

SYSTÈMES DISTRIBUÉS

MASTER INTELLIGENCE
ARTIFICIELLE ET ANALYSE
DES DONNÉES

RÉALISÉE PAR :
ANOADA NOHAYLA

ENCADRÉ PAR :
M. MOHAMED YOUSFI



ACTIVITÉ PRATIQUE N°2

SPRING DATA JPA HIBERNATE

1. INSTALLER INTELLIJ ULTIMATE



2. CRÉER UN PROJET SPRING INITIALIZER AVEC LES DÉPENDANCES JPA, H2, SPRING WEB ET LOMBOCK

The image consists of two screenshots of the IntelliJ Spring Initializr interface. The top screenshot shows the main configuration screen where a "Spring Initializr" generator is selected. The project details include: Server URL: start.spring.io, Name: Anoada_Nohayla_Tp2_1, Location: ~\Desktop\study\school\Master\S2\Systèmes_Distribués\Tp\Anoada_Nohayla_Tp2, Language: Java, Type: Maven, Group: org.example, Artifact: Anoada_Nohayla_Tp2_1, Package name: org.example.anoada_nohayla_tp2_1, JDK: 18 Oracle OpenJDK version 18.0.1, Java: 17, and Packaging: Jar. The bottom screenshot shows the "Added dependencies:" section, which lists: Spring Data JPA, H2 Database, Spring Web, and Lombok.

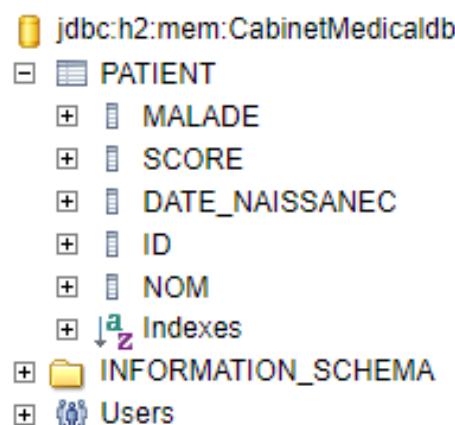
3. CRÉER L'ENTITÉ JPA PATIENT AYANT LES ATTRIBUTS :

- ID DE TYPE LONG
- NOM DE TYPE STRING
- DATE_NAISSANCE DE TYPE DATE
- MALADE DE TYPE BOOLEAN
- SCORE DE TYPE INT

```
@Entity
@Data @NoArgsConstructor @AllArgsConstructor
public class Patient
{
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nom;
    private Date dateNaissance;
    private boolean malade;
    private int score;
}
```

4. CONFIGURER L'UNITÉ DE PERSISTANCE DANS LE FICHER APPLICATION.PROPERTIES

```
spring.application.name=Anoada_Nohayla_Tp2_1
server.port=8088
spring.datasource.url=jdbc:h2:mem:CabinetMedicaldb
spring.datasource.username=nohayla
spring.datasource.password=password
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.h2.console.enabled=true
```



5. CRÉER L'INTERFACE JPA REPOSITORY BASÉE SUR SPRING DATA

```
public interface PatientRepository extends JpaRepository<Patient, Long>
{
}
```

6. TESTER QUELQUES OPÉRATIONS DE GESTION DE PATIENTS :

- AJOUTER DES PATIENTS

```
SimpleDateFormat sdf = new SimpleDateFormat(pattern: "dd/MM/yyyy");

patientRepository.save(new Patient(id: null, nom: "noha", sdf.parse(source: "30/05/2001"), malade: false, score: 0));
patientRepository.save(new Patient(id: null, nom: "salah", sdf.parse(source: "12/08/1998"), malade: false, score: 0));
patientRepository.save(new Patient(id: null, nom: "hind", sdf.parse(source: "10/07/1985"), malade: true, score: 0));
patientRepository.save(new Patient(id: null, nom: "saad", sdf.parse(source: "02/01/2014"), malade: false, score: 0));
patientRepository.save(new Patient(id: null, nom: "farah", sdf.parse(source: "08/06/2002"), malade: true, score: 0));
```

- CONSULTER TOUS LES PATIENTS

```
List<Patient> patients= patientRepository.findAll();
patients.forEach(p->{
    System.out.println(p.toString());
});
```

```
Patient(id=1, nom=noha, dateNaissance=2001-05-30 00:00:00.0, malade=false, score=0)
Patient(id=2, nom=salah, dateNaissance=1998-08-12 00:00:00.0, malade=false, score=0)
Patient(id=3, nom=hind, dateNaissance=1985-07-10 00:00:00.0, malade=true, score=0)
Patient(id=4, nom=saad, dateNaissance=2014-01-02 00:00:00.0, malade=false, score=0)
Patient(id=5, nom=farah, dateNaissance=2002-06-08 00:00:00.0, malade=true, score=0)
```

