Course Submission Cover Sheet

Module: CS6004ES Application Development

Assignment no: 001

Weighting: 30%

Deadline: TBC

Module Leader: Mr. Chamila Karunathilaka **Student ID: E182014**

Please note that there are specific regulations concerning **the use of AI and Academic Misconduct**. Below are extracts from these regulations. By signing, you acknowledge that you have read and understood these extracts.

(signature:) I M Zairaz Date: 2025 Mar 15th

This header sheet should be attached to the work you submit.

Academic Integrity means being honest in your academic work and your studies and making sure that you acknowledge the work of others and giving credit where you have used other people's ideas as part of presenting your arguments. Your assessment submissions must therefore always be entirely your own work, based on your own learning and appropriately referenced including how you have used Generative AI. The University regards the use of Generative AI applications by students to deceive to gain unfair advantage as **academic misconduct**. This usage includes:

- **Plagiarism**, where AI tools are used to generate output and ideas that are presented or submitted as if they were the student's own work, without proper citation or references.

- Where a complete assignment is created using Generative AI and represented as a student's own work, this will be regarded as contract cheating in the same way as commissioning an 'Essay Mill' or other third party to complete your work. Further information can be found on : Guidance on the use of Artificial Intelligence.

CS6004ES –Application Development

Coursework 1 –Main-Sit 2024/25

In this individual coursework, you are tasked to develop and document a Windows Forms application in C. Upon completion, you are required to submit a comprehensive report detailing your implementation process and demonstrate the application in a Viva. It will be assessed using Visual Studio 2017 or any higher version and any features not working in the standard installation of Visual Studio 2017 or any higher version will not be assessed.

Submission Deadline:

Deliverables

Documentation

You are expected to submit a document containing the following sections.

- Installation Guide and User Manual
- Concise Description of Your Logical Solution
- Architecture Diagram, ER Diagram, and UML Diagrams:
- Detailed Description of Classes, Properties, and Methods:
- Own Reflection on Your Experience

Viva voce and Demonstration

After submitting the document, you will be required to demonstrate the functionality of your application in a Viva session.

- Be prepared to explain your design and implementation choices.
- Demonstrate key functionalities, such as adding books, processing orders, managing inventory, and generating reports.
- Discuss your approach to solving problems, testing, and ensuring the application's robustness.

Evaluation Criteria

Your submission and Viva will be evaluated based on the following.

Correctness - Does the application meet the functional requirements (user login, book management, order processing, inventory management, reporting)?

Code Quality - Is your code well-organized, readable, and following object-oriented principles (e.g., encapsulation, inheritance)?

Design - Are the architecture and class design efficient, modular, and maintainable?

Documentation - Is your submission well-documented, with clear explanations in the manual, solution descriptions, and diagrams?

Presentation - How well you present and explain your solution during the Viva.

Case Study: BookHaven - Bookstore Management System (Windows Forms Application)

BookHaven is a mid-sized bookstore specializing in selling various books ranging from fiction to non-fiction, educational materials, and collectibles. BookHaven is seeking to modernize its operations by developing a Windows Forms-based desktop application to streamline inventory management, sales tracking, and customer management. The application should help staff manage books, track customer orders, manage inventory, and generate sales reports.

Requirements for the System

User Login System - Implement a secure login system that allows staff to access the application with appropriate credentials. Different roles such as "Admin" and "Sales Clerk" should have different levels of access.

Book Inventory Management - Provide a feature to add, update, delete, and search for books in the inventory. Each book should have details like title, author, genre, ISBN, price, and stock quantity.

Customer Management - Develop functionality to add and manage customer details. The system should allow the staff to create customer profiles, store contact information, and keep track of purchase history.

Sales Transaction - Create a point-of-sale (POS) module where staff can select books for purchase, calculate the total cost, apply discounts (if any), and generate sales receipts for customers. The system should also update inventory based on sales.

Order Management - Implement an order management system to track customer orders. The system should allow staff to place, update, and manage orders. Customers may either pick up the books in-store or have them delivered.

Supplier Management - Allow the store admin to manage supplier information. This includes adding new suppliers, tracking supplier details (e.g., name, contact info), and generating orders for new stock from suppliers when inventory runs low. Here, suppliers refer to the companies or individuals that provide the bookstore with the books and other related products they sell (Book Publishers, Distributors, etc.).

Admin Dashboard - Create an admin dashboard that provides an overview of system metrics like total sales, inventory levels, customer activity, and staff performance. Admins should also have access to critical settings such as managing users and modifying business details.

Reporting and Analytics - Provide reporting features to generate sales reports, including daily, weekly, and monthly sales summaries. Reports should also highlight the best-selling books and overall inventory status.

Security and Data Protection - Implement security measures such as role-based access control, encryption of sensitive data, and secure storage of customer information to prevent unauthorized access.

Marking Scheme for the CS6004ES Group Coursework

This individual coursework is worth 30% of the module mark. The following are guidelines for marking. Mark each item listed below on a scale of 0 to 5. Then multiply the mark by the weighting indicated, sum it up, and divide it by 2 to get the final mark.

Mark	**Characterized by**
0	No work or work completely irrelevant
1	Work started in the right direction but yielded no results
2	Some results achieved, but with significant deficiencies and/or errors
3	Acceptable results but incomplete, or some good results with minor errors
4	Good results, but there is room for further improvement
5	Excellent results

	Item	**Weight**	***Mark (0 - 5)***	***Weight x Mark***
Implementation				
1	The application user interface	2	5	10.00
2	Task 1: User Login System.	3	5	15.00
3	Task 2: Book Inventory Management	3	5	15.00
4	Task 3: Customer Management	3	5	15.00
5	Task 4: Sales Transaction	4	5	20.00
6	Task 5: Order Management	4	5	20.00
7	Task 6: Supplier Management	3	5	15.00
8	Task 7: Admin Dashboard	3	5	15.00
9	Task 8: Reporting and Analytics	3	5	15.00
10	Task 9: Security and Data Protection	2	5	10.00
Documentation				
A	Installation Guide and User Manual	1	5	5.00
B	A concise description of your logical solution to each of the implemented functions of the application.	1	5	5.00
C	Architecture diagram, ER Diagram, and UML diagrams	1	5	5.00
D	A detailed description of the classes, properties, and methods	1	5	5.00
E	Own reflection of your experience	1	5	5.00
Programming style				
1	Clarity of code which shows the underlying algorithm	1	5	5.00

2	Sensible naming of programmer-defined variables, classes, properties, and methods	1	5	5.00
3	Useful comments in code	1	5	5.00
4	Data validation and exception handling	1	5	5.00
5	Interface design and usability of the system	1	5	5.00
	Total			**200.00**

Contents

1. Installation Guide and User Manual

1.1 System Requirements

1.1.1 Hardware Requirements:

- Operating System: Windows 10/11
- Processor: Intel Core i3 or higher
- RAM: 4GB minimum (8GB recommended)
- Disk Space: At least 10GB free

1.1.2 Software Requirements:

- .NET Framework (4.7.2)
- Visual Studio 2017 or later
- SQL Server (SSMS)

1.2 Downloading the Application

1.2.1 From GitHub:

- Open a command prompt and run the following command:
- https://github.com/imzairaz/BookHaven.git
- Navigate to the downloaded folder.
- Extract the BookHaven.zip

1.3 Installing the Application

1.3.1 Setting up in Visual Studio:

- Open Visual Studio

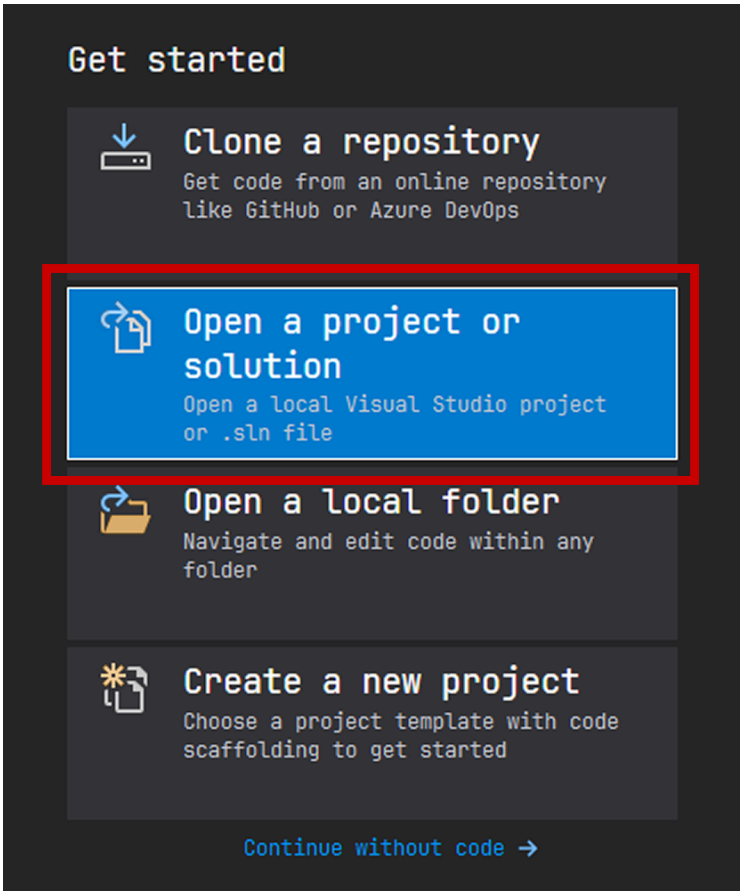


Figure 1 - How to Open the Project 01

- Click **"Open a Project or Solution"**

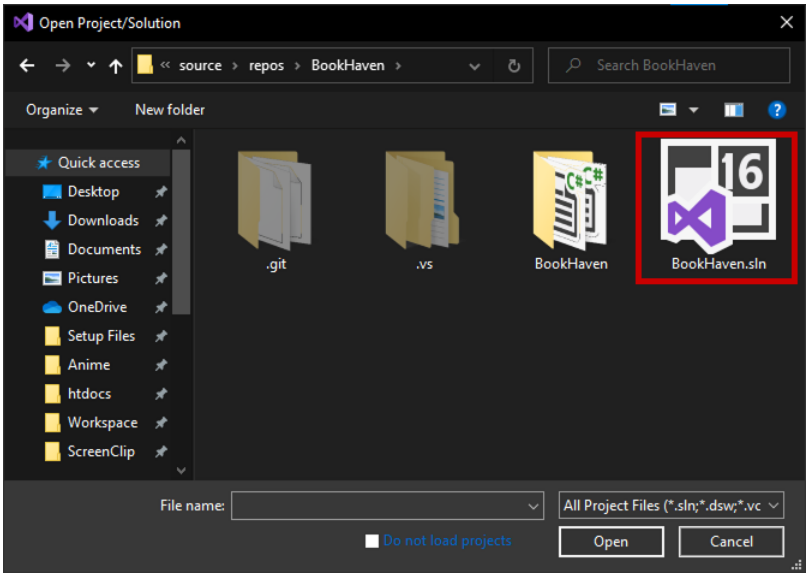


Figure 2 - How to Open the Project 02

- Navigate to the project folder and open the **"BookHaven.sln"** file.
- Select the "**BookHaven.sln"** File then Click the "**Open**" Button

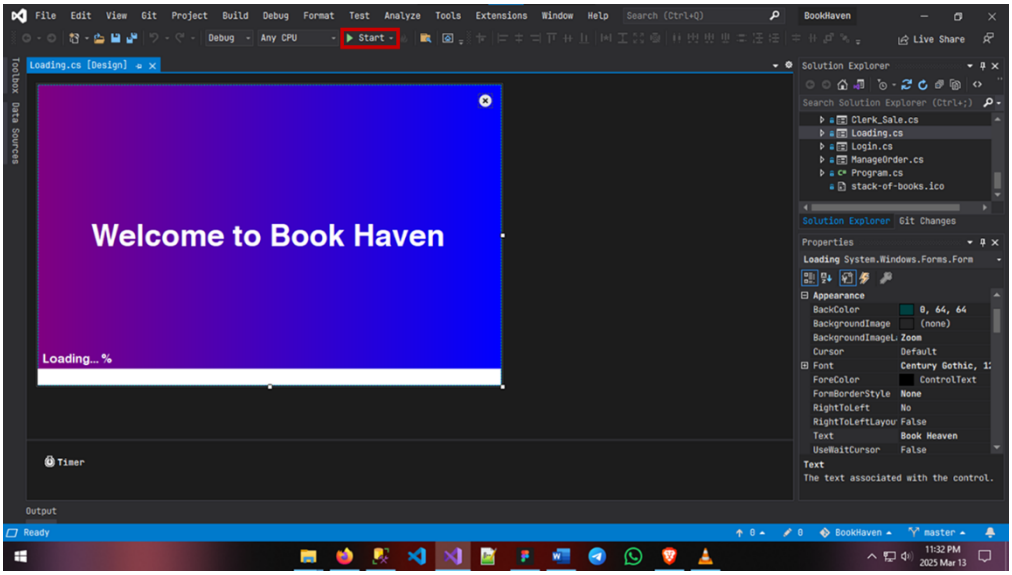


Figure 3 - How to Open the Project 03

- After you open the BookHaven.sln file you will get this interface
- Click the red highlight to run or press F5 to run the application.

1.4 Setting Up the Database

1.4.1 Open SQL Server Management Studio (SSMS).

- Execute the provided `.sql` script to create tables.

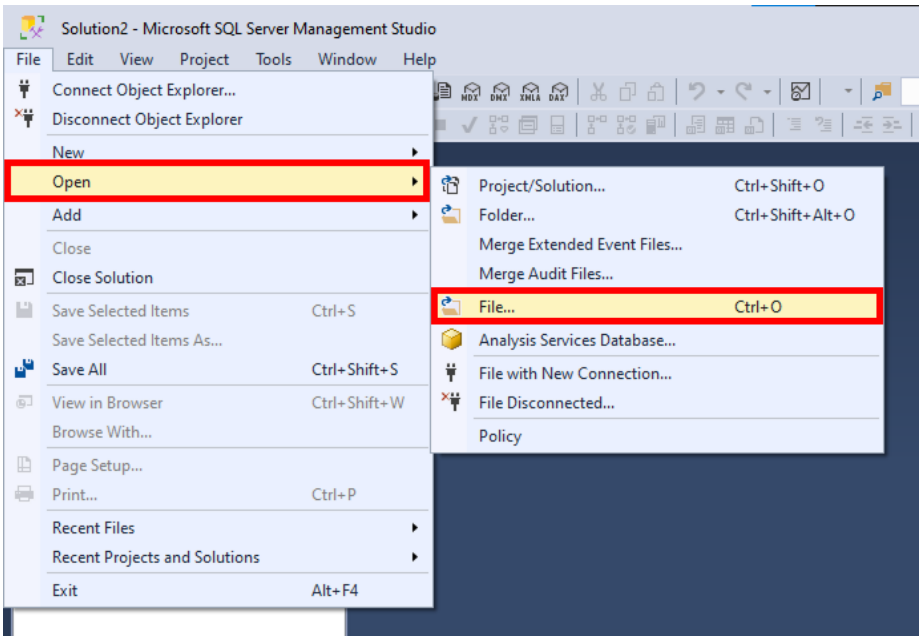


Figure 4 - Database Setup 01

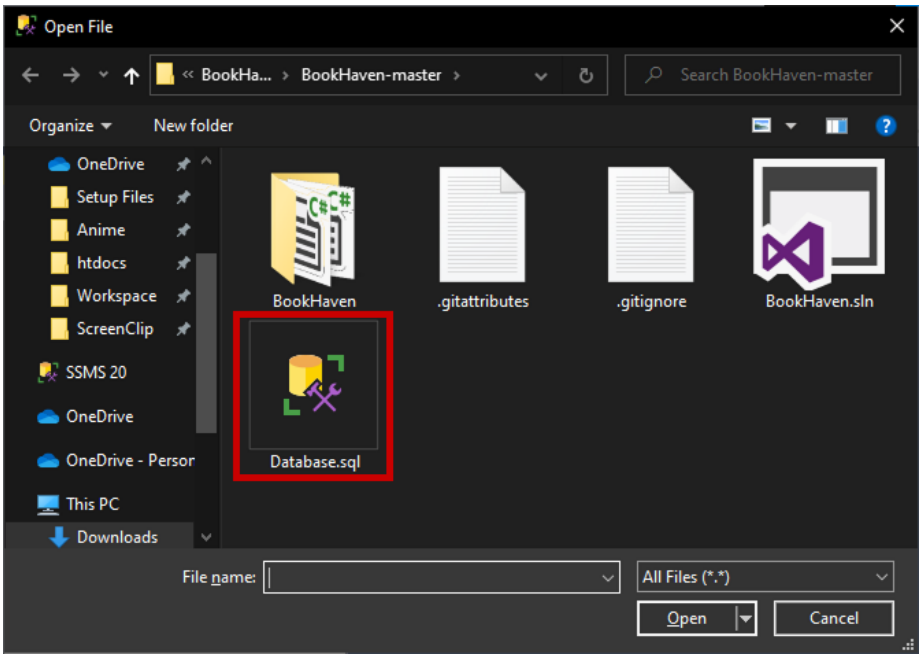


Figure 5 - Database Setup 02

- Execute the provided database.sql script to create database and tables.

