Database Final Project Report

Bookstore Management System

Date: 2026-01-22

Contents

1 Information of Team Members and Responsibilities

1.1 Team Members

Name	Student ID	Email
Member 1	ID	email@example.com
Member 2	ID	email@example.com

1.2 Responsibilities

Member	Responsibilities
Member 1	• Database design and ER diagram modeling • Backend API development and business logic implementation • System testing and documentation
Member 2	• Frontend UI/UX design and implementation • Database schema implementation with Prisma • Deployment and system integration

2 Database Scenario

2.1 Background

In the context of the continuous development of e-commerce, traditional physical bookstores are facing the practical need to transition online. This project builds a complete “online bookstore system” using a browser/server (B/S) architecture, aiming to simulate and implement a real online book sales business process. This system is not only a window for displaying book information but also a comprehensive e-commerce platform that integrates user interaction, order transactions, inventory management, and back-end operations.

By constructing such a system, we break the constraints of traditional bookstores in terms of business hours and geographical space, enabling round-the-clock, cross-regional book retail services. The system is divided into two major modules: the front-end user interface and the back-end management interface, both sharing the same underlying PostgreSQL relational database. This design ensures real-time synchronization and strong consistency of data between the front and back ends.

2.2 Target Users

The system serves two key user roles:

End Consumers: The primary source of system traffic. They browse the website as visitors, view homepage recommendations, filter by book categories, or use the search function to find specific books. After registering and logging in, users have personal accounts that allow them to manage persistent shopping carts and view historical orders. These user scenarios involve high-frequency data retrieval operations requiring quick system response and strict inventory consistency to avoid over-selling.

System Administrators: Responsible for platform operation and maintenance with the highest system authority. They configure basic system parameters, manage user accounts, set access permissions, handle daily tasks such as adding/removing books, adjusting prices, viewing user order details, and handling special situations like refunds.

2.3 Project Objectives

- Build a complete online bookstore management system supporting book browsing, searching, shopping cart, and order processing
- Implement role-based access control with separate user and administrator interfaces
- Ensure data consistency and prevent overselling through transaction management and inventory control
- Provide comprehensive logging and auditing capabilities for system operations and inventory changes
- Design a scalable database architecture supporting future business expansion

3 Requirements Analysis

3.1 Functional Requirements

3.1.1 User Management

- **Registration**: Users provide username, password, email, and full name. System verifies username uniqueness and password complexity. Passwords are hashed using BCrypt before storage
- **Login and Authentication**: Supports username/email + password login. Upon successful login, a Token or Session Cookie is issued
- **Profile Management**: Users can update personal information including contact details and default shipping address
- **Order History**: Users can view historical orders and track order status (pending payment, shipped, etc.)

3.1.2 Book Management

- **Homepage Recommendation**: Display books with highest sales or newly listed books
- **Category Navigation**: Provides tree-like or list-like navigation for book categories
- **Advanced Search**: Supports combined queries by title, author, ISBN with case-insensitive fuzzy matching
- **Book Details**: Display comprehensive information including cover image, price, real-time stock status, description, and sales count
- **Admin Book Maintenance**: Administrators can add, edit, delete books, upload cover images, set categories and pricing
- **Status Control**: Quick control of front-end book visibility through status fields (on-shelf/off-shelf)

3.1.3 Order Management

- **Checkout Process**: Confirm delivery information → Select payment method → Confirm order summary
- **Inventory Deduction**: Automatic inventory deduction and sales count increment upon successful order placement
- **Order Inquiry**: Users view order list with status filtering
- **Admin Order Management**: Administrators can intervene in order process, modify status, confirm shipment, process refunds
- **Shipping Management**: Enter and track logistics information

3.1.4 Shopping Cart

- **State Management**: Users can add books, modify quantities, or remove items at any time
- **Persistence**: Cart data stored in database for cross-session and cross-device synchronization
- **Dynamic Calculation**: Real-time calculation of total cart amount
- **Inventory Pre-judgment**: System prompts when quantity exceeds current inventory

