VP Испит Прирачник

0 Setup

1. Отвори го фајлот **pom.xml** од задачата со InteliJ.
2. Направи Build на проектот.
3. Ако има некој error со dependencies, File → Project Structure → Project SDK
4. Статртувај го проектот.
5. Во /tests/SeleniumTest поставете го вашиот индекс **SubmissionHelper.*index* = "TODO";**

1 Анотирање на /config/DataInitializer

1. **@Component** над DataInitializer.
2. **@PostConstruct** над функција која ги креира податоците која сакаме да се повика при секое стартување на апликацијата (најчесто **init()** или **initData()**).

2 Намести /models/exceptions

1. Ако е генерален exception:
   public InvalidUserCredentialsException() {
   super("Invalid username or password.");}
2. Ако exception вклучува некој id, username, email или др (%s ако е од тип String):
   public InvalidProductIdException(Long id) {
   super(String.*format*("No product with Id = %d found.", id));}

3 Анотирање на /models

1. **@Data** над секоја класа во /models (само ако во класата нема get и set функции за сите променливи), освен врз **public enum**.
2. **@Entity** над секоја класа во /models, освен врз **public enum**.
3. **@Id** над секој примарен клуч на entity.
4. **@GeneratedValue** ако примарниот клуч е автоматски генериран.
5. **@Enumerated(EnumType.*STRING*)** ако се користи променливата за енумерација.
6. **@ManyToOne**, **@OneToMany** за релации меѓу табелтите.
7. **@ManyToMany(fetch = FetchType.*EAGER*)** ако користиме мора да потенцираме каков FetchType има инаку нема да ни работи апликацијата.
8. **@Table(name="shop_user")** ако табелата се вика User мора да се употреби ова за rename.

Дали има custom User model?

Во зависност од тоа дали имаме custom User model, security делот треба да се додаде една од следните функции:

Ако во /models има класа која што има атрибути: username или email и password, тогаш најчесто треба да ја употребиме таа класа како custom User model и да овозможиме најава на корисници со неа (пишува и во security барањето). За да работи security ти треба и кодот од **4**, **7.2**, **9.2**.

Ако во /models нема класа која што има атрибути: username или email и password, тогаш треба да овозможиме најава на корисници со inMemory база на податоци. За да работи security ти треба и кодот од **9.1**.

4 Намести Enumeration

(ако има custom User model)

Ако имаме custom User model и ако во класата има променлива од тип енумерација која се користи да ја чува улогата на корисникот, тогаш:

1. Кај **enum** ставаме **implements GrantedAuthority**
2. Во **enum** пред сите улоги/типови ставаме ***ROLE_***. Пример: ***ROLE_ADMIN***.
3. Во **enum** ја додаваме функцијата:
   ```
   @Override
   public String getAuthority() {
     return name();}
   ```

5 Намести JPARepository во /repository

1. Сите класи во овој пакет треба да се interface, а не класа.
2. **extends JpaRepository<Product, Long>** секој interface мора да наследува од оваа класа. Long е од кој тип е id на Product.

6 Намести PasswordEncoder во ExampleApplication

1. Под функцијата **void main()** стави ја следната функција:
   ```
   @Bean
   public PasswordEncoder passwordEncoder(){
     return new BCryptPasswordEncoder();}
   ```

7 Намести /service/impl

Ако нема, во /service додади пакет /impl и во него додади ги класите според interface во /service.

1. **@Service** над секоја класа во /service/impl.
2. **implements ProductService** каде што се дефинира секоја класа во /service/impl, “Product” зависи од името на entity.
3. **private final ProductRepository productRepository;**
   public ProductServiceImpl(ProductRepository productRepository) {
   this.productRepository = productRepository;}
   за да вклучиме зависност од репозиториум слојот.
4. **Alt + Enter** за имплементација на сите методи од service.

