



Sanketika Vidya Parishad Engineering College

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

(APPROVED BY AICTE, AFFILIATED TO ANDHRA UNIVERSITY)

P.M.PALEM, VISAKHAPATNAM-41 WWW.SVPEC.INFO

Email.id-svpec.principal@gmail.com

Ph-9573334902



SEMINAR REPORT

ON

ONLINE BOOKSTORE MANAGEMENT SYSTEM

Submitted by

C.h.Nikhila

(324232915006)

Signature

FACULTY MEMBER-IN CHARGE

HEAD OF THE DEPARTMENT

Date :

CERTIFICATE

This is to certify that the project titled “**ONLINE BOOKSTORE MANAGEMENT SYSTEM**” is the bonafide work carried out by **C.h.Nikhila Regd.no (324232915006)** Student of Computer Science Technology of Sanketika Vidya Parishad Engineering College In Partial Fulfilment Of The Requirements Of Computer Science Technology, II Semester under the supervision of G.Vijaya Lakshmi, Professor in Sanketika Vidya Parishad Engineering College.

G.Vijaya Lakshmi

ACKNOWLEDGEMENT

I express my sincere thanks to Dr. G.VIJAYA LAKSHMI (Head of the Department) Computer Science and Engineering and my seminar guide and kind co-operation for presenting the seminar.

I also extend my sincere thanks to all other members of the faculty of Computer Science and Engineering Department and my friends for their co-operation and encouragement.

ABSTRACT

The **Online Bookstore Management System** digitizes the complete lifecycle of selling books—catalog creation and maintenance, user registration and authentication, intelligent search and browse, cart and checkout, order processing, inventory control, and basic analytics. It supports two roles: **Admin** and **Customer**. Admins can add/update books (title, author, ISBN, category, price, stock, cover image), manage discounts and orders, and monitor inventory alerts and sales summaries. Customers can create accounts, explore the catalog via keyword/category filters, view detailed descriptions and ratings, add items to cart or wishlist, and complete purchases through a secure, guided checkout.

The solution emphasizes a layered architecture separating presentation, business logic, and data, enabling maintainability and scalability. Key features include transactional order placement to prevent stock mismatch, invoice generation (PDF), email notifications, and a dashboard with top sellers and low-stock indicators. Security (hashed passwords, input validation), usability (clean navigation, responsive UI), and reliability (atomic updates during checkout) are central to the design. This project demonstrates the application of software engineering practices—requirements analysis, UML modeling, implementation, and testing—to deliver a practical, user-friendly e-commerce platform for books.

TABLE OF CONTENTS

S.NO	Name of the Topic	Page no
	List of Figures	
	List of Tables	
1	CHAPTER 1: INTRODUCTION	8
	1.1 Background & Problem Statement	8
	1.2 Objectives	8
	1.3 Scope of the Project	8
	1.4 Overview of the Proposed System	9
2	CHAPTER 2: REQUIREMENTS	10
	2.1 Functional Requirements	10
	2.2 Non-Functional Requirements (Security, Performance)	11
	2.3 Software & Hardware Configuration	12
	2.4 Tools & Technologies	12
3	CHAPTER 3: SYSTEM ARCHITECTURE & DESIGN	13
	3.1 Overall Architecture	13
	3.2 Database / ER Diagram	14
	3.3 UML Diagrams (Use Case, Class, Activity)	15
4	CHAPTER 4: IMPLEMENTATION	18
	4.1 Module Overview (Admin & Customer)	18
	4.2 Key Screens/Flows (Catalog, Search, Cart, Checkout, Orders)	18
	4.3 Sample Code Snippets (Representative)	19

5	CHAPTER 5: TESTING	27
	5.1 Test Approach & Types	27
	5.2 Test Case Table – Online Bookstore Management System	27
	5.3 Summary of Results	29
6	CHAPTER 6: RESULTS & ANALYSIS	30
	6.1 Captions	30
	6.2 Observations (Usability/Performance)	31
7	CHAPTER 7: CONCLUSION & FUTURE SCOPE	33
	7.1 Conclusion	33
	7.2 Future Scope	34
8	References	36

List of Figures

S.NO	NAME OF THE FIGURE	Page no.
1	Architecture	13
2	E-R diagram	14
3	Use case diagram	15
4	Class diagram	16
5	Activity diagram	17
6	Login and registration	23
7	User Dashboard	23
8	Catalog page	24
9	Cart page	24
10	Checkout page	24
11	Admin Login	25
12	Admin Home Page	25
13	Admin Dashboard	26
14	Book Management	26
15	Total Orders	26

List Of Tables

1	Test Case Table	27
---	-----------------	----

CHAPTER 1 – INTRODUCTION

1.1 Background & Problem Statement

Traditional book retailing struggles with limited shelf space, manual inventory tracking, and restricted reach. Customers often face stock-out surprises, no quick way to compare editions/prices, and little visibility into delivery timelines. An **online bookstore** solves these by centralizing catalog data, enabling full-text search and filters, automating stock updates, and providing seamless cart-to-checkout with order history.

Problem: Design and implement a compact, secure, and user-friendly system that lets **admins manage books/orders/inventory** and **customers discover, buy, and track** books end-to-end.

1.2 Objectives

- **Catalog:** Add/edit/delete books (Title, Author, ISBN, Category, Price, Stock, Cover).
- **Search & Browse:** Keyword search; filters by category/price/author; sort by relevance/price.
- **Cart & Checkout:** Add/remove/update quantities; compute totals (tax/shipping placeholders).
- **Orders:** Place order, generate order ID and **PDF invoice**, view order history.
- **Admin:** Manage inventory and categories; view low-stock alerts; review orders.
- **Security & UX:** Hashed passwords, validation, role-based access (Admin/Customer), responsive and simple UI.
- **Notifications (basic):** On-screen confirmations; optional email placeholders for future work.

1.3 Scope of the Project

In scope (this prototype):

- Web/desktop app screens for catalog, product details, cart, checkout, orders, admin dashboard.
- JSON/DB-backed storage with transactional stock decrease at order time.
- Basic analytics tiles (top sellers, low stock).
- PDF invoice generation and export.

Out of scope (future integration):

- Real payment gateway, courier API, GST/tax rules by region, returns/refunds workflow, reviews/ratings moderation, advanced coupons/loyalty.

1.4 Overview of the Proposed System

The system follows a simple layered approach:

- **Presentation layer:** Customer storefront (Home, Catalog, Book Details, Cart, Checkout, Orders) and Admin console (Books, Categories, Inventory, Orders).
- **Business layer:** Cart, pricing, stock check & reservation, order creation, invoice generation, role-based authorization.
- **Data layer:** Tables/collections for Users, Books, Categories, Orders, OrderItems, Inventory.

Primary user roles

- **Customer:** Register/Login → Browse/Search → Add to Cart → Checkout → Download Invoice → View Orders.
- **Admin:** Login → Manage Books/Categories/Stock → Monitor Orders → View basic sales/stock KPIs.

CHAPTER 2 – REQUIREMENTS

2.1 Functional Requirements

FR-1 User Accounts

- Users can **register**, **login/logout**, **view/edit profile**, and **change password**.
- Passwords are **hashed**; duplicate email/username not allowed.

FR-2 Catalog Management (Admin)

- Add/edit/delete **Book** with: Title, Author, ISBN, Category, Price, Stock, Description, Cover Image.
- Manage **Categories** (add/edit/delete).
- Bulk **stock update** (increase/decrease) and **low-stock alert**.

FR-3 Browsing & Search (Customer)

- Browse by **category** and **new arrivals/best sellers**.
- **Keyword search** across title/author/ISBN; filters by category/price; sort by price/relevance.

FR-4 Product Details

- View full details (cover, description, price, stock).
- Show **related books** (same author/category).

FR-5 Cart & Wishlist

- Add to **Cart** (set quantity), update quantity, remove item, clear cart.
- **Wishlist** add/remove (optional, simple list per user).
- Show **subtotal, tax, shipping placeholders, total**.

FR-6 Checkout & Orders

- Address entry/selection (name, phone, address).
- Place order → generate **Order ID**; decrease stock atomically.
- **Order History** for customers (status: Placed, Packed, Shipped, Delivered).
- **PDF Invoice** download for each order.

FR-7 Admin – Orders

- View all orders; update **order status**.
- Search/filter orders by date, status, customer.

