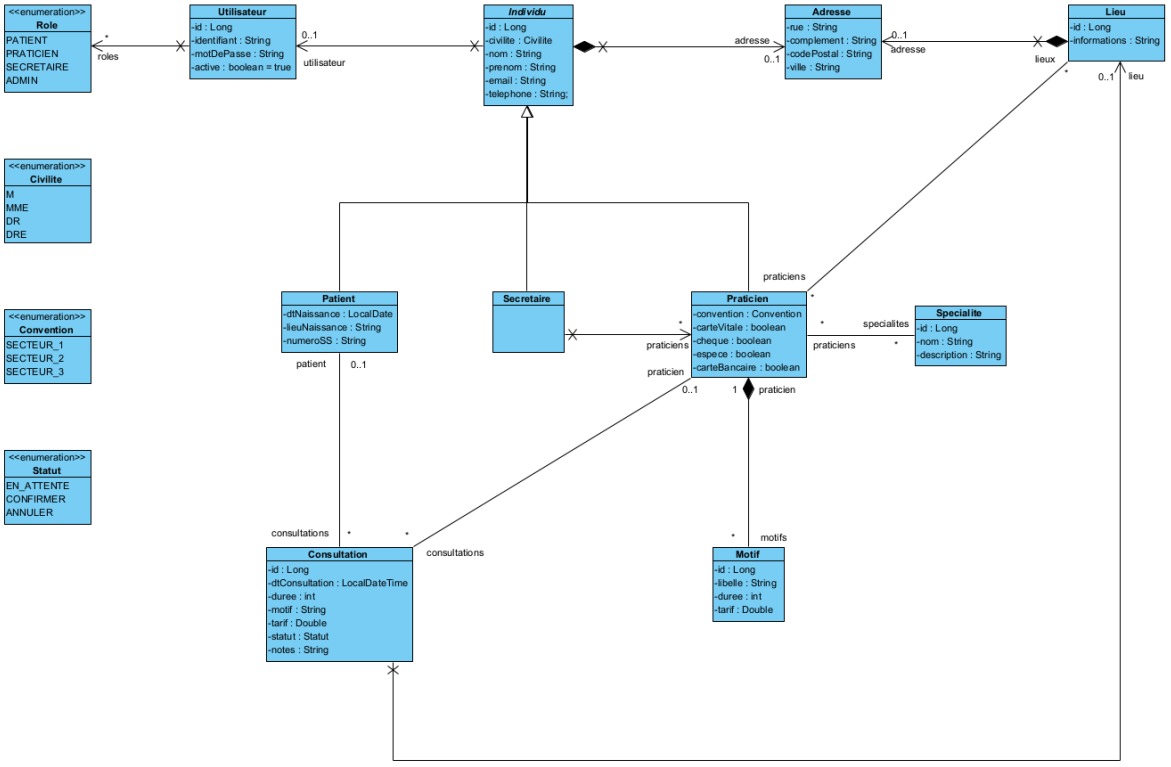
Spring - Ateliers

L'ensemble des ateliers s'appuient sur un projet de réservation de consultations auprès de praticiens.



Atelier 1 : JPA

Exercice 1

Mettre en œuvre le mapping JPA en essayant d'atteindre le script SQL ci-dessous :

```
create table consultation (
  id bigint generated by default as identity,
  dt_consultation timestamp(6),
  duree integer not null,
  motif varchar(255),
  notes varchar(1000),
  statut enum ('ANNULER', 'CONFIRMER', 'EN_ATTENTE'),
  tarif float(53) not null,
  lieu_id bigint,
  patient_id bigint,
  praticien_id bigint,
  primary key (id)
);

create table individu (
  type varchar(15) not null,
  id bigint generated by default as identity,
  codePostal varchar(10),
  complement varchar(255),
```

