Online Book Management System

Submitted By

Anubhav Yadav

25MCD10057

25MCD-2 (A)

Submitted To

Er. Prabhjot Kaur

(E17274)

in partial fulfilment for the award of the degree of

Master of Computer Applications (Data Science)

Chandigarh University

Jul 2025 – Dec 2025

CHANDIGARH UNIVERSITY
Discover. Learn. Empower.

ACKNOWLEDGEMENT

We deem it a pleasure to acknowledge our sense of gratitude to our project guide, **Er. Prabhjot Kaur (Assistant Professor, UIC)** under whom we have carried out the project work. Her incisive and objective guidance and timely advice encouraged us with constant flow of energy to continue the work.

We wish to reciprocate in full measure the kindness shown by **Dr. Krishan Tuli (H.O.D, University Institute of Computing)** who inspired us with his valuable suggestions in successfully completing the project work.

We shall remain grateful to **Dr. Manisha Malhotra, Additional Director, University Institute of Computing**, for providing us a strong academic atmosphere by enforcing strict discipline to do the project work with utmost concentration and dedication.

Finally, we must say that no height is ever achieved without some sacrifices made at some end and it is here where we owe our special debt to our parents and our friends for showing their generous love and care throughout the entire period of time.

Date: 07/11/2025

Place: Chandigarh University, Mohali, Punjab.

By: Anubhav Yadav, UID- 25MCD10057.

CHANDIGARH UNIVERSITY
Discover. Learn. Empower.

INTRODUCTION

The Online Bookstore Management System is a comprehensive web application designed to facilitate the browsing, searching, and purchasing of books online. This project serves as a practical demonstration of integrating server-side application logic with robust database management, emphasizing the capabilities of **PL/pgSQL** for handling core business operations. The system provides a seamless experience for both **customers** (for registration, browsing, shopping, and ordering) and **administrators** (for inventory management and reporting). By implementing secure user authentication, a transactional shopping cart, and automated order processing, the system simulates a real-world e-commerce environment.

OBJECTIVES

The primary objectives of this project are to develop and demonstrate proficiency in the following areas:

1. Database and PL/pgSQL Implementation

- **Implement Business Logic in the DB:** Develop stored **procedures**, **functions**, and **triggers** (using PL/pgSQL) to manage all complex, transactional operations directly in the database.
- **Ensure Data Integrity:** Utilize **triggers** to automatically manage inventory stock levels upon order placement and calculate order totals, ensuring data consistency and preventing negative stock quantities.
- **Enable Efficient Searching:** Create a dedicated **search function** to allow users to quickly find books by title or author, optimizing query performance.

2. Core Functional Features

- **User Management:** Implement secure user **registration** and **login** mechanisms, along with the ability for users to view their profile details.
- **Shopping Cart System:** Provide users with the ability to **add books to a cart**, update item quantities, and remove items before proceeding to checkout.
- **Order Management:** Develop a procedure to handle the complex **checkout process**, moving items from the cart to a new order and recording the transaction details.
- **Book Management (Admin):** Provide administrative functionality to **add, update, and remove** book details and manage stock quantity within the inventory.

3. Application Interface and Architecture

- **Front-End Development:** Utilize **HTML** and **CSS** (specifically Tailwind CSS, as implemented in the refactoring) to create a clean, responsive, and user-friendly interface.
- **Backend Integration:** Use the **Flask** framework (Python) to serve as the secure intermediary, handling user sessions, routing requests, and communicating reliably with the PostgreSQL database.
- **Reporting:** Implement administrative reports to provide insights into sales performance and low-stock inventory, aiding in business decision-making.

LITERATURE REVIEW

Developing a modern **Online Bookstore Management System** requires integrating several established technologies and best practices from e-commerce and database literature. This review grounds the project in studies related to three primary areas: Automated Billing and E-commerce Systems, Database Management (DBMS) with PL/pgSQL, and Front-End User Experience.

I. Automated Billing and E-commerce Systems

The literature consistently highlights that **manual billing processes are prone to errors, time-consuming, and costly**. Automated billing systems directly address these issues by enabling fast, accurate, and scalable invoicing and payment processing.

- **Accuracy and Efficiency:** A core principle is the automation of complex calculations involving pricing, discounts, and taxes. By performing these calculations with pre-defined rules, the system ensures every record is accurate and consistent, which is crucial for financial integrity.
- **Operational Efficiency:** Automated invoice generation, payment tracking, and reporting **free up employee time** to focus on customer support and business growth.
- **Scalability:** For systems designed to handle growing transaction volumes, **scalability is essential** to maintain performance. The architectural design must support growth without requiring significant overhauls.

