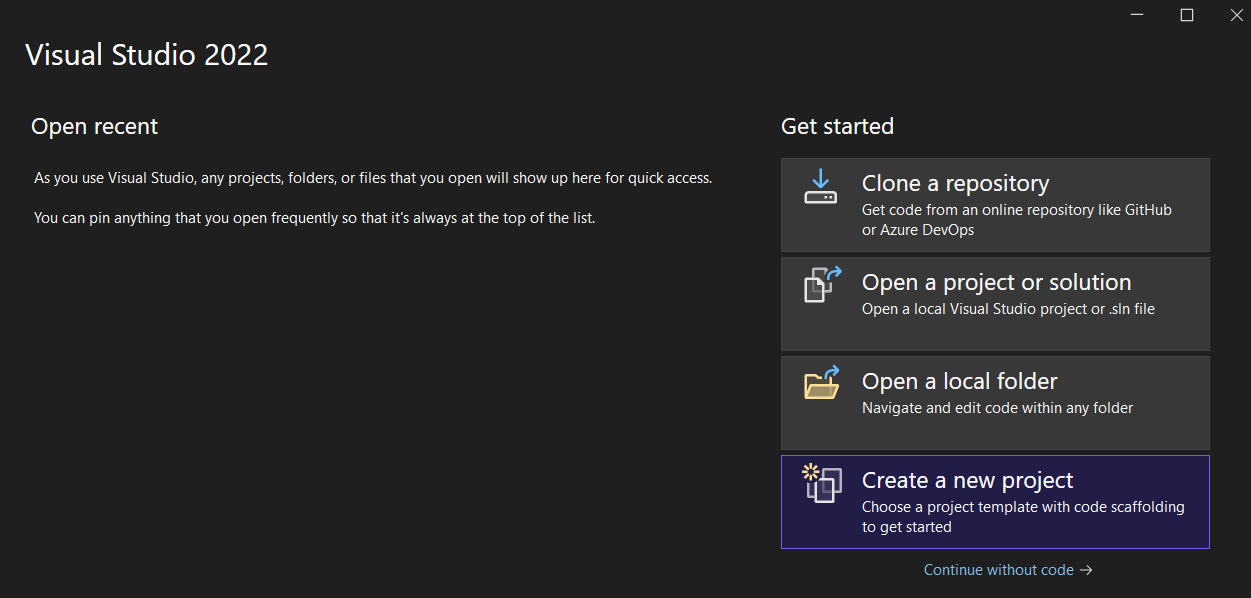
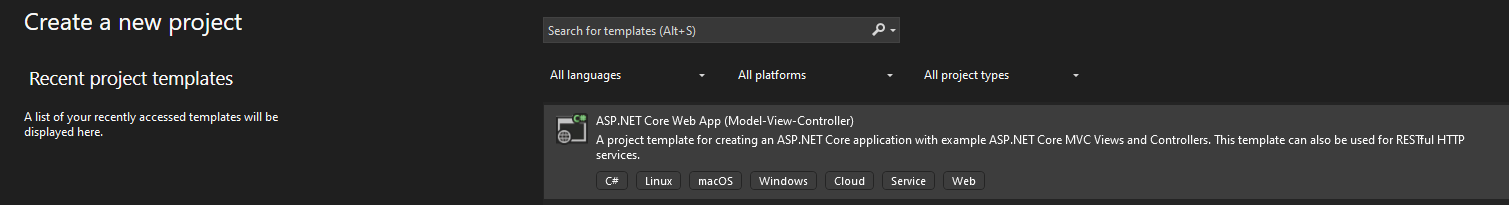
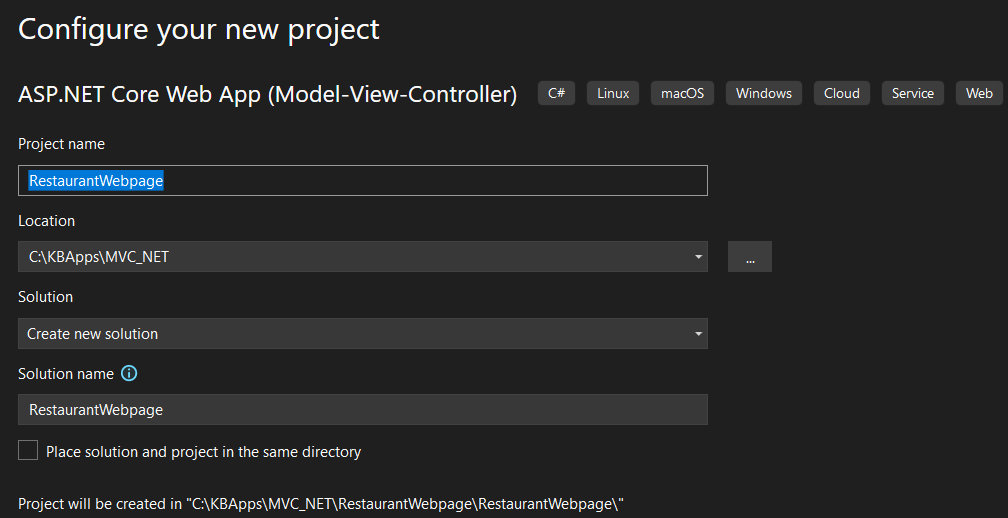
# ASP .NET CORE MVC

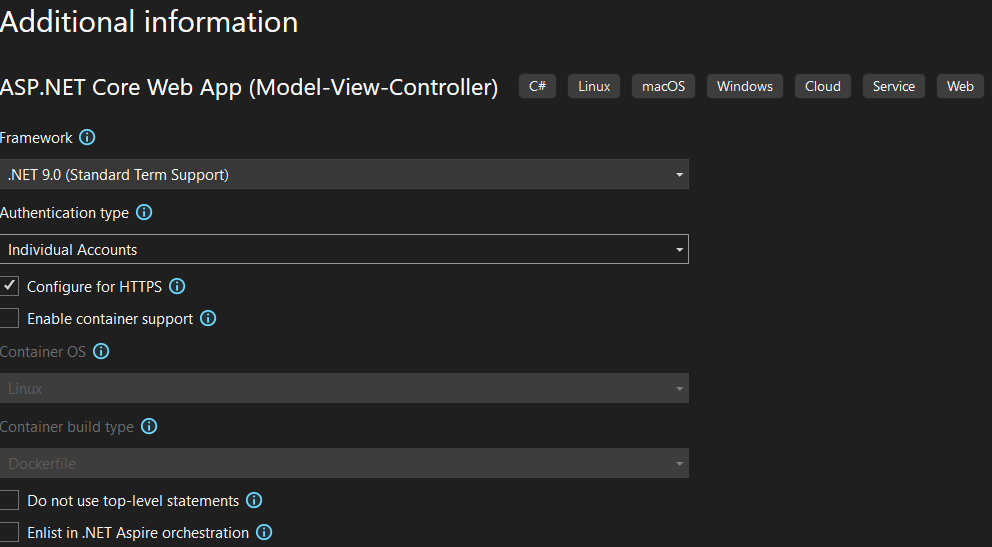
Based on: [ASP.NET Core MVC Tutorial – Full Course to Build YOUR Passion Project!](https://www.youtube.com/watch?v=q9X3SDEZtpw)

## Create MVC project using Visual Studio.









Go to appsettings.json and set the database configuration

{

"ConnectionStrings": {

"DefaultConnection": "Server=(localdb)\\mssqllocaldb;Database=RestaurantWebpageDb;Trusted\_Connection=True;MultipleActiveResultSets=true"

},

"Logging": {

"LogLevel": {

"Default": "Information",

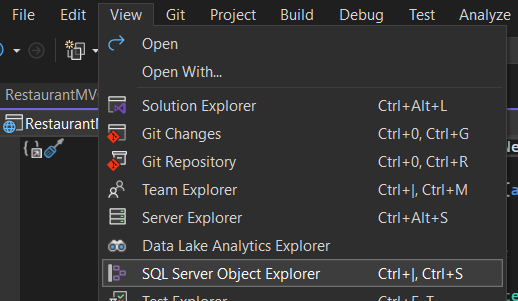
"Microsoft.AspNetCore": "Warning"

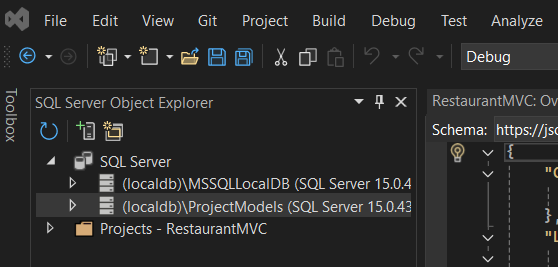
}

},

"AllowedHosts": "\*"

}

Go to:  




If we go to Data/applicationDbContext.cs we can see that it has:

using Microsoft.AspNetCore.Identity.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore;

namespace RestaurantWebpage.Data

{

public class ApplicationDbContext : IdentityDbContext

{

public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)

: base(options)

{

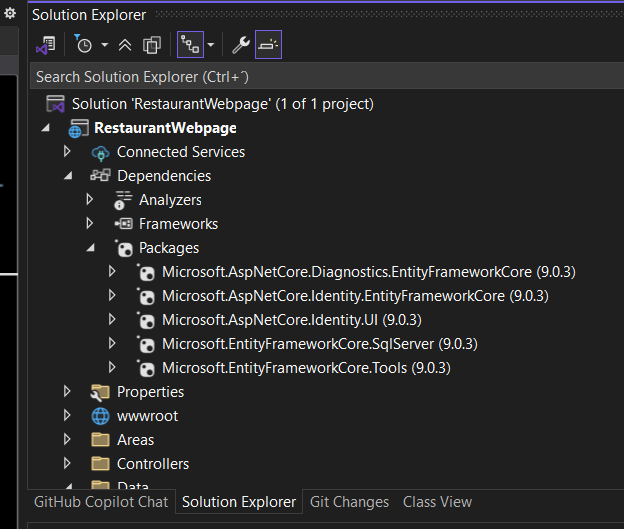
}

}

}

It has IdentityDbContext since we selected in the Authentication type: Individual Accounts.