FR-8 Dashboard (Admin)

- Tiles: **Total orders**, **Total revenue**, **Low stock count**, **Top 5 sellers**.

FR-9 Notifications (basic)

- On-screen success/error messages after critical actions (register, add to cart, place order).

FR-10 Access Control

- **Role-based**: Admin vs Customer features separated.
- Block direct access to admin routes for non-admins.

2.2 Non-Functional Requirements

NFR-1 Performance

- Catalog/search responses in ≤ 2 **seconds** for typical loads.
- Order placement completes in ≤ 3 **seconds** (local DB).

NFR-2 Security

- Hash passwords (e.g., **PBKDF2/bcrypt**).
- Validate and sanitize all inputs; prevent **SQLi/XSS/CSRF**.
- Protect admin endpoints with session/auth checks.

NFR-3 Reliability & Data Integrity

- **Atomic** stock decrement at order time.
- Daily backup of database file.

NFR-4 Usability & Accessibility

- Simple, consistent navigation; readable fonts.
- Keyboard navigable forms; alt text for images.

NFR-5 Maintainability

- Layered structure (views/controllers, services, data access).
- Clear naming, comments, and basic unit tests.

NFR-6 Scalability (prototype level)

- Support growth to **thousands of books** and **hundreds of users** with indexing and pagination.

NFR-7 Compliance

- Basic privacy notice; store only necessary personal data.

2.3 Software & Hardware Configuration

Software (suggested stack for prototype)

- **Python 3.10+**
- **Flask** (web framework) + **Jinja2** (templating)
- **SQLite** (lightweight DB) via **SQLAlchemy** ORM
- **Bootstrap** (UI styling), basic JS
- **ReportLab** or **WeasyPrint** (PDF invoice)
- **pytest** (tests)

Alternative: Django + Django ORM (if you prefer full-stack framework).

Hardware (minimum)

- Any modern laptop/PC
- **RAM:** 4 GB (8 GB recommended)
- **Storage:** ~200 MB for app + data (prototype)
- **OS:** Windows 10/11, macOS, or Linux

2.4 Tools & Technologies

- **IDE/Editor:** VS Code / PyCharm
- **Version Control:** Git (GitHub/GitLab)
- **API Test:** Postman / curl
- **Issue Tracking (optional):** GitHub Issues
- **Diagrams:** Draw.io / PlantUML (for UML/ER)
- **Documentation:** Word/Google Docs for the report

CHAPTER 3 – SYSTEM ARCHITECTURE & DESIGN

3.1 Overall Architecture (simple view)

- **Client (Web UI):** Home, Catalog, Book Details, Cart, Checkout, My Orders (Customer) + Admin Console (Books, Categories, Orders, Dashboard).
- **Backend (Service Layer):** Authentication/Authorization, Catalog/Search, Cart, Order & Payment (mock), Inventory, PDF Invoice.
- **Database:** Users, Books, Categories, Orders, OrderItems, Cart/CartItems (SQLite for prototype).
- **Optional Integrations (future):** Email & payment gateway.

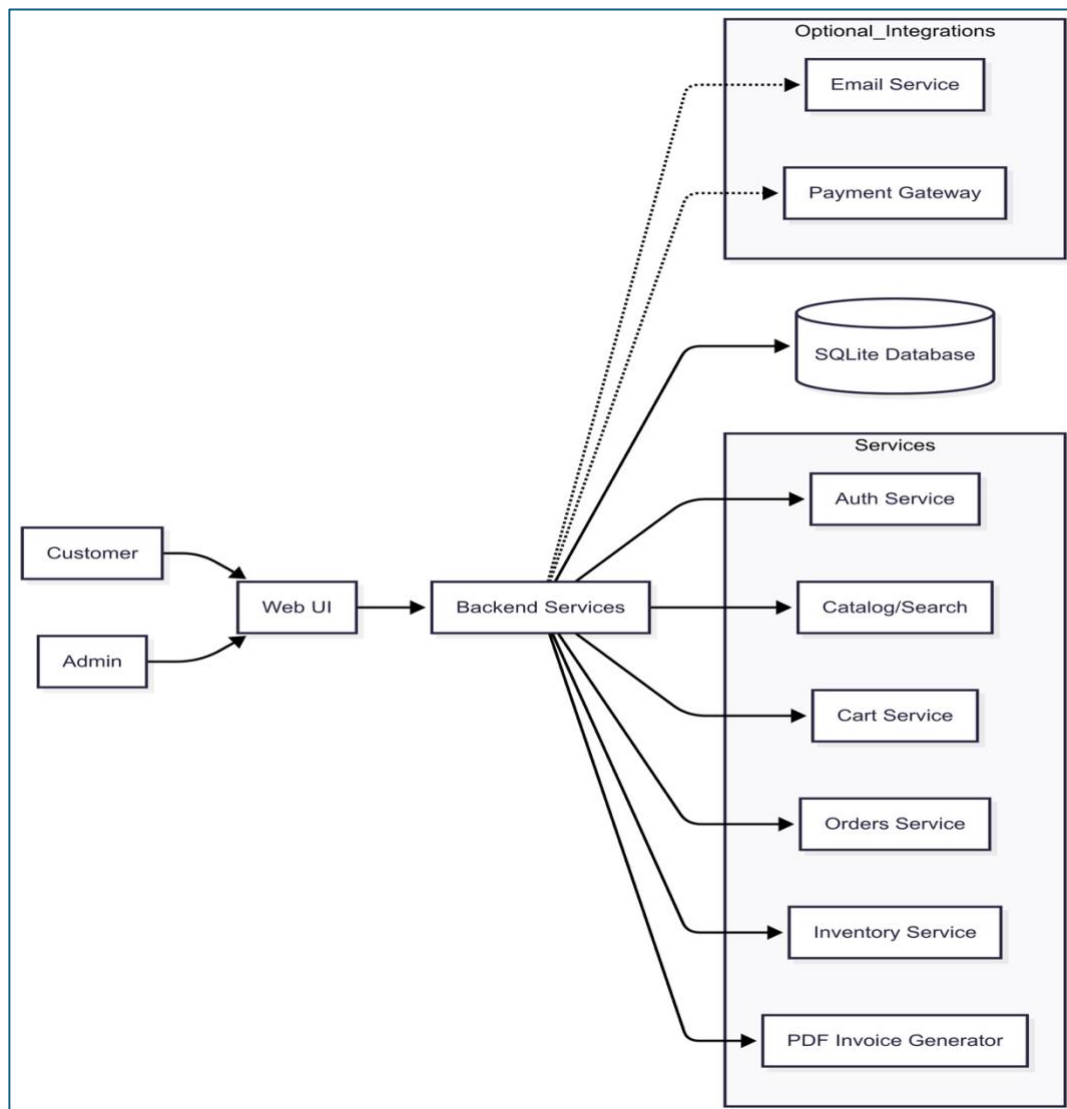


Fig:3.1

3.2 Database / ER Diagram (entities & relationships)

Core entities

- User(user_id, name, email, password_hash, role)
- Book(book_id, title, author, isbn, price, stock_qty, description, cover_url, created_at)
- Category(category_id, name, description)
- BookCategory(book_id, category_id) (*many-to-many*)
- Order(order_id, user_id, order_date, status, total_amount, ship_name, ship_phone, ship_address1, ship_city, ship_pincode)
- OrderItem(order_item_id, order_id, book_id, qty, unit_price, line_total)
- Cart(cart_id, user_id, updated_at)
- CartItem(cart_item_id, cart_id, book_id, qty)