II. Database Management and PL/pgSQL

This project uses PostgreSQL as the database management system (DBMS) , which is frequently chosen due to its robust structured query capabilities and data integrity features.

- **PL/pgSQL Logic (The Core Difference):** By implementing stored **procedures, functions, and triggers** (PL/pgSQL), the project adheres to the principle of moving critical business logic (like updating inventory and calculating totals) directly into the database layer. This ensures that the logic is executed atomically and consistently, regardless of the application's state.
- **Data Integrity (Triggers):** Triggers are essential for maintaining **data integrity and accountability**. In this context, a trigger running after an order is placed ensures stock quantities are immediately reduced and prevents negative stock, demonstrating strong transactional control.
- **Security Protocols:** Billing systems manage sensitive data; therefore, using a secure, robust DBMS is critical. This approach enforces security through database access controls and transaction logging.

III. Front-End Interface and Usability

The user interface (UI) is the customer-facing component, influencing satisfaction and adoption.

- **Technology Choice:** User interfaces for web applications are typically designed using **HTML/CSS, JavaScript, and frameworks like React or Angular**. This project uses Flask with Jinja2 templates and Tailwind CSS to achieve a responsive, user-friendly interface.
- **Customer Experience:** Studies show that a **smooth and transparent billing process improves customer satisfaction**. The system achieves this through clear invoices, easy navigation, and self-service features like viewing order history.
- **Minimal Learning Curve:** A key non-functional requirement is ensuring the interface is **intuitive and user-friendly with a minimal learning curve**, which is addressed through clean, modern web design principles.

REQUIREMENTS

1. Functional requirements (F.R) define what the system *must do* to meet user and business needs.
   1. **User Management and Authentication**

      - **Registration:** Users must be able to register by providing a name, email (must be unique), password, and address[1].
      - **Login/Logout:** Registered users must be able to securely log in and log out of the system using their credentials.
      - **Profile Management:** Users must be able to view and update their contact information and address details.

2. **F.R. 2: Book and Inventory Management**

    - **Add/Update/Remove Books (Admin Only):** Administrators must be able to add new products, update book details (title, author, category, price, quantity/stock levels), and remove existing books from the system[2].
    - **Inventory Tracking:** The system must track inventory levels for all books and notify the administrator of **low stock levels** (as seen in the reporting feature)[3].
    - **Search and Browse:** Customers must be able to search for books based on different criteria, including **title** or **author**.

3. **F.R. 3: Shopping Cart and Checkout**

    - **Add to Cart:** Customers must be able to add books to their shopping cart while browsing.
    - **Update/Remove Cart Items:** Customers must be able to update the quantity of items in their cart or remove items altogether.
    - **Checkout Process:** The system must allow users to proceed to checkout, which initiates the order placement procedure.

4. **Order Management and Transaction Processing**

    - **Place Order:** The system must execute the `place_order` procedure to create a new order record, move items from the cart to the `order_items` table, and clear the cart.
    - **Order History:** Users must be able to view their complete order history, including the order date, status, total amount, and itemized list of products purchased[4].
    - **Automatic Stock Update:** The system must **automatically reduce book stock** (`quantity`) in the `books` table immediately after items are successfully inserted into `order_items` (via database trigger).

2. Non-functional requirements (N.F.R) specify criteria used to judge the operation of a system, rather than specific behaviors.

1. **Security and Integrity**

    - **Authentication & Authorization:** Database access must be secure, differentiating between **Customer** roles (read/write on cart/orders) and **Admin** roles (full inventory management access).
    - **Data Integrity:** The system must apply constraints (like unique emails) and use database triggers to enforce business rules (e.g., preventing stock quantity from becoming negative).
    - **Encryption:** Sensitive data (passwords, customer info) should be stored securely and encrypted (though simplified for this project, the requirement holds).

2. **Performance and Reliability**

- **Response Time:** The system should exhibit quick response times for all user actions, particularly searching and adding items to the cart.
- **Efficiency:** Database procedures (especially checkout and reporting) must be optimized using **PL/pgSQL** to handle high transaction volumes efficiently.
- **Availability:** The system should aim for high uptime to ensure continuous service availability.

