

SWE 203

WEB PROGRAMMING COURSE PROJECT

Dr. Deniz Balta

PROJECT TEAM

Emirhan Tuygun	B211202003
Emirhan Cebiroğlu	B211202065
Şeyma Handekli	B211202069

emirhan.tuygun@ogr.sakarya.edu.tr

emirhan.cebiroglu@ogr.sakarya.edu.tr

seyma.handekli@ogr.sakarya.edu.tr

REPORT

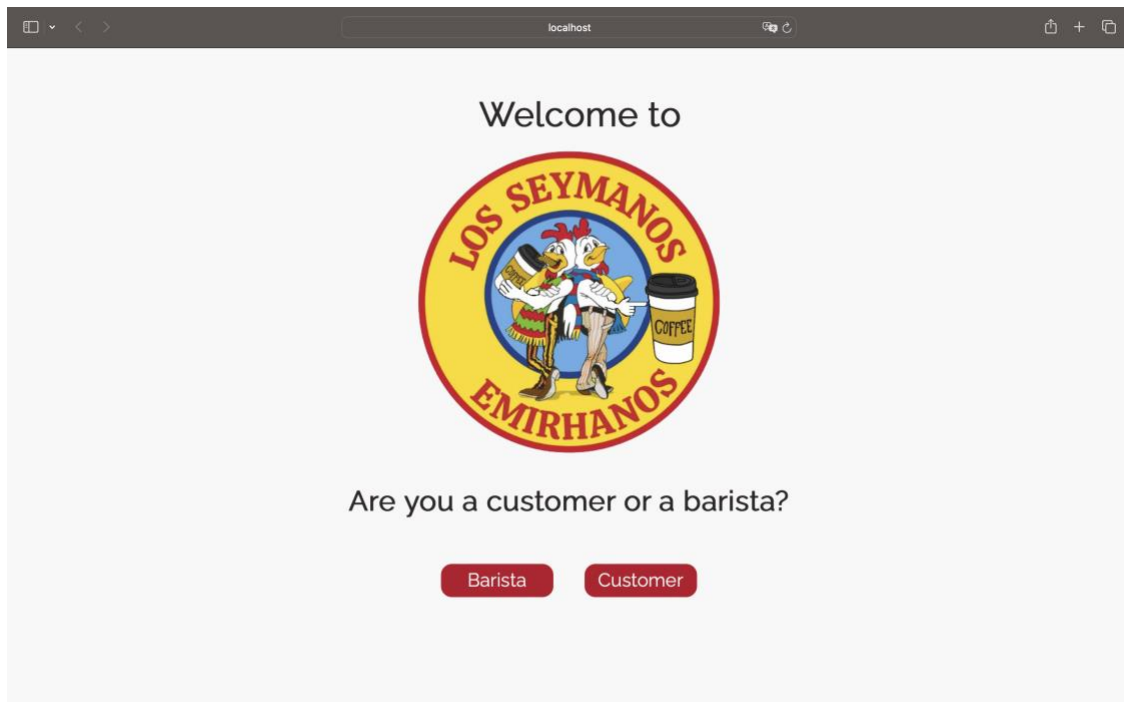
1. Project Description and Aim

We designed this café website to offer a user-friendly interface and efficient database management to meet the needs of both customers and baristas.

