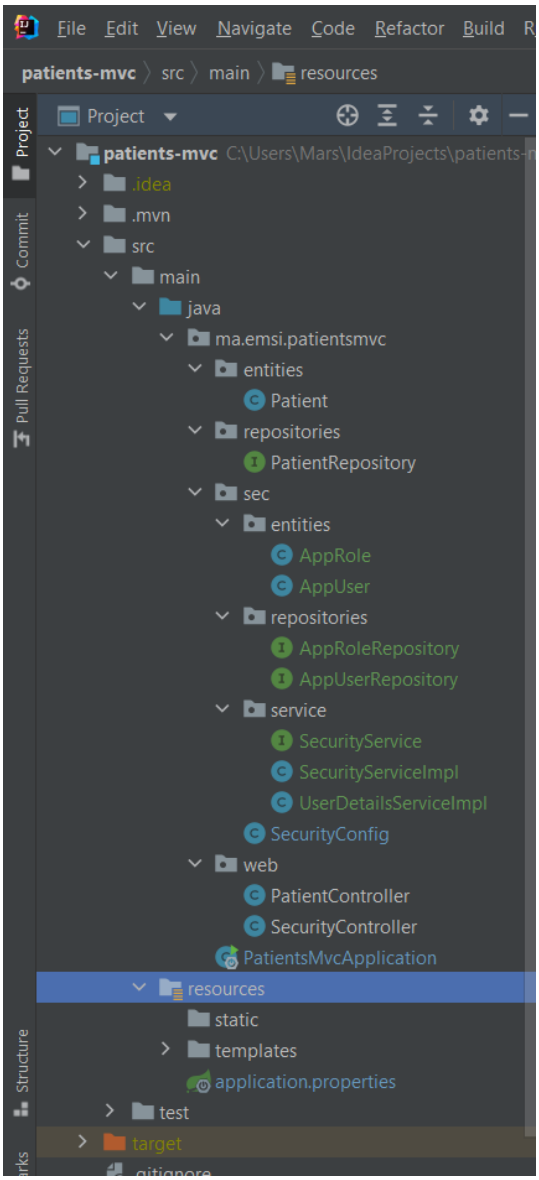
Compte Rendu Controle:

Objectif : Développer une application web de Gestion des Patients basée sur Spring MVC ,Thymeleaf et Spring Data .

Partie 1 :

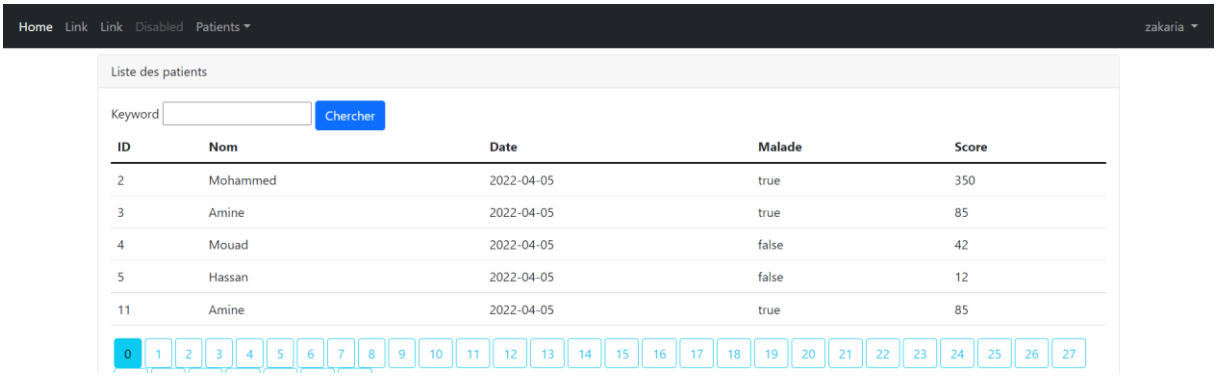
a)Rechercher les patients :

En premier lieu on crée notre projet nommé Patients -mvc sur Intellij avec la structure suivante , on utilisé une base de données h2-database puis on migré vers mysql :



Pour la recherche des patients et la pagination on l'a realisé avec la méthode patients qui prend les paramètres suivants et nous retourne une liste de patients qu'on affiche dans la page patients.html

