

Building a Modern Web Application with ASP.NET Core, SQL Server, and Azure

Carlos Felipe Avilán Payares

Laura Katherine Aviles Reina

Juan Diego Pinzón Fontecha

Software Engineering

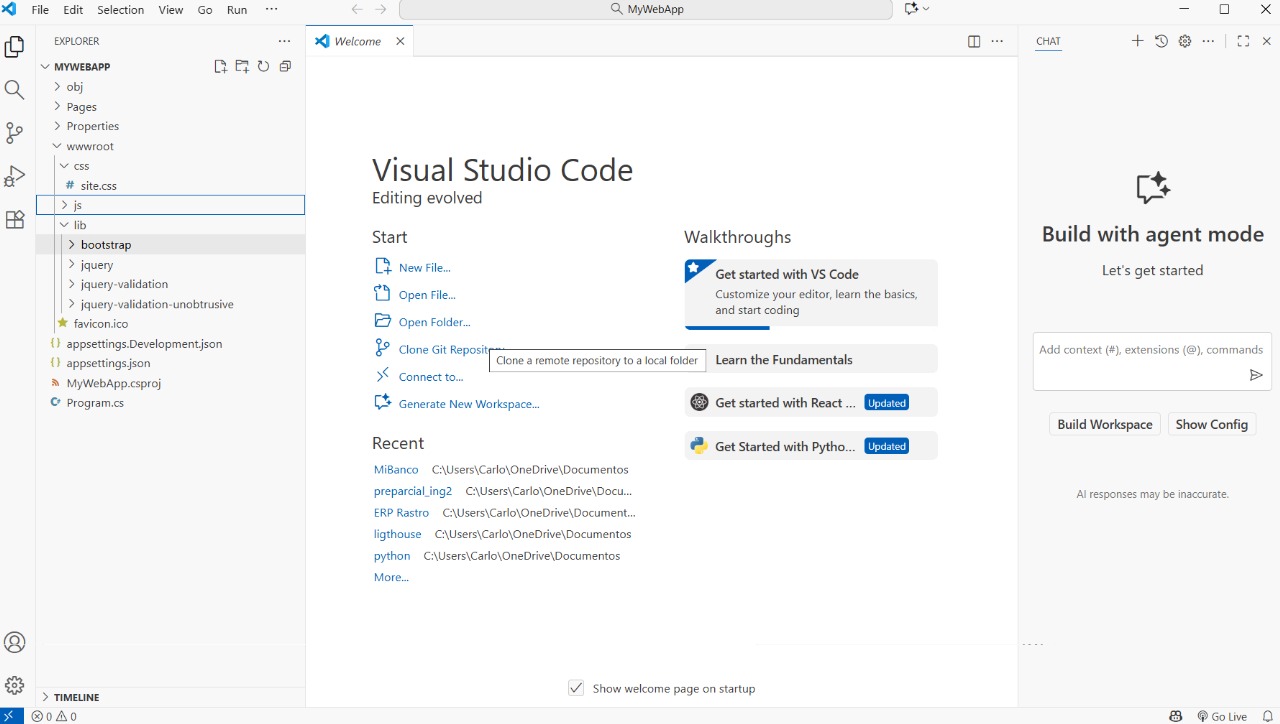
Universidad Manuela Beltrán

Web Engineering II

Carlos Eduardo Mujica Reyes

2025

1. **Part 1: Project Setup and Foundational Concepts**



This image shows the Visual Studio Code editor open to a new project named MyWebApp.

Left Sidebar (Explorer): The file structure is visible, which includes common elements of an ASP.NET Core web application, such as the wwwroot folder for static files (with css and js directories), Pages (for Razor Pages), appsettings.json (for configuration), and the project file (MyWebApp.csproj).

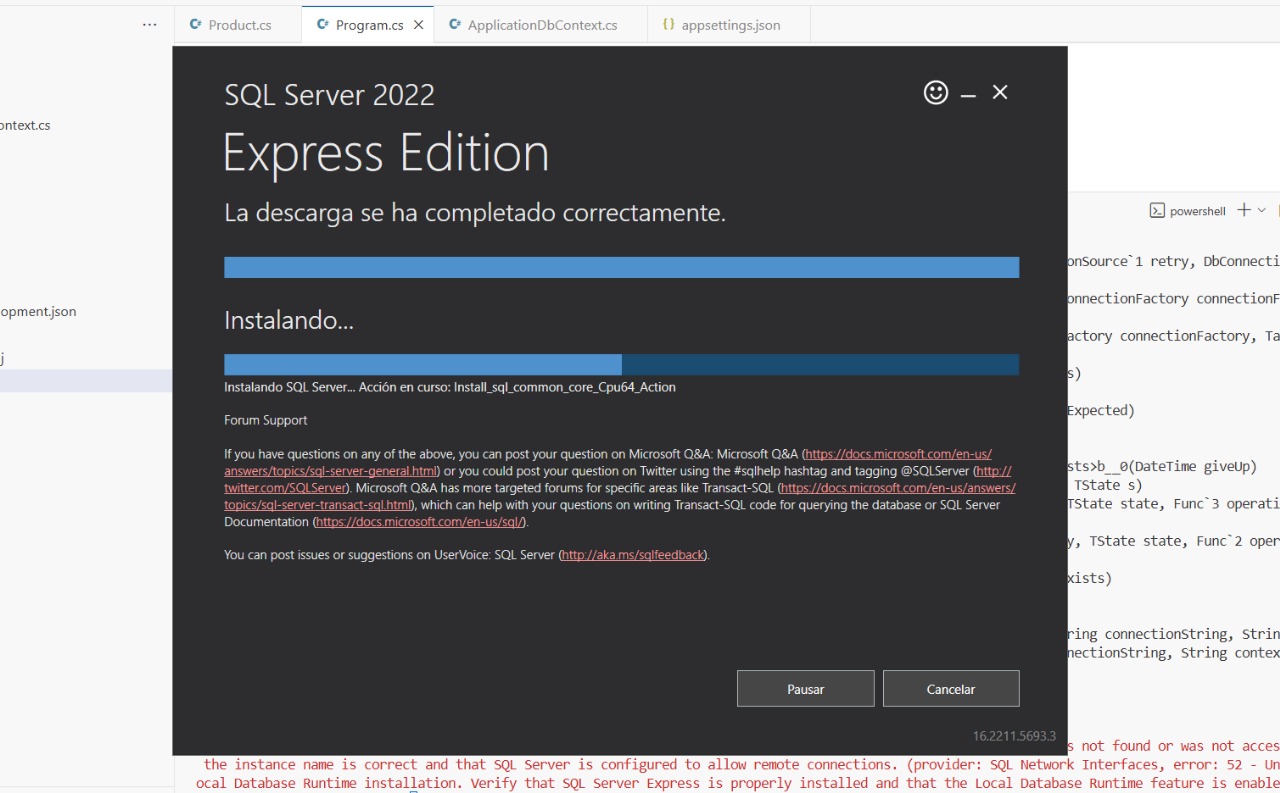
Main Window: The editor is displaying the "Welcome" tab with options to "Start" a new file or open a folder/project. This indicates the developer has just opened or created the basic project scaffold and is beginning their work.

Foundational Concepts: The structure visible (wwwroot, Pages, Program.cs) is the fundamental, default setup for a modern ASP.NET Core web application.



The SDK (Software Development Kit) is a prerequisite for developing, building, and running ASP.NET Core applications.

