**Computer Store Web Application**

for

Master of Science

Information and Communications Technology

ICT 4351 .NET Programming with C#

Richard Magill

University of Denver University College

March 10, 2021

Faculty: Kris Jamsa, PhD

Director: Michael Batty, PhD

Dean: Michael J. McGuire, MLS

**Introduction**

Throughout the Winter 2021 Academic Quarter, our course curriculum has focused on a continuing theme of building a set of “Computer Store Applications” using C# and Visual Studio 2019 as our Integrated Design Environment (IDE). Each week has generally built upon the previous lessons, resulting in steadily more sophisticated and capable applications. Three separate solutions were created:

* A .NET Framework console application with capabilities for orders, mailing lists, and quoting. This application would serve as an internal application for employees.
* A .NET Framework web application for customers, which provided information on products and locations.
* A .NET Core web application with a comprehensive set of capabilities.

This paper will primarily focus on the final .NET Core web application, as the full suite of technologies and techniques are well-represented by that application. Some topics that will be covered include:

* ASP.NET Identity management for user login and authentication
* Razor syntax, including HTML, CSS, and Page Models
* Data handling, including SQLite and XML
* Services, Endpoints, and Configuration as defined within the Startup.cs class
* REST API with Swagger

In addition to discussion of the aforementioned technologies, screenshots of the application will additionally be included.

**ASP.NET Identity**

For professional web applications, few topics are as important as security. ASP.NET Core Identity provides a powerful and standardized API that manages login, users, passwords, roles, and many other critical authentication functions (Anderson 2020). To implement security in this fashion, upon creation of the ASP.NET Core Web Application, I selected “Change Authentication” and then “Individual User Accounts”. Once this was complete, I was able to apply migrations to initialize the database, running an “Update-Database” command in the Nuget Package Manager Console. This creates a full security database in SQL Server, as shown in Figure 1 below:

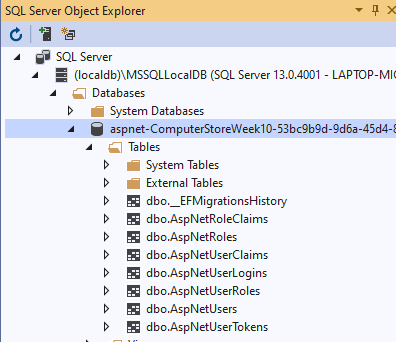


Fig. 1: ASP.NET Core Identity Tables

Next, I added services.Configure<IdentityOptions> to the Startup.cs class to configure services and cookies.

Finally, to complete the setup I scaffolded out Register, Login, Logout, and Register Confirmation files (Anderson 2020). With these key pages now in place, the only thing remaining was to include the [Authorize] attribute to all the pages that I wanted to secure. Figure 2 below shows my application upon startup. Note the functionality for Login, Registration, and Password Recovery:

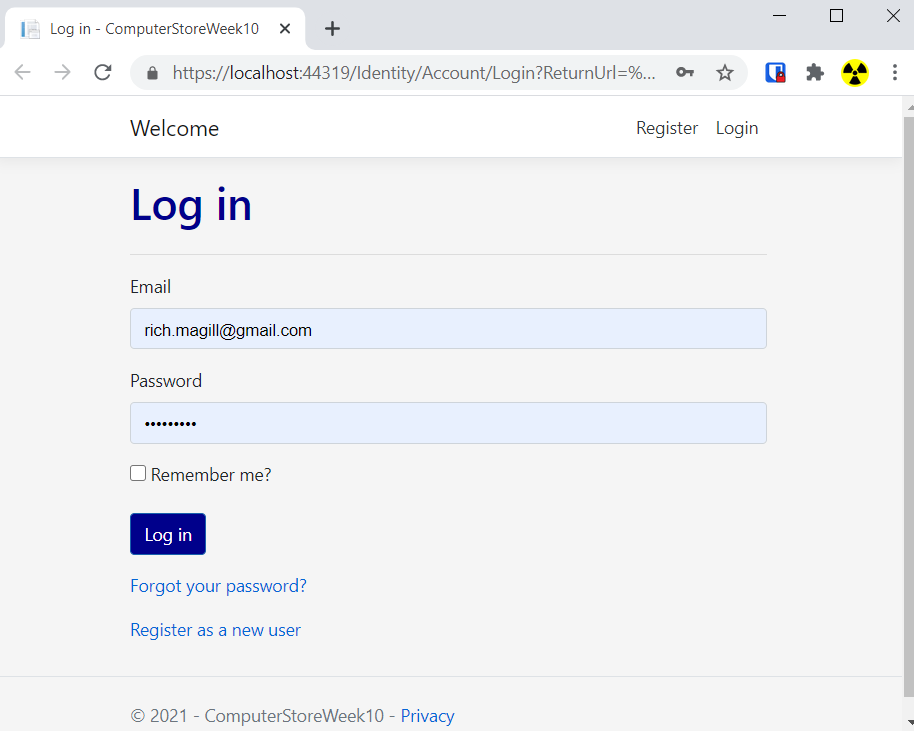


Fig. 2 Computer Store Login

Figure 3 below shows the application Index.html page after authentication is successful.