2. Our Solution

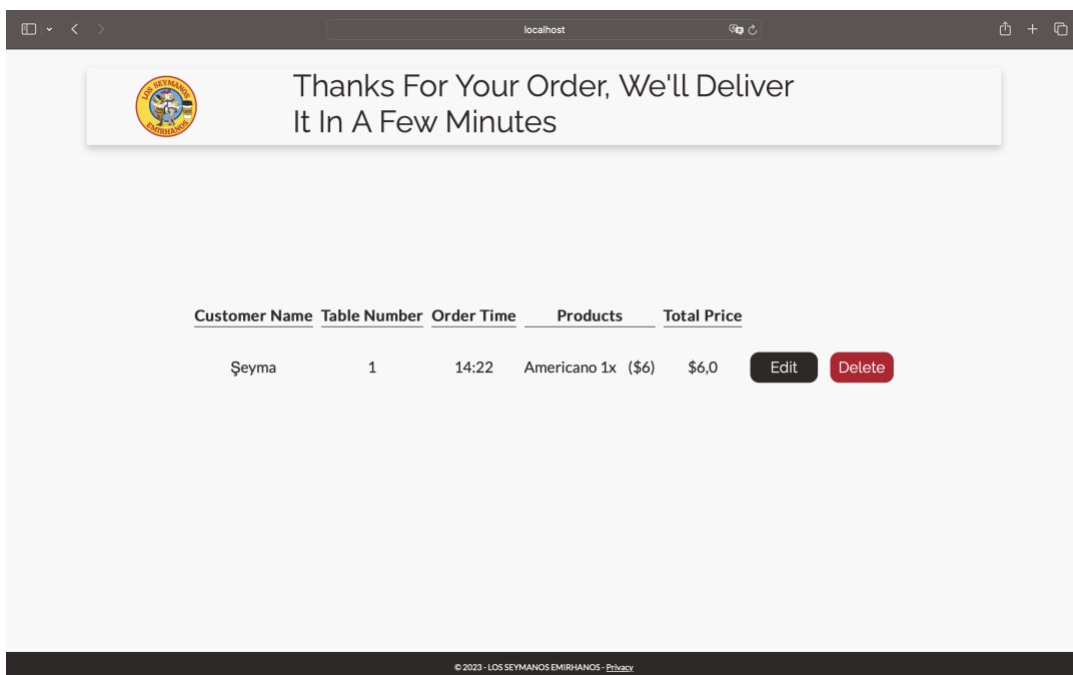
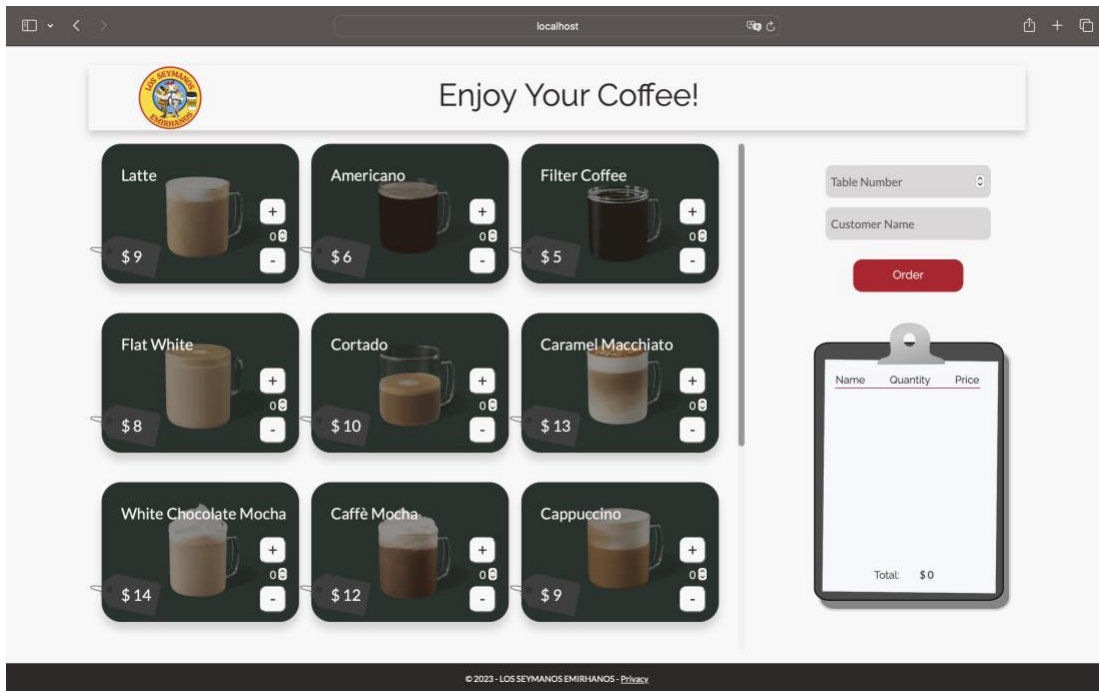
As you can see in our logo, our website, which offers an innovative solution for baristas and customers, is represented by the memorable café name "Los Seymanos Emirhanos".

Visitors are presented with two options when they enter the homepage: "Barista" and "Customer".