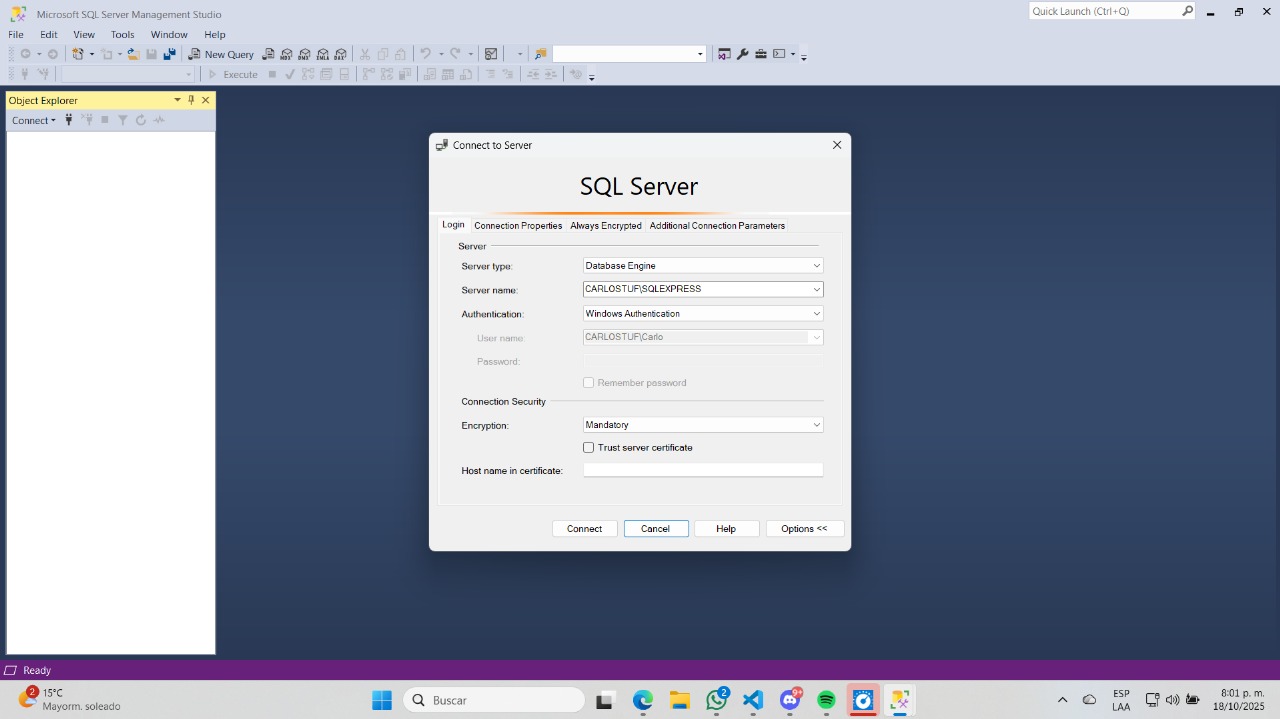
This is a fundamental step in Project Setup—getting the necessary tools installed before writing any code. The page also provides a direct download link and a SHA512 checksum for verifying the integrity of the downloaded file.



The image shows the installation in progress for SQL Server 2022 Express Edition.

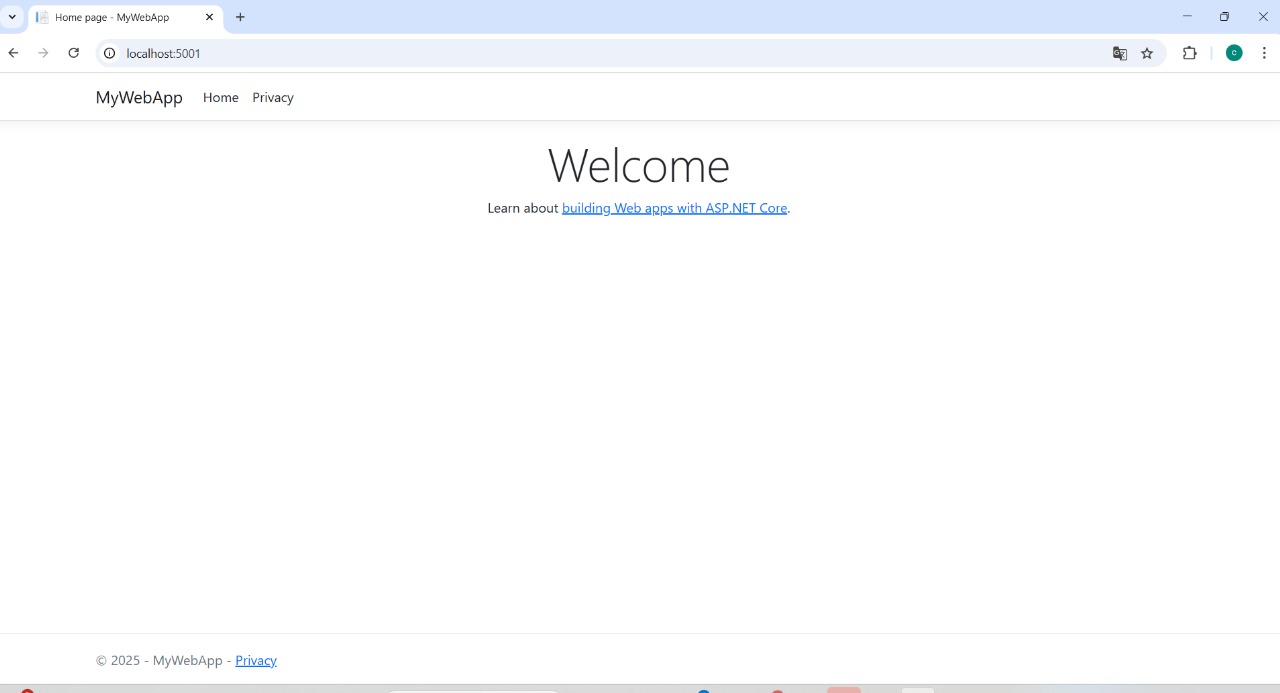
While this tool is for the data layer, its installation is a crucial step in the overall Project Setup—it provides the necessary database engine (mssqllocaldb) referenced by the connection string shown in the previous image.

This is a fundamental infrastructure step that must be completed before the application can successfully connect and manage its data using Entity Framework Core.



The image shows a connection window in Microsoft SQL Server Management Studio (SSMS). The developer is attempting to connect to the SQL Server instance, specifically a local instance named CARLOSTUF\SQLEXPRESS.

SSMS is a graphical tool used to manage the SQL Server database engine. While not strictly part of the ASP.NET Core code, setting up and viewing the database structure is essential for confirming the Entity Framework Core data layer setup (Migrations) is working correctly. This is an infrastructure step related to the foundational tools needed for the project.

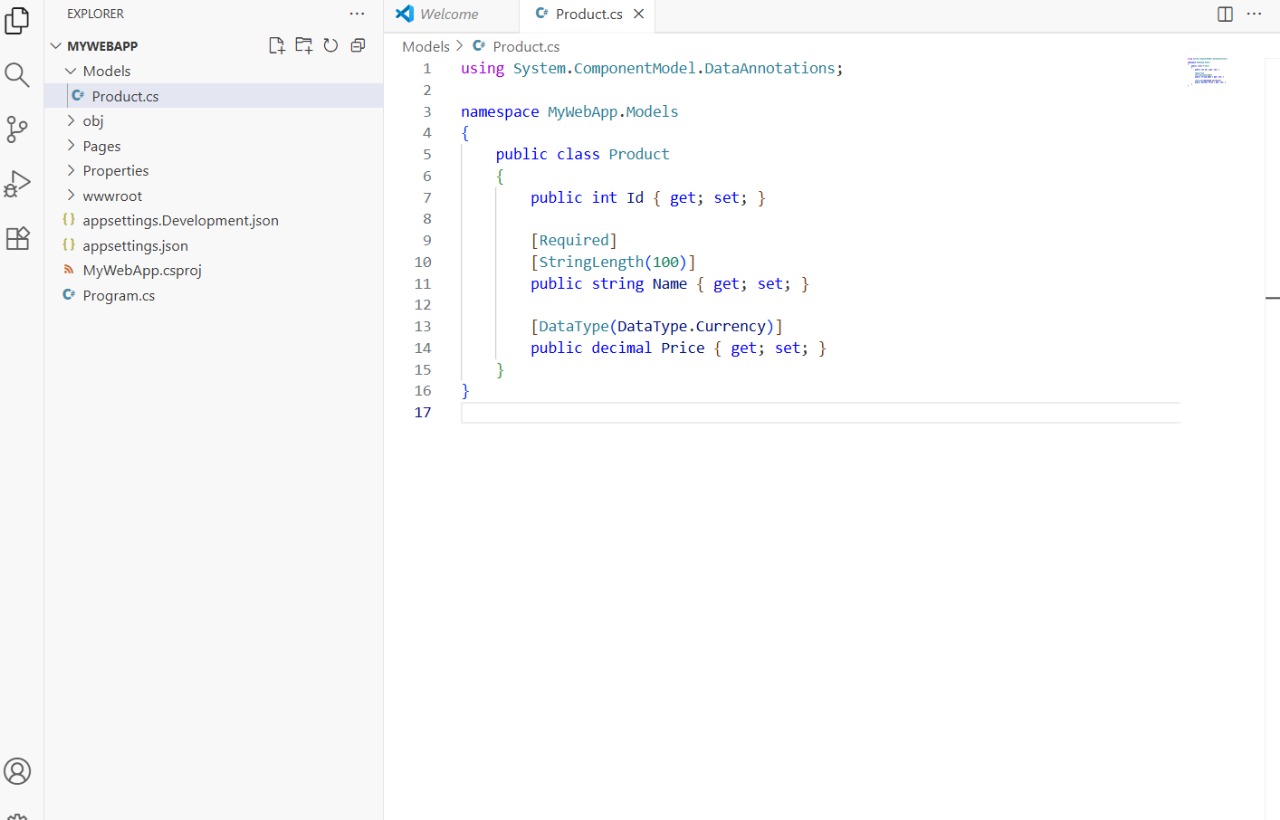


The image shows the default welcome page of the MyWebApp running in a web browser at http://localhost:5001.

