SOFT20181

Internet Application Progreamming Report

Clive Kimari 2020 (N0937738)

Table of Contents

List of Tables

List of Figures

Introduction

This report is the supporting documentation for the book website application that has been creating for SOFT20181 Internet Application Programming Assessment 2.

The application is called E-Books and intends to sell multiple books with a wide range of book genres such as fantasy fiction, Biographies, and Personal finance to name a few. With Some of the books that can be purchased on the E-Books being Can't Hurt Me by David Goggins and Lord of The Rings by J.R.R. Tolkien to name a few. The application consists of multiple pages both for customers and the admin of the page, with the admin being able to login and view and edit orders, books, and book genres. While some of the customer pages allow the user to register and sign in and search for books available on the website, along with a cart page and order page for the user to complete their transaction.

The website has also been designed to be as professional as possible with media responsiveness being implemented as much as possible while also implementing stripe to the website which allows for the customer to view checkout purchases made on the E-Books website. Lastly SQL server management has also been implemented into the code to allow for data such as customer details, books, book genres, cart etc. to be saved on the database through appsettings.json.

Success Criteria

Table 1: Success Criteria

Success Criteria Point	**Priority**	**Implemented?**
Website navigation	Required	Yes, a dark navigation bar has been implemented with the help of Layout.cshtml the navigation bar has been further implemented to look more appealing with CSS and link to the other pages.
Search Box for Customers to find products	Required	Yes, On the home page (where all the books are made available) the user can search for products using the search box on the navigation bar. The box has been implemented in layout using both CSS classes and C# in both the Homecontroller.cs as well as Search.cshtml
Allows customers to add quantity to cart	Required	Yes, for each book the customer will be able choose the number of books they want to buy, and it will be added into shopping cart. This has been done mainly using C# in Homecontroller.cs where BookQty has been made an integer to allow for the books quantity to increase and decrease

Login and Register customer	Required	Yes, customer login and register has been fully implemented to allow users to create and login into their accounts and whenever they logout of their account the cart will always save and remember the books, they left in it. This has been done through using views in the Authcontroller.cs where if statements have been used in logins to validate their account and when a user registers for an account the details will be saved into the database to allow for them to now login with those details.
Admin Page	Required	Yes, an admin page has also been implemented into the website with the hyperlink being implemented into the Layout.cshtml where admin login will be shown in the navigation bar if no one is logged in however the random user cannot create a admin login page as there is only one admin account that is able to add, edit, and delete multiple books and book genres and also view checkouts and time of checkouts for the purchased books.
Contact and or about us page	Required	Contact and about us page have been implemented into the website where the user can view the pages on the navigation bar. It has been implemented in Layout.cshtml and sends a message to the E-Books company through a form that has been made
Checkout page Process Payment	Could	Yes, Process Payment has been implemented mainly using JavaScript and Json with certain keys. Through the stripe website. When the user clicks on the checkout button the stripe JavaScript will activate therefore allowing the user to input fake generated stripe

		bank details where then the user will checkout and the order details of the purchase. will also be saved on the stripe account.

Home page

The homepage (home Index.csthtml) of the website is where the user will be introduced to the website and see a picture of books therefore already indicating that the website is book related. They will also find all the books genres and all the available books that come along with it. The genres just below the picture has been implemented in the Index.cshtml view of the Homecontroller.cs for this a foreach loop has been used for the property of categories which are the book genres, inside this loop the categories id URL has been implemented into it along with the name of category which has been made as a button. This allows for all the categories to be shown along with the books associated with them through the categories id.

```
<h3 class="mb-3 text-black-50">Books Genres</h3>
@foreach (var item in ViewBag.Categories)
{

<a asp-action="Index" asp-route-id="@item.CategoryId" class="btn btn-sm btn-secondary">@item.CategoryName</a>


}
```

Figure 1: Foreach loop used in Index.cshtml

After the genres the books are then shown one by one underneath each other in which each book consists of an image of the front cover, a description of the book, price, add to cart, and lastly a quantity selection for if the user wants to add the product to their basket. The layout of the of the books grid is done through CSS classes and styles through both bootstrap CSS and self-implementation of CSS. As shown below through the book images size.

```
<div id="bookImage">
    <img src="~/images/uploads/bookImages/@item.BookImage" height="200" width="200" alt="img"/>
</div>
```

Figure 2: CSS used in Index.cshtml

The book's details such as price and description has also been implemented through CSS and C# by calling each of the books created along with their features through the item property also shown below where price has been put in a header class along with a brief text of the price accompanied with a £ sign. This data has been taken from the database created for these features where every time a product is created the features will be saved int the SQL Server Management.

```
<h6 class="text-danger">Price:£@item.BookPrice</h6>
```

Figure 3: Item property used in Index.cshtml

Lastly TempData alerts have been implemented in these index page to transfer data from the view to the controller and the controller to view which assists in allowing the also other views, however in this instance, data from an action method has been transferred to a view.

