# Low-Level Design Document

## Library Management System:

Table Of Contents:

## Introduction:

LMS is designed for managing and maintaining libraries in any educational institution through an integrated computerized system. The library management software will allow librarians to operate more productively while handling the typical day-to-day tasks of a library.

In a traditional library management system, everything is done manually. All the library operations and records, including the number of books, genres of books, names of books, records of the students who've issued/returned books, etc., are all done via pen and paper. Naturally, this process requires a significant amount of time, effort, and even human resources. The proposed project seeks to solve all the challenges associated with the traditional library management system. Since it stores and manages all the library records in a computerized database, it eliminates the need for manual record-keeping. The software includes different modules, each of which handles and manages specific library operations. By using this software application, librarians and students need not search the entire library to find a book. They can enter the name and author of the book, and the system will display the list of all the possible books available for that search keyword/phrase. This is one of the best features of this library management software.

## The scope of the Library Management System includes the following:

Book Management: This module manages the entire lifecycle of books in the library, including adding new books, updating book information, and assigning unique identifiers to each book. It also manages the classification and indexing of library materials.

Borrowing and Returning: This module manages the borrowing and returning of books, including issuing library cards to patrons, recording due dates, sending overdue notices, and managing fines and fees.

Reservations: This module allows patrons to reserve books that are currently checked out and be notified when the book is available.

Catalog Search: This module allows patrons to search for books in the library catalog and locate the book on the shelves.

User Management: This module manages user accounts, including creating and deleting accounts, updating user information, and setting user permissions.

The system can be used by libraries in any educational institution to automate their operations and provide efficient services to library users. It can be customized to fit the specific needs of the library and its users.

## Overview of the system architecture:

### User Interface:

This layer provides the graphical user interface for the library management system. It allows users to interact with the system to perform various operations, such as searching the catalog, borrowing books, and managing their accounts.

### Application Services:

This layer provides the business logic and data access services for the library management system. It encapsulates the core functionality of the system, such as cataloging, circulation, reservation, and User Management.

### Data Storage:

This layer provides the data storage and management services for the library management system. It stores all the data related to the library, such as book information, user accounts, circulation history, and other administrative data.

### Integration Services:

This layer provides the integration services for the library management system. It integrates with other systems, such as User Management Module, Search and Retrieval Module, and other library-related systems.

### Security and Authentication:

This layer provides the security and authentication services for the library management system. It ensures that all user data and library resources are protected and secure.

## Overview of the modules and their interactions:

The Library Management System consists of several modules that work together to provide a comprehensive solution for managing library operations. Here is an overview of the modules and their interactions:

- Cataloging Module: This module is responsible for managing the information about the library's collection of books, including adding new books, updating book information, and assigning unique identifiers to each book. It also manages the classification and indexing of library materials.

- Circulation Module: This module manages the borrowing and returning of books, including issuing library cards to patrons, recording due dates, sending overdue notices, and managing fines and fees.

- Reservation Module: This module allows patrons to reserve books that are currently checked out and be notified when the book is available.

- Search Module: This module allows patrons to search for books in the library catalog and locate the book on the shelves.

- User Management Module: This module manages user accounts, including creating and deleting accounts, updating user information, and setting user permissions.

### The modules interact with each other in various ways. For example:

- The Cataloging Module interacts with the Circulation Module to update book availability information when books are checked out or returned.
- The Circulation Module interacts with the User Management Module to verify patron information and manage library card accounts.
- The Reservation Module interacts with the Circulation Module to manage book availability and notify patrons when their reserved books are available.
- The Search Module interacts with the Cataloging Module to retrieve book information and with the Circulation Module to display book availability information.

Overall, the modules of the Library Management System work together to provide an integrated solution for managing all aspects of library operations.

## CATALOGING MODULE:

This process design outlines the steps involved in adding new books, updating book information, and assigning unique identifiers to each book in a Library Management System.