This confirms that the initial Project Setup was successful, the application has compiled, and the basic ASP.NET Core request pipeline is functioning.

The content is the default template, indicating the developer is ready to start navigating to the pages that implement the CRUD Operations (like /Products or /Products/Index) or continue building the application's front-end structure.

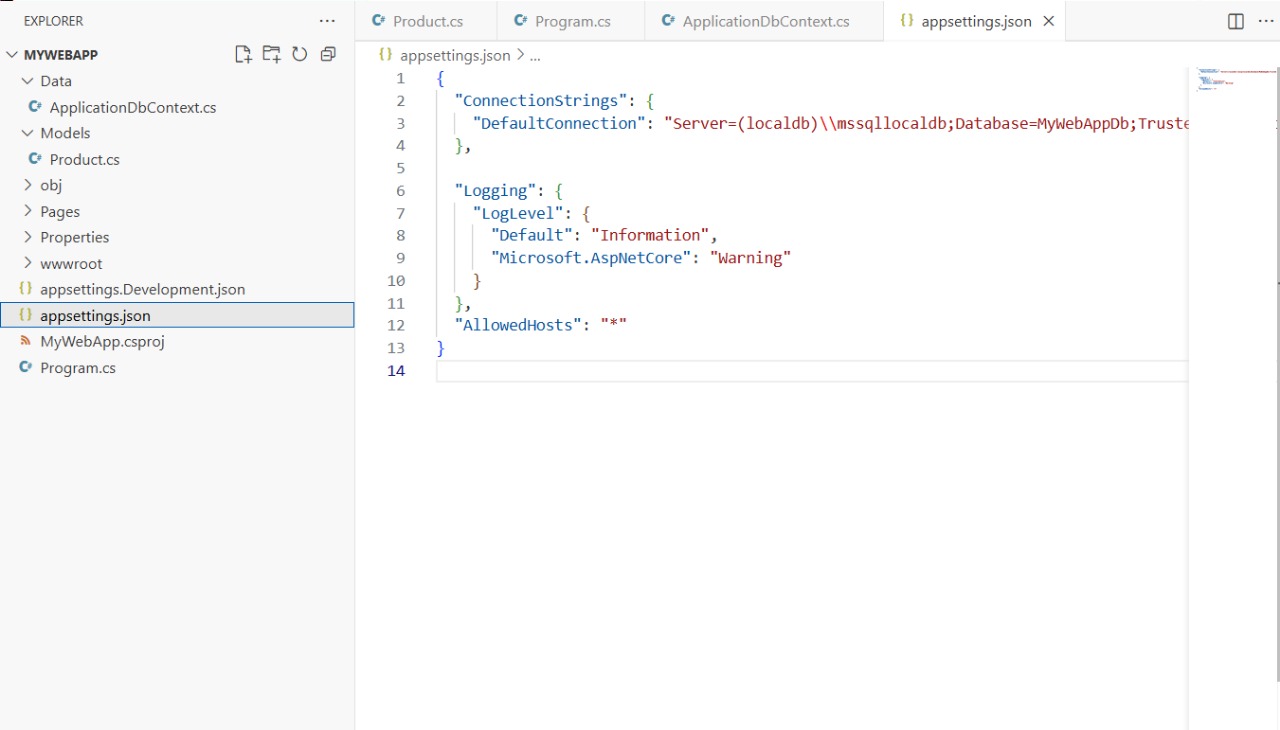
1. **Part 2: Building the Data Layer with Entity Framework Core**



The image shows the creation of a C# Model Class named Product.cs inside a Models folder. This class defines the structure of data (an entity) that will be stored in the database.

It has properties for Id, Name, and Price.

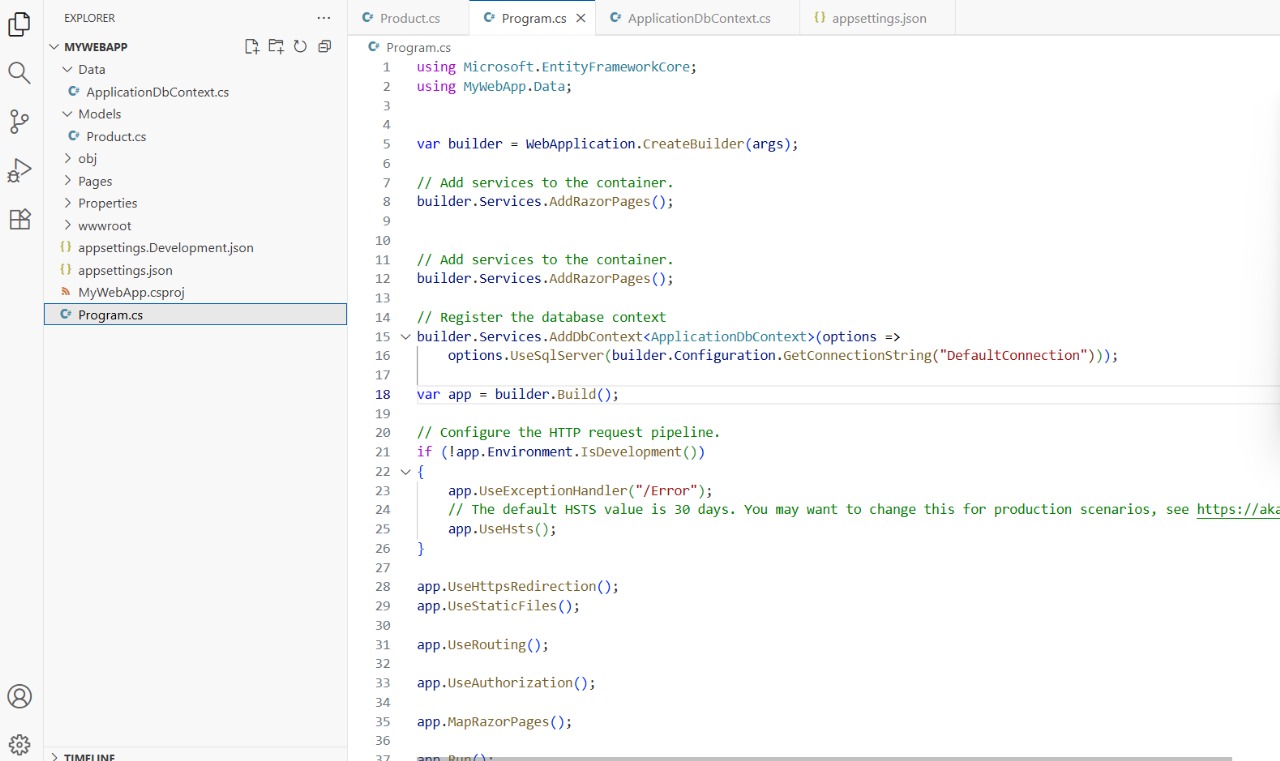
Data Annotations (like [Required], [StringLength(100)], and [DataType(DataType.Currency)]) are being used. These annotations tell Entity Framework Core how to configure the corresponding database column and also aid in front-end validation. This step is essential for Code-First development using Entity Framework Core, where the C# model is used to generate the database schema.



This image shows the appsettings.json file, which is used for application configuration. Specifically, it displays the ConnectionStrings section being configured.

The crucial part is the DefaultConnection entry. This string provides the necessary details (server address, database name: MyWebAppDb) for Entity Framework Core to connect to the database, in this case, a LocalDB instance of SQL Server.

A file named ApplicationDbContext.cs is also visible on the left, which confirms that the DbContext class (the session with the database) is being set up in conjunction with this connection string.



