



JPA - HIBERNATE

M2I Formations 2022



Objectifs de la formation

- Configuration de l'environnement de travail
- Approche de JPA & Hibernate
- Comprendre la gestion des états avec l'entity Manager
- Mettre en place des tests unitaires avec une base de données H2
- Approche explicative sur le cache de bas niveau
- Explications sur la gestion des transactions, et de la session
- Vue globale du contexte de persistance
- Configurer une base de données PostgreSQL
- Créer une application REST Avec un CRUD



Les points de la formation

- Mise en place de tests unitaires avec une base de donnée H2
- Gestion des états JPA
- Gestion des transactions
- Gestion de la session
- Mise en place d'une base de données PostgreSQL
- Création d'un webservice REST avec un CRUD complet



Pré-requis pour ce cours

- ▶ Avoir des connaissances en JAVA
- ▶ Avoir une approche Spring
- ▶ Avoir des connaissances en POO



A qui est destiné ce cours

- Développeurs full-stack
- Architectes

JPA & Hibernate

Persister des données



Vision globale

TEST

TESTS UNITAIRES
H2 DATABASE

CRUD ENSEMBLE

DÉVELOPPER L'APP WEB SPRING-BOOT

Mise en place d'une architecture
REST

création d'un
controller REST

`http://localhost:8080/movie` -> nous renvoyer tous les
films en BDD

`http://localhost:8080/movie/1` -> ça nous récupérera
l'entité movie en BDD avec l'identifiant 1

`http://localhost:8080/movie` -> ça fera une requête post
et ça insérera en BDD un film

CREER UNE BDD
postgresql

Connectera cette BDD à
notre projet



1.

Installation de l'environnement



Installations

- ▶ JAVA17
- ▶ POSTRESQL : Base de données
- ▶ INTELLIJ : IDE
- ▶ DBEAVER : SGBD (Système de gestion de base de données)
- ▶ PGADMIN : SGBD PostgreSQL
- ▶ ADVANCED REST CLIENT : Outil permettant d'appeler des urls REST




2.

Création d'un projet web avec Spring Boot



Spring initializr



Project

☒ Maven Project ☐ Gradle Project

Language

☒ Java ☐ Kotlin ☐ Groovy

Spring Boot

☐ 3.0.0 (SNAPSHOT) ☐ 3.0.0 (M4) ☐ 2.7.4 (SNAPSHOT) ☒ 2.7.3 ☐ 2.6.12 (SNAPSHOT) ☐ 2.6.11

Project Metadata

Group

com.example

Artifact

demo

Name

demo

Description

Demo project for Spring Boot

Package name

com.example.demo


Packaging


☐ Jar ☒ War

Java


☐ 18 ☒ 17 ☐ 11 ☐ 8

Dependencies



ADD DEPENDENCIES...  + B


H2 Database 

Provides a fast in-memory database that supports JDBC API and R2DBC access, with a small (2mb) footprint. Supports embedded and server modes as well as a browser based console application.

Spring Web 

Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.



GENERATE  + ↵

EXPLORE CTRL + SPACE

SHARE...



Ajout de dépendance

```
<dependency>  
    <groupId>org.hibernate</groupId>  
    <artifactId>hibernate-core</artifactId>  
</dependency>
```

```
<dependency>  
    <groupId>org.springframework</groupId>  
    <artifactId>spring-orm</artifactId>  
</dependency>
```



3.

Le context de persistence Hibernate



JPA VS Hibernate

- ▶ JPA : norme
- ▶ Hibernate : nous passons par l'implémentation d'hibernate pour gérer les interactions avec la BDD

