# Spring Fundamentals Exam

# Philately Application

Exam for the ["Spring Fundamentals" course @ SoftUni](https://judge.softuni.org/Contests/4821/Spring-Fundamentals-Retake-Exam-15-August-2024).

***Philately*** *is the collection and study of postage stamps and related postal items. It is a hobby enjoyed by people of all ages worldwide, and it can be both a leisure activity and an academic pursuit. Philately began shortly after introducing the first postage stamp, the Penny Black, in 1840 in Great Britain. As postal systems expanded and evolved, so did the hobby of collecting stamps.*

*Start->*

password: the best23  
username: root

And RUN

There are several requirements you must follow in the implementation:

## Database Requirements

The **Database** of the **Philately** application needs to support **3 entities**:

Next ->

For all ->

@Builder  
@AllArgsConstructor  
@NoArgsConstructor  
@Getter  
@Entity

Без таблица, без конструктор и гетъри и сетъри, зададени са от горните анотации!

### User

* Has an **Id – UUID – this!**
* Has a Username (unique, not null)
  + Username length must be between 3 and 20 characters (inclusive of 3 and 20).
* Has a Password (not null)
  + Password length must be between 3 and 20 characters (inclusive of 3 and 20).
* Has an Email (unique, not null)
  + Must contain '@'.
* Has a List<Stamp> list of the **added** stamps
* Has a List<WishedStamp> list of the wished stamps

@OneToMany(mappedBy = "owner", fetch = FetchType.*EAGER*)  
private List<Stamp> stamps;  
  
@OneToMany(mappedBy = "owner", fetch = FetchType.*EAGER*)  
private List<WishedStamp> wishedStamps;

### Stamp

* Has an **Id – UUID**
* Has a Name (not null)
  + Name length must be between 5 and 20 characters (inclusive of 5 and 20).
* Has a Description (not null) *(a brief description of what the stamp depicts)*
  + Description length must be between 5 and 25 characters (inclusive of 5 and 25)
* Has a Paper (not null) *(the type of paper on which the stamp is printed)* enumeration with values:
  + For WOVE\_PAPER – "Has an even texture without any particular distinguishing features."
  + For LAID\_PAPER – "When held up to the light, shows parallel lines of greater or less width running across the stamp."
  + For GRANITE\_PAPER – "Has tiny specks of coloured fibre in it, which can usually be seen with the naked eye." – описанията няма нужда да ги правим за изпита!
* Has an **Image URL** (not null)
  + Valid image URL containing no more than 150 characters.
* Has an **Owner**
  + A stamp has one Owner, the User who added it.

### WishedStamp

* Has an **Id – UUID**
* Has a Name (not null)
  + Name length must be between 5 and 20 characters (inclusive of 5 and 20).
* Has a Description (not null) *(a brief description of what the stamp depicts)*
  + Description length must be between 5 and 25 characters (inclusive of 5 and 25)
* Has an **Image URL** (not null)
  + Valid image URL containing no more than 150 characters.
* Has an **Owner – who is owning this wished stamp**.

Run и проверяваме в майескюел има ли нещо изобко!

## Page Requirements

Next ->

In index ->

<h3>  
 <a class="primary-text" th:href="@{/login}">Login</a>  
 if you have an account or  
 <a class="primary-text" th:href="@{/register}">Register</a>  
 if you don't have one.  
</h3>

Next ->

In commons

<div class="links">  
 <li>  
 <a th:href="@{/login}">Login</a>  
 </li>  
  
 <li>  
 <a th:href="@{/register}">Register</a>  
 </li>  
  
 <li>  
 <a th:href="@{/logout}">Logout</a>  
 </li>  
</div>

Next -> видзуализация на login register and logout

@Controller  
public class IndexController {  
  
 @GetMapping("/")  
 public String getIndexPage() {  
  
 return "index";  
 }  
  
 @GetMapping("/login")  
public ModelAndView getLoginPage(){  
  
 ModelAndView modelAndView = new ModelAndView();  
 modelAndView.setViewName("login");  
  
 return modelAndView;  
}  
  
@GetMapping("/register")  
public ModelAndView getRegisterPage(){  
  
 ModelAndView modelAndView = new ModelAndView();  
 modelAndView.setViewName("register");  
  
 return modelAndView;  
}  
  
@GetMapping("/logout")  
public String getLogout(HttpSession session){  
  
 session.invalidate();  
  
 return "redirect:/";  
}

And RUN – кликваме на всяко меню от трите вдясно горе и двете в средата и всяко трябва да показва полета за попълване.