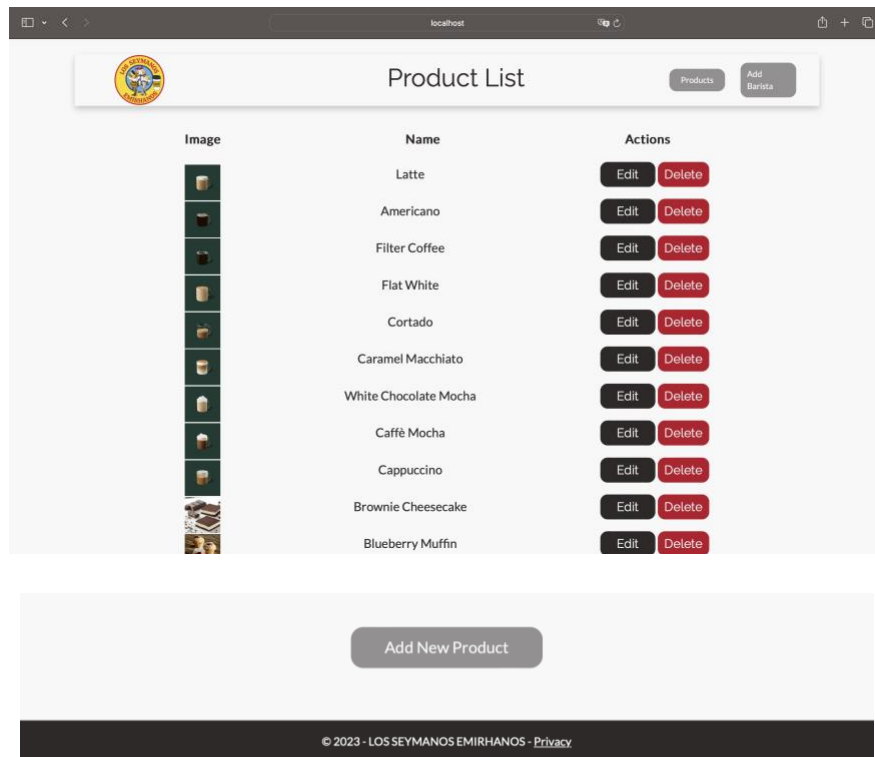
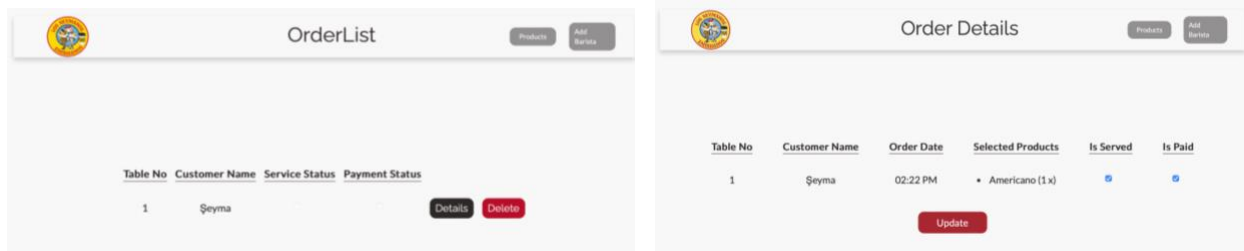
2.1 Workflow for Customers

If the “Customer” option is selected, users are redirected to the order page. Here, visitors can enter their table number and name and select the products they wish to order. When the order is confirmed, you are redirected to the "My Order" page, where you have the option to update or cancel your order.



2.2 Workflow for Baristas

If the "Barista" option is selected, users are redirected to a login page. After logging in, baristas gain access to the order list. On the "Order List" page, they can view the details of orders and update the service status and billing information of orders. Also, in the "Products" section, they can view a list of products and have the functionality to update, delete or add new products.



3. Implementation

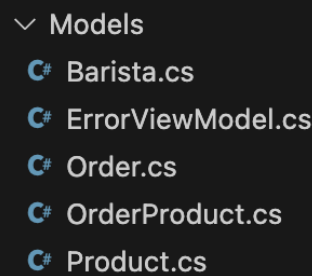
Briefly, our cafe website project is developed using ASP .NET Core MVC and Entity Framework Core and integrated with SQLite database. Our website includes some additional features to make it dynamic.

3.1 Architectural Design

We implemented Model-View-Controller architectural pattern in order to apply an isolated approach for Data and Business Logic.

