Library Management System - Project Report

A full-stack Library Management System web application featuring CRUD operations for book management, JWT-based user authentication, and a responsive React interface. Built with ASP.NET Core 9.0 backend, React 18 with TypeScript frontend, and SQLite database.

Key Metrics:

- 7 RESTful API endpoints
- 12+ React components
- JWT authentication with password hashing
- Fully responsive design
- Search and filter functionality
- Real-time data synchronization
- Local development environment ready

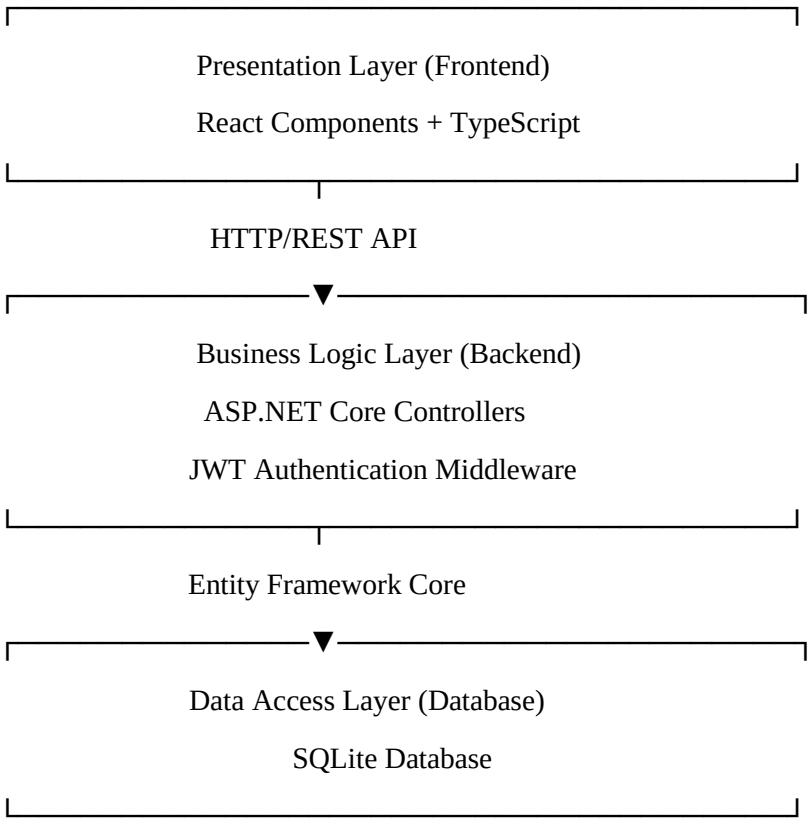
1. Project Overview

1. Objectives
    - Implement complete CRUD functionality for book management
    - Secure user authentication using JWT tokens
    - Create responsive, intuitive user interface
    - Apply RESTful API design principles
    - Demonstrate full-stack development capabilitie

2. Technology Stack
    - Backend - ASP.NET Core 9.0 - REST API, Business Logic
    - Frontend - React 18 + TypeScript - User Interface
    - Database - SQLite - Data Persistence
    - ORM - Entity Framework Core 9.0 - Database Access
    - Auth - JWT - Token-based Authentication
    - HTTP Client - Axios - API Communication
    - Routing - React Router 6 - SPA Navigation

3. System Architecture

Three-Tier Architecture



2. Communication Flow

1. User interacts with React UI
2. Axios sends HTTP requests to API
3. Controllers validate JWT tokens (for protected routes)
4. Business logic processes requests
5. Entity Framework queries database
6. Response sent back to frontend
7. UI updates with new data

3. Backend Implementation

1. Project Structure

   Api/

   ```
   ├── Controllers/
   │   ├── BooksController.cs      # CRUD operations
   │   └── AuthController.cs       # Login/Register
   ├── Models/
   │   ├── Book.cs                 # Book entity
   │   └── User.cs                 # User entity
   ├── Data/
   │   └── AppDbContext.cs         # EF Core context
   ├── Services/
   │   └── TokenService.cs         # JWT generation
   └── Program.cs                  # Configuration
   ```

