**ASSIGNMENT 2 FRONT SHEET**

|  |  |  |  |
| --- | --- | --- | --- |
| **Qualification** | **BTEC Level 5 HND Diploma in Business** | | |
| **Unit number and title** | **Unit 30: Application Development** | | |
| **Submission date** |  | **Date Received 1st submission** |  |
| **Re-submission Date** |  | **Date Received 2nd submission** |  |
| **Student Name** | Truong Tan Phuc | **Student ID** | GCD210070 |
| **Class** | GCD1101 | **Assessor name** | Pham Thanh Son |
| **Student declaration**  I certify that the assignment submission is entirely my own work and I fully understand the consequences of plagiarism. I understand that making a false declaration is a form of malpractice. | | | |
|  |  | **Student’s signature** | A black letter with a curved line  Description automatically generated with medium confidence |

**Grading grid**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **P4** | **P5** | **P6** | **M3** | **M4** | **M5** | **D2** | **D3** |
|  |  |  |  |  |  |  |  |

|  |  |  |
| --- | --- | --- |
| **❒ Summative Feedback: ❒ Resubmission Feedback:** | | |
| **Grade:** | **Assessor Signature:** | **Date:** |
| **Internal Verifier’s Comments:** | | |
| **Signature & Date:** | | |

Table of Contents

[CHAPTER 1: PEER REVIEW AND FEEDBACK ANALYSIS 5](#_Toc154181914)

[1. Formal Questionnaire To Reviews The Business Application (P4) 5](#_Toc154181915)

[2. Collect Review Feedbacks (P4) 9](#_Toc154181916)

[3. Interpret Peer-Review Feedbacks (M3) 14](#_Toc154181917)

[3.1 Interpret Feedbacks 14](#_Toc154181918)

[3.2 Identify Opportunities 16](#_Toc154181919)

[CHAPTER 2: APPLICATION DEVELOPMENT 18](#_Toc154181920)

[1. Folder Structure Of The Application 18](#_Toc154181921)

[2. Source Code Samples Of The Application With Explanation 27](#_Toc154181922)

[2.1 Model 27](#_Toc154181923)

[2.2 Controller 44](#_Toc154181924)

[2.3 View 102](#_Toc154181925)

[3. Final Screenshots Of The Application 155](#_Toc154181926)

[4. Screenshots Of Using Github Or Gitlab To Manage The Source Code 175](#_Toc154181927)

[5. Screenshots Of Using IIS Or Azure For The Application Deployment 177](#_Toc154181928)

[CHAPTER 3: APPLICATION EVALUATION 178](#_Toc154181929)

[1. Review The Performance Of The Application (P6) 178](#_Toc154181930)

[2. Conclude Whether The Application Adapts All Requirements Or It Needs To Be Improved Later (P6) 197](#_Toc154181931)

[3. Analyse the factors that influence the performance of the application (M5) 198](#_Toc154181932)

# CHAPTER 1: PEER REVIEW AND FEEDBACK ANALYSIS

## 1. Formal Questionnaire To Reviews The Business Application (P4)

In order to comprehensively assess and enhance our business application, we have devised a formal questionnaire aimed at soliciting valuable feedback from users and stakeholders. This questionnaire is structured to delve into multiple aspects, including the user experience, identification of issues, proposed solutions, and insights into the strategic development of our application.

This initiative seeks to garner insights that will aid in refining our business application, ensuring it meets the needs and expectations of our diverse user base. By addressing specific queries related to user experiences, potential challenges, proposed solutions, and strategic developmental plans, we aim to gather a holistic understanding of the strengths and areas for improvement within our application.

A screenshot of a web project

Description automatically generated

A screenshot of a computer

Description automatically generated

Figure 1: Survey

A screenshot of a computer

Description automatically generated

Figure 2: Survey

A screenshot of a computer

Description automatically generated

Figure 3: Survey

A screenshot of a computer

Description automatically generated

Figure 4: Survey

A screenshot of a chat

Description automatically generated

Figure 5: Survey

## 2. Collect Review Feedbacks (P4)

As part of our ongoing commitment to enhancing your experience with our product/service, we are excited to launch a comprehensive review feedback initiative. Your insights and opinions are instrumental in shaping the future of our offerings, and this feedback collection process is designed to capture your valuable thoughts.

We understand the significance of user feedback in refining and optimizing our products/services. Your unique perspectives provide us with invaluable information that helps us address concerns, enhance features, and ensure that our offerings align seamlessly with your needs.

This feedback collection is not just a survey but a collaborative effort to build a product/service that exceeds your expectations. We invite you to share your experiences, suggestions, and any challenges you may have encountered. Your feedback will contribute directly to our ongoing commitment to delivering a superior and tailored experience.

Biểu đồ câu trả lời của biểu mẫu. Tên câu hỏi: 1. How do you feel about your experience using the ShopBee app?
. Số lượng câu trả lời: 16 câu trả lời.

Figure 6: Survey Result

Biểu đồ câu trả lời của biểu mẫu. Tên câu hỏi: 2. How do you feel about your experience using the ShopBee app?
. Số lượng câu trả lời: 16 câu trả lời.

Figure 7: Survey Result

Biểu đồ câu trả lời của biểu mẫu. Tên câu hỏi: 3. Does the ShopBee app deliver the features you expect?
. Số lượng câu trả lời: 16 câu trả lời.

Figure 8: Survey Result

Biểu đồ câu trả lời của biểu mẫu. Tên câu hỏi: 4. Do you feel secure about the safety and security of your personal information when using the ShopBee application?
. Số lượng câu trả lời: 16 câu trả lời.

Figure 9: Survey Result

Biểu đồ câu trả lời của biểu mẫu. Tên câu hỏi: 5. How would you rate ShopBee&apos;s user feedback and technical support process?
. Số lượng câu trả lời: 16 câu trả lời.

Figure 10: Survey Result

A screenshot of a chat

Description automatically generated

Figure 11: Survey Result

A screenshot of a chat

Description automatically generated

Figure 12: Survey Result

A screenshot of a chat

Description automatically generated

Figure 13: Survey Result

## 3. Interpret Peer-Review Feedbacks (M3)

### 3.1 Interpret Feedbacks

**Q1. Problems Encountered:**

Feedback: Users faced difficulties in setting up their store, navigating the product addition interface, slow loading times for page images, and challenges in accessing customer support and editing shipping details post-purchase.

Interpretation: There are usability issues and performance concerns affecting the overall user experience. Improvements in onboarding, navigation, performance optimization, and customer support responsiveness are needed.

**Q2. Standout Features:**

Feedback: Users appreciate the efficient search functionality, user-friendly dashboard, promotional tools, seamless integration with e-book formats, analytics dashboard, and collaborative storefront feature.

Interpretation: Positive feedback highlights strong aspects of the application, including search functionality, dashboard usability, promotional capabilities, and collaborative features, which can be leveraged as key selling points.

**Q3. Suggestions for Improvement and New Features:**

Feedback: Users suggest adding promotional discount features, a customer review system, personalized storefronts, pre-order functionality, and multilingual support for global accessibility.

Interpretation: Users are interested in additional features that enhance marketing capabilities, credibility, customization, and global reach. Implementing these suggestions can boost user engagement and satisfaction.

**Q4. Feedback on User Experience through the Interface:**

Feedback: Users suggest streamlining the checkout process, adding tooltips for first-time users, optimizing the mobile app, and providing a progress indicator during the checkout process.

Interpretation: Users seek improvements in interface efficiency, mobile accessibility, and user guidance. Enhancements in these areas can lead to a more seamless and intuitive user experience.

**Q5. Thoughts on Future Development Strategy:**

Feedback: Users recommend integrating with social media, enhancing mobile responsiveness, supporting audiobook formats, expanding to international markets, considering eco-friendly options, and exploring partnerships and digital wallet integration.

Interpretation: Users provide insights into potential strategic directions, including social media integration, mobile optimization, global expansion, environmental considerations, and partnerships, aligning the application with industry trends.

**Overall Interpretation:**

The feedback indicates a mix of positive aspects and areas for improvement in user experience, performance, and feature set. Key opportunities include enhancing onboarding, performance optimization, marketing tools, global accessibility, and aligning with emerging trends such as sustainability and digital wallets.

**Next Steps:**

Prioritize addressing usability issues and performance concerns.

Consider implementing suggested features and improvements.

Leverage positive feedback as marketing points.

Engage with users to gather more specific insights.

Continuously iterate based on user feedback for ongoing improvement.

This feedback provides valuable insights for refining and advancing your project, ensuring it aligns closely with user expectations and industry trends.

### 3.2 Identify Opportunities

**Personalized User Dashboards:**

Opportunity: Enhance the user dashboard by providing customizable and personalized elements. Allow users to arrange widgets or modules based on their preferences, providing a more tailored and user-centric dashboard experience.

**Predictive Analytics for Inventory Management:**

Opportunity: Implement predictive analytics for inventory management. By analyzing past sales data and trends, the system could provide recommendations on optimal stock levels, reducing the likelihood of stockouts or overstock situations.

**Voice Search Functionality:**

Opportunity: Integrate voice search functionality within the application. This feature can cater to users who prefer hands-free navigation and enhance the overall accessibility of your platform.

**Augmented Reality (AR) for Product Visualization:**

Opportunity: Explore the integration of augmented reality for book visualization. Allow users to preview how a book will look on their shelves or in their hands using AR technology, enhancing the online shopping experience.

**Inclusive Design for Accessibility:**

Opportunity: Conduct a thorough accessibility audit and implement inclusive design practices to ensure your application is accessible to users with disabilities. This can involve optimizing for screen readers, improving color contrast, and providing alternative navigation options.

**Dynamic Pricing Algorithms:**

Opportunity: Implement dynamic pricing algorithms that adjust product prices based on real-time market conditions, demand, or user behavior. This can optimize pricing strategies and maximize revenue.

# CHAPTER 2: APPLICATION DEVELOPMENT

## 1. Folder Structure Of The Application

ASP.NET Core's Models, Views, and Controllers (MVC) architectural style was used to create our website, SHOPBEE Book. Below is all of source folder contain source code for project:

Here is all folder and files of my project

(Folder Name: 1670\_ApplicationDevelopment\_ShopBee\_GroupProject)

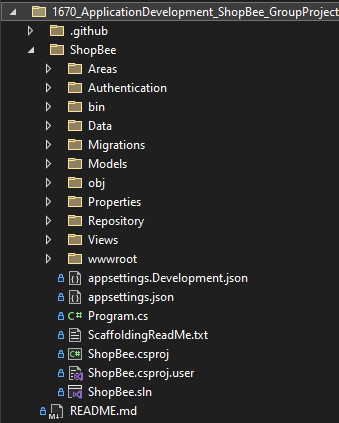


Figure 14: Project structure

The Area folder contains the Admin folder, which has the controller view and the project View.

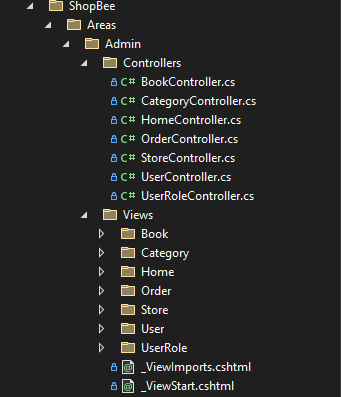


Figure 15: Areas Admin

Customer's Areas folder contains controllers and Views that limit customer access.

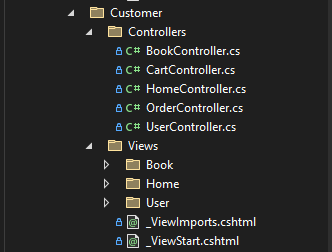


Figure 16: Areas Customer

The Store's Areas folder contains controllers and Views that limit Store access for sales.

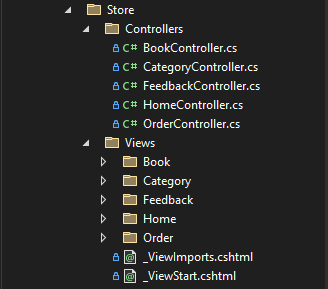


Figure 17: Areas Store

Models folder: It represents the data and business logic for Book, Cart, Category, User, ErrorViewModel, Order, OrderDetail, Feedback, Role, Store, UserRole of my application.

A screenshot of a computer

Description automatically generated

Figure 18: Models

View Models: It represents the data and business logic for BookDetailVM, BookVM, HomeVM, OrderVM, StoreVM, UserVM.

A screenshot of a computer

Description automatically generated

Figure 19: ViewModels

We domain each role to have different tasks as well as different functions. So below I will provide controllers for 3 roles: Admin, Customer, Store.

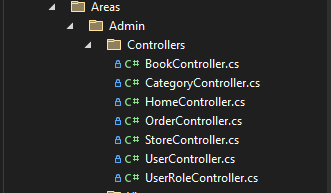


Figure 20: Admin Controller

A screenshot of a computer

Description automatically generated

Figure 21: Customer Controller

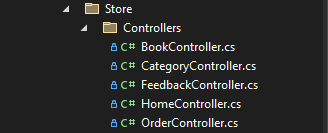


Figure 22: Store Controller

We domain each role to have different tasks as well as different functions. So below I will provide the Views of 3 roles: Admin, Customer, Store.

This is View of Admin Role.

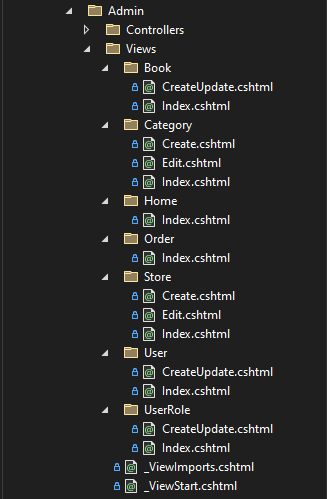


Figure 23: View of Admin

This is View of Customer

A screenshot of a computer

Description automatically generated

Figure 24: View of Customer

This is View of Store

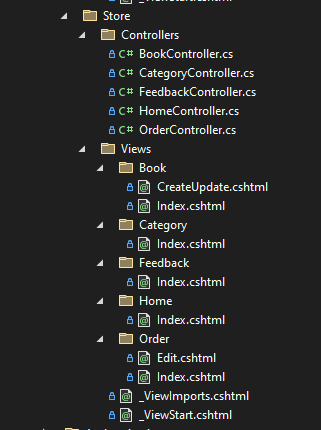


Figure 25: View of Store

Shared folder in View: It's used to store view templates and components that are shared

across multiple pages or views within my application.

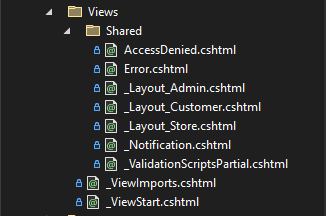


Figure 26: View Shared

wwwroot folder: It's the location where static folders, js, css, font, images, and lib are

stored. These folder are directly served to clients (browsers) by the web server without

any processing by the application.

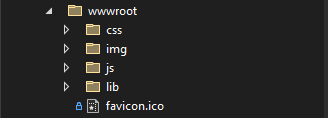


Figure 27: wwwroot

## 2. Source Code Samples Of The Application With Explanation

### 2.1 Model

I develop all model required for my project such as Book, Cart, Category, User, ErrorViewModel, Order, OrderDetail, Feedback, Role, Store, UserRole.

using Microsoft.AspNetCore.Mvc.ModelBinding.Validation;

using System.ComponentModel.DataAnnotations.Schema;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel;

namespace ShopBee.Models

{

    public class Book

    {

        [Key]

        public int Id { get; set; }

        [Required]

        [DisplayName("Book Name")]

        public string Name { get; set; }

        [ValidateNever]

        public int StoreID { get; set; }

        [ForeignKey("StoreID")]

        [ValidateNever]

        public Store Store { get; set; }

        public int CategoryId { get; set; }

        [ForeignKey("CategoryId")]

        [ValidateNever]

        public Category Category { get; set; }

        [Required]

        [DisplayName("Price $")]

        [Column(TypeName = "decimal(18,2)")]

        public decimal ActualPrice { get; set; }

        [Required]

        [DisplayName("Discount Price $")]

        [Column(TypeName = "decimal(18,2)")]

        public decimal DiscountPrice { get; set; }

        public int? Stock {  get; set; }

        [Required]

        [DisplayName("Author")]

        public string? Author { get; set; }

        [DisplayName("Description")]

        public string? Description { get; set; }

        [ValidateNever]

        public string? ImgUrl { get; set; }

        public DateTime? CreateDate { get; set; }

        public DateTime? ModifyDate { get; set; }

    }

}

**Book:** It contains properties int Id, Name, StoreID, CategoryId, ActualPrice, DiscountPrice, Stock, Author, Description, imgUrl, CreateDate, ModifyDate.

The source code file above is part of a web application built with ASP.NET Core, containing a model class named Book. Below is a detailed description of this model class:

Properties:

Id: Is the primary key of the Book object, has data type int and is marked with [Key], indicating that this is the primary key.

Name: Is the name of the book, marked with [Required] to ensure the value cannot be null.

StoreID: Is a foreign key referencing Store, marked with [ValidateNever] to not perform validation when binding data.

Store: Is the Store object referenced through StoreID.

CategoryId: Is a foreign key referencing Category.

Category: Is the Category object referenced through CategoryId.

ActualPrice: Is the original price of the book, has decimal data type with precision of 18 digits and 2 decimal numbers.

DiscountPrice: Is the discounted price of the book, also has decimal data type.

Stock: Is the inventory quantity of the book, can be null.

Author: Is the author of the book, can be null.

Description: Description of the book, can also be null.

ImgUrl: Is the image URL representing the book, can also be null.

CreateDate: Date the book was created.

ModifyDate: Date the book was modified.

Annotations:

[DisplayName("...")]: Use to specify the display name of the attribute when displayed in the user interface.

Data Types:

Use [Column(TypeName = "decimal(18,2)")] to specify the data type of the ActualPrice and DiscountPrice fields as decimal with a precision of 18 digits and 2 decimals.

Foreign Keys:

The StoreID and CategoryId properties are used to refer to Store and Category objects.

Validation:

Use [Required] to ensure that required fields have a value.

[ValidateNever] is used to not perform validation when binding data.

This model class describes information about a book in the store, including attributes such as name, price, inventory, author, description, and image.

using System.ComponentModel.DataAnnotations;

using System.ComponentModel;

namespace ShopBee.Models

{

    public class Category

    {

        [Key]

        public int Id { get; set; }

        [Required]

        [MaxLength(100, ErrorMessage = "Name must be 100 Characters ")]

        [DisplayName("Category Name")]

        public string? Name { get; set; }

        public int? Status { get; set; }

    }

}

**Category:** It contains properties int Id and Name, Status.

The "Category" model class in the source code is a crucial component of the ASP.NET Core web application. Here is a brief explanation of this class:

Id: Represents the primary key property of the category object. Each category has a unique Id value to identify it.

