04 Display Products Index

DISPLAY PRODUCTS FROM THE DATABASE ON AN ADMIN PAGE

Goal

Create an admin page that reads products from the database and displays them.
This proves your database connection works and your MVC flow works.

MVC flow you must understand

Controller gets data from database
Controller sends data to View
View renders HTML with data

Step 1. Create AdminProductController

To Create AdminProductController.cs in Controllers.

Right click on the controller folder and then click on Add and then click controller

Then name the controller AdminProductController

Then inside the file

```
 1. using Microsoft.AspNetCore.Mvc;
 2. using Microsoft.EntityFrameworkCore;
 3. using RTWebApplication.DbServices;
 4. using RTWebApplication.Models;
 5.
 6. namespace RTWebApplication.Controllers
 7. {
 8.     public class ProductController : Controller
 9.     {
10.         private readonly ApplicationDbContext context;
11.         private readonly IWebHostEnvironment webHostEnvironment;
12.
13.         public ProductController(ApplicationDbContext context, IWebHostEnvironment
webHostEnvironment)
14.         {
15.             this.context = context;
16.             this.webHostEnvironment = webHostEnvironment;
17.         }
18.
19.         [HttpGet]
20.         public IActionResult Index(string search)
21.         {
22.             IQueryable<[MODEL NAME]> productsQuery = context.[MODEL NAME];
23.
24.             if (!string.IsNullOrEmpty(search))
25.             {
26.                 string pattern = $"%{search}%";
27.                 productsQuery = productsQuery.Where(p =>
28.                     EF.Functions.Like(p.Name, pattern) ||
29.                     EF.Functions.Like(p.Brand, pattern) ||
30.                     EF.Functions.Like(p.Category, pattern) ||
31.                     EF.Functions.Like(p.Description, pattern)
32.                 );
33.             }
34.             var products = productsQuery.OrderByDescending(p => p.Id).ToList();
35.             return View(products);
36.         }
37.
```

Explanation in detail

1. ApplicationDbContext is injected through the constructor.
2. This is dependency injection. It avoids manually creating the context.
3. IQueryable keeps the query as SQL until ToList runs.
4. OrderByDescending helps show newest items first.
5. View(products) sends a List<Product> to the view.

Step 2. Create the Index view

ENSURE PROGRAM IS NOT RUNNING

THEN RIGHT CLICK ON VIEW in return View() and then a menu should pop up then click on Add view and then name the file THE SAME NAME AS THE METHOD in this case that's Index and that should automatically create the Index view for products

Then in the index view at the top add

```
@model List<Model Name> - most likely Products

```

Then add the page markup.

```
<h2>Products</h2>

<form method="get">
    <input type="text"
           name="search"
           placeholder="Search by name"
           value="@Context.Request.Query["search"]" />
    <button type="submit">Search</button>
</form>

<table class="table">
    <thead>
        <tr>
            <th>Id</th>
            <th>Name</th>
            <th>Price</th>
            <th>Created</th>
        </tr>
    </thead>
    <tbody>
        @foreach (var p in Model)
        {
            <tr>
                <td>@p.Id</td>
                <td>@p.Name</td>
                <td>@p.Price</td>
                <td>@p.CreatedAt</td>
            </tr>
        }
    </tbody>
</table>

```

Explanation

@model tells Razor what type the view expects.

The foreach renders one table row per product.

The search box keeps its value after searching by reading the query string.

Step 3. Add a navigation link to the page

In Views/Shared/_Layout.cshtml add a link.
Place inside the nav list

```
1. <li class="nav-item">
2.     <a class="nav-link text-dark" asp-controller="AdminProduct" asp-
action="Index">Admin Products</a>
3. </li>
4.
```

Explanation

asp-controller and asp-action generate the correct URL.
This avoids hardcoding routes.

Common errors and fixes

Error: The model item passed into the ViewDataDictionary is of type X
Cause: View expects List<Product> but controller returned a different type
Fix: Ensure controller returns View(products) and view uses @model List<Product>

Error: InvalidOperationException. Unable to resolve service for ApplicationDbContext
Cause: DbContext not registered in Program.cs
Fix: Add AddDbContext in Program.cs