1. ### ADD Books:
    - Collect Book Information: The librarian collects all necessary information about the book, including the title, author, publisher, publication date, edition, and ISBN.
    - Create Book Record: Using the collected information, the librarian creates a new record for the book in the Library Management System, including assigning a unique identifier to the book.
    - Enter Book Information: The librarian enters all the book information into the system, including the title, author, publisher, publication date, edition, and ISBN. The system validates the information to ensure that all required fields are filled and that the ISBN is in the correct format.
    - Add Copies: If the library has multiple copies of the book, the librarian adds the number of copies to the book record in the system.
    - Assign Location: The librarian assigns a physical location for the book in the library, such as a shelf number, based on the library's classification and indexing system.
    - Review and Approve: The librarian review's the information entered into the system and approves the record if everything is accurate and complete.

2. ### Update Book Information:
    - If any changes are made to the book information, such as a new edition or updated author information, the librarian updates the book record in the system.

3. ### Remove Books:
    - If a book is lost, damaged, or withdrawn from the collection, the librarian removes the book record from the system.

## Sample Request and Response of the Operation in Catalog module:

### Add New Book:

API Type: POST
URL: http://loaclhost:10000/api/v1/catalog/addbook

Request Payload:

```
{
 "title": "The Alchemist",
 "author": "Paulo Coelho",
 "isbn": "0-06-250217-4",
 "publisher": "HarperTorch",
 "publicationYear": "1988",
 "edition": "1st Edition",
 "genre": "fantasy",
 "language": "English",
 "description": "The Alchemist is the magical story of Santiago, an Andalusian shepherd boy
who yearns to travel in search of a worldly treasure as extravagant as any ever found"
}
```

Response Payload:

```
{
  "BookID": "B1003",
  "ResponseCode": "200",
  "ResponseMsg": "Copy of the book added to the catalog"
}
```

Client Input Validation:

| Error Code | Example | Descriptions | User Info. Message |
|---|---|---|---|
| 101 | `Title=""` | Empty String Validtion | 'Title cannot be empty' |
| 102 | `Author=""` | Empty String Validtion | 'author filed cannot be empty' |
| 103 | `Isbn=""` | Empty String Validtion | 'ISBN cannot be empty' |

## Update Book Information:

API Type: POST
URL: http://loaclhost:10000/api/v1/catalog/updatebook

### Request Payload:

```
{
 "BookID": "B1003",
 "title": "The Alchemist",
 "author": "Paulo Coelho",
 "isbn": "0-06-250217-4",
 "publisher": "HarperTorch",
 "publicationYear": "1988",
 "edition": "1st Edition",
 "genre": "fantasy",
 "language": "English",
 "description": "The Alchemist is the magical story of Santiago, an Andalusian shepherd boy
who yearns to travel in search of a worldly treasure as extravagant as any ever found"
}
```

### Response Payload:

```
{
  "BookID": "B1003",
  "ResponseCode": "200",
  "ResponseMsg": "Book Details Updated Successfully"
}
```

### Client Input Validation:

| **Error Code** | **Example** | **Descriptions** | **User Info. Message** |
|---|---|---|---|
| 101 | BookID=”” | Empty String Validtion | 'BookID cannot be null' |

## Remove Books:

API Type: POST
URL:hhtp://loaclhost:10000/api/v1/catalog/removebook

### Request Payload:

```
{
  "BookID": "B1003"
}
```

### Response Payload:

```
{
   "BookID": "B1003",
   "ResponseCode": "200",
   "ResponseMsg": "Book Removed"
}
```

### Client Input Validation:

| Error Code | Example | Descriptions | User Info. Message |
|---|---|---|---|
| 101 | BookID=”” | Empty String Validtion | 'BookID cannot be null' |

## HTTP Status Codes:

| Http Code | Descriptions | Messages |
|---|---|---|
| 200 | OK | Success |
| 400 | Bad Request | Invalid argument |
| 401 | Unauthorized | Invalid access token |
| 500 | Internal Server Error | Common Error |

## CIRCULATION MODULE:

This process design outlines the steps involved in managing the borrowing and returning of books, including issuing library cards to patrons, recording due dates, sending overdue notices, Check In Books in a Library Management System.

process design for the Circulation module in a Library Management System:

### Register Patrons: The librarian registers patrons and issues library cards with unique identifiers to track their borrowing history.

### Check Out Books: When a patron wants to borrow a book, the librarian scans the patron's library card and the book's barcode to check out the book.

