snakeyaml</artifactId>
    <version>1.33</version>
    <scope>compile</scope>
  </dependency>
</dependencies>
</project>
```

Attention : Il faut assurer la compatibilité entre les sous dépendances de chaque dépendance vis-à-vis les autres dépendances

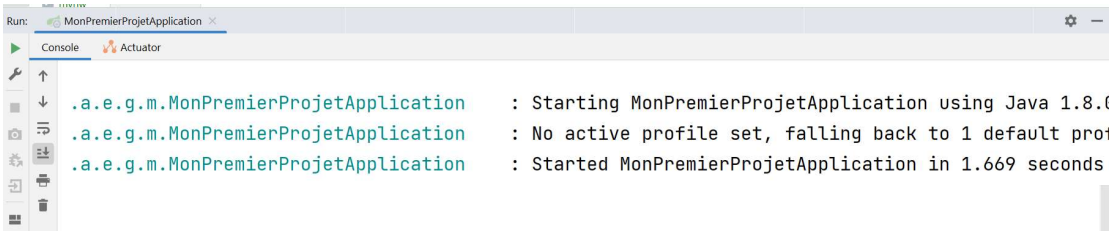
Exemple de dépendances 2 niveau



Projet Spring : Code Java

```java
import ...

1 usage
@SpringBootApplication
public class MonPremierProjetApplication {

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

}
```

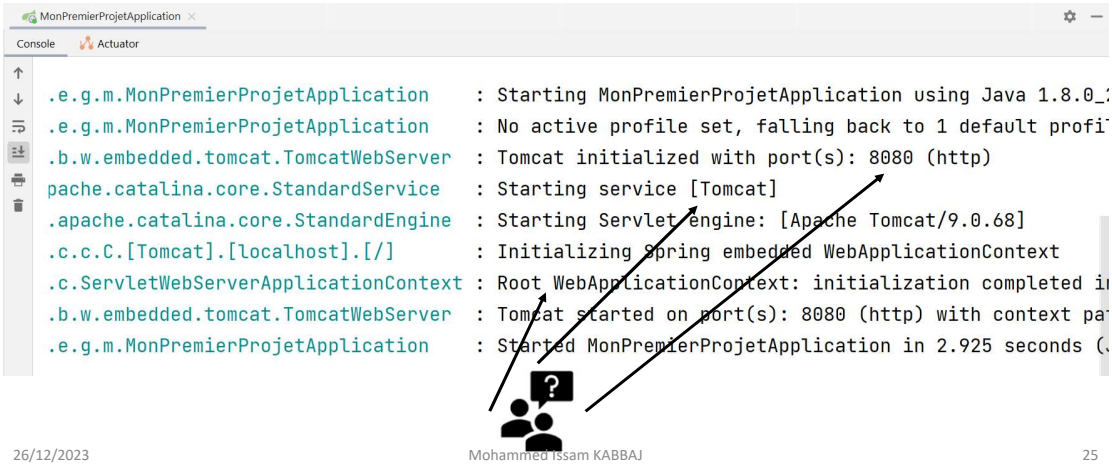
Projet Spring : Première Exécution



Ma première modification

```
19    <dependencies>
20        <dependency>
21            <groupId>org.springframework.boot</groupId>
22            <artifactId>spring-boot-starter-web</artifactId>
23        </dependency>
24
25        <dependency>
26            <groupId>org.springframework.boot</groupId>
27            <artifactId>spring-boot-starter-test</artifactId>
28            <scope>test</scope>
29        </dependency>
30    </dependencies>
```

Nouvelle exécution



Explication

- Nous utilisons la notation **@SpringBootApplication** sur la classe principale de l'application.
- Cette annotation transforme cette classe comme classe de configuration principale d'application.
- Cela équivaut à **@Configuration**, **@EnableAutoConfiguration** et **@ComponentScan** ensemble.

@SpringBootApplication

- **@Configuration** : définie la classe comme source de définitions de bean pour le contexte de l'application.
- **@EnableAutoConfiguration** : indique à Spring Boot de commencer à ajouter des beans en fonction des paramètres de chemin de classe, d'autres beans et de divers paramètres de propriété.
  - Par exemple, si spring-web se trouve sur le POM.XML, cette annotation marque l'application en tant qu'application Web et active des comportements clés, tels que la configuration d'un DispatcherServlet et le démarrage de Tomcat.
- **@ComponentScan** : indique à Spring de rechercher d'autres composants, configurations et services dans le package.

Auto-configuration

- L'annotation **@SpringBootApplication** déclenche la configuration automatique de l'infrastructure Spring
- Au démarrage de l'application, Spring Boot :
  - Scanne toutes classes de @Configuration
    - Classes de configuration spécifiques à l'application
    - Classes de Spring Boot suffixées par AutoConfiguration
    - 3rd party starters
  - Utilise les JAR présents dans le classpath pour prendre des décisions

Comment Changer la configuration

- Plusieurs options sont possible :
  - POM.XML
  - Application.properties
  - Annotation Java
  - Fichiers XML de configuration
  - …

Configuration via POM.xml

pom.xml (MonPremierProjet)

```
19    <dependencies>
20        <dependency>
21            <groupId>org.springframework.boot</groupId>
22            <artifactId>spring-boot-starter</artifactId>
23        </dependency>
24
25        <dependency>
26            <groupId>org.springframework.boot</groupId>
27            <artifactId>spring-boot-starter-test</artifactId>
28            <scope>test</scope>
29        </dependency>
30    </dependencies>
31
```

Configuration via POM.xml

```
19      <dependencies>
20          <dependency>
21              <groupId>org.springframework.boot</groupId>
22              <artifactId>spring-boot-starter-web</artifactId>
23          </dependency>
24
25          <dependency>
26              <groupId>org.springframework.boot</groupId>
27              <artifactId>spring-boot-starter-test</artifactId>
28              <scope>test</scope>
29          </dependency>
30      </dependencies>
```

```
.e.g.m.MonPremierProjetApplication    : Starting MonPremierProjetApplication using Java 1.8.0_
.e.g.m.MonPremierProjetApplication    : No active profile set, falling back to 1 default profi
.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 8080 (http)
pache.catalina.core.StandardService   : Starting service [Tomcat]
.apache.catalina.core.StandardEngine  : Starting Servlet engine: [Apache Tomcat/9.0.68]
.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed i
.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8080 (http) with context pa
.e.g.m.MonPremierProjetApplication    : Started MonPremierProjetApplication in 2.925 seconds (
```

Changement d'une seule dépendance du projet à impliquer le démarrage de Apache Tomcat et sa configuration sur le port HTTP 8080

Configuration via application.properties

pom.xml (MonPremierProjet) | application.properties

```
1   server.port=9999
2
```

```
mierProjetApplication    : Starting MonPremierProjetApplication using Java 1.8.0_211 on kabbaj
mierProjetApplication    : No active profile set, falling back to 1 default profile: "default"
.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 9999 (http)
a.core.StandardService   : Starting service [Tomcat]
ina.core.StandardEngine  : Starting Servlet engine: [Apache Tomcat/9.0.68]
t].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
ServerApplicationContext : Root WebApplicationContext: initialization completed in 1642 ms
.tomcat.TomcatWebServer  : Tomcat started on port(s): 9999 (http) with context path ''
mierProjetApplication    : Started MonPremierProjetApplication in 3.242 seconds (JVM running f
```

Configuration via Annotations

```java
1 usage
@SpringBootApplication
public class MonPremierProjetApplication {

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

    @Bean
    public CommandLineRunner traitementParallele(){
        return args -> {
            System.out.println("Je suis un traitement parallele ");
        };
    }
}
```

```
2022-11-01 23:59:49.925  INFO 17216 --- [           main] m.a.e.g.m.MonPremierProjetApplicati
2022-11-01 23:59:49.930  INFO 17216 --- [           main] m.a.e.g.m.MonPremierProjetApplicati
2022-11-01 23:59:51.343  INFO 17216 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebSe
2022-11-01 23:59:51.355  INFO 17216 --- [           main] o.apache.catalina.core.StandardServ
2022-11-01 23:59:51.356  INFO 17216 --- [           main] org.apache.catalina.core.StandardEn
2022-11-01 23:59:51.584  INFO 17216 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]
2022-11-01 23:59:51.585  INFO 17216 --- [           main] w.s.c.ServletWebServerApplicationCo
2022-11-01 23:59:52.221  INFO 17216 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebSe
2022-11-01 23:59:52.241  INFO 17216 --- [           main] m.a.e.g.m.MonPremierProjetApplicati
Je suis un traitement parallele
```


Attention

Certaines modifications nécessite d'autres éléments pour fonctionner correctement.

```xml
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>8.0.23</version>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>
```

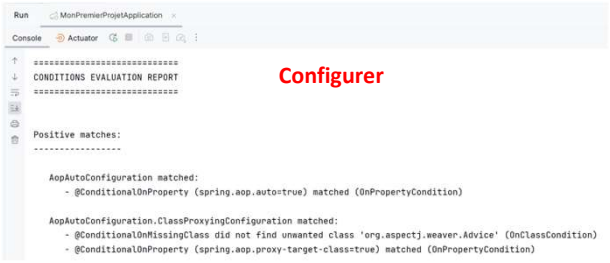
```
Error starting ApplicationContext. To display the conditions report re-run your application wit
2022-11-02 00:07:53.395 ERROR 944 --- [           main] o.s.b.d.LoggingFailureAnalysisReporter

***************************
APPLICATION FAILED TO START
***************************

Description:

Failed to configure a DataSource: 'url' attribute is not specified and no embedded datasource c

Reason: Failed to determine a suitable driver class


Action:
```

Configurer ou pas ?

- Ajouter dans le fichier **application.properties** la ligne suivante :

  logging.level.org.springframework=debug

- puis exécuter



Configurer

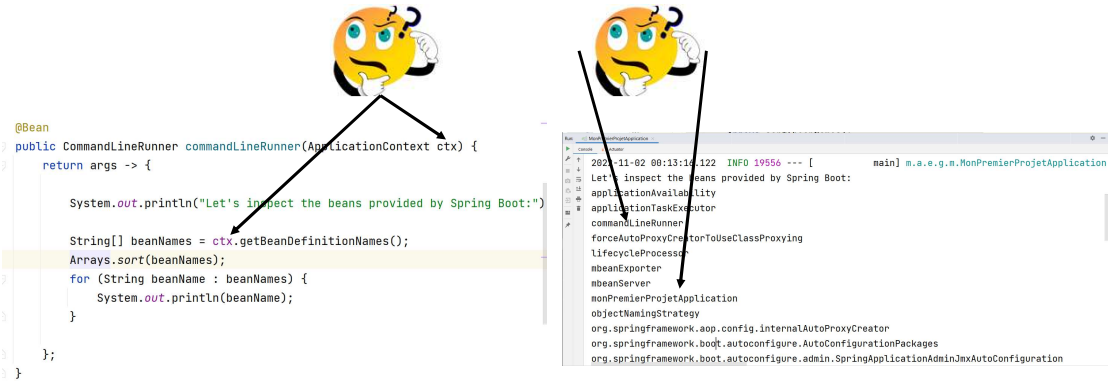
Non Configurer

- Qu'est ce qui est configuré avec spring boot web ?

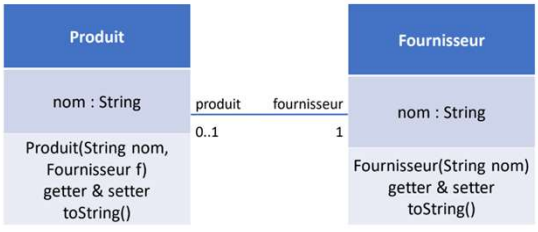
Spring Boot Starter

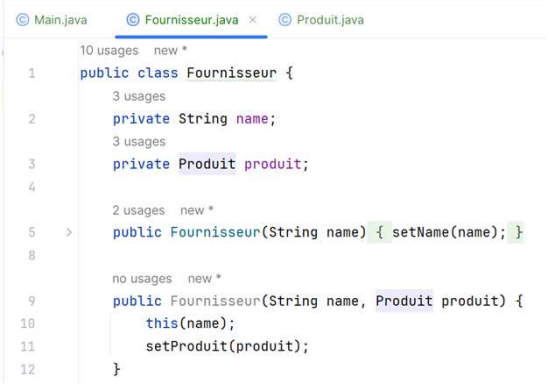
- **Spring Boot** propose une variété de projets de démarrage :
  - **spring-boot-starter-web** pour créer des applications Web avec API REST
    - spring-webmvc, spring-web, spring-boot-starter-tomcat, spring-boot-starter-json
  - **spring-boot-starter-test** pour tests unitaires
  - **spring-boot-starter-data-jpa** pour communiquer avec la base de données à l'aide de JPA
  - **spring-boot-starter-jdbc** pour communiquer avec la base de données à l'aide de JDBC
  - **spring-boot-starter-security** pour sécuriser votre site Web ou votre API REST

Injection des dépendances (ID)

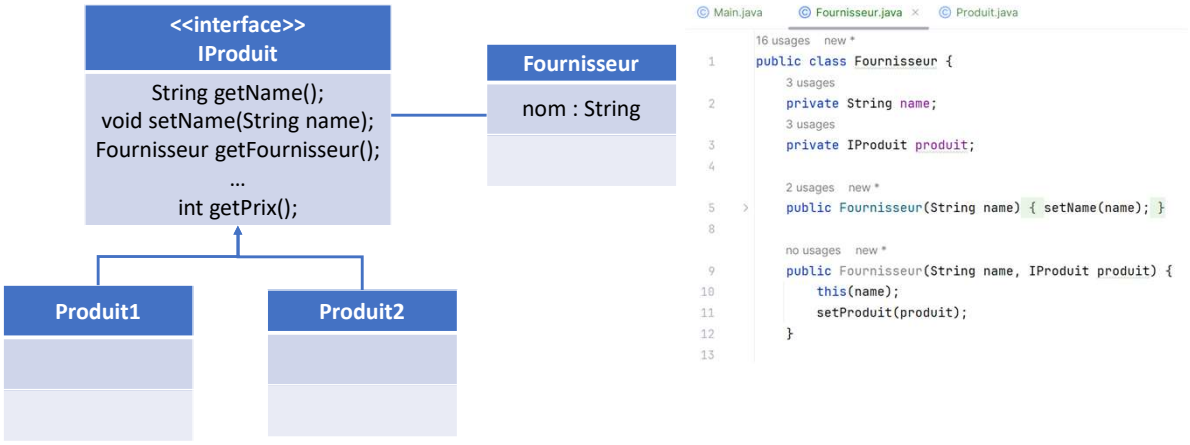


Origine du problème : Dépendance forte

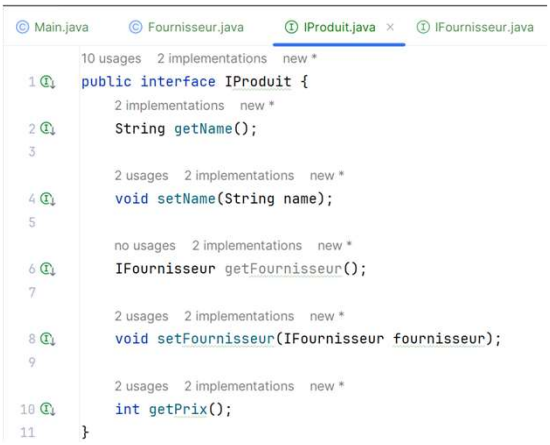


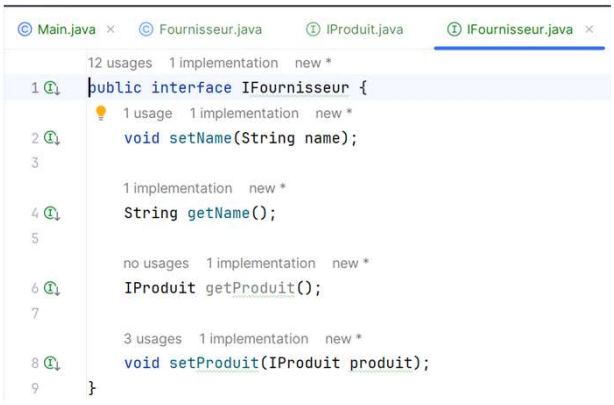


Première réflexion



Aller plus loin dans cette réflexion





Dépendance vis-à-vis de new

Main.java × | Fournisseur.java | IProduit.java | IFournisseur.java | Produit.java