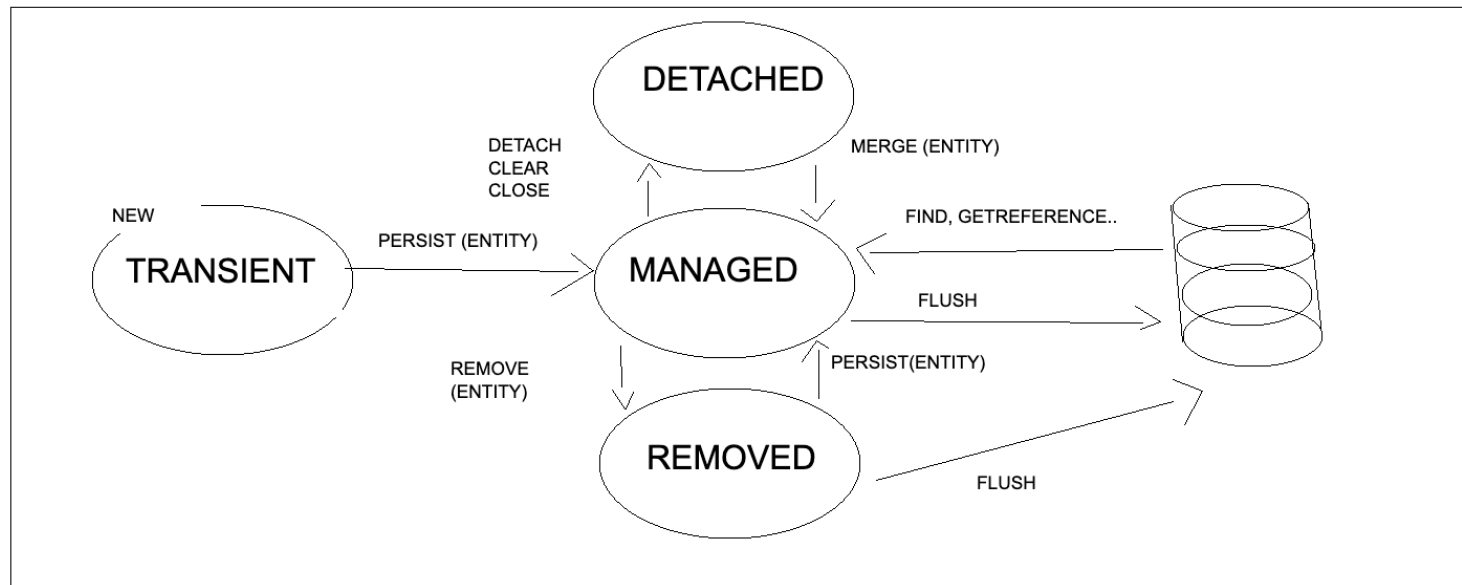


3 règles d'or

- ▶ Savoir si on est dans une session
- ▶ Savoir si on est ou non dans une transaction
- ▶ Se soucier du SQL généré par Hibernate

CYCLE DE VIE D'UNE ENTITÉ JPA

context de persistance = session
 MAP<ClassID, Entity>





Contexte de persistance

- L'ensemble des entités générées et gérées par Hibernate
- Le contexte de persistance est aussi appelée session
- C'est une sorte de map dont la clef est l'identifiant de l'entité et la valeur l'entité elle-même
- Le contexte de persistance est un ensemble d'instances pour entité persistante il y a une seule et unique instance d'entité
- Les instances des entités et leur cycle de vie sont gérés dans ce contexte de persistance

Mise en place des logs

```
<!-- PERMET D'OBSERVER LES INSTRUCTIONS SQL ENVOYÉES À LA BDD -->
<logger name="org.hibernate.SQL" level="DEBUG" />

<!-- Quand une transaction ouverte et fermée -->
<logger name="org.springframework.orm.jpa.JpaTransactionManager" level="TRACE" />

<!-- Ouverture et fermeture de session -->
<logger name="org.hibernate.internal.SessionImpl" level="TRACE" />

<!-- Permet d'afficher les logs de la classe MovieRepository -->
<logger name="com.webstart.projectStart.repository.MovieRepository" level="TRACE" />

<!-- Permet d'afficher les valeurs des paramètres sql -->
<logger name="org.hibernate.type.descriptor.sql.BasicBinder" level="TRACE" />
```

