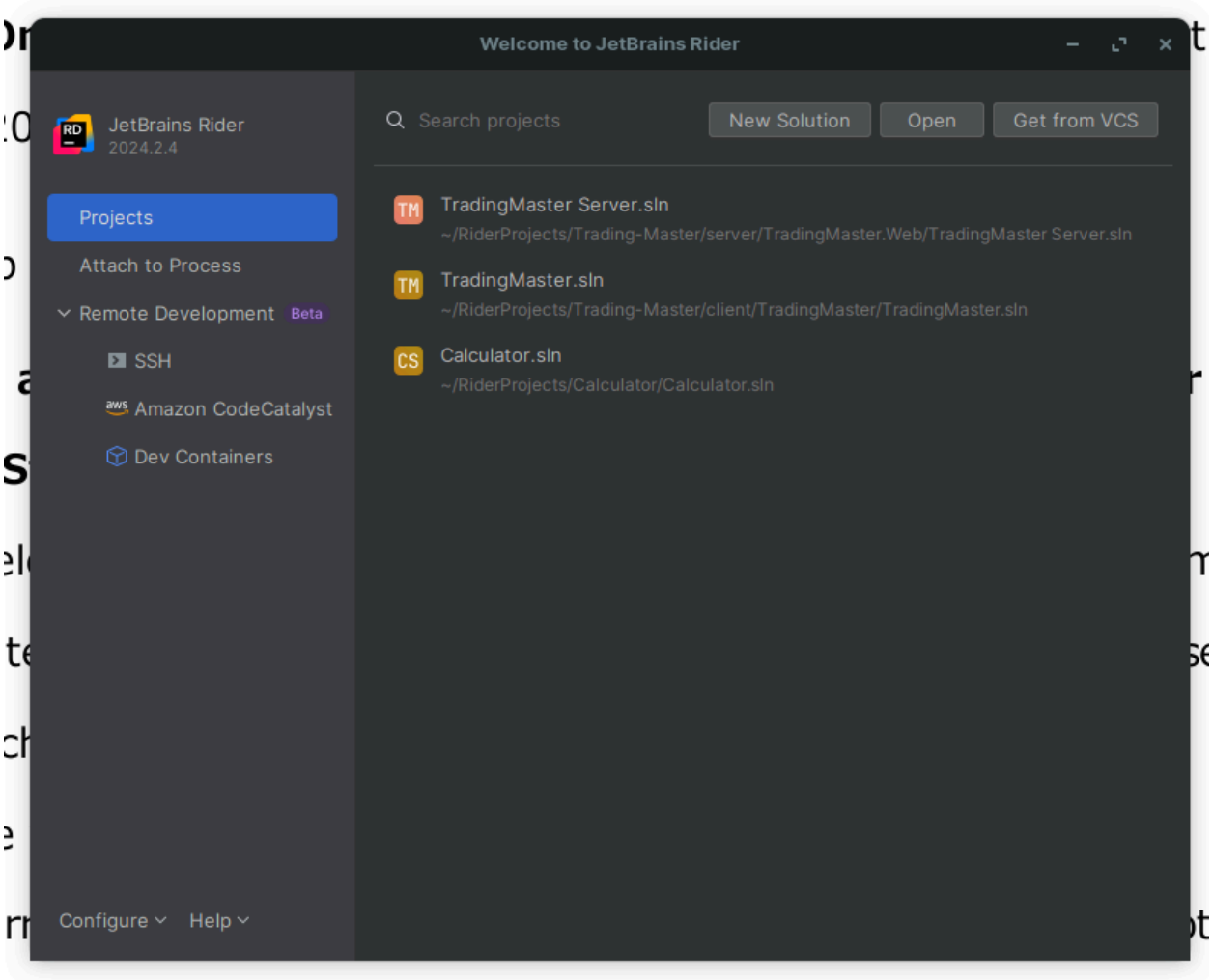
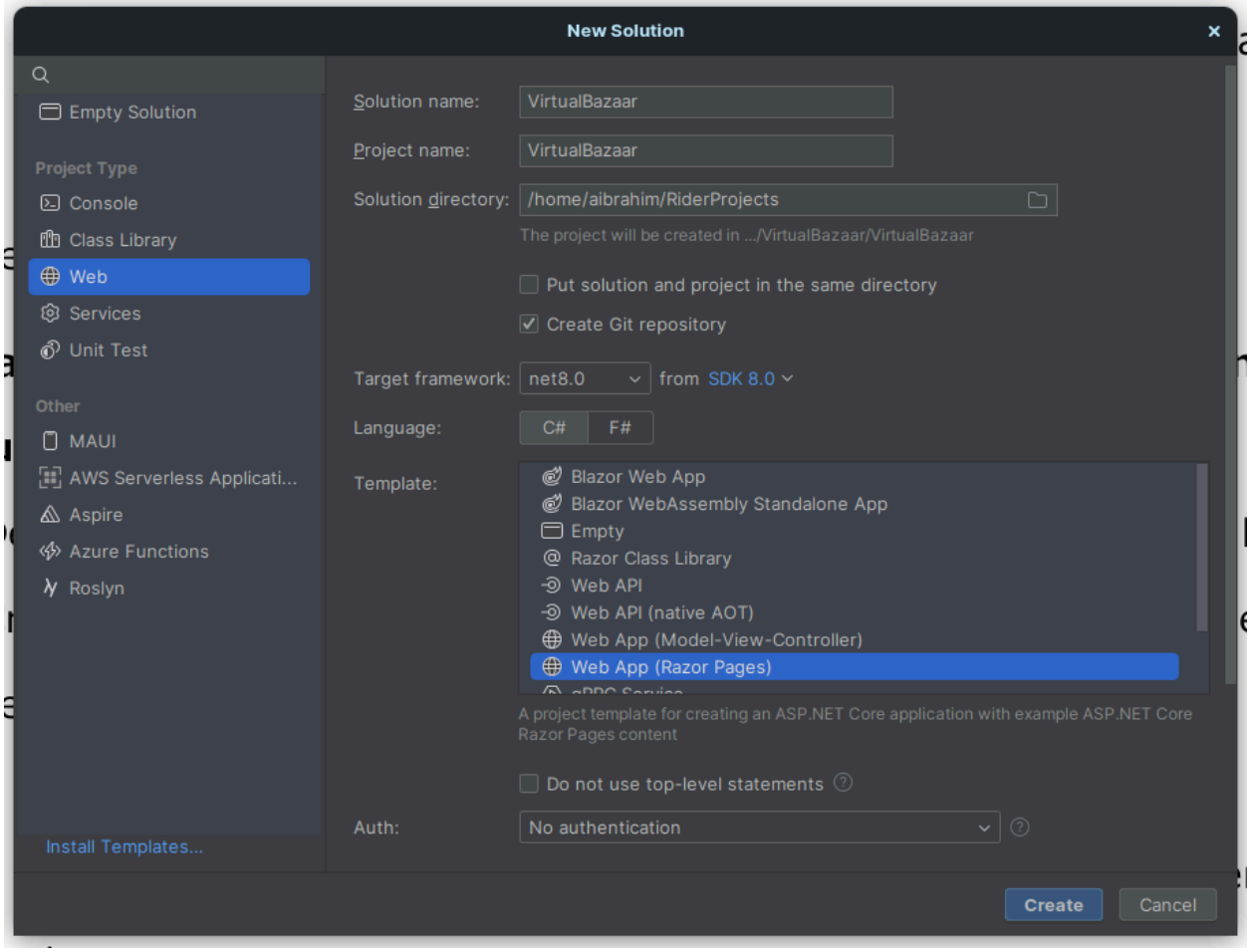
E-Commerce Project Report
Ahmad IBRAHIM - solo