- CONSULTER UN PATIENT

```
Patient patient=patientRepository.findById(3L).get();
System.out.println(patient.getId());
System.out.println(patient.getNom());
System.out.println(patient.getDateNaissance());
System.out.println(patient.isMalade());
System.out.println(patient.getScore());
```

```
3
hind
1985-07-10 00:00:00.0
true
0
```

- CHERCHER DES PATIENTS

```
List<Patient> findByMalade(boolean malade);
```

```
List<Patient> patientsIsMalade = patientRepository.findByMalade(true);
patientsIsMalade.forEach(p->{
    System.out.println(p.toString());
});
```

```
Patient(id=3, nom=hind, dateNaissance=1985-07-10 00:00:00.0, malade=true, score=0)
Patient(id=5, nom=farah, dateNaissance=2002-06-08 00:00:00.0, malade=true, score=0)
```

- METTRE À JOUR UN PATIENT

```
Patient patientEdit=patientRepository.findById(2L).get();
patientEdit.setNom("edit name");
patientRepository.save(patientEdit);
System.out.println(patientEdit.toString());
```

```
Patient(id=2, nom=edit name, dateNaissance=1998-08-12 00:00:00.0, malade=false, score=0)
```

- SUPPRIMER UN PATIENT

```
patientRepository.deleteById(1L);
```

7. MIGRER DE H2 DATABASE VERS MYSQL

```
<dependency>
    <groupId>com.mysql</groupId>
    <artifactId>mysql-connector-j</artifactId>
    <scope>runtime</scope>
</dependency>
```

```
spring.application.name=Anoada_Nohayla_Tp2_1
server.port=8088
spring.datasource.url=jdbc:mysql://localhost:3306/CabinetMedicaldb?createDatabaseIfNotExist=true
spring.datasource.username=root
spring.datasource.password=
spring.jpa.hibernate.ddl-auto=update
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MariaDBDialect
```

Serveur : 127.0.0.1 » Base de données : cabinetmedicaldb » Table : patient						
<input type="checkbox"/> Profilage [Éditer en ligne] [Éditer] [Expliquer SQL] [Créer le code source PHP] [Actualiser]						
<input type="checkbox"/> Tout afficher Nombre de lignes : <select value="25">25</select>						
Options supplémentaires						
← T →	id	date_naissance	malade	nom	score	
<input type="checkbox"/>	2	1998-08-12 00:00:00.000000	0	edit name	0	
<input type="checkbox"/>	3	1985-07-10 00:00:00.000000	1	hind	0	
<input type="checkbox"/>	4	2014-01-02 00:00:00.000000	0	saad	0	
<input type="checkbox"/>	5	2002-06-08 00:00:00.000000	1	farah	0	

8. REPRENDRE LES EXEMPLES DU PATIENT, MÉDECIN, RENDEZ-VOUS, CONSULTATION.

1. ENTITÉS

CET PACKAGE CONTIENT LES CLASSES QUI PRÉSENTENT LES ENTITÉS DE VOTRE APPLICATION, À SAVOIR CONSULTATION, MÉDECIN, PATIENT, RENDEZVOUS, ET L'ÉNUMÉRATION STATUSRDV.

• PATIENT :

CETTE ENTITÉ PRÉSENTE UN PATIENT. ELLE CONTIENT DES INFORMATIONS TELLES QUE L'ID, LE NOM DU PATIENT, SA DATE DE NAISSANCE, SON STATUT DE MALADIE, SON SCORE ET UNE COLLECTION DE RENDEZ-VOUS ASSOCIÉS À CE PATIENT.

```
@Entity
@Data @NoArgsConstructor @AllArgsConstructor
public class Patient
{
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nom;
    @Temporal(TemporalType.DATE)
    private Date dateNaissance;
    private boolean malade;
    private int score;
    @OneToMany(mappedBy = "patient", fetch = FetchType.LAZY)
    private Collection<RendezVous> rendezVous;
}
```

• MÉDECIN :

CETTE ENTITÉ REPRÉSENTE UN MÉDECIN. ELLE CONTIENT DES INFORMATIONS TELLES QUE L'ID, LE NOM DU MÉDECIN, SON EMAIL, SA SPÉCIALITÉ ET UNE COLLECTION DE RENDEZ-VOUS ASSOCIÉS À CE MÉDECIN.

```
@Entity
@Data @NoArgsConstructor @AllArgsConstructor
public class Medecin
{
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nom;
    private String email;
    private String specialite;
    @OneToMany(mappedBy = "medecin", fetch = FetchType.LAZY)
    @JsonProperty(access = JsonProperty.Access.READ_WRITE)
    private Collection<RendezVous> rendezVous;
}
```

• CONSULTATION :

