LAB 4. BUILD ADVANCED WEB APPLICATIONS WITH ASP.NET MVC 5 (1)

1. Introduction

This lab focuses on advanced topics in ASP.NET MVC 5, guiding you through the process of building a functional e-commerce website. We will move beyond in-memory data and connect our application to a real SQL Server database using Entity Framework. You will learn to handle large datasets efficiently by implementing pagination and filtering.

2. Objective

Upon completing this lab, you will be able to:

- Understand and use **Entity Framework** to interact with a SQL Server database.
- Create a complete **Home Page** and a **Product Details Page** for an e-commerce website.
- Implement a product listing page with **pagination** to manage large amounts of data.
- Build a **filtering** feature that allows users to view products by category.

3. Project setup

a. Project Initialization

- Open Visual Studio 2022 and create a new project using the ASP.NET Web Application (.NET Framework) template.
- Name the project **StoreMVC** and select MVC when prompted.

b. Entity Framework and Database Setup

- Open the **Package Manager Console** by navigating to **Tools** -> **NuGet Package Manager** -> **Package Manager Console**.
- Install Entity Framework by running the following command:

```
Install-Package EntityFramework
```

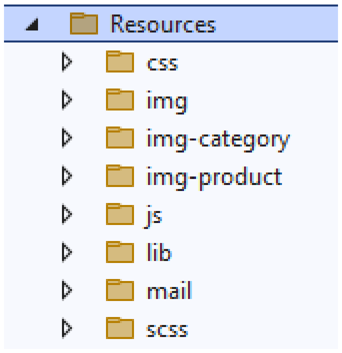
- In the **Web.config** file, add the following connection string inside the **<configuration>** section to connect to your SQL Server database. The name of the connection string must be **StoreMVC**.

```
<connectionStrings>
  <add name="StoreMVC" connectionString="Data
Source=(LocalDb)\MSSQLLocalDB;Initial Catalog=StoreMVC;Integrated
Security=SSPI;" providerName="System.Data.SqlClient" />
</connectionStrings>
```

c. Template and Layout setup

i) Copy Static Files:

- Copy the css, js, img, lib, mail, scss folders from your HTML template into the corresponding folders in your ASP.NET MVC project.



ii) Configure Layout:

- Open the **_Layout.cshtml** file in the Views/Shared folder. This file defines the common structure (header, footer, navigation) for all pages. Replace its content with the HTML structure from your template, making sure to include these essential ASP.NET MVC components:

- **RenderBody()**: This is where the content of each specific view will be rendered.

```
@RenderBody()
```

- **Partial Views**: Use **@Html.Partial()** to include reusable view components like the header or footer.

```
@Html.Partial("_Header")
```

```
@RenderBody()
@Html.Partial("_Footer")
```

- **Reference CSS/JS**: Ensure you correctly link to your CSS and JavaScript files using **@Url.Content()** to generate the correct file paths.

```
<!-- Customized Bootstrap Stylesheet -->
<link href="@Url.Content("~/Resources/css/style.css")" rel="stylesheet">

<!-- JavaScript Libraries -->
<script src="https://code.jquery.com/jquery-3.4.1.min.js"></script>
<script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/js/bootstrap.bundle.min.js"></script>
<script src="@Url.Content("~/Resources/lib/easing/easing.min.js")"></script>
<script
src="@Url.Content("~/Resources/lib/owlcarousel/owl.carousel.min.js")"></script>
```

4. Complete the following requirements

a. Implement Entity Framework

i) Create the Model classes:

- In the Models folder, create two new classes: **Product.cs** and **Category.cs**.
- Add the following code to define the **Product** class:

```
public class Product
{
   public int Id { get; set; }
   [Required]
   public string Name { get; set; }
   [Required]
   public decimal Price { get; set; }
   public string ImageUrl { get; set; }
   public bool IsFeatured { get; set; }
```

```
    public  int CategoryId { get; set; }

    public virtual Category Category { get; set; }
}

public int Id { get; set; }
[Required]
[StringLength(100)]
public string Name { get; set; }
public string ImageUrl { get; set; }

public virtual ICollection<Product> Products { get; set; }
```

ii) Create the DbContext class:

- In the **Models** folder, create a new class named **AppDbContext.cs**. This class will be the bridge between your application and the database.

```
public class AppDbContext : DbContext
    {
        public AppDbContext() : base("name=StoreMVC")
        {
        }


        public DbSet<Product> Products { get; set; }
        public DbSet<Category> Categories { get; set; }
    }
```

- Enable Migrations to use the Code First approach. In the **Package Manager Console**, run these commands in order:

```
Enable-Migrations
Add-Migration InitialCreate
Update-Database
```

- These commands will create a Migrations folder, a migration file, and finally the **Products** and **Categories** tables in your SQL Server database.

b. Build the Home Page

i) Create the HomeViewModel:

- In the **Models** folder, create a new class named HomeViewModel.cs. This class will be used to pass strongly-typed data from the controller to the home page view.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace StoreMVC.Models.View
{
    public class HomeViewModel
    {
        public List<Category> Categories { get; set; }
        public List<Product> AllProducts { get; set; }
        public List<Product> FeaturedProducts { get; set; }
    }
}
```

ii) Create the HomeController:

- Open the **HomeController.cs** file in the Controllers folder.
- Add an Index action method to retrieve a list of products and categories from the database.
- Modify the Index method to access the database, retrieve the necessary data, and pass it to the view.

```
public class HomeController : Controller
{
```

```
private readonly AppDbContext db = new AppDbContext();
public async Task<ActionResult> Index()
{
   try
   {
      var categories = await db.Categories.ToListAsync();
      var allProducts = await db.Products
                     .Where(p => p.IsFeatured == false)
                     .Take(8)
                     .ToListAsync();

      var featuredProduct = await db.Products
                        .Where(p => p.IsFeatured == true)
                        .Take(4)
                        .ToListAsync();

      HomeViewModel homeView = new HomeViewModel
      {
         Categories = categories,
         AllProducts = allProducts,
         FeaturedProducts = featuredProduct
      };

      ViewBag.Title = "Home page";
      return View(homeView);
   }
   catch (Exception ex)
   {
```

```
            return RedirectToAction("Error", "Home");
        }
    }
}
```

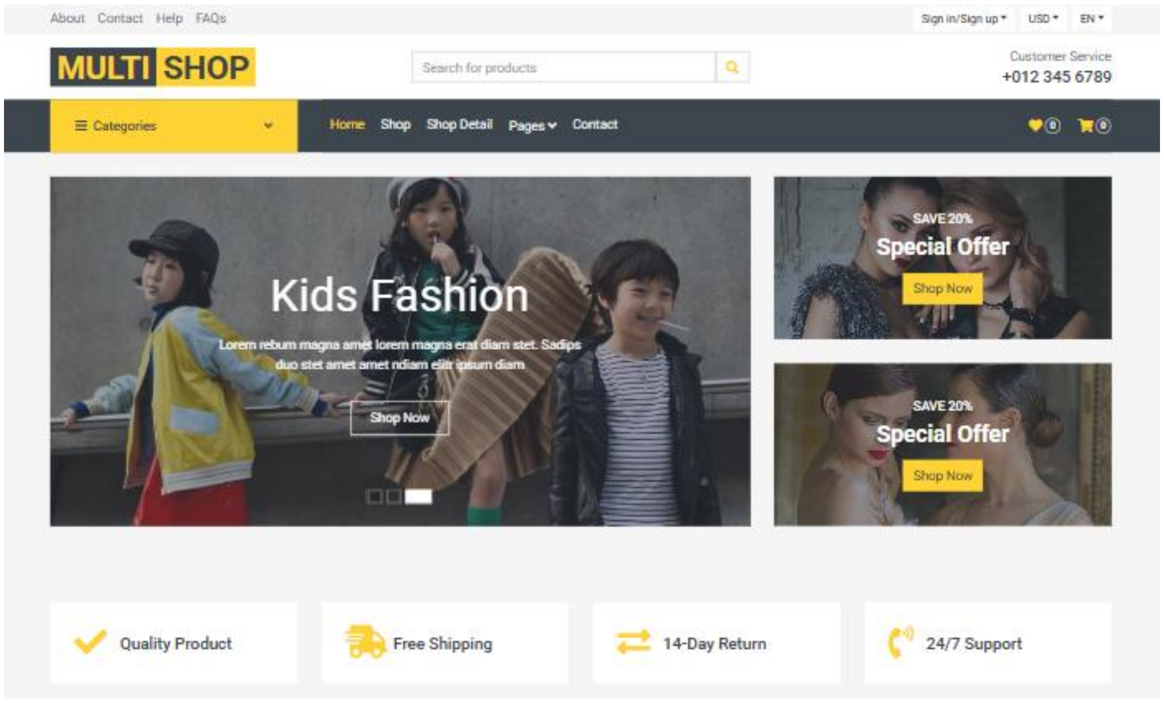
iii) Create Index View:

- Open the **Index.cshtml** file in the Views/Home folder.
- Modify the content of the view to display a list of products by category.
- Use **@model StoreMVC.Models.HomeViewModel** to access the strongly-typed data from the controller.
- Display **Product** in View.

