**ANUDEEP FOUNDATION**

**Library Management Website**

**Project Report**

|  |  |
| --- | --- |
| **Name** | **Enrollment No** |
| **Dipti Balaso Olekar** | **AF0481797** |

**Under Guidance of**

**Rajshri Thete**

**Index**

|  |  |  |
| --- | --- | --- |
| **Sr.no** | **Topic** | **Page no** |
| 1 | Title of Research | 5 |
| 2 | Certificate | 2 |
| 3 | Abstract | 5 |
| 4 | Introduction | 6-9 |
| 5 | Objective | 9- |
| 6 | Programming Language | 9-18 |
| 7 | Data gathering | 19 |
| 8 | System design /UML Diagrams | 20-22 |
| 9 | Proposed system features | 23-25 |
| 10 | Evalution | 26-27 |
| 11 | Future scope | 28 |
| 12 | conclusion | 29 |
| 13 | Biography | 30 |
| 14 | References | 31 |

**Acknowledgement**

The project **“Library management System”** is the Project work carried out by

|  |  |
| --- | --- |
| **Name** | **Enrollment No** |
| **Dipti Balaso Olekar** | **AF0481797** |

We are thankful to my project guide for guiding me to complete the Project.

His suggestions and valuable information regarding the formation of the Project Report have provided me a lot of help in completing the Project and its related topics.

We are also thankful to my family member and friends who were always there to provide support and moral boost up.

**Abstract**

In the modern era of digitalization, the manual handling of library operations has become outdated, inefficient, and time-consuming. Libraries require robust, scalable, and easy-to-use systems that streamline their workflows and provide real-time access to users and administrators. This project, titled **Library Management Website**, aims to develop a comprehensive web-based application that full fills these needs using modern web development technologies.

The system has been developed using **HTML and CSS** for the front end, which ensures a clean, responsive, and intuitive user interface. The backend is powered by **Python**, with the **Django framework**, which provides rapid development, secure authentication, and a well-organized architecture. This combination of technologies allows the development of a scalable and secure application that meets the dynamic needs of library operations.

The application supports two primary types of users: **Admin** and **Student/User**. The **Admin panel** allows librarians or staff to manage book records, register and monitor users, track book issues and returns, and generate reports. The **User panel** provides functionalities for browsing the library catalog, checking book availability, requesting or issuing books, and viewing borrowing history.

**1. Introduction**

In the digital era, the need for efficient and reliable information systems has grown rapidly. Libraries, being essential components of educational institutions and research organizations, have traditionally relied on manual systems for managing books, users, and transactions. However, with growing user bases and the need for quick access to information, traditional methods are no longer sustainable. This gave rise to the idea of developing an automated and web-based Library Management System.

The Library Management Website is designed as a comprehensive solution to modernize the daily operations of a library. It replaces manual registers with a digital database, allowing librarians and users to interact with the system seamlessly through a web interface. The project uses HTML and CSS for building an intuitive and responsive front end, while Python and Django power the back end, handling all business logic, user authentication, and database interactions.

The system caters to two main user roles:

* Administrator (Librarian) – responsible for adding books, managing inventory, tracking borrowed books, handling user registrations, and monitoring overall library usage.
* Users (Students or Readers) – can register on the portal, search for books, view details, issue books, and track their borrowing history.

The system is built using Django’s Model-View-Template (MVT) architecture, which ensures a clean separation of concerns, easier debugging, and better maintenance. All interactions with the database (such as issuing, returning, or updating book records) are handled through Django's Object Relational Mapping (ORM), eliminating the need for writing complex SQL queries manually.

**2. Objectives**

Key objectives of the project include:

* **Automation of Library Tasks:** To eliminate manual processes and automate routine tasks such as issuing, returning, and reserving books.
* **Efficient Book Management:** To maintain a well-organized database of books, including details like author, title, subject, and availability status.
* **User Account Management:** To create and manage accounts for students, staff, and librarians with appropriate access controls.
* **Accurate Record-Keeping:** To ensure accurate tracking of borrowing history, due dates, and fines for overdue books.
* **Search and Catalog Features:** To allow users to easily search for books by title, author, category, or ISBN through an intuitive interface.
* **Security and Data Integrity:** To provide a secure login system and prevent unauthorized access to the system’s features or data.

3. System analysis

**3.1 PROBLEM DEFINITION**

Libraries are an essential part of educational and professional institutions, serving as a hub for information, research, and self-learning. However, traditional library management systems are typically **manual**, involving **physical registers**, **paper-based book tracking**, and **time-consuming processes** for both staff and users. As libraries grow in size and usage, these manual systems become inefficient, error-prone, and difficult to manage.

Key problems faced in traditional library systems include:

1. **Manual record-keeping** leads to frequent errors, data loss, and redundancy.
2. **Inefficient book tracking**, making it hard to monitor which books are available, issued, or overdue.
3. **Time-consuming processes** for book search, issue, and return, especially during peak usage hours.

**3.2 Preliminary Investigation**

**Purpose**

1. To **replace manual, paper-based operations** with a centralized digital system.
2. To allow **real-time tracking** of book availability, issue status, and return deadlines.
3. To enable **students and staff to access the library remotely** through a web interface.
4. To provide **admin-level controls** for managing book inventory, users, and reports.
5. To improve **data accuracy, security, and reporting** through automation and validation.

**Benefits**

**🔹 For Librarians (Admins):**

* **Easy Management:** Add, update, delete, and categorize books from a dashboard.
* **Quick User Access:** Register and manage users effortlessly.
* **Inventory Tracking:** Real-time view of issued/returned books and stock levels.

**🔹 For Users (Students/Readers):**

* **Online Access:** View book availability and request issues from anywhere.
* **Account Management:** Track borrowing history and return dates via their own login.
* **Faster Transactions:** No need to stand in queues for manual book issuance.
* **Search Functionality:** Quickly find books by title, author, category, or availability.
* **Secure Login:** Role-based authentication keeps data secure.

**Proposed System**

The Proposed Library Management Website is a comprehensive digital solution developed to automate and streamline library operations such as book inventory management, user registration, book issuance, returns, and record-keeping. Built using HTML, CSS, Python, and the Django web framework, the system delivers a responsive, scalable, and secure platform accessible via any modern web browser.

1. This system serves two main user groups — Admins (Librarians) and Users (Students or Readers) — each having specific roles and access permissions.  
     
   **User Registration & Login**
   * New users can register through the web interface.
   * Django handles secure login and authentication using hashed passwords.
   * Role-based redirection: Admins and Users see different dashboards.
2. **Admin Panel Functionalities**
   * **Book Management**: Add, update, delete books with details like title, author, publication, quantity, and availability.
   * **User Management**: View registered users, delete or deactivate users if needed.
   * **Issue & Return Management**: Track which user has which book, manage due dates.
3. **User Panel Functionalities**
   * **Book Browsing**: Search and filter books by category, title, author, or availability.
   * **Issue Requests**: Users can request to borrow available books.
   * **Borrowing History**: Users can view their current and past borrowed books.

**3.3 Feasibility Study**

Before initiating the development of the Library Management Website, a thorough feasibility study was conducted to ensure that the project is viable from technical, economic, operational, and legal perspectives. The goal of this study was to determine whether the proposed system could be effectively implemented within the given resources, time frame, and institutional constraints, while solving the existing problems in traditional library management.

From a technical perspective, the proposed system is entirely feasible due to the availability of open-source technologies such as Python, Django, HTML, CSS, and MySQL. These technologies are well-documented, widely supported by the developer community, and compatible with most operating systems and web browsers. The development team has access to adequate tools, development environments like Visual Studio Code, and testing frameworks, which makes the implementation practical and sustainable.

Economically, the project is also considered feasible as it does not require heavy investment in hardware or licensed software. The use of open-source platforms significantly reduces costs associated with development and deployment. Institutions or colleges already possessing basic IT infrastructure—such as computers and internet connectivity—will incur minimal additional expenses. The long-term savings in manpower, time, and error reduction in managing books and users also add to the financial feasibility of the project.

Operational feasibility is assured due to the intuitive nature of the system’s design, which ensures that both administrators and users can easily adapt to its interface without requiring extensive training. The administrative interface is designed to be simple, direct, and effective for managing large volumes of data, while users are provided with an accessible platform to search, issue, and return books with minimal clicks. The system introduces clear improvements over the manual system, such as faster transactions, better record-keeping, and enhanced accessibility.

3.4 Project Planning

Purpose of Project Planning

The purpose of project planning for a Library Management System is to provide a clear and structured roadmap for the successful development and implementation of the system. It begins with defining the objectives and scope of the project, ensuring that the system will meet the needs of its users, such as managing book inventories, member registrations, and issuing and returning books. Project planning helps in identifying and documenting all necessary functional and non-functional requirements, which serve as a foundation for system design and development. It also involves allocating the appropriate resources, including personnel, time, and budget, to avoid any unexpected shortages during the project. Additionally, it plays a vital role in risk management by identifying potential challenges and planning strategies to mitigate them.  
**1. Define Objectives and Scope**

* Clearly outline what the system is intended to achieve (e.g., managing books, members, borrowing/returning processes).
* Establish the boundaries of the project to avoid scope creep.

**2. Identify Requirements**

* Gather functional and non-functional requirements from stakeholders (librarians, students, administrators).
* Ensure that user needs such as search capabilities, fine management, and user registration are documented.

**3. Resource Allocation**

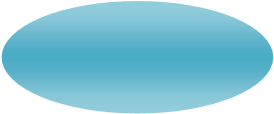
* Determine the human resources (developers, testers), tools, and technologies needed.
* Estimate budget and time requirements.

Phases Covered in the Plan

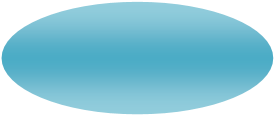
The planning is divided into different phases to ensure smooth execution:

1. **Preliminary Investigation** – Understanding the project scope and objectives.
2. **System Analysis** – Identifying challenges, gathering requirements, and defining solutions.
3. **System Design** – Structuring modules, database design, and UI development.
4. **Coding** – Developing the portal using Python (Django) and integrating sentiment analysis.
5. **Security** – Implementing authentication, data encryption, and user privacy measures.
6. **Testing** – Performing unit testing, integration testing, and user acceptance testing.
7. **Implementation** – Deploying the final system and ensuring smooth operation.

3.5 Project Scheduling



**Stop**



**Start**



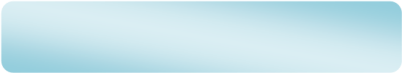
Preliminary



System Analysis



System Design



Coding



Testing



Security



Implementation

3.6 Software Requirement Specification (SRS)

The **Software Requirement Specification (SRS)** document defines the complete set of requirements for the development of a **Library Management System (LMS)**. It serves as a blueprint for both the development team and stakeholders.  
The purpose of this SRS is to outline the functional and non-functional requirements for a Library Management System. The system will manage the cataloging, circulation, and administrative tasks of a library, ensuring smooth operations and efficient user access to library resources.  
  
  
System Overview

The Library Management System (LMS) is designed to streamline the management of library resources, user interactions, and circulation processes. The system is organized into three main modules:

1. **User Module** – Allows users (students, staff, or faculty) to search for books, check availability, reserve books, borrow and return books, and view borrowing history.
2. **Admin Module** – Provides full access to manage books, users, issue/return transactions, fines, and generate reports. The admin can also manage system settings and oversee sub-admin activities.

Software & Hardware Requirements

Software Requirements

* **Frontend**: HTML, CSS, JS
* **Backend**: Python (Django) for data handling and security
* **Database**: MySQL for storing user details
* **Web Server**: Apache for hosting

Hardware Requirements

* **Processor**: Intel i5 or higher
* **RAM**: Minimum 8GB
* **Storage**: At least 100GB for database and media files
* **Connectivity**: Internet access for real-time updates

3.7 Functional Requirements

1. User Module

Users (students, faculty, or staff) can:  
• Search for books by title, author, subject, or ISBN.  
• Browse books by categories such as Fiction, Science, History, Technology, etc.  
• View book details including availability status.  
• Reserve books currently issued to others.  
  
  
  
  
2. Admin Module

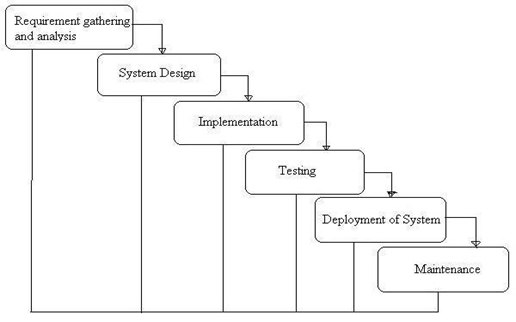
• **Secure Login System** – Access the admin dashboard via authentication with role-based access.  
• **Dashboard Management** – View system statistics and manage book inventories, user accounts  
• **User Management** – Add, update, or delete user profiles and assign roles.  
• **Book Management** – Add, edit, delete, and categorize books.  
• **Issue/Return Management** – Monitor and manage book circulation records.  
• **Fine Management** – Set rules for overdue fines and monitor fine collection.

3.8 Software Engineering Paradigm

The development of the Library Management System follows the **Waterfall Model**, a structured software engineering paradigthat breaks the project into clear sequential phases—Requirement Analysis, System Design, Implementation, Testing, Deployment, and Maintenance. This model ensures thorough documentation and planning at each stage, which is ideal for systems like LMS that have well-defined, predictable functionalities.  
  
Development Model: Adapted Waterfall Model

The development of the Library Management System follows an **Adapted Waterfall Model**, which maintains the structured, phase-by-phase approach of the traditional Waterfall Model while incorporating feedback loops to allow iterative refinement. This hybrid strategy ensures both discipline and flexibility during the development process.  
  
  
  
Key Adaptations in the Waterfall Model:

1. Structured Phase Progression – The project advances through clearly defined stages such as Requirements Gathering, System Design, Implementation, Testing, Deployment, and Maintenance, ensuring systematic progress and accountability.
2. Iterative Refinements – Feedback mechanisms are embedded between phases—especially between testing and development—to allow revisions based on user testing, performance issues, or stakeholder feedback.
3. Defined Milestones – Each phase has specific deliverables and checkpoints that must be completed and approved before progressing, which promotes quality control and minimizes errors.
4. Flexible Adjustments – While the model remains largely sequential, overlapping between certain stages (e.g., early testing during late development) is permitted when needed to improve project efficiency or meet deadlines.



Phases of Development

1. **Requirement Analysis & System Study**
   * Identifying project goals, challenges, and functional specifications.
   * Gathering stakeholder requirements and defining core functionalities.
2. **System Design**
   * Structuring the **database, modules, and architecture**.
   * Designing **user interfaces** for optimal accessibility.
3. **Implementation (Coding)**
   * Backend development using **Python (Django)**.
   * Frontend design using **HTML, CSS, JS**.
   * Database integration with **MySQL**.
4. **Testing & Debugging**
   * Unit testing, integration testing, and usability checks.
   * Debugging for performance improvements.
5. **Deployment & Maintenance**
   * Hosting on a scalable environment.
   * Continuous updates for feature enhancements.

3.9 Data Flow Diagram:

A Data Flow Diagram (DFD) for an LMS is a powerful graphical tool that illustrates the movement and transformation of information within the system. It offers a clear, visual representation of how data flows between various components of the LMS, external entities, and internal processes. Unlike a flowchart, a DFD focuses purely on the flow of data, not the sequence of operations or logical decisions.  
The primary objective of a DFD for an LMS is to define the scope and boundaries of the system from a data perspective. It serves as an invaluable communication tool between system analysts, developers, and all stakeholders involved in the LMS project, providing a shared understanding of the system's data interactions. It acts as a fundamental starting point for understanding existing systems or for designing new ones.  
  
**The following observations about DFDs are essential:**

1. All names should be unique. This makes it easier to refer to elements in the DFD.
2. Remember that DFD is not a flow chart. Arrows is a flow chart that represents the order of events; arrows in DFD represents flowing data. A DFD does not involve any order of events.
3. Suppress logical decisions. If we ever have the urge to draw a diamond-shaped box in a DFD, suppress that urge! A diamond-shaped box is used in flow charts to represents decision points with multiple exists paths of which the only one is taken. This implies an ordering of events, which makes no sense in a DFD.
4. Do not become bogged down with details. Defer error conditions and error handling until the end of the analysis.