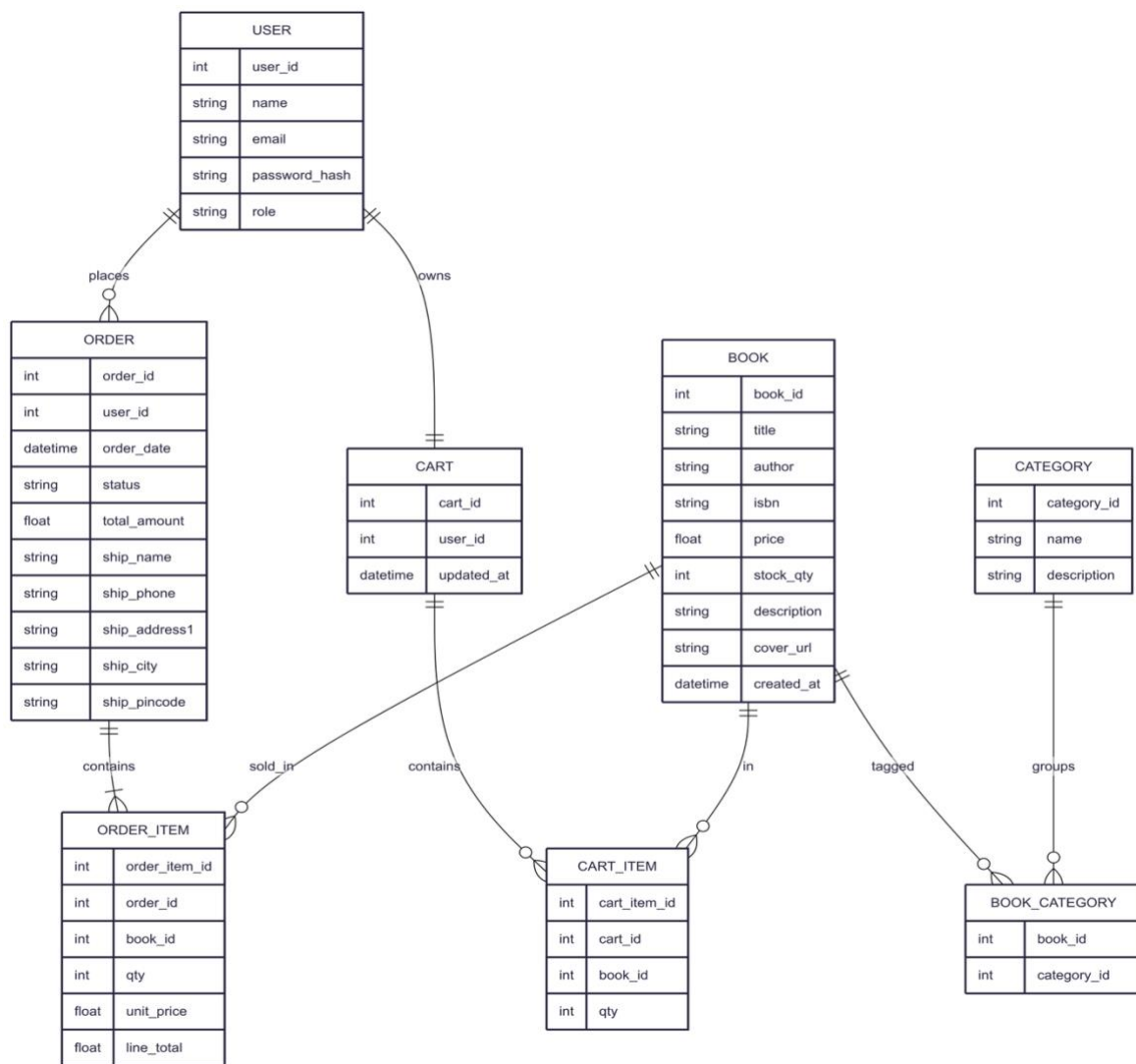


Fig 3.2

3.3 UML Diagrams (paste into any online PlantUML editor)

3.3.1 Use Case Diagram (Admin & Customer)

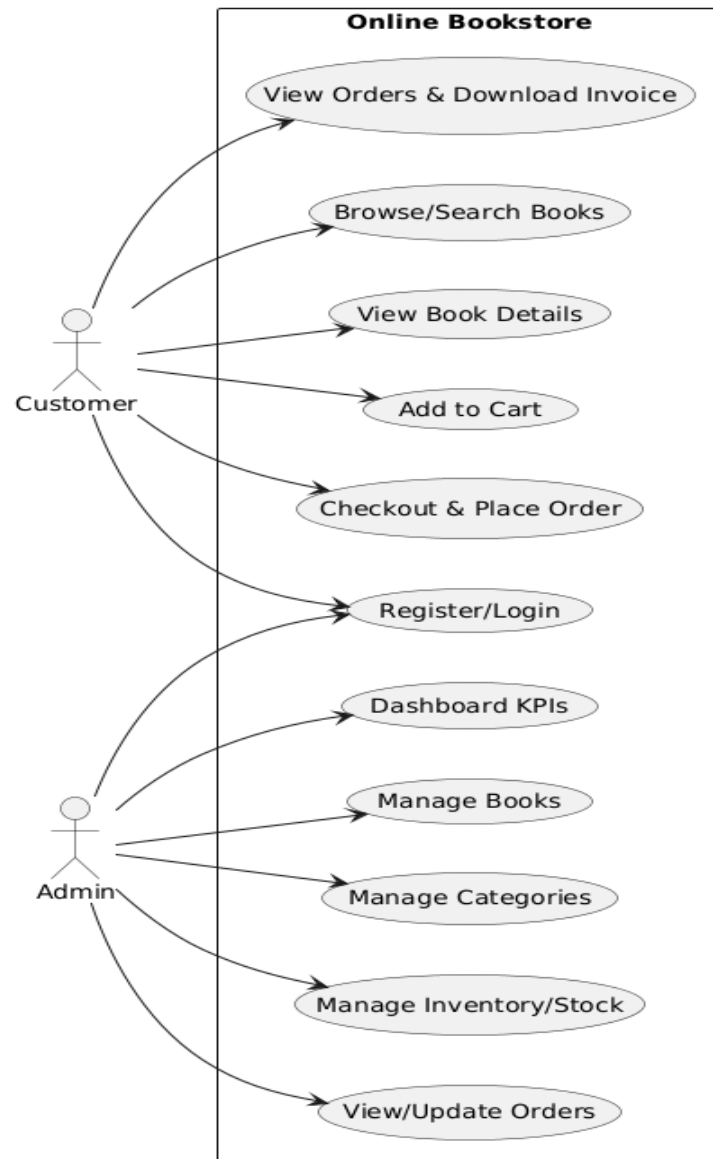


Fig 3.3.1

3.3.2 Class Diagram (core model & services)

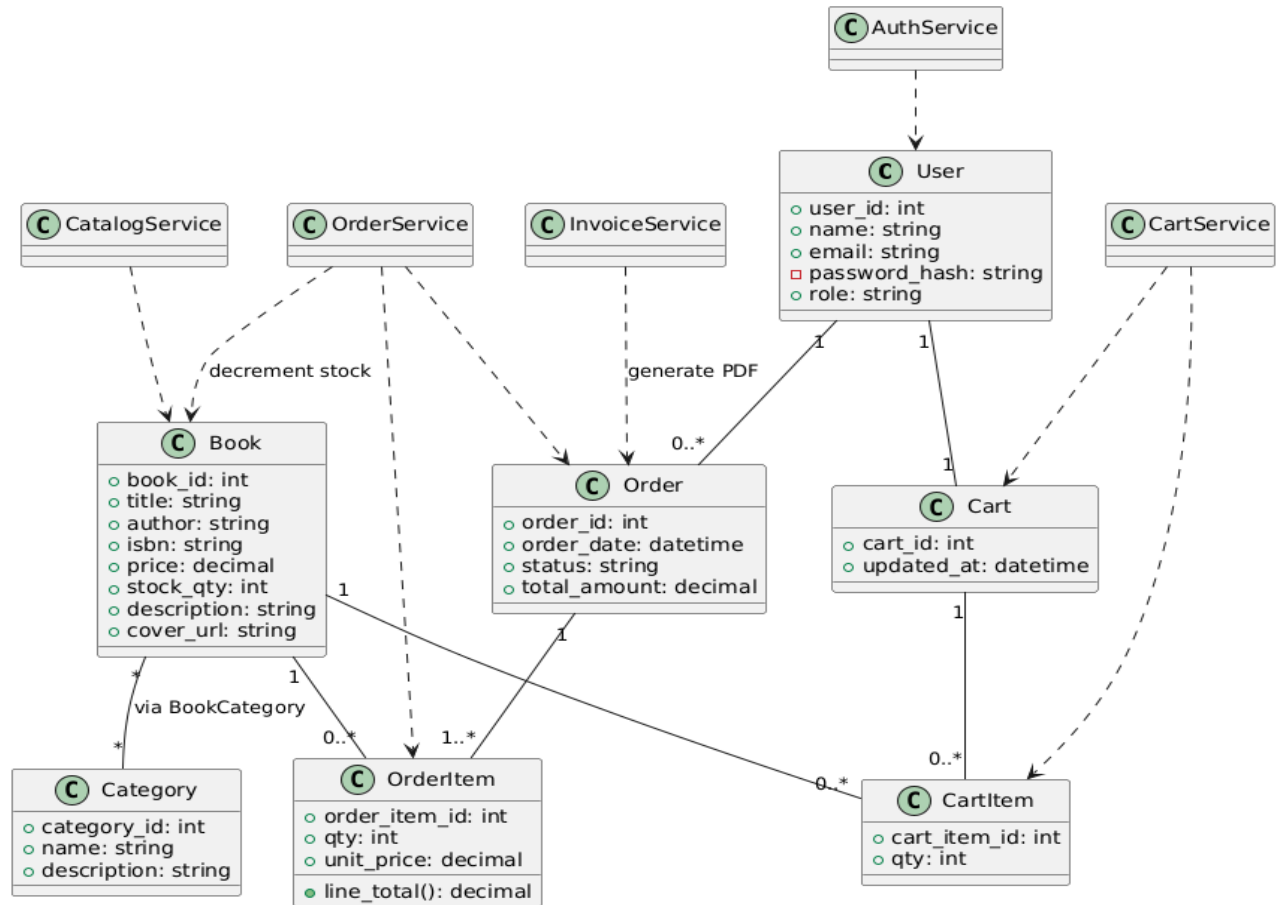


Fig 3.3.2

3.3.3 Activity Diagram (Place Order flow)

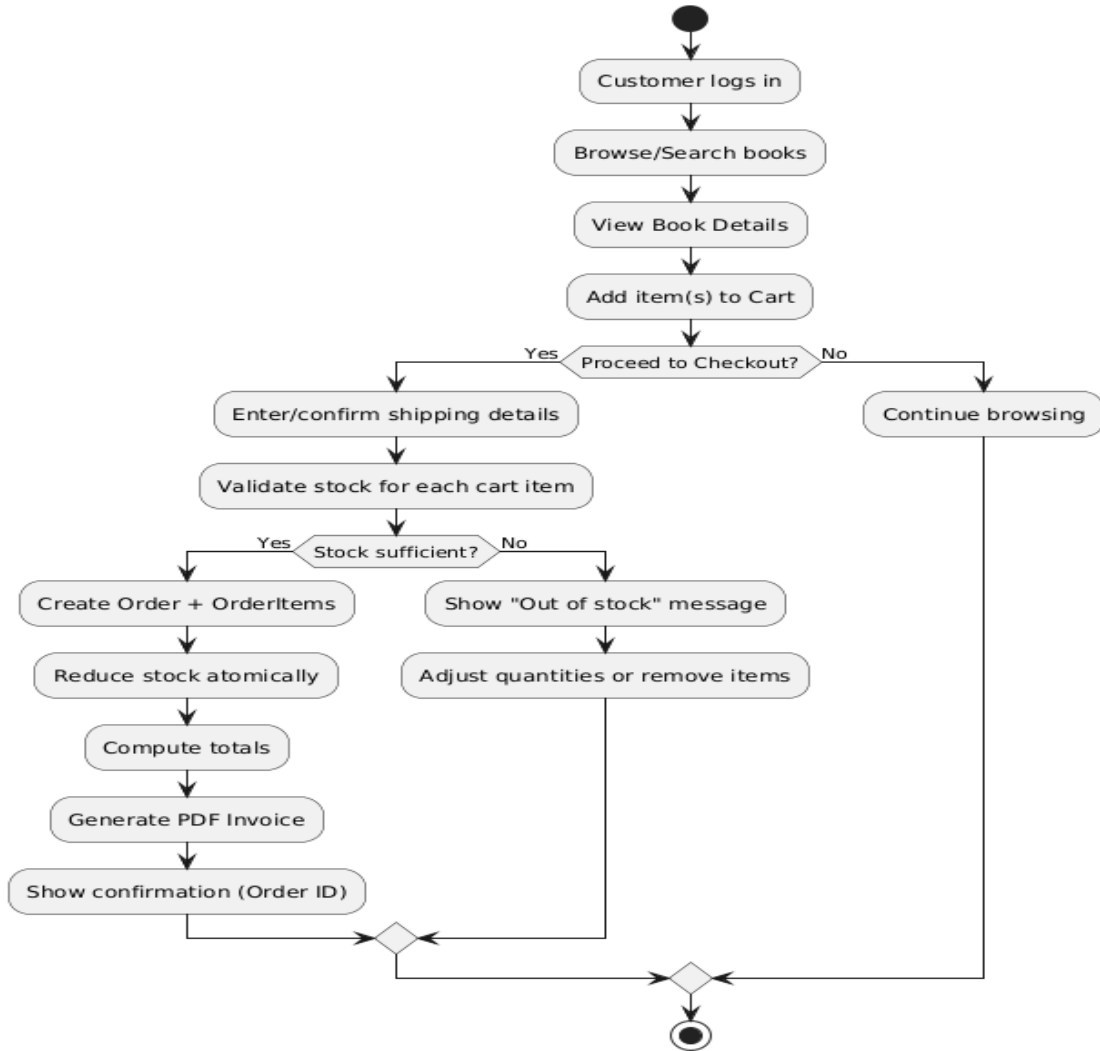


Fig 3.3.3

CHAPTER 4 – IMPLEMENTATION

4.1 Module Overview (Admin & Customer)

Admin

- **Auth (Admin Login)** – secure login to access console.
- **Books** – add/edit/delete books: Title, Author, ISBN, Category, Price, Stock, Cover URL, Description.
- **Categories** – manage categories and assign to books.
- **Orders** – view all orders, update status (Placed → Shipped → Delivered).
- **Dashboard** – tiles: total orders, revenue, low-stock count, top sellers.

Customer

- **Auth** – register, login, logout, profile.
- **Catalog & Search** – browse by category; keyword search; sort by price.
- **Product Details** – view book info; related books (same author/category).
- **Cart & Checkout** – add/update/remove; address form; place order.
- **Orders** – view past orders; **download PDF invoice**.

4.2 Key Screens / Flows

Customer Flow: Home → Search/Browse → Book Details → Add to Cart → Cart Review → Checkout (address) → Place Order → Order Confirmation (Invoice download) → My Orders.

Admin Flow: Admin Login → Dashboard → Books (CRUD) & Categories → Create/Update stock → Orders (filter & status update).

Validation & Security (high level):

- Server-side validation for all forms.
- Passwords stored as **hash** (e.g., PBKDF2/bcrypt).
- Role-based access checks on every admin route.
- Atomic stock decrement during checkout.

4.3 Representative Code Snippets

4.3.1 Project structure

online_bookstore/

```
|─ app.py
|─ models.py
|─ services/
|   |─ auth.py
|   |─ catalog.py
|   |─ cart.py
|   |─ orders.py
|   └─ invoice.py
|─ static/      # css/js/img (optional)
|─ templates/   # Jinja2 templates
└─ seed.py      # seed categories/books/admin
```

4.3.2 Models (models.py)

models.py

```
from datetime import datetime