3.1.5 Administrative Functions

- **Dashboard**: Data visualization overview including total users, orders, sales amount, and inventory warnings
- **Inventory Management**: All inventory changes (purchase, sales, adjustments) recorded in log table with full traceability
- **Role-Based Access Control**: RBAC model supporting custom roles and permission isolation
- **Operation Audit**: Comprehensive logging of all critical operations including operator, timestamp, and change details

3.2 Non-Functional Requirements

3.2.1 Performance Requirements

- **Index Optimization**: Indexes established on high-frequency query fields (title, author, category, order status)
- **Concurrent Processing**: Database transactions and row-level locking prevent overselling in high-concurrency scenarios
- **Quick Response**: Search results returned in milliseconds
- **Scalability**: Database design reserves expansion space for future features

3.2.2 Security Requirements

- **Data Encryption**: User passwords stored with BCrypt hashing, never in plaintext
- **Access Control**: Backend management interface requires Token verification and permission checks
- **Injection Prevention**: Parameterized queries through Prisma ORM prevent SQL injection attacks
- **Audit Trail**: All sensitive operations logged with operator information and timestamps

3.2.3 Usability Requirements

- **Responsive Design**: Modern UI with consistent visual design using Tailwind CSS and DaisyUI
- **Session Persistence**: Shopping cart data maintained across sessions for logged-in users
- **Form Validation**: Immediate feedback on format errors during registration and login
- **Operational Feedback**: Toast notifications for successful actions, modal warnings for errors

4 ER Diagram Design

4.1 User-Side Entities

4.1.1 USER Entity

Represents registered users of the system:

- **user_id**: Unique user identifier (Primary Key)
- **username**: Unique username for authentication
- **password**: BCrypt hashed password
- **email**: Unique email address
- **full_name, phone, address, city, postal_code**: Contact and shipping information
- **status**: User account status (active/inactive)
- **created_at, updated_at**: Timestamp tracking

4.1.2 CATEGORY Entity

Manages book classifications:

- **category_id**: Category identifier (Primary Key)
- **name**: Category name
- **description**: Category description

4.1.3 BOOK Entity

Represents books sold in the system:

- **book_id**: Book identifier (Primary Key)
- **isbn**: Unique ISBN number
- **title, author, publisher**: Basic book information
- **price**: Book price (Decimal)
- **stock_quantity**: Current inventory level
- **description**: Book description text
- **cover_image**: Cover image URL
- **category_id**: Foreign key to CATEGORY
- **status**: Book status (on-shelf/off-shelf)
- **sales_count**: Cumulative sales counter

4.1.4 SHOPPING_CART Entity

Represents user shopping carts:

- **cart_id**: Shopping cart identifier (Primary Key)
- **user_id**: Foreign key to USER (unique, one-to-one)
- **created_at, updated_at**: Timestamp tracking

4.1.5 CART_ITEM Entity

Individual items within shopping carts:

- **cart_item_id**: Cart item identifier (Primary Key)
- **cart_id**: Foreign key to SHOPPING_CART
- **book_id**: Foreign key to BOOK
- **quantity**: Quantity of books
- **added_at**: Timestamp when item was added

4.1.6 ORDER Entity

Represents user orders:

- **order_id**: Order identifier (Primary Key)
- **user_id**: Foreign key to USER

- **total_amount**: Total order amount (Decimal)
- **status**: Order status (pending/paid/shipped)
- **shipping_address**: Delivery address
- **order_date, updated_at**: Timestamp tracking

4.1.7 ORDER_ITEM Entity

Individual items within orders:

- **order_item_id**: Order item identifier (Primary Key)
- **order_id**: Foreign key to ORDER
- **book_id**: Foreign key to BOOK
- **quantity**: Quantity purchased
- **unit_price**: Price at time of purchase (snapshot)

4.2 Administration-Side Entities

4.2.1 ADMIN Entity

Represents system administrators:

- **admin_id**: Administrator identifier (Primary Key)
- **username, password, email**: Authentication information
- **full_name, phone**: Contact information
- **status**: Administrator status
- **last_login_at**: Last login timestamp
- **created_at, updated_at**: Timestamp tracking

4.2.2 ROLE Entity

Defines administrator roles:

- **role_id**: Role identifier (Primary Key)
- **role_name**: Role display name
- **role_key**: Unique role key
- **description**: Role description
- **status**: Role status

4.2.3 ADMIN_ROLE Entity

Associative entity for many-to-many relationship:

- **id**: Association identifier (Primary Key)
- **admin_id**: Foreign key to ADMIN
- **role_id**: Foreign key to ROLE
- **created_at**: Assignment timestamp

4.2.4 OPERATION_LOG Entity

Records administrative operations:

- **log_id**: Log identifier (Primary Key)
- **admin_id, admin_name**: Administrator information
- **module, action**: Operation module and action type
- **target_type, target_id**: Target object information
- **content**: Operation description
- **status**: Operation status
- **created_at**: Operation timestamp

4.2.5 STOCK_LOG Entity

Records inventory changes:

- **log_id**: Log identifier (Primary Key)
- **book_id**: Foreign key to BOOK
- **change_type**: Type of change (in/out/adjustment)
- **change_quantity**: Quantity changed
- **before_quantity, after_quantity**: Inventory snapshots
- **related_order_id**: Related order if applicable
- **operator_id, operator_type**: Operator information
- **remark**: Change remarks
- **created_at**: Change timestamp

4.3 Relationship Description

4.3.1 User-Side Relationships

- **USER ↔ SHOPPING_CART**: One-to-one relationship. Each user has at most one shopping cart
- **SHOPPING_CART ↔ CART_ITEM**: One-to-many. A cart contains multiple items
- **USER ↔ ORDER**: One-to-many. A user can place multiple orders
- **ORDER ↔ ORDER_ITEM**: One-to-many. An order includes multiple items
- **BOOK ↔ CART_ITEM/ORDER_ITEM**: One-to-many. A book may appear in multiple carts and orders
- **CATEGORY ↔ BOOK**: One-to-many. A category contains multiple books

4.3.2 Administration-Side Relationships

- **ADMIN ↔ ROLE**: Many-to-many through ADMIN_ROLE. An administrator can have multiple roles, and a role can be assigned to multiple administrators
- **ADMIN ↔ OPERATION_LOG**: One-to-many. An administrator generates multiple operation logs
- **BOOK ↔ STOCK_LOG**: One-to-many. A book has multiple inventory change records

4.4 ER Diagram

The system uses a dual ER diagram design separating user-side and administration-side concerns:

User-Side ER Diagram focuses on transaction and user behavior modeling, including entities for users, books, categories, shopping carts, and orders with their relationships.

Administration-Side ER Diagram emphasizes access control, logging, and inventory management, including entities for administrators, roles, operation logs, and stock logs.

This separation reduces system complexity and makes business logic clearer, providing a solid foundation for database implementation.

5 Technical Framework

5.1 System Architecture

The system adopts a **modern full-stack Web architecture** with clear separation of responsibilities:

```
Browser Client
   ↓ HTTP/HTTPS RESTful API
Frontend Application (Vue + Vike)
   ↓ Business Requests
Backend Service (Express)
   ↓ ORM Queries
Database (PostgreSQL)
```

This architecture follows the **Frontend-Backend Separation + RESTful API + SSR Support** pattern. The frontend and backend are decoupled through unified API interfaces, with data exchanged via RESTful APIs in JSON format. This separation of presentation logic from business processing logic improves flexibility and maintainability.

The overall technical framework is divided into four core layers:

- **Presentation Layer (Frontend)**: Handles UI rendering and user interaction
- **Business Logic Layer (Backend)**: Processes core business logic and API services
- **Data Persistence Layer (Database)**: Manages data storage and retrieval
- **Supporting Toolchain**: Build tools, package managers, and deployment infrastructure

5.2 Technology Stack

5.2.1 Frontend

- **Framework**: Vue.js 3 - Component-based UI framework
- **Routing & SSR**: Vike - File-based routing with server-side rendering support
- **Styling**: Tailwind CSS + DaisyUI - Utility-first CSS with component library
- **Build Tool**: Vite - Fast development server and optimized production builds
- **Package Manager**: pnpm - Efficient dependency management
- **HTTP Client**: Axios/Fetch - RESTful API communication

