Southampton Solent University

Object Oriented Design and Development

Module Leader:

Amjad Alam / Chathurika Goonawardane

Software Development

Student ID: [Your ID Number]

Student Name: [Your Full Name]

Submission Date: 4th February 2025

Word Count: 1010

Table of content

Introduction

The development of web applications requires a structured approach to Object-Oriented Design and Development to ensure efficiency, maintainability, and scalability. This report presents the second stage of the **BookNook** project, an online bookstore that enables users to browse and purchase books and literary accessories. Building on the foundation established in the AE1 group project, this individual assignment extends the functionality of **BookNook** by implementing additional features such as a **shopping cart system, order history tracking, and role-based authentication** using **Java, JSP, JDBC, and SQLite**.

This document provides a comprehensive analysis and design of the system, including **use case diagrams, robustness diagrams, and sequence diagrams** to illustrate system interactions. The report also details the implementation process, outlining how the group-developed code was adapted and enhanced to meet the new requirements while adhering to software engineering principles such as **separation of concerns** and **object-oriented principles** like **inheritance, encapsulation, and polymorphism**.

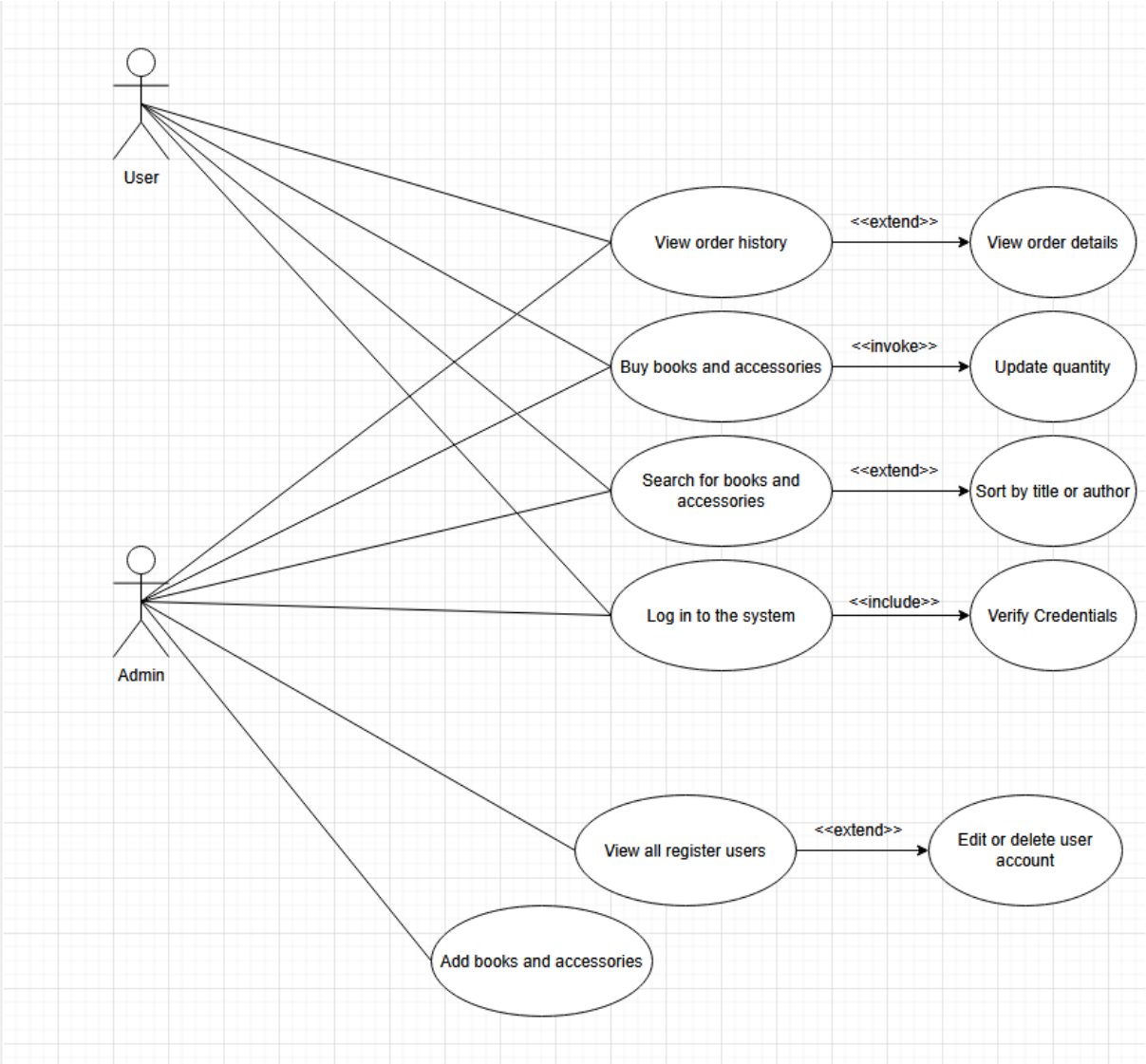
By integrating a **session-based authentication system**, differentiating between **regular users and administrators**, and enabling **inventory management features**, this assignment demonstrates the application of object-oriented principles in a real-world web development scenario.

The following sections cover the **analysis and design of the system**, followed by **implementation details** and a discussion of the **challenges encountered and their resolutions**.

Part 1: Analysis

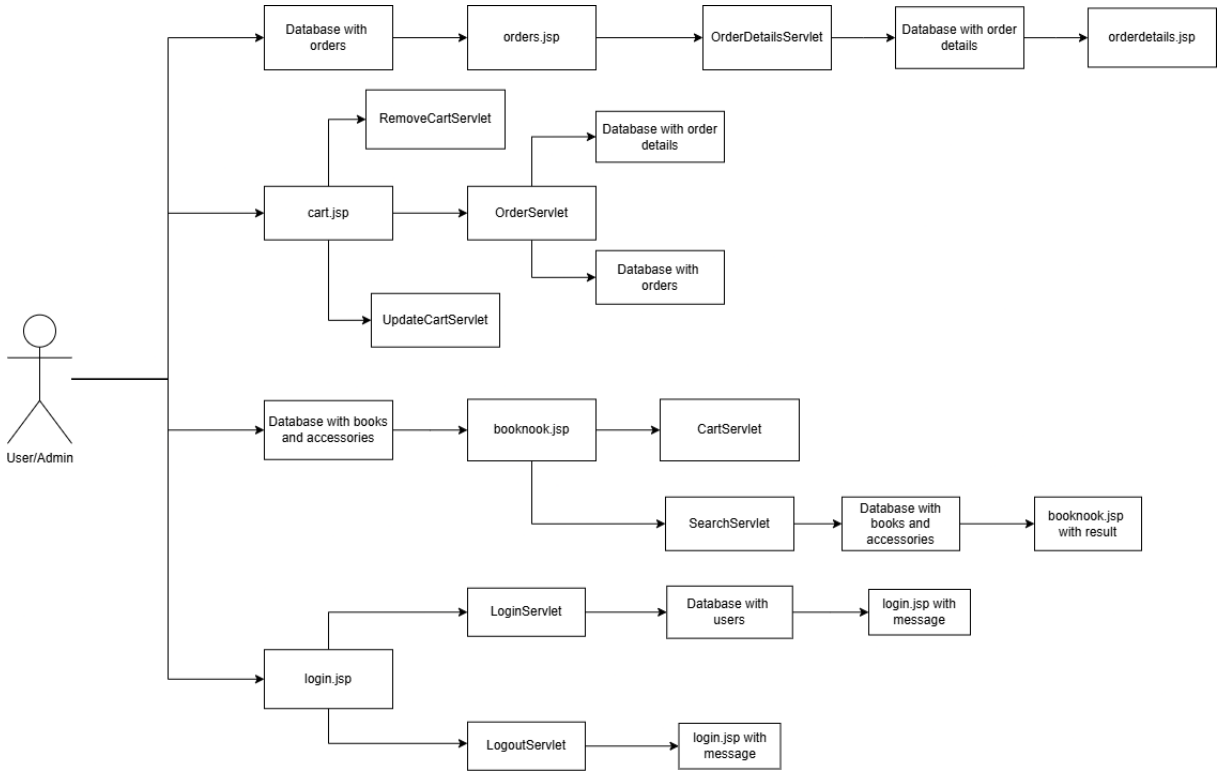
1. Use Cases

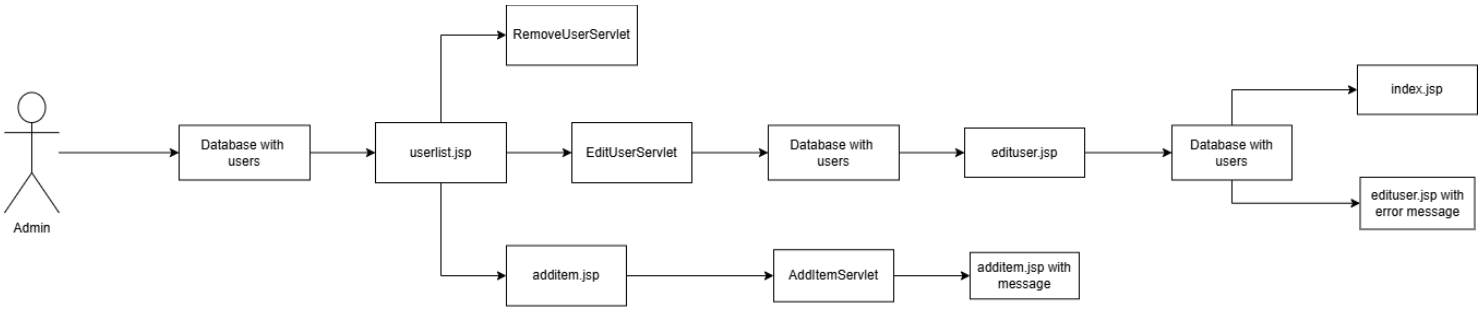
Use case diagrams help in understanding the various functionalities of the system and how users interact with it. Below are the primary actors and their respective use cases:



2. Robustness diagrams

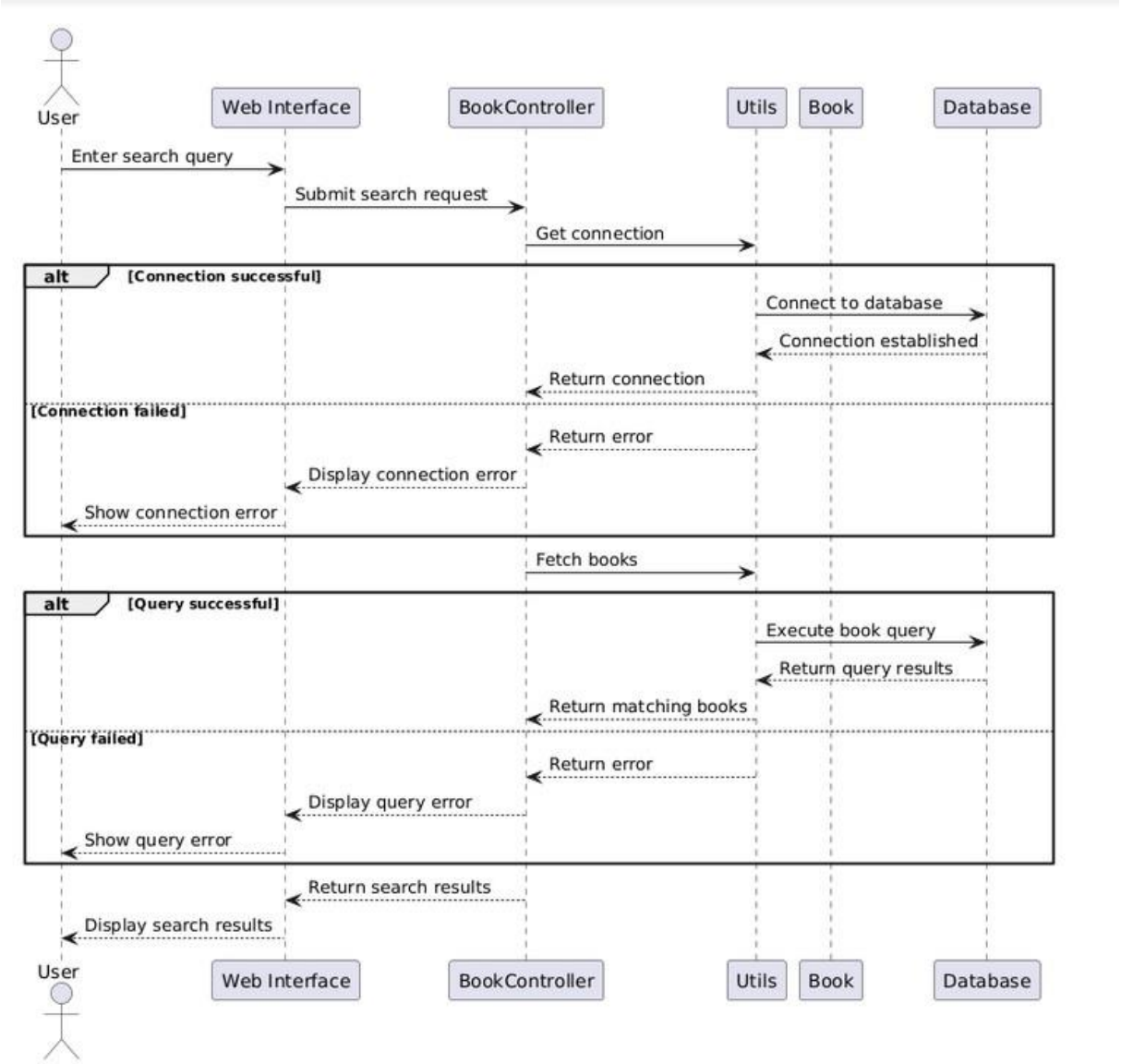
Robustness diagrams illustrate how various objects in the system interact. These diagrams help in designing the logical flow of the **BookNook** application.