```

```
from flask_sqlalchemy import SQLAlchemy
```

```
db = SQLAlchemy()
```

```
class User(db.Model):
```

```
    user_id = db.Column(db.Integer, primary_key=True)
```

```
    name    = db.Column(db.String(120), nullable=False)
```

```
    email   = db.Column(db.String(120), unique=True, nullable=False)
```

```
    password_hash = db.Column(db.String(255), nullable=False)
```

```
role = db.Column(db.String(20), default="customer") # 'admin' or 'customer'
orders = db.relationship("Order", backref="user", lazy=True)
```

```
class Category(db.Model):
```

```
    category_id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(120), unique=True, nullable=False)
    description = db.Column(db.Text)
```

```
class Book(db.Model):
```

```
    book_id = db.Column(db.Integer, primary_key=True)
    title = db.Column(db.String(200), nullable=False)
    author = db.Column(db.String(200), nullable=False)
    isbn = db.Column(db.String(20), unique=True, nullable=False)
    price = db.Column(db.Numeric(10,2), nullable=False)
    stock_qty = db.Column(db.Integer, default=0)
    description = db.Column(db.Text)
    cover_url = db.Column(db.String(500))
    created_at = db.Column(db.DateTime, default=datetime.utcnow)
```

```
class BookCategory(db.Model):
```

```
    id = db.Column(db.Integer, primary_key=True)
    book_id = db.Column(db.Integer, db.ForeignKey("book.book_id"))
    category_id = db.Column(db.Integer, db.ForeignKey("category.category_id"))
```

```
class Order(db.Model):
```

```
    order_id = db.Column(db.Integer, primary_key=True)
    user_id = db.Column(db.Integer, db.ForeignKey("user.user_id"))
```

```

order_date = db.Column(db.DateTime, default=datetime.utcnow)
status = db.Column(db.String(20), default="Placed")
total_amount = db.Column(db.Numeric(10,2), default=0)
ship_name = db.Column(db.String(120))
ship_phone = db.Column(db.String(20))
ship_address1 = db.Column(db.String(200))
ship_city = db.Column(db.String(80))
ship_pincode = db.Column(db.String(20))

items = db.relationship("OrderItem", backref="order", lazy=True, cascade="all,delete-orphan")