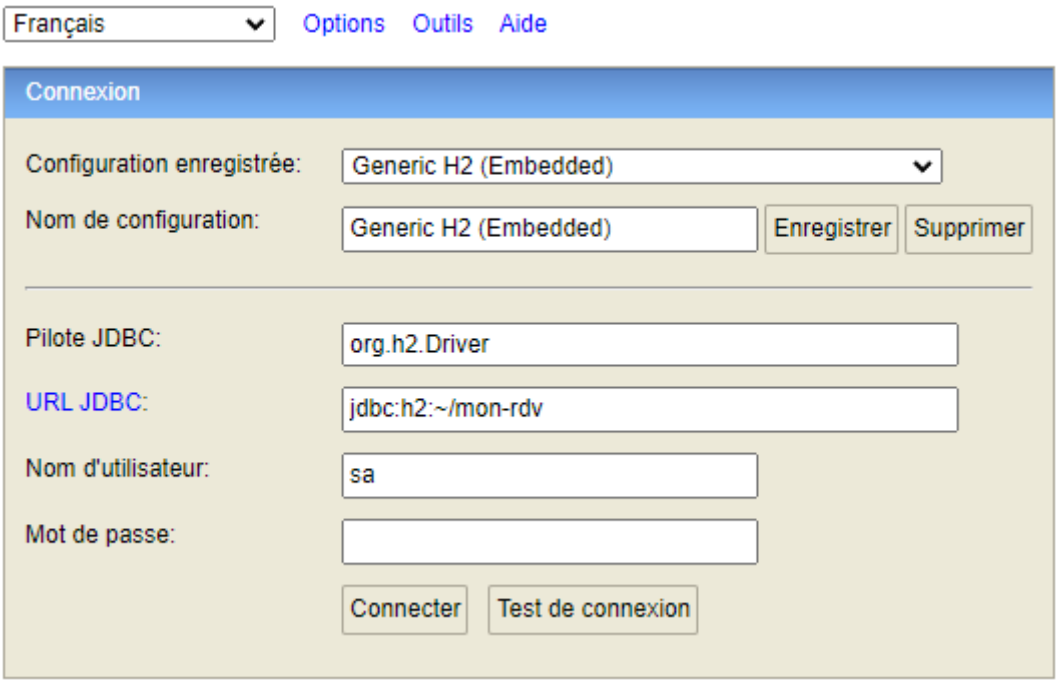
```
  rue varchar(255),
  ville varchar(100),
  civilite enum ('DR', 'DRE', 'M', 'MME'),
  email varchar(255),
  nom varchar(100),
  prenom varchar(100),
  telephone varchar(15),
  carte_bancaire boolean,
  carte_vitale boolean,
  cheque boolean,
  convention enum ('SECTEUR_1', 'SECTEUR_2', 'SECTEUR_3'),
  espece boolean,
  dt_naissance date,
  lieu_naissance varchar(100),
  numero_ss varchar(15),
  utilisateur_id bigint,
  primary key (id)
);

create table lieu (
  id bigint generated by default as identity,
  codePostal varchar(10),
  complement varchar(255),
  rue varchar(255),
  ville varchar(100),
  informations varchar(1000),
  primary key (id)
);

create table motif (
  id bigint generated by default as identity,
  duree integer not null,
  libelle varchar(100),
  tarif float(53),
  praticien_id bigint,
  primary key (id)
);

create table praticien_lieu (
  praticien_id bigint not null, lieu_id bigint not null
);

create table praticien_specialite (
  praticien_id bigint not null, specialite_id bigint not null
);

create table secretaire_praticien (
  secretaire_id bigint not null, praticien_id bigint not null
);

create table specialite (
  id bigint generated by default as identity,
  description varchar(255),
  nom varchar(100),
  primary key (id)
);
```

```
create table utilisateur (
  id bigint generated by default as identity,
  active boolean not null,
  identifiant varchar(50) not null,
  mot_de_passe varchar(100) not null,
  primary key (id)
);

create table utilisateur_roles (
  utilisateur_id bigint not null,
  role enum ('ADMIN', 'PATIENT', 'PRATICIEN', 'SECRETAIRE') not null,
  primary key (utilisateur_id, role)
);

alter table if exists individu add constraint UKteh6v5xpu02hxpehmlx4xop3w unique
(utilisateur_id);
alter table if exists utilisateur add constraint UKo3vqges7u1b1bcp7k9fh5mp8o
unique (identifiant);
alter table if exists consultation add constraint FK5fy7bc2gsbv9ja4ipkn2ngdiu
foreign key (lieu_id) references lieu;
alter table if exists consultation add constraint FKjeldv179g6tt6gvv7fednps7e
foreign key (patient_id) references individu;
alter table if exists consultation add constraint FKs0kkvea1vutv9k077rrju3ku1
foreign key (praticien_id) references individu;
alter table if exists individu add constraint FK8u93rbl6g1is8q0k77id3mlnc foreign
key (utilisateur_id) references utilisateur;
alter table if exists motif add constraint FKq5bhpx5kp7jjbbke8hd008bdd foreign
key (praticien_id) references individu;
alter table if exists praticien_lieu add constraint FKj8pd27vrwngg36fw029oy3knt
foreign key (lieu_id) references lieu;
alter table if exists praticien_lieu add constraint FKpsgprbakrkcprta3gjmjvdnnl
foreign key (praticien_id) references individu;
alter table if exists praticien_specialite add constraint
FK66x0s5aan6g75yfpirj7h8mtn foreign key (specialite_id) references specialite;
alter table if exists praticien_specialite add constraint
FKd8e5j1yoarhtdt20kaw33y6fb foreign key (praticien_id) references individu;
alter table if exists secretaire_praticien add constraint
FKo6pcw05p6iy83pxynaoa9tj0d foreign key (praticien_id) references individu;
alter table if exists secretaire_praticien add constraint
FKq9klqsraj7cxns625hisvvrnw foreign key (secretaire_id) references individu;
alter table if exists utilisateur_roles add constraint
FK9lop304xtodorgho9w56lpjhn foreign key (utilisateur_id) references utilisateur;
```