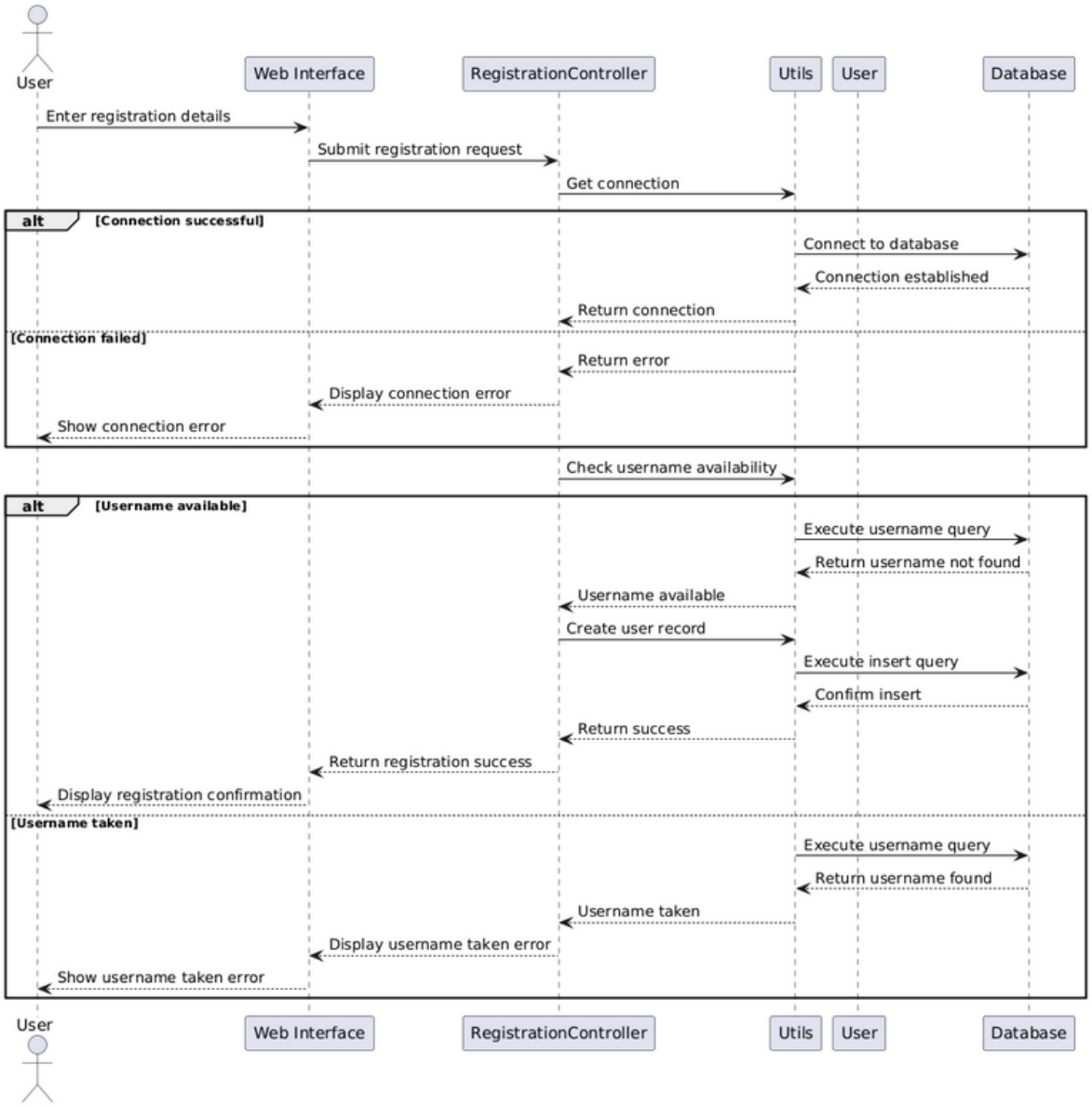


3.Sequence Diagrams

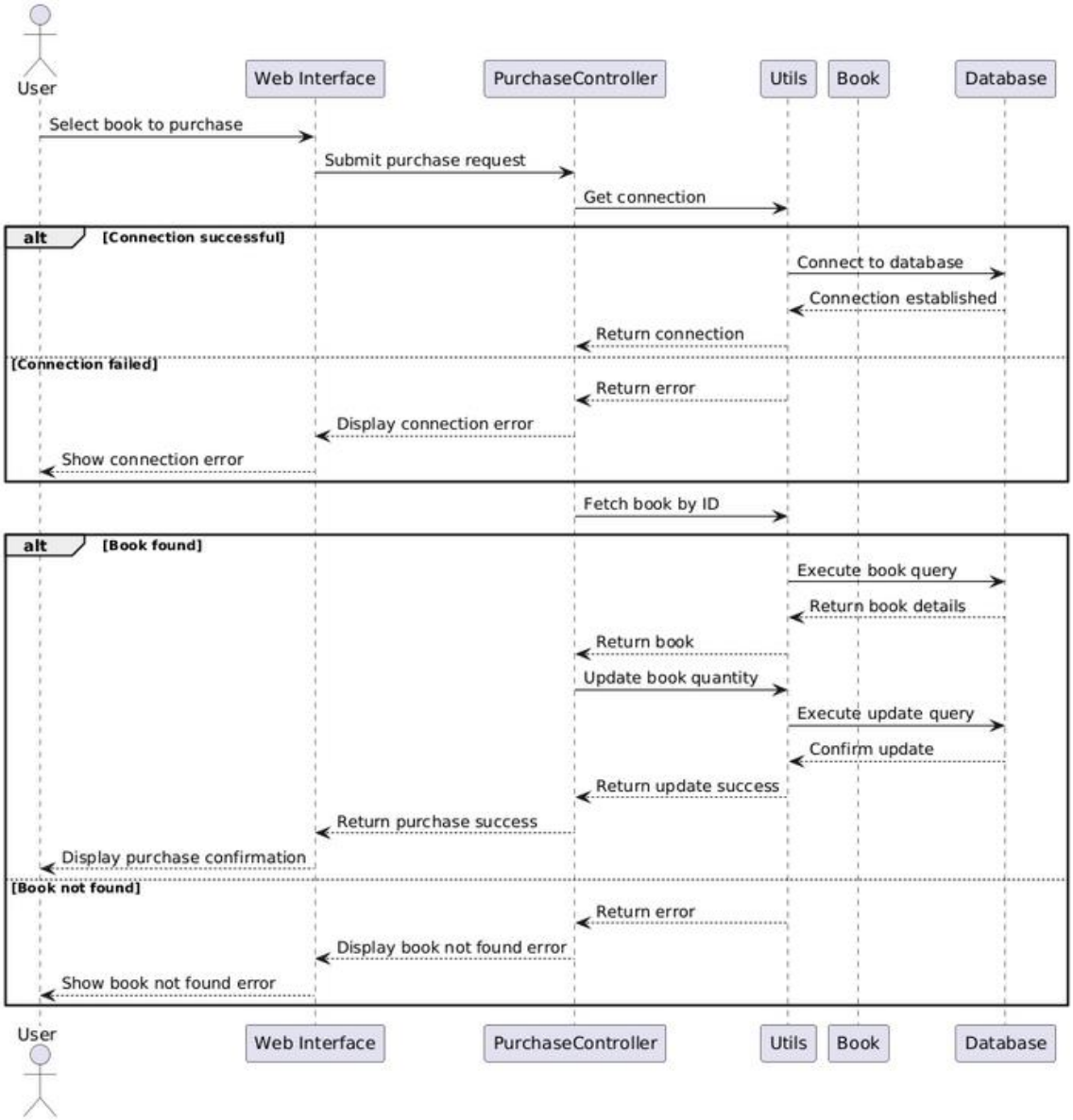
Sequence diagrams provide a step-by-step breakdown of interactions between system components.



Sequence Diagram for searching item



Sequence Diagram for registration



Sequence Diagram purchasing book

Part 2: Implementation

1.Database

First we need to create a database in SQLite for this website. I did this using following code in SQlite:

```
CREATE TABLE Users
(
    UserID INTEGER PRIMARY KEY AUTOINCREMENT,
    Username TEXT NOT NULL,
    Email TEXT NOT NULL,
    Password TEXT NOT NULL,
    Balance DECIMAL(10, 2) NOT NULL DEFAULT 0.00,
    Role TEXT DEFAULT 'user'
);

CREATE TABLE Books
(
    BookID INTEGER PRIMARY KEY AUTOINCREMENT,
    BookTitle TEXT NOT NULL,
    BookAuthor TEXT NOT NULL,
    BookPrice DECIMAL(10, 2) NOT NULL,
    Quantity INTEGER NOT NULL
);

CREATE TABLE Orders
(
    OrderID INTEGER PRIMARY KEY AUTOINCREMENT,
    Username TEXT NOT NULL,
    OrderDate TEXT NOT NULL,
    FOREIGN KEY (Username) REFERENCES Users(Username)
);

CREATE TABLE OrderItems
(
    OrderItemID INTEGER PRIMARY KEY AUTOINCREMENT,
    OrderID INTEGER NOT NULL,
    ItemType TEXT NOT NULL,
    ItemID INTEGER NOT NULL,
    Quantity INTEGER NOT NULL,
    FOREIGN KEY (OrderID) REFERENCES Orders(OrderID)
);

CREATE TABLE "Accessories" (
    AccessoryID INTEGER,
    AccessoryName   TEXT NOT NULL,
    AccessoryPrice  DECIMAL(10,2) NOT NULL,
    Quantity    INTEGER,
    PRIMARY KEY(AccessoryID)
)
```

