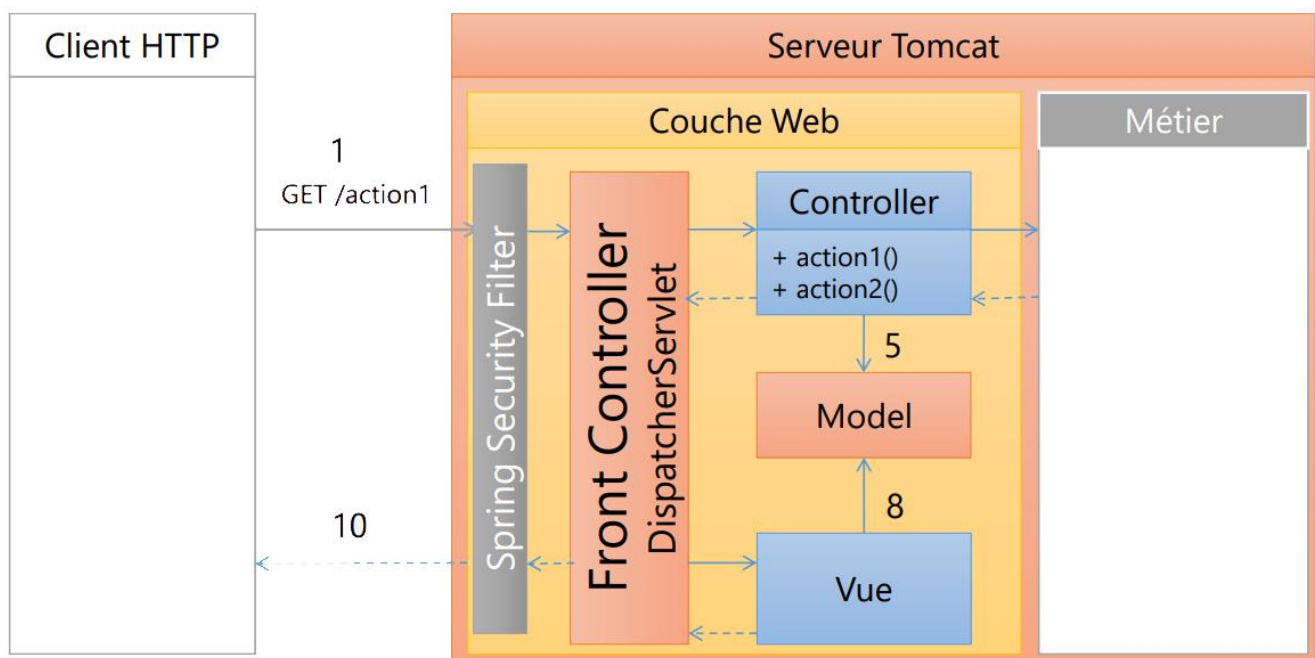




CR DE L'ACTIVITE PRATIQUE : SPRING MVC THYMELEAF

Filière : « Ingénierie Informatique : Big Data et Cloud
Computing » II-BDCC



Réalisé par :

Khadija BENJILALI

Encadré par :