5.2.2 Backend

- **Runtime**: Node.js - JavaScript runtime environment
- **Framework**: Express - Lightweight web application framework
- **Language**: TypeScript - Static type checking for improved code quality
- **API Design**: RESTful architecture - Resource-based API organization
- **Authentication**: Token-based authentication with session management
- **Middleware**: CORS handling, request validation, logging

5.2.3 Database

- **Database System**: PostgreSQL - Relational database management system
- **ORM**: Prisma - Type-safe database client with auto-generated queries
- **Migration Tool**: Prisma Migrate - Schema version control and migration management
- **Query Optimization**: Indexes on high-frequency query fields
- **Transaction Management**: ACID compliance for data consistency

5.2.4 Deployment

- **Hosting Platform**: Vercel - Frontend and SSR deployment
- **Version Control**: Git - Source code management
- **Database Hosting**: PostgreSQL cloud service

- **Environment Management**: Environment variables for configuration

5.3 Database Schema

The database schema is defined using Prisma ORM, which provides type-safe database access and automated migrations. Key schema highlights:

```
// User-side core models
model User {
  id          Int       @id @default(autoincrement())
  username    String    @unique @db.VarChar(50)
  password    String    @db.VarChar(255)
  email       String    @unique @db.VarChar(100)
  fullName    String    @db.VarChar(100)
  phone       String?   @db.VarChar(20)
  address     String?   @db.VarChar(255)
  status      Int       @default(1)
  createdAt   DateTime  @default(now())
  updatedAt   DateTime  @updatedAt

  cart   ShoppingCart?
  orders Order[]
}

model Book {
  id            Int      @id @default(autoincrement())
  isbn          String   @unique @db.VarChar(20)
  title         String   @db.VarChar(200)
  author        String   @db.VarChar(100)
  price         Decimal  @db.Decimal(10, 2)
  stockQuantity Int      @default(0)
  salesCount    Int      @default(0)
  categoryId    Int?
  status        Int      @default(1)

  category   Category?
  cartItems  CartItem[]
  orderItems OrderItem[]
  stockLogs  StockLog[]

  @@index([categoryId])
  @@index([title])
  @@index([author])
}

model Order {
  id              Int      @id @default(autoincrement())
  userId          Int
  totalAmount     Decimal  @db.Decimal(10, 2)
  status          String   @default("pending")
  shippingAddress String   @db.VarChar(255)
  orderDate       DateTime @default(now())

  user  User
  items OrderItem[]

  @@index([userId])
  @@index([status])
}

// Admin-side models
model Admin {
  id          Int       @id @default(autoincrement())
  username    String    @unique
  password    String
  email       String    @unique
  status      Int       @default(1)

  roles         AdminRole[]
  operationLogs OperationLog[]
}

model StockLog {
  id             BigInt   @id @default(autoincrement())
  bookId         Int
  changeType     String   @db.VarChar(20)
  changeQuantity Int
  beforeQuantity Int
  afterQuantity  Int
  operatorType   String   @default("admin")
  createdAt      DateTime @default(now())

  book Book

  @@index([bookId])
  @@index([changeType])
}
```

The schema implements:

- **Referential Integrity**: Foreign key constraints with appropriate cascade rules
- **Performance Optimization**: Strategic indexes on frequently queried fields
- **Data Validation**: Type constraints and default values
- **Audit Trail**: Timestamp fields and logging tables for traceability

6 Functional Design & Software Implementation

6.1 Front-End Store Subsystem

6.1.1 User Account Module

Purpose: Provide user registration, authentication, and profile management capabilities.

Key Features:
- Dual registration/login methods: username+password or email+verification
- BCrypt password hashing for security
- Automatic shopping cart initialization upon registration
- Personal center for profile management
- Order history with status filtering

Implementation:
- Backend `UserService` validates username/email uniqueness
- Password hashing performed before database storage
- Token/Session-based authentication for persistent login state
- User table stores default shipping information (address, city, postal_code)
- Order center aggregates user transactions with status-based filtering

6.1.2 Book Browsing and Retrieval Module

Purpose: Enable users to discover and search for books efficiently.