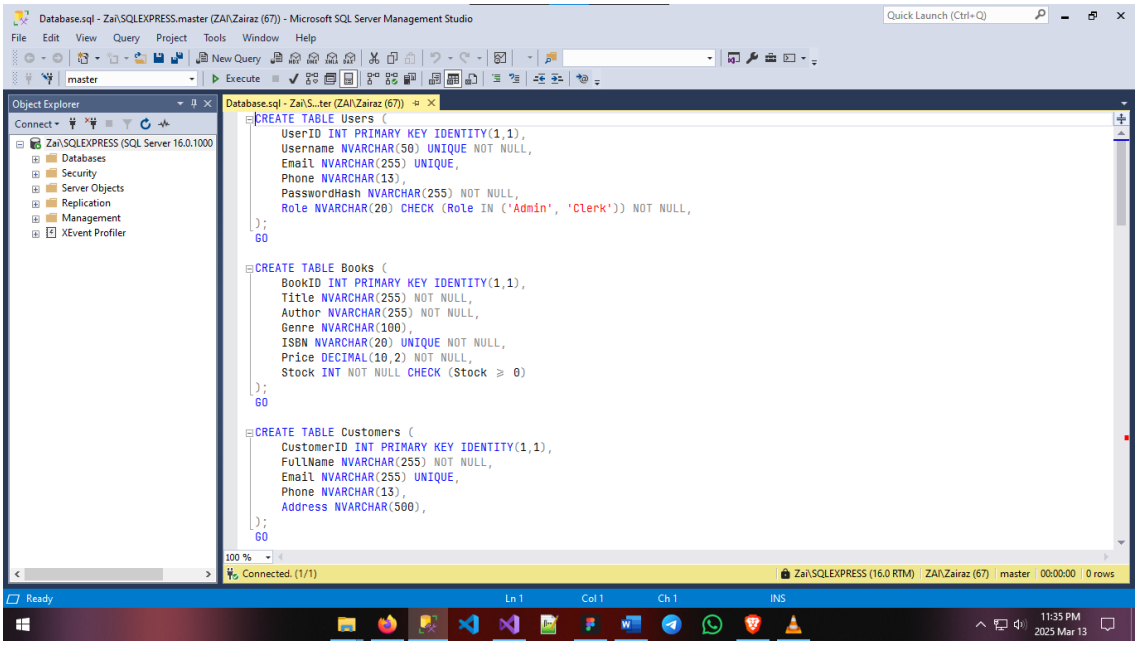


Figure 6 - Database Setup 03

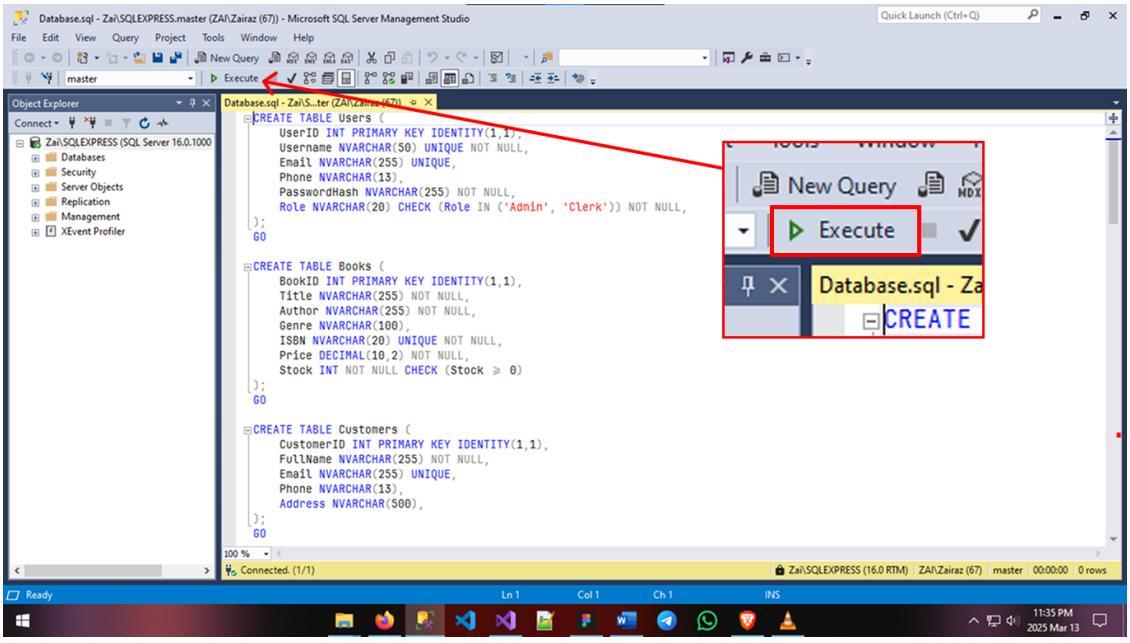


Figure 7 - Database Setup 04

- After Open the Database.sql file on SSMS the click the **"Execute"** Button as shows on above the image

- Update the **connection string** in your application's `DatabaseHelper.cs`

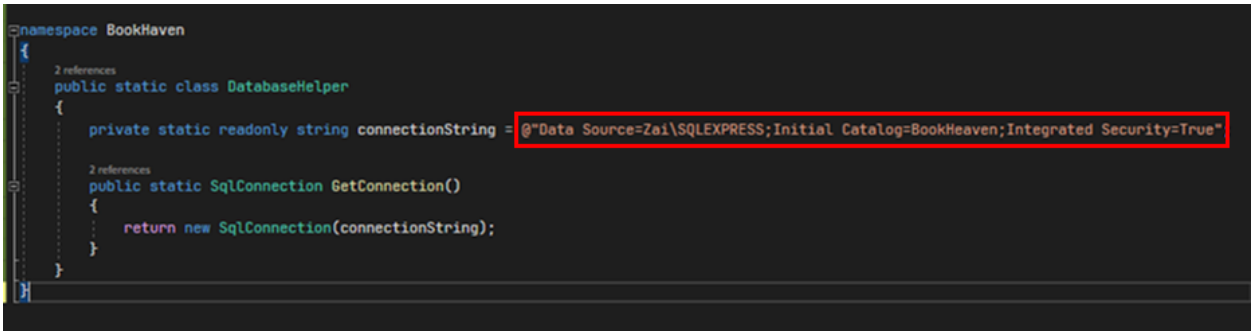


Figure 8 - Database String Main

5. Running the Application

- If running from Visual Studio, press F5.

Issue	**Solution**
Error: Missing .NET Framework	Install from Microsoft's official site.
Error: Cannot connect to the database	Verify SQL Server is running and connection string is correct.
Error: Application crashes	Ensure all system requirements are met.

1.5 User Roles and Access Levels

The BookHaven application implements a role-based access control (RBAC) system to manage user access and permissions. The system has two main user roles: Admin and Sales Clerk, each with different access levels to various parts of the application.

1. **Admin User:**

Access Level: Full access to all features of the application.

Permissions:

- User Management: Can add, edit, or delete user accounts.
- Book Inventory Management: Can add, update, delete, and view books in the inventory.
- Customer Management: Can view, add, and update customer details.
- Sales Transactions: Can view, manage, and generate sales receipts.
- Order Management: Can view, edit, and manage customer orders.
- Supplier Management: Can add or update supplier information and generate orders.
- Reporting and Analytics: Can generate sales reports, monitor inventory levels, and analyze sales data.
- Admin Dashboard: Can access critical system settings and view system-wide metrics such as sales, inventory, and customer activity.

2. **Sales Clerk:**

Access Level: Limited access to only specific features related to sales, inventory, and customer management.

Permissions:

- Book Inventory Management: Can view and search for books, but cannot add, update, or delete books.
- Sales Transactions: Can process sales transactions, calculate total cost, apply discounts, and print receipts for customers.

- Customer Management: Can view customer profiles, add new customers, and track purchase history.
- Order Management: Can view and manage customer orders, but cannot create new supplier orders or access critical settings.
- No Access to Admin Dashboard or Reporting Features: Sales Clerks do not have access to sales reports or overall system metrics.

1.6 How Role-based Access Control Works:

- During the login process, the system checks the credentials provided by the user and assigns roles based on the stored data in the database (e.g., whether the user is an Admin or a Sales Clerk).
- Once logged in, users are redirected to a dashboard based on their role:
    - Admins are taken to the Admin Dashboard.
    - Sales Clerks are taken to the Sales Clerk interface where they can process sales, manage inventory

1.7 How to Use the Application (Step-by-Step Guide)

This section will walk you through the key features of the **BookHaven** application, explaining how to use each part of the system.

Launch the BookHaven application.

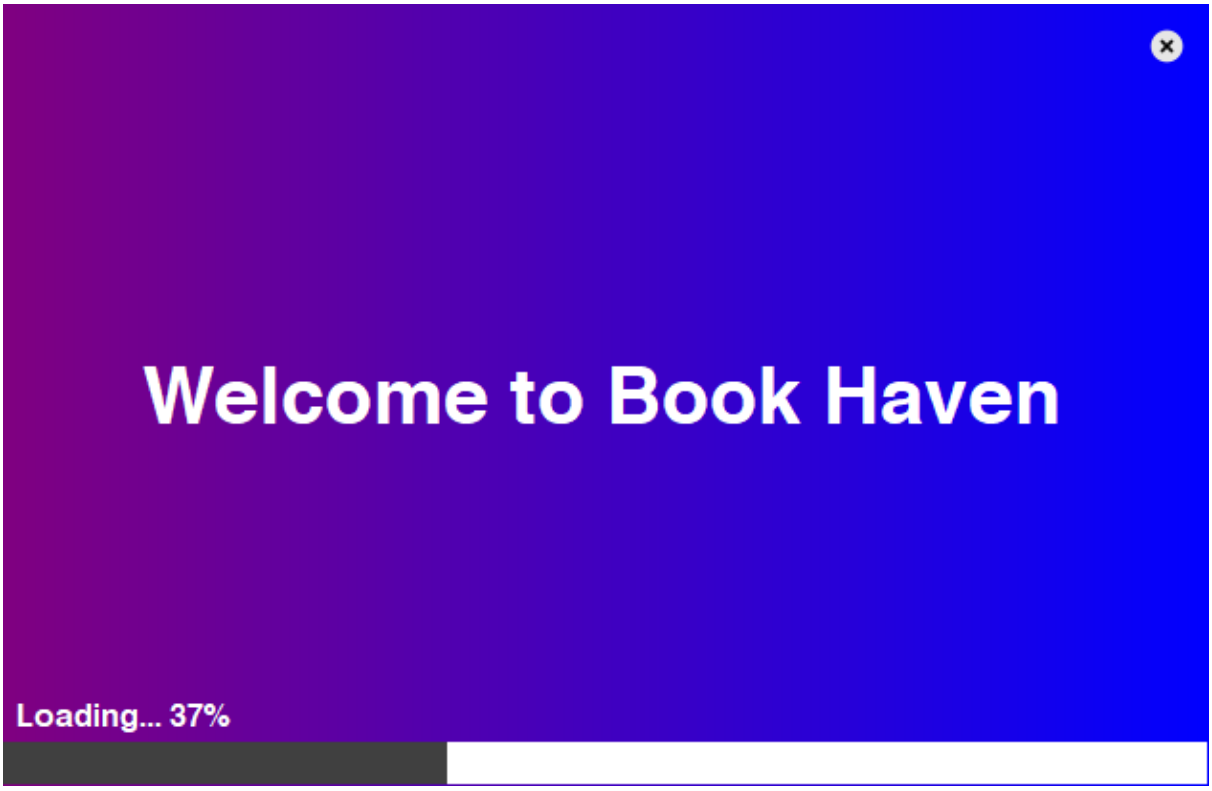


Figure 9 – Launch the Application

Login Form

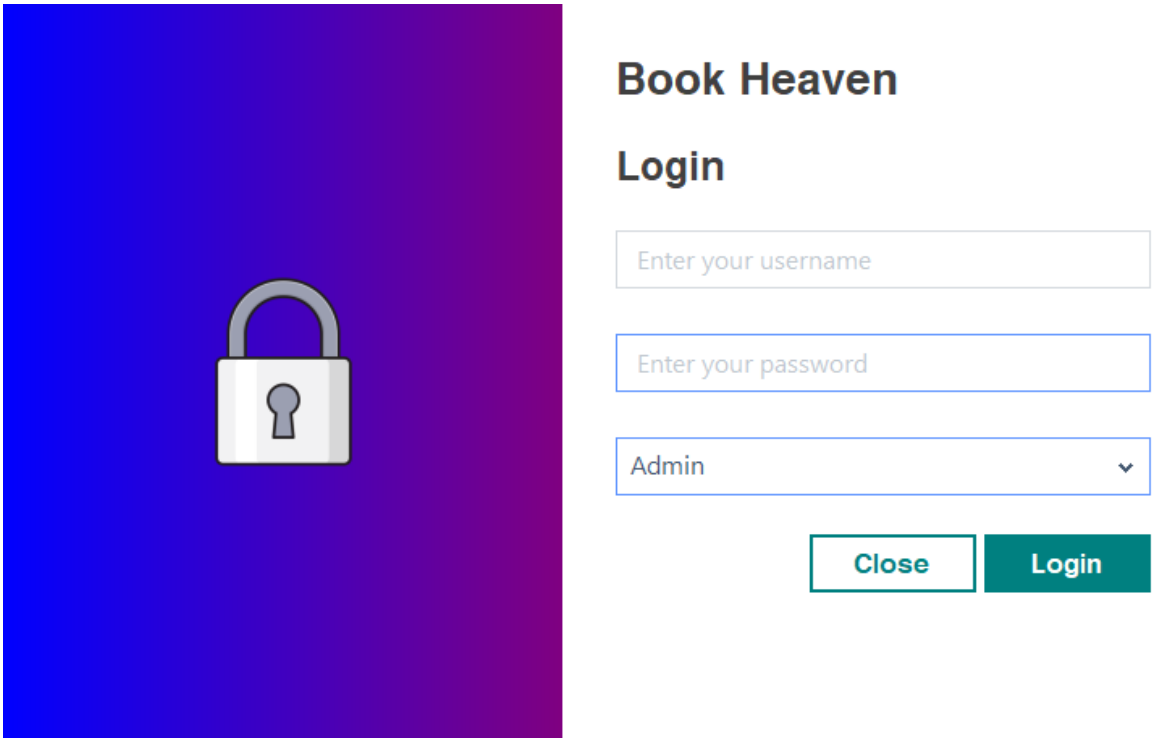


Figure 10 - Login Interface

1. On the login screen, enter your Username and Password.
2. Click the Login button.
    - If you are an Admin, you will be redirected to the Admin Dashboard.
    - If you are a Sales Clerk, you will be taken to the Sales Interface.