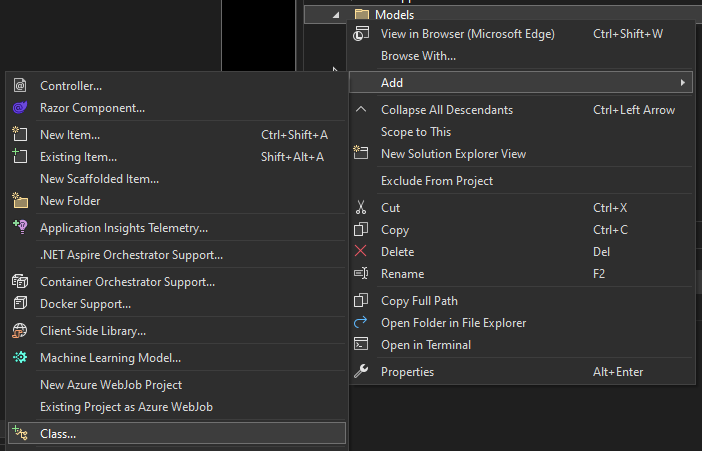
On packages we can see the installed packages:

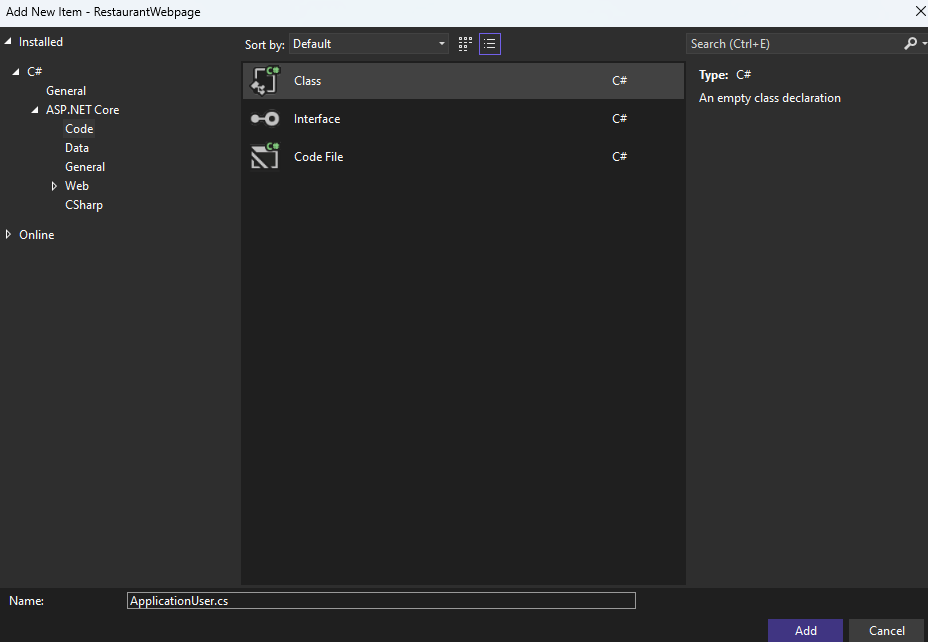


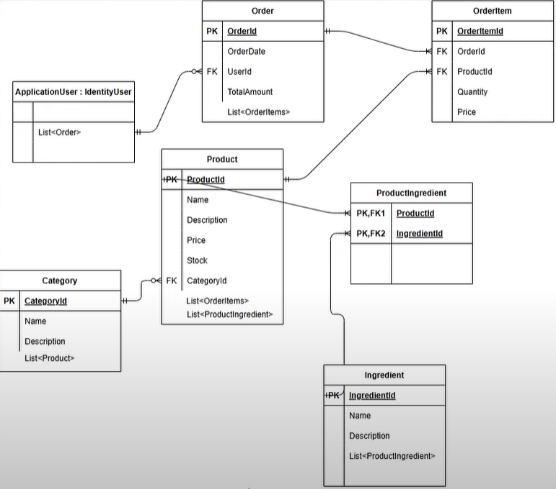
## Creating Models

Create new Model for users called ApplicationUser.cs:

On Models, right click and Add Class:





It should inherit from IdentityUser, note in the model it is related with a list of order:

using Microsoft.AspNetCore.Identity;

namespace RestaurantWebpage.Models

{

public class ApplicationUser: IdentityUser

{

public ICollection<Order>? Orders { get; set; }

}

}

as it is required to have the orders Model, create at Models Order.cs:

namespace RestaurantWebpage.Models

{

public class Order

{

public int OrderId { get; set; }

public DateTime OrderDate { get; set; }

public string? UserId { get; set; }

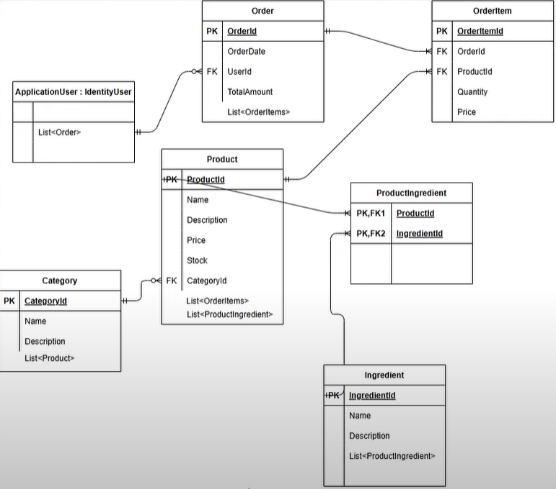
public ApplicationUser User { get; set; }

public decimal TotalAmount { get; set; }

public ICollection<OrderItem> OrderItems { get; set; }

}

}



It also needs to have a Model OrderItem.cs:

namespace RestaurantWebpage.Models

{

public class OrderItem

{

public int OrderItemId { get; set; }

public string OrderId { get; set; }

public Order? Order { get; set; }

public Product? Product { get; set; }

public int Quantity { get; set; }

public decimal Price { get; set; }

}

}

Create also Product.cs model:

namespace RestaurantWebpage.Models

namespace RestaurantWebpage.Models

{

public class Product

{

public int ProductId { get; set; }

public string? Name { get; set; }

public string? Description { get; set; }

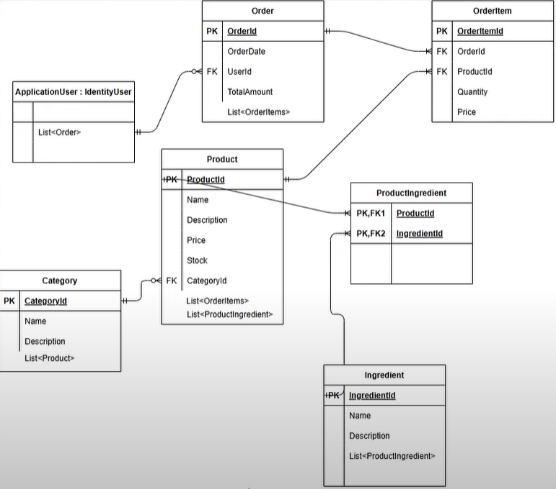
public decimal Price { get; set; }

public int Stock { get; set; }

public int CategoryId { get; set; }

public Category? Category { get; set; } // A product belongs to a category

public ICollection<OrderItem>? OrderItems { get; set; } // A product can be in many order items

 public ICollection<ProductIngredient>? ProductIngredients { get; set; } // A product can have many ingredients

}

}

In Category.cs model:  
namespace RestaurantWebpage.Models

{

public class Category

{

public int CategoryId { get; set; }

public string Name { get; set; }

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

}

}

In ProductIngredient.cs model:

namespace RestaurantWebpage.Models

{

public class ProductIngredient

{

public int ProductId { get; set; }

public Product Product { get; set; }

public int IngredientId { get; set; }

public Ingredient Ingredient { get; set; }

}

}

And in Ingredient.cs model:

namespace RestaurantWebpage.Models

{

public class Ingredient

{

public int IngredientId { get; set; }

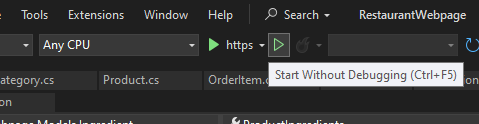
public string Name { get; set; }

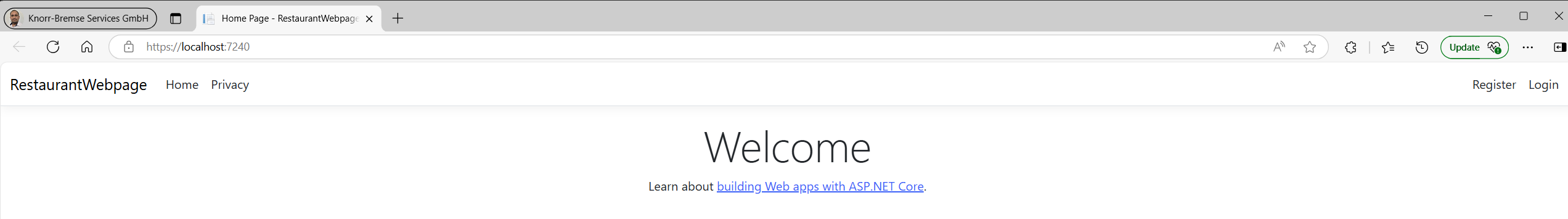
public ICollection<ProductIngredient> ProductIngredients { get; set; }