```
new *
public class Main {
    new *
    public static void main(String[] args) {
        //attention encore une dependence vers la Classe Founisseur
        IFournisseur hp = new Fournisseur( name: "HP");
        //attention encore une dependence vers la Classe Produit
        IProduit x360 = new Produit( name: "Elitebook x360", hp);
```


Première solution

Main.java | Config.txt ×

```
Produit2
Produit2
Fournisseur
setName
setFournisseur
```

```
try {
//          Scanner scanner = new Scanner(System.in);
        Scanner scanner = new Scanner(new FileReader( fileName: "Config.txt"));

        classProduitName = scanner.next();
        classProduit2Name = scanner.next();
        classFournisseurName = scanner.next();
        methodeSetNameName = scanner.next();
        methodeSetFournisseurName = scanner.next();

        // Chargement dynamique des classes
        Class produitClass = Class.forName(classProduitName);
        // Creation d'instance
        // IProduit x360 = new Produit()
        x360 = (IProduit) produitClass.newInstance();
        // Invocation d'une methode d'objet
        Method methodeSetName2 = produitClass.getMethod(methodeSetNameName, new Class[]{String.class});
        // x360.setName("Elitebook x360");
        methodeSetName2.invoke(x360, ...args: "Elitebook x360");
```

Solution Spring : IOC + ID

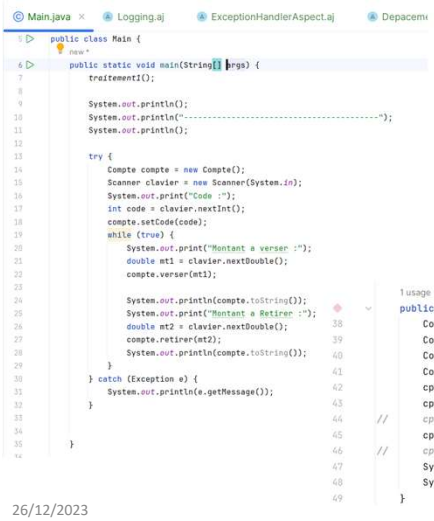
```
new *
public class Main {
    new *
    public static void main(String[] args) {
        //inverse of controle  IOC
        ApplicationContext context = new ClassPathXmlApplicationContext( configLocation: "beans.xml");

        // injection dependance
        IFournisseur hp = (IFournisseur) context.getBean( s: "hp");
        IProduit x360 = (IProduit) context.getBean( s: "x360");
        System.out.println("hp = " + hp);
        System.out.println("x360 = " + x360);

    }
}
```

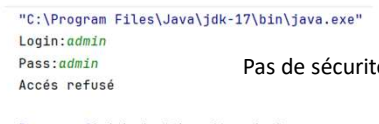
2 - Utiliser

1 - Déclarer

```
<bean id="hp" class="Fournisseur">
    <property name="name" value="HP" />
</bean>
<bean id="x3602" class="Produit3">
    <property name="name" value="Elitebook x3602" />
    <property name="fournisseur" ref="hp" />
</bean>
<bean id="x360" class="Produit3">
    <constructor-arg name="name" value="Elitebook x360" />
    <constructor-arg name="fournisseur" ref="hp" />
</bean>
```


Mieux Encore : se concerter sur le métier





Pas de sécurité dans votre code métier

Contrôle ajustable et parametrable
Prendre en charge de besoin Non Fonctionnels
- Performance
- Traçabilité
…

Programmation par Aspect

- Comment ?

Main.java | Logging.aj | ExceptionHandlerAspect.aj

```
new *
public aspect Logging {


    Logger logger = Logger.getLogger(this.getClass().getName());
    long t1, t2;
    pointcut avant():execution(* *..Compte.*(..));
    pointcut apres():execution(* *..Compte.*(..));

    before():avant() {
        t1 = System.currentTimeMillis();
        logger.info( msg: "************************");
        logger.info( msg: "Avant " + thisJoinPoint.getSignature());
    }

    after():apres() {
        t2 = System.currentTimeMillis();
        logger.info( msg: "************************");
        logger.info( msg: "Apres " + thisJoinPoint.getSignature());
        logger.info( msg: "Duree d'execution=" + (t2 - t1) + " ms");
    }
```

```
package ma.emi.aspect;

new *
public aspect ExceptionHandlerAspect {
    pointcut anyMethod():call(* *.*.*(..));

    after()throwing(Exception e): anyMethod(){
        System.out.println(thisJoinPoint.getSignature());
        System.out.println(e.getClass());
        System.out.println(e.getMessage());
    }
}
```

```
public aspect NbObjet {
    private List<Compte> listComptes = new ArrayList<>();

    pointcut instatiation():call(Compte.new(..));

    Compte around() : instatiation() {
        System.out.println("************************");
        System.out.println("Avant instanciation");
        if (listComptes.size() < 3) {
            Compte p = (Compte) proceed();
            System.out.println("Apres instanciation");
            p.setCode(listComptes.size() + 1);
            listComptes.add(p);
            System.out.println("initialisation de l'instance");
            System.out.println("************************");
            return p;
        } else {
            throw new RuntimeException("Nombre d'instances depassees");
        }
    }
}
```


Programmation par Aspect

Ainsi

```
new *
public aspect SecurityAspect {
    private String login;
    private String pass;
    private Scanner clavier;

    pointcut security():execution(* *.main(..));

    void around() : security() {
        if (login == null) {
            clavier = new Scanner(System.in);
            System.out.print("Login:");
            String l = clavier.next();
            System.out.print("Pass:");
            String p = clavier.next();
            if (l.equals("root") && p.equals("root")) {
                login = l;
                pass = p;
                proceed();
            } else
                System.out.println("Accés refusé");
        }
    }
}
```

Ou encore mieux, ainsi

```
@Configuration
@EnableWebSecurity
public class SecurityConfiguration {

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        // @formatter:off
        http
                .authorizeHttpRequests((authorize) -> authorize
                        .anyRequest().authenticated()
                )
                .httpBasic(withDefaults())
                .formLogin(withDefaults());
        // @formatter:on
        return http.build();
    }

    // @formatter:off
    @Bean
    public UserDetailsService userDetailsService() {
        UserDetails user = User.withDefaultPasswordEncoder()
                .username("user")
                .password("password")
                .roles("USER")
                .build();
        return new InMemoryUserDetailsManager(user);
    }
```

Profiles

Environnements de déploiement

- Plusieurs environnements pour déployer une application
  - dev, QA, stage, prod, …
- Configuration spécifique pour chaque environnement
  - Niveau de log différents (trace, debug, info, warning, error, off)
  - Bases de données différentes
  - web services différentes
  - …
- ➔ la solution est proposée avec les profiles

Profiles (intro)

- Créer vos profiles

application-dev.properties | application-prod.properties

```
1  logging.level.org.springframework=trace
```

application-dev.properties | application-prod.properties

```
1  logging.level.org.springframework=info
```

- Activer le profile souhaité

application.properties | application-dev.properties | application-prod.properties

```
1  logging.level.org.springframework=debug
2  spring.profiles.active=prod
```


Spring Data - JPA

Mapping Objet Relationnel

Objet

- Diagramme de classe (DC)
- Classe
- Attribue
- Objet

Relationnel

- Modèle conceptuel de données (MCD)
- Table
- Champ
- Enregistrement

Mapping Objet Relationnel : Divergence

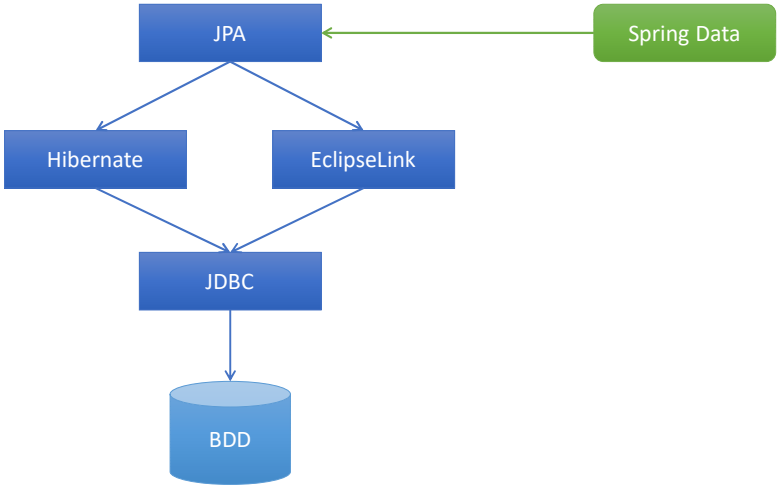
Objet

- DC pour la communication
  - Diagramme dynamique
- Relation bidirectionnelle
  - ManyToMany, ManyToOne, …
- Id implicite
  - Variable de classe mais bon
- Contrainte de domaine implicite
- Héritage
- Contrainte d'entité non déclarée

Relationnel

- Schéma pour réduire la redondance
  - Stockage seulement
- Relation unidirectionnelle
  - PK ➔ FK
- Clé primaire (PK) explicite
  - Il peut être auto incrémental
- Contrainte de domaine explicite
- Héritage non supporté
- Contrainte d'entité

Spring Data JPA



Changement du POM

```
19      <dependencies>
20          <dependency>
21              <groupId>org.springframework.boot</groupId>
22              <artifactId>spring-boot-starter-data-jpa</artifactId>
23          </dependency>
24          <dependency>
25              <groupId>com.h2database</groupId>
26              <artifactId>h2</artifactId>
27          </dependency>
28          <dependency>
29              <groupId>org.springframework.boot</groupId>
30              <artifactId>spring-boot-starter-test</artifactId>
31              <scope>test</scope>
32          </dependency>
33      </dependencies>
```

```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v2.7.5)

2022-11-02 00:27:04.752  INFO 8352 --- [           main] m.a.e.g.m.MonPremierProjetApplication
2022-11-02 00:27:04.757  INFO 8352 --- [           main] m.a.e.g.m.MonPremierProjetApplication
2022-11-02 00:27:05.632  INFO 8352 --- [           main] m.a.e.g.m.MonPremierProjetApplication

Process finished with exit code 0
```

Configuration h2

application.properties

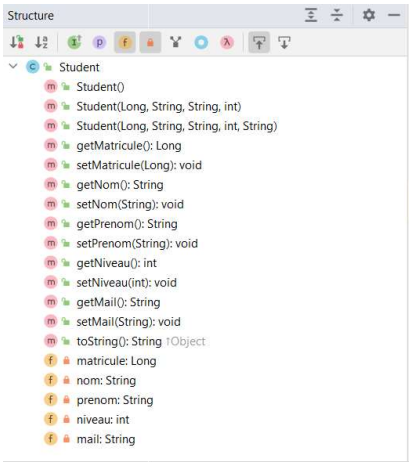
```
spring.datasource.url=jdbc:h2:mem:testdb;
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.jpa.show-sql=true
spring.h2.console.enabled=true
```


Configuration MYSQL

application.properties

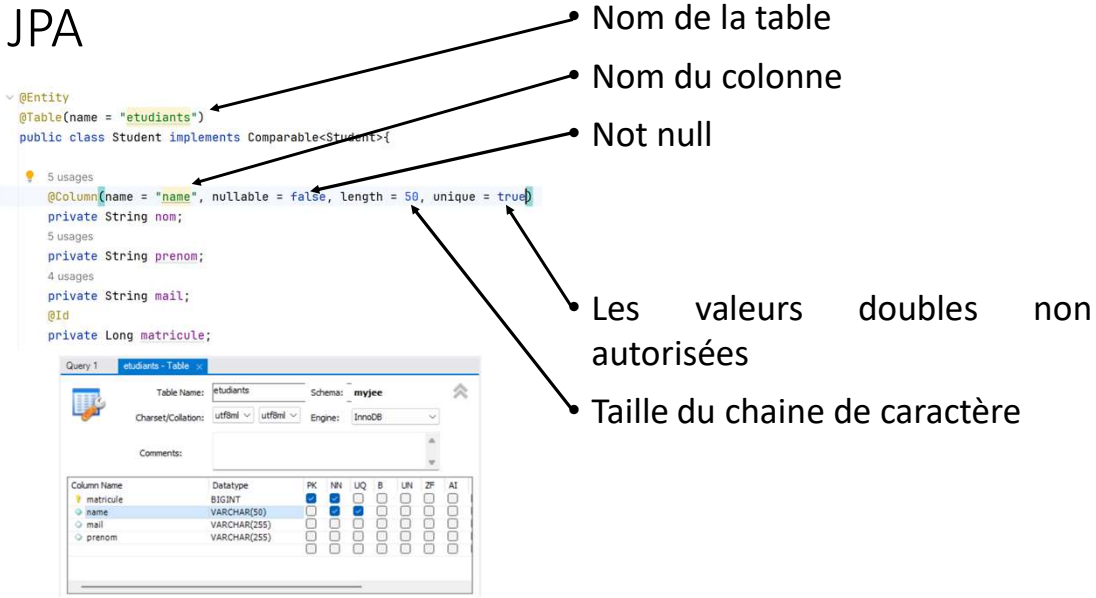
```
spring.datasource.url=jdbc:mysql://localhost:3306/myjee
spring.datasource.username=root
spring.datasource.password=toor
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.jpa.hibernate.ddl-auto=create-drop
spring.jpa.show-sql=true
```

JPA : Object/Relational Mapping



Minimum pour mapper une classe avec une table dans une base de données relationnelle

JPA

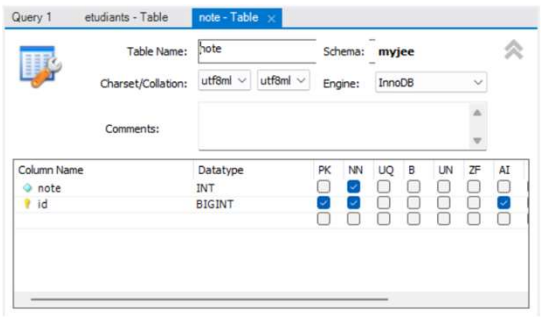


Clé primaire Auto incrémentale