Mohamed YOUSSEFI

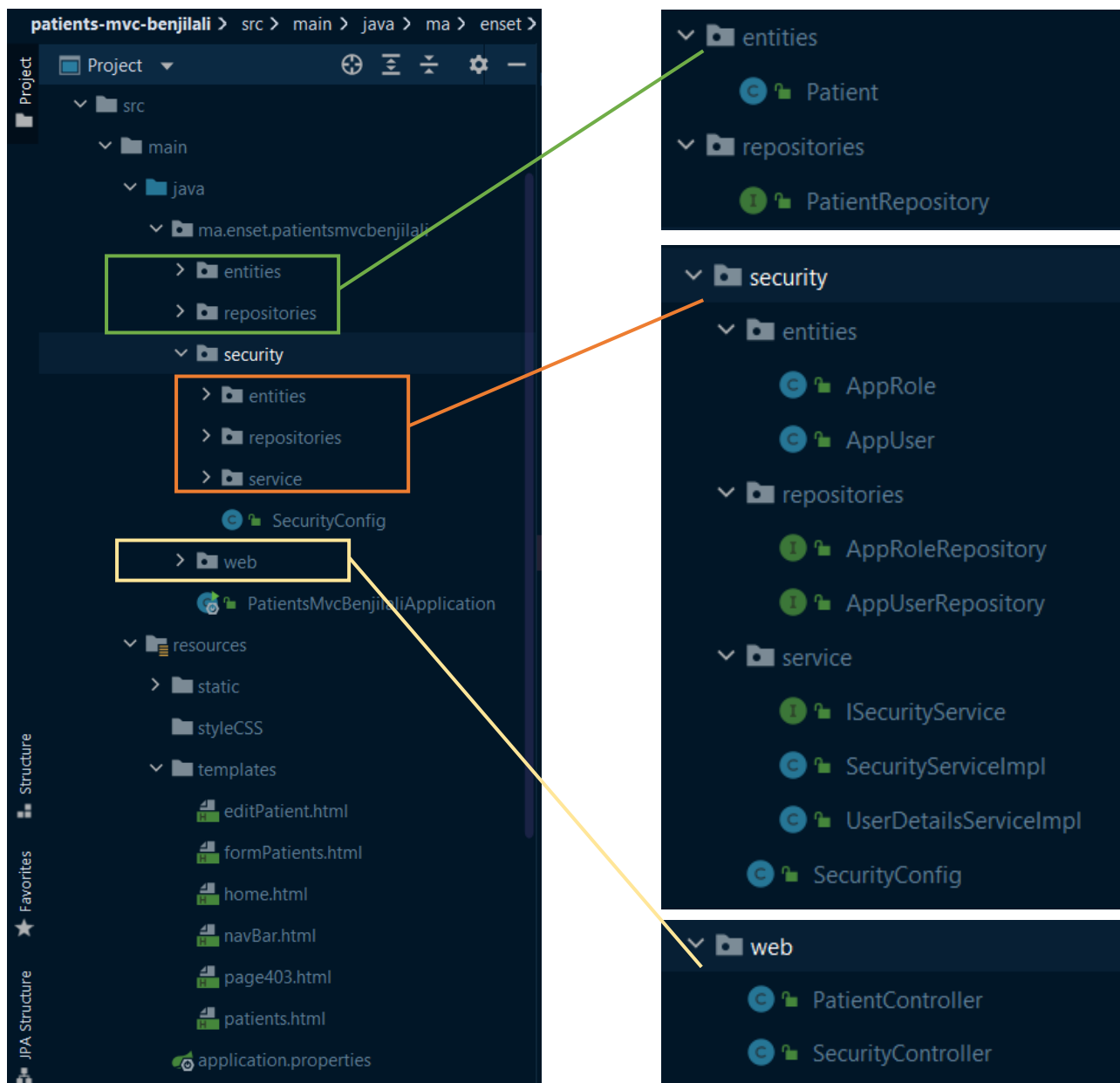
Année Universitaire : 2021-2022

◆ Enoncé

Création d'une application Web JEE basée sur Spring MVC, Thymeleaf et Spring Data JPA qui permet de gérer les patients. L'application doit permettre les fonctionnalités suivantes :

- **Afficher les patients**
- **Faire la pagination**
- **Chercher les patients**
- **Supprimer un patient**
- **Faire des améliorations supplémentaires**

◆ Structure du projet



◆ Fichier de dépendances Maven : pom.xml

```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
  <dependency>
    <groupId>nz.net.ultraq.thymeleaf</groupId>
    <artifactId>thymeleaf-layout-dialect</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
    <optional>true</optional>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-validation</artifactId>
  </dependency>
</dependencies>
```

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<dependency>
  <groupId>org.thymeleaf.extras</groupId>
  <artifactId>thymeleaf-extras-springsecurity5</artifactId>
</dependency>
<!--<dependency>
  <groupId>com.h2database</groupId>
  <artifactId>h2</artifactId> <scope>runtime</scope>
</dependency-->
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
</dependency>
<dependency>
  <groupId>org.projectlombok</groupId>
  <artifactId>lombok</artifactId>
  <optional>true</optional>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-test</artifactId>
  <scope>test</scope>
</dependency>
</dependencies>
```

◆ Application.properties

Fichier de configuration de l'application Spring Boot. Fichier lu au démarrage de Spring