```
<!-- Products Start -->
<div class="container-fluid pt-5 pb-3">
    <h2 class="section-title position-relative text-uppercase mx-xl-5 mb-4"><span
class="bg-secondary pr-3">Featured Products</span></h2>
    <div class="row px-xl-5">
        @if (Model.FeaturedProducts != null && Model.FeaturedProducts.Any())
        {
            foreach (var item in Model.FeaturedProducts)
            {
                <div class="col-lg-3 col-md-4 col-sm-6 pb-1">
                    <!--Featured Product Item-->
                    <div class="product-item bg-light mb-4">
                        <div class="product-img position-relative overflow-hidden">
                            <img class="img-fluid w-100" src="~/Resources/img-
product/@item.ImageUrl" alt="">
                            <div class="product-action">
                                <a class="btn btn-outline-dark btn-square" href=""><i class="fa fa-
shopping-cart"></i></a>
                                <a class="btn btn-outline-dark btn-square" href=""><i class="far fa-
heart"></i></a>
```

```
                                <a class="btn btn-outline-dark btn-square" href=""><i class="fa fa-
sync-alt"></i></a>
                                <a class="btn btn-outline-dark btn-square" href=""><i class="fa fa-
search"></i></a>
                            </div>
                        </div>
                        <div class="text-center py-4">
                            <a class="h6 text-decoration-none text-truncate"
href="@Url.Action("Detail","Store", new { id = item.Id })">@item.Name</a>
                            <div class="d-flex align-items-center justify-content-center mt-2">
                                <h5>$123.00</h5><h6 class="text-muted ml-2"><del>@item.Price
VND</del></h6>
                            </div>
                            <div class="d-flex align-items-center justify-content-center mb-1">
                                <small class="fa fa-star text-primary mr-1"></small>
                                <small class="fa fa-star text-primary mr-1"></small>
                                <small class="fa fa-star text-primary mr-1"></small>
                                <small class="fa fa-star text-primary mr-1"></small>
                                <small class="fa fa-star text-primary mr-1"></small>
                                <small>(99)</small>
                            </div>
                        </div>
                    </div>
                </div>
                <!--Featured Product Item-->
            }
        }
        else
        {
            <span class="text-danger">No Data</span>
        }
    </div>
</div>
<!-- Products End -->
```

- Featured Product and Category are displayed in View like Product.



c. Build the Product Details Page

i) Create the StoreController:

- In the Controllers folder, create a new controller named **StoreController.cs**.
- Add a **Detail** action method that takes a product ID as a parameter, retrieves the corresponding product from the database, and passes it to a view.

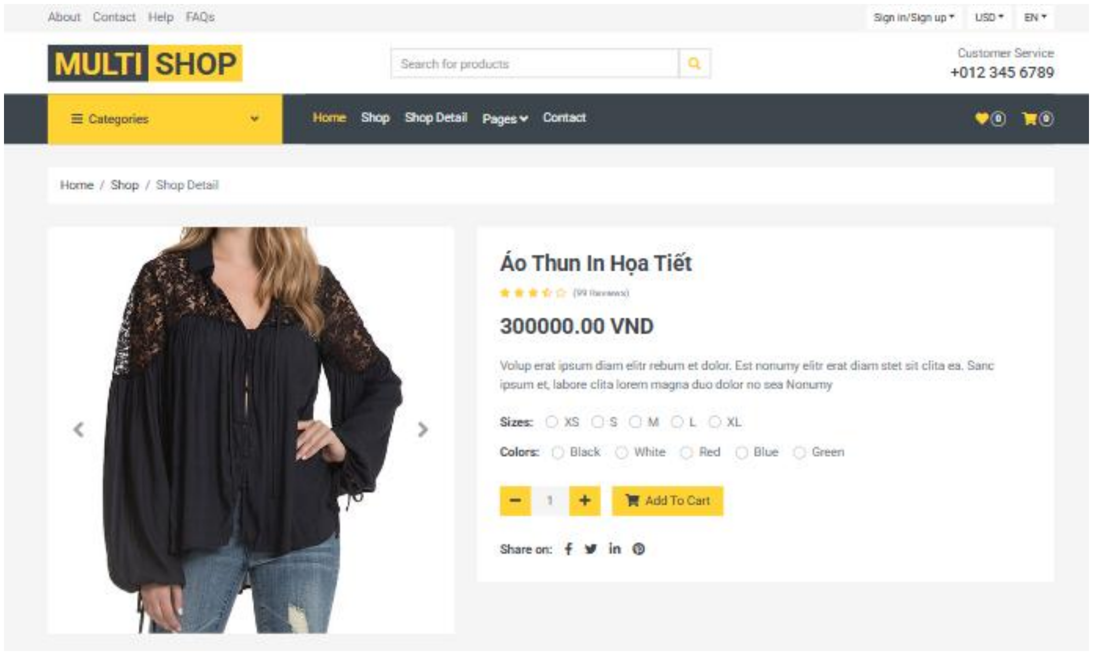
```
using System.Web.Mvc;
using System.Threading.Tasks;
using System.Data.Entity;
using StoreMVC.Models;
```