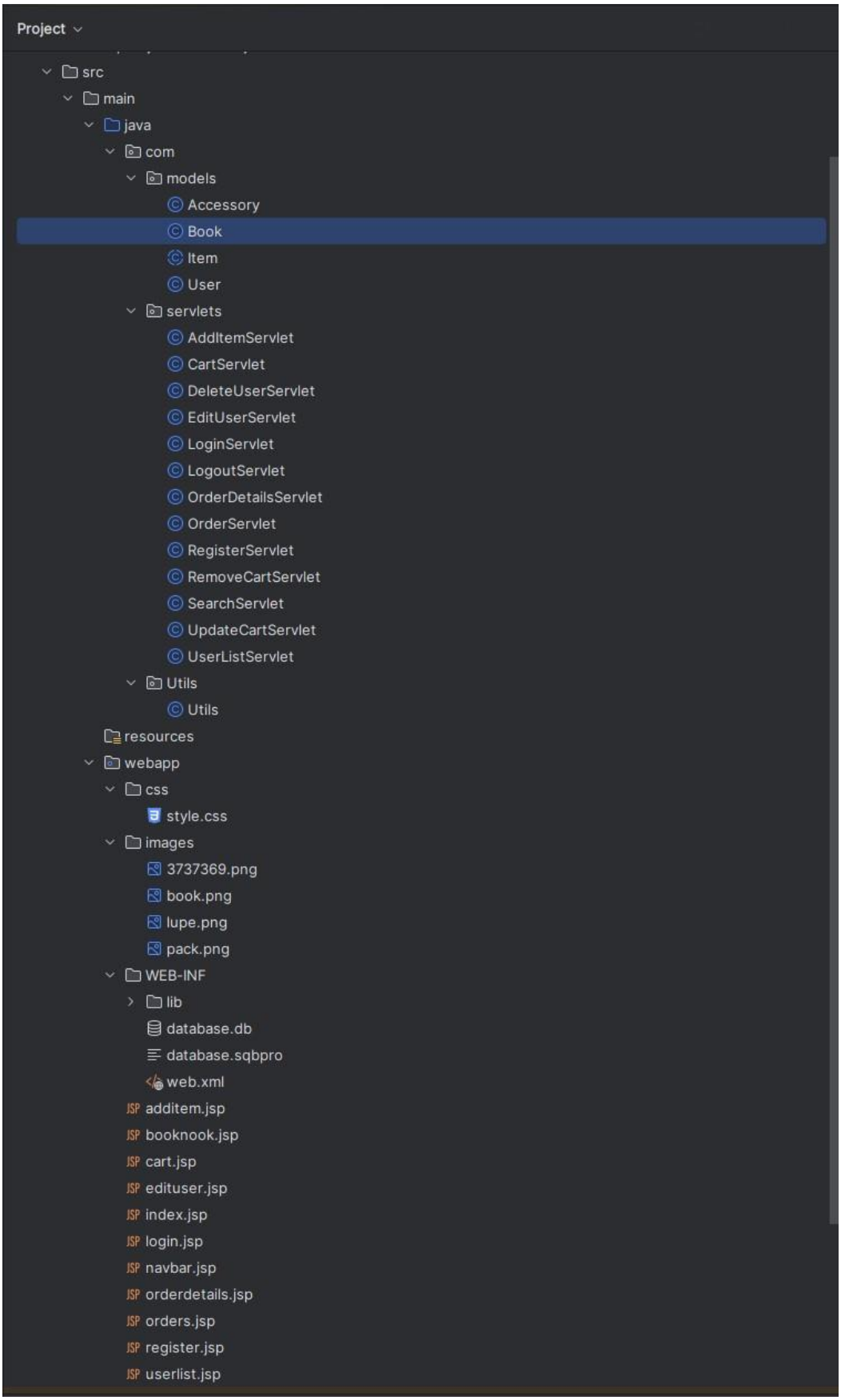
We also need to insert into some values:

```sql
INSERT INTO Accessories (AccessoryName, AccessoryPrice, Quantity) VALUES
('Reading Light', 12.99, 34),
('Book Stand', 25.49, 20),
('Leather Bookmark', 5.99, 100),
('Book Organizer Shelf', 79.99, 10),
('Book Cover Protector', 9.49, 80),
('Magnifying Glass for Reading', 14.99, 25),
('E-Reader Cover', 19.99, 40),
('Pen Holder with Book Design', 29.99, 15),
('Notebook with Inspirational Quotes', 10.99, 60),
('Library-Themed Tote Bag', 18.99, 50);

INSERT INTO Books (BookTitle, BookAuthor, BookPrice, Quantity) VALUES
('The Catcher in the Rye', 'J.D. Salinger', 19.99, 9),
('To Kill a Mockingbird', 'Harper Lee', 14.99, 22),
('1984', 'George Orwell', 12.49, 30),
('Pride and Prejudice', 'Jane Austen', 9.99, 40),
('Moby Dick', 'Herman Melville', 22.99, 10),
('The Great Gatsby', 'F. Scott Fitzgerald', 17.49, 18),
('War and Peace', 'Leo Tolstoy', 39.99, 5),
('Crime and Punishment', 'Fyodor Dostoevsky', 29.99, 8),
('The Hobbit', 'J.R.R. Tolkien', 25.99, 12),
('Brave New World', 'Aldous Huxley', 16.99, 14),
('Fahrenheit 451', 'Ray Bradbury', 13.99, 20),
('Jane Eyre', 'Charlotte Bronte', 14.49, 22),
('Wuthering Heights', 'Emily Bronte', 15.99, 16),
('The Brothers Karamazov', 'Fyodor Dostoevsky', 34.99, 7),
('Les Miserables', 'Victor Hugo', 49.99, 6),
('The Odyssey', 'Homer', 19.99, 11),
('A Tale of Two Cities', 'Charles Dickens', 12.99, 18),
('Sense and Sensibility', 'Jane Austen', 10.49, 24),
('Dracula', 'Bram Stoker', 11.99, 19),
('The Picture of Dorian Gray', 'Oscar Wilde', 14.99, 13);

INSERT INTO Users (Username, Email, Password, Balance, Role) VALUES
('admin', 'admin@gmail.com', 'admin', 965.02, 'admin'),
('admin1', 'admin1@gmail.com', 'admin1', 1000, 'admin'),
('admin2', 'admin2@gmail.com', 'admin2', 1000, 'admin'),
('admin3', 'admin3@gmail.com', 'admin3', 1000, 'admin'),
('tester', 'tester@gmail.com', 'tester', 100, 'user'),
('dev', 'dev@gmail.com', 'dev', 10000, 'user'),
('manager', 'manager@gmail.com', 'manager', 100000, 'user'),
('user', 'user@gmail.com', 'user', 100, 'user');
```

2.Creating website

2.1 Project Struture

The project structure was precisely adapted to the analysis schemes. The code has been divided into files with object models, servlets with unique functions for each servlet, JSP subpages and superficial comments that briefly describe the operation of code fragments. Thanks to this, the code is transparent and every developer in the team will have no problem understanding its content.


Project
src
main
java
com
models
Accessory
Book
Item
User
servlets
AddItemServlet
CartServlet
DeleteUserServlet
EditUserServlet
LoginServlet
LogoutServlet
OrderDetailsServlet
OrderServlet
RegisterServlet
RemoveCartServlet
SearchServlet
UpdateCartServlet
UserListServlet
Utils
Utils
resources
webapp
css
style.css
images
3737369.png
book.png
lupe.png
pack.png
WEB-INF
lib
database.db
database.sqbpro
web.xml
additem.jsp
booknook.jsp
cart.jsp
edituser.jsp
index.jsp
login.jsp
navbar.jsp
orderdetails.jsp
orders.jsp
register.jsp
userlist.jsp

As one can see on this screenshot, the project consists of subfolders with code files that have been divided appropriately.

```java
package com.models;

public class User {
    public String username;
    public String email;
    private String password;
    private String role;


    public String getUsername() { return username; }

    public void setUsername(String username) { this.username = username; }

    public String getEmail() { return email; }

    public void setEmail(String email) { this.email = email; }

    public String getPassword() { return password; }

    public void setPassword(String password) { this.password = password; }

    public String getRole() { return role; }

    public void setRole(String role) { this.role = role; }
}
```

```java
package com.models;

public abstract class Item {
    private String name;
    private double price;
    private int quantity;

    public Item(String name, double price,int quantity) {
        this.name = name;
        this.price = price;
        this.quantity = quantity;
    }

    public String getName() { return name; }

    public double getPrice() { return price; }

    public int getQuantity() { return quantity; }
}
```

```java
package com.models;

public class Book extends Item {

    public int bookID;
    private String Author;

    public Book(int bookID, String bookTitle, String author, double bookPrice, int quantity) {
        super(bookTitle, bookPrice,quantity);
        this.bookID = bookID;
        this.Author = author;
    }

    public String getBookID() { return String.valueOf(bookID); }

    public String getAuthor() { return Author; }

}
```

```java
package com.models;

public class Accessory extends Item {

    public int accessoryID;
    public int quantity;

    public Accessory(int accessoryID, String accessoryName, double accessoryPrice, int quantity) {
        super(accessoryName, accessoryPrice,quantity);
        this.accessoryID = accessoryID;
        this.quantity = quantity;
    }

    public int getAccessoryID() { return accessoryID; }

}
```

The class inheritance technique has also been implemented for clarity and code customization

User

After running this project user should see this menu. There are all the books and accessories and on the navbar there is BookNook which is link to this, register, login and cart which is locked currently (first you need to login).

After scrolling there are also accessories (all of this is booknook.jsp file):

BookNook Register Login

The Picture of Dorian Gray	Oscar Wilde	14.99	13	Add to Cart

Accessory	Name	Price	Quantity	Cart
	Reading Light	12.99	34	Add to Cart
	Book Stand	25.49	20	Add to Cart
	Leather Bookmark	5.99	100	Add to Cart
	Book Organizer Shelf	79.99	10	Add to Cart
	Book Cover Protector	9.49	80	Add to Cart
	Magnifying Glass for Reading	14.99	25	Add to Cart
	E-Reader Cover	19.99	40	Add to Cart
	Pen Holder with Book Design	29.99	15	Add to Cart
	Notebook with Inspirational Quotes	10.99	60	Add to Cart

First interaction is searching for books (This is created using SearchServlet), add to cart is currently locked because first you need to login:

BookNook Register Login

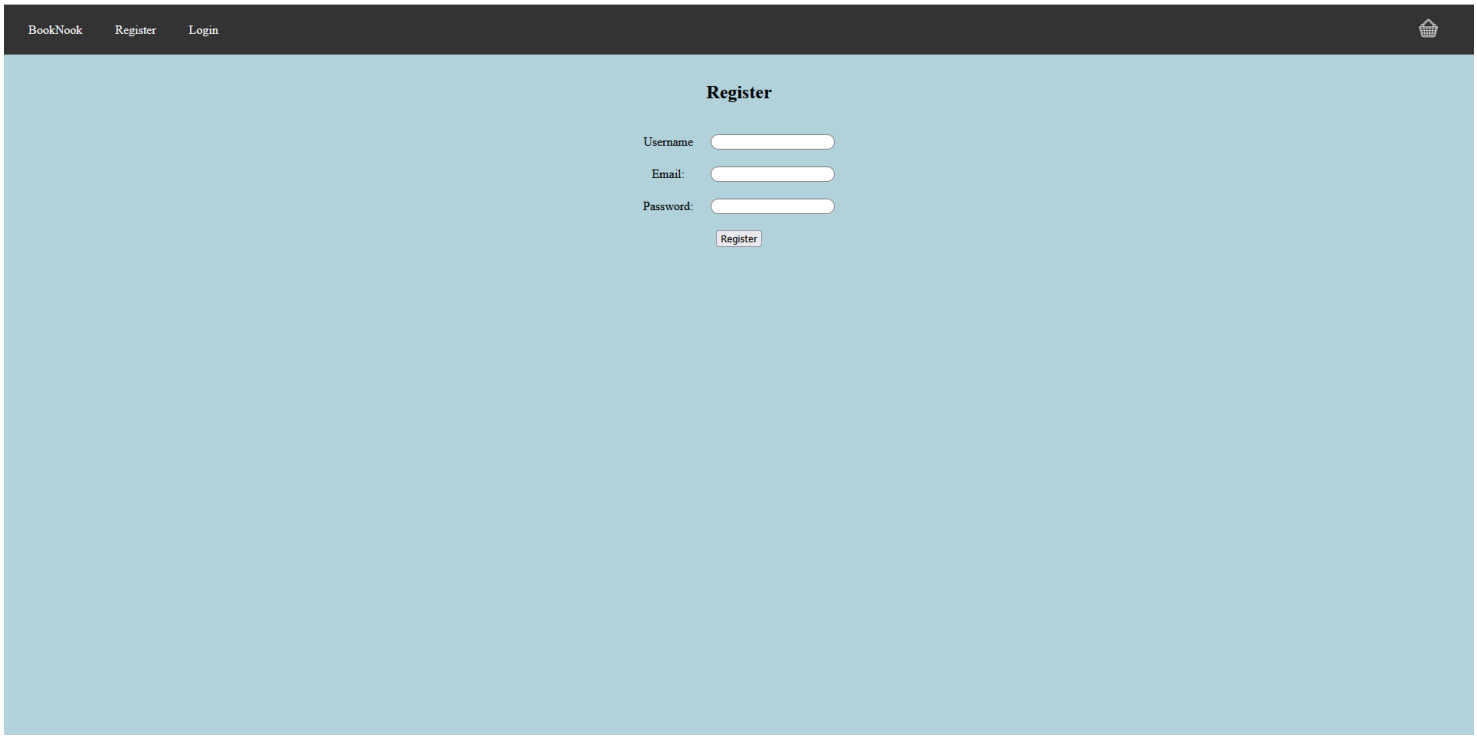
The

Book	Title	Author	Price	Quantity	Cart
	The Catcher in the Rye	J.D. Salinger	19.99	9	Add to Cart
	The Great Gatsby	F. Scott Fitzgerald	17.49	18	Add to Cart
	The Hobbit	J.R.R. Tolkien	25.99	12	Add to Cart
	Wuthering Heights	Emily Bronte	15.99	16	Add to Cart
	The Brothers Karamazov	Fyodor Dostoevsky	34.99	7	Add to Cart
	The Odyssey	Homer	19.99	11	Add to Cart
	The Picture of Dorian Gray	Oscar Wilde	14.99	13	Add to Cart