CETTE ENTITÉ PRÉSENTE UNE CONSULTATION MÉDICALE. ELLE CONTIENT DES INFORMATIONS TELLES QUE L'ID, LA DATE DE LA CONSULTATION, LE RAPPORT DE LA CONSULTATION ET UNE RÉFÉRENCE À UN RENDEZ-VOUS.

```
@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Consultation
{
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private Date dateConsultation;
    private String rapport;
    @OneToOne
    @JsonProperty(access = JsonProperty.Access.READ_WRITE)
    private RendezVous rendezVous;
}
```

• RENDEZVOUS :

PRÉSENTE UN RENDEZ-VOUS ENTRE UN PATIENT ET UN MÉDECIN, AVEC DES ATTRIBUTS TELS QUE LA DATE DU RENDEZ-VOUS, LE STATUT ET LES RÉFÉRENCES AU PATIENT ET AU MÉDECIN.

```
@Entity
@Data @NoArgsConstructor @AllArgsConstructor
public class RendezVous
{
    @Id
    private String id;
    private Date date;
    @Enumerated(EnumType.STRING)
    private StatusRDV status;
    @ManyToOne
    @JsonProperty(access = JsonProperty.Access.READ_WRITE)
    private Patient patient;
    @ManyToOne
    private Medecin medecin;
    @OneToOne(mappedBy = "rendezVous")
    private Consultation consultation;
}
```

• STATUSRDV :

C'EST UNE ÉNUMÉRATION QUI
REPRÉSENTE LES DIFFÉRENTS STATUTS
D'UN RENDEZ-VOUS (EN ATTENTE,
ANNULÉ, TERMINÉ).

```
public enum StatusRDV
{
    1 usage
    PENDING,
    no usages
    CANCELED,
    no usages
    DONE
}
```

The screenshot shows the MySQL Workbench interface. At the top, it displays the server as 127.0.0.1 and the database as cabinetmedicale. Below the header are tabs for Structure, SQL, Rechercher (Search), and Requête (Query). A 'Filtres' (Filters) section contains a search input field labeled 'Contenant le mot:' (Containing the word:). The main area lists four tables:

Table	Action
consultation	★ Parcourir Structure Rechercher
medecin	★ Parcourir Structure Rechercher
patient	★ Parcourir Structure Rechercher
rendez_vous	★ Parcourir Structure Rechercher

Below the table list, it says '4 tables' and 'Somme'.

2. REPOSITORY

LES INTERFACES DANS CE PACKAGE DÉFINISSENT LES OPÉRATIONS CRUD (CREATE, READ, UPDATE, DELETE) POUR CHAQUE ENTITÉ EN UTILISANT SPRING DATA JPA.

• PATIENTREPOSITORY

```
public interface PatientRepository extends JpaRepository<Patient, Long>
{
    no usages
    List<Patient> findByMalade(boolean malade);
    no usages
    List<Patient> findByNom(String nom);
}
```

• MÉDECINREPOSITORY

```
public interface MedecinRepository extends JpaRepository<Medecin, Long>
{
    no usages
    List<Medecin> findByNom(String nom);
}
```

• CONSULTATIONREPOSITORY

```
public interface ConsultationRepository extends JpaRepository<Consultation, Long>
{
    no usages
    List<Consultation> findByDateConsultation(Date dateConsultation);
}
```

• RENDEZVOUSREPOSITORY

```
public interface RendezVousRepository extends JpaRepository<RendezVous, String>
{
    no usages
    List<RendezVous> findByDate(Date date);
}
```

3. SERVICE

CE PACKAGE CONTIENT UNE INTERFACE IHOSPITALSERVICE AINSI QU'UNE IMPLÉMENTATION IHOSPITALSERVICEIMPL QUI GÈRE LA LOGIQUE MÉTIER DE VOTRE APPLICATION, NOTAMMENT LA SAUVEGARDE DES ENTITÉS.

• IHOSPITALSERVICE

```
public interface IHospitalService
{
    1 usage 1 implementation
    Patient savePatient (Patient patient);
    1 usage 1 implementation
    Medecin saveMedecin (Medecin medecin);
    1 usage 1 implementation
    RendezVous saveRDV (RendezVous rendezvous);
    1 usage 1 implementation
    Consultation saveConsultation (Consultation consultation);
}
```

