BSc. (Hons) in Software Engineering
Department of Information Technology
Faculty of Computing

Name of the Student: S.A.P.Dewavishmi
Student Number: IT_IFLS_001/B003/0006
Module: IT3113
Advanced Web Technologies

Submission Date: 07.05.2025

Contents

1. Project Overview

The online book store web application allow users to browse, search and purchase books. The application is integrated used Google Books API to fetch book data and implements a complete e-shopping experience with all required e-commerce functionalities.

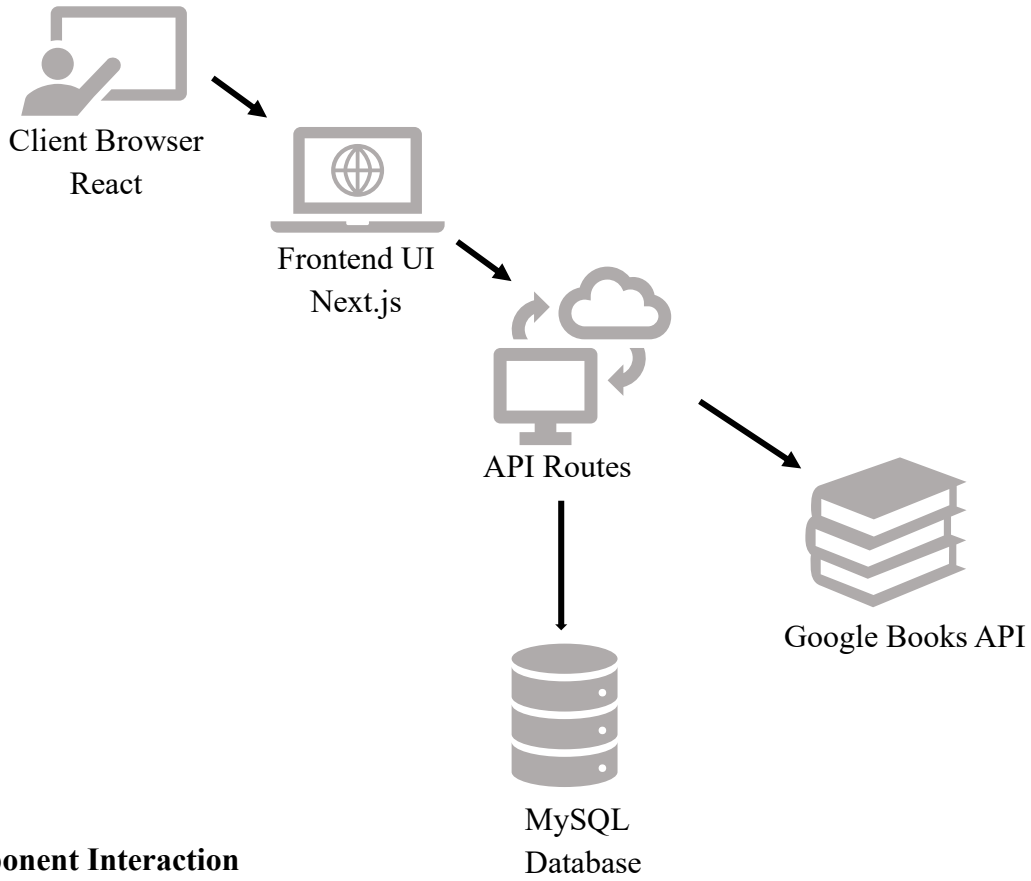
Key Features

- User authentication (registration, login, profile management)
- Book browsing with search and filter capabilities
- Shopping cart functionality with persistent storage
- Checkout process
- Integration with Google Books API
- Responsive design for all device sizes
- Secure user data handling

Technologies Used

- Frontend: React.js, Next.js, TypeScript, Tailwind CSS
- Backend: Next.js API Routes, Node.js
- Database: MySQL with Prisma ORM
- Authentication: JWT (JSON Web Tokens), bcrypt for password hashing
- External API: Google Books API
- State Management: React Context API

2. System Architecture



Component Interaction

1. Client Layer

- User's web browser renders React components
- Manages local state (UI state, form data)
- Communicates with API layer via fetch requests

2. Frontend Layer

- Next.js App Router for page routing
- React components for UI rendering
- Context providers for global state management (Auth, Cart)

3. API Layer

- Next.js API Routes handle HTTP requests
- Implements RESTful API principles

- Manages authentication and authorization
- Communicates with database and external APIs

4. Data Layer

- MySQL database stores user data, orders, etc.
- Prisma ORM provides type-safe database access
- Data models and relationships
-

5. External Services

- Google Books API provides book data
- JWT for authentication tokens

Data Flow:

1. Authentication Flow

- User submits credentials → API validates → JWT token created → Cookie set → User authenticated
- Protected routes check token validity before processing requests

2. Book Search Flow

- User submits search → API receives request → Google Books API queried → Results formatted → Data returned to frontend

3. Shopping Cart Flow

- Client-side cart management using Context API and localStorage
- Cart persists across page refreshes

4. Checkout Flow

- Authenticated user submits order → Order validated → Database transaction → Order confirmation

3. Technical Implementation

3.1 Frontend Development

Layout Components

- app/layout.tsx Main application layout with header and footer
- components/header.tsx Navigation and user menu

Authentication Components

- app/login/page.tsx User login form
- app/signup/page.tsx User registration form
- context/auth-context.tsx Authentication state management

Book Components

- components/book-search.tsx Search functionality
- components/book-grid.tsx Display books in a grid layout
- app/books/page.tsx Book listing page
- app/books/[id]/page.tsx Book details page

Shopping Components

- context/cart-context.tsx Shopping cart state management
- app/cart/page.tsx Shopping cart page

Form Components

- Form validation using React Hook Form and Zod
- Custom form components for consistent styling

Responsive Design Implementation:

The application uses Tailwind CSS's responsive utility classes to ensure proper display across all device sizes.

- Mobile-first approach

- Responsive grid layouts
- Flexible components
- Conditional rendering based on screen size

3.2 Backend Development

API Implementation:

- The backend follows RESTful principles with the following characteristics:
  - JSON data format for all requests and responses
  - Proper HTTP status codes
  - Error handling and validation
  - Authentication middleware

Authentication API

- User registration
- User login
- Session management with JWT

Book API

- Integration with Google Books API
- Book search and filtering
- Book details retrieval

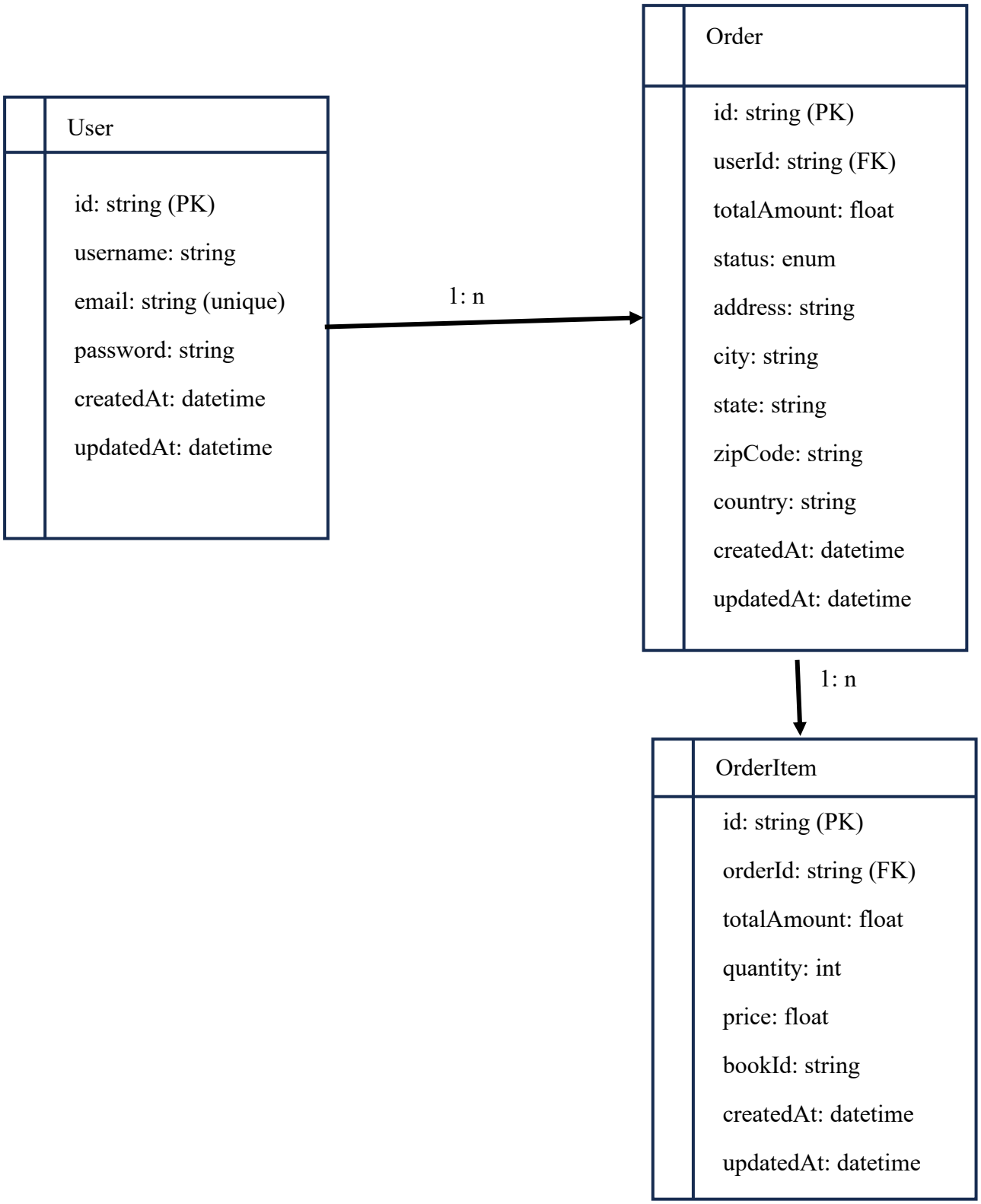
Shopping Cart API

- Cart management
- Order processing
-

Database Integration

- Prisma ORM for type-safe database access
- Database models for users, orders, and order items