Standard symbols for DFDs are derived from the electric circuit diagram analysis and are shown in fig:

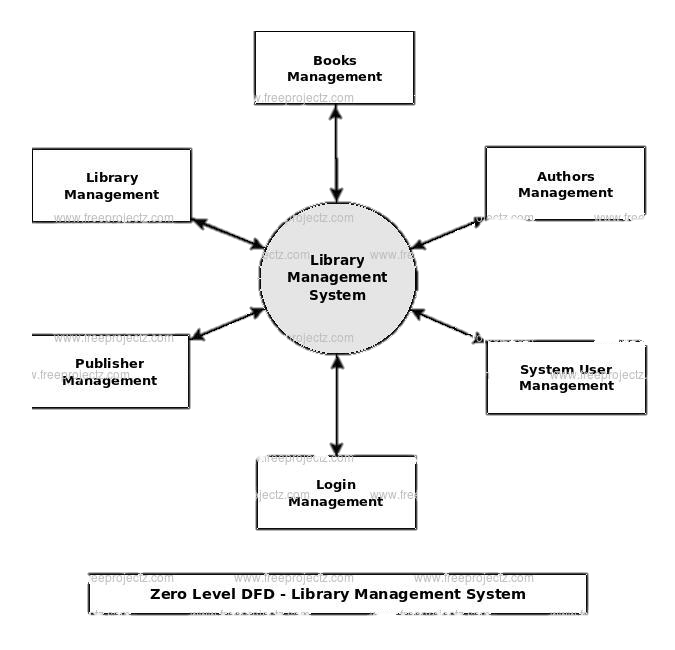
******

**Circle:** A circle (bubble) shows a process that transforms data inputs into data outputs.

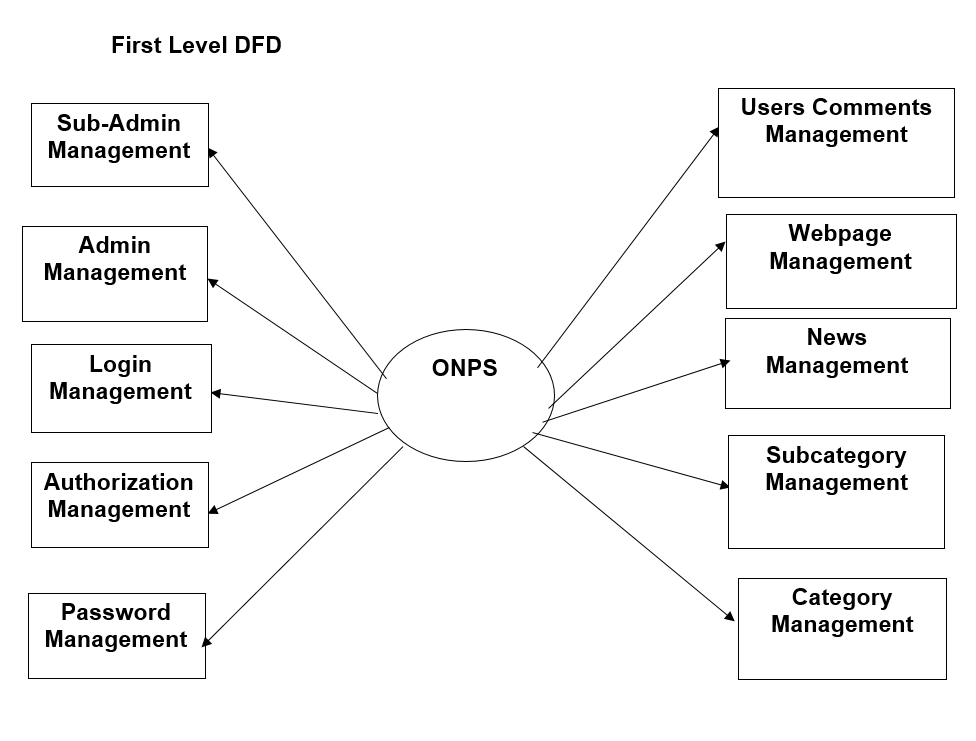
**Data Flow:** A curved line shows the flow of data into or out of a process or data store.

**Data Store:** A set of parallel lines shows a place for the collection of data items. A data store indicates that the data is stored which can be used at a later stage or by the other processes in a different order. The data store can have an element or group of elements.

**Source or Sink**: Source or Sink is an external entity and acts as a source of system inputs or sink of system outputs.

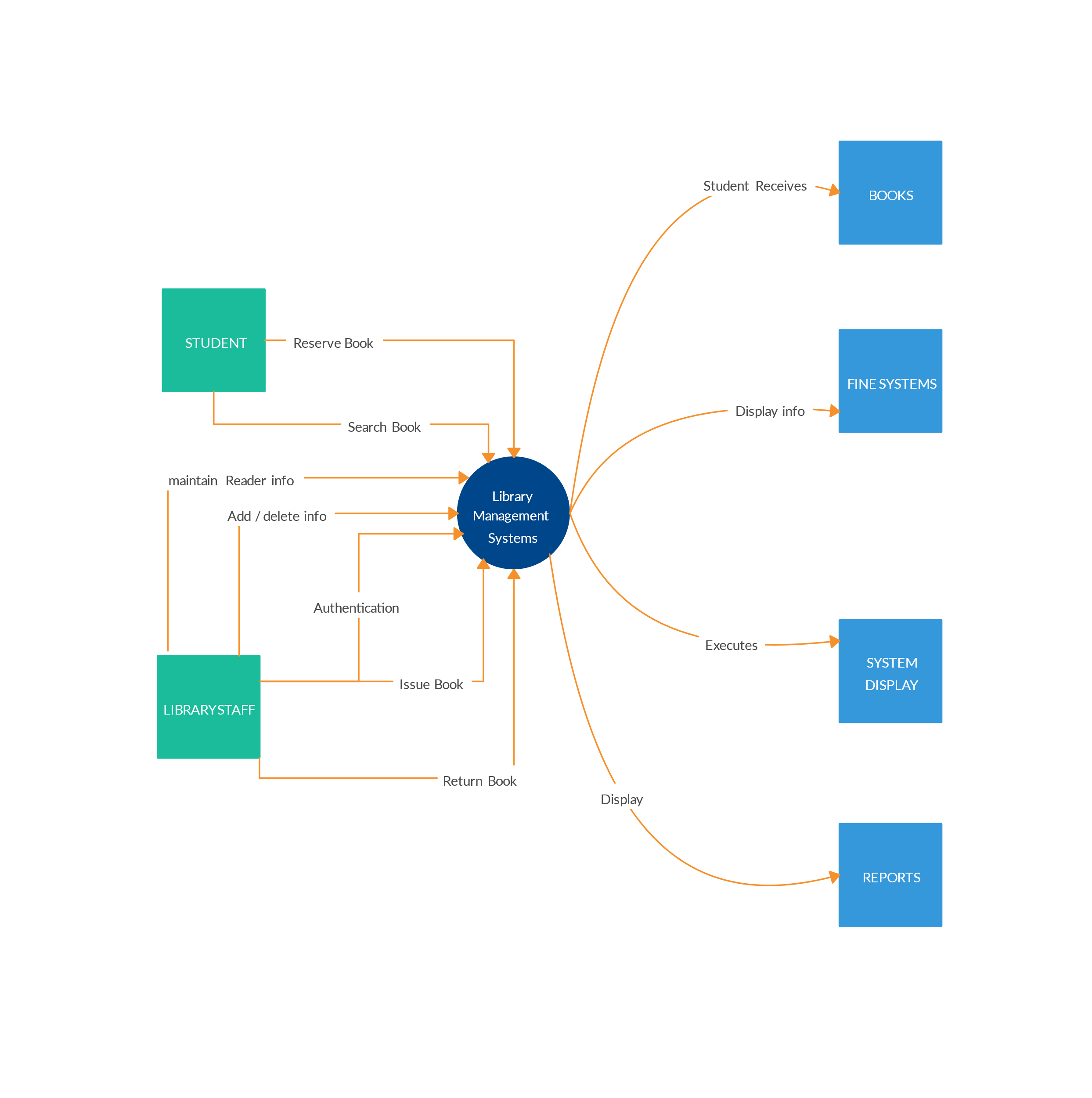


The Zero Level Data Flow Diagram for the Library Management System (LMS) portal illustrates the central system as a single process interacting with several external entities. **Users** (general public and website visitors) exchange data such as search queries, news views, comments, and login/registration information with the ONPS, receiving in return webpages, news articles, search results, comment displays, and login status. **Sub-Admins** provide content management data (news, categories, subcategories, webpages, comments) and user requests. **Admins** interact with the by supplying system configuration data, user management details, security settings, and authorization rules.



LMS

The **First-Level DFD** of ONPS shows how the system handles key functions like login, admin, sub-admin, authorization, and password management, and connects them to content-related modules such as news, category, subcategory, webpage, and user comments management for smooth portal operations.

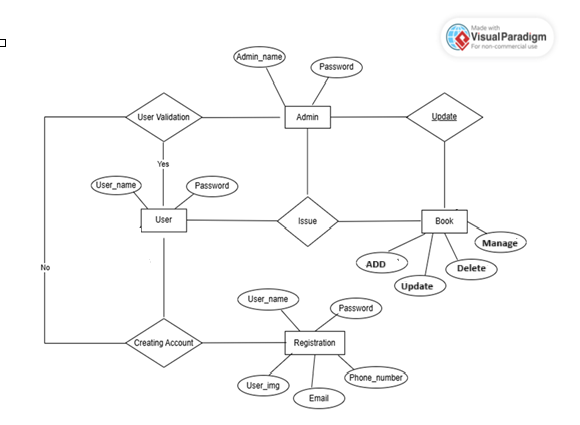


The Library Management System (LMS) encompasses a sophisticated internal workflow for its administrative users, represented by the Second-Level Data Flow Diagram (DFD) focusing on the Admin and Sub-Admin functionalities. This detailed view begins with **Admin Authentication**, where librarians (acting as Admins or Sub-Admins) initiate a login process by submitting their credentials. This process interacts with the **Admin Credentials & Roles** data store to verify the login information and retrieve the associated administrative role.

Upon successful authentication, the system proceeds to **Access Control & Role Management**. Here, the system determines the specific permissions granted to the logged-in administrator based on their role

ER diagram

In a Library Management System, the core entities of **Users** and **Admins** are crucial for defining system access and functionality. The **User** entity is characterized by attributes such as a unique UserID serving as the primary key, Username, Password, Email, and RegistrationDate, facilitating individual user identification and account management. Their relationships within the system primarily involve performing actions like searching for available books, initiating borrowing requests for desired titles, and submitting books for return. Conversely, the **Admin** entity possesses attributes including a unique AdminID as its primary key, Username, Password, Email, and Role/Privileges to define their administrative scope.



4. system design

4.1. Module

Users can:  
• Search and browse books by category (e.g., Fiction, Non-Fiction, Science, History, Technology).  
• View book details including title, author, availability status, and category.  
• Register and log in to their account.  
  
2. Admin Module

Admins have full control over the platform and can:  
• Secure Login System – Authenticate via a secure login to access the admin dashboard.  
• Dashboard Management – Manage book categories, add/edit/delete book records, and monitor system statistics.  
• User Management – Approve user registrations, suspend accounts, and view user borrowing history.  
• Issue and Return Management – Track book issues and returns, update availability, and calculate fines.

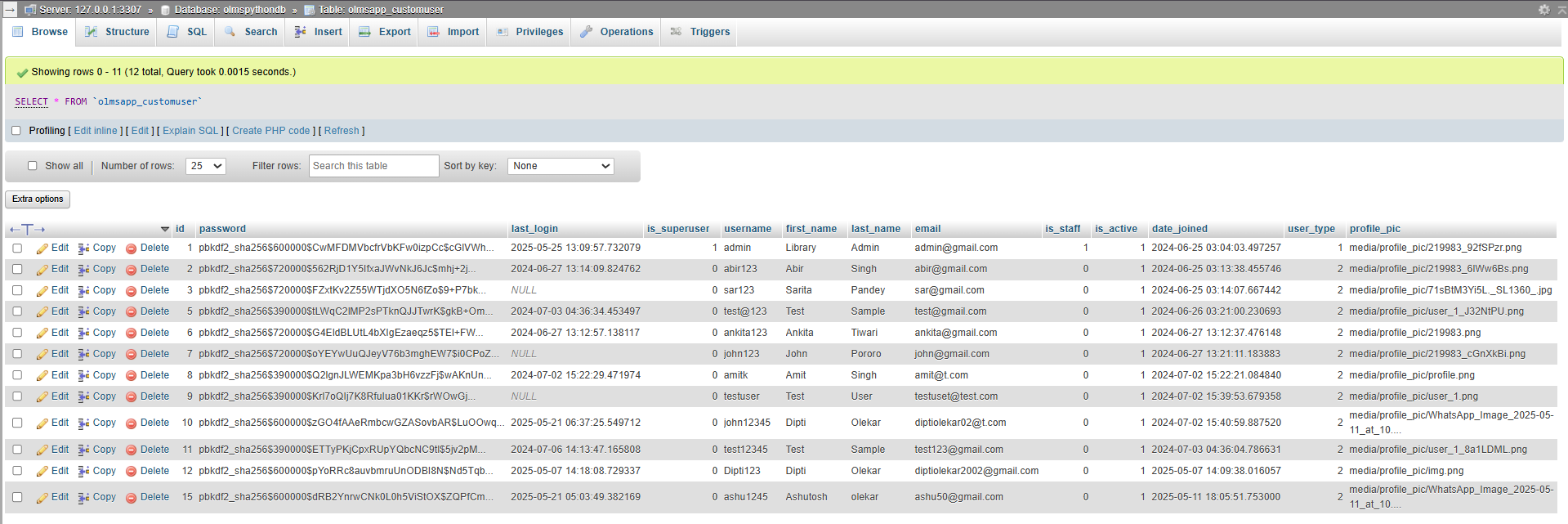
• Manage book inventory (add, update, or remove books) within assigned categories or sections.  
• Handle book issue and return operations.

4.2 DATA STRUCTURE OF ALL MODULES:

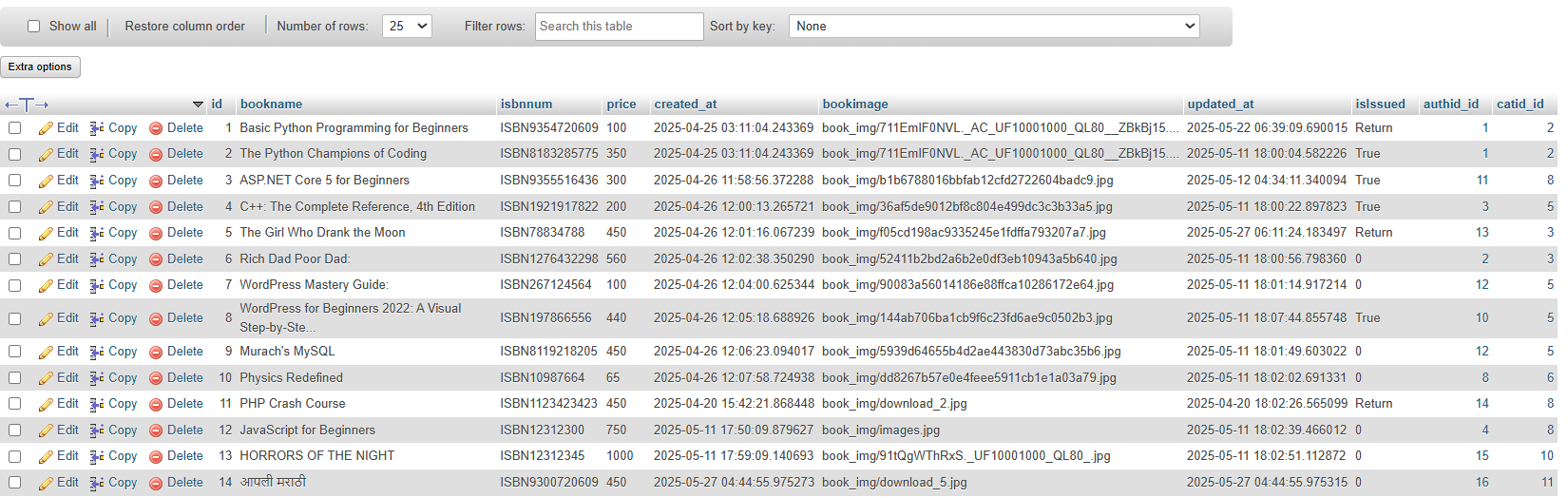
We have designed and organized a comprehensive database for the **Library Management System**. This database can be accessed directly or sequentially by registered users such as librarians, staff, and members. It includes structured elements such as files, records, fields, and characters to ensure smooth operations and easy data retrieval.