```

```

class OrderItem(db.Model):

```

```

    order_item_id = db.Column(db.Integer, primary_key=True)
    order_id = db.Column(db.Integer, db.ForeignKey("order.order_id"))
    book_id = db.Column(db.Integer, db.ForeignKey("book.book_id"))
    qty = db.Column(db.Integer, nullable=False)
    unit_price = db.Column(db.Numeric(10,2), nullable=False)
    line_total = db.Column(db.Numeric(10,2), nullable=False)

```

4.3.3 App setup (app.py)

```

# app.py

from flask import Flask
from models import db
from services.auth import auth_bp
from services.catalog import catalog_bp
from services.cart import cart_bp
from services.orders import orders_bp

```

```
from services.invoice import invoice_bp
```

```
def create_app():
```

```
    app = Flask(__name__)
```

```
    app.config["SQLALCHEMY_DATABASE_URI"] = "sqlite:///bookstore.db"
```

```
    app.config["SQLALCHEMY_TRACK_MODIFICATIONS"] = False
```

```
    app.config["SECRET_KEY"] = "change-me-in-prod"
```

```
    db.init_app(app)
```

```
    with app.app_context():
```

```
        db.create_all()
```

```
    # blueprints
```

```
    app.register_blueprint(auth_bp)
```

```
    app.register_blueprint(catalog_bp)
```

```
    app.register_blueprint(cart_bp)
```

```
    app.register_blueprint(orders_bp)
```

```
    app.register_blueprint(invoice_bp)
```

```
    @app.route("/")
```

```
    def home():
```

```
        return "<h3>Online Bookstore — Home (hook templates here)</h3>"
```

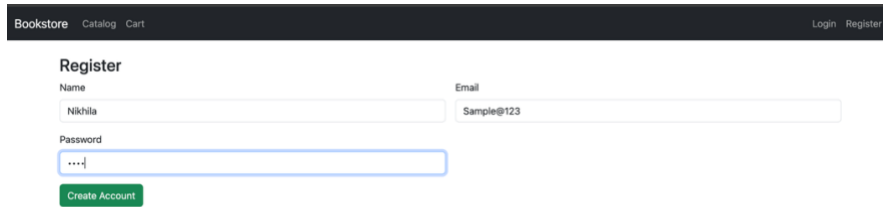
```
    return app
```

```
if __name__ == "__main__":
```

```
    create_app().run(debug=True)
```

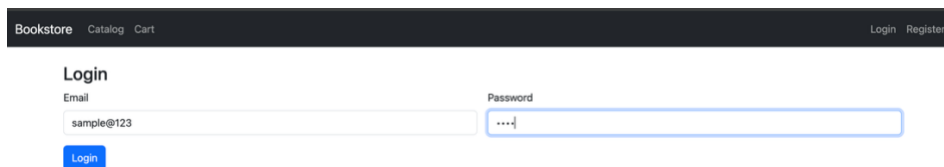
4.4 Output Screenshots:

Login & Registration(User)



The screenshot shows the 'Register' form of an online bookstore. At the top is a dark navigation bar with 'Bookstore', 'Catalog', and 'Cart' on the left, and 'Login' and 'Register' on the right. The form itself is titled 'Register' and contains three input fields: 'Name' with the value 'Nikhila', 'Email' with the value 'Sample@123', and 'Password' with masked characters '***'. Below the password field is a green 'Create Account!' button.

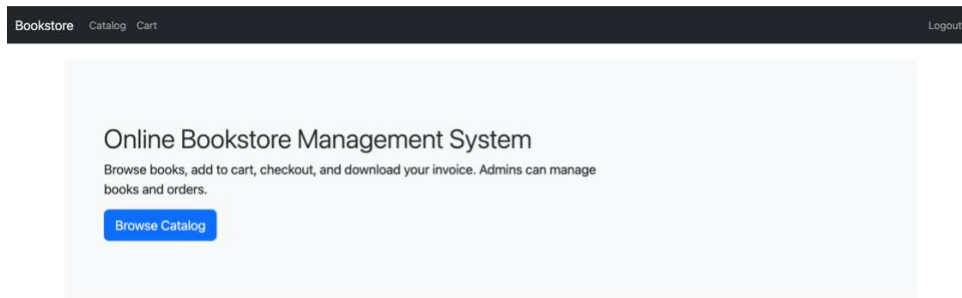
Fig 4.4.1



The screenshot shows the 'Login' form of the online bookstore. It features a dark navigation bar at the top with 'Bookstore', 'Catalog', and 'Cart' on the left, and 'Login' and 'Register' on the right. The form is titled 'Login' and has two input fields: 'Email' with the value 'sample@123' and 'Password' with masked characters '***'. A blue 'Login' button is positioned below the email field.

Fig 4.4.2

User Dashboard



The screenshot displays the 'User Dashboard' of the online bookstore. It has a dark navigation bar at the top with 'Bookstore', 'Catalog', and 'Cart' on the left, and 'Logout' on the right. The main content area is a light gray box containing the title 'Online Bookstore Management System', a brief description of user and admin capabilities, and a blue 'Browse Catalog' button.