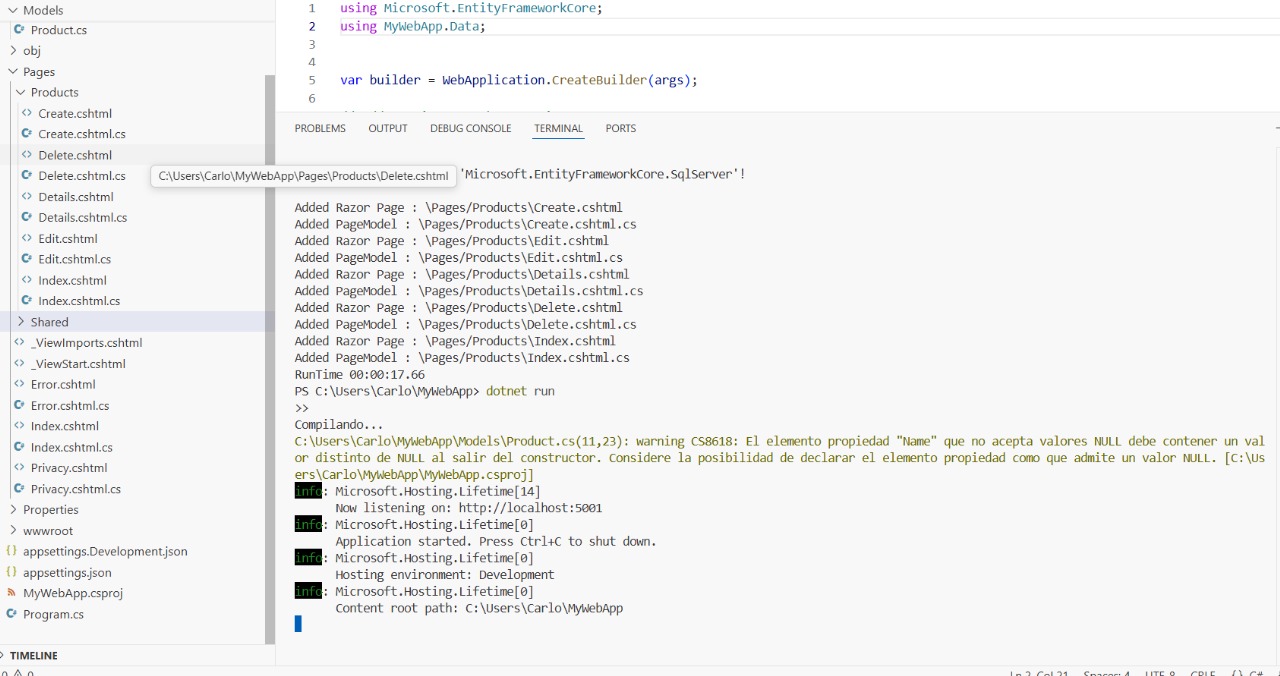
This image shows the Program.cs file, which configures the application's services and middleware. The key action here is registering the database context with the dependency injection container.

On line 15-16, builder.Services.AddDbContext<ApplicationDbContext>(...) is called. This tells the application to create and manage instances of the ApplicationDbContext (the EF Core session) using the connection string defined earlier.

.UseSqlServer() specifies that SQL Server will be used as the database provider.

This code completes the foundational setup for Entity Framework Core, enabling the application to interact with the data layer.

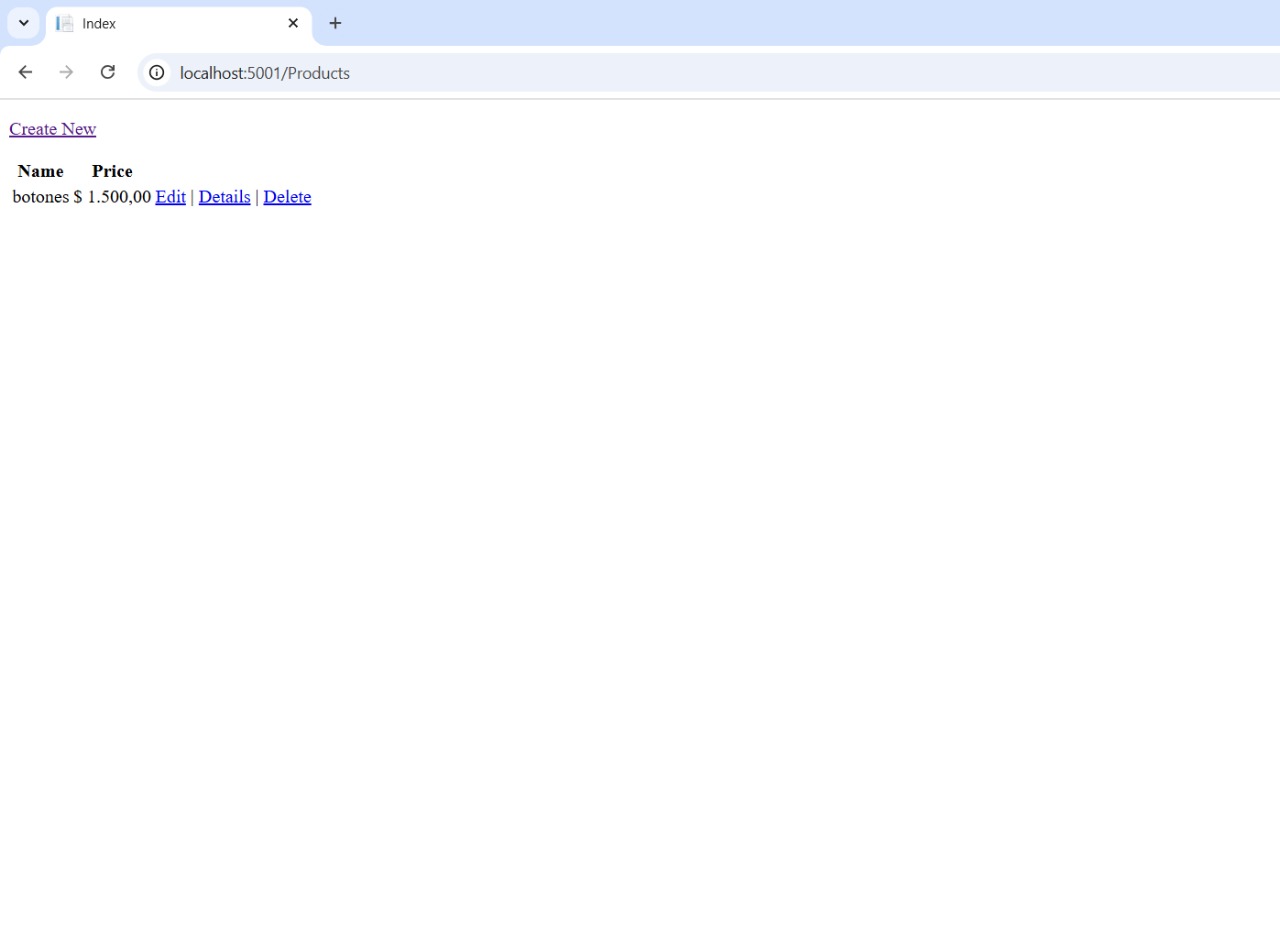
1. **Part 3: Implementing CRUD Operations**



This image shows the output of running the ASP.NET Core application after using a scaffolding tool. The key output is the list of Added Razor Page entries, including: Create.cshtml, Edit.cshtml, Details.cshtml, Delete.cshtml, and Index.cshtml.

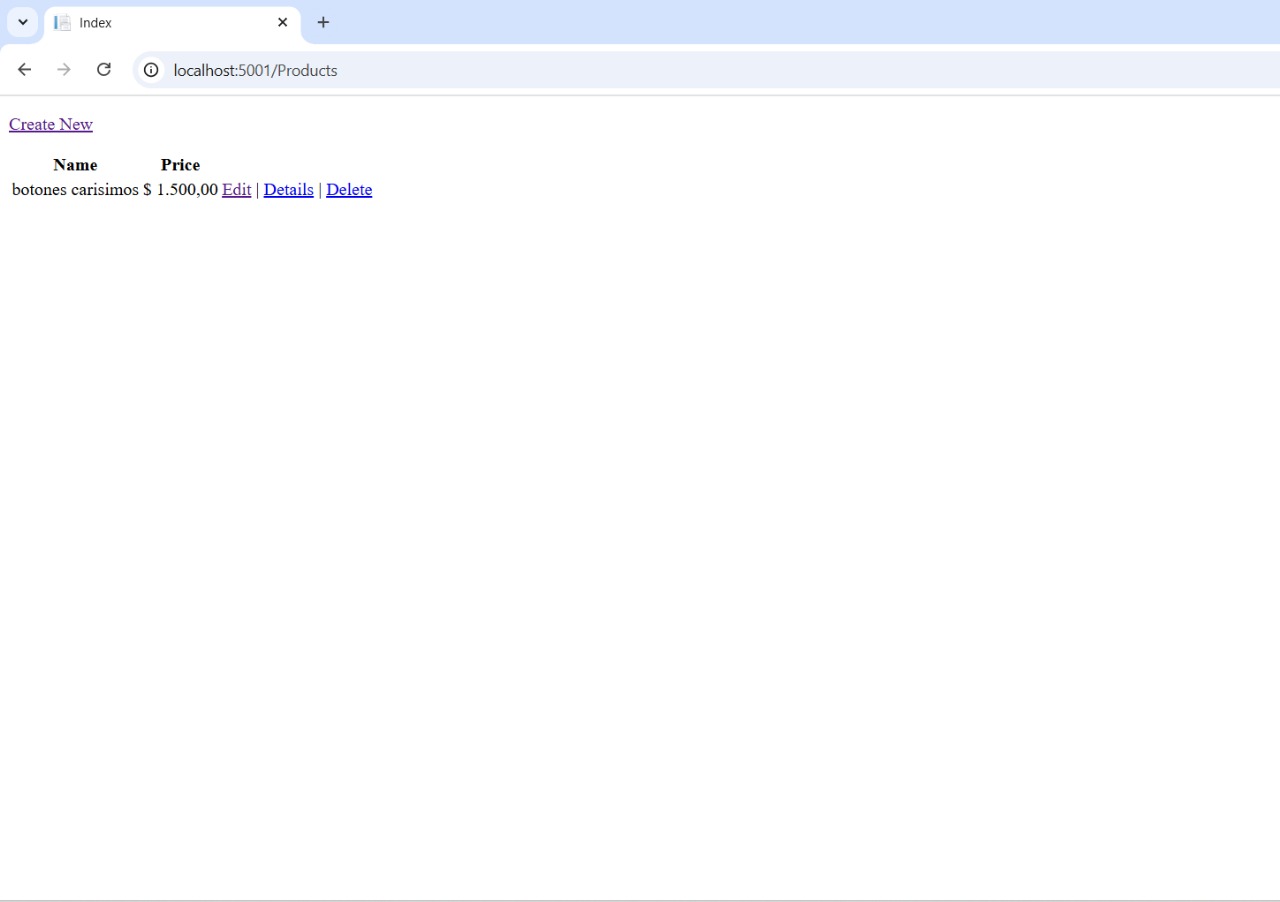
These pages, generated under the Pages/Products folder, correspond directly to the standard CRUD (Create, Read, Update, Delete) operations for the Product entity.