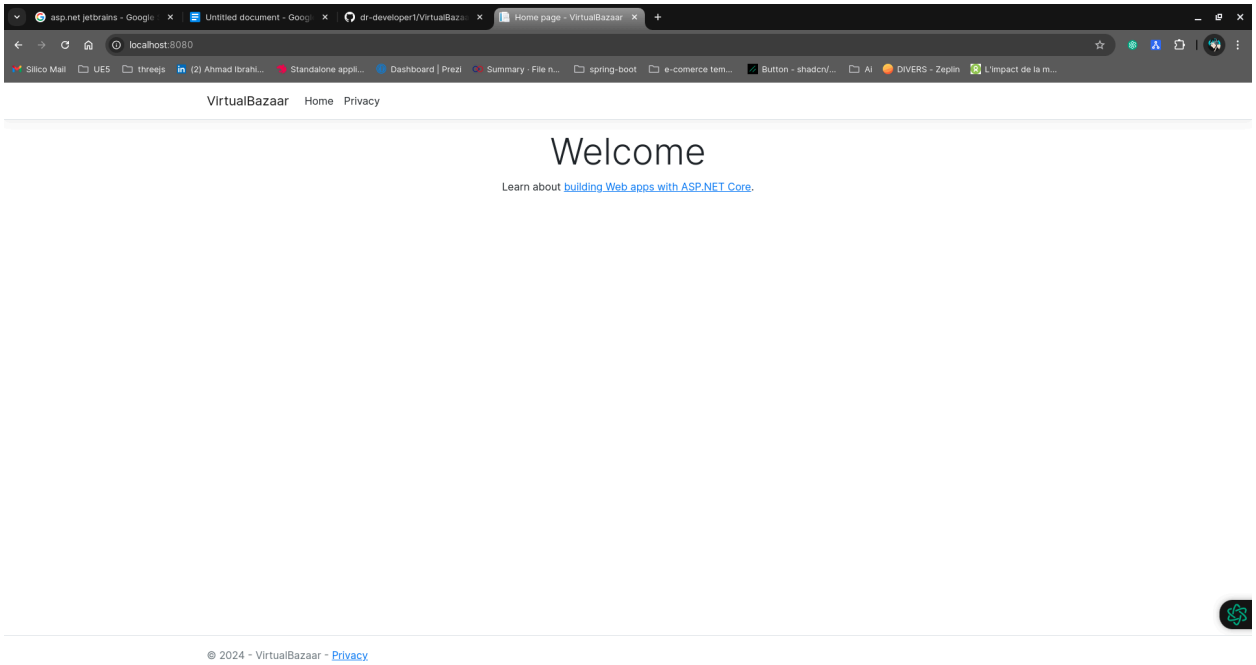



New Solution
Empty Solution
Project Type
Console
Class Library
Web
Services
Unit Test
Other
MAUI
AWS Serverless Applicati...
Aspire
Azure Functions
Roslyn
Install Templates...
Solution name:
VirtualBazaar
Project name:
VirtualBazaar
Solution directory:
/home/aibrahim/RiderProjects
The project will be created in .../VirtualBazaar/VirtualBazaar
Put solution and project in the same directory
Create Git repository
Target framework:
net8.0
from SDK 8.0
Language:
C#
F#
Template:
Blazor Web App
Blazor WebAssembly Standalone App
Empty
Razor Class Library
Web API
Web API (native AOT)
Web App (Model-View-Controller)
Web App (Razor Pages)
A project template for creating an ASP.NET Core application with example ASP.NET Core Razor Pages content
Do not use top-level statements
Auth:
No authentication
Create
Cancel

docker-compose.yml

```yaml
services:
  virtualbazaar:
    image: virtualbazaar
    build:
      context: .
      dockerfile: VirtualBazaar/Dockerfile
    ports:
      - "8080:8080"
      - "8081:8081"
  db:
    image: mcr.microsoft.com/mssql/server:latest
    environment:
      SA_PASSWORD: "Virtu@lBaz@r2024!"
      ACCEPT_EULA: "Y"
    ports:
      - "1433:1433"
    volumes:
      - dbdata:/var/opt/mssql
volumes:
  dbdata:
```

You, 2 minutes ago • Uncommitted changes



1. Introduction

This report details the development of an e-commerce web application using ASP.NET Core MVC, as per the requirements outlined in the project brief. The application implements basic CRUD (Create, Read, Update, Delete) operations and a search functionality for products.

