Library Management System - Project Report Week 4

Date: January 11, 2026
Project: Library Borrowing and Book Tracking System
Development Phase: Contact Management Implementation & User Role Enhancement

---

📋 Executive Summary

This report documents the significant progress made since Week 3, focusing on implementing a comprehensive Contact/Support Management System, enhancing user role management, and improving system documentation. The project continues to follow best practices with proper authentication, authorization, and API documentation through Swagger/OpenAPI.

✅ Completed Features

1. Contact/Support Management System (MAJOR FEATURE)

1.1 Database Schema Design

File Created: `V5__create_contacts_table.sql`

Implemented a comprehensive contacts table with the following structure:

- **Primary fields**: ID, user_id (foreign key), subject, message
- **Status management**: Status field with ENUM values (OPEN, IN_PROGRESS, RESOLVED, CLOSED)
- **Admin response**: Admin response text and responded_at timestamp
- **Audit trail**: created_at and updated_at timestamps via BaseEntity
- **Referential integrity**: Foreign key constraint to users table

```sql
CREATE TABLE contacts (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    user_id BIGINT NOT NULL,
    subject VARCHAR(150) NOT NULL,
    message TEXT NOT NULL,
    status VARCHAR(30) NOT NULL DEFAULT 'OPEN',
    admin_response TEXT,
    created_at DATETIME,
    responded_at DATETIME,
    CONSTRAINT fk_contact_user FOREIGN KEY (user_id) REFERENCES users(id)
);
```

1.2 Backend Implementation

Entity Layer:

- **File:** `Contact.java`
- Extends BaseEntity for automatic timestamp management
- Uses JPA annotations for proper ORM mapping
- Implements ManyToOne relationship with User entity
- Uses ContactStatus enum for status field

Repository Layer:

- **File:** `ContactRepository.java`
- Extends JpaRepository for CRUD operations
- Custom query methods:
  - `findByUserId()` - Get user-specific contacts
  - `findByStatus()` - Filter by status
  - `findByCreatedAtBetween()` - Date range queries
  - `findByUserIdAndStatus()` - Combined filters
  - `countByStatus()` and `countByUserId()` - Analytics support

Service Layer:

- **Files:** `ContactService.java` (interface), `ContactServiceImpl.java` (implementation)
- Implements 5 core operations:
  1. `createContact()` - Create new support request
  2. `getMyContacts()` - Get user's own contacts
  3. `getAllContacts()` - Admin view all contacts
  4. `updateContact()` - Update status and response
  5. `deleteContact()` - Remove contact request
- Includes proper transaction management with `@Transactional`
- Comprehensive logging for all operations
- Input validation and error handling

Controller Layer - Two Separate Controllers:

1. **ContactController** (`/api/contacts`)

   - User-facing endpoints
   - POST `/api/contacts` - Create contact request
   - GET `/api/contacts/my` - View own contacts
   - Requires authentication via JWT
   - Full Swagger documentation with `@Operation` and `@ApiResponses`

2. **AdminContactController** (`/api/admin/contacts`)

   - Admin-only endpoints with `@PreAuthorize("hasRole('ADMIN') or hasRole('LIBRARIAN')")`
   - GET `/api/admin/contacts` - View all contact requests
   - PUT `/api/admin/contacts/{id}` - Update contact status/response
   - DELETE `/api/admin/contacts/{id}` - Delete contact request
   - Role-based access control enforced

1.3 DTOs and Request/Response Models

- **CreateContactRequest**: For creating new contacts
- **UpdateContactStatusRequest**: For admin updates
- **ContactResponse**: Standardized response format including user details

1.4 Contact Status Workflow

- **OPEN**: Newly created contact request
- **IN_PROGRESS**: Admin is working on the request
- **RESOLVED**: Issue has been resolved
- **CLOSED**: Request is closed

---

2. User Role Management Enhancement

2.1 Librarian Promotion to Admin

File Created: `V6__promote_librarian_to_admin.sql`

Implemented database migration to grant admin privileges:

- Ensures ADMIN role exists in the roles table
- Grants ADMIN role to librarian user (librarian@lms.com)
- Avoids duplicate role assignments with NOT EXISTS checks
- Maintains existing LIBRARIAN role (multi-role support)
- Properly handles role-based authorization

```sql
-- Make sure the ADMIN role exists
INSERT INTO roles (name)
SELECT 'ADMIN'
WHERE NOT EXISTS (
    SELECT 1 FROM roles r WHERE r.name = 'ADMIN'
);

-- Grant ADMIN role to the librarian user
INSERT INTO user_roles (user_id, role_id)
SELECT u.id, r.id
FROM users u
JOIN roles r ON r.name = 'ADMIN'
WHERE u.email = 'librarian@lms.com'
  AND NOT EXISTS (
      SELECT 1 FROM user_roles ur
      WHERE ur.user_id = u.id AND ur.role_id = r.id
  );
```