Name: Marked with [Required] to ensure that the category name cannot be null and [MaxLength(100, ErrorMessage = "Name must be 100 Characters ")] to limit the length of the category name to a maximum of 100 characters. The [DisplayName("Category Name")] attribute is used to specify the display name of this field in the user interface.

Status: Represents the property evaluating the status of the category (e.g., active, inactive). It is typed as int?, allowing the value to be null.

This class contains basic properties to represent fundamental information about a category in the system, aiding in the management and display of relevant information within the web application.

using Microsoft.AspNetCore.Mvc.ModelBinding.Validation;

using System.ComponentModel.DataAnnotations.Schema;

using System.ComponentModel.DataAnnotations;

namespace ShopBee.Models

{

    public class Cart

    {

        [Key]

        public int Id { get; set; }

        [Required]

        public int? UserID { get; set; }

        [ForeignKey("UserID")]

        [ValidateNever]

        public User? User { get; set; }

        [Required]

        public int? BookID { get; set; }

        [ForeignKey("BookID")]

        [ValidateNever]

        public Book? Book { get; set; }

        [Required]

        public int? StoreID { get; set; }

        [ForeignKey("StoreID")]

        [ValidateNever]

        public Store? Store { get; set; }

        [Required]

        public int? Quantity { get; set; }

        public string? Status { get; set; }

    }

}

**Cart:** It contains properties int Id and UserID, BookID, StoreID, Quantity.

The "Cart" model class is an essential component of the ASP.NET Core web application. Here is a brief explanation of this class:

Id: Represents the primary key property of the cart object. Each cart item has a unique Id value to identify it.

UserID: Represents the foreign key linking the cart item to a specific user. It is marked as [Required] and associated with the "User" navigation property.

BookID: Represents the foreign key linking the cart item to a specific book. It is marked as [Required] and associated with the "Book" navigation property.

StoreID: Represents the foreign key linking the cart item to a specific store. It is marked as [Required] and associated with the "Store" navigation property.

Quantity: Represents the quantity of the book in the cart. It is marked as [Required] to ensure a quantity value is provided.

Status: Represents the status of the cart item, such as "active" or "inactive." It is typed as a string, allowing for various status values.

This class encapsulates the properties necessary for managing and tracking items within a user's shopping cart, facilitating the e-commerce functionality of the web application.

using Microsoft.AspNetCore.Mvc.ModelBinding.Validation;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace ShopBee.Models

{

    public class Feedback

    {

        [Key]

        public int Id { get; set; }

        [Required]

        public int? BookId { get; set; }

        [ForeignKey("BookId")]

        [ValidateNever]

        public Book Book { get; set; }

        public int? UserId { get; set; }

        [ForeignKey("UserId")]

        [ValidateNever]

        public User User { get; set; }

        public int? OrderId { get; set; }

        [ForeignKey("OrderId")]

        [ValidateNever]

        public Order Order { get; set; }

        public string? Content { get; set; }

        public int? Rating { get; set; }

        public string? Response {  get; set; }

        public DateTime CreateDate {  get; set; }

    }

}

**Feedback:** It contains properties int Id, BookID, UserID, OrderID, Content, Rating, Response, CreateDate.

The "Feedback" model class is a key component of the ASP.NET Core web application, responsible for managing user feedback on various aspects. Here is a concise explanation of this class:

Id: Serves as the primary key for the feedback entity, providing a unique identifier for each feedback item.

BookId: Represents the foreign key linking the feedback to a specific book. It is marked as [Required] and associated with the "Book" navigation property.

UserId: Represents the foreign key linking the feedback to a specific user. It is marked as [Required] and associated with the "User" navigation property.

OrderId: Represents the foreign key linking the feedback to a specific order. It is marked as [Required] and associated with the "Order" navigation property.

Content: Stores the textual content of the feedback provided by the user.

Rating: Holds the numeric rating assigned by the user to express their satisfaction or opinion.

Response: Provides a space for any response or commentary from the system or administrator.

CreateDate: Captures the date and time when the feedback was created.

This class facilitates the collection and storage of user feedback, enhancing user engagement and enabling the system to gather insights into user experiences with books, orders, and the overall platform.

using System.ComponentModel;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace ShopBee.Models

{

    public class Order

    {

        [Key]

        public int Id { get; set; }

        public int StoreId { get; set; }

        [ForeignKey("StoreId")]

        public Store Store { get; set; }

        public int UserId { get; set; }

        [ForeignKey("UserId")]

        public User User { get; set; }

        public int Quantity { get; set; }

        [DisplayName("Price $")]

        [Column(TypeName = "decimal(18,2)")]

        public decimal TotalPrice { get; set; }

        public string Method { get; set; }

        public string Status { get; set; }

        public DateTime CreateDate { get; set; }

    }

}

**Order:** It contains properties int Id, StoreID, UserID, TotalPrice, Method, Status, CreateDate.

The "Order" model class in the ShopBee application is designed to manage and represent customer orders. Here is a brief explanation of its properties:

Id: Serves as the primary key for the order entity, providing a unique identifier for each order.

StoreId: Represents the foreign key linking the order to a specific store. It is associated with the "Store" navigation property.

UserId: Represents the foreign key linking the order to a specific user. It is associated with the "User" navigation property.

Quantity: Indicates the quantity of items included in the order.

TotalPrice: Represents the total cost of the order in dollars, stored as a decimal value with a precision of 18 digits and 2 decimal places.

Method: Specifies the payment method used for the order, whether it's through PayPal or another payment service.

Status: Represents the current status of the order, indicating whether it's pending, processed, shipped, or in another state.

CreateDate: Captures the date and time when the order was created.

This model class is essential for managing and tracking customer orders, associating them with specific users and stores, and storing relevant details such as quantity, total price, and order status.

using Microsoft.AspNetCore.Mvc.ModelBinding.Validation;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace ShopBee.Models

{

    public class OrderDetail

    {

        [Key]

        public int Id { get; set; }

        public int? BookId { get; set; }

        [ForeignKey("BookId")]

        [ValidateNever]

        public Book Book { get; set; }

        public int? OrderId { get; set; }

        [ForeignKey("OrderId")]

        [ValidateNever]

        public Order Order { get; set; }

        public int Quantity { get; set; }

        public double TotalPrice { get; set; }

    }

}

**OrderDetail:** It contains properties int Id, BookID, OrderID, Quantity, TotalPrice, CreateDate.

The "OrderDetail" model class in the ShopBee application is designed to store information about individual items within a customer order. Here is a brief explanation of its properties:

Id: Serves as the primary key for the order detail entity, providing a unique identifier for each order detail.

BookId: Represents the foreign key linking the order detail to a specific book. It is associated with the "Book" navigation property.

OrderId: Represents the foreign key linking the order detail to a specific order. It is associated with the "Order" navigation property.

Quantity: Indicates the quantity of a specific book included in the order detail.

TotalPrice: Represents the total cost of the specific book in the order detail. It is stored as a double precision floating-point number.

This model class is crucial for maintaining a detailed record of each item within a customer order, associating them with specific books and orders, and storing relevant details such as quantity and total price.

using System.ComponentModel.DataAnnotations;

namespace ShopBee.Models

{

    public class Role

    {

        [Key]

        public int Id { get; set; }

        [Required]

        public string? NomalizedName { get; set; }

    }

}

**Role:** It contains properties int Id and NomalizedName.

The "Role" model class in the ShopBee application is designed to represent user roles. Here's a brief explanation of its properties:

Id: Serves as the primary key for the role entity, providing a unique identifier for each role.

NomalizedName: Represents the normalized name of the role. The "Normalized" name is often used for case-insensitive comparisons and is typically derived from the regular name of the role.

This model class is essential for managing and assigning roles to users in the system, providing a structured way to control access and permissions based on user roles.

using Microsoft.AspNetCore.Mvc.ModelBinding.Validation;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace ShopBee.Models

{

    public class Store

    {

        [Key]

        public int Id { get; set; }

        public string? Name { get; set; }

        public int? UserId { get; set; }

        [ForeignKey("UserId")]

        [ValidateNever]

        public User User { get; set; }

        public DateTime CreateDate { get; set; }

    }

}

**Store:** It contains properties int Id and Name, UserID, CreateDate.

The "Store" model class in the ShopBee application is designed to represent stores. Here's a brief explanation of its properties:

Id: Serves as the primary key for the store entity, providing a unique identifier for each store.

Name: Represents the name of the store. It is a string property that holds the name of the store.

UserId: Represents the user ID associated with the store. It is a foreign key referencing the "User" model, indicating the owner or user associated with the store.

User: Represents the user associated with the store. It is a navigation property allowing access to the associated user information.

CreateDate: Represents the date when the store was created. It is a DateTime property indicating the creation timestamp of the store.

This model class is crucial for managing information about stores in the system, including their names, associated owners, and creation timestamps.

using System.ComponentModel.DataAnnotations;

namespace ShopBee.Models

{

    public class User

    {

        public enum GenderType

        {

            Male,

            Female

        }

        [Key]

        public int Id { get; set; }

        [Required]

        [EmailAddress]

        public string? Email { get; set; }

        [Required]

        [StringLength(100, MinimumLength = 8)]

        [DataType(DataType.Password)]

        public string? Password { get; set; }

        [Required]

        public string? Name { get; set; }

        [Required]

        public GenderType Gender { get; set; }

        public string? Phone { get; set; }

        public string? Adress { get; set; }

        public string? avtURL { get; set; }

        public DateTime CreateDate { get; set; }

        public DateTime ModifyDate { get; set; }

    }

}

**User:** It contains properties int Id, Email, Password, Name, Gender, Phone, Address, avtUrl, CreateDate, ModifyDate.

The "User" model class in the ShopBee application represents user entities. Below is a summary of its properties:

Id: Serves as the primary key for the user entity, providing a unique identifier for each user.

Email: Represents the email address of the user. It is required and must be a valid email address.

Password: Represents the password associated with the user. It is required, and its length must be between 8 and 100 characters.

Name: Represents the name of the user. It is a required field.

Gender: Represents the gender of the user, defined by the "GenderType" enumeration, which includes values for "Male" and "Female."

Phone: Represents the phone number of the user. It is an optional field.

Address: Represents the address of the user. It is an optional field.

avtURL: Represents the URL of the user's avatar. It is an optional field.

CreateDate: Represents the date when the user account was created.

ModifyDate: Represents the date when the user account was last modified.

This model class captures essential information about users, including their authentication credentials, personal details, and profile-related information such as avatar and contact details.

using Microsoft.AspNetCore.Mvc.ModelBinding.Validation;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace ShopBee.Models

{

    public class UserRole

    {

        [Key]

        public int Id { get; set; }

        public int? UserId { get; set; }

        [ForeignKey("UserId")]

        [ValidateNever]

        public User User { get; set; }

        public int? RoleId { get; set; }

        [ForeignKey("RoleId")]

        [ValidateNever]

        public Role Role { get; set; }

    }

}

**UserRole:** It contains properties int Id, UserID, RoleID.

The "UserRole" model class in the ShopBee application is designed to manage the roles assigned to users. Here is a summary of its properties:

Id: Serves as the primary key for the user role entity, providing a unique identifier for each user role.

UserId: Represents the foreign key linking the user role to a specific user. It is associated with the "User" model.

User: Represents the navigation property allowing access to the associated user entity.

RoleId: Represents the foreign key linking the user role to a specific role. It is associated with the "Role" model.

Role: Represents the navigation property allowing access to the associated role entity.

This model class establishes a relationship between users and roles, allowing for role-based access control and permissions within the ShopBee application. Roles provide a way to categorize and manage user access levels, ensuring a secure and organized system.

### 2.2 Controller

#### 2.2.1 Admin

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Rendering;

using Microsoft.AspNetCore.Authorization;

using ShopBee.Models.ViewModels;

using ShopBee.Models;

using ShopBee.Repository.IRepository;

using ShopBee.Authentication;

namespace ShopBee.Areas.Admin.Controllers

{

    [Area("Admin")]

    [AdminAuthentication()]

    public class BookController : Controller

    {

        //private readonly ApplicationDBContext \_dbContext;

        private readonly IUnitOfWork \_unitOfWork;

        private readonly IWebHostEnvironment \_webhost;

        public BookController(IUnitOfWork unitOfWork, IWebHostEnvironment webhost)

        {

            \_unitOfWork = unitOfWork;

            \_webhost = webhost;

        }

        public IActionResult Index()

        {

            return View();

        }

        public IActionResult CreateUpdate(int? id)

        {

            BookVM bookVM = new BookVM()

            {

                MyCategories = \_unitOfWork.Category.GetAll().

                Select(u => new SelectListItem

                {

                    Text = u.Name,

                    Value = u.Id.ToString()

                }),

                MyStores = \_unitOfWork.Store.GetAll().

                Select(u => new SelectListItem

                {

                    Text = u.Name,

                    Value = u.Id.ToString()

                }),

                Book = new Book()

            };

            if (id == null || id == 0)

            {

                //Create new Book

                return View(bookVM);

            }

            else

            {

                //Update a Book

                bookVM.Book = \_unitOfWork.Book.Get(book => book.Id == id);

                return View(bookVM);

            }

        }

        [HttpPost]

        public IActionResult CreateUpdate(BookVM bookVM, IFormFile? file)

        {

            if (ModelState.IsValid)

            {

                string wwwRootPath = \_webhost.WebRootPath;

                if (file != null)

                {

                    string fileName = Guid.NewGuid().ToString() + Path.GetExtension(file.FileName);

                    string bookPath = Path.Combine(wwwRootPath, "img/bookImg");

                    if (!string.IsNullOrEmpty(bookVM.Book.ImgUrl))

                    {

                        //Delete old image

                        var oldImagePath = Path.Combine(wwwRootPath, bookVM.Book.ImgUrl.TrimStart('\\'));

                        if (System.IO.File.Exists(oldImagePath))

                        {

                            System.IO.File.Delete(oldImagePath);

                        }

                    }

                    using (var fileStream = new FileStream(Path.Combine(bookPath, fileName), FileMode.Create))

                    {

                        file.CopyTo(fileStream);

                    }

                    bookVM.Book.ImgUrl = @"/img/bookImg/" + fileName;

                }

                if (bookVM.Book.Id == 0)

                {

                    bookVM.Book.CreateDate = DateTime.Today.Date;

                    bookVM.Book.ModifyDate = DateTime.Today.Date;

                    \_unitOfWork.Book.Add(bookVM.Book);

                    TempData["success"] = "Book created succesfully";

                }

                else

                {

                    bookVM.Book.ModifyDate = DateTime.Today.Date;

                    \_unitOfWork.Book.Update(bookVM.Book);

                    TempData["success"] = "Book updated succesfully";

                }

                \_unitOfWork.Save();

                return RedirectToAction("Index");

            }

            else

            {

                bookVM.MyCategories = \_unitOfWork.Category.GetAll().

                            Select(u => new SelectListItem

                            {

                                Text = u.Name,

                                Value = u.Id.ToString()

                            });

                bookVM.MyStores = \_unitOfWork.Store.GetAll().

                            Select(u => new SelectListItem

                            {

                                Text = u.Name,

                                Value = u.Id.ToString()

                            });

                return View(bookVM);

            }

        }

        [HttpGet]

        public IActionResult GetAll()

        {

            List<Book> obj = \_unitOfWork.Book.GetAll(includeProperties: "Category,Store").ToList();

            return Json(new { data = obj });

        }

        [HttpDelete]

        public IActionResult Delete(int id)

        {

            var bookDelete = \_unitOfWork.Book.Get(u => u.Id == id);

            if (bookDelete == null)

            {

                return Json(new { success = false, message = "Error while deleting" });

            }

            \_unitOfWork.Book.Remove(bookDelete); \_unitOfWork.Save();

            return Json(new { success = true, message = "Delete Successful" });

        }

    }

}

**BookController:** In BookController I have methods like Index(), CreateUpdate(), GetAll(), Delete(int id). Here I use UnitOfWork to manipulate the database. IWebHostEnvironment in ASP.NET Core is an important interface used to expose information about the environment in which the application is operating. This object provides properties such as EnvironmentName to identify the environment (like "Development" or "Production"), ContentRootPath and WebRootPath to get the full path to the directory containing the resources and the application root directory. Through IWebHostEnvironment, applications are able to adapt to environmental conditions and manage resources effectively. The CreateUpdate() method is that we combine both creation and update in one method. If id = null, we will create a new book, and if id is not null, we will get the book with the id to update, followed by the methods delete and getAll() is to get the list of Book.

using Microsoft.AspNetCore.Mvc;

using ShopBee.Authentication;

using ShopBee.Models;

using ShopBee.Repository.IRepository;

namespace ShopBee.Areas.Admin.Controllers

{

    [Area("Admin")]

    [AdminAuthentication()]

    public class CategoryController : Controller

    {

        private readonly IUnitOfWork \_unitOfWork;

        public CategoryController(IUnitOfWork db)

        {

            \_unitOfWork = db;

        }

        public IActionResult Index()

        {

            List<Category> objCategoryList = \_unitOfWork.Category.GetAll().ToList();

            return View(objCategoryList);

        }

        public IActionResult Create()

        {

            return View();

        }

        [HttpPost]

        public IActionResult Create(Category obj)

        {

            if (ModelState.IsValid)

            {

                \_unitOfWork.Category.Add(obj);

                \_unitOfWork.Save();

                TempData["success"] = "Category created successfully";

                return RedirectToAction("Index");

            }

            return View();

        }

        public IActionResult Edit(int? id)

        {

            if (id == null || id == 0)

            {

                return NotFound();

            }

            Category? categoryFromDb = \_unitOfWork.Category.Get(u => u.Id == id);

            //CategoryModel? categoryFromDb1 = \_db.Categories.FirstOrDefault(u=>u.Id==id);

            //CategoryModel? categoryFromDb2 = \_db.Categories.Where(u=>u.Id==id).FirstOrDefault();

            if (categoryFromDb == null)

            {

                return NotFound();

            }

            return View(categoryFromDb);

        }

        [HttpPost]

        public IActionResult Edit(Category obj)

        {

            if (ModelState.IsValid)

            {

                \_unitOfWork.Category.Update(obj);

                \_unitOfWork.Save();

                TempData["success"] = "Category edited successfully";

                return RedirectToAction("Index");

            }

            return View();

        }

        #region API CALLS

        [HttpGet]

        public IActionResult GetAll()

        {

            List<Category> obj = \_unitOfWork.Category.GetAll().ToList();

            return Json(new { data = obj });

        }

        [HttpDelete]

        public IActionResult Delete(int id)

        {

            var categoryDelete = \_unitOfWork.Category.Get(u => u.Id == id);

            if (categoryDelete == null)

            {

                return Json(new { success = false, message = "Error while deleting" });

            }

            \_unitOfWork.Category.Remove(categoryDelete); \_unitOfWork.Save();

            return Json(new { success = true, message = "Delete Successful" });

        }

        #endregion

    }

}

**CategoryController**: In CategoryController, there are methods Index(), Create(), Edit(), GetAll(), Delete(int id). In the constructor we use UnitOfWork to be able to interact with the database part. The Index action is retrieved for all categories in the database using the unitOfWork.GetAll() function. The Details, Create, Edit, and Delete actions are missing from the provided code, but they typically handle displaying details of a specific category, creating new categories, updating categories, and deleting corresponding categories. response.

using Microsoft.AspNetCore.Mvc;

using ShopBee.Authentication;

using ShopBee.Data;

using ShopBee.Models;

using ShopBee.Models.ViewModels;

using ShopBee.Repository.IRepository;

namespace ShopBee.Areas.Admin.Controllers

{

    [Area("Admin")]

    [AdminAuthentication()]

    public class HomeController : Controller

    {

        private readonly IUnitOfWork \_unitOfWork;

        public HomeController(IUnitOfWork unitOfWork)

        {

            \_unitOfWork = unitOfWork;

        }

        public IActionResult Index()

        {

            int numberOfBooks = \_unitOfWork.Book.GetNumberOfBooks();

            int numberOfUsers = \_unitOfWork.User.GetNumberOfUsers();

            int numberOfOrders = \_unitOfWork.Order.GetNumberOfOrders();

            int numberOfStores = \_unitOfWork.Store.GetNumberOfStores();

            ViewBag.BookModel = new BookVM { NumberOfBooks = numberOfBooks };

            ViewBag.UserModel = new UserVM { NumberOfUsers = numberOfUsers };

            ViewBag.OrderModel = new OrderVM { NumberOfOrders = numberOfOrders };

            ViewBag.StoreModel = new StoreVM { NumberOfStores = numberOfStores };

            return View();

        }

    }

}

**HomeController:** In HomeController, I only use the Index() action to display data, in which I use UnitOfWork to interact with the database. In the Index action, I have GetNumberOfBooks, GetNumberOfUsers, GetNumberOfOrders, GetNumberOfStores to get data to display in the dashboard.