2. Running the Project with Docker

Our e-commerce application can be easily run using Docker, which ensures consistency across different environments. Follow these steps to run the project:

- Ensure Docker and Docker Compose are installed on your system.
- Clone the project repository:
-
- Build and run the Docker containers:

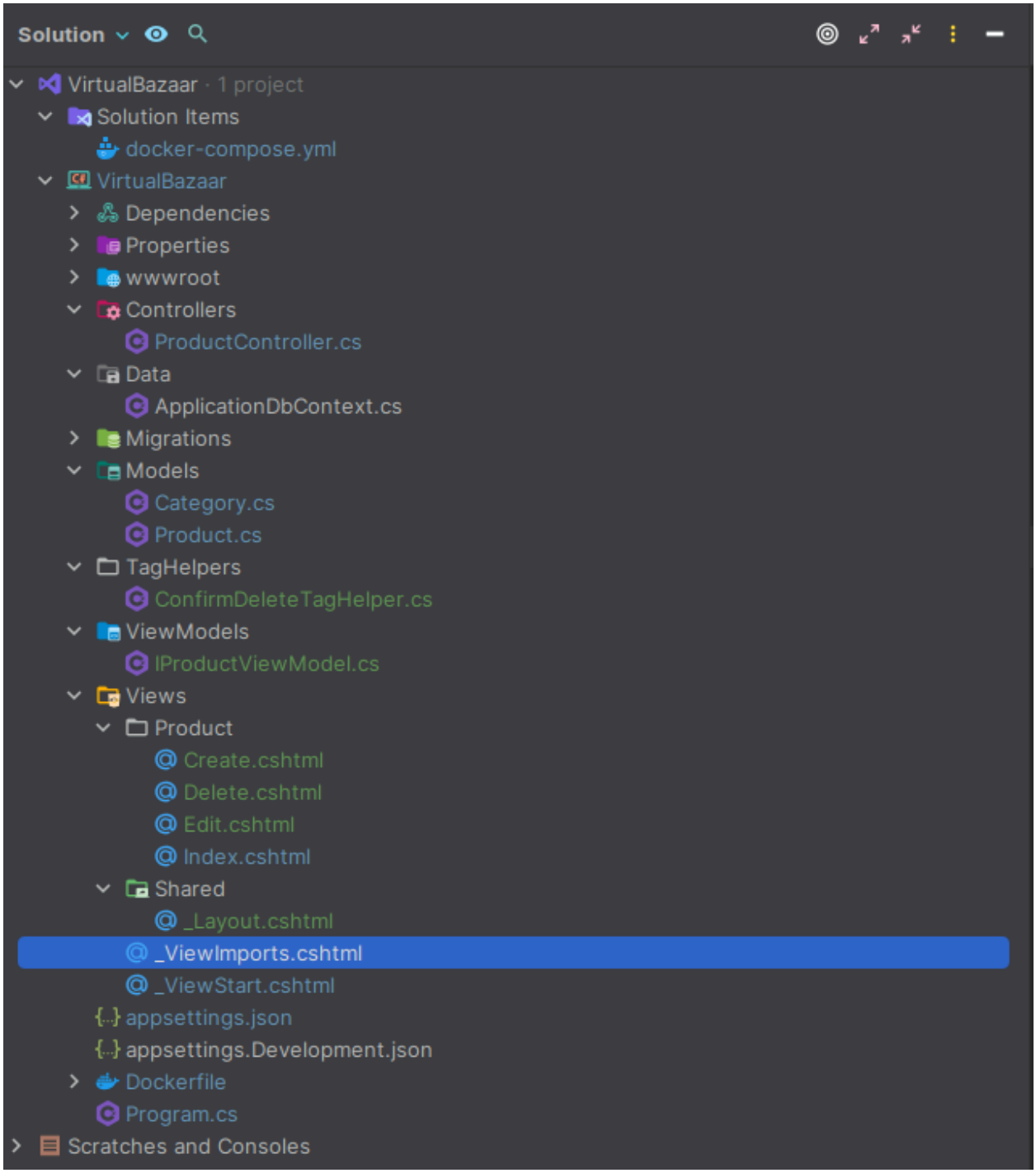
- docker-compose up --build
- Once the containers are up and running, you can access the application by opening a web browser and navigating to:
- http://localhost:8080
- To stop the application, use:
- docker-compose down