4. Database Schema



5. API Documentation

5.1 External API Integration

Google Books API Integration

- The application integrates with the Google Books API to fetch book data.
- Base URL https://www.googleapis.com/books/v1/volumes
- Used in,
  - Book Search

    GET https://www.googleapis.com/books/v1/volumes?q={searchQuery}

  - Book Details

    GET https://www.googleapis.com/books/v1/volumes/{volumeId}

Implementation Details

- API requests are made server-side in Next.js API routes
- Responses are transformed to match the application's data model
- Error handling for API failures
- Optional caching for improved performance

5.2 Internal API Integration

Authentication

- Register User POST /api/auth/signup
- Login User POST/api/auth/login
- Logout User POST/api/auth/logout
- Get Current User POST/api/auth/current

Book

- Search Books GET /api/books?q={searchQuery}&limit={limit}
- Get book details GET /api/books/{id}
- Featured Books GET /api/books/featured

Order

- Create order POST /api/orders
- Get order details GET /api/orders/{id}

6. UI Documentation

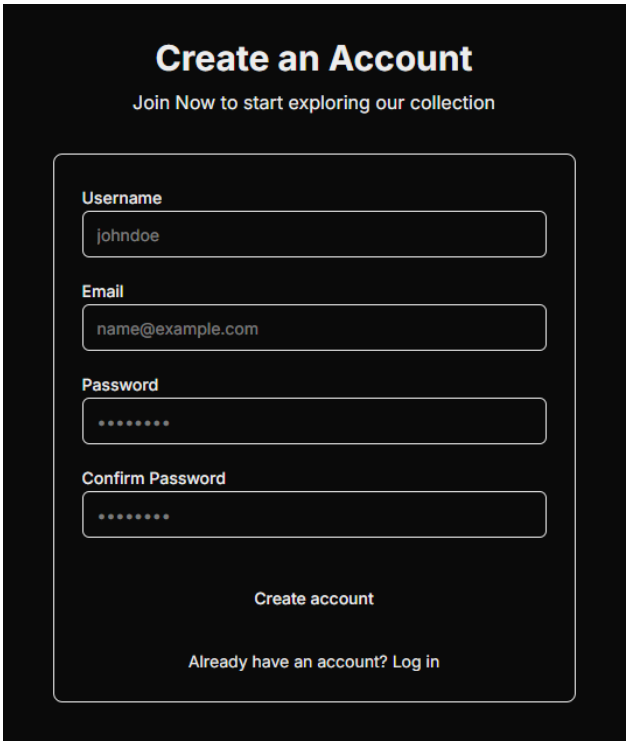
6.1 User Authentication Screens

Registration Page

Allow new users to create an account.

Form Validation

- Username: Minimum 3 characters
- Email: Valid email format
- Password: Minimum 6 characters
- Confirm password: Must match password

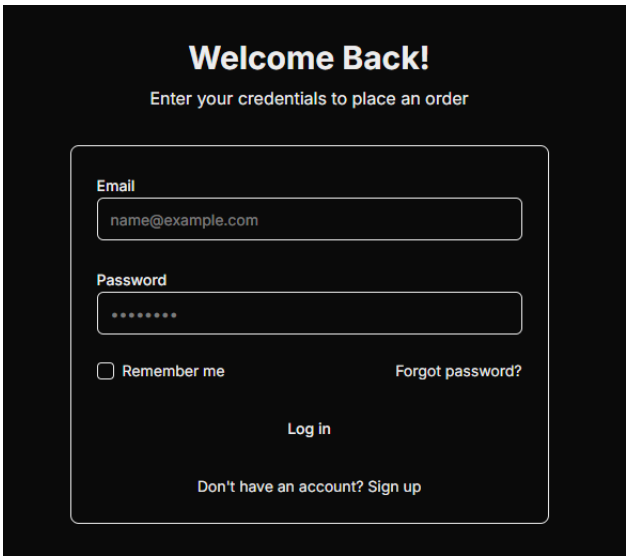


Login Page

Allow already registered users to log in.

Form Validation

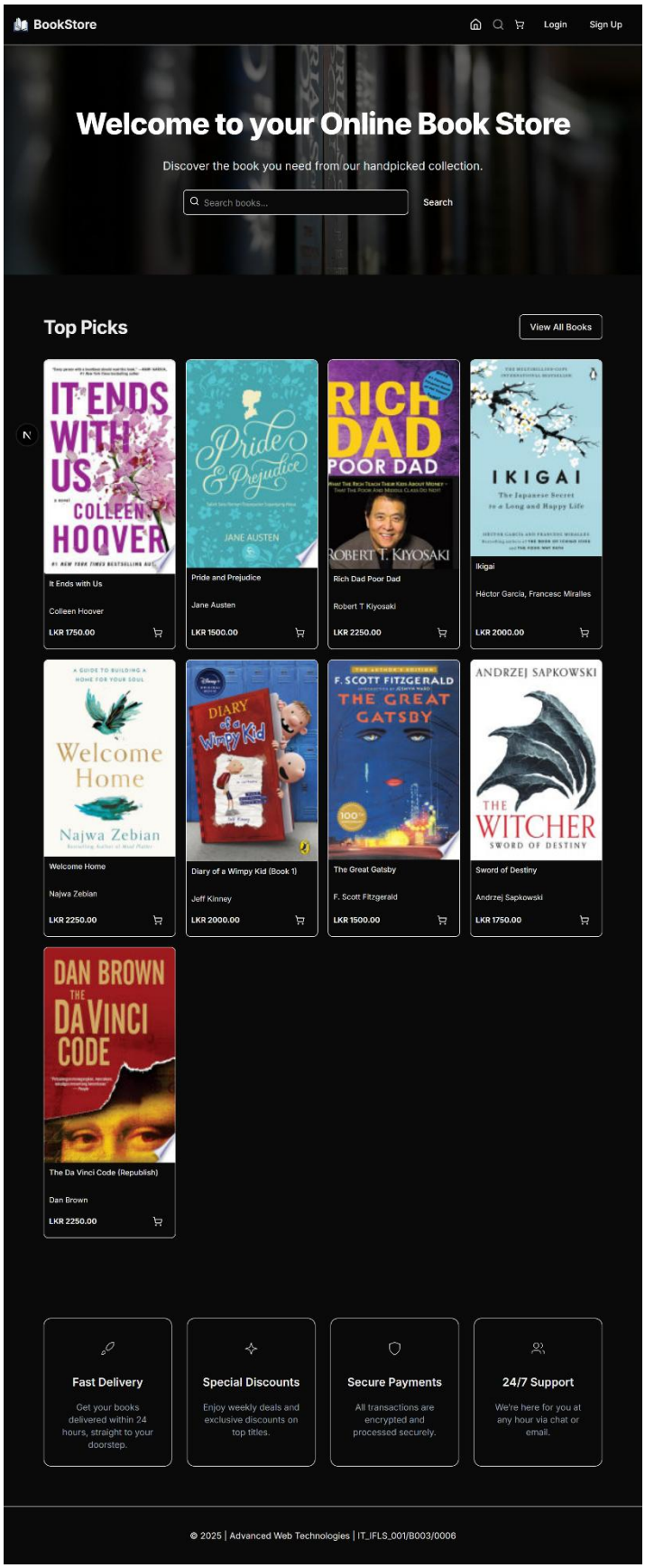
- Email: Valid email format
- Password: Required field