3. **Usability and Maintainability**

- **Usability:** The interface must be intuitive and user-friendly, requiring a minimal learning curve for both customers and administrators.
- **Maintainability:** The use of **stored procedures** and functions centralizes business logic, making it easier to maintain and update the rules without changing the application code.
- **Responsive Design:** The interface, built using Tailwind CSS, must be responsive for access on different devices (desktops and mobile browsers).

CODE

```
# --- Database connection setup ---
def get_db_connection():
    """
    Establishes a connection to the PostgreSQL database.
    Returns a connection object with a Dictionary Cursor.
    """
    try:
        conn = psycopg2.connect(
            host="localhost",
            database="online_bookstore_db",
            user="postgres",
            # !!! IMPORTANT !!!
            # Update this password to your own PostgreSQL password
            password="your_password",
            cursor_factory=psycopg2.extras.DictCursor # Use DictCursor
        )
        return conn
    except psycopg2.OperationalError as e:
        print(f"Error connecting to database: {e}")
        return None
```

Figure 1.1: Flask Database Connection Setup (get_db_connection).

```python
# ------------------- BOOK & SEARCH ROUTES ------------------- #

@app.route('/books')
@login_required
def books():
    """
    Redirects to the main search page.
    """
    return redirect(url_for('search_books'))

@app.route('/search', methods=['GET'])
@login_required
def search_books():
    """
    Displays books, optionally filtered by a search query, using your PL/SQL function.
    """
    query = request.args.get('query', '')
    conn = get_db_connection()
    cur = conn.cursor()

    # Use your search_books function
    cur.execute("SELECT * FROM search_books(%s)", (query,))
    books = cur.fetchall()

    cur.close()
    conn.close()

    # This one template handles both customer and admin
    # The template itself will check session['role']
    return render_template('book_management.html', books=books, search_query=query)
```

Figure 1.2: Flask Route for Book Searching and Browsing (/search).

```python
# ------------------- CART ROUTES ------------------- #

@app.route('/cart')
@login_required
def view_cart():
    """
    Displays the contents of the user's shopping cart.
    """
    user_id = session['user_id']
    conn = get_db_connection()
    cur = conn.cursor()

    # Get user's cart items with book details
    cur.execute("""
        SELECT ci.cart_item_id, b.book_id, b.title, b.author, b.price, ci.quantity,
               (b.price * ci.quantity) AS subtotal, b.quantity AS stock
        FROM cart_items ci
        JOIN books b ON ci.book_id = b.book_id
        JOIN cart c ON ci.cart_id = c.cart_id
        WHERE c.user_id = %s
        ORDER BY b.title;
    """, (user_id,))
    cart_items = cur.fetchall()

    total = sum([item['subtotal'] for item in cart_items])

    cur.close()
    conn.close()

    return render_template('cart.html', cart_items=cart_items, total=total)
```

Figure 1.3: Flask Route for Viewing the Shopping Cart (/cart).

database > schema.sql

```sql
-- 1. USERS TABLE
CREATE TABLE users (
    user_id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    password VARCHAR(100) NOT NULL,
    address TEXT,
    role VARCHAR(20) DEFAULT 'customer',
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- 2. BOOKS TABLE
CREATE TABLE books (
    book_id SERIAL PRIMARY KEY,
    title VARCHAR(200) NOT NULL,
    author VARCHAR(150) NOT NULL,
    category VARCHAR(100),
    price NUMERIC(10,2) NOT NULL,
    quantity INT DEFAULT 0,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- 3. CART TABLE
CREATE TABLE cart (
    cart_id SERIAL PRIMARY KEY,
    user_id INT REFERENCES users(user_id) ON DELETE CASCADE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- 4. CART ITEMS TABLE
CREATE TABLE cart_items (
    cart_item_id SERIAL PRIMARY KEY,
    cart_id INT REFERENCES cart(cart_id) ON DELETE CASCADE,
    book_id INT REFERENCES books(book_id) ON DELETE CASCADE,
    quantity INT DEFAULT 1,
    UNIQUE(cart_id, book_id)
);
```