Business Logic:

- Allows librarians to have both LIBRARIAN and ADMIN roles
- Enables access to admin-level contact management
- Maintains proper role hierarchy

- Supports Spring Security's role-based access control

---

3. Comprehensive API Documentation

3.1 Contact API Documentation

File Created: `CONTACT_API.md`

Complete API reference including:

- **Endpoint descriptions**: All user and admin endpoints
- **Request/response examples**: JSON format samples
- **Security requirements**: Authentication and authorization
- **Status values**: All contact status enums
- **Database schema**: Table structure documentation
- **Error handling**: HTTP status codes and error scenarios

Key sections:

- User Contact Endpoints (create, view own)
- Admin Contact Endpoints (view all, update, delete)
- Contact status values and workflow
- Contact response format specification
- Database schema details
- Error handling scenarios (404, 400, 401)

---

4. Enhanced Swagger/OpenAPI Integration

Status: ✅ FULLY OPERATIONAL

4.1 Implementation Details

- **Swagger UI:** Accessible at `http://localhost:8080/swagger-ui/index.html`
- **OpenAPI JSON:** Available at `http://localhost:8080/api-docs`
- **Dependency:** SpringDoc OpenAPI 2.7.0

4.2 Features Configured

- Custom API information (title, description, version)
- JWT Bearer token authentication support
- Server configuration
- Operations and tags sorting
- Package scanning for controllers
- Public access to Swagger endpoints

4.3 Enhanced Controller Documentation

All controllers now include:

- `@Tag` annotations for API grouping
- `@Operation` annotations with summaries and descriptions
- `@ApiResponses` for all possible HTTP responses
- `@Parameter` annotations for request parameters
- `@SecurityRequirement` for authentication needs
- Schema definitions for request/response models

🏗 Technical Architecture

Current System Components

API Layers

1. **User Management APIs** (`/api/users`)
    - CRUD operations
    - User status management
    - Role assignment
2. **Authentication APIs** (`/api/login`, `/api/register`)
    - JWT-based authentication
    - User registration
    - Login/logout functionality
3. **Contact Management APIs**
    - User contact endpoints (`/api/contacts`)
    - Admin contact endpoints (`/api/admin/contacts`)
4. **System APIs** (`/api/system`)
    - System status
    - Health checks
    - Documentation links

Security Architecture

- **JWT Authentication**: Token-based authentication
- **Role-Based Access Control**: ADMIN, LIBRARIAN, STUDENT roles
- **Method-level Security**: `@PreAuthorize` annotations
- **Password Encryption**: BCrypt password encoder
- **Public Endpoints**: Swagger UI, login, register, static assets
- **Protected Endpoints**: All other APIs require JWT token

Database Migrations (Flyway)

- V1: Initial schema (users, roles, books, borrowings)
- V2: Alter users table

- V3: Add book descriptions and images
- V4: Update sample users
- V5: Create contacts table ✅ NEW
- V6: Promote librarian to admin ✅ NEW

📂 Project Structure Updates

New Files Added

Database Migrations

```
src/main/resources/db/migration/
├── V5__create_contacts_table.sql ✅ NEW
└── V6__promote_librarian_to_admin.sql ✅ NEW
```

Java Source Files

```
src/main/java/com/demo/lms/
├── controller/
│   └── contact/
│       ├── ContactController.java ✅ NEW
│       └── AdminContactController.java ✅ NEW
├── service/
│   └── contact/
│       ├── ContactService.java ✅ NEW
│       └── ContactServiceImpl.java ✅ NEW
├── repository/
│   └── ContactRepository.java ✅ NEW
├── model/
│   ├── entity/
│   │   └── Contact.java ✅ NEW
│   └── enums/
│       └── ContactStatus.java ✅ NEW
├── dto/
│   ├── request/
│   │   ├── CreateContactRequest.java ✅ NEW
│   │   └── UpdateContactStatusRequest.java ✅ NEW
│   └── response/
│       └── ContactResponse.java ✅ NEW
└── exception/
    └── ContactNotFoundException.java ✅ NEW
```

Documentation Files

```
document/
├── CONTACT_API.md ✅ NEW
├── AUTH_SETUP.md (existing)
├── SWAGGER_IMPLEMENTATION.md (existing)
├── USER_CRUD_API.md (existing)
├── ERROR_HANDLING.md (existing)
├── QUICK_START.md (existing)
└── README.md (existing)
```