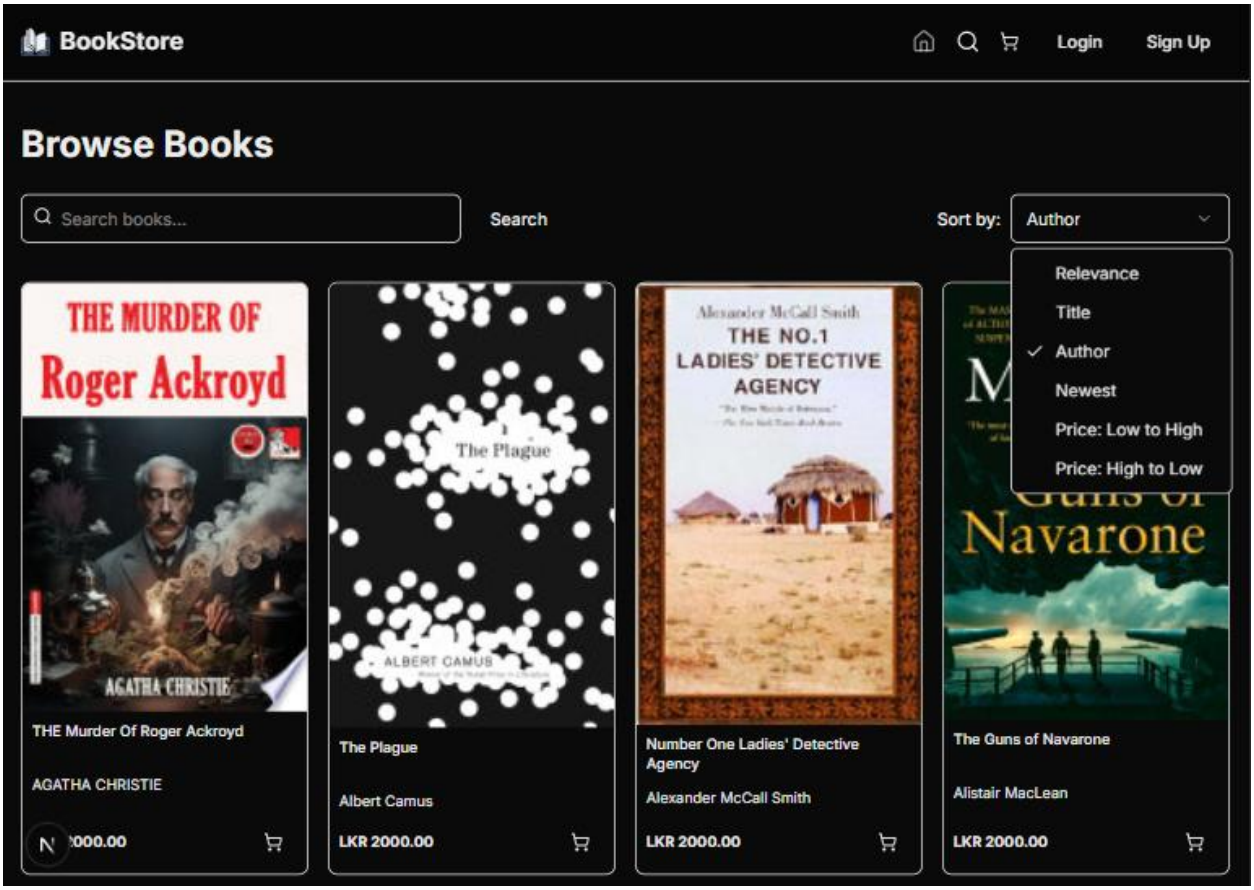


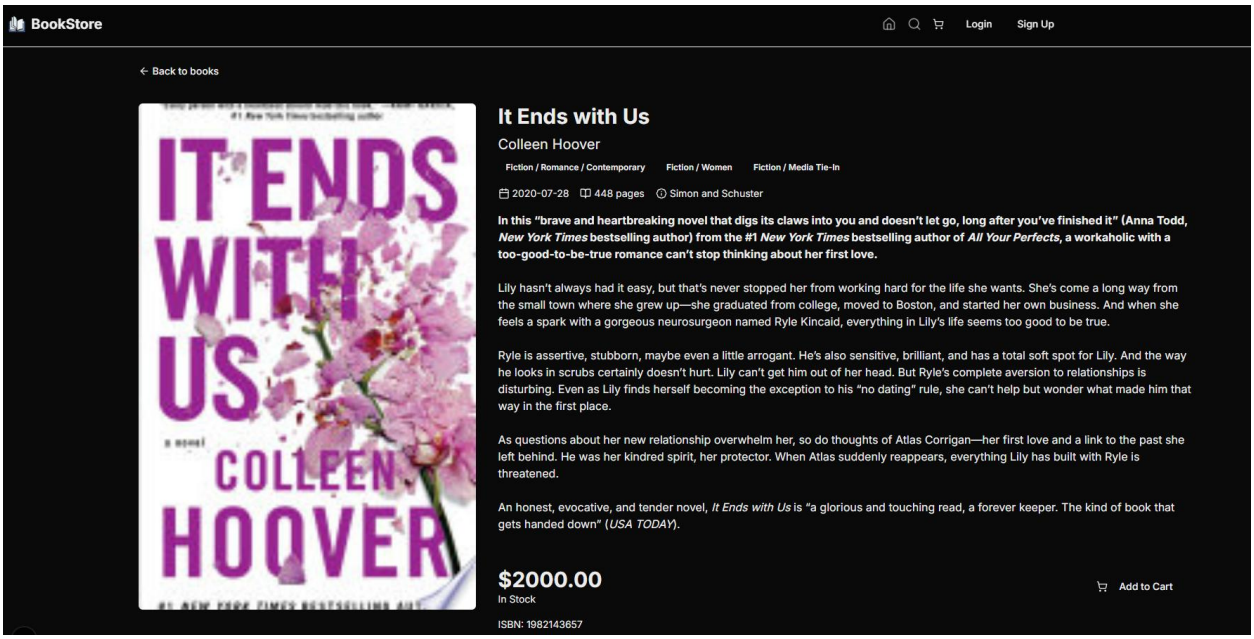
6.2 Book Listing and Search

Components and Features

- Home page with Search bar
- Search Page with Filter/ Sort by relevance, title, author, or price
- Detailed book page with individual book details (Title, Author, Description, Availability, Add to Cart button)
- Responsive book grid


BookStore
Login
Sign Up
Welcome to your Online Book Store
Discover the book you need from our handpicked collection.
Search books...
Search
Top Picks
View All Books
It Ends with Us
Colleen Hoover
LKR 1750.00
Pride and Prejudice
Jane Austen
LKR 1500.00
Rich Dad Poor Dad
Robert T Kiyosaki
LKR 2250.00
Ikigai
Hector Garcia, Francesc Miralles
LKR 2000.00
Welcome Home
Najwa Zebian
LKR 2250.00
Diary of a Wimpy Kid (Book 1)
Jeff Kinney
LKR 2000.00
The Great Gatsby
F. Scott Fitzgerald
LKR 1500.00
Sword of Destiny
Andrzej Sapkowski
LKR 1750.00
The Da Vinci Code (Republish)
Dan Brown
LKR 2250.00
Fast Delivery
Get your books delivered within 24 hours, straight to your doorstep.
Special Discounts
Enjoy weekly deals and exclusive discounts on top titles.
Secure Payments
All transactions are encrypted and processed securely.
24/7 Support
We're here for you at any hour via chat or email.
© 2025 | Advanced Web Technologies | IT_IFLS_001/B003/0006

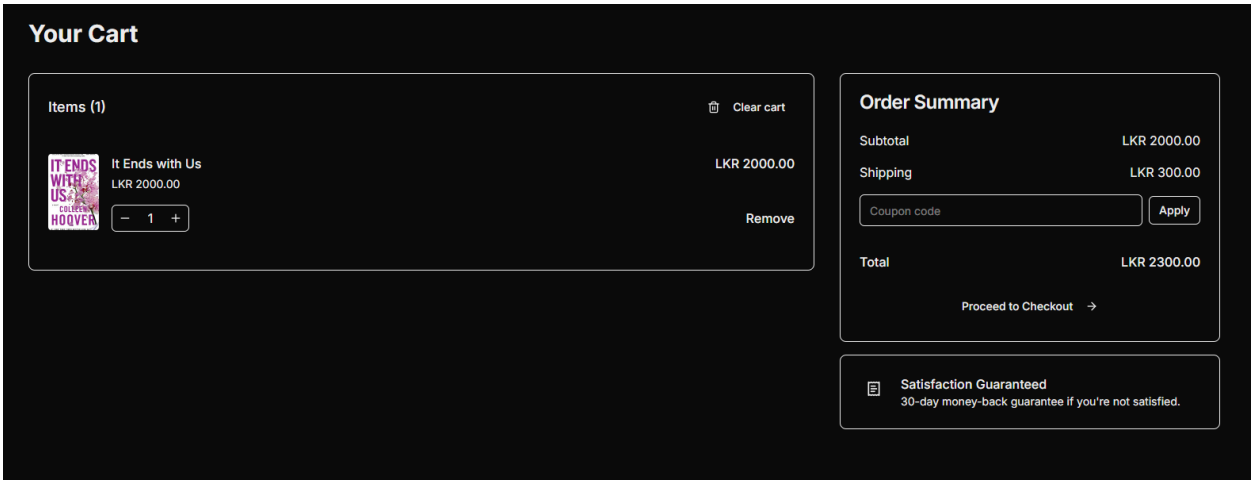

BookStore
Login
Sign Up
Browse Books
Search books...
Search
Sort by:
Author
Relevance
Title
Author
Newest
Price: Low to High
Price: High to Low
THE MURDER OF
Roger Ackroyd
AGATHA CHRISTIE
THE Murder Of Roger Ackroyd
AGATHA CHRISTIE
The Plague
ALBERT CAMUS
The Plague
Albert Camus
LKR 2000.00
Alexander McCall Smith
THE NO.1
LADIES' DETECTIVE
AGENCY
Number One Ladies' Detective Agency
Alexander McCall Smith
LKR 2000.00
Navarone
The Guns of Navarone
Alistair MacLean
LKR 2000.00


BookStore
Login
Sign Up
← Back to books
IT ENDS
WITH
US
COLLEEN
HOOVER
It Ends with Us
Colleen Hoover
Fiction / Romance / Contemporary
Fiction / Women
Fiction / Media Tie-In
2020-07-28
448 pages
Simon and Schuster
In this "brave and heartbreaking novel that digs its claws into you and doesn't let go, long after you've finished it" (Anna Todd, New York Times bestselling author) from the #1 New York Times bestselling author of All Your Perfects, a workaholic with a too-good-to-be-true romance can't stop thinking about her first love.
Lily hasn't always had it easy, but that's never stopped her from working hard for the life she wants. She's come a long way from the small town where she grew up—she graduated from college, moved to Boston, and started her own business. And when she feels a spark with a gorgeous neurosurgeon named Ryle Kincaid, everything in Lily's life seems too good to be true.
Ryle is assertive, stubborn, maybe even a little arrogant. He's also sensitive, brilliant, and has a total soft spot for Lily. And the way he looks in scrubs certainly doesn't hurt. Lily can't get him out of her head. But Ryle's complete aversion to relationships is disturbing. Even as Lily finds herself becoming the exception to his "no dating" rule, she can't help but wonder what made him that way in the first place.
As questions about her new relationship overwhelm her, so do thoughts of Atlas Corrigan—her first love and a link to the past she left behind. He was her kindred spirit, her protector. When Atlas suddenly reappears, everything Lily has built with Ryle is threatened.
An honest, evocative, and tender novel, It Ends with Us is "a glorious and touching read, a forever keeper. The kind of book that gets handed down" (USA TODAY).
$2000.00
In Stock
Add to Cart
ISBN: 1982143657

6.3 Shopping Cart

Components and Features

- Quantity adjustments
- Remove item
- Clear cart
- Order summary
- Calculate subtotal, shipping and total prices
- Proceed to checkout



6.4 Checkout

Components and Features

- Multistep checkout process
- Shipping address form with pre-filled Full Name and Email
- Payment method selection and card payment details form
- Review order summary and place order button

Back to cart

Checkout

1 Shipping 2 Payment 3 Review

Shipping Information

Full Name: user two

Email: user2@abc.com

Address:

City:

Province:

ZIP Code:

Country: Sri Lanka

Phone (optional):

Shipping Method

- (●) Standard Shipping — Delivery in 3-5 business days — LKR 300
- ( ) Express Shipping — Delivery in 1-2 business days — LKR 500

Continue to Payment ›

Order Summary

It Ends with Us × 1	LKR 2000.00
Subtotal	LKR 2000.00
Shipping	LKR 300
Total	LKR 2300.00

Secure Checkout
Your payment information is encrypted and secure.