Admin

Admin Dashboard:

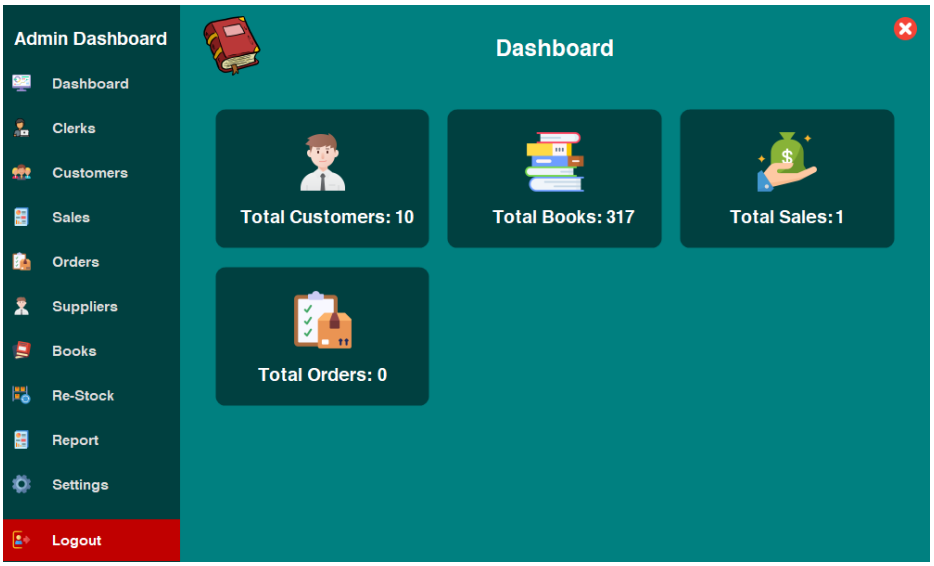


Figure 11 - Admin Dashboard

- Admin Dashboard Statistics
- Shows (Total Customer, Total Books, Total Sales, Total Orders)

Manage Clerk:

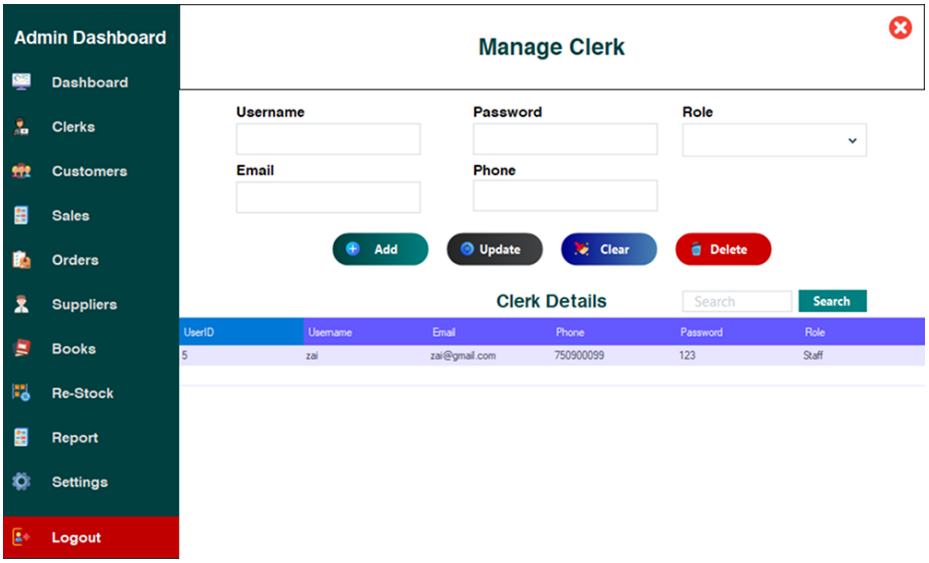


Figure 12 - Manage Clerk

Click on the “Clerk” option to add, update, or delete and user accounts(Sale Clerk).

Book Inventory:

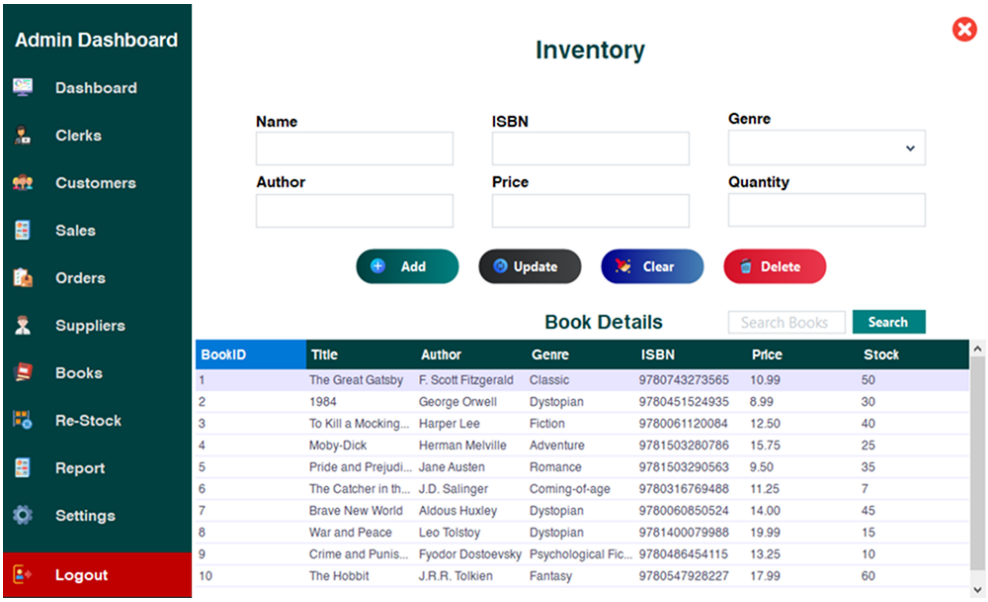


Figure 13 - Inventory

- From the **"Inventory"** section, you can add, update, and delete books.
- You can also search for books based on parameters like title, author, genre

Restock:

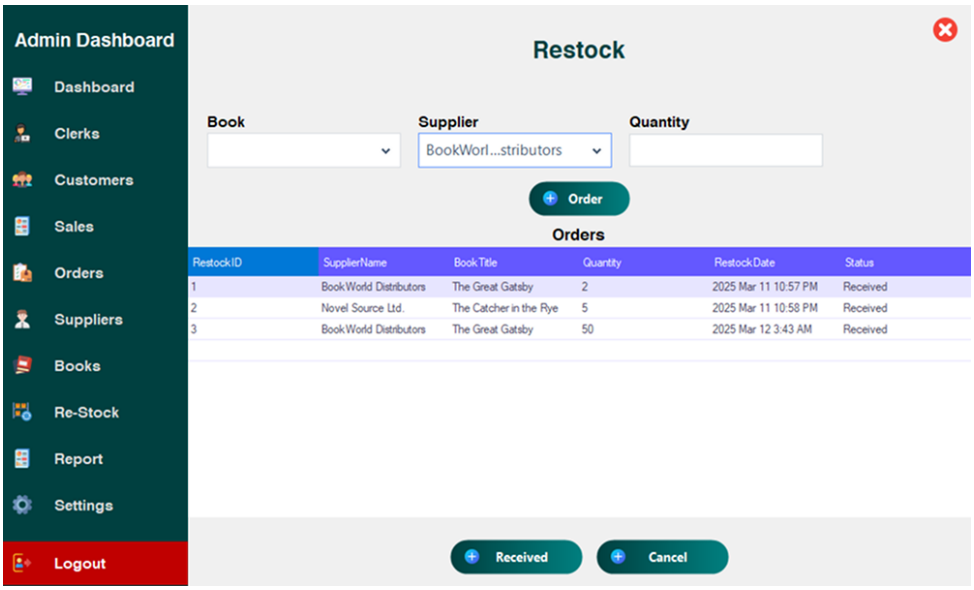


Figure 14 - Restock Book

- From the "Re-Stock" Section, Can order Book and update the status it's received or cancel

Manage Supplier:

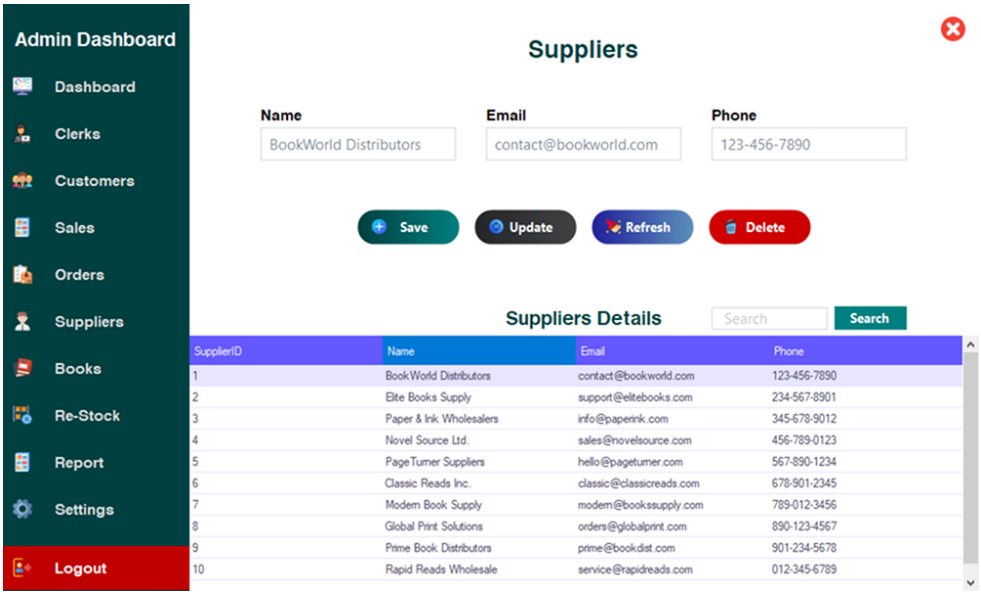


Figure 15 – Manage Supplier

- In the "Suppliers" section, you can add, update or Delete supplier information, manage contacts

Clerk

Clerk Sales

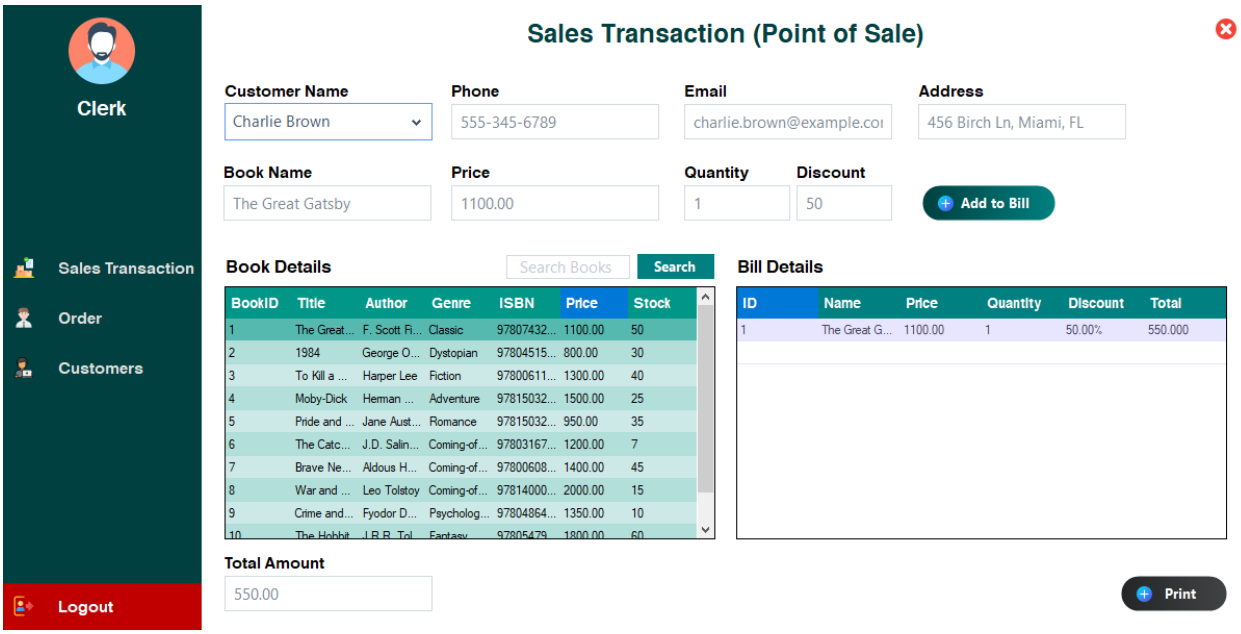


Figure 16 - Clerk Sales (POS)

- To start a sale, click on the “Sales Transaction”
- Search for books by title, author, or genre, and add and enter any applicable discounts (if allowed). them to the Bill
- The system will calculate the total and allow you to print a receipt for the customer.
- Upon completing the sale, the inventory will be automatically updated to reflect the books sold.

Manage Customers:

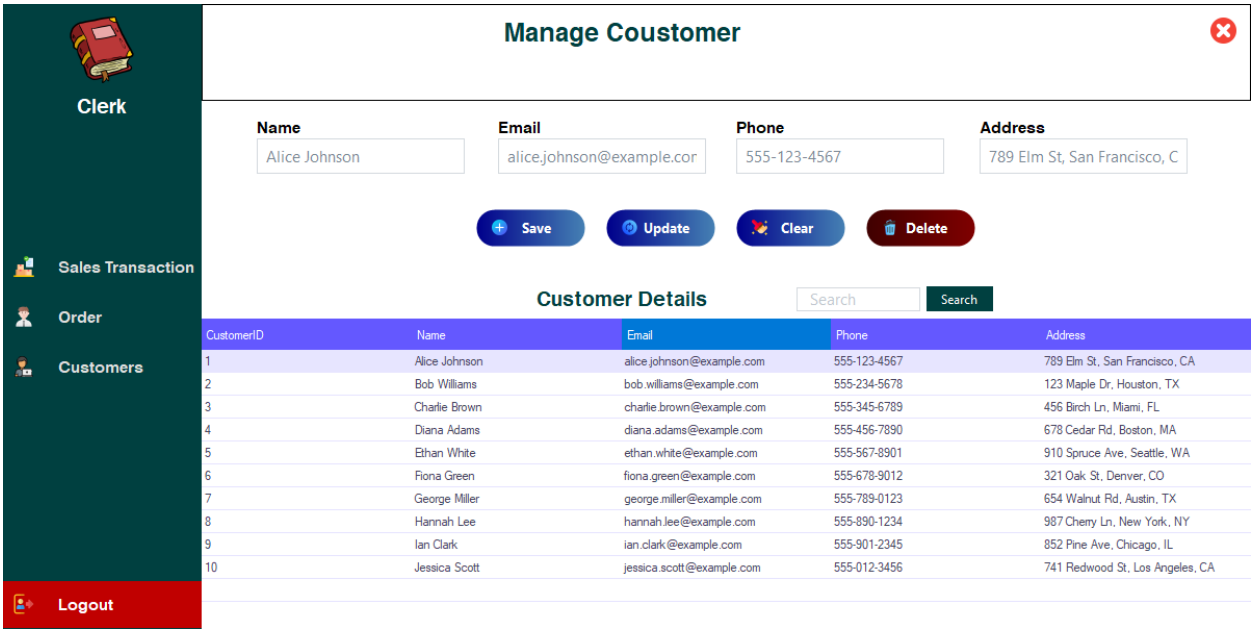


Figure 17 - Manage Customers

- Click on “Customers” to view existing customer profiles.
- You can add a new customer by clicking “Add Customer” and entering details such as name, email, and phone number, Address.
- You can also track a customer's purchase history to assist them with future purchases.

Orders

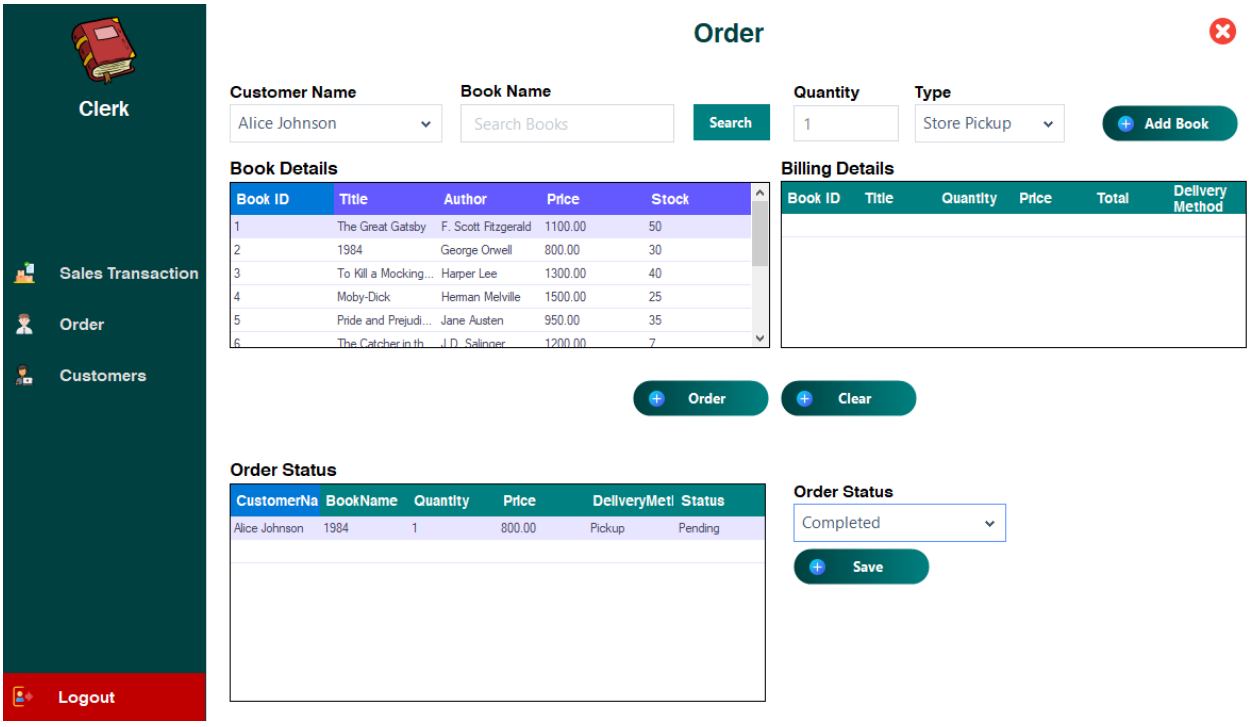


Figure 18 - Order Form

- Make a order for customers request
- In the “Orders” section, you can view and manage customer orders.
- Orders can be marked as “Delivered” or “Picked Up” based on the status of the customer’s request.