```
4 usages
@Entity
public class Note {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    4 usages
    private int note;
```

```
Note n = new Note(15);
System.out.println(n);
n = noteRepository.save(n);
System.out.println(n);
```

```
2023-11-15T11:16:34.139+01:00  INFO 25484 ---
Note{id=null, note=15}
Hibernate: insert into note (note) values (?)
Note{id=1, note=15}
```



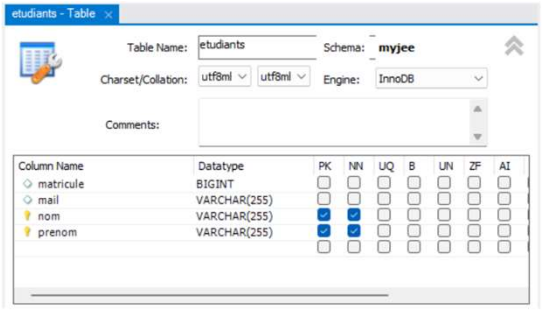
Id auto incrémentale ➔

1. Pas d'id dans le constructeur
2. La récupération de l'objet après avoir persister dans la base de données pour récupérer son id

Clé primaire composée

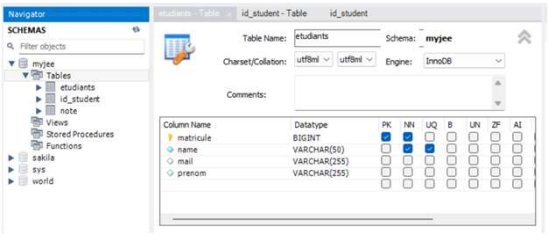
```
@Embeddable
public class StudentId implements Serializable {
    4 usages
    private String nom;
    4 usages
    private String prenom;
```

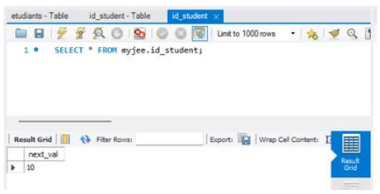
```
@Entity
@Table(name = "etudiants")
public class Student implements Comparable<Student>{

    6 usages
    @EmbeddedId
    private StudentId identifiant;
```



Clé primaire autres options

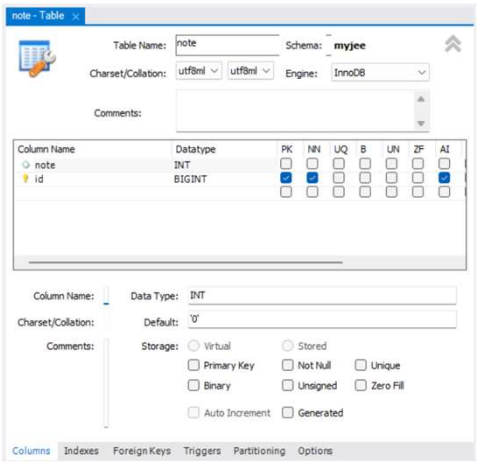
```java
@Entity
@Table(name = "etudiants")
public class Student implements Comparable<Student>{

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE,
            generator = "idStudent")
    @SequenceGenerator(name = "idStudent", initialValue = 10)
    private Long matricule;
    5 usages
    @Column(name = "name", nullable = false, length = 50, unique = true)
    private String nom;
```





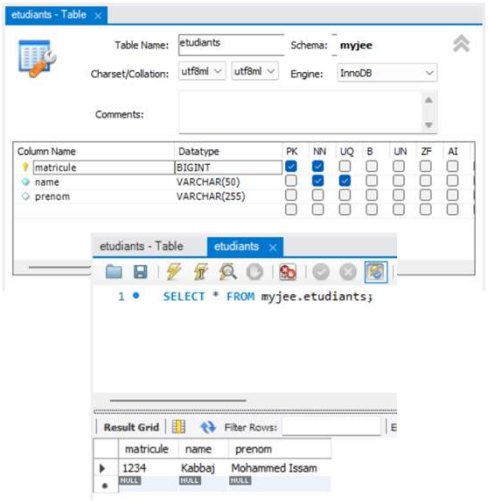
Valeur par défaut

```java
@Entity
public class Note {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    5 usages
    @Column(columnDefinition = "integer default 0")
    private int note;
```



Données non persistantes

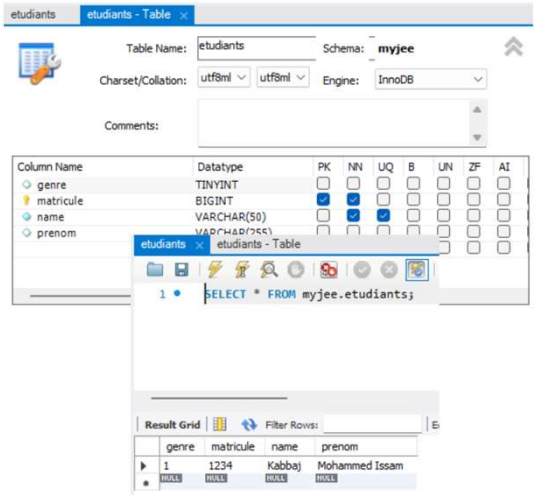
```java
@Entity
@Table(name = "etudiants")
public class Student implements Comparable<Student> {

    @Id
    private Long matricule;
    4 usages
    @Column(name = "name", nullable = false, length = 50, unique = true)
    private String nom;
    4 usages
    private String prenom;
    4 usages
    @Transient
    private String mail;
```



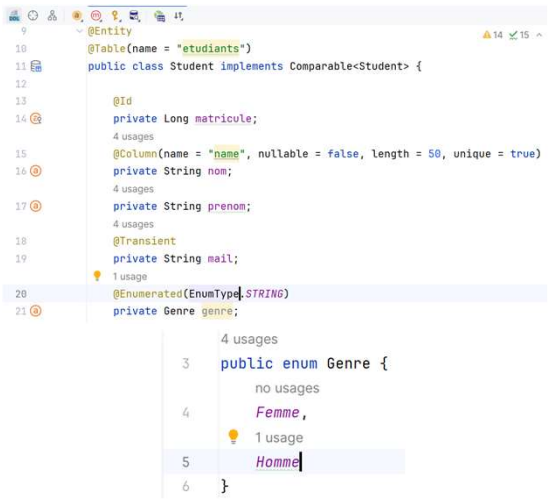
Persistante des énumérations sans annotation

```java
@Entity
@Table(name = "etudiants")
public class Student implements Comparable<Student> {

    @Id
    private Long matricule;
    4 usages
    @Column(name = "name", nullable = false, length = 50, unique = true)
    private String nom;
    4 usages
    private String prenom;
    4 usages
    @Transient
    private String mail;
    1 usage
    private Genre genre;
```

```java
4 usages
public enum Genre {
    no usages
    Femme,
    1 usage
    Homme
}
```

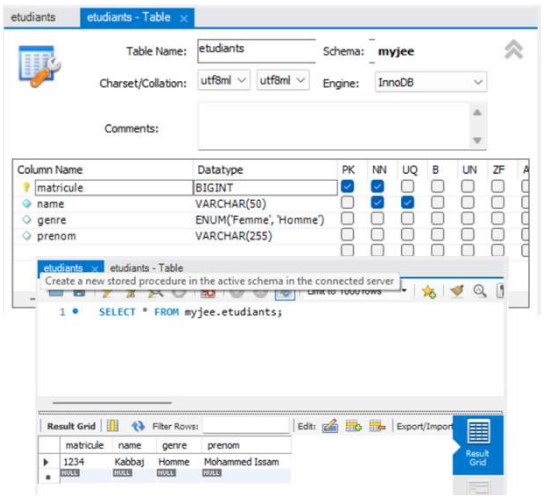


Persistante des énumérations avec annotation



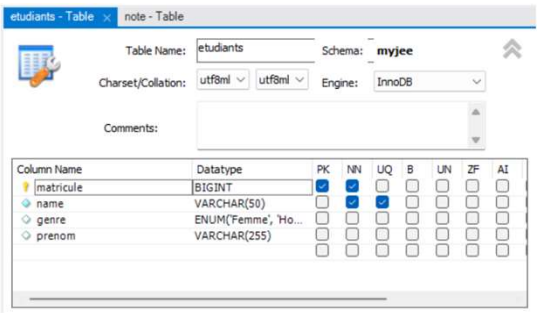
```java
@Entity
@Table(name = "etudiants")
public class Student implements Comparable<Student> {

    @Id
    private Long matricule;
    @Column(name = "name", nullable = false, length = 50, unique = true)
    private String nom;
    private String prenom;
    @Transient
    private String mail;
    @Enumerated(EnumType.STRING)
    private Genre genre;
```

```java
public enum Genre {
    Femme,
    Homme
}
```



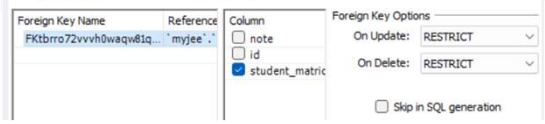
OneToOne Unidirectionnelle (1/2)

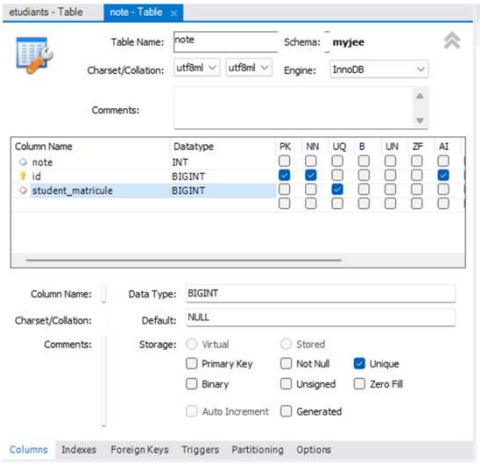
```java
@Entity
@Table(name = "etudiants")
public class Student implements Comparable<Student> {

    @Id
    private Long matricule;
    @Column(name = "name", nullable = false, length = 50, unique = true)
    private String nom;
    private String prenom;
    @Transient
    private String mail;
    private Genre genre;
```



OneToOne Unidirectionnelle (2/2)

```
5 usages
@Entity
public class Note {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    5 usages
    @Column(columnDefinition = "integer default 0")
    private int note;

    4 usages
    @OneToOne
    private Student student;
```



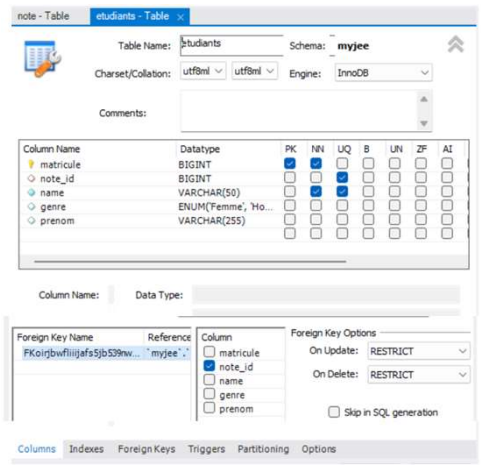


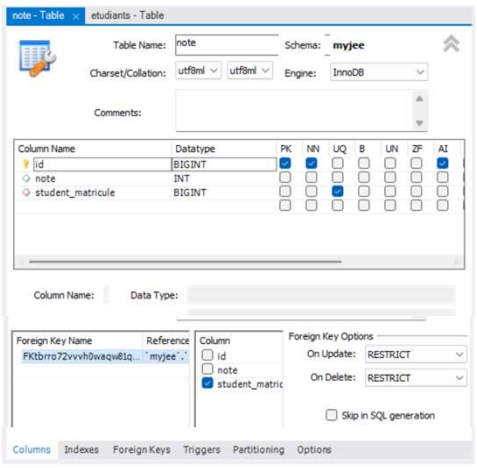
OneToOne Bidirectionnelle Erreur (1/2)

```
@Entity
@Table(name = "etudiants")
public class Student implements Comparable<Student> {

    @Id
    private Long matricule;
    4 usages
    @Column(name = "name", nullable = false, length = 50, unique = true)
    private String nom;
    4 usages
    private String prenom;
    4 usages
    @Transient
    private String mail;
    1 usage
    @Enumerated(EnumType.STRING)
    private Genre genre;
    2 usages
    @OneToOne
    private Note note;
```

```
@Entity
public class Note {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    5 usages
    @Column(columnDefinition = "integer default 0")
    private Long note;

    4 usages
    @OneToOne
    private Student student;
```

OneToOne Bidirectionnelle Erreur (2/2)



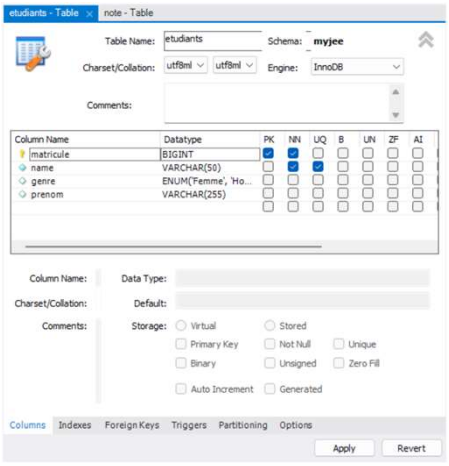


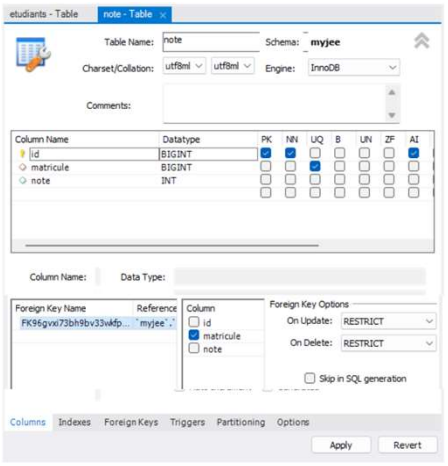
OneToOne Bidirectionnelle (1/2)

```java
@Entity
@Table(name = "etudiants")
public class Student implements Comparable<Student> {

    @Id
    private Long matricule;
    4 usages
    @Column(name = "name", nullable = false, length = 50, unique = true)
    private String nom;
    4 usages
    private String prenom;
    4 usages
    @Transient
    private String mail;
    1 usage
    @Enumerated(EnumType.STRING)
    private Genre genre;
    2 usages
    @OneToOne(mappedBy = "student")
    private Note note;
```