2. API Endpoints

   Authentication:

   - POST /api/auth/register - User registration
   - POST /api/auth/login   - User login (returns JWT)

   Books (Protected with JWT):

   - GET   /api/books   - Get all books
   - GET   /api/books/{id}   - Get book by ID
   - POST   /api/books   - Create book [Auth Required]
   - PUT   /api/books/{id}   - Update book [Auth Required]
   - DELETE /api/books/{id}   - Delete book [Auth Required]

3. Key Features

   Security Implementation:

- Password hashing with HMACSHA512
- Unique salt per user
- JWT tokens with 60-minute expiration
- Token validation middleware
- CORS configuration for frontend access

4. Data Validation:
    - [Required(ErrorMessage = "Title is required.")]
    - [StringLength(200, MinimumLength = 1)]
    - public string Title { get; set; }

5. Database Context:
    - Automatic migrations
    - LINQ queries
    - Change tracking
    - Transaction management

4. Frontend Implementation

1. Project Structure

```
library-frontend/
├── components/
│   ├── Navbar.tsx            # Navigation
│   ├── PrivateRoute.tsx      # Route protection
│   ├── BookDetailsModal.tsx  # Details popup
│   └── ConfirmModal.tsx      # Delete confirmation
├── pages/
│   ├── BookList.tsx          # Main book listing
│   ├── BookForm.tsx          # Add/Edit form
│   ├── Login.tsx             # Authentication
│   └── Register.tsx          # User registration
├── services/
│   ├── bookService.ts        # Book API calls
│   └── authService.ts        # Auth API calls
```

└── types/

└── Book.ts # TypeScript interfaces

2. Key Components

BookList Component:

- Displays books in responsive table
- Real-time search and filter
- Click-to-view details modal
- Edit and delete actions
- Authentication-aware UI

BookForm Component:

- Single form for create/update
- Real-time validation
- Success/error notifications
- Pre-populated data for edits

Authentication:

- Login/Register forms
- Token storage in localStorage
- Automatic token inclusion in requests
- Protected routes for authenticated pages

TypeScript Integration:

```
export interface Book {
  id?: number;
  title: string;
  author: string;
  description?: string;
  createdAt?: string;
}
```

State Management

- React Hooks (useState, useEffect)
- LocalStorage for token persistence
- Real-time UI updates after operations
- Context API for global auth state

5. Database Design

Books Table:

- Id INTEGER - PRIMARY KEY, AUTO INCREMENT
- Title TEXT - NOT NULL, MAX 200 chars
- Author TEXT - NOT NULL, MAX 100 chars
- Description TEXT
- CreatedAt DATETIME

Users Table:

- Id INTEGER - PRIMARY KEY, AUTO INCREMENT
- Username TEXT - UNIQUE, NOT NULL
- PasswordHash BLOB - NOT NULL
- PasswordSalt BLOB - NOT NULL
- CreatedAt DATETIME - NOT NULL, DEFAULT NOW

Migration Strategy

- Entity Framework Core migrations
- Version-controlled schema changes
- Easy rollback capabilities
- Production-ready for Azure SQL migration

6. Authentication & Security

Password Security

- Hashing Process:
- User provides password
- Generate random salt (unique per user)
- Hash = HMACSHA512(password + salt)
- Store hash and salt in database

Login: Repeat process and compare hashes

JWT Authentication

Token Structure:

```
{
  "sub": "username",
  "exp": 1702234567,
  "iat": 1702230967
}
```

Authentication Flow:

User Login → Validate Credentials → Generate JWT

↓

Frontend stores token in localStorage

↓

Protected Request → Include token in Authorization header

↓

Backend validates token → Process request

Security Measures Implemented

- Password hashing (never stored plain text)
- Unique salt per user
- JWT token expiration (60 minutes)
- Token signature verification
- CORS policy configuration
- SQL injection prevention (EF Core)
- XSS protection (React escaping)
- Input validation (frontend + backend)
- HTTPS ready for production