2. Concise Description of Your Logical Solution to Each Implemented Function

2.1 User Login System

Code Screenshot 1: Login Button Click Event

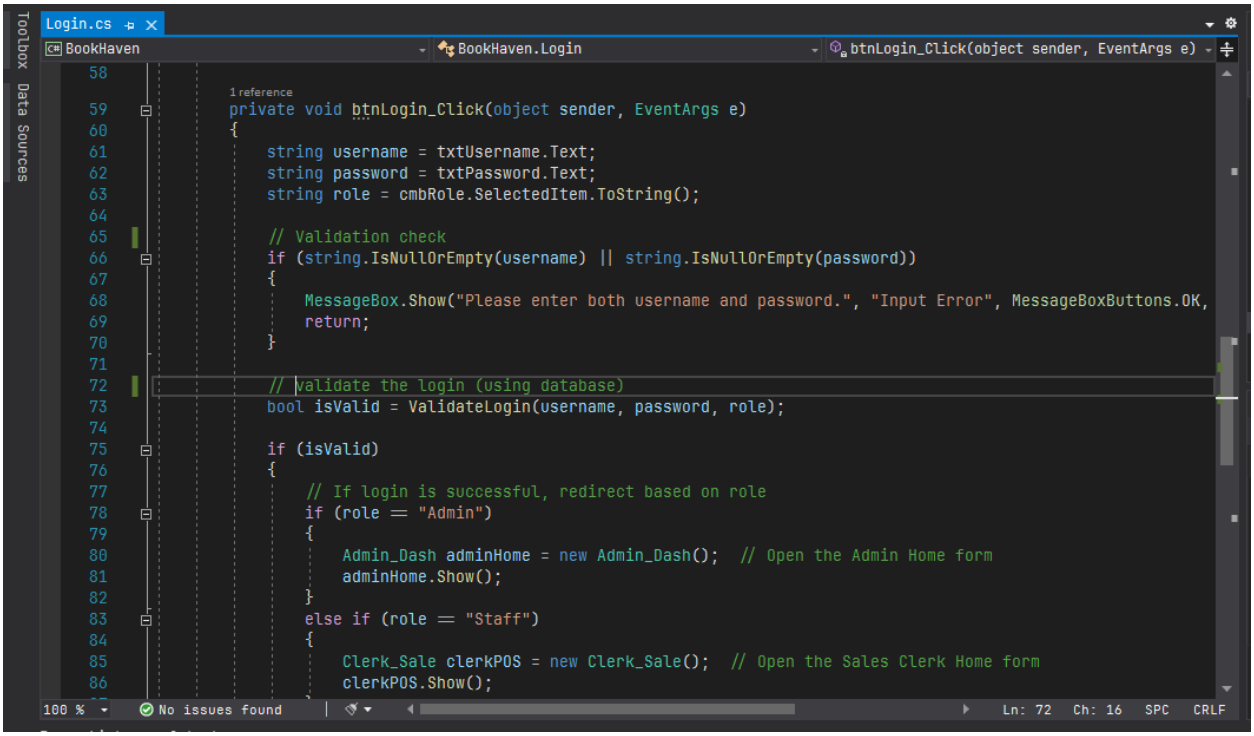


Figure 19 - Login Function

1. Username and Password Check: First, the code checks if both the username and password fields are filled in. If not, it shows a message asking the user to fill both fields.
2. Validate the Login: After that, the code checks the username, password, and role against what's stored in the database. If everything matches, it moves to the next step.
3. Redirect to Dashboard: If the login is successful, the user is sent to their home screen based on their role:
    - Admin: Takes the user to the Admin Dashboard.
    - Staff: Takes the user to the Sales Clerk Dashboard.
4. Error Handling: If the login fails (wrong username, password, or role), an error message is shown.

Code Screenshot 2: Validate Login Method

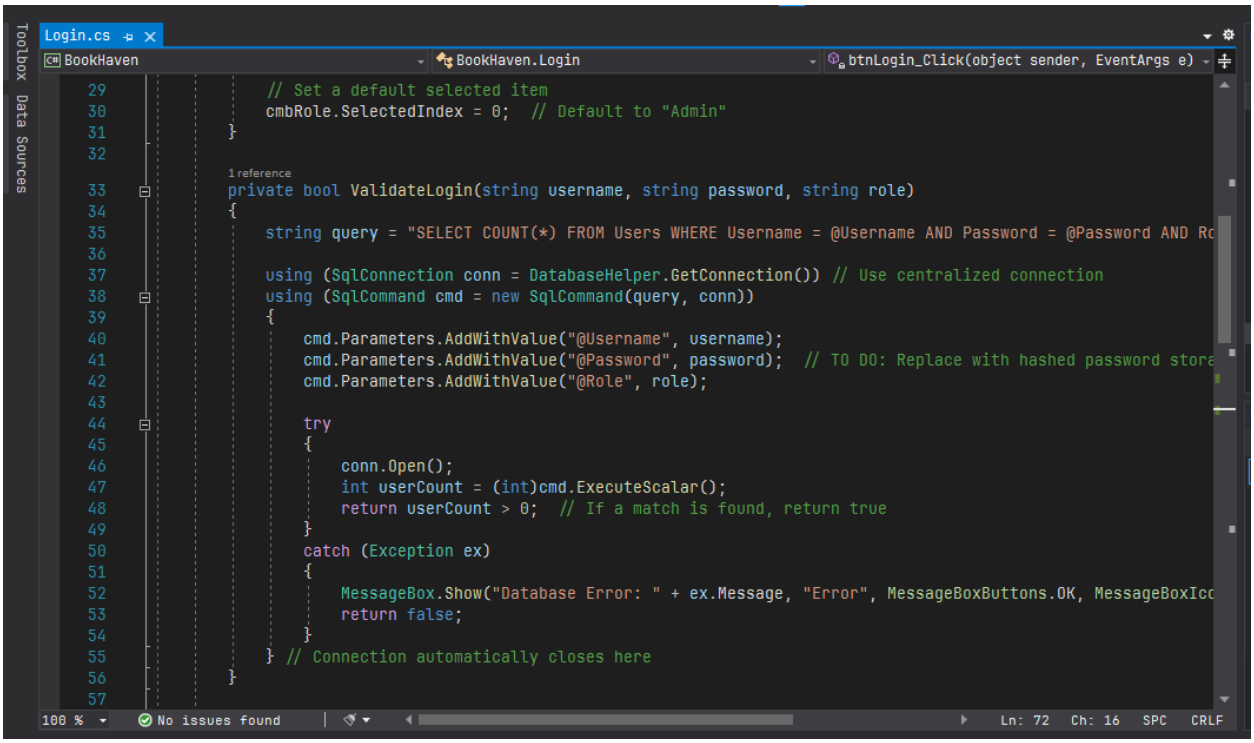


Figure 20 - Login Validation

Explanation: This method checks the database to make sure the username, password, and role are correct. It runs a query on the database, and if it finds a match, it returns true. If there's no match, it returns false.

2.2 Book Inventory Management

In the Admin_Books form, various functionalities are implemented for book inventory management using the CRUD operations (Create, Read, Update, Delete). Below is an explanation of each important part of the code.

1. Loading Data:

When the form is loaded, the Admin_Book_Load method calls LoadData() to display all books in the DataGridView (a table on the form showing all books).

```
1 reference
private void Admin_Book_Load(object sender, EventArgs e)
{
    LoadData();  // Load data into the DataGridView when the form loads
}
```

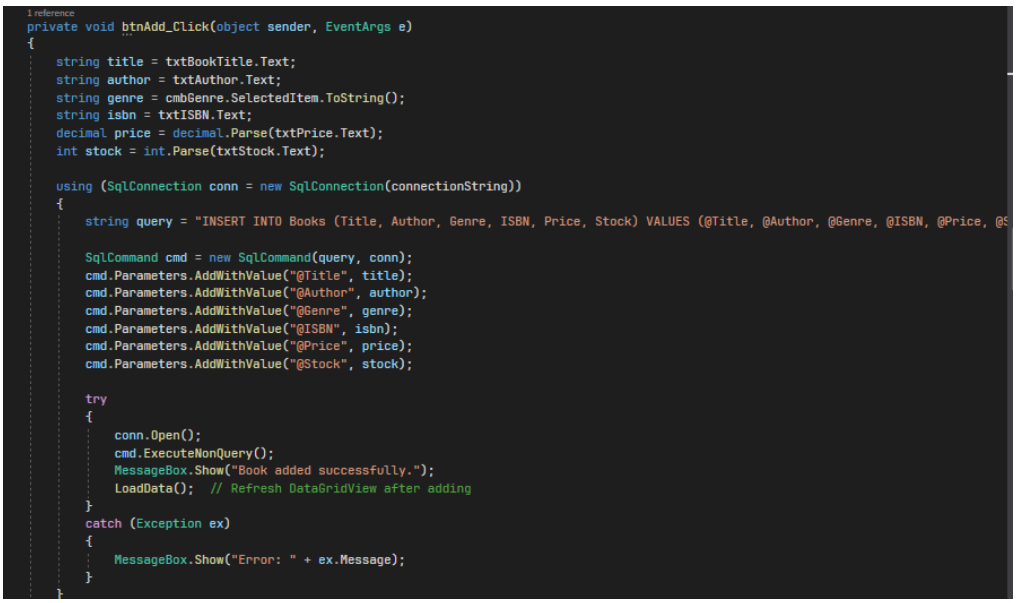
Figure 21 - Load Data

Explanation:

When the Admin_Books form is opened, it automatically loads and displays all books from the Books table in the database.

2. Adding a Book:

The btnAdd_Click method is responsible for adding new books to the inventory by collecting data from textboxes and combo boxes.



```
1 reference
private void btnAdd_Click(object sender, EventArgs e)
{
    string title = txtBookTitle.Text;
    string author = txtAuthor.Text;
    string genre = cmbGenre.SelectedItem.ToString();
    string isbn = txtISBN.Text;
    decimal price = decimal.Parse(txtPrice.Text);
    int stock = int.Parse(txtStock.Text);

    using (SqlConnection conn = new SqlConnection(connectionString))
    {
        string query = "INSERT INTO Books (Title, Author, Genre, ISBN, Price, Stock) VALUES (@Title, @Author, @Genre, @ISBN, @Price, @S

        SqlCommand cmd = new SqlCommand(query, conn);
        cmd.Parameters.AddWithValue("@Title", title);
        cmd.Parameters.AddWithValue("@Author", author);
        cmd.Parameters.AddWithValue("@Genre", genre);
        cmd.Parameters.AddWithValue("@ISBN", isbn);
        cmd.Parameters.AddWithValue("@Price", price);
        cmd.Parameters.AddWithValue("@Stock", stock);

        try
        {
            conn.Open();
            cmd.ExecuteNonQuery();
            MessageBox.Show("Book added successfully.");
            LoadData();  // Refresh DataGridView after adding
        }
        catch (Exception ex)
        {
            MessageBox.Show("Error: " + ex.Message);
        }
    }
```

Figure 22 - Button Add Functions

Explanation: When the Add Book button is clicked, the application collects the book details (title, author, genre, etc.) from the user input and adds a new book to the Books table in the database.

3. Updating a Book:

The btnUpdate_Click method allows the admin to update the details of an existing book in the inventory.

```
1 reference
private void btnUpdate_Click(object sender, EventArgs e)
{
    if (dgvBook.SelectedRows.Count > 0)
    {
        int bookID = Convert.ToInt32(dgvBook.SelectedRows[0].Cells[0].Value); // Get BookID from selected row
        string title = txtBookTitle.Text;
        string author = txtAuthor.Text;
        string genre = cmbGenre.SelectedItem.ToString();
        string isbn = txtISBN.Text;
        decimal price = decimal.Parse(txtPrice.Text);
        int stock = int.Parse(txtStock.Text);

        using (SqlConnection conn = new SqlConnection(connectionString))
        {
            string query = "UPDATE Books SET Title = @Title, Author = @Author, Genre = @Genre, ISBN = @ISBN, Price = @Price, Stock =

            SqlCommand cmd = new SqlCommand(query, conn);
            cmd.Parameters.AddWithValue("@Title", title);
            cmd.Parameters.AddWithValue("@Author", author);
            cmd.Parameters.AddWithValue("@Genre", genre);
            cmd.Parameters.AddWithValue("@ISBN", isbn);
            cmd.Parameters.AddWithValue("@Price", price);
            cmd.Parameters.AddWithValue("@Stock", stock);
            cmd.Parameters.AddWithValue("@BookID", bookID);

            try
            {
                conn.Open();
                cmd.ExecuteNonQuery();
                MessageBox.Show("Book updated successfully.");
                LoadData();  // Refresh DataGridView after updating
            }
```

Figure 23 - Button Update

Explanation: When the Update button is clicked, the selected book's details are retrieved, and the BookID (primary key) is used to update the specific book in the database.

4. Deleting a Book:

The btnDelete_Click method allows the admin to delete a book from the inventory.

```
1 reference
private void btnDelete_Click(object sender, EventArgs e)
{
    if (dgvBook.SelectedRows.Count > 0)
    {
        int bookID = Convert.ToInt32(dgvBook.SelectedRows[0].Cells[0].Value); // Get BookID from selected row

        using (SqlConnection conn = new SqlConnection(connectionString))
        {
            string query = "DELETE FROM Books WHERE BookID = @BookID";

            SqlCommand cmd = new SqlCommand(query, conn);
            cmd.Parameters.AddWithValue("@BookID", bookID);

            try
            {
                conn.Open();
                cmd.ExecuteNonQuery();
                MessageBox.Show("Book deleted successfully.");
                LoadData();  // Refresh DataGridView after deleting
            }
            catch (Exception ex)
            {
                MessageBox.Show("Error: " + ex.Message);
            }
        }
    }
    else
    {
        MessageBox.Show("Please select a book to delete.");
    }
}
```

Figure 24 - Button delete

Explanation: When the Delete button is clicked, the selected book is deleted from the Books table in the database using its BookID.

5. Searching for a Book:

The btnSearch_Click method allows searching for books by title or author.

```
1 reference
private void btnClear_Click(object sender, EventArgs e)
{
    txtBookTitle.Clear();
    txtAuthor.Clear();
    cmbGenre.SelectedIndex = -1;  // Clears the selected item in ComboBox
    txtISBN.Clear();
    txtPrice.Clear();
    txtStock.Clear();
}
1 reference
private void btnSearch_Click(object sender, EventArgs e)
{
    string searchQuery = txtSearch.Text;

    try
    {
        using (SqlConnection conn = new SqlConnection(connectionString))
        {
            string query = "SELECT * FROM Books WHERE Title LIKE @SearchQuery OR Author LIKE @SearchQuery";
            SqlDataAdapter adapter = new SqlDataAdapter(query, conn);
            adapter.SelectCommand.Parameters.AddWithValue("@SearchQuery", "%" + searchQuery + "%");
            DataTable dataTable = new DataTable();

            conn.Open();
            adapter.Fill(dataTable);

            dgvBook.DataSource = dataTable;  // Display search results in DataGridView
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show("Error searching data: " + ex.Message);
    }
}
```

Figure 25 - Button Search

Explanation: The Search functionality allows the admin to search for books based on the title or author. It uses the LIKE SQL operator to search for matches.

2.3 Customer Management

This class handles the customer management functionality for the Clerk. It allows the clerk to add, update, delete, search, and view customer information. The DataGridView is used to display customer data, and SQL is used to interact with the database.

Customer Data into DataGridView:

```
1 reference
private void Clerk_Customer_Load(object sender, EventArgs e)
{
    LoadData();  // Load data into the DataGridView when the form loads
}

// Method to load customer data into the DataGridView
5 references
private void LoadData()
{
    try
    {
        using (SqlConnection conn = new SqlConnection(connectionString))
        {
            string query = "SELECT * FROM Customers"; // SQL Query to get all customers
            SqlDataAdapter adapter = new SqlDataAdapter(query, conn);
            DataTable dataTable = new DataTable();

            conn.Open();
            adapter.Fill(dataTable);

            dgvCustomer.DataSource = dataTable;  // Set the DataGridView's data source to the DataTable
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show("Error loading data: " + ex.Message);
    }
}
```

Figure 26 - Customer Data Grid View

Explanation: The LoadData() method connects to the database, retrieves all customer records from the Customers table, and binds this data to the DataGridView.

Adding a New Customer:

It allows the clerk to add a new customer by entering their details.

```
1 reference
private void btnAdd_Click(object sender, EventArgs e)
{
    string name = txtName.Text;
    string email = txtEmail.Text;
    string phone = txtPhone.Text;
    string address = txtAddress.Text;

    using (SqlConnection conn = new SqlConnection(connectionString))
    {
        string query = "INSERT INTO Customers (Name, Email, Phone, Address) VALUES (@Name, @Email, @Phone, @Address)";

        SqlCommand cmd = new SqlCommand(query, conn);
        cmd.Parameters.AddWithValue("@Name", name);
        cmd.Parameters.AddWithValue("@Email", email);
        cmd.Parameters.AddWithValue("@Phone", phone);
        cmd.Parameters.AddWithValue("@Address", address);

        try
        {
            conn.Open();
            cmd.ExecuteNonQuery();
            MessageBox.Show("Customer added successfully.");
            LoadData();  // Refresh the DataGridView after adding a new customer
            ClearFields();  // Clear the input fields
        }
        catch (Exception ex)
        {
            MessageBox.Show("Error: " + ex.Message);
        }
    }
}
```

Figure 27 - Button Add on Customers

Explanation: The btnAdd_Click method collects data from the textboxes and inserts it into the Customers table in the database. After the insertion, it refreshes the DataGridView to show the updated list of customers and clears the input fields.