Figure 4: TempData implemented in Index.cshtml

About Us page

Whether the user has an account or not they will be able view both the about and Contact Us page where in the About Us page they will be able to read a brief description of what the E-Books Website is about and what it has to offer. There are then hyperlinks to social medias such as Instagram and Twitter that have been implemented into the website using HTML and CSS along with font awesome to give the buttons icons.

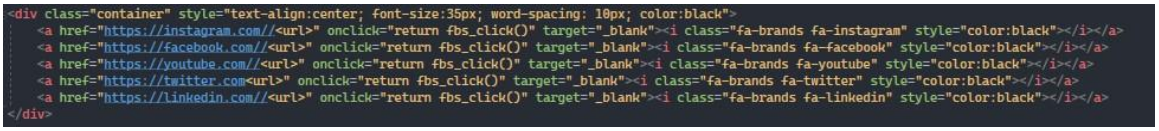

Figure 5: Hyperlinks being implemented into AboutUs.cshtml

Contact Us Page

If the Customer decides that they want to contact E-Books then by clicking the 'Contact Us' button they will be sent to a page where they can fill out their full name, email address, and then a message to send the website where they will then receive an alert saying that E-books will message the customer as soon as possible. This has been done through creating views in the Homecontroller.Cs where CSS has been used to allow the user to input certain details that will be saved under a certain name in the database.

Figure 6: Input classes being made under certain attributes in ContactUs.cshtml

Login and Registration Page

Registration

Before the user can add or remove items and checkout, they must either log in or register an account on the website. In terms of the registration page, the user will be asked for a details such as full name, email, phone, password, and address. This must input the correct forms of both phone number and email address which has been done using basic regex to create an expression validating certain characters as shown below. Once the user has imputed the correct inputs and pressed register the user's details will be saved on the database.

```
[Required(AllowEmptyStrings = false, ErrorMessage = "Please enter your phone")]
[RegularExpression(@"^\(?([0-9]{3})\)?[-. ]?([0-9]{3})[-. ]?([0-9]{4})$",
            ErrorMessage = "Entered phone format is not valid.")]
3 references
public string CustomerPhone { get; set; }
```

Figure 7: Regex implemented to validate the user input for phone number in Customer.cs

Login

If the user has an account already made, then they will be able to login and the Authcontroller.cs will validate whether the user is real or not as the both the email and password must resemble one in the database which has been done firstly through a variable to set the condition and then an if/else statement for the outcome of the user input. Once the user has successfully logged in, they can then add products into their cart.

```
var result = _context.Customers.Where(e => e.CustomerEmail == user.CustomerEmail && e.CustomerPassword == user.CustomerPassword).FirstOrDefault();
```

Figure 8: variable condition implemented in AuthController.cs

Cart

When the user has decided to add a book to the cart the book will be put in the checkout basket in the navigation bar and the checkout page table where the user will have the choice to add to the quantity of the specified book or delete the book, all while this is being done the subtotal amount will calculate the price of everything in the basket and constantly update.

The book has been added to the cart through `@Html.DisplayFor(modelItem => item.bookName)` as this expression calls certain features from the Index.cshtml. As shown above the code, it is calling the book name. As for the user being able to increase, decrease, and delete books, this has been done in Cart.cshtml though calling functions in HomeController.cs.

```
<a asp-action="IncreaseQuantity" asp-route-id="@item.cartId" class="text-success">
    <i class="fa fa-plus-square"></i>
</a>
</td>
<td>
    £@Html.DisplayFor(modelItem => item.bookTotalPrice)
</td>
<td>
    <a asp-action="RemoveFromCart" asp-route-id="@item.cartId" class="text-danger">

        <i class="fa fa-trash"></i>
    </a>
</td>
```

