Created Solution

* ASPNET\_WebApplication
* ASPNET\_API
* Business
* DataAccess
* Test.NUnitTestProject
* Test.UIEdgeDriver

Adding Dapper to DataAccess

Adding Microsoft.EntityFrameworkCore to DataAccess

<https://dotnetdetail.net/how-to-add-swagger-to-asp-net-core-3-0-web-api/>

<https://docs.microsoft.com/en-us/aspnet/core/tutorials/getting-started-with-swashbuckle?view=aspnetcore-3.1&tabs=visual-studio>

Adding Swashbuckle.AspNetCore to ASPNET\_API \*\* Use Pre-Release since we’re on .net 3.1

\*\* Add in Registration Codes.

Adding Microsoft.AspNetCore.Mvc.Formatters.Json to ASPNET\_API

<http://localhost:5000/swagger/index.html>

Config Updates

* Web\_API: Change Settings for API to be 5001

Test Project

* Update Solution to .NET Core 3.1
* Update Nuget Packages for Test project

Within DataAccess Classes

* Add in Customer & DB Context Classes
* Create Database
* BootStrapper
* CREATE TABLE [dbo].[LogInformation](
* [LogId] [int] IDENTITY(1,1) NOT NULL,
* [Product] [varchar](100) NULL, -- Application its coming from
* [Type] [varchar](50) NULL, -- Informational / Error
* [Class] [varchar](50) NULL,
* [Method] [varchar](50) NULL,
* [Message] [varchar](4000) NULL,
* [Source] [varchar](4000) NULL
* ) ON [PRIMARY]
* GO
* CREATE TABLE [dbo].[Customer](
* [CustomerId] [int] IDENTITY(1,1) NOT NULL,
* [FirstName] [varchar](100) NULL,
* [LastName] [varchar](100) NULL,
* [State] [varchar](2) NULL
* ) ON [PRIMARY]
* GO

Everyone seems to be writing services and WebAPI (Application Programming Interface) these days. It’s a huge *buzz* word and one that everyone wants on their resume. This is not really new either so I suspect you’ve been writing them already. The sample controller that Microsoft puts into its solution works but it may not show you the best approach to writing services. Do you return every response the same? Why does the ValuesController in the sample return some with a List, some a string, and others with Void? If you are familiar with WebAPI, you may already know the service can return different status codes to represent different responses. With these response types, what are you returning? There are multiple response codes and I’m guilty of writing them wrong. In attempting to #LearnGrowShare, I had initially started to watch a Pluralsight course on *Building a RESTful API with ASP.NET Core* <https://app.pluralsight.com/library/courses/asp-dot-net-core-restful-api-building/table-of-contents>. While I haven’t finished this entire course, I started to get exposed to the different codes and my lack of knowledge on best API practices. Microsoft’s documentation here <https://docs.microsoft.com/en-us/aspnet/web-api/overview/getting-started-with-aspnet-web-api/action-results> goes more into the different way to response from your services. Since there are so many different ways to respond, I wasn’t fully aware how Microsoft handled some of these cases until I found this article <https://www.exceptionnotfound.net/http-status-codes-in-asp-net-web-api-a-guided-tour/>.

Add in Customer API

* Add in Controller
* Return Customer Information

services.AddDbContext<IBootStrapperContext>(options =>

options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));

Adding Microsoft.EntityFrameworkCore to API

Adding Microsoft.EntityFrameworkCore.SqlServer to API

Adding Scutor to API

services.Scan(scan =>

scan.FromCallingAssembly()

.AddClasses()

.AsMatchingInterface());

Setup Unit testing of your Controllers

* Add Moq to Unit test Project.
* Write Test for GetAll Customers
* Add in additional Nuget Packages
* ASPNET\_API
* DataAccess

\*\* Write a few different Tests to Confirm working process. (Rich to fill out)

Now Lets work on the UI

* Add Controller
* MVC Controller with Read/Write actions
* CustomerController
* Connect to WebAPI Solution
* Create Model
* Create View
* Then Add in all the rest of the Views
  + Details
  + Create
  + Edit
  + Delete

<https://dotnetcoretutorials.com/2017/09/23/using-automapper-asp-net-core/>

<https://code-maze.com/automapper-net-core/>

Add in Auto Mapper to Service Layer

AutoMapper.Extensions.Microsoft.DependencyInjection

Add in Service Layer DTO

* CustomerDTO

Within View, Hide ID Fields

**\*\*** This seems like the right link to add in the aspnetusertables. Still need to confirm and do this.

<https://www.c-sharpcorner.com/article/asp-net-core-identity-tables-customization-with-visual-studio-2017/>

Add ASP.NET Identity Tables to Solution.

<https://entityframework.net/knowledge-base/28636511/how-to-create-asp-net-identity-tables-inside-existing-database->

Install system.identitymodel.tokens.jwt to API

UGH… Going to need to do this a few more times.

Setup Routing

Create Login

Create Token