Project Validation Summary

Overview

This document outlines all validation implemented across the Library Management System project. Validation is primarily implemented at the **business logic layer** (services and controllers) and **HTML form level** (client-side), rather than using JPA validation annotations.

---

1. User Registration Validation

Location: AuthController.java

Business Logic Validation

```
@PostMapping("/register")
public String registerUser(@ModelAttribute User user, RedirectAttributes
redirectAttributes) {
    // Username uniqueness check
    if (userRepository.existsByUsername(user.getUsername())) {
        redirectAttributes.addFlashAttribute("error", "Username already
exists");
        return "redirect:/register";
    }

    // Email uniqueness check
    if (userRepository.existsByEmail(user.getEmail())) {
        redirectAttributes.addFlashAttribute("error", "Email already
exists");
        return "redirect:/register";
    }

    // Admin email restriction
    String email = user.getEmail() == null ? "" :
user.getEmail().toLowerCase().trim();
    user.setEmail(email);

    if (email.contains("@admin.library.com")) {
        redirectAttributes.addFlashAttribute("error", "This email is not
allowed for registration");
        return "redirect:/register";
    }

    // Password encoding
    user.setPassword(passwordEncoder.encode(user.getPassword()));

    // Assign default role
    Role memberRole = roleRepository.findByName("MEMBER")
        .orElseThrow(() -> new RuntimeException("Member role not found"));
```

```
    user.getRoles().add(memberRole);

    userRepository.save(user);
    redirectAttributes.addFlashAttribute("success", "Registration
successful! Please login.");
    return "redirect:/login";
}
```

HTML Form Validation

File: register.html

Client-side Validations:

- Username: `required`
- Full Name: `required`
- Email Type: Radio button selection required (Personal or Library)
- Personal Email: `type="email"` validation when selected
- Library Email: Username input with auto-formatting to `@member.library.com`
- Password: `required`, `minlength="6"`

Form Handling:

```
<!-- Email Type Selection -->
<input type="radio" name="emailType" id="personalEmail" value="personal"
required>
<input type="radio" name="emailType" id="libraryEmail" value="library"
required>

<!-- Dynamic Email Input -->
<input type="email" class="form-control" id="email" th:field="*{email}"
placeholder="example@gmail.com">

<!-- Password Validation -->
<input type="password" class="form-control" id="password"
       th:field="*{password}" required minlength="6">
<div class="form-text">Password must be at least 6 characters.</div>

<!-- Form Submission with prepareEmail() validation -->
<form th:action="@{/register}" th:object="${user}" method="post"
onsubmit="return prepareEmail()">
```

JavaScript Validation Function:

```
function toggleEmailInput() {
    // Shows/hides email input based on selection
    // Sets required attributes dynamically
}
```

```
function prepareEmail() {
    // Prepares email based on email type before submission
    // For library email: username + '@member.library.com'
    // Returns true to allow form submission
}
```

---

2. Login Validation

Location: login.html

HTML Form Validation:

```
<!-- Username Input -->
<input type="text" class="form-control" id="username" name="username"
required autofocus>

<!-- Password Input -->
<input type="password" class="form-control" id="password" name="password"
required>
```

Server-side (Spring Security):

- Authentication through `CustomUserDetailsService`
- Password comparison with BCrypt encoded passwords
- Error message: "Invalid username or password"

---

3. Book Management Validation

Location: BookController.java

Create/Update Book Validation

Business Logic:

```
@PostMapping("/admin/books")
public String createBook(@ModelAttribute Book book, RedirectAttributes
redirectAttributes) {
    try {
        bookService.saveBook(book);
        redirectAttributes.addFlashAttribute("success", "Book added
successfully!");
    } catch (Exception e) {
        redirectAttributes.addFlashAttribute("error", "Error adding book: "
+ e.getMessage());
    }
    return "redirect:/admin/books";
```

```
}

@PostMapping("/admin/books/{id}")
public String updateBook(@PathVariable Long id, @ModelAttribute Book book,
                         RedirectAttributes redirectAttributes) {
    try {
        bookService.updateBook(id, book);
        redirectAttributes.addFlashAttribute("success", "Book updated
successfully!");
    } catch (Exception e) {
        redirectAttributes.addFlashAttribute("error", "Error updating book:
" + e.getMessage());
    }
    return "redirect:/admin/books";
}
```

HTML Form Validation

File: books/form.html

```
<!-- Title Input -->
<input type="text" class="form-control" id="title"
       th:field="*{title}" required>

<!-- Author Input -->
<input type="text" class="form-control" id="author"
       th:field="*{author}" required>

<!-- ISBN Input (Unique constraint in database) -->
<input type="text" class="form-control" id="isbn"
       th:field="*{isbn}" required>

<!-- Genre Selection -->
<select class="form-select" id="genre" th:field="*{genre}" required>
    <option value="">Select Genre</option>
    <option value="Fiction">Fiction</option>
    <option value="Non-Fiction">Non-Fiction</option>
    <option value="Science">Science</option>
    <option value="History">History</option>
    <option value="Biography">Biography</option>
    <option value="Mystery">Mystery</option>
    <option value="Romance">Romance</option>
    <option value="Technology">Technology</option>
    <option value="Self-Help">Self-Help</option>
    <option value="Other">Other</option>
</select>

<!-- Quantity Input -->
<input type="number" class="form-control" id="quantity"
       th:field="*{quantity}" min="1" required>
```