> Le nom des contraintes unicité ou de clé étrangères auto-générés n'entrent pas dans le cadre de l'exercice.
>
> Pour des questions de praticité, le SGBDR utilisée est embarqué dans l'application H2.

Vous pourrez utiliser la classes *MainJpa* qui se trouve dans le répertoire source *src/main/test* pour initialiser l'EntityManagerFactory basé sur l'unité de persistence *mon-rdv-pu* que vous trouverez dans le fichier *src/main/resources/persistence.xml*.

Vous pourrez une console d'administration web pour H2 en lançant la classes H2Console que vous trouverez dans *src/main/test* également.



Exercice 2

Créer le Repository de gestion des consultations et l'intégrer dans le Singleton MonRdvApplication

Modifier le test MonRdvApplicationTest pour créer deux consultations en les reliant à un patient, un praticien dans un lieu donné.

Exercice 3

Dans les interfaces IConsultationRepository, IIndividuRepository, IUtilisateurRepository, vous trouverez des signatures de méthodes qu'il faudra implémentés dans les classes correspondantes en codant les requêtes JPQL

Atelier 2 : Spring + JPA

Exercice 1

Mise en place d'un projet JPA avec spring-orm et spring-tx

1. Rajouter les dépendences :
    - spring-context, spring-orm, spring-tx, spring-test en version 6.1.8
    - jakarta.transaction-api en version 2.0.1
    - commons-dbcp2 en version 2.12.0
2. Créer une classe de configuration Spring dans spring.formation.config qui créera ces beans :
    1. DataSource pour la connection à la BDD en utilisant commons-dbcp2
    2. EntityManagerFactory pour initialiser JPA en utilisant la Datasource créée précedemment
    3. TransactionManager pour préparer à l'injection automatique par Spring des transactions
        1. Penser à activer via l'annotation adéquat dans la classe de configuration l'utilisation des annotations @Transactional