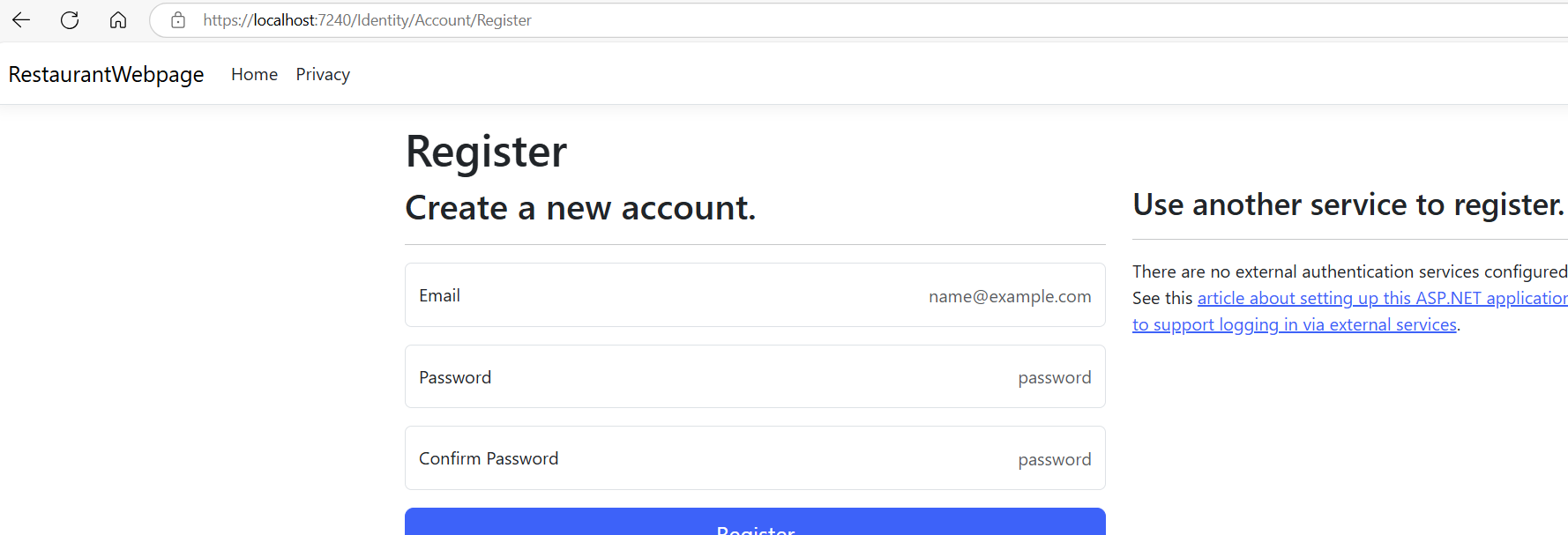
}

}

Execute the project:







Modify ApplicationDbContext.cs

Al poner ApplicationUser como tipo genérico de IdentityDbContext, le estás diciendo a Entity Framework que en lugar de usar la clase predeterminada IdentityUser, use tu clase personalizada ApplicationUser para la gestión de los usuarios. Esto te permite agregar propiedades adicionales a los usuarios, como Nombre, FechaDeNacimiento, Direccion, etc.

using Microsoft.AspNetCore.Identity.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore;

using RestaurantWebpage.Models;

namespace RestaurantWebpage.Data

{

public class ApplicationDbContext : IdentityDbContext<ApplicationUser>

{

public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)

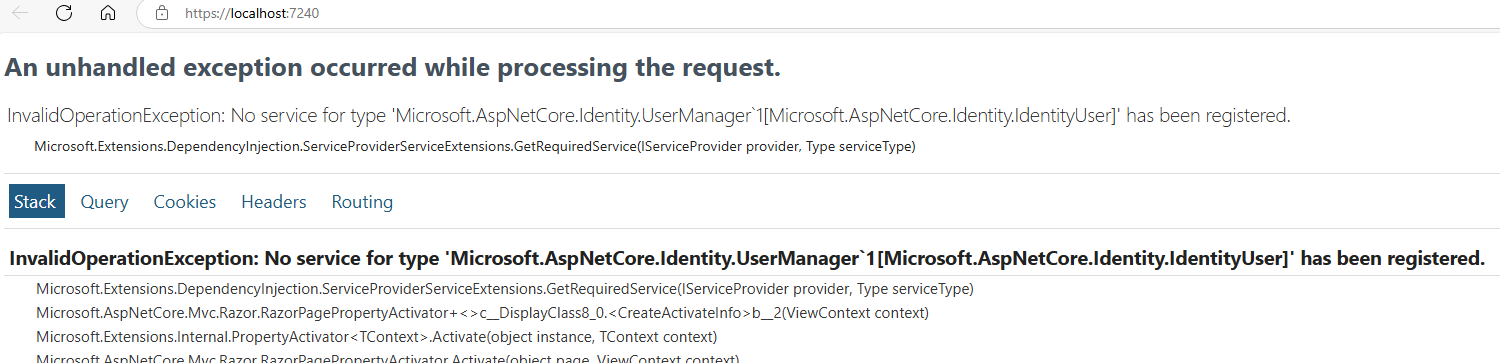
: base(options)

{

}

}

}

If run the app, you probably will see something like:  


So in Views/Shared/\_LoginPartial.cshtml replace IdentityUser by ApplicationUser:

@using Microsoft.AspNetCore.Identity

@inject SignInManager< ApplicationUser> SignInManager

@inject UserManager< ApplicationUser > UserManager

<ul class="navbar-nav">

@if (SignInManager.IsSignedIn(User))

{

<li class="nav-item">

<**a** class="nav-link text-dark" **asp-area**="Identity" **asp-page**="/Account/Manage/Index" title="Manage">Hello @User.Identity?.Name!</**a**>

</li>

<li class="nav-item">

<**form** class="form-inline" **asp-area**="Identity" **asp-page**="/Account/Logout" **asp-route-returnUrl**="@Url.Action("Index", "Home", new { area = "" })">

<button type="submit" class="nav-link btn btn-link text-dark">Logout</button>

</**form**>

</li>

}

else

{

<li class="nav-item">

<**a** class="nav-link text-dark" **asp-area**="Identity" **asp-page**="/Account/Register">Register</**a**>

</li>

<li class="nav-item">

<**a** class="nav-link text-dark" **asp-area**="Identity" **asp-page**="/Account/Login">Login</**a**>

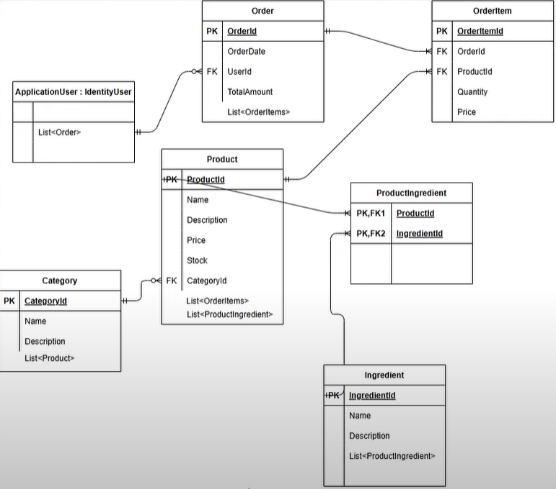
</li>

}

</ul>

## Add tables to the Db, add relationship and data.

In ApplicationDbContext.cs add tables for the database using DbSet, and define relationships for Product and Product ingredient, and finally add data to some tables:



using Microsoft.AspNetCore.Identity.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore;

using RestaurantWebpage.Models;

namespace RestaurantWebpage.Data

{

public class ApplicationDbContext : IdentityDbContext<ApplicationUser> // Use ApplicationUser to manage users.

{

public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)

: base(options)

{

}

public DbSet<Product> Products { get; set; }

public DbSet<Category> Categories { get; set; }

public DbSet<Order> Orders { get; set; }

public DbSet<OrderItem> OrderItems { get; set; }

public DbSet<Ingredient> Ingredients { get; set; }

public DbSet<ProductIngredient> ProductIngredients { get; set; }

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

base.OnModelCreating(modelBuilder);

// Define composite key and relationships for ProductIngredient

modelBuilder.Entity<ProductIngredient>()

.HasKey(pi => new { pi.ProductId, pi.IngredientId }); // Primary key of table ProductIngredient will be a combination of fields ProductId and IngredientId.

modelBuilder.Entity<ProductIngredient>()

.HasOne(pi => pi.Product)

.WithMany(p => p.ProductIngredients)

.HasForeignKey(pi => pi.ProductId);

modelBuilder.Entity<ProductIngredient>()

.HasOne(pi => pi.Ingredient)

.WithMany(i => i.ProductIngredients)

.HasForeignKey(pi => pi.IngredientId);

//Seed Data

modelBuilder.Entity<Category>().HasData(

new Category { CategoryId = 1, Name = "Appetizer" },

new Category { CategoryId = 2, Name = "Entree" },

new Category { CategoryId = 3, Name = "Side Dish" },

new Category { CategoryId = 4, Name = "Dessert" },

new Category { CategoryId = 5, Name = "Beverage" }

);

modelBuilder.Entity<Ingredient>().HasData(

//add mexican restaurant ingredients here

new Ingredient { IngredientId = 1, Name = "Beef" },

new Ingredient { IngredientId = 2, Name = "Chicken" },

new Ingredient { IngredientId = 3, Name = "Fish" },

new Ingredient { IngredientId = 4, Name = "Tortilla" },

new Ingredient { IngredientId = 5, Name = "Lettuce" },

new Ingredient { IngredientId = 6, Name = "Tomato" }

);

modelBuilder.Entity<Product>().HasData(

//Add mexican restaurant food entries here

new Product

{

ProductId = 1,

Name = "Beef Taco",

Description = "A delicious beef taco",

Price = 2.50m,

Stock = 100,

CategoryId = 2

},

new Product

{

ProductId = 2,

Name = "Chicken Taco",

Description = "A delicious chicken taco",

Price = 1.99m,

Stock = 101,

CategoryId = 2

},

new Product

{

ProductId = 3,

Name = "Fish Taco",

Description = "A delicious fish taco",

Price = 3.99m,

Stock = 90,

CategoryId = 2

}

);

modelBuilder.Entity<ProductIngredient>().HasData(

new ProductIngredient { ProductId = 1, IngredientId = 1 },

new ProductIngredient { ProductId = 1, IngredientId = 4 },

new ProductIngredient { ProductId = 1, IngredientId = 5 },

new ProductIngredient { ProductId = 1, IngredientId = 6 },

new ProductIngredient { ProductId = 2, IngredientId = 2 },

new ProductIngredient { ProductId = 2, IngredientId = 4 },

new ProductIngredient { ProductId = 2, IngredientId = 5 },

new ProductIngredient { ProductId = 2, IngredientId = 6 },

new ProductIngredient { ProductId = 3, IngredientId = 3 },

new ProductIngredient { ProductId = 3, IngredientId = 4 },

new ProductIngredient { ProductId = 3, IngredientId = 5 },

new ProductIngredient { ProductId = 3, IngredientId = 6 }

);