3.1.1 Model

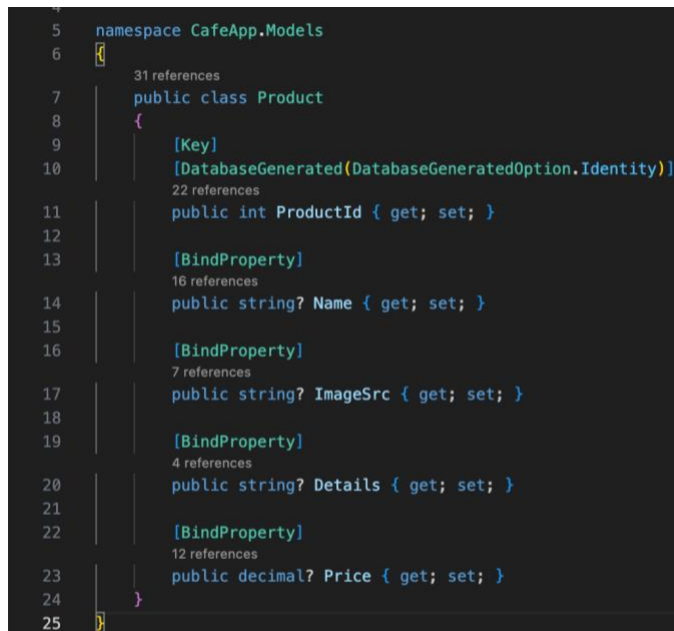
We described our data entities and business logic in Models. In our web project, there are 5 models named Product, Barista, Order, Order Product and ErrorView.



```

  Models
  C# Barista.cs
  C# ErrorViewModel.cs
  C# Order.cs
  C# OrderProduct.cs
  C# Product.cs

```



```

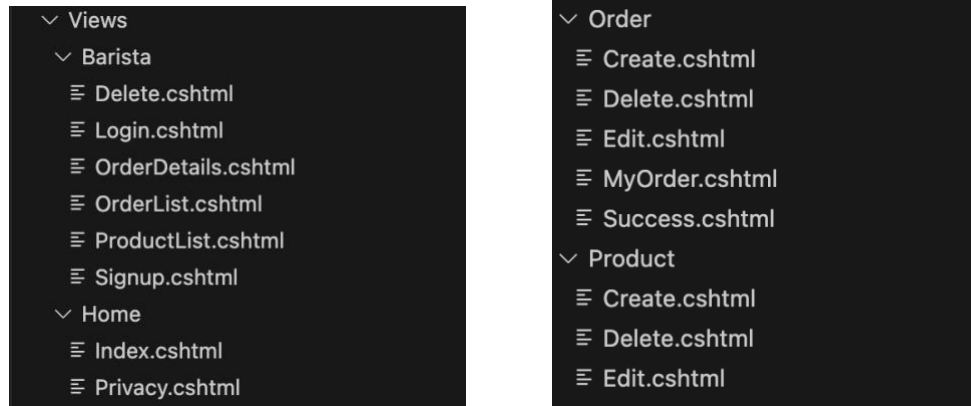
5 namespace CafeApp.Models
6 {
7     31 references
8     public class Product
9     {
10         [Key]
11         [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
12         22 references
13         public int ProductId { get; set; }
14
15         [BindProperty]
16         16 references
17         public string? Name { get; set; }
18
19         [BindProperty]
20         7 references
21         public string? ImageSrc { get; set; }
22
23         [BindProperty]
24         4 references
25         public string? Details { get; set; }
26
27         [BindProperty]
28         12 references
29         public decimal? Price { get; set; }
30     }
31 }

```

3.1.2 View

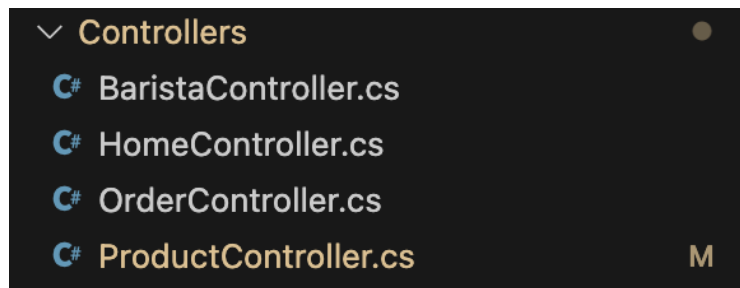
In the MVC framework, views are indeed responsible for the presentation layer of the application, which includes everything the user sees and interacts with in their web browser.

