Ecole Normale Supérieure de
l'Enseignement Technique Mohammedia
Université Hassan II de Casablanca

ENSET

ا لمدرسة العليا لأساتذة التعليم التقني
المحمدية
جامعة الحسن الثاني بالدار البيضاء

DEPARTEMENT MATHEMATIQUES ET INFORMATIQUE

Compte rendu de l'activité pratique 8

Technologie Web JEE : http, Servlet, JSP, MVC Spring MVC, Thymeleaf
Parti III : authentification avec "User Details Service"

Filière :
« Génie du Logiciel et des Systèmes Informatiques Distribués »
GLSID2

Module : Architecture Distribuée et Middlewares

Élaboré par :

ELMAJNI Khaoula

Encadré par :

M. YOUSSFI Mohammed

Année Universitaire : 2021-2022

Introduction

Dans ce travail on va continuer au développement web avec le framework Spring et notamment son outil Spring Boot, et on va bien illustrer l'Architecture Web et comment l'échange des données se fait, et bien précisément l'architecture Web JEE avec le design pattern **Spring MVC**, et on va expérimenter cette architecture avec un rendu coté serveur avec le générateur de template Thymeleaf, aussi on va initier la sécurité avec Spring Security en faisant un système d'authentification.

Dans cette 3éme partie du développement web avec Spring Boot, on va faire une authentification avec avec le service « **User Details Service** » après qu'on a expérimenté les 2 méthodes :

- Authentification en mémoire
- Authentification avec JDBC

Description

Notre application une application qui permet de gérer des patients.

Chaque patient est défini par :

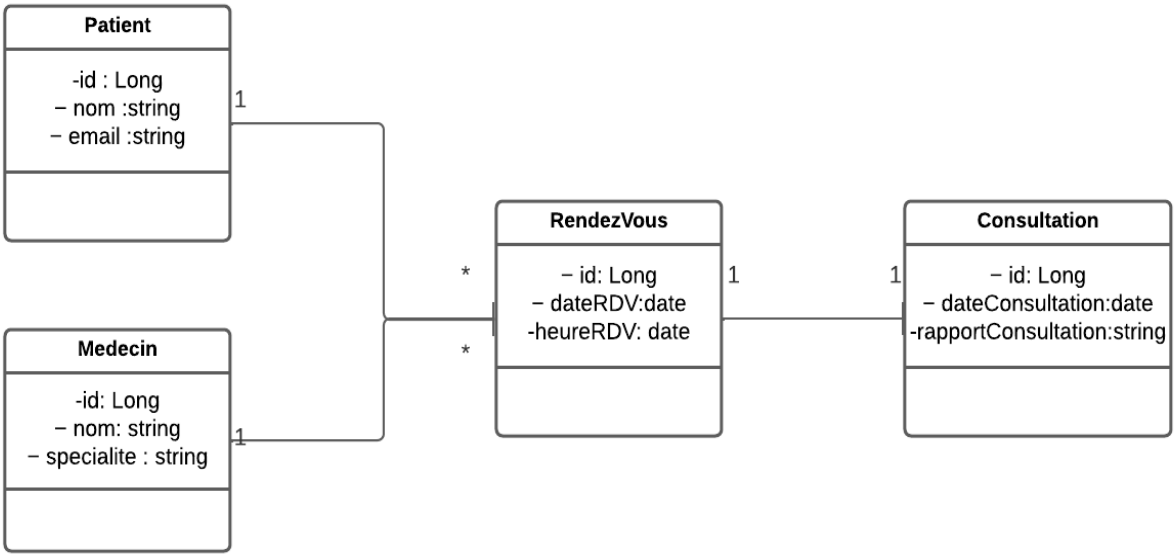
- Son ID de type Long
- Son Nom de type String
- Sa Date de naissance
- Un attribut qui indique si le patient est malade ou non
- Un score de type int

Les données sont stockées dans une base de données MySQL La couche web respecte MVC coté serveur.