Figure 9: Increase and decrease quality being called in cart.cshtml

In which the code has been written in Homecontroler.cs to both add and decrease quantity through firstly reading the database and again setting a condition where both bookId and FKBookId have must be the same as well as making sure the certain customer signed in cart is edited through turn the string of customer id into an integer. this is allowed to happen as bookID and Customer ID has been linked together in a model to allow for the only the specified users cart to chart and not every other account cart.

```
var _contextCart = _context.Carts.Where(x => x.FkBookId == bookId && x.FkCustomerId==Int32.Parse(customerId)).FirstOrDefault();
```

Figure 10: variable condition for editing the quantity of the book items in cart

Lastly an if/else statement for if the user is increasing quantity for a book that is already in the cart or, if the user is adding a book that wasn't already in the cart, where the total price and quantity of the cart will change for the specific customer if the quantity is either increased or decreased as shown below.

```
var _contextCart = _context.Carts.Where(x => x.FkBookId == bookId && x.FkCustomerId==Int32.Parse(customerId)).FirstOrDefault();
if (_contextCart != null)
{
    //returing the total price and quanity for cart for specific customer,
    //if book is already added into then only quanity will be increased for specific book
    _contextCart.TotalPrice = _contextCart.TotalPrice+( _contextBook.BookPrice * bookQty);
    _contextCart.Qty = _contextCart.Qty + bookQty;
    _context.Carts.Update(_contextCart);
    //updating the quantity after adding the book into cart
    _contextBook.BookQty = _contextBook.BookQty - bookQty;
    _context.Books.Update(_contextBook);
    _context.SaveChanges();
}
else
{
    //if book is adding first time into cart then this condition will be executed, for specific customer
    var data = new Cart();
    data.FkCustomerId = Int32.Parse(customerId);
    data.FkBookId = bookId;
    data.Qty = bookQty;
    data.TotalPrice = _contextBook.BookPrice * bookQty;
    _context.Carts.Add(data);
    _contextBook.BookQty = _contextBook.BookQty - bookQty;
    _context.Books.Update(_contextBook);
    _context.SaveChanges();
}
```

Figure 11:if statement for editing books in the cart

Once satisfied with the books they would like to purchase, the user can go to checkout by pressing the green checkout button which will send them to the checkout page.

User-input for the Checkout Page

At the checkout the customer will be prompted to add their details for their name, number, and their address. This has firstly been done by making sure the user has inputted details into the boxes, so just like every other validation on the website needing user input validation classes has been used.

```
<div asp-validation-summary="ModelOnly" class="text-danger"></div>
```

Figure 12: ModelOnly class used Checkout.cshtml

Label elements have then been implemented for all the user-input boxes to generate a property for the model as well as an input to enable the customer to type an input for that certain property where it will then be stored into the data base.

```
<label asp-for="FullName" class="control-label"></label>
<input asp-for="FullName" class="form-control" />
<span asp-validation-for="FullName" class="text-danger"></span>
```

Figure 13: label tag helper on Checkout.cshtml

Checkout with Stripe

When the user clicks on payment the customer will be directed to a form where they can fill out their bank details (fake details from https://stripe.com/docs/testing) to complete their payment. Through implementing https://checkout.stripe.com/checkout.js firstly in wwwroot and then calling it into checkout.cshtml this has allowed for majority of the template to be designed, while the validation process of the bank card has been done by through wwwrott lib.

```
<script src="https://checkout.stripe.com/checkout.js" class="stripe-button"
```

Figure 14: Stripe being called in Checkout.cshtml

Admin Page

When it comes to the admin page the admin will firstly need to sign to access editing features, the code used here is the same that is done with customer login. Once successfully signed in the admin will have access to Manage Book Genres, Manage Books, and mange orders.