Key Features:
- Homepage “Hot Recommendations” based on sales count
- Multi-level category navigation system
- Fuzzy search across title, author, publisher, ISBN, and description
- Comprehensive book detail pages with real-time inventory

Implementation:
- Homepage queries books sorted by `salesCount` DESC, limited to top 8
- Category filtering uses JOIN between books and categories tables
- Search implements case-insensitive LIKE queries across multiple fields
- Detail page displays cover image, pricing, stock status, and description
- Real-time inventory check prevents display of out-of-stock items

6.1.3 Shopping Cart Module

Purpose: Provide persistent, cross-device shopping cart functionality.

Key Features:
- Database-backed persistence (not cookie-based)
- Real-time quantity adjustment and item management
- Automatic total price calculation
- Inventory pre-check before checkout

Implementation:
- Cart data stored in `ShoppingCart` and `CartItem` tables
- One-to-one relationship between User and ShoppingCart
- Unique constraint on (cart_id, book_id) prevents duplicates
- Frontend displays real-time total by summing (quantity × price)
- Pre-checkout validation ensures sufficient inventory and active status

6.1.4 Order Processing Module

Purpose: Handle the complete order lifecycle from checkout to fulfillment.

Key Features:
- Multi-step checkout process with address confirmation
- Atomic transaction processing for data consistency
- Automatic inventory deduction and sales tracking
- Order status management (pending/paid/shipped)

Implementation: Order creation uses Prisma Transaction mechanism with four atomic operations:

1. Generate records in `Order` and `OrderItem` tables
2. Deduct `stockQuantity` and increment `salesCount` in books table
3. Create `StockLog` entry with type "sale"
4. Clear settled items from shopping cart

Transaction rollback occurs if inventory insufficient, preventing overselling.

6.2 Back-End Management Subsystem

6.2.1 Book and Inventory Management Module

Purpose: Enable administrators to manage book catalog and inventory.

Key Features:
- Complete CRUD operations for books
- Cover image upload and rich-text descriptions
- Manual inventory adjustments with full audit trail
- Low-stock warning dashboard (threshold: 10 units)

Implementation:
- Book maintenance through RESTful API endpoints
- All inventory changes recorded in `StockLog` table
- Log fields include: change_type (in/out/adjust), quantities, operator info
- Dashboard aggregates books where `stockQuantity < 10`
- Status field controls front-end visibility (on-shelf/off-shelf)

6.2.2 Order Processing Module

Purpose: Provide administrators with order management capabilities.

Key Features:
- Site-wide order list with advanced filtering
- Status updates and shipping information entry
- Order detail view for refund/after-sales handling
- Monthly revenue statistics

Implementation:
- Order queries support filtering by order_id, date, status, user_id
- `updateOrderStatus` API endpoint for status transitions
- Shipping updates modify order status to "shipped"
- Revenue calculation: SUM(total_amount) WHERE status='completed' AND MONTH(order_date)=current_month

6.2.3 User and Permission Management Module

Purpose: Ensure system security through RBAC and comprehensive auditing.

Key Features:
- User account monitoring and freeze capability
- RBAC model with custom roles (Super Admin, Operator, etc.)
- Comprehensive operation logging for all critical actions
- Permission isolation through role-based access control

Implementation:
- User search supports fuzzy matching on username/email
- Account freeze sets user `status` to 0
- Many-to-many Admin-Role relationship via `AdminRole` table
- All sensitive operations logged in `OperationLog` table
- Log captures: admin_id, module, action, target_type, target_id, content, timestamp

6.3 Key Algorithms and Logic

6.3.1 Inventory Deduction with Concurrency Control

```
// Prisma transaction ensures atomicity
await prisma.$transaction(async (tx) => {
  // 1. Lock book row and check inventory
  const book = await tx.book.findUnique({
    where: { id: bookId }
  });

  if (book.stockQuantity < quantity) {
    throw new Error('Insufficient inventory');
  }

  // 2. Update inventory and sales
  await tx.book.update({
    where: { id: bookId },
    data: {
      stockQuantity: { decrement: quantity },
      salesCount: { increment: quantity }
    }
  });

  // 3. Create stock log
  await tx.stockLog.create({
    data: {
      bookId,
      changeType: 'sale',
      changeQuantity: -quantity,
      beforeQuantity: book.stockQuantity,
      afterQuantity: book.stockQuantity - quantity,
      operatorType: 'system'
    }
  });
});
```