Our project utilizes views to provide a user-friendly interface for navigating the café's offerings.



3.1.3 Controller

In our project, we have used controllers to manage user interactions and data processing. They take user inputs through GET and POST requests, coordinate with models for data processing, and select views for response generation, thus directing the workflow and routing of the application.



3.2 Technologies

3.2.1 ASP .NET Core

In our project, we adopted ASP.NET Core MVC, a powerful and modern framework for building web applications. This framework empowered us with a structured approach to developing our application, with a clear separation of topics and regular code organization.

The features we added to our project such as Routing, Layouts, ViewBag/ViewData, Tag Helpers, Forms, Form Validations and Partial Views increase the sustainability and scalability of our application.

Routing

We directed the views through the controller.

```
[HttpGet]
[Route("Product/Create")]
0 references
public IActionResult Create()
{
    return View();
}

[HttpPost]
[ValidateAntiForgeryToken]
[Route("Product/Create")]
0 references
public async Task<IActionResult> Create([Bind("ProductId,Name,ImageSrc,Details,Price")] Product product)
{
    if (ModelState.IsValid)
    {
        _context.Products.Add(product);
        await _context.SaveChangesAsync();
        return RedirectToAction("ProductList", "Barista");
    }

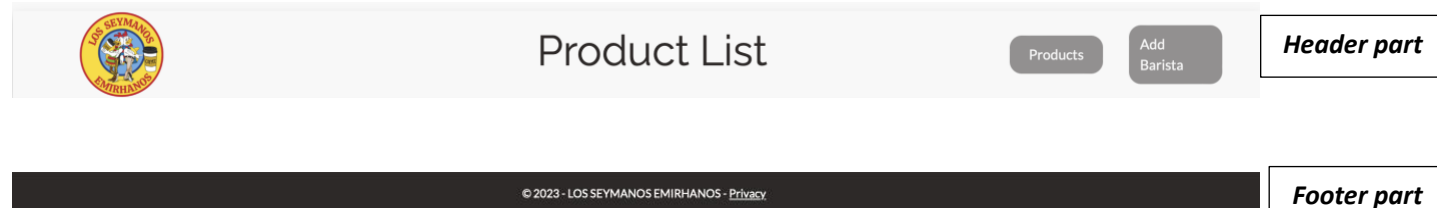
    return View(product);
}
```

Layout

We designed the header and footer sections of all our dynamic pages using layout to create a consistent and cohesive look.

```
▼ Shared
  ≡ _Layout.cshtml
  # _Layout.cshtml.css
  ≡ _Layout2.cshtml
  ≡ _ValidationScriptsPartial.cshtml
  ≡ Error.cshtml
  ≡ _ViewImports.cshtml
  ≡ _ViewStart.cshtml
```

```
≡ _ViewStart.cshtml M X
CafeApp > Views > ≡ _ViewStart.cshtml
1  @{
2      Layout = "_Layout";
3  }
4
```



```

<body id="body">
  <header>
    <nav style="display: flex; align-items: center; justify-content: space-between;">
      <div style="display: flex; justify-content: space-between; align-items: center; width: 82%;">
        @{
          string currentPath = ViewContext.HttpContext.Request.Path;
          string controller = currentPath.Contains("Barista") || currentPath.Contains("Product") ? "Barista" :
            "Home";
          string action = currentPath.Contains("Barista") || currentPath.Contains("Product") ? "OrderList" :
            "Index";
        }
        <a asp-controller="@controller" asp-action="@action"></a>
        <h2 style="font-family:'Raleway',sans-serif; font-size: 48px; margin-left: 150px;">@ViewData["Header"]
        </h2>
      </div>
    </nav>
    <div>
      @if (controller == "Barista")
      {
        <a style="text-decoration: none; color: #f8f8f8; background-color: rgba(45,41,40,0.5); padding: 10px 22px;
          border-radius: 14px; transition: all 0.3s;" href="ProductList"
          onmouseenter="this.style.opacity=0.9" onmouseout="this.style.opacity=1">Products</a>
        <a style="text-decoration: none; margin-left: 30px; color: #f8f8f8; background-color: rgba(45,41,40,0.5); padding: 10px 13px;
          border-radius: 14px; transition: all 0.3s;" href="Signup" onmouseenter="this.style.opacity=0.9"
          onmouseout="this.style.opacity=1">Add Barista</a>
      }
    </div>
  </header>