Creating Book Genres & Book List

One of the three choices the admin has is to edit the book genres and create new genres, for the user to create a new book genre a Create.cshtml view has been made from the ManageCategoriesController.cs where again an input has been created to allow the admin to create a book genre in which it is saved to CategoryName which is a model string inside Category.cs that will send and store all of the Categories created into the SQL Server, the equivalent is done with Book List, the same process has bene for creating a book also..

```
17 references
public int CategoryId { get; set; }
[Required(AllowEmptyStrings = false, ErrorMessage = "Please enter the category")]
[DisplayName("Book Genre")]
15 references
public string CategoryName { get; set; }

2 references
public virtual ICollection<Book> Books { get; set; }
```

Figure 15: Assigned value of name variable in Category.cs

Editing Book Genres/ Book List

When the admin decides to edit the details of the book genre the ManageCategoriesController.cs will send them to the Edit view where the user will again be prompted to input a new name for the Book Genre. Once the admin has confirmed a new name for a book genre the ManageCategoriesController.cs edit section will proceed to update the change and then save the changes inside a try a catch loop as shown below with the if statement first validation that the Model conditions were first met, the same is done when editing the Book List.

```
if (ModelState.IsValid)
{
    try
    {
        _context.Update(category);
        await _context.SaveChangesAsync();
    }
    catch (DbUpdateConcurrencyException)
    {
        if (!CategoryExists(category.CategoryId))
        {
            return NotFound();
        }
        else
        {
            throw;
        }
    }
    return RedirectToAction(nameof(Index));
}
return View(category);
```

Figure 16: If statment inside post Edit for ManageCategoriesController.cs

Details of Book Genres/ Book List

If the admin decides to view the details of the book, then when they click on details it will show both the Id and the name of the book genre, where then the admin can decide to either edit the name by clicking on edit and be sent back to the edit page again or they can go back to the Book Genre list. The display of the name is shown through DisplayNameFor which will give the name of the property without the value therefore displaying the header BookGenre while DisplayFor will generate the html string for the object property therefore displaying the name of the book genre, the same is done when viewing the details of Book List.

```
<dt>
    @Html.DisplayNameFor(model => model.CategoryName)
</dt>
<dd>
    @Html.DisplayFor(model => model.CategoryName)
</dd>
```

Figure 17: How the Name of the Book Genre is displayed in Edit.cshtml

Deleting of Book Genres/ Book List

When deciding to delete a book genre or a book the admin will firstly click on the delete button and then be directed onto another page where they can decide to get back to the list of book genres/book list or confirm the deletion of the product by pressing the delete button where it will then no longer show in the specified list. Once the admin has confirmed to delete either book genre/book list the ManageCategoriesController.cs delete section will proceed to remove the changes and then save the changes.

```
var category = await _context.Categories.FindAsync(id);
_context.Categories.Remove(category);
await _context.SaveChangesAsync();
return RedirectToAction(nameof(Index));
```

Figure 18: Process of the specified property being deleted in ManageCategoriesController.cs

Manage Orders

In terms of managing books genres/book list, the same can be said for managing orders with the main differences being the editing details as the order will be where the admin can view the time of purchase, name, phone, address, and order status. They will also be able to change the status by going into the edit page.in the Homecontroller.cs. Order date has been implemented by converting the current date time object into a string at the moment the user presses payment in Checkout,cshtml.

```
order.OrderDate = DateTime.Now.ToString();
```

Figure 19: OrderDate date and time being converted into a string

Challenges Encountered

One of the struggles encountered with the website that was built would be that on the home page where all the products are displayed the responsiveness of the product sections are lacking as the description and the add to cart button appear partially outside of their specified box. This error would mainly be due to the layout and the CSS of the product boxes.

Summary and Conclusion

In conclusion a large majority of the project build was successful in both the design and functionality of the website as shown through the multiple pages and certain user inputs need such as signing in and registering for both customer and admin. Secondly the admin pages was a complete success with the admin functionality being fully functioning enabling the use of editing, checking details, and deleting books which was done through JavaScript and C# in the controllers and views. However, there were also challenges that came with building this website as explained previously with one being the responsiveness in the homepage, this would mainly be down to timing of project completion. In conclusion the project has enabled me to learn and implement C#, Regex, and json while also being able to further implement and understand JavaScript and C++ even further.

Admin Details

Username – clivekimari

Password – clivekimari123

Sources and References

https://stripe.com/docs/testing

https://checkout.stripe.com/checkout.js

https://dotnet.microsoft.com/en-us/apps/aspnet

https://www.youtube.com/watch?v=1ck9LIBxO14&t=2481s

https://www.youtube.com/watch?v=hZ1DASYd9rk

https://fontawesome.com/

https://getbootstrap.com/

https://www.youtube.com/watch?v=_u3of33pZa4