A. **findById()**: внатер во функцијата кога се користи repository за findById, мора да се фрли exception.
.findById(id).orElseThrow(()->new InvalidProductIdException(id));

B. **listAll()**:
.findAll();

C. **create()**:
Product product = new Product(вредности);
return productRepository.save(product);

D. **update()**:
Product product = findById(id);
product.setName(name);
product.setPrice(price);
return productRepository.save(product);

E. **delete()**:
Product product = findById(id);
productRepository.delete(product);
return product;

7.1 Филтер функција во /service/impl

Најчесто во /service/imp класа ќе имаме функција која треба да ги излиста сите објекти од таа класа според некои 2 критериуми. Во функцијата имаме 3 различни if проверки и 4 различни return:

1. if(name == null && categoryId == null)
   return productRepository.findAll();
2. if(name == null)
   return productRepository.findAllByCategoriesContaining(category);
3. if(categoryId == null)
   return productRepository.findAllByNameLike(nameSearch);
4. return productRepository.findAllByNameLikeAndCategoriesContaining(nameSearch, category);

Функциите кои ќе ги користиме во return потребно е да ги дефинираме во соодветниот JPARepository.

7.2 User Service

(ако има custom User model)

Ако имаме custom User модел тогаш треба следното да го направиме во /service/impl класата на custom User model:

1. Ја додаваме зависноста **PasswordEncoder**:
   private final PasswordEncoder passwordEncoder;
   public UserServiceImpl(PasswordEncoder passwordEncoder) {
   this.passwordEncoder = passwordEncoder;}
2. Во /service/impl ако има функции каде што се работи со password треба да користиме **passwordEncoder.encode(password);** најчесто во **create()** и **update()**.
3. На декларацијата на сервисот се додава implements UserDetailsService.
4. **Alt + Enter** за да ја додадеме функцијата loadUserByUsername().
5. Функцијата loadUserByUsername(String s):
   User user = userRepository.findByUsername(s).orElseThrow(()-> new InvalidUsernameException(s));
   return new org.springframework.security.core.userdetails.User(
   user.getUsername(),
   user.getPassword(),
   Collections.*singletonList*(user.getRole()));

- Ако корисникот треба да се login со email, ќе употребиме **.findByEmail(s)** наместо **.findByUsername(s)**.
- Во **return** ќе употребиме **.getEmail()** наместо .getUsername() .
- **Collections.*singletonList*(user.getRole())** се користи за да се земе ROLE на User, ако ROLE е од тип **enum**.
  Ако role е од тип String тогаш над **return** ставаме:
  GrantedAuthority grantedAuthority = new SimpleGrantedAuthority(user.getRole());
  List<GrantedAuthority> authorities = Collections.*singletonList*(grantedAuthority);

- Во JPARepository кој е соодветен на service-от кој implements UserDetailsService треба да ја додадеме **findBy**(Username или Email зависно од задачата) функција.

8 Најчести controller функции

List all:

1. Ја анотираме фунцкијата со **@GetMapping()** и соодветните url адреси, најчесто {"/", "/products"}.
2. Сите променливи кои ги прима функцијата ги анотираме со **@RequestParam(required = false)**.
3. Додаваш **Model model** како влезна променлива.

4. Ги додаваш динамичките податоци во модел **model.addAttribute("products", products);**, за да се прикажат на страницата.
5. Ако сакаме да додадеме динамички податоци од тип **enum** тогаш model.addAttribute("positions", PlayerPosition.*values*());
6. Ја прикажуваш страницата **return "list";** (најчесто е list.html).
7. Во **list.html** додаваме name="" атрибут на филтер **<input>** полињата.
8. Во **list.html** во филтер **<form>** тагот НЕ додаваме ништо бидејќи не правиме POST барање.
9. Во **list.html** ако сакаш да се креира нов елемент за секој објект тогаш th:each="product : ${products}"
   Најчесто во **<tr>** и **<option>**.
10. Во **list.html** се додава **th:text="${product.price}"** ако сакаме да прикажеме некој текст од променливите,
    th:text="${genre.name}" за користење со **enum**.
11. Во **list.html th:value="${product.id}"** најчесто се користи кога треба да селектираме нешто кое треба да има динамичка вредност.
    th:value="${genre.toString()}" за користење со **enum**.

GET Add Form:

1. Ја анотираме фунцкијата со **@GetMapping()** и соодветните url адреси, најчесто {"/products/add"}.
2. Додаваш **Model model** како влезна променлива.
3. Додаваш динамички податоци **model.addAttribute("categories",this.categoryService.listAll());** , најчесто ако треба да селектираш од некоја листа на објекти.
4. **return "form";** го прикажуваме form.html
5. Во **list.html** во **<a>** тагот каде што е копчето ADD ќе додадеш **href=""** во согласност со URL патеката на controller функцијата.
6. Во **form.html** каде што ќе имаш селектирање на некој тип на објект во **<option>** тагот ставаме **th:each="cat:${categories}"**, **th:value** и **th:text**.