}

}

}

In Program.cs use ApplicationUser instead of IdentityUser:

using Microsoft.AspNetCore.Identity;

using Microsoft.EntityFrameworkCore;

using RestaurantWebpage.Data;

using RestaurantWebpage.Models;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

var connectionString = builder.Configuration.GetConnectionString("DefaultConnection") ?? throw new InvalidOperationException("Connection string 'DefaultConnection' not found.");

builder.Services.AddDbContext<ApplicationDbContext>(options =>

options.UseSqlServer(connectionString));

builder.Services.AddDatabaseDeveloperPageExceptionFilter();

builder.Services.AddDefaultIdentity<ApplicationUser>(options => options.SignIn.RequireConfirmedAccount = true)

.AddEntityFrameworkStores<ApplicationDbContext>();

builder.Services.AddControllersWithViews();

var app = builder.Build();

// Configure the HTTP request pipeline.

if (app.Environment.IsDevelopment())

{

app.UseMigrationsEndPoint();

}

else

{

app.UseExceptionHandler("/Home/Error");

// The default HSTS value is 30 days. You may want to change this for production scenarios, see https://aka.ms/aspnetcore-hsts.

app.UseHsts();

}

app.UseHttpsRedirection();

app.UseRouting();

app.UseAuthorization();

app.MapStaticAssets();

app.MapControllerRoute(

name: "default",

pattern: "{controller=Home}/{action=Index}/{id?}")

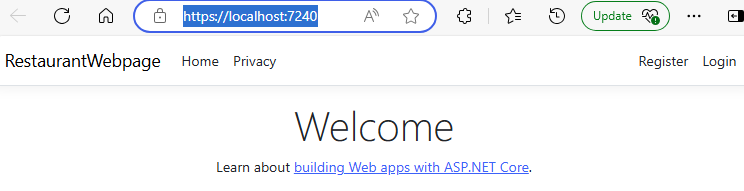
.WithStaticAssets();

app.MapRazorPages()

.WithStaticAssets();

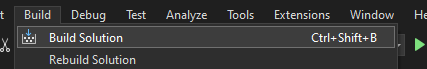
app.Run();

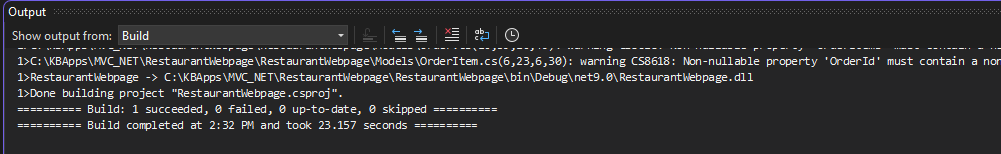
Execute the project, it should work correctly:



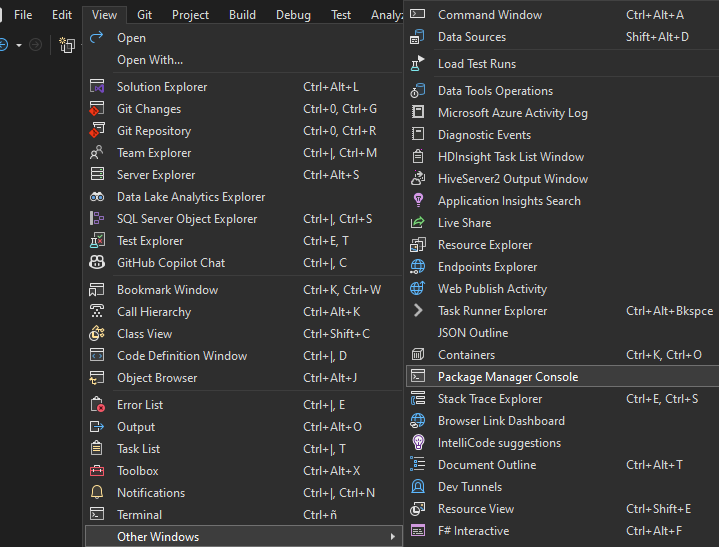
## Add migration

First, build to see if any error.

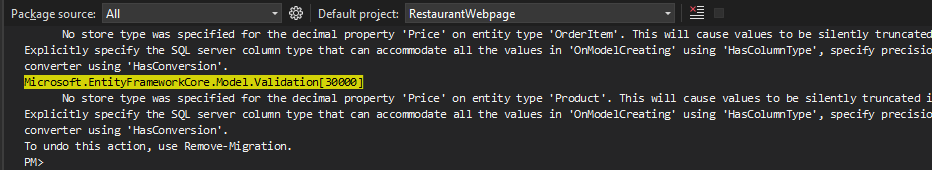




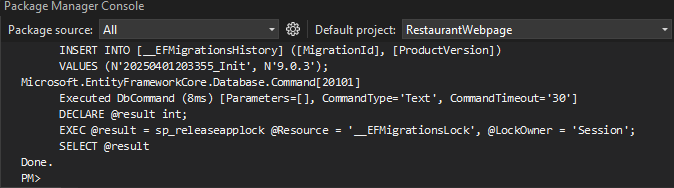
Then in the Package Manager Console run migrations to create the tables:

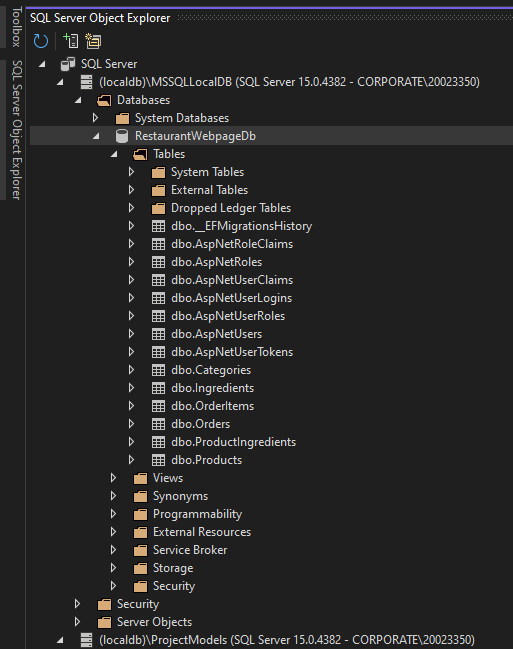
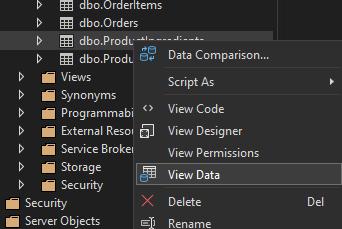
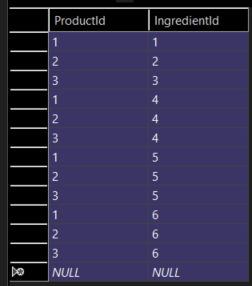


add-migration Init

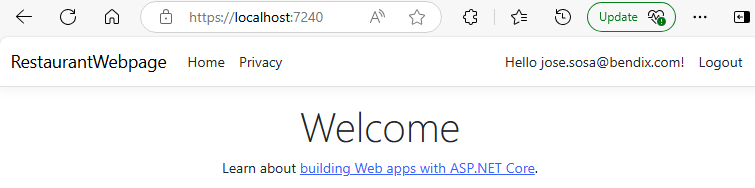


update-database



Create an account and login ([jose.sosa@bendix.com](mailto:jose.sosa@bendix.com) P@ssw0rd)



## Many to Many (Product and Ingredients)

Add in Ingredient.cs model [ValidateNever]

using Microsoft.AspNetCore.Mvc.ModelBinding.Validation;

namespace RestaurantWebpage.Models

{

public class Ingredient

{

public int IngredientId { get; set; }

public string Name { get; set; }

[ValidateNever]

public ICollection<ProductIngredient> ProductIngredients { get; set; }

}

}

In Product.cs model, add:

using System.ComponentModel.DataAnnotations.Schema;

using Microsoft.AspNetCore.Mvc.ModelBinding.Validation;

namespace RestaurantWebpage.Models

{

public class Product

{

public int ProductId { get; set; }

public string? Name { get; set; }

public string? Description { get; set; }

public decimal Price { get; set; }

public int Stock { get; set; }

public int CategoryId { get; set; }

[NotMapped] // This attribute tells Entity Framework Core not to map this property to the database. In other words, it will not be stored in the product table.

public IFormFile? ImageFile { get; set; } // IFormfile: This property represents a file uploaded from a form (for example, a product image uploaded by the user).

public string? ImageUrl { get; set; } = "https://via.placeholder.com/150"; // A default URL is set, which points to a placeholder image, in case a specific image is not uploaded.