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Rendering;

using ShopBee.Authentication;

using ShopBee.Models;

using ShopBee.Models.ViewModels;

using ShopBee.Repository.IRepository;

namespace ShopBee.Areas.Admin.Controllers

{

    [Area("Admin")]

    [AdminAuthentication()]

    public class OrderController : Controller

    {

        private readonly IUnitOfWork \_unitOfWork;

        public OrderController(IUnitOfWork db)

        {

            \_unitOfWork = db;

        }

        public IActionResult Index()

        {

            return View();

        }

        #region API CALLS

        [HttpGet]

        public IActionResult GetAll()

        {

            List<Order> obj = \_unitOfWork.Order.GetAll(includeProperties: "User").ToList();

            return Json(new { data = obj });

        }

        [HttpDelete]

        public IActionResult Delete(int id)

        {

            var OrderDelete = \_unitOfWork.Order.Get(u => u.Id == id);

            if (OrderDelete == null)

            {

                return Json(new { success = false, message = "Error while deleting" });

            }

            \_unitOfWork.Order.Remove(OrderDelete); \_unitOfWork.Save();

            return Json(new { success = true, message = "Delete Successful" });

        }

        #endregion

    }

}

**OrderController:** Controller "OrderController" is part of the ASP.NET Core project that manages orders in the ShopBee website. Using the Dependency Injection function, this controller is responsible for handling order-related requests in the application's Admin area.

Action methods include:

Index: Returns a view, there is no specific processing logic in this action.

GetAll: Returns a list of orders and related user information in JSON form, serving AJAX requests from the client side.

Delete: Delete an order based on ID and return the deletion result via JSON.

The IUnitOfWork object is injected into the controller to manage interaction with the database. Thereby, the controller provides important API calls to display and manage orders, and defines admin-only access rights through the attribute.

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Rendering;

using ShopBee.Authentication;

using ShopBee.Models;

using ShopBee.Models.ViewModels;

using ShopBee.Repository.IRepository;

namespace ShopBee.Areas.Admin.Controllers

{

    [Area("Admin")]

    [AdminAuthentication()]

    public class StoreController : Controller

    {

        private readonly IUnitOfWork \_unitOfWork;

        public StoreController(IUnitOfWork db)

        {

            \_unitOfWork = db;

        }

        public IActionResult Index()

        {

            return View();

        }

        public IActionResult Create()

        {

            StoreVM storeVM = new StoreVM()

            {

                MyUsers = \_unitOfWork.User.GetAll().

                 Select(u => new SelectListItem

                 {

                     Text = u.Name,

                     Value = u.Id.ToString()

                 }),

                Store = new ShopBee.Models.Store()

            };

            return View(storeVM);

        }

        [HttpPost]

        public IActionResult Create(StoreVM storeVM)

        {

            if (ModelState.IsValid)

            {

                storeVM.Store.CreateDate = DateTime.Today.Date;

                \_unitOfWork.Store.Add(storeVM.Store);

                \_unitOfWork.Save();

                TempData["success"] = "Category created successfully";

                return RedirectToAction("Index");

            }

            else

            {

                storeVM.MyUsers = \_unitOfWork.User.GetAll().

                            Select(u => new SelectListItem

                            {

                                Text = u.Name,

                                Value = u.Id.ToString()

                            });

                return View(storeVM);

            }

        }

        public IActionResult Edit(int? id)

        {

            if (id == null || id == 0)

            {

                return NotFound();

            }

            StoreVM storeVM = new StoreVM()

            {

                MyUsers = \_unitOfWork.User.GetAll().

                 Select(u => new SelectListItem

                 {

                     Text = u.Name,

                     Value = u.Id.ToString()

                 }),

                Store = new ShopBee.Models.Store()

            };

            storeVM.Store = \_unitOfWork.Store.Get(store => store.Id == id);

            if (storeVM == null)

            {

                return NotFound();

            }

            return View(storeVM);

        }

        [HttpPost]

        public IActionResult Edit(StoreVM storeVM)

        {

            if (ModelState.IsValid)

            {

                storeVM.Store.CreateDate = DateTime.Today;

                \_unitOfWork.Store.Update(storeVM.Store);

                \_unitOfWork.Save();

                TempData["success"] = "Store edited successfully";

                return RedirectToAction("Index");

            }

            return View();

        }

        #region API CALLS

        [HttpGet]

        public IActionResult GetAll()

        {

            List<ShopBee.Models.Store> obj = \_unitOfWork.Store.GetAll(includeProperties: "User").ToList();

            return Json(new { data = obj });

        }

        [HttpDelete]

        public IActionResult Delete(int id)

        {

            var storeDelete = \_unitOfWork.Store.Get(u => u.Id == id);

            if (storeDelete == null)

            {

                return Json(new { success = false, message = "Error while deleting" });

            }

            \_unitOfWork.Store.Remove(storeDelete); \_unitOfWork.Save();

            return Json(new { success = true, message = "Delete Successful" });

        }

        #endregion

    }

}

**StoreController:** Controller "StoreController" plays an important role in the ASP.NET Core project of the ShopBee website, located in the admin area (Admin). This source code is designed to manage information about stores, including functions to create, edit, delete and display store lists.

In the new creation (`Create`), this controller uses the `StoreVM` object to display an interface with a list of users and information about the store. The input data is checked, and if valid, the new store information is added to the database and redirected to the home page.

The edit section (`Edit`) allows administrators to adjust store information based on ID. The information is updated after validation, and then redirected to the home page.

Additionally, the controller provides API calls to interact with the database, including retrieving a list of stores and deleting stores based on ID. These functions respond to AJAX requests from the client and return results via JSON format.

Overall, "StoreController" plays an important role in managing website store information, providing efficient administrative functions and flexible interaction with the database.

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Rendering;

using ShopBee.Authentication;

using ShopBee.Models;

using ShopBee.Models.ViewModels;

using ShopBee.Repository.IRepository;

namespace ShopBee.Areas.Admin.Controllers

{

    [Area("Admin")]

    [AdminAuthentication()]

    public class UserController : Controller

    {

        private readonly IUnitOfWork \_unitOfWork;

        private readonly IWebHostEnvironment \_webhost;

        public UserController(IUnitOfWork db, IWebHostEnvironment webhost)

        {

            \_unitOfWork = db;

            \_webhost = webhost;

        }

        public IActionResult Index()

        {

            List<User> objUserList = \_unitOfWork.User.GetAll().ToList();

            return View(objUserList);

        }

        public IActionResult CreateUpdate(int? id)

        {

            UserVM userVM = new UserVM()

            {

                MyUsers = \_unitOfWork.User.GetAll().

                Select(u => new SelectListItem

                {

                    Text = u.Name,

                    Value = u.Id.ToString()

                }),

                User = new User()

            };

            if ( id == null || id == 0)

            {

                //Create new User

                return View(userVM);

            }

            else

            {

                //Update a User

                userVM.User = \_unitOfWork.User.Get(user => user.Id == id);

                return View(userVM);

            }

        }

        [HttpPost]

        public IActionResult CreateUpdate(UserVM userVM, IFormFile? file)

        {

            if (ModelState.IsValid)

            {

                string wwwRootPath = \_webhost.WebRootPath;

                if (file != null)

                {

                    string fileName = Guid.NewGuid().ToString() + Path.GetExtension(file.FileName);

                    string userPath = Path.Combine(wwwRootPath, "img/userAvt");

                    if (!string.IsNullOrEmpty(userVM.User.avtURL))

                    {

                        //Delete old image

                        var oldImagePath = Path.Combine(wwwRootPath, userVM.User.avtURL.TrimStart('\\'));

                        if (System.IO.File.Exists(oldImagePath))

                        {

                            System.IO.File.Delete(oldImagePath);

                        }

                    }

                    using (var fileStream = new FileStream(Path.Combine(userPath, fileName), FileMode.Create))

                    {

                        file.CopyTo(fileStream);

                    }

                    userVM.User.avtURL = @"/img/userAvt/" + fileName;

                }

                if (userVM.User.Id == 0)

                {

                    userVM.User.CreateDate = DateTime.Today.Date;

                    userVM.User.ModifyDate = DateTime.Today.Date;

                    \_unitOfWork.User.Add(userVM.User);

                    TempData["success"] = "User created succesfully";

                }

                else

                {

                    userVM.User.ModifyDate = DateTime.Today.Date;

                    \_unitOfWork.User.Update(userVM.User);

                    TempData["success"] = "User updated succesfully";

                }

                \_unitOfWork.Save();

                return RedirectToAction("Index");

            }

            else

            {

                UserVM userVMNew = new UserVM()

                {

                    MyUsers = \_unitOfWork.User.GetAll().

                                Select(u => new SelectListItem

                                {

                                    Text = u.Name,

                                    Value = u.Id.ToString()

                                }),

                    User = new User()

                };

                return View(userVMNew);

            }

        }

        public IActionResult Edit(int? id)

        {

            if (id == null || id == 0)

            {

                return NotFound();

            }

            User? userFromDb = \_unitOfWork.User.Get(u => u.Id == id);

            if (userFromDb == null)

            {

                return NotFound();

            }

            return View(userFromDb);

        }

        [HttpPost]

        public IActionResult Edit(User obj)

        {

            if (ModelState.IsValid)

            {

                \_unitOfWork.User.Update(obj);

                \_unitOfWork.Save();

                TempData["success"] = "User edited successfully";

                return RedirectToAction("Index");

            }

            return View();

        }

        #region API CALLS

        [HttpGet]

        public IActionResult GetAll()

        {

            List<User> obj = \_unitOfWork.User.GetAll().ToList();

            return Json(new { data = obj });

        }

        [HttpDelete]

        public IActionResult Delete(int id)

        {

            var userDelete = \_unitOfWork.User.Get(u => u.Id == id);

            if (userDelete == null)

            {

                return Json(new { success = false, message = "Error while deleting" });

            }

            \_unitOfWork.User.Remove(userDelete); \_unitOfWork.Save();

            return Json(new { success = true, message = "Delete User Successful" });

        }

        #endregion

    }

}

**UserController**:

Controller "UserController" is a main part of the ASP.NET Core application in the ShopBee website, located in the admin area (Admin). Below is a summary of the source code:

Namespace and Dependency Injection:

Use the necessary namespaces and perform Dependency Injection to inject objects implementing IUnitOfWork and IWebHostEnvironment into the controller.

Attribute and Area Routing:

This controller belongs to the "Admin" area and requires admin access via the [AdminAuthentication()] attribute.

Constructors and Dependency Injection:

The UserController constructor receives two objects, one implementing IUnitOfWork and one implementing IWebHostEnvironment, via dependency injection.

Action Methods:

Index: Returns a view displaying a list of users from the database.

CreateUpdate: Displays the interface to create new or update users based on ID. Process input data and save user image if available.

Edit: Allows editing user information based on ID.

Both CreateUpdate and Edit action methods check the validity of the data, perform corresponding operations, and redirect to the home page.

API CALLS - Action Methods for API:

GetAll: Returns a list of users from the database as JSON, serving AJAX requests from the client side.

Delete: Delete a user based on ID and return results via JSON.

Overall, "UserController" is responsible for managing user information, providing functions corresponding to CRUD and interacting with the database, and supports API calls to work with data from the client side.

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Rendering;

using ShopBee.Authentication;

using ShopBee.Models;

using ShopBee.Models.ViewModels;

using ShopBee.Repository;

using ShopBee.Repository.IRepository;

namespace ShopBee.Areas.Admin.Controllers

{

    [Area("Admin")]

    [AdminAuthentication()]

    public class UserRoleController : Controller

    {

        private readonly IUnitOfWork \_unitOfWork;

        private readonly IWebHostEnvironment \_webhost;

        public UserRoleController(IUnitOfWork db, IWebHostEnvironment webhost)

        {

            \_unitOfWork = db;

            \_webhost = webhost;

        }

        public IActionResult Index()

        {

            return View();

        }

        public IActionResult CreateUpdate(int? id)

        {

            UserVM userVM = new UserVM()

            {

                MyUsers = \_unitOfWork.User.GetAll().

                Select(u => new SelectListItem

                {

                    Text = u.Email,

                    Value = u.Id.ToString()

                }),

                MyRoles = \_unitOfWork.Role.GetAll().

                Select(u => new SelectListItem

                {

                    Text = u.NomalizedName,

                    Value = u.Id.ToString()

                }),

                UserRole = new UserRole()

            };

            if (id == null || id == 0)

            {

                return View(userVM);

            }

            else

            {

                userVM.UserRole = \_unitOfWork.UserRole.Get(userrole => userrole.Id == id);

                return View(userVM);

            }

        }

        [HttpPost]

        public IActionResult CreateUpdate(UserVM userVM,string? email\_temp)

        {

            bool checkDuplicated = false;

            var checkEmailOfUser = \_unitOfWork.User.Get(user => user.Email == email\_temp);

            if (checkEmailOfUser != null && ModelState.IsValid)

            {

                List<UserRole> userRolesList = \_unitOfWork.UserRole.GetAll().ToList();

                userVM.UserRole.UserId = checkEmailOfUser.Id;

                foreach (var userRole in userRolesList)

                {

                    if (userRole.UserId == userVM.UserRole.UserId && userRole.RoleId == userVM.UserRole.RoleId)

                    {

                        checkDuplicated = true;

                        break;

                    }

                }

                if (checkDuplicated == false)

                {

                    \_unitOfWork.UserRole.Add(userVM.UserRole);

                    TempData["success"] = "User Role created succesfully";

                    \_unitOfWork.Save();

                    return RedirectToAction("Index");

                }

                else

                {

                    TempData["error"] = "This User had this Role Before";

                    userVM.MyUsers = \_unitOfWork.User.GetAll().

                    Select(u => new SelectListItem

                    {

                        Text = u.Email,

                        Value = u.Id.ToString()

                    });

                    userVM.MyRoles = \_unitOfWork.Role.GetAll().

                    Select(u => new SelectListItem

                    {

                        Text = u.NomalizedName,

                        Value = u.Id.ToString()

                    });

                    return View(userVM);

                }

            }

            else

            {

                if (checkEmailOfUser == null) {

                    TempData["error"] = "This email does not belong to any User in the system";

                }

                userVM.MyUsers = \_unitOfWork.User.GetAll().

                Select(u => new SelectListItem

                {

                    Text = u.Email,

                    Value = u.Id.ToString()

                });

                userVM.MyRoles = \_unitOfWork.Role.GetAll().

                Select(u => new SelectListItem

                {

                    Text = u.NomalizedName,

                    Value = u.Id.ToString()

                });

                return View(userVM);

            }

        }

        #region API CALLS

        [HttpGet]

        public IActionResult GetAll()

        {

            List<UserRole> obj = \_unitOfWork.UserRole.GetAll(includeProperties: "Role,User").ToList();

            return Json(new { data = obj });

        }

        [HttpDelete]

        public IActionResult Delete(int id)

        {

            var userRole = \_unitOfWork.UserRole.Get(u => u.Id == id);

            if (userRole == null)

            {

                return Json(new { success = false, message = "Error while deleting" });

            }

            \_unitOfWork.UserRole.Remove(userRole); \_unitOfWork.Save();

            return Json(new { success = true, message = "Delete Successful" });

        }

        #endregion

    }

}

**UserRoleController:** Controller "UserRoleController" plays an important role in the ASP.NET Core system of the ShopBee website, located in the admin area (Admin). This source code focuses on managing user rights information, including the functions of creating, editing, deleting and displaying user rights lists.

In the create and update section (`CreateUpdate`), the controller uses the `UserVM` object to display the interface for creating or updating user permissions based on ID. The input data is checked for validity and then user permission information is added or updated. At the same time, check duplicates to ensure the uniqueness of user permissions and display corresponding messages.

Additionally, the controller provides API calls to interact with the database, including retrieving a list of user permissions and removing user permissions based on ID. These functions respond to AJAX requests from the client and return results via JSON format.

Overall, "UserRoleController" is responsible for managing user rights information of the ShopBee website, providing efficient administrative functions and flexible interaction with the database.

#### 2.2.2 Store

**Book Controller:**

**Controller Structure**

The BookController is a part of the Store area (as indicated by [Area("Store")]) within the application.

[StoreAuthentication()] suggests that this controller might have custom authentication logic defined for accessing its actions.

**Constructor Injection**

Constructor injection is used to inject dependencies (IUnitOfWork and IWebHostEnvironment) required by the controller.