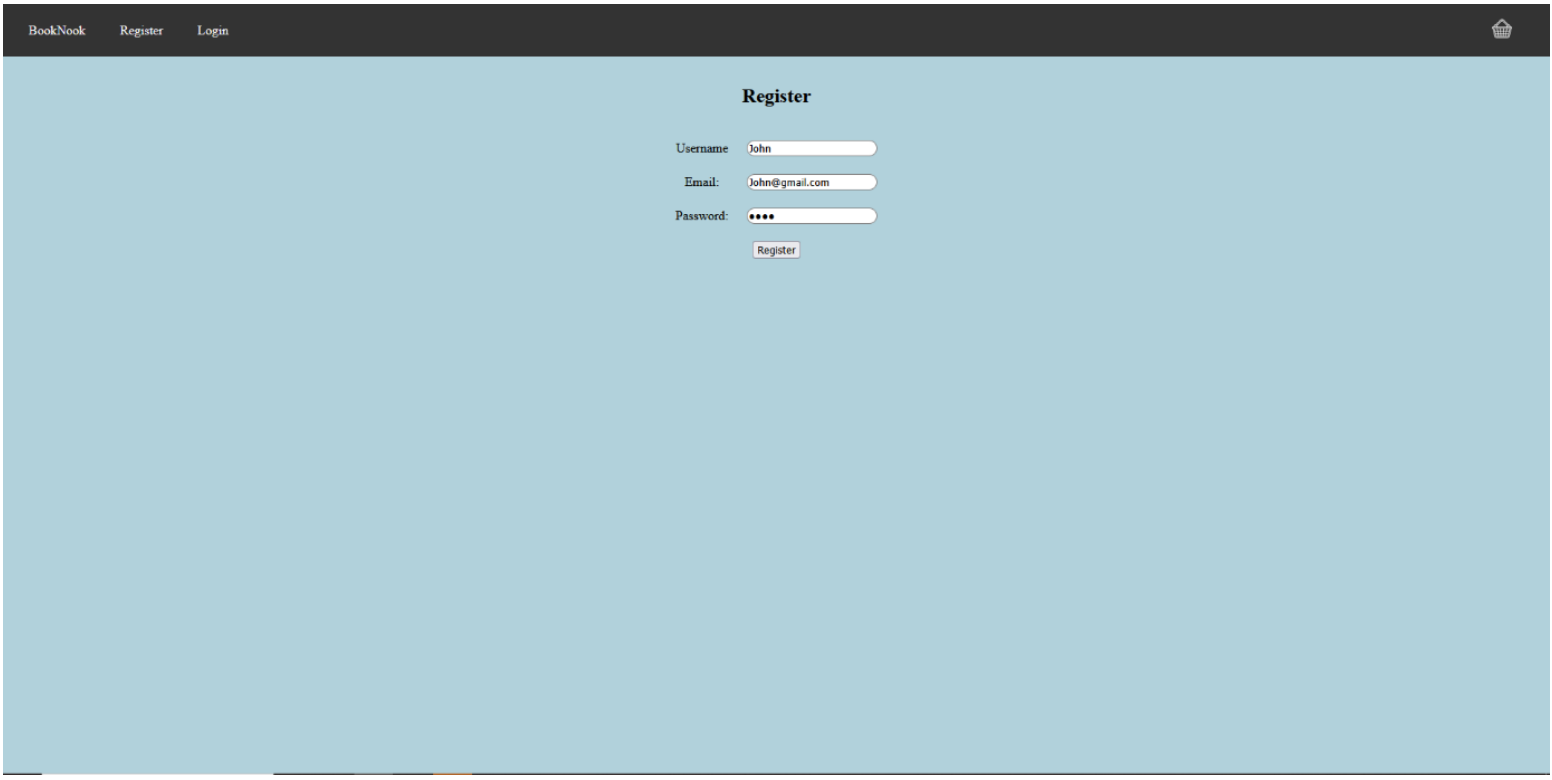
```java
package com.servlets;

import ...

@WebServlet("/SearchServlet")
public class SearchServlet extends HttpServlet {

    public ArrayList<Book> books = new ArrayList<>();

    public SearchServlet() { super(); }
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        //default redirect to itself preventing unwanted get actions performed by user which can lead to malfunctions
        request.getRequestDispatcher( s: "/index.jsp").forward(request, response);
    }
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

        String searchStr = request.getParameter( s: "search-input");
        books = new ArrayList<>();

        try {
            Class.forName( className: "org.sqlite.JDBC");
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }

        String path = getServletContext().getRealPath( s: "/WEB-INF/database.db");
        //search for books basing on the input
        String sql = "SELECT * FROM Books WHERE BookTitle LIKE ? OR BookAuthor LIKE ?";
        try (Connection conn = DriverManager.getConnection( url: "jdbc:sqlite:" + path);
             PreparedStatement pstmt = conn.prepareStatement(sql)) {
            String searchTerm = "%" + searchStr + "%";
            pstmt.setString( parameterIndex: 1, searchTerm);
            pstmt.setString( parameterIndex: 2, searchTerm);

            try (ResultSet rs = pstmt.executeQuery()) {
                // check if we have results
                boolean foundResults = false;
                while (rs.next()) {
                    foundResults = true;

                    Book book = new Book(rs.getInt( columnLabel: "BookID"), rs.getString( columnLabel: "BookTitle")
                    , rs.getString( columnLabel: "BookAuthor"), rs.getDouble( columnLabel: "BookPrice"), rs.getInt( columnLabel: "Quantity"));
                    books.add(book);
                    request.setAttribute( s: "books", books);
                }

                if (!foundResults) {
                    request.setAttribute( s: "books",  o: "noresult");
                }
            }

        } catch (SQLException e) {
            e.printStackTrace();
        }
        request.getRequestDispatcher( s: "index.jsp").forward(request, response);

    }
```

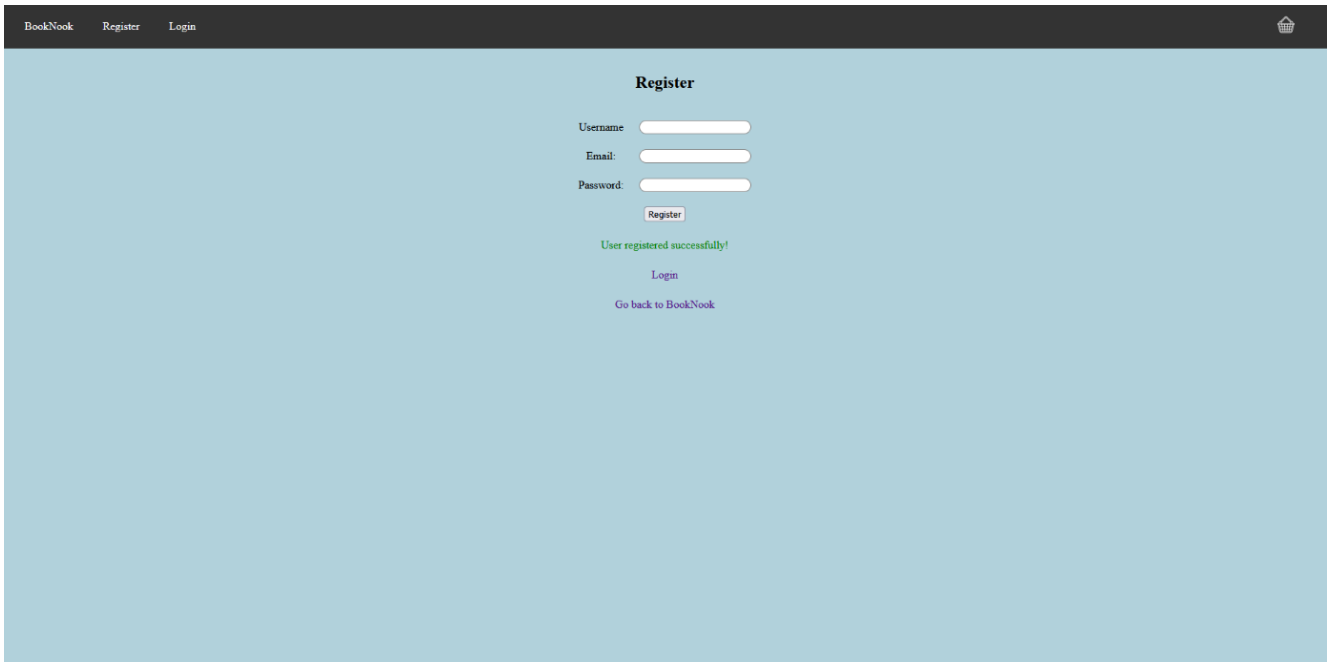
Search Servlet

So you need to register account first, after clicking it this form shows up (register.jsp):



We put in some values, then click register button and we can see that user is registered correctly.