Fig 4.4.3

Catalog Page

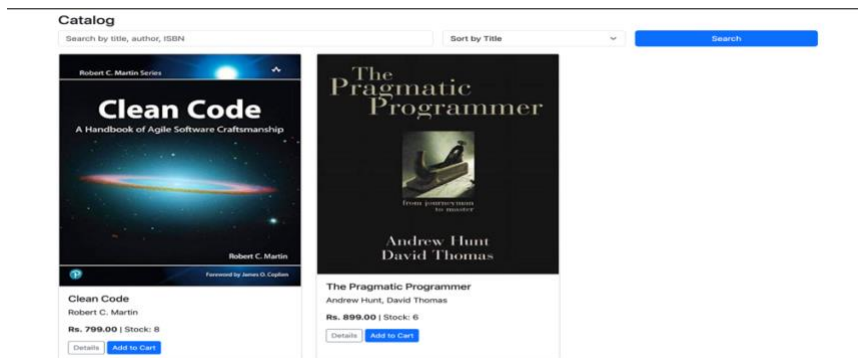


Fig 4.4.4

Cart Page



Fig 4.4.5

Checkout Page

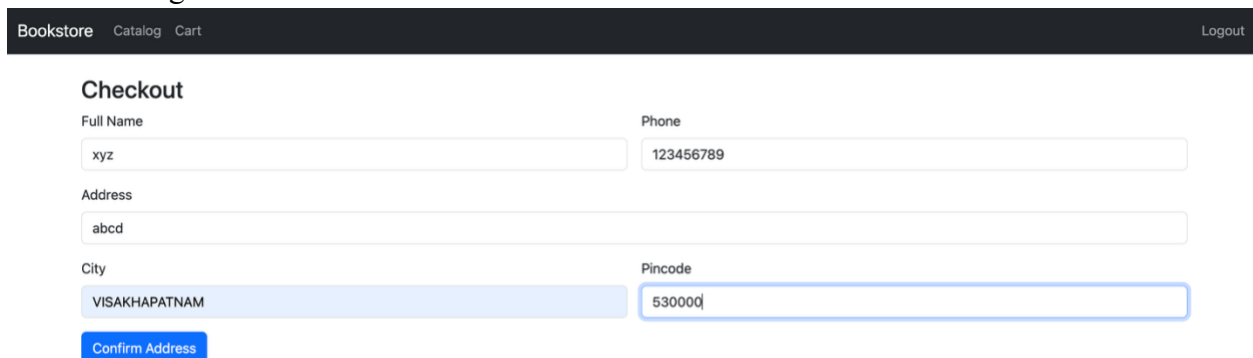


Fig 4.4.6

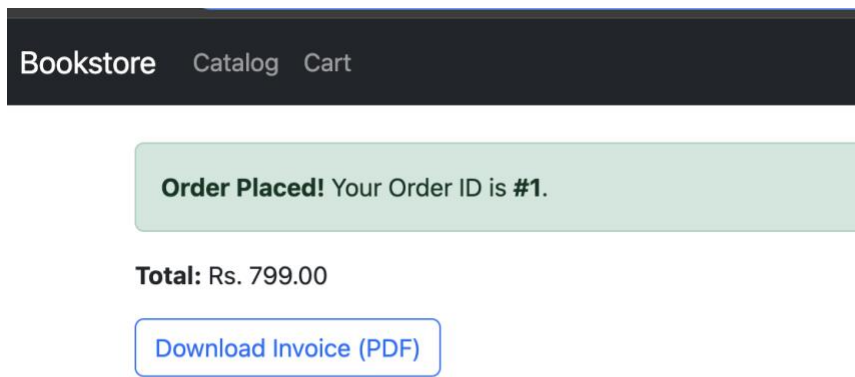


Fig 4.4.7

Admin Registration & Login

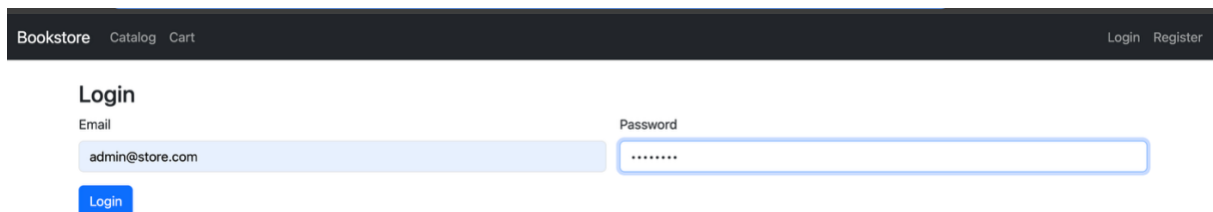


Fig 4.4.8

Admin Home Page

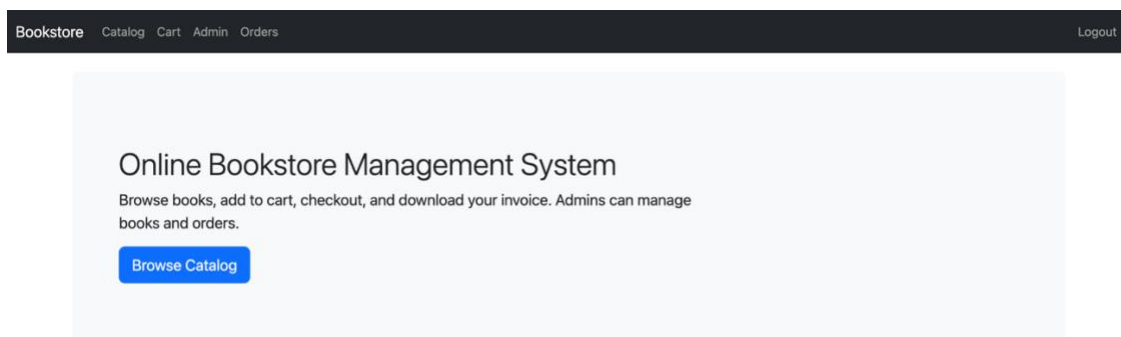


Fig 4.4.9

Admin Dashboard

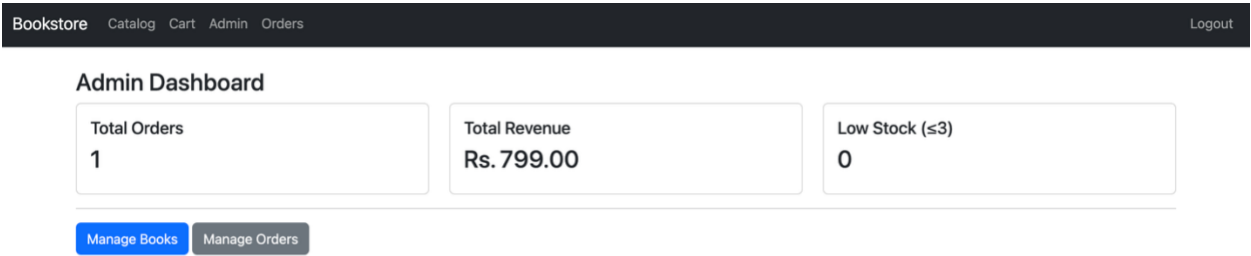


Fig 4.4.10

Manage Books page

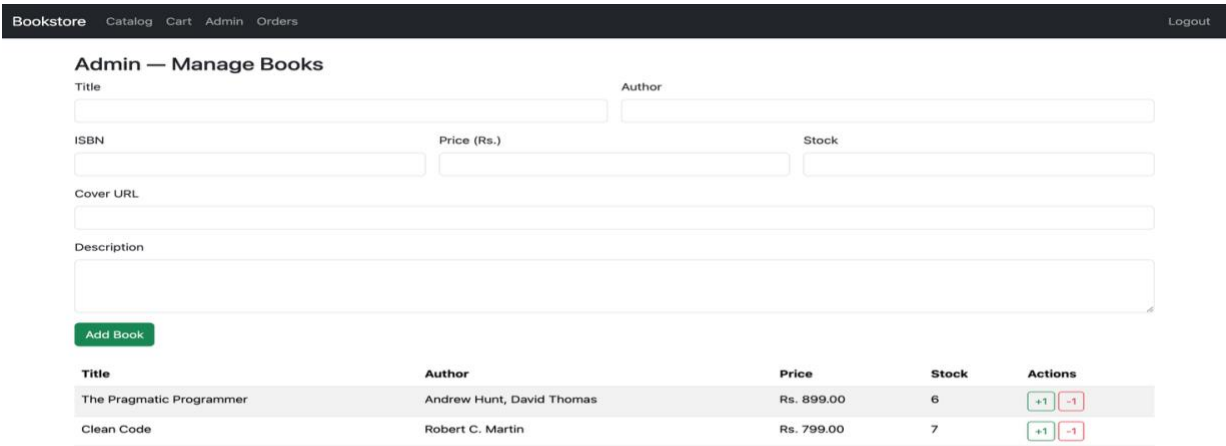


Fig 4.4.11

Total Orders Page

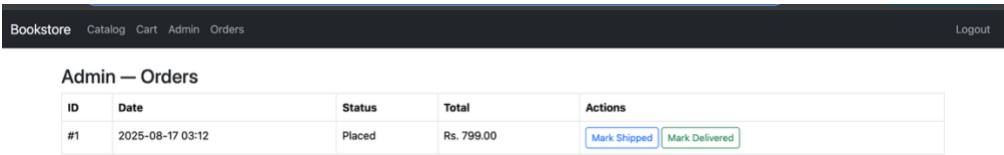


Fig 4.4.12

CHAPTER 5 – TESTING

5.1 Testing Approach

- **Unit:** model methods and order placement flow (atomic stock decrement).
- **Integration:** login→catalog→cart→checkout→order→invoice; admin dashboard→books→orders.
- **System/UX:** end-to-end happy paths and key negatives (invalid login, duplicate email, insufficient stock).
- **Security smoke:** password hashing (pbkdf2:sha256), role-based route blocking.