**Actions**:

* **Index():** This action returns a view for displaying a list of books. Currently, it returns an empty view.
* **CreateUpdate(int? id):** Handles both creating a new book and updating an existing book. If id is null or 0, it presents a view to create a new book. Otherwise, it retrieves the book by its ID and presents a view to update its details.
* **CreateUpdate(BookVM bookVM, IFormFile? file):** Handles the HTTP POST request for creating/updating a book. It handles file uploads for the book's image, saves the image to the server, and updates the book details accordingly. It also handles form validation.
* **GetAll():** An API endpoint that retrieves all books associated with the currently logged-in store owner (retrieved using session information).
* **Delete(int id):** An API endpoint to delete a book by its ID. It fetches the book, removes it from the repository, and saves changes.

**Explanation of Actions:**

* **Dependency Injection:** The controller relies on an IUnitOfWork to interact with the data layer. This pattern abstracts away data access, promoting modularity and testability.
* **ViewModel Usage (BookVM):** BookVM appears to be a ViewModel used to pass data to and from views. It contains data related to books, categories, and stores.
* **File Upload Handling:** The code manages file uploads for book images, ensuring uniqueness by appending a GUID to the file name and storing it in a specified directory.
* **Session Usage:** The controller uses session information to retrieve the currently logged-in user's ID, presumably to associate books with the respective store owner.
* using Microsoft.AspNetCore.Mvc;
* using Microsoft.AspNetCore.Mvc.Rendering;
* using Microsoft.AspNetCore.Authorization;
* using ShopBee.Models.ViewModels;
* using ShopBee.Models;
* using ShopBee.Repository.IRepository;
* using ShopBee.Authentication;
* using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
* namespace ShopBee.Areas.Store.Controllers
* {
* [Area("Store")]
* [StoreAuthentication()]
* public class BookController : Controller
* {
* //private readonly ApplicationDBContext \_dbContext;
* private readonly IUnitOfWork \_unitOfWork;
* private readonly IWebHostEnvironment \_webhost;
* public BookController(IUnitOfWork unitOfWork, IWebHostEnvironment webhost)
* {
* \_unitOfWork = unitOfWork;
* \_webhost = webhost;
* }
* public IActionResult Index()
* {
* return View();
* }
* public IActionResult CreateUpdate(int? id)
* {
* BookVM bookVM = new BookVM()
* {
* MyCategories = \_unitOfWork.Category.GetAll().Where(c => c.Status == 1).
* Select(u => new SelectListItem
* {
* Text = u.Name,
* Value = u.Id.ToString()
* }),
* MyStores = \_unitOfWork.Store.GetAll().
* Select(u => new SelectListItem
* {
* Text = u.Name,
* Value = u.Id.ToString()
* }),
* Book = new Book()
* };
* if (id == null || id == 0)
* {
* //Create new Book
* return View(bookVM);
* }
* else
* {
* //Update a Book
* bookVM.Book = \_unitOfWork.Book.Get(book => book.Id == id);
* return View(bookVM);
* }
* }
* [HttpPost]
* public IActionResult CreateUpdate(BookVM bookVM, IFormFile? file)
* {
* if (ModelState.IsValid)
* {
* string wwwRootPath = \_webhost.WebRootPath;
* if (file != null)
* {
* string fileName = Guid.NewGuid().ToString() + Path.GetExtension(file.FileName);
* string bookPath = Path.Combine(wwwRootPath, "img/bookImg");
* if (!string.IsNullOrEmpty(bookVM.Book.ImgUrl))
* {
* //Delete old image
* var oldImagePath = Path.Combine(wwwRootPath, bookVM.Book.ImgUrl.TrimStart('\\'));
* if (System.IO.File.Exists(oldImagePath))
* {
* System.IO.File.Delete(oldImagePath);
* }
* }
* using (var fileStream = new FileStream(Path.Combine(bookPath, fileName), FileMode.Create))
* {
* file.CopyTo(fileStream);
* }
* bookVM.Book.ImgUrl = @"/img/bookImg/" + fileName;
* }
* if (bookVM.Book.Id == 0)
* {
* var UserIdGet = HttpContext.Session.GetString("UserId");
* int.TryParse(UserIdGet, out int storeOwnerId);
* ShopBee.Models.Store store = \_unitOfWork.Store.Get(u => u.UserId == storeOwnerId);
* bookVM.Book.StoreID = store.Id;
* bookVM.Book.CreateDate = DateTime.Today;
* bookVM.Book.ModifyDate = DateTime.Today;
* \_unitOfWork.Book.Add(bookVM.Book);
* TempData["success"] = "Book created succesfully";
* }
* else
* {
* bookVM.Book.ModifyDate = DateTime.Today;
* \_unitOfWork.Book.Update(bookVM.Book);
* TempData["success"] = "Book updated succesfully";
* }
* \_unitOfWork.Save();
* return RedirectToAction("Index");
* }
* else
* {
* bookVM.MyCategories = \_unitOfWork.Category.GetAll().Where(c => c.Status == 1).
* Select(u => new SelectListItem
* {
* Text = u.Name,
* Value = u.Id.ToString()
* });
* bookVM.MyStores = \_unitOfWork.Store.GetAll().
* Select(u => new SelectListItem
* {
* Text = u.Name,
* Value = u.Id.ToString()
* });
* return View(bookVM);
* }
* }
* #region API CALLS
* [HttpGet]
* public IActionResult GetAll()
* {
* var UserIdGet = HttpContext.Session.GetString("UserId");
* int.TryParse(UserIdGet, out int storeOwnerId);
* ShopBee.Models.Store store = \_unitOfWork.Store.Get(u => u.UserId == storeOwnerId);
* List<Book> books = \_unitOfWork.Book.GetAll(includeProperties: "Category,Store").Where(u => u.StoreID == store.Id).ToList();
* return Json(new { data = books });
* }
* [HttpDelete]
* public IActionResult Delete(int id)
* {
* var bookDelete = \_unitOfWork.Book.Get(u => u.Id == id);
* if (bookDelete == null)
* {
* return Json(new { success = false, message = "Error while deleting" });
* }
* \_unitOfWork.Book.Remove(bookDelete); \_unitOfWork.Save();
* return Json(new { success = true, message = "Delete Successful" });
* }
* #endregion
* }
* }

**Category Controller:**

**Controller Structure**

The CategoryController is within the Store area ([Area("Store")]) of the application.

[StoreAuthentication()] suggests that this controller might require authentication logic specific to the store area.

**Constructor Injection**

The controller expects an IUnitOfWork dependency to interact with the data layer. This dependency is injected into the constructor.

**Actions:**

* Index(): This action renders a view to display the list of categories. As it stands, it returns an empty view.
* Index(Category obj): This is the HTTP POST action corresponding to the Index view. It receives a Category object as a parameter, presumably from a form submission.
* If the model state is valid (meaning the model has passed validation checks), it sets the Status property of the category object to 0, indicating a pending status.
* It adds the category object to the repository using the UnitOfWork and saves changes.
* Sets a success message in TempData to notify the user that their request has been received and pending approval.
* Redirects back to the Index action to display the view.

**Explanation of Actions:**

* Dependency Injection: The controller relies on an IUnitOfWork to interact with the data layer. The use of this pattern abstracts away data access, ensuring separation of concerns and testability.
* Category Handling: The controller allows the addition of categories by receiving a Category object via HTTP POST. It assumes there's a form submitting data for creating new categories.
* TempData: Used here to pass a success message to be displayed on the subsequent view (Index view), indicating that the category addition request has been received and needs admin approval.
* using Microsoft.AspNetCore.Mvc;
* using ShopBee.Authentication;
* using ShopBee.Models;
* using ShopBee.Repository.IRepository;
* namespace ShopBee.Areas.Store.Controllers
* {
* [Area("Store")]
* [StoreAuthentication()]
* public class CategoryController : Controller
* {
* private readonly IUnitOfWork \_unitOfWork;
* public CategoryController(IUnitOfWork db)
* {
* \_unitOfWork = db;
* }
* public IActionResult Index()
* {
* return View();
* }
* [HttpPost]
* public IActionResult Index(Category obj)
* {
* if (ModelState.IsValid)
* {
* obj.Status = 0;
* \_unitOfWork.Category.Add(obj);
* \_unitOfWork.Save();
* TempData["success"] = "We have received your request, please wait for the Admin team to approve this content, thank you.";
* return RedirectToAction("Index");
* }
* return View();
* }
* }
* }

**Feedback Controller:**

**Controller Structure**

The FeedbackController resides within the Store area ([Area("Store")]) of the application.

[StoreAuthentication()] suggests that this controller might require specific authentication logic related to the store area.

**Constructor Injection**

The controller expects two dependencies injected via the constructor: IUnitOfWork (for data operations) and IWebHostEnvironment (to access the web hosting environment).

**Actions:**

* **Index**(): This action renders a view to display feedback-related information. Presently, it returns an empty view.

**API CALLS:**

* **GetAll**(): An API endpoint accessed via HTTP GET that retrieves feedback related to books associated with the currently logged-in store owner.
* Retrieves the currently logged-in user's ID from the session.
* Uses the IUnitOfWork to retrieve the store ID based on the user ID.
* Retrieves a list of book IDs associated with the store.
* Fetches feedbacks related to these book IDs, including their associated books.
* Returns a JSON response containing the feedback data.

Explanation of Actions:

Dependency Injection: The controller relies on an IUnitOfWork to interact with the data layer and IWebHostEnvironment to access hosting-related information.

GetAll() API Endpoint: Retrieves feedback data related to books associated with the store owner.

Session Usage: Utilizes the session to retrieve the currently logged-in user's ID, presumably the store owner's ID, to fetch relevant feedback.

using Microsoft.AspNetCore.Mvc;

using ShopBee.Authentication;

using ShopBee.Models;

using ShopBee.Repository.IRepository;

using System.Linq;

namespace ShopBee.Areas.Store.Controllers

{

    [Area("Store")]

    [StoreAuthentication()]

    public class FeedbackController : Controller

    {

        private readonly IUnitOfWork \_unitOfWork;

        private readonly IWebHostEnvironment \_webhost;

        public FeedbackController(IUnitOfWork unitOfWork, IWebHostEnvironment webhost)

        {

            \_unitOfWork = unitOfWork;

            \_webhost = webhost;

        }

        public IActionResult Index()

        {

            return View();

        }

        #region API CALLS

        [HttpGet]

        public IActionResult GetAll()

        {

            var UserIdGet = HttpContext.Session.GetString("UserId");

            int.TryParse(UserIdGet, out int storeOwnerId);

            ShopBee.Models.Store store = \_unitOfWork.Store.Get(u => u.UserId == storeOwnerId);

            List<int> bookIds = \_unitOfWork.Book.GetAll().Where(u => u.StoreID == store.Id).Select(b => b.Id).ToList();

            List<Feedback> feedbacks = \_unitOfWork.Feedback.GetAll(includeProperties: "Book").Where(u => bookIds.Contains((int)u.BookId)).ToList();

            return Json(new { data = feedbacks });

        }

        #endregion

    }

}

**Order Controller**

**Controller Structure**

The OrderController resides within the Store area ([Area("Store")]) of the application.

[StoreAuthentication()] suggests that this controller might have specific authentication logic related to the store area.

Constructor Injection

The controller expects an IUnitOfWork dependency to interact with the data layer. This dependency is injected into the constructor.

**Actions:**

* **Index**(): This action renders a view to display a list of orders. Presently, it returns an empty view.
* **Details(int orderId):** This action fetches details of a specific order based on the provided orderId. It retrieves order details from the database, including associated books for each order detail.
* Utilizes the IUnitOfWork to get order details and corresponding book information.
* Populates an OrderVM (Order View Model) with order details and the specific order itself with related user information, returning this information to the view.

**API CALLS:**

* GetAll(): An API endpoint accessed via HTTP GET that retrieves all orders along with user information.
* Retrieves a list of all orders including associated user details from the database.
* Returns a JSON response containing the order data.
* Delete(int id): An API endpoint to delete a specific order by its ID.
* Fetches the order based on the provided ID using IUnitOfWork.
* Removes the order from the repository and saves changes.
* Returns a JSON response indicating the success or failure of the deletion process.

**Explanation of Actions:**

Dependency Injection: The controller relies on an IUnitOfWork to interact with the data layer.

Details() Action: Fetches specific order details and related book information to be displayed in the view using an OrderVM.

API Endpoints: Provides functionality to retrieve all orders and delete a specific order through HTTP GET and DELETE methods, respectively.

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Rendering;

using Microsoft.VisualStudio.Web.CodeGenerators.Mvc.Templates.Blazor;

using ShopBee.Authentication;

using ShopBee.Models;

using ShopBee.Models.ViewModels;

using ShopBee.Repository.IRepository;

namespace ShopBee.Areas.Store.Controllers

{

    [Area("Store")]

    [StoreAuthentication()]

    public class OrderController : Controller

    {

        private readonly IUnitOfWork \_unitOfWork;

        public OrderController(IUnitOfWork db)

        {

            \_unitOfWork = db;

        }

        public IActionResult Index()

        {

            return View();

        }

        public IActionResult Details(int orderId)

        {

            var orderDetails = \_unitOfWork.OrderDetail.GetAll().Where(u => u.OrderId == orderId).ToList();

            foreach (var orderDetail in orderDetails)

            {

                orderDetail.Book = \_unitOfWork.Book.Get(b => b.Id == orderDetail.BookId);

            }

            OrderVM orderVM = new OrderVM()

            {

                DetailsOfOderList = orderDetails,

                Order = \_unitOfWork.Order.Get(u => u.Id == orderId, includeProperties:"User"),

            };

            return View(orderVM);

        }

        #region API CALLS

        [HttpGet]

        public IActionResult GetAll()

        {

            List<Order> obj = \_unitOfWork.Order.GetAll(includeProperties: "User").ToList();

            return Json(new { data = obj });

        }

        [HttpDelete]

        public IActionResult Delete(int id)

        {

            var OrderDelete = \_unitOfWork.Order.Get(u => u.Id == id);

            if (OrderDelete == null)

            {

                return Json(new { success = false, message = "Error while deleting" });

            }

            \_unitOfWork.Order.Remove(OrderDelete); \_unitOfWork.Save();

            return Json(new { success = true, message = "Delete Successful" });

        }

        #endregion

    }

}

#### 2.2.3 Customer

using Microsoft.AspNetCore.Mvc;

using ShopBee.Models;

using ShopBee.Models.ViewModels;

using ShopBee.Repository;

using ShopBee.Repository.IRepository;

namespace ShopBee.Areas.Customer.Controllers

{

    [Area("Customer")]

    public class BookController : Controller

    {

        private readonly IUnitOfWork \_unitOfWork;

        public BookController(IUnitOfWork db)

        {

            \_unitOfWork = db;

        }

        public IActionResult Index()

        {

            return View();

        }

        public IActionResult Details(int? id)

        {

            if (id == null || id == 0)

            {

                return NotFound();

            }

            BookDetailVM bookDetailVM = new BookDetailVM();

            bookDetailVM.book = \_unitOfWork.Book.Get(c=> c.Id == id,includeProperties: "Store,Category");

            bookDetailVM.feedbacks = \_unitOfWork.Feedback.GetFeedbackByBook(bookDetailVM.book.Id);

            return View(bookDetailVM);

        }

        [HttpPost]

        public IActionResult Details(int storeId, int bookId, int quantity)

        {

            if (bookId == null || bookId == 0)

            {

                return NotFound();

            }

            string strUserId = HttpContext.Session.GetString("UserId");

            if (string.IsNullOrEmpty(strUserId)) {

                return RedirectToRoute(new RouteValueDictionary(new

                {

                    area = "Customer",

                    controller = "User",

                    action = "Login",

                }));

            } else

            {

                if (quantity == 0)

                {

                    TempData["error"] = "Out of Stock";

                    return RedirectToAction("Details", bookId);

                }

                int userId = int.Parse(strUserId);

                Cart cart = new Cart();

                cart.StoreID = storeId;

                cart.BookID = bookId;

                cart.UserID = userId;

                cart.Quantity = quantity;

                \_unitOfWork.Cart.Add(cart);

                \_unitOfWork.Save();

                HttpContext.Session.SetString("Cart", \_unitOfWork.Cart.GetNumbersOfItems(userId).ToString());

                BookDetailVM bookDetailVM = new BookDetailVM();

                bookDetailVM.book = \_unitOfWork.Book.Get(c => c.Id == bookId, includeProperties: "Store,Category");

                bookDetailVM.feedbacks = \_unitOfWork.Feedback.GetFeedbackByBook(bookDetailVM.book.Id);

                return RedirectToAction("Details", bookId);

            }

        }

    }

}

**BookController:**

The "BookController" controller is an important component of the ASP.NET Core application on the ShopBee website, located in the Customer area. The source code of this controller is designed to manage and display information about books, creating opportunities for users to visit and explore products on the website.

With the use of Dependency Injection and appropriate namespaces, "BookController" can interact strongly with objects implementing IUnitOfWork. In the action section of the controller:

Index: Returns an interface so that the user can conveniently view the list of books.

Details: Displays detailed information about a book based on ID. If the ID is invalid, the user will be redirected to the NotFound page. Detailed data includes information about the book, related stores and categories, along with user reviews.

using Microsoft.AspNetCore.Mvc;

using ShopBee.Authentication;

using ShopBee.Models;

using ShopBee.Models.ViewModels;

using ShopBee.Repository.IRepository;

namespace ShopBee.Areas.Customer.Controllers

{

    [Area("Customer")]

    [CustomerAuthentication()]

    public class CartController : Controller

    {

        private readonly IUnitOfWork \_unitOfWork;

        public CartController(IUnitOfWork db)

        {

            \_unitOfWork = db;

        }

        public IActionResult Index()

        {

            int userId = int.Parse(HttpContext.Session.GetString("UserId"));

            CartVM cartVM = new CartVM();

            cartVM.carts = \_unitOfWork.Cart.GetCartByUser(userId);

            foreach (var cart in cartVM.carts)

            {

                cartVM.totalPrice = (decimal)(cartVM.totalPrice + (cart.Quantity \* cart.Book.DiscountPrice));

            }

            return View(cartVM);

        }

        public IActionResult RemoveBookToCart(int? id)

        {

            if (id == null || id == 0)

            {

                return NotFound();

            }

            Cart cart = \_unitOfWork.Cart.Get(c=> c.Id == id);

            \_unitOfWork.Cart.Remove(cart);

            \_unitOfWork.Save();

            int userId = int.Parse(HttpContext.Session.GetString("UserId"));

            HttpContext.Session.SetString("Cart", \_unitOfWork.Cart.GetNumbersOfItems(userId).ToString());

            return RedirectToAction("Index");

        }

    }

}

**HomeController:**

The "HomeController" controller in the Customer area of the ShopBee website is the central hub for managing actions related to the home page. Built on the ASP.NET Core platform, this controller's source code demonstrates a focus on providing an intuitive and convenient experience for users when exploring book products on the website.