[ValidateNever]

public Category? Category { get; set; } // A product belongs to a category

[ValidateNever]

public ICollection<OrderItem>? OrderItems { get; set; } // A product can be in many order items

[ValidateNever]

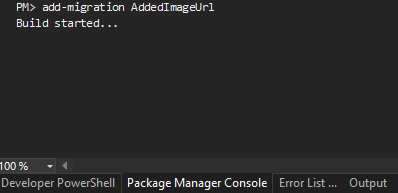
public ICollection<ProductIngredient>? ProductIngredients { get; set; } // A product can have many ingredients

}

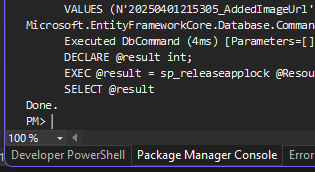
}

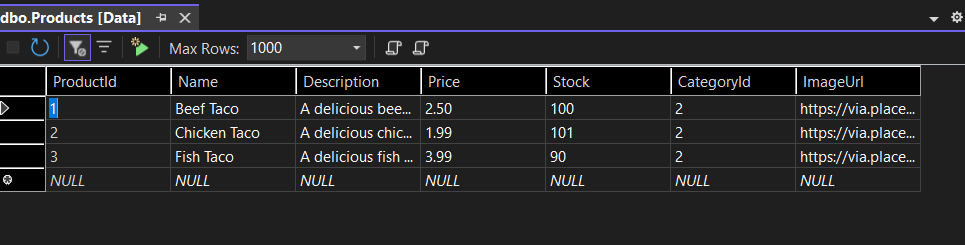
## Add migration for AddedImageUrl:

add-migration AddedImageUrl



update-database





## Create View using interface and Repository

Create folder Interfaces and there a new interface called IRepository.cs.

There you define the method signatures for the class that implements the interface. Esta interfaz IRepository<T> define las operaciones CRUD (Crear, Leer, Actualizar, Eliminar) para entidades de tipo T. Está diseñada para ser genérica, lo que significa que puedes usarla con cualquier tipo de entidad (como Product, Order, etc.). Todos los métodos son **asíncronos**, lo que permite trabajar con tareas que pueden tomar tiempo, como el acceso a la base de datos, sin bloquear el hilo principal.

using RestaurantWebpage.Models;

namespace RestaurantWebpage.Interfaces

{

public interface IRepository<T> where T : class

{

Task<IEnumerable<T>> GetAllAsync();

Task<T> GetByIdAsync(int id, QueryOptions<T> options);

Task AddAsync(T entity);

Task UpdateAsync(T entity);

Task DeleteAsync(int id);

}

}

Create in Models the QueryOptions.cs:

using System.Linq.Expressions;

namespace RestaurantWebpage.Models

{

public class QueryOptions<T> where T : class

{

public Expression<Func<T, Object>> OrderedBy { get; set; } = null!;

public Expression<Func<T, bool>> Where { get; set; } = null!;

private string[] includes = Array.Empty<string>();

public string Includes

{

set => includes = value.Replace(" ", "").Split(',');

}

public string[] GetIncludes() => includes;

public bool HasWhere => Where != null;

public bool HasOrderBy => OrderedBy != null;

}

}

Create the Repository folder with new item of type class: Repository.css

using Microsoft.EntityFrameworkCore;

using RestaurantWebpage.Data;

using RestaurantWebpage.Interfaces;

using RestaurantWebpage.Models;

namespace RestaurantWebpage.Repository

{

public class Repository<T> : IRepository<T> where T : class

{

protected ApplicationDbContext \_context { get; set; }

private DbSet<T> \_dbSet { get; set; }

public Repository(ApplicationDbContext context)

{

\_context = context;

\_dbSet = context.Set<T>();

}

public Task AddAsync(T entity)

{

throw new NotImplementedException();

}

public Task DeleteAsync(int entity)

{

throw new NotImplementedException();

}

public async Task<IEnumerable<T>> GetAllAsync()

{

return await \_dbSet.ToListAsync();

}

public Task<T> GetByIdAsync(int id, QueryOptions<T> options)

{

throw new NotImplementedException();

}

public Task UpdateAsync(T entity)

{

throw new NotImplementedException();

}

Task<T> IRepository<T>.AddAsync(T entity)

{

throw new NotImplementedException();

}

Task<T> IRepository<T>.UpdateAsync(T entity)

{

throw new NotImplementedException();

}

public Task<T> DeleteAsync(int id)

{

throw new NotImplementedException();

}

}

}

Add in Models IngredientController.cs:

using Microsoft.AspNetCore.Mvc;

using RestaurantWebpage.Data;

using RestaurantWebpage.Models;

using RestaurantWebpage.Repository;

namespace RestaurantWebpage.Controllers

{

public class IngredientController : Controller

{

private Repository<Ingredient> ingredients;

public IngredientController(ApplicationDbContext context)

{

ingredients = new Repository<Ingredient>(context);

}

public async Task<IActionResult> Index()

{

return View(await ingredients.GetAllAsync());

}

}

}

For testing update Program.cs with:

using Microsoft.AspNetCore.Identity;

using Microsoft.EntityFrameworkCore;

using RestaurantWebpage.Data;

using RestaurantWebpage.Models;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

var connectionString = builder.Configuration.GetConnectionString("DefaultConnection") ?? throw new InvalidOperationException("Connection string 'DefaultConnection' not found.");

builder.Services.AddDbContext<ApplicationDbContext>(options =>

options.UseSqlServer(connectionString));

builder.Services.AddDatabaseDeveloperPageExceptionFilter();

builder.Services.AddDefaultIdentity<ApplicationUser>(options => options.SignIn.RequireConfirmedAccount = true)

.AddEntityFrameworkStores<ApplicationDbContext>();

builder.Services.AddControllersWithViews();

var app = builder.Build();

// Configure the HTTP request pipeline.

if (app.Environment.IsDevelopment())

{

app.UseMigrationsEndPoint();

}

else

{

app.UseExceptionHandler("/Home/Error");

// The default HSTS value is 30 days. You may want to change this for production scenarios, see https://aka.ms/aspnetcore-hsts.

app.UseHsts();

}

app.UseHttpsRedirection();

app.UseRouting();

app.UseAuthorization();

app.MapStaticAssets();

app.MapControllerRoute(

name: "default",

pattern: "{controller=Ingredient}/{action=Index}/{id?}")

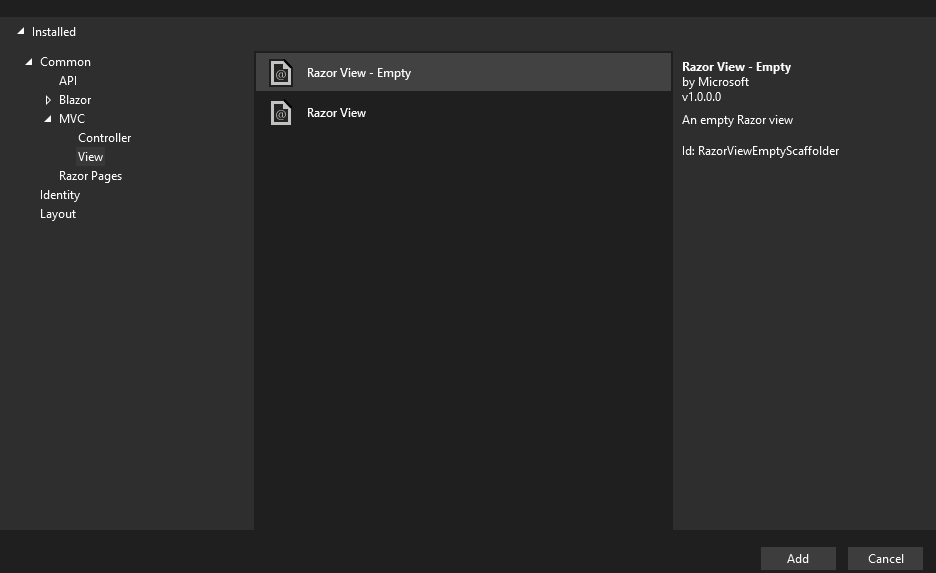
.WithStaticAssets();

app.MapRazorPages()

.WithStaticAssets();

app.Run();

Add in Views the Ingredient folder and there the Index.cshtml:



@model IEnumerable<Ingredient>

@{

ViewBag.Title = "List of Restaurant Ingredients";

}

<h1>Ingredients</h1>

<table class="table">

<thead>

<tr>

<th>

@Html.DisplayNameFor(model => model.Name)

</th>

<th></th>

</tr>

</thead>

<tbody>

@foreach (Ingredient i in Model)

{

<tr>

<td>

@Html.DisplayFor(modelItem => i.Name)