    4. PersistenceExceptionTranslationPostProcessor pour encapsuler les Exceptions des Repositories dans des exceptions "standards" Spring
3. Supprimer la classe de Singleton RdvApplication => du rouge va apparaitre partout (c'est normal)
4. Modifier les classes de repository pour en faire des beans spring de type @Repository
5. Injecter avec l'annotation spécifique l'EntityManager dans chaque classes de repository
6. Configuration avec l'annotation l'injection automatique des transactions
7. Adapter les méthodes de l'ensemble des classes de Repository
8. Modifier la classe de test MonApplicationTest pour charger la configuration Spring et changer la manière de récupérer les beans Spring (voir slide 27 dans Support SPRING INTRO)

Exercice 2

Mise en place d'un fichier de properties pour externaliser certaines configurations

1. Créer un fichier db.properties qui contiendra ces valeurs :
    - db.driver
    - db.url
    - db.username
    - db.password
    - db.maxConnection
2. Modifier la configuration Spring pour charger ce fichier en mémoire (voir slide 49 sur Support Spring intro)
3. Injecter la classe Environment de Spring dans la configuration
4. Modifier les différentes configurations pour utiliser les propriétés du fichier db.properties
5. Retester le bon fonctionnement

Atelier 3 : Spring Data JPA

Exercice 1

Adapter notre projet pour utiliser les JPARepository générés par Spring en lieu et place de nos classes *RepositoryJpa

> Remarque : les requêtes spécifiques dans les interfaces ont été mises en commentaire ainsi que les tests de ceux-ci

1. Rajouter la dépendence :
    - spring-data-jpa en version 3.3.6
2. Modifier la classes de configuration Spring pour activer les JpaRepository en précisant le package (voir slide 12 sur Support Spring Data JPA)
3. Supprimer la super interface IRepository (des erreurs vont apparaîtrent)
4. Modifier les interfaces pour héritées de l'interface JpaRepository
5. Supprimer les *RepositoryJpa désormais inutiles
6. Retester l'application

Exercice 2

Faire en sorte que les requêtes préalablement commentées refonctionnent.

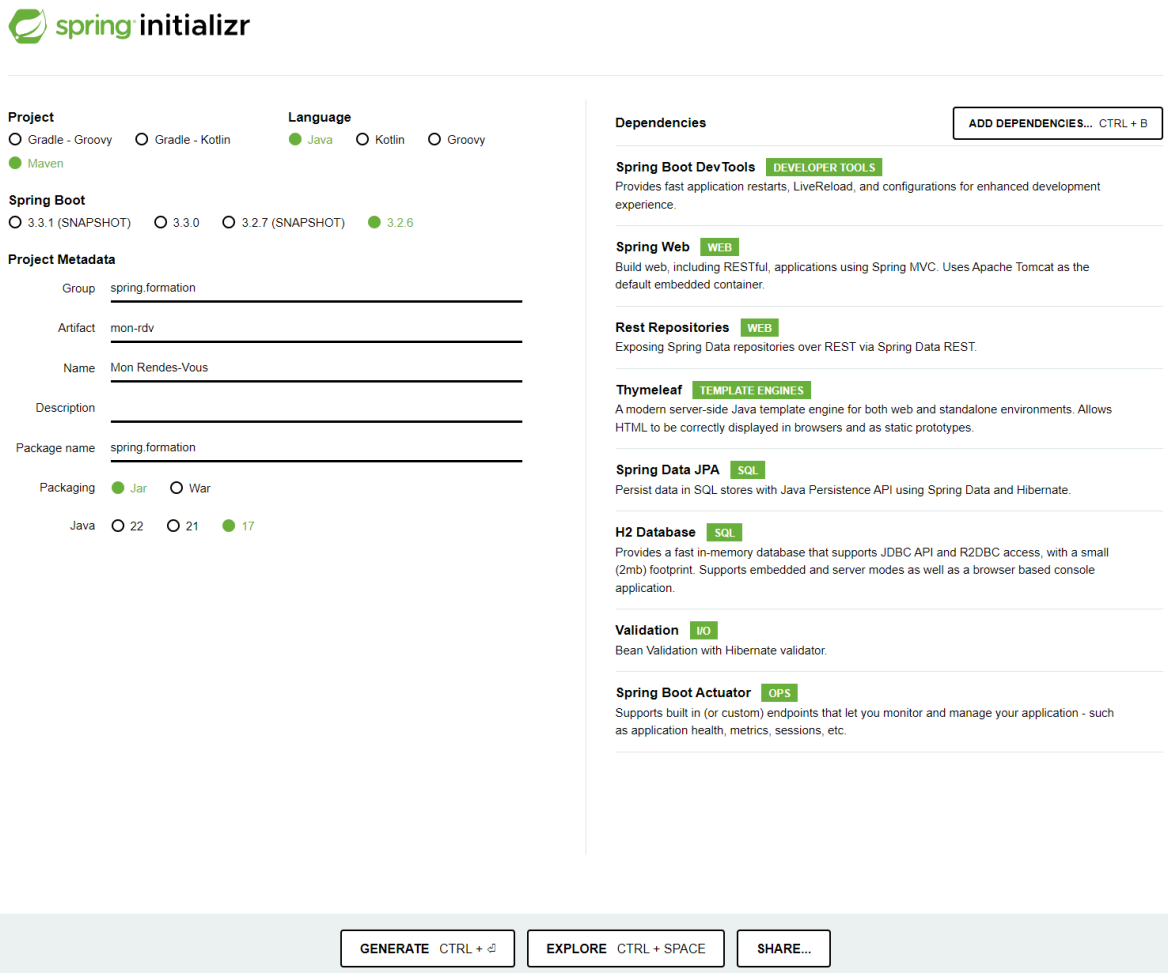
Utiliser les 3 méthodes vues dans le cours :