Through the use of Dependency Injection to integrate IUnitOfWork, "HomeController" has the ability to interact flexibly with the database and methods of the IUnitOfWork object. The main actions of the controller include:

Index: Displays the home page with a diverse list of related books and categories. In addition, providing a book search function based on the search string, helping users easily locate and select.

FilterByCategory: Filter books based on the selected category. If the category ID is invalid, the user will be redirected to the NotFound page. Search results are displayed right on the home page.

FilterByPrice: Filter books based on price. If the ID is invalid, the NotFound page will be displayed. Search results, along with the ability to sort books by ascending or descending price, help users easily sort information.

GetAllBook: Provides a list of all books in JSON format, serving client-side AJAX requests and optimizing the interactive experience.

Overall, "HomeController" not only plays the role of managing home page information but also creates a favorable and positive environment for users to explore and visit book products on the ShopBee website.

using Microsoft.AspNetCore.Mvc;

using ShopBee.Authentication;

using ShopBee.Models;

using ShopBee.Models.ViewModels;

using ShopBee.Repository.IRepository;

using System.Diagnostics;

namespace ShopBee.Areas.Customer.Controllers

{

    [Area("Customer")]

    //[RoleAuthentication()]

    public class HomeController : Controller

    {

        private readonly IUnitOfWork \_unitOfWork;

        public HomeController(IUnitOfWork db)

        {

            \_unitOfWork = db;

        }

        public IActionResult Index(string? searchString)

        {

            HomeVM homeVM = new HomeVM();

            homeVM.categories = \_unitOfWork.Category.GetAll().ToList();

            if (string.IsNullOrEmpty(searchString))

            {

                homeVM.books = \_unitOfWork.Book.GetAll().ToList();

            } else

            {

                homeVM.books = \_unitOfWork.Book.GetBookBySearch(searchString);

            }

            return View(homeVM);

        }

        public IActionResult FilterByCategory(int? id)

        {

            if (id == null || id == 0) {

                return NotFound();

            }

            HomeVM homeVM = new HomeVM();

            homeVM.categories = \_unitOfWork.Category.GetAll().ToList();

            homeVM.books = \_unitOfWork.Book.GetAllBookByCategory(id).ToList();

            return View("Index", homeVM);

        }

        public IActionResult FilterByPrice(int? id)

        {

            if (id == null || id == 0)

            {

                return NotFound();

            }

            HomeVM homeVM = new HomeVM();

            homeVM.categories = \_unitOfWork.Category.GetAll().ToList();

            homeVM.books = \_unitOfWork.Book.GetAllBookSort();

            if (id == 2)

            {

                homeVM.books.Reverse();

            }

            return View("Index", homeVM);

        }

        [HttpGet]

        public IActionResult GetAllBook()

        {

            List<Book> obj = \_unitOfWork.Book.GetAll().ToList();

            return Json(obj);

        }

    }

}  
using Microsoft.AspNetCore.Mvc;

using ShopBee.Authentication;

using ShopBee.Models;

using ShopBee.Models.ViewModels;

using ShopBee.Repository.IRepository;

namespace ShopBee.Areas.Customer.Controllers

{

    [Area("Customer")]

    [CustomerAuthentication()]

    public class OrderController : Controller

    {

        private readonly IUnitOfWork \_unitOfWork;

        public OrderController(IUnitOfWork db)

        {

            \_unitOfWork = db;

        }

        public IActionResult Index()

        {

            int userId = int.Parse(HttpContext.Session.GetString("UserId"));

            OrderUserVM vm = new OrderUserVM();

            vm.orders = \_unitOfWork.Order.GetOrderByUser(userId);

            foreach (Order order in vm.orders)

            {

                List<OrderDetail> orderDetails = \_unitOfWork.OrderDetail.GetOrderDetailsByOrder(order.Id);

                vm.ordersDetail.Add(orderDetails);

            }

            return View(vm);

        }

        public IActionResult Checkout()

        {

            int userId = int.Parse(HttpContext.Session.GetString("UserId"));

            int numberItem = \_unitOfWork.Cart.GetNumbersOfItems(userId);

            if (numberItem == 0)

            {

                return RedirectToAction("Index","Cart", new {area = "Customer"});

            }

            List<Cart> carts = \_unitOfWork.Cart.GetCartByUser(userId);

            List<List<Cart>> cartsFilter = carts.GroupBy(cart => cart.StoreID).Where(group => group.Count() >= 1) .Select(group => group.ToList()).ToList();

            foreach (List<Cart> listcart in cartsFilter)

            {

                Order order = new Order();

                order.UserId = userId;

                order.Quantity = numberItem;

                order.TotalPrice = 0;

                foreach (Cart cart in listcart)

                {

                    order.StoreId = (int)cart.StoreID;

                    order.TotalPrice = (decimal)(order.TotalPrice + (cart.Quantity \* cart.Book.DiscountPrice));

                    \_unitOfWork.Cart.Remove(cart);

                }

                order.Method = "Paypal";

                order.Status = "Pending";

                order.CreateDate = DateTime.Now;

                int orderId = \_unitOfWork.Order.CreateOrder(order);

                foreach (var cart in listcart)

                {

                    OrderDetail orderDetail = new OrderDetail();

                    orderDetail.BookId = cart.BookID;

                    orderDetail.OrderId = orderId;

                    orderDetail.Quantity = (int)cart.Quantity;

                    orderDetail.TotalPrice = (double)(orderDetail.Quantity \* cart.Book.DiscountPrice);

                    Book book = \_unitOfWork.Book.Get(c => c.Id == cart.BookID);

                    book.Stock = book.Stock - cart.Quantity;

                    \_unitOfWork.Book.Update(book);

                    \_unitOfWork.OrderDetail.Add(orderDetail);

                }

            }

            HttpContext.Session.SetString("Cart", \_unitOfWork.Cart.GetNumbersOfItems(userId).ToString());

            \_unitOfWork.Save();

            return RedirectToAction("Index");

        }

    }

}

**OrderController:**

The "OrderController" controller in the Customer section of the ShopBee website plays a key role in the process of managing and processing orders. The code is built on the ASP.NET Core platform and shows special attention to user authentication and data integration through the IUnitOfWork object.

This controller contains two main actions:

Index: Displays a list of the current user's orders, including detailed information about each order. Each order is presented clearly for users to easily track and check.

Checkout: Manage the checkout process, checking the user's cart to ensure that there is at least one line item for checkout. If so, the system creates corresponding orders for each store, along with updating the inventory quantity and removing the paid item from the cart. Order information and the number of items in the cart are updated to reflect the change.

Overall, "OrderController" not only performs the ordering function but also ensures data integrity and provides a consistent and convenient shopping experience for users on the ShopBee website.

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Mvc;

using Newtonsoft.Json;

using ShopBee.Authentication;

using ShopBee.Models;

using ShopBee.Models.ViewModels;

using ShopBee.Repository;

using ShopBee.Repository.IRepository;

namespace ShopBee.Areas.Customer.Controllers

{

    [Area("Customer")]

    //[RoleAuthentication()]

    public class UserController : Controller

    {

        private readonly IUnitOfWork \_unitOfWork;

        private readonly IWebHostEnvironment \_webhost;

        public UserController(IUnitOfWork db, IWebHostEnvironment webhost)

        {

            \_unitOfWork = db;

            \_webhost = webhost;

        }

        public IActionResult Login()

        {

            return View();

        }

        [HttpPost]

        public IActionResult Login(string email, string password)

        {

            User user = \_unitOfWork.User.Login(email, password);

            if (user != null)

            {

                HttpContext.Session.SetString("UserName", user.Name);

                HttpContext.Session.SetString("UserId", user.Id.ToString());

                HttpContext.Session.SetString("UserAvt", user.avtURL);

                var userRoles = \_unitOfWork.User.GetUserRoles(user.Id);

                HttpContext.Session.SetString("UserRoles", userRoles);

                HttpContext.Session.SetString("Cart", \_unitOfWork.Cart.GetNumbersOfItems(user.Id).ToString());

                return RedirectToAction("Index", "Home", new { area = "Customer" });

            }

            else

            {

                TempData["error"] = "Invalid Account";

                return View();

            }

            /\*else if (email.ToLower() == "customer" && password.ToLower() == "customer")

            {

                HttpContext.Session.SetString("UserRoles", "Customer");

            }\*/

           ;

        }

        public IActionResult Register()

        {

            return View();

        }

        [HttpPost]

        public IActionResult Register(User user, int gender, IFormFile file)

        {

            string wwwRootPath = \_webhost.WebRootPath;

            if (file != null)

            {

                string fileName = Guid.NewGuid().ToString() + Path.GetExtension(file.FileName);

                string avtPath = Path.Combine(wwwRootPath, "img/userAvt");

                using (var fileStream = new FileStream(Path.Combine(avtPath, fileName), FileMode.Create))

                {

                    file.CopyTo(fileStream);

                }

                user.avtURL = @"/img/userAvt/" + fileName;

            }

            //user.avtURL = "4cc2377e-8594-43ee-9022-3f72815880dd.jpg";

            user.CreateDate = DateTime.Now;

            user.ModifyDate = DateTime.Now;

            if (gender == 0)

            {

                user.Gender = Models.User.GenderType.Male;

            }

            else if (gender == 1)

            {

                user.Gender = Models.User.GenderType.Female;

            }

            \_unitOfWork.User.Add(user);

            TempData["success"] = "Account created succesfully";

            \_unitOfWork.Save();

            return RedirectToAction("Login");

        }

        public IActionResult Logout()

        {

            HttpContext.Session.Remove("UserId");

            HttpContext.Session.Remove("UserName");

            HttpContext.Session.Remove("UserRoles");

            HttpContext.Session.Remove("UserAvt");

            HttpContext.Session.Remove("Cart");

            return RedirectToAction("Index", "Home", new { area = "Customer" });

        }

        public IActionResult EditProfile()

        {

            int userId = int.Parse(HttpContext.Session.GetString("UserId"));

            User user = \_unitOfWork.User.Get(b => b.Id == userId);

            return View(user);

        }

        [HttpPost]

        public IActionResult EditProfile(IFormFile? file, int gender, User user, string password)

        {

            if (!\_unitOfWork.User.CheckPassword(user.Id, password))

            {

                TempData["error"] = "Wrong Password";

            } else

            {

                string wwwRoothPath = \_webhost.WebRootPath;

                if (file != null)

                {

                    string fileName = Guid.NewGuid().ToString() + Path.GetExtension(file.FileName);

                    string bookPath = Path.Combine(wwwRoothPath, "img/userAvt");

                        // Delete Old Image

                        var oldImagePath = Path.Combine(wwwRoothPath, user.avtURL.TrimStart('/'));

                        if (System.IO.File.Exists(oldImagePath))

                        {

                            System.IO.File.Delete(oldImagePath);

                        }

                    using (var fileStream = new FileStream(Path.Combine(bookPath, fileName), FileMode.Create))

                    {

                        file.CopyTo(fileStream);

                    }

                    user.avtURL = @"/img/userAvt/" + fileName;

                }

                if (gender == 0)

                {

                    user.Gender = Models.User.GenderType.Male;

                }

                else if (gender == 1)

                {

                    user.Gender = Models.User.GenderType.Female;

                }

                \_unitOfWork.User.Update(user);

                HttpContext.Session.SetString("UserAvt", user.avtURL);

                \_unitOfWork.Save();

                TempData["success"] = "Change Successfully";

            }

            return View(user);

        }

        public IActionResult ChangePassword()

        {

            return View();

        }

        [HttpPost]

        public IActionResult ChangePassword(string currentPassword,string newPassword, string confirmNewPassword)

        {

            if (newPassword != confirmNewPassword)

            {

                TempData["error"] = "Password Doesn't Match";

            } else

            {

                int userId = int.Parse(HttpContext.Session.GetString("UserId"));

                if (!\_unitOfWork.User.CheckPassword(userId, currentPassword))

                {

                    TempData["error"] = "Invalid Password";

                } else

                {

                    User user = \_unitOfWork.User.Get(b => b.Id == userId);

                    user.Password = newPassword;

                    \_unitOfWork.User.Update(user);

                    \_unitOfWork.Save();

                    TempData["success"] = "Update Password Successfully";

                }

            }

            return View();

        }

        public IActionResult AccessDenied()

        {

            return View();

        }

    }

}

**UserController:**

The "UserController" controller in the Customer section of the ShopBee website plays an important role in managing user-related functions. Below is a summary description of the important actions in this source code:

Login: This action handles the user login process. The user provides login information, including email and password. If the login information is correct, the user is redirected to the home page of the customer section. Information such as the user's name, ID, and role are stored in the session.

Register: This action allows users to create a new account by providing personal information and an avatar. After successful registration, users can log in and use the website's functions.

Logout: This action takes the user out of the system, deletes information related to their session and redirects them to the home page.

Edit profile (EditProfile): Allows users to edit their personal information and avatar.

Change Password: Allows users to change their password after verifying the current password. If the new password does not match, the user will receive a corresponding notification.

Access Denied Notice (AccessDenied): Displays a notification page when a user tries to access a part of the website that they do not have permission to access.

The code integrates ASP.NET Core Identity to efficiently manage authentication and user management. Additionally, it uses IUnitOfWork to interact with the database and perform operations related to users and accounts.

### 2.3 View

#### 2.3.1 Admin

This is Home/Index:

@{

    var bookModel = ViewBag.BookModel as BookVM;

    var userModel = ViewBag.UserModel as UserVM;

    var orderModel = ViewBag.OrderModel as OrderVM;

    var storeModel = ViewBag.StoreModel as StoreVM;

}

<!-- dashboard inner -->

<div class="dashboard">

    <div class="container-fluid">

        <div class="row column\_title">

            <div class="col-md-12">

                <div class="page\_title">

                    <h2>Dashboard</h2>

                </div>

            </div>

        </div>

        <div class="row column1">

            <div class="col-md-6 col-lg-3">

                <div class="full counter\_section margin\_bottom\_30">

                    <div class="couter\_icon">

                        <div>

                            <i class="fa-solid fa-user fa-xl" style="color: #f2cd18;"></i>

                        </div>

                    </div>

                    <div class="counter\_no">

                        <div>

                            <h5 style="padding: 25px;">User: @userModel.NumberOfUsers</h5>

                        </div>

                    </div>

                </div>

            </div>

            <div class="col-md-6 col-lg-3">

                <div class="full counter\_section margin\_bottom\_30">

                    <div class="couter\_icon">

                        <div>

                            <i class="fa-solid fa-book fa-xl" style="color: #ff9500;"></i>

                        </div>

                    </div>

                    <div class="counter\_no">

                        <div>

                            <h5 style="padding: 25px;">Books:  @bookModel.NumberOfBooks</h5>

                        </div>

                    </div>

                </div>

            </div>

            <div class="col-md-6 col-lg-3">

                <div class="full counter\_section margin\_bottom\_30">

                    <div class="couter\_icon">

                        <div>

                            <i class="fa-solid fa-cart-shopping" style="color: #d40808;"></i>

                        </div>

                    </div>

                    <div class="counter\_no">

                        <div>

                            <h5 style="padding:25px 10px;">Orders:  @orderModel.NumberOfOrders</h5>

                        </div>

                    </div>

                </div>

            </div>

            <div class="col-md-6 col-lg-3">

                <div class="full counter\_section margin\_bottom\_30">

                    <div class="couter\_icon">

                        <div>

                            <i class="fa-solid fa-store fa-xl" style="color: #00ff4c;"></i>

                        </div>

                    </div>

                    <div class="counter\_no">

                        <div>

                            <h5 style="padding:25px 10px;">Stores:  @storeModel.NumberOfStores</h5>

                        </div>

                    </div>

                </div>

            </div>

        </div>

    </div>

</div>

<script>

    let myChart = document.getElementById('myChart').getContext('2d');

    // Global Options

    Chart.defaults.global.defaultFontFamily = 'Lato';

    Chart.defaults.global.defaultFontSize = 18;

    Chart.defaults.global.defaultFontColor = '#777';

    let massPopChart = new Chart(myChart, {

        type: 'bar', // bar, horizontalBar, pie, line, doughnut, radar, polarArea

        data: {

            labels: ['Boston', 'Worcester', 'Springfield', 'Lowell', 'Cambridge', 'New Bedford'],

            datasets: [{

                label: 'Population',

                data: [

                    617594,

                    181045,

                    153060,

                    106519,

                    105162,

                    95072

                ],

                //backgroundColor:'green',

                backgroundColor: [

                    'rgba(255, 99, 132, 0.6)',

                    'rgba(54, 162, 235, 0.6)',

                    'rgba(255, 206, 86, 0.6)',

                    'rgba(75, 192, 192, 0.6)',

                    'rgba(153, 102, 255, 0.6)',

                    'rgba(255, 159, 64, 0.6)',

                    'rgba(255, 99, 132, 0.6)'

                ],

                borderWidth: 1,

                borderColor: '#777',

                hoverBorderWidth: 3,

                hoverBorderColor: '#000'

            }]

        },

        options: {

            title: {

                display: true,

                text: 'Largest Cities In Massachusetts',

                fontSize: 25

            },

            legend: {

                display: true,

                position: 'right',

                labels: {

                    fontColor: '#000'

                }

            },

            layout: {

                padding: {

                    left: 50,

                    right: 0,

                    bottom: 0,

                    top: 0

                }

            },

            tooltips: {

                enabled: true

            }

        }

    });

</script>

<!-- jQuery -->

<script src="~/js/admin/jquery.min.js"></script>

<script src="~/js/admin/popper.min.js"></script>

<script src="~/js/admin/bootstrap.min.js"></script>

<!-- chart js -->

<script src="~/js/admin/Chart.min.js"></script>

<script src="~/js/admin/Chart.bundle.min.js"></script>

<script src="~/js/admin/utils.js"></script>

<script src="~/js/admin/analyser.js"></script>

<!-- wow animation -->

<script src="~/js/admin/animate.js"></script>

<!-- select country -->

<script src="~/js/admin/bootstrap-select.js"></script>

<!-- owl carousel -->

<script src="~/js/admin/owl.carousel.js"></script>

<!-- nice scrollbar -->

<script src="~/js/admin/perfect-scrollbar.min.js"></script>

<!-- sidebar -->

<script>

    var ps = new PerfectScrollbar('#sidebar');

