

# BookVerse

A Project Report

*In the partial fulfillment of the award of the degree of*

**B.Sc**

under

**Academy of Skill Development**



*Submitted by*

**Siddhanta Roy  
Shitij Bhadra  
Rajen Allen Burney  
Praket Jaiswal**



## St. Xavier's College (Autonomous), Kolkata



### Certificate from the Mentor

This is to certify that **Siddhanta Roy , Shitij Bhadra , Rajen Allen Burney and Praket Jaiswal** have completed the project titled **BookVerse** under my supervision during the period from **May** to **July** which is in partial fulfillment of requirements for the award of the [B.Sc](#) and submitted to Department **Computer Science of St. Xavier's College (Autonomous) , Kolkata.**

**Date:**

---

*Signature of the Mentor*

## **Acknowledgment**

I take this opportunity to express my deep gratitude and sincerest thanks to my project mentor, **Mr. Subhojit Santra** for giving the most valuable suggestions, helpful guidance, and encouragement in the execution of this project work.

I would like to give a special mention to my colleagues. Last but not least I am grateful to all the faculty members of the **Academy of Skill Development** for their support.

# CONTENT PAGE

<b>1. COMPANY PROFILE</b>	<b>1</b>
ASD Technologies	1
ASD Collaborations	1
Associations and Accreditations	1
<b>2. INTRODUCTION</b>	<b>2</b>
2A.OBJECTIVE	3
2B.SCOPE	4
1. Book Display System:	4
2. Category Sorting:	4
3. Sales Highlights :	4
4. Purchase Functionality:	4
5. Admin Panel:	4
6. User Interface:	4
<b>3. SYSTEM ANALYSIS</b>	<b>5</b>
3A.IDENTIFICATION OF NEED	6
3B.FEASIBILITY STUDY	7
3C.WORKFLOW	8
Waterfall Model Design:	8
Iterative Waterfall Design:	8
Advantages:	9
Disadvantages:	9
Applications:	10
<b>3D .STUDY OF THE SYSTEM</b>	<b>11</b>
• Register:	11
• Verify Account:	11
• Login:	11
• Home:	11
• Books:	11
• About:	11
• Account:	11
• User Interface:	11
• Admin Interface:	11
3E.INPUT AND OUTPUT	12
INPUT:	12
OUTPUT:	12
<b>3F.SOFTWARE REQUIREMENT SPECIFICATIONS</b>	<b>13</b>

The developer is responsible for:-----	13
Functional Requirements:-----	13
A. User Registration and authentication	
B. Browse and Search:-----	13
C. Books Display: -----	13
Hardware Requirements:-----	13
Software Requirements:-----	13
3G.SOFTWARE ENGINEERING PARADIGM APPLIED-----	14
<b>4. SYSTEM DESIGN</b> -----	<b>15</b>
<b>4A. DATA FLOW DIAGRAM</b> -----	<b>16</b>
DFD Notation:-----	17
DFD Example:-----	17
Database Input Output-----	17
Rules for constructing a Data Flow Diagram-----	18
• LEVEL 0 DFD OR CONTEXT DIAGRAM:-----	18
1. LEVEL 1 DFD: (for user):-----	19
2. LEVEL 1 DFD: (for admin):-----	20
<b>4B. SEQUENCE DIAGRAM</b> -----	<b>21</b>
How to draw Use Case Diagram?-----	24
4C. SCHEMA DIAGRAM-----	27
<b>5. UI SNAPSHOT</b> -----	<b>28</b>
❖ FRONTEND: -----	28
1.Home Page:-----	28
2.Login Page:-----	29
3.Register Page-----	32
4.Admin Dashboard:-----	36
5.Add New Books(Admin):-----	40
6.Put On Sale(Admin):-----	43
7.User Dashboard-----	46
8.Shop:-----	50
9.Cart:-----	56
10.Checkout:-----	59
11.Order History:-----	63
12.All Orders(Admin):-----	66
13.About Us:-----	68
❖ BACKEND:-----	71
• Server.js:-----	71
1.User and Admin Data:-----	72
• User.js:-----	72
2.Book Data:-----	73
• Book.js:-----	73
3.Order Data:-----	74

	• Order.js:-----	74
6. Conclusion:-----		75
7. FUTURE SCOPE & FURTHER ENHANCEMENTS -----		76
❖ Future scope:-----		76
❖ Further enhancement:-----		77
8. BIBLIOGRAPHY-----		78

# **1. COMPANY PROFILE**

**Academy of Skill Development (ASD)**, formerly known as Ardent Computech Private Limited, is an ISO 9001:2015 certified Software Development and Training Company based in India. Operating independently since 2003, the organization has recently undergone a strategic merger with **ASD Technologies**, enhancing its global outreach and service offerings.

## **ASD Technologies**

ASD Technologies delivers high-end IT services across the UK, USA, Canada, and India. Its core competencies lie in the development of customized application software, encompassing end-to-end solutions including system analysis, design, development, implementation, and training. The company also provides expert consultancy and electronic security solutions. Its clientele spans educational institutions, entertainment companies, resorts, theme parks, the service industry, telecom operators, media, and diverse business sectors.

## **ASD Collaborations**

**ASD Collaborations**, the Research, Training, and Development division of ASD, offers professional IT-enabled services and industrial training programs. These are tailored for freshers and professionals from B.Tech, M.Tech, MBA, MCA, BCA, and MSc backgrounds. ASD provides Summer Training, Winter Training, and Industrial Training to eligible candidates. High-performing students may qualify for stipends, scholarships, and additional benefits based on performance and mentor recommendations.

## **Associations and Accreditations**

ASD is affiliated with the **National Council of Vocational Training (NCVT)** under the **Directorate General of Employment & Training (DGET)**, Ministry of Labour & Employment, Government of India. The institution upholds strict quality standards under ISO 9001:2015 certification and is dedicated to bridging the gap between academic knowledge and industry skills through innovative training programs.

## 2. INTRODUCTION

As the digital world continues to evolve rapidly, so too does the way we interact with books. Traditional bookstores and libraries, while cherished, often struggle to meet the demands of today's fast-paced, convenience-driven society. BookVerse was born out of a desire to simplify the book-buying process and bring a curated library of titles to users in just a few clicks. In an age where people value efficiency and clarity, BookVerse steps in as a modern solution for online book discovery and purchase.

Unlike complex learning platforms or feature-heavy e-readers, BookVerse intentionally embraces simplicity. It allows users to explore a growing collection of books, view detailed descriptions, and buy instantly—without unnecessary distractions or complicated navigation. Whether a user is interested in newly released titles, popular best sellers, or discounted books under the **Sale** section, the platform ensures a consistent, hassle-free browsing experience.

Users can also sort books by genres, which helps narrow down searches and personalizes the shopping process. To support its operations, BookVerse features an admin dashboard that enables backend users to manage book listings, update inventory, and keep track of customer orders. The platform is fast, minimal, and user-centric, making it ideal for anyone looking to buy books without the clutter of modern web applications. With BookVerse, the focus is simple: browse, click, and buy.



## **2A.OBJECTIVE**

The main objective of BookVerse is to serve as a simple, effective, and easy-to-use online platform that focuses solely on the purchase and discovery of books. Users can browse a wide selection of titles categorized by genres, popularity, and recency. Whether someone is searching for the latest arrivals, best sellers, or discounted books on sale, BookVerse ensures that the discovery process is smooth and satisfying.

A core function of the platform is the seamless buying experience. Once a user finds a book they like, they can proceed with a quick and user-friendly checkout system. There are no distractions, just a clean interface built for purpose.

To support the backend, the platform includes an admin dashboard. This panel allows the administrators to manage the book listings, update book information, track orders, and handle inventory efficiently. The entire platform is developed with minimalism and functionality in mind, making it ideal for users who simply want to explore and buy books without unnecessary complexity.

BookVerse does not aim to replicate digital reading platforms or build reader communities. Instead, its objective is singular and focused: to make book discovery and purchase as convenient and clear-cut as possible.

## **2B.SCOPE**

BookVerse is built as a simple yet effective book e-commerce system. The primary scope of the platform includes the following features:

1. **Book Display System:** Users can view a wide collection of books with essential details such as title, author, price, and cover image.
2. **Category Sorting:** Books are organized and searchable by genre, helping users quickly find what they're looking for.
3. **Sales Highlights:**
  - a. **Sale Section:** Books available at a discounted price.
  - b. **Best Selling Section:** Top-selling books displayed prominently.
  - c. **New Arrivals:** Recently added books featured separately.
4. **Purchase Functionality:** Users can select books and proceed to a simple and secure checkout process.
5. **Admin Panel:**
  - a. Add, edit, and delete books.
  - b. Categorize and tag books (e.g., Sale, New Arrival).
  - c. View and manage order history and customer transactions.
6. **User Interface:** Intuitive layout for ease of navigation and quick interaction.

### **3. SYSTEM ANALYSIS**

### **3A.IDENTIFICATION OF NEED**

In recent years, the shift toward online shopping and digital access has transformed user expectations across industries, including book retail. Many readers today seek a faster, simpler way to browse and purchase books without the hassle of visiting physical stores or navigating overly complex platforms. BookVerse addresses this need by providing a clean, intuitive space where books are showcased attractively and transactions are straightforward.

There is a noticeable gap in the availability of lightweight, minimal platforms tailored for users who simply want to buy books. Many existing solutions either overload users with extra features like reviews, recommendations, and digital reading tools or require multiple steps just to complete a basic purchase. BookVerse removes that friction.

Additionally, book retailers often require a backend solution to track inventory, manage listings, and monitor order history. Without a centralized admin system, keeping digital storefronts updated can be time-consuming and inefficient. BookVerse solves this by providing an integrated admin dashboard to support daily operations with ease.

- The need for BookVerse is driven by:
- The growing preference for online shopping over physical bookstores.
- A demand for faster, cleaner browsing and purchasing experiences.
- The lack of simple, book-specific platforms without clutter or excessive features.
- The necessity for an efficient admin interface to manage inventory and orders.

BookVerse exists to streamline the book buying process—for both customers and administrators.

### **3B.FEASIBILITY STUDY**

- **The development and deployment of BookVerse have been assessed from multiple feasibility perspectives—technical, economic, operational, and legal—to ensure that the project is both achievable and sustainable.**
- **Technical Feasibility:**  
**BookVerse is built using modern web development tools and frameworks that are well-supported, scalable, and suitable for lightweight applications. The platform requires only basic infrastructure to run efficiently—such as a web server, database, and frontend framework—making it highly practical from a technical standpoint. Given the simplicity of the system (book listing, sorting, purchase, and admin panel), it does not require complex architecture or high-end server configurations.**
- **Economic Feasibility:**  
**Since BookVerse focuses on essential features, the cost of development and maintenance remains low. It avoids expensive integrations like AI recommendation systems or e-readers, and instead invests in usability and reliability. Moreover, by enabling book sales, the platform has potential to generate revenue from day one, either via direct transactions or future expansion into affiliate partnerships with publishers or book distributors.**
- **Operational Feasibility:**  
**The system is designed to be easily manageable. Admins can add or update book listings, handle orders, and monitor inventory with minimal training. Users can find and purchase books without logging in or navigating unnecessary steps. This ease of use ensures smooth day-to-day operation and positive user engagement without requiring significant operational resources.**
- **Legal Feasibility:**  
**BookVerse complies with basic e-commerce regulations such as secure transactions, proper data handling, and transparent order management. Since it does not store sensitive user data beyond order records and book listings, compliance with data privacy regulations (like GDPR or India's PDP Bill) is straightforward and manageable.**

### **3C.WORKFLOW**

This Document plays a vital role in the development life cycle (SDLC) as it describes the complete requirements of the system. It is meant for use by the developers and will be the basic during the testing phase. Any changes made to the requirements in the future will have to go through a formal change approval process.

The Waterfall Model was the first Process Model to be introduced. It is also referred to as a linear-sequential life cycle model. It is very simple to understand and use. In a waterfall model, each phase must be completed before the next phase can begin and there is no overlapping in the phases.

The waterfall model is the earliest SDLC approach that was used for software development. The waterfall Model illustrates the software development process in a linear sequential flow; hence it is also referred to as a linear-sequential life cycle model. This means that any phase in the development process begins only if the previous phase is complete. In the waterfall model phases do not overlap.

#### **□ Waterfall Model Design:**

The waterfall approach was the first SDLC Model to be used widely in Software Engineering to ensure the success of the project. In “The Waterfall” approach, the whole process of software development is divided into separate phases. In the Waterfall model, typically, the Outcome of one phase acts as the input for the next phase sequentially.

#### **□ Iterative Waterfall Design:**

**Definition:** The Iterative Waterfall Model is a variation of the traditional Waterfall model, which is a linear and sequential software development methodology. In the Iterative Waterfall Model, the development process is divided into small, manageable cycles, allowing for the revisiting and refinement of phases before progressing to the next stage. It combines the systematic structure of the Waterfall model with the flexibility of iterative development.

The sequential phases in Iterative Waterfall model are:

- **Requirement Gathering and Analysis:** All possible requirements of the system to be developed are captured in this phase and documented in a requirement specification doc.
- **System Design:** The requirement specifications from the first phase are studied in this phase and system design is prepared. System Design helps in specifying hardware and system requirements and also helps in defining overall system architecture.
- **Implementation:** With inputs from system design, the system is first developed in small programs called units, which are integrated in the next phase. Each unit is developed and tested for its functionality which is referred to as Unit Testing.
- **Integration and Testing:** All the units developed in the implementation phase are integrated into a system after testing each unit. Post integration the entire system is tested for any faults and failures.
- **Deployment of the system:** Once the functional and non-functional testing is done, the product is deployed in the customer environment or released into the market.
- **Maintenance:** Some issues come up in the client environment. To fix those issues patches are released. Also to enhance the product some better versions are released. Maintenance is done to deliver these changes in the customer environment.

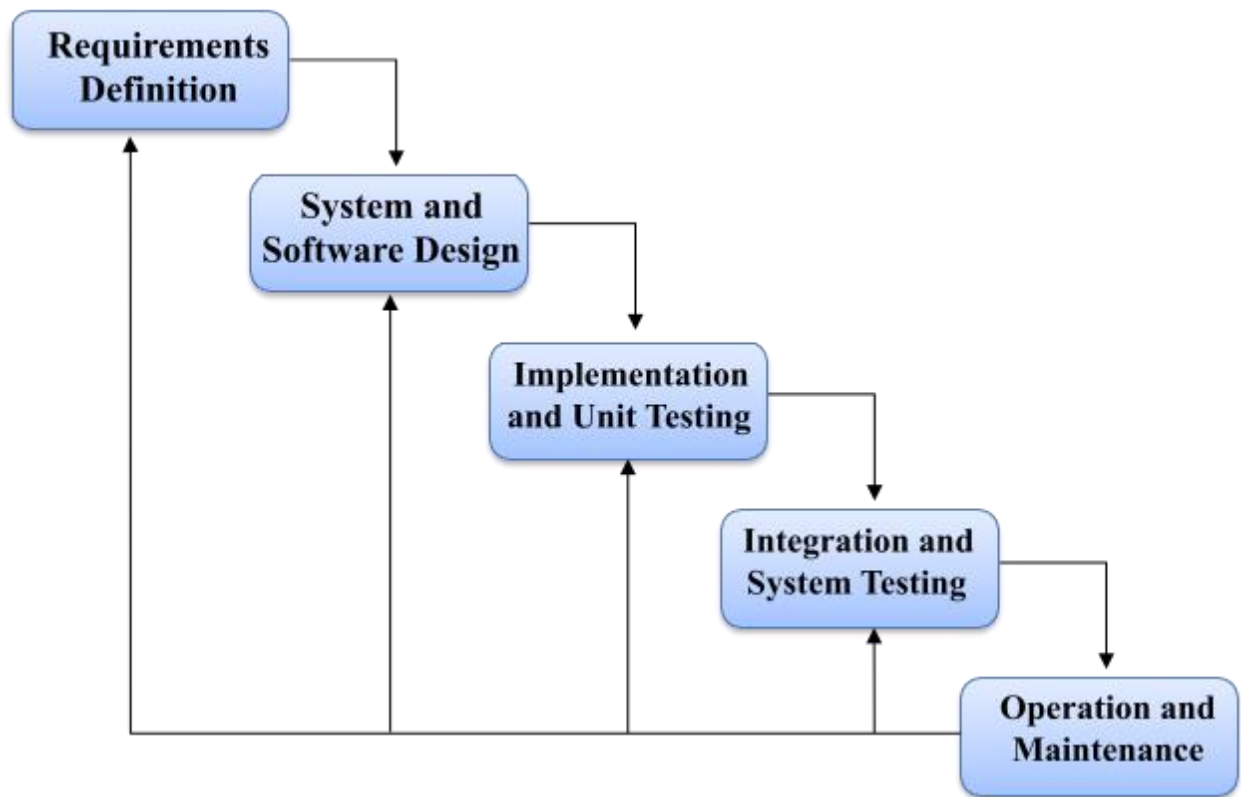
All these phases are cascaded to each other in progress and are seen as flowing steadily downwards (like a waterfall) through the phases. The next phase is started only after the defined set of goals are achieved for the previous phase and it is signed off, so the name “Iterative Waterfall Model”. In this model, phases do not overlap.

· **Advantages:**

- 1 . **Flexibility:** Iterations permit adjustments based on feedback.
- 2 . **Early Delivery:** Partial systems can be delivered incrementally.
- 3 . **Risk Management:** Identifying and addressing issues early in the process.

· **Disadvantages:**

1. **Increased Complexity:** The iterative nature can make the process more complex.
2. **Potential for Scope Creep:** Frequent iterations may lead to scope changes.
3. **Resource Intensive:** Continuous revisiting of phases may demand more resources.



· **Applications:**

The Iterative Waterfall Model is suitable for projects with evolving or unclear requirements. It is commonly used in software development projects where regular feedback and refinement are essential.

Additionally, it is applicable in scenarios where partial system delivery is beneficial, allowing stakeholders to assess progress and make adjustments.



### **3D .STUDY OF THE SYSTEM**

**Modules:** The modules used in this software are as follows:

- **Register:**
  - 1 . **User Register:** Here, the user will register to buy any book.
  - 2 . **Admin Register:** Here, the admin will register to handle all the databases.
- **Verify Account:** When a user login using a wrong password or username then it will display an error message.
- **Login:**
  - 1 . **User Login:** Here users will login to see available books and buy them.
  - 2 . **Admin Login:** Here admin will log in to handle all the data.
- **Home:** This page helps to navigate to all the other pages.
- **Books:**
  - 1. **User interface:** This page shows the available books that the users can purchase. They can also search for books if available.
  - 2. **Admin interface:** In this page, the Admin can view or delete books.
- **About:** This page will show the details about the website developers.
- **Account:**
  - 1. **User Interface:**
    - a) **Dashboard:** Here, the user can see the books he/she can purchase.
  - 2. **Admin Interface:**
    - a) **Admin Dashboard:** Here the Admin can manage , update and add books .

### **3E.INPUT AND OUTPUT**

The main inputs, outputs and the major function the details are:

#### ☐ **INPUT:**

1. Users can log in by entering their **credentials** on the login page.

#### ☐ **OUTPUT:**

1. Users can view the **available books and buy any books.**
2. The **admin** can access a **centralized database** that includes details of books, **add any book and put any book on sale** ensuring efficient system management.