This process encapsulates all dependencies and configurations, making it simple to run the e-commerce application on any system with Docker installed.

3. Project Structure

The project follows the Model-View-Controller (MVC) architecture, typical of ASP.NET Core applications. Key components include:

- Models: Represent the data structure
- Views: Handle the presentation layer
- Controllers: Manage the application logic
- TagHelpers: Provide custom HTML-like syntax for complex rendering



4. Models

The `Product` class is the primary model, representing items for sale:

```
public class Product
{
    public int Id { get; set; }

    [Required]
    [StringLength(100, MinimumLength = 3)]
    public required string Name { get; set; }

    public DateTime CreatedDate { get; set; } =
DateTime.Now;
    [Required] [Range(0.01, 10000.00)]
    public float Price { get; set; }

    [EnumDataType(typeof(Category))]
    public Category Category { get; set; } =
Category.Electronics;

    public string ImageUrl { get; set; }
}
```

```
public enum Category
{
    Electronics,
    Clothing,
    Books,
    HomeDecor,
    Sports
}
```

5. Controllers

The ProductController handles CRUD operations and search functionality:

```
using ...

namespace VirtualBazaar.Controllers;

[Route("[controller]/[action]")]
9 usages  aibrahim *
public class ProductController(ILogger<ProductController> logger)
    : Controller   aibrahim, Yesterday • create product controller
{
    private static readonly List<Product> Products = [];

    10 usages  aibrahim *
    public IActionResult Index(string searchString, Category? category)
    {
        logger.LogInformation("Retrieving all products");

        var filteredItems:List<Product> = Products;

        if (!string.IsNullOrEmpty(searchString))
        {
            filteredItems = filteredItems.Where(i:Product => i.Name.Contains(searchString, StringComparison.OrdinalIgnoreCase)) // IEnumerable<Product>
                .ToList(); // List<Product>
        }

        if (category.HasValue)
        {
            filteredItems = filteredItems.Where(i:Product => i.Category == category.Value).ToList();
        }

        var viewModel = new ProductViewModel
            { Products = filteredItems, SearchString = searchString, SelectedCategory = category };
        return View(viewModel);
    }

    2 usages  aibrahim
```

6. Views

Views were created for each CRUD operation:

- Index.cshtml: Displays product list and search functionality
- Create.cshtml: Form for adding new products
- Edit.cshtml: Form for editing existing products
- Delete.cshtml: Confirmation page for product deletion

7. Custom Tag Helpers

A custom ConfirmDeleteTagHelper was implemented to enhance the delete functionality:

```
[HtmlTargetElement("confirm-delete", TagStructure =
TagStructure.WithoutEndTag)]
public class ConfirmDeleteTagHelper : TagHelper
{
    public string Message { get; set; } = "Are you sure you want to
delete this item?";

    public override void Process(TagHelperContext context,
TagHelperOutput output)
    {
        output.TagName = "button";
        output.Attributes.SetAttribute("type", "submit");
        output.Attributes.SetAttribute("class", "btn btn-danger");
        output.Attributes.SetAttribute("onclick", $"return
confirm('{Message}');");
        output.Content.SetContent("Delete");
    }
}
```