</script>

<!-- custom js -->

<script src="~/js/admin/custom.js"></script>

**This is Admin/Book/Createupdate.cshtml**

@model BookVM

<h1> @(Model.Book.Id == 0 ? "Create" : "Edit") a Book</h1>

<form method="post" enctype="multipart/form-data">

    <div class="border p-3 mt-4">

        <div class="row">

            <input asp-for="Book.Id" hidden />

            <input asp-for="Book.CreateDate" hidden />

            <div class="col-10">

                <div class="mb-3 row p-1">

                    <label asp-for="Book.Name" class="p-0"></label>

                    <input asp-for="Book.Name" class="form-control" />

                    <span asp-validation-for="Book.Name" class="text-danger"></span>

                </div>

                <div class="row mb-3">

                    <label asp-for="Book.Store"></label>

                    <select asp-for="@Model.Book.StoreID" asp-items="@Model.MyStores" class="form-select form-control "></select>

                </div>

                <div class="row mb-3">

                    <label asp-for="Book.CategoryId"></label>

                    <select asp-for="@Model.Book.CategoryId" asp-items="@Model.MyCategories" class=" form-select form-control"></select>

                </div>

                <div class="row mb-3">

                    <label asp-for="Book.ActualPrice"></label>

                    <span asp-validation-for="Book.ActualPrice" class="text-danger"></span>

                    <input asp-for="Book.ActualPrice" class="form-control" >

                </div>

                <div class="row mb-3">

                    <label asp-for="Book.DiscountPrice" class="control-label">Discount Price $</label>

                    <input asp-for="Book.DiscountPrice" class="form-control" />

                    <span asp-validation-for="Book.DiscountPrice" class="text-danger"></span>

                </div>

                <div class="row mb-3">

                    <label asp-for="Book.Stock" class="control-label">Stock</label>

                    <input asp-for="Book.Stock" class="form-control" />

                    <span asp-validation-for="Book.Stock" class="text-danger"></span>

                </div>

                <div class="row mb-3">

                    <label asp-for="Book.Author"></label>

                    <input asp-for="Book.Author" class="form-control" />

                    <span asp-validation-for="Book.Author" class="text-danger"></span>

                </div>

                <div class="row mb-3">

                    <label asp-for="Book.Description"></label>

                    <input asp-for="Book.Description" class="form-control" />

                    <span asp-validation-for="Book.Description" class="text-danger"></span>

                </div>

                <div class="row mb-3">

                    <label asp-for="Book.ImgUrl"></label>

                    <img class="col-2" src="@Model.Book.ImgUrl"  />

                    <input type="file" name="file" class="form-control">

                </div>

                <div class="row mb-3" role="group">

                    @if (Model.Book.Id == 0)

                    {

                        <button type="submit" class="btn btn-primary">Create</button>

                    }

                    else

                    {

                        <button type="submit" class="btn btn-primary">Update</button>

                    }

                    <a asp-controller="Book" asp-action="Index" class="btn btn-primary">

                        Back to List

                    </a>

                </div>

            </div>

        </div>

    </div>

</form>

@section Scripts {

    @{

        <partial name="\_ValidationScriptsPartial" />

    }

}

**This is Admin/Book/Index.cshtml**

@{

    ViewData["Title"] = "Book Management";

}

@model List<Book>

<div class="card shadow border-0 my-4">

    <div class="card-header bg-secondary bg-gradient ml-0 py-3">

        <div class="row">

            <div class="col-12 text-center">

                <h2 class="py-2">Book Managment</h2>

            </div>

        </div>

    </div>

    <div class="card-body p-5">

        <div class="row pb-3">

            <div class="col-6">

            </div>

            <div class="col-6 text-end">

                <a asp-action="CreateUpdate" asp-controller="Book" class="btn btn-primary">

                    <i class="bi bi-plus-circle"></i>  Create New Book

                </a>

            </div>

        </div>

        <table id="tblDataBookAdmin" class="table table-bordered table-striped">

            <thead>

                <tr>

                    <th>Id</th>

                    <th>Book Name</th>

                    <th>Store Owner</th>

                    <th>Category Name</th>

                    <th>Actual Price</th>

                    <th>Discount Price</th>

                    <th>Stock</th>

                    <th>Author</th>

                    <th>Description</th>

                    <th>Createdate</th>

                    <th>ModifyDate</th>

                    <th>Action</th>

                </tr>

            </thead>

        </table>

    </div>

</div>

@section Scripts {

    <script src="~/js/table.js"></script>

}

**This is Admin/Category/Create.cshtml page**

@model Category

<form method="POST">

    <div class="boder p-3 mt-4">

        <div class="row pb-2">

            <h2 class="text-primary">Create Category</h2>

            <hr/>

        </div>

        <div class="p-0" asp-validation-summary="All"></div>

        <div class="mb-3 row p-1">

            <label asp-for="Name" class="p-0"></label>

            <input asp-for="Name" class="form-control" />

            <span asp-validation-for="Name" class="text-danger"></span>

        </div>

        <div class="mb-3 row p-1">

            <label asp-for="Status" class="p-0"></label>

            <select asp-for="Status" class="form-control">

                <option value="1">Active</option>

                <option value="0">Pending</option>

            </select>

            <span asp-validation-for="Status" class="text-danger"></span>

        </div>

        <div class="row">

            <button type="submit" class="btn btn-outline-primary form-control" style="width:150px">Create</button>

            <div class ="col-6 col-md-3">

                <a asp-controller="Category" asp-action="Index" type="submit" class="btn btn-outline-secondary form-control" style="width:150px">Go Back</a>

            </div>

        </div>

    </div>

</form>

@section Scripts{

    @{

        <partial name="\_ValidationScriptsPartial"/>

    }

}

**This is Edit View Page Admin**

@model Category

<form method="POST">

    <input asp-for="Id" hidden />

    <div class="boder p-3 mt-4">

        <div class="row pb-2">

            <h2 class="text-primary">Edit Category</h2>

            <hr />

        </div>

        <div class="p-0" asp-validation-summary="All"></div>

        <div class="mb-3 row p-1">

            <label asp-for="Name" class="p-0"></label>

            <input asp-for="Name" class="form-control" />

            <span asp-validation-for="Name" class="text-danger"></span>

        </div>

        <div class="mb-3 row p-1">

            <label asp-for="Status" class="p-0"></label>

            <select asp-for="Status" class="form-control">

                <option value="1">Active</option>

                <option value="0">Pending</option>

            </select>

            <span asp-validation-for="Status" class="text-danger"></span>

        </div>

        <div class="row">

            <button type="submit" class="btn btn-outline-primary form-control" style="width:150px">Update</button>

            <div class="col-6 col-md-3">

                <a asp-controller="Category" asp-action="Index" type="submit" class="btn btn-outline-secondary form-control" style="width:150px">Go Back</a>

            </div>

        </div>

    </div>

</form>

@section Scripts {

    @{

        <partial name="\_ValidationScriptsPartial" />

    }

}

**This is View Index Admin Page**

@{

    ViewData["Title"] = "Category Management";

}

@model List<Category>

<div class="card shadow border-0 my-4">

    <div class="card-header bg-secondary bg-gradient ml-0 py-3">

        <div class="row">

            <div class="col-12 text-center">

                <h2 class="py-2">Category Managment</h2>

            </div>

        </div>

    </div>

    <div class="card-body p-4">

        <div class="row pb-3">

            <div class="col-6">

            </div>

            <div class="col-6 text-end">

                <a asp-action="Create" asp-controller="Category" class="btn btn-primary">

                    <i class="bi bi-plus-circle"></i>  Create New Category

                </a>

            </div>

        </div>

        <table id="tblDataCategory" class="table table-bordered table-striped">

            <thead>

                <tr>

                    <th>Id</th>

                    <th>

                        Category Name

                    </th>

                    <th>

                        Status

                    </th>

                    <th>

                        Action

                    </th>

                </tr>

            </thead>

        </table>

    </div>

</div>

@section Scripts {

    <script src="~/js/table.js"></script>

}

**This is Order Index View Admin Page**

@{

    ViewData["Title"] = "Order Management";

}

@model List<Order>

<div class="card shadow border-0 my-4">

    <div class="card-header bg-secondary bg-gradient ml-0 py-3">

        <div class="row">

            <div class="col-12 text-center">

                <h2 class="py-2">Order Managment</h2>

            </div>

        </div>

    </div>

    <div class="card-body p-4">

        <table id="tblDataOrderAdmin" class="table table-bordered table-striped">

            <thead>

                <tr>

                    <th>User</th>

                    <th>TotalPrice</th>

                    <th>Method</th>

                    <th>CreateDate</th>

                    <th>Status</th>

                    <th>Action</th>

                </tr>

            </thead>

        </table>

    </div>

</div>

@section Scripts {

    <script src="~/js/table.js"></script>

}

**This is of Store Admin Create View Page**

@model StoreVM

<form method="POST">

    <div class="boder p-3 mt-4">

        <div class="row pb-2">

            <h2 class="text-primary">Create Store</h2>

            <hr/>

        </div>

        <div class="p-0" asp-validation-summary="All"></div>

        <div class="mb-3 row p-1">

            <label asp-for="Store.Name" class="p-0"></label>

            <input asp-for="Store.Name" class="form-control" />

            <span asp-validation-for="Store.Name" class="text-danger"></span>

        </div>

        <div class="row mb-3">

            <label asp-for="Store.User"></label>

            <select asp-for="@Model.Store.UserId" asp-items="@Model.MyUsers" class="form-select form-control "></select>

        </div>

        <div class="row">

            <button type="submit" class="btn btn-outline-primary form-control" style="width:150px">Create</button>

            <div class ="col-6 col-md-3">

                <a asp-controller="Store" asp-action="Index" type="submit" class="btn btn-outline-secondary form-control" style="width:150px">Go Back</a>

            </div>

        </div>

    </div>

</form>

@section Scripts{

    @{

        <partial name="\_ValidationScriptsPartial"/>

    }

}

**Edit View page Admin**

@model StoreVM

<form method="POST">

    <input asp-for="Store.Id" hidden />

    <input asp-for="Store.CreateDate" hidden />

    <div class="boder p-3 mt-4">

        <div class="row pb-2">

            <h2 class="text-primary">Edit Store</h2>

            <hr />

        </div>

        <div class="p-0" asp-validation-summary="All"></div>

        <div class="mb-3 row p-1">

            <label asp-for="Store.Name" class="p-0"></label>

            <input asp-for="Store.Name" class="form-control" />

            <span asp-validation-for="Store.Name" class="text-danger"></span>

        </div>

        <div class="row mb-3">

            <label asp-for="Store.User"></label>

            <select asp-for="@Model.Store.UserId" asp-items="@Model.MyUsers" class="form-select form-control "></select>

        </div>

        <div class="row">

            <button type="submit" class="btn btn-outline-primary form-control" style="width:150px">Update</button>

            <div class="col-6 col-md-3">

                <a asp-controller="Store" asp-action="Index" type="submit" class="btn btn-outline-secondary form-control" style="width:150px">Go Back</a>

            </div>

        </div>

    </div>

</form>

@section Scripts {

    @{

        <partial name="\_ValidationScriptsPartial" />

    }

}

**This is Store Index View Page Admin**

@{

    ViewData["Title"] = "Store Management";

}

@model List<Store>

<div class="card shadow border-0 my-4">

    <div class="card-header bg-secondary bg-gradient ml-0 py-3">

        <div class="row">

            <div class="col-12 text-center">

                <h2 class="py-2">Store Managment</h2>

            </div>

        </div>

    </div>

    <div class="card-body p-4">

        <div class="row pb-3">

            <div class="col-6">

            </div>

            <div class="col-6 text-end">

                <a asp-action="Create" asp-controller="Store" class="btn btn-primary">

                    <i class="bi bi-plus-circle"></i>  Create New Store

                </a>

            </div>

        </div>

        <table id="tblDataStoreAdmin" class="table table-bordered table-striped">

            <thead>

                <tr>

                    <th>Id</th>

                    <th>Name</th>

                    <th>User</th>

                    <th>Create Date</th>

                    <th>Action</th>

                </tr>

            </thead>

        </table>

    </div>

</div>

@section Scripts {

    <script src="~/js/table.js"></script>

}

**User CreateUpdate Admin View Page**

@model UserVM

<form method="POST" enctype="multipart/form-data">

    <div class="boder p-3 mt-4" style="padding-bottom:100px">

        <input asp-for="User.Id" hidden />

        <div class="row pb-2">

            <h1> @(Model.User.Id == 0 ? "Create" : "Edit") a User</h1>

        </div>

        <div class="p-0" asp-validation-summary="All"></div>

        <div class="mb-3 row p-1">

            <label asp-for="User.Email" class="p-0"></label>

            <input asp-for="User.Email" class="form-control" />

            <span asp-validation-for="User.Email" class="text-danger"></span>

        </div>

        <div class="mb-3 row p-1">

            <label asp-for="User.Password" class="p-0"></label>

            <input asp-for="User.Password" class="form-control" />

            <span asp-validation-for="User.Password" class="text-danger"></span>

        </div>

        <div class="mb-3 row p-1">

            <label asp-for="User.Name" class="p-0"></label>

            <input asp-for="User.Name" class="form-control" />

            <span asp-validation-for="User.Name" class="text-danger"></span>

        </div>

        <div class="mb-3 row p-1">

            <label asp-for="User.Gender" class="p-0"></label>

            <input asp-for="User.Gender" class="form-control" />

            <span asp-validation-for="User.Gender" class="text-danger"></span>

        </div>

        <div class="mb-3 row p-1">

            <label asp-for="User.Phone" class="p-0"></label>

            <input asp-for="User.Phone" class="form-control" />

            <span asp-validation-for="User.Phone" class="text-danger"></span>

        </div>

        <div class="mb-3 row p-1">

            <label asp-for="User.Adress" class="p-0"></label>

            <input asp-for="User.Adress" class="form-control" />

            <span asp-validation-for="User.Adress" class="text-danger"></span>

        </div>

        <div class="row mb-3">

            <label asp-for="User.avtURL"></label>

            <img class="col-2" src="@Model.User.avtURL" style="border-radius:5px; border:1px; solid: #bbb9b9; width=100%" />

            <input type="file" name="file" class="form-control">

        </div>

        <div class="btn group" role="group">

            @if (Model.User.Id == 0)

            {

                <button type="submit" class="btn btn-primary">Create</button>

            }

            else

            {

                <button type="submit" class="btn btn-primary">Update</button>

            }

            <a asp-controller="User" asp-action="Index" class="btn btn-primary">

                Back to List

            </a>

        </div>

    </div>

</form>

@section Scripts{

    @{

        <partial name="\_ValidationScriptsPartial"/>

    }

}

**This is User Admin Index View Page**

@{

    ViewData["Title"] = "User Management";

}

@model List<User>

<div class="card shadow border-0 my-4">

    <div class="card-header bg-secondary bg-gradient ml-0 py-3">

        <div class="row">

            <div class="col-12 text-center">

                <h2 class="py-2">Category Managment</h2>

            </div>

        </div>

    </div>

    <div class="card-body p-4">

        <div class="row pb-3">

            <div class="col-6">

            </div>

            <div class="col-6 text-end">

                <a asp-action="CreateUpdate" asp-controller="User" class="btn btn-primary">

                    <i class="bi bi-plus-circle"></i>  Create New User

                </a>

            </div>

        </div>

        <table id="tblDataUserAdmin" class="table table-bordered table-striped">

            <thead>

                <tr>

                    <th>Id</th>

                    <th>Email </th>

                    <th>Phone </th>

                    <th>Password</th>

                    <th>Name</th>

                    <th>Gender</th>

                    <th>Address</th>

                    <th>Actions</th>

                </tr>

            </thead>

        </table>

    </div>

</div>

@section Scripts {

    <script src="~/js/table.js"></script>

}

**This is Admin CreateUpdate User View Page**

@model UserVM

<form method="POST" enctype="multipart/form-data">

    <div class="boder p-3 mt-4" style="padding-bottom:100px">

        <div class="row pb-2">

            <h1> Create a User Role</h1>

        </div>

        <div class="p-0" asp-validation-summary="All"></div>

        <div class="mb-3 row p-1">

            <label asp-for="@Model.MyUsers" class="p-0">Email</label>

            <input name="email\_temp" class="form-control" />

        </div>

        <div class="row mb-3">

            <label asp-for="UserRole.RoleId"></label>

            <select asp-for="@Model.UserRole.RoleId" asp-items="@Model.MyRoles" class=" form-select form-control"></select>

        </div>

        <div class="btn group" role="group">

            <button type="submit" class="btn btn-primary">Create</button>

            <a asp-controller="UserRole" asp-action="Index" class="btn btn-primary">

                Back to List

            </a>

        </div>

    </div>

</form>

@section Scripts{

    @{

        <partial name="\_ValidationScriptsPartial"/>

    }

}

**This is Admin UserRole Index View Page**

@{

    ViewData["Title"] = "User Role Management";

}

<div class="card shadow border-0 my-4">

    <div class="card-header bg-secondary bg-gradient ml-0 py-3">

        <div class="row">

            <div class="col-12 text-center">

                <h2 class="py-2">User Role Managment</h2>

            </div>

        </div>

    </div>

    <div class="row pb-3">

        <div class="col-6">

        </div>

        <div class="col-6 text-end">

            <a asp-action="CreateUpdate" asp-controller="UserRole" class="btn btn-primary">

                <i class="bi bi-plus-circle"></i>  Create New User Role

            </a>

        </div>

    </div>

    <div class="card-body p-4">

        <div class="row pb-3">

            <div class="col-6">

            </div>

        </div>

        <table id="tblDataUserRoleAdmin" class="table table-bordered table-striped">

            <thead>

                <tr>

                    <th>Email</th>

                    <th>Role Active</th>

                    <th>Actions</th>

                </tr>

            </thead>

        </table>

    </div>

</div>

@section Scripts {

    <script src="~/js/table.js"></script>

}

#### 2.3.2 Store

**CreateUpdate Book:**

@using ShopBee.Models.ViewModels

@model BookVM

<h1> @(Model.Book.Id == 0 ? "Create" : "Edit") a Book</h1>

<form method="post" enctype="multipart/form-data">

    <div class="border p-3 mt-4">

        <div class="row">

            <input asp-for="Book.Id" hidden />

            <input asp-for="Book.CreateDate" hidden />

            <input asp-for="@Model.Book.StoreID" hidden />

            <div class="col-10">

                <div class="mb-3 row p-1">

                    <label asp-for="Book.Name" class="p-0"></label>

                    <input asp-for="Book.Name" class="form-control" />

                    <span asp-validation-for="Book.Name" class="text-danger"></span>

                </div>

                <div class="row mb-3">

                    <label asp-for="Book.CategoryId"></label>

                    <select asp-for="@Model.Book.CategoryId" asp-items="@Model.MyCategories" class=" form-select form-control"></select>