## **3F.SOFTWARE REQUIREMENT SPECIFICATIONS**

Software Requirements Specification provides an overview of the entire project. It is a description of a software system to be developed, laying out functional and non-functional requirements. The software requirements specification document enlists enough necessary requirements that are required for the project development. To derive the requirements we need to have a clear and thorough understanding of the project to be developed. This is prepared after detailed communication with the project team and the customer.

### **The developer is responsible for:**

- Developing the system, which meets the SRS and solves all the requirements of the system.
- Demonstrating the system and installing the system at the client's location after acceptance testing is successful.
- Submitting the required user manual describing the system interfaces to work on it and also the documents of the system.

### **Functional Requirements:**

#### **A. User Registration and Authentication:**

1. Users should be able to create accounts securely.
2. The system should authenticate users and manage login sessions.

#### **B. Browse and Search:**

1. Users should be able to browse and search for books.

#### **C. Books Display:**

1. Each User should have detailed and up-to-date books with prices.
2. Users should be able to view books and buy them easily.

### **Hardware Requirements:**

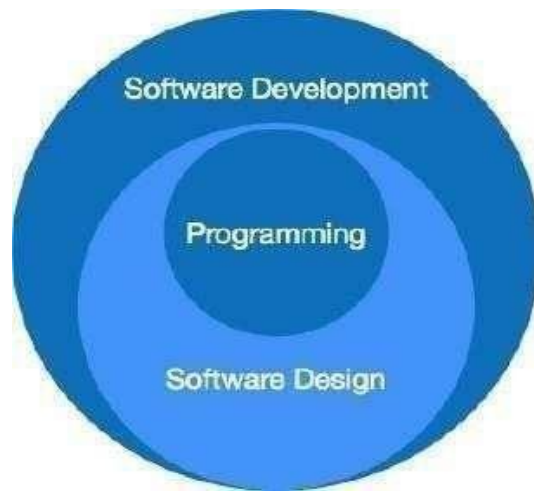
- 1 . Computer has Intel I3 Processor
2. 8 GB RAM
3. SSD-ROM Drive

### **Software Requirements:**

1. Windows 11 OS
2. Visual Studio Code
3. Mongo DB Atlas
4. Node js
5. React js

### **3G.SOFTWARE ENGINEERING PARADIGM APPLIED**

Software paradigms refer to the methods and steps, which are taken while designing the software. There are many methods proposed and are in work today, but we need to see where in software engineering these paradigms stand. These can be combined into various categories, though each of them is contained in one another.



The programming paradigm is a subset of Software design paradigm which is further a subset of the Software development paradigm.

There are two levels of reliability. The first is meeting the right requirements. A careful and thorough systems study is needed to satisfy this aspect of reliability. The second level of systems reliability involves the actual work delivered to the user. At this level, the system's reliability is interwoven with software engineering and development.

There are three approaches to reliability.

1. **Error avoidance:** Prevents errors from occurring in software.
2. **Error detection and correction:** In this approach, errors are recognized whenever they are encountered, and correcting the error by the effect of the error of the system does not fail.
3. **Error tolerance:** In this approach, errors are recognized whenever they occur, but enables the system to keep running through degraded performance or Applying values that instruct the system to continue process.

## **4. SYSTEM DESIGN**

## **4A. DATA FLOW DIAGRAM**

A data flow diagram (DFD) is a graphical representation of the "flow" of data through an information system, modeling its process aspects. A DFD is often used as a preliminary step to create an overview of the system, which can later be elaborated.

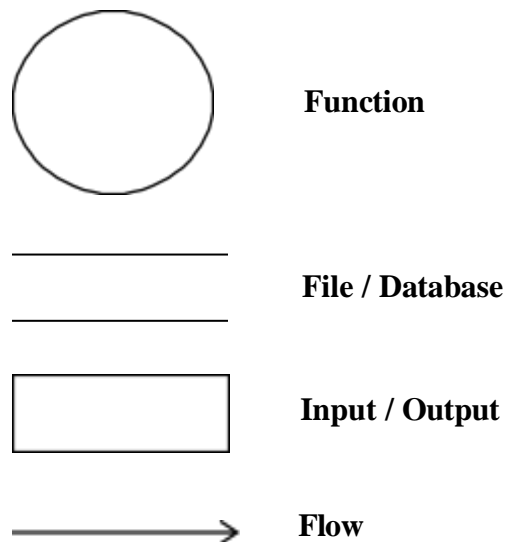
DFD can also be used for the visualization of data processing (structured design). A DFD shows what kind of information will be input to and output from the system, where the data will come from and go to, and where the data will be stored. It does not show information about the timing of the process or information about whether processes will operate in sequence or in parallel (which is shown on a flowchart).

This context-level DFD is next "exploded", to produce a Level 1 DFD that shows some of the detail of the system being modeled. The Level 1 DFD shows how the system is divided into sub-systems (processes), each of which deals with one or more of the data flows to or from an external agent, and which together provide all of the functionality of the system as a whole. It also identifies internal data stores that must be present for the system to do its job and shows the flow of data between the various parts of the system.

Data flow diagrams are one of the three essential perspectives of the structured-systems analysis and design method SSADM. The sponsor of a project and the end users will need to be briefed and consulted throughout all stages of a system's evolution. With a data flow diagram, users can visualize how the system will operate, what the system will accomplish, and how the system will be implemented. The old system's data-flow diagrams can be drawn up and compared with.

How any system is developed can be determined through a data flow diagram model. In the book of developing a set of leveled data flow diagrams, the analyst/designer is forced to address how the system may be decomposed into component sub-systems and to identify the transaction data in the data model. Data flow diagrams can be used in both the Analysis and Design phase of the SDLC. There are different notations to draw data flow diagrams. Defining different visual representations for processes, data stores, data flow, and external entities.

### **DFD Notation:**



### **DFD Example:**



### **Steps to Construct Data Flow Diagram:**

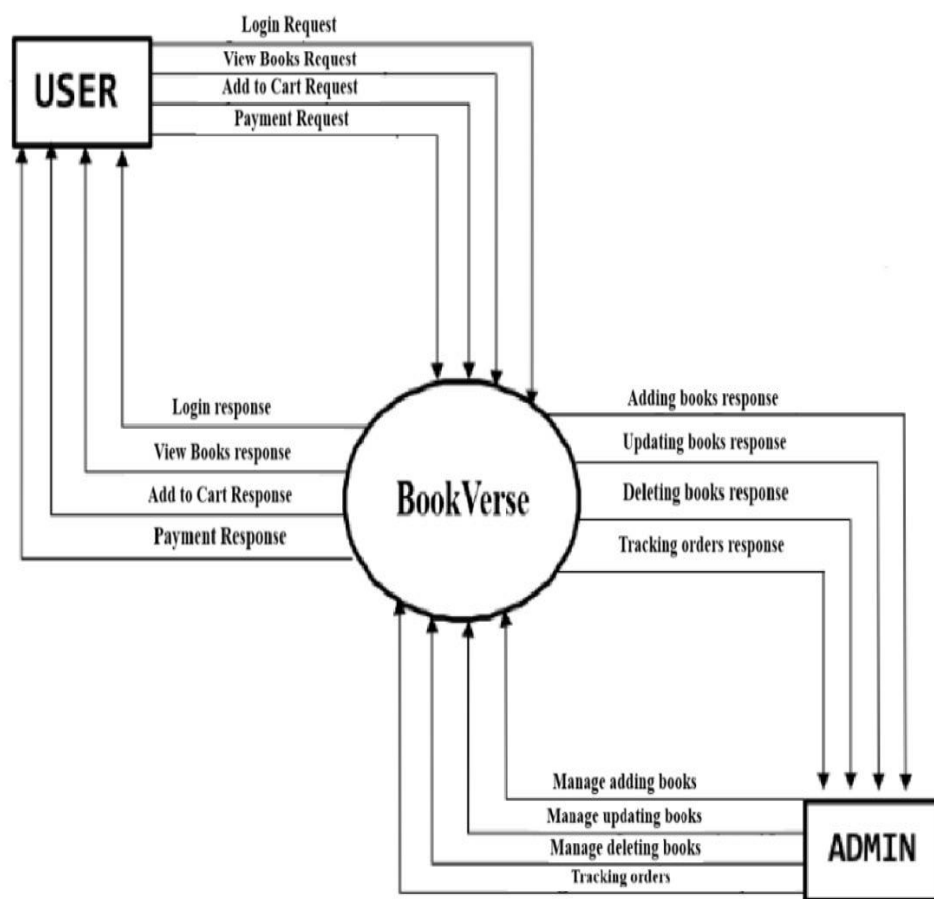
Four Steps are generally used to construct a DFD.

- ☐ Process should be named and referred for easy reference. Each name should be representative of the reference.
- ☐ The destination of flow is from top to bottom and from left to right.
- ☐ When a process is distributed into lower-level details they are numbered.
- ☐ The names of data stores, sources, and destinations are written in capital letters.

## Rules for constructing a Data Flow Diagram:

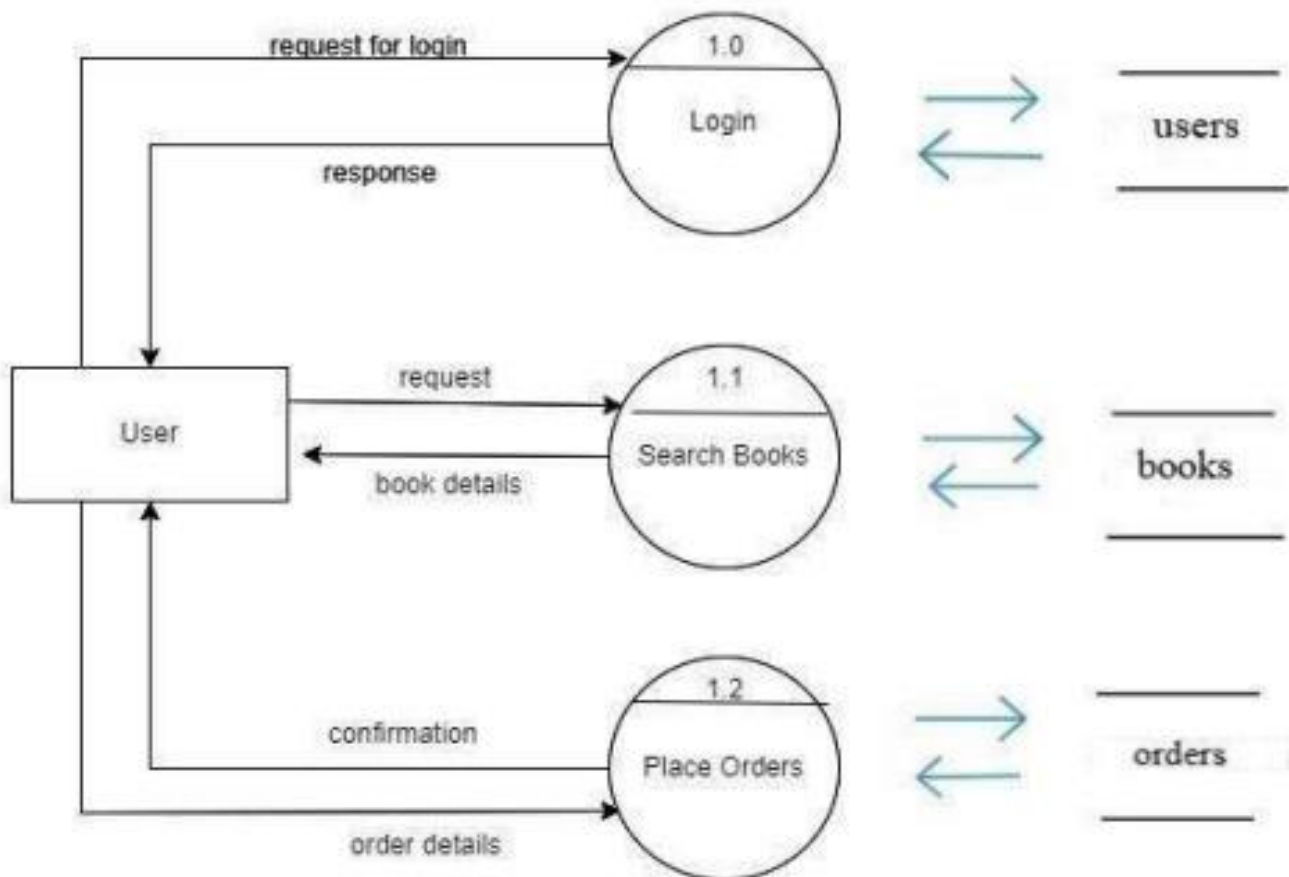
- Arrows should not cross each other.
- Squares, Circles, and Files must bear a name.
- Decomposed data flow squares and circles can have the same names.
- Draw all data flow around the outside of the diagram.

### • LEVEL 0 DFD OR CONTEXT DIAGRAM:

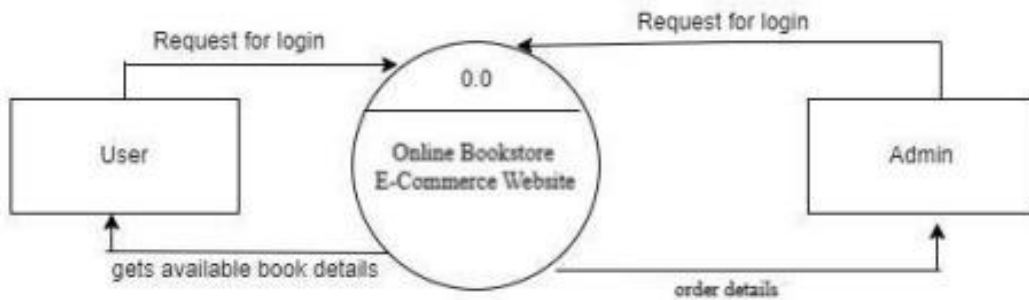




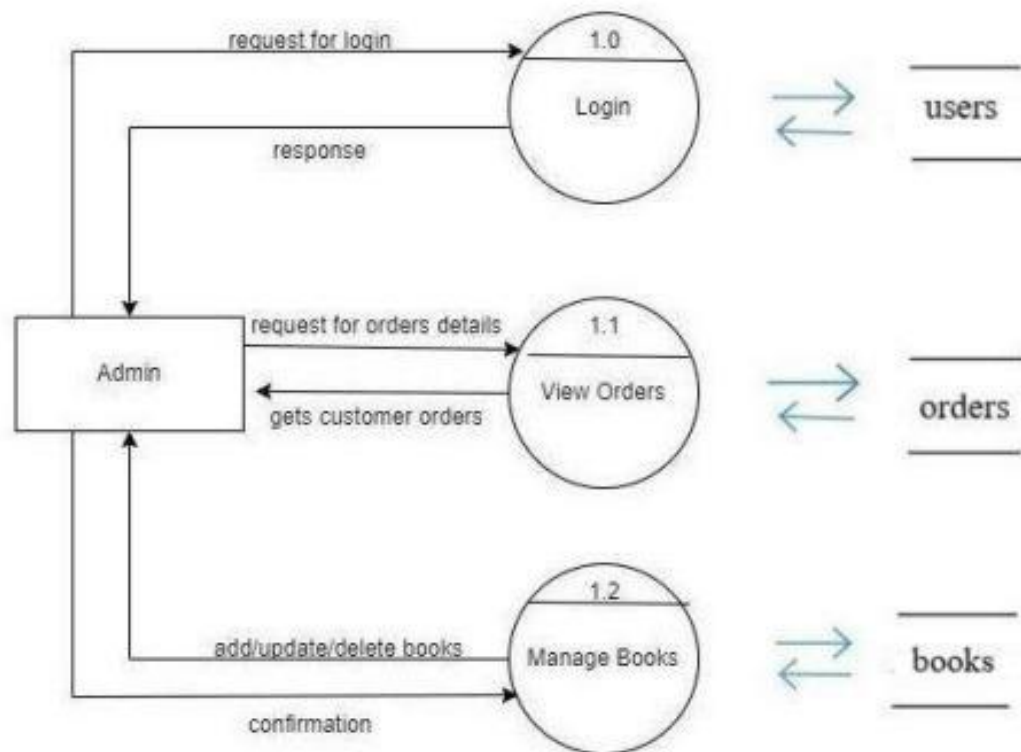
- LEVEL 1 DFD:



**LEVEL 1 DFD FOR USER**



**Level 0**



**LEVEL 1 DFD FOR ADMIN**

#### **4B.SEQUENCE DIAGRAM**

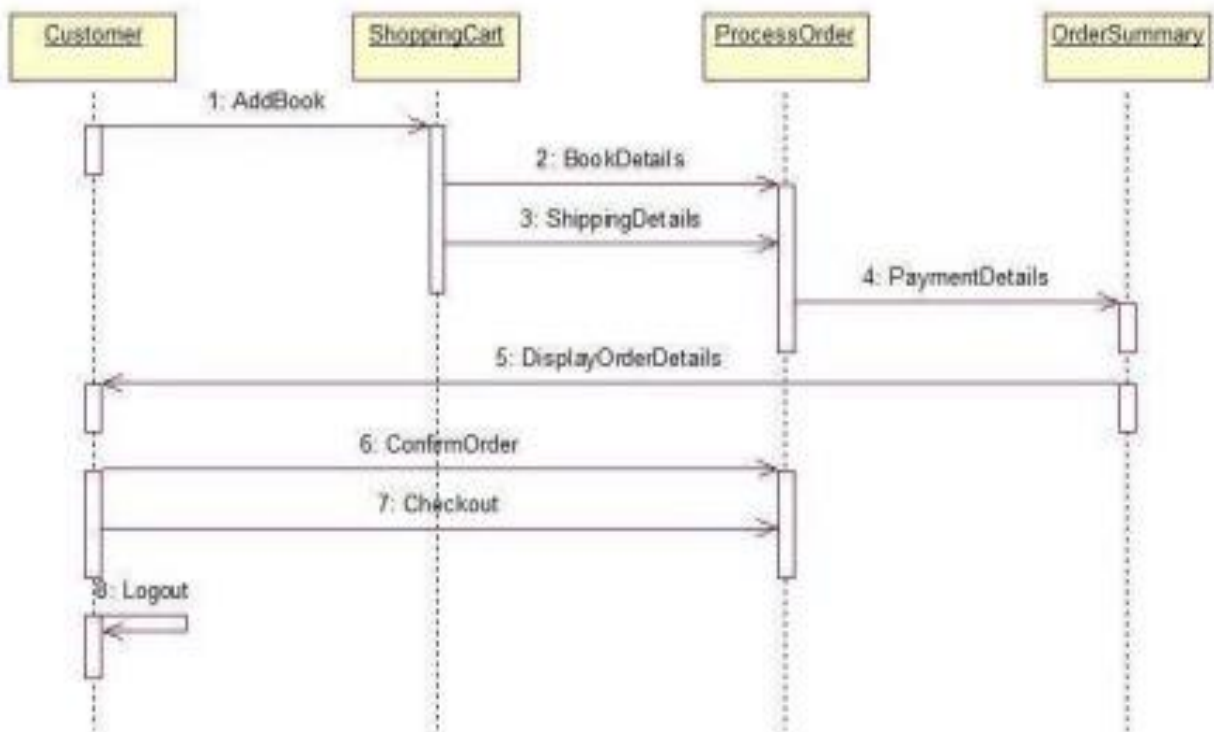
Sequence diagrams can be useful reference diagrams for businesses and other organizations. Try drawing a sequence diagram to:

- Represent the details of a UML use case.
- Model the logic of a sophisticated procedure, function, or operation.
- See how tasks are moved between objects or components of a process.
- Plan and understand the detailed functionality of an existing or future scenario.