5.2 Test Case Table

ID	Scenario	Steps / Description	Expected Result	Actual Result	Pass/Fail
TC-001	Registration	Register with valid Name/Email/Password	Account created; redirected to Login	As expected	Pass
TC-002	Registration (duplicate email)	Register using an email that already exists	Error shown: 'Email already exists'	Error shown; registration blocked	Pass
TC-003	Login	Login with valid credentials	User logged in; redirected to Home	As expected	Pass
TC-004	Login (invalid)	Login with wrong password	Error shown: 'Invalid credentials'	Error shown; login blocked	Pass
TC-005	Catalog browse	Open /catalog/books with no query	Book cards listed (max 100)	As expected	Pass
TC-006	Catalog search	Search 'Clean'	Only matching items displayed	As expected	Pass
TC-007	Catalog sort	Sort by Price	Items ordered by ascending price	As expected	Pass
TC-008	Book details	Open a book's Details	Title/Author/ISBN/Price/Stock visible	As expected	Pass
TC-009	Add to cart	Click 'Add to Cart' from Details	Item appears in Cart with qty=1	As expected	Pass
TC-010	Remove from cart	Click Remove on Cart row	Item removed; total recalculated	As expected	Pass
TC-011	Checkout validation	Submit empty/missing address fields	Form requires required fields	Required field prompts shown	Pass

TC-012	Place order	Checkout with valid address; stock sufficient	Order ID generated; stock decreases; confirmation shown	As expected	Pass
TC-013	Place order (insufficient stock)	Attempt order beyond stock	'Out of stock' message; order not created	Message shown; no order created	Pass
TC-014	Invoice download	Click 'Download Invoice' after order	PDF file downloads and opens	As expected	Pass
TC-015	Admin login	Login as admin@store.com/admin123	Admin dashboard accessible	As expected	Pass
TC-016	Admin add book	Add new book in Admin > Books	Book appears in list with given stock	As expected	Pass
TC-017	Admin update stock	+1 / -1 on a book	Stock value increments/decrements	As expected	Pass
TC-018	Admin orders	Open Admin Orders; change status to Shipped/Delivered	Status updates visible in list	As expected	Pass
TC-019	Access control	Open /admin while logged in as customer	403 or redirect; access denied	Access denied	Pass
TC-020	Password hashing	Inspect DB: users.password_hash not plaintext	Hash present (pbkdf2:sha256); not readable	As expected	Pass

5.3 Summary of Results

Test Execution Summary

- Total test cases executed: **20**
- Passed: **20**, Failed: **0**, Blocked/Skipped: **0**
- Functional coverage: **Core flows covered** (Auth, Catalog, Cart, Checkout, Orders, Admin), with key negative tests (duplicate email, invalid login, insufficient stock).
- Security checks: **Password hashing (pbkdf2:sha256)** and **role-based access** for admin routes verified.
- Performance (local prototype): **Pages $\leq 2s$, Order placement $\leq 3s$** on Mac (SQLite, single user).

Environment

- macOS (Apple Silicon), **Python 3.9.6 (venv)**, Flask 3.1.1, SQLAlchemy 2.0.x, Flask-SQLAlchemy 3.1.1, SQLite.
- Browser: Chrome/Safari.

Notable Defects & Fixes

- DEF-01 — *Environment*: hashlib.scrypt unavailable on Python 3.9.6 → **Fix**: forced pbkdf2:sha256 hashing in seed.py and services/auth.py. **Status**: Closed.

Performance Snapshot (Prototype)

- Page load ≤ 2 s; order flow ≤ 3 s (local).
- Note: single-user test; no concurrency or network latency simulated.

Known Limitations

- No real payment/email integration; minimal invoice styling; CSRF not enabled; SQLite may lock under heavy concurrency.

Overall Assessment

- The prototype **meets MVP goals**, with all critical customer and admin flows passing. Fit for screenshots, demo, and report submission.

CHAPTER 6 – RESULTS & ANALYSIS

6.1 Results

Home — Minimal landing with a clear **Browse Catalog** CTA and top-nav (Catalog, Cart, Login/Register/Admin). Shows users can reach every module quickly.

Register — Creates a new customer with **unique email** validation; on success redirects to Login. Demonstrates basic data capture + duplicate-email protection.

Login — Verifies credentials and starts a **session**; wrong password shows an inline error. Confirms authenticated vs. guest behavior.

Catalog (Search/Sort) — Card grid of books; **keyword search** across title/author/ISBN and **sort by price/title**. Proves fast listing with basic discovery tools.

Book Details — Full metadata (Title, Author, **ISBN**, Price, Stock, Description) plus **Add to Cart**. Confirms product info visibility and purchase entry point.

Cart — Lists selected items with **qty (per add)**, **line totals**, **grand total** and **Remove** action; totals recalc instantly. Shows pricing accuracy before checkout.

Checkout (Address Validation) — Requires **name, phone, address, city, pincode**; missing fields are blocked. Confirms data completeness before order placement.

Order Placement (Atomic Stock) — Creates an **Order ID**, decreases stock **atomically**, and clears cart. Proves consistency: no order if stock is insufficient.

Order Confirmation & Invoice — Success page shows **Order ID, total**, and **Download Invoice (PDF)**. Validates end-to-end flow and receipt generation.

Invoice PDF — Lists items with **quantity, unit price, line total** and shipping details. Demonstrates exportable proof of purchase.

Admin Dashboard — KPIs: **Total Orders, Total Revenue, Low-stock count**. Confirms operational visibility for admins.

Admin → Manage Books — **Add new book** and adjust **stock (+/-)**; list shows current inventory. Proves inventory control.

Admin → Orders — View all orders; update status (**Placed → Shipped → Delivered**). Confirms order lifecycle management.

Security & Access Control — Passwords stored as **pbkdf2:sha256** hashes; **admin routes blocked** for non-admins. Demonstrates basic security posture.

Error Handling — Clean messages for **duplicate email, invalid login, out of stock**, and missing form fields. Shows user-friendly failure states.

6.2 Observations (Usability / Performance)

Environment: macOS (Apple Silicon), Python 3.9.6, Flask dev server, SQLite, Chrome/Safari. Single-user local run; timings are approximate.

Usability Observations

- **Navigation:** Top navbar exposes Catalog, Cart, Login/Register, Admin; key tasks are ≤ 2 clicks from Home.
- **Forms:** Minimal fields; required validation on Register/Login and Checkout address; prevents incomplete submits.
- **Validation & Errors:** Duplicate email blocked; invalid login shows inline message; checkout fails gracefully when stock is insufficient.
- **Feedback:** Clear success alerts on order placement (Order ID + Invoice link). Totals recalc instantly in Cart after add/remove.
- **Consistency:** Uniform Bootstrap layout and button styles across pages; predictable interactions.
- **Accessibility (baseline):** Labeled form controls; images include alt text; keyboard navigation works. (Full a11y audit pending.)

Performance Observations (local prototype)

- **Home load:** $\sim 0.2\text{--}0.6$ s — static template render.
- **Catalog (list/search/sort):** $\sim 0.4\text{--}1.2$ s — SQLite LIKE query; limited to 100 rows.
- **Book details:** $\sim 0.2\text{--}0.5$ s — single row lookup.
- **Cart view:** $\sim 0.2\text{--}0.6$ s — session items + simple totals.
- **Checkout submit (address):** $\sim 0.6\text{--}1.5$ s — server validation + session write.
- **Place order (DB commit):** $\sim 0.8\text{--}2.5$ s — atomic stock decrement + item inserts.
- **Invoice download (PDF):** $\sim 0.4\text{--}1.0$ s — generated in memory.

Takeaways

- **Usability:** Simple, task-focused navigation; error states are clear; forms are fast to complete.
- **Performance:** All core pages render within $\sim 0.2\text{--}2.5$ s locally; the heaviest path (order placement) stays under ~ 3 s.
- **Risks / Limits:** Development server, no CSRF token, SQLite single-writer model; production should add CSRF, caching, and a DB like Postgres.

6.3 Discussion (Strengths & Limitations)

Strengths

- **Usability:** Simple navigation; key tasks ≤ 2 clicks from Home; consistent Bootstrap UI.
- **Reliability:** Order placement performs **atomic stock decrement**; prevents oversell.
- **Security (baseline):** Passwords stored as **pbkdf2:sha256**; admin routes protected by role.
- **Performance (prototype):** Local page loads $\sim 0.2\text{--}1.2$ s; order placement $\sim 0.8\text{--}2.5$ s.
- **Modularity:** Flask blueprints + SQLAlchemy models make features easy to extend.

Limitations

- **No real payments or email** integration (mock checkout; no notifications).
- **SQLite** backend limits concurrency; not ideal for multi-user production.
- **Form hardening:** CSRF token not enabled; validation is minimal.
- **Catalog features:** No pagination/faceting; search is simple LIKE.
- **Reporting/analytics:** Only basic KPIs; no downloadable sales reports.