Configuration de la BDD pou nos tests avec H2 Database

```
@Bean // on est train d'ajouter au context spring entityManager
public LocalContainerEntityManagerFactoryBean entityManagerFactory() { // responsable de faire les interactions avec la BDD
    LocalContainerEntityManagerFactoryBean em = new LocalContainerEntityManagerFactoryBean();
    em.setDataSource(dataSourceH2());
    em.setPackagesToScan(new String[] { "com.webstart.projectStart.entity", "com.webstart.projectStart.converter" }); // il est

    JpaVendorAdapter vendorAdapter = new HibernateJpaVendorAdapter();
    em.setJpaVendorAdapter(vendorAdapter);
    em.setJpaProperties(additionalProperties());

    return em;
}

@Bean
public DataSource dataSourceH2() { // on va l'utiliser pour nos tests unitaires
    DriverManagerDataSource dataSource = new DriverManagerDataSource();
    dataSource.setDriverClassName("org.h2.Driver");
    dataSource.setUrl("jdbc:h2:mem:db;DB_CLOSE_DELAY=-1;DB_CLOSE_ON_EXIT=FALSE");
    dataSource.setUsername("sa");
    dataSource.setPassword("");

    return dataSource;
}

@Bean
public PlatformTransactionManager transactionManager() {
    JpaTransactionManager transactionManager = new JpaTransactionManager();
    transactionManager.setEntityManagerFactory(entityManagerFactory().getObject());

    return transactionManager;
}

private Properties additionalProperties() {
    Properties properties = new Properties();
    properties.setProperty("hibernate.hbm2ddl.auto", "create-drop"); // hibernate va créer la BDD et à chaque supprimer la supp
    properties.setProperty("hibernate.dialect", "org.hibernate.dialect.H2Dialect"); // le dialect à utiliser pour communiquer a
    return properties;
}
```



Entity Manager et ses méthodes

- ▶ Il est associé au contexte de persistance
- ▶ L'API entity manager est utilisé pour créer et supprimer des instances d'entités persistantes, ou encore pour récupérer ces entités par leur clef primaire, ou pour effectuer des requêtes sur ces entités



Initialiser des données dans la partie test du projet

```
@SqlConfig(dataSource = "dataSourceH2", transactionManager =  
"transactionManager")  
@Sql({"/data/data-test.sql"})
```

```
*.sql files are supported in other JetBrains IDEs  
1 insert into Movie (name, id, certification) values ('Training Day', -1L, 3L)  
2 insert into Movie (name, id, certification) values ('Boyz in the hood', -2L, 2L)
```



@Transactional

- ▶ Utilisé pour modifier des données en base
- ▶ Ouvrir une transaction
- ▶ Ouvrir la session
- ▶ On l'utilise sur les méthodes `persist()`, `merge()`...



JPQL - Java Persistence Query Language

- ▶ Langage de requêtes proche du SQL qui est défini par la spécification JPA

```
public List<Movie> getAll() {  
    List<Movie> movies = entityManager.createQuery("from Movie",  
Movie.class).getResultList();  
    return movies;  
}
```



La méthode `getReference()`

- La méthode `getReference()` nous renvoie un proxy
- On parle de chargement à la demande
- Cette méthode nous renvoie une référence vers l'entité mais pas ses données
- Si je ne suis pas dans une session ouverte, je peux pas récupérer ses données
- Si c'est le cas, Hibernate va se charger de récupérer les données en base à la seule et unique condition que l'entité soit à l'état managed (en session)



LazyInitialisationException

- ▶ Une des erreurs les plus fréquentes avec Hibernate
- ▶ Se produit lorsque l'on cherche à récupérer des données d'un proxy dans un endroit dans lequel on ne devrait pas le faire (lorsque mon entité n'est pas à l'état managed)
- ▶ @Transactional obligatoire lorsque l'on souhaite récupérer des données à la demande



La méthode flush()

- ▶ Force l'entity manager à envoyer les modifications en BDD



Dirty Checking

- ▶ Fonctionnalités du contexte de persistance
- ▶ Concerne que les entités managed
- ▶ Lorsqu'on récupère une entité, Hibernate la met dans sa map de session mais aussi dans une sorte de collection, une sorte de photographie de cette entité. Au moment du flush, il va comparé notre objet et la photo qu'il a pris. Si il ya des différences entre les deux, va Updater les données en BDD.
- ▶ En gros, si je récupère un entité en BDD et je modifie la valeur de ses attributs, Hibernate va l'identifié comme 'sale' si l'entité est à l'état managed et mettre à jour les données en BDD.