7. Core Features

1. Book Management (CRUD)

- Create: Add books with title, author, description
- Read: View all books with search/filter
- Update: Edit existing book details
- Delete: Remove books with confirmation

2. User Authentication

- Secure registration with validation
- Login with JWT token generation
- Persistent sessions (localStorage)
- Protected routes enforcement

3. Search & Filter

- Real-time search (title, author)
- Case-insensitive filtering
- Instant results without reload

4. User Experience

- Book details modal on click
- Loading indicators
- Success/error notifications
- Confirmation dialogs
- Smooth transitions
- Mobile-responsive design

5. Additional Features

- Custom book-themed favicon (SVG)
- Consistent branding and colors
- Form validation with feedback
- Formatted dates display
- Dark mode ready CSS structure

8. Challenges & Solutions

Technical Challenges Overcome

1. CORS Configuration

- Challenge: Frontend couldn't connect to backend API
- Solution: Configured CORS policy to allow frontend origin
- Learning: Browser security policies and proper configuration

2. JWT Token Management

- Challenge: Token storage and automatic inclusion in requests
- Solution: localStorage + Axios interceptors
- Learning: Client-side security considerations

3. Database Schema Evolution

- Challenge: Adding CreatedAt field after deployment
- Solution: Entity Framework migration
- Learning: Importance of schema planning and migrations

4. TypeScript Type Safety

- Challenge: Type mismatches between frontend/backend
- Solution: Created matching TypeScript interfaces
- Learning: Benefits of strong typing

5. Form State Management

- Challenge: Reusing form for both create and edit
- Solution: Conditional logic based on URL parameters
- Learning: React component reusability patterns

6. State Synchronization

- Challenge: Keeping UI in sync with backend
- Solution: useEffect hooks and proper state updates
- Learning: React lifecycle management

7. Error Handling

- Challenge: Graceful API error handling
- Solution: Try-catch blocks with user-friendly messages
- Learning: Defensive programming importance

8. Responsive Design

- Challenge: Making UI work on all screen sizes
- Solution: Mobile-first CSS with media queries
- Learning: Modern responsive techniques

9. Testing & Validation

Test Coverage

Authentication Tests:

- Register with valid credentials
- Register with duplicate username (error handling)
- Login with correct credentials
- Login with incorrect password (error handling)
- Access protected routes without token (denied)
- Access protected routes with valid token (allowed)

Book Management Tests:

- Create book with all fields
- Create book with required fields only
- Create book with missing fields (validation error)
- Update existing book
- Delete book with confirmation
- Search books by title
- Filter books by author

10. Key Learning Outcomes

Technical Skills Acquired

Backend Development:

- ASP.NET Core Web API architecture
- Entity Framework Core and ORM concepts
- RESTful API design principles
- JWT authentication implementation
- Password security (hashing, salting)

- Dependency injection patterns
- Middleware configuration

Frontend Development:

- React component architecture
- TypeScript type safety
- State management with hooks
- Asynchronous programming (async/await)
- HTTP client integration (Axios)
- React Router navigation
- Form handling and validation

Database:

- Relational database design
- SQL and LINQ queries
- Database migrations
- Entity relationships
- Data validation

Security:

- Authentication vs Authorization
- Token-based authentication
- Password security best practices
- CORS and web security
- Input validation and sanitization

Software Engineering Principles

Applied Concepts:

- Clean code principles
- Separation of concerns
- DRY (Don't Repeat Yourself)
- Single Responsibility Principle
- Three-tier architecture
- Component-based architecture
- Git version control
- Incremental development
- Testing and debugging strategies

Professional Skills

- Breaking complex projects into tasks
- Time management

- Technical documentation
- Problem-solving and research
- Reading documentation effectively
- Code commenting and clarity