The console also shows the application has started and is listening on http://localhost:5001, meaning the developer is now ready to test the implemented CRUD functionality in a browser.

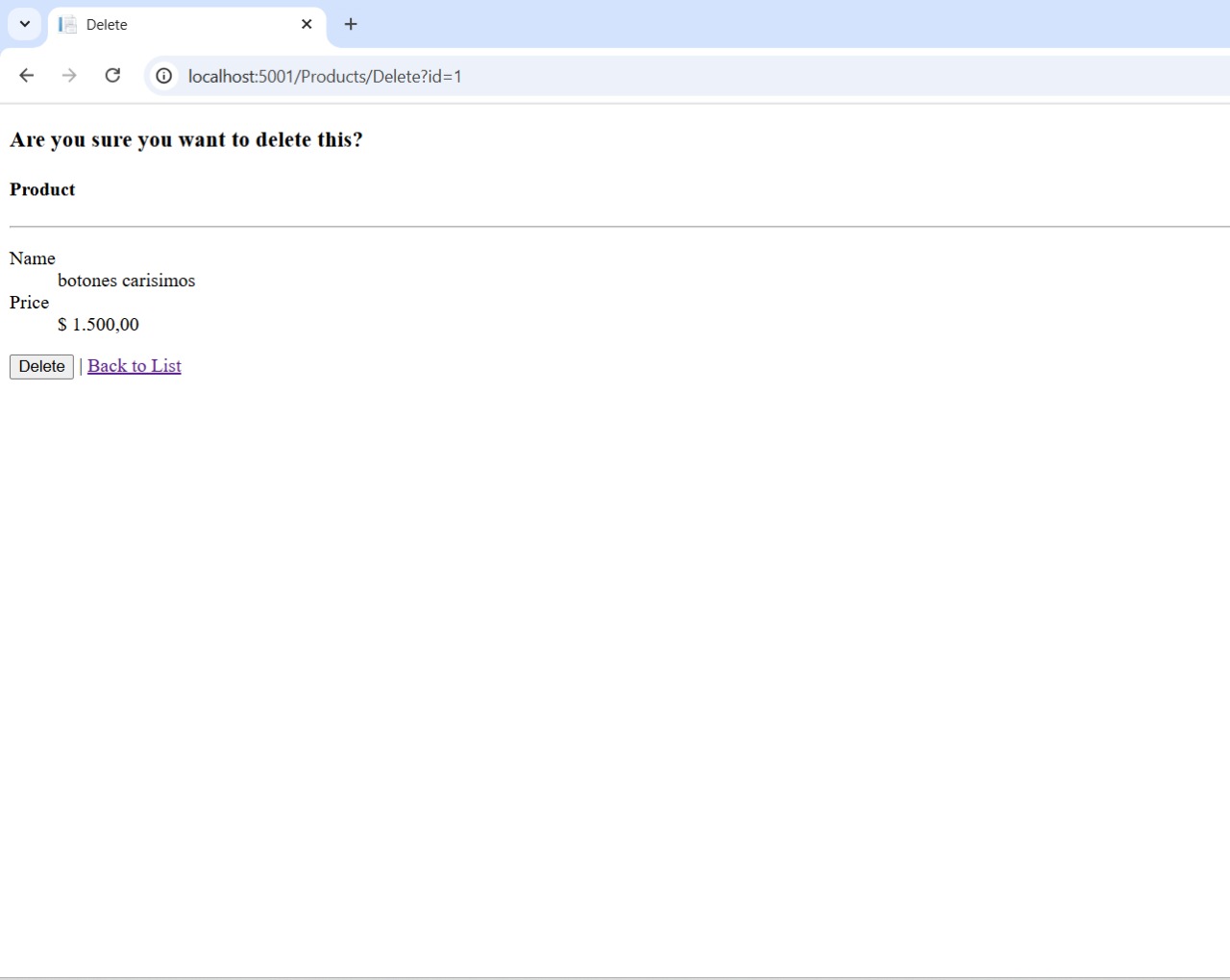


This image shows the Index Razor Page for the Product entity running in the browser at /Products. It displays a list of products (currently one item: "botones") with its price.

Crucially, it provides links for the other CRUD operations: Create New, Edit, Details, and Delete. This confirms that the data layer (Entity Framework Core) is correctly connected and the scaffolded pages are functioning, allowing users to Read the data and access the other manipulation tools.