Le cache de premier niveau

- ▶ La session Hibernate est aussi appelée cache de premier niveau
- ▶ Si on lui demande des objets qu'il connaît déjà, il va les chercher dans son cache plutôt que d'aller re-faire une requête en base de données.
- ▶ Il peut le faire tant que la session est ouverte



@Entity

- ▶ Permet d'identifier une classe Java en entité persistante et donc de créer la table en base



@Table

- ▶ Permet de renommer le nom d'une table en BDD lors de sa création



Mapping

- ▶ Il permet lier une entité Java à une table en base
- ▶ Field Access : via les attributs
- ▶ Property Access : via les getters



Les énumérations

- ▶ L'énumération permet de définir un type avec un nombre fini de valeurs possibles. Il est possible de manipuler ce nouveau type comme une constante, comme un attribut, un paramètre et une variable.

Les énumérations

```
public enum Certification {  
  
    TOUS_PUBLIC( key: 1, description: "Tous public"),  
    INTERDITS_AU_MOINS_12( key: 2, description: "Interdits au moins de 12 ans"),  
    INTERDITS_AU_MOINS_16( key: 3, description: "Interdits au moins de 16 ans"),  
    INTERDITS_AU_MOINS_18( key: 4, description: "Interdits au moins de 18 ans");  
  
    private Integer key;  
    private String description;  
  
    private Certification(Integer key, String description) {  
        this.key = key;  
        this.description = description;  
    }  
  
    public Integer getKey() {  
        return key;  
    }  
  
    public void setKey(Integer key) {  
        this.key = key;  
    }  
  
    public String getDescription() {  
        return description;  
    }  
  
    public void setDescription(String description) {  
        this.description = description;  
    }  
}
```



Les converters

- ▶ Nous permettent de stocker le strict minimum en BDD tout en gardant une application maintenable
- ▶ Il permet par exemple de sauvegarder un chiffre entier lié à un enum en BDD
- ▶ Et de nous retourner l'enum lié à la récupération d'un integer en Base
- ▶ Les converters nous font la conversion lors d'un insert et d'un select

TP - CRUD

Créer un webservices REST

Configuration de la BDD avec PostgreSQL

```
@Bean
public LocalContainerEntityManagerFactoryBean entityManagerFactory() {
    LocalContainerEntityManagerFactoryBean em = new LocalContainerEntityManagerFactoryBean();
    em.setDataSource(dataSource());
    em.setPackagesToScan(new String[] { "com.webstart.projectStart.entity", "com.webstart.projectStart.converter" });

    JpaVendorAdapter vendorAdapter = new HibernateJpaVendorAdapter();
    em.setJpaVendorAdapter(vendorAdapter);
    em.setJpaProperties(additionalProperties());

    return em;
}

@Bean
public DataSource dataSource() {
    DriverManagerDataSource dataSource = new DriverManagerDataSource();
    dataSource.setDriverClassName("org.postgresql.Driver"); // modifier le driver
    dataSource.setUrl("jdbc:postgresql://localhost:5432/jawher"); // modifier l'url
    dataSource.setUsername("postgres"); // modifier le userName
    dataSource.setPassword("root"); // modifier le mdp

    return dataSource;
}

@Bean
public PlatformTransactionManager transactionManager() {
    JpaTransactionManager transactionManager = new JpaTransactionManager();
    transactionManager.setEntityManagerFactory(entityManagerFactory().getObject());

    return transactionManager;
}

private Properties additionalProperties() {
    Properties properties = new Properties();
    properties.setProperty("hibernate.ddl.auto", "update");
    properties.setProperty("hibernate.dialect", "org.hibernate.dialect.PostgreSQLDialect"); // modifier le dialect
    return properties;
}
```



Ajouter le driver POSTRESQL et le starter web Spring

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

```
<dependency>  
  <groupId>org.postgresql</groupId>  
  <artifactId>postgresql</artifactId>  
  <version>42.4.2</version>  
  <scope>runtime</scope>  
</dependency>
```



4.

Rappel sur les annotations Spring



Contexte spring et conteneur léger

- ▶ Un contexte d'application contient la définition des objets que le conteneur doit créer ainsi que leurs interdépendances. L'API du Spring Framework définit l'interface `ApplicationContext`.