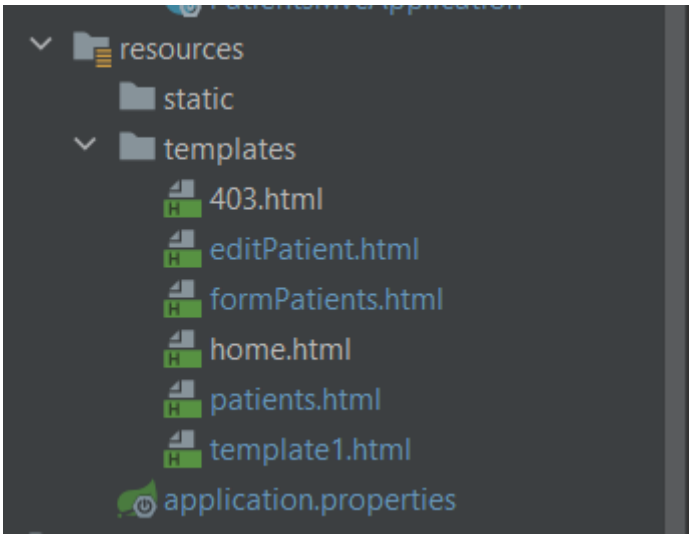
```
public String patients(Model model,
                       @RequestParam(name="page",defaultValue = "0") int page,
                       @RequestParam(name="size",defaultValue = "5")int size,
                       @RequestParam(name = "keyword",defaultValue = "")String keyword
){
    Page<Patient> pagePatients = patientRepository.findByNomContains(keyword,PageRequest.of(page,size));
    model.addAttribute( attributeName: "listPatients",pagePatients.getContent());
    model.addAttribute( attributeName: "pages",new int[pagePatients.getTotalPages()] );
    model.addAttribute( attributeName: "currentPage",page);
    model.addAttribute( attributeName: "keyword",keyword);
    return "patients";
}
```



Et seul l'administrateur supprimer un patient , ce qui sera expliqué par la suite

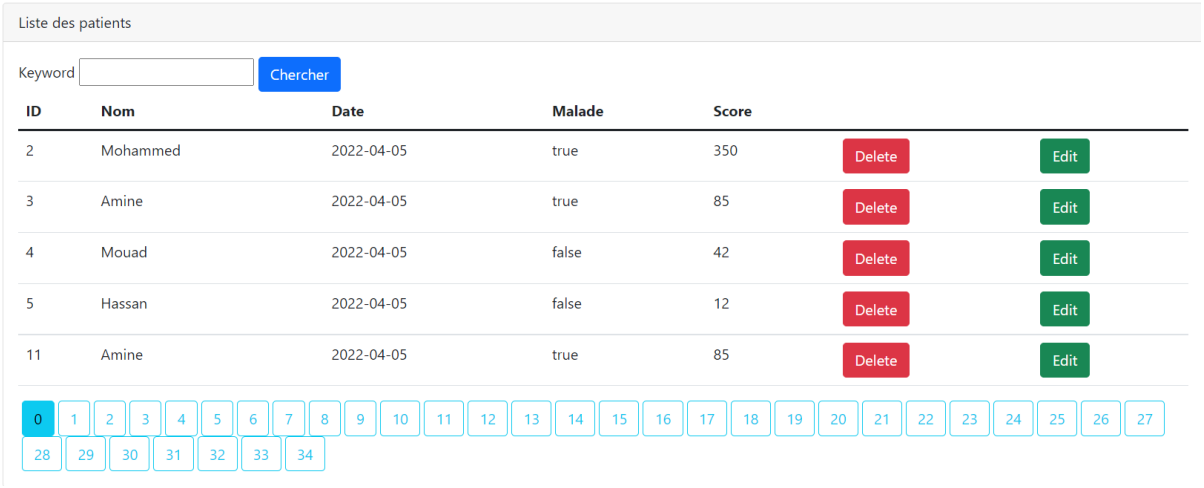
Partie 2 : Page template, Ajout des patients, validation des formulaires, édition et mise à jour des patients

Création du dossier Templates qui regroupera les fichiers html utilisés pour : la template de base , le formulaire d'ajout et modification des patients , la page home ainsi que la page d'erreur 403



a) Ajout des patients et validation des formulaires

Comme étant mentionné avant seul un administrateur a le droit de modifier ou supprimer un patient de la liste :



Partie 3 : Spring Security Stratégies : InMemoryAuthentication et JDBCAuthentication

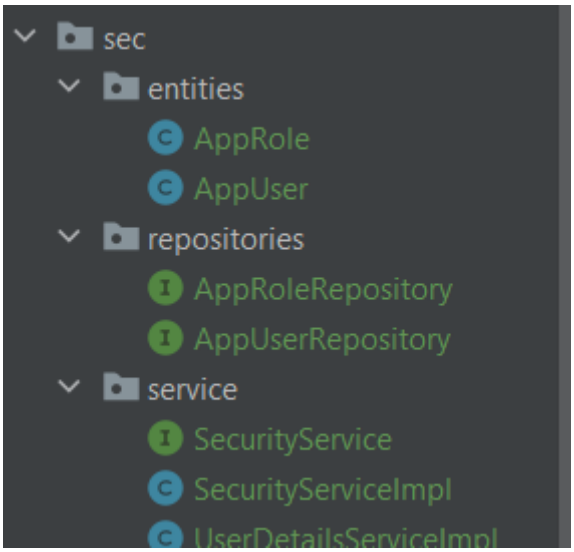
On a utilisé deux stratégies d'authentification , la première est InMemoryAuthentication avec laquelle on crée quelques utilisateurs qui seront stockés dans la mémoire avec leur username et pwd qui serviront pour s'authentifier , chaque user aura son rôle : le premier un user normal et l'autre un administrateur/user