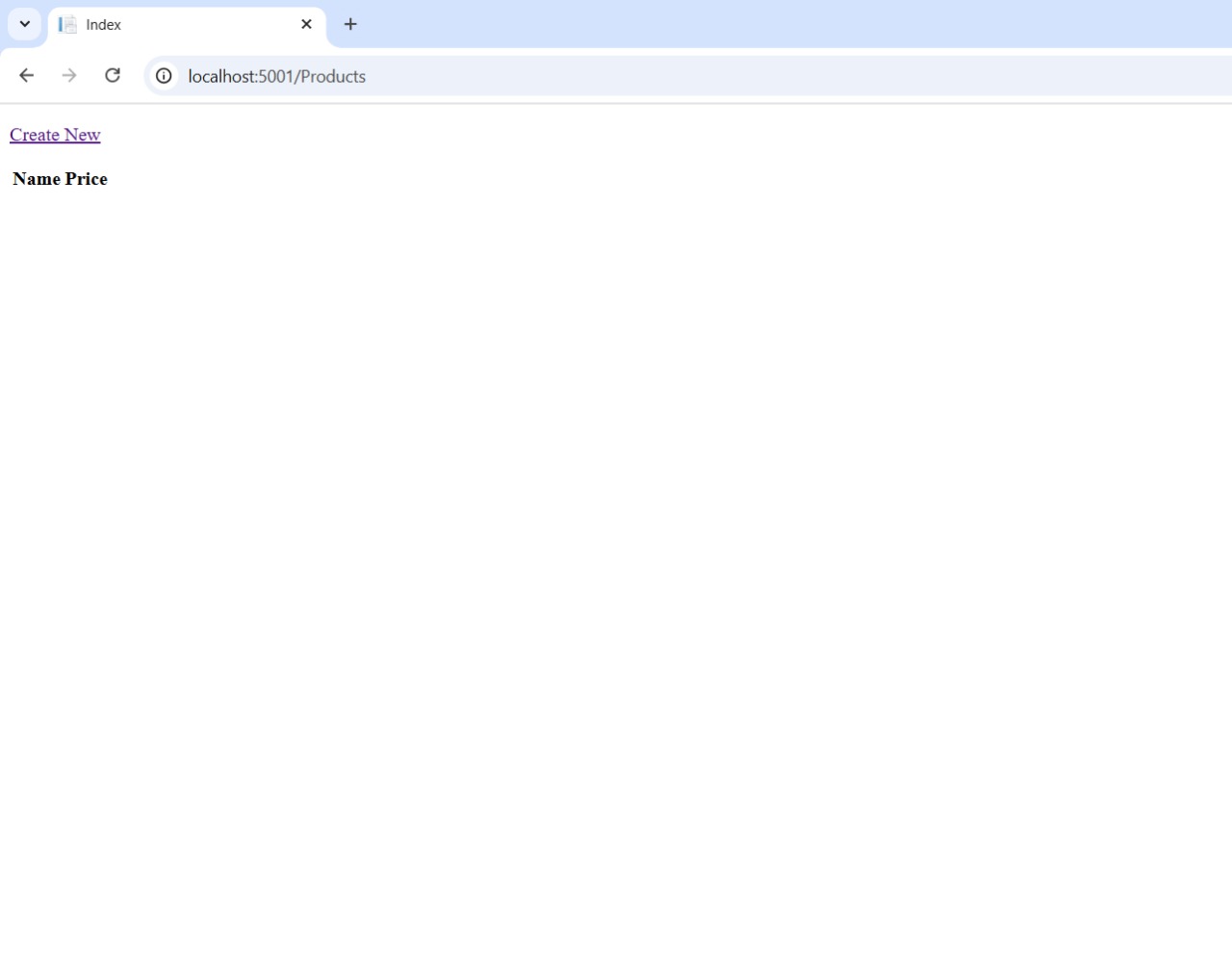


The presence of Edit, Details, and Delete links confirms the full suite of CRUD actions is accessible for the displayed product.



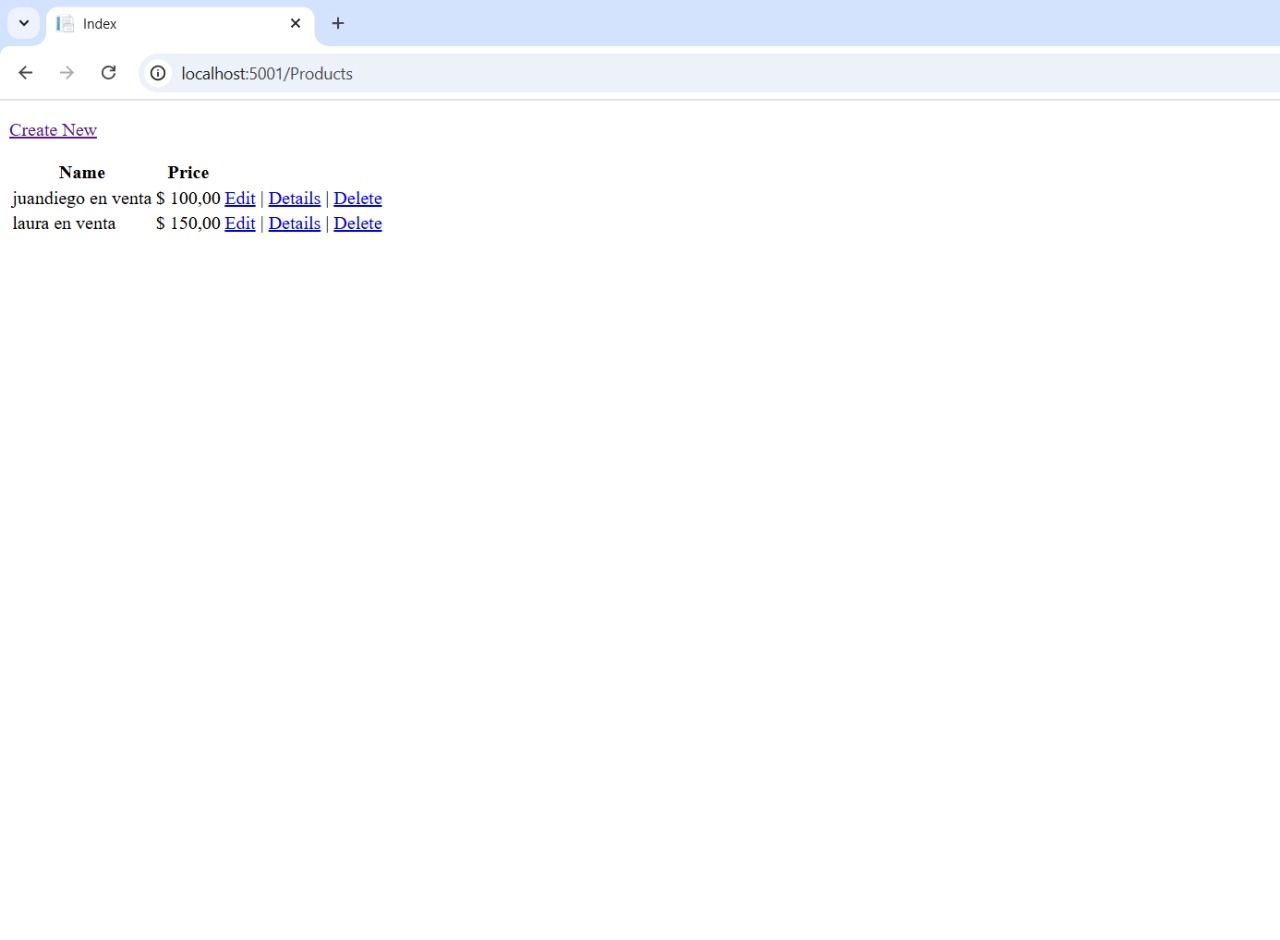
This image shows the Delete confirmation page for a product, accessed via the URL /Products/Delete?id=1. The page displays the data for the specific product (ID 1, Name: "botones carisimos") and asks the user, "Are you sure you want to delete this?"

This is the final step in the Delete part of the CRUD cycle. It's a standard web security/usability practice to require confirmation before permanently removing data from the database.



This image shows the /Products Index page after a successful Delete operation.The table rows are empty, with only the column headers (Name and Price) and the Create New link remaining.

The product that was viewed in the previous image is no longer displayed. This confirms the Delete part of the CRUD functionality worked correctly, and the record was removed from the database via Entity Framework Core.

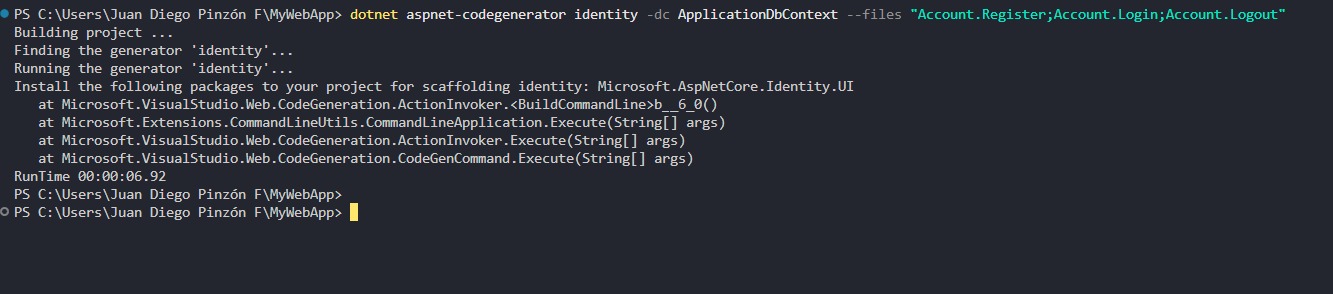


This image shows the /Products Index page again, now displaying a list of two new products ("juandiego en venta" and "laura en venta").

This re-confirms the successful implementation of the Create (or Update) and Read parts of the CRUD cycle.

The application is fully capable of managing product data, showing multiple records retrieved from the database via Entity Framework Core.

1. **Part 4: Securing Your Application with ASP.NET Core Identity**

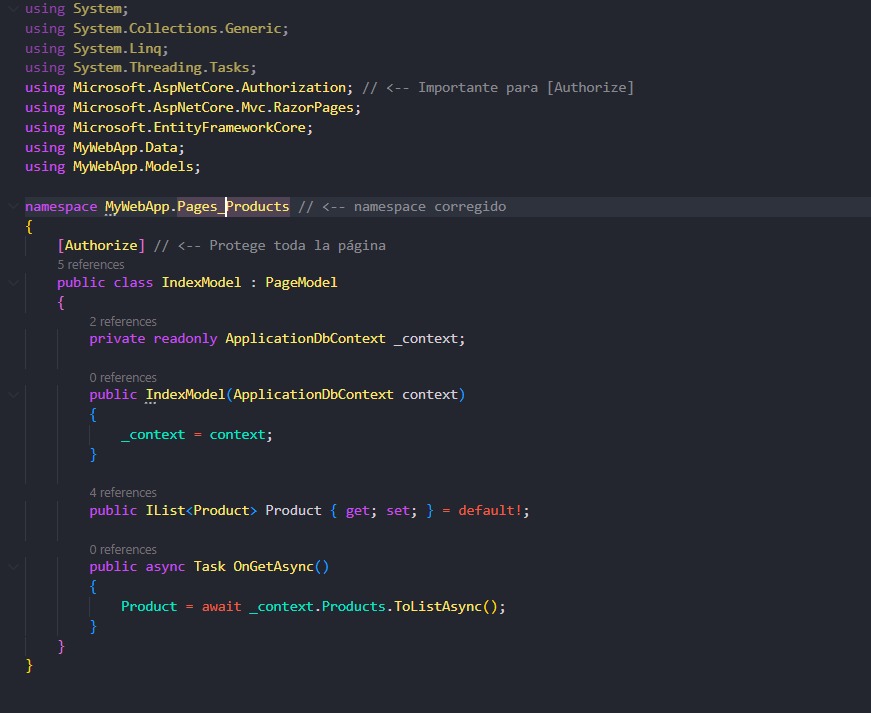


This image shows a command executed in the terminal: dotnet aspnet-codegenerator identity.... This command uses the ASP.NET Core Code Generator to scaffold Identity pages.