Updating Customer

It allows the clerk to update an existing customer's details.

```
private void btnUpdate_Click(object sender, EventArgs e)
{
    if (dgvCustomer.SelectedRows.Count > 0)
    {
        int customerId = Convert.ToInt32(dgvCustomer.SelectedRows[0].Cells[0].Value); // Get CustomerID from selected row
        string name = txtName.Text;
        string email = txtEmail.Text;
        string phone = txtPhone.Text;
        string address = txtAddress.Text;

        using (SqlConnection conn = new SqlConnection(connectionString))
        {
            string query = "UPDATE Customers SET Name = @Name, Email = @Email, Phone = @Phone, Address = @Address WHERE CustomerID = (

            SqlCommand cmd = new SqlCommand(query, conn);
            cmd.Parameters.AddWithValue("@Name", name);
            cmd.Parameters.AddWithValue("@Email", email);
            cmd.Parameters.AddWithValue("@Phone", phone);
            cmd.Parameters.AddWithValue("@Address", address);
            cmd.Parameters.AddWithValue("@CustomerID", customerId);

            try
            {
                conn.Open();
                cmd.ExecuteNonQuery();
                MessageBox.Show("Customer updated successfully.");
                LoadData();  // Refresh the DataGridView after updating
                ClearFields();  // Clear the input fields
            }
            catch (Exception ex)
            {
                MessageBox.Show("Error: " + ex.Message);
            }
```

Figure 28 - button update customers

Explanation: The btnUpdate_Click method checks if a row is selected. It retrieves the selected customer's CustomerID, then updates the corresponding record in the database with the new data from the input fields.

Deleting a Customer

It allows the clerk to delete a customer from the database.

```
private void btnDelete_Click(object sender, EventArgs e)
{
    if (dgvCustomer.SelectedRows.Count > 0)
    {
        int customerId = Convert.ToInt32(dgvCustomer.SelectedRows[0].Cells[0].Value); // Get CustomerID from selected row

        using (SqlConnection conn = new SqlConnection(connectionString))
        {
            string query = "DELETE FROM Customers WHERE CustomerID = @CustomerID";

            SqlCommand cmd = new SqlCommand(query, conn);
            cmd.Parameters.AddWithValue("@CustomerID", customerId);

            try
            {
                conn.Open();
                cmd.ExecuteNonQuery();
                MessageBox.Show("Customer deleted successfully.");
                LoadData();  // Refresh the DataGridView after deleting
            }
            catch (Exception ex)
            {
                MessageBox.Show("Error: " + ex.Message);
            }
        }
    }
    else
    {
        MessageBox.Show("Please select a customer to delete.");
    }
}
```

Figure 29 - button delete customers

Explanation: The btnDelete_Click method deletes the customer record corresponding to the selected row's CustomerID from the database.

Searching for Customers:

It allows the clerk to search for customers by name or email.

```
// Method to search for customers based on name or email
1 reference
private void btnSearch_Click(object sender, EventArgs e)
{
    string searchQuery = txtSearch.Text;

    try
    {
        using (SqlConnection conn = new SqlConnection(connectionString))
        {
            string query = "SELECT * FROM Customers WHERE Name LIKE @SearchQuery OR Email LIKE @SearchQuery";
            SqlDataAdapter adapter = new SqlDataAdapter(query, conn);
            adapter.SelectCommand.Parameters.AddWithValue("@SearchQuery", "%" + searchQuery + "%");
            DataTable dataTable = new DataTable();

            conn.Open();
            adapter.Fill(dataTable);

            dgvCustomer.DataSource = dataTable;  // Display search results in DataGridView
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show("Error searching data: " + ex.Message);
    }
}
```

Figure 30 - button search customers

Explanation: The btnSearch_Click method searches for customers in the database by matching their name or email with the search term entered by the clerk. The results are displayed in the DataGridView.

2.4 Sales Transaction

Adding a Book to the Bill:

This allows the clerk to add a selected book to the sale, applying any discounts and updating the total bill.

```
1 reference
private void btnAddToBill_Click(object sender, EventArgs e)
{
    // Get the values from the form fields
    string bookTitle = txtBookName.Text;
    decimal bookPrice = Convert.ToDecimal(txtPrice.Text);
    int quantity = Convert.ToInt32(txtQuantity.Text);
    decimal discountPercent = Convert.ToDecimal(txtDiscount.Text);

    // Calculate the discount amount (as a percentage of the total price)
    decimal discountAmount = (bookPrice * quantity) * (discountPercent / 100);

    // Calculate the total price after discount
    decimal totalPrice = (bookPrice * quantity) - discountAmount;

    int bookId = Convert.ToInt32(dgvBook.SelectedRows[0].Cells["BookID"].Value);

    dgvBill.Rows.Add(bookId, bookTitle, bookPrice, quantity, discountPercent.ToString("F2") + "%", totalPrice); // Add discountPercent

    // Update the total amount
    totalAmount += totalPrice;
    txtTotal.Text = totalAmount.ToString("F2");
}
```

Figure 31 - Button Add to Bill

Explanation: The btnAddToBill_Click method collects the book details, such as the title, price, quantity, and discount, from the form fields. It calculates the discount amount and then calculates the final total for that book. After that, it adds the book to the dgvBill DataGridView, updating the total bill amount. The totalAmount variable keeps track of the total price of all books added, which is displayed in the txtTotal textbox.

Saving the Sale:

This method saves the sale transaction to the database, including the sale details and the inventory update.

```
private bool SaveSale()
{
    using (SqlConnection conn = new SqlConnection(connectionString))
    {
        conn.Open();
        SqlTransaction transaction = conn.BeginTransaction();

        try
        {
            if (!decimal.TryParse(txtDiscount.Text, out decimal discount))
            {
                MessageBox.Show("Invalid discount format.");
                return false;
            }

            if (!decimal.TryParse(txtTotal.Text, out decimal totalAmount))
            {
                MessageBox.Show("Invalid total amount format.");
                return false;
            }

            // Insert sale record
            SqlCommand cmd = new SqlCommand("INSERT INTO Sales (CustomerID, Discount, Total) VALUES (@CustomerID, @Discount, @Total);
            cmd.Parameters.AddWithValue("@CustomerID", cmbCustomerName.SelectedValue ?? DBNull.Value);
            cmd.Parameters.AddWithValue("@Discount", discount);
            cmd.Parameters.AddWithValue("@Total", totalAmount);
            int saleId = Convert.ToInt32(cmd.ExecuteScalar()); // Capture SaleID

            // Insert sale items and update inventory
            foreach (DataGridViewRow row in dgvBill.Rows)
            {
                if (row.IsNewRow) continue;

                // Ensure BookID is valid
```

Figure 32 - Save Sales Function

```
    {
        if (row.IsNewRow) continue;

        // Ensure BookID is valid
        if (row.Cells["BookID"].Value == null || !int.TryParse(row.Cells["BookID"].Value.ToString(), out int bookId))
        {
            MessageBox.Show("Invalid Book ID.");
            return false;
        }

        // Validate Quantity and Price
        if (!int.TryParse(row.Cells["Quantity"].Value.ToString(), out int quantity))
        {
            MessageBox.Show("Invalid Quantity format.");
            return false;
        }

        if (!decimal.TryParse(row.Cells["Price"].Value.ToString(), out decimal price))
        {
            MessageBox.Show("Invalid Price format.");
            return false;
        }

        decimal total = price * quantity; // Calculate the row's total

        // Insert each sale item
        SqlCommand itemCmd = new SqlCommand("INSERT INTO SaleDetails (SaleID, BookID, Quantity, Price, Total) VALUES (@SaleID,
        itemCmd.Parameters.AddWithValue("@SaleID", saleId);
        itemCmd.Parameters.AddWithValue("@BookID", bookId);
        itemCmd.Parameters.AddWithValue("@Quantity", quantity);
        itemCmd.Parameters.AddWithValue("@Price", price);
        itemCmd.Parameters.AddWithValue("@Total", total);
        itemCmd.ExecuteNonQuery();
```

Figure 33 - Save Sales Function 02

Explanation: The SaveSale method performs the entire sale transaction. First, it validates the discount and total amount. Then it inserts the sale record into the Sales table and stores each item (book) in the SaleDetails table, ensuring the SaleID is associated with the correct book. Before updating the stock, it checks whether there is enough stock available for each book. If any book has insufficient stock, an error message is displayed. If everything is successful, the transaction is committed, and the sale is saved to the database. If there's an error, the transaction is rolled back, and the error is shown.

Printing the Receipt:

What it does: It generates and displays a receipt for the completed sale, showing the customer and sale details.

```
1 reference
private void btnPrint_Click(object sender, EventArgs e)
{
    // First, save the sale before printing the receipt
    bool saleSaved = SaveSale();  // Automatically saves the sale when print button is clicked

    if (!saleSaved)
    {
        MessageBox.Show("There was an error saving the sale. Please try again.");
        return;
    }

    // Now, generate and display the receipt
    StringBuilder receipt = new StringBuilder();

    // Store details
    receipt.AppendLine("Book Heaven Store - Receipt");
    receipt.AppendLine("----------------------------------------");

    // Access customer name from the ComboBox's selected item
    DataRowView selectedCustomer = cmbCustomerName.SelectedItem as DataRowView;
    if (selectedCustomer != null)
    {
        string customerName = selectedCustomer["Name"].ToString();
        receipt.AppendLine($"Customer: {customerName}");
    }

    // Customer details
    receipt.AppendLine($"Phone: {txtCusPhone.Text}");
    receipt.AppendLine($"Email: {txtCusEmail.Text}");
    receipt.AppendLine($"Address: {txtCusAddress.Text}");
    receipt.AppendLine("----------------------------------------");

    // Bill items
    foreach (DataGridViewRow row in dgvBill.Rows)
```

Figure 34 - Print Receipt

Explanation: The btnPrint_Click method is responsible for generating a printable receipt for the completed sale. First, it ensures the sale is saved by calling the SaveSale method. If the sale is successfully saved, it generates a receipt by appending the customer details and each item in the bill to a StringBuilder. The receipt is then displayed in a message box with the total amount.