**Customized Tables Details**

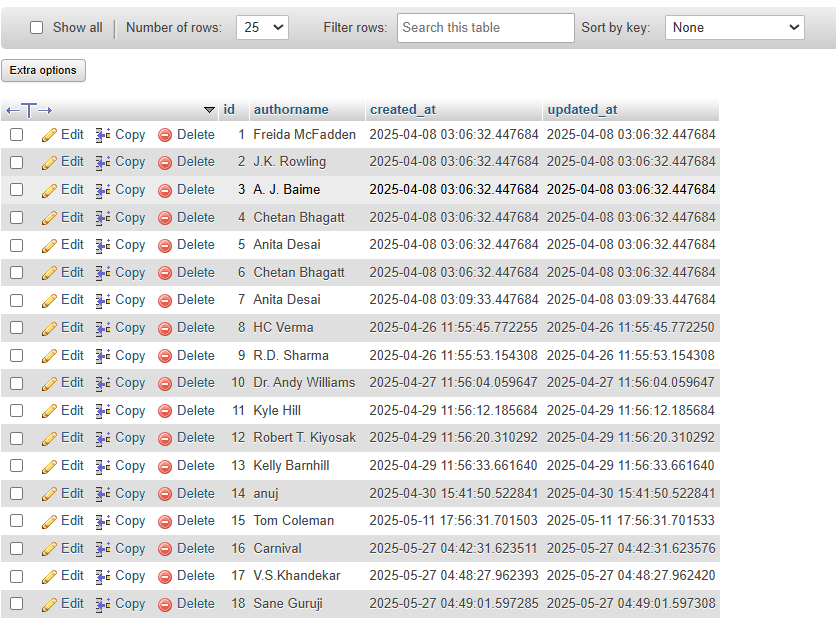
admin and **user Table :**



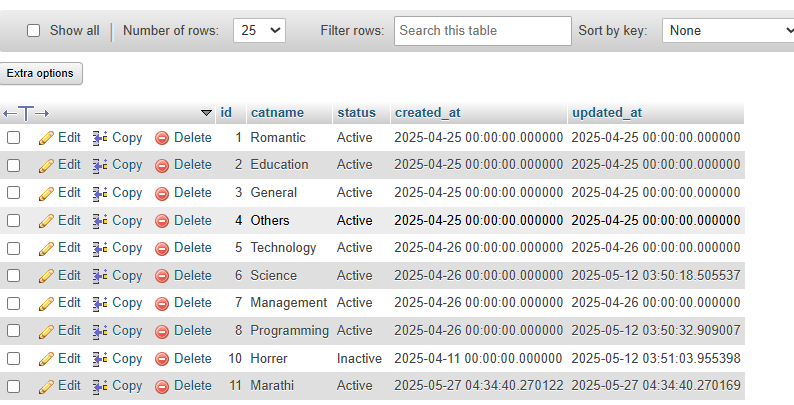
**Book Table: Table name(**olmsapp\_book)



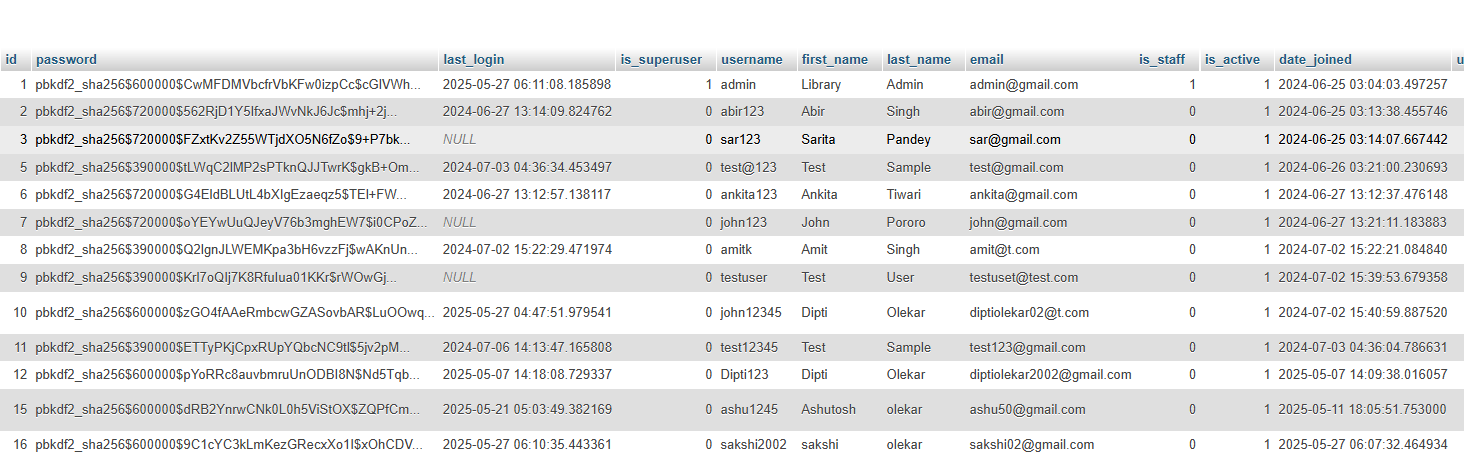
**Author Table: Table name(**olmsapp\_author)



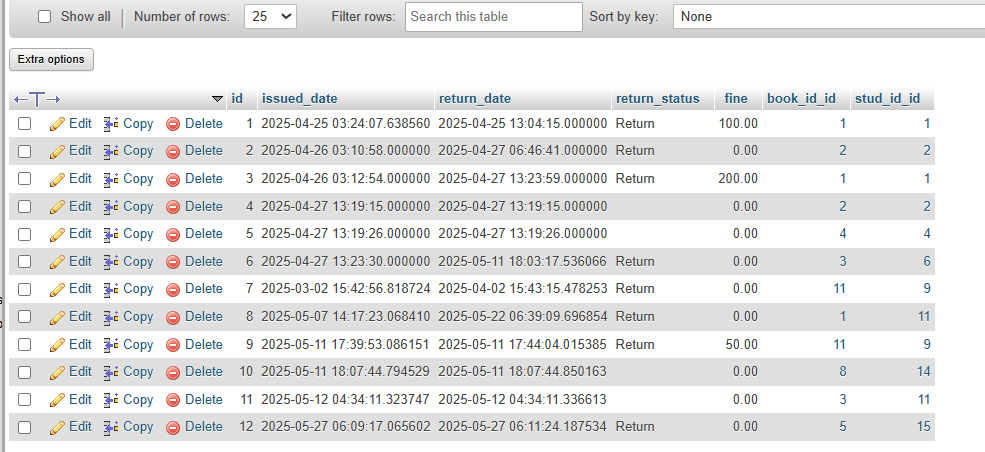
**Category Table: Table name(**olmsapp\_category)



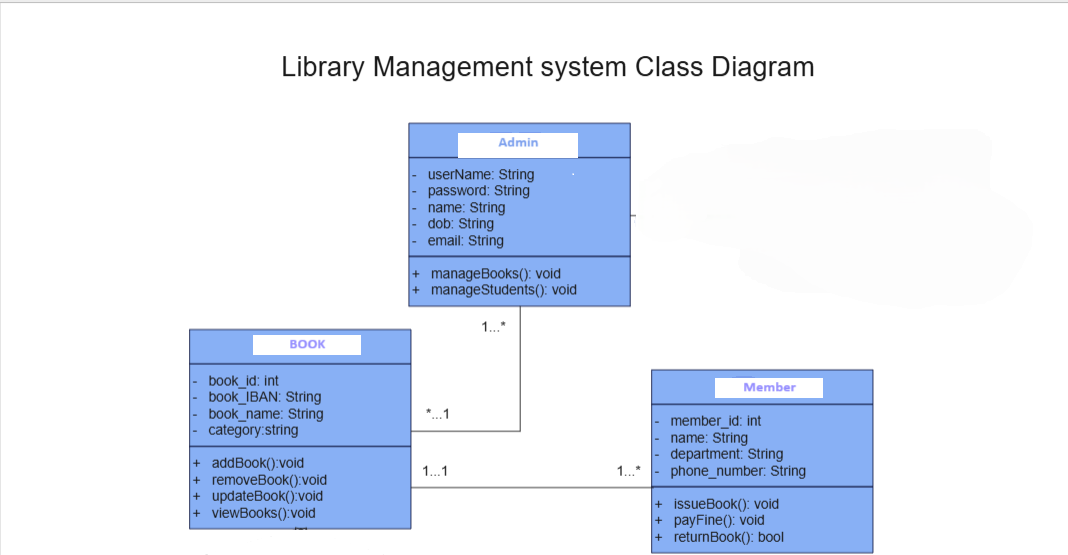
**User Table: Table name(**olmsapp\_user)



**Issuebookdetails Table: Table name(**olmsapp\_issuebookdetails)



**Relationship between tables (Class Diagram)**

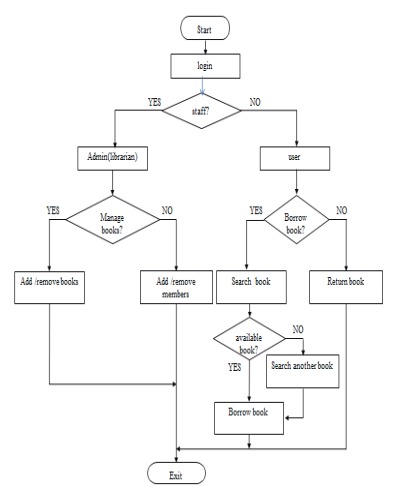


4.3 PROCEDURAL DESIGN:  
Process logic (flowchart ) of each module

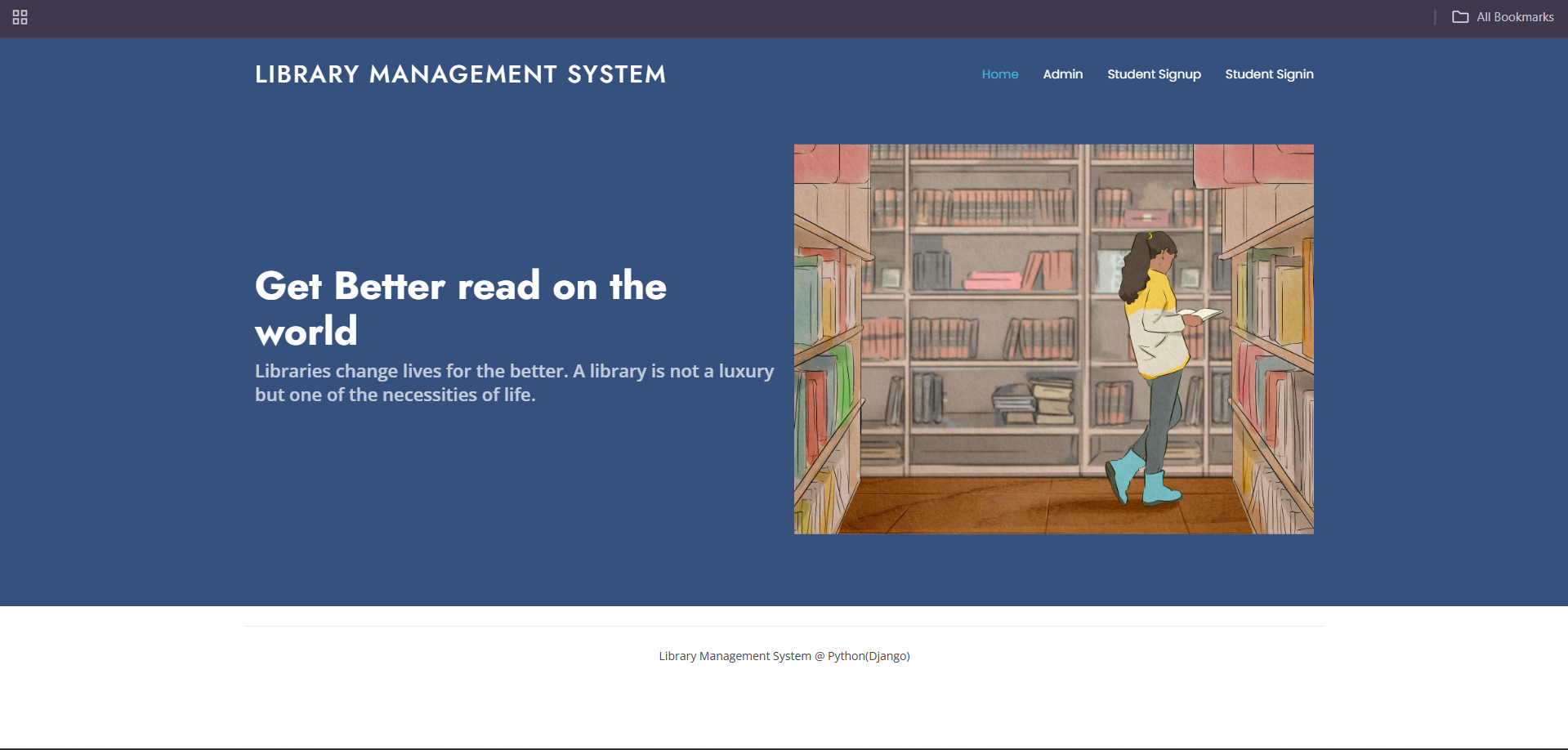
**4.3.1 User Panel Design**

The **User Panel** is designed for ease of access and user-friendly navigation, allowing library members to interact with the system efficiently. Users can visit the homepage and browse books by category or use the search feature to locate specific titles. While book browsing is available without login, borrowing and account management require users to **register or log in**. After logging in, users are directed to their **dashboard**, where they can view and update their profile, see a list of currently borrowed books, check due dates, request new books, return books, and view any pending fines. Notifications may be sent for due dates and overdue books.  
  