--dc ApplicationDbContext specifies the existing Entity Framework Core context where the Identity tables will be created.

--files "Account.Register;Account.Login;Account.Logout" indicates that the fundamental user authentication pages (registration, login, and logout) are being generated.

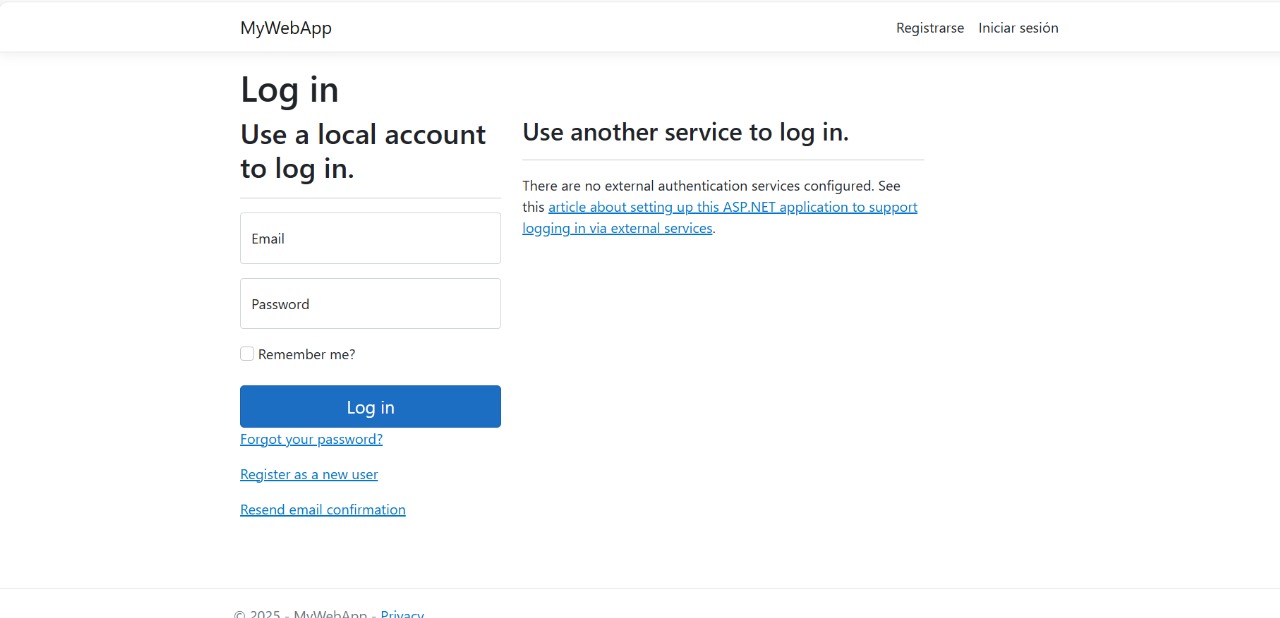
This is the essential step for integrating ASP.NET Core Identity (the user management/authentication system) into the application.



This image shows a C# code file for a Razor Page (IndexModel of the Products page) being modified to enforce security using the [Authorize] attribute. The [Authorize] attribute is placed directly above the IndexModel class definition.

This attribute mandates that any user attempting to access the Products Index page (and thus the CRUD functionality) must be logged in. If they're not, ASP.NET Core Identity will redirect them to the login page.

This is a crucial step in Securing Your Application by protecting specific content or functionality based on user authentication.



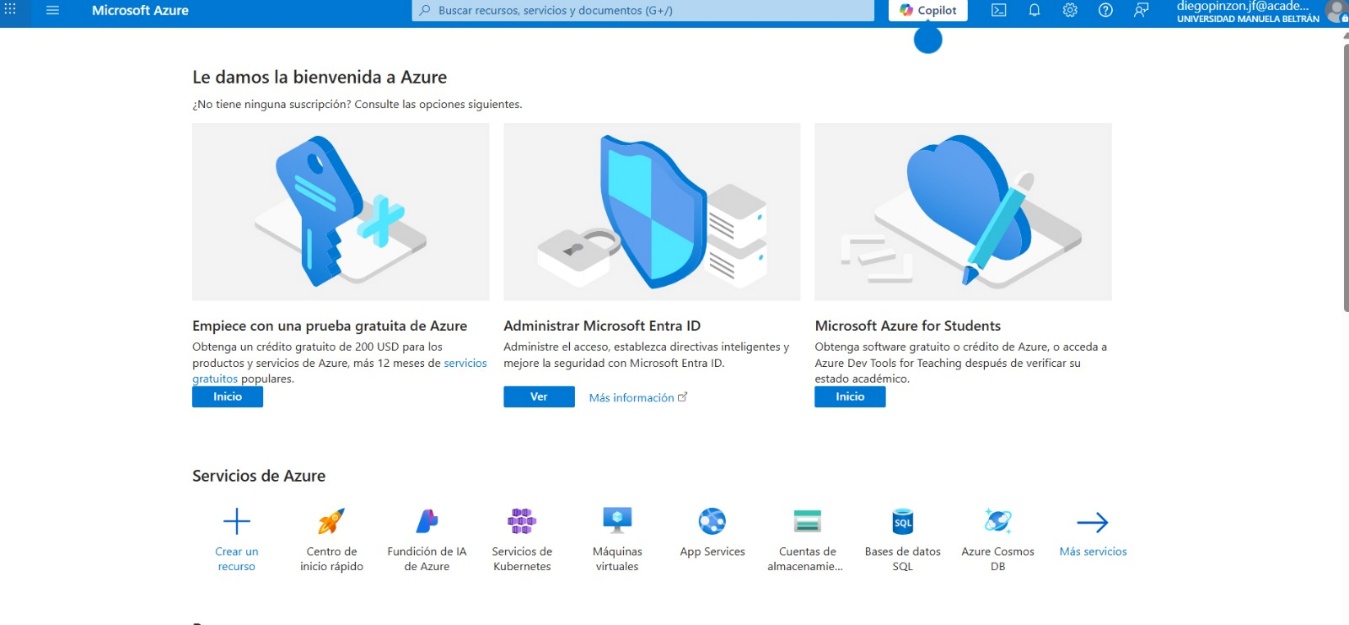
This image shows the Login page of the MyWebApp running in a browser.

This page was automatically generated (scaffolded) after executing the dotnet aspnet-codegenerator identity command (Image 15).

It provides fields for Email and Password for a local account and links for Forgot your password?, Register as a new user, and Resend email confirmation.

This is the functional result of setting up ASP.NET Core Identity to handle user authentication, confirming the application is now secured and ready to manage users.

1. **Part 5: Version Control and Automated Deployment**

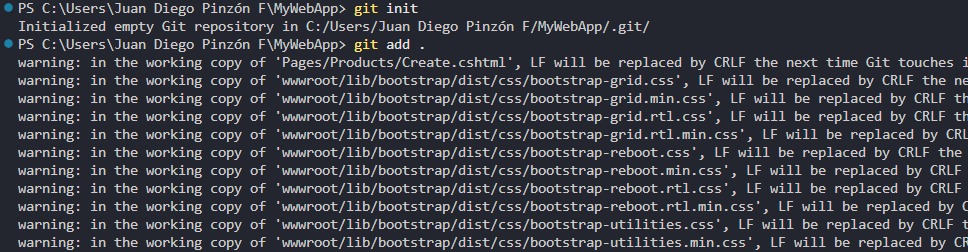


This image shows the Microsoft Azure portal's welcome screen.

Azure is a cloud computing platform used for deployment and hosting of web applications, databases, and other services.

The options shown (e.g., App Services, SQL Databases, Create a resource) are all part of setting up a production environment.