@SpringBootApplication

- ▶ Définit classe de configuration Spring pour une application SpringBoot
- ▶ Il lance également le Scan des classes à effectuer pour la création des objets dans le context Spring



@Controller et @RestController

- ▶ Controller est responsable de fournir des données à la vue souvent via un modèle
- ▶ @Controller : permet de définir une classe comme un controller, mapper les différentes requêtes http
- ▶ @RestController : permet de définir une classe comme un controller, mapper les différentes requêtes http dans une application REST



Controller REST

- @RequestMapping : Permet de préfixer url pour les méthodes d'un controller
- @PostMapping : Permet de capter un scénario POST sur une URL
- @GetMapping : Permet de capter un scénario GET sur une URL
- @PutMapping : Permet de capter un scénario PUT sur une URL
- @DeleteMapping : Permet de capter un scénario DELETE sur une URL



@Service

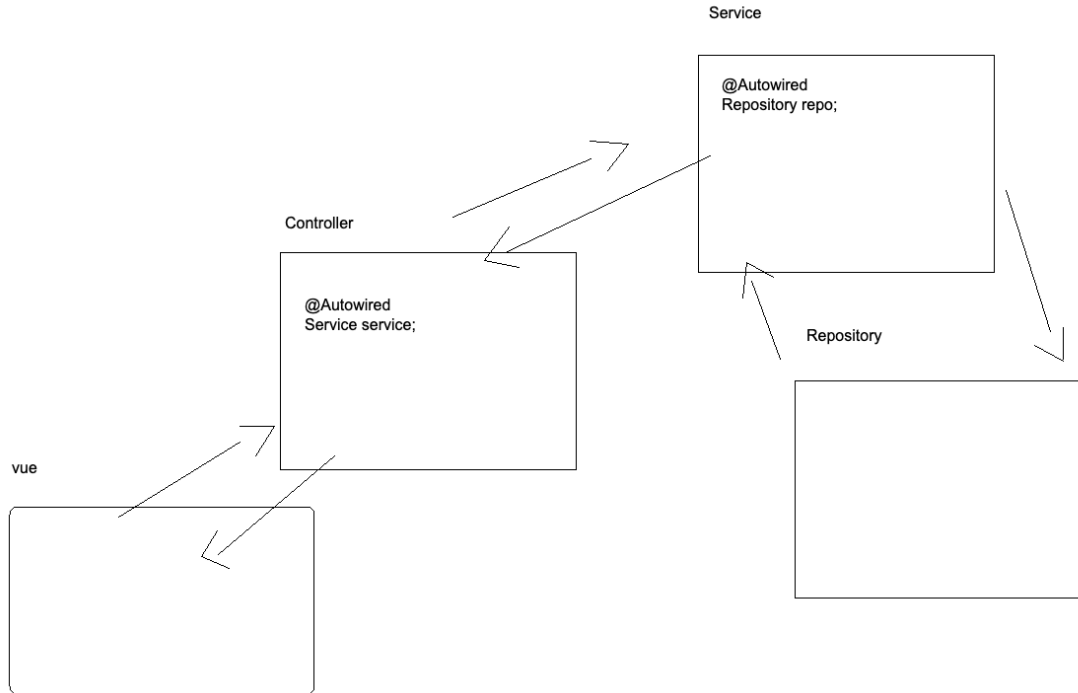
- ▶ Cette annotation permet de définir une classe comme un service dans le contexte spring
- ▶ Le service est appelé par le controller
- ▶ Le service appelle le repository



@Repository

- ▶ Cette annotation permet de définir une classe JAVA comme classe repository dans le context Spring
- ▶ Cette classe permet les interactions avec la BDD

Rappel des dépendances





Controller REST - part 2

- ▶ `@PathVariable` : permet de récupérer la valeur d'une paramètre dans une URL
- ▶ `@RequestBody` : permet de récupérer sous format json le corps d'une requête



@Autowired

- ▶ Annotation permet d'injecter une dépendance dans une classe depuis le context Spring