```
1  #spring.datasource.url=jdbc:h2:mem:patients_db_khadija
2  #spring.h2.console.enabled=true
3  spring.datasource.url=jdbc:mysql://localhost:3306/patients_db_ben?createDatabaseIfNotExist=true
4  spring.datasource.username=root
5  spring.datasource.password=
6  spring.jpa.hibernate.ddl-auto=update
7  spring.jooq.sql-dialect=org.hibernate.dialect.MariaDBDialect
8  server.port=8082
9  spring.jpa.show-sql=true
10 #spring.thymeleaf.cache=false
11 #spring.main.allow-circular-references=true
```

◆ Application Spring Boot

```
15 @SpringBootApplication
16 public class PatientsMvcBenJilaliApplication {
17
18     public static void main(String[] args) { SpringApplication.run(PatientsMvcBenJilaliApplication.class, args); }
21
22     @Bean
23     PasswordEncoder passwordEncoder(){
24         return new BCryptPasswordEncoder();
25     }
26 }
```

```

// @Bean
CommandLineRunner start(PatientRepository patientRepository){
    return args -> {
        Stream.of("Ahlam", "Salma", "Amjad", "Mohammed").forEach(nom -> {
            patientRepository.save(new Patient(id: null, nom, new Date(), Math.random() < 0.5 ? false : true, score: 129));
        });
        patientRepository.findAll().forEach(p -> {
            System.out.println(p.getNom());
        });
    };
}

// @Bean
CommandLineRunner saveUsers(ISecurityService iSecurityService){
    return args -> {
        iSecurityService.saveNewUser(username: "khadija", password: "1234", rePassword: "1234");
        iSecurityService.saveNewUser(username: "Akram", password: "1234", rePassword: "1234");
        iSecurityService.saveNewUser(username: "Hassan", password: "1234", rePassword: "1234");

        iSecurityService.saveNewRole(roleName: "ADMIN", description: "");
        iSecurityService.saveNewRole(roleName: "USER", description: "");

        iSecurityService.addRoleToUser(username: "khadija", roleName: "ADMIN");
        iSecurityService.addRoleToUser(username: "khadija", roleName: "USER");
        iSecurityService.addRoleToUser(username: "Akram", roleName: "USER");
        iSecurityService.addRoleToUser(username: "Hassan", roleName: "USER");
    };
}

```

◆ Couche DAO

➤ Entité JPA : Patient

```

@Entity
@Data @NoArgsConstructor @AllArgsConstructor
public class Patient
{
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @NotEmpty
    @Size(min = 4, max = 40)
    private String nom;

    @Temporal(TemporalType.DATE)
    @DateTimeFormat(pattern = "yyyy-MM-dd")
    private Date dateNaissance;
    private boolean malade;
    @DecimalMin("100")
    private int score;
}

```

➤ Interfaces DAO basées sur Spring Data

Ajouter une méthode à l'interface JpaRepository qui permet de retourner une page de patients ont le nom contient un mot clé :

```

@Repository
public interface PatientRepository extends JpaRepository<Patient, Long> {
    Page<Patient> findByNomContains(String kw, Pageable pageable);
}

```

◆ Test de la couche DAO

| | | | | id | date_naissance | malade | nom | score |
|--------------------------|----------|----------|-------------|----|----------------|--------|----------|-------|
| <input type="checkbox"/> | ✎ Éditer | ✚ Copier | 🗑 Supprimer | 1 | 2022-04-18 | 1 | Ahlam | 129 |
| <input type="checkbox"/> | ✎ Éditer | ✚ Copier | 🗑 Supprimer | 2 | 2022-04-18 | 0 | Salma | 129 |
| <input type="checkbox"/> | ✎ Éditer | ✚ Copier | 🗑 Supprimer | 3 | 2022-04-18 | 0 | Amjad | 129 |
| <input type="checkbox"/> | ✎ Éditer | ✚ Copier | 🗑 Supprimer | 4 | 2022-04-18 | 0 | Mohammed | 129 |

◆ Couche Web

➤ Templates

Les vues de l'application qui seront interprétées côté serveur par le moteur de vue Tymeleaf :

➤ Contrôleur