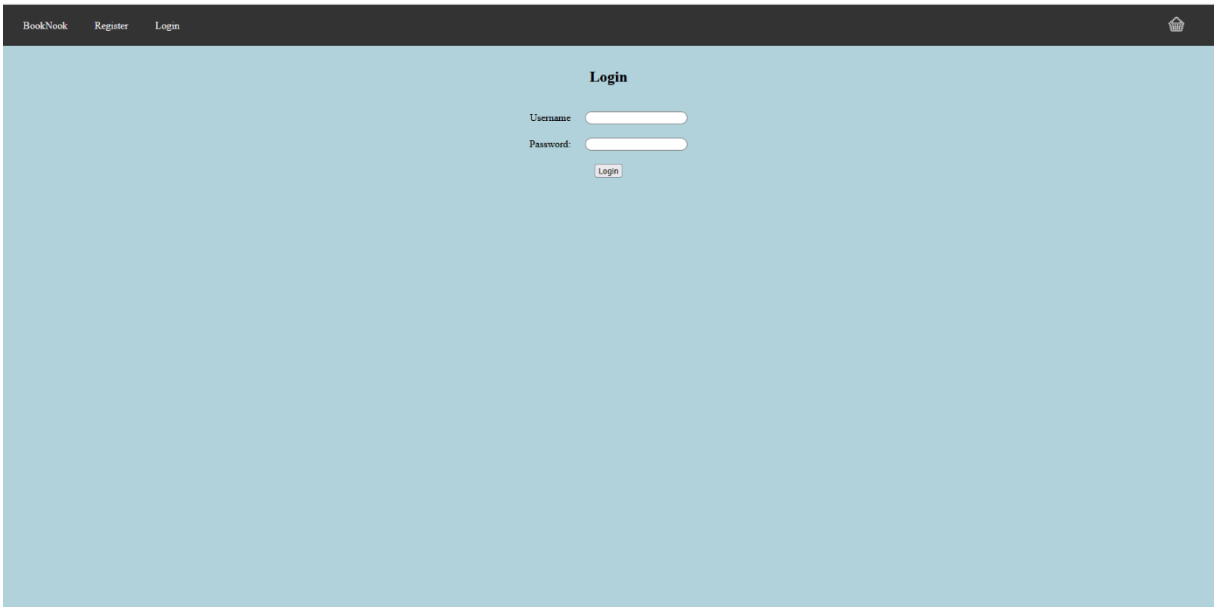


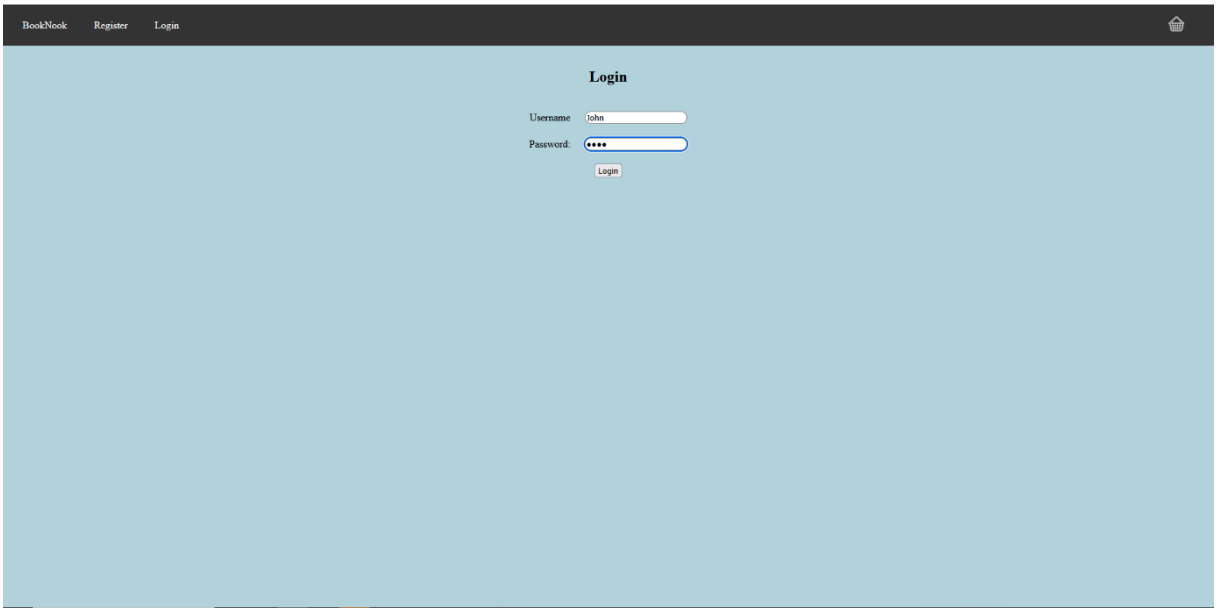


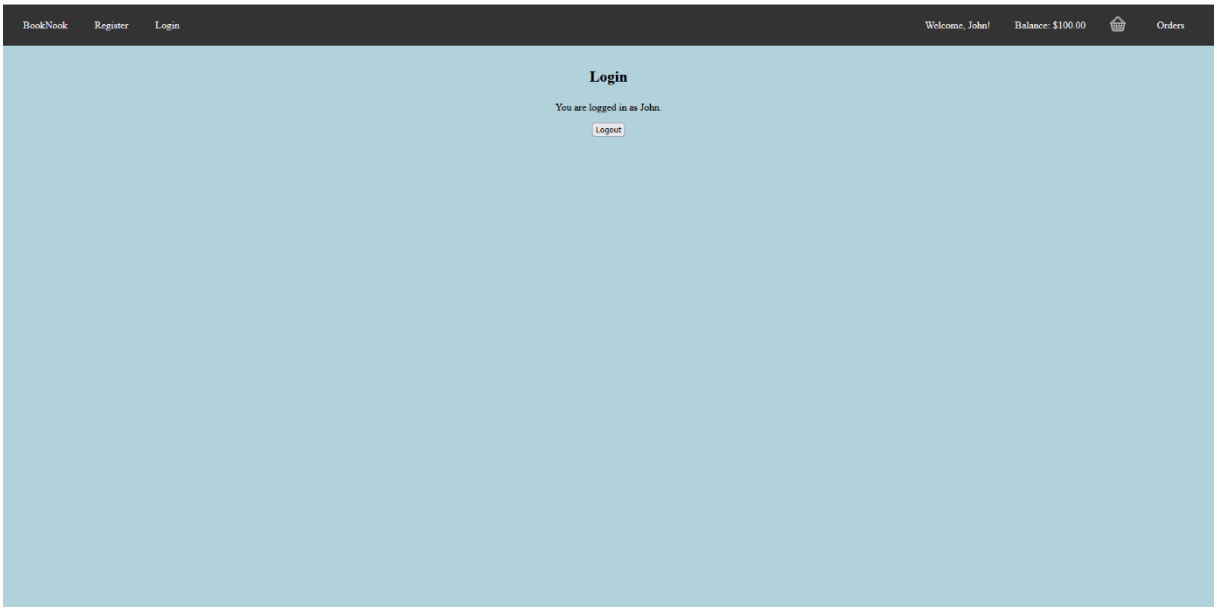

```java
package com.servlets;

import ...

@WebServlet("/RegisterServlet")
public class RegisterServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    public RegisterServlet() { super(); }

    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        //default redirect to itself preventing unwanted get actions performed by user which can lead to malfunctions
        request.getRequestDispatcher( s: "/register.jsp").forward(request, response);
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        String username = request.getParameter( s: "username");
        String email = request.getParameter( s: "email");
        String password = request.getParameter( s: "password");

        try {
            //check if user with the same username or email already exists
            if (userAlreadyExists(username, email, getServletContext())){
                request.setAttribute( s: "message",  o: "Failed to register user in database: User with the same username or email already exists.");
            } else {
                // if doesn't, register user in database
                try {
                    registerUser(username, email, password, getServletContext());
                    request.setAttribute( s: "message",  o: "User registered successfully!");
                } catch (Exception e) {
                    request.setAttribute( s: "message",  o: "Failed to register user in database, please try again.");
                    e.printStackTrace();
                }
            }
        } catch (ClassNotFoundException e) {
            throw new RuntimeException(e);
        } catch (SQLException e) {
            throw new RuntimeException(e);
        }
        //send the message to the page
        request.getRequestDispatcher( s: "index.jsp?page=register").forward(request, response);
    }
```

Register Servlet

So now we can log into this account with login link on the navigation bar.







```java
package com.servlets;

import |...

@WebServlet("/LoginServlet")
public class LoginServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    public LoginServlet() { super(); }

    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        //default redirect to itself preventing unwanted get actions performed by user which can lead to malfunctions
        request.getRequestDispatcher( s: "/login.jsp").forward(request, response);
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        String username = request.getParameter( s: "username");
        String password = request.getParameter( s: "password");
        boolean isError = false;

        try {
            String role = authenticateUser(username, password);
            //authenticate user by checking its role in the database
            if (role != null) {
                double balance = getUserBalance(username);
                HttpSession session = request.getSession();
                session.setAttribute( s: "username", username);
                session.setAttribute( s: "balance", balance);
                session.setAttribute( s: "role", role);
                request.setAttribute( s: "message",  o: "Login successful!");
                //if no role has been found then send the proper message
            } else {
                request.setAttribute( s: "message",  o: "Invalid username or password.");
                isError = true;
            }
        } catch (Exception e) {
            request.setAttribute( s: "message",  o: "An error occurred during login.");
            isError = true;
            e.printStackTrace();
        }
        //send the message to the page
        request.setAttribute( s: "isError", isError);
        request.getRequestDispatcher( s: "index.jsp?page=login").forward(request, response);
    }
```

Login Servlet

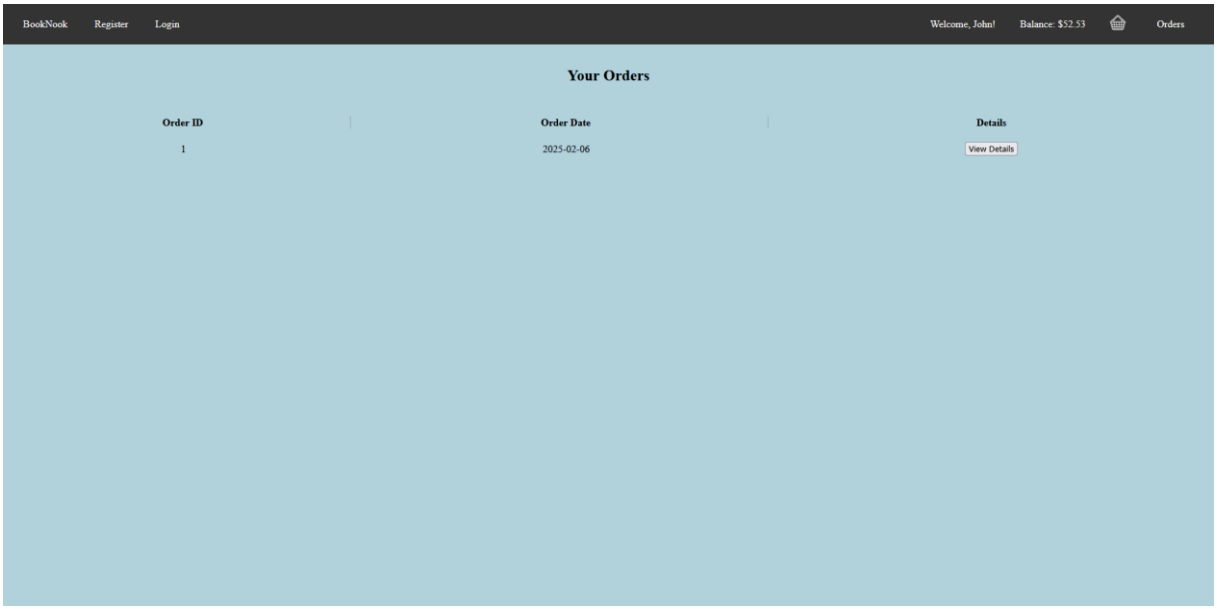
Now when we are logged in we can finally buy some items. So we go back to BookNook and choose items that we want, by cliking on Add to Cart, items are being added to John's cart, and we can see that by clicking on cart icon on the top right.

In cart you can change quantity of an item and also remove it from the cart. When we are ready to place our order we just need to click on the place order button and if everything has been done correctly you should see this window:

```java
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import java.io.IOException;
import java.util.ArrayList;

@WebServlet("/CartServlet")
public class CartServlet extends HttpServlet {

    public CartServlet() {
        super();
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        //default redirect to itself preventing unwanted get actions performed by user which can lead to malfunctions
        request.getRequestDispatcher( s: "/index.jsp?page=cart").forward(request, response);
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        int itemID = request.getParameter( s: "BookID") != null ? Integer.parseInt(request.getParameter( s: "BookID")) : -1;
        int itemID2 = request.getParameter( s: "AccessoryID") != null ? Integer.parseInt(request.getParameter( s: "AccessoryID")) : -1;

        HttpSession session = request.getSession();
        //get a cart from session
        //(array lists which store all the book and accessory id's)
        ArrayList<Integer> bookCart = (ArrayList<Integer>) session.getAttribute( s: "bookCart");
        ArrayList<Integer> accessoryCart = (ArrayList<Integer>) session.getAttribute( s: "accessoriesCart");

        //create new cart for session
        if (bookCart == null) {
            bookCart = new ArrayList<>();
        }
        if (accessoryCart == null) {
            accessoryCart = new ArrayList<>();
        }
        //add id's to carts
        if(itemID != -1) {
            bookCart.add(itemID);
        }
        if(itemID2 != -1) {
            accessoryCart.add(itemID2);
        }
        session.setAttribute( s: "bookCart", bookCart);
        session.setAttribute( s: "accessoriesCart", accessoryCart);

        //send message to the page
        request.getRequestDispatcher( s: "index.jsp?page=booknook").forward(request, response);
    }
}
```

Cart Servlet

Now we can go to the Orders. Here you can see order that you have placed and you also can see order details by clicking view Details button.



```java
package com.servlets;


import ...

@WebServlet("/OrderServlet")
public class OrderServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;
    public OrderServlet() {}
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        //default redirect to itself preventing unwanted get actions performed by user which can lead to malfunctions
        request.getRequestDispatcher( s: "/index.jsp?page=cart").forward(request, response);
    }
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        HttpSession session = request.getSession();

        String username = session.getAttribute( s: "username").toString();

        double totalPrice = (double) session.getAttribute( s: "totalPrice");
        double userBalance = (double) session.getAttribute( s: "balance");
        totalPrice = Double.parseDouble(String.format("%.2f", totalPrice));
        userBalance = Double.parseDouble(String.format("%.2f", userBalance));

        ArrayList<Integer> bookIDs = (ArrayList<Integer>) session.getAttribute( s: "bookCart");
        ArrayList<Integer> accessoryIDs = (ArrayList<Integer>) session.getAttribute( s: "accessoriesCart");

        try {
            Class.forName( className: "org.sqlite.JDBC");
        } catch (ClassNotFoundException e) {
            request.setAttribute( s: "messageOrder",  o: "Transaction failed. Please try again.");
            throw new RuntimeException(e);
        }

        // get ID of a user that is making a transaction
        int userId = -1;
        try {
            userId = getUserId(username, getServletContext());
        } catch (ClassNotFoundException | SQLException e) {
            request.setAttribute( s: "messageOrder",  o: "Transaction failed. Please try again.");
            throw new RuntimeException(e);

        }

        if(userId != -1) {
            if(userBalance < totalPrice) {
                //check if user has enough money to buy items in cart
                request.setAttribute( s: "messageOrder",  o: "Transaction failed. Insufficient funds.");
            } else {
                try {
                    //update all data in database
                    addOrder(username, getServletContext());
                    addOrderDetails(getLastOrderId(username, getServletContext()), bookIDs, accessoryIDs, getServletContext());
                    updateBooksQuantity(bookIDs, getServletContext());
                    updateAccessoriesQuantity(accessoryIDs, getServletContext());
                    updateUserBalance(userId, totalPrice, getServletContext(), session);
                } catch (ClassNotFoundException | SQLException e) {
                    request.setAttribute( s: "messageOrder",  o: "Transaction failed. Please try again.");
```