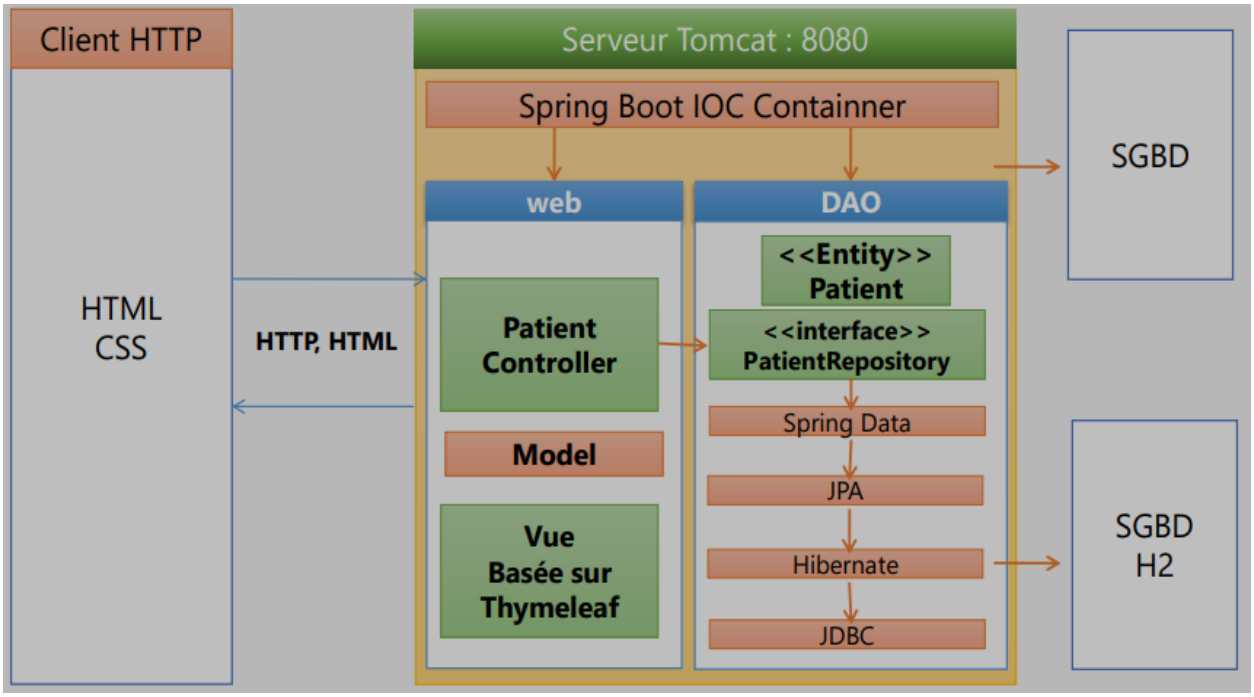
```
@Entity
@Data @NoArgsConstructor
@AllArgsConstructor
public class Patient {
    @Id @GeneratedValue(strategy =
GenerationType.IDENTITY)
    private Long id;
    @NotEmpty
    @Size(min = 3, max = 50)
    private String nom;
    @Temporal(TemporalType.DATE)
```

```
    @DateTimeFormat(pattern = "yyyy-MM-dd")
    private Date dateNaissance;
    private boolean malade;
    @DecimalMin("0")
    private int score;
}
```