```

Header part

```

<div>
  <main role="main">
    @RenderBody()
  </main>
</div>

<footer>
  &copy; 2023 - LOS SEYMANOS EMIRHANOS - <a asp-controller="Home" asp-action="Privacy">Privacy</a>
</footer>
<script src="~/lib/jquery/dist/jquery.min.js"></script>
<script src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js"></script>
<script src="~/js/site.js" asp-append-version="true"></script>
@await RenderSectionAsync("Scripts", required: false)
</body>
</html>

```

Footer part

ViewBag / ViewData

We have used ViewBag and ViewData for dynamic data transfer between controllers and views.

```
<h2 style="font-family:'Raleway',sans-serif; font-size: 48px; margin-left: 150px;">@ViewData["Header"]</h2>
```

```
<title>@ViewBag.Title - LOS SEYMANOS EMIRHANOS</title>
```

```

@{
  ViewData["Header"] = "Product List";
}

```

```

@{
  ViewBag.Title = "OrderList";
  ViewData["Header"] = "OrderList";
}

```


TempData

We've have used TempData to to transfer data from view to controller, controller to view, or from one action method to another action method of the same or a different controller.

```
TempData["Error Message"] = "Invalid email or password!";
```

```
<div class="error-message">  
|   <div id="errorMessage">@TempData["Error Message"]</div>  
</div>
```

Tag Helpers

We've also used tag helpers, an extension of the Razor Views concept that allows the building of dynamic web pages by using an HTML-like syntax.

```
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
```

```
<form class="form" asp-controller="Order" asp-action="Create" method="post" onsubmit="return check()">  
| <div>  
| </div>  
</form>
```

Forms

We utilize forms to facilitate user interactions and data submissions, enabling a dynamic and engaging experience on our website.

```
<div id="OrderForm">  
  <form class="form" asp-controller="Order" asp-action="Create" method="post" onsubmit="return check()">  
    <div>  
      <input asp-for="Item2.OrderId" type="hidden" name="orderId" value="@Model.Item2.OrderId">  
    </div>  
  
    <div>  
      <input class="loginInput" asp-for="Item2.TableNo" name="tableNo" required  
        placeholder="Table Number">  
    </div>  
  
    <div>  
      <input class="loginInput" asp-for="Item2.CustomerName" name="customerName" required  
        placeholder="Customer Name">  
    </div>  
  
    <div>  
      <input asp-for="Item2.TotalPrice" type="hidden" id="totalPrice" name="totalPrice" value="0">  
    </div>  
  
    <button class="orderButton" type="submit">Order</button>  
  </form>  
</div>
```

Form Validations

In our project, we have used form validations to ensure the integrity and accuracy of user input, enhancing both the usability and security of our application.

```
[HttpPost]
0 references
public async Task<IActionResult> Create(int orderId, int tableNo, string customerName, decimal totalPrice)
{
    if (ModelState.IsValid)
    {
        var order = await _context.Orders.FindAsync(orderId);

        if (order != null)
        {
            order.TableNo = tableNo;
            order.CustomerName = customerName;
            order.OrderDate = DateTime.Now;
            order.TotalPrice = totalPrice;
        }
        else
        {
            return Json(new { success = false });
        }

        await _context.SaveChangesAsync();

        _context.ClearNullOrders();

        return RedirectToAction("MyOrder", new { id = orderId });
    }
    else
    {
        return Json(new { success = false });
    }
}
```

Partial View

In our project, we have used Partial Views to modularize and reuse common parts of our web pages, which increases maintainability and reduces redundancy in our code.

```
_OrderDetails.cshtml X
Views > Shared > _OrderDetails.cshtml
1 <div style="background-image: url('@Url.Content("~/images/OrderNote.svg")'); background-position: center;
2     background-repeat: no-repeat; background-size: cover; width: 350px; height: 470px;
3     padding-top: 75px; padding-left: 45px; padding-bottom: 100px; position: relative;
4     margin-left: 35px;" id="OrderDetails">
5
6     <div class="tableContainer" style="height: 100%; overflow: auto; width: 78%; padding-right: 15px;">
7         <table style="position: relative; width: 100%; display: flex; flex-direction: column;
8             justify-content: space-between;" id="tbl">
9             <tr style="display: flex; justify-content: space-between; align-items: center;
10                 border-bottom: 1px solid #b80f2b; margin-bottom: 15px;">
11                 <td style="font-family: 'Raleway', sans-serif;">Name</td>
12                 <td style="font-family: 'Raleway', sans-serif;">Quantity</td>
13                 <td style="font-family: 'Raleway', sans-serif;">Price</td>
14             </tr>
15         </table>
16     </div>
17
18     <div style="position: absolute;
19         bottom: 15%; left: 30%; font-family: 'Raleway', sans-serif;">Total:</div>
20     <div style="position: absolute;
21         bottom: 15%; left: 50%;" id="total">$ 0</div>
22 </div>
```

```
@await Html.PartialAsync("../Shared/_OrderDetails")
```