### Register Page (logged out user)

After successful registration, the user should be redirected to the **login** page.

Next ->

Create DTO -> и изписване на предупреждения при невалидни данни

@Data

public class RegisterRequest {

@NotBlank(message = "Username cannot be empty")  
 @Size(min = 3, max = 20, message = "Username must be between 3 and 20 characters")  
 private String username;  
 @NotBlank(message = "Email cannot be empty")  
 @Email(message = "Not a valid email format")  
 private String email;  
 @NotBlank(message = "Password cannot be empty")  
 @Size(min = 3, max = 20, message = "Password must be between 3 and 20 characters")  
 private String password;  
 @NotBlank(message = "Password cannot be empty")  
 @Size(min = 3, max = 20, message = "Password must be between 3 and 20 characters")  
 private String confirmPassword;  
}

Next ->

@GetMapping("/register")  
public ModelAndView getRegisterPage(){  
  
 ModelAndView modelAndView = new ModelAndView();  
 modelAndView.setViewName("register");  
  
 modelAndView.addObject("registerRequest", new RegisterRequest());  
  
 return modelAndView;  
}

Next -> in register – оправяме съобщения за грешки и път към полетата! На всички!

<form class="welcome login" th:method="POST" th:action="@{/register}" th:object="${registerRequest}">  
 <h1 class="text-center mt-2"><span class="badge badge-pill badge-dark">Register</span></h1>

<input type="text" class="form-control" id="username" name="username" th:field="\*{username}"/>  
 <small th:if="${#fields.hasErrors('username')}" th:errors="\*{username}" class="bg-danger text-light rounded" style="font-size: 15px">Username error message</small>  
</div>  
  
<div class="form-group">  
 <div class="label-holder text-white textCol d-flex justify-content-center">  
 <label for="email" class="h4 mb-2">Email</label>  
 </div>  
 <input type="email" class="form-control" id="email" name="email" th:field="\*{email}"/>  
 <small th:if="${#fields.hasErrors('email')}" th:errors="\*{email}" class="bg-danger text-light rounded">Email error message</small>  
</div>  
  
<div class="form-group">  
 <div class="label-holder text-white textCol d-flex justify-content-center">  
 <label for="password" class="h4 mb-2">Password</label>  
 </div>  
 <input type="password" class="form-control" id="password" name="password" th:field="\*{password}"/>  
 <small th:if="${#fields.hasErrors('password')}" th:errors="\*{password}" class="bg-danger text-light rounded">Password error message</small>  
</div>  
  
<div class="form-group">  
 <div class="label-holder text-white textCol d-flex justify-content-center">  
 <label for="confirmPassword" class="h4 mb-2">Confirm Password</label>  
 </div>  
 <input type="password" class="form-control" id="confirmPassword" name="confirmPassword" th:field="\*{confirmPassword}"/>  
 <small th:if="${#fields.hasErrors('confirmPassword')}" th:errors="\*{confirmPassword}" class="bg-danger text-light rounded">Confirm Password error message</small>  
</div>

Next ->

@Controller  
public class IndexController {  
  
 private final UserService userService;  
  
 @Autowired  
 public IndexController(UserService userService) {  
 this.userService = userService;  
 }

@PostMapping("/register")  
public String doRegister(@Valid RegisterRequest registerRequest, BindingResult bindingResult){  
  
  
 if (bindingResult.hasErrors()){  
  
 return "register";  
 }  
  
 userService.register(registerRequest);  
  
 return "redirect:/login";  
}

Next -> Тук ще обработим новия потребител, когато искаме да го регистрираме!

