Institute of Technology of Cambodia

Topic : Library Tracking System

Lecture: Roeun Pacharoth

Our Team

ROM Tola
Project manager

SOKHOM Panha
Menber

PRAVE Vinuth
Member

SRENG Panha
Member

Table Content

Introduction

- **Project Overview**

  This project is a Library Tracking System built using Spring Boot to manage books, members, and borrowing activities.

  - Security and Authentication Lead – Handles login, roles.
  - Handles main entity CRUD
  - Handles secondary module CRUD.
  - Frontend/Thymeleaf: Templates, access control.
  - Database

Introduction

- **Project Objective**

  This project build especially for librarians and library members to keep track on borrow record. Librarians can manage member, books and borrow record. Meanwhile, members can only view their borrow recorder.

Project Planing

Requirement Analysis

- Identify Role
- Define core features

System Design

- Designing Database Schema
- Designing UI

Technologies & Tools

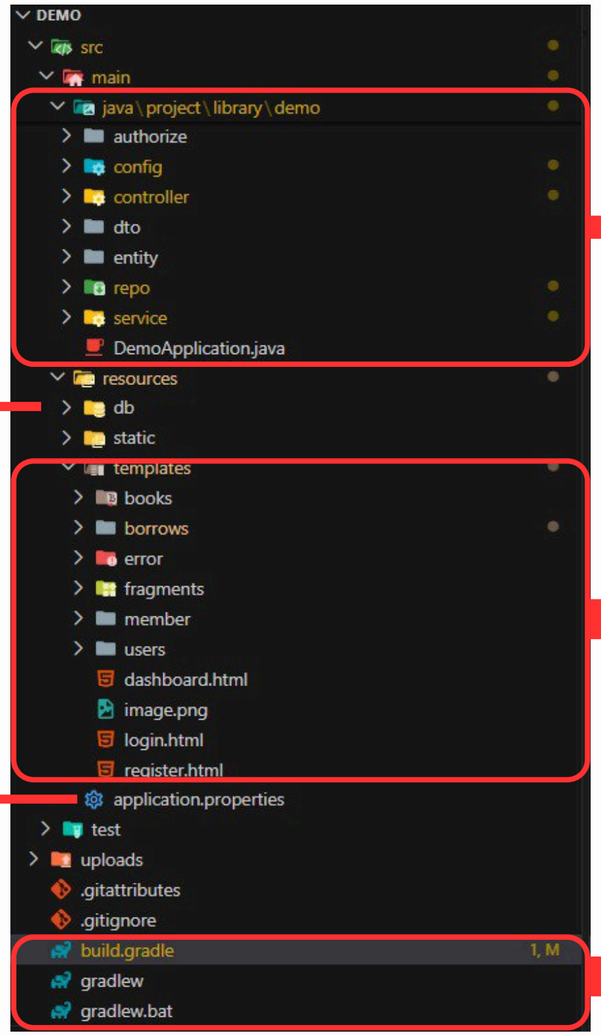
Technologies & Tools

Dependencies

```
dependencies {
    // Spring Boot Starters
    implementation 'org.springframework.boot:spring-boot-starter-web'
    implementation 'org.springframework.boot:spring-boot-starter-data-jpa:3.5.7'
    implementation 'org.springframework.boot:spring-boot-starter-thymeleaf:3.5.7'
    implementation 'org.springframework.boot:spring-boot-starter-validation:3.5.7'
    implementation 'org.springframework.boot:spring-boot-starter-security:3.5.7'
    implementation 'org.springframework.boot:spring-boot-starter-actuator:3.5.7'

    // Database
    implementation 'mysql:mysql-connector-java:8.0.33'

    // Flyway Migration
    implementation 'org.flywaydb:flyway-core:11.17.0'
    implementation 'org.flywaydb:flyway-mysql:11.17.0'

    // Validation
    implementation 'jakarta.validation:jakarta.validation-api'

    // Thymeleaf Security
    implementation 'org.thymeleaf.extras:thymeleaf-extras-springsecurity6:3.1.2.RELEASE'

    // JWT
    implementation 'io.jsonwebtoken:jjwt:0.13.0'
    runtimeOnly 'io.jsonwebtoken:jjwt-impl:0.13.0'
    runtimeOnly 'io.jsonwebtoken:jjwt-jackson:0.13.0'

    // Testing
    testImplementation 'org.springframework.boot:spring-boot-starter-test'
    testRuntimeOnly 'org.junit.platform:junit-platform-launcher'
}
```

Architecture

Folder Struture



Java package

Databases file

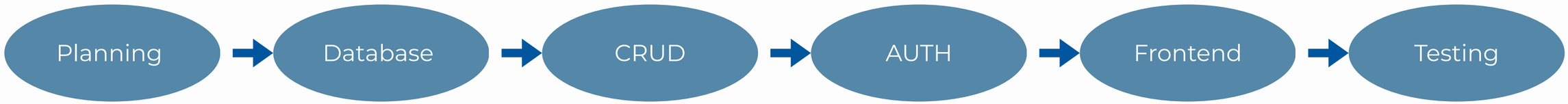
Front-end (UI)

Spring Boot configuration

Build Files

Architecture

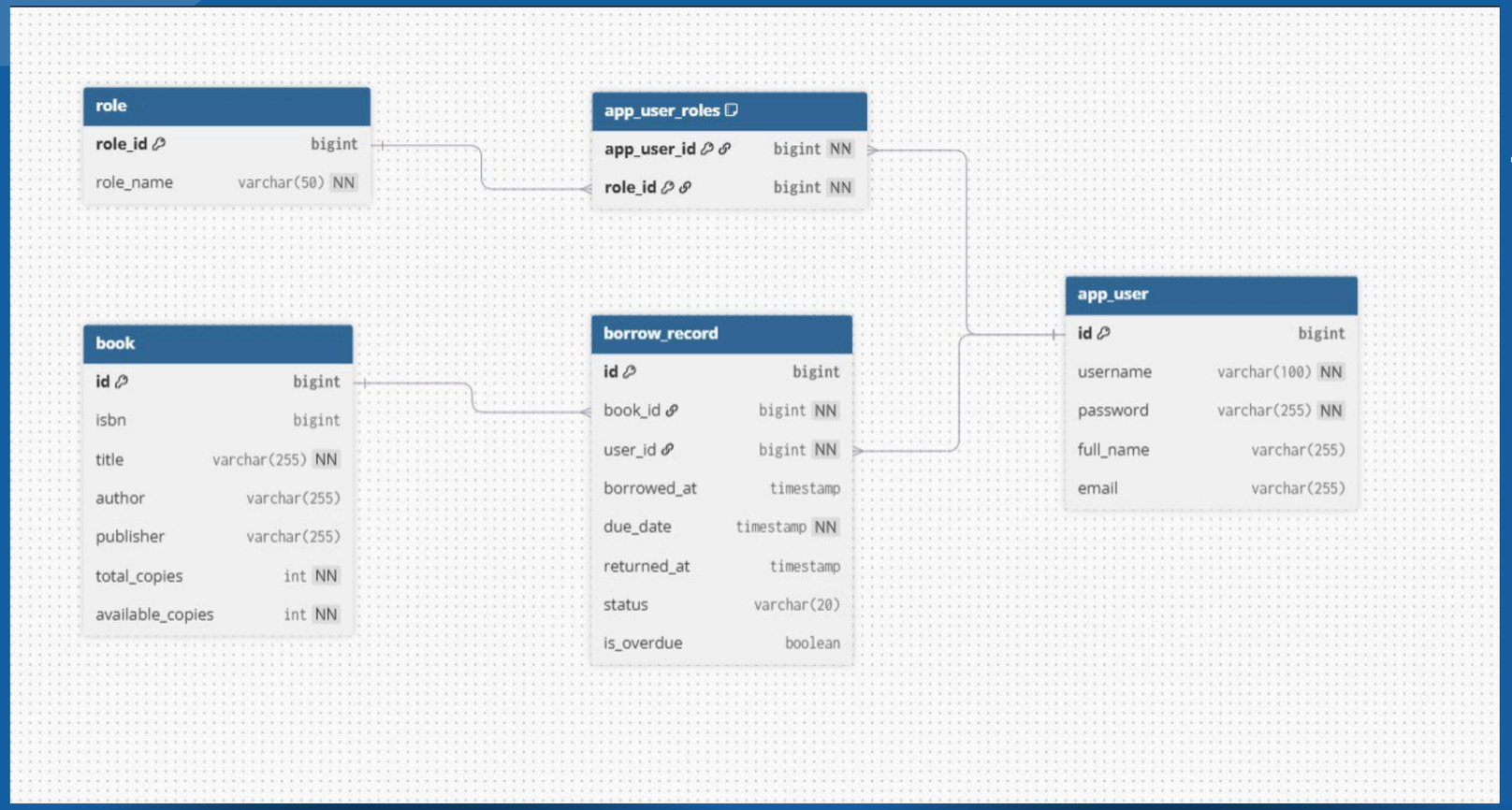
Design Workflow



Task Division

TASK	OWNER
Pull Request, merge	ROM Tola
Database / Entity	SOKHOM Panha
Frontend /UI	PRAVE Vinuth
CRUD	SRENG Panha
Spring Security / JWT	PRAVE Vinuth / SOKHOM Panha
Code Review / fix bug	ROM Tola

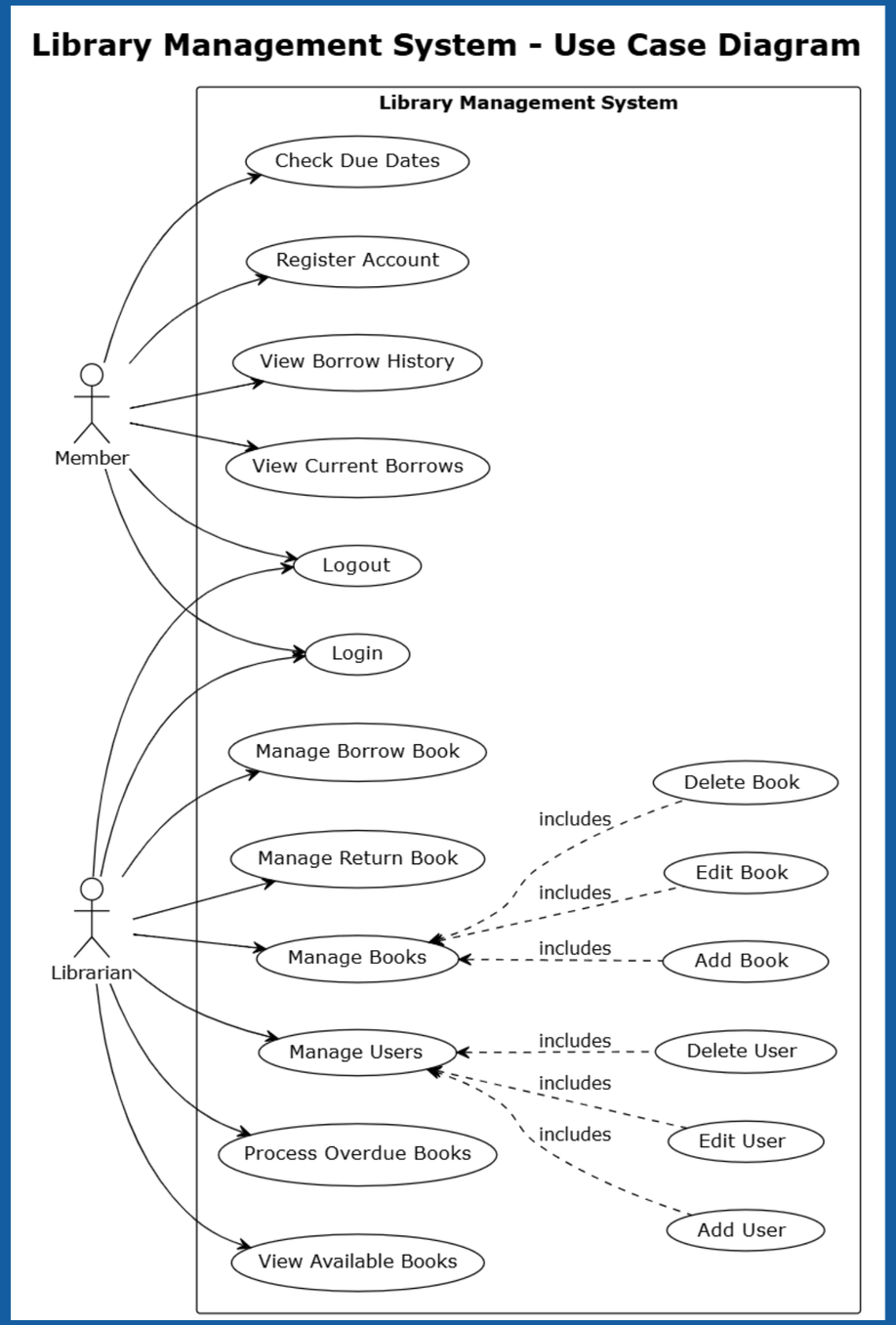
Database Design



+ Relationships
- Users ↔ Roles (Many-to-One)
- Users ↔ Borrows (Many-to-Many)
- Books ↔ Borrows (One-to-Many)

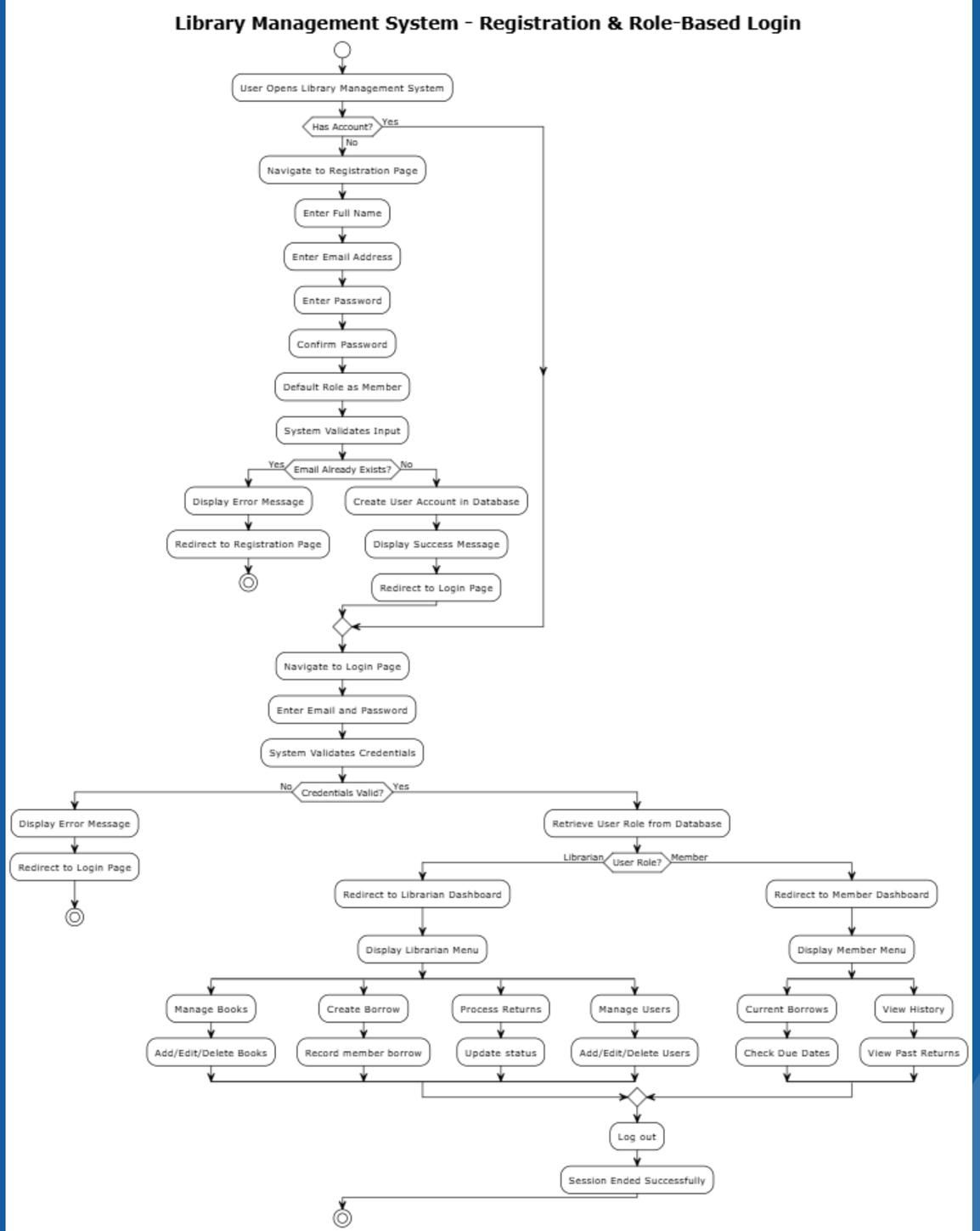
UML Design

Use case diagram



UML Design

LOGIN-REGISTER ACTIVITY DIAGRAM



API Endpoint Design

I. Authentication

Authentication	
GET	/login
POST	/api/login
GET	/register
POST	/api/register

API Endpoint Design

II. Librarian

Librarian	
GET	/dashboard

API Endpoint Design

II. Librarian

book management	
GET	/admin/book
GET	/admin/book/new
POST	/admin/book/new
GET	/admin/book/edit/{id}
POST	/admin/book/edit/{id}
GET	/admin/book/edit/{id}

API Endpoint Design

II. Librarian

borrow management	
GET	/admin/borrows
GET	/admin/borrows/new
POST	/admin/borrows/borrow
POST	/admin/borrows/delete/{id}

POST	/admin/borrows/return/{id}

API Endpoint Design

II. Librarian

Users management	
GET	/admin/users
GET	/admin/users/new
POST	/admin/users/save
GET	/admin/users/edit/{id}
GET	/admin/users/delete/{id}

API Endpoint Design

III. Member

Member	
GET	/member/home
GET	/member/borrows/current
GET	/member/borrows/history

Implementation

```
.authorizeHttpRequests(auth -> auth
            // Public endpoints
            .requestMatchers(
                "/login", "/login**", "/doLogin",
                "/register", "/api/login", "/api/register",
                "/error", "/favicon.ico"
            ).permitAll()

            // Static resources
            .requestMatchers(
                "/css/**", "/js/**", "/images/**", "/fonts/**",
                "/static/**", "/resources/**", "/webjars/**",
"/assets/**").permitAll()

            // Librarian-only endpoints
            .requestMatchers("/admin/**").hasRole("LIBRARIAN")

            // Member-accessible endpoints
            .requestMatchers(
                "/member/**",
                "/books",
                "/borrow/**", "/return/**", "/myloans", "/profile"
            ).hasRole("MEMBER")

            // Any other request requires authentication
            .anyRequest().authenticated()
        )
```

ROLE LIBRARIAN can access to page admin all

ROLE MEMBER can access to page member all, book,borrow,return,myloans and profile

Implementation

Uses BCrypt to hash passwords with strength 12

```java
@Bean
public PasswordEncoder passwordEncoder() {
    return new BCryptPasswordEncoder(12);
}
```

Generate Token using username

```java
package project.library.demo.config;
import java.util.Date;
import java.util.Base64;
import javax.crypto.SecretKey;
import org.springframework.stereotype.Component;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.security.Keys;
import io.jsonwebtoken.Claims;

@Component
public class JwtUtil {
    private JwtProperties properties;
    private SecretKey key;

    public JwtUtil(JwtProperties properties){
        this.properties=properties;
        this.key= Keys.hmacShaKeyFor(Base64.getDecoder().decode(properties.getSecret()));
    }
    public String generateToken(String username){
        Date now = new Date();
        Date expiry = new Date(now.getTime() + properties.getExp());
        return Jwts.builder().subject(username).issuedAt(new
Date()).expiration(expiry).signWith(key).compact();
    }
    public String extractUsername(String token){
        return extractAllClaims(token).getSubject();
    }

    private Claims extractAllClaims(String token){
        return Jwts.parser()
         .verifyWith(key)
            .build()
            .parseSignedClaims(token)
            .getPayload();
    }
}
```

Implementation

This page to control the ROLE:

- LIBRARIAN
- MEMBER

```java
package project.library.demo.controller;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.web.authentication.AuthenticationSuccessHandler;
import org.springframework.stereotype.Component;

import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;

import java.io.IOException;
import java.util.Collection;

@Component
public class CustomLoginSuccessHandler implements AuthenticationSuccessHandler {

    @Override
    public void onAuthenticationSuccess(HttpServletRequest request,
                                        HttpServletResponse response,
                                        Authentication authentication)
            throws IOException, ServletException {

        Collection<? extends GrantedAuthority> authorities =
authentication.getAuthorities();
        boolean isAdmin = authorities.stream()
                .anyMatch(a -> a.getAuthority().equals("ROLE_LIBRARIAN"));
        boolean isMember = authorities.stream()
                .anyMatch(a -> a.getAuthority().equals("ROLE_MEMBER"));

        if (isAdmin) {
            response.sendRedirect("/dashboard");
        } else if (isMember) {
            response.sendRedirect("/member/home");
        } else {
            response.sendRedirect("/login?error=true");
        }
    }
}
```

It role librarian we go page dashboard

It role members we go page /member/home

It not two page is error

Implementation

```
@GetMapping
    public String list(Model model) {
        model.addAttribute("books", bookService.findAll());
        return "books/list";
    }
```

```
    // ADD FORM - /books/new
    @GetMapping("/new")
    public String createForm(Model model) {
        model.addAttribute("book", new Book());
        return "books/form";
    }
```

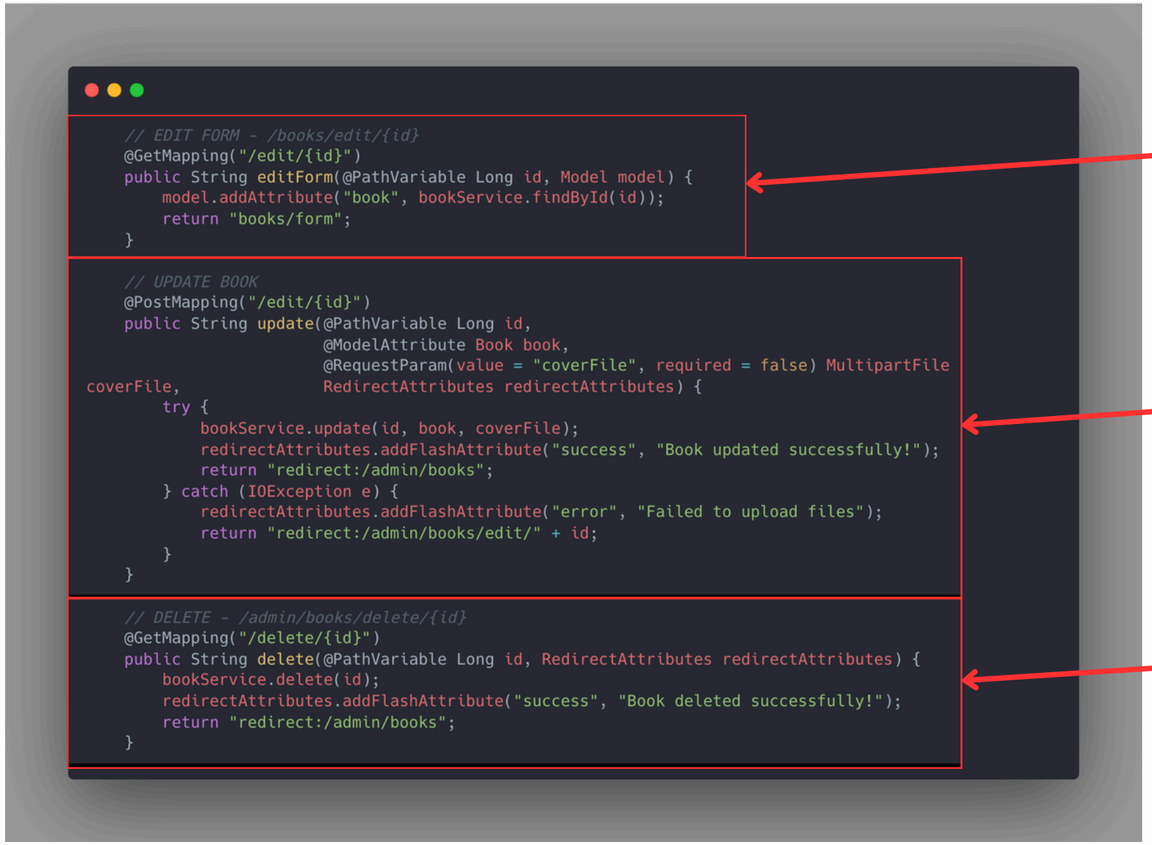
```
    // SAVE NEW BOOK
    @PostMapping("/new")
    public String create(@ModelAttribute Book book,
                         @RequestParam(value = "coverFile", required = false) MultipartFile
coverFile,               RedirectAttributes redirectAttributes) {
        try {
            bookService.create(book, coverFile);
            redirectAttributes.addFlashAttribute("success", "Book added successfully!");
            return "redirect:/admin/books";
        } catch (IOException e) {
            redirectAttributes.addFlashAttribute("error", "Failed to upload files");
            return "redirect:/admin/books/new";
        }
    }
  return go(f, seed, [])
}
```

This function displays the book list

The function is the form add book

This component is save when we input to form

Implementation



```
    // EDIT FORM - /books/edit/{id}
    @GetMapping("/edit/{id}")
    public String editForm(@PathVariable Long id, Model model) {
        model.addAttribute("book", bookService.findById(id));
        return "books/form";
    }
```

```
    // UPDATE BOOK
    @PostMapping("/edit/{id}")
    public String update(@PathVariable Long id,
                         @ModelAttribute Book book,
                         @RequestParam(value = "coverFile", required = false) MultipartFile
coverFile,               RedirectAttributes redirectAttributes) {
        try {
            bookService.update(id, book, coverFile);
            redirectAttributes.addFlashAttribute("success", "Book updated successfully!");
            return "redirect:/admin/books";
        } catch (IOException e) {
            redirectAttributes.addFlashAttribute("error", "Failed to upload files");
            return "redirect:/admin/books/edit/" + id;
        }
    }
```

```
    // DELETE - /admin/books/delete/{id}
    @GetMapping("/delete/{id}")
    public String delete(@PathVariable Long id, RedirectAttributes redirectAttributes) {
        bookService.delete(id);
        redirectAttributes.addFlashAttribute("success", "Book deleted successfully!");
        return "redirect:/admin/books";
    }
```

This form edit the component book

After input the form this function have save data input

The function have delete the book

Implementation

```java
    // Show "Add Borrow" form
    @GetMapping("/new")
    public String showBorrowForm(Model model) {
        model.addAttribute("borrowRecord", new BorrowRecord());
        model.addAttribute("books", bookService.findAll());
        model.addAttribute("users", userService.getAllMembers());
        return "borrows/form";
    }

    // Handle "Add Borrow" submission
    @PostMapping("/borrow")
    public String addBorrow(@ModelAttribute BorrowRecord borrowRecord,
                            RedirectAttributes redirectAttributes) {
        try {

            // Set borrow date and due date
            Timestamp now = Timestamp.valueOf(LocalDateTime.now());
            borrowRecord.setBorrowAt(now);
            borrowRecord.setDueDate(Timestamp.valueOf(now.toLocalDateTime().plusDays(14)));
            borrowRecord.setStatus("BORROWED");
            borrowRecord.setOverdue(false);

            // Save borrow record using IDs
            borrowService.borrowBookByIds(borrowRecord.getUserId(), borrowRecord.getBookId());

            redirectAttributes.addFlashAttribute("success", "Book borrowed successfully!");
        } catch (Exception e) {
            redirectAttributes.addFlashAttribute("error", "Cannot borrow book: " +
e.getMessage());
        return "redirect:/admin/borrows";
    }

    // Handle "Return Book" action
    @PostMapping("/return/{id}")
    public String returnBook(@PathVariable Long id,
                             RedirectAttributes redirectAttributes) {
        try {
            borrowService.returnBook(id);
            redirectAttributes.addFlashAttribute("success", "Book returned successfully!");
        } catch (Exception e) {
            redirectAttributes.addFlashAttribute("error", "Cannot return book: " +
e.getMessage());
        return "redirect:/admin/borrows";
    }
```

This form handle user borrow book

This function handle day when member borrow the current day count to 14 day is Overdue day

The function when book is return

Conclusion

This project is built for librarians and library members to track borrowing records. Librarians can manage members, books, and borrow records, while members can only view their own records. The system helps make library management more organized and efficient.

Difficulties

- Security Implementation
- Hard to build with Thymeleaf
- Conflict code with teammate
- Database Design is problem

Conclusion

Future Improvements

- More security features
- Improves User Experience
- Add more feature for member and Librarians
- Code quality

THANK YOU!