```
public class StoreController : Controller
{
   private readonly AppDbContext db = new AppDbContext();

   public async Task<ActionResult> Detail(int id)
{
   try
   {
      var product = await db.Products.FindAsync(id);

      if (product == null)
      {
         return View("Error");
      }

      ViewBag.Title = "Product detail page";
      return View(product);
   }
   catch (Exception ex)
   {
      ViewBag.Error = ex.Message;
      return View("Error");
   }
}

}
```

ii) Create the Details View:

- In the Views folder, create a new subfolder named Store.

- Right-click the Store folder, select Add -> View..., and name it **Details.cshtml**.
- Paste the content from your provided template file for the details page.



Áo Thun In Họa Tiết

300000.00 VND

Volup erat ipsum diam elitr rebum et dolor. Est nonumy elitr erat diam stet sit clita ea. Sanc ipsum et, labore clita lorem magna duo dolor no sea Nonumy

d. Build the Product List Page with Pagination

i) Add the Index action to StoreController:

- Modify the **StoreController** to handle the main product list page. This method will fetch all products and prepare them for pagination.

```
private readonly int pageSize = 5;
public async Task<ActionResult> Index(int? page)
{
   try
   {
      int pageNumber = (page ?? 1);


      var products = await db.Products.OrderBy(p => p.Id)
                        .Skip((pageNumber - 1) * pageSize)
                        .Take(pageSize)
                        .ToListAsync();
```

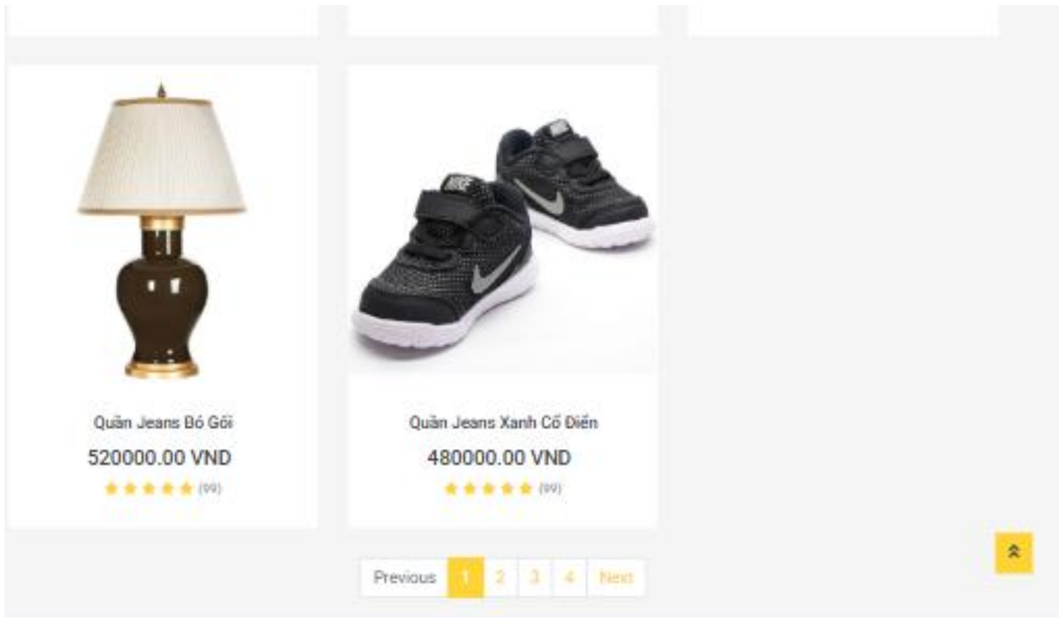
```
        int totalProduct = await db.Products.CountAsync();

