Software Requirements Specification (SRS)

Library Management System (LMS)

---

1. Problem Statement

The Library Management System (LMS) is a comprehensive solution that automates the management of books, users, categories, authors, publishers, and borrowing transactions. The system implements a role-based architecture with inheritance (User → Customer / Employee / Admin) to ensure data consistency, avoid duplication, and provide scalable user management with appropriate access controls.

---

2. System Requirements

2.1 Functional Requirements

The Library Management System supports the following role-based functionalities:

2.1.1 Authentication System

- User registration (Customer signup)
- User login with role-based redirection
- Input validation (email format, phone number, password strength)
- Duplicate email prevention

2.1.2 Customer Functions

- Search books by title
- Browse all available books
- Borrow books with availability checking
- Return books with automatic late fee calculation
- View book details (title, ISBN, language, pages, etc.)

2.1.3 Employee Functions

- Add new books to the inventory
- Edit existing book information (title, ISBN, copies, language, pages, year)
- Remove books from the system
- Manage book metadata (authors, publishers, categories)

- Update book relationships and associations

2.1.4 Administrator Functions

- Add new employees to the system
- Remove employees and customers
- Edit employee and customer information
- Complete user management across all roles
- System-wide administrative control

2.1.5 System Functions

- Display library policies
- Calculate and track late fees (5 EGP per day)
- Real-time inventory management
- Automatic status tracking for borrowed books
- Date-based due date management

---

3. Entities and Attributes

3.1 User (Base Entity)

- **user_id** (PK, AUTO_INCREMENT)
- **first_name** (VARCHAR(50))
- **last_name** (VARCHAR(50))
- **email** (VARCHAR(100), UNIQUE)
- **phone_no** (VARCHAR(15))
- **username** (VARCHAR(50), UNIQUE)
- **password** (VARCHAR(100))
- **role** (ENUM: 'Employee', 'Customer', 'Admin')

3.2 Customer (Inherits from User)

- **customer_id** (PK, FK → User.user_id)
- **address** (VARCHAR(100))

3.3 Employee (Inherits from User)

- **employee_id** (PK, FK → User.user_id)

3.4 Admin (Inherits from User)

- **admin_id** (PK, FK → User.user_id)

3.5 Book

- **book_id** (PK, AUTO_INCREMENT)
- **title** (VARCHAR(100), NOT NULL)
- **isbn** (VARCHAR(20), UNIQUE)
- **language** (VARCHAR(50))
- **no_of_copies** (INT)
- **no_of_pages** (INT)
- **released_year** (INT)
- **category_id** (FK → Category.category_id)
- **publisher_id** (FK → Publisher.publisher_id)
- **author_id** (FK → Author.author_id)

3.6 Category

- **category_id** (PK, AUTO_INCREMENT)
- **category_name** (VARCHAR(50), NOT NULL)

3.7 Author

- **author_id** (PK, AUTO_INCREMENT)
- **author_name** (VARCHAR(100))

3.8 Publisher

- **publisher_id** (PK, AUTO_INCREMENT)
- **publisher_name** (VARCHAR(100))

3.9 Borrow

- **issued_id** (PK, AUTO_INCREMENT)
- **book_id** (FK → Book.book_id)
- **customer_id** (FK → Customer.customer_id)

- **issue_date** (DATE)
- **due_date** (DATE)
- **return_date** (DATE)
- **status** (ENUM: 'borrowed', 'returned', 'late', DEFAULT 'borrowed')
- **fees** (DECIMAL for late fees)

3.10 BookAuthor (Junction Table)

- **book_id** (FK → Book.book_id)
- **author_id** (FK → Author.author_id)

3.11 Policies

- **policy_id** (PK, AUTO_INCREMENT)
- **policy_type** (VARCHAR(50))
- **policy_text** (TEXT)

---

4. Relationships & Cardinality

4.1 User Specialization

- **User → Customer**: 1:1 (ISA relationship)
- **User → Employee**: 1:1 (ISA relationship)
- **User → Admin**: 1:1 (ISA relationship)