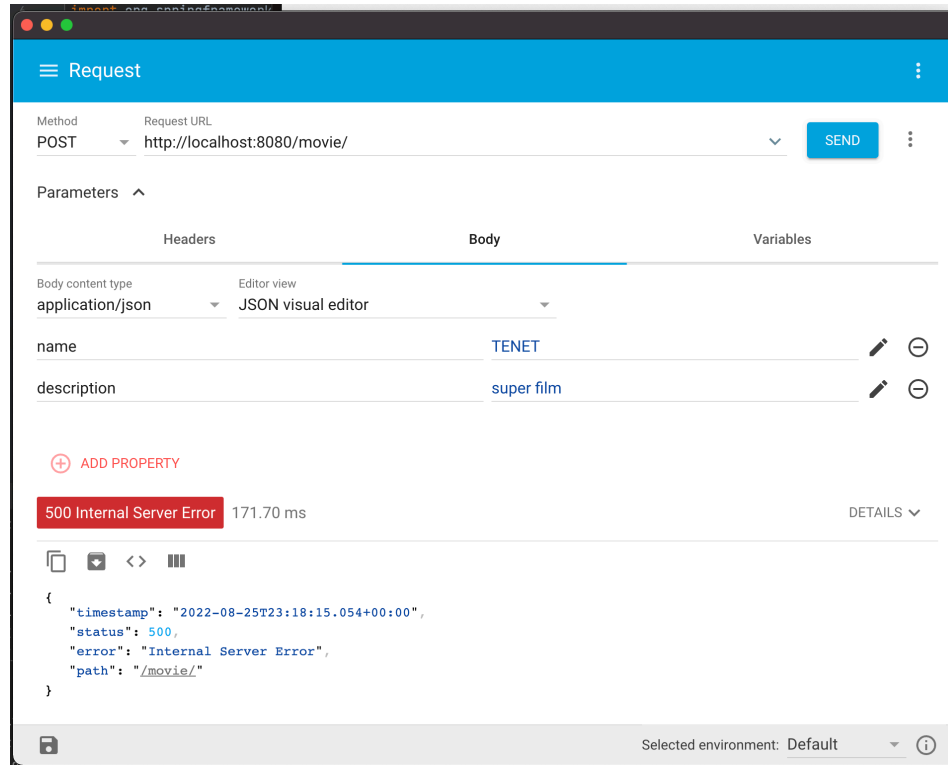


`application.properties`

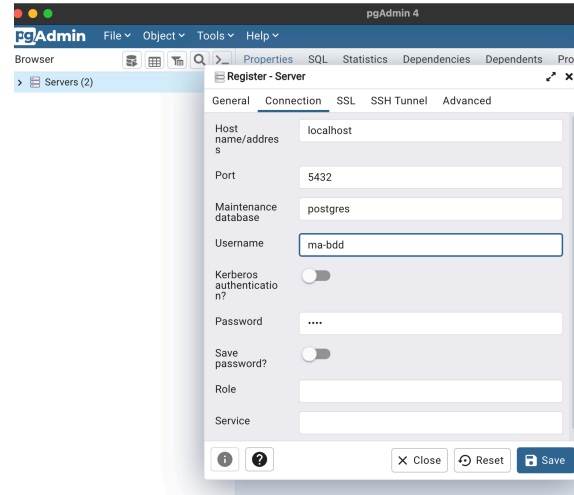
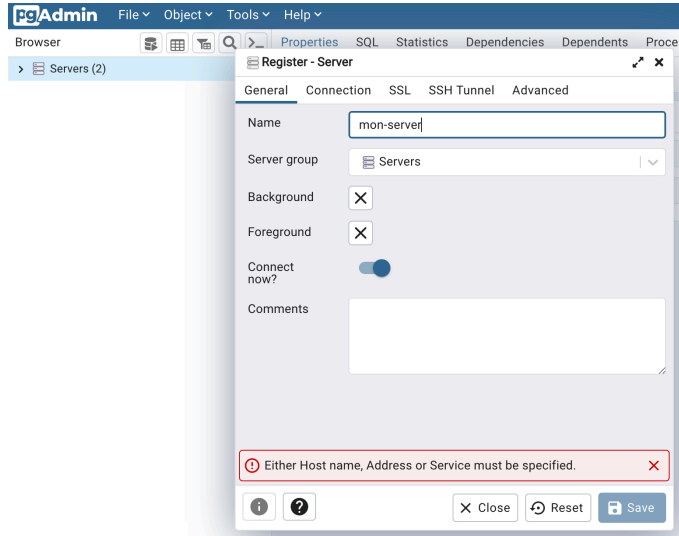
- ▶ C'est le fichier de configuration avec les propriétés Spring
- ▶ Il permet par exemple de changer le port pour le déploiement de notre application