Order Servlet pt.1

```java
                    throw new RuntimeException(e);
                }
            }
        }

        request.setAttribute( s: "messageOrder",  o: "Transaction completed. Thank you for your order!");
        //remove items from cart by reseting both attributes
        session.setAttribute( s: "bookCart",  o: null);
        session.setAttribute( s: "accessoriesCart",  o: null);
        //send the message to the page
        request.getRequestDispatcher( s: "index.jsp?page=cart").forward(request, response);

    }
    protected int getUserId(String username, ServletContext context) throws ClassNotFoundException, SQLException {
        int id = -1;
        String path = context.getRealPath( s: "/WEB-INF/database.db");
        String sql = "SELECT UserID FROM Users WHERE Username = ?";

        try (Connection conn = DriverManager.getConnection( url: "jdbc:sqlite:" + path);
             PreparedStatement pstmt = conn.prepareStatement(sql)) {

            pstmt.setString( parameterIndex: 1, username);
            try (ResultSet rs = pstmt.executeQuery()) {
                if (rs.next()) {
                    id = rs.getInt( columnLabel: "UserID");
                }
            }

        } catch (SQLException e) {
            e.printStackTrace();
        }
        return id;
    }
    protected void addOrder(String username, ServletContext context) throws ClassNotFoundException, SQLException {
        String path = context.getRealPath( s: "/WEB-INF/database.db");
        String sql = "INSERT INTO Orders (Username, OrderDate) VALUES (?, ?);";
        String currentDate = LocalDate.now().format(DateTimeFormatter.ofPattern("yyyy-MM-dd"));

        try (Connection conn = DriverManager.getConnection( url: "jdbc:sqlite:" + path);
             PreparedStatement pstmt = conn.prepareStatement(sql)) {

            pstmt.setString( parameterIndex: 1, username);
            pstmt.setString( parameterIndex: 2, currentDate);

            pstmt.executeUpdate();
        }
    }
```

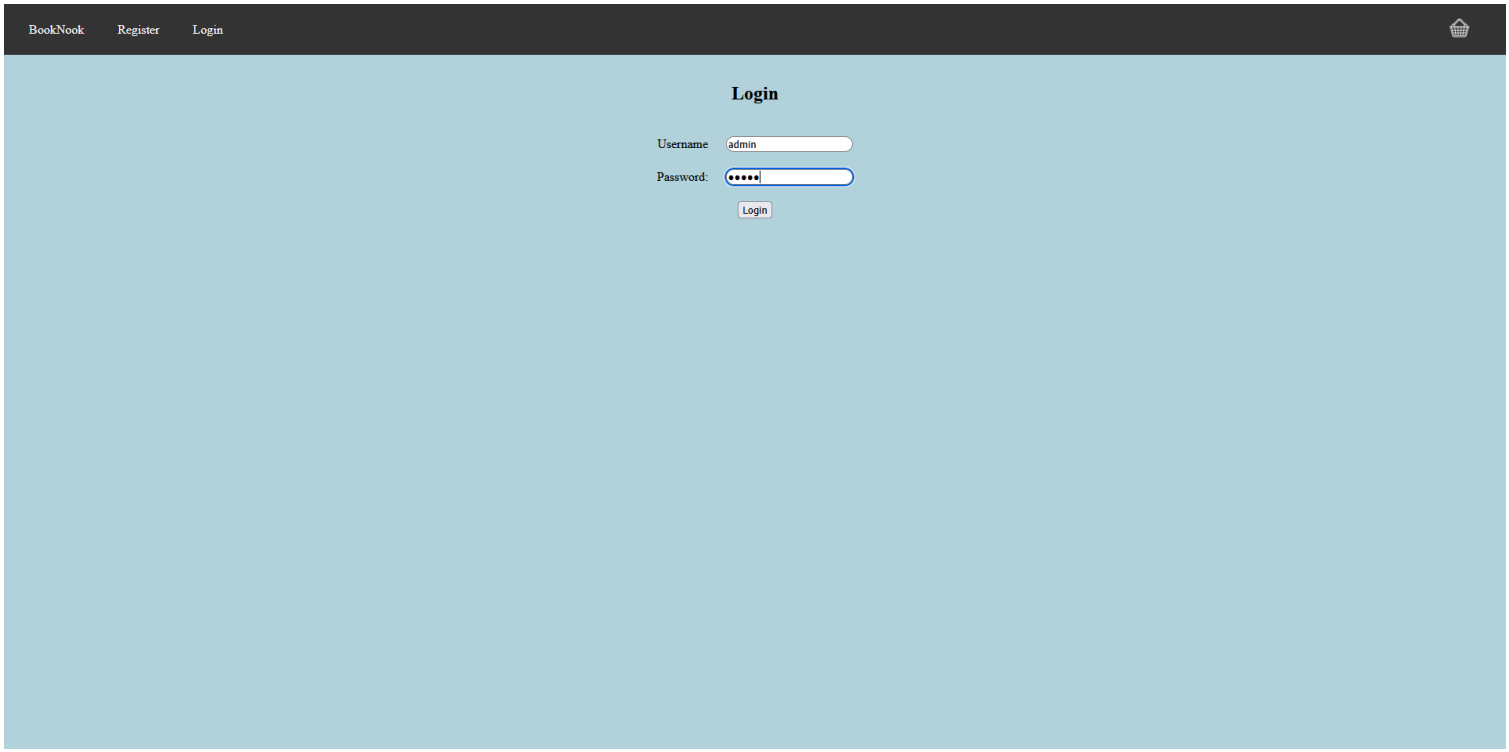
Order Servlet pt.2

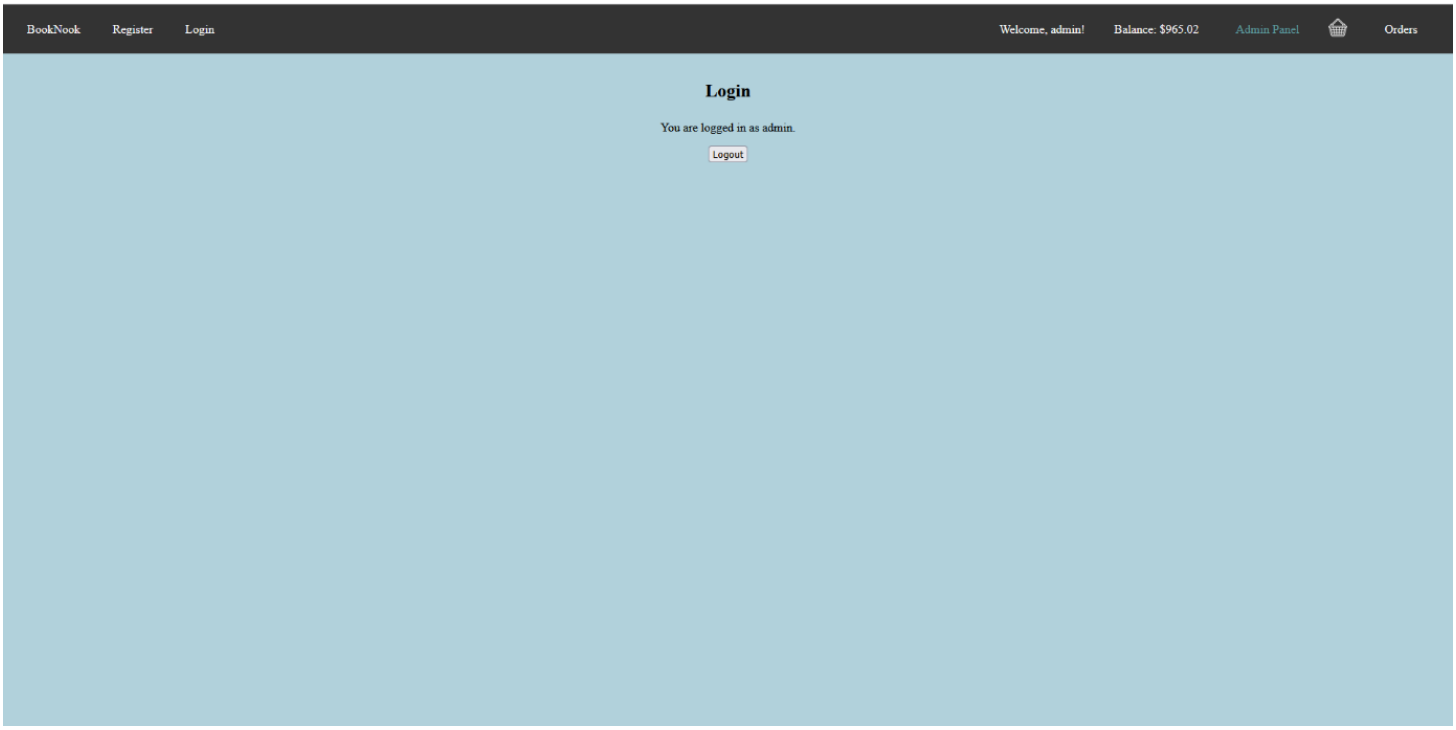
```java
package com.servlets;

import ...

@WebServlet("/OrderDetailsServlet")
public class OrderDetailsServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        int orderId = Integer.parseInt(request.getParameter( s: "orderId"));
        ArrayList<Map<String, Object>> orderItems = new ArrayList<>();
        //execute sql query that get all order details from database
        try {
            Class.forName( className: "org.sqlite.JDBC");
            String path = getServletContext().getRealPath( s: "/WEB-INF/database.db");
            try (Connection conn = DriverManager.getConnection( url: "jdbc:sqlite:" + path)) {
                String sql = "SELECT oi.ItemType, oi.Quantity, " +
                        "CASE WHEN oi.ItemType = 'book' THEN b.BookTitle " +
                        "WHEN oi.ItemType = 'accessory' THEN a.AccessoryName END AS ItemName, " +
                        "CASE WHEN oi.ItemType = 'book' THEN b.BookPrice " +
                        "WHEN oi.ItemType = 'accessory' THEN a.AccessoryPrice END AS ItemPrice " +
                        "FROM OrderItems oi " +
                        "LEFT JOIN Books b ON oi.ItemID = b.BookID AND oi.ItemType = 'book' " +
                        "LEFT JOIN Accessories a ON oi.ItemID = a.AccessoryID AND oi.ItemType = 'accessory' " +
                        "WHERE oi.OrderID = ?";
                try (PreparedStatement pstmt = conn.prepareStatement(sql)) {
                    pstmt.setInt( parameterIndex: 1, orderId);
                    try (ResultSet rs = pstmt.executeQuery()) {
                        while (rs.next()) {
                            Map<String, Object> item = new HashMap<>();
                            item.put("ItemType", rs.getString( columnLabel: "ItemType"));
                            item.put("Quantity", rs.getInt( columnLabel: "Quantity"));
                            item.put("ItemName", rs.getString( columnLabel: "ItemName"));
                            item.put("ItemPrice", rs.getDouble( columnLabel: "ItemPrice"));
                            orderItems.add(item);
                        }
                    }
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        //send all data to the page
        request.setAttribute( s: "orderItems", orderItems);
        request.getRequestDispatcher( s: "orderdetails.jsp").forward(request, response);
    }
}
```

Order Details Servlet

This is all the functionality for user now let us logout and log in as a admin.