#### Popular Sequence Diagram Uses

- Usage Scenario – A usage scenario is a diagram of how your system could potentially be used. It's a great way to make sure that you have worked through the logic of every usage scenario for the system.
- Method Logic - Just as you might use a UML sequence diagram to explore the logic of a use case, you can use it to Usage Scenario - A usage scenario is a diagram of how your system could potentially be used. It's a great way to explore the logic of any function, procedure, or complex process.

- Service Logic - If you consider a service to be a high-level method used by different clients, a sequence diagram is an ideal way to map that out.



A Use case diagram at its simplest is a representation of a user's interaction with the system that shows the relationship between the user and the different use cases in which the user is involved. A use case diagram can identify the different types of users of a system and the different use cases and will often be accompanied by other types of diagrams as well.

So only static behavior is not sufficient to model a system rather dynamic behavior is more important than static behavior. In UML there are five diagrams available to model dynamic nature and a use case diagram is one of them. Now as we have to discuss that the use case diagram is dynamic in nature there should be some internal or external factors for making the interaction.

These internal and external agents are known as actors. So, use case diagrams consist of actors, use cases, and their relationships. The diagram is used to model the system/subsystem of an application. A single-use case diagram captures a particular functionality of a system.

So, to model the entire system numbers of use case diagrams are used. The purpose of a use case diagram is to capture the dynamic aspect of a system. But this definition is too generic to describe the purpose.

Because the other four diagrams (activity, sequence, collaboration, and State chart) are also having the same purpose. So, we will look into some specific purpose that will distinguish it from the other four diagrams.

Use case diagrams are used to gather the requirements of a system including internal and external influences. These requirements are mostly design requirements. So, when a system is analyzed to gather its functionalities use cases are prepared and actors are identified.

Now when the initial task is complete use case diagrams are modeled to present the outside view. So, in brief, the purposes of use case diagrams can be as follows:

- ☐ Used to gather requirements of a system.
- ☐ Used to get an outside view of a system.
- ☐ Identify external and internal factors influencing the system.

## **How to draw Use Case Diagram?**

Use case diagrams are considered for high level requirement analysis of a system. So, when the requirements of a system are analyzed, the functionalities are captured in use cases.

So, we can say that uses cases are nothing but the system functionalities written in an organized manner. Now the second things which are relevant to the use cases are the actors. Actors can be defined as something that interacts with the system.

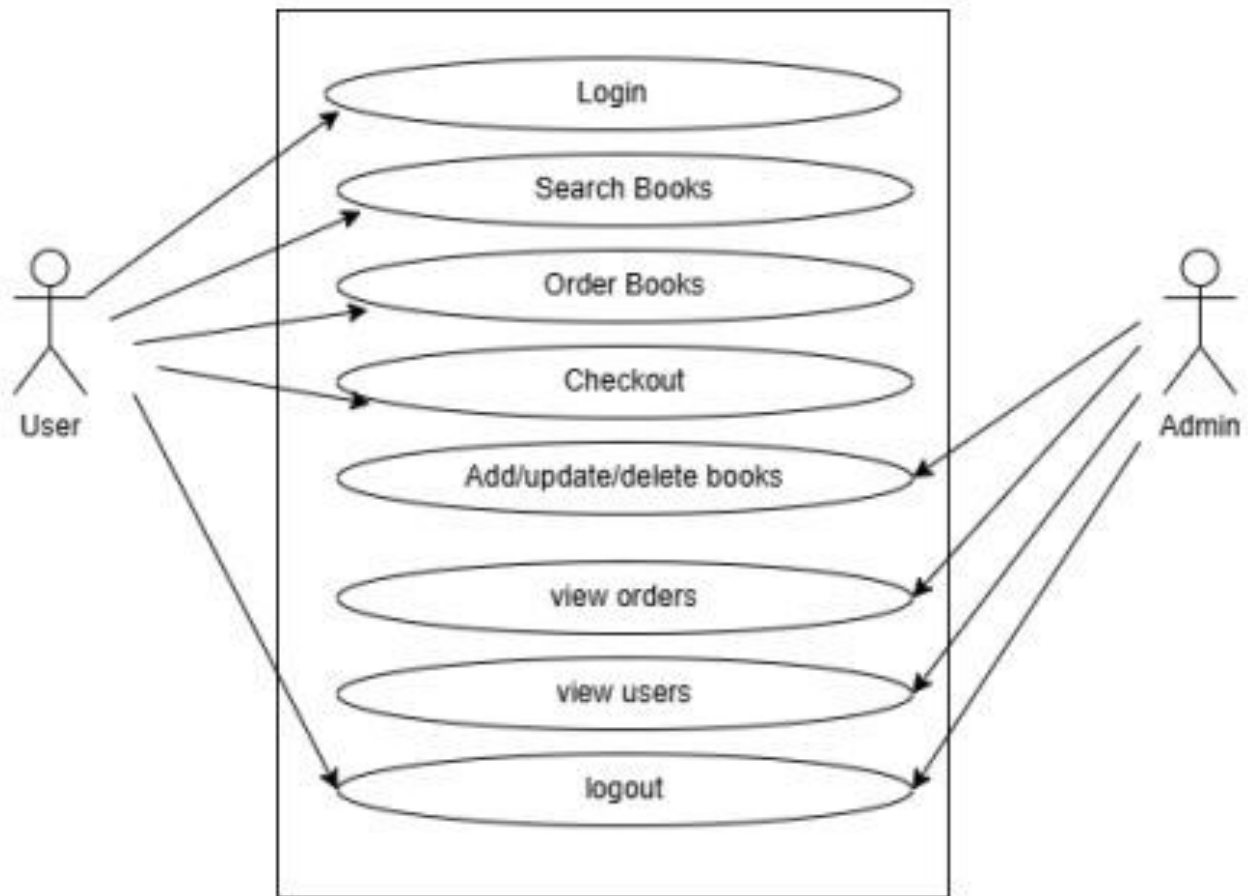
The actors can be human user, some internal applications or may be some external applications. So, in a brief when we are planning to draw use case diagram, we should have the following items identified.

- ☐ Functionalities to be represented as a use case
- ☐ Actors
- ☐ Relationships among the use cases and actors.

Use case diagrams are drawn to capture the functional requirements of a system. So, after identifying the above items we have to follow the following guidelines to draw an efficient use case diagram.

- ☐ The name of a use case is very important. So, the name should be chosen in such a way so that it can identify the functionalities performed.
- ☐ Give a suitable name for actors.
- ☐ Show relationships and dependencies clearly in the diagram.
- ☐ Do not try to include all types of relationships. Because the main purpose of the diagram is to identify requirements.
- ☐ Use note whenever required to clarify some important point

- **USE CASE DIAGRAM:**

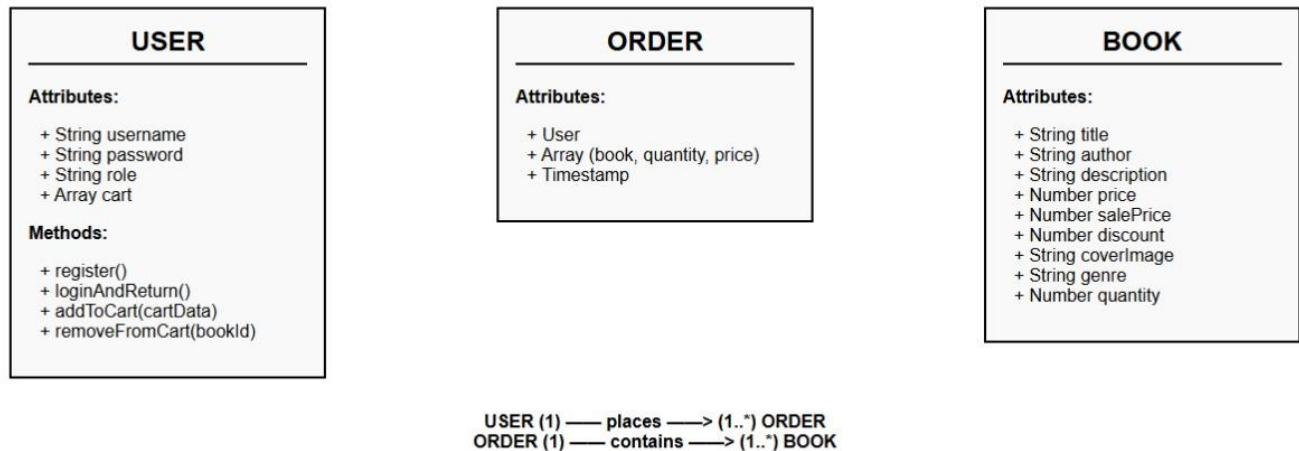


## 4C.SCHEMA DIAGRAM

The schema is an abstract structure or outline representing the logical view of the database as a whole. Defining categories of data and relationships between those categories, database schema design makes data much easier to retrieve, consume, manipulate, and interpret.

DB schema design organizes data into separate entities, determines how to create relationships between organized entities, and influences the applications of constraints on data. Designers create database schema to give other database users, such as programmers and analysts, a logical understanding of data.

### BookVerse Schema Design

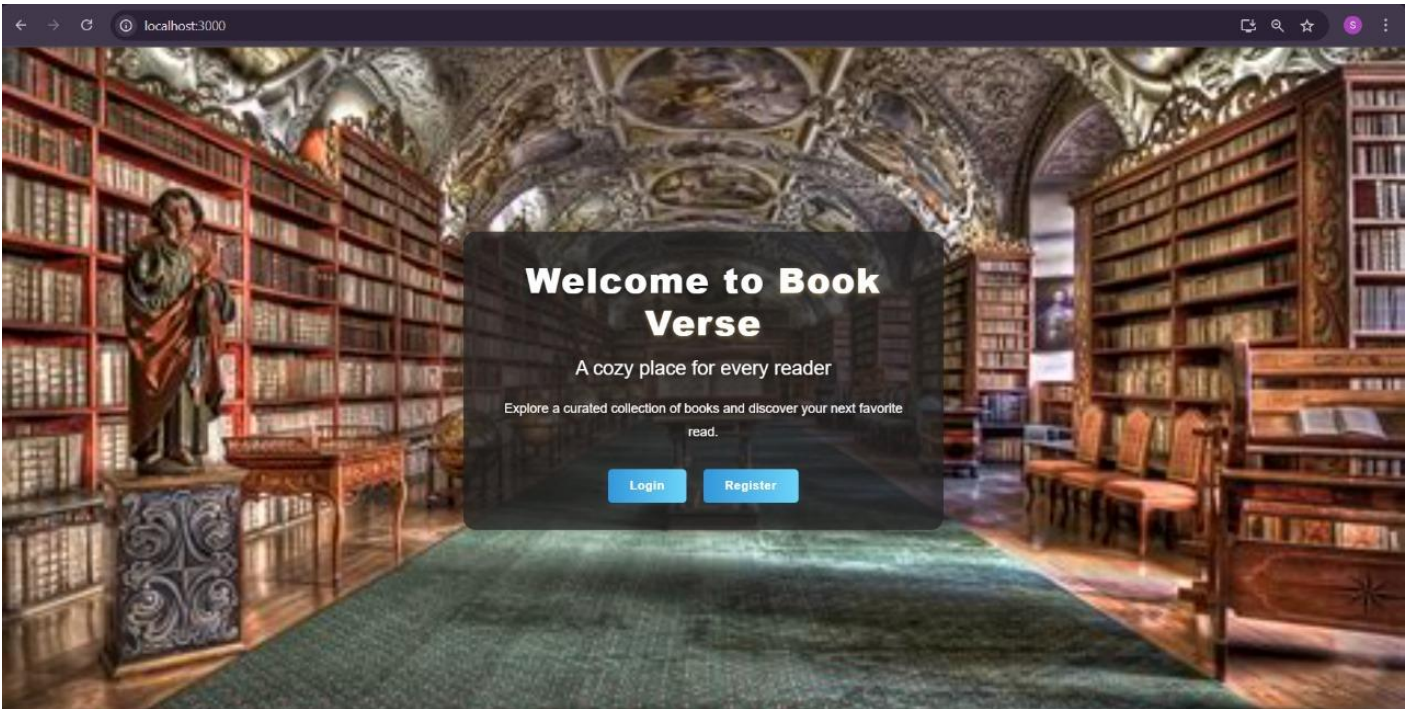




## 5. UI SNAPSHOT

### ❖ FRONTEND :-

#### 1) Home Page:



### ✓ CODE

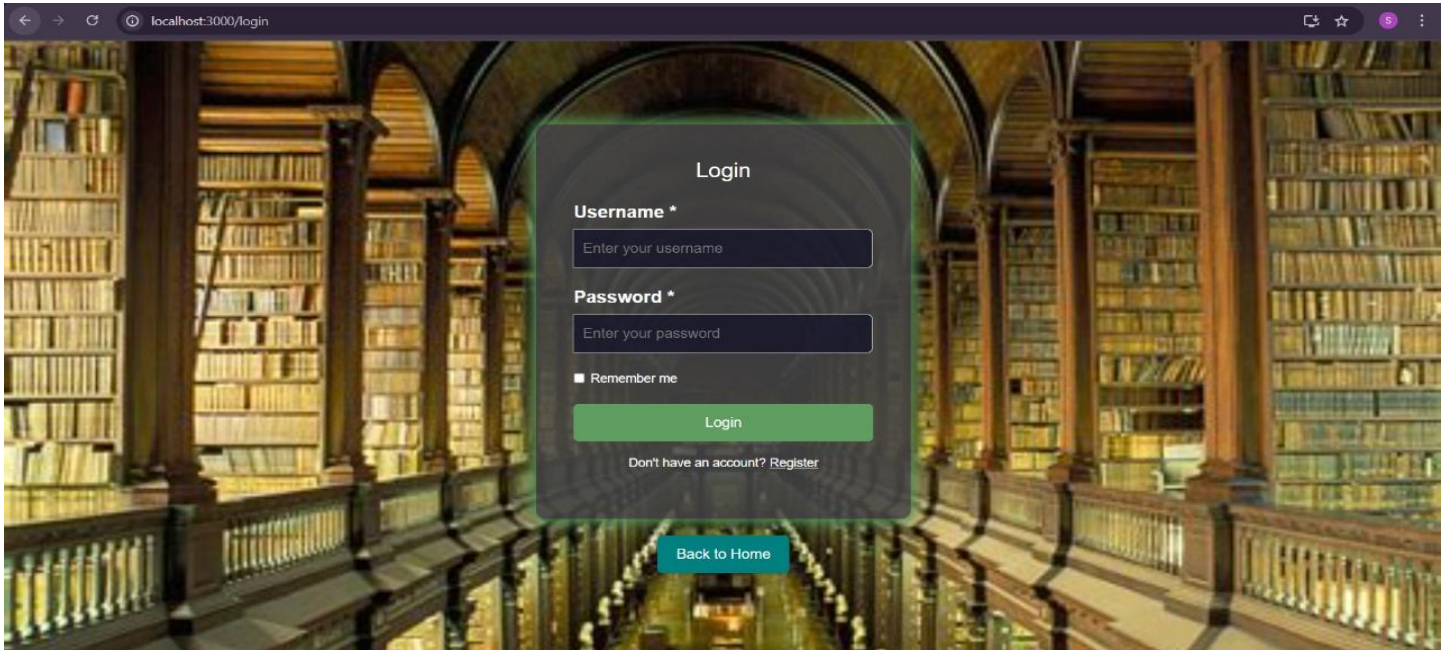
```
import React from 'react';
import { Link } from 'react-router-dom';
import './Home.css';

const Home = () => {
  return (
    <div className="home-bg">
      <div className="home-container">
        <header className="home-content">
          <h1>Welcome to <span className="brand">Book Verse</span></h1>
          <h4 className="subtitle">A cozy place for every reader</h4>
          <p className="description">
            Explore a curated collection of books and discover your next favorite read.
          </p>
          <div className="auth-links">
            <Link to="/login" className="auth-link">Login</Link>
            <Link to="/register" className="auth-link">Register</Link>
          </div>
        </header>
      </div>
    </div>
  )
}
```

```
);  
};
```

```
export default Home;
```