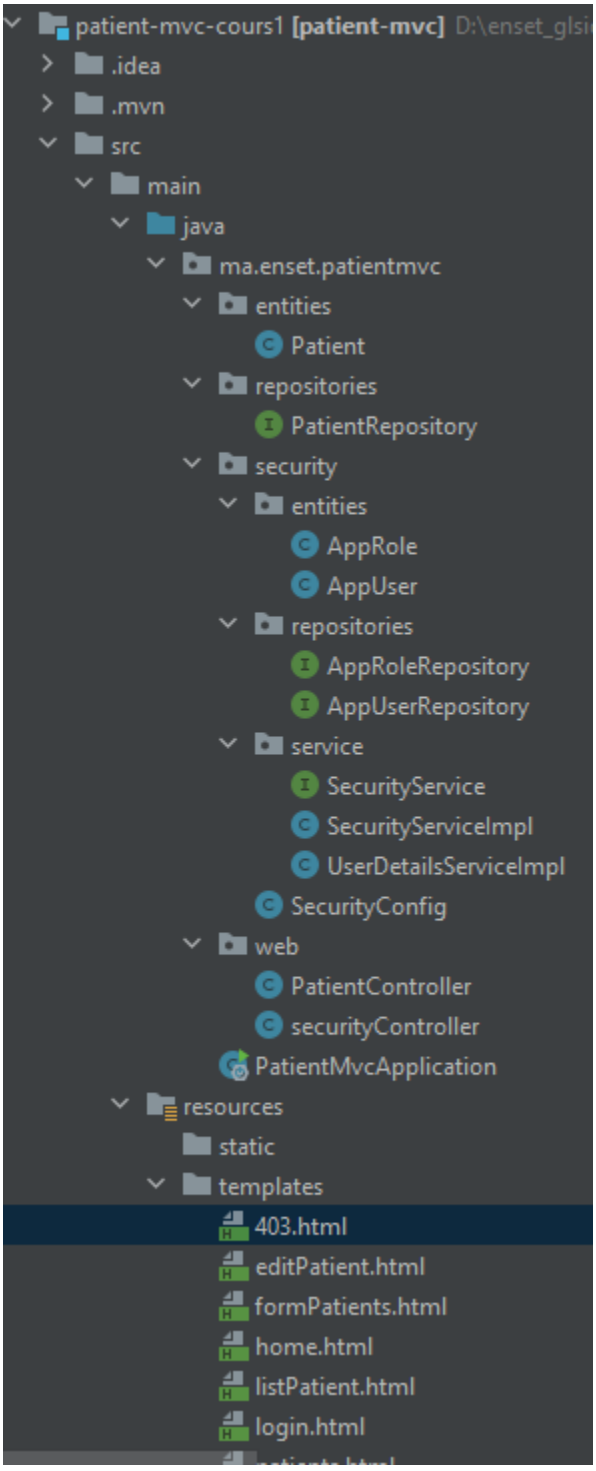
Le modèle conceptuel de données de l'application

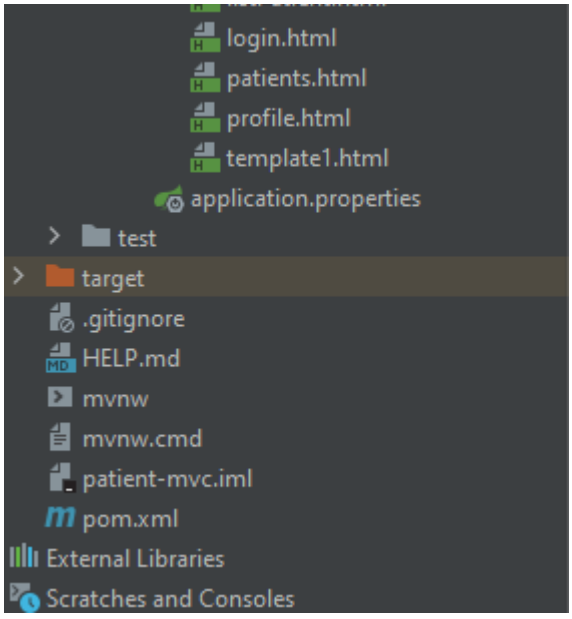


Architecture du projet



Structure du projet





Gestion d'authentification avec User Details Service

La classe d'utilisateurs:

```
@Entity
@Data @NoArgsConstructor @AllArgsConstructor
public class AppUser {
    @Id
    private String userId;
    @Column(unique = true)
    private String username;
    private String password;
    private Boolean active;
    @ManyToMany(fetch = FetchType.EAGER)
    private List<AppRole> appRoles = new ArrayList<>();
}
```