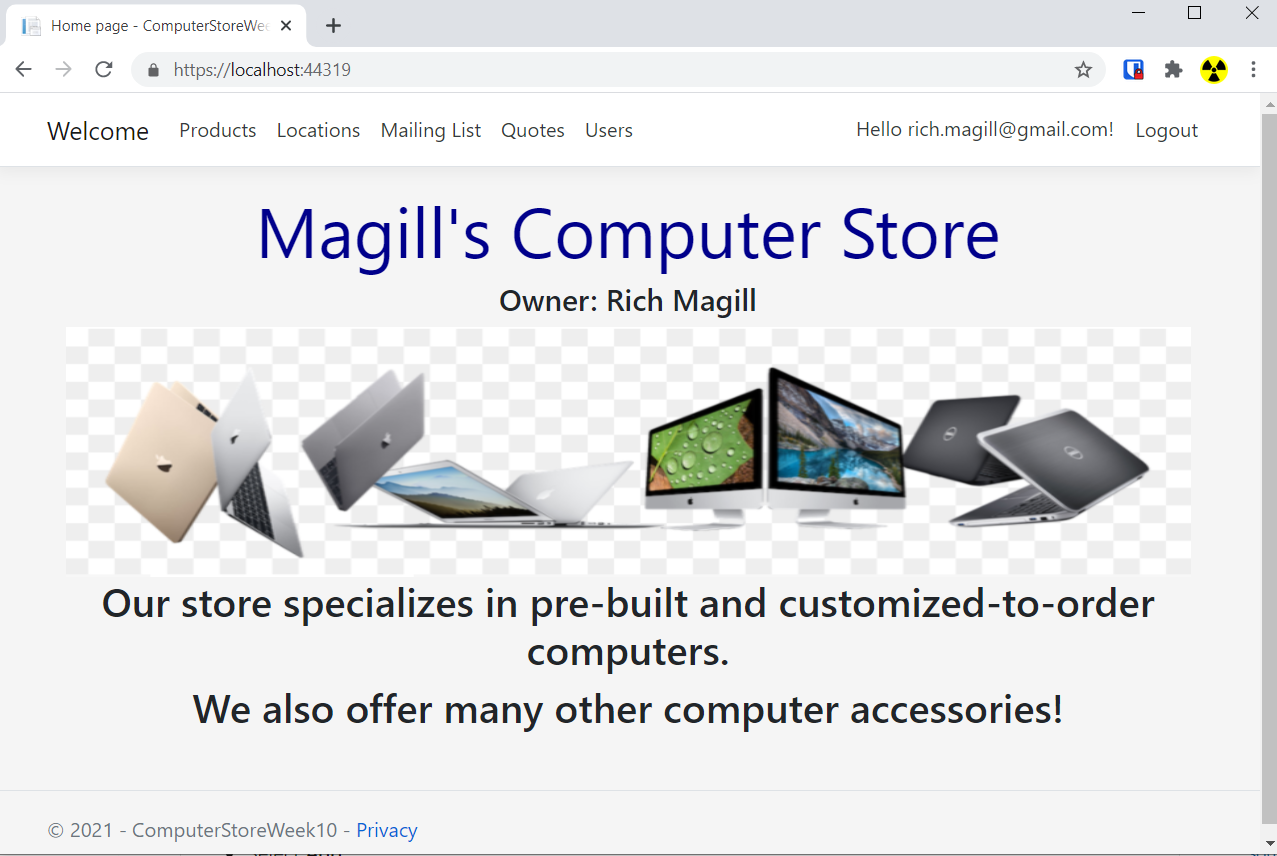


Figure 3: Index.html start page

**Razor Syntax**

The pages are done in Razor syntax, meaning that HTML and C# are actually used together within the same page. As opposed to the Model-View-Controller “MVC” model, the Razor page is known as a Model-View-ViewModel “MVVM” framework (Watson 2017). As shown in Figure 4, the C# elements are annotated in my page by the @ symbol. Here we can see that the HTML elements (<table>, <tr>, <td>, etc.) are essentially interwoven with the C# elements (foreach, class properties, Model, etc). The “foreach” looping mechanism combined with HTML allows for creation of grid-like tables and other repeating structures.



Figure 4: Razor syntax

The Razor page also contains links to a Cascading Style Sheet (CSS) to give the site a consistently styled look and feel. Figure 5 shows the Products page:

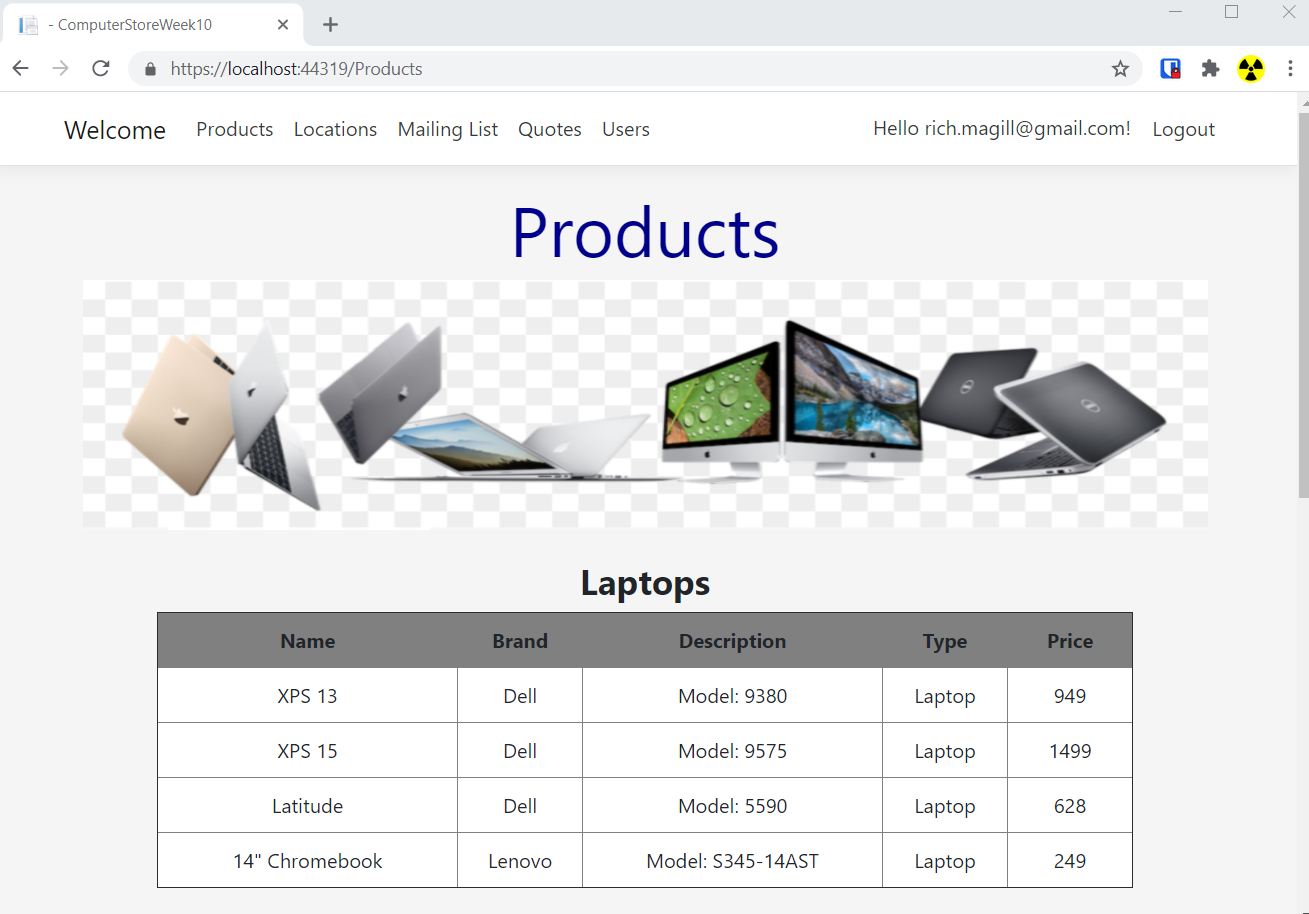


Figure 5: Products page

The “code-behind” of the Product Page model implements LINQ to filter the various product types into the correct tables, as shown in Figure 6:



Figure 6: Use of LINQ in the Product Page Model

**Data Handling**

As a learning exercise, instead of having a single data source (like an SQL database), this application includes a variety of data handling methods and tools, including Excel File IO, XML, MS SQL Server, and an SQLite database.

The MS SQL Server database uses Entity Framework to provide the authentication and identity management in the application. Product data is stored in the SQLite database and is handled via a database service. XML is used for Locations and Orders. Figure 7 shows a snippet of part of the XML in the store\_locations.xml file:

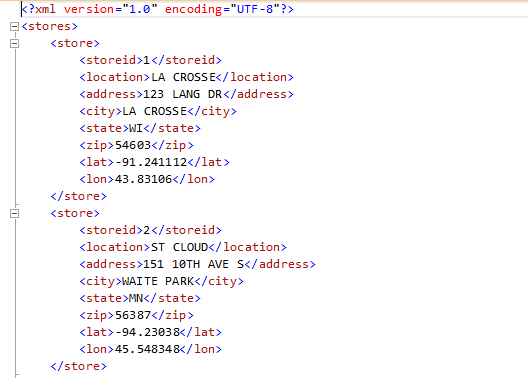


Figure 7: XML Store Location data

This XML is read into the application using XDocument and LINQ to XML (Agarwal 2010). Figure 8 shows a code sample of the method used to populate and filter the Locations page model.

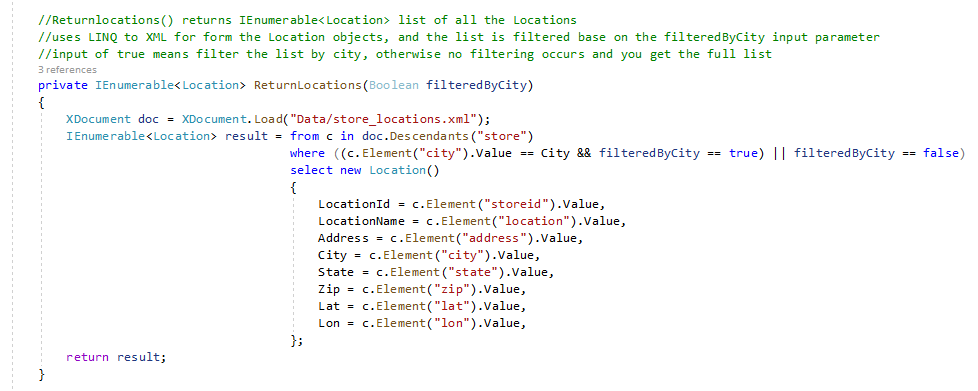


Figure 8: LINQ to XML to populate Locations page model

Figure 9 below shows the rendered Locations page. Note the Filter by City function that was implemented in the LINQ to XML method described above:

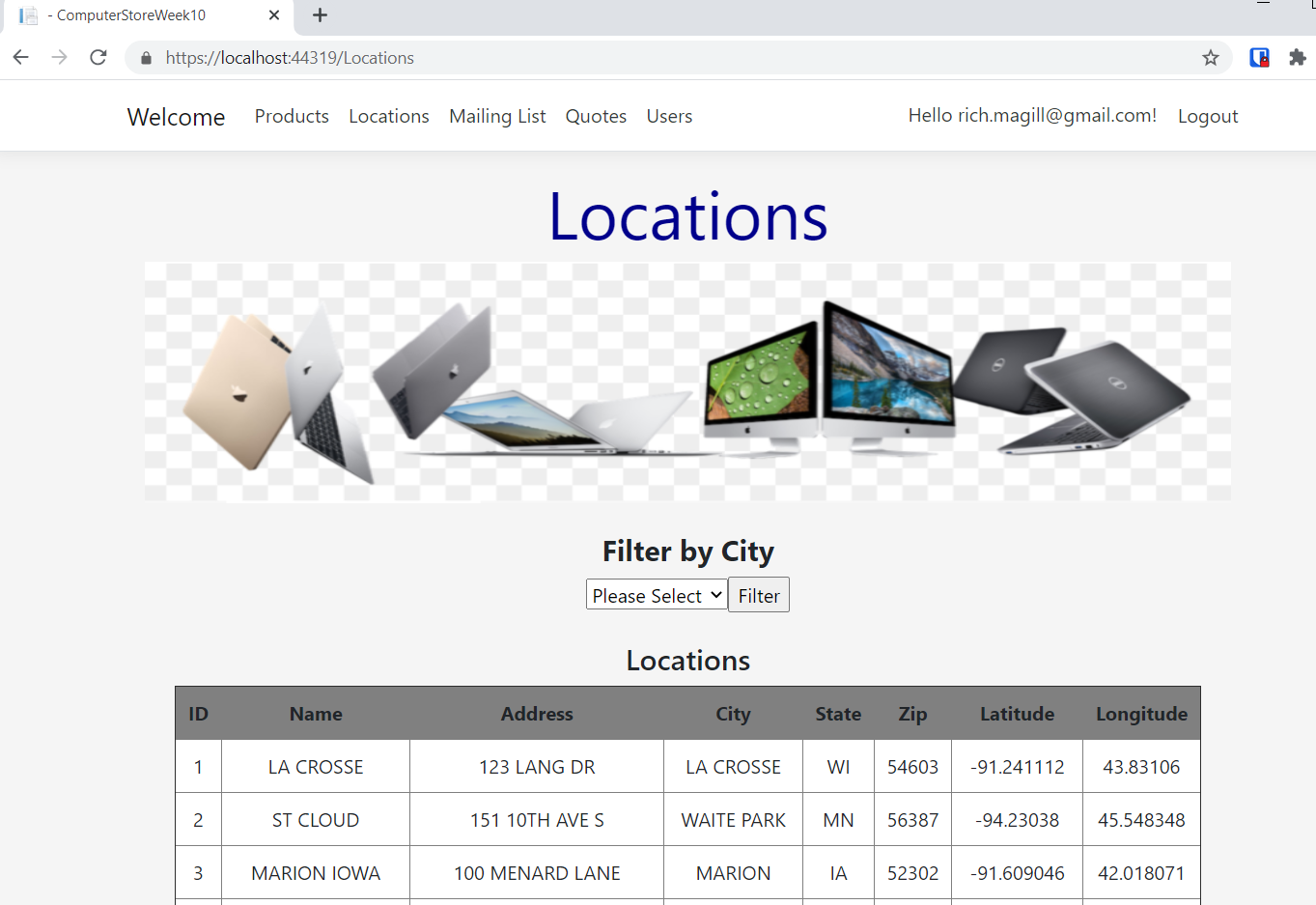


Figure 9: Locations page

**Startup.cs Class**

The Startup.cs class in this project contains many critical service, endpoint, and configuration elements. For example, in Figure 10 we can see the registration of many services implementing IServiceCollection, including Swagger, a DBContext service that configures Identity using EntityFramework, Razor pages, and my SQLite database service.

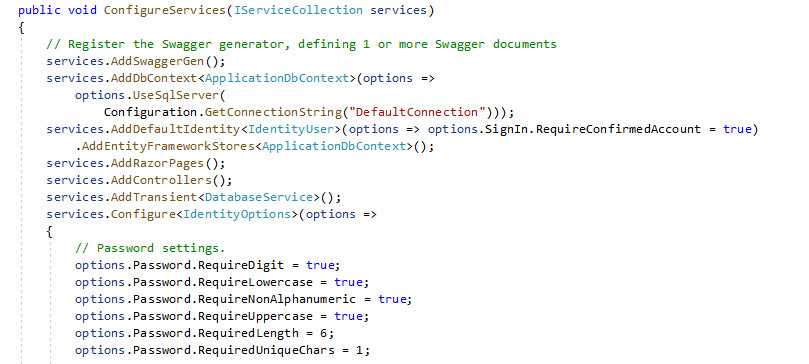


Figure 10: Configure Services in Startup.cs Class

**REST API**

In addition to all these required elements, I also implemented a Products REST API with Swagger (Hanselman and Richardson 2019). The REST API is a common and increasingly valuable JSON-based method of communication between applications, using HTTP requests. In order to implement this code, I created a ProductsApiController.cs class with a couple of simple HTTPGet methods:



Figure 11: HTTPGet methods in ProductsApiController

I then mapped the controller endpoint in the Startup.cs class:

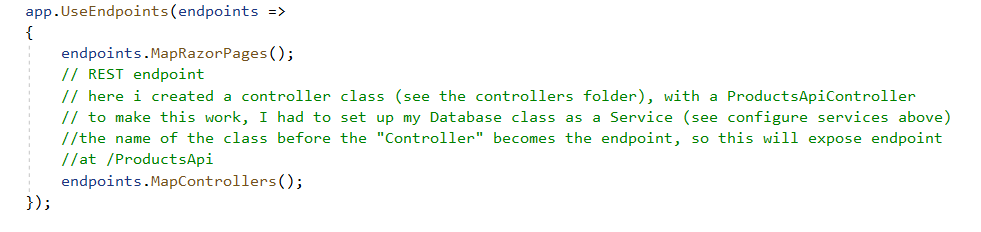


Figure 12: Controller Endpoint

I then used Nuget to get a package from Swagger (Swashbuckle.AspNetCore) and configured swagger endpoints as demonstrated in Figure 13 (microsoft.com 2020). Figure 14 shows the Swagger REST API interface:

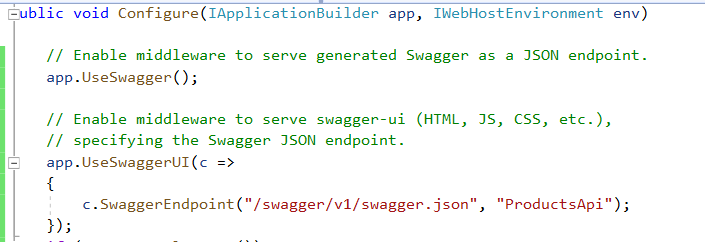


Figure 13: Swagger Endpoint configuration

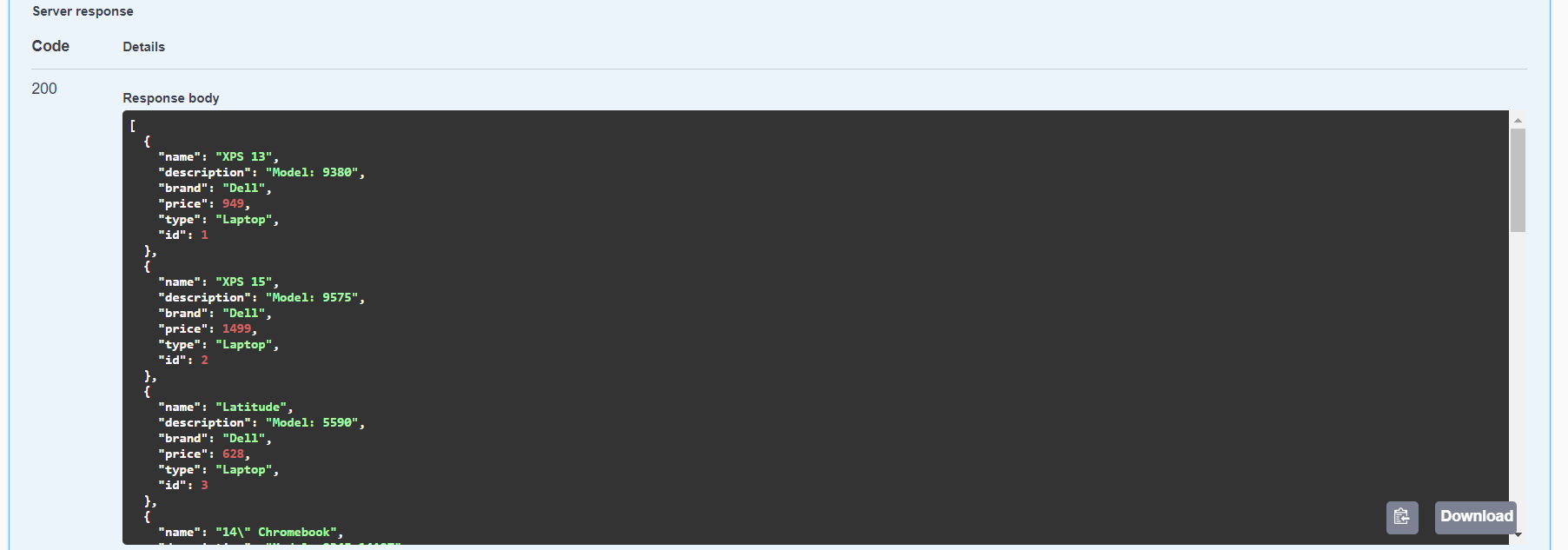
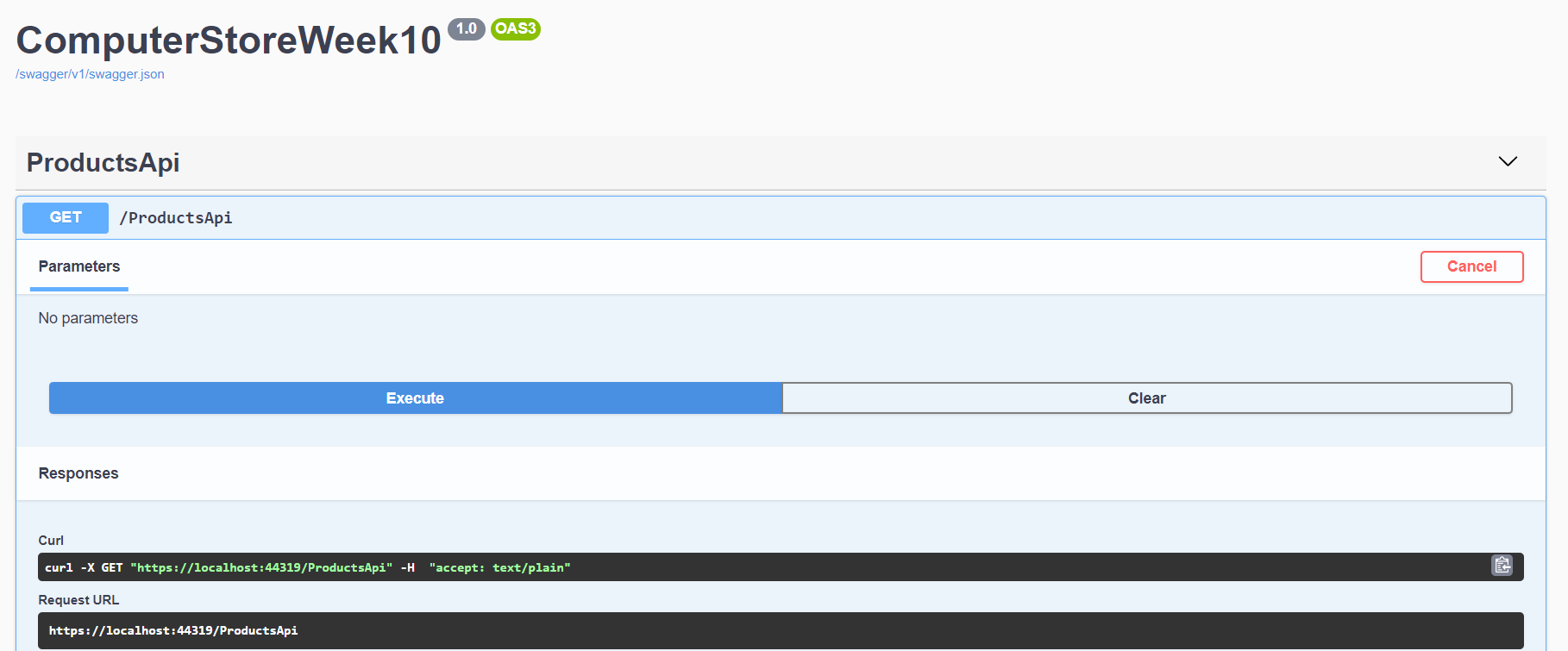