11. Development Environment & Deployment Readiness

Development Setup

- Backend: dotnet run on localhost:5013
- Frontend: npm start on localhost:3000
- Database: SQLite (local file)
- Version Control: Git with GitHub repository

Deployment Readiness

Project is production-ready with:

- Clean architecture and separation of concerns
- Environment-based configuration files
- Database migrations for easy schema deployment
- CORS configuration for cross-origin requests
- JWT authentication for secure API access
- Comprehensive error handling
- Input validation on both frontend and backend
- Responsive design for all devices

Deployment Options:

- Cloud Platforms: Azure, AWS, Google Cloud, Heroku
- Database: Can migrate from SQLite to SQL Server, PostgreSQL, or MySQL
- Frontend Hosting: Azure Static Web Apps, Netlify, Vercel
- CI/CD: GitHub Actions, Azure DevOps, Jenkins

12. Future Enhancements

Planned Features

1. Advanced Functionality:

- Multi-criteria search with sorting
- Pagination for large datasets
- Book categories and tags
- User roles (Admin, Librarian, Member)
- Borrowing system with due dates

- Fine calculation for overdue books

2. User Management:

- User profiles with avatars
- Password change functionality
- Email verification
- Password reset via email
- Two-factor authentication

3. Reporting:

- Popular books dashboard
- User activity reports
- Inventory statistics
- Export to PDF/Excel

4. Enhanced UI:

- Book cover image uploads
- Star rating system
- User reviews and comments
- Dark mode toggle
- Accessibility improvements (WCAG)
- Multi-language support

Technical Improvements

Performance:

- Implement caching (Redis)
- Database query optimization
- Frontend code splitting
- Image lazy loading
- API response compression

Testing:

- Unit tests (xUnit, Jest)
- Integration tests
- E2E tests (Cypress)
- Automated CI/CD testing

DevOps:

- Docker containerization
- Kubernetes orchestration
- Monitoring (Application Insights)
- Centralized logging
- Automated backups

13. Conclusion

Project Achievements

This Library Management System successfully demonstrates comprehensive full-stack development capabilities with:

- Complete CRUD Implementation - All operations functional and tested
- Secure Authentication - Industry-standard JWT with password hashing
- Responsive UI - Works seamlessly across all devices
- RESTful Architecture - Clean, maintainable API design
- Type Safety - TypeScript prevents runtime errors
- Production Ready - Deployment-ready architecture
- Professional Code - Well-documented and organized

Skills Demonstrated

- Full-stack web development (React + ASP.NET Core)
- Database design and ORM usage
- Authentication and security implementation
- API design and documentation
- Version control with Git and GitHub
- Problem-solving and debugging
- Professional documentation

Key Insights

1. Security First: Authentication must be designed into the system from the start, not added later.

2. Type Safety Matters: TypeScript caught numerous potential bugs during development.

3. Validation Everywhere:Both frontend and backend validation are essential - never trust client-side only.

4. Iterative Development: Building incrementally and testing continuously leads to stable applications.

5. User Experience: Technical functionality is important, but UI/UX determines real-world success.

6. Documentation Value: Good documentation saves time and prevents confusion later.

7. Modern Frameworks: React and ASP.NET Core accelerate development while maintaining quality.

Professional Impact

This project provides:

- Portfolio-ready demonstration of full-stack capabilities
- Practical experience with enterprise-level technologies
- Understanding of complete software development lifecycle
- Foundation for continued learning and growth
- Deployment-ready architecture and best practices
- Real-world problem-solving skills

Final Reflection

The Library Management System project successfully bridges theoretical knowledge with practical application. The challenges encountered - from CORS configuration to state management - provided invaluable hands-on experience that complements classroom learning. The implemented security measures, clean architecture, and responsive design demonstrate readiness for professional software development roles.

The comprehensive feature set and production-ready architecture showcase ability to deliver complete, user-ready applications. This project serves as both an academic achievement and a stepping stone toward a career in software engineering.

Report Prepared By : K.K.Indumini Theekshana De Alwis

Date: December 7, 2025

Project: Library Management System