La classe des roles:

```
@Entity
@Data @NoArgsConstructor @AllArgsConstructor
public class AppRole {

    @Id @GeneratedValue(strategy =
GenerationType.IDENTITY)
    private Long roleId;
    @Column(unique = true)
    private String roleName;
    private String description;
}
```

Les interfaces 'repositories':

Qui vont declarer les signatures des méthodes nécessaires

```
public interface AppRoleRepository extends
JpaRepository<AppRole,Long> {
    AppRole findByRoleName(String roleName);
}
```

```
public interface AppUserRepository extends
JpaRepository<AppUser,String> {
    AppUser findByUsername(String username);
}
```

La sécurité:

Interface 'SecurityService' qui va declarer les signatures des méthodes pour les traitements des utilisateurs et ses roles:

```
public interface SecurityService {
    AppUser saveNewUser(String username,String password,
String rePwd);
    AppRole saveNewRole(String roleName, String
description);
    void addRoleToUser(String username, String roleName);
    void removeRoleFromUser(String username, String
roleName);
    AppUser loadUserByUsername(String username);
}
```

Implementation de l'interface 'SecurityService':

```
@Service
@Slf4j //permet de donner un attribut appelé log ==>
permet de loger
@AllArgsConstructor
@Transactional
public class SecurityServiceImpl implements
SecurityService {
    private AppUserRepository appUserReository;
    private AppRoleRepository appRoleRepository;
    private PasswordEncoder passwordEncoder;

    @Override
    public AppUser saveNewUser(String username, String
password, String rePwd) {
        if (!password.equals(rePwd)) throw new
RuntimeException("Password not match");
        String hashedPWD =
passwordEncoder.encode(password);
        AppUser appUser = new AppUser();
        appUser.setUserId(UUID.randomUUID().toString());
        appUser.setUsername(username);
        appUser.setPassword(hashedPWD);
```

```
        appUser.setActive(true);
        AppUser savedAppUser =
appUserReosistory.save(appUser);
        return savedAppUser;
    }

    @Override
    public AppRole saveNewRole(String roleName, String
description) {
        AppRole appRole =
appRoleRepository.findByRoleName(roleName);
        if (appRole != null)
            throw new RuntimeException("Role "+roleName+"
already exist!!!");
        appRole = new AppRole();
        appRole.setRoleName(roleName);
        appRole.setDescription(description);
        AppRole savedAppRole =
appRoleRepository.save(appRole);
        return savedAppRole;
    }
    @Override
    public void addRoleToUser(String username, String
roleName) {
        AppUser appUser =
appUserReosistory.findByUsername(username);
        if (appUser == null)
            throw new RuntimeException("User not found");
        AppRole appRole =
appRoleRepository.findByRoleName(roleName);
        if (appRole == null)
            throw new RuntimeException("Role not found");
        appUser.getAppRoles().add(appRole);
    }
    @Override
    public void removeRoleFromUser(String username,
String roleName) {
        AppUser appUser =
appUserReosistory.findByUsername(username);
        if (appUser == null)
            throw new RuntimeException("User not found");
```

```
        AppRole appRole =
appRoleRepository.findByRoleName(roleName);
        if (appRole == null)
            throw new RuntimeException("Role not found");
        appUser.getAppRoles().remove(appRole);
    }
    @Override
    public AppUser loadUserByUsername(String username) {
        return appUserReository.findByUsername(username);
    }
}
```