## 2) Login Page:



### ✓ CODE

```
import React, { useState, useEffect, useContext } from 'react';  
import axios from 'axios';  
import { useNavigate } from 'react-router-dom';  
import { AuthContext } from '../context/AuthContext';  
import './Login.css';
```

```
// Import local images
```

```
import bg1 from '../components/bgg1.jpg';  
import bg2 from '../components/bgg2.jpg';  
import bg3 from '../components/bgg3.jpg';  
import bg4 from '../components/bgg4.jpg';
```

```
const images = [bg1, bg2, bg3, bg4];
```

```
const Login = () => {  
  const [username, setUsername] = useState("");  
  const [password, setPassword] = useState("");  
  const [bgImage, setBgImage] = useState(images[0]);  
  const [error, setError] = useState("");  
  const { login } = useContext(AuthContext);  
  const navigate = useNavigate();
```

```

// Cycle background images every 5 seconds
useEffect(() => {
  let index = 0;
  const interval = setInterval(() => {
    index = (index + 1) % images.length;
    setBgImage(images[index]);
  }, 5000);
  return () => clearInterval(interval);
}, []);

const handleSubmit = async (e) => {
  e.preventDefault();
  setError(""); // Clear previous errors

  try {
    const response = await axios.post('http://localhost:5000/api/auth/login', {
      username,
      password,
    });

    const userData = {
      token: response.data.token,
      role: response.data.role,
      id: response.data.id,
    };
    localStorage.setItem('user', JSON.stringify(userData));
    login(response.data);

    if (response.data.role === 'admin') {
      navigate('/admin');
    } else {
      navigate('/Home1');
    }
  } catch (err) {
    if (err.response?.status === 400) {
      setError('Incorrect username or password.');
```

```

    {error && (
    <div className="error-message">
    {error}
    </div>
    )}

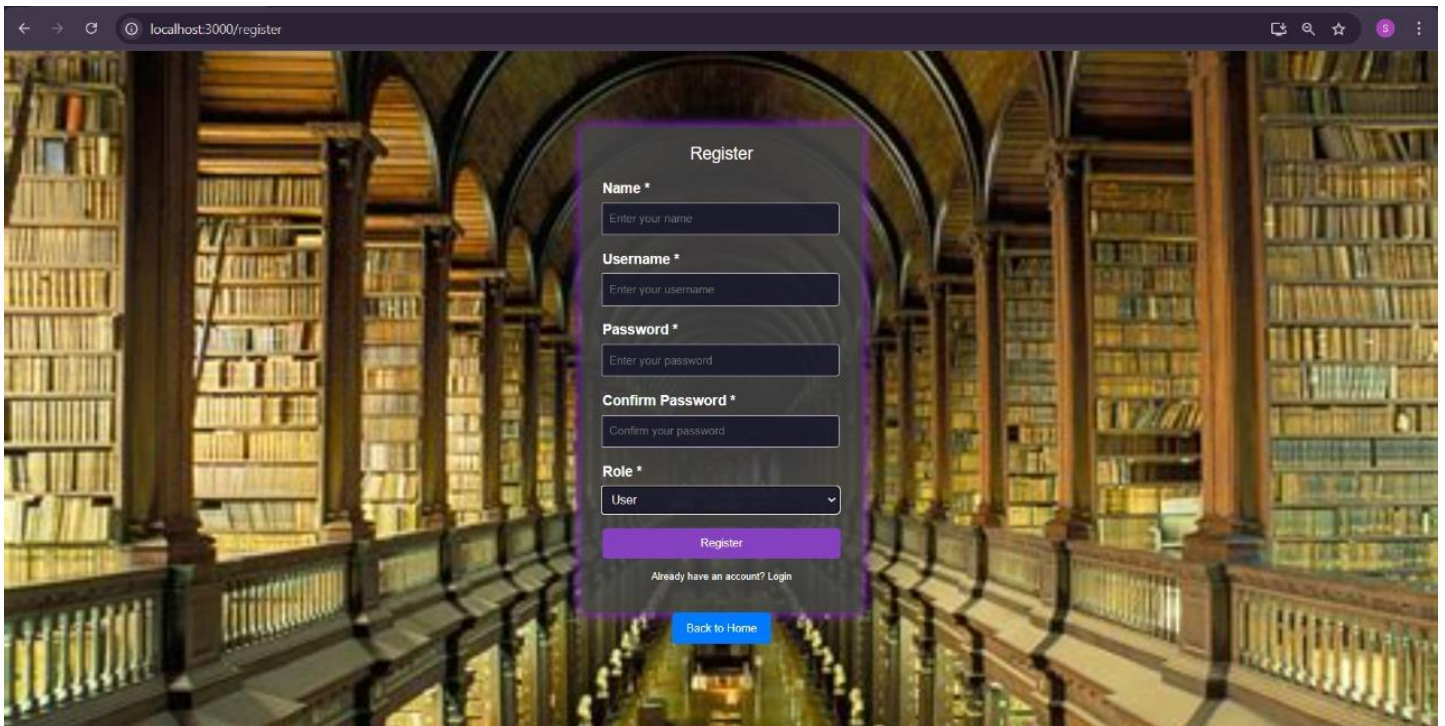
    <form onSubmit={handleSubmit}>
    <div className="input-group">
    <label htmlFor="username">Username<span className='red'> *</span></label>
    <input
    type="text"
    id="username"
    placeholder="Enter your username"
    value={username}
    onChange={(e) => setUsername(e.target.value)}
    required
    />
    </div>
    <div className="input-group">
    <label htmlFor="password">Password<span className='red'> *</span></label>
    <input
    type="password"
    id="password"
    placeholder="Enter your password"
    value={password}
    onChange={(e) => setPassword(e.target.value)}
    required
    />
    </div>
    <div className="remember-me">
    <label>
    <input type="checkbox" /> Remember me
    </label>
    </div>
    <button type="submit" className="login-button">Login</button>
    </form>
    <div className="register-link">
    <p>Don't have an account? <a href="/register" className='register-btn'>Register</a></p>
    </div>
    </div>
    <div className="back-button-container">
    <button onClick={() => navigate('/')} className="back-button-login">
    Back to Home
    </button>
    </div>
    </div>
  );
};

```



export default Login;

### 3) Register Page:



#### ✓ CODE

```
// Register.js
import React, { useState, useEffect } from 'react';
import axios from 'axios';
import { useNavigate } from 'react-router-dom';
import './Register.css';

// Import local background images
import bg1 from '../components/bgg1.jpg';
import bg2 from '../components/bgg2.jpg';
import bg3 from '../components/bgg3.jpg';
import bg4 from '../components/bgg4.jpg';

const images = [bg1, bg2, bg3, bg4];

const Register = () => {
  const [name, setName] = useState("");
  const [username, setUsername] = useState("");
  const [password, setPassword] = useState("");
  const [confirmPassword, setConfirmPassword] = useState("");
  const [role, setRole] = useState('user');
```

```

const [adminPassword, setAdminPassword] = useState("");
const [error, setError] = useState("");
const [bgImage, setBgImage] = useState(images[0]);
const navigate = useNavigate();

// Background image slideshow
useEffect(() => {
  let index = 0;
  const interval = setInterval(() => {
    index = (index + 1) % images.length;
    setBgImage(images[index]);
  }, 5000);
  return () => clearInterval(interval);
}, []);

const handleSubmit = async (e) => {
  e.preventDefault();
  if (password !== confirmPassword) {
    alert('Passwords do not match');
    return;
  }
  if (role === 'admin' && adminPassword !== 'Admin@123') {
    setError('Invalid admin password');
    return;
  }

  try {
    const response = await axios.post('http://localhost:5000/api/auth/register', {
      username,
      password,
      role,
    });
    console.log('Registration successful:', response.data);
    navigate('/login');
  } catch (err) {
    console.error('Registration error:', err.response?.data || err.message);
    setError(err.response?.data?.message || 'Registration failed. Please try again.');
```

```

  }
};

return (
  <div
    className="register-container"
    style={{ backgroundImage: url(`${bgImage}`) }}
  >
    <div className="register-box">
      <h1>Register</h1>
      <form onSubmit={handleSubmit}>
        <div className="input-group">
          <label htmlFor="name" className="params">
```

```
Name<span className="red"> *</span>
</label>
<input
type="text"
id="name"
placeholder="Enter your name"
value={name}
onChange={(e) => setName(e.target.value)}
/>
</div>
<div className="input-group">
<label htmlFor="username" className="params">
Username<span className="red"> *</span>
</label>
<input
type="text"
id="username"
placeholder="Enter your username"
value={username}
onChange={(e) => setUsername(e.target.value)}
/>
</div>
<div className="input-group">
<label htmlFor="password" className="params">
Password<span className="red"> *</span>
</label>
<input
type="password"
id="password"
placeholder="Enter your password"
value={password}
onChange={(e) => setPassword(e.target.value)}
/>
</div>
<div className="input-group">
<label htmlFor="confirmPassword" className="params">
Confirm Password<span className="red"> *</span>
</label>
<input
type="password"
id="confirmPassword"
placeholder="Confirm your password"
value={confirmPassword}
onChange={(e) => setConfirmPassword(e.target.value)}
/>
</div>
<div className="input-group">
<label htmlFor="role">
Role<span className="red"> *</span>
</label>
```

```

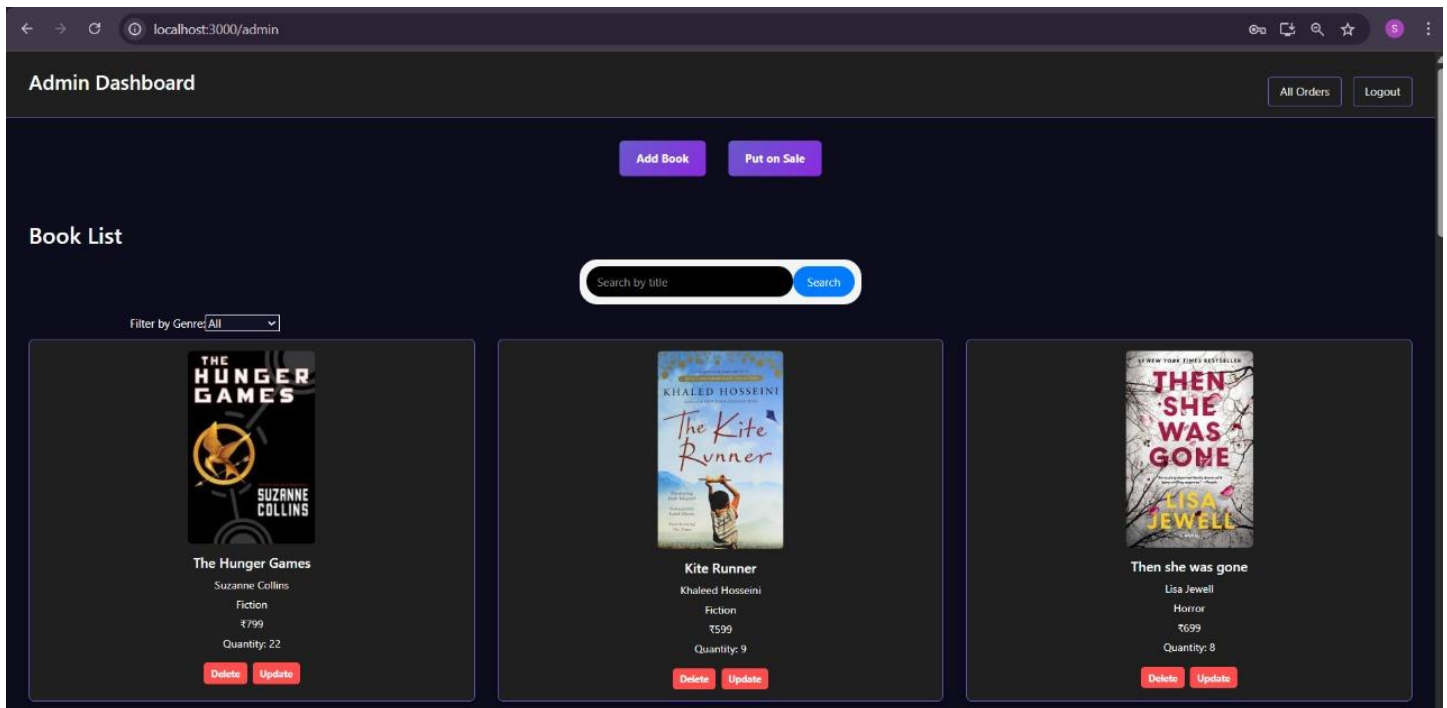
<select
id="role"
value={role}
onChange={(e) => setRole(e.target.value)}
>
<option value="user">User</option>
<option value="admin">Admin</option>
</select>
</div>
{role === 'admin' && (
<div className="input-group">
<label htmlFor="adminPassword">
Admin Password<span className="red"> *</span>
</label>
<input
type="password"
id="adminPassword"
placeholder="Enter admin password"
value={adminPassword}
onChange={(e) => setAdminPassword(e.target.value)}
/>
</div>
)}
{error && <p className="error-message">{error}</p>}
<button type="submit" className="register-button">
Register
</button>
</form>
<div className="login-link">
<p>
Already have an account? <a href="/login">Login</a>
</p>
</div>
</div>
<button onClick={() => navigate('/') } className="back-button">
Back to Home
</button>
</div>
);
};

export default Register;

```



#### 4) Admin Dashboard:



#### ✓ CODE

```
import React, { useState, useEffect, useContext } from 'react';
import axios from 'axios';
import { AuthContext } from '../context/AuthContext';
import { useNavigate } from 'react-router-dom';
import UpdateBookForm from './UpdateBookForm';
import './AdminDashboard.css';
import AdminSearchBar from './AdminSearchBar';
import AdminFilterBar from './AdminFilterBar';
```

```
const AdminDashboard = () => {
  const [selectedBook, setSelectedBook] = useState(null);
  const [books, setBooks] = useState([]);
  const [filteredBooks, setFilteredBooks] = useState([]);
  const [newBook, setNewBook] = useState({
    title: "",
    author: "",
    genre: "",
    description: "",
    price: "",
    coverImage: "",
    quantity: 0,
  });
```

```
  const [searchQuery, setSearchQuery] = useState("");
```

```

const [selectedGenre, setSelectedGenre] = useState("");

const { user, logout } = useContext(AuthContext);
const navigate = useNavigate();

const fetchBooks = async () => {
  try {
    const response = await axios.get('http://localhost:5000/api/books');
    setBooks(response.data);
    setFilteredBooks(response.data);
  } catch (err) {
    console.error('Error fetching books:', err);
  }
};

useEffect(() => {
  fetchBooks();
}, []);

useEffect(() => {
  let result = books;

  if (searchQuery) {
    result = result.filter((book) =>
      book.title.toLowerCase().includes(searchQuery.toLowerCase())
    );
  }

  if (selectedGenre) {
    result = result.filter(
      (book) => book.genre.toLowerCase() === selectedGenre.toLowerCase()
    );
  }

  setFilteredBooks(result);
}, [searchQuery, selectedGenre, books]);

const handleDeleteBook = async (bookId) => {
  try {
    await axios.delete('http://localhost:5000/api/books/${bookId}', {
      headers: {
        Authorization: Bearer ${user.token},
      },
    });
    fetchBooks();
  } catch (err) {
    console.error('Error deleting book:', err);
  }
};

```

```

const handleUpdateClick = (book) => {
  setSelectedBook(book);
};

const handleUpdateComplete = () => {
  setSelectedBook(null);
  fetchBooks();
};

const handleLogout = () => {
  logout();
  navigate('/login');
};

const handleSearch = (title) => {
  setSearchQuery(title);
};

const handleFilter = (genre) => {
  setSelectedGenre(genre);
};

return (
  <div className="admin-dashboard">
    <header className="admin-header">
      <h1>Admin Dashboard</h1>
      <div className="button-container">
        <button className="all-orders-button" onClick={() => navigate('/all-orders')}>
          All Orders
        </button>
        <button className="logout-button" onClick={handleLogout}>
          Logout
        </button>
      </div>
    </header>

    <div className="add-book-box">
      <button className="add-book-btn" onClick={() => navigate('/add-book')}>
        Add Book
      </button>
      <button className="put-on-sale-btn" onClick={() => navigate('/put-on-sale')}>
        Put on Sale
      </button>
    </div>

    <div className="book-container">
      <h2 className="book_list">Book List</h2>
      <AdminSearchBar onSearch={handleSearch} />
      <AdminFilterBar onFilter={handleFilter} />
      <div className="book-grid">

```

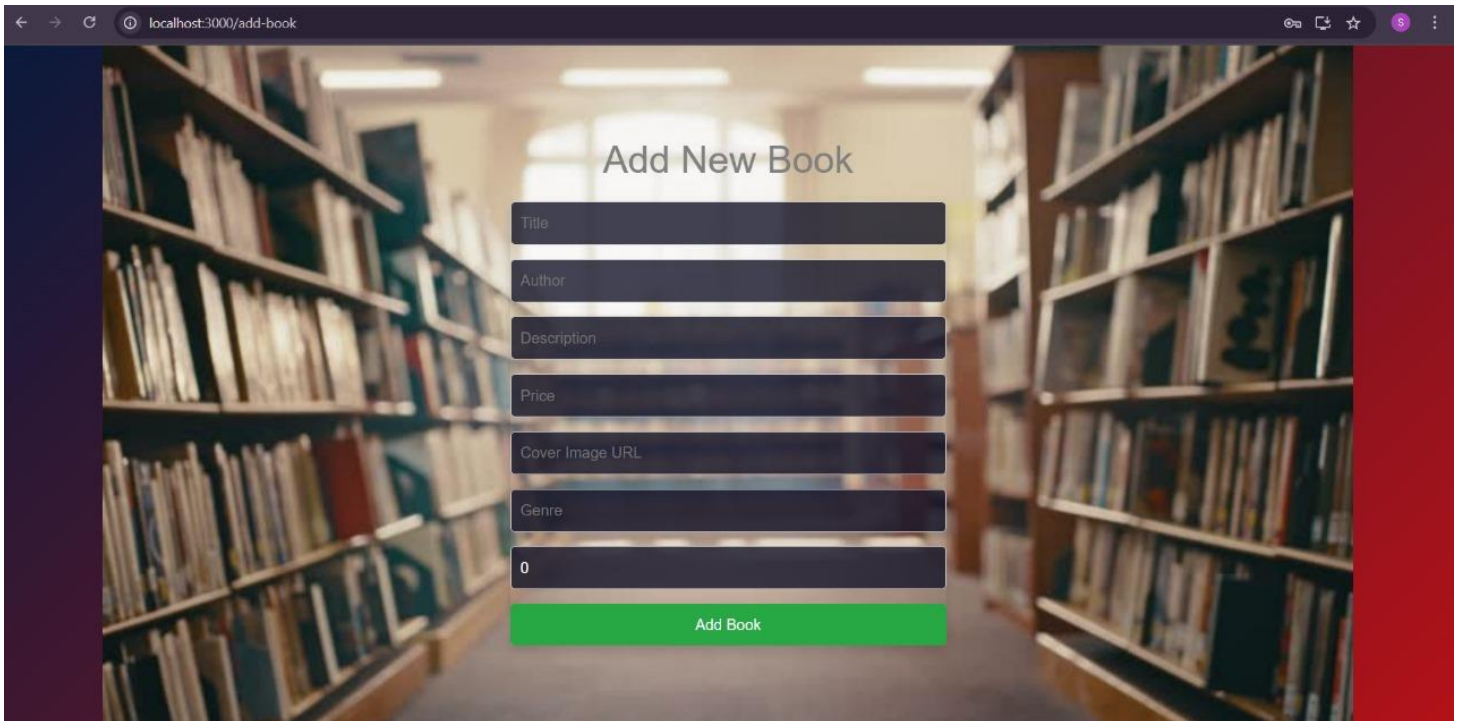
```

    {filteredBooks.map((book) => (
      <div key={book._id} className="book-item">
        {selectedBook && selectedBook._id === book._id ? (
          <UpdateBookForm book={selectedBook} onUpdate={handleUpdateComplete} />
        ) : (
          <div className="Book-Container">
            <img src={book.coverImage} alt={book.title} className="Book-Image" />
            <div className="book-details">
              <h3 className="book_title">{book.title}</h3>
              <p className="book_author">{book.author}</p>
              <p className="book_genre">{book.genre}</p>
              <p className="book_desc">{book.description}</p>
              <p className="book_price">₹{book.price}</p>
              <p className="book_quantity">Quantity: {book.quantity}</p>
              <div className="button-row">
                <button
                  onClick={() => handleDeleteBook(book._id)}
                  className="book-delete-btn"
                >
                  Delete
                </button>
                <button
                  onClick={() => handleUpdateClick(book)}
                  className="book-update-button"
                >
                  Update
                </button>
              </div>
            </div>
          </div>
        )}
      </div>
    ))}
  </div>
</div>
);
};

```

```
export default AdminDashboard;
```

#### 4) Add New Books(Admin):



The screenshot shows a web browser window with the address bar displaying 'localhost:3000/add-book'. The main content area features a background image of a library aisle with bookshelves. Overlaid on this is a form titled 'Add New Book'. The form contains seven input fields: 'Title', 'Author', 'Description', 'Price', 'Cover Image URL', 'Genre', and a quantity field with the value '0'. A green button labeled 'Add Book' is positioned at the bottom of the form.