3. Architecture Diagram, ER Diagram, and UML Diagrams

Architectural Diagram

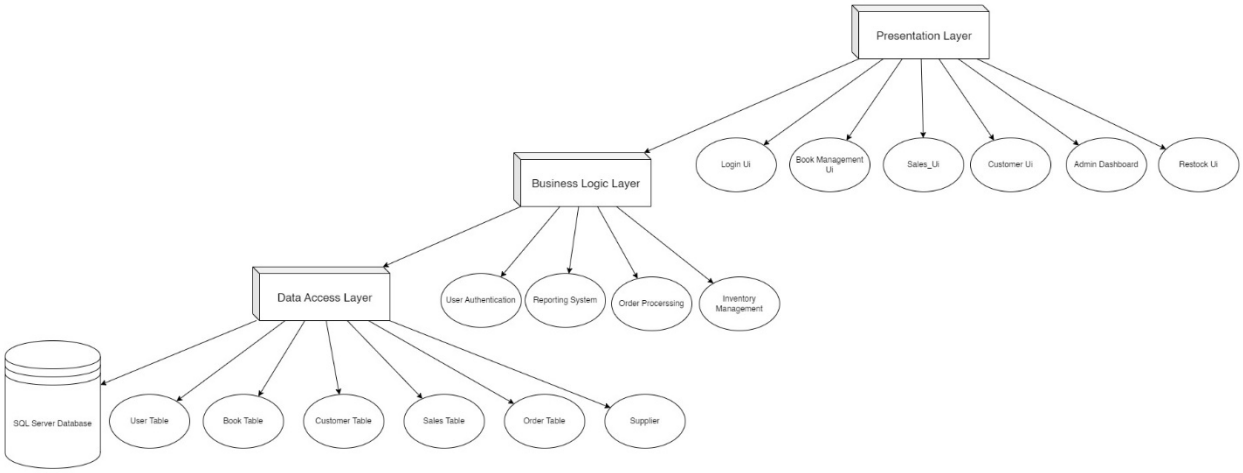


Figure 35 - Architectural Diagram

ER Diagram

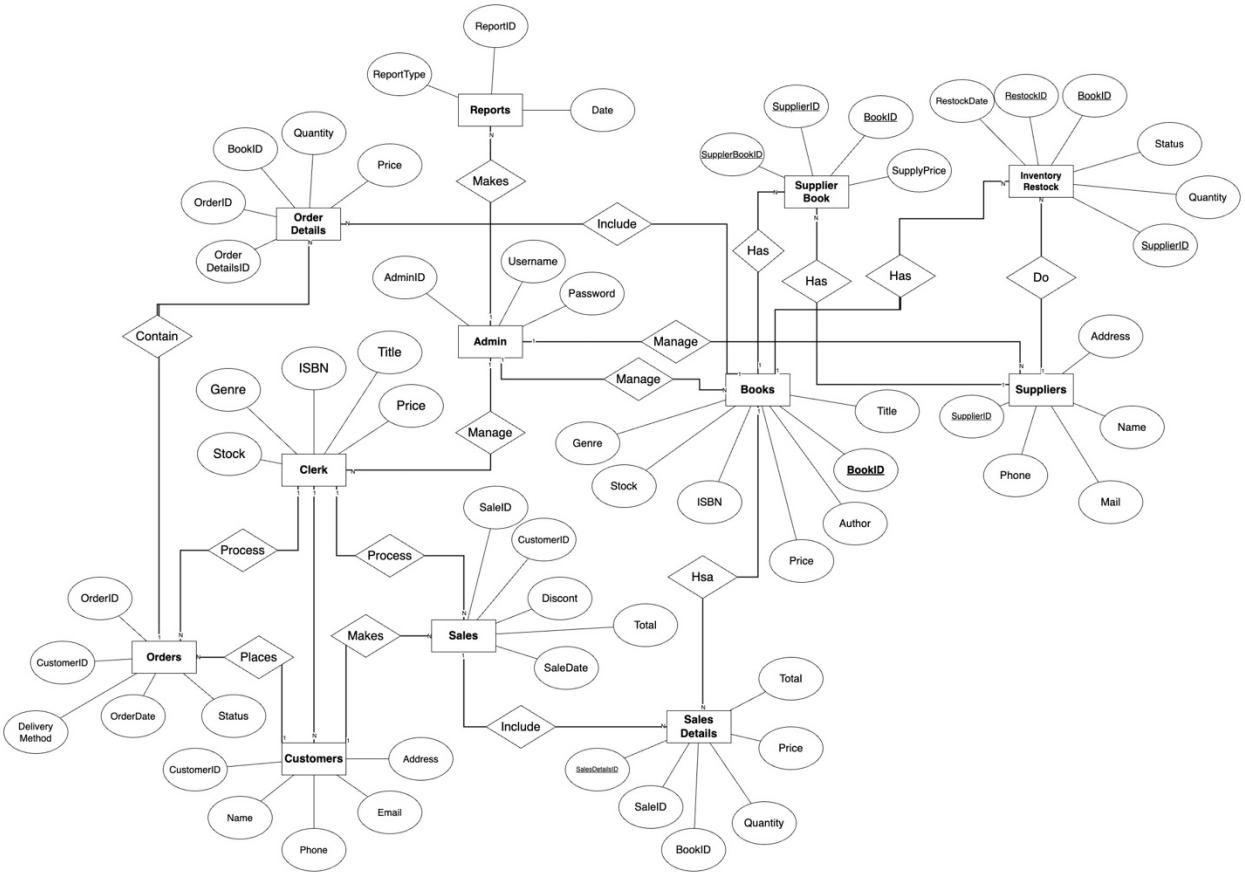


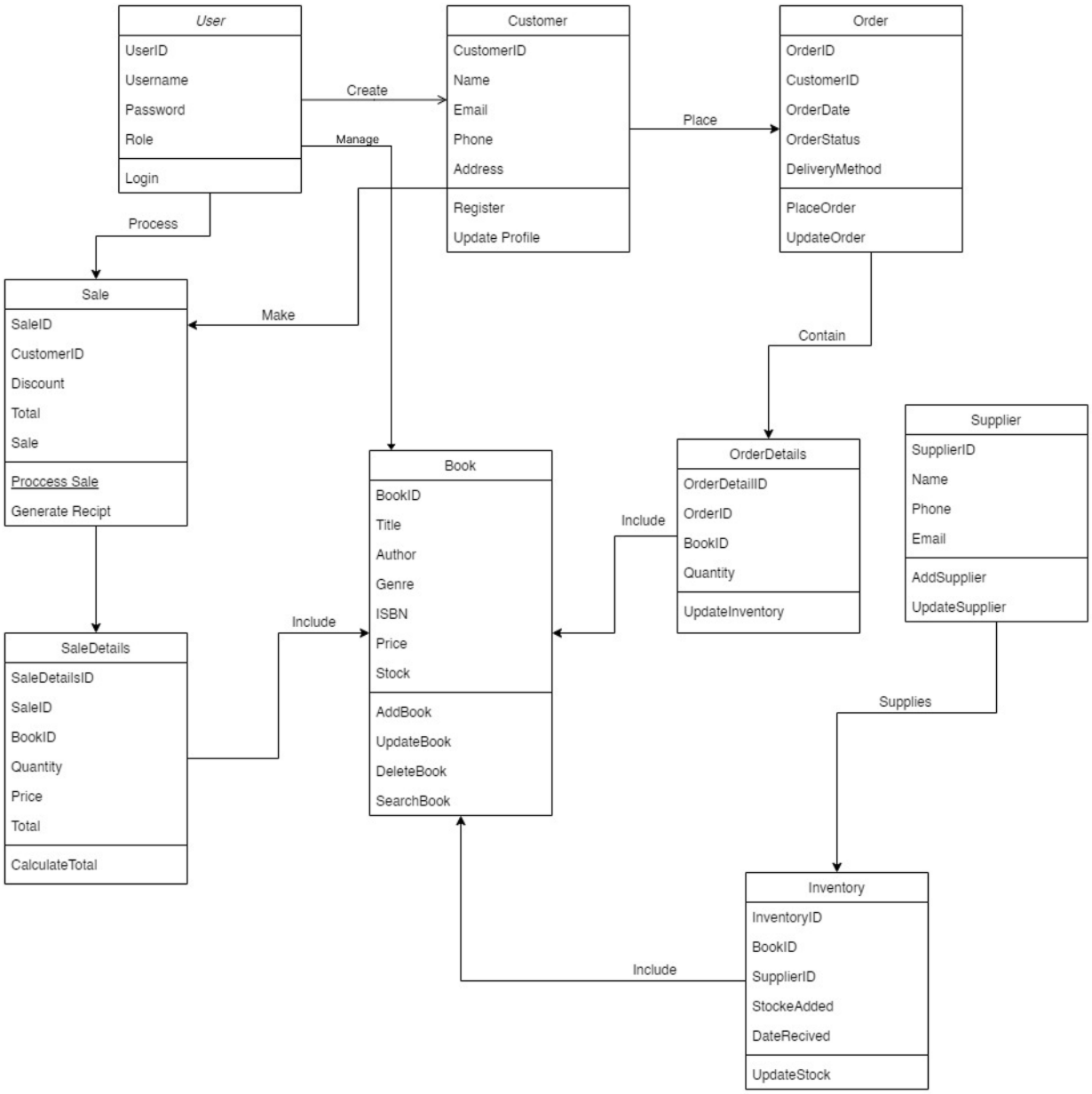
Figure 36 - ER Diagram

UML Diagrams

Use Case Diagram:

Figure 37 - Use Case Diagram

Class Diagram



4. Detailed Description of Classes, Properties, and Methods:

Class: `Admin_Books`

The `Admin_Books` class represents a Windows Forms page that allows the administrator to manage the bookstore's inventory of books. The functionalities provided include adding, updating, deleting, searching for books, and managing other aspects of the bookstore such as navigating to other forms.

Properties

1. connectionString (`string`)

- Type: `string`
- Description: Holds the connection string used to connect to the SQL Server database (`BookHeaven`). It defines the server, database, and the authentication mode used.

Methods

1. Constructor (`Admin_Books()`)

- Description: This is the constructor for the `Admin_Books` form. It initializes the form and binds the form load event (`Admin_Book_Load`) and the `CellClick` event for the `dgvBook` DataGridView to handle user interactions.
- Usage: Called when the form is instantiated.
- Parameters: None.

2. Admin_Book_Load(object sender, EventArgs e)

- Description: This event handler is called when the form is loaded. It calls `LoadData()` to populate the DataGridView with book data.
- Usage: Triggered on form load.

3. LoadData()

- Description: This method fetches all book records from the database and populates the `DataGridView` (`dgvBook`) with the retrieved data.
- Usage: Called on form load and after adding, updating, or deleting a book.
- Parameters: None.
- SQL Query: `"SELECT  FROM Books"`

4. dgvBook_CellContentClick(object sender, DataGridViewCellEventArgs e)

- Description: This method handles the event when a cell in the `dgvBook` DataGridView is clicked. It loads the clicked row's data into the textboxes and combo box (`txtBookTitle`, `txtAuthor`, `cmbGenre`, etc.) for editing.
- Usage: Triggered when a cell in the DataGridView is clicked.
- Parameters:
    - `sender`: The object that raised the event.
    - `e`: Provides event data, such as the row and column index of the clicked cell.

5. btnDash_Click(object sender, EventArgs e)

- Description: This method opens the `Admin_Dash` form and hides the current `Admin_Books` form.
- Usage: Triggered by the "Dashboard" button click.
- Parameters: `sender` and `e` (standard event arguments).

6. btnClerk_Click(object sender, EventArgs e)

- Description: This method opens the `Admin_Clerk` form and hides the current form.
- Usage: Triggered by the "Clerk" button click.
- Parameters: `sender` and `e`.

7. btnSupplier_Click(object sender, EventArgs e)

- Description: This method opens the `Admin_Suppliers` form and hides the current form.
- Usage: Triggered by the "Suppliers" button click.
- Parameters: `sender` and `e`.

8. btnInventory_Click(object sender, EventArgs e)

- Description: This method opens the `Admin_Books` form (current form) and hides the current form. This can be used to refresh or reopen the current form.
- Usage: Triggered by the "Inventory" button click.
- Parameters: `sender` and `e`.

9. btnReport_Click(object sender, EventArgs e)

- Description: This method opens the `Admin_Reports` form and hides the current form.
- Usage: Triggered by the "Reports" button click.
- Parameters: `sender` and `e`.

10. btnSettings_Click(object sender, EventArgs e)

- Description: This method opens the `Admin_Settings` form and hides the current form.
- Usage: Triggered by the "Settings" button click.
- Parameters: `sender` and `e`.

11. btnLogout_Click(object sender, EventArgs e)

- Description: This method opens the `Login` form and hides the current form, effectively logging the admin out.
- Usage: Triggered by the "Logout" button click.
- Parameters: `sender` and `e`.

12. btnClose_Click(object sender, EventArgs e)

- Description: This method closes the application when the "Close" button is clicked.
- Usage: Triggered by the "Close" button click.
- Parameters: `sender` and `e`.

13. btnAdd_Click(object sender, EventArgs e)

- Description: This method adds a new book to the database by extracting information from the textboxes and combo box (e.g., `txtBookTitle`, `txtAuthor`, `txtPrice`, etc.). The data is then inserted into the `Books` table in the database.
- Usage: Triggered by the "Add" button click.
- Parameters: `sender` and `e`.
- SQL Query: `"INSERT INTO Books (Title, Author, Genre, ISBN, Price, Stock) VALUES (@Title, @Author, @Genre, @ISBN, @Price, @Stock)"`

14. btnUpdate_Click(object sender, EventArgs e)

- Description: This method updates an existing book's details based on the selected row in the `dgvBook` DataGridView. The changes are then saved to the database.
- Usage: Triggered by the "Update" button click.
- Parameters: `sender` and `e`.
- SQL Query: `"UPDATE Books SET Title = @Title, Author = @Author, Genre = @Genre, ISBN = @ISBN, Price = @Price, Stock = @Stock WHERE BookID = @BookID"`

15. btnDelete_Click(object sender, EventArgs e)

- Description: This method deletes the selected book from the database by using the `BookID` from the selected row in the DataGridView.
- Usage: Triggered by the "Delete" button click.
- Parameters: `sender` and `e`.
- SQL Query: `"DELETE FROM Books WHERE BookID = @BookID"`

16. btnClear_Click(object sender, EventArgs e)

- Description: This method clears all input fields (textboxes and combo box) in the form to reset the form's state.
- Usage: Triggered by the "Clear" button click.
- Parameters: `sender` and `e`.

17. btnSearch_Click(object sender, EventArgs e)

- Description: This method searches for books based on the input provided in the `txtSearch` textbox. It retrieves the results from the database and displays them in the `dgvBook` DataGridView.
- Usage: Triggered by the "Search" button click.
- Parameters: `sender` and `e`.
- SQL Query: `"SELECT  FROM Books WHERE Title LIKE @SearchQuery OR Author LIKE @SearchQuery"`

18. btnBookOrder_Click(object sender, EventArgs e)

- Description: This method opens the `Admin_Restock` form for placing book orders and hides the current form.
- Usage: Triggered by the "Book Order" button click.
- Parameters: `sender` and `e`.

Summary of the Structure

The `Admin_Books` class is a Windows Form that facilitates managing a bookstore's inventory through a series of UI elements such as textboxes, combo boxes, and buttons. These components interact with a SQL Server database to perform CRUD (Create, Read, Update, Delete) operations on the books.

- The DataGridView (`dgvBook`) is used to display the book data.
- Buttons handle user interactions for adding, updating, deleting, and searching for books.
- The SQL Server connection is established using `SqlConnection`, and commands are executed with `SqlCommand` for interacting with the database.

Class: `Admin_Restock`

The `Admin_Restock` class represents a Windows Forms page that allows the administrator to manage restock orders for the bookstore. The functionalities include marking restock orders as received or canceled, placing new restock orders, and refreshing the list of pending, received, or canceled restock orders.

Properties

1. connectionString (`string`)

- Type: `string`
- Description: Holds the connection string used to connect to the SQL Server database (`BookHeaven`). It specifies the server, database, and authentication mode used to interact with the database.

Methods

1. Constructor (`Admin_Restock()`)

- Description: The constructor initializes the form and hooks up event handlers for form load (`RestockOrderForm_Load`) and other UI interactions. The form loads initial data like pending orders and available books for restock.
- Usage: Called when the form is instantiated.
- Parameters: None.

2. RestockOrderForm_Load(object sender, EventArgs e)

- Description: This event handler is called when the form is loaded. It refreshes the list of orders displayed in the DataGridView by calling the `RefreshOrders()` method.
- Usage: Triggered on form load to refresh the order list.
- Parameters:

- `sender`: The object that raised the event (in this case, the form).
- `e`: Event data containing additional information about the event.

3. RefreshOrders()

- Description: This method fetches the current orders from the `InventoryRestock` table and displays them in the `dgvOrders` DataGridView. It includes orders with statuses "Pending", "Received", or "Canceled".
- Usage: Called when the form loads and after performing actions like marking an order as received or canceled.
- Parameters: None.

SQL Query:

```sql
SELECT ir.RestockID, s.Name AS SupplierName, b.Title AS BookTitle, ir.Quantity, ir.RestockDate, ir.Status
FROM InventoryRestock ir
JOIN Suppliers s ON ir.SupplierID = s.SupplierID
JOIN Books b ON ir.BookID = b.BookID
WHERE ir.Status IN ('Pending', 'Received', 'Canceled');
```

4. btnReceived_Click(object sender, EventArgs e)

- Description: This method is triggered when the "Mark as Received" button is clicked. It updates the status of the selected restock order to "Received" and updates the stock of the corresponding book in the `Books` table.
- Usage: Triggered by the "Received" button click.

- Parameters:
- `sender`: The object that raised the event.
- `e`: Event data containing additional information about the event.

5. btnCanceled_Click(object sender, EventArgs e)

- Description: This method is triggered when the "Mark as Canceled" button is clicked. It updates the status of the selected restock order to "Canceled".
- Usage: Triggered by the "Canceled" button click.
- Parameters:
- `sender`: The object that raised the event.
- `e`: Event data containing additional information about the event.

6. btnOrder_Click(object sender, EventArgs e)

- Description: This method is triggered when the "Place Order" button is clicked. It creates a new restock order with the selected book, supplier, and quantity. It inserts a new record into the `InventoryRestock` table.
- Usage: Triggered by the "Place Order" button click.
- Parameters:
- `sender`: The object that raised the event.
- `e`: Event data containing additional information about the event.

7. Form1_Load(object sender, EventArgs e)

- Description: This method loads the list of books with stock less than 5 into the `cmbBook` ComboBox and the list of suppliers into the `cmbSupplier` ComboBox.
- Usage: Triggered on form load to populate combo boxes with data for selecting books and suppliers.
- Parameters:
    - `sender`: The object that raised the event.
    - `e`: Event data containing additional information about the event.

8. btnBookOrder_Click(object sender, EventArgs e)

Description: This method opens the `Admin_Restock` form again and hides the current form. It ensures the user remains in the restock order section.

Usage: Triggered by the "Book Orders" button click.

Parameters: `sender` and `e` (standard event arguments).

Summary

The `Admin_Restock` form provides a user interface for managing restock orders in a bookstore's inventory. It allows the admin to:

1. Place restock orders for books with low stock.
2. Mark orders as received and update the inventory accordingly.
3. Cancel orders if necessary.
4. View a list of current orders and refresh the data when needed.

Each button on the form is associated with specific logic that interacts with the database to update or fetch data for the administrator. The combination of form events, data handling, and SQL queries ensures that the application works effectively for managing restocks in the bookstore.

Test Case

Test Case No: 01

Function (Login): If username, password and roles is correct then login should be successful

Test Data:

- Username: admin
- Password: 123

Expect Results: Admin Login Successfully

Actual Results: Admin Login Successfully

Status: Pass

Test Results (Evidence)

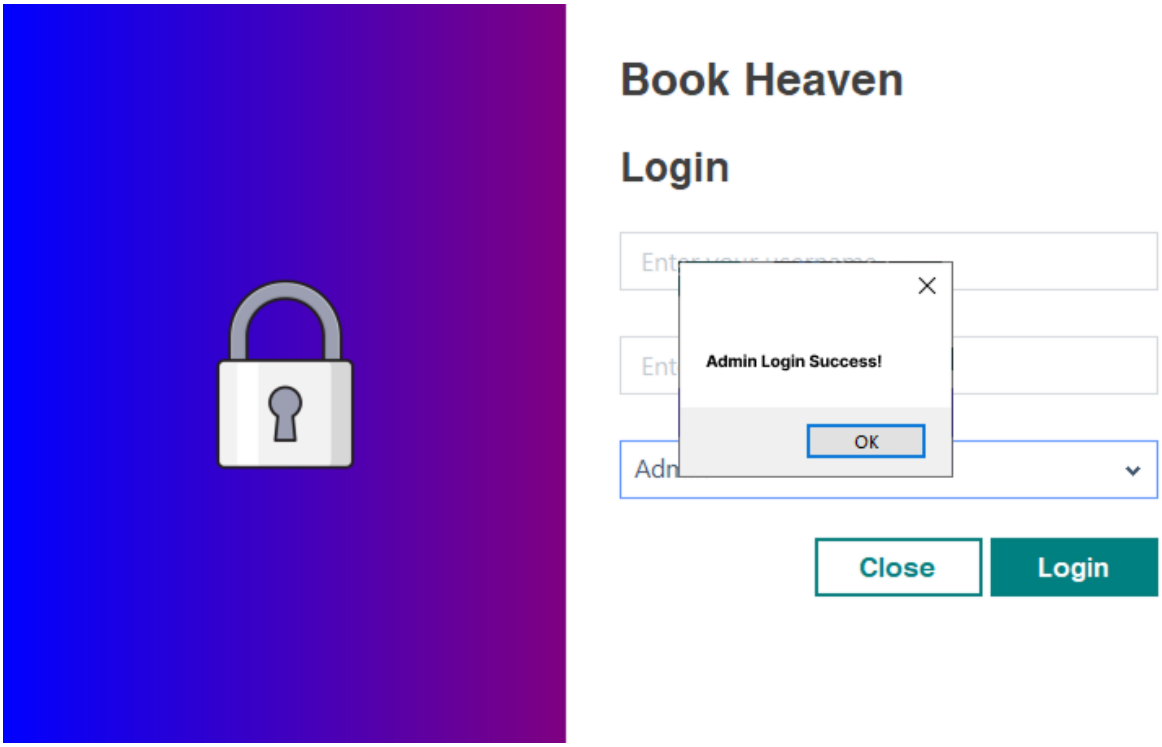


Figure 38 - Test Login Success

Test Case No: 02

Function (Login): If username, password and roles is incorrect then login should be unsuccessful

Test Data:

- Username: admin
- Password: 123456

Expect Results: Invalid username, password or role

Actual Results: Invalid username, password or role

Status: Pass

Test Results (Evidence)