This image represents the first step in the final phase: moving the completed application code from the local development environment to a public cloud environment, which falls under Automated Deployment.

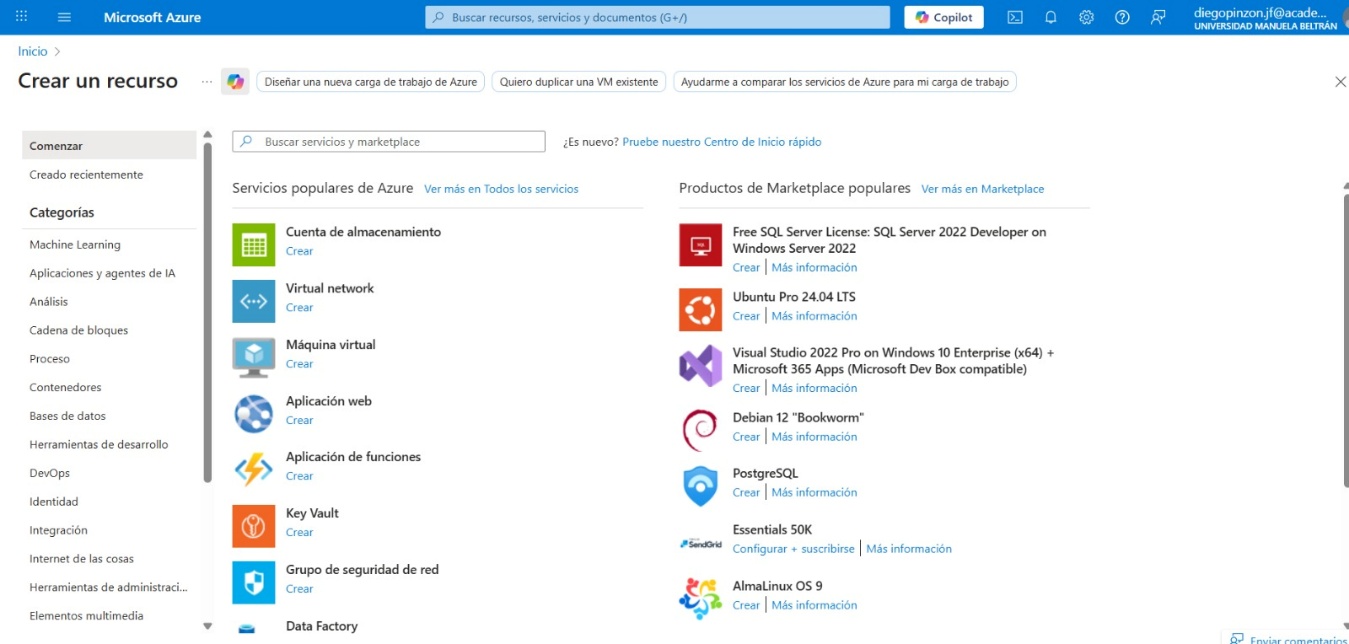


This image shows two fundamental commands being executed in the terminal for Version Control using Git:

git init: Initializes an empty Git repository in the current project directory (F:\MyWebApp). This prepares the folder to track changes to the code.

git add .: Stages all current files (including the recently scaffolded pages and static assets) for the next commit. The warnings about line endings (LF/CRLF) are typical when configuring Git on a Windows system.

These actions mark the beginning of managing the codebase with version control, a prerequisite for robust deployment.

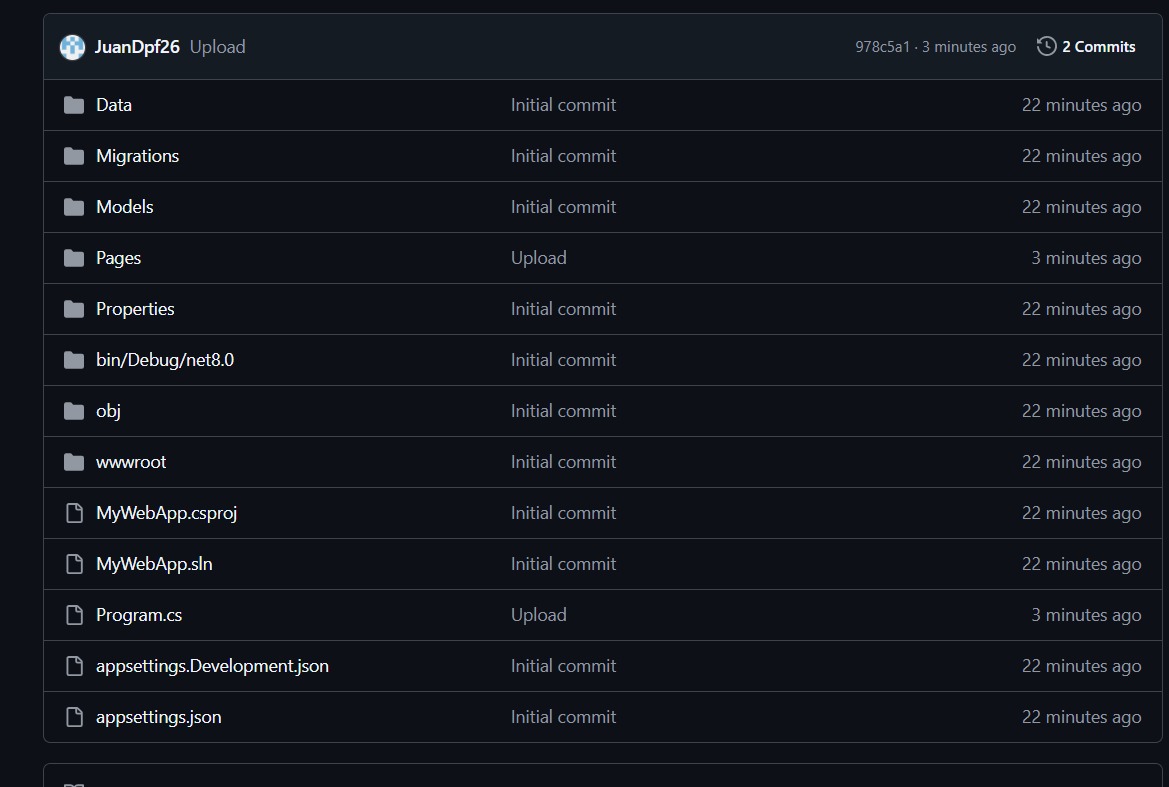


This image shows the "Create a resource" page within the Microsoft Azure portal.

The developer is browsing various services for hosting and supporting the application.

Options like "Aplicación web" (Web App) are the direct targets for deploying the ASP.NET Core application. Other services like SQL Server and Virtual Network are also visible.

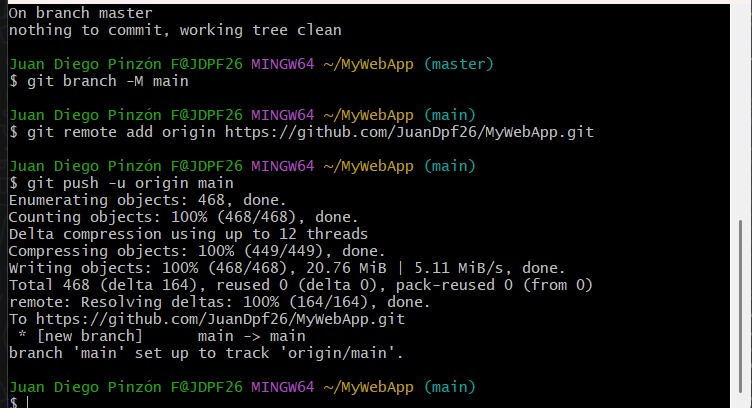
This is a step in the Automated Deployment phase, where the developer selects the cloud infrastructure needed to host the application publicly.



This image shows a file listing within a Git repository hosted online (likely GitHub or Azure DevOps, given the context).

It displays the project's file structure (Data, Models, Pages, Program.cs, etc.) and the commit history for those files (e.g., "Initial commit," "Upload").

This confirms that the local source code was successfully pushed to a remote version control repository. This is a mandatory step for collaborative development and is the foundation for setting up Automated Deployment pipelines (like Azure DevOps or GitHub Actions) to host the application.



This image shows a series of essential Git commands executed in the terminal to prepare the code for the remote repository and deployment.

git branch -M main: Renames the local branch from master to main, which is the current industry standard for the default branch name.

git remote add origin ...: Links the local Git repository to the remote GitHub repository (https://github.com/JuanDpf26/MyWebApp.git).

git push -u origin main: Pushes the entire local code history from the main branch to the remote repository on GitHub. The output confirms the push was successful, making the code available for version control and subsequent Automated Deployment.