#### ✓ CODE

```
import React, { useState } from 'react';
import axios from 'axios';
import { useNavigate } from 'react-router-dom';
import './AddBookForm.css';
```

```
const AddBookForm = () => {
  const [formData, setFormData] = useState({
    title: "",
    author: "",
    description: "",
    price: "",
    coverImage: "",
    genre: "",
    quantity: 0,
  });
  const navigate = useNavigate();
```

```
  const handleChange = (e) => {
    const { name, value } = e.target;
    setFormData({ ...formData, [name]: value });
  };
}
```

```

const handleSubmit = async (e) => {
  e.preventDefault();
  try {
    await axios.post('http://localhost:5000/api/books', formData, {
      headers: {
        Authorization: Bearer ${JSON.parse(localStorage.getItem('user')).token},
      },
    });
    navigate('/admin'); // Redirect to admin dashboard
  } catch (err) {
    console.error('Error adding book:', err);
  }
};

```

```

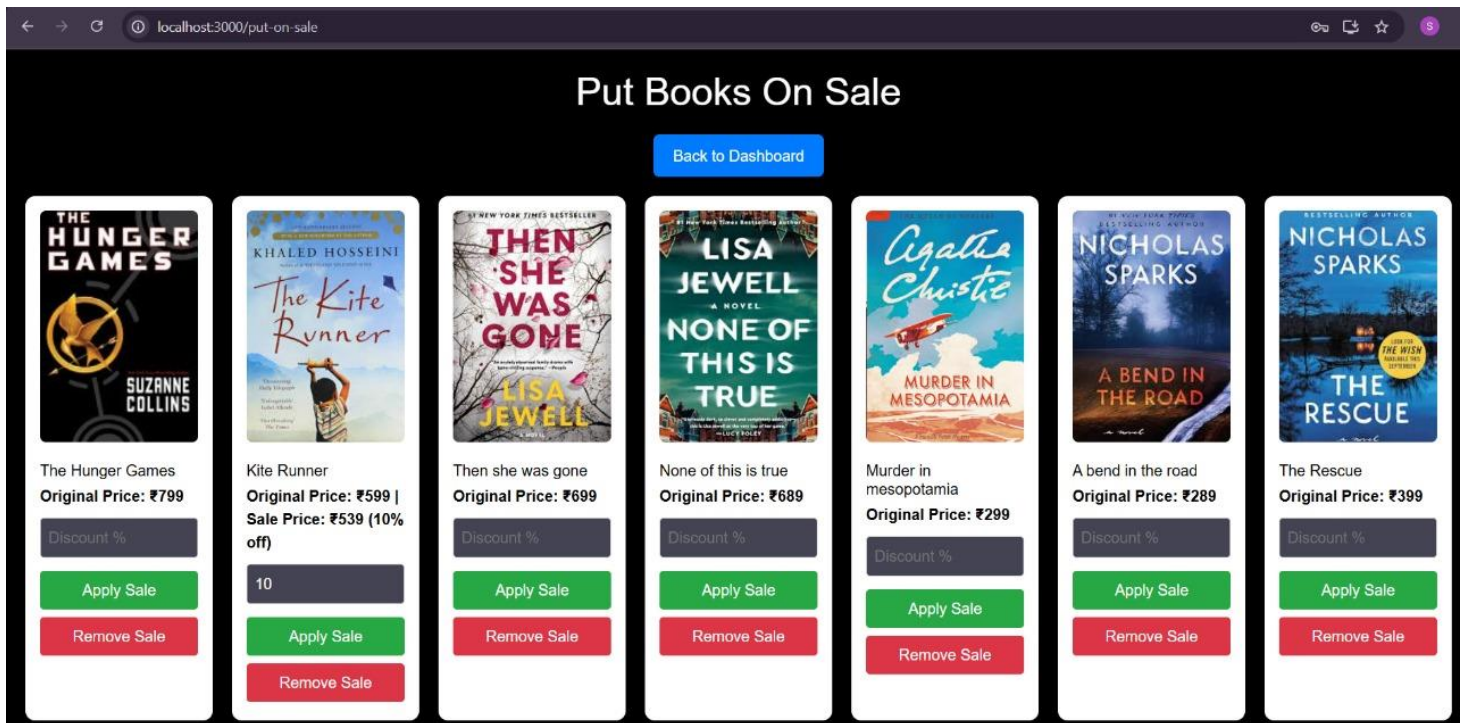
return (
  <div className='container'>
    <div className="add-book-form">
      <h1 class="nm">Add New Book</h1>
      <form onSubmit={handleSubmit}>
        <input
          type="text"
          name="title"
          placeholder="Title"
          value={formData.title}
          onChange={handleChange}
        />
        <input
          type="text"
          name="author"
          placeholder="Author"
          value={formData.author}
          onChange={handleChange}
        />
        <input
          type="text"
          name="description"
          placeholder="Description"
          maxLength="280"
          value={formData.description}
          onChange={handleChange}
        />
        <input
          type="number"
          name="price"
          placeholder="Price"
          value={formData.price}
          onChange={handleChange}
        />
        <input

```

```
type="text"
name="coverImage"
placeholder="Cover Image URL"
value={formData.coverImage}
onChange={handleChange}
/>
<input
type="text"
name="genre"
placeholder="Genre"
value={formData.genre}
onChange={handleChange}
/>
<input
type="number"
name="quantity"
placeholder="Quantity"
value={formData.quantity}
onChange={handleChange}
/>
<button type="submit">Add Book</button>
</form>
</div>
</div>
);
};

export default AddBookForm;
```

## 6) Put On Sale(Admin):



## ✓ CODE

```
import React, { useState, useEffect, useContext } from 'react';
import axios from 'axios';
import { AuthContext } from '../context/AuthContext';
import { useNavigate } from 'react-router-dom';
import './PutOnSale.css';

const PutOnSale = () => {
  const [books, setBooks] = useState([]);
  const [discounts, setDiscounts] = useState({});
  const { user } = useContext(AuthContext);
  const navigate = useNavigate();

  useEffect(() => {
    fetchBooks();
  }, []);

  const fetchBooks = async () => {
    try {
      const response = await axios.get('http://localhost:5000/api/books');
```



```

setBooks(response.data);

// ☒ Pre-fill discounts with existing discount from DB
const initialDiscounts = {};
response.data.forEach(book => {
  initialDiscounts[book._id] = book.discount || ""; // If no discount, use empty string
});
setDiscounts(initialDiscounts);

} catch (err) {
  console.error('Error fetching books:', err);
}
};

const handleDiscountChange = (bookId, value) => {
  setDiscounts({
    ...discounts,
    [bookId]: value
  });
};

const handlePutOnSale = async (bookId) => {
  try {
    const discount = discounts[bookId];
    if (!discount || discount <= 0 || discount >= 100) {
      alert('Please enter a valid discount percentage (1-99).');
      return;
    }

    await axios.patch(
      http://localhost:5000/api/books/${bookId}/sale,
      { discount: discount },
      {
        headers: {
          Authorization: Bearer ${user.token},
        },
      }
    );

    alert('Book marked as On Sale!');
    fetchBooks();
  } catch (err) {
    console.error('Error putting book on sale:', err);
  }
};

const handleRemoveSale = async (bookId) => {
  try {
    await axios.patch(
      http://localhost:5000/api/books/${bookId}/remove-sale,

```

```

    {}},
    {
      headers: {
        Authorization: Bearer ${user.token},
      },
    }
  );

  alert('Sale removed from book!');
  fetchBooks();
} catch (err) {
  console.error('Error removing sale:', err);
}
};
return (
  <div className="put-on-sale-page">
    <h1>Put Books On Sale</h1>
    <button className="back-button" onClick={() => navigate('/admin')}>
      Back to Dashboard
    </button>
    <div className="sale-book-list">
      {books.map((book) => (
        <div key={book._id} className="sale-book-item">
          <img src={book.coverImage} alt={book.title} className="sale-book-image" />
          <div className="sale-book-info">
            <h3>{book.title}</h3>
            <p>
              Original Price: ₹{book.price}
              {book.salePrice && (
                <div> | Sale Price: ₹{book.salePrice} ({book.discount}% off)</div>
              )}
            </p>
            <input
              type="number"
              placeholder="Discount %"
              value={discounts[book._id] || ""}
              onChange={(e) => handleDiscountChange(book._id, e.target.value)}
              min="1"
              max="99"
            />
            <button className="apply-button" onClick={() => handlePutOnSale(book._id)}>
              Apply Sale
            </button>
            <button className="remove-button" onClick={() => handleRemoveSale(book._id)}>
              Remove Sale
            </button>
          </div>
        </div>
      ))}
    </div>
  </div>
);

```

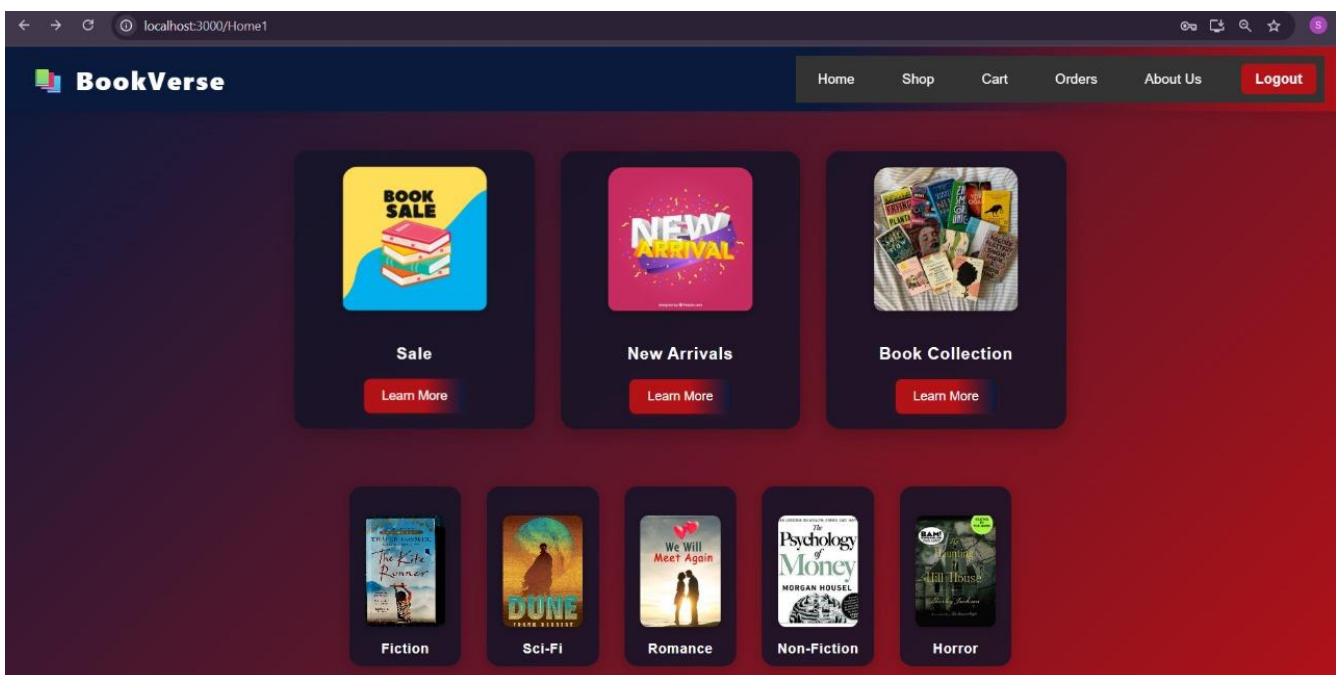
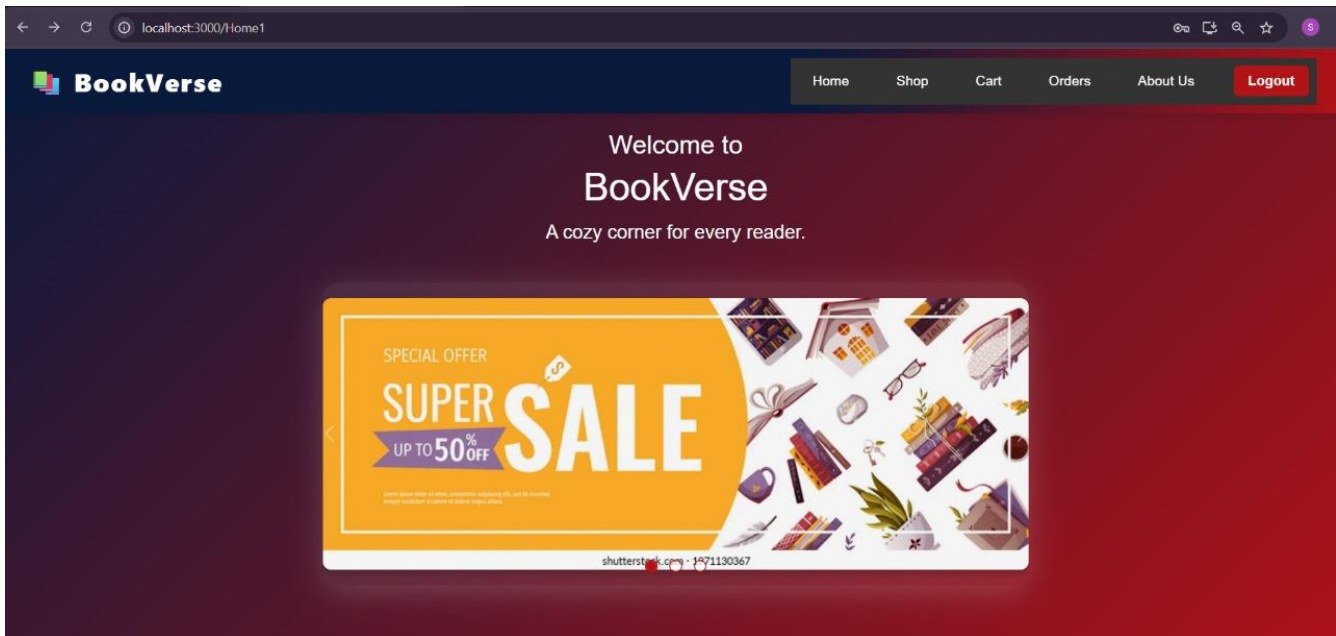
```

    </div>
  );
};

export default PutOnSale;

```

## 7) User Dashboard:



✓ CODE

```

import React from 'react';
import Header from './Header';
import './Home1.css';
import { useNavigate } from 'react-router-dom';

const Home1 = () => {
  const navigate = useNavigate();

  return (
    <div>
      <Header />
      <div className="home1-body">
        <h3>Welcome to</h3>
        <h1>BookVerse</h1>
        <p>A cozy corner for every reader.</p>

        <div
          id="carouselExampleCaptions"
          className="carousel slide"
          data-bs-ride="carousel"
          data-bs-interval="3000" // Optional: 3 seconds per slide
        >
          <div className="carousel-indicators">
            <button type="button" data-bs-target="#carouselExampleCaptions" data-bs-slide-to="0"
              className="active" aria-current="true" aria-label="Slide 1"></button>
            <button type="button" data-bs-target="#carouselExampleCaptions" data-bs-slide-to="1" aria-
              label="Slide 2"></button>
            <button type="button" data-bs-target="#carouselExampleCaptions" data-bs-slide-to="2" aria-
              label="Slide 3"></button>
          </div>
          <div className="carousel-inner">
            <div className="carousel-item active">
              
            </div>
            <div className="carousel-item">
              
            </div>
            <div className="carousel-item">
              
            </div>
          </div>
        </div>
      </div>
    </div>
  );
};

```

```
</div>
<button className="carousel-control-prev" type="button" data-bs-
target="#carouselExampleCaptions" data-bs-slide="prev">
<span className="carousel-control-prev-icon" aria-hidden="true"></span>
<span className="visually-hidden">Previous</span>
</button>
<button className="carousel-control-next" type="button" data-bs-
target="#carouselExampleCaptions" data-bs-slide="next">
<span className="carousel-control-next-icon" aria-hidden="true"></span>
<span className="visually-hidden">Next</span>
</button>
</div>
```

```
<div className="promo-container">
<div className="promo-card">
<div className="promo-image">

</div>
<div className="promo-content">
<div className="promo-title">Sale</div>
<button
className="promo-button"
onClick={() => navigate('/shop?sale=true')}
>
Learn More
</button>
</div>
</div>
```

```
<div className="promo-card">
<div className="promo-image">

</div>
<div className="promo-content">
<div className="promo-title">New Arrivals</div>
<button
className="promo-button"
onClick={() => navigate('/shop?sort=newest')}
>
Learn More
</button>
</div>
</div>
```

```
<div className="promo-card">
<div className="promo-image">

</div>
<div className="promo-content">
<div className="promo-title">Book Collection</div>
```

```
<button  
  className="promo-button"  
  onClick={() => navigate('/shop')}  
>
```

```
  Learn More
```

```
</button>
```

```
</div>
```

```
</div>
```

```
</div>
```

```
<div className="genre-container">
```

```
<div className="genre-box" onClick={() => navigate('/shop?genre=Fiction')}>
```

```

```

```
<div className="genre-label">Fiction</div>
```

```
</div>
```

```
<div className="genre-box" onClick={() => navigate('/shop?genre=Sci-Fi')}>
```

```

```

```
<div className="genre-label">Sci-Fi</div>
```

```
</div>
```

```
<div className="genre-box" onClick={() => navigate('/shop?genre=Romance')}>
```

```

```

```
<div className="genre-label">Romance</div>
```

```
</div>
```

```
<div className="genre-box" onClick={() => navigate('/shop?genre=Non-Fiction')}>
```

```

```

```
<div className="genre-label">Non-Fiction</div>
```

```
</div>
```

```
<div className="genre-box" onClick={() => navigate('/shop?genre=Horror')}>
```

```

```

```
<div className="genre-label">Horror</div>
```

```
</div>
```

```
</div>
```

```
</div>
```