**4.3.2 Admin Panel Design**

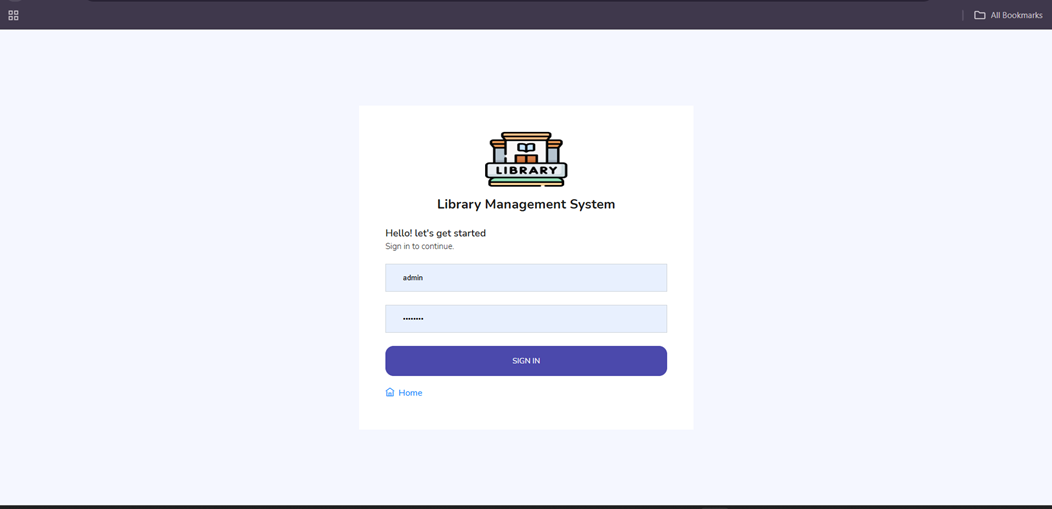
The **Admin Panel** of the Library Management System is the central control interface that allows library administrators to manage all backend operations efficiently. Upon secure login, the admin is taken to a dashboard that offers access to various functional modules. In the **Book Management module**, admins can add, edit, delete