@Service  
public class UserService {  
  
 public void register(RegisterRequest registerRequest){

}  
  
  
}

Next ->

@Repository  
public interface UserRepository extends JpaRepository<User, UUID> {   
   
}

@Repository  
public interface WishedStampRepository extends JpaRepository<WishedStamp, UUID> {  
}

@Repository  
public interface StampRepository extends JpaRepository<Stamp, UUID> {  
  
  
}

Next -> Сега трябва да проверим в сървис дали новия потребител го има – виждаме кои са му уникалните променливи – това са името и имейла. Тоест тях ще проверим има ли ги или не и тогава ще го регистрираме!

@Service  
public class UserService {  
  
 private final UserRepository userRepository;

private final PasswordEncoder passwordEncoder;

@Autowired  
 public UserService(UserRepository userRepository, PasswordEncoder passwordEncoder) {  
 this.userRepository = userRepository;  
 this.passwordEncoder = passwordEncoder;  
 }  
  
 public void register(RegisterRequest registerRequest){  
  
 Optional<User> optionalUser = userRepository.findByUsernameOrEmail(registerRequest.getUsername(), registerRequest.getEmail());  
  
 if (optionalUser.isPresent()){  
 throw new RuntimeException("User with this email/username already exist.");  
 }  
  
  
  
 }  
}

Next ->

@Repository  
public interface UserRepository extends JpaRepository<User, Long> {  
  
  
 Optional<User> findByUsernameOrEmail(String username, String email);

Next -> in Config

@Configuration  
public class BeanConfig {  
  
 @Bean  
 public PasswordEncoder encoder(){  
 return new BCryptPasswordEncoder();  
 }  
}

Next -> UserService

public void register(RegisterRequest registerRequest){  
  
Optional<User> optionalUser = userRepository.findByUsernameOrEmail(registerRequest.getUsername(), registerRequest.getEmail());  
  
if (optionalUser.isPresent()){  
 throw new RuntimeException("User with this email/username already exist.");  
}  
  
User user = User.*builder*()  
 .username(registerRequest.getUsername())  
 .email(registerRequest.getEmail())  
 .password(passwordEncoder.encode(registerRequest.getPassword()))  
 .build();

userRepository.save(user);  
  
}

AND RUN и проверяваме дали изписва грешки при неправилни данни в полетата!

A screenshot of a computer

Description automatically generated

### Register Page validations

A screenshot of a phone

Description automatically generated

### Index Page (logged out user)

A screenshot of a cell phone

Description automatically generated

### Login Page (logged out user)

Next ->

@Data  
public class LoginRequest {  
  
 @NotBlank(message = "Username cannot be empty")  
 @Size(min = 3, max = 20, message = "Username must be between 3 and 20 characters")  
 private String username;  
  
 @NotBlank(message = "Password cannot be empty")  
 @Size(min = 3, max = 20, message = "Password must be between 3 and 20 characters")  
 private String password;  
  
  
}

Next ->

@GetMapping("/login")  
public ModelAndView getLoginPage(){  
  
 ModelAndView modelAndView = new ModelAndView();  
 modelAndView.setViewName("login");  
  
 modelAndView.addObject("loginRequest", new LoginRequest());  
  
 return modelAndView;  
}

Next -> in login

<form class="welcome login" th:method="POST" th:action="@{/login}" th:object="${loginRequest}">

Преместваме този ред –

<div class="bg-danger text-light rounded">Error messages</div>

Тук –

<input type="text" class="form-control" id="username" name="username"/>  
 <div class="bg-danger text-light rounded">Error messages</div>  
</div>

И попълваме –

<input type="text" class="form-control" id="username" name="username" th:field="\*{username}"/>  
 <div th:if="${#fields.hasErrors('username')}" th:errors="\*{username}" class="bg-danger text-light rounded">Error messages</div>  
</div>