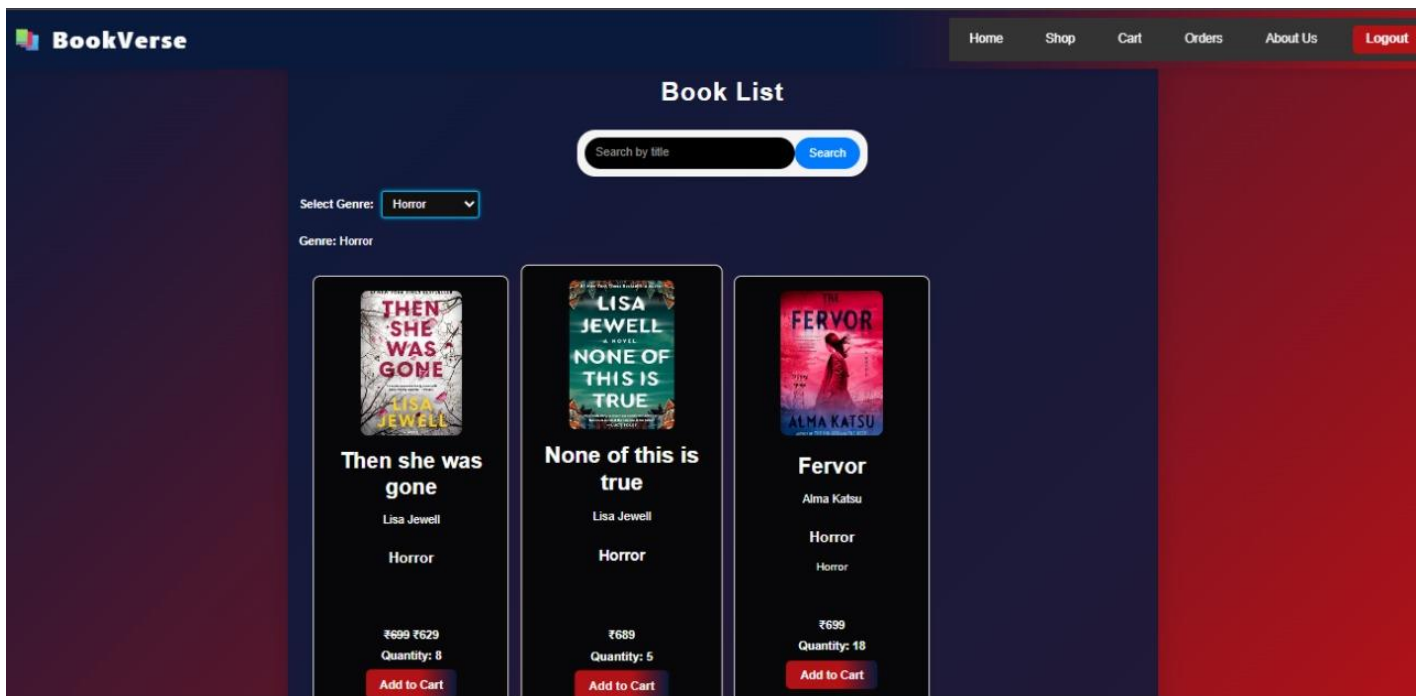
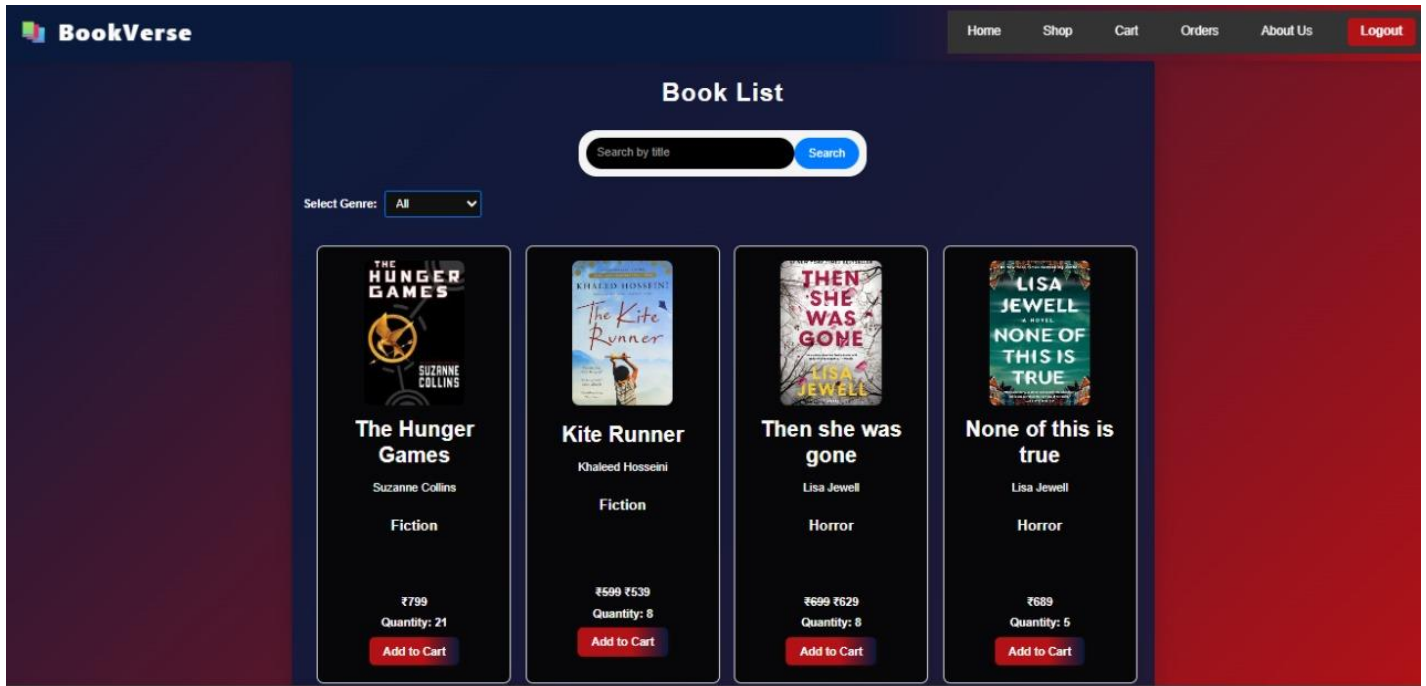
```
</div>
```

```
);
```

```
};
```

```
export default Home1;
```

## 8) Shop:



✓ CODE

```

import React, { useEffect, useState, useContext } from 'react';
import axios from 'axios';
import { CartContext } from '../context/CartContext';
import { AuthContext } from '../context/AuthContext';
import { useNavigate, useLocation } from 'react-router-dom';
import SearchBar from './SearchBar';
import './BookList.css';
import Header from './Header';

const BookList = () => {
  const [books, setBooks] = useState([]);
  const [filteredBooks, setFilteredBooks] = useState([]);
  const [genres, setGenres] = useState([]);
  const [filterGenre, setFilterGenre] = useState("");
  const [searchTitle, setSearchTitle] = useState("");
  const [isNewest, setIsNewest] = useState(false);

  const { addToCart, clearCart } = useContext(CartContext);
  const { logout } = useContext(AuthContext);

  const navigate = useNavigate();
  const location = useLocation();

  const fetchBooks = async () => {
    try {
      const params = new URLSearchParams(location.search);
      let url = 'http://localhost:5000/api/books';

      if (params.toString()) {
        url += '?' + params.toString();
      }

      const response = await axios.get(url);
      setBooks(response.data);
      setFilteredBooks(response.data);

      if (!params.get('genre') && !params.get('sort') && !params.get('sale')) {
        const uniqueGenres = [...new Set(response.data.map(book => book.genre))];
        setGenres(uniqueGenres);
      }
    } catch (err) {
      console.error('Error fetching books:', err);
    }
  };

  useEffect(() => {
    fetchBooks();
  }, [location.search]);

  useEffect(() => {

```



```

const params = new URLSearchParams(location.search);
const genre = params.get('genre');
const sort = params.get('sort');
const sale = params.get('sale');

if (books.length === 0) return;

if (genre) {
  setFilterGenre(genre);
  applyFilters(searchTitle, genre);
  setIsNewest(false);
} else if (sort === 'newest') {
  setIsNewest(true);
  const sortedBooks = [...books].sort(
    (a, b) => new Date(b.createdAt) - new Date(a.createdAt)
  );
  setFilteredBooks(sortedBooks.slice(0, 8));
} else if (sale === 'true') {
  setIsNewest(false);
  setFilteredBooks(books);
} else {
  setFilteredBooks(books);
  setFilterGenre("");
  setIsNewest(false);
}
}, [location.search, books]);

const applyFilters = (title, genre) => {
  let filtered = books;

  if (title) {
    filtered = filtered.filter(book =>
      book.title.toLowerCase().includes(title.toLowerCase())
    );
  }

  if (genre && genre !== 'All') {
    filtered = filtered.filter(book =>
      book.genre.toLowerCase() === genre.toLowerCase()
    );
  }

  setFilteredBooks(filtered);
};

const handleSearch = (title) => {
  setSearchTitle(title);
  setIsNewest(false);
  applyFilters(title, filterGenre);
};

```

```

const handleFilter = (genre) => {
  setFilterGenre(genre);
  setIsNewest(false);
  applyFilters(searchTitle, genre);
};

const handleAddToCart = async (bookId) => {
  await addToCart(bookId);

  const updatedBooks = books.map(book =>
    book._id === bookId && book.quantity > 0
    ? { ...book, quantity: book.quantity - 1 }
    : book
  );

  setBooks(updatedBooks);
  applyFilters(searchTitle, filterGenre);

  alert('Book added to cart!');
};

const handleLogout = () => {
  clearCart();
  logout();
  navigate('/home1');
};

const params = new URLSearchParams(location.search);
const hasGenreParam = params.get('genre');
const hasSaleParam = params.get('sale') === 'true';

return (
  <div>
    <Header onLogout={handleLogout} />
    <div className="book-list">
      <h1>Book List</h1>

      <SearchBar
        searchTerm={searchTitle}
        setSearchTerm={setSearchTitle}
        onSearch={handleSearch}
      />

      {!isNewest && !hasGenreParam && !hasSaleParam && (
        <div className="genre-dropdown">
          <label htmlFor="genre">Select Genre: </label>
          <select
            id="genre"
            value={filterGenre}

```

```

onChange={(e) => handleFilter(e.target.value)}
>
<option value="All">All</option>
{genres.map((genre) => (
<option key={genre} value={genre}>{genre}</option>
))}
</select>
</div>
)}

{(filterGenre && filterGenre !== 'All') || searchTitle ? (
<p className="active-filter">
{filterGenre && filterGenre !== 'All' && (
<span><strong>Genre:</strong> {filterGenre} </span>
)}
{searchTitle && (
<span><strong>Search:</strong> {searchTitle}</span>
)}
</p>
): null}

{isNewest && (
<p className="active-filter">
<strong>Showing:</strong> Newest Arrivals
</p>
)}

<div className="User_books">
{filteredBooks.map((book) => (
<div key={book._id} className="book">
<table className="book-table">
<tbody>
<tr className="book-image">
<td className="image-data">
<img
src={book.coverImage || 'https://via.placeholder.com/150'}
alt={book.title}
className="user-book-image"
/>
</td>
</tr>
<tr className="book-details">
<td className="padding_none">
<h2 className="user-book-title">{book.title}</h2>
<p className="user-book-author">{book.author}</p>
<h2 className="user-book-genre">{book.genre}</h2>
</td>
</tr>
<tr>
<td className="book-desc">

```

```

<h2 className="user-book-desc">{book.description}</h2>
</td>
</tr>
<tr className="book-buttons">
<td className="padding_none">
<p className="user-book-author">
{book.salePrice && book.salePrice < book.price ? (
<
<span style={{ textDecoration: 'line-through', color: 'red' }}>
₹{book.price}
</span>{' '}
<span style={{ fontWeight: 'bold', color: 'green' }}>
₹{book.salePrice}
</span>
</>
) : (
<>₹{book.price}</>
)}
</p>
<p className="user-book-author">Quantity: {book.quantity}</p>
{book.quantity > 0 ? (
<button onClick={() => handleAddToCart(book._id)}>
Add to Cart
</button>
) : (
<p style={{ color: 'red' }}>Sold Out</p>
)}
</td>
</tr>
</tbody>
</table>
</div>
))}
</div>
</div>
</div>
);
};

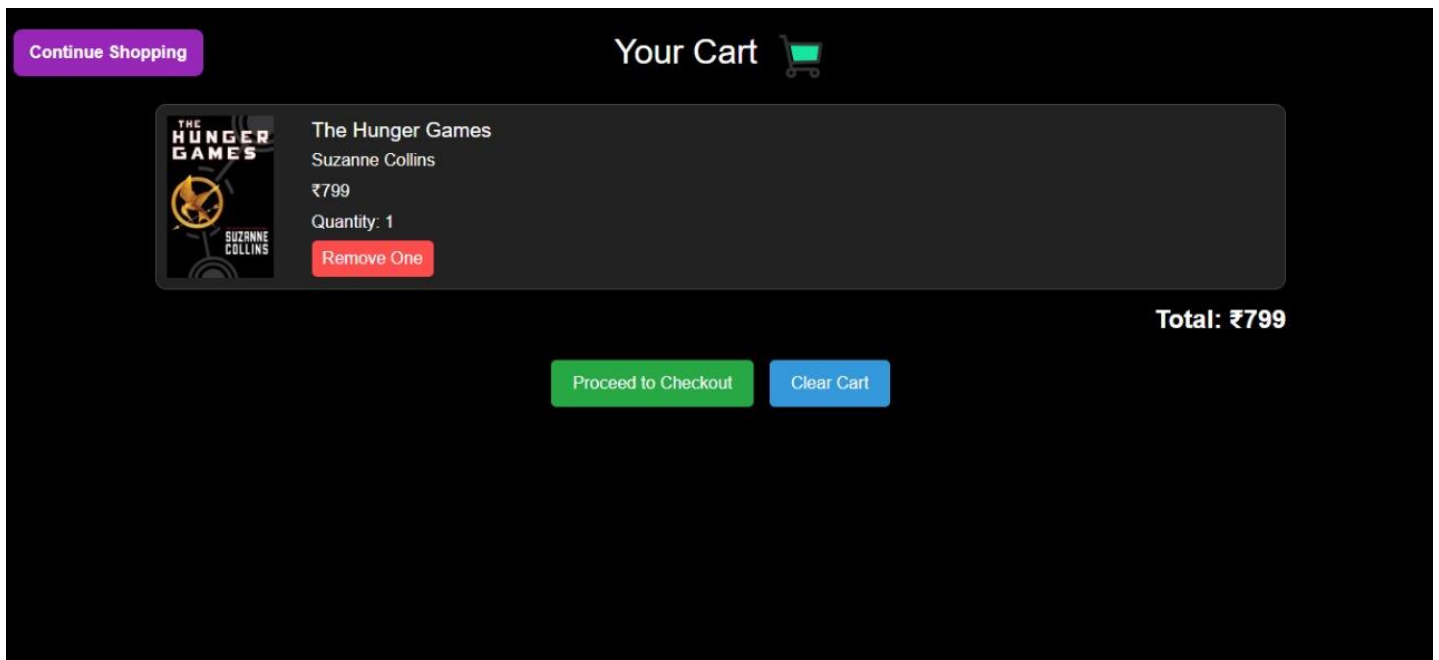
```

```

export default BookList;

```

## 9) Cart:



## ✓ CODE

```
import React, { useContext, useEffect } from 'react';
import { CartContext } from '../context/CartContext';
import './Cart.css';
import { useNavigate } from 'react-router-dom';

const Cart = () => {
  const { cart, removeFromCart, clearCart, fetchCart } = useContext(CartContext);
  const navigate = useNavigate();

  useEffect(() => {
    fetchCart();
  }, [fetchCart]);

  return (
    <div className="cart-container">
      <div className="cart-header">
        <h1>Your Cart</h1>
```

```


</div>

```

```

<button
className="continue-shopping-btn"
onClick={() => navigate('/shop')}
>
Continue Shopping
</button>

```

```

{cart.length === 0 ? (
<p className="empty-cart">Your cart is empty.</p>
) : (
<div className="cart-items">
{cart.map((item) => (
<div key={item.bookId._id} className="cart-item">
<img
src={item.bookId.coverImage || 'https://via.placeholder.com/150'}
alt={item.bookId.title}
/>
<div className="book-details">
<h2>{item.bookId.title}</h2>
<p>{item.bookId.author}</p>
<p>
{item.bookId.salePrice && item.bookId.salePrice < item.bookId.price ? (
<span style={{ textDecoration: 'line-through', color: 'red' }}>
₹{item.bookId.price}
</span>{' '}
<span style={{ fontWeight: 'bold', color: 'green' }}>
₹{item.bookId.salePrice}
</span>
</>
) : (
<span>₹{item.bookId.price}</span>
)}
</p>
<p>Quantity: {item.quantity}</p>
<button onClick={() => removeFromCart(item.bookId._id)}>Remove One</button>
</div>
</div>
))}

```

```

<p className="total-price">
Total: ₹

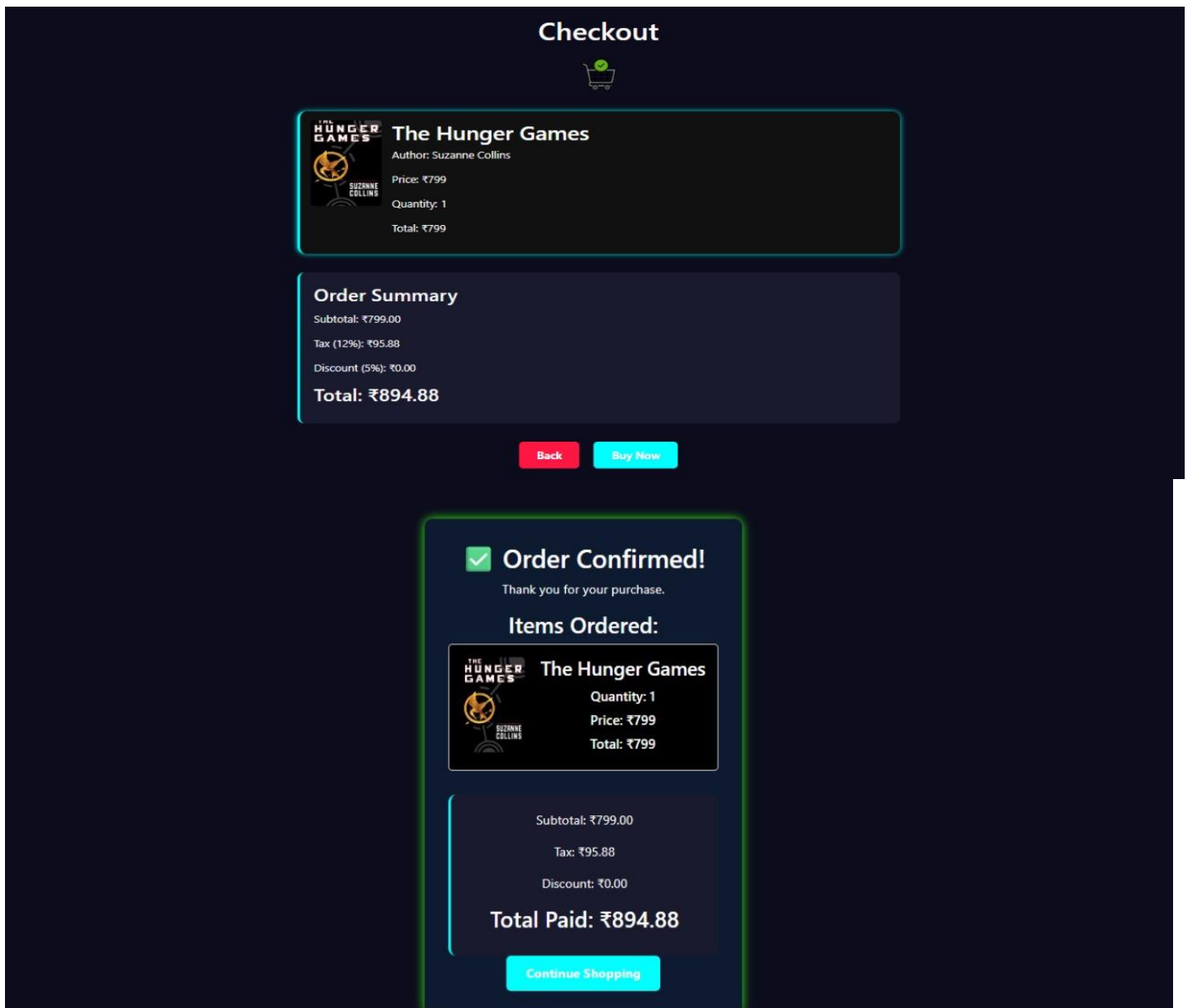
```

```
{cart.reduce((total, item) => {  
  const price =  
  item.bookId.salePrice && item.bookId.salePrice < item.bookId.price  
  ? item.bookId.salePrice  
  : item.bookId.price;  
  return total + price * item.quantity;  
}, 0)}  
</p>
```

```
<div className="cart-actions">  
  <button  
    className="checkout-button"  
    onClick={() => navigate('/checkout')}  
    disabled={cart.length === 0}  
  >  
    Proceed to Checkout  
  </button>  
  <button  
    className="clear-cart-button"  
    onClick={clearCart}  
    disabled={cart.length === 0}  
  >  
    Clear Cart  
  </button>  
</div>  
</div>  
)}  
</div>  
);  
};
```

```
export default Cart;
```