Back to cart

Checkout

✓ Shipping 2 Payment 3 Review

Payment Information

Payment Method
( ) Cash on Delivery (COD) (●) Card

Card Number: 1234 5678 9012 3456

Name on Card:

Expiry Date: MM/YY

CVC: 123

☐ Save card for future purchases

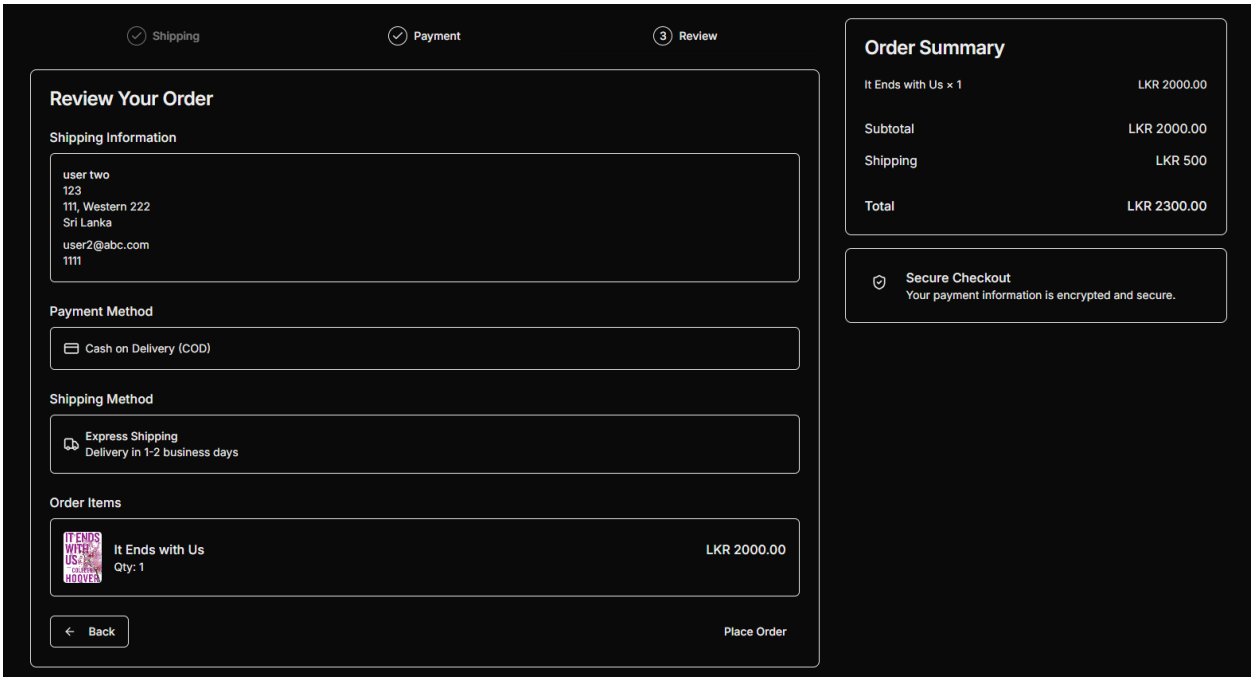
Your payment information is secure and encrypted

← Back Review Order ›

Order Summary

It Ends with Us × 1	LKR 2000.00
Subtotal	LKR 2000.00
Shipping	LKR 500
Total	LKR 2300.00

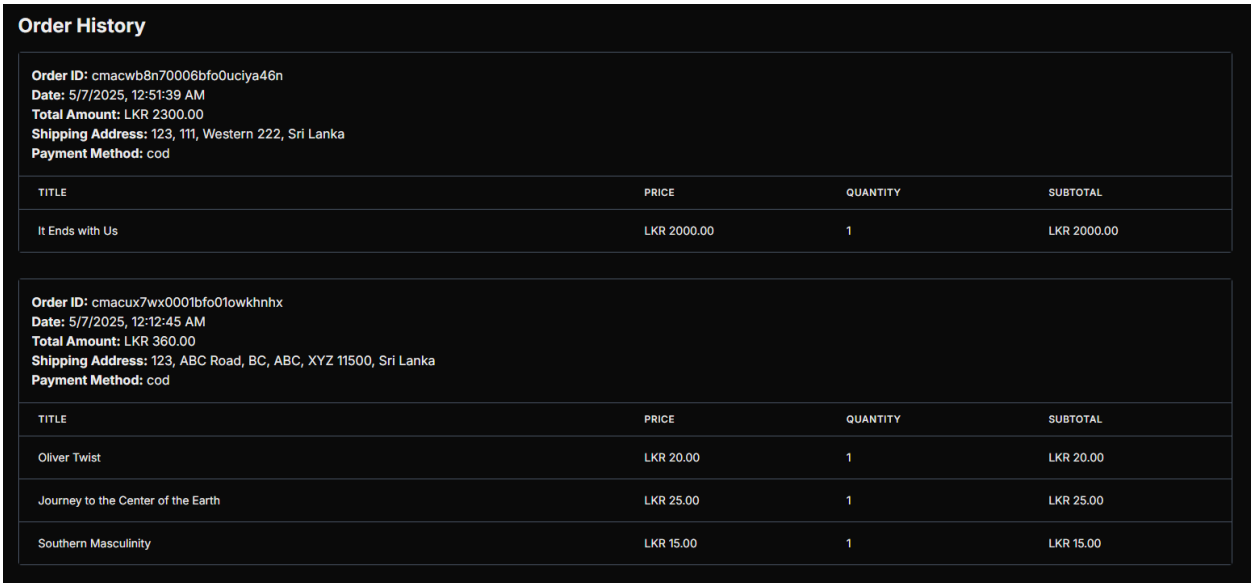
Secure Checkout
Your payment information is encrypted and secure.



6.5 Order History

Components and Features

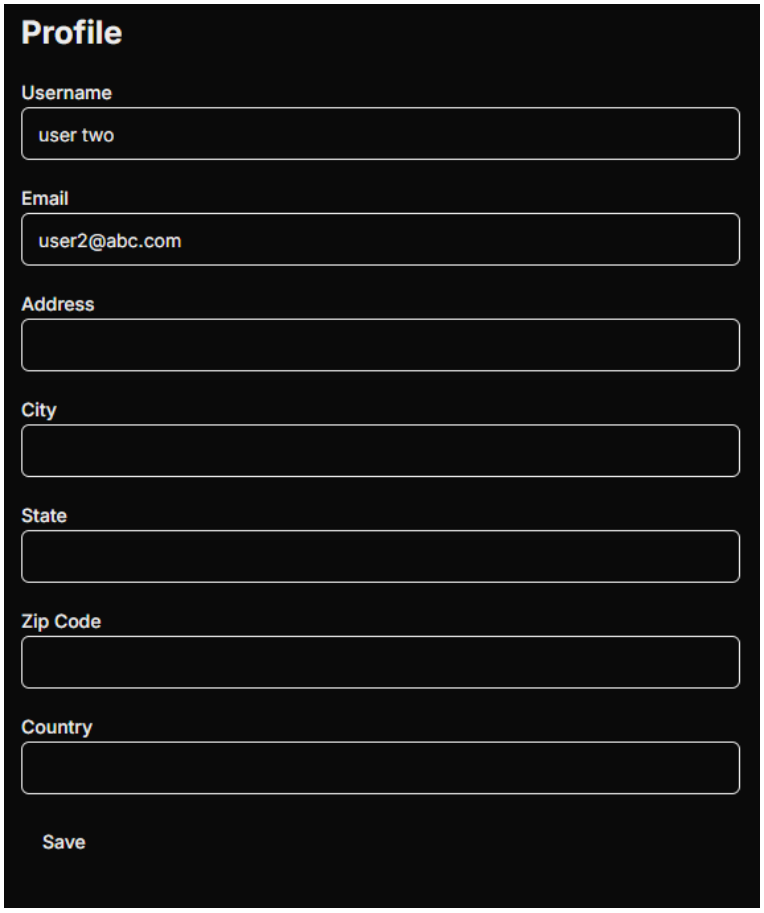
- View order history details in table view



6.6 User Profile

Features

- View and edit personal details
- Save changes



7. Security Implementation

Authentication Security

1. Password Security

- Passwords are hashed using bcrypt with a salt factor of 10
- Plain text passwords are never stored or transmitted
- Password validation enforces minimum length and complexity

2. JWT Implementation

- JSON Web Tokens used for authentication
- Tokens stored in HTTP-only cookies to prevent JavaScript access
- Token expiration set to 24 hours (or 30 days with "remember me")
- Secure and SameSite cookie attributes in production

3. Session Management

- Server-side validation of JWT tokens
- Automatic token invalidation on logout
- Token refresh mechanism (if implemented)

Data Security

1. Database Security

- Parameterized queries via Prisma ORM to prevent SQL injection
- Limited database user permissions
- Sensitive data encryption

2. Error Handling

- Custom error handling to prevent exposure of sensitive information
- Structured error responses

8. Setup and Deployment Guide

Prerequisites

- Node.js (v16.0.0 or higher)
- MySQLServer (v8.0 or higher)
- XAMMP server
- npm or yarn package manager
- Git