Entity Constraints

File: Book.java

```
@Column(nullable = false)
private String title;

@Column(nullable = false)
private String author;

@Column(unique = true, nullable = false)    // ISBN must be unique
private String isbn;

@Column(nullable = false)
private String genre;

@Column(nullable = false)
private Integer quantity;

@Column(nullable = false)
private Boolean isAvailable = true;
```

Search Validation

File: BookController.java

```
@GetMapping("/books/search")
public String searchBooks(@RequestParam(required = false) String keyword,
Model model) {
    if (keyword != null && !keyword.isEmpty()) {
        model.addAttribute("books", bookService.searchBooks(keyword));
    } else {
        model.addAttribute("books", bookService.getAllBooks());
    }
    model.addAttribute("keyword", keyword);
    return "books/search";
}
```

---

4. Book Borrowing Validation

Location: BorrowController.java

Borrow Book Validation

Business Logic:

```
@PostMapping("/borrow/book/{bookId}")
public String borrowBook(@PathVariable Long bookId,
                         Authentication authentication,
                         RedirectAttributes redirectAttributes) {
    try {
        User user = userRepository.findByUsername(authentication.getName())
            .orElseThrow(() -> new RuntimeException("User not found"));

        Book book = bookService.getBookById(bookId)
            .orElseThrow(() -> new RuntimeException("Book not found"));

        borrowService.borrowBook(user, book);
        redirectAttributes.addFlashAttribute("success", "Book borrowed
successfully! Return deadline is 14 days.");
    } catch (Exception e) {
        redirectAttributes.addFlashAttribute("error", "Error borrowing
book: " + e.getMessage());
    }

    return "redirect:/borrow/available";
}
```

Borrow Service Validation

File: BorrowService.java

```
@Transactional
public BorrowRecord borrowBook(User user, Book book) {
    // Availability check
    if (!book.getIsAvailable()) {
        throw new RuntimeException("Book is not available");
    }

    // Set book as unavailable
    book.setIsAvailable(false);
    bookService.saveBook(book);

    // Create borrow record with 14-day return deadline
    LocalDate borrowDate = LocalDate.now();
    LocalDate returnDeadline = borrowDate.plusDays(14);

    BorrowRecord borrowRecord = new BorrowRecord(user, book, borrowDate,
returnDeadline);
    return borrowRepository.save(borrowRecord);
}

@Transactional
public BorrowRecord returnBook(Long borrowRecordId) {
    BorrowRecord record = borrowRepository.findById(borrowRecordId)
        .orElseThrow(() -> new RuntimeException("Borrow record not
```

```
found"));

    // Set actual return date
    record.setActualReturnDate(LocalDate.now());

    // Validate return status based on deadline
    if (LocalDate.now().isAfter(record.getReturnDeadline())) {
        record.setStatus("RETURNED_LATE");
    } else {
        record.setStatus("RETURNED");
    }

    // Make book available again
    Book book = record.getBook();
    book.setIsAvailable(true);
    bookService.saveBook(book);

    return borrowRepository.save(record);
}

@Transactional
public void updateOverdueRecords() {
    List<BorrowRecord> activeRecords =
borrowRepository.findByStatus("BORROWED");
    LocalDate today = LocalDate.now();

    for (BorrowRecord record : activeRecords) {
        if (today.isAfter(record.getReturnDeadline())) {
            record.setStatus("OVERDUE");
            borrowRepository.save(record);
        }
    }
}
```

Entity Constraints

File: BorrowRecord.java

```
@ManyToOne(fetch = FetchType.EAGER)
@JoinColumn(name = "user_id", nullable = false)
private User user;

@ManyToOne(fetch = FetchType.EAGER)
@JoinColumn(name = "book_id", nullable = false)
private Book book;

@Column(name = "borrow_date", nullable = false)
private LocalDate borrowDate;

@Column(name = "return_deadline", nullable = false)
private LocalDate returnDeadline;
```

```
@Column(name = "actual_return_date")
private LocalDate actualReturnDate;

@Column(name = "status", length = 20, nullable = false)
private String status; // BORROWED, RETURNED, OVERDUE
```

---

5. Entity-Level Validation

User Entity

File: User.java

```
@Column(name = "username", length = 50, unique = true, nullable = false)
private String username;

@Column(name = "password", length = 255, nullable = false)
private String password;

@Column(name = "full_name", length = 100, nullable = false)
private String fullName;

@Column(name = "email", length = 100, unique = true, nullable = false)
private String email;
```

Constraints:

- Username: max 50 chars, unique, not null
- Password: max 255 chars, not null
- Full Name: max 100 chars, not null
- Email: max 100 chars, unique, not null