                </div>

                <div class="row mb-3">

                    <label asp-for="Book.ActualPrice"></label>

                    <span asp-validation-for="Book.ActualPrice" class="text-danger"></span>

                    <input asp-for="Book.ActualPrice" class="form-control" >

                </div>

                <div class="row mb-3">

                    <label asp-for="Book.DiscountPrice" class="control-label">Discount Price $</label>

                    <input asp-for="Book.DiscountPrice" class="form-control" />

                    <span asp-validation-for="Book.DiscountPrice" class="text-danger"></span>

                </div>

                <div class="row mb-3">

                    <label asp-for="Book.Stock" class="control-label">Stock</label>

                    <input asp-for="Book.Stock" class="form-control" />

                    <span asp-validation-for="Book.Stock" class="text-danger"></span>

                </div>

                <div class="row mb-3">

                    <label asp-for="Book.Author"></label>

                    <input asp-for="Book.Author" class="form-control" />

                    <span asp-validation-for="Book.Author" class="text-danger"></span>

                </div>

                <div class="row mb-3">

                    <label asp-for="Book.Description"></label>

                    <input asp-for="Book.Description" class="form-control" />

                    <span asp-validation-for="Book.Description" class="text-danger"></span>

                </div>

                <div class="row mb-3">

                    <label asp-for="Book.ImgUrl"></label>

                    <img class="col-2" src="@Model.Book.ImgUrl"  />

                    <input type="file" name="file" class="form-control">

                </div>

                <div class="row mb-3" role="group">

                    @if (Model.Book.Id == 0)

                    {

                        <button type="submit" class="btn btn-primary">Create</button>

                    }

                    else

                    {

                        <button type="submit" class="btn btn-primary">Update</button>

                    }

                    <a asp-controller="Book" asp-action="Index" class="btn btn-primary">

                        Back to List

                    </a>

                </div>

            </div>

        </div>

    </div>

</form>

@section Scripts {

    @{

        <partial name="\_ValidationScriptsPartial" />

    }

}

**Index of book in Store:**

@{

    ViewData["Title"] = "Book Management";

}

@model List<Book>

<div class="card shadow border-0 my-4">

    <div class="card-header bg-secondary bg-gradient ml-0 py-3">

        <div class="row">

            <div class="col-12 text-center">

                <h2 class="py-2">Book Managment</h2>

            </div>

        </div>

    </div>

    <div class="card-body p-5">

        <div class="row pb-3">

            <div class="col-6">

            </div>

            <div class="col-6 text-end">

                <a asp-action="CreateUpdate" asp-controller="Book" class="btn btn-primary">

                    <i class="bi bi-plus-circle"></i>  Create New Book

                </a>

            </div>

        </div>

        <table id="tblDataBookStoreOwner" class="table table-bordered table-striped">

            <thead>

                <tr>

                    <th>Id</th>

                    <th>Book Name</th>

                    <th>Category Name</th>

                    <th>Actual Price</th>

                    <th>Discount Price</th>

                    <th>Stock</th>

                    <th>Author</th>

                    <th>Description</th>

                    <th>Actions</th>

                </tr>

            </thead>

        </table>

    </div>

</div>

@section Scripts {

    <script src="~/js/table.js"></script>

}

**Category Index:**

@model Category

<form method="POST">

    <div class="boder p-3 mt-4">

        <div class="row pb-2">

            <h2 class="text-primary">Create Category</h2>

            <hr />

        </div>

        <div class="p-0" asp-validation-summary="All"></div>

        <div class="mb-3 row p-1">

            <label asp-for="Name" class="p-0"></label>

            <input asp-for="Name" class="form-control" />

            <span asp-validation-for="Name" class="text-danger"></span>

        </div>

        <div class="row">

            <button type="submit" class="btn btn-outline-primary form-control" style="width:150px">Create</button>

            <div class="col-6 col-md-3">

                <a asp-controller="Category" asp-action="Index" type="submit" class="btn btn-outline-secondary form-control" style="width:150px">Go Back</a>

            </div>

        </div>

    </div>

</form>

@section Scripts {

    @{

        <partial name="\_ValidationScriptsPartial" />

    }

}

**Feedback Index :**

@{

    ViewData["Title"] = "Feedback Management";

}

@model List<Book>

<div class="card shadow border-0 my-4">

    <div class="card-header bg-secondary bg-gradient ml-0 py-3">

        <div class="row">

            <div class="col-12 text-center">

                <h2 class="py-2">Book Managment</h2>

            </div>

        </div>

    </div>

    <div class="card-body p-5">

        <table id="tblDataFeedbackStoreOwner" class="table table-bordered table-striped">

            <thead>

                <tr>

                    <th>Book Name</th>

                    <th>Content</th>

                    <th>Rating</th>

                    <th>Response</th>

                    <th>Last Feedback</th>

                    <th>Action </th>

                </tr>

            </thead>

        </table>

    </div>

</div>

@section Scripts {

    <script src="~/js/table.js"></script>

}

#### 2.3.3 Customer

**Book Details View Page;**

@using ShopBee.Models.ViewModels

@model BookDetailVM;

<head>

    <link rel="stylesheet" href="~/css/customer/productdetail.css">

</head>

<div class="section">

    <div class="image-container">

        <img src="@Model.book.ImgUrl" alt="Product Image" class="product-image">

    </div>

    <div class="product-info">

        <h1>@Model.book.Name</h1>

        <p class="store-name">Store: @Model.book.Store.Name</p>

        @{

            if (Model.book.Stock == 0)

            {

                <p class="product-status-outofstock">Out of Stock</p>

            } else

            {

                <p class="product-status-available">Available</p>

            }

        }

        <p class="author-name">Author: @Model.book.Author</p>

        <p class="category">Category: @Model.book.Category.Name</p>

        <p>@Model.book.Description</p>

        <div class="actual-price">$@Model.book.ActualPrice</div>

        <p> </p>

        <div class="discount-price">$@Model.book.DiscountPrice</div>

        <form method="post">

            <input name="storeId" value="@Model.book.StoreID" hidden />

            <input name="bookId" value="@Model.book.Id" hidden />

            <div class="quantity">

                <label for="quantity">Quantity:</label>

                <button id="decrease">-</button>

                @{

                    if (Model.book.Stock == 0)

                    {

                        <input type="text" name="quantity" id="quantity" value="0" readonly>

                    }

                    else

                    {

                        <input type="text" name="quantity" id="quantity" value="1" readonly>

                    }

                }

                <button type="button" id="increase">+</button>

            </div>

            <button type="submit" class="buy-button">Add To Cart</button>

        </form>

    </div>

</div>

<script>

    document.getElementById('increase').addEventListener('click', function () {

        var quantityInput = document.getElementById('quantity');

        var currentQuantity = parseInt(quantityInput.value);

        var maxStock = parseInt(@Model.book.Stock); // Lấy giá trị Stock từ Model

        if (currentQuantity < maxStock) {

            quantityInput.value = currentQuantity + 1;

        }

    });

    document.getElementById('decrease').addEventListener('click', function () {

        var quantityInput = document.getElementById('quantity');

        var currentQuantity = parseInt(quantityInput.value);

        if (currentQuantity > 1) {

            quantityInput.value = currentQuantity - 1;

        }

    });

</script>

<div class="section feedback-section">

    <h1 class="feedback-header">FEEDBACK</h1>

    <div class="feedback-components">

        @{

            foreach (Feedback feedback in Model.feedbacks)

            {

                <div class="feedback-item">

                    <div class="user-avatar">

                        <img src="@feedback.User.avtURL" alt="User 2 Avatar">

                    </div>

                    <div class="user-info">

                        <div class="user-name">@feedback.User.Name</div>

                        <div class="user-rating">

                            @{

                                for (int i=0; i<feedback.Rating; i++)

                                {

                                    <i class="star-checked far fa-star"></i>

                                }

                                for (int i=0; i<5-feedback.Rating; i++)

                                {

                                    <i class="star-unchecked far fa-star"></i>

                                }

                            }

                        </div>

                        <div class="feedback-text">@feedback.Content</div>

                        <div class="store-feedback">Store response: @feedback.Response</div>

                    </div>

                </div>

            }

            <div class="feedback-item">

                <div class="user-avatar">

                    <img src="~/img/userAvt/64b550d9-fd3e-4677-8727-b3c6295d4fc2.jpg" alt="User 1 Avatar">

                </div>

                <div class="user-info">

                    <div class="user-name">John Doe </div>

                    <div class="user-rating">

                        <i class="star-checked far fa-star"></i>

                        <i class="star-checked far fa-star"></i>

                        <i class="star-checked far fa-star"></i>

                        <i class="star-checked far fa-star"></i>

                        <i class="star-unchecked far fa-star"></i>

                    </div>

                </div>

            </div>

        }

    </div>

    <!-- Add more feedback items as needed -->

</div>

**Cart index View Page**

@using ShopBee.Models.ViewModels

@model CartVM

<head>

    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">

    <link rel="stylesheet" href="~/css/customer/cart.css">

</head>

<div class="container">

    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>

    <h2 class="text-center mb-4">Shobee Cart</h2>

    <ul class="list-group">

        @{

            foreach (Cart cart in Model.carts)

            {

                <li class="list-group-item">

                    <div class="row">

                        <div class="col-md-2">

                            <img src="@cart.Book.ImgUrl" alt="Product 1" class="img-fluid product-image">

                        </div>

                        <div class="col-md-10">

                            <div class="product-details">

                                <div class="product-info">

                                    <p class="product-name">@cart.Book.Name</p>

                                    <p class="product-price">Price: $@cart.Book.DiscountPrice</p>

                                    <p class="product-quantity">Quantity: @cart.Quantity</p>

                                </div>

                                <div class="text-right">

                                    <a asp-area="Customer" asp-controller="Cart" asp-action="RemoveBookToCart" asp-route-id="@cart.Id" class="remove-btn">&times;</a>

                                </div>

                            </div>

                        </div>

                    </div>

                </li>

            }

        }

    </ul>

    <div class="text-right mt-4">

        <p class="font-weight-bold total-amount">Total: $@Model.totalPrice</p>

        <a asp-area="Customer" asp-controller="Order" asp-action="Checkout" class="btn checkout-btn">Checkout</a>

    </div>

</div>

**Home Index View Page**

@using ShopBee.Models.ViewModels

@model HomeVM;

@{

    ViewData["Title"] = "Home Page";

}

<script src="~/js/homepage\_product.js"></script>

<div class="container">

    <div class="grid wide">

        <div class="row sm-gutter">

            <div class="col l-2 m-0 c-0">

                <!-- category -->

                <nav class="category">

                    <h3 class="category-heading">

                        <i class="category-heading-icon fas fa-list-ul"></i>

                        CATEGORY

                    </h3>

                    <div class="category-group">

                        <ul class="category-group-list">

                        @{

                            foreach (var category in Model.categories)

                            {

                                <li class="category-group-item">

                                        <a asp-area="Customer" asp-controller="Home" asp-action="FilterByCategory" asp-route-id="@category.Id" class="category-group-title">

                                        @category.Name

                                    </a>

                                </li>

                            }

                        }

                        </ul>

                    </div>

                </nav>

            </div>

            <div class="col l-10 m-12 c-12">

                <!-- home filter -->

                <div class="home-filter hide-on-mobile-tablet">

                    <div class="home-filter-control">

                        <p class="home-filter-title">Sorted by:</p>

                        <button class="btn btn--primary home-filter-btn">Popular</button>

                        <div class="btn home-filter-sort">

                            <p class="home-filter-sort-btn">Price</p>

                            <i class="fas fa-sort-amount-down-alt"></i>

                            <ul class="home-filter-sort-list">

                                <li>

                                    <a asp-area="Customer" asp-controller="Home" asp-action="FilterByPrice" asp-route-id="1" class="home-filter-sort-item-link">

                                        Descending

                                        <i class="fas fa-sort-amount-down-alt"></i>

                                    </a>

                                </li>

                                <li>

                                    <a asp-area="Customer" asp-controller="Home" asp-action="FilterByPrice" asp-route-id="2" class="home-filter-sort-item-link">

                                        Ascending

                                        <i class="fas fa-sort-amount-up-alt"></i>

                                    </a>

                                </li>

                            </ul>

                        </div>

                    </div>

                </div>

                <!-- home product -->

                <div class="home-product">

                    <div class="row sm-gutter">

                        @{

                            int itemsPerPage = 12;  // Số sản phẩm mỗi trang

                            int totalItems = Model.books.Count();

                            int totalPages = (int)Math.Ceiling((double)totalItems / itemsPerPage);

                            int currentPage = Context.Request.Query["page"].Count > 0 ? Convert.ToInt32(Context.Request.Query["page"]) : 1;

                            int startIndex = (currentPage - 1) \* itemsPerPage;

                            for (int i = startIndex; i < startIndex + itemsPerPage && i < totalItems; i++)

                            {

                                var item = Model.books.ElementAt(i);

                                // Rest of your HTML code for displaying product item

                                <div data="${item.id}" class="col l-3 m-6 c-12 home-product-item">

                                    <a class="home-product-item-link" asp-area="Customer" asp-controller="Book" asp-action="Details" asp-route-id="@item.Id">

                                        <div class="home-product-item\_\_img" style="background-image: url('@item.ImgUrl');"></div>

                                        <div class="home-product-item\_\_info">

                                            <h4 class="home-product-item\_\_name">@item.Name</h4>

                                            <div class="home-product-item\_\_price">

                                                <p class="home-product-item\_\_price-old">@item.ActualPrice $</p>

                                                <p class="home-product-item\_\_price-new">@item.DiscountPrice $</p>

                                                <i class="home-product-item\_\_ship fas fa-shipping-fast"></i>

                                            </div>

                                            <div class="home-product-item\_\_footer">

                                                <p class="author-name">@item.Author</p>

                                                <div class="home-product-item\_\_rating-star">

                                                    <i class="star-checked far fa-star"></i>

                                                    <i class="star-checked far fa-star"></i>

                                                    <i class="star-checked far fa-star"></i>

                                                    <i class="star-checked far fa-star"></i>

                                                    <i class="star-checked far fa-star"></i>

                                                </div>

                                            </div>

                                            <div class="home-product-item\_\_sale-off">

                                                <div class="home-product-item\_\_sale-off-value">@Math.Round((item.ActualPrice - item.DiscountPrice) / item.ActualPrice \* 100)%</div>

                                            </div>

                                        </div>

                                        <div class="home-product-item-footer">View Detail</div>

                                    </a>

                                </div>

                            }

                        }

                    </div>

                </div>

                    @{

                        for (int i =0; i< 10; i++)

                        {

                        }

                    }

                    <!--<div id="list-product" class="row sm-gutter"></div>-->

                <ul class="pagination home-product-pagination">

                    @for (int i = 1; i <= totalPages; i++)

                    {

                        <li class="pagination-item @(i == currentPage ? "pagination-item--active" : "")">

                            <a href="?page=@i" class="pagination-item-link">@i</a>

                        </li>

                    }

                </ul>

            </div>

        </div>

    </div>

</div>

**Privacy View Page of Home**

@{

    ViewData["Title"] = "Privacy Policy";

}

<h1>@ViewData["Title"]</h1>

<p>Use this page to detail your site's privacy policy.</p>

**Index of Order View Page:**

@using ShopBee.Models.ViewModels

@model OrderUserVM

<head>

    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">

    <link rel="stylesheet" href="~/css/customer/order.css">

    <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>

    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>

</head>

<div class="container">

    <h2 class="text-center mb-4">Order Management</h2>

    <ul class="nav nav-tabs">

        <li class="nav-item">

            <a class="nav-link active" id="pending-tab" data-toggle="tab" href="#pending">Pending Orders</a>

        </li>

        <li class="nav-item">

            <a class="nav-link" id="successful-tab" data-toggle="tab" href="#successful">Successful Orders</a>

        </li>

    </ul>

    <div class="tab-content">

        <div class="tab-pane fade show active" id="pending">

            <div class="order-list">

                @{

                    foreach(Order item in Model.orders)

                    {

                        if (item.Status == "Pending")

                        {

                            <div class="order-item">

                                <ul class="product-list">

                                    @{

                                        List<OrderDetail> orderDetails = Model.ordersDetail[Model.orders.IndexOf(item)];

                                        foreach (OrderDetail orderDetail in orderDetails)

                                        {

                                            <li class="product-list-item">

                                                <div class="product-info">

                                                    <div>

                                                        <img src="@orderDetail.Book.ImgUrl" alt="Product 2" class="img-fluid product-image">

                                                        <span class="ml-3">@orderDetail.Book.Name</span>

                                                        <span class="quantity">x @orderDetail.Quantity</span>

                                                    </div>

                                                </div>

                                            </li>

                                        }

                                    }

                                </ul>

                                <p class="store-name">@item.Store.Name</p>

                                <p class="total">Total: $@item.TotalPrice</p>

                                <button class="review-button">Pending</button>

                            </div>

                        }

                    }

                }

            </div>

        </div>

        <div class="tab-pane fade" id="successful">

            <div class="order-list">

                @{

                    foreach (Order item in Model.orders)

                    {

                        if (item.Status == "Successful")

                        {

                            <div class="order-item">

                                <ul class="product-list">

                                    @{

                                        List<OrderDetail> orderDetails = Model.ordersDetail[Model.orders.IndexOf(item)];

                                        foreach (OrderDetail orderDetail in orderDetails)

                                        {

                                            <li class="product-list-item">

                                                <div class="product-info">

                                                    <div>

                                                        <img src="@orderDetail.Book.ImgUrl" alt="Product 2" class="img-fluid product-image">

                                                        <span class="ml-3">@orderDetail.Book.Name</span>

                                                        <span class="quantity">x @orderDetail.Quantity</span>

                                                    </div>

                                                </div>

                                            </li>

                                        }

                                    }

                                </ul>

                                <p class="store-name">@item.Store.Name</p>

                                <p class="total">Total: $@item.TotalPrice</p>

                                <button class="review-button">Feedback</button>

                            </div>

                        }

                    }

                }

            </div>

        </div>

    </div>

</div>

<script>

    $(document).ready(function () {

        $('#pending-tab').on('click', function () {

            $('#successful-tab').removeClass('active');

            $(this).addClass('active');

            $('#successful').removeClass('show active');

            $('#pending').addClass('show active');

        });

        $('#successful-tab').on('click', function () {

            $('#pending-tab').removeClass('active');

            $(this).addClass('active');

            $('#pending').removeClass('show active');

            $('#successful').addClass('show active');

        });

    });

</script>

**Index Of Product View Page**

<head>

    <link rel="stylesheet" href="~/css/customer/productdetail.css">

</head>

<div class="section">

    <div class="image-container">

        <img src="~/img/bookImg/37ae814e-b93d-419e-915f-1678005087d8.png" alt="Product Image" class="product-image">