🔧 Technical Improvements

1. Error Handling

- Custom exception: `ContactNotFoundException`
- Integrated with `GlobalExceptionHandler`
- Consistent error response format
- Proper HTTP status codes

2. Validation

- Jakarta validation annotations (`@Valid`, `@NotNull`, `@Size`)
- Input validation in DTOs
- Business logic validation in services

3. Logging

- SLF4J with Lombok `@Slf4j`
- Comprehensive logging at INFO and DEBUG levels
- Error logging with stack traces
- Request/response logging in controllers

4. Transaction Management

- `@Transactional` for write operations
- `@Transactional(readOnly = true)` for read operations
- Proper isolation and propagation

5. Code Quality

- Consistent naming conventions
- Proper separation of concerns (Controller → Service → Repository)
- Builder pattern for response objects
- Comprehensive JavaDoc comments
- Clean code principles

🧪 Testing & Verification

API Testing Capabilities

1. **Swagger UI**: Interactive API testing at http://localhost:8080/swagger-ui/index.html
2. **Manual Testing**: Postman/cURL compatible
3. **Authentication**: JWT token support in Swagger

Test Scenarios Covered

- ✅ Create contact request
- ✅ View user's own contacts
- ✅ Admin view all contacts
- ✅ Admin update contact status
- ✅ Admin respond to contacts
- ✅ Admin delete contacts
- ✅ User not found error handling
- ✅ Contact not found error handling
- ✅ Authorization checks (role-based)

📊 Database Schema Status

Tables Implemented

1. **users** - User accounts
2. **roles** - System roles (ADMIN, LIBRARIAN, STUDENT)
3. **user_roles** - Many-to-many relationship
4. **books** - Book catalog
5. **borrowings** - Borrowing records
6. **contacts** - Support requests ✅ NEW

Relationships

- users → contacts (one-to-many)
- users → borrowings (one-to-many)
- books → borrowings (one-to-many)
- users → roles (many-to-many via user_roles)

🚀 Build & Deployment Status

Build Configuration

- **Build Tool**: Maven
- **Java Version**: 17
- **Spring Boot Version**: Latest
- **Database**: MySQL/H2 (configurable)
- **Migration Tool**: Flyway

Build Commands

```
# Clean and package
./mvnw.cmd clean package -DskipTests

# Run application
java -jar target/LMS-0.0.1-SNAPSHOT.jar --server.port=8080
```

Application Status

- ✅ Build: SUCCESS
- ✅ Database Migrations: Applied successfully
- ✅ Server: Running on port 8080
- ✅ Swagger UI: Accessible
- ✅ Authentication: Working
- ✅ Contact APIs: Operational

📈 Progress Metrics

Code Statistics

- **New Controllers**: 2
- **New Services**: 1 interface + 1 implementation
- **New Repositories**: 1
- **New Entities**: 1
- **New DTOs**: 3
- **New Exceptions**: 1
- **New Database Migrations**: 2
- **New Documentation Files**: 1
- **API Endpoints Added**: 5

Feature Completion

- ✅ Contact Management System: 100%
- ✅ User Role Enhancement: 100%
- ✅ API Documentation: 100%
- ✅ Swagger Integration: 100%

🔄 Compared to Previous Reports

Evolution from Week 3

Week 3 Focus:

- Error handling enhancement
- User API documentation
- Swagger implementation
- Authentication setup

Week 4 (Current) Focus:

- Contact/Support system (NEW MAJOR FEATURE)
- Role management enhancement
- Admin capabilities
- Multi-role support

New Capabilities Added

1. **Support Ticket System**: Users can now submit support requests
2. **Admin Management**: Admins can view, update, and respond to contacts
3. **Status Tracking**: Contact requests have status workflow
4. **Multi-Role Support**: Users can have multiple roles simultaneously
5. **Enhanced Authorization**: Role-based endpoint protection

---

📚 Documentation Coverage

Complete Documentation Set

1. **CONTACT_API.md** ✅ NEW
   - Contact endpoint reference
   - Request/response formats
   - Status workflow
   - Error scenarios
2. **AUTH_SETUP.md** (existing)
   - Authentication implementation
   - JWT configuration
   - Security matrix
3. **SWAGGER_IMPLEMENTATION.md** (existing)
   - Swagger setup guide
   - API testing instructions
   - Authentication in Swagger
4. **USER_CRUD_API.md** (existing)
   - User management endpoints
   - CRUD operations
   - Status management
5. **ERROR_HANDLING.md** (existing)
   - Error response structure
   - Exception types
   - HTTP status codes
6. **QUICK_START.md** (existing)

  - 5-minute setup guide
  - Prerequisites
  - First API test

7. **README.md** (existing)