. IHOSPITALSERVICE

CETTE STRUCTURE DE CODE MET EN PLACE UNE COUCHE DE SERVICE POUR GÉRER LES OPÉRATIONS CRUD (CREATE, READ, UPDATE, DELETE) SUR LES ENTITÉS DE L'APPLICATION HOSPITALIÈRE, EN UTILISANT SPRING DATA JPA POUR L'INTERACTION AVEC LA BASE DE DONNÉES.

```
@Service
@Transactional
public class IHospitalServiceImpl implements IHospitalService {
    2 usages
    private PatientRepository patientRepository;
    2 usages
    private MedecinRepository medecinRepository;
    2 usages
    private RendezVousRepository rendezvousRepository;
    2 usages
    private ConsultationRepository consultationRepository;

    public IHospitalServiceImpl(PatientRepository patientRepository,
                                MedecinRepository medecinRepository,
                                RendezVousRepository rendezvousRepository,
                                ConsultationRepository consultationRepository)
    {
        this.patientRepository = patientRepository;
        this.medecinRepository = medecinRepository;
        this.rendezvousRepository = rendezvousRepository;
        this.consultationRepository = consultationRepository;
    }

    1 usage
    @Override
    public Patient savePatient(Patient patient) { return patientRepository.save(patient); }

    1 usage
    @Override
    public Medecin saveMedecin(Medecin medecin) { return medecinRepository.save(medecin); }

    1 usage
    @Override
    public RendezVous saveRDV(RendezVous rendezvous) {
        rendezvous.setId(UUID.randomUUID().toString());
        return rendezvousRepository.save(rendezvous);
    }

    1 usage
    @Override
    public Consultation saveConsultation(Consultation consultation) {
        return consultationRepository.save(consultation);
    }
}
```

4. MAIN APPLICATION CLASS

LA CLASSE PRINCIPALE ANOADANOHAYLATP21APPLICATION QUI DÉMARRE L'APPLICATION SPRING BOOT ET INITIALISE LES DONNÉES DE BASE.

```
@SpringBootApplication
public class AnoadaNohaylaTp21Application //implements CommandLineRunner
{
    @Autowired
    public static void main(String[] args) { SpringApplication.run(AnoadaNohaylaTp21Application.class, args); }

    @Bean
    CommandLineRunner start(
        IHospitalService hospitalService,
        PatientRepository patientRepository,
        MedecinRepository medecinRepository,
        RendezVousRepository rendezVousRepository,
        ConsultationRepository consultationRepository
    ) {
        return args -> {
            // Ajout des patients
            Stream.of(...values: "Nohayla", "Hind", "Mohamed", "Hassan", "Najat")
                .forEach(name -> {
                    Patient patient = new Patient();
                    patient.setNom(name);
                    patient.setDateNaissance(new Date());
                    patient.setMalade(false);
                    patient.setScore(2);
                    hospitalService.savePatient(patient);
                });
            // Ajout des médecins
            Stream.of(...values: "Dr. Mohamed", "Dr. Hassan", "Dr. Najat")
                .forEach(name -> {
                    Medecin medecin=new Medecin();
                    medecin.setNom(name);
                    medecin.setEmail(name+"@gmail.com");
                    medecin.setSpecialite(Math.random()>0.5?"Cardio":"Dentiste");
                    hospitalService.saveMedecin(medecin);
                });
            // Ajout des rendez-vous
            Patient patient = patientRepository.findById(1L).orElseThrow();
            Medecin medecin = medecinRepository.findById(1L).orElseThrow();

            RendezVous rendezVous = new RendezVous();
            rendezVous.setDate(new Date());
            rendezVous.setStatus(StatusRDV.PENDING);
            rendezVous.setPatient(patient);
            rendezVous.setMedecin(medecin);
            hospitalService.saveRDV(rendezVous);

            // Ajout des consultations
            Consultation consultation = new Consultation();
            consultation.setDateConsultation(new Date());
            consultation.setRapport("Rapport de consultation");
            consultation.setRendezVous(rendezVous);
            hospitalService.saveConsultation(consultation);
        };
    }
}
```

Base de données : cabinetmedicale » Table : patient

	id	date_naissance	malade	nom	score
Supprimer	1	2024-04-13	0	Nohayla	2
Supprimer	2	2024-04-13	0	Hind	2
Supprimer	3	2024-04-13	0	Mohamed	2
Supprimer	4	2024-04-13	0	Hassan	2
Supprimer	5	2024-04-13	0	Najat	2

Base de données : cabinetmedicale » Table : medecin

	id	email	nom	specialite
Supprimer	1	Dr. Mohamed@gmail.com	Dr. Mohamed	Cardio
Supprimer	2	Dr. Hassan@gmail.com	Dr. Hassan	Cardio
Supprimer	3	Dr. Najat@gmail.com	Dr. Najat	Dentiste

Base de données : cabinetmedicale » Table : consultation

	id	date_consultation	rapport	rendez_vous_id
Supprimer	1	2024-04-13 03:25:31.000000	Rapport de consultation	1781b559-a475-4cf6-b12b-01d33cf8673a
Supprimer	2	2024-04-14 16:16:49.000000	Rapport de consultation	2e6f25bf-cc6b-4e30-ac9b-cc9d10f7db64
Supprimer	3	2024-04-14 16:40:33.000000	Rapport de consultation	3e07f148-8b54-48e5-8e5d-5844dc6b073f