POST Add Object:

1. Ја анотираме фунцкијата со **@PostMapping()** и соодветните url адреси, најчесто {"/products"}.
2. **@RequestParam** се анотираат сите влезни променливи во функцијата.
3. **@RequestParam** + @DateTimeFormat(iso = DateTimeFormat.ISO.*DATE*) ако влезната променлива е од тип LocalDate или LocalDateTime.
4. this.service.create() се креира нов објект.
5. **return "redirect:/products";** откако ќе се submit form, треба да се направи редирекција кон друга страница.
6. Во **form.html** во **<form>** се додава th:method="POST"
7. Во **form.html** во **<form>** се додава th:action="@{'/products'}" за да се користи за create.

8. Во **form.html** сите **<input>** ќе им дадеш name="" атрибут соодветен на името на влезните променливи кои ги прима функцијата.

GET Edit Form:

1. Ја анотираме фунцкијата со **@GetMapping()** и соодветните url адреси, најчесто {"/products/{id}/edit"}.
2. Додаваме **@PathVariable** пред **Long id** и додаваме **Model model**.
3. **model.addAttribute("product", this.service.findById(id));** за да ги прикажеме податоците за product кој сакаме да го изменуваме.
4. Додаваш динамички податоци на form model.addAttribute("categories",this.categoryService.listAll()); , најчесто ако треба да селектираш објект од некоја листа.
5. **return "form";** се користи истата форма за Edit и за Add.
6. Во **list.html** во **<a>** тагот каде што се наоѓа EDIT доадаваме **th:href="@{'/products/{id}/edit' (id=${product.id})}"**.
7. Во **form.html** за секое **<input>** поле **th:value="${product?.name}"** name е во зависност од полето.
8. Ако сакаме елементот од листата да е веќе селектиран и ако тој елемент е објект во **form.html** во **<option>** поле додаваме **th:selected="${product != null and product.category != null and product.category.id == cat.id}"**
   Ако елементот во **<select>** е **enum** тогаш користиме **th:selected="${product != null and product.type == type}"**

POST Edit Object:

1. Функцијата се анотира со **@PostMapping({"/products/{id}"})**.
2. **Long id** се анотира со **@PathVariable**.
3. Сите влезни промеливи освен **Long id** се анотираат со **@RequestParam**.
4. **@RequestParam** + @DateTimeFormat(iso = DateTimeFormat.ISO.*DATE*) ако влезната променлива е од тип LocalDate или LocalDateTime.
5. Се користи this.service.update().
6. return "redirect:/products"; се прави редирекција.
7. Во **form.html** во **<form>** се полето **th:action** се менува во **th:action="@{${product!=null} ? '/products/' + ${product.id} : '/products' }"** за да се знае дали **form.html** се користи за create или update на објект.

POST Delete Object:

1. Ако на **list.html** DELETE копчето е во **<form>** коритиме **@PostMapping**, ако е во **<a>** таг тогаш користиме **@GetMapping**.
2. **({"/products/{id}/delete"})** секогаш прима URL со ID на обиектот кој треба да се избрише.
3. **@PathVariable Long id** се анотира влезното ID.
4. Најчесто прави redirect до list страницата **return"redirect:/products";**.

5. Во **list.html** во **<form>** каде што се наоѓа копчето DELETE додаваме **th:method="POST"** и **th:action="@{'/products/{id}/delete' (id=${product.id})}"** или
   Во **list.html** во **<a>** тагот на DELETE додаваме **th:href="@{'/products/{id}/delete' (id=${product.id})}"**

POST Vote Object:

1. Функцијата се анотира со **@PostMapping({"/products/{id}/vote"})**.
2. **Long id** се анотира со **@PathVariable**.
3. Ја користиме фунцкијата **this.playerService.vote(id);**.
4. Се прави redirect кон list страницата **return "redirect:/products";**.
5. Во **list.html** во **<form>** тагот каде што се наоѓа копчето VOTE додаваме th:action="@{'/products/{id}/vote' (id=${product.id})}" и th:method="POST"