Record Due Date: The system records the due date for each borrowed book based on the library's borrowing policies and the patron's borrowing history.

Renew Books: If a patron wants to renew a borrowed book, the librarian extends the due date in the system, provided there are no holds on the book.

### Send Overdue Notices: The system automatically sends overdue notices to patrons who have not returned their borrowed books by the due date.

### Check In Books: When a patron returns a book, the librarian scans the book's barcode to check it in and update the book's status in the system.

## Sample Request and Response of the Operation in Circulation Module:

### Register Patrons:

API Type: POST
URL:http://localhost:10020/api/v1/circulation/register

### Request Payload:

```
{
  "patron_name": "Santosh Kumar",
  "address": "Hyderabad",
  "phone_number": "8074217636",
  "email": "stalgeri@eniquesolutions.com"
}
```

### Response Payload:

```
{
  "library_card_number": "LIB100021",
  "expiration_date": "2024-03-20",
  "patron_name": "Saranaya Kota",
  "status": "Active"
 }
```

### Client Input Validation:

| **Error Code** | **Example** | **Descriptions** | **User Info. Message** |
|---|---|---|---|
| 101 | `"patron_name": ""` | Empty String Validtion | 'Patron name should not be empty' |
| 102 | `phone_number=””` | Input String Validtion | 'Phone Number should only be in digits and of length 10' |
| 103 | `email=””` | Input String Validtion | 'Email Id is not in a valid format' |
| 104 | `email=”example@gmail.com”` | Input String Validtion | 'Email ID already Registered' |

### Check Out Books:

API Type: POST
URL:hhttp://localhost:10020/api/v1/circulation/checkout

## Request Payload:

```
{
  "library_card_number": "LIB100021",
  "book_id": "B1007",
  "Renew": {
    "transactionID": "",
    "isRenew": "false"

  }
}
```

## Response Payload:

```
{
  "transaction_id": "T10007",
  "book_id": "B1007",
  "due_date": "2023-04-03",
  "status": "Checked Out"
}
```

## Client Input Validation:

| **Error Code** | **Example** | **Descriptions** | **User Info. Message** |
|---|---|---|---|
| 101 | `"library_card_number": ""` | Empty String Validtion | 'Library Card Number can not be null' |
| 102 | book_id=”” | Empty String Validtion | 'BookID can not be null' |
| 103 | `"library_card_number": "LIB100019"` | Input String Validtion | 'Library card Number is not valid… please Try agian.' |
| 104 | `"transactionID": "T10006"` | Input String Validtion | 'Library Card num is not valid for the given transaction' |

## Check In Books:

API Type: POST
URL:hhttp://localhost:10020/api/v1/circulation/checkin

## Request Payload:

```
{
  "transaction_id": "T10003",
  "book_id": "B1005"
}
```

## Response Payload:

```
{
  "transaction_id": "T10003",
  "book_id": "B1005",
  "return_date": "2023-03-20",
  "status": "Returned"
}
```

## Client Input Validation:

| Error Code | Example | Descriptions | User Info. Message |
|---|---|---|---|
| 101 | `"transaction_id": ""` | Empty String Validtion | 'Transaction ID cannot be null' |
| 102 | `book_id=""` | Empty String Validtion | 'BookID can not be null' |
| 103 | `"transaction_id": "T10007"` | Input String Validtion | 'Transaction ID is not valid' |
| 104 | `book_id=" B1003"` | Input String Validtion | 'BookID is not valid' |

## Sending Overdue Notices:

API Type: Schedular(1 week)

- Collect the information of all the overdue items from “LIB_BORROWING” table.
- Collect the information of the patrons who have overdue from “LIB_CARD” table.

- Send notice mail to each patron.
- Store the notice details on “LIB_NOTICEINFO” table

## Sample XML required for the process:

### 1. OverdueInfo

```
{
  "library_card_number": "123456789",
  "book_id": "B0001",
  "due_date": "2023-04-01",
  "overdue_days": 5,
  "email": "jsmith@email.com"
}
```

### 2. Notice

```
{
  "notice_id": "N0001",
  "library_card_number": "123456789",
  "book_id": "B0001",
  "notice_date": "2023-04-06",
  "status": "sent"
}
```

## RESERVATION MODULE:

This process design outlines the steps involved in allowing patrons to reserve books that are currently checked out and be notified when the book is available in a Library Management System.

process design for the Book Reservation module in a Library Management System:

*Reserve book*: Patrons search the library catalog to find books they want to reserve, and the system displays the availability of the book.