## 10) Checkout:



## ✓ CODE

```
import React, { useContext, useState } from 'react';
import { CartContext } from '../context/CartContext';
import { useNavigate } from 'react-router-dom';
import axios from 'axios';
import './Checkout.css';
```

```
const Checkout = () => {
  const { cart, clearCart } = useContext(CartContext);
```



```
const [orderConfirmed, setOrderConfirmed] = useState(false);
const [orderDetails, setOrderDetails] = useState(null);
const navigate = useNavigate();

const subtotal = cart.reduce((total, item) => {
const price = item.bookId.salePrice && item.bookId.salePrice < item.bookId.price
? item.bookId.salePrice
: item.bookId.price;
return total + price * item.quantity;
}, 0);

const tax = subtotal * 0.12;
const discount = subtotal > 1000 ? subtotal * 0.05 : 0;
const total = subtotal + tax - discount;

const handleBuyNow = async () => {
try {
const user = JSON.parse(localStorage.getItem('user'));
if (!user?.token) throw new Error('User not logged in');

await axios.post(
'http://localhost:5000/api/cart/checkout',
{},
{ headers: { Authorization: Bearer ${user.token} } }
);

// Deep clone the cart before clearing it
const frozenCart = cart.map(item => ({
...item,
bookId: { ...item.bookId },
quantity: item.quantity
})));

setOrderDetails({ cart: frozenCart, subtotal, tax, discount, total });
setOrderConfirmed(true);
clearCart();
} catch (err) {
console.error('Checkout error:', err.response?.data || err.message);
}
};

return (
<div className="checkout">
{!orderConfirmed ? (
<>
<h1 className="checkout-title">Checkout</h1>

<div className="cart-items-container">
  {cart.map((item) => {
    const price = item.bookId.salePrice && item.bookId.salePrice < item.bookId.price
    ? item.bookId.salePrice
    : item.bookId.price;
    return (
      <div className="cart-card" key={item.bookId._id}>
        <img src={item.bookId.coverImage} alt={item.bookId.title} />
        <div className="cart-details">
          <h2>{item.bookId.title}</h2>
          <p>Author: {item.bookId.author}</p>
          <p>
            Price:{' '}
            {item.bookId.salePrice && item.bookId.salePrice < item.bookId.price ? (
              <span className="striketrough">₹{item.bookId.price}</span>{' '}
              <span className="highlight">₹{item.bookId.salePrice}</span>
            ) : (
              <span>₹{item.bookId.price}</span>
            )}
          </p>
          <p>Quantity: {item.quantity}</p>
          <p>Total: ₹{item.quantity * price}</p>
        </div>
      </div>
    );
  })}
</div>

<div className="summary-box">
  <h3>Order Summary</h3>
  <p>Subtotal: ₹{subtotal.toFixed(2)}</p>
  <p>Tax (12%): ₹{tax.toFixed(2)}</p>
  <p>Discount (5%): ₹{discount.toFixed(2)}</p>
  <h3>Total: ₹{total.toFixed(2)}</h3>
</div>

<div className="checkout-actions">
  <button onClick={() => navigate('/cart')} className="btn-secondary">Back</button>
  <button onClick={handleBuyNow} className="btn-primary">Buy Now</button>
</div>

) : (
  <div className="order-confirmation">
    <h2>👍 Order Confirmed!</h2>
    <p>Thank you for your purchase.</p>
    <div className="order-summary">
      <h3>Items Ordered:</h3>

```

```

{orderDetails.cart.map((item) => {
const price = item.bookId.salePrice && item.bookId.salePrice < item.bookId.price
? item.bookId.salePrice
: item.bookId.price;
return (
<div className="order-item" key={item.bookId._id}>
<img src={item.bookId.coverImage} alt={item.bookId.title} />
<div>
<h4>{item.bookId.title}</h4>
<p>Quantity: {item.quantity}</p>
<p>Price: ₹{price}</p>
<p>Total: ₹{item.quantity * price}</p>
</div>
</div>
);
)}}
<div className="summary-box">
<p>Subtotal: ₹{orderDetails.subtotal.toFixed(2)}</p>
<p>Tax: ₹{orderDetails.tax.toFixed(2)}</p>
<p>Discount: ₹{orderDetails.discount.toFixed(2)}</p>
<h3>Total Paid: ₹{orderDetails.total.toFixed(2)}</h3>
</div>
</div>
<button onClick={() => navigate('/shop')} className="btn-primary">
Continue Shopping
</button>
</div>
)}
</div>
);
};

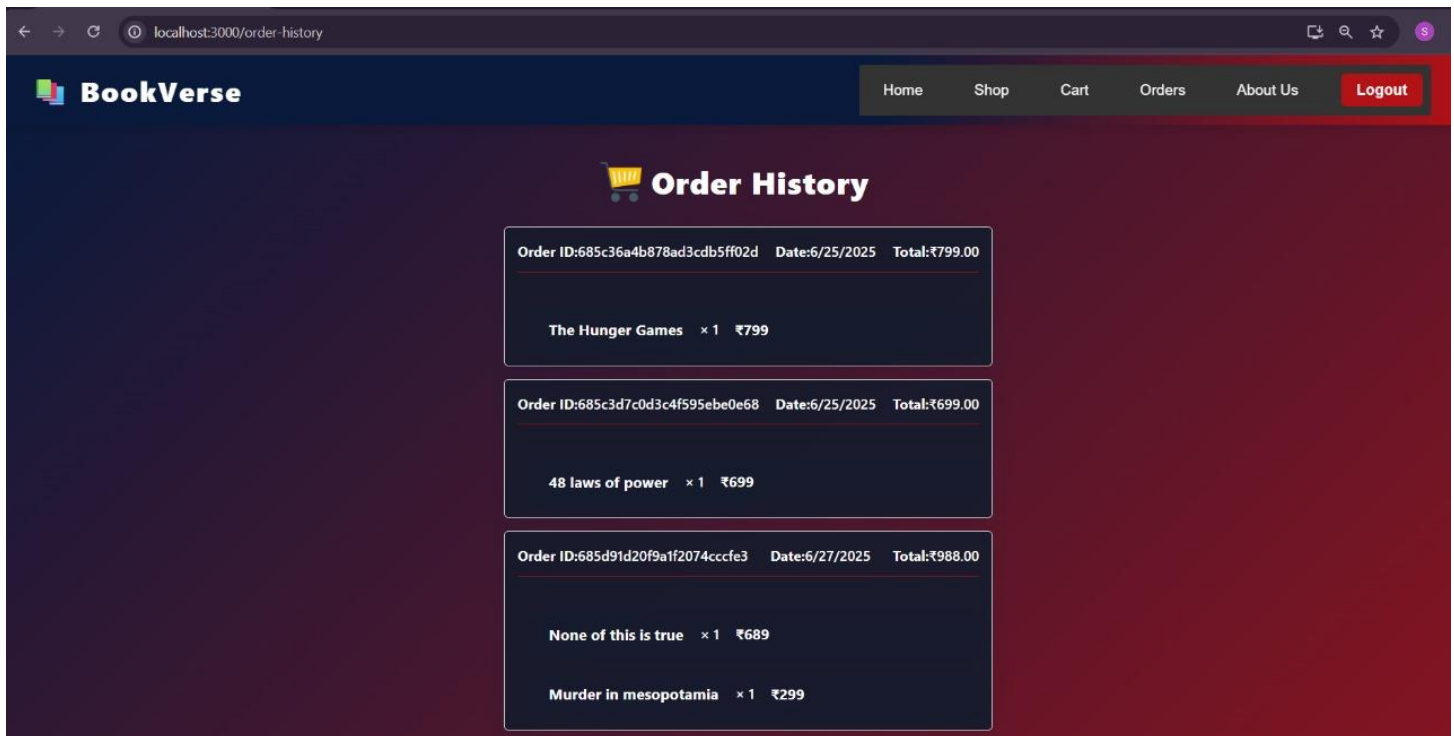
```

```

export default Checkout;

```

## 11)Order History:



### ✓ CODE

```
import React, { useEffect, useState, useContext } from 'react';
import axios from 'axios';
import Header from './Header';
import { AuthContext } from '../context/AuthContext';
import './OrderHistory.css'; // Optional: Add styles for this component
```

```
const OrderHistory = () => {
  const [orders, setOrders] = useState([]);
  const { user } = useContext(AuthContext);
```

```
  useEffect(() => {
    const fetchOrderHistory = async () => {
      try {
        console.log('Fetching order history for user:', user);
        const response = await axios.get('http://localhost:5000/api/orders', {
          headers: {
            Authorization: Bearer ${user.token},
          },
        });
```

```

});
console.log('Order history response:', response.data);
setOrders(response.data);
} catch (err) {
console.error('Error fetching order history:', err);
}
}

if (user) {
fetchOrderHistory();
}
}, [user]);

return (
<>
<Header />
<div className="order-history">
<h1>

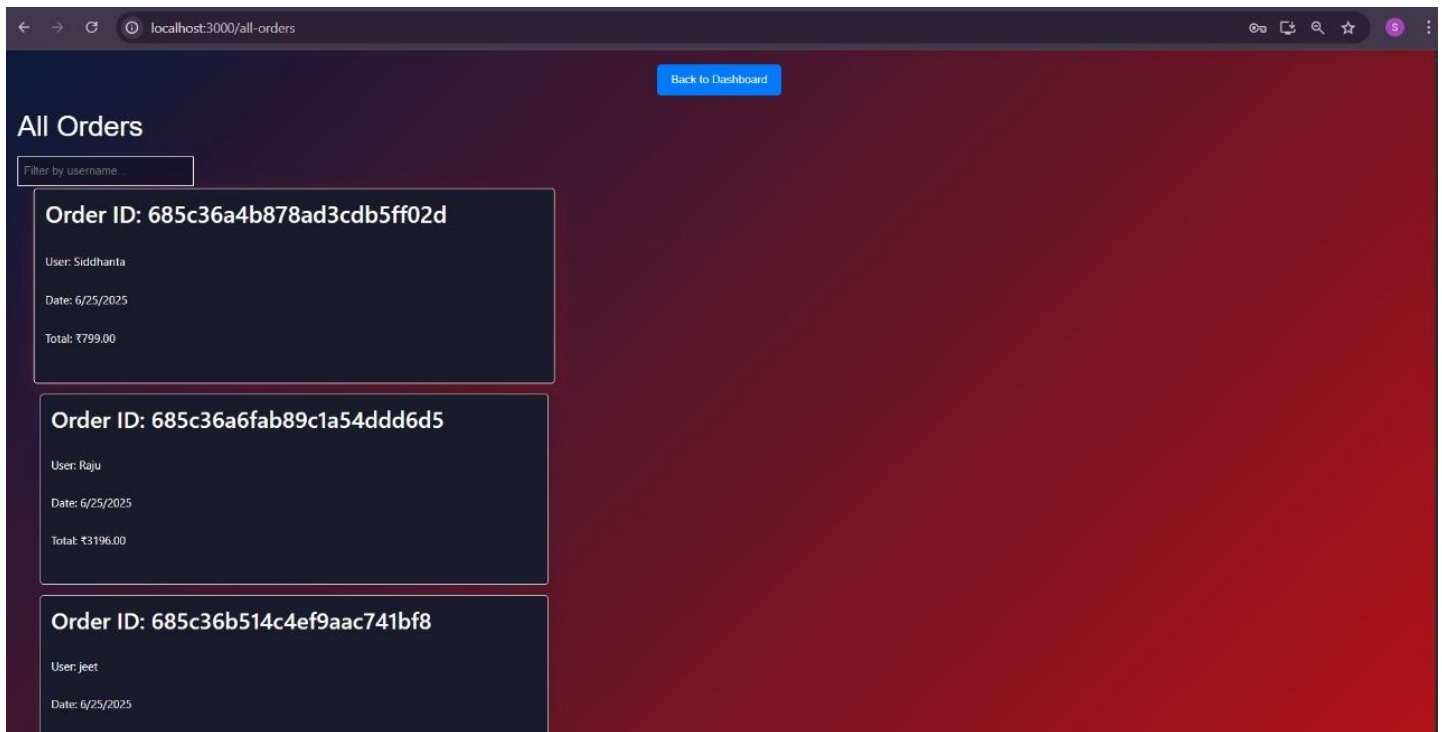
Order History
</h1>
{orders.length === 0 ? (
<p>No orders found.</p>
) : (
<ul>
{orders.map((order) => (
<li key={order._id} className="order-item">
<div className="order-summary-row">
<div>
<span className="order-label">Order ID:</span>
<span className="order-value">{order._id}</span>
</div>
<div>
<span className="order-label">Date:</span>
<span className="order-value">
{new Date(order.createdAt).toLocaleDateString()}
</span>
</div>
<div>
<span className="order-label">Total:</span>
<span className="order-value">₹ {order.total.toFixed(2)}</span>
</div>
</div>
<ul className="order-books-list">
{order.items
.filter((item) => item.bookId)

```

```
.map((item) => (  
  <li key={ item.bookId._id} className="order-book-item">  
    <span className="book-title">{ item.bookId.title}</span>  
    <span className="book-qty">× { item.quantity}</span>  
    <span className="book-price">₹ {item.bookId.price}</span>  
  </li>  
  )}  
</ul>  
</li>  
  )}  
</ul>  
  )}  
</div>  
</>  
);  
};
```

```
export default OrderHistory;
```

## 12) All Orders(admin):



### ✓ CODE

```
import React, { useEffect, useState, useContext } from 'react';
import axios from 'axios';
import { AuthContext } from '../context/AuthContext';
import { useNavigate } from 'react-router-dom';
import './AllOrders.css';

const AllOrders = () => {
  const [orders, setOrders] = useState([]);
  const [filterUser, setFilterUser] = useState(""); // ✓ new state for user name filter
  const { user } = useContext(AuthContext);
  const navigate = useNavigate();

  useEffect(() => {
    const fetchAllOrders = async () => {
      try {
        const response = await axios.get('http://localhost:5000/api/all-orders', {
          headers: {
            Authorization: Bearer ${user.token},
          },
        });
        setOrders(response.data);
      } catch (error) {
        console.log(error);
      }
    };
    fetchAllOrders();
  });
}
```

```

    } catch (err) {
      console.error('Error fetching all orders:', err);
    }
  };

  if (user) {
    fetchAllOrders();
  }
}, [user]);

// ✅ Filter orders based on filterUser
const filteredOrders = orders.filter((order) =>
  order.userId?.username?.toLowerCase().includes(filterUser.toLowerCase())
);

return (
  <div className="all-orders">
    <button className="back-button" onClick={() => navigate('/admin')}>
      Back to Dashboard
    </button>
    <h1>All Orders</h1>

    { /* ✅ Search input */ }
    <input
      type="text"
      placeholder="Filter by username..."
      value={filterUser}
      onChange={(e) => setFilterUser(e.target.value)}
      className="filter-input"
    />

    {filteredOrders.length === 0 ? (
      <p>No orders found.</p>
    ) : (
      <ul>
        {filteredOrders.map((order) => (
          <li key={order._id} className="order-item">
            <h2>Order ID: {order._id}</h2>
            <p>User: {order.userId?.username || 'Unknown User'}</p>
            <p>Date: {new Date(order.createdAt).toLocaleDateString()}</p>
            <p>Total: ₹{order.total.toFixed(2)}</p>
            <ul>
              {order.items
                .filter((item) => item.bookId)
                .map((item) => (
                  <li key={item.bookId._id}>
                    {item.bookId.title} - {item.quantity} x ₹{item.bookId.price}
                  </li>
                ))}
            </ul>
          </li>
        ))}
      </ul>
    )}
  </div>
);

```



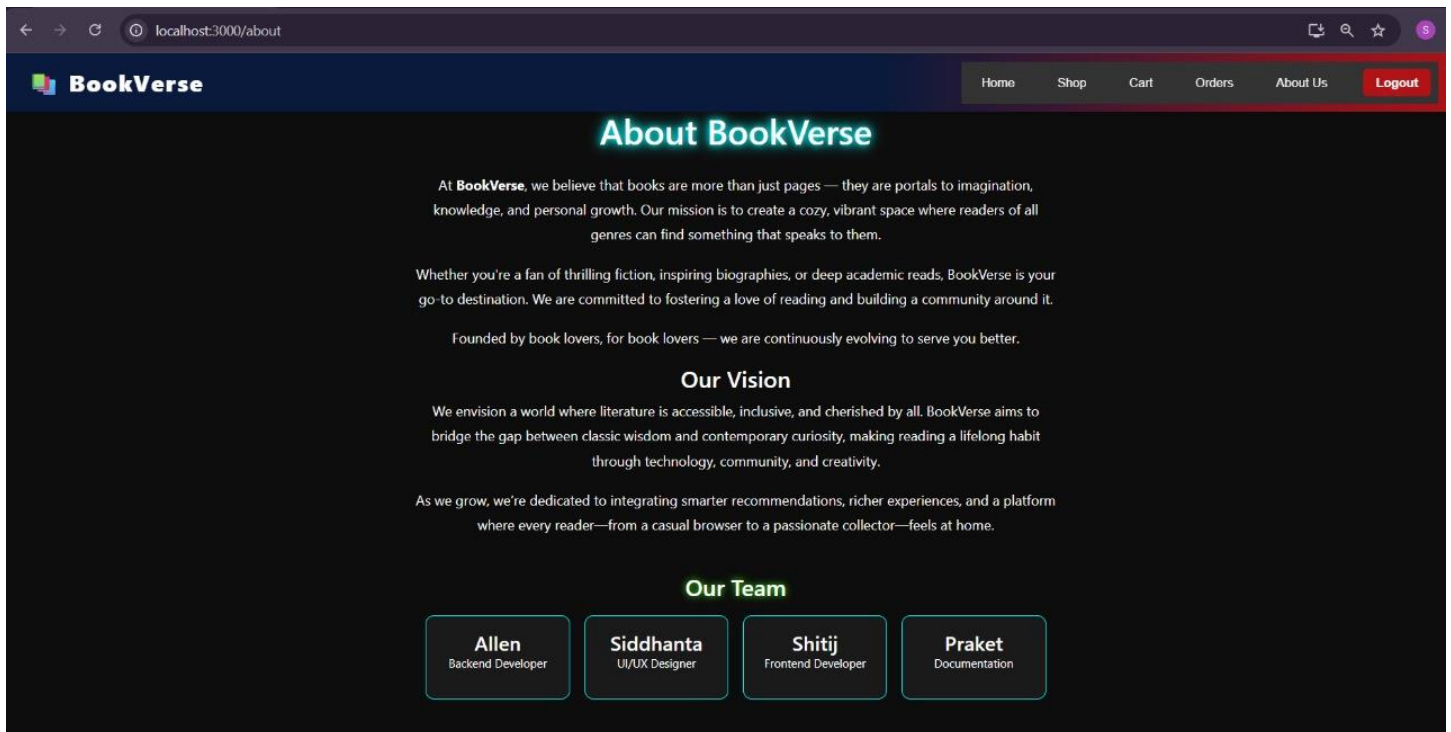
```

</li>
))}
</ul>
)}
</div>
);
};

```

```
export default AllOrders;
```

### 13) About Us:



### ✓ CODE

```

import React from 'react';
import './About.css';
import Header from './Header';

const About = () => {
  return (
    <>
    <Header />
    <div className="about-page">
    <div className="about-content">
    <h1 className="neon-title">About BookVerse</h1>

```

<p className="about-text">

At <strong>BookVerse</strong>, we believe that books are more than just pages — they are portals to imagination, knowledge, and personal growth. Our mission is to create a cozy, vibrant space where readers of all genres can find something that speaks to them.