9A Spring Security

(ако класата SecurityConfig налседува од WebSecurityConfigurerAdapter**)**

1. Во функцијата configure(WebSecurity web) искоментрија ја линијата web.ignoring().antMatchers("/**");.
2. Креирај нова функција:

```
@Override
protected void configure(HttpSecurity http) throws Exception {
  http
     .csrf().disable()
        .authorizeRequests()
        .antMatchers("/").permitAll()
        .antMatchers("/products").permitAll()
        .antMatchers("/products/**/vote").hasRole("USER")
        .anyRequest().hasRole("ADMIN")
        .and()
     .formLogin().permitAll()
        .failureUrl("/login?error=BadCredentials")
        .defaultSuccessUrl("/products", true)
        .and()
     .logout()
        .clearAuthentication(true)
        .invalidateHttpSession(true)
        .deleteCookies("JSESSIONID")
        .logoutSuccessUrl("/");}
```

   Оваа функција ги има сите општи работи за config на SpringSecurity.
3. Во **list.html** ако сакаме html елемент да е видлив само за одреден тип на корисник во тагот на тој елемент саваме **sec:authorize="hasRole('ROLE_ADMIN')"**.

'ROLE_ADMIN' е ако сакаме да е видлив само за ADMIN, 'ROLE_USER' е ако сакаме да е видлив само за USER.
Ако се **enum** користи кој implements GrantedAuthority тогаш во **list.html** може да се користи sec:authorize="hasAuthority('ADMIN')", а во configure **.hasAuthority("USER")**

- **.antMatchers("/products").authenticated()** се користи ако сакаме само логирани корисници да имаат пристап до таа страна.

4. Ја додаваме функцијата од **9.1** или **9.2** (зависи од задачата).

9B Spring Security

(ако класата наследува од SecurityFilterChain
или не налседува од ништо**)**

1. Ја анотираме класата со **@EnableWebSecurity**
2. Ја коментриаме **@Bean** анотацијата над public **WebSecurityCustomizer webSecurityCustomizer()** функцијата.
3. Ја додаваме функцијата:

```
@Bean
public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception
{
  http.csrf(AbstractHttpConfigurer::disable)
        .authorizeRequests( (requests) -> requests
              .antMatchers("/")
              .permitAll()
              .anyRequest()
              .hasRole("ADMIN"))
        .formLogin((form) -> form
              .permitAll()
              .failureUrl("/login?error=BadCredentials")
              .defaultSuccessUrl("/products", true))
        .logout((logout) -> logout
              .logoutUrl("/logout")
              .clearAuthentication(true)
              .invalidateHttpSession(true)
              .deleteCookies("JSESSIONID")
              .logoutSuccessUrl("/"));
  return http.build();}
```

4. Ја додаваме функцијата од **9.1** или **9.2** (зависи од задачата).

9.1 Spring security+

(ако нема custom User model)

Овој код се користи само ако податоците за корисниците треба да ни доаѓаат од inMemory (нема ниеден /service/impl кој што имплементира **UserDetailsService**).

1. Вклучи **private final PasswordEncoder passwordEncoder;** зависност користејќи конструктор.
2. Ја додаваме функцијата:
   @Override
   protected void configure(AuthenticationManagerBuilder auth) throws Exception {
   auth
   .inMemoryAuthentication()
   .withUser("admin").password(passwordEncoder.encode("admin")).roles("ADMIN")
   .and()
   .withUser("user").password(passwordEncoder.encode("user")).roles("USER");}
   Со оваа фунцкија можеме да се log in. Кредeнцијали (user, user), (admin, admin).

9.2 Spring security+

(ако има custom User model)

1. Ги следиме чекорите во **4** и **7.2**
2. Во SecurityConfig вклучуваме зависност **private final UserDetailsService userDetailsService;**.
3. Во SecurityConfig класта ја додаваме функцијата:
   public AuthenticationManager authManager(HttpSecurity http) throws Exception {
   AuthenticationManagerBuilder authenticationManagerBuilder =
   http.getSharedObject(AuthenticationManagerBuilder.class);
   authenticationManagerBuilder.userDetailsService(userDetailsService);
   return authenticationManagerBuilder.build();}

10 Тестови

1. Направив run на сите Selenium тестови.
2. Ако има errors провери ги URL адресите кои се користат во тестовите.

Со среќа колеги и колешки