3.2.2 Entity Framework Core and SQLite

In our project, we have used Entity Framework Core with SQLite for efficient data management. This powerful combination has simplified the way we interact with our database, allowing us to benefit from the full potential of ORM. We connected to our database using DbContext and utilized migrations to keep our database updated.

```
builder.Services.AddDbContext<DataContext>(  
    Options =>  
    {  
        var config = builder.Configuration;  
        var conString = config.GetConnectionString("database");  
        Options.UseSqlite(conString);  
    });
```

▼ Migrations

- 20231219074342_CafeDB.cs
- 20231219074342_CafeDB.Design...
- 20231223144051_isAuthAdded.cs
- 20231223144051_isAuthAdded.D...
- DataContextModelSnapshot.cs

```
namespace CafeApp.Data  
{  
    13 references  
    public class DataContext : DbContext  
    {  
        17 references  
        public DbSet<Order> Orders { get; set; }  
  
        11 references  
        public DbSet<Product> Products { get; set; }  
  
        16 references  
        public DbSet<OrderProduct> OrderProducts { get; set; }  
  
        3 references  
        public DbSet<Barista> Baristas { get; set; }  
  
        2 references  
        public Barista? FindByEmail(string email)  
        {  
            return Baristas.SingleOrDefault(b => b.Email == email);  
        }  
  
        2 references  
        public async void ClearNullOrders()  
        {  
            var existingOrderList = await Orders.Where(o => o.TableNo == null).ToListAsync();  
  
            if (existingOrderList != null)  
            {  
                var orderIds = existingOrderList.Select(op => op.OrderId).ToList();  
  
                var orderProductWiththeOrderIdList = await OrderProducts  
                    .Where(op => orderIds.Contains(op.OrderId))  
                    .ToListAsync();  
  
                OrderProducts.RemoveRange(orderProductWiththeOrderIdList);  
                Orders.RemoveRange(existingOrderList);  
            }  
  
            await SaveChangesAsync();  
        }  
    }  
}
```

Product Data

	ProductId	Name	ImageSrc	Details	Price
	Filter	Filter	Filter	Filter	Filter
1	1	Latte	~/images/latte.jpg	It's a delightful beverage.	9
2	2	Americano	~/images/americano.jpg	It's a delightful beverage.	6
3	3	Filter Coffee	~/images/filter_coffee.jpg	It's a delightful beverage.	5
4	4	Flat White	~/images/flat_white.jpg	It's a delightful beverage.	8
5	5	Cortado	~/images/cortado.jpg	It's a delightful beverage.	10
6	6	Caramel Macchiato	~/images/caramel_macchiato.jpg	It's a delightful beverage.	13
7	7	White Chocolate Mocha	~/images/white_chocolate_mocha.jpg	It's a delightful beverage.	14
8	8	Caffè Mocha	~/images/caffe_mocha.jpg	It's a delightful beverage.	12
9	9	Cappuccino	~/images/cappuccino.jpg	It's a delightful beverage.	9
10	10	Brownie Cheesecake	~/images/brownie_cheesecake.jpg	It's a delightful dessert.	10
11	11	Blueberry Muffin	~/images/blueberry_muffin.jpg	It's a delightful dessert.	8
12	12	Chocolate Croissant	~/images/chocolate_croissant.jpg	It's a delightful dessert.	7
13	13	Chocolate Cookie	~/images/chocolate_cookie.jpg	It's a delightful dessert.	5
14	14	Coffee Cake	~/images/coffee_cake.jpg	It's a delightful dessert.	6
15	15	Lemon Loaf	~/images/lemon_loaf.jpg	It's a delightful dessert.	4

This assignment was completed entirely through group work. Each page was made together.