```
//@RestController => pour la couche côté client c-à-d le serveur return Json est le client qui va récupérer html
@Controller
@AllArgsConstructor
public class PatientController
{
    private PatientRepository patientRepository;

    @GetMapping(path = "/user/index")
    public String patientsFunction(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( attributeNames: "listePatients", pagePatients.getContent());
        model.addAttribute( attributeNames: "pagesNumber", new int[pagePatients.getTotalPages()]);
        model.addAttribute( attributeNames: "currentPage", page);
        model.addAttribute( attributeNames: "keyword", keyword);
        //return une vue qui s'appelle patients.html
        return "patients";
    }
}
```

```

@GetMapping("/{id}/admin/delete")
public String deleteFunction(Long id,
                             @RequestParam(defaultValue = "") String keyword,
                             @RequestParam(defaultValue = "0") int page){
    patientRepository.deleteById(id);
    return "redirect:/user/index?page="+page+"&keyword="+keyword;
}

@GetMapping("/") public String home() { return "home"; }

// Rendu coté client retourner une liste Json des patients
// @RestController
@GetMapping("/{id}/user/patients")
@ResponseBody //obligatoire
public List<Patient> patientList(){
    //DispatcherServlet comprendre qu'il s'agit pas d'une vue(String) et il va convertir en format Json
    return patientRepository.findAll();
}

@GetMapping("/{id}/admin/formPatients")
public String formPatients(Model model){
    model.addAttribute( attributeName: "patientform", new Patient());
    return "formPatients";
}

//utiliser pour ajouter et pour faire la mise à jour
//si l'id est null => ajouter
//sinon => modifier
@PostMapping(path = "{id}/admin/save")
public String save(Model model, @Valid Patient patient, BindingResult bindingResult,
                    @RequestParam(defaultValue = "") String keyword,
                    @RequestParam(defaultValue = "0") int page){
    //valeur par défaut @RequestParam
    if(bindingResult.hasErrors())
    {if(patient.getId()==null) return "formPatients";
      else return "editPatient";
    }
    patientRepository.save(patient);
    return "redirect:/user/index?page="+page+"&keyword="+keyword;
}

@GetMapping(path = "{id}/admin/editPatient")
public String editPatient(Model model, Long id, String keyword, int page){
    Patient patient = patientRepository.findById(id).orElse( other: null);
    if(patient==null) throw new RuntimeException("Patient introuvable");
    model.addAttribute( attributeName: "patientEdit", patient);
    model.addAttribute( attributeName: "page", page);
    model.addAttribute( attributeName: "keyword", keyword);
    return "editPatient";
}

```

◆ Ressources :

➤ Utilisation des Layouts :

Templates Généralement toutes les page d'une application web partagent le même contenu html (Header, footer, menus, etc..) Pour éviter des faire des copies coller dans toutes les pages, il est important de définir une page template « **navBar.html** »

qui définit • toutes les parties fixes de toutes les pages (header, footer, menus, etc...) et déclarer les sections qui changeront de contenu en fonction de chaque page.

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org"
      xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
      xmlns:sec="http://www.thymeleaf.org/extras/spring-security">
<head...>
<body>

<nav class="navbar navbar-expand-lg navbar-dark bg-dark">
  <div class="container-fluid">
</nav>

<section layout:fragment="content1">

</section>
</body>
</html>
```

➤ Vue : patients.html

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org"
      xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
      xmlns:sec="http://www.thymeleaf.org/extras/spring-security"
      layout:decorate="navBar">
<head...>
<body>
<div layout:fragment="content1">
  <div class="container mt-4">
    <div class="card">
      <div class="card-header">Liste des patients</div>
      <div class="card-body">
        <form method="get" th:action="@{/user/index}" class="form-control mb-4" >
          <label class="col-form-label">Key word</label>
          <input type="text" name="keyword" th:value="${keyword}" style="padding: 4px; margin: 5px">
          <button type="submit" class="btn btn-primary" >Chercher</button>
        </form>
        <table class="table">
        <nav aria-label="..." style="margin-top: 40px">
      </div>
    </div>
  </div>
</div>
</body>
</html>
```

➤ Autres vues