4.2 Book Relationships

- **Category → Books**: 1:M (One category contains many books)
- **Publisher → Books**: 1:M (One publisher publishes many books)
- **Author → Books**: 1:M (One author writes many books)
- **Book ↔ Author**: M:N (Many-to-many via BookAuthor junction table)

4.3 Transaction Relationships

- **Customer → Borrow**: 1:M (One customer has many borrow records)
- **Book → Borrow**: 1:M (One book can be borrowed multiple times)

---

5. User Roles and Capabilities

5.1 Customer (Inherits from User)

Primary Functions:

- Browse complete book catalog
- Search books by title with detailed information display
- Borrow available books (automatic inventory update)
- Return borrowed books (automatic fee calculation if late)
- View personal borrowing history and status

Business Rules:

- Can only borrow books with available copies (no_of_copies > 0)
- Must provide valid return date
- Subject to late fees: 5 EGP per day after due date
- Account creation requires valid email (@gmail.com) and Egyptian phone number (+201xxxxxxxx)

5.2 Employee (Inherits from User)

Primary Functions:

- Complete book inventory management (Add/Edit/Remove)
- Manage book metadata and relationships
- Handle author, publisher, and category associations
- Update book details including copies, language, pages, publication year
- Dynamic creation of new authors, publishers, and categories

Advanced Capabilities:

- Multi-field book editing with granular control
- Referential integrity management
- Inventory tracking and availability updates
- Comprehensive book catalog maintenance

5.3 Administrator (Inherits from User)

Primary Functions:

- Complete employee lifecycle management (Add/Edit/Remove)

- Customer account management and maintenance
- System-wide user administration
- Access to all system functions and data

Administrative Powers:

- Create and remove employee accounts
- Edit user information across all roles
- System configuration and user management
- Override capabilities for system maintenance

---

6. System Architecture

6.1 Design Patterns

- **Inheritance**: User base class with Customer, Employee, Admin specializations
- **MVC-inspired**: Separation of concerns with core business logic, database layer, and service layer
- **Factory Pattern**: Dynamic creation of authors, publishers, and categories
- **Connection Management**: Centralized database connection handling

6.2 Data Validation Rules

- **Email**: Must end with @gmail.com
- **Phone**: Must follow Egyptian format (+201xxxxxxxxx, exactly 13 characters)
- **Password**: Minimum 6 characters
- **Duplicate Prevention**: Email and username uniqueness enforced

6.3 Business Logic

- **Automatic Inventory**: Real-time copy count updates during borrow/return
- **Late Fee Calculation**: 5 EGP per day for overdue books
- **Status Tracking**: Automatic status updates (borrowed/returned/late)
- **Data Integrity**: Foreign key constraints and referential integrity

---

7. Non-Functional Requirements

7.1 Security

- Role-based access control
- SQL injection prevention through parameterized queries
- Input validation and sanitization
- Password-based authentication

7.2 Performance

- Efficient database queries with proper indexing
- Real-time inventory updates
- Optimized join operations for book details

7.3 Reliability

- Transaction consistency for borrow/return operations
- Data integrity constraints
- Error handling and recovery mechanisms

7.4 Usability

- Command-line interface with clear menu navigation
- Intuitive role-based workflows
- Comprehensive error messages and user feedback

---

8. System Constraints

8.1 Technical Constraints

- Python 3.7+ runtime environment
- MySQL 8.0+ database server
- mysql-connector-python dependency

8.2 Business Constraints

- Single library system (not multi-tenant)
- Egyptian phone number format requirement
- Gmail-only email validation
- Fixed late fee structure

8.3 Operational Constraints

- Command-line interface only
- Single-user session handling
- Manual database setup required

---

9. Future Enhancements

9.1 Potential Improvements

- Web-based user interface
- Multi-tenant support for multiple libraries
- Advanced reporting and analytics
- Book reservation system
- Digital book management
- SMS/Email notifications for due dates

9.2 Scalability Considerations

- Database optimization for large datasets
- Caching layer implementation
- Multi-user concurrent access
- API development for external integrations