Advanced Rest CLIENT - POST

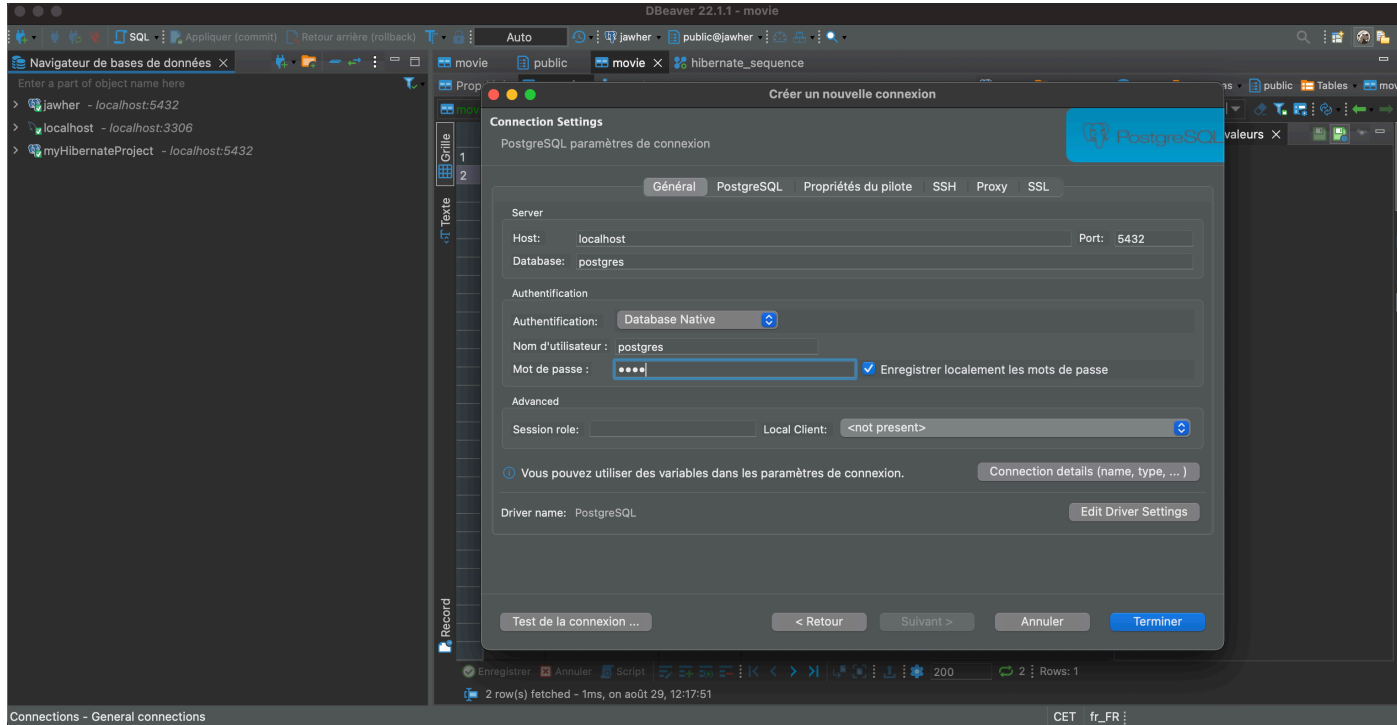


Créer une BDD postgreSQL Avec PgAdmin





Connecter une BDD postgresql à DBeaver



THE END.

**Avec tous nos remerciements et toutes
nos félicitations pour avoir suivi ce
cursus.**

Samih Habbani : s.habbani@coderbase.io