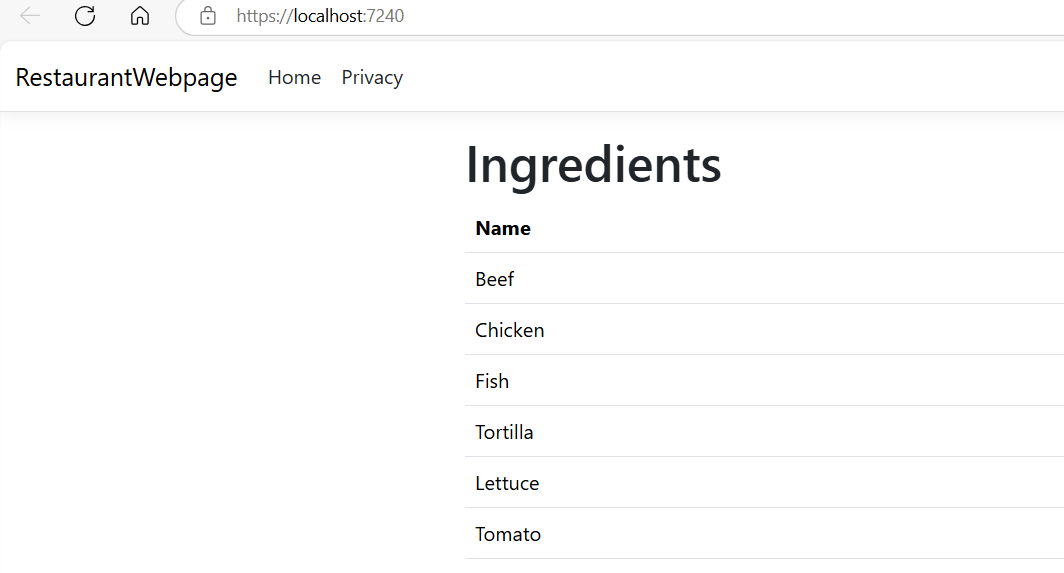
</td>

</tr>

}

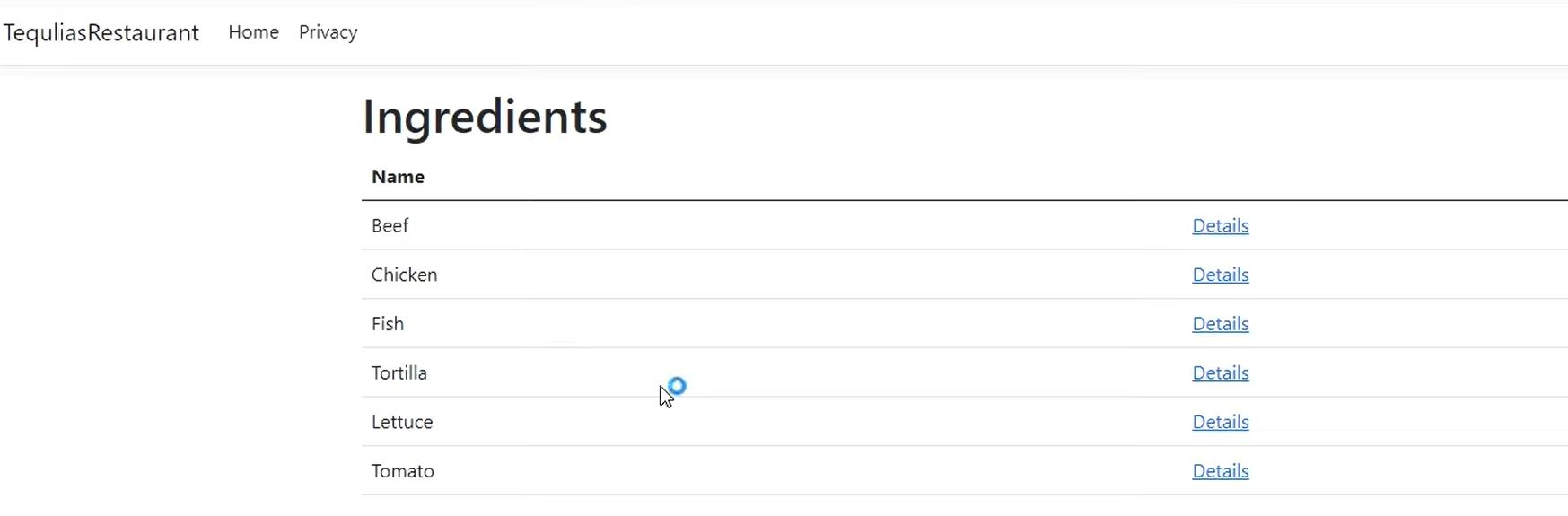
</tbody>

</table>



## Add Details to the ingredients view

If we want to see the details of an ingredient:



Add into Views/Ingredient/Index.cshtml:  
@model IEnumerable<Ingredient>

@{

ViewBag.Title = "List of Restaurant Ingredients";

}

<h1>Ingredients</h1>

<table class="table">

<thead>

<tr>

<th>

@Html.DisplayNameFor(model => model.Name)

</th>

<th></th>

</tr>

</thead>

<tbody>

@foreach (Ingredient i in Model)

{

<tr>

<td>

@Html.DisplayFor(modelItem => i.Name)

</td>

<td>

<**a** **asp-action**="Details" **asp-route-id**="@i.IngredientId">Details</**a**>

</td>

</tr>

}

</tbody>

</table>

Go to IngredientController.cs and add the details of the product that is associated:  
using Microsoft.AspNetCore.Mvc;

using RestaurantWebpage.Data;

using RestaurantWebpage.Models;

using RestaurantWebpage.Repository;

namespace RestaurantWebpage.Controllers

{

public class IngredientController : Controller

{

private Repository<Ingredient> ingredients;

public IngredientController(ApplicationDbContext context)

{

ingredients = new Repository<Ingredient>(context);

}

public async Task<IActionResult> Index()

{

return View(await ingredients.GetAllAsync());

}

public async Task<IActionResult> Details(int id)

{

return View(await ingredients.GetByIdAsync(id, new QueryOptions<Ingredient>() {Includes= "ProductIngredients.Product" }));

}

}

}

In Views/Ingredient create the view Details.cs:

@model Ingredient

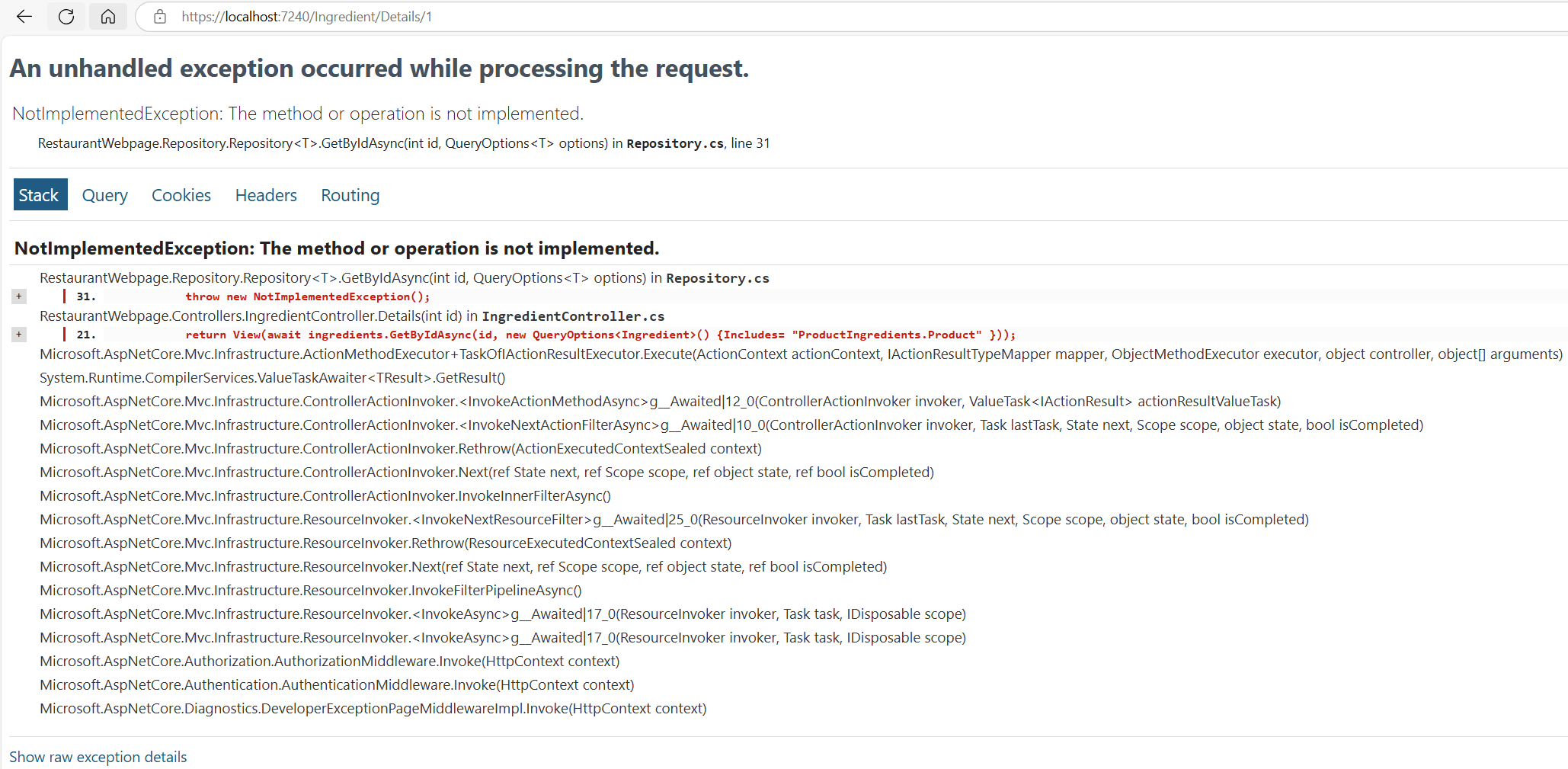
@{

ViewBag.Title = "Ingredient Details";

}

<h1>@Model.Name Details</h1>

Then, execute the project, and select the Details of an ingredient:



Add in Repository.cs:

using Microsoft.EntityFrameworkCore;

using RestaurantWebpage.Data;

using RestaurantWebpage.Interfaces;

using RestaurantWebpage.Models;

namespace RestaurantWebpage.Repository

{

public class Repository<T> : IRepository<T> where T : class

{

protected ApplicationDbContext \_context { get; set; }

private DbSet<T> \_dbSet { get; set; }

public Repository(ApplicationDbContext context)

{

\_context = context;

\_dbSet = context.Set<T>();

}

public Task AddAsync(T entity)

{

throw new NotImplementedException();

}

public Task DeleteAsync(int entity)

{

throw new NotImplementedException();

}

public async Task<IEnumerable<T>> GetAllAsync()

{

return await \_dbSet.ToListAsync();

}

public async Task<T> GetByIdAsync(int id, QueryOptions<T> options)