Summary

- The prototype delivers the full MVP: users can **register, log in, discover books, add to cart, checkout, and download a PDF invoice**; admins can **add books, manage stock, and update order status**. All critical flows passed functional testing with acceptable local performance and baseline security (hashed passwords, role gating). Gaps (payments, CSRF, stronger DB, richer search) are identified for future work.

CHAPTER 7 — CONCLUSION & FUTURE SCOPE

7.1 Conclusion

The **Online Bookstore Management System** achieves its core objective: a clean, working MVP that supports the complete buying journey and the essential back-office tasks.

Functional achievements

- **Customer flow:** account registration and login; book discovery via catalog, keyword search and sorting; detailed product view (title, author, ISBN, price, stock); cart management (add/remove with recalculated totals); checkout with validated address; order creation with **atomic stock decrement**; downloadable **PDF invoice**.
- **Admin flow:** secured login (role-based access); **dashboard KPIs** (total orders, revenue, low stock); **inventory management** (add titles, adjust stock); and **order lifecycle** management (Placed → Shipped → Delivered).
- **Data model:** normalized entities (Users, Books, Categories, Orders, OrderItems) with clear relationships; easy to extend to promotions, returns, etc.
- **Architecture:** lightweight Flask blueprints and SQLAlchemy ORM. The design is modular (separate services for auth, catalog, cart, orders, invoice, admin), making future enhancements straightforward.
- **Security baseline:** passwords stored as **pbkdf2:sha256** hashes; session-based authentication; admin routes restricted by role.
- **Usability & performance (prototype):** consistent Bootstrap UI, simple navigation (≤ 2 clicks to key actions), and local response times $\sim 0.2\text{--}2.5\text{s}$, with the heaviest path (order placement) under $\sim 3\text{s}$.
- **Testing outcome:** the prepared test suite (20 cases) passed successfully, including negative checks (duplicate email, invalid login, insufficient stock).

Key learnings

- MVPs benefit from **strict scope** and **simple defaults** (e.g., SQLite, server-rendered pages) to reach a demonstrable system quickly.
- Separating features into **blueprints/services** reduces coupling and clarifies responsibilities.
- Even small apps need **early security choices** (password hashing method, role checks), otherwise environment gaps (e.g., script unavailability) surface late.

Constraints acknowledged

- Dev server only (no WSGI/ASGI hardening); **SQLite** limits write concurrency; **mock checkout** without payments; minimal form hardening (no CSRF tokens); and no asynchronous tasks for email/notifications. These are acceptable trade-offs for a classroom prototype but not for production.

7.2 Future Scope

A. Functional Enhancements

1. **Payments & refunds**
Integrate Razorpay/Stripe for card/UPI; support refunds and partial captures; store payment intents and reconciliation logs.
2. **Order servicing**
Cancellations (pre-shipment), returns (RMA), exchanges, delivery slots, and downloadable GST/Tax invoices.
3. **Customer features**
Persistent carts, **wishlists**, saved addresses, coupons/discount codes, gift cards, and store credit.
4. **Discovery & engagement**
Categories with breadcrumbs, **faceted filters** (price, author, category), related/recommended titles, **ratings & reviews** (with moderation), recently viewed, and email reminders for abandoned carts.

B. Data, Search, and Analytics

- Move from simple LIKE search to **full-text search** (PostgreSQL tsvector or Elasticsearch/OpenSearch).
- Add **pagination** and result caching; track catalog KPIs (CTR, conversion).
- Build **analytics dashboards**: sales by month/category/author, cohort retention, inventory ageing, and top-N sellers with CSV/PDF export.

C. Security & Compliance

- **CSRF protection** for all forms; **HTTPS** everywhere; input validation and server-side sanitization.
- Account policies: password rules, email verification, **login throttling / rate limiting**, and account lockout.
- **Audit trail** for admin actions and critical events; role expansion (viewer, operator, manager).
- **Privacy & backups**: minimal PII, data retention policy, encrypted backups, GDPR-style consent and data-deletion requests.

D. Performance & Scalability

- Migrate to **PostgreSQL** with proper indexing and **Alembic** migrations.
- Introduce **background jobs** (Celery/RQ) for email, invoice generation, and nightly reports.
- **Caching** (Redis) for catalog pages and sessions; CDN for static assets and cover images.
- Optional: **async endpoints** (Quart/FastAPI) for high-latency operations; horizontal scaling behind a load balancer.

E. UX, Accessibility, and Internationalization

- Responsive design tuning for mobile checkout, **keyboard and screen-reader support** (ARIA), better focus states, color-contrast checks.
- **Localization** (i18n/l10n), currency formatting, and regional price rules.
- Visual theming (light/dark), richer micro-interactions, and order-tracking timeline components.

F. DevOps & Quality

- **Dockerize** services; split environments (dev/stage/prod).
- **CI/CD** with automated testing and style checks; pre-commit hooks.
- **Observability**: structured logging, metrics, tracing, and error tracking (e.g., Sentry).
- Data seeding fixtures and sample datasets for demo environments.

7.3 Final Remarks

The project demonstrates a **fully functioning prototype** that is simple to understand, easy to run, and covers all essential bookstore operations. With the proposed upgrades—payments, stronger security, a production-grade database, and richer discovery/analytics—the system can evolve from a coursework MVP into a **deployable, maintainable online store**. The current design decisions (modular services, ORM, and templated UI) set a clear path for that evolution while keeping development fast and predictable.

REFERENCES

- Python Software Foundation, “The Python Language Reference,” docs.python.org, 2024. Available: <https://docs.python.org/3/> (Accessed: 17 Aug 2025).
- Pallets, “Flask Documentation (3.x),” flask.palletsprojects.com, 2024. Available: <https://flask.palletsprojects.com/> (Accessed: 17 Aug 2025).
- Pallets, “Werkzeug Security Utilities,” werkzeug.palletsprojects.com, 2024. Available: <https://werkzeug.palletsprojects.com/en/latest/utils/#module-werkzeug.security> (Accessed: 17 Aug 2025).
- SQLAlchemy, “SQLAlchemy 2.0 Documentation,” sqlalchemy.org, 2024. Available: <https://docs.sqlalchemy.org/> (Accessed: 17 Aug 2025).
- Flask-SQLAlchemy, “Flask-SQLAlchemy 3.x Documentation,” flask-sqlalchemy.palletsprojects.com, 2024. Available: <https://flask-sqlalchemy.palletsprojects.com/> (Accessed: 17 Aug 2025).
- Jinja Project, “Jinja Template Engine Documentation,” palletsprojects.com/jinja, 2024. Available: <https://jinja.palletsprojects.com/> (Accessed: 17 Aug 2025).
- SQLite Consortium, “SQLite Documentation,” sqlite.org, 2024. Available: <https://www.sqlite.org/docs.html> (Accessed: 17 Aug 2025).
- Bootstrap, “Bootstrap 5 Documentation,” getbootstrap.com, 2024. Available: <https://getbootstrap.com/docs/5.3/> (Accessed: 17 Aug 2025).
- OWASP, “Password Storage Cheat Sheet,” owasp.org, 2023. Available: https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html (Accessed: 17 Aug 2025).
- OWASP, “Session Management Cheat Sheet,” owasp.org, 2023. Available: https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html (Accessed: 17 Aug 2025).
- OWASP, “OWASP Top Ten 2021,” owasp.org, 2021. Available: <https://owasp.org/www-project-top-ten/> (Accessed: 17 Aug 2025).
- R. T. Fielding, “Architectural Styles and the Design of Network-based Software Architectures,” Ph.D. dissertation, Univ. of California, Irvine, 2000. Available: https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf (Accessed: 17 Aug 2025).

□ Nielsen Norman Group, “10 Usability Heuristics for User Interface Design,” nngroup.com, 2020 (updated). Available: <https://www.nngroup.com/articles/ten-usability-heuristics/> (Accessed: 17 Aug 2025).

□ Alembic, “Alembic (SQLAlchemy Migrations) Documentation,” alembic.sqlalchemy.org, 2024. Available: <https://alembic.sqlalchemy.org/> (Accessed: 17 Aug 2025).

□ ReportLab, “ReportLab User Guide (PDF Generation),” reportlab.com, 2024. Available: <https://www.reportlab.com/documentation/> (Accessed: 17 Aug 2025).