<input type="password" class="form-control" id="password" name="password" th:field="\*{password}"/>  
 <div th:if="${#fields.hasErrors('password')}" th:errors="\*{password}" class="bg-danger text-light rounded">Error messages</div>  
</div>

Next ->

@PostMapping("/login")  
  
public String loginUser(@Valid LoginRequest loginRequest, BindingResult bindingResult) {  
  
 if (bindingResult.hasErrors()) {  
 return "login";  
 }  
  
 User user = userService.login(loginRequest);  
  
 return "redirect:/home";  
}

Next -> UserService

public User login(LoginRequest loginRequest) {  
  
 Optional<User> optionalUser = userRepository.findByUsername(loginRequest.getUsername());  
  
 if (optionalUser.isEmpty()){  
 throw new RuntimeException("Incorrect username or password.");  
 }  
  
 User user = optionalUser.get();  
 if (!passwordEncoder.matches(loginRequest.getPassword(), user.getPassword())){  
 throw new RuntimeException("Incorrect username or password.");  
 }  
  
 return user;  
}

Next -> in IndexController – Щом се логне потребител, активираме сесия!

@PostMapping("/login")  
  
public String loginUser(@Valid LoginRequest loginRequest, BindingResult bindingResult, HttpSession session) {  
  
 if (bindingResult.hasErrors()) {  
 return "login";  
 }  
  
 User user = userService.login(loginRequest);  
 session.setAttribute("user\_id", user.getId());  
  
 return "redirect:/home";  
}

When initializing a new user account, enabling an HTTP session and assigning the "user\_id" attribute to the session for proper user identification and management is essential!

Next -> in IndexController

@GetMapping("/home")  
public ModelAndView getHomePage(HttpSession session){  
  
 UUID userId = (UUID) session.getAttribute("user\_id");  
 User user = userService.getById(userId);  
  
 ModelAndView modelAndView = new ModelAndView();  
 modelAndView.setViewName("home");  
  
 modelAndView.addObject("user", user);  
  
 return modelAndView;  
  
}

Next -> in UserService

public User getById(UUID userId) {

return userRepository.findById(userId).orElseThrow(() -> new RuntimeException("User with id [%s] does not exist.".formatted(userId)));

}

And RUN

A screenshot of a login page

Description automatically generated

### Login Page validations

A screenshot of a login screen

Description automatically generated

### Home Page (without having any stamps when a logged-in user)

Next ->

Първо оправяме пътищата – какво виждаме когато сме логнати и ако не сме !!!

<header>  
 <nav>  
 <div class="home">  
 <a href="/">Philately App</a>  
 <a th:if="${user}" href="/">Add Stamp</a>  
 </div>  
 <div class="links">  
 <li th:unless="${user}">  
 <a th:href="@{/login}">Login</a>  
 </li>  
  
 <li th:unless="${user}">  
 <a th:href="@{/register}">Register</a>  
 </li>  
  
 <li th:if="${user}">  
 <a th:href="@{/logout}">Logout</a>  
 </li>  
 </div>  
 </nav>  
</header>

And RUN

Next ->

A screenshot of a computer

AI-generated content may be incorrect.

### Add stamp (when a logged-in user)

Next -> in commons

<a th:if="${user}" th:href="@{/stamps/new}">Add Stamp</a>

Next ->

@Controller  
@RequestMapping("/stamps")  
public class StampController {  
  
  
 private final UserService userService;  
  
 @Autowired  
 public StampController(UserService userService) {  
 this.userService = userService;  
 }  
  
  
 @GetMapping("/new")  
 public ModelAndView getNewStampPage(HttpSession session){  
  
 UUID userId = (UUID) session.getAttribute("user\_id");  
 User user = userService.getById(userId);  
  
 ModelAndView modelAndView = new ModelAndView();  
 modelAndView.setViewName("add-stamp");  
  
 modelAndView.addObject("user", user);

*//добавяме и потребителя, защото това определя какво ще виждаме в хедъра като бутони, когато сме в стамп страницата*

return modelAndView;  
 }  
  
  
}

A screenshot of a computer screen

AI-generated content may be incorrect.