Figure 14: REST API with Swagger

**Conclusions**

While this application is relatively simple in appearance, it demonstrates a wide array of modern and powerful coding methods and technologies. The ASP.NET Core foundation for Identity, EntityFramework, Rest API, Razor pages, and the MVVM model essentially form state-of-the art components developed in the C# language and the Visual Studio 2019 IDE.

**References**

Anderson, Rick 2020. “Introduction to Identity on ASP.NET Core” Microsoft.com, 7/15/2020. <https://docs.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-5.0&tabs=visual-studio>

Anderson, Rick 2020. “Scaffold Identity in ASP.NET Core project” Microsoft.com, 5/1/2020. <https://docs.microsoft.com/en-us/aspnet/core/security/authentication/scaffold-identity?view=aspnetcore-5.0&tabs=visual-studio>

Watson, Matt 2017. “ASP.NET Razor Pages vs MVC: How Do Razor Pages Fit in Your Toolbox?” stackify.com. August 16, 2017. <https://stackify.com/asp-net-razor-pages-vs-mvc/>

Agarwal, Suprotim 2010. “Using LINQ to represent XML Data as an object” dotnetcurry.com, 4/7/2010. <https://www.dotnetcurry.com/ShowArticle.aspx?ID=490>

Hanselman, Scott and Richardson, Leslie 2019. “Making a Simple API, ASP.NET Core 101 (7/13)” Youtube.com. September 23, 1019. <https://www.youtube.com/watch?v=04IP3Yl8-64>

Microsoft.com, 2020. “Getting Started with Swashbuckle and ASP.NET Core” 6/26/2020. <https://docs.microsoft.com/en-us/aspnet/core/tutorials/getting-started-with-swashbuckle?view=aspnetcore-5.0&tabs=visual-studio>