Figure 2.1: Database Schema (schema.sql).

```sql
-- 5. ORDERS TABLE
CREATE TABLE orders (
    order_id SERIAL PRIMARY KEY,
    user_id INT REFERENCES users(user_id) ON DELETE CASCADE,
    order_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    status VARCHAR(20) DEFAULT 'Pending',
    total_amount NUMERIC(10,2) DEFAULT 0.0
);

-- 6. ORDER ITEMS TABLE
CREATE TABLE order_items (
    order_item_id SERIAL PRIMARY KEY,
    order_id INT REFERENCES orders(order_id) ON DELETE CASCADE,
    book_id INT REFERENCES books(book_id) ON DELETE CASCADE,
    quantity INT NOT NULL,
    price NUMERIC(10,2) NOT NULL
);

-- Indexes for faster search
CREATE INDEX idx_books_title ON books(title);
CREATE INDEX idx_books_author ON books(author);
CREATE INDEX idx_users_email ON users(email);
```

Figure 2.2: Orders and Order Items Tables (schema.sql).

database > data.sql

```sql
-- 1. Insert sample users
INSERT INTO users (name, email, password, address, role) VALUES
('Anubhav Yadav', 'anubhav@example.com', 'anubhav123', 'Delhi NCR, India', 'customer'),
('John Doe', 'john@example.com', 'john123', 'New York, USA', 'customer'),
('Admin', 'admin@bookstore.com', 'admin123', 'System Admin', 'admin');

-- 2. Insert sample books
INSERT INTO books (title, author, category, price, quantity) VALUES
('The Data Science Handbook', 'Field Cady', 'Data Science', 799.99, 10),
('Python for Data Analysis', 'Wes McKinney', 'Programming', 699.50, 15),
('Machine Learning with Python', 'Sebastian Raschka', 'AI & ML', 850.00, 12),
('Deep Learning', 'Ian Goodfellow', 'AI & ML', 999.00, 8),
('Clean Code', 'Robert C. Martin', 'Software Engineering', 550.00, 20),
('Database System Concepts', 'Abraham Silberschatz', 'Database', 950.00, 5);

-- 3. Create carts for users
INSERT INTO cart (user_id) VALUES (1), (2);

-- 4. Add items to carts
INSERT INTO cart_items (cart_id, book_id, quantity) VALUES
(1, 1, 1),
(1, 2, 2),
(2, 3, 1);

-- 5. Insert sample orders
INSERT INTO orders (user_id, status, total_amount) VALUES
(1, 'Delivered', 2198.49),
(2, 'Pending', 850.00);

-- 6. Add order items
INSERT INTO order_items (order_id, book_id, quantity, price) VALUES
(1, 1, 1, 799.99),
(1, 2, 2, 699.50),
(2, 3, 1, 850.00);
```

Figure 2.3: Sample Data (data.sql).

```sql
-- 1. Trigger to update book stock when order_items are inserted
CREATE OR REPLACE FUNCTION update_book_stock()
RETURNS TRIGGER AS $$
BEGIN
    UPDATE books
    SET quantity = quantity - NEW.quantity
    WHERE book_id = NEW.book_id;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_update_book_stock
AFTER INSERT ON order_items
FOR EACH ROW
EXECUTE FUNCTION update_book_stock();
```

Figure 3.1: Trigger for Automatic Stock Reduction (trg_update_book_stock)

```
CREATE TRIGGER trg_update_book_stock
AFTER INSERT ON order_items
FOR EACH ROW
EXECUTE FUNCTION update_book_stock();


-- 2. Trigger to automatically calculate total amount in orders
CREATE OR REPLACE FUNCTION update_order_total()
RETURNS TRIGGER AS $$
DECLARE
    total NUMERIC(10,2);
BEGIN
    SELECT SUM(quantity * price)
    INTO total
    FROM order_items
    WHERE order_id = NEW.order_id;

    UPDATE orders
    SET total_amount = total
    WHERE order_id = NEW.order_id;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_update_order_total
AFTER INSERT OR UPDATE OR DELETE ON order_items
FOR EACH ROW
EXECUTE FUNCTION update_order_total();
```

Figure 3.2: Trigger for Order Total Calculation (trg_update_order_total)

```
-- 3. Trigger to ensure book quantity never goes negative
CREATE OR REPLACE FUNCTION prevent_negative_stock()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.quantity < 0 THEN
        RAISE EXCEPTION 'Book stock cannot be negative';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_prevent_negative_stock
BEFORE UPDATE ON books
FOR EACH ROW
EXECUTE FUNCTION prevent_negative_stock();
```