{

IQueryable<T> query = \_dbSet; // IQueryable<T>: Es una interfaz que representa una consulta de datos que no se ejecuta hasta que se evalúa. \_dbSet: Es una propiedad que representa la colección de datos de tipo T en la base de datos.

if (options.HasWhere)

{

query = query.Where(options.Where);

}

if (options.HasOrderBy)

{

query = query.OrderBy(options.OrderedBy);

}

foreach (string include in options.GetIncludes())

{

query = query.Include(include);

}

var key = \_context.Model.FindEntityType(typeof(T)).FindPrimaryKey().Properties.FirstOrDefault(); // Gives the primary key of the entity.

string primaryKeyName = key?.Name; // GEt key name of the primary key, ie. IngredientId

return await query.FirstOrDefaultAsync(e => EF.Property<int>(e, primaryKeyName) == id);

}

public Task UpdateAsync(T entity)

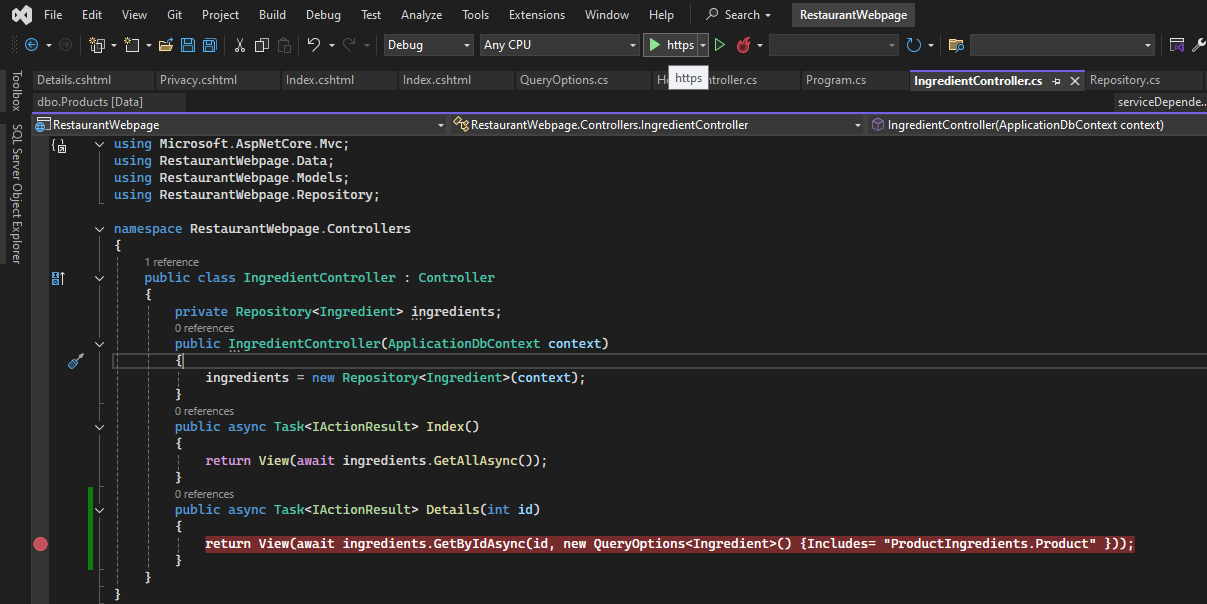
{

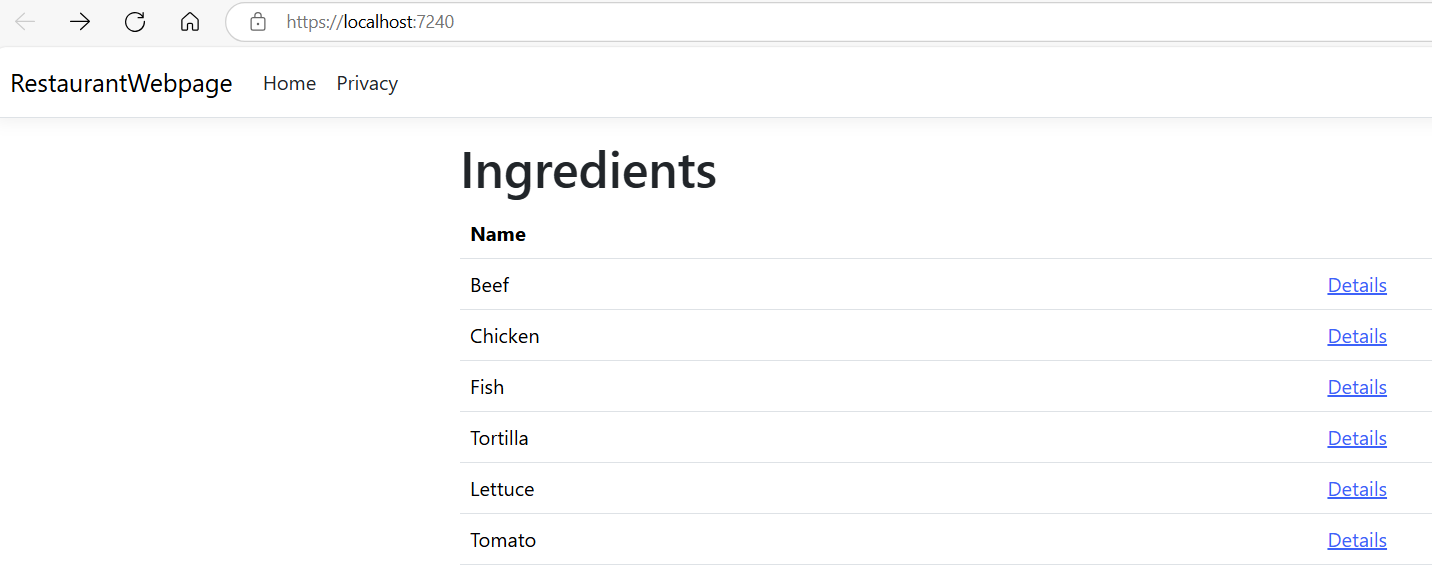
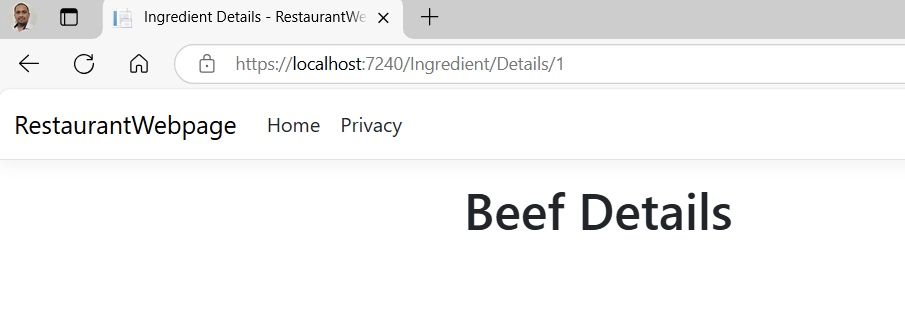
throw new NotImplementedException();

}

}

}

In IngredientController.cs add a breakpoint and execute:  


We want to know what Products are associated with the Ingredient.

In Details.cshtml:

@model Ingredient

@{

ViewBag.Title = "Ingredient Details";

}

<h1>@Model.Name Details</h1>

@if(Model.ProductIngredients != null && Model.ProductIngredients.Any())

{

<h4>Products</h4>

<table class="table">

<thead>

<tr>

<th>Product Name</th>

<th>Description</th>

<th>Price</th>

</tr>

</thead>

<tbody>

@foreach (var productIngredient in Model.ProductIngredients)