8. Search Functionality

The search feature allows users to filter products by name and category

9. Styling and Layout

Bootstrap was used for styling, with a custom layout implemented in _Layout.cshtml

10. Image Handling

Product images are uploaded and stored in the /wwwroot/images/ directory

11. Docker Implementation

The application is containerized using Docker

12. CRUD Operations Demonstration

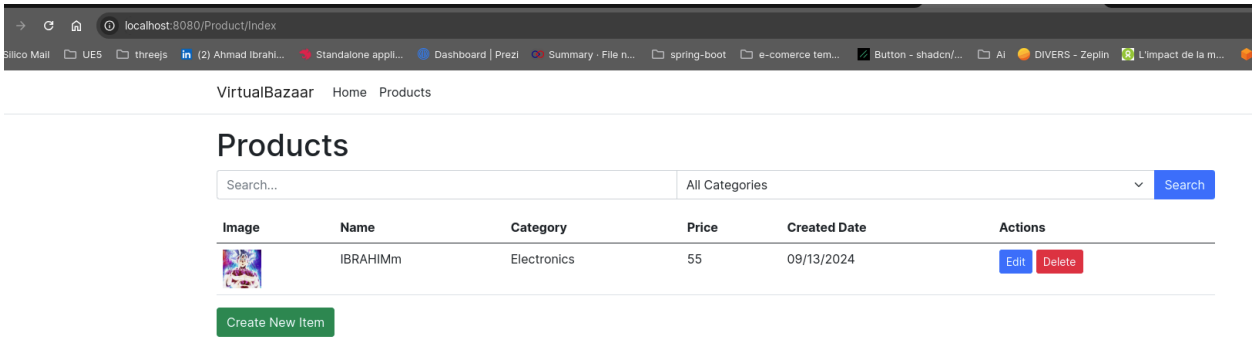
```
10 usages  aibrahim *
public IActionResult Index(string searchString, Category? category){...}

2 usages  aibrahim
public IActionResult Create(){...}

[HttpPost]
2 usages  aibrahim *
public IActionResult Create(Product product, IFormFile image){...}

2 usages  aibrahim *
public IActionResult Edit(int? id){...}

[HttpPost]
2 usages  aibrahim *
public IActionResult Edit(Product product, IFormFile image){...}

1 usage  aibrahim *
public IActionResult Delete(int? id){...}

[HttpPost]
1 usage  aibrahim *
public IActionResult DeleteConfirmed(int id){...}
```