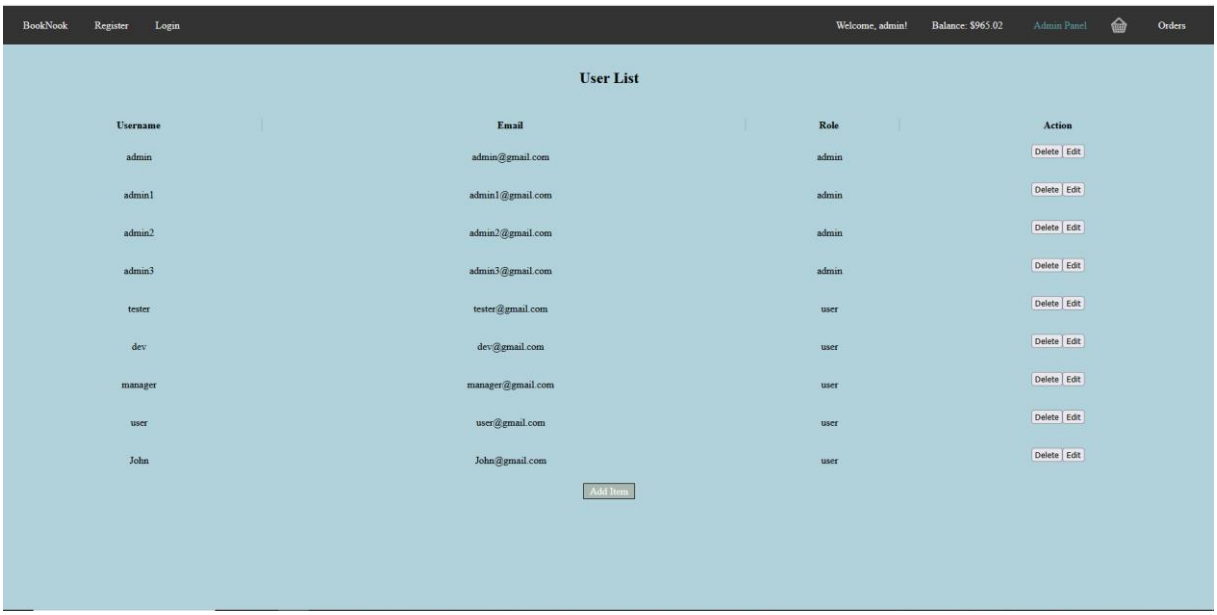
Admin



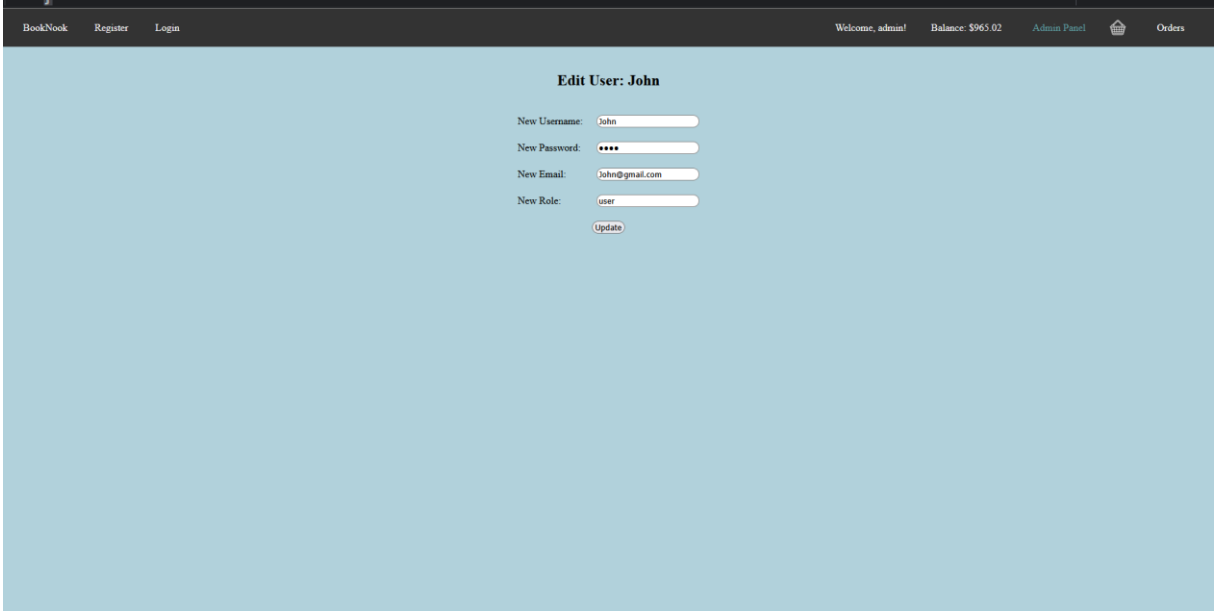


After logging as Admin, a new option labeled *Admin Panel* becomes visible on the navigation bar.

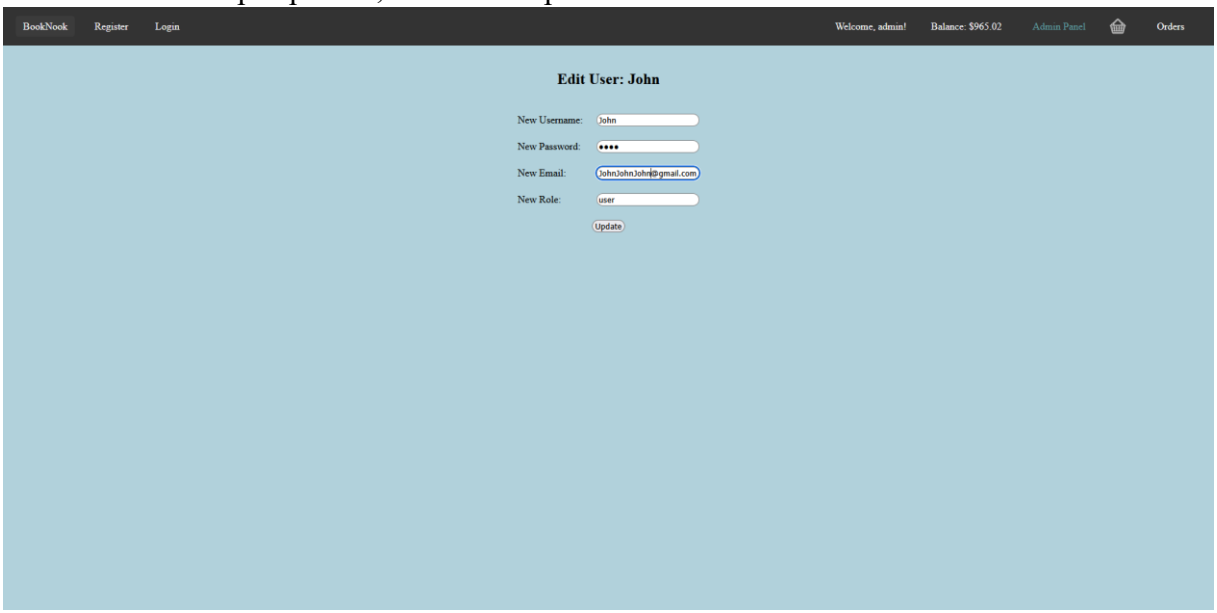
In Admin Panel, administrators can access the *User List* and the *Add Item* feature. Let us begin by exploring the User List. This tab shows all registered users and provides options to edit or delete users.

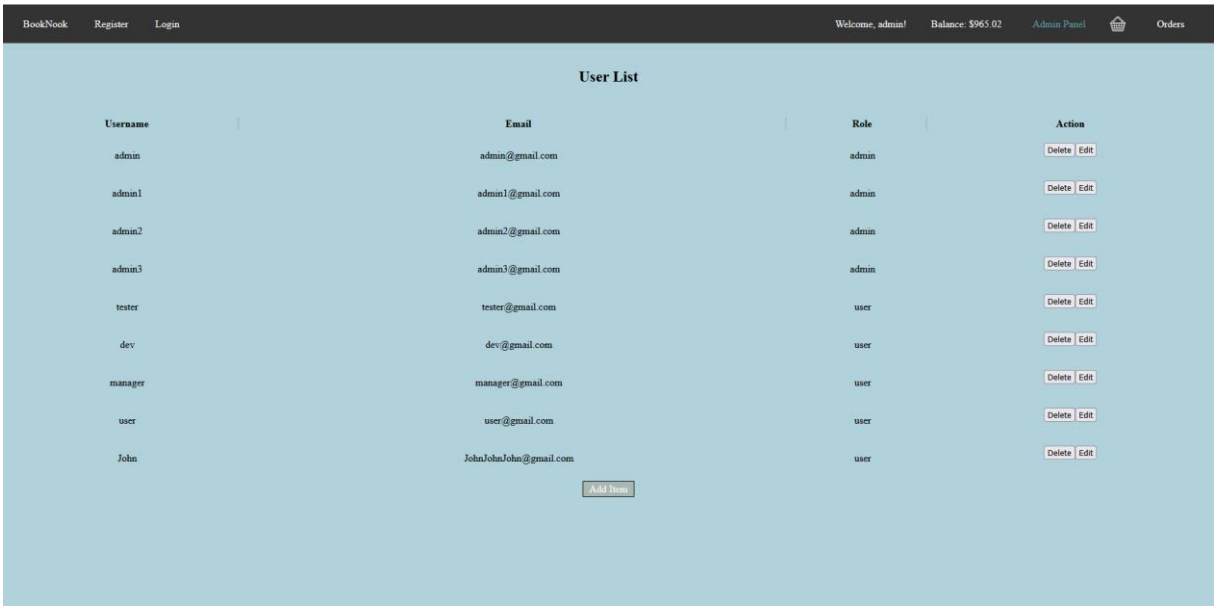


```java
@WebServlet("/UserListServlet")
public class UserListServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;  no usages
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        HttpSession session = request.getSession( b: false);
        //checking if the user requesting is admin
        if (session != null && "admin".equals(session.getAttribute( s: "role"))) {
            try {
                List<User> users = getUsers();
                request.setAttribute( s: "users", users);
                //send all data to the page
                request.getRequestDispatcher( s: "userlist.jsp").forward(request, response);
            } catch (Exception e) {
                e.printStackTrace();
                response.sendError(HttpServletResponse.SC_INTERNAL_SERVER_ERROR,  s: "An error occurred while fetching the user list.");
            }
        } else {
            response.sendError(HttpServletResponse.SC_FORBIDDEN,  s: "Access denied.");
        }
    }
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        doGet(request, response);
    }
    private List<User> getUsers() throws Exception {  1 usage
        List<User> users = new ArrayList<>();
        Class.forName( className: "org.sqlite.JDBC");
        String path = getServletContext().getRealPath( s: "/WEB-INF/database.db");
        String sql = "SELECT Username, Email, Role FROM Users";
        //execute query that gets username, email and role of all users in database
        try (Connection conn = DriverManager.getConnection( url: "jdbc:sqlite:" + path);
             PreparedStatement pstmt = conn.prepareStatement(sql);
             ResultSet rs = pstmt.executeQuery()) {
            while (rs.next()) {
                User user = new User();
                user.setUsername(rs.getString( columnLabel: "Username"));
                user.setEmail(rs.getString( columnLabel: "Email"));
                user.setRole(rs.getString( columnLabel: "Role"));
                users.add(user);
            }
        }
        return users;
    }
```