{

<tr>

<td>@productIngredient.Product.Name</td>

<td>@productIngredient.Product.Description</td>

<td>@productIngredient.Product.Price</td>

</tr>

}

</tbody>

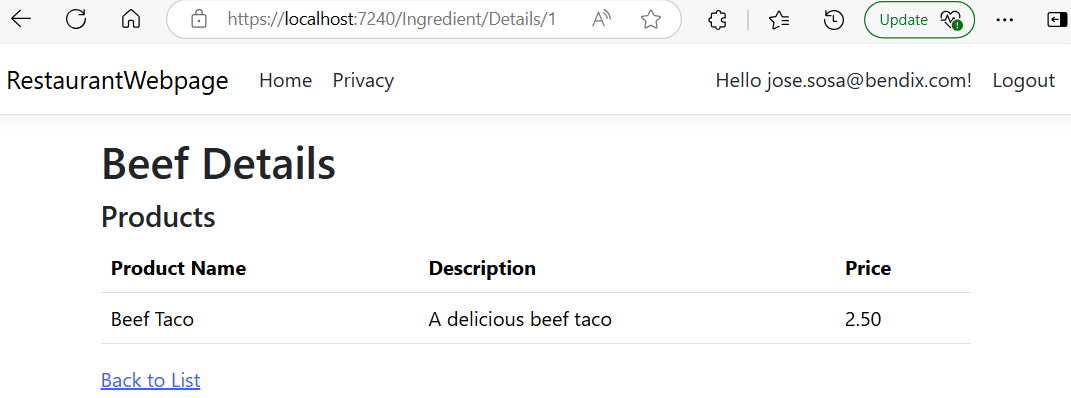
</table>

}

<div>

<**a** **asp-action**="Index">Back to List</**a**>

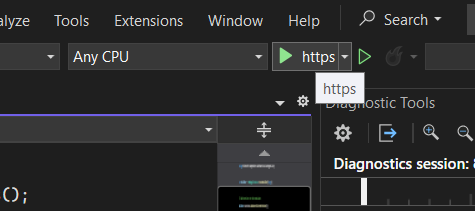
</div>

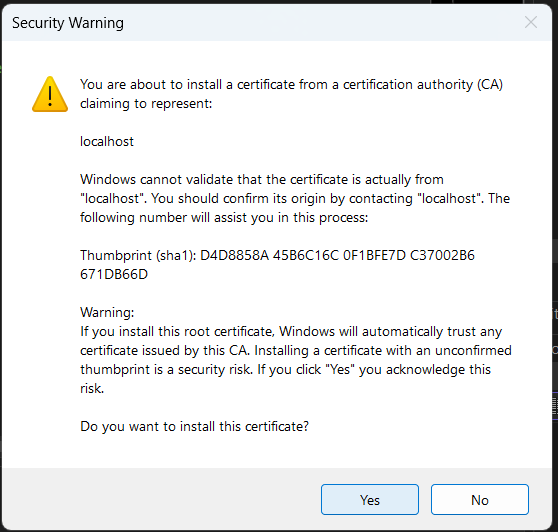
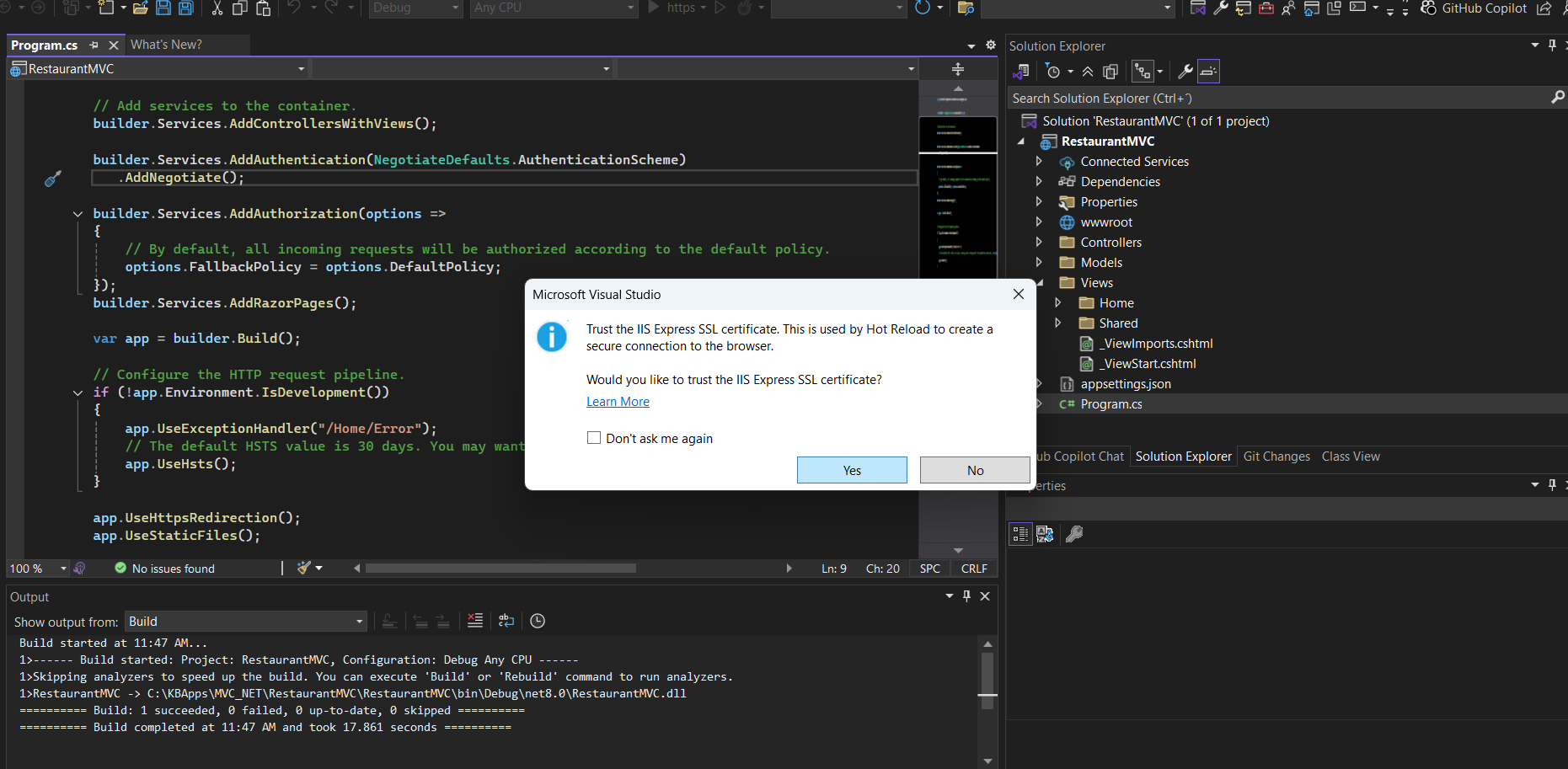


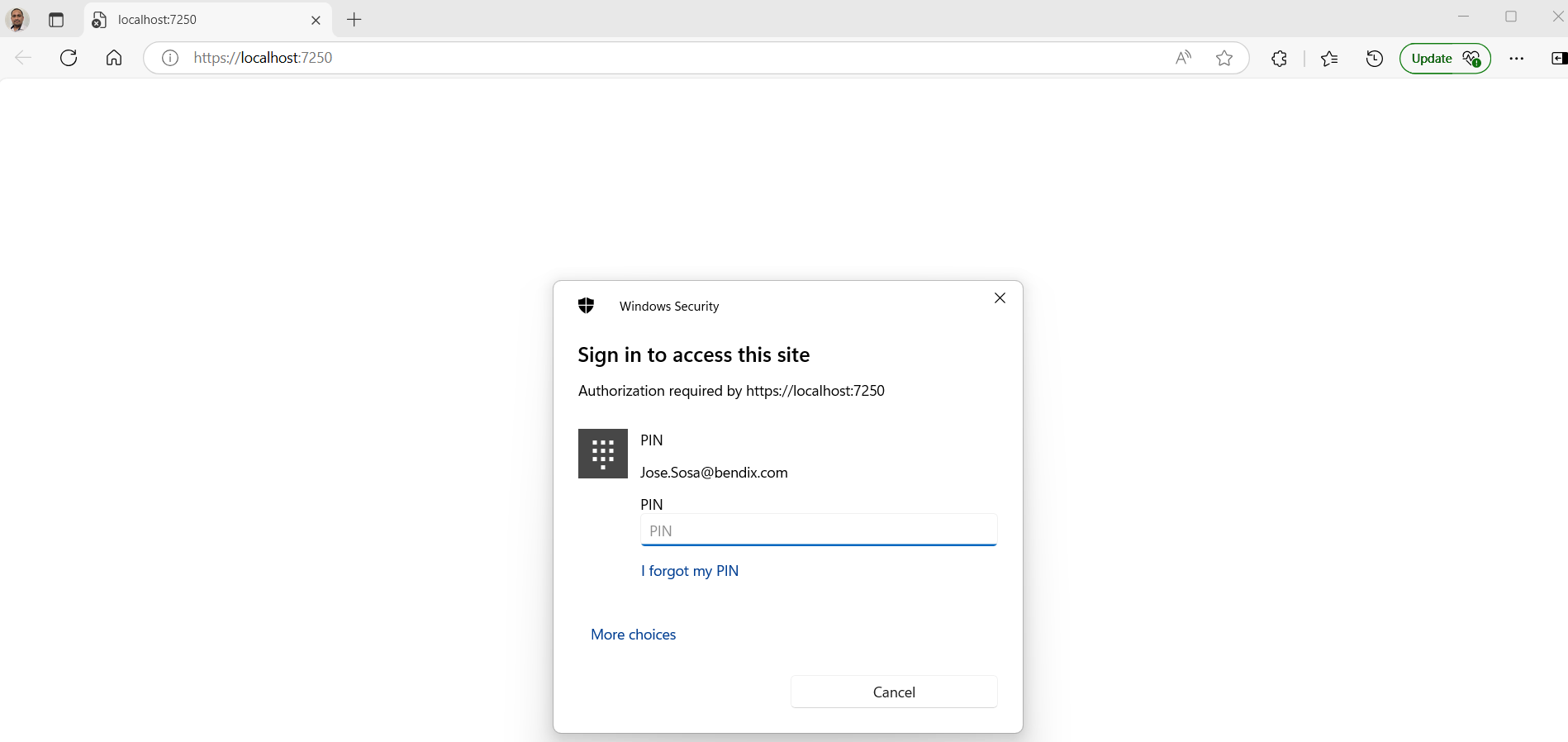
tengo: divs\_bom\_docs\_locator = self.get\_element\_locator("divs\_bom\_docs") divs\_bom\_docs = self.driver.find\_elements(\*divs\_bom\_docs\_locator) if divs\_bom\_docs: for element in reverse(divs\_bom\_docs): aria\_level = element.get\_attribute("aria-level") is\_expanded = element.get\_attribute("aria-expanded") si existe el atributo aria-expanded, si no de None o algo ais

With windows identity, based on: [.NET 7 WEB API WINDOWS AUTHENTICATION GET CURRENT WINDOWS USER.](https://www.youtube.com/watch?v=MtLu8_U2Kcg)

To run the project:







asegurarte de que el navegador está configurado correctamente para enviar las credenciales de Windows automáticamente,

**Para Edge**:

1. Abre Microsoft Edge.
2. En la barra de direcciones, ve a edge://flags/.
3. Busca la opción Enable Integrated Authentication in Windows y habilítala.
4. Reinicia Edge.

**Asegúrate de que el navegador confíe en tu sitio**

* Verifica que el navegador esté configurado para permitir la autenticación integrada sin pedir credenciales repetidas. Si no se configura correctamente, los navegadores pueden pedir las credenciales en cada intento.
* Si estás utilizando **localhost** como URL de desarrollo, asegúrate de que **localhost** esté agregado a las configuraciones del navegador para permitir la autenticación automática.

Close the browser.

In the Controllers folder, right click and select Add --> Controller:



