Project 4: Library Borrowing && Tracking Systems

PHORN SOPHATH

👥 Secondary Entity - Members

Responsible for Member Management.

- [ ] **Member Navigation** (Display member name and password updating in the navigation bar)

How it works:

> - **Member Navigation:** The navigation bar will show the logged-in member's name and provide an option to update their password for enhanced security.

```
34      @GetMapping("/")
35      public String dashboardHome(Model model) {
36          // 1. nav & Context Info
37          model.addAttribute("userName", "Nisal Gunasekara");
38          model.addAttribute("userRole", "Admin");
39          model.addAttribute("activePage", "dashboard");
40
```

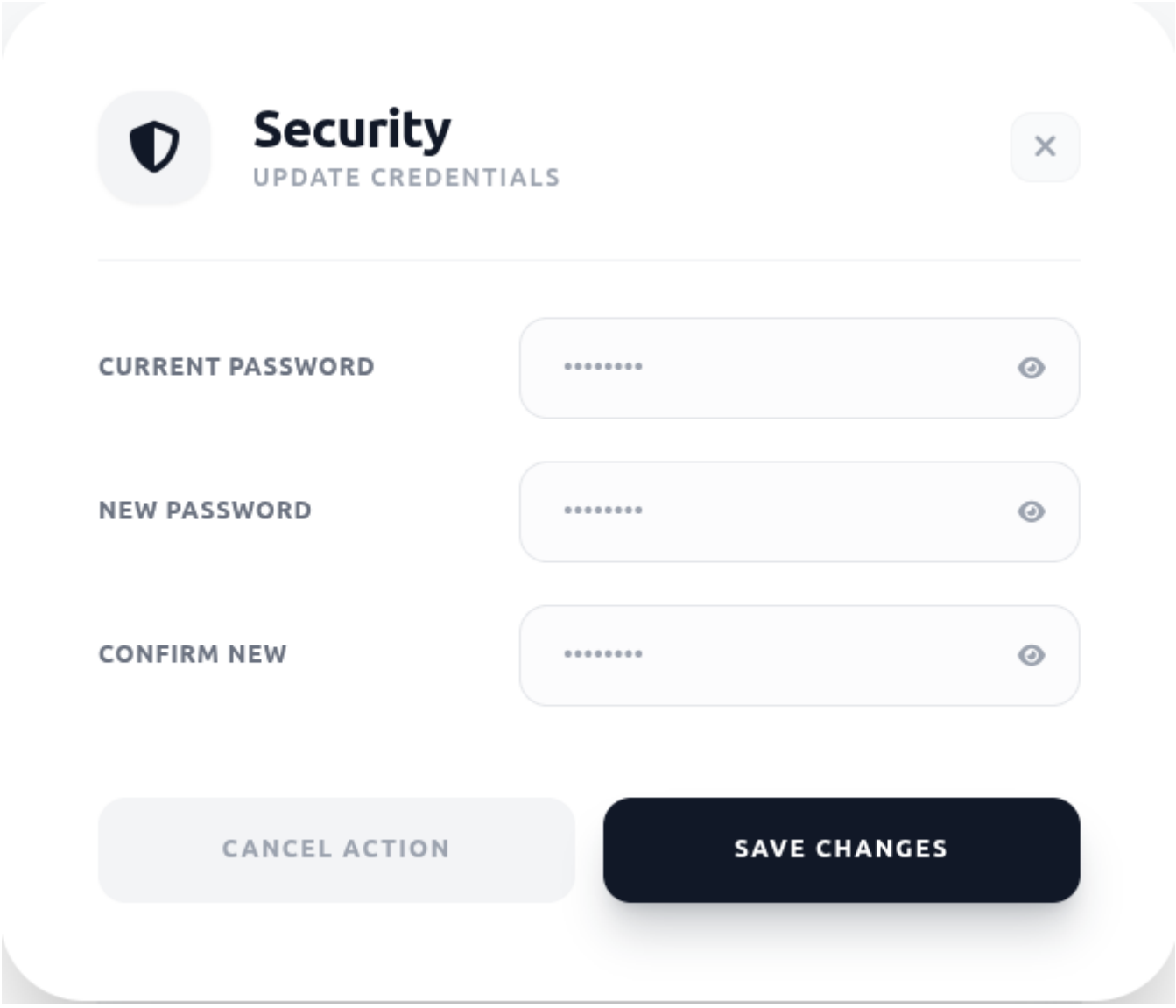
- The image above in the dashboard controller shows how to retrieve member information from the database to display in the navigation bar. But now it is just providing the static name, role.
- use `model.addAttribute("userName", "Nisal Gunasekara");` and `model.addAttribute("userRole", "Admin");` to pass the member name and role to the view.

```
    <nav class="fixed top-0 right-0 left-64 h-16 bg-white/80 backdrop-blur-md z-40 border-b border-gray-100">
        <div th:replace="~{fragments/navigation :: navigation}"></div>
    </nav>
```

- Then the data is assigned to the navigation fragment in the layout dashboard as shown in the image above to update the data I used `th:text=${variable}`.

```
<div class="flex flex-col" onclick="toggleModal(true)" style="cursor: pointer;">
    <span class="text-lg font-bold text-gray-900 leading-tight" th:text="${userName ?: 'Guest User'}">User Name</span>
    <span class="text-xs font-bold text-blue-600 uppercase tracking-wider" th:text="${userRole ?: 'Member'}">Role</span>
</div>
```

- In the navigation bar HTML, use `${userName}` and `${userRole}` to display the member's name, role dynamically.
- use `model.addAttribute("activePage", "dashboard");` to highlight the active page in the sidebar(make tab which user clicks unique).
- When librarians click on the settings icon in the navigation bar, they will be rendering the "Update Credentials" layout to the password update page.



How it works:

- Its html+css in navigation fragment

```
<form th:action="@{/admin/update-credentials}" method="POST" class="space-y-7">
    <div class="space-y-6">
        <div class="flex items-center justify-between gap-8">
            <label class="text-sm font-black text-gray-500 uppercase tracking-wider">Current Password</label>
            <div class="relative w-80">
                <input type="password" name="currentPassword" id="currentPass" required placeholder="••••••••"
                    class="w-full px-6 py-4 rounded-2xl border border-gray-200 focus:outline-none focus:ring-4 focus:ring-blue-500/10 focus:border-blue-500 transition-all bg-gray-50/50">
                <button type="button" onclick="togglePasswordVisibility('currentPass', this)" class="absolute right-5 top-1/2 -translate-y-1/2 text-gray-400 hover:text-gray-600">
                    <i class="fa-solid fa-eye text-sm"></i>
                </button>
            </div>
        </div>

        <div class="flex items-center justify-between gap-8">
            <label class="text-sm font-black text-gray-500 uppercase tracking-wider">New Password</label>
            <div class="relative w-80">
                <input type="password" name="newPassword" id="newPass" required placeholder="••••••••"
                    class="w-full px-6 py-4 rounded-2xl border border-gray-200 focus:outline-none focus:ring-4 focus:ring-blue-500/10 focus:border-blue-500 transition-all bg-gray-50/50">
                <button type="button" onclick="togglePasswordVisibility('newPass', this)" class="absolute right-5 top-1/2 -translate-y-1/2 text-gray-400 hover:text-gray-600">
                    <i class="fa-solid fa-eye text-sm"></i>
                </button>
            </div>
        </div>

        <div class="flex items-center justify-between gap-8">
            <label class="text-sm font-black text-gray-500 uppercase tracking-wider">Confirm New</label>
            <div class="relative w-80">
                <input type="password" name="confirmPassword" id="confirmPass" required placeholder="••••••••"
                    class="w-full px-6 py-4 rounded-2xl border border-gray-200 focus:outline-none focus:ring-4 focus:ring-blue-500/10 focus:border-blue-500 transition-all bg-gray-50/50">
                <button type="button" onclick="togglePasswordVisibility('confirmPass', this)" class="absolute right-5 top-1/2 -translate-y-1/2 text-gray-400 hover:text-gray-600">
                    <i class="fa-solid fa-eye text-sm"></i>
                </button>
            </div>
        </div>
    </div>

    <div class="flex gap-4 pt-6">
        <button type="button" onclick="toggleModal(false)"
            class="flex-1 py-5 bg-gray-100 text-gray-400 font-black rounded-2xl hover:bg-gray-200 hover:text-gray-600 transition-all uppercase tracking-widest text-sm">
            Cancel Action
        </button>
        <button type="submit"
            class="flex-1 py-5 bg-gray-900 text-white font-black rounded-2xl hover:bg-black transition-all shadow-xl shadow-gray-900/20 uppercase tracking-widest text-sm">
            Save Changes
        </button>
    </div>
</form>
```

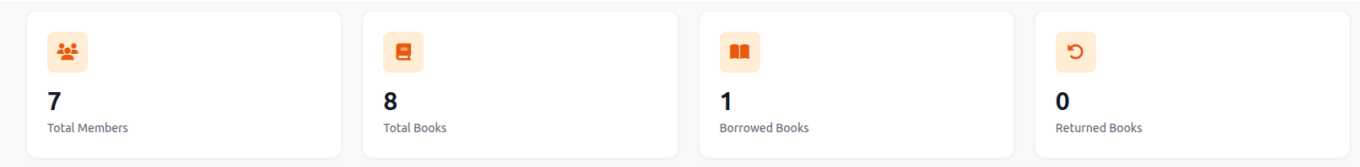
- Then it interacts with the route `/members/update-credentials` in the member controller to render the update password UI as shown in the image above by naming the input fields are the same as the parameters in dashboard controller and the controller method works with userService to check the current password is the same as old password or not and update the password accordingly then the userService will return boolean value to tell controller is successful or not.

```
47
48      public boolean updateAdminPassword(String name, String oldPwd, String newPwd) {
49      // findByName now returns Optional<User> to prevent NullPointerExceptions
50          return userRepository.findByName(name)
51              .map(admin -> {
52                  // Validate the current password matches before saving new one
53                  if (admin.getPassword().equals(oldPwd)) {
54                      admin.setPassword(newPwd);
55                      userRepository.save(admin);
56                      return true;
57                  }
58                  return false;
59              }).orElse(false);
60      }
```

- To display success or error messages.

```
function showToast(message, type = 'success') {
    const container = document.getElementById('toastContainer');
    const toast = document.createElement('div');

    // Set colors based on success or error
    const bgColor = type === 'success' ? 'bg-black' : 'bg-red-600';
    const icon = type === 'success' ? 'fa-check-circle' : 'fa-exclamation-circle';

    toast.className = `${bgColor} text-white px-8 py-5 rounded-2xl shadow-2xl flex items-center gap-4 animate-in slide-in-from-right
    toast.innerHTML = `
        <i class="fa-solid ${icon} text-xl"></i>
        <div class="flex flex-col">
            <span class="font-black text-xs uppercase tracking-widest">${type}</span>
            <span class="font-medium text-sm">${message}</span>
        </div>
    `;

    container.appendChild(toast);

    // Auto-remove after 4 seconds
    setTimeout(() => {
        toast.classList.add('animate-out', 'fade-out', 'slide-out-to-right');
        setTimeout(() => toast.remove(), 500);
    }, 4000);
}

// Add this inside your script block
window.addEventListener('load', () => {
    const params = new URLSearchParams(window.location.search);
    const status = params.get('status');

    if (status === 'success') {
        showToast("Credentials updated successfully!", "success");
    } else if (status === 'error') {
        showToast("Invalid current password.", "error");

        // 1. Open the modal
        toggleModal(true);

        // 2. Target the white card inside the modal for the shake
        const modalCard = document.querySelector('#credentialsModal > div');
        if (modalCard) {
            modalCard.classList.add('animate-shake');

            // 3. Remove the class after animation finishes so it can be re-triggered
            setTimeout(() => {
                modalCard.classList.remove('animate-shake');
            }, 500);
        }
```

- [ ] **Total Cards on Admin Dashboard** (Display total members, total books, total borrowed books, total overdue books)



How it works:

> - **Total Cards on Admin Dashboard:** The admin dashboard will feature cards that display key statistics such as total members, total books, total borrowed books, and total overdue books for quick insights.

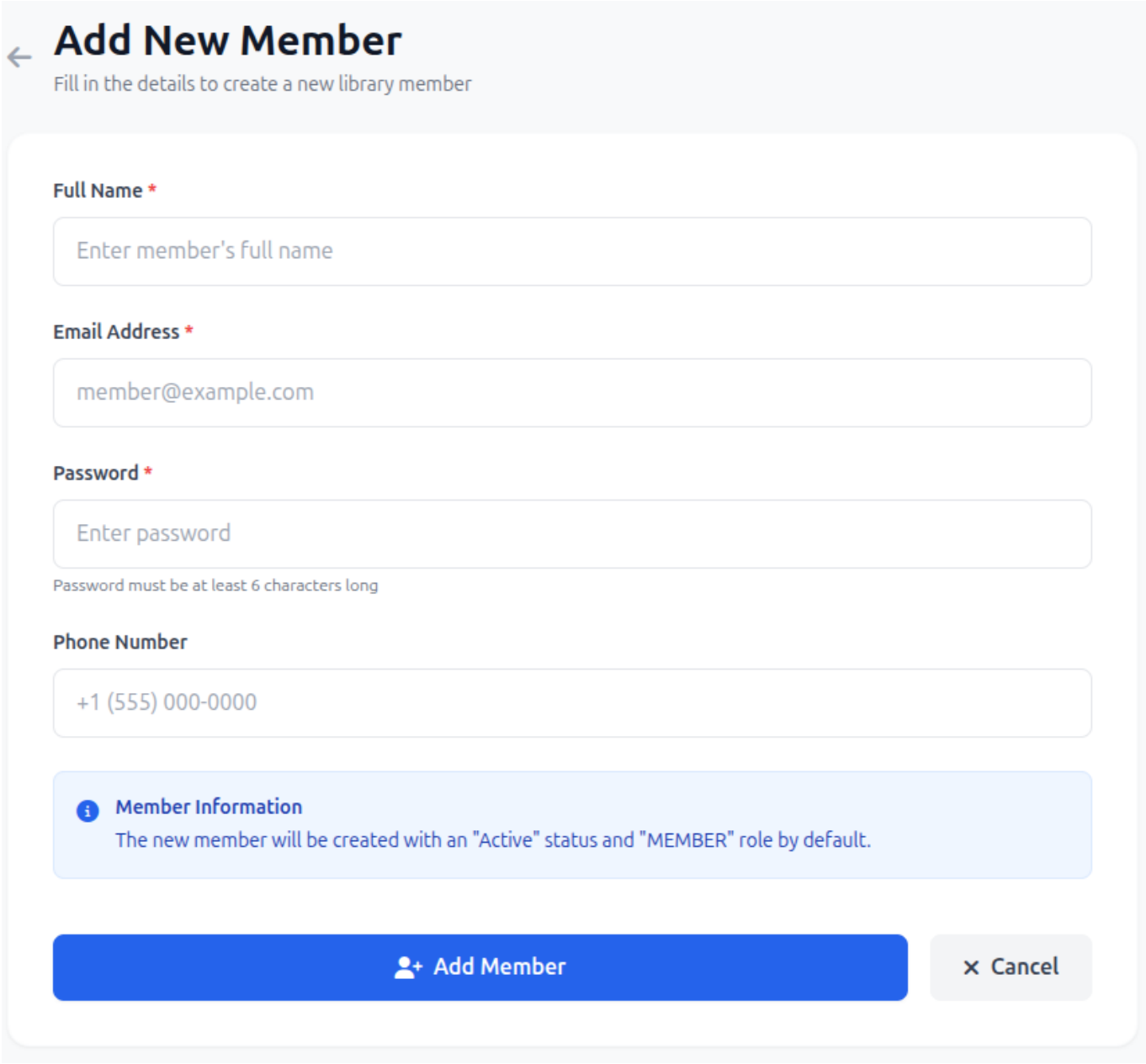
```
// 2. Summary Stats (Using BookService)
model.addAttribute("totalBooks", bookService.getTotalBooks());
model.addAttribute("borrowedBooks", bookService.getCountByStatus(status: "BORROWED"));
model.addAttribute("returnedBooks", bookService.getCountByStatus(status: "RETURNED"));
model.addAttribute("notBorrowed", bookService.getCountByStatus(status: "AVAILABLE"));

// 3. User Stats (Using UserService)
model.addAttribute("totalUsers", userService.getTotalMemberCount());
```

> - controller method interacts with userService -> userRepository and bookService -> bookRepository to get the counts of members, books, borrowed books, and overdue books from the database and pass them to the view using model attributes -> dashboard layout -> totalCard fragment.

```
public long getTotalMemberCount() {
    return userRepository.countByRole(User.Role.MEMBER);
}
```

```
// Filter by Role (Admin or Member)
Show AOT-generated Implementation, Query, etc...
List<User> findByRole(User.Role role);
```

- [ ] **Add new Member Form** (Form to register new members with validation)



How it works:

- Its html+css form will render when librarian clicks on the "Add New Member" by route "/members/add".

```
</head>
<body class="bg-[#F8F9FA] overflow-x-hidden font-sans antialiased text-gray-900">
    <aside class="fixed top-0 left-0 h-screen w-64 z-50">
        <div th:replace="~{fragments/sidebar :: sidebar}"></div>
    </aside>

    <div class="flex flex-col min-h-screen pl-64">

        <nav class="fixed top-0 right-0 left-64 h-16 bg-white/80 backdrop-blur-md z-40 border-b border-gray-100">
            <div th:replace="~{fragments/navigation :: navigation}"></div>
        </nav>

        <main class="p-10 mt-16 mx-auto w-full max-w-4xl">
            <!-- Header -->
            <div class="flex items-center gap-4 mb-6">
                <a href="/members" class="text-gray-400 hover:text-gray-600 transition-colors">
                    <i class="fa-solid fa-arrow-left text-xl"></i>
                </a>
                <div>
                    <h1 class="text-3xl font-bold text-gray-900">Add New Member</h1>
                    <p class="text-sm text-gray-500 mt-1">Fill in the details to create a new library member</p>
                </div>
            </div>

            <!-- Success Message -->
            <div th:if="${successMessage}" class="mb-6 p-4 bg-green-100 border border-green-300 text-green-700 rounded-lg flex items-center gap-3">
                <i class="fa-solid fa-check-circle"></i>
                <span th:text="${successMessage}">Success message</span>
            </div>

            <!-- Error Message -->
            <div th:if="${errorMessage}" class="mb-6 p-4 bg-red-100 border border-red-300 text-red-700 rounded-lg flex items-center gap-3">
                <i class="fa-solid fa-exclamation-circle"></i>
                <span th:text="${errorMessage}">Error message</span>
            </div>

            <!-- Form -->
            <div class="bg-white rounded-2xl shadow-sm border border-gray-50 p-8">
                <form action="/members/add" method="post" class="space-y-6">

                    <!-- Name Field -->
                    <div>
                        <label for="name" class="block text-sm font-semibold text-gray-700 mb-2">
                            Full Name <span class="text-red-500">*</span>
                        </label>
                        <input type="text"
                               id="name"
                               name="name"
```

- The form interacts with the route `/members/add` in the member controller to handle the form submission. The controller method works with userService to validate the input data and save the new member to the database.

```
    @PostMapping("/members/add")
    public String addMember(@RequestParam String name,
                            @RequestParam String email,
                            @RequestParam String password,
                            @RequestParam(required = false) String phone,
                            RedirectAttributes redirectAttributes) {
        try {
            User newUser = new User();
            newUser.setName(name);
            newUser.setEmail(email);
            newUser.setPassword(password);
            newUser.setRole(User.Role.MEMBER);
            newUser.setStatus(User.Status.Active);

            userService.createUser(newUser);

            redirectAttributes.addFlashAttribute("successMessage", "Member added successfully!");
            return "redirect:/members";
        } catch (Exception e) {
            redirectAttributes.addFlashAttribute("errorMessage", "Failed to add member: " + e.getMessage());
            return "redirect:/members/add";
        }
    }
```

- The data are sent to set in user model and then passed to userService to save the new member.

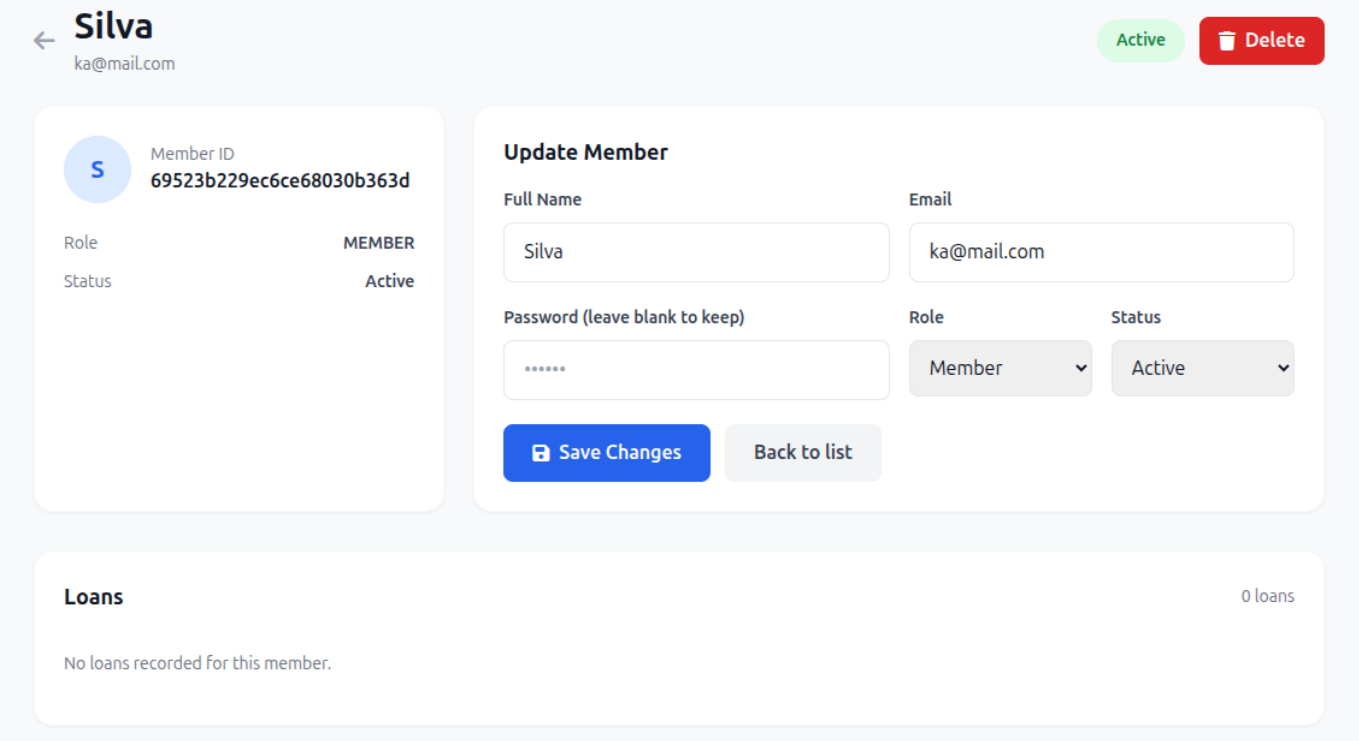
```
    public String getId() { return id; }
    public void setId(String id) { this.id = id; }
    public String getName() { return name; }
    public void setName(String name) { this.name = name; }
    public Role getRole() { return role; }
    public void setRole(Role role) { this.role = role; }
    public Status getStatus() { return status; }
    public void setStatus(Status status) { this.status = status; }
    public String getEmail() { return email; }
    public void setEmail(String email) { this.email = email; }
    public String getPassword() { return password; }
    public void setPassword(String password) { this.password = password; }
}
```

- The userService method to add a new member.

```
    // Create a new user (member)
    public User createUser(User user) {
        return userRepository.save(user);
    }
```

- [ ] **Update Member Form** (Form to update existing member details with validation)

- librarian can edit member information by clicking the "view" text next to each member in the member list, which will render the edit member form.



How it works:

- The edit member form interacts with the route `/members/{id}/update` in the member controller to handle the form submission. The controller method works with userService to validate the updated data and save the changes to the database.

```
<!-- Edit Form -->
<div class="bg-white rounded-2xl shadow-sm border border-gray-50 p-6 lg:col-span-2">
    <h2 class="text-lg font-bold text-gray-900 mb-4">Update Member</h2>
    <form th:action="@{'/members/' + ${member.id} + '/update'}" method="post" class="space-y-5">
        <div class="grid grid-cols-1 md:grid-cols-2 gap-4">
            <div>
```

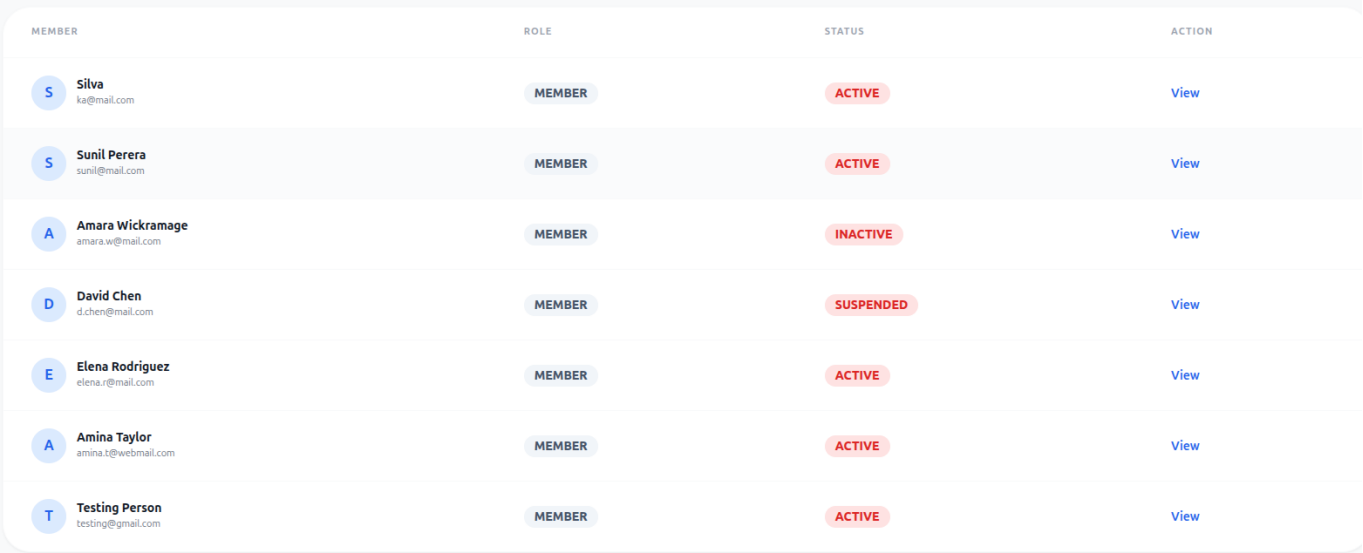
- The data are sent to set in user model and then passed to userService to update the member.

```
@PostMapping("/members/{id}/update")
public String updateMember(@PathVariable("id") String id,
                           @RequestParam String name,
                           @RequestParam String email,
                           @RequestParam(required = false) String password,
                           @RequestParam User.Role role,
                           @RequestParam User.Status status,
                           RedirectAttributes redirectAttributes) {
    return userService.getUserById(id)
        .map(user -> {
            user.setName(name);
            user.setEmail(email);
            user.setRole(role);
            user.setStatus(status);
            if (password != null && !password.isBlank()) {
                user.setPassword(password);
            }
            userService.updateUser(user);
            redirectAttributes.addFlashAttribute("successMessage", "Member updated successfully");
            return "redirect:/members/" + id;
        })
        .orElseGet(() -> {
            redirectAttributes.addFlashAttribute("errorMessage", "Member not found");
            return "redirect:/members";
        });
}
```

- The userService method to update an existing member.

```
// Update an existing user (member)
public User updateUser(User user) {
    return userRepository.save(user);
}
```

- ☐ **View All Member** (View all members in a table)



How it works:

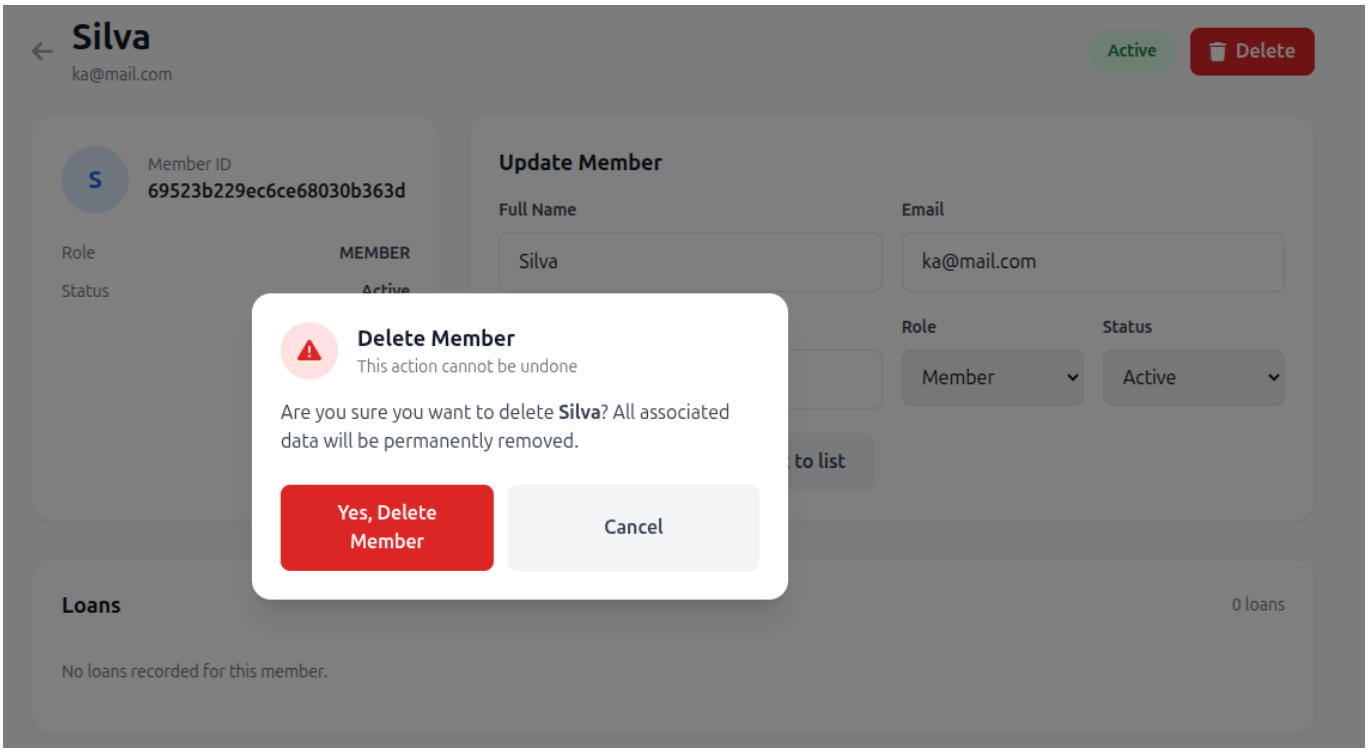
- The member list page interacts with the route `/members` in the member controller to retrieve all members from the database. The controller method works with userService to get the list of members and pass them to the view.

```java
@GetMapping("/members")
public String memberDisplay(@RequestParam(required = false) String search,
                            @RequestParam(required = false) String status,
                            Model model) {
    model.addAttribute("userName", "Nisal Gunasekara");
    model.addAttribute("activePage", "Admin");

    List<User> membersList;

    // Apply search if present
    if (search != null && !search.trim().isEmpty()) {
        membersList = userService.searchMembers(search);
        model.addAttribute("searchQuery", search);
    }
    // Apply status filter if present
    else if (status != null && !status.isEmpty()) {
        try {
            User.Status filterStatus = User.Status.valueOf(status);
            membersList = userService.filterMembersByStatus(filterStatus);
            model.addAttribute("filterStatus", status);
        } catch (IllegalArgumentException e) {
            membersList = userService.getMembers();
        }
    }
    // Default: show all members
    else {
        membersList = userService.getMembers();
    }

    model.addAttribute("membersList", membersList);

    // Add statistics
    model.addAttribute("totalMembers", userService.getTotalMemberCount());
    model.addAttribute("activeMembers", userService.getActiveMemberCount());
    model.addAttribute("inactiveMembers", userService.getInactiveMemberCount());

    return "layout/member_dash";
}
```

- The userService method to get all members.

```java
// Fetch only Members for the Member Table component
public List<User> getMembers() {
    return userRepository.findByRole(User.Role.MEMBER);
}
```

- [ ] **Delete Member** (Functionality to delete a member)



How it works:

- Its html+css in member list fragment

```
<!-- Delete Confirmation Modal -->
<div id="deleteModal" class="hidden fixed inset-0 bg-black bg-opacity-50 z-50 flex items-center justify-center p-4">
    <div class="bg-white rounded-2xl shadow-xl max-w-md w-full p-6">
        <div class="flex items-center gap-4 mb-4">
            <div class="w-12 h-12 bg-red-100 rounded-full flex items-center justify-center">
                <i class="fa-solid fa-triangle-exclamation text-red-600 text-xl"></i>
            </div>
            <div>
                <h3 class="text-lg font-bold text-gray-900">Delete Member</h3>
                <p class="text-sm text-gray-500">This action cannot be undone</p>
            </div>
        </div>
        <p class="text-gray-700 mb-6">
            Are you sure you want to delete <strong th:text="${member.name}">this member</strong>?
            All associated data will be permanently removed.
        </p>
        <div class="flex gap-3">
            <form th:action="@{'/members/' + ${member.id} + '/delete'}" method="post" class="flex-1">
                <button type="submit" class="w-full px-4 py-3 bg-red-600 hover:bg-red-700 text-white font-semibold rounded-lg transition-colors">
                    Yes, Delete Member
                </button>
            </form>
            <button onclick="document.getElementById('deleteModal').classList.add('hidden')"
                    class="flex-1 px-4 py-3 bg-gray-100 hover:bg-gray-200 text-gray-700 font-semibold rounded-lg transition-colors">
                Cancel
            </button>
        </div>
    </div>
</div>
```

- The delete member functionality is triggered when the librarian clicks the "Delete" button next to a member in the member list. This action interacts with the route `/members/{id}/delete` in the member controller. The controller method works with userService to delete the specified member from the database.

```
@PostMapping("/members/{id}/delete")
public String deleteMember(@PathVariable("id") String id, RedirectAttributes redirectAttributes) {
    try {
        userService.deleteUser(id);
        redirectAttributes.addFlashAttribute("successMessage", "Member deleted successfully");
    } catch (Exception e) {
        redirectAttributes.addFlashAttribute("errorMessage", "Failed to delete member: " + e.getMessage());
    }
    return "redirect:/members";
}
```

- The userService method to delete a member by ID.

```
72      // Delete a user by ID
73      public void deleteUser(String id) {
74          userRepository.deleteById(id);
75      }
```