Aim of the Project

The Library Management System is designed to efficiently manage book records, allowing users to borrow, return, search for books, and track availability. The system includes an administrative panel for managing books and a user-friendly interface for library users.

Features of the System

User Mode Features

- **Book Borrowing and Returning**: Users can borrow available books and return them before or on the due date.
- **Search Functionality**: Users can search for books by title, author, or category.
- **Book Availability Tracking**: Users can view whether a book is available or currently borrowed.

Admin Mode Features

- **Add, Edit, and Remove Books**: Administrators can add new books, update existing records, or remove books from the system.
- **Manage Borrowed Books**: The admin can track which books are borrowed and their due dates.
- **Set Book Categories**: Books can be assigned to different categories for easy organization and searching.

Technologies and Packages Used

- **Python (Flask)**: Backend framework for handling web requests and database operations.
- **MySQL**: Database management system for storing book records and user transactions.
- **HTML**, **CSS**, **JavaScript**: Frontend technologies for user interface design.
- **Jinja2**: Templating engine for rendering dynamic content in Flask.
- **MySQL Connector**: Python package for connecting Flask with MySQL.

Project Structure

```
Library_Management_System/
|— static/                  # CSS and JavaScript files
|— templates/               # HTML templates
|— app.py                   # Main Flask application
|— database.sql             # Database schema
|— requirements.txt         # Required Python packages
|— README.md                # Project documentation
```

Setup and Installation

1. Install the required dependencies:

   ```
   pip install -r requirements.txt
   ```

2. Set up the MySQL database using `database.sql`.
3. Run the Flask application:

   ```
   python app.py
   ```

4. Access the system in a web browser at `http://127.0.0.1:5000`.

GitHub Repository

The full source code for this project is available on GitHub:
https://github.com/KevinAiCloud/LibraryManagementSystem

Source Code

app.py file

app.py > update_book

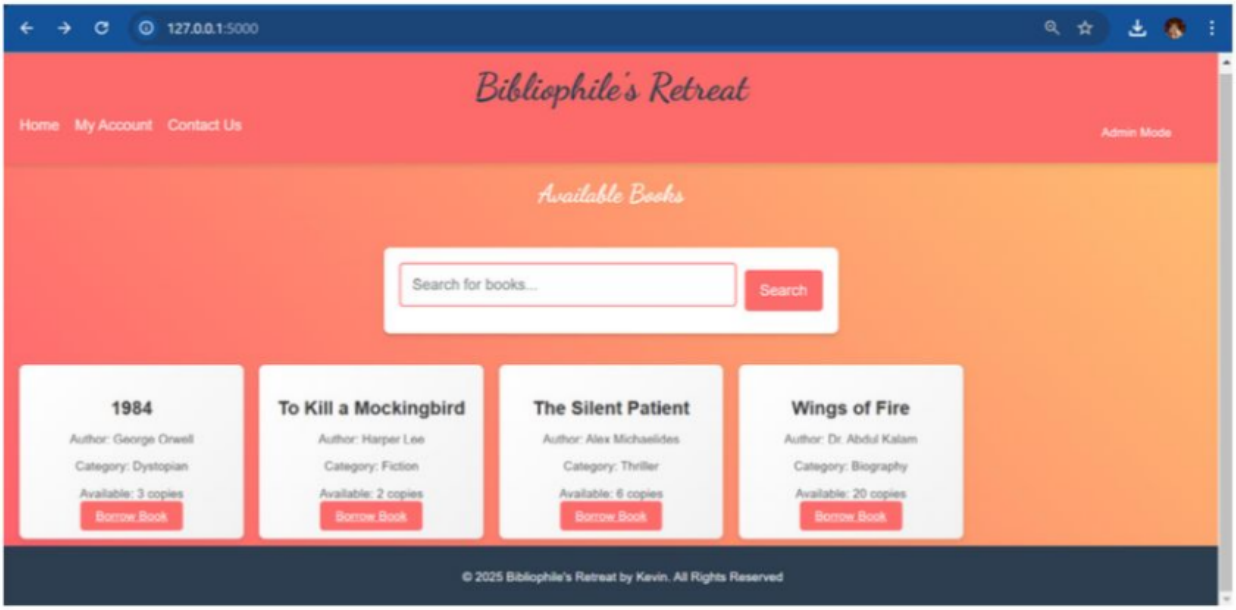
```python
from flask import Flask, render_template, request, redirect, url_for, flash #  type: ignore
import datetime

app = Flask(__name__)
app.secret_key = "your_secret_key_here"

# Sample Data: Books and Users (In-memory storage)
books = [
    {"id": 1, "title": "The Great Gatsby", "author": "F. Scott Fitzgerald", "category": "Fiction", "available": 5},
    {"id": 2, "title": "1984", "author": "George Orwell", "category": "Dystopian", "available": 3},
    {"id": 3, "title": "To Kill a Mockingbird", "author": "Harper Lee", "category": "Fiction", "available": 2},
    {"id": 4, "title": "The Silent Patient", "author": "Alex Michaelides", "category": "Thriller", "available": 6}
]

borrowed_books = []  # Track borrowed books
users = []  # User management (for simplicity)

@app.route("/admin", methods=["GET"])
def admin_dashboard():
    search_query = request.args.get('search', '').lower()  # Get search query from URL args, default to empty string

    # Filter books based on search query
    if search_query:
        filtered_books = [book for book in books if search_query in book['title'].lower() or
                          search_query in book['author'].lower() or search_query in book['category'].lower()]
    else:
        filtered_books = books  # Show all books if no search query

    return render_template("admin_dashboard.html", books=filtered_books, users=users)

# Add Book Route
@app.route("/admin/add", methods=["GET", "POST"])
def add_book():
    if request.method == "POST":
        title = request.form["title"]
        author = request.form["author"]
        category = request.form["category"]
        available = int(request.form["available"])

        new_book = {
            "id": len(books) + 1,  # Assign a new ID
            "title": title,
            "author": author,
            "category": category,
            "available": available
        }
        books.append(new_book)
        flash(f"Book '{title}' added successfully!", "success")
        return redirect(url_for("admin_dashboard"))
    return render_template("add_book.html")

# Update Book Route
@app.route("/admin/update/<int:book_id>", methods=["GET", "POST"])
def update_book(book_id):
    book = next((b for b in books if b["id"] == book_id), None)

    if not book:
        flash("Book not found.", "error")
        return redirect(url_for("admin_dashboard"))

    if request.method == "POST":
        # Update the book
        book["title"] = request.form["title"]
        book["author"] = request.form["author"]
```

```
        book["category"] = request.form["category"]
        book["available"] = int(request.form["available"])

        # Redirect back to the admin dashboard
        flash(f"Book '{book['title']}' updated successfully!", "success")
        return redirect(url_for("admin_dashboard"))

    return render_template("update_book.html", book=book)

# Remove Book Route
@app.route("/admin/remove/<int:book_id>")
def remove_book(book_id):
    global books
    books = [b for b in books if b["id"] != book_id]
    flash("Book removed successfully!", "success")
    return redirect(url_for("admin_dashboard"))

@app.route("/borrow/<int:book_id>")
def borrow_book(book_id):
    book = next((b for b in books if b["id"] == book_id), None)
    if book and book["available"] > 0:
        book["available"] -= 1
        borrowed_books.append({
            "book_id": book_id,
            "borrow_date": datetime.datetime.now(),
            "due_date": datetime.datetime.now() + datetime.timedelta(days=14)  # 14 days due
        })
        flash(f"Book '{book['title']}' successfully borrowed!", "success")  # Flash message
    else:
        flash("Sorry, this book is unavailable!", "error")
    return redirect(url_for("index"))

@app.route("/overdue")
def overdue_books():
    overdue = []
    for record in borrowed_books:
        book = next((b for b in books if b["id"] == record["book_id"]), None)
        if book and record["due_date"] < datetime.datetime.now():
            overdue.append({
                "book": book,
                "borrow_date": record["borrow_date"],
                "due_date": record["due_date"],
                "fine": (datetime.datetime.now() - record["due_date"]).days * 1  # $1 per day fine
            })
    return render_template("overdue_books.html", overdue=overdue)

@app.route("/")

def index():
    return render_template("index.html", books=books)
if __name__ == "__main__":
    app.run(debug=True)
```