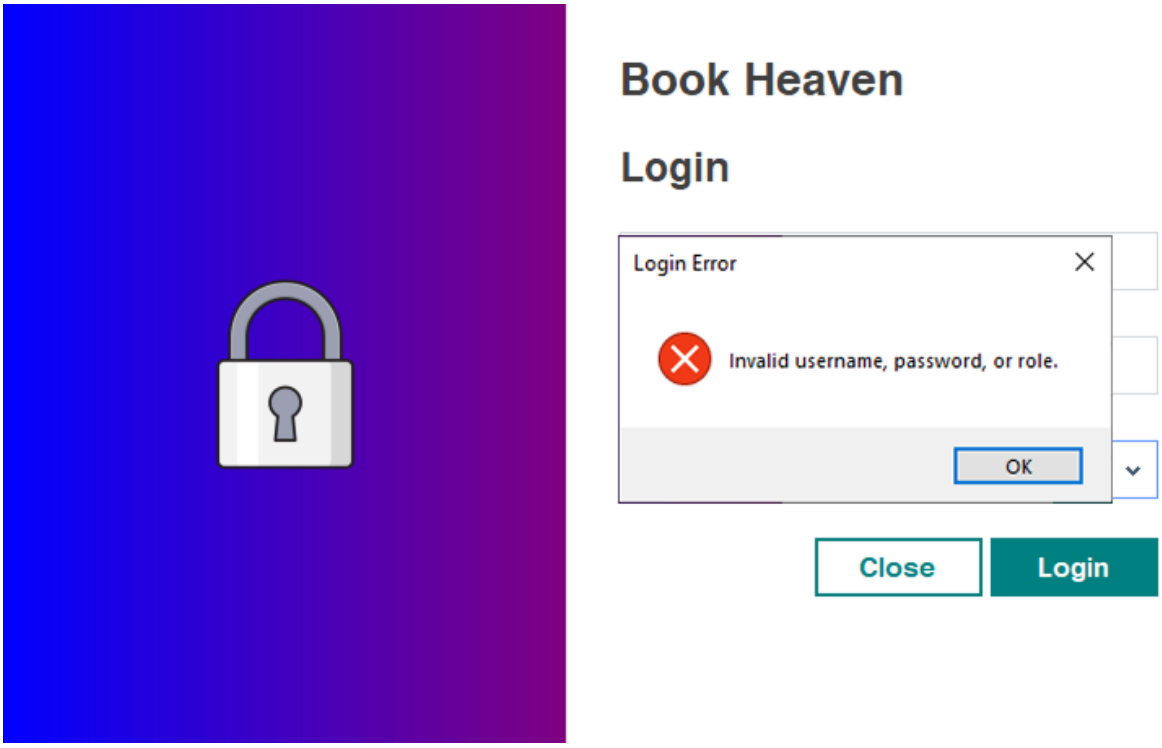


Figure 39 - Test Login Wrong

Test Case No: 03

Function: Add Book

Test Data:

- Name, Author, ISBN, Genre, Price, Stock

Expect Results: Book Added Successfully

Actual Results: Book Added Successfully

Status: Pass

Test Results (Evidence)

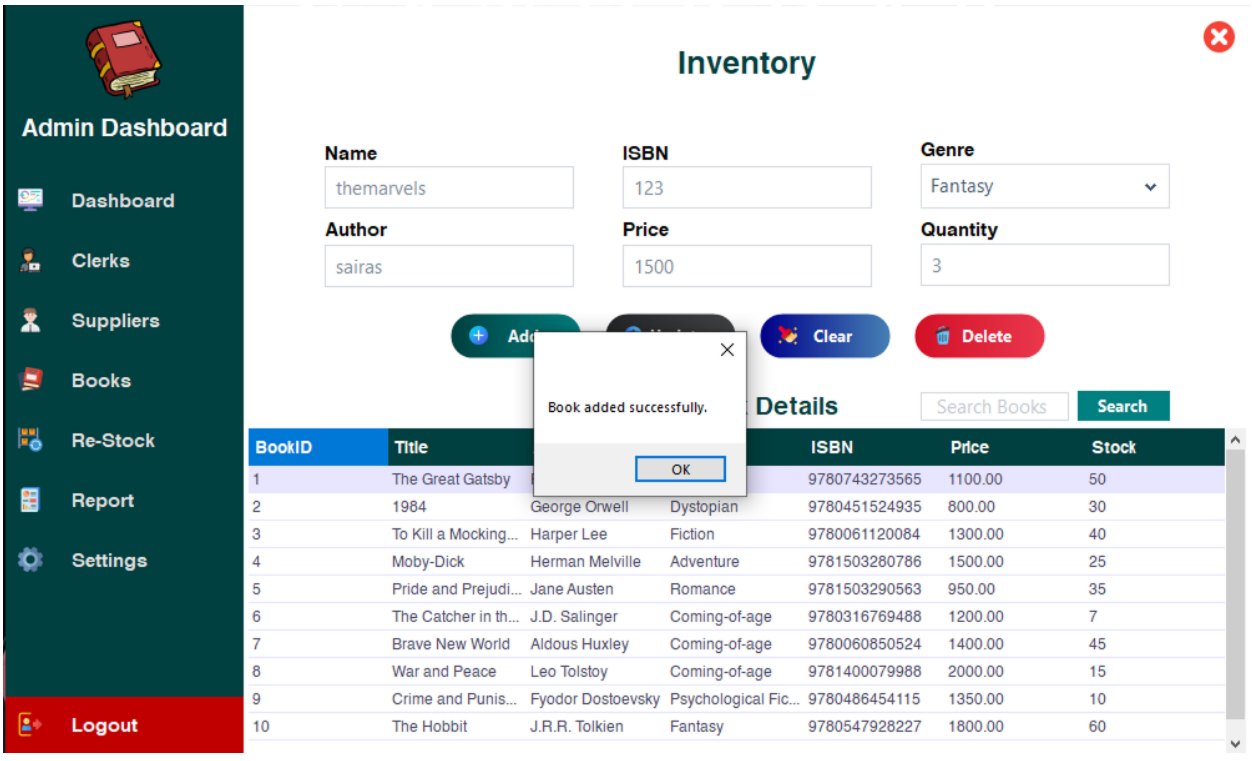


Figure 40 - Test Book Add

Test Case No: 04

Function: Book Update

Test Data:

- Price: Change Price 1500 to 2000

Expect Results: Book Update Successfully

Actual Results: Book Update Successfully

Status: Pass

Test Results (Evidence)

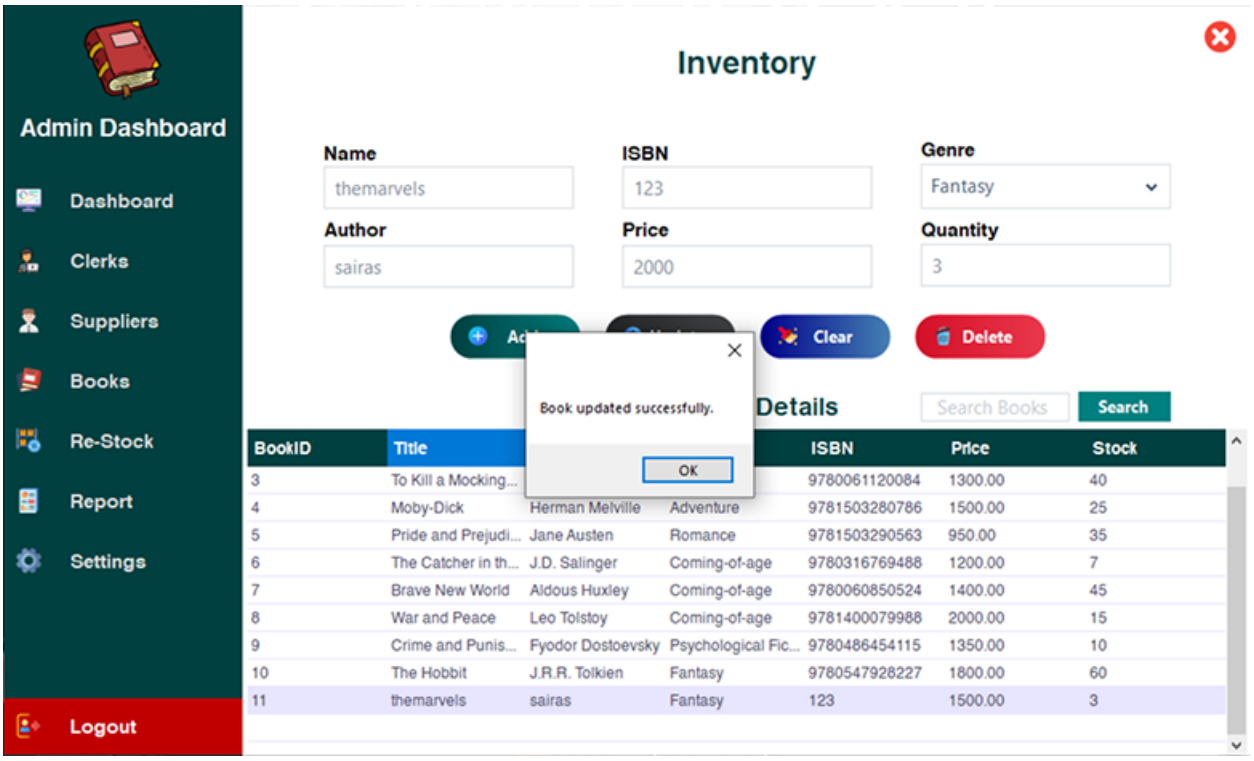


Figure 41 - Test Book Update

Test Case No: 05

Function: Re-Stock Book From Supplier

Test Data:

- Add 10 Quantity

Expect Results: Restock order created successfully.

Actual Results: Restock order created successfully.

Status: Pass

Test Results (Evidence)

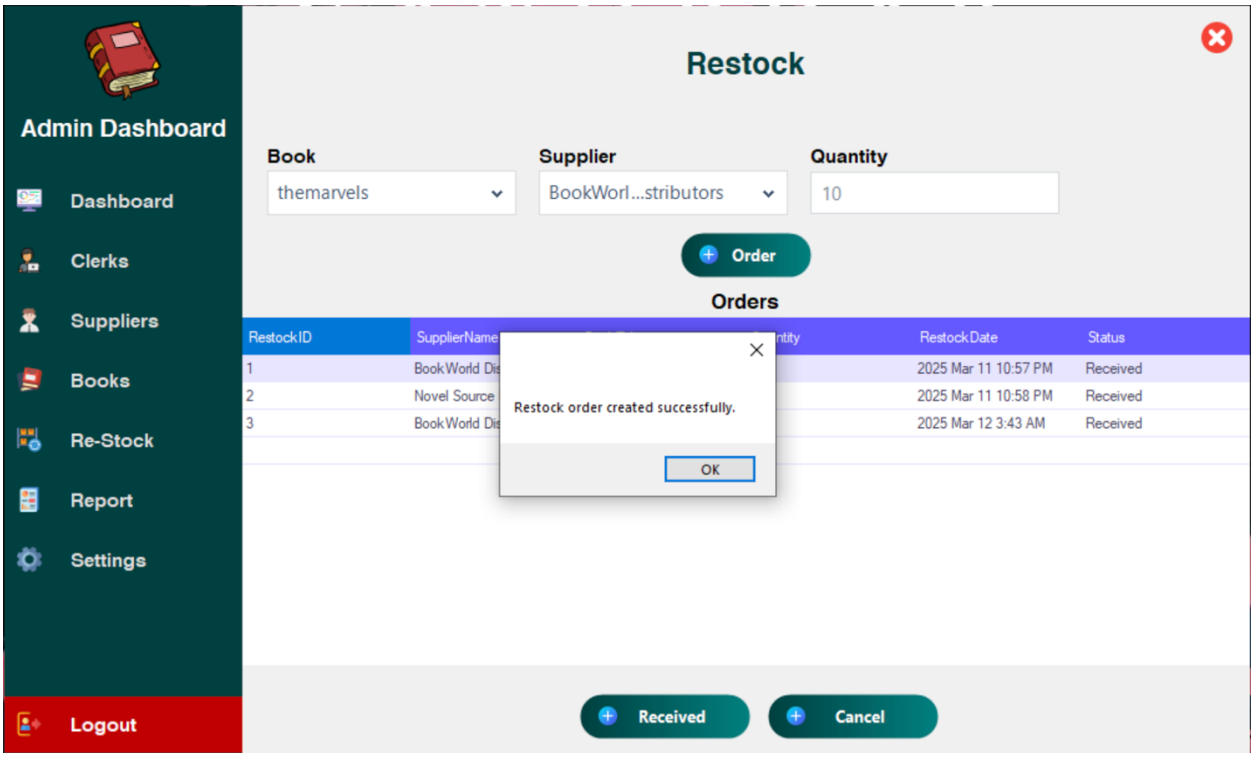


Figure 42 - Test Restock Order Success

Test Case No: 06

Function: Ordered Book Marked as Received

Test Data:

- If Book Received need to mark as ordered book received

Expect Results: Order Marked as Received

Actual Results: Order Marked as Received

Status: Pass

Test Results (Evidence)

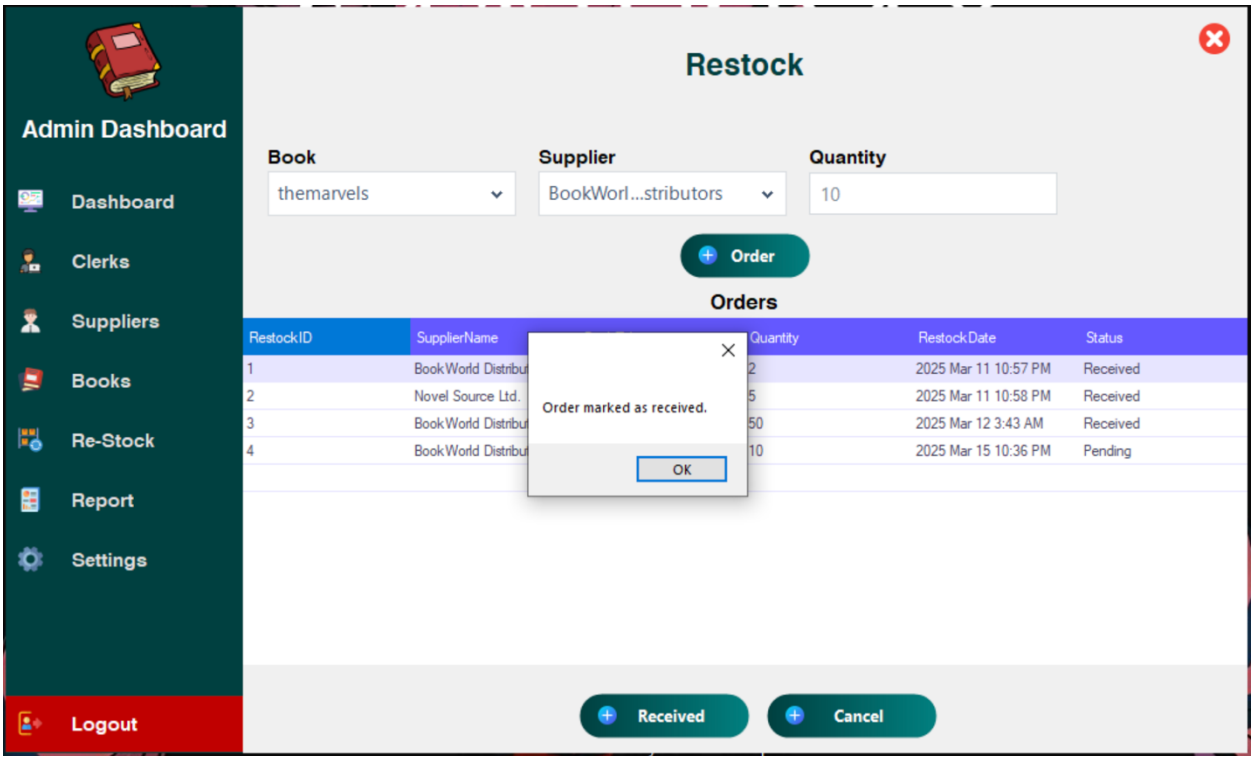


Figure 43 - Order marked as recieved

Test Case No: 07

Function: Sale Book

Test Data:

- Select Customer
- Select Book
- Quantity
- If Discount Have

Expect Results: the print receipt needs to show

Actual Results: the print receipt needs to show

Status: Pass

Test Results (Evidence)