*Place a Hold*: If a book is checked out, patrons can place a hold on the book in the system by clicking a button or selecting a menu option.

*Verify Patron Information*: The system verifies that the patron has a valid library card and that their contact information is up to date.

*Record Reservation*: The system records the patron's reservation in the system, including the book title, the patron's contact information, and the date the hold was placed.

*Notify Patrons*: When a reserved book is returned, the system automatically sends notifications to the patrons in the reservation queue, informing them that the book is available for pickup.

## Sample Request and Response payload for Reservation Module:

### Reserve Book:

API Type: POST
URL:http://localhost:10020/api/v1/reservation/reserve

## Request Payload:

```
{
    "cardNo": "LIB100002",
    "book_id": "B1005",
    "reservation_date": "2023-04-01"
}
```

## Response Payload:

```
{
    "reserveBookRes": {
        "reservation_id": "R11002",
        "cardNo": "LIB100002",
        "book_id": "B1005",
        "reservation_date": "2023-04-01",
        "status": "active",
        "email": "kpolagoni@eniquesolutions.com"
    }
}
```

## Client Input Validation:

| Error Code | Example | Descriptions | User Info. Message |
|---|---|---|---|
| 101 | `"cardNo": ""` | Empty String Validtion | 'Card num cannot be null' |
| 102 | `"book_id": ""` | Empty String Validtion | 'Book ID cannot be null' |
| 103 | `" reservation_date": ""` | Empty String Validtion | 'Reservation Date cannot be null' |
| 104 | `"cardNo": "LIB100006"` | Input String Validtion | 'No Record for the given Card number' |
| 105 | `"cardNo": "LIB100006"` | Input String Validtion | 'The Library card is not active' |

## SEARCH AND RETRIEVAL MODULE:

This process design outlines the steps involved in allowing patrons to search for books in the library catalog and locate the book on the shelves in a Library Management System.

process design for the Book Search module in a Library Management System:

### Search Catalog:

- Patrons search the library catalog by entering keywords, author name, title, subject, or other search criteria.
- Display Results: The system displays a list of books matching the search criteria, along with information about the book such as title, author, publisher, publication date, and availability status.
- Filter Results: Patrons can filter and sort the search results based on various criteria such as author, publication year, subject, and availability status.

## Check Book Availability:

- Patrons can check the availability of a book in the library by viewing its status, location, call number.
- Locate Book: If a book is available, patrons can use the call number to locate the book on the shelves.
- Manage Reservations: If a book is checked out, patrons can place a hold on the book and be notified when the book is available.

## Request and Response Payload for Search and Retrieval Module:

### Search Catalog:

API Type: GET
URL:http://localhost:10040/api/v1/search

### Request Payload:

**QUERY PARAMETERS**

| Parameter Name | Optional /Not Optional | Description |
|---|---|---|
| Search_Query | Not Optional | name of the title or author to search |
| Search_Type | Not Optional | field to search for |

| | | |
|---|---|---|
| sort_by | Optional | TITLE, AUTHOR (should not accept other than these values, should show the Error msg) |
| sort_order | Optional | Asc/Desc (should not accept other than these values, should show the Error msg) |
| page_number | Optional | int Page Number - Default: 1 (user input), if user input is given, then the value should be validated.<br><br>Page>0 |
| page_size | Optional | int Rows Per page - Default: 100 (configurable), if user input is given, then the value should be validated,<br><br>1. Limit>0<br><br>Limit< (configurable variable) |