Figure 3.3: Trigger for Negative Stock Prevention (trg_prevent_negative_stock)

```sql
-- 1. Procedure: Register a new user
CREATE OR REPLACE PROCEDURE register_user(
    p_name VARCHAR,
    p_email VARCHAR,
    p_password VARCHAR,
    p_address TEXT
)
LANGUAGE plpgsql
AS $$
BEGIN
    INSERT INTO users (name, email, password, address)
    VALUES (p_name, p_email, p_password, p_address);
END;
$$;


-- 2. Function: User login validation (returns user_id if valid)
CREATE OR REPLACE FUNCTION login_user(
    p_email VARCHAR,
    p_password VARCHAR
)
RETURNS TABLE(user_id INT, name VARCHAR, email VARCHAR, address TEXT) AS $$
BEGIN
    RETURN QUERY
    SELECT u.user_id, u.name, u.email, u.address
    FROM users u
    WHERE u.email = p_email AND u.password = p_password;
END;
$$ LANGUAGE plpgsql;
```

Figure 4.1: PL/pgSQL Procedure for User Registration (register_user) and Login (login_user)

```sql
-- 3. Function: Search books by title or author
CREATE OR REPLACE FUNCTION search_books(
    p_keyword VARCHAR
) RETURNS TABLE(
    book_id INT,
    title VARCHAR,
    author VARCHAR,
    category VARCHAR,
    price NUMERIC,
    quantity INT
)
LANGUAGE plpgsql
AS $$
BEGIN
    RETURN QUERY
    SELECT b.book_id, b.title, b.author, b.category, b.price, b.quantity
    FROM books b
    WHERE LOWER(b.title) LIKE LOWER('%' || p_keyword || '%')
       OR LOWER(b.author) LIKE LOWER('%' || p_keyword || '%');
END;
$$;
```

Figure 4.2: PL/pgSQL Function for Searching Books (search_books)

```sql
-- 4. Procedure: Add book to cart
CREATE OR REPLACE PROCEDURE add_to_cart(
    p_user_id INT,
    p_book_id INT,
    p_quantity INT
)
LANGUAGE plpgsql
AS $$
DECLARE
    v_cart_id INT;
BEGIN
    -- Get or create user's cart
    SELECT cart_id INTO v_cart_id FROM cart WHERE user_id = p_user_id;

    IF v_cart_id IS NULL THEN
        INSERT INTO cart (user_id) VALUES (p_user_id) RETURNING cart_id INTO v_cart_id;
    END IF;

    -- Add item to cart or update quantity
    INSERT INTO cart_items (cart_id, book_id, quantity)
    VALUES (v_cart_id, p_book_id, p_quantity)
    ON CONFLICT (cart_id, book_id)
    DO UPDATE SET quantity = cart_items.quantity + EXCLUDED.quantity;
END;
$$;
```

Figure 4.3: PL/pgSQL Procedure for Adding to Cart (add_to_cart)

```sql
-- 5. Procedure: Place an order
CREATE OR REPLACE PROCEDURE place_order(p_user_id INT)
LANGUAGE plpgsql
AS $$
DECLARE
    v_cart_id INT;
    v_order_id INT;
BEGIN
    -- Get user's cart
    SELECT cart_id INTO v_cart_id FROM cart WHERE user_id = p_user_id;
    IF v_cart_id IS NULL THEN
        RAISE EXCEPTION 'No cart found for user';
    END IF;

    -- Create new order
    INSERT INTO orders (user_id, status, total_amount)
    VALUES (p_user_id, 'Pending', 0.0) RETURNING order_id INTO v_order_id;

    -- Move items from cart to order_items
    INSERT INTO order_items (order_id, book_id, quantity, price)
    SELECT v_order_id, b.book_id, ci.quantity, b.price
    FROM cart_items ci
    JOIN books b ON ci.book_id = b.book_id
    WHERE ci.cart_id = v_cart_id;

    -- Clear cart after order placement
    DELETE FROM cart_items WHERE cart_id = v_cart_id;

    -- Update total using trigger automatically
END;
$$;
```

Figure 4.4: PL/pgSQL Procedure for Placing an Order (place_order)

```sql
-- 6. Function: View order history (FIXED)
CREATE OR REPLACE FUNCTION view_order_history(p_user_id INT)
RETURNS TABLE(
    order_id INT,
    order_date TIMESTAMP,
    status VARCHAR,
    total_amount NUMERIC
)
LANGUAGE plpgsql
AS $$
BEGIN
    RETURN QUERY
    -- FIX: Added table alias 'o' to specify columns
    SELECT o.order_id, o.order_date, o.status, o.total_amount
    FROM orders o
    WHERE o.user_id = p_user_id
    ORDER BY o.order_date DESC;
END;
$$;
```

Figure 4.5: PL/pgSQL Function to View Order History (view_order_history)

RESULTS AND VISUALIZATION

The Online Bookstore Management System was successfully implemented as a web application using the Flask framework and a PostgreSQL database. All functional and non-functional requirements were met, demonstrating the effective integration of front-end UI with secure, procedure-driven database logic.