  - Project overview
  - Getting started
  - Documentation index

🎯 Key Achievements

1. Production-Ready Contact System

- Complete CRUD operations
- Role-based access control
- Proper error handling
- Comprehensive validation
- Full API documentation

2. Enhanced User Management

- Multi-role support
- Librarian admin promotion
- Flexible role assignment
- Proper authorization

3. Professional API Design

- RESTful conventions
- Consistent response formats
- Proper HTTP status codes
- Swagger/OpenAPI documentation

4. Security Best Practices

- JWT authentication
- Role-based authorization
- Method-level security
- Protected endpoints

5. Code Quality

- Clean architecture
- Separation of concerns
- Comprehensive logging
- Transaction management
- Exception handling

🔜 Future Enhancements (Recommendations)

Short-Term

1. **Email Notifications**: Send emails when contacts are created/responded
2. **Contact Categories**: Add category field for better organization
3. **Priority Levels**: Urgent, high, normal, low priorities
4. **Attachment Support**: Allow file attachments in contact requests

Medium-Term

1. **Contact Analytics Dashboard**: Statistics and reports
2. **Auto-Assignment**: Automatically assign contacts to admins
3. **SLA Management**: Service level agreement tracking
4. **Bulk Operations**: Update multiple contacts at once

Long-Term

1. **AI-Powered Responses**: Suggest responses using AI
2. **Live Chat Integration**: Real-time support
3. **Knowledge Base**: FAQ and self-service portal
4. **Mobile App Support**: iOS/Android applications

🎓 Learning Outcomes

Technical Skills Demonstrated

1. **Spring Boot Framework**: Controllers, services, repositories
2. **Spring Security**: Authentication and authorization
3. **JPA/Hibernate**: Entity relationships and queries
4. **Flyway Migrations**: Database version control
5. **Swagger/OpenAPI**: API documentation
6. **RESTful API Design**: Best practices and conventions
7. **Error Handling**: Exception handling strategies
8. **Transaction Management**: Data consistency
9. **Logging**: Application monitoring
10. **Git Version Control**: Code management

Software Engineering Practices

- Clean code principles
- SOLID principles
- Design patterns (Builder, Repository, Service)
- API-first development
- Documentation-driven development
- Test-driven development mindset

📞 System Capabilities Summary

User Capabilities

- ☑ Register and login
- ☑ Submit contact/support requests
- ☑ View own contact history
- ☑ Borrow books
- ☑ View book catalog

Librarian Capabilities

- ☑ All user capabilities
- ☑ View all contact requests
- ☑ Update contact status
- ☑ Respond to contacts
- ☑ Delete contacts
- ☑ Manage book inventory

Admin Capabilities

- ☑ All librarian capabilities
- ☑ User management (CRUD)
- ☑ Role assignment
- ☑ System configuration
- ☑ Full contact management

🏁 Conclusion

Week 4 has been highly productive with the successful implementation of a complete Contact/Support Management System. This feature significantly enhances the Library Management System by providing a structured way for users to communicate with administrators and request support. The implementation follows industry best practices with proper authentication, authorization, error handling, and comprehensive documentation.

The multi-role support enhancement allows for more flexible user management, enabling staff members to have multiple roles simultaneously. This architectural improvement provides better scalability and role management for future features.

All features are production-ready with full Swagger documentation, proper error handling, and comprehensive logging. The codebase maintains high quality standards with clean architecture, proper separation of concerns, and adherence to SOLID principles.

Overall Status: ☑ ON TRACK

📋 Appendix

Sample Test Scenarios

Test 1: Create Contact Request

```
POST http://localhost:8080/api/contacts?userId=1
Authorization: Bearer {token}
Content-Type: application/json

{
  "subject": "Book not available",
  "message": "I'm trying to borrow 'The Great Gatsby' but it's not in the
catalog."
}
```

Test 2: Admin View All Contacts

```
GET http://localhost:8080/api/admin/contacts
Authorization: Bearer {admin-token}
```

Test 3: Admin Update Contact

```
PUT http://localhost:8080/api/admin/contacts/1
Authorization: Bearer {admin-token}
Content-Type: application/json

{
  "status": "RESOLVED",
  "adminResponse": "The book has been added to the catalog. You can now borrow
it."
}
```

Contact Request Lifecycle

1. User creates contact → Status: OPEN
2. Admin sees contact in queue
3. Admin updates status → Status: IN_PROGRESS
4. Admin responds to user
5. Admin marks as resolved → Status: RESOLVED
6. Admin closes ticket → Status: CLOSED

Report Generated: January 11, 2026
Author: Development Team
Version: 1.0
Project: Library Management System