    </div>

    <div class="product-info">

        <h1>Product Name</h1>

        <p class="product-status">Status: Available</p>

        <p class="author-name">Author: [Tên Tác giả]</p>

        <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed et libero sed odio bibendum convallis.</p>

        <div class="actual-price">$149.99</div>

        <p> </p>

        <div class="discount-price">$129.99</div>

        <div class="quantity">

            <label for="quantity">Quantity:</label>

            <button id="decrease">-</button>

            <input type="text" id="quantity" value="1" readonly>

            <button id="increase">+</button>

        </div>

        <a href="#" class="buy-button">Buy Now</a>

    </div>

</div>

<script>

    document.getElementById('increase').addEventListener('click', function () {

        var quantityInput = document.getElementById('quantity');

        var currentQuantity = parseInt(quantityInput.value);

        quantityInput.value = currentQuantity + 1;

    });

    document.getElementById('decrease').addEventListener('click', function () {

        var quantityInput = document.getElementById('quantity');

        var currentQuantity = parseInt(quantityInput.value);

        if (currentQuantity > 1) {

            quantityInput.value = currentQuantity - 1;

        }

    });

</script>

<div class="section feedback-section">

    <div class="feedback-components">

        <div class="feedback-item">

            <div class="user-avatar">

                <img src="~/img/userAvt/4cc2377e-8594-43ee-9022-3f72815880dd.jpg" alt="User 1 Avatar">

            </div>

            <div class="user-info">

                <div class="user-name">John Doe </div>

                <div class="user-rating">

                    <i class="star-checked far fa-star"></i>

                    <i class="star-checked far fa-star"></i>

                    <i class="star-checked far fa-star"></i>

                    <i class="star-checked far fa-star"></i>

                    <i class="star-unchecked far fa-star"></i>

                </div>

                <div class="feedback-text">Great product! Fast delivery and good quality.</div>

                <div class="store-feedback">-> Thank you for your positive feedback!</div>

            </div>

        </div>

        <div class="feedback-item">

            <div class="user-avatar">

                <img src="~/img/userAvt/4cc2377e-8594-43ee-9022-3f72815880dd.jpg" alt="User 2 Avatar">

            </div>

            <div class="user-info">

                <div class="user-name">Jane Smith</div>

                <div class="user-rating">

                    <i class="star-checked far fa-star"></i>

                    <i class="star-checked far fa-star"></i>

                    <i class="star-checked far fa-star"></i>

                    <i class="star-unchecked far fa-star"></i>

                    <i class="star-unchecked far fa-star"></i>

                </div>

                <div class="feedback-text">I love it! The product exceeded my expectations.</div>

                <div class="store-feedback">Store response: We're glad to hear that! Thank you for choosing our product.</div>

            </div>

        </div>

        <div class="feedback-item">

            <div class="user-avatar">

                <img src="~/img/userAvt/4cc2377e-8594-43ee-9022-3f72815880dd.jpg" alt="User 2 Avatar">

            </div>

            <div class="user-info">

                <div class="user-name">Jane Smith</div>

                <div class="user-rating">

                    <i class="star-checked far fa-star"></i>

                    <i class="star-checked far fa-star"></i>

                    <i class="star-checked far fa-star"></i>

                    <i class="star-checked far fa-star"></i>

                    <i class="star-unchecked far fa-star"></i>

                </div>

                <div class="feedback-text">I love it! The product exceeded my expectations.</div>

                <div class="store-feedback">Store response: We're glad to hear that! Thank you for choosing our product.</div>

            </div>

        </div>

    </div>

    <!-- Add more feedback items as needed -->

</div>

**User Changepassword**

<link rel="stylesheet" href="~/css/customer/changepassword.css">

<partial name="\_Notification" />

<div class="body\_changepassword">

    <form class="changepassword-form" action="#" method="post" id="changePasswordForm">

        <h2>Đổi Mật Khẩu</h2>

        <label for="currentPassword">Mật khẩu hiện tại:</label>

        <input type="password" name="currentPassword" required>

        <label for="newPassword">Mật khẩu mới:</label>

        <input type="password" name="newPassword" required>

        <label for="confirmNewPassword">Xác nhận mật khẩu mới:</label>

        <input type="password" name="confirmNewPassword" required>

        <input type="submit" value="Đổi mật khẩu">

    </form>

</div>

**User Edit Profile**

@model User

<link rel="stylesheet" href="~/css/customer/editprofile.css">

<div class="body\_editprofile">

    <form class="editprofile-form" action="#" method="post" enctype="multipart/form-data" id="profileForm">

        <input asp-for="@Model.Email" hidden type="text">

        <input asp-for="@Model.Id" hidden />

        <input asp-for="@Model.avtURL" hidden />

        <h2>Change Information</h2>

        <div class="avatar-container" id="avatarContainer">

            <img id="currentAvatar" src="@Model.avtURL" alt="Avatar">

        </div>

        <div class="upload-btn-wrapper">

            <button class="btn-image">Chọn ảnh mới</button>

            <input type="file" name="file" accept="image/\*" id="file" onchange="updateAvatar()">

        </div>

        <label for="name">Tên:</label>

        <input asp-for="@Model.Name" required>

        <label for="gender">Giới tính:</label>

        <select id="gender" name="gender" required>

            @{

                if (Model.Gender == ShopBee.Models.User.GenderType.Male)

                {

                    <option value="0" selected>Male</option>

                    <option value="1">Female</option>

                } else

                {

                    <option value="0">Male</option>

                    <option value="1" selected>Female</option>

                }

            }

        </select>

        <label for="phone">Số điện thoại:</label>

        <input asp-for="@Model.Phone" type="tel" required>

        <label for="address">Địa chỉ:</label>

        <textarea asp-for="@Model.Adress" rows="3" required></textarea>

        <label for="password">Mật khẩu:</label>

        <input type="password" name="password" required>

        <input type="submit" value="Lưu thay đổi">

    </form>

    <script>

        function updateAvatar() {

            var input = document.getElementById('file');

            var container = document.getElementById('avatarContainer');

            var currentAvatar = document.getElementById('currentAvatar');

            var file = input.files[0];

            if (file) {

                var reader = new FileReader();

                reader.onload = function (e) {

                    currentAvatar.src = e.target.result;

                };

                reader.readAsDataURL(file);

            }

        }

    </script>

</div>

**User Login**

<head>

    <meta charset="UTF-8">

    <title>Login - Shopbee</title>

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <link rel="stylesheet" href="~/css/login.css" />

    <link rel="stylesheet" href="//cdnjs.cloudflare.com/ajax/libs/toastr.js/latest/css/toastr.min.css" />

</head>

<body>

    <div class="login-container">

        <h2>Login</h2>

        <form class="login-form" method="post">

            <div class="form-group">

                <label for="email">Email:</label>

                <input type="text" id="email" name="email" required>

            </div>

            <div class="form-group">

                <label for="password">Password:</label>

                <input type="password" id="password" name="password" required>

            </div>

            <div class="form-group">

                <button type="submit">Login</button>

            </div>

        </form>

        Don't have and account yet? <a asp-area="Customer" asp-controller="User" asp-action="Register">Register</a>

    </div>

</body>

@{

    Layout = null;

}

**User Register Page**

@model User

<head>

    <meta charset="UTF-8">

    <title>Register - Shopbee</title>

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <link rel="stylesheet" href="~/css/register.css" />

</head>

<body>

    <div class="register-container">

        <h2>Register</h2>

        <form class="register-form" method="post" enctype="multipart/form-data">

            <div class="form-group">

                <label for="email">Email:</label>

                <input asp-for="Email" type="email" required>

            </div>

            <div class="form-group">

                <label for="password">Password:</label>

                <input asp-for="Password" type="password" required>

            </div>

            <div class="form-group">

                <label for="name">Name:</label>

                <input type="text" asp-for="Name" required>

            </div>

            <div class="form-group">

                <label for="gender">Gender:</label>

                <select id="gender" name="gender" required>

                    <option value="0">Male</option>

                    <option value="1">Female</option>

                </select>

            </div>

            <div class="form-group">

                <label for="phone">Phone:</label>

                <input type="tel" asp-for="Phone" required>

            </div>

            <div class="form-group">

                <label for="address">Address:</label>

                <input type="text" asp-for="Adress" required>

            </div>

            <div class="form-group">

                <label for="image">Image:</label>

                <input type="file" id="file" name="file" accept="image/\*" required>

            </div>

            <div class="form-group">

                <button type="submit">Register</button>

            </div>

        </form>

        Already have an account? <a asp-area="Customer" asp-controller="User" asp-action="Login">Login</a>

    </div>

</body>

@{

    Layout = null;

}

@section Scripts {

    @{

        <partial name="\_ValidationScriptsPartial" />

    }

}

## 3. Final Screenshots Of The Application

This is Home Page when user is not Login.

A screenshot of a computer

Description automatically generated

**Login Page:**

A screenshot of a login screen

Description automatically generated

**Register Page:**

A screenshot of a computer

Description automatically generated

**Role Admin**

Home Page:

A screenshot of a book

Description automatically generated

DashBoard Admin:

A screenshot of a computer

Description automatically generated

Category Management Page:

A screenshot of a computer

Description automatically generated

Book Management Page:

A screenshot of a computer

Description automatically generated

Store Management Page:

A screenshot of a computer

Description automatically generated

Order Management Page:

A screenshot of a computer

Description automatically generated

User Management Page:

A screenshot of a computer

Description automatically generated

UserRole Management:

A screenshot of a computer

Description automatically generated

**Customer Role:**

Home Page

A screenshot of a book

Description automatically generated

Book Detail:

A screenshot of a computer

Description automatically generated

Cart Page:

A screenshot of a computer

Description automatically generated

Order Page:

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Store View Page:**

**Book Management:**

**A screenshot of a computer

Description automatically generated**

**OrderManagement:**

**A screenshot of a computer

Description automatically generated**

**Feedback:**

**A screenshot of a computer

Description automatically generated**

**Add Category:**

A screenshot of a computer

Description automatically generated

## 4. Screenshots Of Using Github Or Gitlab To Manage The Source Code

A screenshot of a computer

Description automatically generated

Figure 28: Github

Source code management through GitHub represents a pivotal tool for individual developers and collaborative teams engaged in software projects. At its core, Git, the technology underpinning GitHub, offers robust version control capabilities. This means you can meticulously trace alterations in your codebase over time, effortlessly revert to previous iterations, and collaborate without the looming threat of data loss. GitHub, acting as a collaborative platform, seamlessly facilitates teamwork among developers.

One of Git's standout features is its ability to let multiple team members work on the same codebase concurrently, each in their respective branches. This parallel development approach ensures that individual contributions remain isolated until they're ready to be merged into the main codebase. Pull Requests (PRs) emerge as a pivotal feature in GitHub, serving as a mechanism for code review, constructive discussion, and the eventual integration of changes.

Pull Requests, in the GitHub context, provide a structured and organized way for team members to scrutinize and assess each other's code modifications. This systematic process not only upholds the integrity of the codebase but also serves as a proactive means of identifying and rectifying potential bugs early in the development cycle. Furthermore, the collaborative nature of Pull Requests fosters the exchange of knowledge and best practices among team members, contributing to a more cohesive and knowledgeable development team.

In essence, GitHub, coupled with Git, transcends being a mere version control system; it becomes a dynamic collaboration hub that empowers developers to collectively enhance code quality, streamline workflows, and fortify the foundation for successful software development projects.

This is link Github of my project: <https://github.com/qngoc07012002/1670_ApplicationDevelopment_ShopBee_GroupProject/tree/master/ShopBee>

## 5. Screenshots Of Using IIS Or Azure For The Application Deployment

A screenshot of a computer

Description automatically generated

Figure 29: Database

We use the IIS Database Server database hosted on a physical server using the ubuntu 20.04 operating system. Once we're done, we go through creating accounts and giving access to my team members. This database on the server does not have an interface, so we use SSMS to access it

# CHAPTER 3: APPLICATION EVALUATION

## 1. Review The Performance Of The Application (P6)

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case** | **Test Case** | **Expect Result** | **Actual Result** |
| 1 | View Landing Page | Success | Not Yet |
| 2 | Login | Success | Not Yet |
| 3 | Register | Success | Not Yet |
| 4 | Search Book by Name, Author | Success | Not Yet |
| 5 | Search Book by Category | Success | Not Yet |
| 6 | Search Book by Price | Success | Not Yet |
| 7 | View Book Detail | Success | Not Yet |
| 8 | Add Book to Cart | Success | Not Yet |
| 9 | Checkout | Success | Not Yet |
| 10 | Edit Profile | Success | Not Yet |
| 11 | Change Password | Success | Not Yet |
| 12 | CRUD Category | Success | Not Yet |
| 13 | CRUD Book | Success | Not Yet |
| 14 | CRUD Store | Success | Not Yet |
| 15 | CRUD User | Success | Not Yet |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **Action** | **Expected Result** | **Actual Result** | **Conclusion** |
| 1 | Visit Shopbee's Home Page and the content is displayed (Visit the website shopbee.xyz) | Home Page Is displayed and the content is fully displayed |  | PASS |
| 2 | User login into the Website (email: [hieu@gmail.com](mailto:hieu@gmail.com), password: Hieu2172003) | Login Successfully and Move to Home Page |  | PASS |
| 3 | Users register for an account on the Website | Notification Register Successful and Move to Login Page |  | PASS |
| 4 | Search for books by keywords of Book Name or Author | Type “Đế chế ký hiệu” on search bar and Enter. Book or Author contain “Đế chế ký hiệu” will display |  | PASS |
| 5 | User selects Book Category on HomePage | Select Comics in Category List. Comics genre books are displayed |  | PASS |
| 6 | Users sort books by ascending and descending price | Click On Descending and Ascending Option.  Book will. The books will be arranged in ascending and descending order |  | PASS |
| 7 | See detailed information of the book | Click on Books. Users will be redirected to the book's detailed information page and Feedbacks. |  | PASS |
| 8 | User adds books to cart | Select quantity and Click Add to Cart on the product details page. Click on the cart icon, users can see the books in the cart. |  | PASS |
| 9 | The user proceeds to checkout the cart | The user presses Checkout in Cart, the order is successfully created and the user is redirected to the Order page to view the order. |  | PASS |
| 10 | Users change their personal information |  |  | PASS |
| 11 | Users change their password |  |  | PASS |
| 12 | CRUD Category | Admin View, Create, Edit, Delete Categories without encountering any errors |  | PASS |
| 13 | CRUD Book | Admin View, Create, Edit, Delete Books without encountering any errors |  | PASS |
| 14 | CRUD Store | Admin View, Create, Edit, Delete Stores without encountering any errors |  | PASS |
| 15 | CRUD User | Admin View, Create, Edit, Delete Users without encountering any errors |  | PASS |

## 2. Conclude Whether The Application Adapts All Requirements Or It Needs To Be Improved Later (P6)

**Continuous Monitoring and Iterative Improvement:**

The critical review suggests that the application is well-prepared to meet the current requirements. However, the dynamic nature of technology and user expectations necessitates continuous monitoring and iterative improvements. Regular assessments and updates should be part of the development cycle to address emerging challenges and incorporate new features.

**User Feedback and Evolution of Requirements:**

User feedback is crucial for gauging the actual user experience and identifying areas for improvement. As users interact with the platform, their feedback can reveal insights that may lead to enhancements or new requirements. The application should be adaptable to evolving user needs and industry trends.

**Scalability for Future Growth:**

While the application demonstrates scalability considerations, its ability to handle future growth needs ongoing assessment. As the user base and transaction volume increase, the application should be equipped to scale seamlessly. Regular performance monitoring and scalability testing can help ensure that the system remains robust under varying workloads.

**Security Posture and Compliance:**

Security is an ongoing concern, and the application should adapt to new security threats and compliance requirements. Regular security audits, updates, and patches are essential to maintaining a secure environment.

In conclusion, while the ShopBee application appears to meet the current requirements well, its long-term success will depend on the commitment to continuous improvement, responsiveness to user feedback, scalability for future growth, and vigilance in addressing security considerations. Regular updates and adaptability to changing circumstances will contribute to the sustained success of the application.

## 3. Analyse the factors that influence the performance of the application (M5)

**Design Stage:**

**Scalability:**

Review: The application architecture has been designed to scale efficiently.

Observation: Both horizontal and vertical scaling options have been considered.

Recommendation: The choice of technologies and frameworks aligns well with scalability goals.

**User Interface (UI) Design:**

Review: The UI is intuitive and responsive.

Observation: The UI design meets user expectations and industry standards.

Recommendation: Usability testing has been recommended to further refine the UI.

**Development Stage:**

**Code Quality:**

Review: Code quality is emphasized, and maintainable code is encouraged.

Observation: Adherence to coding standards and best practices is evident.

Recommendation: Ongoing emphasis on modular and reusable code is recommended.

**Database Optimization:**

Review: Database schema and queries are optimized.

Observation: Indexing and caching strategies have been implemented.

Recommendation: Regular reviews and optimizations of database queries are encouraged.

**Security Measures:**

Review: Robust security measures are integrated into the application.

Observation: Regular updates and patching of security vulnerabilities are in place.

Recommendation: The continued conduct of security audits during development is suggested.

**Testing Stage:**

**Performance Testing:**

Review: Thorough performance testing has been conducted.

Observation: Scalability bottlenecks have been identified and addressed.

Recommendation: The implementation of simulated user loads for peak traffic scenarios is a positive practice.

**Usability Testing:**

Review: Usability testing has been conducted to validate the UI.

Observation: User experience issues have been identified and addressed.

Recommendation: The integration of usability testing as a regular practice is beneficial.

**Security Testing:**

Review: Thorough security testing has been performed.

Observation: Effective implementation of security measures has been ensured.

Recommendation: The ongoing identification and remediation of security vulnerabilities during testing are crucial.

**Reflective Discussion on Previously Identified Risks:**

**User Authentication and Authorization:**

Review: Authentication mechanisms are robust, and regular testing is emphasized.

Observation: Multi-factor authentication has been recommended for enhanced security.

Recommendation: Ongoing vigilance and updates in authentication procedures are encouraged.

**Scalability Challenges:**

Review: The scalability of the application architecture has been evaluated.

Observation: Proactive monitoring mechanisms for identifying scalability issues are in place.

Recommendation: Continued emphasis on scalability, especially as the user base grows, is recommended.

**Conclusion:**

The critical review indicates that the ShopBee application has been well-designed, developed, and tested with a focus on performance, security, and scalability. The identified recommendations are aimed at further enhancing the application's robustness and sustainability. Continuous monitoring, updates, and adherence to best practices will contribute to the long-term success of the platform.