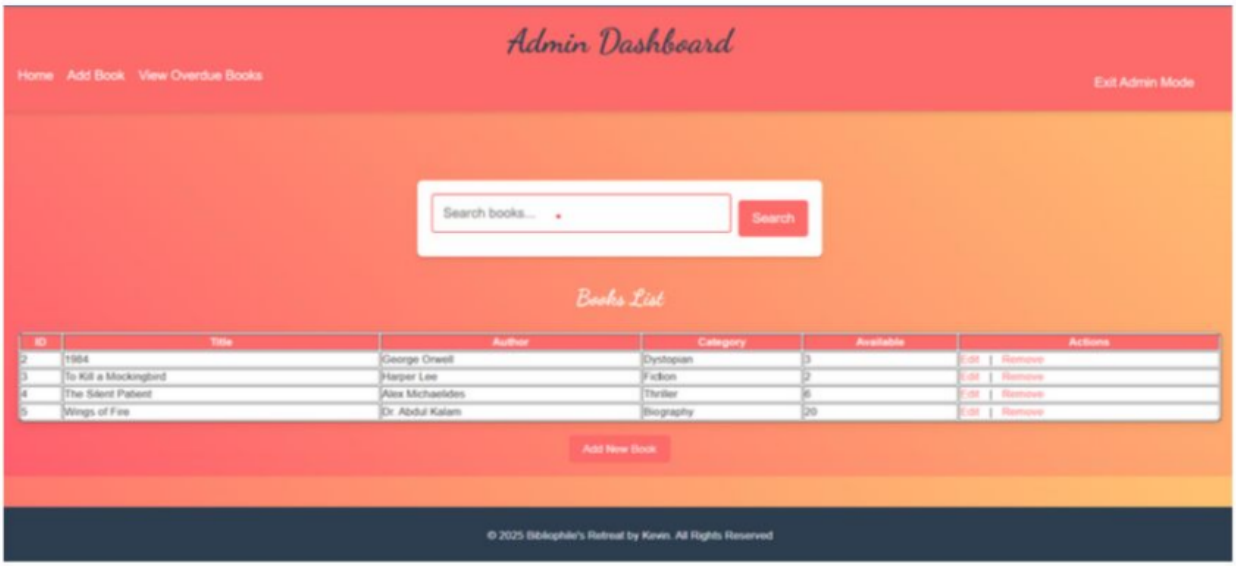
Screenshots and Demonstration

Below are some screenshots showcasing the key features of the Library Management System:

1. Home Page



2. Admin Dashboard



3. Search Functionality

Conclusion

The Library Management System provides an efficient way to manage books in a library setting. It simplifies book borrowing, returning, and searching while offering administrators complete control over book inventory. The system ensures a structured and organized approach to library management, making it a valuable tool for both users and administrators.