```java
@Entity
public class Note {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    5 usages
    @Column(columnDefinition = "integer default 0")
    private Long note;

    5 usages
    @OneToOne()
    @JoinColumn(name = "matricule") //au lieu de student_matricule
    private Student student;
```

OneToOne Bidirectionnelle (2/2)



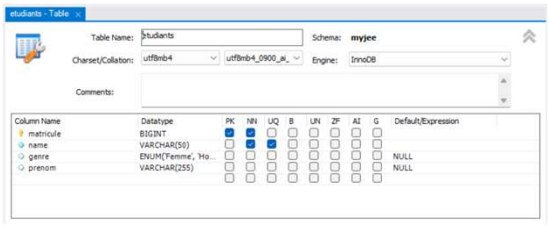


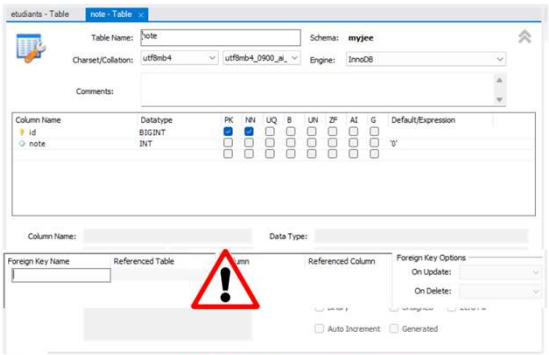
OneToOne Bidirectionnelle partage PK (1/2)

```java
@Entity
@Table(name = "etudiants")
public class Student implements Comparable<Student> {

    @Id
    private Long matricule;
    4 usages
    @Column(name = "name", nullable = false, length = 50, unique = true)
    private String nom;
    4 usages
    private String prenom;
    4 usages
    @Transient
    private String mail;
    1 usage
    @Enumerated(EnumType.STRING)
    private Genre genre;

    2 usages
    @OneToOne(mappedBy = "student")
    Note note;
```

```java
@Entity
public class Note {

    @Id
    @GeneratedValue(generator="gen")
    @GenericGenerator(name="gen", strategy="foreign",
            parameters=@Parameter(name="property", value="student"))
    private Long id;
    5 usages
    @Column(columnDefinition = "integer default 0")
    private long note;

    5 usages
    @OneToOne
    @PrimaryKeyJoinColumn
    private Student student;
```

OneToOne Bidirectionnelle partage PK (2/2)

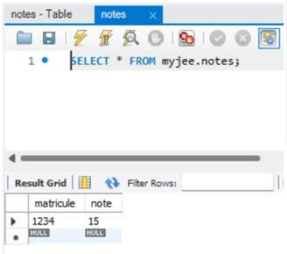


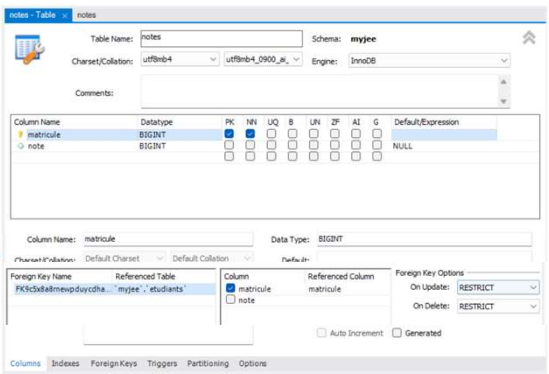



```
2023-11-15T19:06:59.741+01:00 DEBUG 17140 --- [           main] o.s.orm.jpa.JpaTransactionManager        : Closing JPA En
Student{nom='Kabbaj', prenom='Mohammed Issam', mail='kabbaj@emi.ac.ma', matricule=1234}
Note{id=1234, note=15, student=Student{nom='Kabbaj', prenom='Mohammed Issam', mail='kabbaj@emi.ac.ma', matricule=1234}}
```


Entity en deux Tables

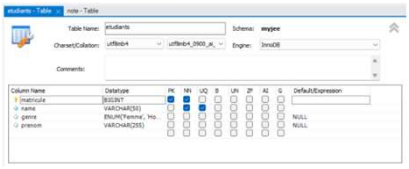
```java
@Entity
@Table(name = "etudiants")
@SecondaryTable(name="notes", pkJoinColumns = @PrimaryKeyJoinColumn(name = "matricule") )
public class Student implements Comparable<Student> {

    @Id
    private Long matricule;
    4 usages
    @Column(name = "name", nullable = false, length = 50, unique = true)
    private String nom;
    4 usages
    private String prenom;
    1 usage
    @Enumerated(EnumType.STRING)
    private Genre genre;

    2 usages
    @Column(name = "note", table = "notes")
    private long note;
```



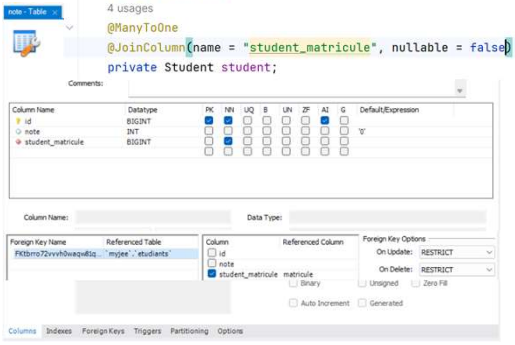


ManyToOne Unidirectionnelle

```java
@Entity
@Table(name = "etudiants")
public class Student implements Comparable<Student> {

    @Id
    private Long matricule;
    4 usages
    @Column(name = "name", nullable = false, length = 50, unique = true)
    private String nom;
    4 usages
    private String prenom;
    4 usages
    @Transient
    private String mail;
    1 usage
    @Enumerated(EnumType.STRING)
    private Genre genre;
```



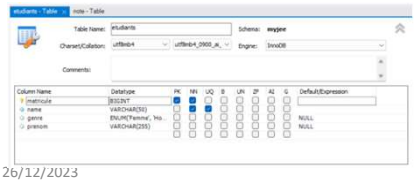
```java
@Entity
public class Note {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    4 usages
    @Column(columnDefinition = "integer default 0")
    private long note;
    4 usages
    @ManyToOne
    @JoinColumn(name = "student_matricule", nullable = false)
    private Student student;
```



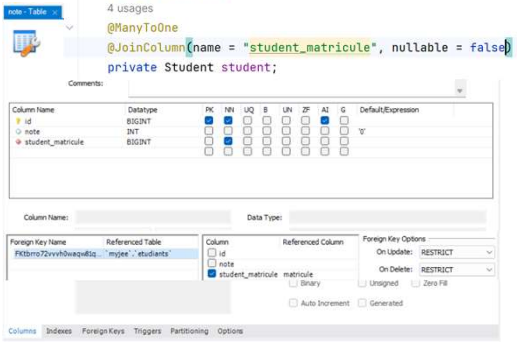
OneToMany Unidirectionnelle

ManyToOne bidirectionnelle

```java
@Entity
@Table(name = "etudiants")
public class Student implements Comparable<Student> {

    @Id
    private Long matricule;
    4 usages
    @Column(name = "name", nullable = false, length = 50, unique = true)
    private String nom;
    4 usages
    private String prenom;
    4 usages
    @Transient
    private String mail;
    1 usage
    @Enumerated(EnumType.STRING)
    private Genre genre;
    6 usages
    @OneToMany(mappedBy = "student", fetch = FetchType.EAGER)
    private List<Note> notes = new ArrayList<>();
```



```java
@Entity
public class Note {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    4 usages
    @Column(columnDefinition = "integer default 0")
    private long note;
    4 usages
    @ManyToOne
    @JoinColumn(name = "student_matricule", nullable = false)
    private Student student;
```

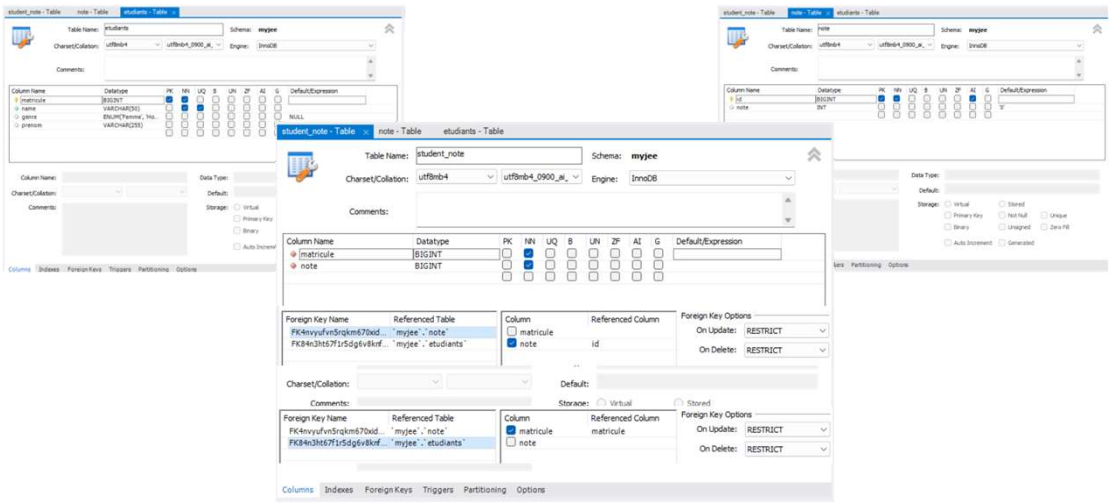


ManyToMany (1/2)

```java
@Entity
@Table(name = "etudiants")
public class Student implements Comparable<Student> {

    @Id
    private Long matricule;
    4 usages
    @Column(name = "name", nullable = false, length = 50, unique = true)
    private String nom;
    4 usages
    private String prenom;
    4 usages
    @Transient
    private String mail;
    1 usage
    @Enumerated(EnumType.STRING)
    private Genre genre;
    6 usages
    @ManyToMany(mappedBy = "students", fetch = FetchType.EAGER)
    private List<Note> notes = new ArrayList<>();
```

```java
@Entity
public class Note {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    4 usages
    @Column(columnDefinition = "integer default 0")
    private long note;
    8 usages
    @ManyToMany(fetch = FetchType.EAGER)
    @JoinTable(
            name = "student_note",
            joinColumns = @JoinColumn(name = "note"),
            inverseJoinColumns = @JoinColumn(name = "matricule"))
    private List<Student> students = new ArrayList<>();
```

ManyToMany (2/2)



Classe d'association(1/2)

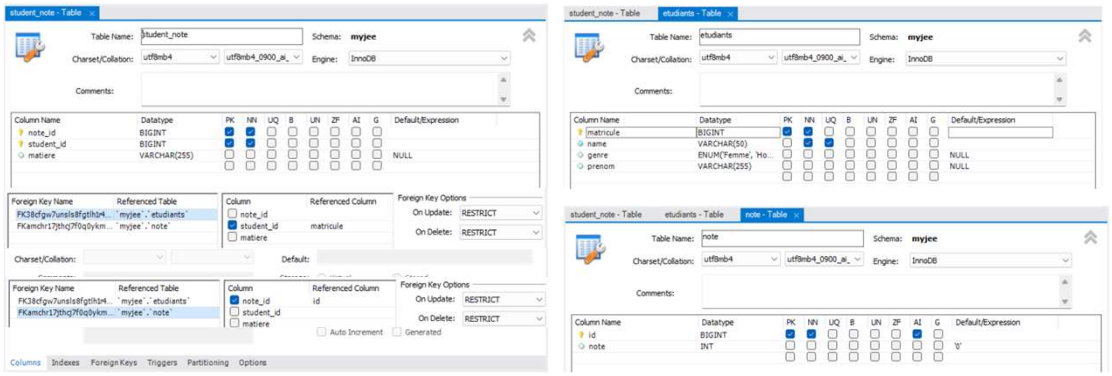
```java
@Entity
public class StudentNote {
    4 usages
    @EmbeddedId
    private StudentNotePK id;

    5 usages
    @ManyToOne
    @MapsId("student_id")
    @JoinColumn(name = "student_id")
    private Student student;
    5 usages
    @ManyToOne
    @MapsId("note_id")
    @JoinColumn(name = "note_id")
    private Note note;

    4 usages
    private String matiere;

    2 usages
    public StudentNote(Student student, Note note, String matiere) {
        this.id = new StudentNotePK(student.getMatricule(), note.getId());
        this.student = student;
        this.note = note;
        this.matiere = matiere;
    }
```

```java
@Embeddable
public class StudentNotePK implements Serializable {
    3 usages
    @Column(name = "student_id")
    Long studentId;
    3 usages
    @Column(name = "note_id")
    Long noteId;
```

```java
@Entity
@Table(name = "etudiants")
public class Student implements Comparable<Student> {

    @Id
    private Long matricule;
    4 usages
    @Column(name = "name", nullable = false, length = 50, unique = true)
    private String nom;
    4 usages
    private String prenom;
    4 usages
    @Transient
    private String mail;
    1 usage
    @Enumerated(EnumType.STRING)
    private Genre genre;
    6 usages
    @OneToMany(mappedBy = "student", fetch = FetchType.EAGER)
    private List<StudentNote> notes = new ArrayList<>();
```

```java
@Entity
public class Note {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    4 usages
    @Column(columnDefinition = "integer default 0")
    private long note;
    6 usages
    @OneToMany(mappedBy = "note", fetch = FetchType.EAGER)
    private List<StudentNote> students = new ArrayList<>();
```

Classe d'association(1/2)



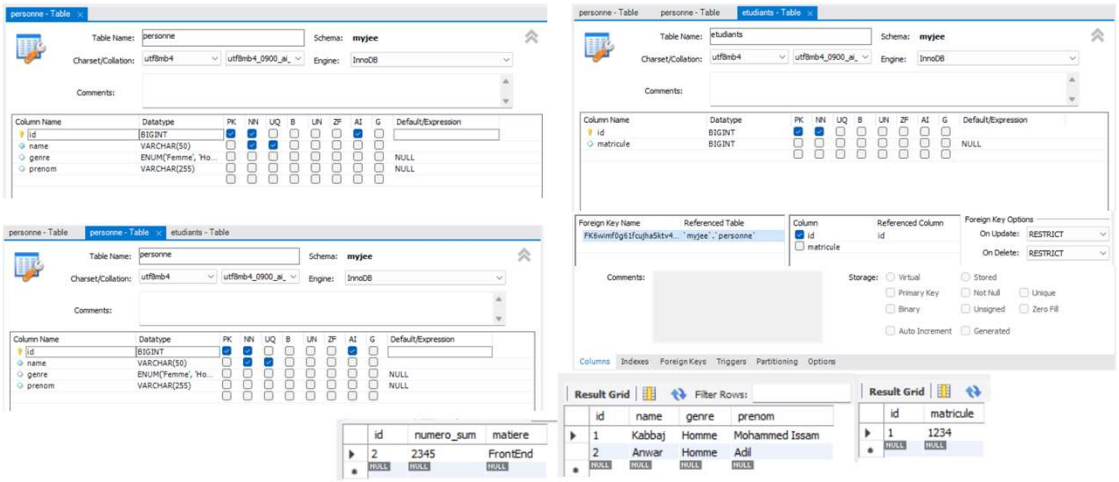
Héritage par jointure des tables