### Add stamp validation

Next -> dto

@Data  
public class CreateNewStamp {  
  
 @NotBlank  
 @Size(min = 5, max = 20, message = "Name length must be between 5 and 20 characters!")  
 private String name;  
  
 @NotBlank  
 @Size(min = 5, max = 25, message = "Description length must be between 5 and 25 characters!")  
 private String description;  
  
 @NotBlank(message = "Please enter a valid image URL!")  
 @URL(message = "Please enter a valid image URL!")  
 private String imageUrl;  
  
 @NotNull(message = "You must select a type of paper!")  
 private Paper paper;  
  
  
}

Next ->

@GetMapping("/new")  
public ModelAndView getNewStampPage(HttpSession session){  
  
 UUID userId = (UUID) session.getAttribute("user\_id");  
 User user = userService.getById(userId);  
  
 ModelAndView modelAndView = new ModelAndView();  
 modelAndView.setViewName("add-stamp");  
  
 modelAndView.addObject("user", user);  
 *//добавяме и потребителя, защото това определя какво ще виждаме в хедъра като бутони, когато сме в стамп страницата* modelAndView.addObject("createNewStamp", new CreateNewStamp());  
  
 return modelAndView;  
}

Next -> Когато имаме селектиране на категория марки примерно , изтриваме тук редовете с изброените категории и е показано тук какво се пише!

<header th:replace="~{fragments/commons::nav}"></header>  
<main>  
 <form class="welcome add-offer-form" th:method="POST" th:action="@{/stamps}" th:object="${createNewStamp}">  
 <h1 class="text-center mt-2"><span class="badge badge-pill badge-dark">Add stamp</span></h1>  
  
 <div class="form-group">  
 <div class="label-holder text-white textCol d-flex justify-content-center">  
 <label for="name" class="h4 mb-2">Name</label>  
 </div>  
 <input type="text" class="form-control" id="name" name="name" value="" th:field="\*{name}"/>  
 <small th:if="${#fields.hasErrors('name')}" th:errors="\*{name}" class="bg-danger text-light rounded">Name error message</small>  
 </div>  
  
 <div class="form-group">  
 <div class="label-holder text-white textCol d-flex justify-content-center">  
 <label for="description" class="h4 mb-2">Description</label>  
 </div>  
 <input type="text" class="form-control" id="description" name="description" value="" th:field="\*{description}"/>  
 <small th:if="${#fields.hasErrors('description')}" th:errors="\*{description}" class="bg-danger text-light rounded">Description error message</small>  
 </div>  
  
 <div class="form-group">  
 <div class="label-holder text-white textCol d-flex justify-content-center">  
 <label for="imageUrl" class="h4 mb-2">Image URL</label>  
 </div>  
 <textarea type="text" class="form-control" id="imageUrl" name="imageUrl" th:field="\*{imageUrl}"></textarea>  
 <small th:if="${#fields.hasErrors('imageUrl')}" th:errors="\*{imageUrl}" class="bg-danger text-light rounded">Image URL error message</small>  
 </div>  
  
 <div class="form-group">  
 <div class="text-white label-holder d-flex justify-content-center">  
 <label for="paper" class="h4 mb-2">Paper</label>  
 </div>  
 <select class="browser-default custom-select" id="paper" name="paper" th:field="\*{paper}">  
 <option th:each="paper : ${T(Philately.stamp.model.Paper).values()}" th:value="${paper}" th:text="${paper}"></option>  
  
 </select>  
 <small th:if="${#fields.hasErrors('paper')}" th:errors="\*{paper}" class="bg-danger text-light rounded">Paper error message</small>  
  
 </div>  
  
 <div class="button-holder d-flex justify-content-center">  
 <button type="submit" class="btn btn-info mb-3">Add stamp</button>  
 </div>  
  
 </form>

Next ->

@Controller  
@RequestMapping("/stamps")  
public class StampController {  
  
  
 private final UserService userService;  
 private final StampService stampService;  
  
 @Autowired  
 public StampController(UserService userService, StampService stampService) {  
 this.userService = userService;  
 this.stampService = stampService;  
 }  
  