</p>

<p className="about-text">

Whether you're a fan of thrilling fiction, inspiring biographies, or deep academic reads, BookVerse is your go-to destination. We are committed to fostering a love of reading and building a community around it.

</p>

<p className="about-text">

Founded by book lovers, for book lovers — we are continuously evolving to serve you better.

</p>

{/\*  Vision Section \*/}

<div className="about-vision">

<h2 className="vision-heading">Our Vision</h2>

<p className="about-text">

We envision a world where literature is accessible, inclusive, and cherished by all. BookVerse aims

to bridge the gap between classic wisdom and contemporary curiosity, making reading a lifelong habit

through technology, community, and creativity.

</p>

<p className="about-text">

As we grow, we're dedicated to integrating smarter recommendations, richer experiences, and a platform

where every reader—from a casual browser to a passionate collector—feels at home.

</p>

</div>

{/\*  Team Section \*/}

<div className="about-team">

<h2 className="team-heading">Our Team</h2>

<div className="team-grid">

<div className="team-card">

<h3>Allen</h3>

<p>Backend Developer</p>

</div>

<div className="team-card">

<h3>Siddhanta</h3>

<p>UI/UX Designer</p>

</div>

<div className="team-card">



## ❖ BACKEND:-

- **Server.js:**

```
const express = require('express');
const mongoose = require('mongoose');
const cors = require('cors');
const dotenv = require('dotenv');
const bookRoutes = require('./routes/bookRoutes');
const cartRoutes = require('./routes/cartRoutes');

dotenv.config();

const app = express();
const PORT = process.env.PORT || 5000;
const MONGO_URI = process.env.MONGO_URI;
const authRoutes = require('./routes/authRoutes');
const orderRoutes = require('./routes/orderRoutes');

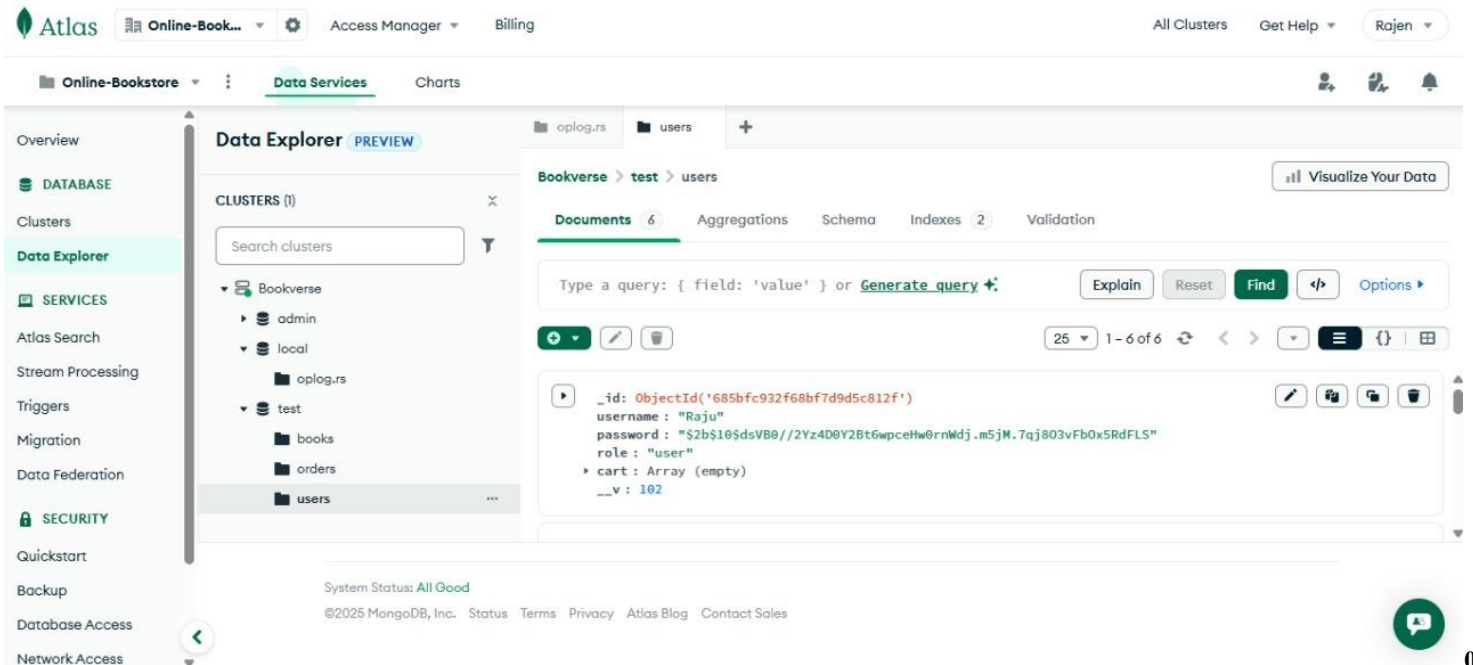
// Middleware
app.use(cors());
app.use(express.json());

// Connect to MongoDB
mongoose.connect(MONGO_URI)
  .then(() => console.log('MongoDB connected'))
  .catch(err => console.log(err));

// Routes
app.use('/api/books', bookRoutes);
app.use('/api/cart', cartRoutes);
app.use('/api/auth', authRoutes);
app.use('/api', orderRoutes);

// Start server
app.listen(PORT, () => console.log(Server running on port ${PORT}));
```

## 1) USER AND ADMIN DATA:



0

### • User.js:

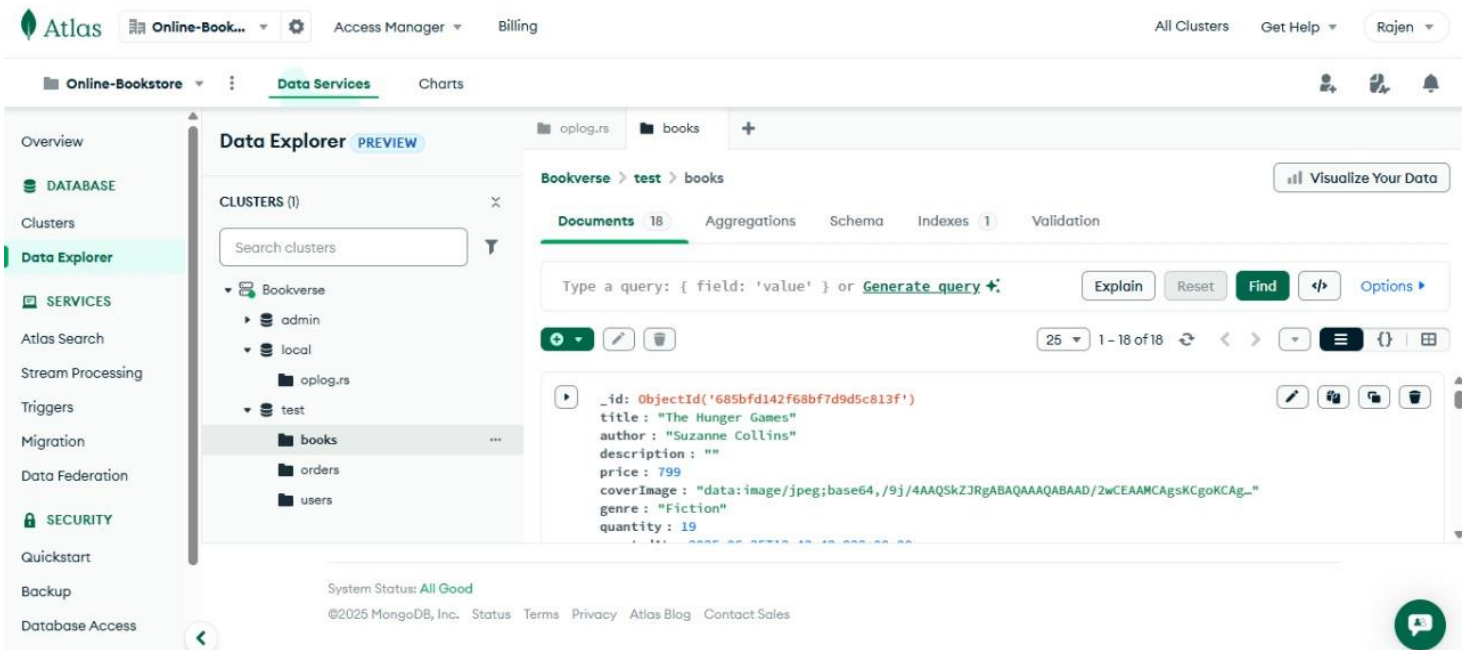
```
const mongoose = require('mongoose');
const bcrypt = require('bcryptjs');
```

```
const UserSchema = new mongoose.Schema({
  username: { type: String, required: true, unique: true },
  password: { type: String, required: true },
  role: { type: String, enum: ['user', 'admin'], default: 'user' },
  cart: [
    {
      bookId: { type: mongoose.Schema.Types.ObjectId, ref: 'Book' },
      quantity: { type: Number, default: 1 },
    },
  ],
});
```

```
UserSchema.pre('save', async function (next) {
  if (this.isModified('password')) {
    this.password = await bcrypt.hash(this.password, 10);
  }
  next();
});
```

```
module.exports = mongoose.model('User', UserSchema);
```

## 2) BOOK DATA:



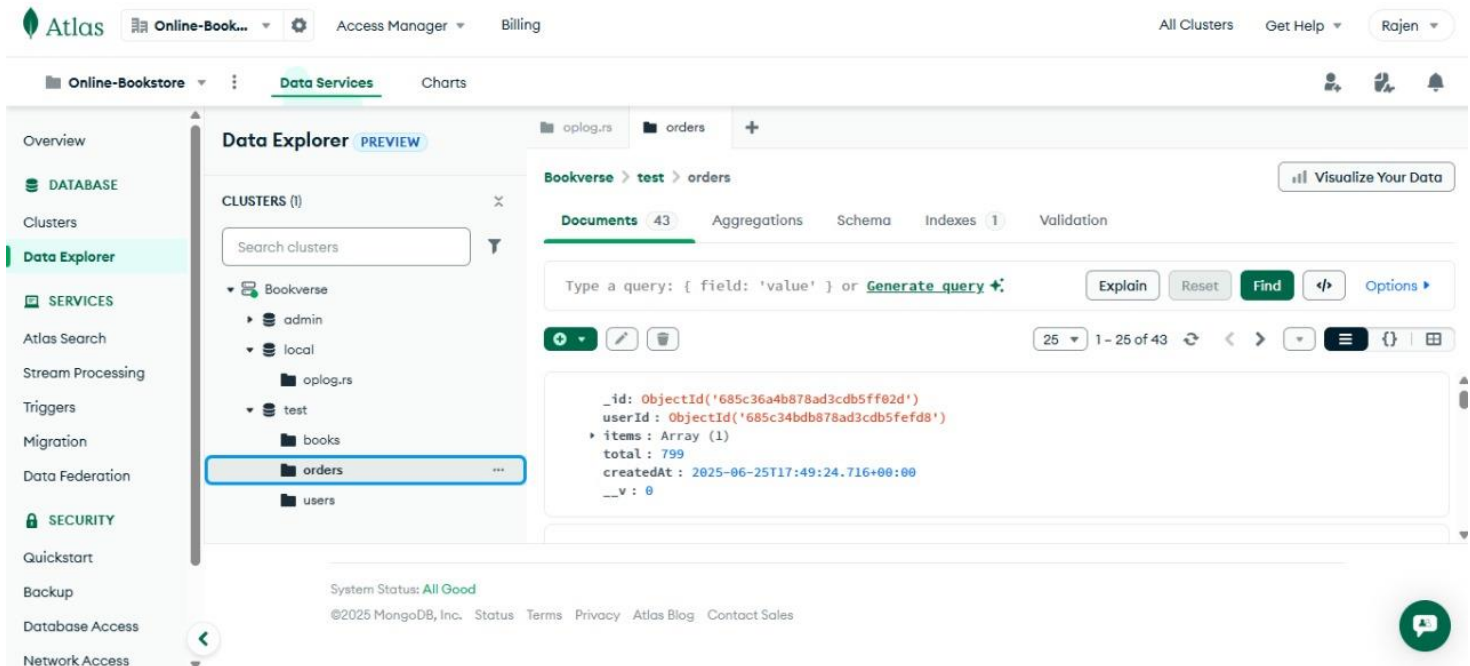
### • Book.js :-

```
const mongoose = require('mongoose');
```

```
const BookSchema = new mongoose.Schema({
  title: { type: String, required: true },
  author: { type: String, required: true },
  description: { type: String },
  price: { type: Number, required: true }, // original price
  salePrice: { type: Number }, // discounted price
  discount: { type: Number }, // discount percentage (optional)
  coverImage: { type: String },
  genre: { type: String, required: true },
  quantity: { type: Number, default: 0, required: true },
}, { timestamps: true });
```

```
module.exports = mongoose.model('Book', BookSchema);
```

### 3) ORDER DATA:



- Order.js :-

```
// backend/models/Order.js
const mongoose = require('mongoose');

const OrderSchema = new mongoose.Schema({
  userId: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true }, // Reference to the
  user who placed the order
  items: [
    {
      bookId: { type: mongoose.Schema.Types.ObjectId, ref: 'Book', required: true }, // Reference to
      the book
      quantity: { type: Number, required: true }, // Quantity of the book
      price: { type: Number, required: true }, // Price of the book at the time of purchase
    },
  ],
  total: { type: Number, required: true }, // Total price of the order
  createdAt: { type: Date, default: Date.now }, // Order creation date
});

module.exports = mongoose.model('Order', OrderSchema);
```

## **6. CONCLUSION**

BookVerse is not just another book-selling website—it is a refined, thoughtfully developed digital platform built for users who crave simplicity and reliability in the online shopping space. The project bridges the gap between outdated, physical book-buying methods and overwhelming, feature-heavy digital solutions by offering a clean, streamlined experience that delivers exactly what the user needs—and nothing more.

By emphasizing clarity, responsiveness, and an intuitive design, BookVerse makes it easier for users to browse, select, and purchase books across genres and categories. The inclusion of featured sections like Best Selling, Sale, and New Arrivals enhances discoverability and encourages users to explore more. The genre-based filtering system ensures that users can always find books tailored to their personal preferences without feeling lost or overloaded.

From a business and operational standpoint, the integrated admin dashboard simplifies backend tasks such as updating inventory, managing book details, and reviewing order histories. This not only reduces the workload for the administrator but also ensures that the platform remains up to date and relevant with minimal effort.

What truly sets BookVerse apart is its commitment to remaining lightweight while being effective. It doesn't try to be everything—it focuses on one thing and does it well: providing a smooth, frustration-free online bookstore experience. No logins, no ads, no pop-ups—just a digital space where books are king.

In conclusion, BookVerse stands as a prime example of how thoughtful design, clear objectives, and essential functionalities can come together to build a successful digital product. It meets the real-world needs of book buyers and sellers alike, proving that sometimes, less really is more.



## 7) .FUTURE SCOPE & FURTHER ENHANCEMENTS – BookVerse

### ❖ **Future Scope:**

The future of BookVerse holds significant potential for innovation, enhanced usability, and global reach. Key areas for development include:

- **AI-Powered Recommendations**

Integrate machine learning algorithms to provide intelligent book suggestions based on user preferences, past purchases, ratings, and browsing behavior.

- **Gamified Reading Experience**

Introduce gamification elements like reading challenges, reward points, leaderboards, and achievement badges to make reading more engaging and social.

- **Augmented Reality Previews**

Utilize AR technology to let users preview books on shelves or simulate real-world reading environments, enhancing user interaction and excitement before buying.

- **Offline Reading Mode**

Enable downloading of eBooks for offline access, ensuring uninterrupted reading for users in low-connectivity areas.

- **Voice & Audio Integration**

Expand your library to include audiobooks and integrate with voice assistants (e.g., Alexa, Google Assistant) for hands-free browsing and reading support.

- **Global Accessibility**

Incorporate multilingual support to serve diverse readers across the globe, enhancing inclusivity and user reach.

- **Social Reading & Reviews**

Allow users to form book clubs, share public reading lists, and collaboratively review or annotate books, encouraging a stronger community bond.

## ❖ Further Enhancements:

### 1. User Experience Improvements

Smart Dashboards: Personalized UI showing current reads, wishlist, reading history, and genre-based recommendations.

Dark Mode & High Contrast Themes: For comfort during long reading sessions and improved accessibility.

Neon Accents & Animated UI: Enrich the visual design with fluid transitions and vibrant contrast for a more premium feel.

### 2. Content & Book Tools

Smart Book Search & Filters: Advanced search with filters for genre, rating, language, discount, author, etc.

AI Book Summarizer: Summarize long books using AI to help readers quickly decide.

In-app Reader & Notes: Built-in eBook reader with note-taking and highlight features.

### 3. Community & Social Features

Live Author Sessions & Webinars: Organize virtual meet-and-greets or Q&A sessions with authors.

Reader Forums & Clubs: Build niche communities where users can discuss books and genres they love.

User-Generated Content: Let users submit reviews, blog posts, or even short stories.

### 4. Data & Personalization

Reading Progress Tracker: Track and visualize reading stats (e.g., time spent, chapters read).

Smart Notifications: Reminders based on reading habits or wishlist price drops.

AI-driven Personalized Homepage: Real-time updates based on behavior and preferences.

### 5. Technical Upgrades

PWA & Offline Support: Turn BookVerse into a Progressive Web App for native-like performance and offline access.

Cloud Sync: Sync reading position, notes, and wishlist across devices.

Integration with School Libraries or Publishers: Collaborate with academic institutions to expand digital reach.

## **8) BIBLIOGRAPHY**

- [www.youtube.com](http://www.youtube.com)
- [www.codepen.io](http://www.codepen.io)
- [www.google.com](http://www.google.com)
- [www.googlefont.com](http://www.googlefont.com)
- [www.react.our](http://www.react.our)
- [www.codingworld.com](http://www.codingworld.com)
- [www.w3schools.com](http://www.w3schools.com)