```java
@Entity
@Inheritance(strategy = InheritanceType.JOINED)
public class Personne {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    4 usages
    @Column(name = "name", nullable = false, length = 50, unique = true)
    private String nom;
    4 usages
    private String prenom;
    1 usage
    @Enumerated(EnumType.STRING)
    private Genre genre;
```

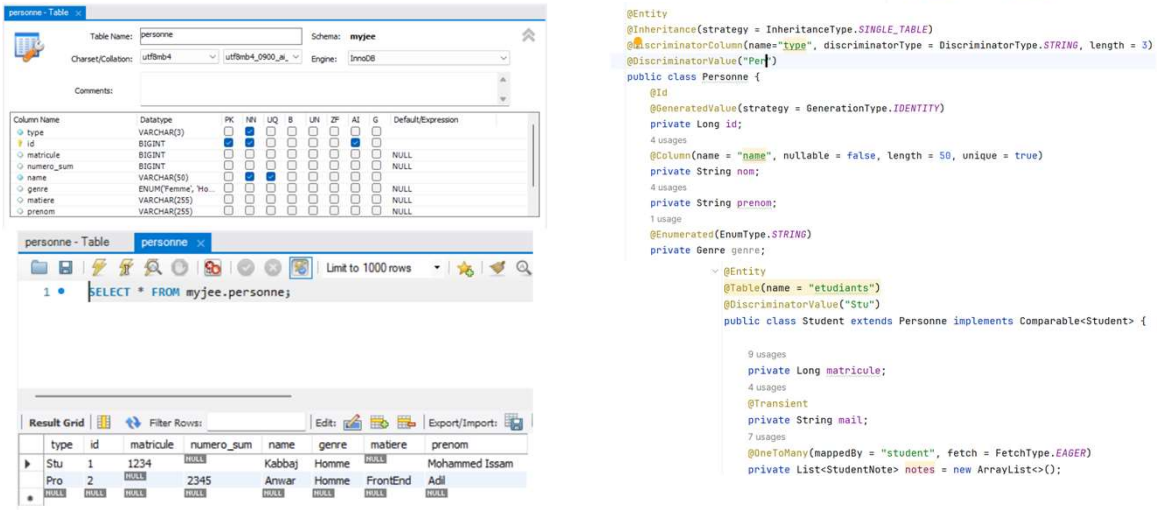
```java
@Entity
public class Professor extends Personne {
    4 usages
    private Long numeroSum;
    4 usages
    private String matiere;
```

```java
@Entity
@Table(name = "etudiants")
public class Student extends Personne implements Comparable<Student> {

    9 usages
    private Long matricule;
    4 usages
    @Transient
    private String mail;
    7 usages
    @OneToMany(mappedBy = "student", fetch = FetchType.EAGER)
    private List<StudentNote> notes = new ArrayList<>();
```

Héritage par jointure des tables



Héritage par une Table



```
@Entity
@DiscriminatorValue("Pro")
public class Professor extends Personne {
    4 usages
    private Long numeroSum;
    4 usages
    private String matiere;
```

```
@Entity
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="type", discriminatorType = DiscriminatorType.STRING, length = 3)
@DiscriminatorValue("Per")
public class Personne {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    4 usages
    @Column(name = "name", nullable = false, length = 50, unique = true)
    private String nom;
    4 usages
    private String prenom;
    1 usage
    @Enumerated(EnumType.STRING)
    private Genre genre;
```

```
@Entity
@Table(name = "etudiants")
@DiscriminatorValue("Stu")
public class Student extends Personne implements Comparable<Student> {

    9 usages
    private Long matricule;
    4 usages
    @Transient
    private String mail;
    7 usages
    @OneToMany(mappedBy = "student", fetch = FetchType.EAGER)
    private List<StudentNote> notes = new ArrayList<>();
```

Héritage par plusieurs tables (1/2)

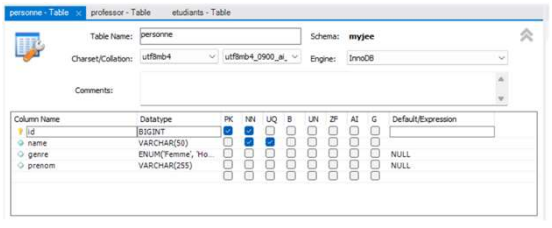
```java
4 usages  2 inheritors
@Entity
@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
public class Personne {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    4 usages
    @Column(name = "name", nullable = false, length = 50, unique = true)
    private String nom;
    4 usages
    private String prenom;
    1 usage
    @Enumerated(EnumType.STRING)
    private Genre genre;
```

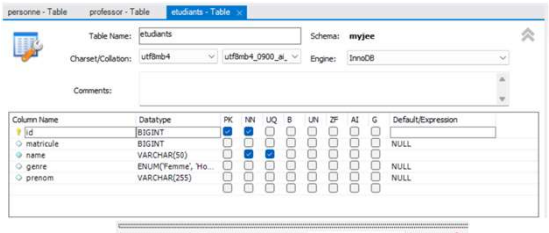
```java
@Entity
@Table(name = "etudiants")
public class Student extends Personne implements Comparable<Student> {

    9 usages
    private Long matricule;
    4 usages
    @Transient
    private String mail;
    7 usages
    @OneToMany(mappedBy = "student", fetch = FetchType.EAGER)
    private List<StudentNote> notes = new ArrayList<>();
```

```java
@Entity
public class Professor extends Personne {
    4 usages
    private Long numeroSum;
    4 usages
    private String matiere;
```

Clé primaire ne doit pas être auto générer par IDENTITY

Héritage par plusieurs tables (2/2)



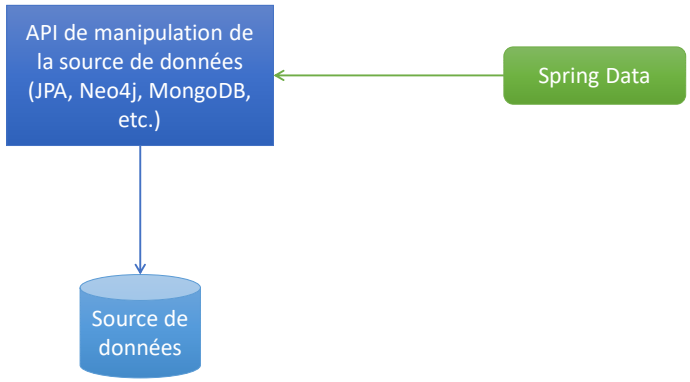


Spring Data

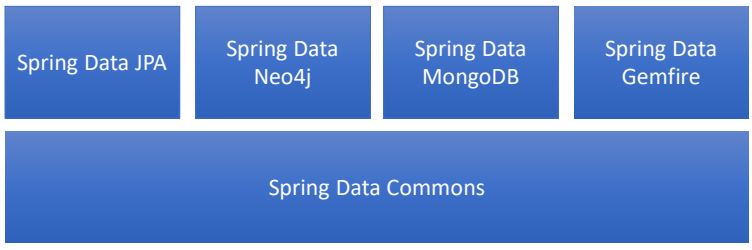
Spring Data - Introduction

- Module Spring
- Son but :
  - Faciliter l'écriture des couches d'accès aux données.
  - Tenter d'offrir une abstraction commune pour l'accès aux données quelques soient les sources de données sous-jacentes, tout en prenant en compte les spécificités de celles-ci
- Sources de données : JPA, Neo4j, MongoDB, GemFire, Hadoop, ElasticSearch, REST, Redis, Couchbase, …

Spring Data



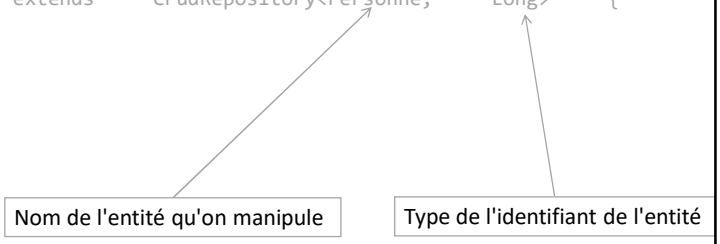
Spring Data



Spring Data JPA

- Principe de base
- Pour écrire le DAO pour un type d'entités, il faut étendre certaines interfaces fournies par Spring Data :

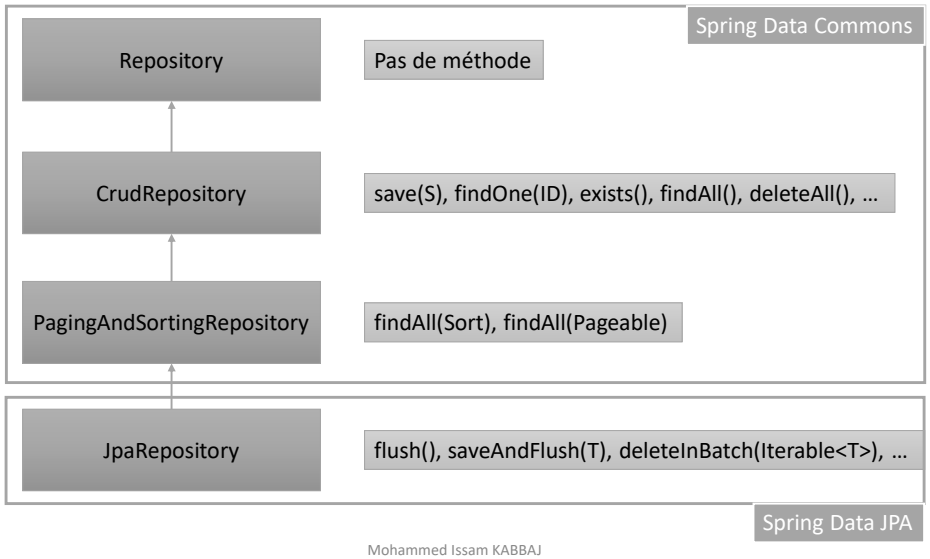
```
public interface PersonneDao extends CrudRepository<Personne, Long> {

}
```



Spring Data JPA

- Les différentes interfaces :
  - **Repository** : vous ne ferez pas grand-chose avec, hormis les méthodes que vous ajouterez
  - **CrudRepository** : vous aurez des fonctionnalités CRUD de base
  - **PagingAndSortingRepository** : vous aurez en plus des méthodes pour la pagination et le tri
  - **JpaRepository** : vous aurez en plus des méthodes propres à JPA

Spring Data JPA



Spring Data JPA

- Vous pouvez aussi écrire une requête juste avec le nom de la méthode
- Exemple :

```
public interface RequetesPersonnaliseesDao extends CrudRepository<Personne, Long> {
   public Personne findByNom(String nom);
   public Personne findByNomOrPrenom(String nom, String prenom);
   public Personne findByVelo(Velo velo);
   public Personne findByVeloAndNom(Velo velo, String nom);
   List<Person> findDistinctPeopleByLastnameOrFirstname(String lastname, String firstname);
   List<Person> findByLastnameAndFirstnameAllIgnoreCase(String lastname, String firstname);
   List<Person> findByLastnameOrderByFirstnameAsc(String lastname);
…}
```

Spring Data JPA

- Il existe une série de mots-clés pour écrire sa requête
- (cf. annexe de la documentation de référence)

Logique	Mot-clé Spring Data
GREATER_THAN	GreaterThan, IsGreaterThan
IN	In, IsIn
IS	Is, Equals, (or no keyword)
IS_NOT_NULL	NotNull, IsNotNull
IS_NULL	Null, IsNull
LESS_THAN	LessThan, IsLessThan
LIKE	Like, IsLike
NOT	Not, IsNot
NOT_IN	NotIn, IsNotIn
NOT_LIKE	NotLike, IsNotLike
REGEX	Regex, MatchesRegex, Matches
…	…

Spring Data JPA

On peut utiliser tout un tas d'autres mots clés dans le nom des méthodes pour construire nos requêtes :

- And
- Or
- GreaterThan
- LessThan
- Like
- IsNull
- OrderBy
- ...

Spring Data JPA

- Spring va créer une requête à partir des propriétés et des mots-clés mentionnés dans le nom de la méthode
- Si on fait une recherche à partir d'une « sous-propriété », on donne à Spring le chemin vers celle-ci.
- Exemple : Si Person a une propriété Address qui a une propriété ZipCode, on peut faire :

```
List<Person> findByAddressZipCode(ZipCode zipCode);
```


Spring Data JPA

- S'il y a ambiguïté dans les propriétés, on peut mettre un « _ ». Exemple :

  Si la classe Person a un attribut addressZip et un autre address (de type Address qui contient ZipCode) :

```
List<Person> findByAddress_ZipCode(ZipCode zipCode);
```

Spring Data JPA

- Certains types de Spring sont automatiquement reconnus. Du coup, on peut ajouter des paramètres de pagination et de tri :

```
Page<User> findByLastname(String lastname, Pageable pageable);
List<User> findByLastname(String lastname, Sort sort);
List<User> findByLastname(String lastname, Pageable pageable);
```


Spring Data JPA

- Exemple :

```
public interface RequetesPersonnaliseesDao extends CrudRepository<Personne, Long> {
     public List<Personne> findByNomStartingWith(String nom, Sort ordreTri);
}
public void testRecuperationParNomEtTri() {
    String baseNom = "aaa";
    // sauvegarde des personnes avec pour nom : "baseNom"+i, avec i={0, 1, 2, 3}
    final List<Personne> listePersonnes =
     this.requetesPersonnaliseesDao.findByNomStartingWith(baseNom,new Sort(Direction.DESC, "nom"));
    for (Personne personne : listePersonnes)
    { System.out.println(personne.getNom()); }
}
```

Affiche : aaa3, aaa2, aaa1, aaa0

Spring Data JPA

- Requêtes nommées :
  - On peut les mettre dans le META-INF/orm.xml ou en annotations dans l'entité
  - Exemple :

```
@Entity
@NamedQuery(name = "User.findByEmailAddress",
query = "select u from User u where u.emailAddress = ?1")
public class User { …
}
```

  - Et dans le repositories, on ne fait que déclarer la méthode :

```
public interface UserRepository
extends JpaRepository<User, Long> {
       User findByEmailAddress(String emailAddress);
}
```


Spring Data JPA

- Vous pouvez aussi ajouter l'annotation @Query si vos noms de méthodes sont beaucoup trop longues
- Exemple :

```
@Query("from Personne p where p.nom = ?1 and p.prenom = ?2")
public Personne maRequêteAvecQueryDeRechercheParNomEtPrenom(String nom, String prenom);
```

- Ca marche aussi avec les requêtes de modification, pour lesquelles il faut l'annotation @Modifying :