Setup

1. Clone the Repository
2. Install dependencies in project folder
    - npm install bcryptjs jsonwebtoken @prisma/client zod @hookform/resolvers lucide-react next-themes
    - npm install -D prisma @types/bcryptjs @types/jsonwebtoken
3. Configure environment variables in .env file in root directory
    - DATABASE_URL="mysql://username:password@localhost:3306/bookshop"
    - JWT_SECRET="your-secure-jwt-secret-key"
    - GOOGLE_BOOKS_API_KEY="your-google-books-api-key"
4. Setup the MySQL database
    - Create database
    - Initialize prisma npx prisma init --datasource-provider mysql
    - Apply migration npx prisma migrate dev --name init
    - Generate client npx prisma generate
5. Start the development server
    - Run Apache and MySQL from xampp control panel
    - npm run dev

9. Code Documentation

Register Page

```python
"use client"

import { useState } from "react"
import Link from "next/link"
import { useRouter } from "next/navigation"
import { z } from "zod"
import { useForm } from "react-hook-form"
import { zodResolver } from "@hookform/resolvers/zod"
import { Button } from "@/components/ui/button"
import { Form, FormControl, FormField, FormItem, FormLabel, FormMessage } from "@/components/ui/form"
import { Input } from "@/components/ui/input"
import { useToast } from "@/components/ui/use-toast"

const formSchema = z
  .object({
    username: z.string().min(3, { message: "Username must be at least 3 characters" }).max(50),
    email: z.string().email({ message: "Please enter a valid email address" }),
    password: z.string().min(6, {
      message: "Password must be at least 6 characters",
    }),
    confirmPassword: z.string(),
  })
  .refine((data) => data.password === data.confirmPassword, {
    message: "Passwords do not match",
    path: ["confirmPassword"],
  })

export default function SignupPage() {
  const [isLoading, setIsLoading] = useState(false)
  const router = useRouter()
  const { toast } = useToast()

  const form = useForm<z.infer<typeof formSchema>>({
    resolver: zodResolver(formSchema),
    defaultValues: {
      username: "",
      email: "",
      password: "",
      confirmPassword: "",
    },
  })

  async function onSubmit(values: z.infer<typeof formSchema>) {
    setIsLoading(true)
    try {
      const response = await fetch("/api/auth/signup", {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({
          username: values.username,
          email: values.email,
          password: values.password,
        }),
      })

      const data = await response.json()

      if (!response.ok) {
        throw new Error(data.message || "Something went wrong")
      }

      toast({
        title: "Account created",
        description: "Your account has been created successfully. Please log in.",
      })
      router.push("/login")
    } catch (error: any) {
      toast({
        variant: "destructive",
        title: "Registration failed",
        description: error.message || "Something went wrong",
      })
    } finally {
      setIsLoading(false)
    }
  }

  return (
    <div class="container mx-auto px-4 py-12">
      <div class="flex flex-col items-center justify-center space-y-8">
        <div class="text-center space-y-2">
          <h1 class="text-3xl font-bold">Create an Account</h1>
          <p class="text-muted-foreground">Join Now to start exploring our collection</p>
        </div>
        <div class="w-full max-w-md space-y-6 rounded-lg border p-6 shadow-sm">
          <Form {...form}>
            <form onSubmit={form.handleSubmit(onSubmit)} class="space-y-4">
              <FormField
                control={form.control}
                name="username"
                render={({ field }) => (
                  <FormItem>
                    <FormLabel>Username</FormLabel>
                    <FormControl>
                      <Input placeholder="johndoe" {...field} />
                    </FormControl>
                    <FormMessage />
                  </FormItem>
                )}
              />
              <FormField
                control={form.control}
                name="email"
                render={({ field }) => (
                  <FormItem>
                    <FormLabel>Email</FormLabel>
                    <FormControl>
                      <Input placeholder="name@example.com" {...field} />
                    </FormControl>
                    <FormMessage />
                  </FormItem>
                )}
              />
              <FormField
                control={form.control}
                name="password"
                render={({ field }) => (
                  <FormItem>
                    <FormLabel>Password</FormLabel>
                    <FormControl>
                      <Input type="password" placeholder="********" {...field} />
                    </FormControl>
                    <FormMessage />
                  </FormItem>
                )}
              />
              <FormField
                control={form.control}
                name="confirmPassword"
                render={({ field }) => (
                  <FormItem>
                    <FormLabel>Confirm Password</FormLabel>
                    <FormControl>
                      <Input type="password" placeholder="********" {...field} />
                    </FormControl>
                    <FormMessage />
                  </FormItem>
                )}
              />
              <Button type="submit" className="w-full mt-4" disabled={isLoading}>
                {isLoading ? "Creating account..." : "Create account"}
              </Button>
            </form>
          </Form>
          <div class="text-center text-sm">
            Already have an account? {" "}
            <Link href="/login" class="text-primary hover:underline">
              Log in
            </Link>
          </div>
        </div>
      </div>
    </div>
  )
}
```

Register authentication

```
import { type NextRequest, NextResponse } from "next/server"
import bcrypt from "bcryptjs"
import { db } from "@/lib/db"

export async function POST(request: NextRequest) {
  try {
    const { username, email, password } = await request.json()

    // Basic validation
    if (!username || !email || !password) {
      return NextResponse.json({ message: "Missing required fields" }, { status: 400 })
    }

    // Check if user with this email already exists
    const existingUser = await db.user.findUnique({
      where: { email },
    })

    if (existingUser) {
      return NextResponse.json({ message: "User with this email already exists" }, { status: 400 })
    }

    // Hash password
    const hashedPassword = await bcrypt.hash(password, 10)

    // Create new user
    const newUser = await db.user.create({
      data: {
        username,
        email,
        password: hashedPassword,
      },
    })

    // Return success response (without sensitive data)
    return NextResponse.json(
      {
        message: "User created successfully",
        user: {
          id: newUser.id,
          username: newUser.username,
          email: newUser.email,
        },
      },
      { status: 201 },
    )
  } catch (error) {
    console.error("Error creating user:", error)
    return NextResponse.json({ message: "An error occurred while creating the user" }, { status: 500 })
  }
}
```

Login Page

```python
"use client"

import { useState } from "react"
import Link from "next/link"
import { useRouter } from "next/navigation"
import { z } from "zod"
import { useForm } from "react-hook-form"
import { zodResolver } from "@hookform/resolvers/zod"
import { Button } from "@/components/ui/button"
import { Form, FormControl, FormField, FormItem, FormLabel, FormMessage } from "@/components/ui/form"
import { Input } from "@/components/ui/input"
import { Checkbox } from "@/components/ui/checkbox"
import { useToast } from "@/components/ui/use-toast"
import { useAuth } from "@/context/auth-context"

const formSchema = z.object({
  email: z.string().email({ message: "Please enter a valid email address" }),
  password: z.string().min(6, {
    message: "Password must be at least 6 characters",
  }),
  rememberMe: z.boolean().default(false),
})

export default function LoginPage() {
  const [isLoading, setIsLoading] = useState(false)
  const router = useRouter()
  const { toast } = useToast()
  const { login } = useAuth()

  const form = useForm<z.infer<typeof formSchema>>({
    resolver: zodResolver(formSchema),
    defaultValues: {
      email: "",
      password: "",
      rememberMe: false,
    },
  })

  async function onSubmit(values: z.infer<typeof formSchema>) {
    setIsLoading(true)
    try {
      await login(values.email, values.password, values.rememberMe)
      toast({
        title: "Login successful",
        description: "Welcome back to BookHaven",
      })
      router.push("/")
    } catch (error: any) {
      toast({
        variant: "destructive",
        title: "Login failed",
        description: error.message || "Please check your credentials and try again",
      })
    } finally {
      setIsLoading(false)
    }
  }

  return (
    <div class=Name="container mx-auto px-4 py-12">
      <div class=Name="flex flex-col items-center justify-center space-y-8">
        <div class=Name="text-center space-y-2">
          <h1 class=Name="text-3xl font-bold">Welcome Back!</h1>
          <p class=Name="text-muted-foreground">Enter your credentials to place an order</p>
        </div>
        <div class=Name="w-full max-w-md space-y-6 rounded-lg border p-6 shadow-sm">
          <form {...form}>
            <form onSubmit={form.handleSubmit(onSubmit)} class=Name="space-y-6">
              <formfield
                control={form.control}
                name="email"
                render={({ field }) => (
                  <FormItem>
                    <FormLabel>Email</FormLabel>
                    <FormControl>
                      <Input placeholder="name@example.com" {...field} />
                    </FormControl>
                    <FormMessage />
                  </FormItem>
                )}
              />
              <FormField
                control={form.control}
                name="password"
                render={({ field }) => (
                  <FormItem>
                    <FormLabel>Password</FormLabel>
                    <FormControl>
                      <Input type="password" placeholder="********" {...field} />
                    </FormControl>
                    <FormMessage />
                  </FormItem>
                )}
              />
              <div class=Name="flex items-center justify-between">
                <FormField
                  control={form.control}
                  name="rememberMe"
                  render={({ field }) => (
                    <FormItem class=Name="flex flex-row items-center space-x-2 space-y-0">
                      <FormControl>
                        <Checkbox checked={field.value} onCheckedChange={field.onChange} />
                      </FormControl>
                      <FormLabel class=Name="text-sm font-normal">Remember me</FormLabel>
                    </FormItem>
                  )}
                />
                <Link href="/forgot-password" class=Name="text-sm text-muted-foreground hover:text-primary">
                  Forgot password?
                </Link>
              </div>
              <Button type="submit" class=Name="w-full" disabled={isLoading}>
                {isLoading ? "Logging in..." : "Log in"}
              </Button>
            </form>
          </Form>
          <div class=Name="text-center text-sm">
            Don&apos;t have an account?{" "}
            <Link href="/signup" class=Name="text-primary hover:underline">
              Sign up
            </Link>
          </div>
        </div>
      </div>
    </div>
  )
}
```