Book Entity

File: Book.java

Constraints:

- Title: not null
- Author: not null
- ISBN: unique, not null
- Genre: not null
- Quantity: not null, minimum 1 (via form input `min="1"`)
- IsAvailable: not null, defaults to true

Role Entity

File: Role.java

```
@Column(name = "name", length = 50, unique = true, nullable = false)
private String name;
```

Constraints:

- Name: max 50 chars, unique, not null

---

6. Security Validation

Location: SecurityConfig.java

```
@Bean
public SecurityFilterChain securityFilterChain(HttpSecurity http) throws
Exception {
    http
        .authorizeHttpRequests(auth -> auth
            .requestMatchers("/css/**", "/js/**", "/images/**").permitAll()
            .requestMatchers("/login", "/register").permitAll()
            .requestMatchers("/admin/**").hasRole("LIBRARIAN")         //
Admin access control
            .requestMatchers("/borrow/**").hasAnyRole("MEMBER",
"LIBRARIAN")  // Member access control
            .requestMatchers("/").authenticated()
            .anyRequest().authenticated()                               //
All other requests require authentication
        )
        .formLogin(form -> form
            .loginPage("/login")
            .defaultSuccessUrl("/", true)
            .permitAll()
        )
        .logout(logout -> logout
            .logoutUrl("/logout")
            .logoutSuccessUrl("/login?logout")
            .permitAll()
        )
        .userDetailsService(userDetailsService);

    return http.build();
}
```

Password Encoding:

```
@Bean
public PasswordEncoder passwordEncoder() {
    return new BCryptPasswordEncoder();
}
```

---

7. Error Handling & User Feedback

Controller-Level Error Handling

Pattern Used Across All Controllers:

```
try {
    // Operation
    redirectAttributes.addFlashAttribute("success", "Success message");
} catch (Exception e) {
    redirectAttributes.addFlashAttribute("error", "Error: " +
e.getMessage());
}
return "redirect:/path";
```

HTML Error/Success Messages

Login Page:

```
<div th:if="${param.error}" class="alert alert-danger" role="alert">
    <i class="bi bi-exclamation-triangle"></i> Invalid username or
password.
</div>

<div th:if="${param.logout}" class="alert alert-success" role="alert">
    <i class="bi bi-check-circle"></i> You have been logged out
successfully.
</div>

<div th:if="${success}" class="alert alert-success" role="alert">
    <i class="bi bi-check-circle"></i> <span th:text="${success}"></span>
</div>
```

Registration Page:

```
<div th:if="${error}" class="alert alert-danger" role="alert">
    <i class="bi bi-exclamation-triangle"></i> <span th:text="${error}">
</span>
</div>
```

Borrow Page:

```
<div th:if="${error}" class="alert alert-danger alert-dismissible fade
show" role="alert">
    <i class="bi bi-exclamation-triangle"></i> <span th:text="${error}">
</span>
    <button type="button" class="btn-close" data-bs-dismiss="alert">
</button>
</div>

<div class="alert alert-info" role="alert">
    <i class="bi bi-info-circle"></i> All borrowed books must be returned
within <strong>14 days</strong>.
</div>
```

---

8. Summary of Validation Types

Validation Type	Location	Examples
Database Constraints	Entity annotations	`@Column(nullable = false, unique = true)`
Business Logic	Services & Controllers	Book availability, username/email uniqueness
HTML Form Validation	Form inputs	`required`, `type="email"`, `minlength="6"`
JavaScript Validation	register.html	Email type toggle, email preparation
Spring Security	SecurityConfig	Role-based access control, authentication
Error Handling	Controllers	Try-catch blocks with user-friendly messages

---

9. Validation Workflow Example: Book Borrowing

1. **User clicks "Borrow Book"** → Browser sends request to `/borrow/book/{bookId}`
2. **Authentication check** → Spring Security verifies user is logged in
3. **User lookup** → Controller retrieves authenticated user from repository
4. **Book lookup** → Controller retrieves book from repository
5. **Service validation** → BorrowService checks if book is available
    - If not available: throws exception → caught → user sees error message
    - If available: proceed to step 6
6. **Business logic** → Set book availability to false, create borrow record with 14-day deadline
7. **Database save** → Save BorrowRecord entity (constraints enforce non-null fields)
8. **Redirect with feedback** → Success message shown to user

---

10. Recommended Enhancements

While the current validation is functional, consider adding:

1. **Bean Validation Annotations** (Jakarta Validation):

```
@NotBlank(message = "Username is required")
@Size(min = 3, max = 50)
private String username;

@Email(message = "Invalid email format")
private String email;
```

2. **Custom Validators** for complex business rules

3. **Validation Result Binding** in controllers:

```
public String registerUser(@Valid @ModelAttribute User user,
                           BindingResult bindingResult) {
    if (bindingResult.hasErrors()) {
        // Handle validation errors
    }
}
```

4. **API-Level Validation** for REST endpoints (if added in future)

Document Generated: 2026-01-14