  
 @GetMapping("/new")  
 public ModelAndView getNewStampPage(HttpSession session){  
  
 UUID userId = (UUID) session.getAttribute("user\_id");  
 User user = userService.getById(userId);  
  
 ModelAndView modelAndView = new ModelAndView();  
 modelAndView.setViewName("add-stamp");  
  
 modelAndView.addObject("user", user);  
 *//добавяме и потребителя, защото това определя какво ще виждаме в хедъра като бутони, когато сме в стамп страницата* modelAndView.addObject("createNewStamp", new CreateNewStamp());  
  
 return modelAndView;  
 }  
  
 @PostMapping()  
  
 public String createNewStamp(@Valid CreateNewStamp createNewStamp, BindingResult bindingResult, HttpSession session) {  
  
 if (bindingResult.hasErrors()) {  
 return "add-stamp";  
 }  
  
 UUID userId = (UUID) session.getAttribute("user\_id");  
 User user = userService.getById(userId);  
  
 stampService.create(createNewStamp, user);  
  
  
 return "redirect:/home";  
 }  
}

Next ->

@Service  
public class StampService {  
  
 private final StampRepository stampRepository;  
 private final WishedStampRepository wishedStampRepository;  
  
 @Autowired  
 public StampService(StampRepository stampRepository, WishedStampRepository wishedStampRepository) {  
 this.stampRepository = stampRepository;  
 this.wishedStampRepository = wishedStampRepository;  
 }  
  
 public void create(CreateNewStamp createNewStamp, User user) {  
  
 Stamp stamp = Stamp.*builder*()  
 .name(createNewStamp.getName())  
 .description(createNewStamp.getDescription())  
 .imageUrl(createNewStamp.getImageUrl())  
 .paper(createNewStamp.getPaper())  
 .owner(user)  
 .build();  
  
 stampRepository.save(stamp);  
  
 }  
}

Next - > тук първо изтриваме към src името някакво и слагаме това <img th:src="${stamp.imageUrl}" alt="The Queen" width="70px" height="70px">

<p class="mySticky bg-gray text-dark user" th:text="${'Welcome' + user.username}"></p>

<div>  
 <div class="home-fields" style="height: 80vh; overflow: auto">  
 <h3 class="my-stamps">My Stamps</h3>  
 <table class="my-table table-striped table-dark">  
 <tr class="my-stamps" th:each="stamp : ${user.stamps}">  
 <td>  
 <img th:src="${stamp.imageUrl}" alt="The Queen" width="70px" height="70px">  
 </td>  
 <td class="my-cell-container">  
 <div>Name: <span th:text="${stamp.name}"></span></div>  
 <div>Description: <span th:text="${stamp.description}"></span></div>  
 <div>Used paper: <span th:text="${stamp.paper}"></span></div>  
 </td>  
 </tr>  
 </table>  
 </div>  
</div>

And RUN welcome user

A screenshot of a phone

AI-generated content may be incorrect.

### Home Page (with stamps, when a logged-in user)

* Note: The left top section (My Stamps) of the page should display **all the current logged user stamps** from the database which he added, if there are any.
* Note: The right top section (Offered Stamps) of the page should display **all offered stamps added by the rest of the users** (except the current logged one).
* Note: The bottom right section (My Wishlist) of the page should display **wished stamps by the current logged-in user**.

Next ->

@Controller  
public class IndexController {  
  
 private final UserService userService;  
 private final StampService stampService;

@GetMapping("/home")  
public ModelAndView getHomePage(HttpSession session){  
  
 UUID userId = (UUID) session.getAttribute("user\_id");  
 User user = userService.getById(userId);  
  
 List<Stamp> allStamps = stampService.getAll();

Next -> stampService

public List<Stamp> getAll() {  
  
 return stampRepository.findAll();  
}

Next ->

@GetMapping("/home")  
public ModelAndView getHomePage(HttpSession session){  
  
 UUID userId = (UUID) session.getAttribute("user\_id");  
 User user = userService.getById(userId);  
  