Login Authentication

```
import { type NextRequest, NextResponse } from "next/server"
import bcrypt from "bcryptjs"
import jwt from "jsonwebtoken"
import { cookies } from "next/headers"
import { db } from "@/lib/db"

// JWT secret key (should be in .env file in production)
const JWT_SECRET = process.env.JWT_SECRET || "your-secret-key"

export async function POST(request: NextRequest) {
  try {
    const { email, password, rememberMe } = await request.json()

    // Basic validation
    if (!email || !password) {
      return NextResponse.json({ message: "Email and password are required" }, { status: 400 })
    }

    // Find user by email
    const user = await db.user.findUnique({
      where: { email },
    })

    if (!user) {
      return NextResponse.json({ message: "Invalid email or password" }, { status: 401 })
    }

    // Compare passwords
    const passwordMatch = await bcrypt.compare(password, user.password)

    if (!passwordMatch) {
      return NextResponse.json({ message: "Invalid email or password" }, { status: 401 })
    }

    // Create JWT token
    const token = jwt.sign({ id: user.id, username: user.username, email: user.email }, JWT_SECRET, {
      expiresIn: rememberMe ? "30d" : "1d",
    })

    // Set cookie
    ;(await
      // Set cookie
      cookies()).set({
      name: "auth_token",
      value: token,
      httpOnly: true,
      path: "/",
      secure: process.env.NODE_ENV === "production",
      maxAge: rememberMe ? 30 * 24 * 60 * 60 : 24 * 60 * 60, // 30 days or 1 day
      sameSite: "strict",
    })

    // Return success response (without sensitive data)
    return NextResponse.json({
      message: "Login successful",
      user: {
        id: user.id,
        username: user.username,
        email: user.email,
      },
    })
  } catch (error) {
    console.error("Login error:", error)
    return NextResponse.json({ message: "An error occurred during login" }, { status: 500 })
  }
}
```

Main book listing page with API integration

```
import { type NextRequest, NextResponse } from "next/server"

// Google Books API endpoint
const GOOGLE_BOOKS_API = "https://www.googleapis.com/books/v1/volumes"

export async function GET(request: NextRequest) {
  try {
    const searchParams = request.nextUrl.searchParams
    const query = searchParams.get("q") || ""
    const limit = Number.parseInt(searchParams.get("limit") || "40")
    const sort = searchParams.get("sort") || "relevance"

    // Determine orderBy parameter for Google Books API
    let orderBy = "relevance"
    if (sort === "relevance" || sort === "newest") {
      orderBy = sort
    }

    // If no query, return some popular books
    if (!query) {
      const response = await fetch(`${GOOGLE_BOOKS_API}?q=subject:fiction&orderBy=${orderBy}&maxResults=40`)

      if (!response.ok) {
        throw new Error("Failed to fetch books from Google Books API")
      }

      const data = await response.json()

      let books = data.items?.map(formatBookData).slice(0, limit) || []

      // Manual sorting for title, author, price
      books = sortBooks(books, sort)

      return NextResponse.json(books)
    }

    // Fetch books based on the query
    const params = new URLSearchParams()
    params.append("q", query)
    params.append("orderBy", orderBy)
    params.append("maxResults", "40")

    const response = await fetch(`${GOOGLE_BOOKS_API}?${params.toString()}`)

    if (!response.ok) {
      throw new Error("Failed to fetch books from Google Books API")
    }

    const data = await response.json()

    let books = data.items?.map(formatBookData).slice(0, limit) || []

    // Manual sorting for title, author, price
    books = sortBooks(books, sort)

    return NextResponse.json(books)
  } catch (error) {
    console.error("Error fetching books:", error)
    return NextResponse.json({ error: "Failed to fetch books" }, { status: 500 })
  }
}

// Helper function to sort books manually
function sortBooks(books: any[], sort: string) {
  switch (sort) {
    case "title":
      return books.sort((a, b) => {
        const firstWordA = a.title.split(" ")[0].toLowerCase()
        const firstWordB = b.title.split(" ")[0].toLowerCase()
        return firstWordA.localeCompare(firstWordB, undefined, { sensitivity: 'base' })
      })
    case "author":
      return books.sort((a, b) => {
        const authorA = (a.authors && a.authors.length > 0) ? a.authors[0].split(" ")[0].toLowerCase() : ""
        const authorB = (b.authors && b.authors.length > 0) ? b.authors[0].split(" ")[0].toLowerCase() : ""
        return authorA.localeCompare(authorB, undefined, { sensitivity: 'base' })
      })
    case "price-low":
      return books.sort((a, b) => a.price - b.price)
    case "price-high":
      return books.sort((a, b) => b.price - a.price)
    default:
      return books
  }
}

// In-memory cache for book prices by book id
const priceCache = new Map<string, number>()

// Helper function to format book data
function formatBookData(item: any) {
  const volumeInfo = item.volumeInfo || {}
  const saleInfo = item.saleInfo || {}

  // Check if price is cached for this book
  if (priceCache.has(item.id)) {
    return {
      id: item.id,
      title: volumeInfo.title || "Unknown Title",
      authors: volumeInfo.authors || [],
      publishedDate: volumeInfo.publishedDate || "",
      description: volumeInfo.description || "",
      thumbnail: volumeInfo.imageLinks?.thumbnail || "",
      price: priceCache.get(item.id)!,
    }
  }

  // Convert available price to LKR by multiplying by 300, else assign fixed default price of 2000 LKR
  const price = saleInfo.retailPrice?.amount ? saleInfo.retailPrice.amount * 300 : 2000

  // Cache the price
  priceCache.set(item.id, price)

  return {
    id: item.id,
    title: volumeInfo.title || "Unknown Title",
    authors: volumeInfo.authors || [],
    publishedDate: volumeInfo.publishedDate || "",
    description: volumeInfo.description || "",
    thumbnail: volumeInfo.imageLinks?.thumbnail || "",
    price,
  }
}
```

Search Book

```
"use client"

import { useState, useEffect, SetStateAction } from "react"
import { useRouter, usePathname, useSearchParams } from "next/navigation"
import { Search } from "lucide-react"
import { Input } from "./ui/input"
import { Button } from "./ui/button"

export function BookSearch() {
  const router = useRouter()
  const pathname = usePathname()
  const searchParams = useSearchParams()
  const initialQuery = searchParams.get("q") || ""
  const [searchQuery, setSearchQuery] = useState(initialQuery)

  useEffect(() => {
    const handler = setTimeout(() => {
      const trimmedQuery = searchQuery.trim()
      const url = new URL(window.location.href)
      if (trimmedQuery) {
        url.searchParams.set("q", trimmedQuery)
      } else {
        url.searchParams.delete("q")
      }
      router.replace(url.toString(), { scroll: false })
    }, 500)

    return () => clearTimeout(handler)
  }, [searchQuery, router])

  const handleSubmit = (e: React.FormEvent) => {
    e.preventDefault()
    const trimmedQuery = searchQuery.trim()
    if (trimmedQuery) {
      router.push(`/books?q=${encodeURIComponent(trimmedQuery)}`)
    }
  }

  return (
    <form onSubmit={handleSubmit} className="flex w-full items-center space-x-2">
      <div className="relative flex-1">
        <Search className="absolute left-2.5 top-2.5 h-4 w-4 text-muted-foreground" />
        <Input
          type="search"
          placeholder="Search books..."
          className="pl-8"
          value={searchQuery}
          onChange={(e: { target: { value: SetStateAction<string> } }) => setSearchQuery(e.target.value)}
        />
      </div>
      <Button type="submit">Search</Button>
    </form>
  )
}
```