```
@Query("update Personne p set p.nom = :nom where p.id = :id")
@Modifying
public int metAJourNom(@Param("nom") String nom,  @Param("id") Long id);
```

- On peut nommer les paramètres avec @Param

Spring Data JPA

- On peut mettre plusieurs arguments que Spring comprendra en fonction de leur déclaration dans la méthode

```
@Query("from Personne p where p.nom = :nom and p.prenom = :prenom")
public Personne maRequêteAvecQueryDeRechercheParNomEtPrenom(String nom, String prenom);
```

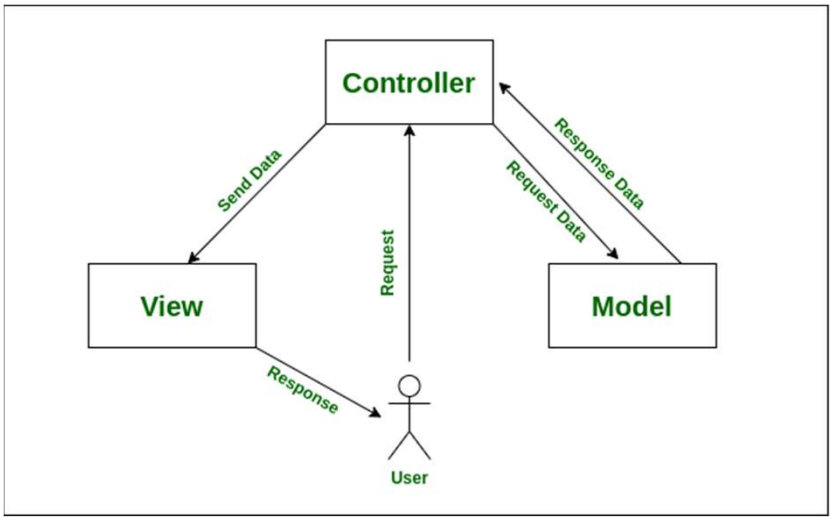
- @Query prend l'ascendant sur les requêtes nommées

Spring Data JPA

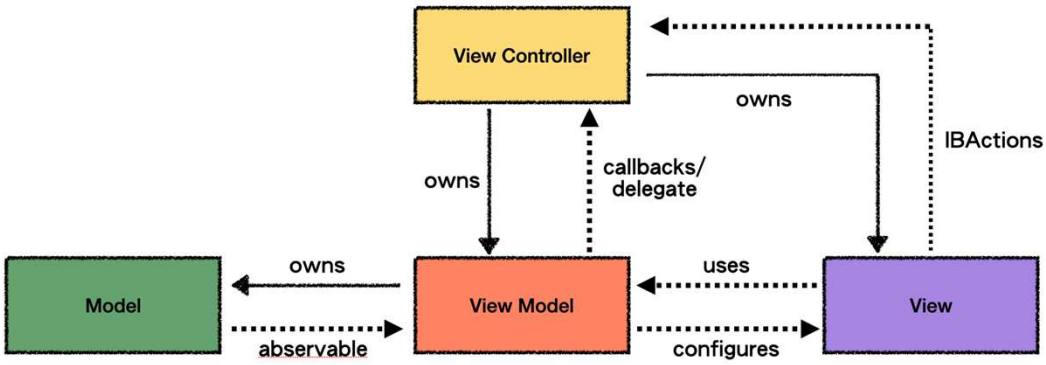
- Que ce soit pour @Query ou @NamedQuery, on peut mettre du code SQL natif (respectivement avec l'attribut nativeQuery et @NamedNativeQuery)

Spring Web

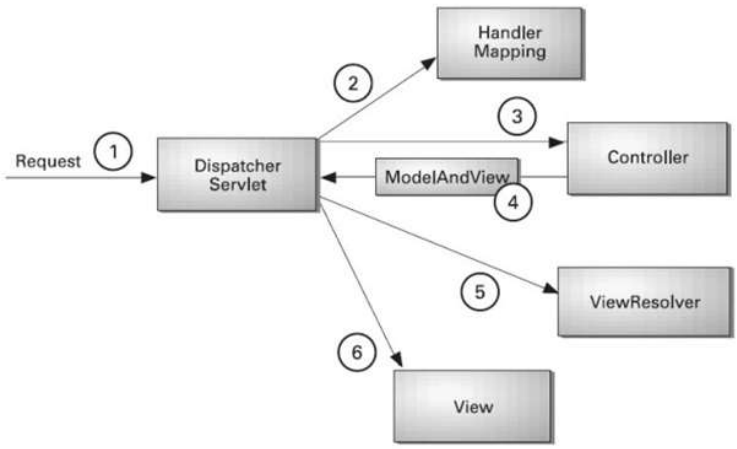
MVC



MVVM



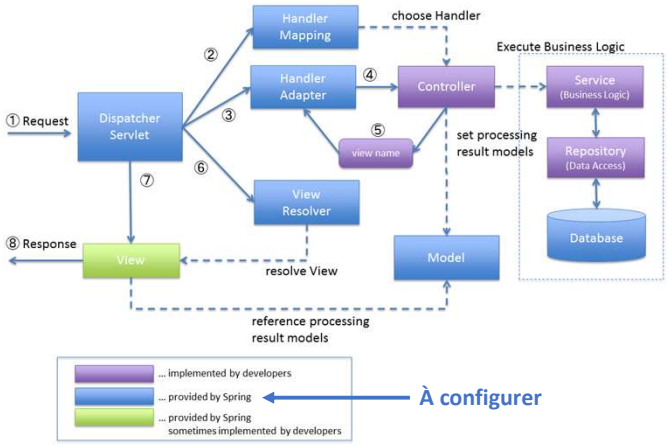
Spring MVC



Workflow Spring MVC

1. La demande sera reçue par le contrôleur frontal, c'est-à-dire DispatcherServlet.
2. DispatcherServlet transmet cette demande à HandlerMapping.
    1. HandlerMapping trouvera le contrôleur approprié pour la requête;
    2. Le HandlerMapping envoie les détails du contrôleur au DispatcherServlet.
3. Le DispatcherServlet appelle le contrôleur identifié par le HandlerMapping.
    1. Le contrôleur traitera la requête en appelant la méthode appropriée et en préparant les données.
    2. Il peut appeler une certaine logique métier ou extraire directement les données de la base de données.
4. Le contrôleur enverra ModelAndView (données du modèle et nom de la vue) à DispatcherServlet.
5. Une fois que DispatcherServlet reçoit l'objet ModelAndView, il le transmet à ViewResolver pour qu'il trouve la vue appropriée.
    1. ViewResolver identifiera la vue et la renverra à DispatcherServlet.
6. Le DispatcherServlet appellera la vue appropriée identifiée par ViewResolver.
    1. La vue crée une réponse sous forme de HTML et l'envoie au DispatcherServlet.
7. Le DispatcherServlet envoie la réponse au navigateur. Le navigateur rendra le code HTML et l'affichera à l'utilisateur final.

Spring MVC



Database as API

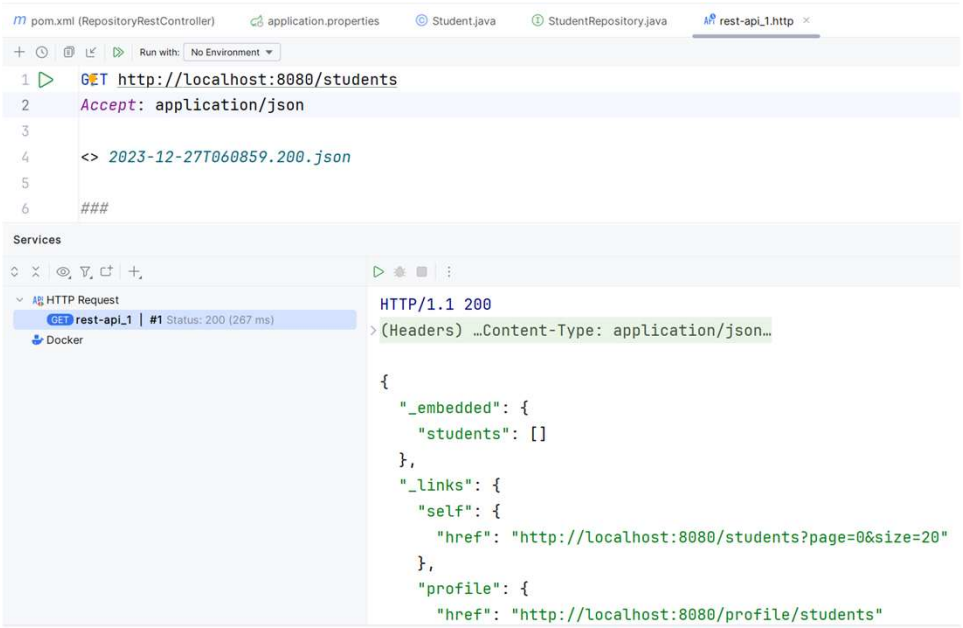
- Modifier le POM en ajoutant la dépendance suivante :

```xml
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-rest</artifactId>
</dependency>
```

- Modifier vos interfaces Repository en ajoutant l'annotation **RestController**

```
@RepositoryRestResource(collectionResourceRel = "students", path = "students")
public interface StudentRepository extends JpaRepository<Student, Long> {
}
```

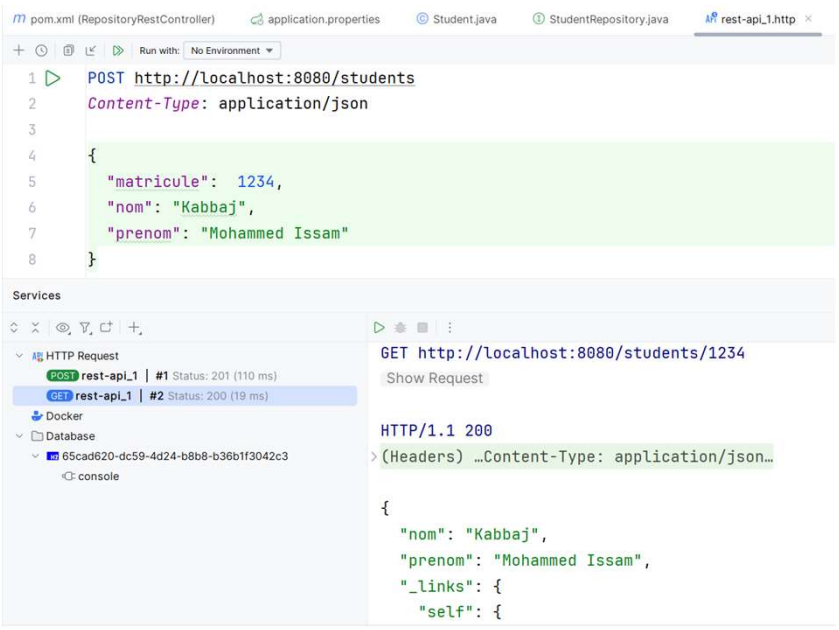

GET



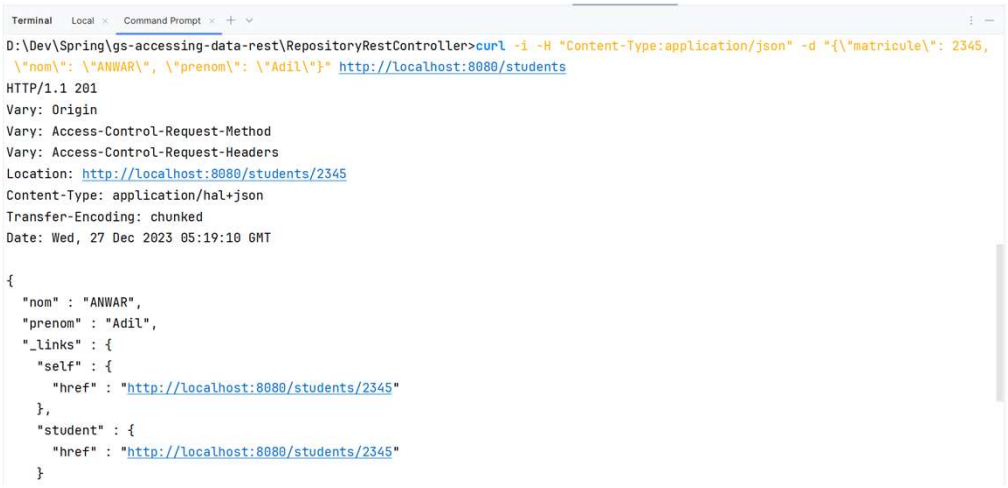
GET

```
D:\Dev\Spring\gs-accessing-data-rest\RepositoryRestController>curl http://localhost:8080/students
{
  "_embedded" : {
    "students" : [ ]
  },
  "_links" : {
    "self" : {
      "href" : "http://localhost:8080/students?page=0&size=20"
    },
    "profile" : {
      "href" : "http://localhost:8080/profile/students"
    }
  },
  "page" : {
    "size" : 20,
    "totalElements" : 0,
    "totalPages" : 0,
```


POST



POST



PUT

```
D:\Dev\Spring\gs-accessing-data-rest\RepositoryRestController>curl -X PUT -H "Content-Type:application/json" -d "{\"nom\": \"KABBAJ\", \"prenom\": \"Adil\"}" http://localhost:8080/students/1234
{
  "nom" : "KABBAJ",
  "prenom" : "Adil",
  "_links" : {
    "self" : {
      "href" : "http://localhost:8080/students/1234"
    },
    "student" : {
      "href" : "http://localhost:8080/students/1234"
    }
  }
}
```

PATCH

```
D:\Dev\Spring\gs-accessing-data-rest\RepositoryRestController>curl -X PATCH -H "Content-Type:application/json" -d "{ \"prenom\": \
"Mohammed Issam\"}" http://localhost:8080/students/2345
{
  "nom" : "ANWAR",
  "prenom" : "Mohammed Issam",
  "_links" : {
    "self" : {
      "href" : "http://localhost:8080/students/2345"
    },
    "student" : {
      "href" : "http://localhost:8080/students/2345"
    }
  }
}
```

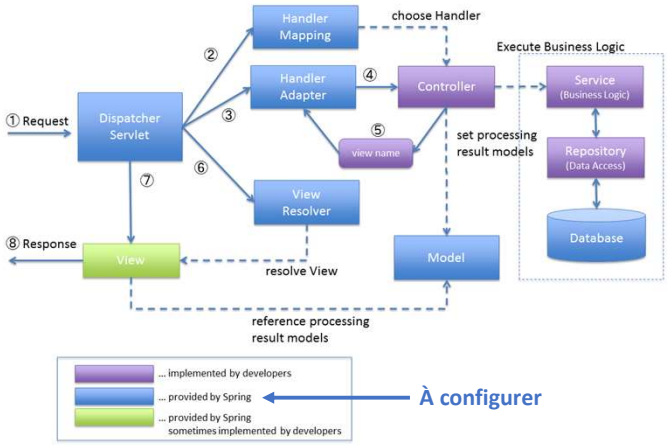

Database as API - DELETE