Base de données : cabinetmedicale » Table : rendez_vous

	id	date	status	medecin_id	patient_id
Supprimer	1781b559-a475-4cf6-b12b-01d33cf8673a	2024-04-13 03:25:31.000000	PENDING	1	1
Supprimer	2e6f25bf-cc6b-4e30-ac9b-cc9d10f7db64	2024-04-14 16:16:49.000000	PENDING	1	1
Supprimer	3e07f148-8b54-48e5-8e5d-5844dc6b073f	2024-04-14 16:40:33.000000	PENDING	1	1
Supprimer	debfda26-3334-435a-8832-5221ca471e7f	2024-04-14 16:48:49.000000	PENDING	1	1

5. WEB

CE PACKAGE CONTIENT LES CONTRÔLEURS REST QUI EXPOSENT LES ENDPOINTS POUR INTERAGIR AVEC VOTRE APPLICATION.

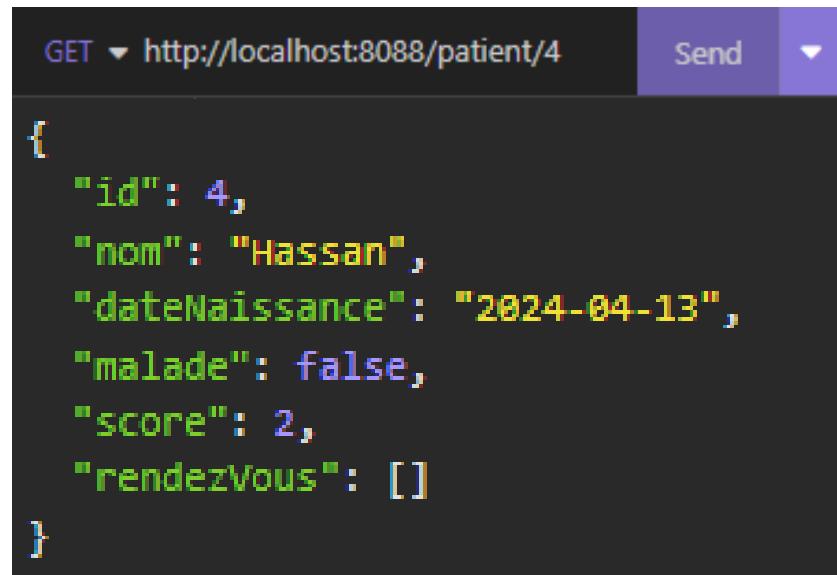
```
@RestController
public class PatientRestController
{
    @Autowired
    private PatientRepository patientRepository;

    @GetMapping("/patients")
    public List<Patient> patientList() { return patientRepository.findAll(); }
```

```
[{"id": 1, "nom": "Nohayla", "dateNaissance": "2024-04-13", "malade": false, "score": 2, "rendezVous": [{"id": "1781b559-a475-4cf6-b12b-01d33cf8673a", "date": "2024-04-13T03:25:31.000+00:00", "status": "PENDING", "patient": {"id": 1, "nom": "Nohayla", "dateNaissance": "2024-04-13", "malade": false, "score": 2, "rendezVous": [{"id": "1781b559-a475-4cf6-b12b-01d33cf8673a", "date": "2024-04-13T03:25:31.000+00:00", "status": "PENDING", "patient": {"id": 1,
```

CE CONTRÔLEUR REST EXPOSE UN POINT DE TERMINAISON "/PATIENTS" QUI RENVOIE LA LISTE DE TOUS LES PATIENTS STOCKÉS DANS LA BASE DE DONNÉES LORSQUE VOUS EFFECTUEZ UNE REQUÊTE GET À CET URL.

```
@GetMapping("/patient/{id}")
public ResponseEntity<Patient> getPatientById(@PathVariable Long id)
{
    Optional<Patient> optionalPatient = patientRepository.findById(id);
    return optionalPatient.map(ResponseEntity::ok).orElseGet(() -> ResponseEntity.notFound().build());
}
```