Et JDBCAuthentication qui a comme source de data la base de donnée déjà crée pat-db et recherche le username et pwd passé lors de la création

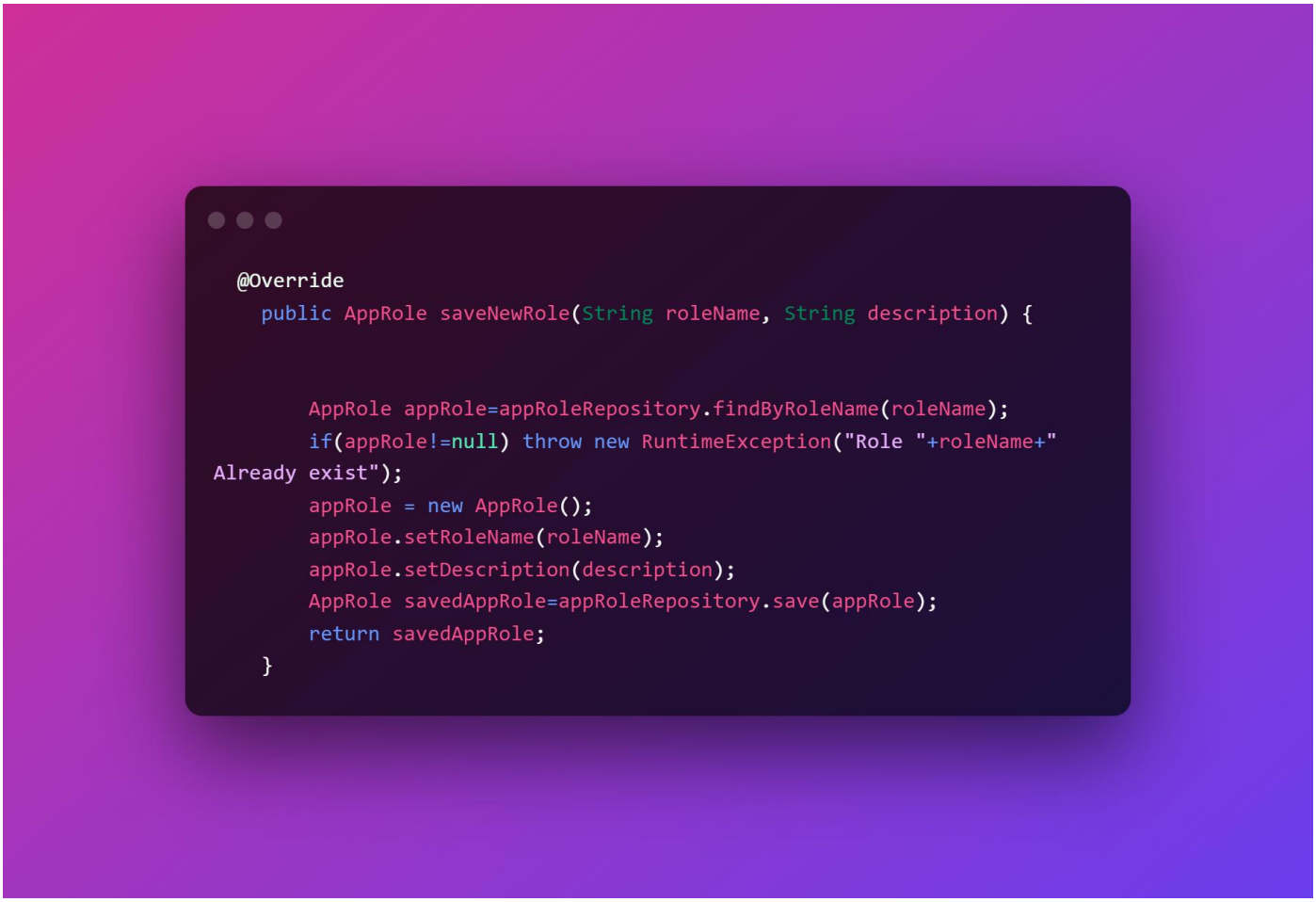
```
/*
String encodedPWD=passwordEncoder.encode("1234");
System.out.println(encodedPWD);
auth.inMemoryAuthentication().withUser("user1").password(encodedPWD).roles("USER");
auth.inMemoryAuthentication().withUser("user2").password(passwordEncoder.encode("1111")).roles("USER");
auth.inMemoryAuthentication().withUser("admin").password(passwordEncoder.encode("2345")).roles("USER","ADMIN");

/*auth.jdbcAuthentication()
        .dataSource(dataSource)
        .usersByUsernameQuery("select username as principal, password as credentials, active from users where username=?")
        .authoritiesByUsernameQuery("select username as principal, role as role from users_roles where username=?")
        .rolePrefix("ROLE_")
        .passwordEncoder(passwordEncoder);*/
auth.userDetailsService(userDetailsService);
```

Partie 4 : Spring Security (Stratégie : UserDetailsService)



Ayant separé le role et le user ainsi que les répositories , en ajoutant les services Security et son implémentation qui contient les méthodes qu'on a redéfinit depuis l'interface


@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;
}