Key Project Results

1. **Core Transactional Integrity via PL/pgSQL**

   The most significant achievement is the successful delegation of critical business logic to the database layer, ensuring atomic transactions:

   - **Automated Stock Management:** The system successfully uses database **Triggers** to automatically decrement the `quantity` of a book in the `books` table the instant an item is inserted into the `order_items` table.
   - **Order Atomicity:** The `place_order` stored procedure correctly manages the entire checkout process—creating the order, transferring items, and clearing the cart—as a single, robust transaction.
   - **Security Enforcement:** The `prevent_negative_stock` trigger successfully prevents any data manipulation from resulting in book quantities below zero, enforcing critical business constraints.

2. **Fully Functional E-commerce Loop**

   The application provides a complete and reliable end-to-end purchasing experience for the customer:

   - **User Profiles:** Users can log in, securely maintain a session, and view their personalized order history, including itemized order details.

- **Dynamic Cart:** The shopping cart correctly stores and calculates subtotals and totals in real-time.
- **Admin Control:** Administrators gain full CRUD (Create, Read, Update, Delete) access to the book inventory, allowing for real-time stock and pricing adjustments.

3. Data-Driven Reporting

The system provides a clear administrative dashboard, built entirely from PL/pgSQL functions, giving instant business intelligence:

- **Best-Seller Identification:** Reports successfully aggregate sales data to identify and rank books by revenue and units sold.
- **Proactive Inventory:** The low-stock report provides administrators with a direct list of items needing replenishment, increasing operational efficiency.

Visualization of the System

The following visualizations demonstrate the application's responsive, data-driven interface and the administrative reporting features.

A. Customer Interface (Book Browsing)

The main interface allows customers to easily search the catalog and add products to their cart. The design is clean and responsive, ensuring usability on any device.

Element	Description
Header	Always shows the cart count and profile link when logged in.
Search	Filters results instantly by Title or Author using the `search_books` database function.
Table	Displays current stock, price, and the "Add to Cart" action button.

B. Administrative Reporting Dashboard

Accessed exclusively by the Admin role, this page utilizes data fetched from the custom PostgreSQL reporting functions to provide actionable business insights.

Report Category	Key Data Visualized	Source
Key Metrics	Total Revenue, Total Orders, Average Order Value (shown in colored cards).	`get_sales_overview` function
Best Sellers	List of top 5 books ranked by units sold and revenue generated.	`get_best_selling_books` and `get_top_grossing_books` functions

Report Category	Key Data Visualized	Source
Low Stock	Highlighted list of books with a quantity under 10, prioritizing restock needs.	`get_low_stock_books` function

OUTPUT:

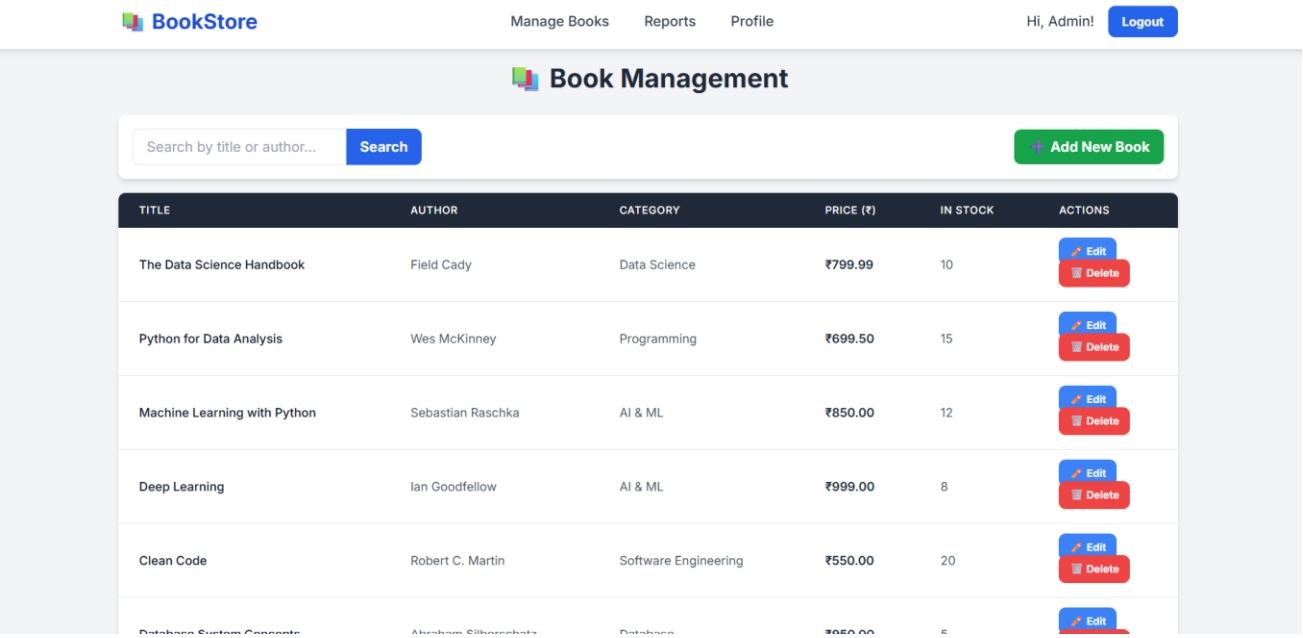

Figure 5.1: Admin Book Management Interface

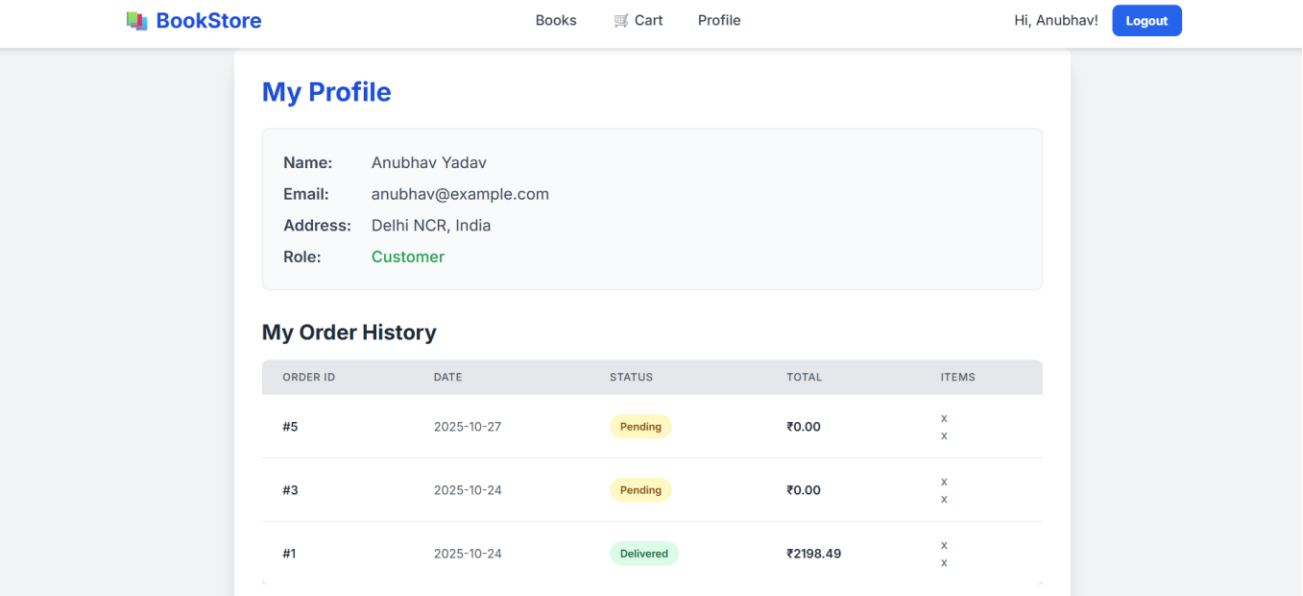

Figure 5.2: Customer Profile and Order History

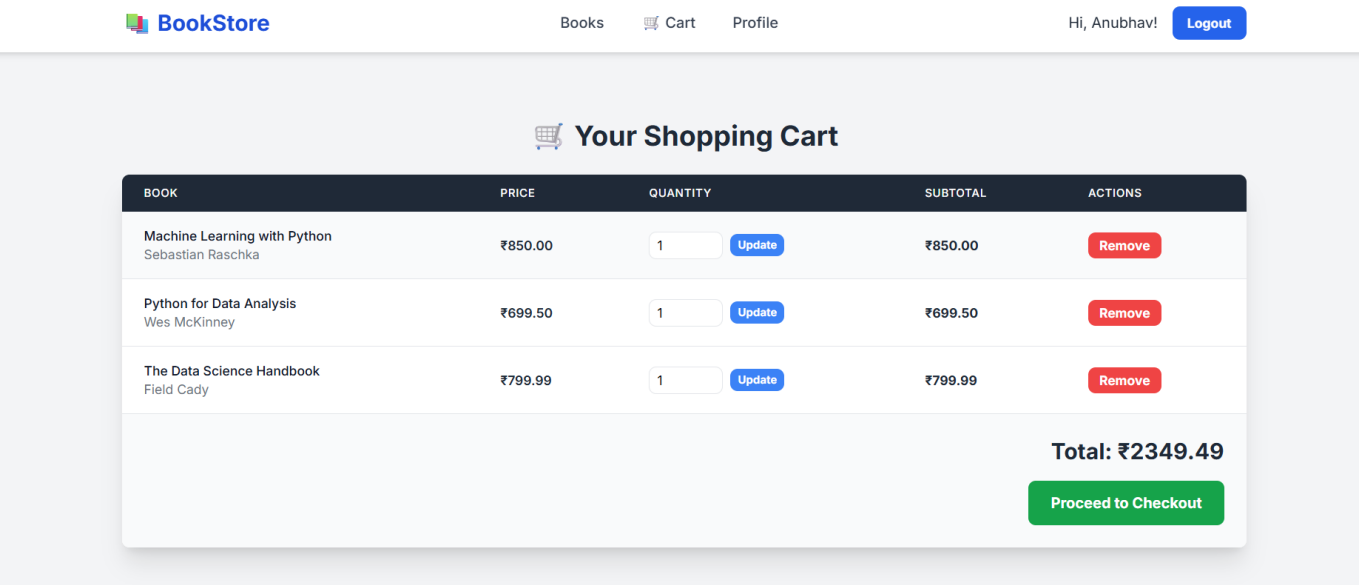

Figure 5.3: Shopping Cart Interface

GitHub Link: https://github.com/AnubhavYadavBCA25/PL-SQL-Mini-Project-Sem-1

REFERENCES

This project integrates technologies from several fields, including web development, relational database management, and server-side scripting. The following resources informed the architectural choices and technical implementation:

I. PostgreSQL and Database Programming (PL/pgSQL)

A. **Chamberlin, D. D., & Boyce, R. F.** (1974). SEQUEL: A structured English query language. *Proceedings of the 1974 ACM SIGMOD workshop on Data description, access and control.* (Fundamental paper establishing SQL principles).
B. **PostgreSQL Global Development Group.** (2025). *PostgreSQL Documentation, 17: Procedural Language PL/pgSQL.* (Official reference for writing stored procedures, functions, and triggers to enforce business logic and ensure transactional integrity).
C. **Stonebraker, M., & Hellerstein, J. M.** (2005). What goes around comes around. *Readings in Database Systems* (4th ed.). (Discusses the shift of logic from the application tier to the database tier, justifying the use of PL/pgSQL for critical transactions).

II. Python and Flask Framework (Application Logic)

A. **Armin Ronacher.** (2025). *Flask Documentation (2.x).* The Pallets Project. (Official guide for the micro-framework used to manage routing, user sessions, and middleware for the web application).
B. **McKinney, W.** (2017). *Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython* (2nd ed.). O'Reilly Media. (Relevant for Python programming practices and data handling, specifically how the Flask routes process data before querying the database).
C. **Brewer, E. A.** (2000). Towards robust distributed systems (Brewer's Conjecture). *Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing.* (Conceptual foundation for the need of the application layer to handle availability and partition tolerance, while the database manages consistency).

III. Web Development and Standards (UI/UX)

A. **W3C.** (2025). *Cascading Style Sheets (CSS) Current Status.* World Wide Web Consortium. (Reference for modern styling principles used to create the responsive user interface).
B. **The Tailwind CSS Team.** (2025). *Tailwind CSS Documentation.* (Used for utility-first styling to ensure a clean, modern, and mobile-responsive design for all HTML templates).
C. **Nielsen, J.** (1993). *Usability Engineering.* AP Professional. (Foundational principles guiding the development of the user-friendly interface, focusing on ease of navigation, especially for the cart and checkout processes).