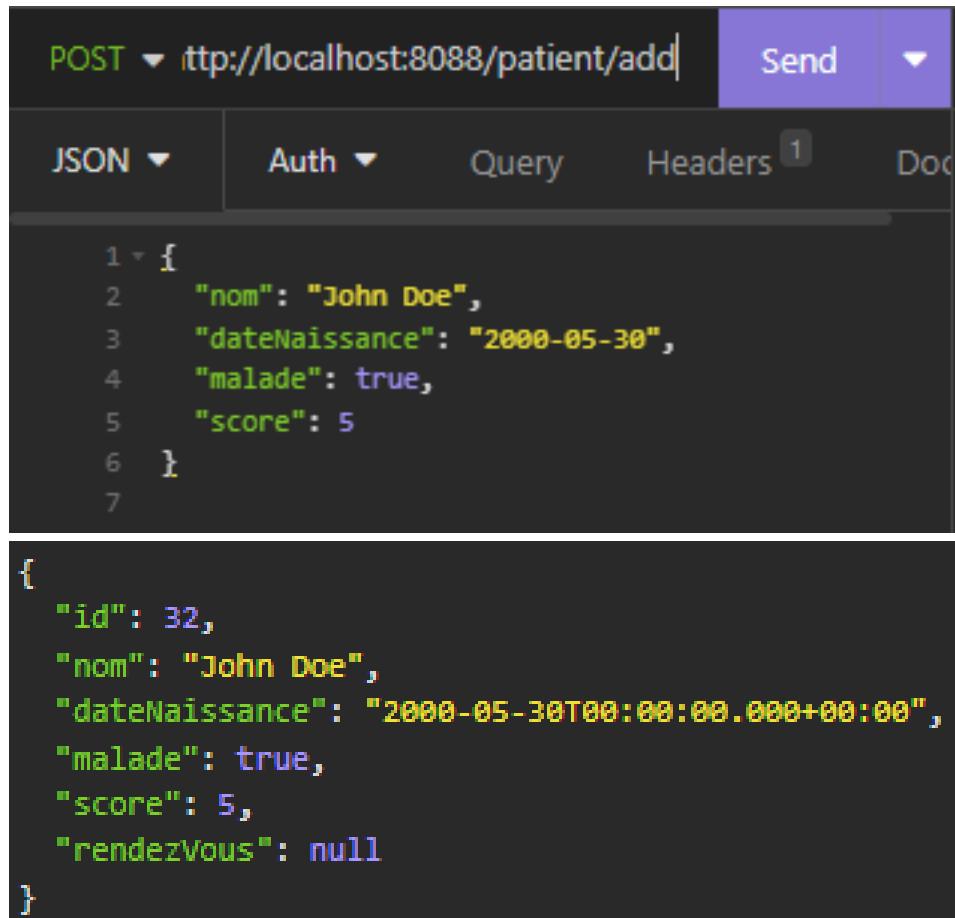


A screenshot of a Postman request window. The method is set to 'GET' and the URL is 'http://localhost:8088/patient/4'. The 'Send' button is visible. The response body is displayed below:

```
{
  "id": 4,
  "nom": "Hassan",
  "dateNaissance": "2024-04-13",
  "malade": false,
  "score": 2,
  "rendezVous": []
}
```

CETTE MÉTHODE PERMET DE RÉCUPÉRER UN PATIENT SPÉCIFIQUE EN FONCTION DE SON ID EN UTILISANT L'URL "/PATIENT/{ID}". SI LE PATIENT EST TROUVÉ, IL EST RENVOYÉ AVEC UN STATUT HTTP 200. SINON, UNE RÉPONSE HTTP 404 EST RENVOYÉE POUR INDICER QUE LE PATIENT N'A PAS ÉTÉ TROUVÉ.

```
@PostMapping("/patient/add")
public ResponseEntity<Patient> addPatient(@RequestBody Patient patient)
{
    Patient savedPatient = patientRepository.save(patient);
    return ResponseEntity.ok(savedPatient);
}
```



CETTE MÉTHODE PERMET D'AJOUTER UN NOUVEAU PATIENT À LA BASE DE DONNÉES EN UTILISANT UNE REQUÊTE POST SUR L'URL "/PATIENT/ADD". ELLE RENVOIE UNE RÉPONSE HTTP 200 AVEC LE PATIENT AJOUTÉ EN TANT QUE CORPS DE LA RÉPONSE POUR INDICER QUE L'OPÉRATION S'EST BIEN DÉROULÉE.

```

@PutMapping("patient/{id}")
public ResponseEntity<Patient> updatePatient(@PathVariable Long id, @RequestBody Patient patientDetails) {
    Optional<Patient> optionalPatient = patientRepository.findById(id);
    if (optionalPatient.isPresent()) {
        Patient existingPatient = optionalPatient.get();
        existingPatient.setNom(patientDetails.getNom());
        existingPatient.setDateNaissance(patientDetails.getDateNaissance());
        existingPatient.setMalade(patientDetails.isMalade());
        existingPatient.setScore(patientDetails.getScore());
        patientRepository.save(existingPatient);
        return ResponseEntity.ok(existingPatient);
    } else {
        return ResponseEntity.notFound().build();
    }
}

```

PUT ▾ http://localhost:8088/patient/26 | Send ▾

JSON ▾ Auth ▾ Query Headers 1 Doc

```

1  {
2      "nom": "edit nom",
3      "dateNaissance": "2015-06-30",
4      "malade": false,
5      "score": 25
6  }

```

```

{
  "id": 32,
  "nom": "John Doe",
  "dateNaissance": "2000-05-30T00:00:00.000+00:00",
  "malade": true,
  "score": 5,
  "rendezVous": null
}