```
D:\Dev\Spring\gs-accessing-data-rest\RepositoryRestController>curl -X DELETE  http://localhost:8080/students/2345
{
  "nom" : "ANWAR",
  "prenom" : "Mohammed Issam",
  "_links" : {
    "self" : {
      "href" : "http://localhost:8080/students/2345"
    },
    "student" : {
      "href" : "http://localhost:8080/students/2345"
    }
  }
}
D:\Dev\Spring\gs-accessing-data-rest\RepositoryRestController>curl http://localhost:8080/students/2345
```

Spring MVC : Dependence

```
19      <dependencies> Edit Starters...
20          <dependency>
21              <groupId>org.springframework.boot</groupId>
22              <artifactId>spring-boot-starter-web</artifactId>
23          </dependency>
24
25          <dependency>
26              <groupId>org.springframework.boot</groupId>
27              <artifactId>spring-boot-starter-test</artifactId>
28              <scope>test</scope>
29          </dependency>
30      </dependencies>
```


Spring MVC



Controller

```
5   @Controller
6   public class Controlleur {
7
        no usages
8       public String Coucou(){
9           return "Hello";
10      }
11  }
```

Son HandlerMapping ?

HandlerMapping

```
8   @Controller
9   public class Controlleur {
10
11      @RequestMapping(path = "/coco", method = RequestMethod.GET)
12      public String Coucou(){
13          return "Hello";
14      }
15  }
```

ViewResolver ?

ViewResolver

Controlleur.java × pom.xml (Controller)

```
9   public class Controlleur {
10
11      @RequestMapping(value = "/coco", method = RequestMethod.GET)
12      @ResponseBody
13      public String coucou() { return "Hello"; }
16  }
```

Terminal Local × + ⌄

```
D:\Dev\Spring\Controller>curl http://localhost:8080/coco
Hello
D:\Dev\Spring\Controller>
```


Méthode

Controlleur.java × pom.xml (Controller)

```
8   @Controller
9   public class Controlleur {
10
11      @RequestMapping(value = "/coco", method = RequestMethod.GET)
12      @ResponseBody
13      public String coucou1() { return "Hello"; }
16      @RequestMapping(value = "/coco", method = RequestMethod.POST)
17      @ResponseBody
18      public String coucou2() { return "Bonjour"; }
21  }
```

Terminal Local × + ⌄

```
D:\Dev\Spring\Controller>curl http://localhost:8080/coco
Hello
D:\Dev\Spring\Controller>curl -X POST http://localhost:8080/coco
Bonjour
D:\Dev\Spring\Controller>
```

RestController

Controlleur.java × pom.xml (Controller)

```
8
9   @RestController
10  public class Controlleur {
11
12      @RequestMapping(value = "/coco", method = RequestMethod.GET)
13      public String coucou1() { return "Hello"; }
16      @RequestMapping(value = "/coco", method = RequestMethod.POST)
17      public String coucou2() { return "Bonjour"; }
20  }
```

Terminal Local × +

```
D:\Dev\Spring\Controller>curl -X POST http://localhost:8080/coco
Bonjour
D:\Dev\Spring\Controller>curl http://localhost:8080/coco
Hello
D:\Dev\Spring\Controller>
```


@XXXMapping

Controlleur.java × pom.xml (Controller)

```
6   @RestController
7   public class Controlleur {
8
9       @GetMapping(value = "/coco")
10      public String coucou1() { return "Hello"; }
13      @PostMapping(value = "/coco")
14      public String coucou2() { return "Bonjour"; }
17  }
```

Terminal Local × +

```
D:\Dev\Spring\Controller>curl http://localhost:8080/coco
Hello
D:\Dev\Spring\Controller>curl -X POST http://localhost:8080/coco
Bonjour
D:\Dev\Spring\Controller>
```

RequestMapping

Controlleur.java × pom.xml (Controller)

```
6   @RestController
7   @RequestMapping("api/v1")
8   public class Controlleur {
9
10      @GetMapping(value = "/coco")
11      public String coucou1() { return "Hello"; }
14      @PostMapping(value = "/coco")
15      public String coucou2() { return "Bonjour"; }
18  }
```

Terminal Local × +

```
D:\Dev\Spring\Controller>curl http://localhost:8080/coco
{"timestamp":"2023-12-27T09:48:33.233+00:00","status":404,"error":"Not Found","path":"/coco"}
D:\Dev\Spring\Controller>curl http://localhost:8080/api/v1/coco
Hello
D:\Dev\Spring\Controller>curl -X POST http://localhost:8080/api/v1/coco
Bonjour
D:\Dev\Spring\Controller>
```


Liste des étudiants

StudentController.java × pom.xml (RestController)

```
11  @RequestMapping("api/v1/students")
12  public class StudentController {
13
        11 usages
14      StudentRepository studentRepository;
15
16      StudentController(StudentRepository studentRepository) { this.studentRepository = studentRepository; }
19
20      @GetMapping("/")
21      public List<Student> studentList(){
22          return studentRepository.findAll();
23      }
```

Terminal Local × Command Prompt × +

```
D:\Dev\Spring\RestController>curl http://localhost:8080/api/v1/students/
[]
D:\Dev\Spring\RestController>
```

RequestBody

PathVariable

PathVariable

StudentController.java × pom.xml (RestController)

```
50      @GetMapping("/{nom}/{prenom}")
51      public List<Student> student(@PathVariable String nom, @PathVariable String prenom){
52          return studentRepository.findByNomAndPrenom(nom, prenom);
53      }
```

Terminal Local × Command Prompt × +

```
D:\Dev\Spring\RestController>curl http://localhost:8080/api/v1/students/ANWAR/Adil
[{"matricule":2345,"nom":"ANWAR","prenom":"Adil"}]
D:\Dev\Spring\RestController>curl http://localhost:8080/api/v1/students/KABBAJ/Mohammed%20Issam
[{"matricule":1234,"nom":"KABBAJ","prenom":"Mohammed Issam"}]
D:\Dev\Spring\RestController>
```


RequestParam

StudentController.java × pom.xml (RestController)

```
55      @GetMapping("/search")
56      public List<Student> student(@RequestParam String nom) { return studentRepository.findByNom(nom); }
59
```

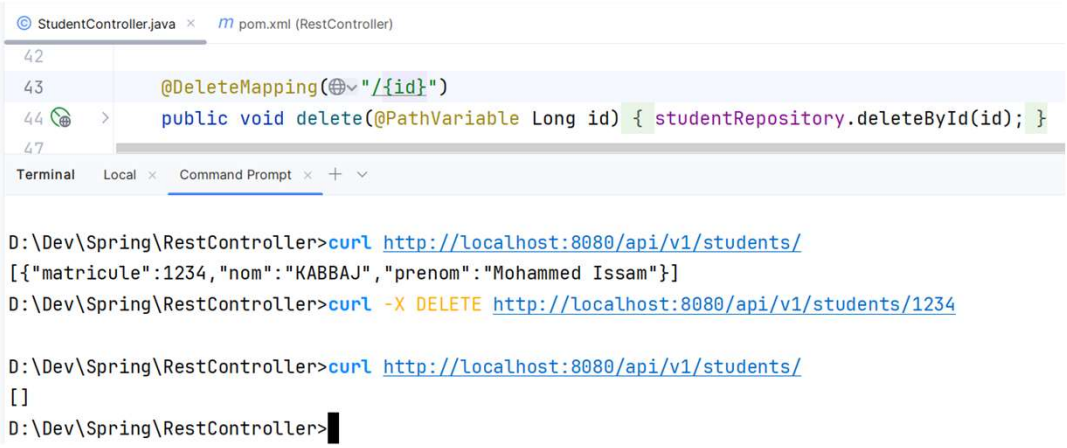
Terminal Local × Command Prompt × +

```
D:\Dev\Spring\RestController>curl http://localhost:8080/api/v1/students/search
{"timestamp":"2023-12-27T08:49:59.607+00:00","status":400,"error":"Bad Request","path":"/api/v1/students/search"}
D:\Dev\Spring\RestController>curl http://localhost:8080/api/v1/students/search?nom=KABBAJ
[{"matricule":1234,"nom":"KABBAJ","prenom":"Mohammed Issam"}]
D:\Dev\Spring\RestController>
```

Put


StudentController.java
pom.xml (RestController)
@PutMapping("/{matricule}")
public void studentList(@RequestBody Student s, @PathVariable Long matricule){
studentRepository.findById(matricule)
.map(student -> {
student.setNom(s.getNom());
student.setPrenom(s.getPrenom());
return studentRepository.save(student);
})
.orElseGet(()-> studentRepository.save(s));
}
Terminal
Local
Command Prompt
D:\Dev\Spring\RestController>curl http://localhost:8080/api/v1/students/
[{"matricule":1234,"nom":"KABBAJ","prenom":"Mohammed Issam"}]
D:\Dev\Spring\RestController>curl -X PUT http://localhost:8080/api/v1/students/1234 -H "Content-type:application/json" -d "{\"nom\
": \"KABBAJ\", \"prenom\": \"Adil\"}"
D:\Dev\Spring\RestController>curl http://localhost:8080/api/v1/students/
[{"matricule":1234,"nom":"KABBAJ","prenom":"Adil"}]
D:\Dev\Spring\RestController>

Delete


StudentController.java
pom.xml (RestController)
@DeleteMapping("/{id}")
public void delete(@PathVariable Long id) { studentRepository.deleteById(id); }
Terminal
Local
Command Prompt
D:\Dev\Spring\RestController>curl http://localhost:8080/api/v1/students/
[{"matricule":1234,"nom":"KABBAJ","prenom":"Mohammed Issam"}]
D:\Dev\Spring\RestController>curl -X DELETE http://localhost:8080/api/v1/students/1234
D:\Dev\Spring\RestController>curl http://localhost:8080/api/v1/students/
[]
D:\Dev\Spring\RestController>

HttpSession

StudentController.java × pom.xml (RestController)

Commit Alt+0

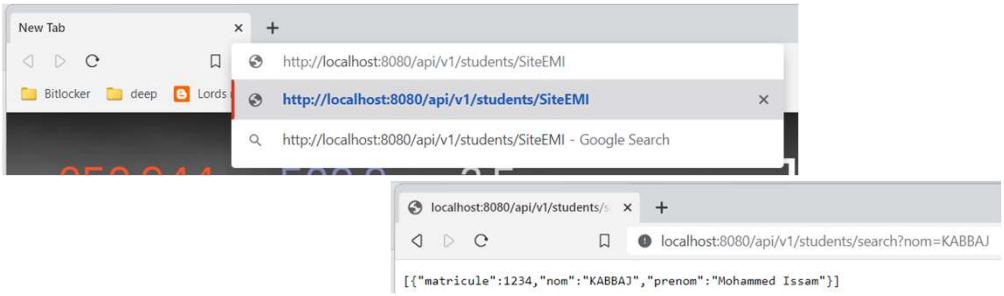
```
@GetMapping("/examen/{matricule}")
public void studentExamen(HttpSession session, @PathVariable Long matricule){
    Student s = studentRepository.findById(matricule).get();
    List<Student> students = (List<Student>) session.getAttribute( s: "sessionExamen");

    if(students == null) {
        students = new ArrayList<>();
        session.setAttribute( s: "sessionExamen", students);
    }
    students.add(s);
}

@GetMapping("/studentpassexamen")
public List<Student> studentsExamen(HttpSession session){
    return (List<Student>) session.getAttribute( s: "sessionExamen");
}
```


Session Test

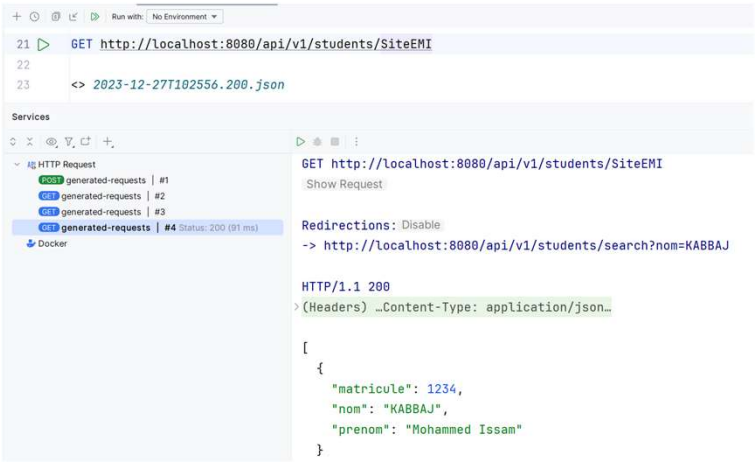
Redirection

StudentController.java × pom.xml (RestController)

```
75    @GetMapping(value = "/SiteEMI")
76    public ModelAndView emi(){
77        return new ModelAndView( viewName: "redirect:/api/v1/students/search?nom=KABBAJ");
78    }
```



Redirection



Profiles : propriétés de configuration

- Créer une classe

application-dev.properties | application-prod.properties | ServiceConfiguration.java

```
no usages
@ConfigurationProperties(prefix = "service")
@Component
class ServiceConfiguration {
    2 usages
    private String url;
    2 usages
    private String username;
    2 usages
    private String password;

    public String getUrl() {
        return url;
    }

    public void setUrl(String url) {
        this.url = url;
    }

    public String getUsername() {
```


Profiles : propriétés de configuration

- Définir les valeurs dans application.properties

application.properties | ServiceConfiguration.java

```
logging.level.org.springframework=debug
spring.profiles.active=prod
monservice.url=http://www.exemple.com/
monservice.username=defaultuser
monservice.password=servicepassword
```

application-dev.properties | ServiceConfiguration.java

```
logging.level.org.springframework=trace
monservice.url=http://localhost:9999/
monservice.username=devuser
monservice.password=mysecretpassword
```

application-prod.properties | ServiceConfiguration.java

```
logging.level.org.springframework=info
monservice.url=http://service.emi.ac.ma/
monservice.username=produser
monservice.password=complexepassword
```

Profiles : propriétés de configuration

- Utiliser les valeurs

```
2023-11-14T01:31:19.011+01:00 TRACE 202
http://localhost:9999/
devuser
mysecretpassword
2023-11-14T01:31:19.016+01:00 TRACE 202
```

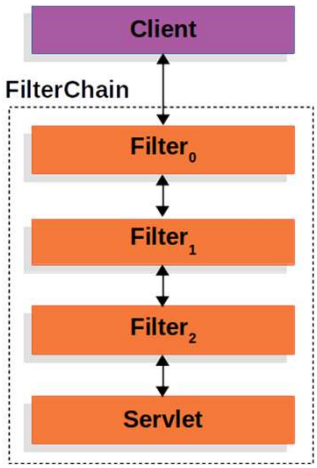
```
2023-11-14T01:29:19.430+01:00  INFO 240
http://service.emi.ac.ma/
produser
complexepassword
```

MonPremierProjetApplication.java

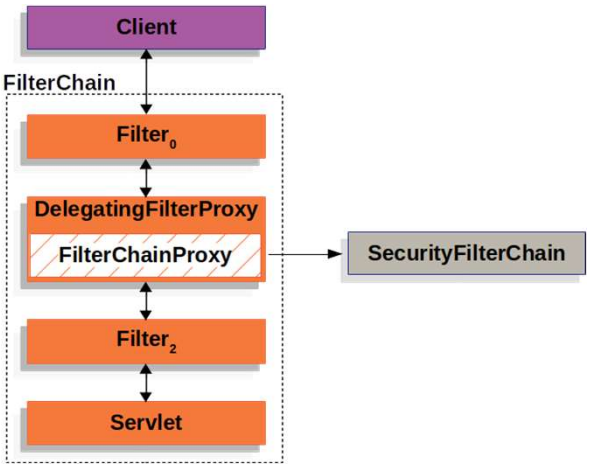
```java
package ma.ac.emi.ginfo.monpremierprojet;

import ...