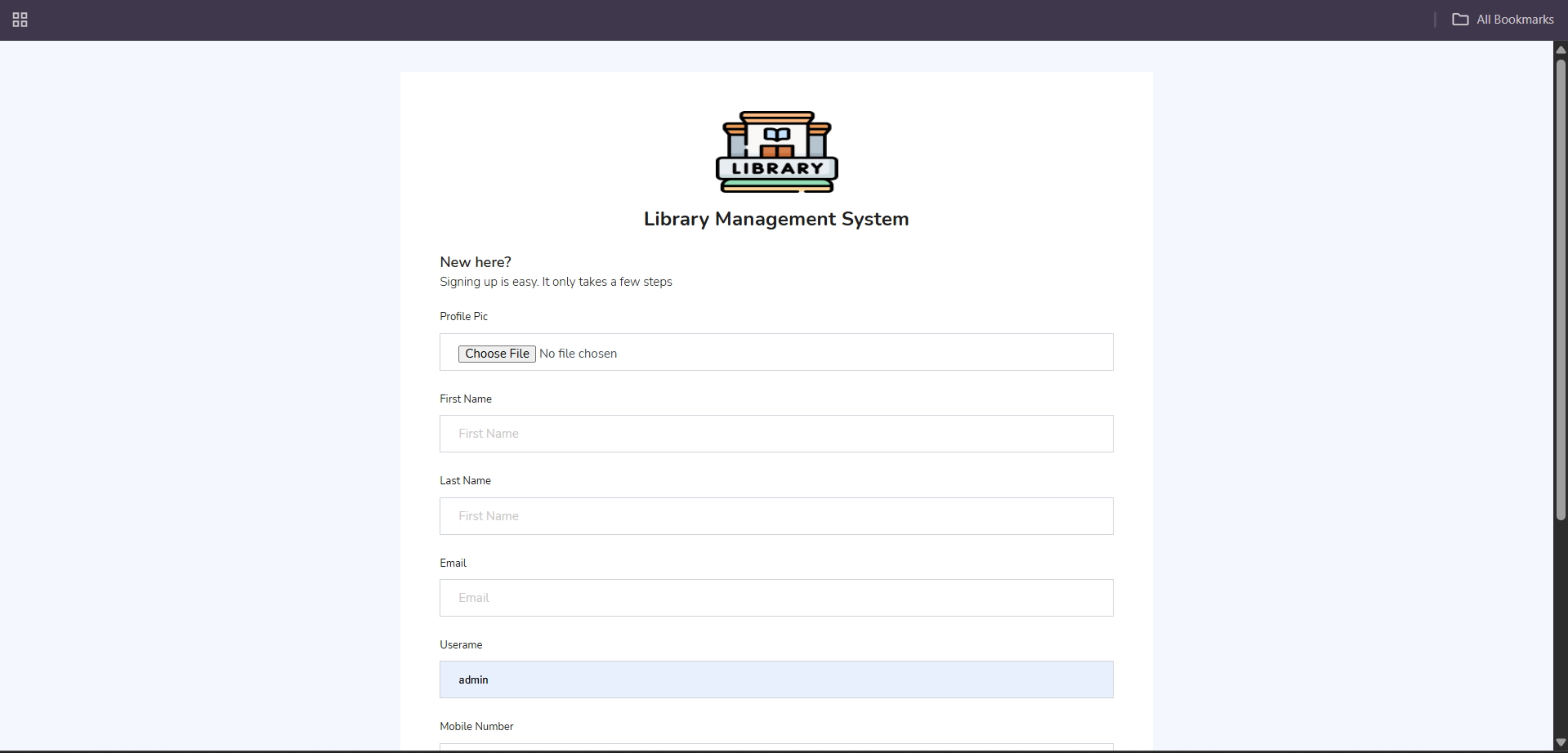


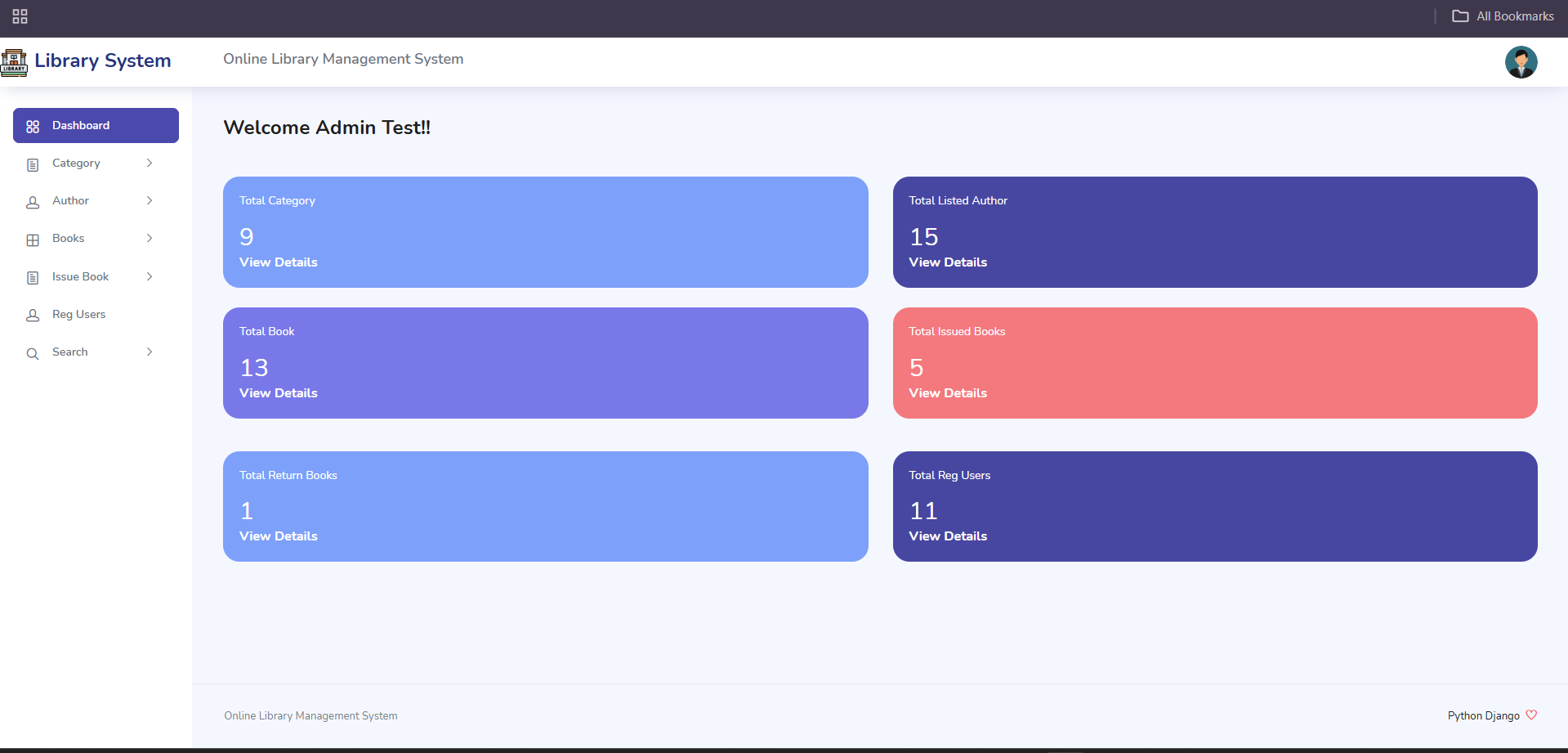
**Home Page**

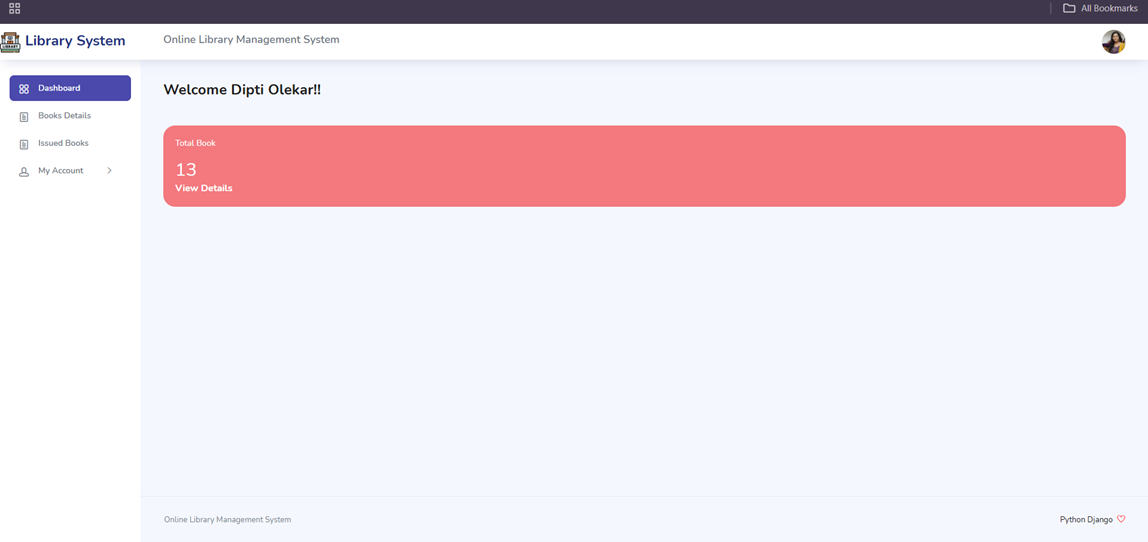
****

**Admin Page**

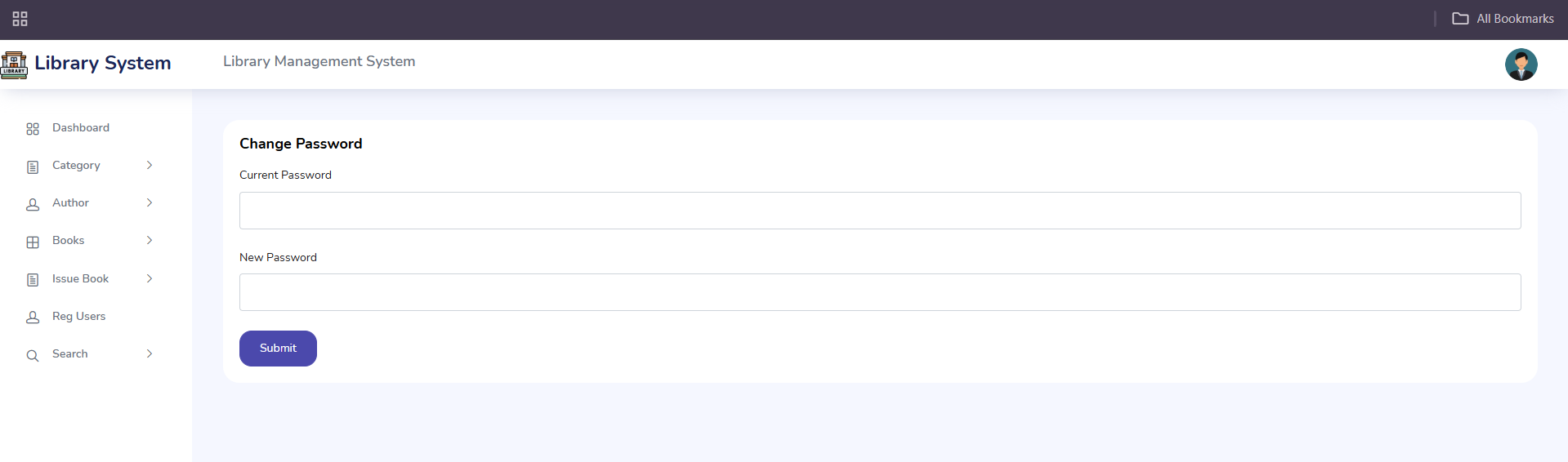


**User Signup Page**

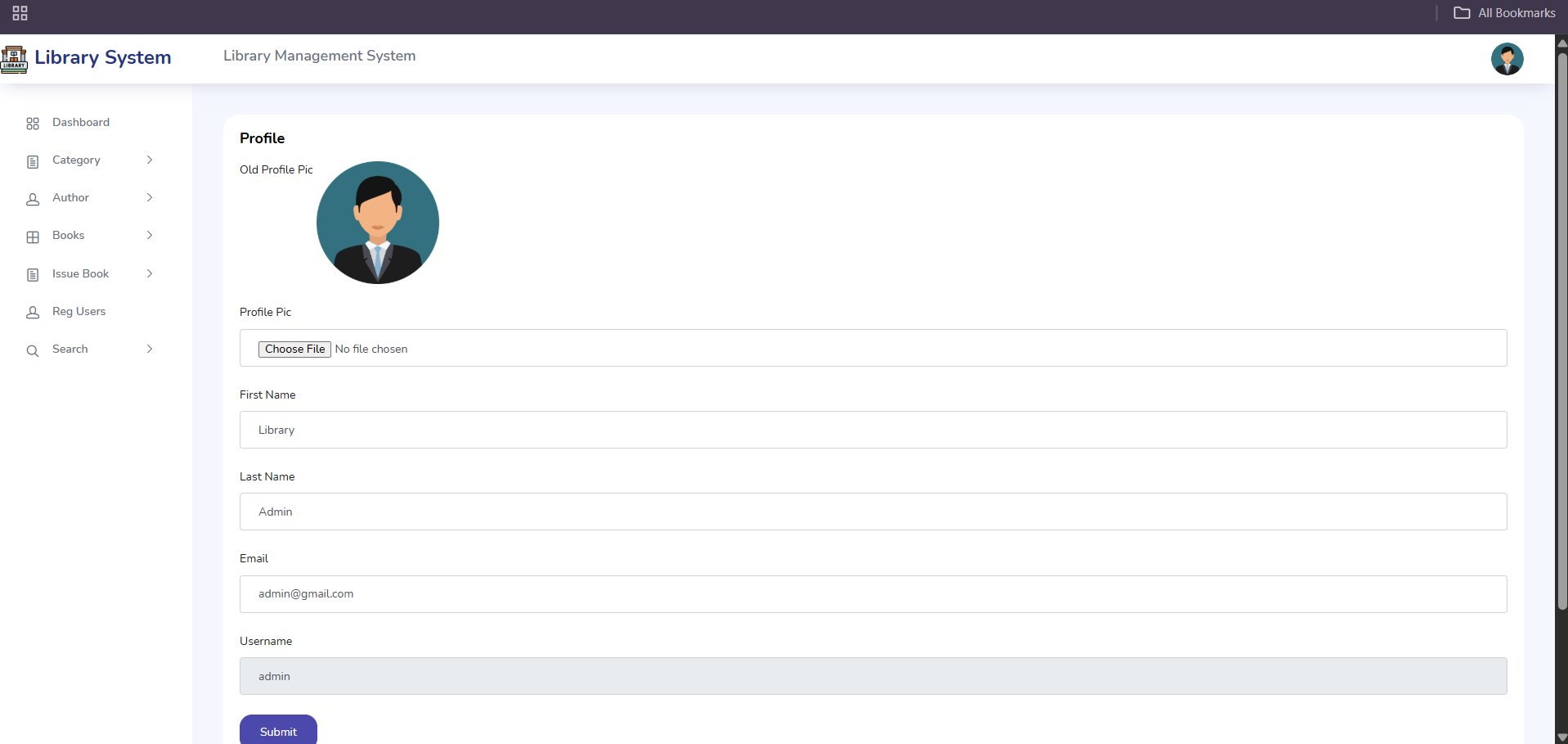
**Admin Dashboard   
  
**

**User Dashboard** 

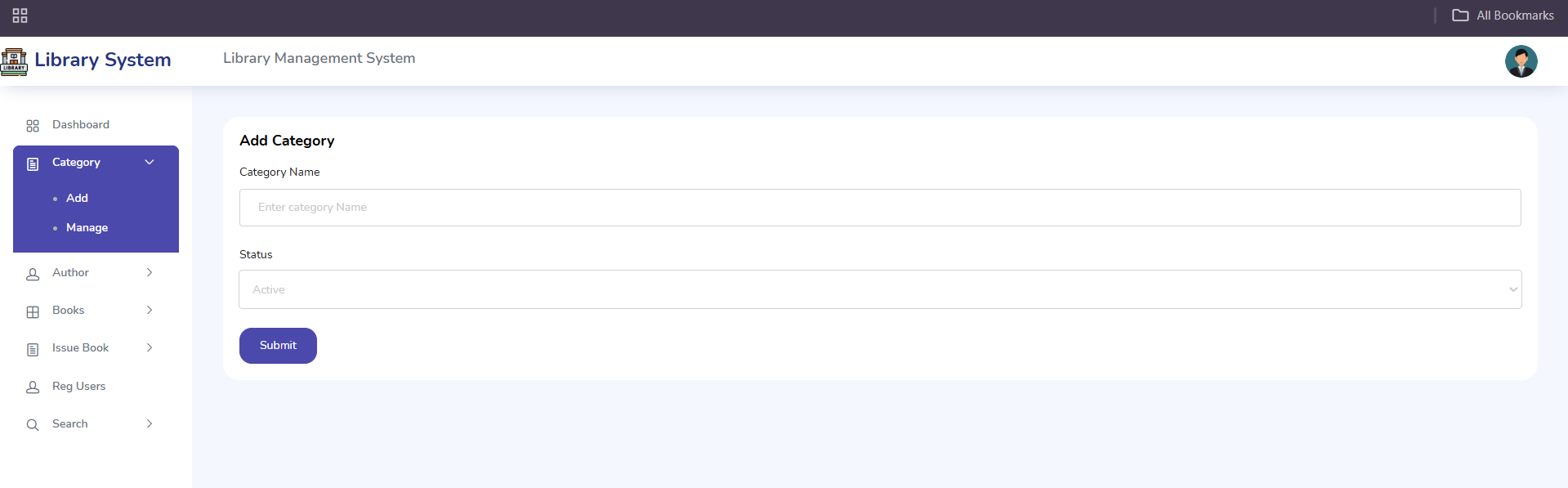
**Change Password**

****

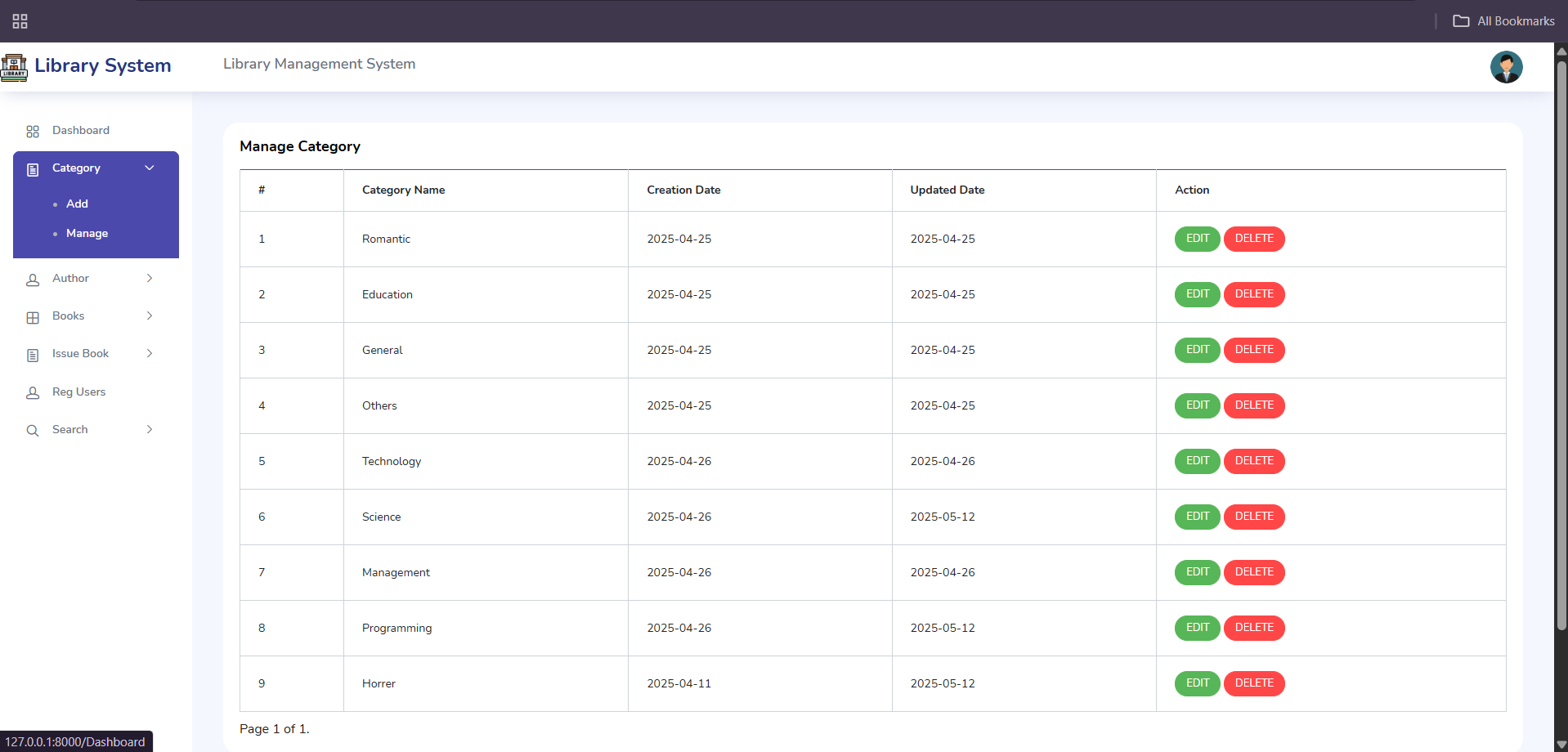
**Admin Profile**

****

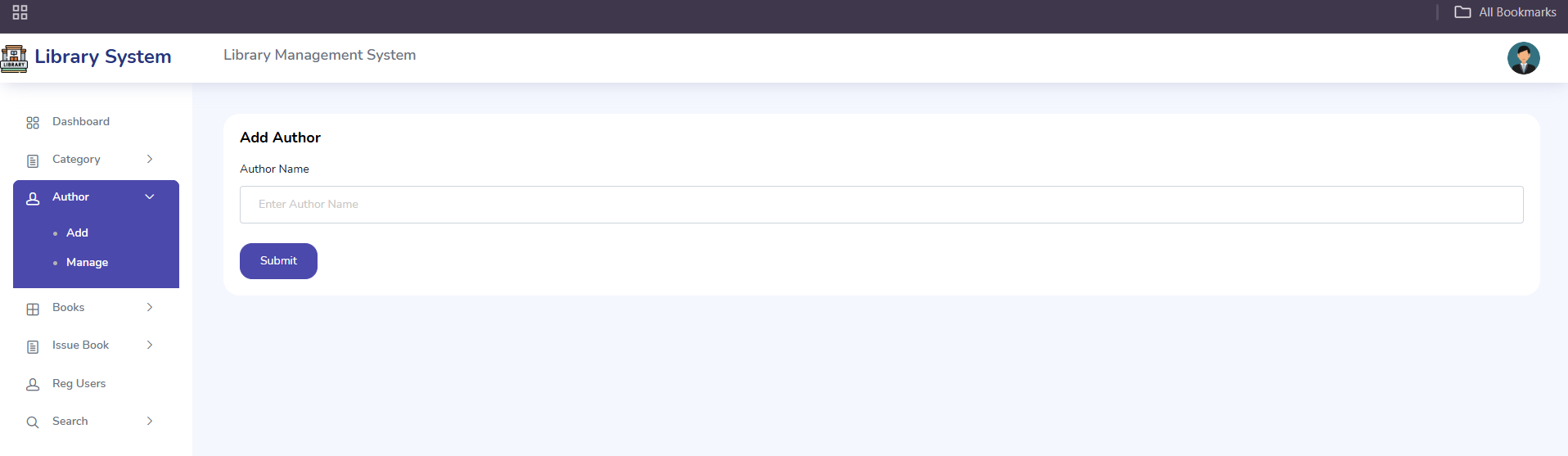
**Add Category**

****

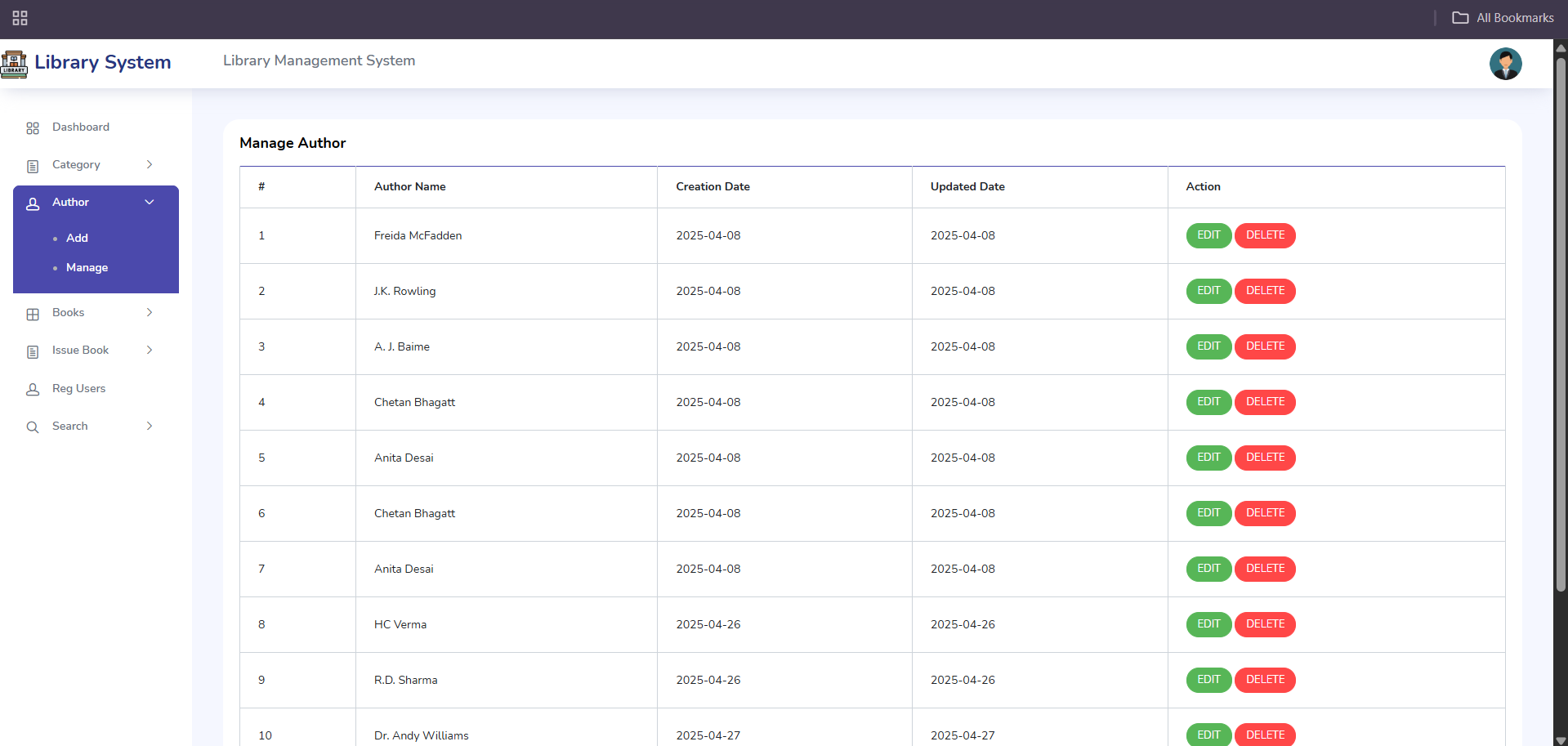
**Manage Category**

****

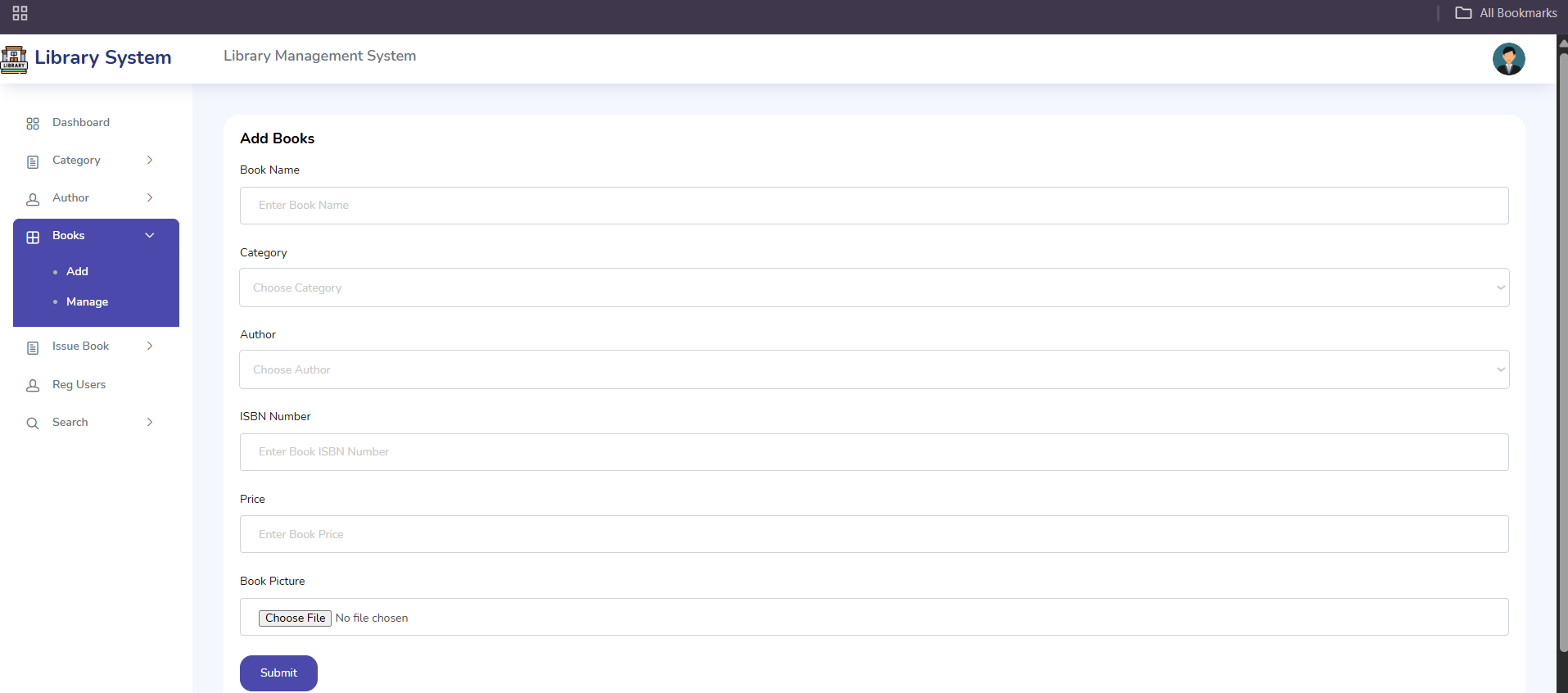
**Add Author**

****

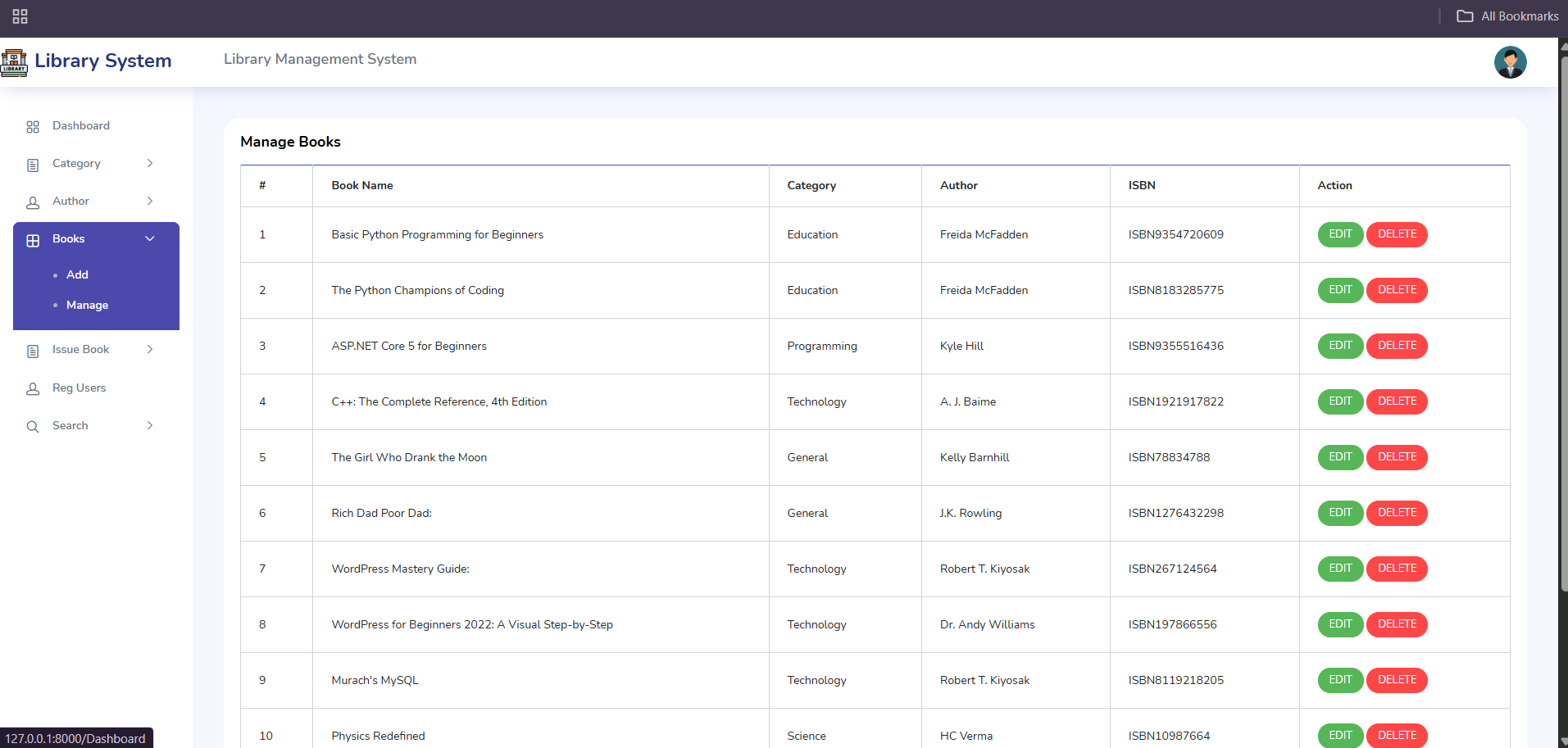
**Manage Author**

****

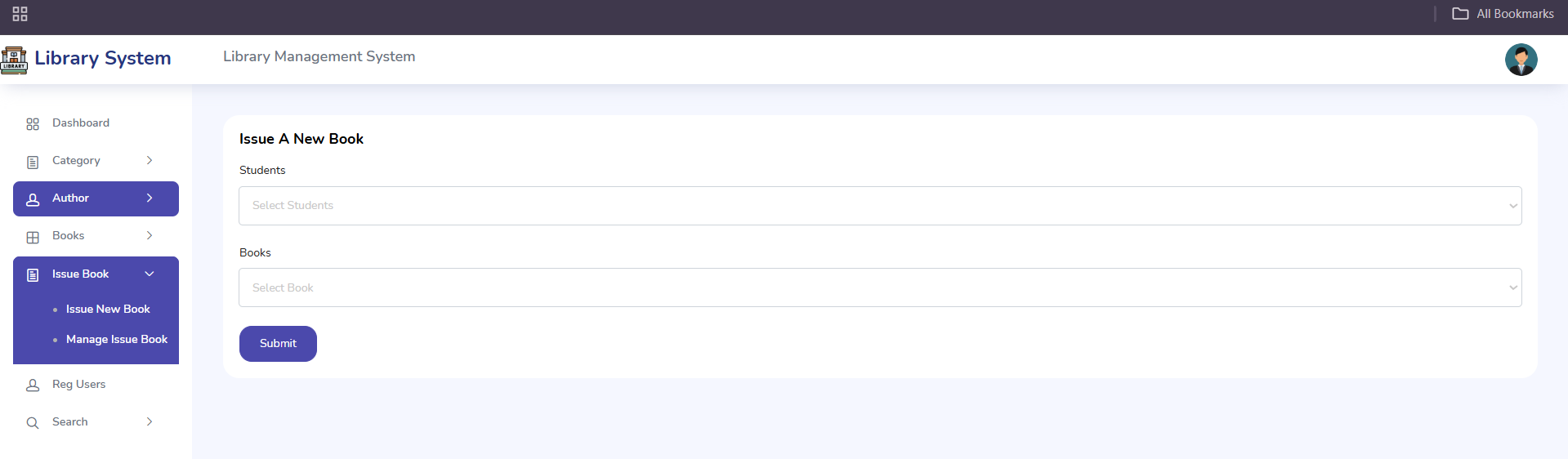
**Add Books**

****

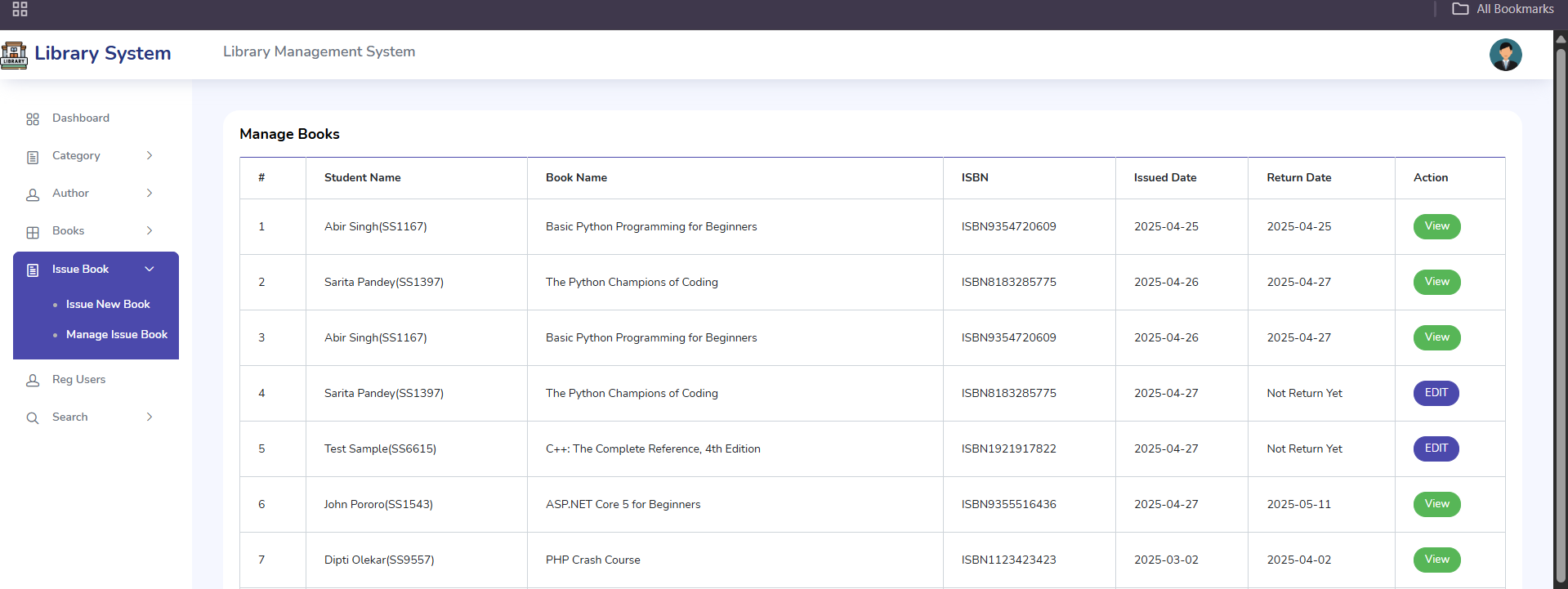
**Manage Books**



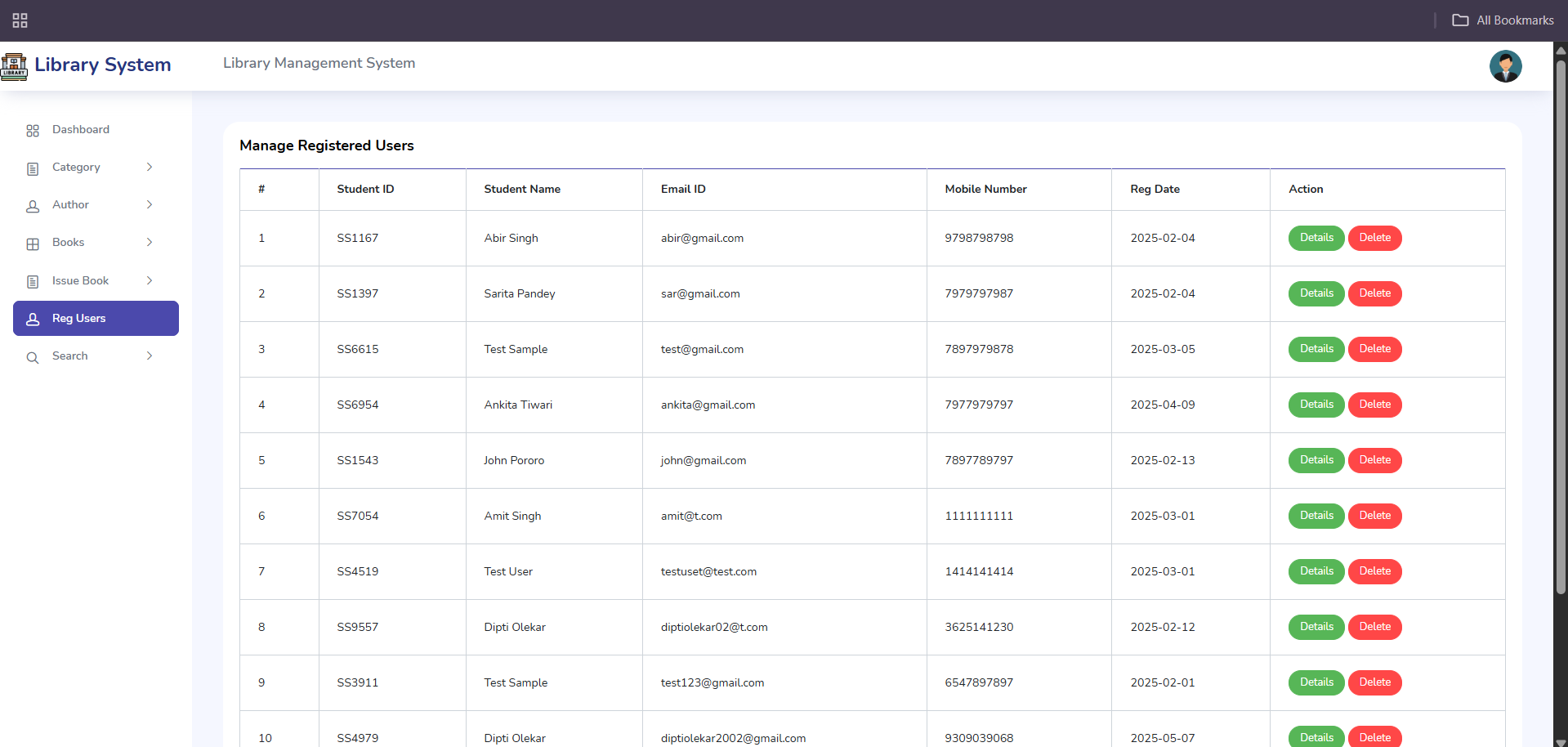