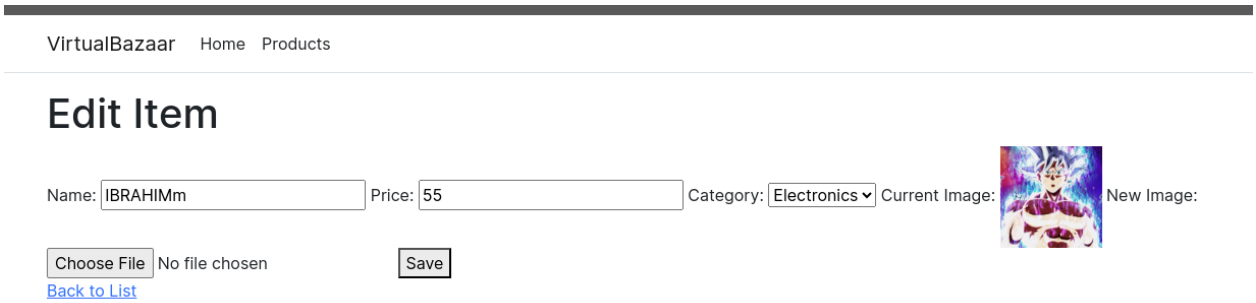
13. Challenges and Solutions

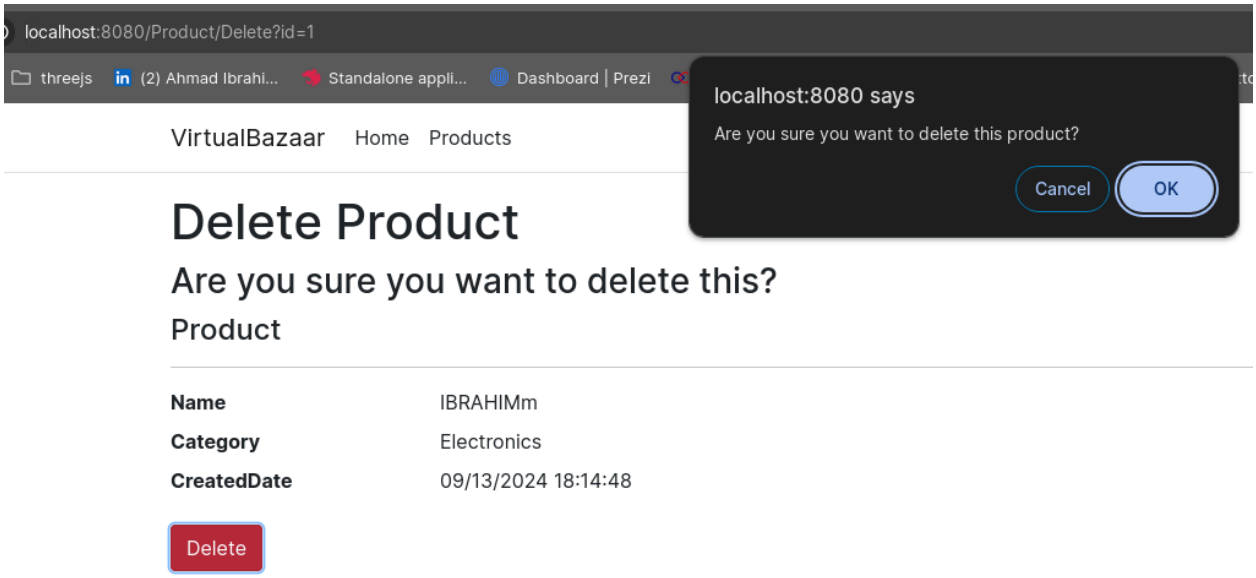
One significant challenge was handling file permissions for image uploads within the Docker container. This was resolved by adjusting the Dockerfile to create the necessary directory with appropriate permissions.

14. Conclusion

This project successfully implements the required CRUD operations, search functionality, and custom tag helpers using ASP.NET Core MVC. It demonstrates proficiency in C#, Razor syntax, and Docker containerization.


localhost:8080/Product/Index
VirtualBazaar
Home
Products
Products
Search...
All Categories
Search
Image
Name
Category
Price
Created Date
Actions
IBRAHIMm
Electronics
55
09/13/2024
Edit
Delete
Create New Item


VirtualBazaar
Home
Products
Edit Item
Name:
IBRAHIMm
Price:
55
Category:
Electronics
Current Image:
New Image:
Choose File
No file chosen
Save
Back to List


localhost:8080/Product/Delete?id=1
threejs
(2) Ahmad Ibrahi...
Standalone appli...
Dashboard | Prezi
localhost:8080 says
Are you sure you want to delete this product?
Cancel
OK
VirtualBazaar
Home
Products
Delete Product
Are you sure you want to delete this?
Product
Name
IBRAHIMm
Category
Electronics
CreatedDate
09/13/2024 18:14:48
Delete

localhost:8080/Product/Create

threejs | (2) Ahmad Ibrahi... | Standalone appli... | Dashboard | Prezi | Summary · File n... | spring-boot | e-comerce tem... | Button - shadcn/... | Ai | DIVERS - Zeplin | L'Impact de la m

VirtualBazaar Home Products

Create Product

Name: Price: Category: Electronics Image: Choose File No file chosen Create

Back to List