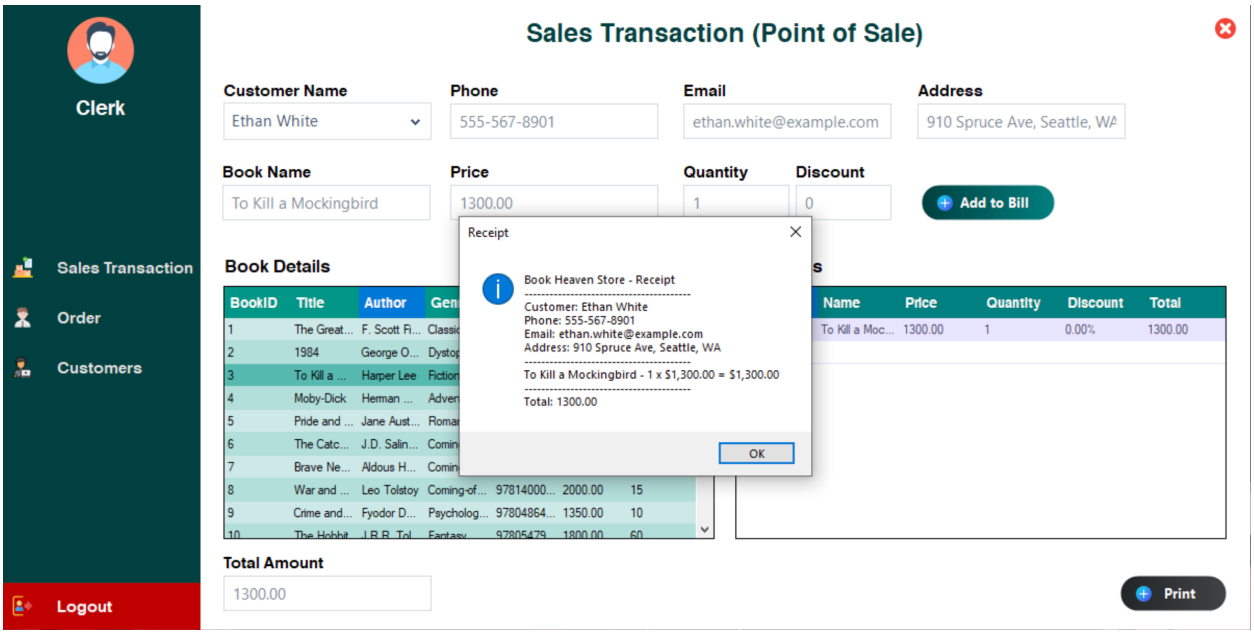


Figure 44 - Test Get Receipt

Test Case No: 08

Function: Order for Customer

Test Data:

- Select Customer
- Quantity
- Delivery Type: Pick-Up

Expect Results: Order saved Successfully

Actual Results: Order saved Successfully

Status: Pass

Test Results (Evidence)

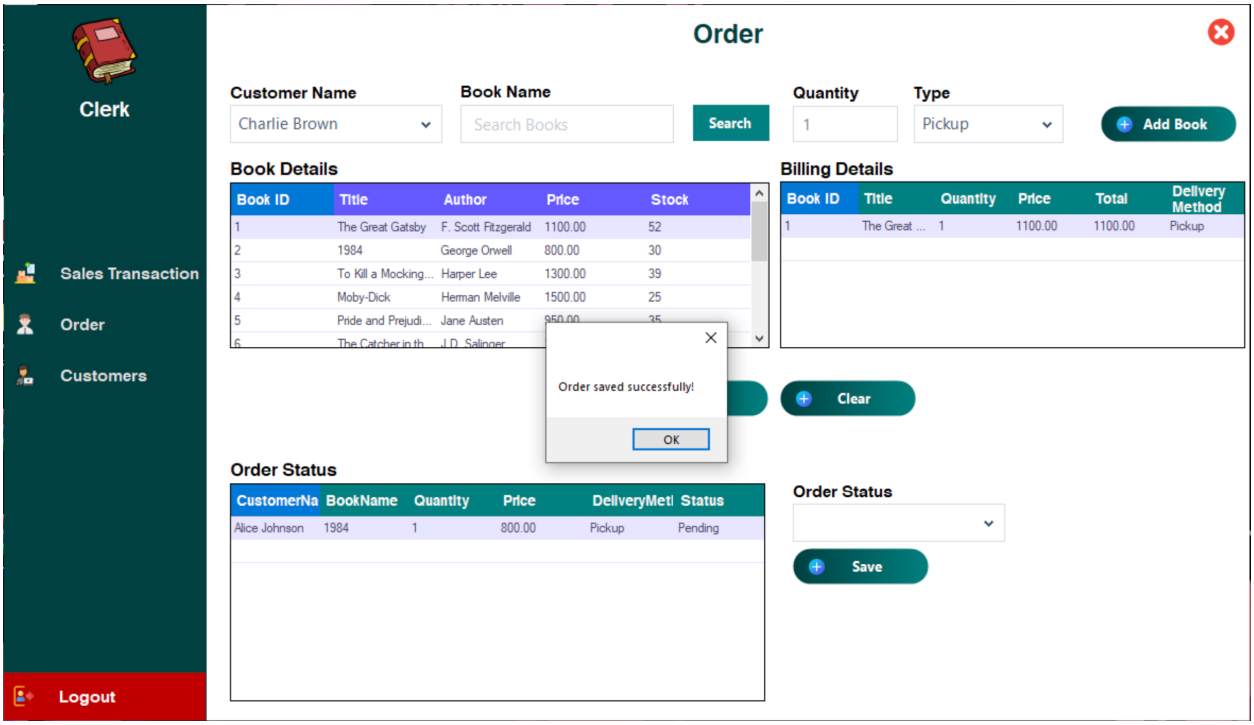


Figure 45 - Make Book Order for Customer

Test Case No: 09

Function: Add Customer

Test Data:

- Name: Sainas
- Email: gta@gmail.com
- Phone: 0751607200
- Address: Kinniya

Expect Results: Customer Added Successfully

Actual Results: Customer Added Successfully

Status: Pass

Test Results (Evidence)

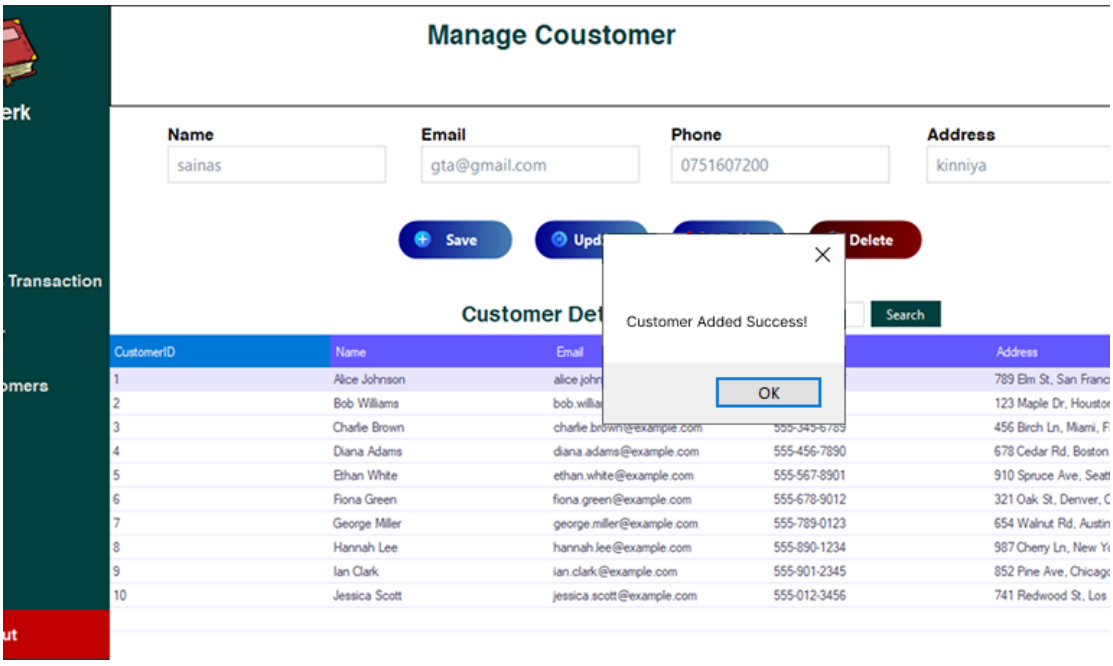


Figure 46 - Add Customer

Test Case No: 10

Function: Customer Update

Test Data:

- Change Name Ethan to Ethan White
-

Expect Results: Customer update successfully

Actual Results: Customer update successfully

Status: Pass

Test Results (Evidence)

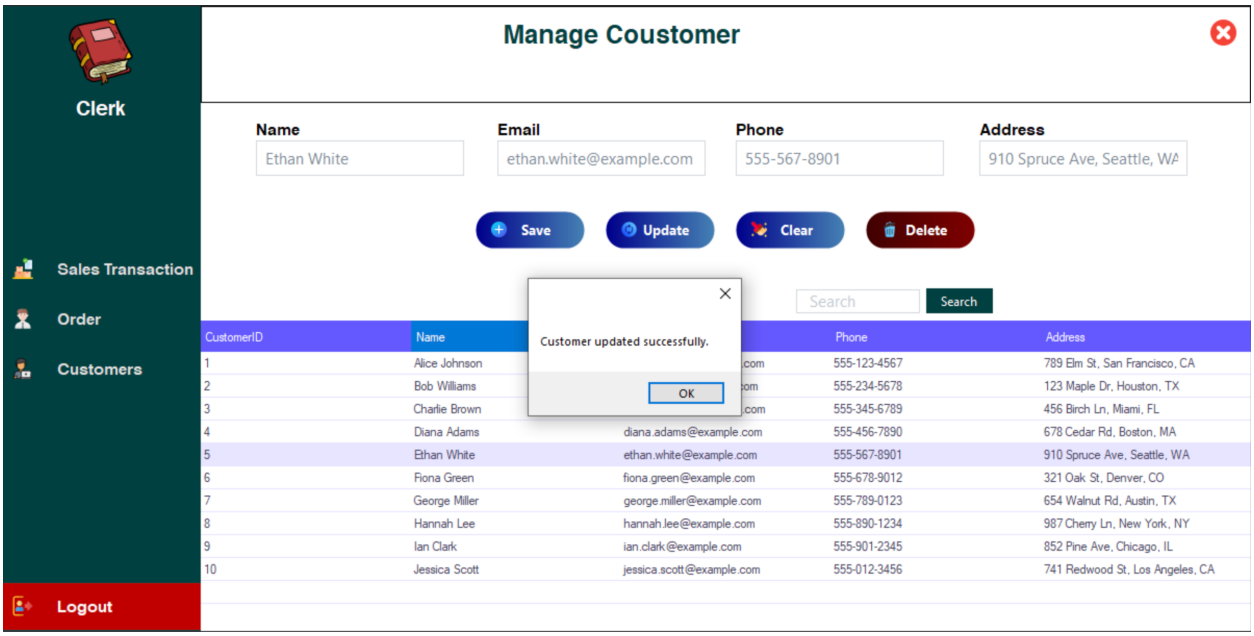


Figure 47 - Update Customer

Own Reflection on My Experience

Throughout the development of the BookHaven bookstore management system, I have gained a comprehensive understanding of both front-end and back-end development. The system includes various interfaces tailored to both Admins and Clerks, and my experience involved designing, coding, testing, and refining each part of the system. Here's a reflection on the key lessons and experiences I gained while working on this project.

1. Designing and Developing User Interfaces (UI)

One of the most challenging yet rewarding parts of this project was designing the interfaces for Admins and Clerks. Each user type had its own specific needs and functionalities, which required careful planning to ensure both usability and efficiency.

Admin Interfaces: The Admin role needs to manage core business functions, including overseeing books, suppliers, customers, restocking, and generating reports. This required creating interfaces like:

- Admin_Dash: The dashboard to overview the system.
- Admin_Book: A form to manage the bookstore's inventory.
- Admin_Clerk: An interface to handle staff and user management.
- Admin_Restock: The page for managing stock replenishment orders.
- Admin_Report: To generate and view reports.
- Admin_Customers: A section for managing customers.

- Clerk Interfaces: The Clerk role focuses more on sales and customer interactions. Interfaces I developed for this role include:
  - Clerk_Sales: Point of Sale (POS) interface where clerks process transactions.
  - Clerk_Customers: A page for customer management.
  - Clerk_Order: An interface for managing customer orders.

The process of creating these forms taught me the importance of simplicity and clarity in UI design. As both Admins and Clerks have different responsibilities, I ensured that their respective dashboards and pages were intuitive and well-organized.

2. Handling User Authentication

To ensure secure access to different parts of the system, I implemented user login functionality. The Login form validates user credentials, determining whether the user is an Admin or a Clerk, and then redirects them to their respective interfaces.

This aspect of the project made me realize how important security and role-based access control are in real-world applications. Ensuring that only authorized users have access to certain functionalities is crucial for protecting sensitive data and ensuring smooth system operations.

3. Backend Integration and Database Interaction

The project heavily relied on SQL Server for managing data, and the interaction between the front-end (UI) and back-end (database) was vital for ensuring that operations like managing books, orders, inventory, and customers worked seamlessly. Here are a few important backend features I implemented:

- Admin_Books Management: Adding, updating, and deleting books from the inventory required interacting with the database to store and retrieve book data efficiently.
- Admin_Restock: Restock orders were placed and tracked, requiring me to create queries that would update both the restock records and the inventory data (like updating stock levels).

- Admin_Report: Reports had to aggregate and summarize data across different tables (e.g., sales, inventory) to provide meaningful insights. Writing efficient SQL queries for this was both challenging and rewarding.
- Clerk_Sales: At the core of the Clerk functionality was managing customer purchases in real-time. This involved dynamic interactions with the database to record sales, update inventory, and track customer details.

4. Event-Driven Programming and Forms Management

As the application uses Windows Forms, event-driven programming played a significant role. Every button click, form load, or data grid interaction triggered specific events. Handling these events correctly was crucial for user experience. For example:

- CRUD Operations: The Admin_Book and Clerk_Order interfaces involved creating, reading, updating, and deleting data. Each operation required careful validation to ensure that no erroneous data was entered.
- Data Binding: I used DataGridViews for displaying data like books, orders, and customers. Dynamically updating the grid after a CRUD operation was essential for a responsive UI.

5. Error Handling and Validation

I learned the importance of input validation and error handling in any system. Ensuring that the user cannot enter invalid data—such as non-numeric values in fields that expect numbers—was a key learning experience. I incorporated:

- Validation for Numeric Fields: Ensuring fields like stock quantity, price, and order quantities were entered correctly.

- Null Checks and Error Messages: If a user attempted to perform an operation without selecting the necessary data (like selecting an item or customer), I made sure to provide clear and helpful error messages.
- Transaction Management: In the case of restocking and order processing, ensuring database transactions were handled correctly was essential to maintain consistency and avoid data corruption.

6. Security Considerations

Another major aspect I paid attention to was ensuring secure communication with the database. By using parameterized queries, I was able to protect the application from SQL injection attacks. Additionally, I thought about role-based access control to ensure that only authorized users could perform certain operations. The Admin role had access to sensitive information and system-wide functionalities, while Clerks only had access to sales-related data and customer interactions.

7. User Experience (UX) Design

Designing the user experience for both the Admin and Clerk roles helped me appreciate the importance of intuitive navigation and workflow. Users should always know where they are in the application and what action they can take next. Through feedback loops such as confirmation dialogs and error messages, I worked to ensure the system was both informative and user-friendly.

8. Testing and Debugging

During the development of this system, thorough testing was crucial. I tested various operations, such as adding books, processing sales, and handling restock orders, to ensure they were functioning as expected. Debugging complex interactions between the database and the user interface, especially when handling edge cases (like a failed database connection or empty fields), was challenging but provided valuable insights.

9. Lessons Learned

- Importance of Planning: The project reinforced the importance of planning the structure and flow of the application carefully before diving into code. Designing clear roles for users (Admins and Clerks) and their responsibilities helped ensure the system met all functional requirements.
- Seamless UI/UX: An intuitive user interface is essential, especially for employees like clerks who rely on fast, error-free transactions. The design of forms, data grids, and action buttons was essential to creating a smooth user experience.

- Database Optimization: I realized how important it is to write efficient SQL queries, particularly when dealing with large datasets. Optimizing database operations helped improve the performance of the application, especially for queries related to order processing and inventory updates.
- Error Prevention and User Validation: Effective error prevention and user input validation are key to preventing application crashes or unexpected behaviors. It's essential to think through edge cases and common user mistakes to design a resilient system.

10. Future Improvements

While the project is functional, I can see several areas for improvement:

- Automated Testing: Integrating automated tests, particularly for database operations, would improve the stability of the system.
- Enhanced Reporting: Admins could benefit from more advanced report generation, such as graphical data representations (charts, graphs) for inventory, sales, and restock history.
- Cloud Integration: Implementing cloud services for backup, scaling, and centralized database management could improve the application's reliability and accessibility.
- Mobile Access: Given the increasing reliance on mobile devices, making the system mobile-friendly or creating a mobile app for on-the-go access could be valuable.

Conclusion

Overall, this project provided me with a valuable opportunity to hone my skills in full-stack development, SQL database management, user interface design, and security best practices. The experience of creating a multi-role system with distinct functionalities for Admins and Clerks was both challenging and rewarding. As I reflect on the process, I feel more confident in my ability to develop robust and efficient desktop applications that can address real-world business needs.