**Issue New Book**

****

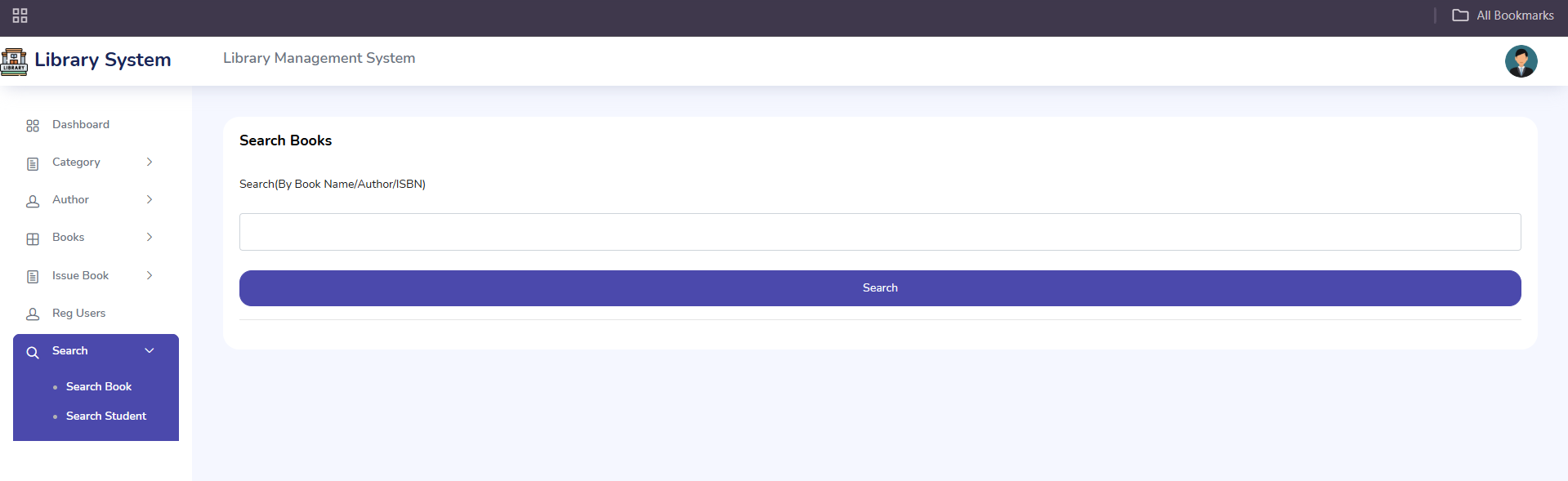
**Manage Issue Books**



**Manage Registered Users**



**Search Books Page**



Code  
**1.Adminview.py** :  
from django.shortcuts import render , redirect # type: ignore

from django.contrib.auth.decorators import login\_required # type: ignore

from django.contrib.auth import authenticate, login,logout # type: ignore

from django.contrib import messages # type: ignore

from olmsapp.models import CustomUser,Category,Author,Book,Student,Issuedbookdetails

from django.contrib.auth import get\_user\_model # type: ignore

from django.core.paginator import Paginator, EmptyPage, PageNotAnInteger # type: ignore

from django.shortcuts import render, get\_object\_or\_404 # type: ignore

import decimal

from django.db.models import Q # type: ignore

User = get\_user\_model()