Cart Page

```
'use client'

import Link from 'next/link'
import { useState } from 'react'
import { Trash2, Plus, Minus, ArrowRight, ShoppingBag, Receipt[illegible] } from "lucide-react"
import { Button } from "@/components/ui/button"
import { Card, CardContent, CardFooter, CardHeader, CardTitle } from "@/components/ui/card"
import { Input } from "@/components/ui/input"
import { Separator } from "@/components/ui/separator"
import { useCart } from "@/context/cart-context"
import { useAuth } from "@/context/auth-context"

export default function CartPage() {
  const { items, removeItem, updateQuantity, subtotal, clearCart } = useCart()
  const { user } = useAuth()
  const, [couponCode, setCouponCode] = useState("")

  const shippingCost = 500
  const total = subtotal + shippingCost

  const handleQuantityChange = (id: string, newQuantity: number) => {
    if (newQuantity >= 1) {
      updateQuantity(id, newQuantity)
    }
  }

  if (items.length === 0) {
    return (
      <div className="container mx-auto px-4 py-12">
        <h1 className="text-3xl font-bold mb-8">Your Cart</h1>
        <div class="text-center py-12 space-y-4">
          <div class="mx-auto flex h-20 w-20 items-center justify-center rounded-full bg-muted">
            <ShoppingBag class="h 10 w 10" />
          </div>
          <h2 class="text-xl font-medium">Your cart is empty</h2>
          <p class="text-muted-foreground">Looks like you haven't added anything to your cart yet.</p>
          <Button asChild class="mt 4">
            <Link href="/books">Browse Books</Link>
          </Button>
        </div>
      </div>
    )
  }

  return (
    <div class="container mx-auto px-4 py-12">
      <h1 class="text-3xl font-bold mb-8">Your Cart</h1>
      <div class="grid grid cols 1 gap 8 lg:grid cols 3">
        <div class="lg:col span 2">
          <div class="rounded-lg border">
            <div class="p-6">
              <div class="flex items center justify between pb 4">
                <h2 class="text lg font-medium">Items ({items.length})</h2>
                <Button variant="ghost" size="sm" onClick={clearCart} class="text-muted-foreground">
                  <Trash2 class="mr-2 h-4 w-4" />
                  Clear cart
                </Button>
              </div>
              <Separator />
              <div class="divide y">
                {items.map(({item}) => (
                  <div key={item.id} class="py-6">
                    <div class="flex items-start gap 4">
                      <div class="h 24 w 16 flex shrink 0 overflow hidden rounded bg muted">
                        {item.thumbnail ? (
                          <Image
                            src={item.thumbnail || "/placeholder.svg"}
                            alt={item.title}
                            class="h-full w-full object-cover object-center"
                          />
                        ) : (
                          <div class="flex h full w full items center justify center text muted foreground text xs">
                            No image
                          </div>
                        )}
                      </div>
                      <div class="flex flex-1 flex-col">
                        <div class="flex justify-between">
                          <div>
                            <h3 class="font-medium">
                              <Link href={`/books/${item.id}`} class="hover:underline">
                                {item.title}
                              </Link>
                            </h3>
                            <p class="mt-1 text-sm text-muted-foreground">KES {item.price.toFixed(2)}</p>
                          </div>
                          <p class="text medium">KES {(item.price * item.quantity).toFixed(2)}</p>
                        </div>
                        <div class="mt-4 flex items-center justify-between">
                          <div class="flex items center border rounded md">
                            <button
                              type="button"
                              class="px-2"
                              onClick={() => handleQuantityChange(item.id, item.quantity - 1)}
                              aria label="Decrease quantity"
                              >
                                <Minus class="h-3 w-3" />
                              </button>
                              <span class="px 2 py 1 text center w 8">{item.quantity}</span>
                              <button
                                type="button"
                                class="px 2"
                                onClick={() => handleQuantityChange(item.id, item.quantity + 1)}
                                aria-label="Increase quantity"
                                >
                                  <Plus class="h 4 w 4" />
                                </button>
                              </div>
                              <button
                                type="button"
                                onClick={() => removeItem(item.id)}
                                class="font-medium text-primary hover:text-primary/80"
                              >
                                Remove
                              </button>
                            </div>
                          </div>
                        </div>
                      </div>
                    ))}
                  </div>
                </div>
              </div>
            </div>
            <div class="lg:col span 1">
              <Card>
                <CardHeader>
                  <CardTitle>Order Summary</CardTitle>
                </CardHeader>
                <CardContent class="space-y-4">
                  <div class="flex justify-between">
                    <span>Subtotal</span>
                    <span>KES {subtotal.toFixed(2)}</span>
                  </div>
                  <div class="flex justify-between">
                    <span>Shipping</span>
                    <span>KES {shippingCost.toFixed(2)}</span>
                  </div>
                  <div class="flex items-center gap-2">
                    <Input placeholder="Coupon code" value={couponCode} onChange={(e) => setCouponCode(e.target.value)} />
                    <Button variant="outline" size="sm">
                      Apply
                    </Button>
                  </div>
                  <Separator />
                  <div class="flex justify-between font-medium">
                    <span>Total</span>
                    <span>KES {total.toFixed(2)}</span>
                  </div>
                </CardContent>
                <CardFooter>
                  <Button asChild class="w full">
                    <Link href={user ? "/checkout" : "/login?redirect=/checkout"}>
                      Proceed to Checkout
                      <ArrowRight class="ml-2 h-4 w-4" />
                    </Link>
                  </Button>
                </CardFooter>
              </Card>
              <Card class="mt 4">
                <CardContent class="pt-6">
                  <div class="flex items-start space-x-4">
                    <div class="rounded-full bg-primary/10 p-2">
                      <ReceiptText class="h 5 w 5 text primary" />
                    </div>
                    <div>
                      <h2 class="font-medium">Satisfaction Guaranteed</h2>
                      <p class="text sm text muted foreground">30 day money back guarantee if you're not satisfied.</p>
                    </div>
                  </div>
                </CardContent>
              </Card>
            </div>
          </div>
        </div>
      )
}
```

Cart context management

```typescript
"use client"

import { createContext, useContext, useState, useEffect, type ReactNode } from "react"

export type CartItem = {
  id: string
  title: string
  price: number
  thumbnail: string
  quantity: number
}

type CartContextType = {
  items: CartItem[]
  addItem: (item: CartItem) => void
  removeItem: (id: string) => void
  updateQuantity: (id: string, quantity: number) => void
  clearCart: () => void
  subtotal: number
}

const CartContext = createContext<CartContextType | undefined>(undefined)

const CART_STORAGE_KEY = "bookshop-cart"

export function CartProvider({ children }: { children: ReactNode }) {
  const [items, setItems] = useState<CartItem[]>([])

  useEffect(() => {
    // Load cart from localStorage on initial render
    const savedCart = localStorage.getItem(CART_STORAGE_KEY)
    if (savedCart) {
      try {
        setItems(JSON.parse(savedCart))
      } catch (error) {
        console.error("Failed to parse cart from localStorage", error)
      }
    }
  }, [])

  useEffect(() => {
    // Save cart to localStorage whenever it changes
    localStorage.setItem(CART_STORAGE_KEY, JSON.stringify(items))
  }, [items])

  const addItem = (item: CartItem) => {
    setItems((prevItems) => {
      const existingItem = prevItems.find((i) => i.id === item.id)

      if (existingItem) {
        return prevItems.map((i) => (i.id === item.id ? { ...i, quantity: i.quantity + item.quantity } : i))
      }

      return [...prevItems, item]
    })
  }

  const removeItem = (id: string) => {
    setItems((prevItems) => prevItems.filter((item) => item.id !== id))
  }

  const updateQuantity = (id: string, quantity: number) => {
    if (quantity <= 0) {
      removeItem(id)
      return
    }

    setItems((prevItems) => prevItems.map((item) => (item.id === id ? { ...item, quantity } : item)))
  }

  const clearCart = () => {
    setItems([])
  }

  const subtotal = items.reduce((total, item) => total + item.price * item.quantity, 0)

  return (
    <CartContext.Provider
      value={{
        items,
        addItem,
        removeItem,
        updateQuantity,
        clearCart,
        subtotal,
      }}
    >
      {children}
    </CartContext.Provider>
  )
}

export const useCart = () => {
  const context = useContext(CartContext)
  if (context === undefined) {
    throw new Error("useCart must be used within a CartProvider")
  }
  return context
}
```

Checkout page with API integration

Checkout order confirmation

```
"use client"

import Link from "next/link"
import { useEffect, useState } from "react"
import { CheckCircle, Package, BookOpen, ArrowRight } from "lucide-react"
import { Button } from "@/components/ui/button"
import { Card, CardContent } from "@/components/ui/card"
import { Separator } from "@/components/ui/separator"
import { useRouter } from "next/navigation"
import { useAuth } from "@/context/auth-context"

export default function ConfirmationPage() {
  const router = useRouter()
  const { user } = useAuth()
  const [orderNumber, setOrderNumber] = useState("")
  const [estimatedDelivery, setEstimatedDelivery] = useState("")
  const [isLoading, setIsLoading] = useState(true)

  useEffect(() => {
    // Check if user came from checkout flow
    const checkOrderStatus = async () => {
      try {
        // Try to get the most recent order
        if (user?.id) {
          const response = await fetch(`/api/orders?userId=${user.id}`)

          if (response.ok) {
            const orders = await response.json()

            if (orders && orders.length > 0) {
              // Get the most recent order
              const latestOrder = orders[0]
              setOrderNumber(latestOrder.id)

              // Calculate estimated delivery date (3-7 business days from order date)
              const orderDate = new Date(latestOrder.created_at)
              const deliveryDate = new Date(orderDate)
              deliveryDate.setDate(orderDate.getDate() + 5 + Math.floor(Math.random() * 3))

              // Format date as Month Day, Year
              const options: Intl.DateTimeFormatOptions = {
                year: "numeric",
                month: "long",
                day: "numeric",
              }
              setEstimatedDelivery(deliveryDate.toLocaleDateString("en-US", options))
              setIsLoading(false)
              return
            }
          }
        }

        // If no order found or error, generate placeholder data
        generatePlaceholderData()
      } catch (error) {
        console.error("Error fetching order:", error)
        generatePlaceholderData()
      }
    }

    const generatePlaceholderData = () => {
      // Generate random order number
      const randomOrderNumber = Math.floor(100000000 + Math.random() * 900000000).toString()
      setOrderNumber(`ORD-${randomOrderNumber}`)

      // Calculate estimated delivery date (3-7 business days from now)
      const today = new Date()
      const deliveryDate = new Date(today)
      deliveryDate.setDate(today.getDate() + 3 + Math.floor(Math.random() * 5))

      // Format date as Month Day, Year
      const options: Intl.DateTimeFormatOptions = {
        year: "numeric",
        month: "long",
        day: "numeric",
      }
      setEstimatedDelivery(deliveryDate.toLocaleDateString("en-US", options))
      setIsLoading(false)
    }

    checkOrderStatus()
  }, [user])

  if (isLoading) {
    return (
      <div class="min-h-screen bg-gray-50 dark:bg-gray-900 flex-col-center">
        <p>Loading order information...</p>
      </div>
    )
  }

  return (
    <div class="min-h-screen bg-gray-50 dark:bg-gray-900">
      <div class="text-center mb-8">
        <div class="inline-flex h-20 w-20 items-center justify-center rounded-full bg-primary/10 mb-4">
          <CheckCircle class="h-10 w-10 text-primary" />
        </div>
        <h1 class="text-3xl font-bold">Order Confirmed!</h1>
        <p class="text-muted-foreground mt-2">
          Thank you for your purchase. Your order has been received and is being processed.
        </p>
      </div>

      <Card class="mb-8">
        <CardContent class="pt-6">
          <div class="space-y-4">
            <div class="flex justify-between">
              <span class="text-muted-foreground">Order Number:</span>
              <span class="font-medium">{orderNumber}</span>
            </div>
            <div class="flex justify-between">
              <span class="text-muted-foreground">Order Date:</span>
              <span class="font-medium">{new Date().toLocaleDateString()}</span>
            </div>
            <div class="flex justify-between">
              <span class="text-muted-foreground">Estimated Delivery:</span>
              <span class="font-medium">{estimatedDelivery}</span>
            </div>
            <Separator />
            <div class="flex justify-between">
              <span class="text-muted-foreground">Payment Method:</span>
              <span class="font-medium">Credit Card</span>
            </div>
            <div class="flex justify-between">
              <span class="text-muted-foreground">Shipping Method:</span>
              <span class="font-medium">Standard Shipping</span>
            </div>
          </div>
        </CardContent>
      </Card>

      <div class="grid gap-6 md:grid-cols-2 mb-8">
        <Card>
          <CardContent class="pt-6">
            <div class="flex items-start space-x-4">
              <div class="rounded-full bg-primary/10 p-2">
                <Package class="h-5 w-5 text-primary" />
              </div>
              <div>
                <h3 class="font-medium">Track Your Order</h3>
                <p class="text-sm text-muted-foreground mt-1">
                  You will receive an email with tracking information once your order ships.
                </p>
                <Button variant="link" class="px-0 mt-2" asChild>
                  <Link href={user ? "/account/orders" : "#"}>View Order Status</Link>
                </Button>
              </div>
            </div>
          </CardContent>
        </Card>

        <Card>
          <CardContent class="pt-6">
            <div class="flex items-start space-x-4">
              <div class="rounded-full bg-primary/10 p-2">
                <BookOpen class="h-5 w-5 text-primary" />
              </div>
              <div>
                <h3 class="font-medium">Need Help?</h3>
                <p class="text-sm text-muted-foreground mt-1">
                  If you have any questions about your order, please contact our customer service.
                </p>
                <Button variant="link" class="px-0 mt-2" asChild>
                  <Link href="/contact">Contact Support</Link>
                </Button>
              </div>
            </div>
          </CardContent>
        </Card>
      </div>

      <div class="text-center">
        <Button asChild>
          <Link href="/books">
            Continue Shopping
            <ArrowRight class="ml-2 h-4 w-4" />
          </Link>
        </Button>
      </div>
    </div>
  )
}
```