```

CETTE MÉTHODE PERMET DE METTRE À JOUR LES INFORMATIONS D'UN PATIENT EXISTANT DANS LA BASE DE DONNÉES EN UTILISANT UNE REQUÊTE PUT SUR L'URL "/PATIENT/{ID}". ELLE RENVOIE UNE RÉPONSE HTTP 200 AVEC LE PATIENT MIS À JOUR EN TANT QUÉ CORPS DE LA RÉPONSE POUR INDICER QUÉ L'OPÉRATION

S'EST BIEN DÉROULÉE. SI LE PATIENT AVEC L'IDENTIFIANT DONNÉ N'EST PAS TROUVÉ, ELLE RENVOIE UNE RÉPONSE HTTP 404 (NOT FOUND).

```
@DeleteMapping("/patient/{id}")
public ResponseEntity<Void> deletePatient(@PathVariable Long id)
{
    Optional<Patient> optionalPatient = patientRepository.findById(id);
    if (optionalPatient.isPresent()) {
        patientRepository.deleteById(id);
        return ResponseEntity.noContent().build();
    } else {
        return ResponseEntity.notFound().build();
    }
}
```

DELETE ▾ http://localhost:8088/patient/32 | Send ▾

CETTE MÉTHODE PERMET DE SUPPRIMER UN PATIENT EXISTANT DE LA BASE DE DONNÉES EN UTILISANT UNE REQUÊTE DELETE SUR L'URL "/PATIENT/{ID}". ELLE RENVOIE UNE RÉPONSE HTTP 204 (NO CONTENT) POUR INDICER QUE LA SUPPRESSION S'EST BIEN DÉROULÉE. SI LE PATIENT AVEC L'IDENTIFIANT DONNÉ N'EST PAS TROUVÉ, ELLE RENVOIE UNE RÉPONSE HTTP 404 (NOT FOUND).

G. FICHIER DE CONFIGURATION

CÉ FICHIER CONTIENT LES CONFIGURATIONS DE BASE DE L'APPLICATION, TELLES QUE LE NOM DE L'APPLICATION, LE PORT, L'URL DE LA BASE DE DONNÉES, LE NOM D'UTILISATEUR, LE MOT DE PASSE....

```
spring.application.name=Anoada_Nohayla_Tp2_1
server.port=8088
spring.datasource.url=jdbc:mysql://localhost:3306/CabinetMedicale?createDatabaseIfNotExist=true
spring.datasource.username=root
spring.datasource.password=
spring.jpa.hibernate.ddl-auto=update
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MariaDBDialect
```

9. REPRENDRE LES EXEMPLES USERS ET ROLES

1. ENTITÉS

CES ENTITÉS REPRÉSENTENT UN SYSTÈME DE GESTION DE RÔLES ET D'UTILISATEURS, OÙ CHAQUE UTILISATEUR PEUT AVOIR PLUSIEURS RÔLES ET CHAQUE RÔLE PEUT ÊTRE ASSOCIÉ À PLUSIEURS UTILISATEURS.

. USER :

```
@Entity @Table(name = "USERS")
@Data @NoArgsConstructor @AllArgsConstructor
public class User
{
    @Id
    private String userId;
    @Column(name = "USER_NAME", unique = true, length = 20)
    private String userName;
    @JsonProperty(access = JsonProperty.Access.WRITE_ONLY)
    private String password;
    @ManyToMany(mappedBy = "users", fetch = FetchType.EAGER)
    private List<Role> roles=new ArrayList<>();
}
```

• ROLE :

```
@Entity @Table(name = "ROLES")
@Data @NoArgsConstructor @AllArgsConstructor
public class Role
{
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String desc;
    @Column(unique = true,length = 20)
    private String roleName;
    @ManyToMany(fetch = FetchType.EAGER)
    @ToString.Exclude
    @JsonProperty(access = JsonProperty.Access.WRITE_ONLY)
    private List<User> users=new ArrayList<>();
}
```

2. REPOSITORY

CES CODE MONTRÉNT DES INTERFACES DE REPOSITORY UTILISÉES POUR INTERAGIR AVEC LES ENTITÉS ROLE ET USER DANS LE CONTEXTE D'UNE APPLICATION SPRING BOOT AVEC JPA.

• USERREPOSITORY :

```
public interface UserRepository extends JpaRepository<User, String>
{
    2 usages
    User findUserByUserName(String userName);
}
```

• ROLEREPOSITORY :

```
@Repository
public interface RoleRepository extends JpaRepository<Role, String>
{
    1 usage
    Role findByRoleName(String roleName);
}
```

3. SERVICE

CES CLASSES ET MÉTHODES CONSTITUENT UN SERVICE DE GESTION DES UTILISATEURS ET DES RÔLES, OFFRANT DES FONCTIONNALITÉS TELLES QUE L'AJOUT D'UTILISATEURS, L'AJOUT DE RÔLES, L'ATTRIBUTION DE RÔLES AUX UTILISATEURS, LA RECHERCHE D'UTILISATEURS ET DE RÔLES, AINSI QUE L'AUTHENTIFICATION DES UTILISATEURS.