```
▼ templates
  editPatient.html
  formPatients.html
  home.html
  navBar.html
  page403.html
  patients.html
```

◆ SPRING SECURITY

Spring Security est un module de Spring qui permet de sécuriser les applications Web. Spring Security configure des filtres qui permet d'intercepter les requêtes HTTP et de vérifier si l'utilisateur authentifié dispose des droits d'accès à la ressource demandée.

Les actions du contrôleur ne seront invoquées que si l'utilisateur authentifié dispose de l'un des rôles attribués à l'action.

➤ SecurityConfig.java

```
import javax.sql.DataSource;

@Configuration
@EnableWebSecurity //va etre instancié en premier
public class SecurityConfig extends WebSecurityConfigurerAdapter
{
    @Autowired
    private DataSource dataSource ;
    @Autowired
    private UserDetailsServiceImpl userDetailsService;
    @Autowired
    private PasswordEncoder passwordEncoder;

    //pour spécifier les users
    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception
    {
        /*
        String encodePWD = passwordEncoder.encode("1234");
        System.out.println("=====> " + encodePWD);
        //stocker dans la mémoire
        auth.inMemoryAuthentication().withUser("user1")
            .password(passwordEncoder.encode("1111")).roles("USER")
            .and().withUser("user2").password(encodePWD).roles("USER");

        auth.inMemoryAuthentication().withUser("admin").password(encodePWD).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);
    }

    //pour spécifier les droit d'accès de chaque user
    @Override
    protected void configure(HttpSecurity http) throws Exception
    {
        http.formLogin(); //je veux utiliser un formulaire d'authentification fournit par Spring Security
        //http.formLogin().loginPage("/login"); ma propre formulaire (créer méthode dans controller)

        http.authorizeRequests().antMatchers("/").permitAll();
        http.authorizeRequests().antMatchers("/admin/**").hasAuthority("ADMIN");
        http.authorizeRequests().antMatchers("/user/**").hasAuthority("USER");
        http.authorizeRequests().antMatchers("/webjars/**").permitAll();

        http.authorizeRequests().anyRequest().authenticated();
        // http.authorizeHttpRequests().anyRequest().authenticated(); //toutes les requetes http nécessite une authentification