        ViewBag.TotalPages = (int)Math.Ceiling((double)totalProduct / pageSize);
        ViewBag.CurrentPage = pageNumber;


        ViewBag.Title = "Store page";
        return View(products);
    }
    catch (Exception ex)
    {
        ViewBag.Error = ex.Message;
        return View("Error");
    }
}
```

ii) Create the Index View:

- In the Views/Store folder, create a new view named **Index.cshtml**.
- Add the necessary code to display the product list and the pagination links. This code will use ViewBag to get the pagination information.

```
<div class="col-12">
    @if (ViewBag.TotalPages > 1)
    {
        <nav>
            <ul class="pagination justify-content-center">


                <!-- Nút Previous -->
                <li class="page-item @(ViewBag.CurrentPage == 1 ? "disabled" : "")">
                    <a class="page-link" href="@Url.Action("Index", new { page =
ViewBag.CurrentPage - 1 })">Previous</a>
```

```
            </li>
            @for (int i = 1; i <= ViewBag.TotalPages; i++)
            {
                <li class="page-item @(i == ViewBag.CurrentPage ? "active" : "")">
                    <a class="page-link" href="@Url.Action("Index", new { page = i
})">@i</a>
                </li>
            }
            <li class="page-item @(ViewBag.CurrentPage == ViewBag.TotalPages ?
"disabled" : "")">
                <a class="page-link" href="@Url.Action("Index", new { page =
ViewBag.CurrentPage + 1 })">Next</a>
            </li>
        </ul>
    </nav>
  }
</div>
```



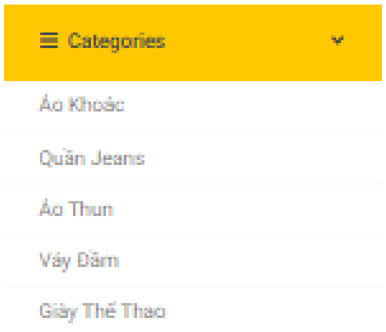
e. Implement the Category Navbar

A partial view is a reusable view component. You will create one to display the list of categories and then add it to your main layout.

- Create a partial view named **_CategoryNavbar.cshtml** in the **Views/Shared** folder. This view will list all categories as links.
- Next, create a new controller named **CategoryController.cs** in the **Controllers** folder. Add a CategoryNavbar action to pass a list of categories to the partial view.

```
using StoreMVC.Models;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

public class CategoryController : Controller
{
   private readonly AppDbContext db = new AppDbContext();


   [ChildActionOnly]
   public  PartialViewResult _CategoryNavbar()
   {
      List<Category> categories = db.Categories.ToList();
      return PartialView(categories);
   }
}
```

- Finally, use **@Html.Action("_CategoryNavbar","Category")** in your main **_Layout.cshtml** to render the category list on every page.



5. Exercise

- Implement Category Filtering
- In this exercise, you will add the functionality to filter products by category. This involves modifying the **StoreController** to accept a category ID and creating a partial view for the category navigation menu.
- Modify the Index action in StoreController.cs to accept an optional categoryId parameter. The query will then be adjusted to filter products based on this ID.

6. Summary

In this lab, you successfully built an advanced e-commerce website using ASP.NET MVC 5. You learned how to use Entity Framework to connect to a SQL Server database, manage data with pagination, and filter products by category. The skills you've acquired—from database interaction to efficient data display—are fundamental for building any robust web application.