@SpringBootApplication
public class MonPremierProjetApplication {

    @Autowired
    private ServiceConfiguration configuration;
    public static void main(String[] args) { SpringApplication.run(MonPremierProj

    @Bean
    public CommandLineRunner useService(){
        return args->{
            System.out.println(configuration.getUrl());
            System.out.println(configuration.getUsername());
            System.out.println(configuration.getPassword());
        };
    }
}
```


Spring Sécurité

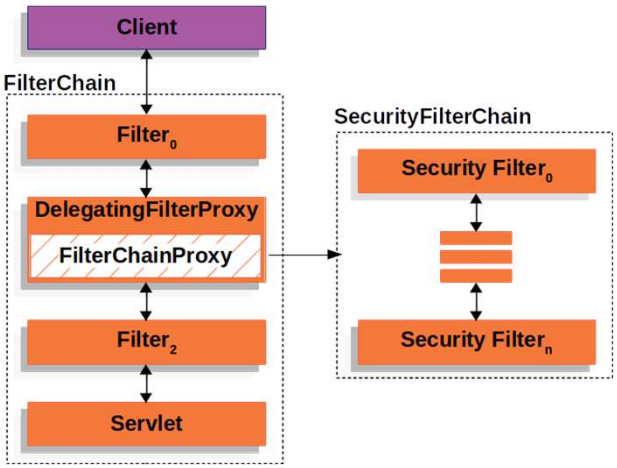
Filtres



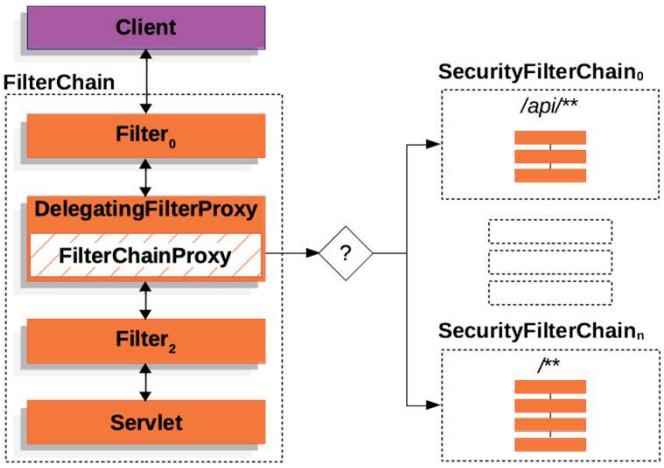
Sécurité autant que filtre



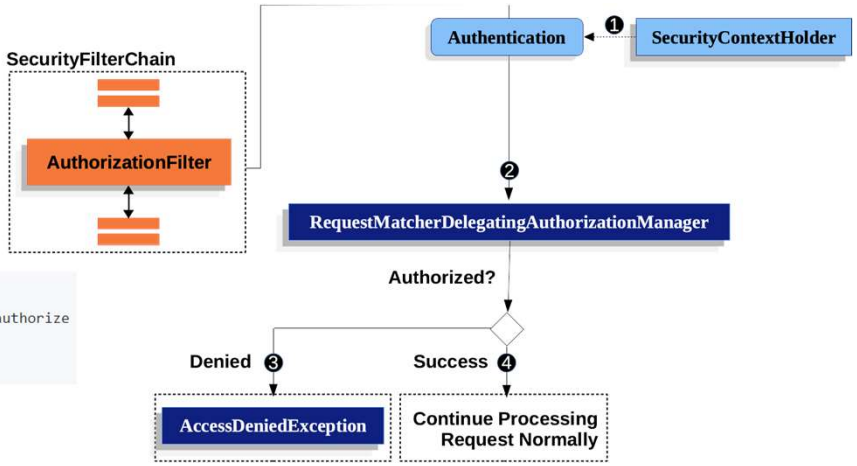
SecurityFilterChain



Multiple SecurityFilterChain



Sécurité basé sur les requêtes



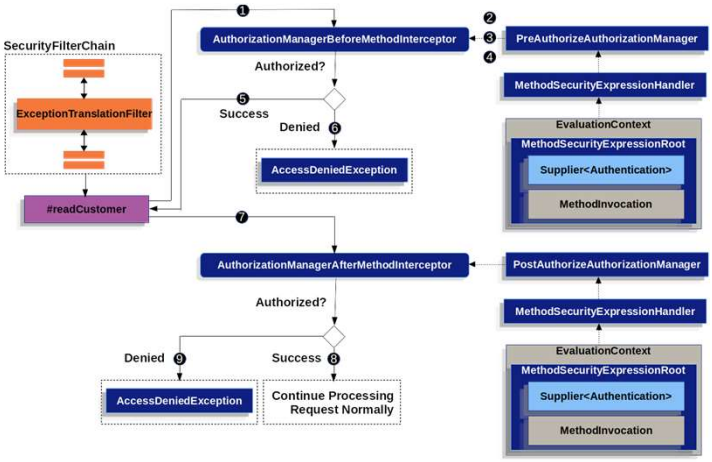

```
http
    .authorizeHttpRequests((authorize) -> authorize
        .anyRequest().authenticated()
    )
```


Example

```
@Bean
SecurityFilterChain web(HttpSecurity http) throws Exception {
    http
        // ...
        .authorizeHttpRequests(authorize -> authorize                                    ❶
            .dispatcherTypeMatchers(FORWARD, ERROR).permitAll() ❷
            .requestMatchers("/static/**", "/signup", "/about").permitAll()          ❸
            .requestMatchers("/admin/**").hasRole("ADMIN")                               ❹
            .requestMatchers("/db/**").access(allOf(hasAuthority('db'), hasRole('ADMIN')))   ❺
            .anyRequest().denyAll()                                                      ❻
        );

    return http.build();
}
```

Sécurité basé sur les méthodes

```
@Service
public class MyCustomerService {
    @PreAuthorize("hasAuthority('permission:read')")
    @PostAuthorize("returnObject.owner == authentication.name")
    public Customer readCustomer(String id) { ... }
}
```



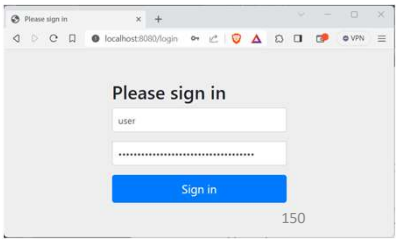
Spring security

- ajouter la dépendance suivante au pom de notre application.

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

- **Spring Boot nous fournit donc un mot de passe par défaut.**

```
 .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.0.4.RELEASE)
2018-09-10 22:44:09.059  INFO 645 --- [           main] .s.s.UserDetailsServiceAutoConfiguration :

Using generated security password: c657aef6-758a-409d-ac02-814ff4df55be
```



Spring security

- Dans Spring Boot, en ajoutant simplement le démarreur de sécurité, la configuration automatique configure un certain nombre de fonctionnalités de sécurité par défaut telles que :
  - Créez un seul utilisateur avec le nom d'utilisateur « user » et le mot de passe généré automatiquement.
  - Configure automatiquement l'authentification de base pour l'accès à l'API.
  - Configure un formulaire de connexion par défaut pour les applications Web.
  - La protection CSRF (Cross-Site Request Forgery) est activée par défaut pour empêcher les attaques CSRF.
  - Permet la surveillance et le suivi des activités liées à la sécurité dans l'application.
  - Configure la gestion de session, le délai d'expiration de session et la protection de fixation de session.
  - Les mots de passe sont automatiquement cryptés à l'aide de bcrypt par défaut.
  - Et plus.

WebSecurityConfig

```
14  @Configuration
15  @EnableWebSecurity
16  public class WebSecurityConfig {
```

Authorization

```
@Bean
public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {

    http
            .authorizeHttpRequests((authorize) -> authorize
                    .anyRequest().authenticated()
            )
            .httpBasic(withDefaults())
            .formLogin(withDefaults());
    return http.build();
}
```


Authentification

```
@Bean
public UserDetailsService users() {
    UserDetails user = User.builder()
            .username("user")
            .password("{bcrypt}$2a$10$GRLdNijSQMUvl/au9ofL.eDwmoohzzS7.rmNSJZ.0FxO/BTk76klW")
            .roles("USER")
            .build();
    UserDetails admin = User.builder()
            .username("admin")
            .password("{bcrypt}$2a$10$GRLdNijSQMUvl/au9ofL.eDwmoohzzS7.rmNSJZ.0FxO/BTk76klW")
            .roles("USER", "ADMIN")
            .build();
    return new InMemoryUserDetailsManager(user, admin);
}
```

Authentification

```java
@Bean
public InMemoryUserDetailsManager userDetailsService() {
    UserDetails user = User.withDefaultPasswordEncoder()
            .username("user")
            .password("password")
            .roles("USER")
            .build();
    return new InMemoryUserDetailsManager(user);
}
```


Login page

login.html

```html
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:th="https://www.thymeleaf.org"
      xmlns:sec="https://www.thymeleaf.org/thymeleaf-extras-springsecurity3">
    <head>
        <title>Spring Security Example </title>
    </head>
    <body>
        <div th:if="${param.error}">
            Invalid username and password.
        </div>
        <div th:if="${param.logout}">
            You have been logged out.
        </div>
        <form th:action="@{/login}" method="post">
            <div><label> User Name : <input type="text" name="username"/> </label></div>
            <div><label> Password: <input type="password" name="password"/> </label></div>
            <div><input type="submit" value="Sign In"/></div>
        </form>
    </body>
</html>
```

Endpoint security

Utilisateur Principal

hello.html

```
1  <!DOCTYPE html>
2  <html xmlns="http://www.w3.org/1999/xhtml" xmlns:th="https://www.thymeleaf.org"
3        xmlns:sec="https://www.thymeleaf.org/thymeleaf-extras-springsecurity3">
4      <head>
5          <title>Hello World!</title>
6      </head>
7      <body>
8          <h1 th:inline="text">Hello [[${#httpServletRequest.remoteUser}]]!</h1>
9          <form th:action="@{/logout}" method="post">
10             <input type="submit" value="Sign Out"/>
11         </form>
12     </body>
13 </html>
```

Utilisateur principal

Endpoint security

Test de Get

```
D:\Dev\Spring\RestController>curl http://localhost:8080/api/v1/students/

D:\Dev\Spring\RestController>curl http://localhost:8080/api/v1/students/ -u user:password
[]
```

Test de POST

```
D:\Dev\Spring\RestController>curl -i -H "Content-Type:application/json" -d "{\"matricule\": 1234, \"nom\": \"KABBAJ\", \"prenom\":
 \"Mohammed Issam\"}" http://localhost:8080/api/v1/students/
HTTP/1.1 401
Vary: Origin
Vary: Access-Control-Request-Method
Vary: Access-Control-Request-Headers
Set-Cookie: JSESSIONID=1D23FEAA0481859324E60C59F4836A6A; Path=/; HttpOnly
X-Content-Type-Options: nosniff
X-XSS-Protection: 0
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
X-Frame-Options: DENY
WWW-Authenticate: Basic realm="Realm"
Content-Length: 0
Date: Wed, 27 Dec 2023 10:38:43 GMT
```


Création Base de données

- create table users(
- username varchar_ignorecase(50) not null primary key,
- password varchar_ignorecase(500) not null,
- enabled boolean not null
-);

- create table authorities (
- username varchar_ignorecase(50) not null,
- authority varchar_ignorecase(50) not null,
- constraint fk_authorities_users foreign key(username) references users(username)
-);
- create unique index ix_auth_username on authorities (username,authority);

Déclaration de la DataSource

```
@Bean
DataSource dataSource() {
    return new EmbeddedDatabaseBuilder()
            .setType(EmbeddedDatabaseType.H2)
            .addScript(JdbcDaoImpl.DEFAULT_USER_SCHEMA_DDL_LOCATION)
            .build();
}
```


Définition de l'utilisateur

```
@Bean
UserDetailsManager users(DataSource dataSource) {
    UserDetails user = User.builder()
            .username("user")
            .password("{bcrypt}$2a$10$GRLdNijSQMUvl/au9ofL.eDwmoohzzS7.rmNSJZ.0FxO/BTk76klW")
            .roles("USER")
            .build();
    UserDetails admin = User.builder()
            .username("admin")
            .password("{bcrypt}$2a$10$GRLdNijSQMUvl/au9ofL.eDwmoohzzS7.rmNSJZ.0FxO/BTk76klW")
            .roles("USER", "ADMIN")
            .build();
    JdbcUserDetailsManager users = new JdbcUserDetailsManager(dataSource);
    users.createUser(user);
    users.createUser(admin);
    return users;
}
```

Sécurité à travers Spring DATA

```
@Bean
public SecurityEvaluationContextExtension securityEvaluationContextExtension() {
    return new SecurityEvaluationContextExtension();
}
```


Actuator

Actuator

- Spring Boot Actuator nous permet de surveiller notre application, d'obtenir des informations sur le projet, de recueillir des métriques ou simplement de vérifier l'état de notre application/base de données.
- Fournit un certain nombre de points de terminaison :
  - beans : Liste complète des beans de Spring dans votre application
  - health : Informations sur l'état de l'application
  - metrics : Métriques d'application
  - mappings : Détails sur les mappages de requêtes
- Seul health est activé par defaut
- Pour activer les autres, il faut ajouter à application.propertites

*management.endpoints.web.exposure.include=**

or

management.endpoints.web.exposure.include=health,metrics

Actuator

- localhost:8080/actuator
- http://localhost:8080/actuator/health
- http://localhost:8080/actuator/beans
- http://localhost:8080/actuator/configprops
- http://localhost:8080/actuator/env
- http://localhost:8080/actuator/metrics
- localhost:8080/actuator/metrics/http.server.requests
  - A quoi correspond les métriques affichées ?

Actuator

- Comment connaitre la version de l'application en cours d'exécution ?
- Terminaison Info vous fournie des informations importantes sur le contexte de votre application, comme il vous permet de les enrichir avec des informations personnalisées.
- Modifier le fichier de application.properties

management.info.env.enabled=true

management.info.os.enabled=true

info.app.version=1.0-Beta

- Consulter le terminaison Info
  - http://localhost:8080/actuator/info