@login\_required(login\_url='/')

def ADD\_CATEGORY(request):

    if request.method == "POST":

        catname = request.POST.get('catname')

        status = request.POST.get('status')

        cat =Category(

            catname=catname,

            status=status,

        )

        cat.save()

        messages.success(request,'Category has been added succeesfully!!!')

        return redirect("add\_category")

    return render(request,'admin/add-category.html')

@login\_required(login\_url='/')

def MANAGE\_CATEGORY(request):

    cat\_list = Category.objects.all()

    paginator = Paginator(cat\_list, 10)  # Show 10 categories per page

    page\_number = request.GET.get('page')

    try:

        categories = paginator.page(page\_number)

    except PageNotAnInteger:

        # If page is not an integer, deliver first page.

        categories = paginator.page(1)

    except EmptyPage:

        # If page is out of range (e.g. 9999), deliver last page of results.

        categories = paginator.page(paginator.num\_pages)

    context = {'categories': categories,

    }

    return render(request, 'admin/manage\_category.html', context)

@login\_required(login\_url='/')

def DELETE\_CATEGORY(request,id):

    cat = Category.objects.get(id=id)

    cat.delete()

    messages.success(request,'Record Delete Succeesfully!!!')

    return redirect('manage\_category')

login\_required(login\_url='/')

def UPDATE\_CATEGORY(request,id):

    cat = Category.objects.get(id=id)

    context = {

         'cat':cat,

    }

    return render(request,'admin/update\_category.html',context)

login\_required(login\_url='/')

def UPDATE\_CATEGORY\_DETAILS(request):

        if request.method == 'POST':

          cat\_id = request.POST.get('cat\_id')

          catname = request.POST.get('catname')

          status = request.POST.get('status')

          category = Category.objects.get(id=cat\_id)

          category.catname = catname

          category.status = status

          category.save()

          messages.success(request,"Your category detail has been updated successfully")

          return redirect('manage\_category')

        return render(request, 'admin/update\_category.html')

@login\_required(login\_url='/')

def ADD\_AUTHOR(request):

    if request.method == "POST":

        authorname = request.POST.get('authorname')

        authorinfo =Author(

            authorname=authorname,

        )

        authorinfo.save()

        messages.success(request,'Author info has been added succeesfully!!!')

        return redirect("add\_author")

    return render(request,'admin/add-author.html')

@login\_required(login\_url='/')

def MANAGE\_AUTHOR(request):

    auth\_list = Author.objects.all()

    paginator = Paginator(auth\_list, 10)  # Show 10 authors per page

    page\_number = request.GET.get('page')

    try:

        authors = paginator.page(page\_number)

    except PageNotAnInteger:

        # If page is not an integer, deliver first page.

        authors = paginator.page(1)

    except EmptyPage:

        # If page is out of range (e.g. 9999), deliver last page of results.

        authors = paginator.page(paginator.num\_pages)

    context = {'authors': authors,

    }

    return render(request, 'admin/manage\_author.html', context)

@login\_required(login\_url='/')

def DELETE\_AUTHOR(request,id):

    auth = Author.objects.get(id=id)

    auth.delete()

    messages.success(request,'Record Delete Succeesfully!!!')

    return redirect('manage\_author')

login\_required(login\_url='/')

def UPDATE\_AUTHOR(request,id):

    auth = Author.objects.get(id=id)

    context = {

         'auth':auth,

    }

    return render(request,'admin/update\_author.html',context)

login\_required(login\_url='/')

def UPDATE\_AUTHOR\_DETAILS(request):

        if request.method == 'POST':

          auth\_id = request.POST.get('auth\_id')

          authorname = request.POST.get('authorname')

          authors = Author.objects.get(id=auth\_id)

          authors.authorname = authorname

          authors.save()

          messages.success(request,"Your author detail has been updated successfully")

          return redirect('manage\_author')

        return render(request, 'admin/update\_author.html')

@login\_required(login\_url='/')

def ADD\_BOOKS(request):

    categories = Category.objects.all()

    authors = Author.objects.all()

    if request.method == "POST":

        bookname = request.POST.get('bookname')

        catid = request.POST.get('catid')

        authid = request.POST.get('authid')

        isbnnum = request.POST.get('isbnnum')

        price = request.POST.get('price')

        bookimage = request.FILES.get('bookimage')

        try:

            category = Category.objects.get(id=catid)

            author = Author.objects.get(id=authid)

        except (Category.DoesNotExist, Author.DoesNotExist):

            messages.error(request, 'Invalid category or author ID')

            return redirect('add\_books')

        bookinfo = Book(

            bookname=bookname,

            catid=category,

            authid=author,

            isbnnum=isbnnum,

            price=price,

            bookimage=bookimage,

            isIssued='0'

        )

        bookinfo.save()

        messages.success(request, 'Book info has been added successfully!')

        return redirect('add\_books')

    context = {

        'categories': categories,

        'authors': authors,

    }

    return render(request, 'admin/add-books.html', context)

@login\_required(login\_url='/')

def MANAGE\_BOOKS(request):

    book\_list = Book.objects.all()

    paginator = Paginator(book\_list, 10)  # Show 10 books per page

    page\_number = request.GET.get('page')

    try:

        books = paginator.page(page\_number)

    except PageNotAnInteger:

        # If page is not an integer, deliver first page.

        books = paginator.page(1)

    except EmptyPage:

        # If page is out of range (e.g. 9999), deliver last page of results.

        books = paginator.page(paginator.num\_pages)

    context = {'books': books,

    }

    return render(request, 'admin/manage\_books.html', context)

@login\_required(login\_url='/')

def DELETE\_BOOKS(request,id):

    books = Book.objects.get(id=id)

    books.delete()

    messages.success(request,'Record Delete Succeesfully!!!')

    return redirect('manage\_books')

login\_required(login\_url='/')

def UPDATE\_BOOKS(request,id):

    books = Book.objects.get(id=id)

    context = {

         'books':books,

    }

    return render(request,'admin/update\_books.html',context)

login\_required(login\_url='/')

def UPDATE\_BOOKS\_DETAILS(request):

        if request.method == 'POST':

          book\_id = request.POST.get('book\_id')

          bookname = request.POST.get('bookname')

          authid = request.POST.get('authid')

          cat\_id = request.POST.get('catid')

          isbnnum = request.POST.get('isbnnum')

          price = request.POST.get('price')

          bookimage = request.FILES.get('bookimage')

          try:

            books = get\_object\_or\_404(Book,id=book\_id)

            category = get\_object\_or\_404(Category, id=cat\_id)

            author = get\_object\_or\_404(Author, id=authid)

            books.bookname = bookname

            books.authid = author

            books.catid = category

            books.isbnnum = isbnnum

            books.price = price

            if bookimage:

                books.bookimage = bookimage

            books.save()

            messages.success(request,"Your author detail has been updated successfully")

            return redirect('manage\_books')

          except (Author.DoesNotExist, Category.DoesNotExist, Book.DoesNotExist):

            messages.error(request, "Invalid ID provided for subcategory, category, or news post")

            return redirect('update\_books')

        return render(request, 'admin/update\_books.html')

login\_required(login\_url='/')

def ISSUE\_BOOK(request):

    students = Student.objects.all()

    books = Book.objects.all()

    context = {'books': books,

    'students':students

    }

    return render(request,'admin/issue\_book.html',context)

@login\_required(login\_url='/')

def ISSUE\_BOOK(request):

    if request.method == 'POST':

        book\_id = request.POST.get('bookid')

        stud\_id = request.POST.get('stuid')

        try:

            # Fetch the Book instance

            book = Book.objects.get(id=book\_id)

            student = Student.objects.get(id=stud\_id)

            # Create the issued book record

            issued\_book = Issuedbookdetails.objects.create(

                book\_id=book,  # Pass the Book instance

                stud\_id=student,

            )

            issued\_book.save()

            # Update the isIssued field in the Book model

            book.isIssued = True  # Assuming isIssued is a BooleanField

            book.save()

            messages.success(request, 'Book issued successfully!')

            return redirect('issue\_book')  # Replace with your success URL

        except Exception as e:

            messages.error(request, f'Error issuing book: {e}')

            return redirect('issue\_book')  # Redirect back to the issue book page

    else:

        students = Student.objects.all()

        # Filter books that have not been issued

        books = Book.objects.filter(Q(isIssued='0') | Q(isIssued='Return'))

        context = {

            'books': books,

            'students': students

        }

        return render(request, 'admin/issue\_book.html', context)

@login\_required(login\_url='/')

def MANAGE\_ISSUEDBOOKS(request):

    issuebook\_list = Issuedbookdetails.objects.all()

    paginator = Paginator(issuebook\_list, 10)  # Show 10 issued\_books per page

    page\_number = request.GET.get('page')

    try:

        issued\_books = paginator.page(page\_number)

    except PageNotAnInteger:

        # If page is not an integer, deliver first page.

        issued\_books = paginator.page(1)

    except EmptyPage:

        # If page is out of range (e.g. 9999), deliver last page of results.

        issued\_books = paginator.page(paginator.num\_pages)

    context = {'issued\_books': issued\_books,

    }

    return render(request, 'admin/manage\_issuedbook.html', context)

login\_required(login\_url='/')

def UPDATE\_IBSTATUS(request,id):

    iss\_books = Issuedbookdetails.objects.get(id=id)

    context = {

         'iss\_books':iss\_books,

    }

    return render(request,'admin/update\_issue\_book\_details.html',context)

login\_required(login\_url='/')

def UPDATE\_IBSTATUS\_DETAILS(request):

        if request.method == 'POST':

          book\_id = request.POST.get('bookid')

          issbkid = request.POST.get('issbk\_id')

          fine = request.POST.get('fine')

          try:

            books = get\_object\_or\_404(Book,id=book\_id)

            issbks = get\_object\_or\_404(Issuedbookdetails, id=issbkid)

            books.isIssued = "Return"

            issbks.return\_status = "Return"

            issbks.fine = fine

            books.save()

            issbks.save()

            messages.success(request,"Issue book detail has been updated successfully")

            return redirect('manage\_issued\_books')

          except (Book.DoesNotExist, Issuedbookdetails.DoesNotExist):

            messages.error(request, "Invalid ID provided for book or issue book")

            return redirect('update\_ib\_status')

        return render(request, 'admin/manage\_issuedbook.html')

@login\_required(login\_url='/')

def MANAGE\_REGUSERS(request):

    student\_list = Student.objects.all()

    paginator = Paginator(student\_list, 10)  # Show 10 student list per page

    page\_number = request.GET.get('page')

    try:

        student\_list = paginator.page(page\_number)

    except PageNotAnInteger:

        # If page is not an integer, deliver first page.

        student\_list = paginator.page(1)

    except EmptyPage:

        # If page is out of range (e.g. 9999), deliver last page of results.

        student\_list = paginator.page(paginator.num\_pages)

    context = {'student\_list': student\_list,

    }

    return render(request, 'admin/manage\_regusers.html', context)

@login\_required(login\_url='/')

def DELETE\_REGUSERS(request, id):

    try:

        student = get\_object\_or\_404(Student, id=id)

        custom\_user = student.admin  # Access the related CustomUser

        student.delete()  # This will also delete the associated CustomUser because of the on\_delete=models.CASCADE

        custom\_user.delete()

        messages.success(request, 'Record deleted successfully!')

    except Exception as e:

        messages.error(request, f'Error deleting record: {e}')

    return redirect('manage\_regusers')

@login\_required(login\_url='/')

def STUDENT\_LIB\_HISTORY(request,id):

    issuebook\_list = Issuedbookdetails.objects.filter(stud\_id=id)

    paginator = Paginator(issuebook\_list, 10)  # Show 10 issued\_books per page

    page\_number = request.GET.get('page')

    try:

        issued\_books = paginator.page(page\_number)

    except PageNotAnInteger:

        # If page is not an integer, deliver first page.

        issued\_books = paginator.page(1)

    except EmptyPage:

        # If page is out of range (e.g. 9999), deliver last page of results.

        issued\_books = paginator.page(paginator.num\_pages)

    context = {'issued\_books': issued\_books,

    }

    return render(request, 'admin/student\_lib\_history.html', context)

@login\_required(login\_url='/')

def SEARCHBOOK(request):

    if request.method == "GET":

        query = request.GET.get('query', '')

        if query:

            searchbook = Book.objects.filter(bookname\_\_icontains=query) | Book.objects.filter(authid\_\_authorname\_\_icontains=query) | Book.objects.filter(isbnnum\_\_icontains=query)

            messages.info(request, "Search against " + query)

            return render(request, 'admin/search-book.html', {'searchbook': searchbook, 'query': query})

        else:

            return render(request, 'admin/search-book.html', {})

@login\_required(login\_url='/')

def SEARCHREGUSERS(request):

    if request.method == "GET":

        query = request.GET.get('query', '')

        if query:

            searchreguser = Student.objects.filter(mobilenumber\_\_icontains=query) | Student.objects.filter(admin\_\_first\_name\_\_icontains=query) | Student.objects.filter(admin\_\_last\_name\_\_icontains=query) | Student.objects.filter(studentid\_\_icontains=query)

            messages.info(request, "Search against " + query)

            return render(request, 'admin/search-regusers.html', {'searchreguser': searchreguser, 'query': query})

        else:

            return render(request, 'admin/search-regusers.html', {})

**2. Asgi.py :**  
"""

ASGI config for olms project.

It exposes the ASGI callable as a module-level variable named ``application``.

For more information on this file, see

https://docs.djangoproject.com/en/5.0/howto/deployment/asgi/

"""

import os

from django.core.asgi import get\_asgi\_application # type: ignore

os.environ.setdefault('DJANGO\_SETTINGS\_MODULE', 'olms.settings')

application = get\_asgi\_application()

**3 . Settings.py :**

"""

Django settings for olms project.

Generated by 'django-admin startproject' using Django 5.0.6.