- Par convention de nommage
- Par @Query et @Param (si besoin)
- Par @NamedQuery et @Param (si besoin)

Atelier 4 : Spring Boot

Exercice 1

Générer une application Spring Boot en utilisant le site https://start.spring.io/



Reprendre l'ensemble des éléments du projet Spring Data JPA et refaire fonctionner le test.

Penser à configurer le fichier application.properties pour qu'il pointe sur la bonne BDD (voir slide 30 du Support Spring Boot)

Exercice 2

Mettre en œuvre un Controleur "classique" d'accueil

1. Créer une classe HomeController dans le package spring.formation.web
    1. Spécifier qu'il s'agit d'un Controller "standard"

2. Créer une méthode qui répond à l'url "/accueil" et qui renvoit vers la page home.html (celle-ci affiche un simple

   # Bonjour le monde

   dans le body)
3. Lancer l'application et tester l'url

2. Modifier HomeController pour que la méthode "/accueil" puisse :
   1. Réceptionner un paramètre nom en paramètre de l'url
   2. Renseigner la valeur de ce paramètre dans un attribut du Model nommé monNom
   3. Modifier l'affichage de la page home.html pour afficher Bonjour le monde ou Bonjour suivi de monNom

Exercice 3

Créer un controlleur REST avec les méthodes standards

1. Créer un RestController pour le Lieu dans le package spring.formation.api
2. Mettre en œuvre les 5 méthodes :
   - getAll()
   - get()
   - post()
   - put()
   - delete()
3. Tester avec postman le bon fonctionnement

Exercice 4

Modifier le mapping Jackson et/ou les requêtes JPQL pour que l'on récupère les lieux depuis son API avec les praticiens associés

> Attention : l'option spring.jpa.open-in-view a été désactivée

Exercice 5

Utilisation des @JSonView pour choisir les éléments de la serialisation JSON.