Order history

```
"use client";

import { useEffect, useState } from "react";
import { useAuth } from "@/context/auth-context";

type OrderItem = {
  id: string;
  bookId: string;
  title: string;
  price: number;
  quantity: number;
};

type Order = {
  id: string;
  totalAmount: number;
  address: string;
  city: string;
  state: string;
  zipCode: string;
  country: string;
  paymentMethod: string;
  createdAt: string;
  items: OrderItem[];
};

export default function OrdersPage() {
  const { user } = useAuth();
  const [orders, setOrders] = useState<Order[]>([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState<string | null>(null);

  useEffect(() => {
    async function fetchOrders() {
      if (!user) {
        setError("User not logged in");
        setLoading(false);
        return;
      }
      try {
        const res = await fetch(`/api/orders?userId=${user.id}`);
        if (!res.ok) {
          throw new Error("Failed to fetch orders");
        }
        const data = await res.json();
        setOrders(data.orders);
      } catch (err: any) {
        setError(err.message || "Unknown error");
      } finally {
        setLoading(false);
      }
    }
    fetchOrders();
  }, [user]);

  if (loading) return <p>Loading orders...</p>;
  if (error) return <p>Error: {error}</p>;
  if (orders.length === 0) return <p>No orders found.</p>;

  return (
    <div className="container mx-auto p-4 text-foreground bg-background">
      <h1 className="text-2xl font-bold mb-4">Order History</h1>
      <div className="overflow-x-auto">
        {orders.map((order) => (
<div key={order.id} className="mb-8 border border-gray-600 rounded shadow-sm bg-muted">
  <div className="p-4 border-b border-gray-600 bg-background">
              <p><strong>Order ID:</strong> {order.id}</p>
              <p><strong>Date:</strong> {new Date(order.createdAt).toLocaleString()}</p>
              <p><strong>Total Amount:</strong> LKR {order.totalAmount.toFixed(2)}</p>
              <p><strong>Shipping Address:</strong> {order.address}, {order.city}, {order.state} {order.zipCode}, {order.country}</p>
              <p><strong>Payment Method:</strong> {order.paymentMethod}</p>
            </div>
            <table className="min-w-full divide-y divide-gray-600">
              <thead className="bg-background">
                <tr>
              <th className="w-1/2 px-6 py-3 text-left text-xs font-medium text-muted-foreground uppercase tracking-wider">Title</th>
              <th className="w-1/6 px-6 py-3 text-left text-xs font-medium text-muted-foreground uppercase tracking-wider">Price</th>
              <th className="w-1/6 px-6 py-3 text-left text-xs font-medium text-muted-foreground uppercase tracking-wider">Quantity</th>
              <th className="w-1/6 px-6 py-3 text-left text-xs font-medium text-muted-foreground uppercase tracking-wider">Subtotal</th>
                </tr>
              </thead>
              <tbody className="bg-background divide-y divide-border divide-gray-600">
                {order.items.map((item) => (
                  <tr key={item.id}>
                    <td className="w-1/2 px-6 py-4 whitespace-nowrap text-sm text-foreground">{item.title}</td>
                    <td className="w-1/6 px-6 py-4 whitespace-nowrap text-sm text-foreground">LKR {item.price.toFixed(2)}</td>
                    <td className="w-1/6 px-6 py-4 whitespace-nowrap text-sm text-foreground">{item.quantity}</td>
                    <td className="w-1/6 px-6 py-4 whitespace-nowrap text-sm text-foreground">LKR {(item.price * item.quantity).toFixed(2)}</td>
                  </tr>
                ))}
              </tbody>
            </table>
          </div>
        ))}
      </div>
    </div>
  );
}
```

Profile page

```python
"use client"

import { useState, useEffect } from "react"
import { useRouter } from "next/navigation"
import { useAuth } from "@/context/auth-context"
import { Button } from "@/components/ui/button"
import { Input } from "@/components/ui/input"
import { Label } from "@/components/ui/label"
import { useToast } from "@/components/ui/use-toast"

export default function ProfilePage() {
  const { user, isLoading } = useAuth()
  const router = useRouter()
  const { toast } = useToast()

  const [formData, setFormData] = useState({
    username: "",
    email: "",
    address: "",
    city: "",
    state: "",
    zipCode: "",
    country: "",
  })

  useEffect(() => {
    if (user) {
      setFormData({
        username: user.username || "",
        email: user.email || "",
        address: user.address || "",
        city: user.city || "",
        state: user.state || "",
        zipCode: user.zipCode || "",
        country: user.country || "",
      })
    }
  }, [user])

  const handleChange = (e: React.ChangeEvent<HTMLInputElement>) => {
    setFormData({ ...formData, [e.target.name]: e.target.value })
  }

  const handleSubmit = async (e: React.FormEvent) => {
    e.preventDefault()
    try {
      const response = await fetch("/api/auth/profile", {
        method: "PUT",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify(formData),
      })
      if (!response.ok) {
        const error = await response.json()
        throw new Error(error.message || "Failed to update profile")
      }
      toast({ title: "Profile updated successfully", variant: "success" })
      router.refresh()
    } catch (error: any) {
      toast({ title: "Error", description: error.message, variant: "destructive" })
    }
  }

  if (isLoading) return <p>Loading...</p>
  if (!user) return <p>Please login to view your profile.</p>

  return (
    <div className="max-w-xl mx-auto p-4">
      <h1 className="text-2xl font-bold mb-4">Profile</h1>
      <form onSubmit={handleSubmit} className="space-y-4">
        <div>
          <Label htmlFor="username">Username</Label>
          <Input id="username" name="username" value={formData.username} onChange={handleChange} required />
        </div>
        <div>
          <Label htmlFor="email">Email</Label>
          <Input id="email" name="email" type="email" value={formData.email} onChange={handleChange} required />
        </div>
        <div>
          <Label htmlFor="address">Address</Label>
          <Input id="address" name="address" value={formData.address} onChange={handleChange} />
        </div>
        <div>
          <Label htmlFor="city">City</Label>
          <Input id="city" name="city" value={formData.city} onChange={handleChange} />
        </div>
        <div>
          <Label htmlFor="state">State</Label>
          <Input id="state" name="state" value={formData.state} onChange={handleChange} />
        </div>
        <div>
          <Label htmlFor="zipCode">Zip Code</Label>
          <Input id="zipCode" name="zipCode" value={formData.zipCode} onChange={handleChange} />
        </div>
        <div>
          <Label htmlFor="country">Country</Label>
          <Input id="country" name="country" value={formData.country} onChange={handleChange} />
        </div>
        <Button type="submit">Save</Button>
      </form>
    </div>
  )
}
```

10. Summary

This is a comprehensive full-stack e-commerce application designed to provide an intuitive online bookshop experience. Built using React and Next.js for the frontend with TypeScript for type safety, the application leverages the power of Tailwind CSS and shadcn/ui components to deliver a responsive, accessible user interface that works seamlessly across all device sizes.

The backend architecture employs Next.js API routes to create a RESTful API system that communicates with a MySQL database through Prisma ORM, ensuring type-safe database operations and simplified data management. This website implements robust user authentication using JWT (JSON Web Tokens) stored in HTTP-only cookies, with passwords securely hashed using bcrypt.

The application integrates with the Google Books API to provide an extensive catalog of books with detailed information, while maintaining a local database schema that efficiently tracks users, orders, and order items with appropriate relationships and constraints.

Core features include user registration and authentication, comprehensive book browsing with search and filter capabilities, detailed book information pages, a persistent shopping cart implemented using React Context API and localStorage, and a complete checkout process with order management. The database design supports all essential e-commerce functionality with tables for users (storing authentication and address information), orders (tracking purchase details and status), and order items (maintaining book-specific information at the time of purchase).

The application follows modern web development best practices including responsive design, proper error handling, input validation using Zod, and security measures to protect against common vulnerabilities.