La classe 'UserDetailsServiceImpl' qui hérite de l'interface 'UserDetailsService':

```
@Service
public class UserDetailsServiceImpl implements
UserDetailsService {
    @Autowired
    private SecurityService securityService;

    /*public UserDetailsServiceImpl(SecurityService
securityService) {
        this.securityService = securityService;
    }*/

    @Override
    public UserDetails loadUserByUsername(String
username) throws UsernameNotFoundException {
        AppUser appUser =
securityService.loadUserByUsername(username);
        //en utilisant la programmation imperative
classique
        /*Collection<GrantedAuthority> authorities = new
ArrayList<>();
        appUser.getAppRoles().forEach(role->{
            SimpleGrantedAuthority authority = new
SimpleGrantedAuthority(role.getRoleName());
            authorities.add(authority);
        });*/

        //en utilisant l'API des streams
```

```
        Collection<GrantedAuthority> authorities1 =
appUser
                .getAppRoles()
                .stream()
                .map(role-> new
SimpleGrantedAuthority(role.getRoleName()))
                .collect(Collectors.toList());

        User user = new
User(appUser.getUsername(),appUser.getPassword(),authorit
ies1);
        return user;
    }
}
```

Ensuite dans la classe de configuration de sécurité on va déclarer un objet de 'UserDetailsServiceImpl' et on va faire un l'injection des dépendances avec l'annotation @Autowired:

```
public class SecurityConfig extends
WebSecurityConfigurerAdapter {
      //code...
    @Autowired
    private PasswordEncoder passwordEncoder;
      //code...
}
```

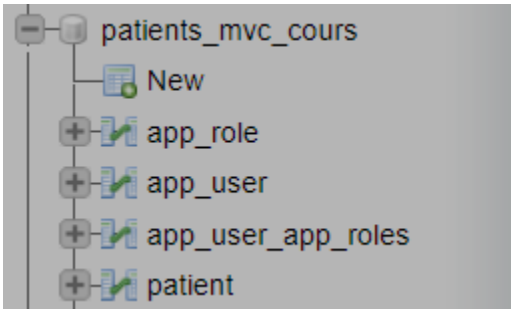
Puis dans la méthode 'configure' :

```
protected void configure(AuthenticationManagerBuilder
auth) throws Exception {
    /* la stratégie comment spring security va chercher
les users*/
    //authentification avec User Details Service
    auth.userDetailsService(userDetailsService);

}
```

la base de des données:

les tables qui ont utilisés dans l'authentification par la suite:



La tables des utilisateurs:

	user_id	active	password	username
Edit Copy Delete	1856a929-7610-4e01-b2d4-e1ee05374cbe	1	$2a$10$/O94g3PM7huB7s6ekLspZ.38fkB5Apj1wte6X7j0WGp...	khaoula
Edit Copy Delete	81542843-f074-4a8e-9f4e-b57be4e8b55b	1	$2a$10$uQbsexAok4.cbA6q.g388.eZdPeW26VlcGlB9tj.nXO...	ahmed
Edit Copy Delete	922c660e-f73e-4bd4-b8b6-5a1957e094fd	1	$2a$10$zfNGUQkvL/AIX/o9bfYCkeUMjD17FVPJKVifyTBXmQ/...	mohammed

La tables des roles:

	role_id	description	role_name
Edit Copy Delete	1		USER
Edit Copy Delete	2		ADMIN

La tables d'association:

app_user_user_id	app_roles_role_id
1856a929-7610-4e01-b2d4-e1ee05374cbe	1
1856a929-7610-4e01-b2d4-e1ee05374cbe	2
922c660e-f73e-4bd4-b8b6-5a1957e094fd	1
81542843-f074-4a8e-9f4e-b57be4e8b55b	1

Conclusion

Ce travail nous a permis de bien connaitre l'architecture web, et on a initié avec l'architecture web JEE et le design pattern Spring MVC, et on a travaillé avec le rendu coté serveur avec le moteur de template thymeleaf, aussi l'authentification avec Spring Security en utilisant le service **'User Details Service'**.