 List<Stamp> allStamps = stampService.getAll();  
  
 ModelAndView modelAndView = new ModelAndView();  
 modelAndView.setViewName("home");  
  
 modelAndView.addObject("user", user);  
 modelAndView.addObject("allStamps", allStamps);  
  
 return modelAndView;  
  
}

Next -> home – правим да виждаме всички други марки на другите

<div class="col-6 mt-1 mb-1">  
 <div class="home-fields" style="height: 40vh; overflow: auto">  
 <h3 class="my-stamps">Offered Stamps</h3>  
 <ul class="list-group list-group-vertical text-dark">  
 <li th:each="stamp : ${allStamps}" th:if="${stamp.owner.id != user.id}"

>

<div class="img1">  
 <img id="image" width="85px"  
 th:src="${stamp.imageUrl}"  
 alt="US Mail"  
 height="85" style="padding-top: 0.3em; padding-left: 0.3em; margin-top: 2.3em">  
 </div>  
 <div class="stamp-info">  
 <p>Name: <span th:text="${stamp.name}"></span></p>  
 <p>Description: <span th:text="${stamp.description}"></span></p>  
 </div>  
</div>  
<div class="buttons-info">  
 <div class="favorite">  
 <form action="" method="">  
 <button class="btn-outline-info btn">To Wishlist</button>  
 </form>  
 </div>  
</div>  
<div class="second-info">  
 <p th:text="${'Added by: ' + stamp.owner.username}"></p>  
</div>

RUN

Next -> сега ще прибавяме в нашия любим списък от общите марки

<div class="buttons-info">  
 <div class="favorite">  
 <form th:action="@{'/stamps/' + ${stamp.id} + '/wishlist'}" th:method="POST">  
 <button class="btn-outline-info btn">To Wishlist</button>  
 </form>  
 </div>

Next ->

@PostMapping("/{id}/wishlist")  
 public String createNewWishListItemForStamp(@PathVariable UUID id, HttpSession session){  
  
 UUID userId = (UUID) session.getAttribute("user\_id");  
 User user = userService.getById(userId);  
  
 stampService.createWished(id, user);  
  
 return "redirect:/home";  
  
}

Next-> stamp service

public void createWished(UUID stampId, User user) {  
  
 Stamp stamp = getById(stampId);  
  
 WishedStamp wishedStamp = WishedStamp.*builder*()  
 .imageUrl(stamp.getImageUrl())  
 .name(stamp.getName())  
 .description(stamp.getDescription())  
 .owner(user)  
 .build();  
  
 wishedStampRepository.save(wishedStamp);  
  
}

public Stamp getById(UUID id) {

return stampRepository.findById(id).orElseThrow(() -> new RuntimeException("Stamp with id [%s] does not exist.".formatted(id)));

}

Next -> трябва да ги виждаме марките в любимия списък

<div class="home-fields" style="height: 39vh; overflow: auto">  
 <h3 class="my-stamps">My Wishlist</h3>  
 <div class="wish-list">  
 <table class="my-table my-wishlist">  
 <tr class="my-stamps" th:each="stamp : ${user.wishedStamps}">

Изтриваме единия пасаж, той е статичен – всичко:

<tr class="my-stamps">  
 <td>  
 <img src="https://postalmuseum.si.edu/sites/default/files/0\_217665\_1a.jpg" alt="US Mail" width="55px" height="55" style="padding-top: 0.3em; padding-left: 0.3em">  
 </td>  
 <td style="text-align: start; padding-left: 5px">US Mail</td>  
 <td style="width: 15%; padding-top: 8px; padding-right: 5px">  
 <form action="" method="">  
 <button class="btn-danger btn">Remove</button>  
 </form>  
 </td>  
</tr>