6.3.2 Fuzzy Search Implementation

```
// Multi-field fuzzy search with case-insensitive matching
const books = await prisma.book.findMany({
  where: {
    OR: [
      { title: { contains: keyword, mode: 'insensitive' } },
      { author: { contains: keyword, mode: 'insensitive' } },
      { publisher: { contains: keyword, mode: 'insensitive' } },
      { isbn: { contains: keyword } },
      { description: { contains: keyword, mode: 'insensitive' } }
    ],
    status: 1 // Only active books
  },
  include: {
    category: true
  },
  orderBy: {
    salesCount: 'desc' // Prioritize popular books
  }
});
```

6.4 Database Operations

6.4.1 CRUD Operations

- **Create**: User registration, book addition, order creation, cart item insertion - all use Prisma's type-safe `create()` method with automatic validation
- **Read**: Book browsing, order queries, user profile retrieval - implemented with `findMany()`, `findUnique()`, and `findFirst()` with filtering and pagination
- **Update**: Profile updates, inventory adjustments, order status changes - use `update()` and `updateMany()` with WHERE clauses and optimistic locking
- **Delete**: Cart item removal, book deletion - implement soft deletes via status field or hard deletes with CASCADE constraints

6.4.2 Complex Queries

Order with Items and Book Details:

```
const order = await prisma.order.findUnique({
  where: { id: orderId },
  include: {
    user: true,
    items: {
      include: {
        book: {
          include: { category: true }
        }
      }
    }
  }
});
```

Monthly Revenue Aggregation:

```
const revenue = await prisma.order.aggregate({
  where: {
    status: 'completed',
    orderDate: {
      gte: startOfMonth,
      lte: endOfMonth
    }
  },
  _sum: { totalAmount: true }
});
```

Low Stock Alert:

```
const lowStockBooks = await prisma.book.findMany({
  where: {
    stockQuantity: { lt: 10 },
    status: 1
  },
  orderBy: { stockQuantity: 'asc' }
});
```

7 Results Display

7.1 User Interface

7.1.1 Homepage

Screenshot placeholder: Homepage Interface

Figure 1: Homepage Interface

7.1.2 Book Listing Page

Screenshot placeholder: Book Listing Page

Figure 2: Book Listing Page

7.1.3 Shopping Cart Page

Screenshot placeholder: Shopping Cart Interface

Figure 3: Shopping Cart Interface

7.1.4 Order Management Page

Screenshot placeholder: Order Management Interface

Figure 4: Order Management Interface

7.1.5 Admin Dashboard

Screenshot placeholder: Admin Dashboard

Figure 5: Admin Dashboard

7.2 Functional Testing Results

7.2.1 Test Case 1: User Registration and Login

- **Test Description**:
- **Expected Result**:
- **Actual Result**:
- **Status**: ✓ Pass / ✗ Fail

7.2.2 Test Case 2: Book Search and Filter

- **Test Description**:
- **Expected Result**:
- **Actual Result**:
- **Status**: ✓ Pass / ✗ Fail

7.2.3 Test Case 3: Add to Cart and Checkout

- **Test Description**:
- **Expected Result**:
- **Actual Result**:
- **Status**: ✓ Pass / ✗ Fail

7.2.4 Test Case 4: Order Management

- **Test Description**:
- **Expected Result**:
- **Actual Result**:
- **Status**: ✓ Pass / ✗ Fail

7.2.5 Test Case 5: Admin Operations

- **Test Description**:

- **Expected Result**:
- **Actual Result**:
- **Status**: ✓ Pass / ✗ Fail

7.3 Performance Analysis

- **Response Time**:
- **Concurrent Users**:
- **Database Query Performance**:

8 Conclusion

8.1 Summary

8.2 Challenges and Solutions

8.3 Future Improvements

-
-
-

9 References

1.
2.
3.