## Response Payload:

```
{
    "BookDetails": [
        {
            "book_id": "B1008",
            "title": "The Lost Symbol",
            "author": "Dan Brown",
            "isbn": "978-0-385-50422-5",
            "publisher": "Doubleday",
            "publish_year": "2009",
            "edition": "1st Edition",
            "category": "fiction",
            "language": "English",
            "description": "The Lost Symbol is a 2009 novel written by American
writer Dan Brown. It is a thriller set in Washington, D.C., after the events of The
Da Vinci Code, and relies on Freemasonry for both its recurring theme and its major
characters.",
            "status": "available"
        },
        {
            "book_id": "B1009",
            "title": "Angels & Demons",
            "author": "Dan Brown",
            "isbn": "978-0-671-02735-2",
            "publisher": "Pocket Books",
            "publish_year": "2000",
            "edition": "1st Edition",
            "category": "fiction",
            "language": "English",
            "description": "Leonardo Vetra, one of CERN's top physicists who have
discovered how to create antimatter, is murdered, his chest branded with an
ambigram of the word 'Illuminati', an ancient anti-religious organization thought
extinct.",
            "status": "available"
        }
    ],
    "control": {
        "sort_by": "TITLE",
        "sort_order": "DESC",
        "page_number": "1",
        "page_size": "10",
        "nextpageURL": null,
        "ErrorCode": {
            "nil": true
        },
        "ErrorMsg": null
    }
}
```

## Client Input Validation:

| Error Code | Example | Descriptions | User Info. Message |
|---|---|---|---|
| 101 | Search_Query= | Empty String Validation | 'Search Query cannot be null' |
| 102 | Search_Type= | Empty String Validation | 'Search Type cannot be null' |
| 103 | sort_by=sdefe | Input String Validation | "Sort by can only accept 'TITLE' and 'AUTHOR' as value" |
| 104 | sort_order=sdfs | Input String Validation | "SortOrder can only accept 'ASC' and 'DESC' as value"' |
| 105 | page_number=0 | Input String Validation | 'page number should be greater then 0' |
| 106 | page_number=fss | Input String Validation | 'page number should only contain numbers' |
| 107 | page_size=0 | Input String Validation | 'Page size sould be greater then 0' |

|  |  |  |  |
|---|---|---|---|
| 108 | page_size=102 | Input String Validation | 'page size should be less then or equal to 100' |

## Check Book Availability:

API Type: GET
URL: http://localhost:10040/api/v1/checkAvail

## Request Payload:

**QUERY PARAMETERS**

| **Parameter Name** | **Optional /Not Optional** | **Description** |
|---|---|---|
| book_id | Not Optional | Varchar |

## Response Payload:

```
{
    "bookAvailDetails": {
        "book_id": "B1004",
        "title": "Ikigai",
        "author": "Hector Garcia, Francesc Miralles",
        "availability": "available"
    }
}
```

If the book is not available, the response payload would be

```
{
    "bookAvailDetails": {
        "book_id": "B1005",
        "title": "The Alchemist",
        "author": "Paulo Coelho",
        "availability": "not available",
        "return_date": "2023-03-31"
    }
```

```
        }
```

If error is present

```
        {
            "control": {
                "ErrorCode": "401",
                "ErrorMsg": "No Record for the give BookID"
            }
        }
```

## Client Input Validation:

| **Error Code** | **Example** | **Descriptions** | **User Info. Message** |
|---|---|---|---|
| 101 | book_id= | Empty String Validation | 'BookId cannot be null' |
| 401 | book_id=1001 | Input String Validation | "No Record for the given BookID" |

## USER MANAGEMENT MODULE:

This process design outlines the steps involved in managing user accounts in a Library Management System.

process design for the User Management Module in a Library Management System:

### Create Account:

- A librarian creates a new user account by entering the user's information, such as name, address, email, and phone number, into the system.
- Verify Account: The system verifies the user's information and checks for any duplicate or conflicting records.
- Assign User Roles: The librarian assigns a role or permission level to the user based on their job function or responsibilities, such as librarian, student, faculty.
- The system sets permissions for each user based on their role, such as the ability to check out books, add new books.

### Manage User Information:

- The system manages user information, including updating user profiles, resetting passwords, and deleting accounts.

### Login:

- The System Verifies the user login information, if success then logs in the user otherwise sends the error message.