Adapter le RestController du praticien pour utiliser le mode JSONView et rajouter un nouveau service rest avec un mode détaillé qui inclut la liste des spécialités

Exercice 6

Créer une nouvelle méthode Rest d'inscription pour le patient : /api/patient/inscription

Celle-ci prendra en données d'entrée :

```
{
    "civilite": "M",
    "nom": "NORIS",
```

```
    "prenom": "Chick",
    "email": "chuckpierre@gmail.com",
    "telephone": "0606060606",
    "rue": "1 rue de Rennes",
    "complement": "Tout en haut",
    "codePostal": "44310",
    "ville": "Nantes",
    "identifiant": "chuck666",
    "motDePasse": "123456",
    "dtNaissance": "1965-03-15",
    "lieuNaissance": "Angers",
    "numeroSS": "165034412245124"
}
```

Et devra créer un Patient et son adresse et un utilisateur associé avec le bon rôle

Atelier 5 : Spring Security

Exercice 1

Mettre en place une sécurité basique sur les url commençant par /api

1. Rajouter les starters pour la sécurité dans le pom.xml :

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

2. Créer une classe de configuration Spring pour la sécurité dans le package spring.formation.config
3. Créer un UserDetailsService fictif (en mémoire) qui créera des utilisateurs de test

```
@Bean
public UserDetailsService inMemory(PasswordEncoder passwordEncoder) {
    InMemoryUserDetailsManager manager = new InMemoryUserDetailsManager();
    manager.createUser(

 User.withUsername("patient01").password(passwordEncoder.encode("123456")).ro
les("PATIENT").build());
    manager.createUser(

 User.withUsername("praticien01").password(passwordEncoder.encode("123456")).
roles("PRATICIEN").build());
    manager.createUser(

 User.withUsername("admin").password(passwordEncoder.encode("123456")).roles(
"ADMIN").build());

 manager.createUser(User.withUsername("secretaire01").password(passwordEncode
r.encode("123456"))
            .roles("SECRETAIRE").build());
    return manager;
}
```

4. Créer une bean pour encoder les mots de passe

```
@Bean
public PasswordEncoder passwordEncoder() {
    return new BCryptPasswordEncoder();
}
```

5. Activer le mode d'authentification en Http Basic

```
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
    http.httpBasic(Customizer.withDefaults());
    ...
}
```

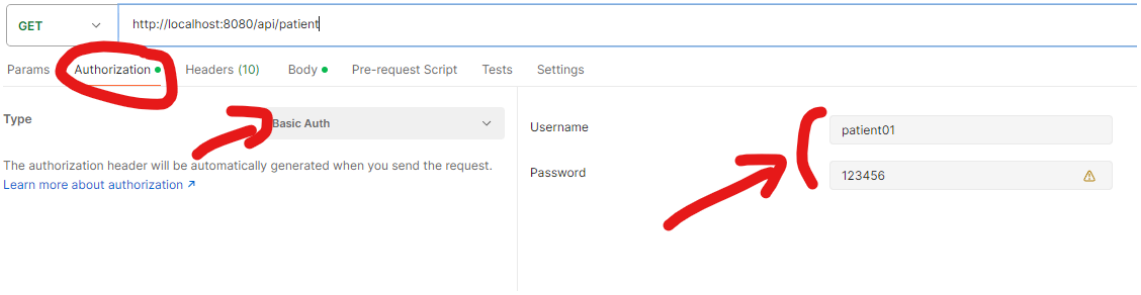
6. Mettre en place des règles d'accès :
    - /api/** : il faut être authentifier
    - /** : autorisé à tout le monde

```
http.authorizeHttpRequests(authorize -> {
    authorize.requestMatchers("/api/**").authenticated();
    authorize.requestMatchers("/**").permitAll();
});
```

7. Désactiver la protection CSRF pour les urls /api/**

```
http.csrf(c -> c.ignoringRequestMatchers("/api/**"));
```

8. Tester le bon fonctionnement dans postman


GET
http://localhost:8080/api/patient
Params
Authorization
Headers (10)
Body
Pre-request Script
Tests
Settings
Type
Basic Auth
The authorization header will be automatically generated when you send the request.
Learn more about authorization
Username
patient01
Password
123456