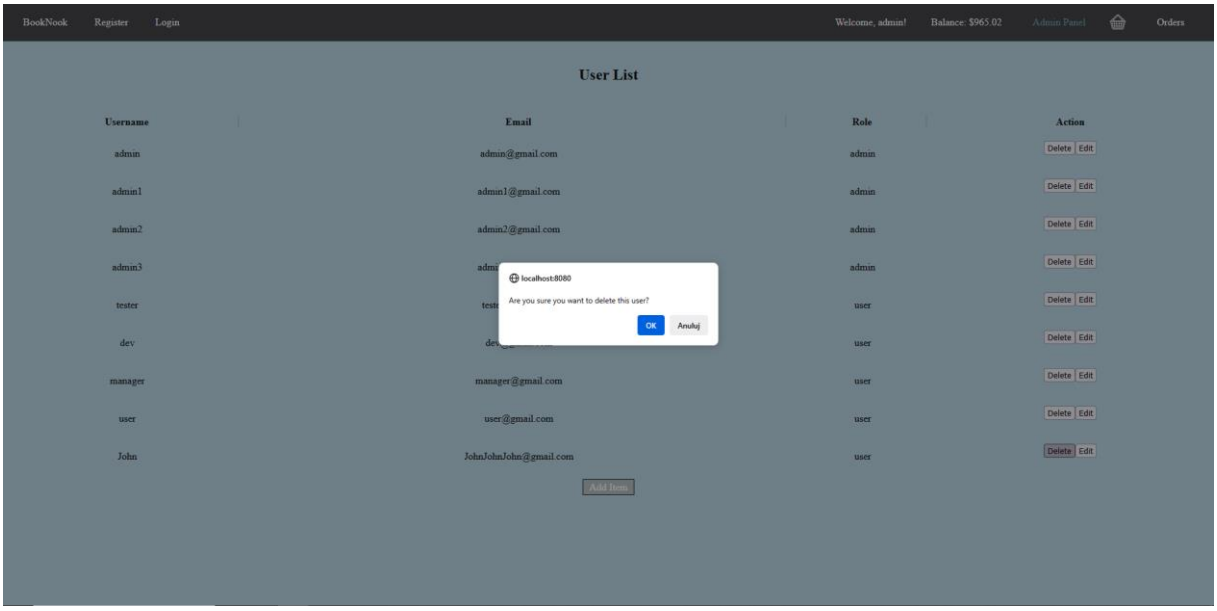


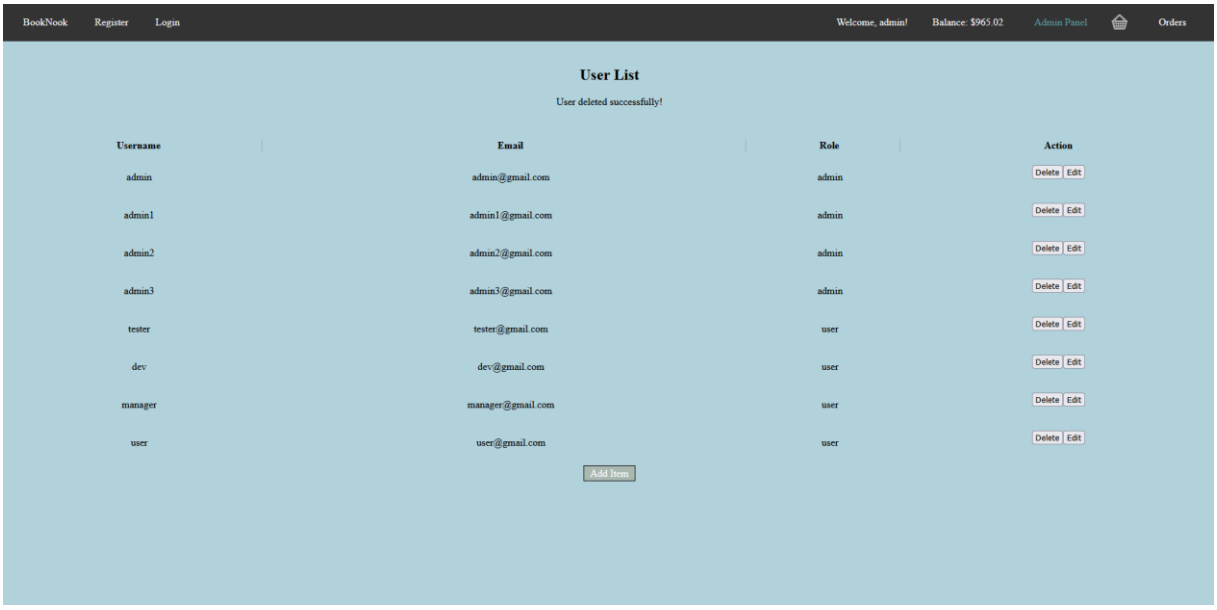
The “Edit User” interface allows admin to modify user account details. For demonstration purposes, we will update John's email address:





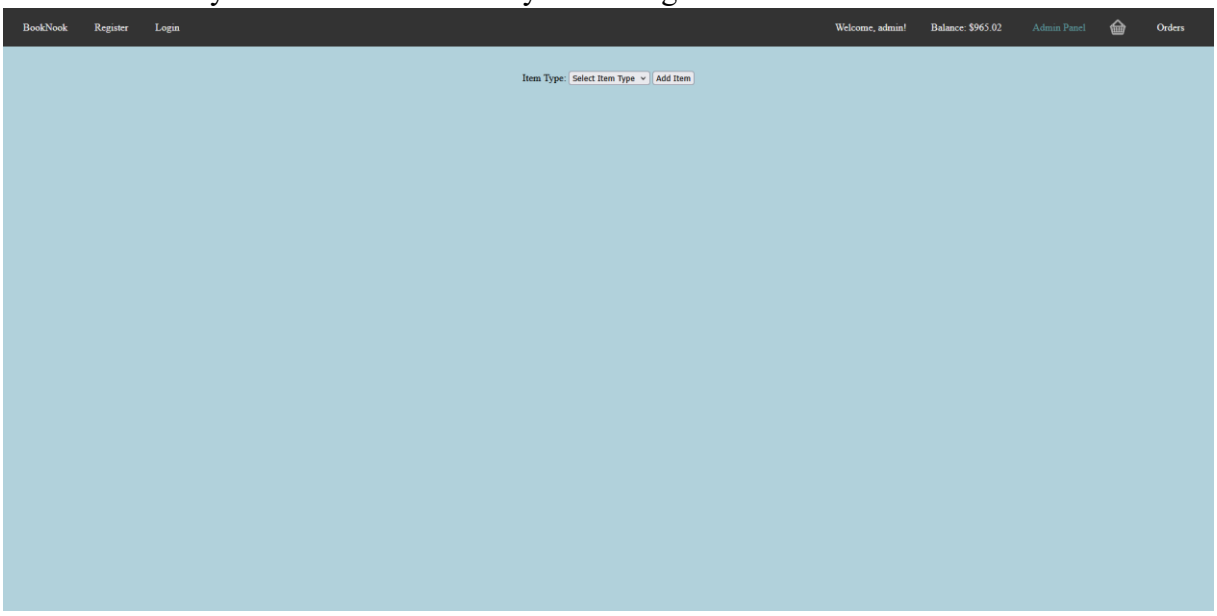
After clicking update and then returning to admin panel we can see that John email has been successfully updated. Now let us proceed to delete his account:



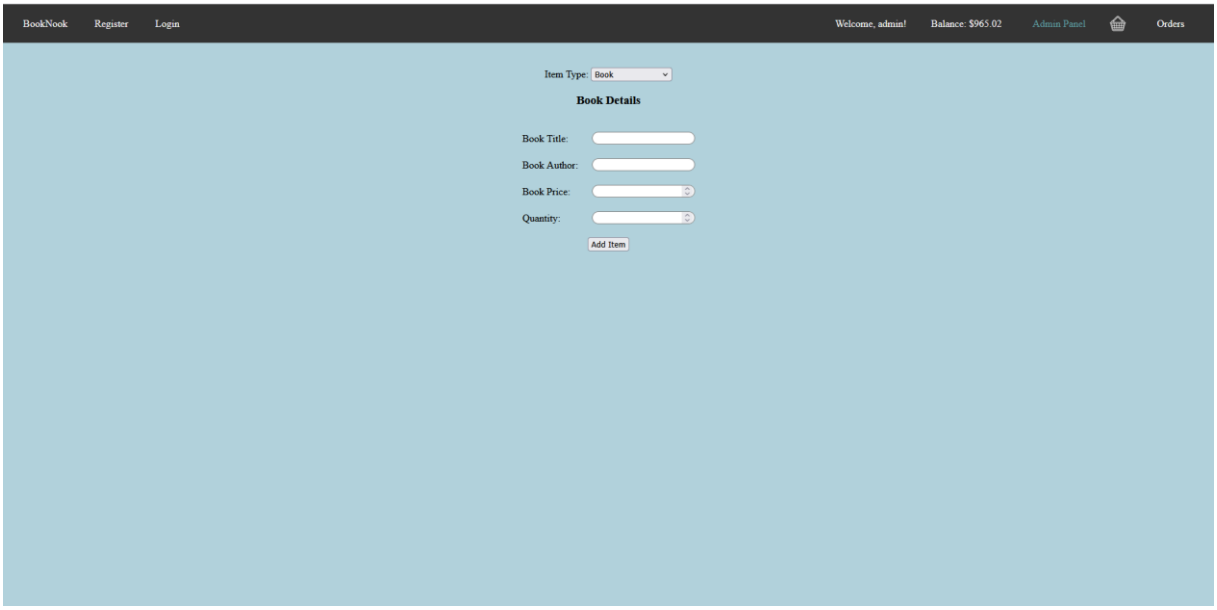


By clicking delete button and confirming the action, John's account is removed from the User List and database.

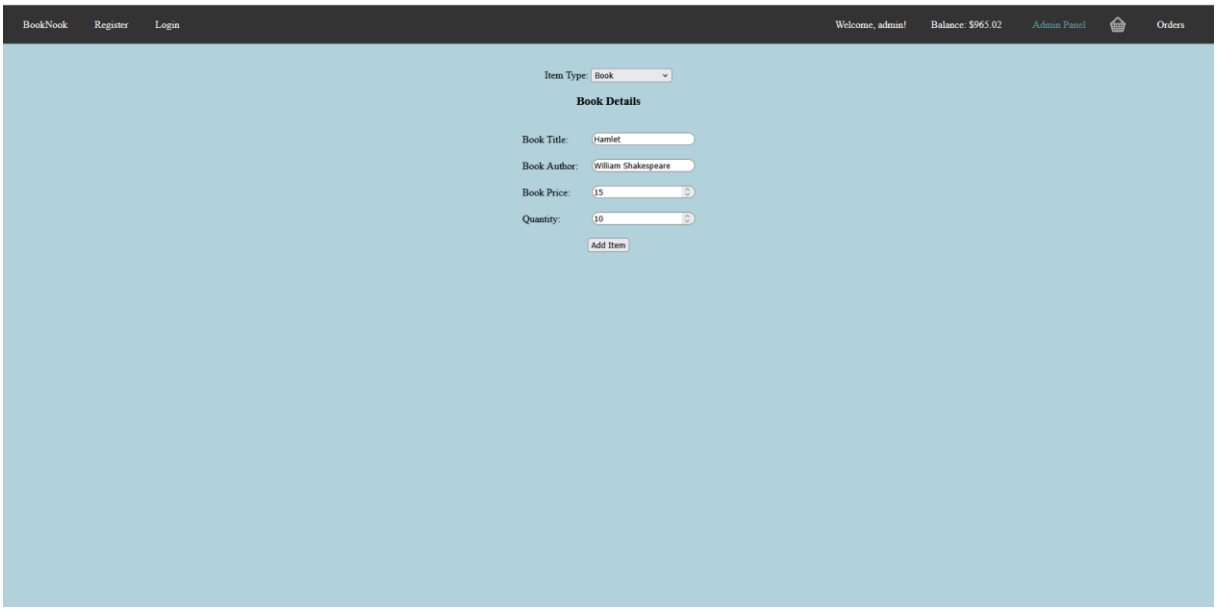
Now let us try to add new Item by clicking Add Item button:



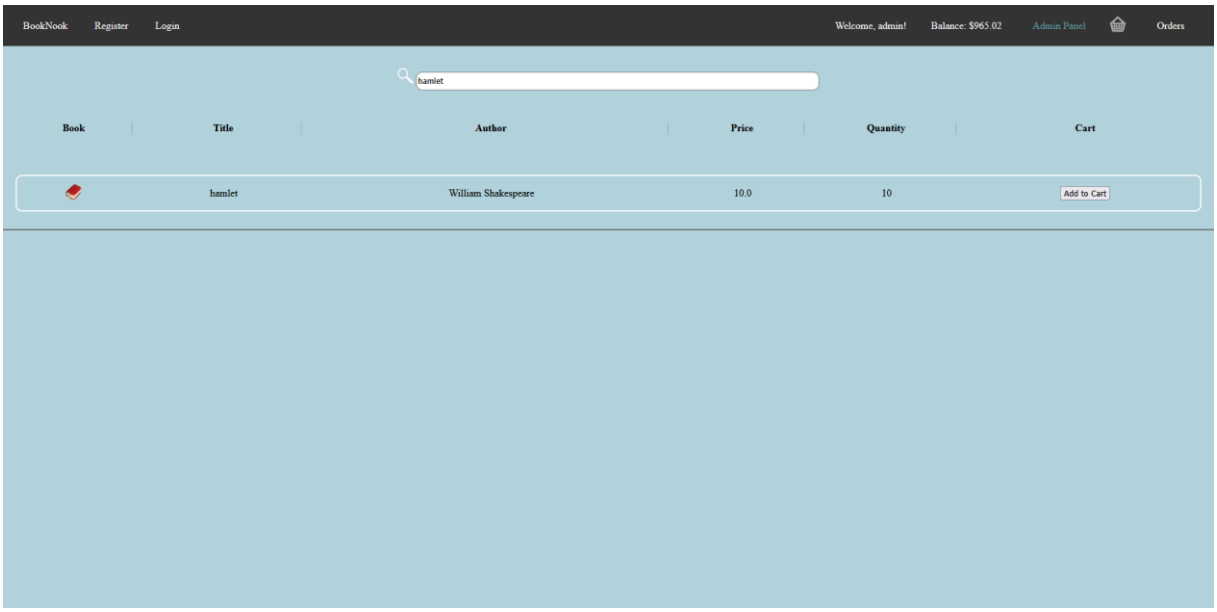
First we have to select if we want to add book or accessories. Lets add book:



We insert some values and now we click add item.



Now we can go back to BookNook and search hamlet and you can see that it is there:



Conclusion

The **BookNook** project successfully implements an online bookstore using **Java, JSP, JDBC, and SQLite** while following **Object-Oriented Design principles**. The system includes essential functionalities such as **user authentication, shopping cart management, order placement, and administrative controls**. The structured approach to **design and implementation** ensures that the project remains **scalable, maintainable, and efficient**. The project meets the defined requirements and showcases how **Object-Oriented Design** can be effectively applied in web development.