• USERSERVICE

```
public interface UserService
{
    1 usage 1 implementation
    User addNewUser(User user);
    1 usage 1 implementation
    Role addNewRole(Role role);
    1 usage 1 implementation
    User findUserByUserName(String userName);
    1 usage 1 implementation
    Role findRoleByRoleName(String roleName);
    6 usages 1 implementation
    void addRoleToUser(String userName, String roleName);
    1 usage 1 implementation
    User authenticate(String userName, String password);
}
```

• USERSERVICEIMPL

```
@Service
@Transactional
@AllArgsConstructor
public class UserServiceImpl implements UserService
{
    private UserRepository userRepository;
    private RoleRepository roleRepository;
    1 usage
    @Override
    public User addNewUser(User user)
    {
        user.setUserId(UUID.randomUUID().toString());
        return userRepository.save(user);
    }
}
```

```
@Override
public Role addNewRole(Role role)
{
    return roleRepository.save(role);
}

1 usage
@Override
public User findUserByUserName(String userName)
{
    return userRepository.findUserByUserName(userName);
}

1 usage
@Override
public void addRoleToUser(String userName, String roleName)
{
    User user=findUserByUserName(userName);
    Role role=findRoleByRoleName(roleName);
    if(user.getRoles()!=null)
    {
        user.getRoles().add(role);
        role.getUsers().add(user);
    }
    userRepository.save(user);
}

1 usage
@Override
public User authenticate(String userName, String password)
{
    User user=userRepository.findUserByUserName(userName);
    if(user==null)
    {
        throw new RuntimeException("Bad credentials");
    }
    if(user.getPassword().equals(password))
    {
        return user;
    }
    throw new RuntimeException("Bad credentials");
}
```

4. MAIN APPLICATION CLASS

LA CLASSE PRINCIPALE ANOADANOHAYLATP21APPLICATION QUI DÉMARRE L'APPLICATION SPRING BOOT ET INITIALISE LES DONNÉES DE BASE.

```
SpringBootApplication
public class AnoadaNohaylaTp22Application

    public static void main(String[] args)
    {
        SpringApplication.run(AnoadaNohaylaTp22Application.class, args);
    }

    @Bean
    CommandLineRunner start(UserService userService)
    {
        return args -> {
            Stream.of(...values: "admin", "user1", "user2", "user3", "user4", "user5")
                .forEach(name -> {
                    User u=new User();
                    u.setUserName(name);
                    u.setPassword("123456");
                    userService.addNewUser(u);
                });
            Stream.of(...values: "Admin", "Student", "User")
                .forEach(name -> {
                    Role r=new Role();
                    r.setRoleName(name);
                    userService.addNewRole(r);
                });
            userService.addRoleToUser(userName: "user1", roleName: "User");
            userService.addRoleToUser(userName: "user2", roleName: "User");
            userService.addRoleToUser(userName: "user3", roleName: "Student");
            userService.addRoleToUser(userName: "user4", roleName: "User");
            userService.addRoleToUser(userName: "user5", roleName: "Student");
            userService.addRoleToUser(userName: "admin", roleName: "Admin");

            try
            {
                User user=userService.authenticate(userName: "user1", password: "13456")
                System.out.println(user.getUserId());
                System.out.println(user.getUserName());
                System.out.println("Roles=>");
                user.getRoles().forEach(r-> {
                    System.out.println("Role=>" + r);
                });
            }
            catch(Exception e)
            {
                e.printStackTrace();
            }
        };
    }
}
```

5. WEB

LE CONTRÔLEUR USERCONTROLLER TRAITE LES REQUÊTES GET POUR OBTENIR LES DÉTAILS D'UN UTILISATEUR PAR SON NOM D'UTILISATEUR, CONVERTISSANT EN SUITE LES RÉSULTATS EN JSON POUR LA RÉPONSE.

```
@RestController
public class UserController
{
    @Autowired
    private UserService userService;

    @GetMapping("/users/{username}")
    public User user(@PathVariable String username)
    {
        User user =userService.findUserByUserName(username);
        return user;
    }
}
```



G. FICHIER DE CONFIGURATION

CES PROPRIÉTÉS SONT DES PARAMÈTRES DE CONFIGURATION POUR UNE APPLICATION SPRING BOOT UTILISANT UNE BASE DE DONNÉES H2 EN MÉMOIRE.

```
spring.application.name=Anoada_Nohayla_Tp2_2
spring.h2.console.enabled=true
spring.datasource.url=jdbc:h2:mem:Users_db
server.port=8088
spring.datasource.username=nohayla
spring.datasource.password=password
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
```