For more information on this file, see

https://docs.djangoproject.com/en/5.0/topics/settings/

For the full list of settings and their values, see

https://docs.djangoproject.com/en/5.0/ref/settings/

"""

from pathlib import Path

import os

# Build paths inside the project like this: BASE\_DIR / 'subdir'.

BASE\_DIR = Path(\_\_file\_\_).resolve().parent.parent

# Quick-start development settings - unsuitable for production

# See https://docs.djangoproject.com/en/5.0/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!

SECRET\_KEY = 'django-insecure-u#%ukrm!5aysp@vub7u6v98$j4f8jb21ia2dtcm3!yuz6cso$d'

# SECURITY WARNING: don't run with debug turned on in production!

DEBUG = True

ALLOWED\_HOSTS = []

# Application definition

INSTALLED\_APPS = [

    'django.contrib.admin',

    'django.contrib.auth',

    'django.contrib.contenttypes',

    'django.contrib.sessions',

    'django.contrib.messages',

    'django.contrib.staticfiles',

    'olmsapp',

]

MIDDLEWARE = [

    'django.middleware.security.SecurityMiddleware',

    'django.contrib.sessions.middleware.SessionMiddleware',

    'django.middleware.common.CommonMiddleware',

    'django.middleware.csrf.CsrfViewMiddleware',

    'django.contrib.auth.middleware.AuthenticationMiddleware',

    'django.contrib.messages.middleware.MessageMiddleware',

    'django.middleware.clickjacking.XFrameOptionsMiddleware',

]

ROOT\_URLCONF = 'olms.urls'

TEMPLATES = [

    {

        'BACKEND': 'django.template.backends.django.DjangoTemplates',

        'DIRS': ['templates'],

        'APP\_DIRS': True,

        'OPTIONS': {

            'context\_processors': [

                'django.template.context\_processors.debug',

                'django.template.context\_processors.request',

                'django.contrib.auth.context\_processors.auth',

                'django.contrib.messages.context\_processors.messages',

            ],

        },

    },

]

WSGI\_APPLICATION = 'olms.wsgi.application'

# Database

# https://docs.djangoproject.com/en/5.0/ref/settings/#databases

DATABASES = {

    'default': {

        'ENGINE': 'django.db.backends.mysql',

        'NAME': 'olmspythondb',

        'USER': 'root',

        'PASSWORD': 'Diptiole@0207',

        'HOST': 'localhost',

        'PORT': '3306',

    }

}

# Password validation

# https://docs.djangoproject.com/en/5.0/ref/settings/#auth-password-validators

AUTH\_PASSWORD\_VALIDATORS = [

    {

        'NAME': 'django.contrib.auth.password\_validation.UserAttributeSimilarityValidator',

    },

    {

        'NAME': 'django.contrib.auth.password\_validation.MinimumLengthValidator',

    },

    {

        'NAME': 'django.contrib.auth.password\_validation.CommonPasswordValidator',

    },

    {

        'NAME': 'django.contrib.auth.password\_validation.NumericPasswordValidator',

    },

]

# Internationalization

# https://docs.djangoproject.com/en/5.0/topics/i18n/

LANGUAGE\_CODE = 'en-us'

TIME\_ZONE = 'UTC'

USE\_I18N = True

USE\_TZ = True

# Static files (CSS, JavaScript, Images)

# https://docs.djangoproject.com/en/5.0/howto/static-files/

STATIC\_URL = '/static/'

STATIC\_ROOT = '/static/'

STATICFILES\_DIRS = [

    os.path.join(BASE\_DIR,'static')

]

# Default primary key field type

# https://docs.djangoproject.com/en/5.0/ref/settings/#default-auto-field

DEFAULT\_AUTO\_FIELD = 'django.db.models.BigAutoField'

AUTH\_USER\_MODEL = 'olmsapp.CustomUser'

**4 . Stuviews.py :**

from django.shortcuts import render , redirect

from django.contrib.auth.decorators import login\_required

from django.contrib.auth import authenticate, login,logout

from django.contrib import messages

from olmsapp.models import CustomUser,Category,Author,Book,Student,Issuedbookdetails

from django.contrib.auth import get\_user\_model

from django.core.paginator import Paginator, EmptyPage, PageNotAnInteger

from django.shortcuts import render, get\_object\_or\_404

import random

User = get\_user\_model()

def STUDENT\_REG(request):

    if request.method == "POST":

        random\_digits = random.randint(1000, 9999)

        studentid = f"SS{random\_digits}"

        pic = request.FILES.get('pic')

        first\_name = request.POST.get('first\_name')

        last\_name = request.POST.get('last\_name')

        username = request.POST.get('username')

        email = request.POST.get('email')

        mobno = request.POST.get('mobno')

        password = request.POST.get('password')

        if CustomUser.objects.filter(email=email).exists():

            messages.warning(request,'Email already exist')

            return redirect('signup')

        if CustomUser.objects.filter(username=username).exists():

            messages.warning(request,'Username already exist')

            return redirect('signup')

        else:

            user = CustomUser(

               first\_name=first\_name,

               last\_name=last\_name,

               username=username,

               email=email,

               user\_type=2,

               profile\_pic = pic,

            )

            user.set\_password(password)

            user.save()

            student = Student(

                admin = user,

                studentid= studentid,

                mobilenumber=mobno,

            )

            student.save()

            messages.success(request,'Signup Successfully')

            return redirect('signup')

    return render(request,'student/registration.html')

@login\_required(login\_url='/')

def ISSUEDBOOKS(request):

    stu\_admin = request.user

    stu\_reg = Student.objects.get(admin=stu\_admin)

    issuebook\_list = Issuedbookdetails.objects.filter(stud\_id=stu\_reg)

    paginator = Paginator(issuebook\_list, 10)  # Show 10 issued\_books per page

    page\_number = request.GET.get('page')

    try:

        issued\_books = paginator.page(page\_number)

    except PageNotAnInteger:

        # If page is not an integer, deliver first page.

        issued\_books = paginator.page(1)

    except EmptyPage:

        # If page is out of range (e.g. 9999), deliver last page of results.

        issued\_books = paginator.page(paginator.num\_pages)

    context = {'issued\_books': issued\_books,

    }

    return render(request, 'student/issuedbooks.html', context)

@login\_required(login\_url='/')

def BOOKSDETAILS(request):

       return render(request,'student/books\_details.html')

@login\_required(login\_url='/')

def BOOKSDETAILS(request):

    book\_list = Book.objects.all()

    paginator = Paginator(book\_list, 12)  # Show 10 books per page

    page\_number = request.GET.get('page')

    try:

        books = paginator.page(page\_number)

    except PageNotAnInteger:

        # If page is not an integer, deliver first page.

        books = paginator.page(1)

    except EmptyPage:

        # If page is out of range (e.g. 9999), deliver last page of results.

        books = paginator.page(paginator.num\_pages)

    context = {'books': books,

    }

    return render(request, 'student/books\_details.html', context)

**5 . urls.py :**

"""

URL configuration for olms project.

The `urlpatterns` list routes URLs to views. For more information please see:

    https://docs.djangoproject.com/en/5.0/topics/http/urls/

Examples:

Function views

    1. Add an import:  from my\_app import views

    2. Add a URL to urlpatterns:  path('', views.home, name='home')

Class-based views

    1. Add an import:  from other\_app.views import Home

    2. Add a URL to urlpatterns:  path('', Home.as\_view(), name='home')

Including another URLconf

    1. Import the include() function: from django.urls import include, path

    2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))

"""

from django.contrib import admin

from django.urls import path

from .import views,adminviews,stuviews

from django.conf import settings

from django.conf.urls.static import static

urlpatterns = [

    path('admin/', admin.site.urls),

    path('base/', views.BASE, name='base'),

    path('Dashboard', views.DASHBOARD, name='dashboard'),

    path('Login', views.LOGIN, name='login'),

    path('doLogin', views.doLogin, name='doLogin'),

    path('', views.Index, name='index'),

    path('doLogout', views.doLogout, name='logout'),

    path('AdminProfile', views.ADMIN\_PROFILE, name='admin\_profile'),

    path('AdminProfile/update', views.ADMIN\_PROFILE\_UPDATE, name='admin\_profile\_update'),

    path('Password', views.CHANGE\_PASSWORD, name='change\_password'),

    path('Admin/AddCategory', adminviews.ADD\_CATEGORY, name='add\_category'),

    path('Admin/ManageCategory', adminviews.MANAGE\_CATEGORY, name='manage\_category'),

    path('Admin/DeleteCategory/<str:id>', adminviews.DELETE\_CATEGORY, name='delete\_category'),

    path('UpdateCategory/<str:id>', adminviews.UPDATE\_CATEGORY, name='update\_category'),

    path('UpdateCategoryDetails', adminviews.UPDATE\_CATEGORY\_DETAILS, name='update\_category\_details'),

    path('Admin/AddAuthor', adminviews.ADD\_AUTHOR, name='add\_author'),

    path('Admin/ManageAuthor', adminviews.MANAGE\_AUTHOR, name='manage\_author'),

    path('Admin/DeleteAuthor/<str:id>', adminviews.DELETE\_AUTHOR, name='delete\_author'),

    path('UpdateAuthor/<str:id>', adminviews.UPDATE\_AUTHOR, name='update\_author'),

    path('UpdateAuthorDetails', adminviews.UPDATE\_AUTHOR\_DETAILS, name='update\_author\_details'),

    path('Admin/AddBooks', adminviews.ADD\_BOOKS, name='add\_books'),

    path('Admin/ManageBooks', adminviews.MANAGE\_BOOKS, name='manage\_books'),

    path('Admin/DeleteBooks/<str:id>', adminviews.DELETE\_BOOKS, name='delete\_books'),

    path('UpdateBooks/<str:id>', adminviews.UPDATE\_BOOKS, name='update\_books'),

    path('UpdateBooksDetails', adminviews.UPDATE\_BOOKS\_DETAILS, name='update\_books\_details'),

    path('Admin/IssueBook', adminviews.ISSUE\_BOOK, name='issue\_book'),

    path('Admin/ManageIssuedBooks', adminviews.MANAGE\_ISSUEDBOOKS, name='manage\_issued\_books'),

    path('UpdateIBStatus/<str:id>', adminviews.UPDATE\_IBSTATUS, name='update\_ib\_status'),

    path('Admin/UpdateIBstatusdetails', adminviews.UPDATE\_IBSTATUS\_DETAILS, name='update\_ibstattus\_details'),

    path('Admin/ManageRegUsers', adminviews.MANAGE\_REGUSERS, name='manage\_regusers'),

    path('Admin/DeleteRegUsers/<str:id>', adminviews.DELETE\_REGUSERS, name='delete\_regusers'),

    path('Admin/Studenlibhistory/<str:id>', adminviews.STUDENT\_LIB\_HISTORY, name='student\_lib\_history'),

    path('Admin/SearchBook', adminviews.SEARCHBOOK, name='search\_books'),

    path('Admin/SearchRegusers', adminviews.SEARCHREGUSERS, name='search\_regusers'),

    path('Student/IssuedBooks', stuviews.ISSUEDBOOKS, name='issued\_books'),

    path('Student/BooksDetails', stuviews.BOOKSDETAILS, name='books\_details'),

    path('Student/StuReg', stuviews.STUDENT\_REG, name='signup'),

]+static(settings.MEDIA\_URL, document\_root = settings.MEDIA\_ROOT)

**6 . Views.py :**

from django.shortcuts import render , redirect

from django.contrib.auth.decorators import login\_required

from django.contrib.auth import authenticate, login,logout

from django.contrib import messages

from olmsapp.models import CustomUser,Category,Author,Book,Student,Issuedbookdetails

from django.contrib.auth import get\_user\_model

User = get\_user\_model()

def BASE(request):

       return render(request,'base.html')

def Index(request):

       return render(request,'index.html')

def LOGIN(request):

    return render(request,'login.html')

def doLogin(request):

    if request.method == 'POST':

        username = request.POST.get('username')

        password = request.POST.get('password')

        user = authenticate(request, username=username, password=password)

        if user is not None:

            login(request, user)

            user\_type = user.user\_type

            if user\_type == '1' or user\_type == 1:  # Check for both string and integer comparison

                return redirect('dashboard')

            elif user\_type == '2' or user\_type == 2:  # Check for both string and integer comparison

                return redirect('dashboard')

        else:

            messages.error(request, 'Username or Password is not valid')

        # If authentication fails or user\_type doesn't match expected values, redirect to login page

        return redirect('login')

    else:

        # If the request method is not POST, redirect to the login page with an error message

        messages.error(request, 'Invalid request method')

        return redirect('login')

def doLogout(request):

    logout(request)

    return redirect('login')

@login\_required(login\_url = '/')

def DASHBOARD(request):

    cat\_count = Category.objects.all().count()

    author\_count = Author.objects.all().count()

    book\_count = Book.objects.all().count()

    issbook\_count = Book.objects.filter(isIssued=True).count()  # Corrected the filter

    regusers\_count = Student.objects.all().count()

    retbook\_count = Book.objects.filter(isIssued='Return').count()

    context = {

        'cat\_count': cat\_count,

        'author\_count': author\_count,

        'book\_count': book\_count,

        'issbook\_count': issbook\_count,

        'regusers\_count': regusers\_count,

        'retbook\_count' : retbook\_count,

    }

    return render(request, 'dashboard.html', context)

@login\_required(login\_url = '/')

def ADMIN\_PROFILE(request):

    user = CustomUser.objects.get(id = request.user.id)

    context = {

        "user":user,

    }

    return render(request,'profile.html',context)

@login\_required(login\_url = '/')

def ADMIN\_PROFILE\_UPDATE(request):

    if request.method == "POST":

        profile\_pic = request.FILES.get('profile\_pic')

        first\_name = request.POST.get('first\_name')

        last\_name = request.POST.get('last\_name')

        email = request.POST.get('email')

        username = request.POST.get('username')

        print(profile\_pic)

        try:

            customuser = CustomUser.objects.get(id = request.user.id)

            customuser.first\_name = first\_name

            customuser.last\_name = last\_name

            customuser.email = email

            if profile\_pic !=None and profile\_pic != "":

               customuser.profile\_pic = profile\_pic

            customuser.save()

            messages.success(request,"Your profile has been updated successfully")

            return redirect('admin\_profile')

        except:

            messages.error(request,"Your profile updation has been failed")

    return render(request, 'profile.html')

login\_required(login\_url='/')

def CHANGE\_PASSWORD(request):

     context ={}

     ch = User.objects.filter(id = request.user.id)

     if len(ch)>0:

            data = User.objects.get(id = request.user.id)

            context["data"]:data

     if request.method == "POST":

        current = request.POST["cpwd"]

        new\_pas = request.POST['npwd']

        user = User.objects.get(id = request.user.id)

        un = user.username

        check = user.check\_password(current)

        if check == True:

          user.set\_password(new\_pas)

          user.save()

          messages.success(request,'Password Change  Succeesfully!!!')

          user = User.objects.get(username=un)

          login(request,user)

        else:

          messages.success(request,'Current Password wrong!!!')

          return redirect("change\_password")

     return render(request,'change-password.html')

Report  
  
The Library Management System (LMS) project is a software application designed to streamline and automate the management of a library’s resources and operations. This system plays a vital role in managing books, members, borrowing records, and inventory, thereby reducing manual work and improving efficiency. The primary goal of this project is to create a user-friendly platform that allows librarians to maintain accurate records, track book availability, and manage the issuing and returning process with ease. In addition, it provides features for searching books by title, author, or subject, making it easier for users to locate resources quickly.

Developed using technologies such as HTML, CSS, JavaScript, and a backend like PHP or Python with a database like MySQL, the LMS project ensures secure and fast data management. The system includes modules for administrator login, member registration, catalog management, and fine calculation for overdue books. Administrators have full control to add, update, or remove book records and manage user accounts. Users can log in to view available books, request to borrow them, and check their borrowing history. With digital tracking, the system also helps prevent book loss or duplication.

This project is especially beneficial for educational institutions and public libraries that manage a large volume of materials and users. By automating routine tasks, the Library Management System enhances accuracy, saves time, and minimizes errors associated with manual record-keeping. Furthermore, it supports better resource planning and improves the overall library experience for users. In conclusion, the LMS project represents an essential step toward modernizing library services, making them more accessible, efficient, and reliable in the digital age.

Future scope  
  
The future scope of the Library Management System (LMS) project is vast and promising, as it aligns with the growing need for digital transformation in educational and public institutions. With the rapid advancement of technology, future enhancements can significantly expand the system's capabilities, offering a more seamless and intelligent user experience. One major area of development is the integration of cloud-based services, allowing remote access to library resources from anywhere and enabling real-time synchronization of data across multiple branches or campuses. This will be particularly beneficial for institutions with decentralized library networks.

Another key direction is the incorporation of artificial intelligence (AI) and machine learning to provide personalized recommendations based on users' reading history and preferences, similar to how modern e-commerce platforms operate. AI can also assist in automating inventory checks, predicting demand for certain books, and managing overdue fines more intelligently. Additionally, mobile app integration will enhance accessibility, enabling users to search, reserve, and renew books directly from their smartphones, thus promoting convenience.

Security and data privacy will also become a growing concern, so implementing robust encryption, secure authentication, and role-based access control will be essential. Furthermore, incorporating feedback mechanisms and usage analytics can help librarians make informed decisions regarding book procurement and user services.

Bibliography

The bibliography of the Library Management System (LMS) project includes a variety of sources that contributed to the understanding, design, and development of the system. Key references include academic books, research papers, online articles, and official documentation related to software development, database management, and library science. Books such as *"Modern Library Management"* by Dr. S.R. Ranganathan and *"Library and Information Science Textbook"* have provided foundational insights into the traditional practices and modern expectations of library operations. These works helped in understanding how libraries function and the areas that benefit most from automation.

**Books and Research Articles:**

1. **"Five Laws of Library Science" by Dr. S.R. Ranganathan**
   * An essential foundation for library management concepts.
   * https://archive.org/details/fivelawsoflibrar031895mbp
2. **"Library and Information Science Textbook" by Arvind Kumar**
   * Covers modern practices in library systems.
   * https://www.amazon.in/Library-Information-Science-Arvind-Kumar/dp/9352692993
3. **"Development of Library Management System" – Research Paper**
   * Available on ResearchGate.
   * <https://www.researchgate.net/publication/347205312_Development_of_Library_Management_System>