Next ->

<table class="my-table my-wishlist">  
 <tr class="my-stamps" th:each="stamp : ${user.wishedStamps}">  
 <td>  
 <img th:src="${stamp.imageUrl}" alt="US Mail" width="55px" height="55" style="padding-top: 0.3em; padding-left: 0.3em">  
 </td>  
 <td style="text-align: start; padding-left: 5px" th:text="${stamp.name}"></td>  
 <td style="width: 15%; padding-top: 8px; padding-right: 5px">  
 <form action="" method="">  
 <button class="btn-danger btn">Remove</button>  
 </form>  
 </td>  
 </tr>  
</table>

Next ->

<td style="width: 15%; padding-top: 8px; padding-right: 5px">  
 <form th:action="@{'/stamps/' + ${stamp.id}}" th:method="DELETE">  
 <button class="btn-danger btn">Remove</button>  
 </form>  
</td>

Next -> stampcontroller

@DeleteMapping("/{id}")  
public String deleteWishedStamp(@PathVariable UUID id){  
  
 stampService.deleteWishedStampById(id);  
  
 return "redirect:/home";  
}

Next ->stampService

public void deleteWishedStampById(UUID id) {  
 wishedStampRepository.deleteById(id);  
}

A screenshot of a computer

AI-generated content may be incorrect.

The templates have been given to you in the application skeleton, so make sure you implement the pages correctly.

**NOTE**: The templates should look as shown above.

**NOTE**: The templates do **NOT** **require** **additional** **CSS** for you to write. Only **bootstrap** and the **given CSS** are enough.

## Functional Requirements

The Functionality Requirements describe the functionality that the application must support.

The application should provide **Guest** (not logged in) users with the functionality to **log in**, **register** and view the **Index** page.

The application should provide **User** (logged in) with the functionality to **log out, add** a new stamp **(Add stamp page), view** all stamps **(Home page)** and **add** a stamp from **"Offered Stamps"** to his **Wishlist**.

Philately Application in the navbar should **redirect** to the appropriate URL depending on whether the **user is logged in**.

The application should provide functionality for **adding stamps** with the following paper types - WOVE\_PAPER**,** LAID\_PAPER**, or** GRANITE\_PAPER.

The stamps should be separated into four different sections according to their ownership.

**At the top**, above all sections in a banner, the **username** of the logged-in user should be displayed:   
**"Welcome current logged-in User"**.

### My Stamps section

This section displays added stamps by the **currently** logged-in **User**. When the **User** adds a **new** stamp, it should appear here.

### Offered Stamps section

This section shows stamps added and offered by **all other users** who use the app.

When the user clicks on the "**To Wishlist" button of some stamp** in the **"Offered Stamps"** section, he adds the stamp to **his "My Wishlist"** section. You should **not delete** this stamp from **DB** and you should **not** remove it from the **"Offered Stamps"** section, yet. The stamp should be added to the **"My Wishlist"** collection with wished stamps of the currently logged-in **User**. The added stamp should appear in the **"My Wishlist"** section of the **User.**

### My Wishlist section

This section displays the Wishlist of stamps that the **currently** logged-in **user** has marked as desired. Users should have the ability to manage their Wishlist by removing stamps individually using the "**Remove**" button. Upon removal, the selected stamp should be instantly deleted from the "**Wishlist"**, ensuring that the list dynamically updates to reflect the changes.

The application should store its data in a **MySQL** database.

## Security Requirements

The Security Requirements are mainly access requirements. Configurations about which users can access specific functionalities and pages.

You will be provided with a pre-configured **session interceptor** class designed to manage access to both authorized and unauthorized pages/endpoints. Additionally, an implementation of the **WebMvcConfigurer** interface is included, ensuring access to static resource folders within your application.

Your main task here would be to create a session for the user upon login and on every subsequent request to fetch the **"user\_id"** attribute from the user's session.

When uploading a solution it is necessary to make an archived file of **src** + **pom.xml**. Go to **SoftUni Learning System** select the archived file from your device and press the "**Upload Homework**" button.

**Only the LAST uploaded file is saved.**

## Scoring

### Database – 10 points.

### Pages – 25 points.

### Functionality – 40 points.

### Security – 5 points.

### Validations – 15 points.

### Code Quality – 5 points.