## Request and Response payload for User Management Module:

### Create User:

API Type: POST
URL:http://localhost:10050/api/v1/userManagement/newUser

### Request Payload:

```
{
    "name":"santosh kumar",
    "email":"stalgeri@eniquesoltions.com",
    "password":"Santosh@1",
    "role":"admin"
}
```

### Response Payload:

```
{
    "data": {
        "user_id": "U10001",
        "name": "santosh kumar",
        "email": "stalgeri@eniquesoltions.com",
        "role": "admin"
    },
    "control": {
        "nil": true
    }
}
```

### Client Input Validation:

| Error Code | Example | Descriptions | User Info. Message |
|---|---|---|---|
| 101 | "name":”” | Empty String Validtion | "name cannot be null" |
| 102 | "email":"satosh@” | Input String Validtion | 'Email is not valid" |
| 103 | "password":”santosh” | Input String Validtion | "Password is not valid .....password should contains at least one lowercase letter, one uppercase letter, and one digit,and one special character(@, $, !, %, *, ?, |

|  |  |  |  |
|---|---|---|---|
|  |  |  | &amp;), and is at least 8 characters long" |
| 104 | "role":”” | Empty String Validtion | "role cannot be null"' |
| 105 | “email”:santosh@test.com | Input String Validtion | 'email is already Registered' |

## Update User Info:

API Type: POST
URL: http://localhost:10050/api/v1/userManagement/updateUser

## Request Payload:

```
{
    "user_id":"U10001",
    "name":"",
    "email":"",
    "password":"Santosh@2",
    "role":""
}
```

## Response Payload:

```
{
    "data": {
        "user_id": "U10001",
        "name": "santosh kumar",
        "email": "stalgeri@eniquesoltions.com",
        "role": "admin"
    },
    "control": {
        "nil": true
    }
}
```

If the user is not found:

```
{
    "data": {
        "nil": true
    },
    "control": {
        "errorCode": "401",
        "errorMsg": "No Record for the given UserID"
    }
}
```

Client Input Validation:

| Error Code | Example | Descriptions | User Info. Message |
|---|---|---|---|
| 101 | "user_id":”” | Empty String Validtion | 'userID cannot be null' |
| 102 | "email":"satosh@” | Input String Validtion | 'Email is not valid' |
| 103 | "password":”santosh” | Input String Validtion | "Password is not valid .....password should contains at least one lowercase letter, one uppercase letter, and one digit,and one special character(@, $, !, %, *, ?, &amp;), and is at least 8 characters long" |
| 401 | "user_id":”U10010” | Input String Validtion | 'No Record for the given UserID' |

## Delete User:

API Type: POST
URL: http://localhost:10050/api/v1/userManagement/deleteUser

## Request Payload:

```
{
    "user_id":"U10006"
}
```

## Response Payload:

```
{
    "ResponseMsg": "User deleted successfully",
    "control": {
        "nil": true
    }
}
```

If the user is not found:

```
{
    "control": {
        "errorCode": "401",
        "errorMsg": "User not found."
    }
}
```

## Client Input Validation:

| **Error Code** | **Example** | **Descriptions** | **User Info. Message** |
|---|---|---|---|
| 101 | "user_id":”” | Empty String Validtion | 'userID cannot be null' |
| 401 | "user_id":”ahaa” | Input String Validtion | 'User not found.' |

## Login:

API Type: POST
URL: http://localhost:10060/v1/LMS/login

## Request Payload:

```
{
    "username":"stalgeri@eniquesoltions.com",
    "password":"Santosh@1"
}
```

## Response Payload:

```
{
  "LoginRes":{
    "resCode":"200",
    "resMsg":"Login successfull.."
  }
}
```

### If the user info is incorrect:

```
{
  "LoginRes":{
    "resCode":"103",
    "resMsg":"Password is incorrect.. please try again"
  }
}
```

## Client Input Validation:

| Error Code | Example | Descriptions | User Info. Message |
|---|---|---|---|
| 101 | "username":”” | Empty String Validtion | 'username cannot be null' |
| 102 | "password":"" | Empty String Validtion | 'password cannot be empty' |
| 103 | "password":"sahdj" | Input String Validtion | 'Password is incorrect.. please try again' |
| 401 | "username":”sgaha” | Input String Validtion | 'No Record for the user found' |