        http.exceptionHandling().accessDeniedPage("/403"); // page 403
    }
}
```

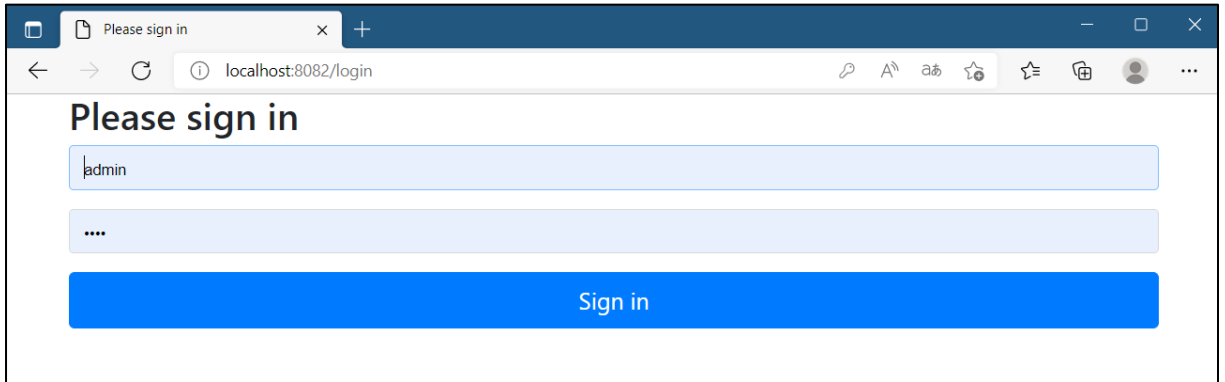
/* Pour le cas ou les utilisateurs sont connues et fixes d'une manière statique */

/* Pour le cas ou les utilisateurs sont stockés dans une base la même base de données de l'application */

/* Toutes les requêtes HTTP avec URL /admin/** nécessitent d'être authentifié avec un Utilisateur ayant le rôle ADMIN*/

/* Toutes les requêtes HTTP avec URL /user/** nécessitent d'être authentifié avec un utilisateur ayant le rôle USER */

➤ Interfaces de Spring Security de l'authentification



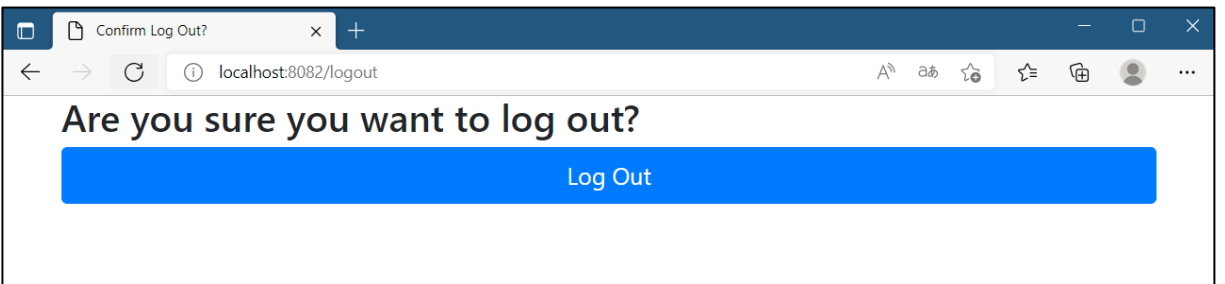
A browser window showing the Spring Security login page. The address bar displays 'localhost:8082/login'. The page title is 'Please sign in'. It features a text input field containing 'admin', a password input field with masked characters '....', and a blue 'Sign in' button.

Please sign in

admin

....

Sign in

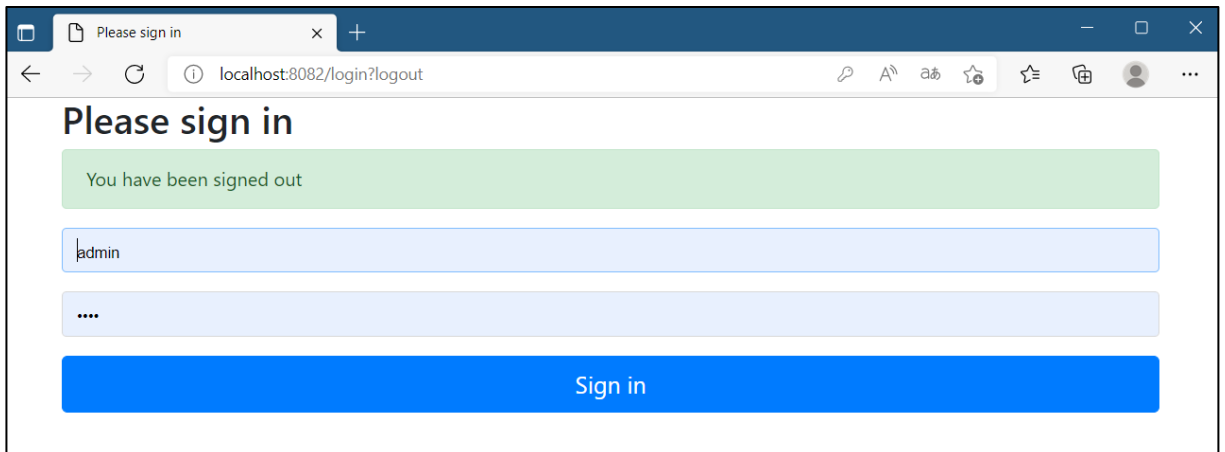


A browser window showing the Spring Security logout confirmation page. The address bar displays 'localhost:8082/logout'. The page title is 'Confirm Log Out?'. It features a blue 'Log Out' button.

Confirm Log Out?

Are you sure you want to log out?

Log Out



A browser window showing the Spring Security login page after a successful logout. The address bar displays 'localhost:8082/login?logout'. The page title is 'Please sign in'. A green message box at the top states 'You have been signed out'. Below it are the same login fields as the first screenshot: a text input field containing 'admin', a password input field with masked characters '....', and a blue 'Sign in' button.

Please sign in

You have been signed out

admin

....

Sign in