Worksheet 1: Developing web services I

# Objectives

* Develop Web Application Programming Interface (API).
* Test the Web API.

### Task 1: Creating REST Web API Project using ASP.NET MVC

**HTTP** is not just for serving up web pages. It is also a powerful platform for building APIs that expose services and data. HTTP is simple, flexible, and ubiquitous. Almost any platform that you can think of has an HTTP library, so HTTP services can reach a broad range of clients, including browsers, mobile devices, and traditional desktop applications.

**ASP.NET Web API** is a framework for building web APIs on top of the **.NET Framework**. This lab exercises will show you the basics of creating a web application with an **ASP.NET Web API** back end. We will be using **Entity Framework 6** for the data layer, and Knockout.js for the client-side **JavaScript** application.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Client**  Desktop  ASP.NET  ASP.NET MVC  Vue js  AngularJS  JQuery |  | |  |  |  |  |  | | --- | --- | --- | --- | --- | | Web API Controller |  | Data Access Repository | 4 | ADO.NET  Entity  Framework |   **Web API Service** | **Database** |

Software used in this lab exercise is Visual Studio 2019.

In this exercise, you will create a Web API using **ASP.NET Web API 2 with Entity Framework**  to expose its database data through **Web API services**.

We will be incrementally build up the various pieces shown in this architecture to emulate the use of microservices. For the client, we will be using a Web Client. The web client will consume both the Product and Comment Web API and display the information as a web page for the user. Do note that the Product and Comment Web API are developed on different endpoint and port number. This means that both of them are different VS project (ie. You will have 2 instances of vs2019 running the Product and feedback project concurrently.) These 2 vs2019 projects can be viewed as 2 separate independent microservices serving different purposes for the entire eco-system. The Web client can be any webpage hosted in any web server. For simplicity, the web client to be used in the lab will simply be a single page webpage (using jQuery) running off locally from the computer filesystem.

<https://localhost:44200/api/Feedbacks>

<https://localhost:44366/api/Products>

Product Service

Feedback Service

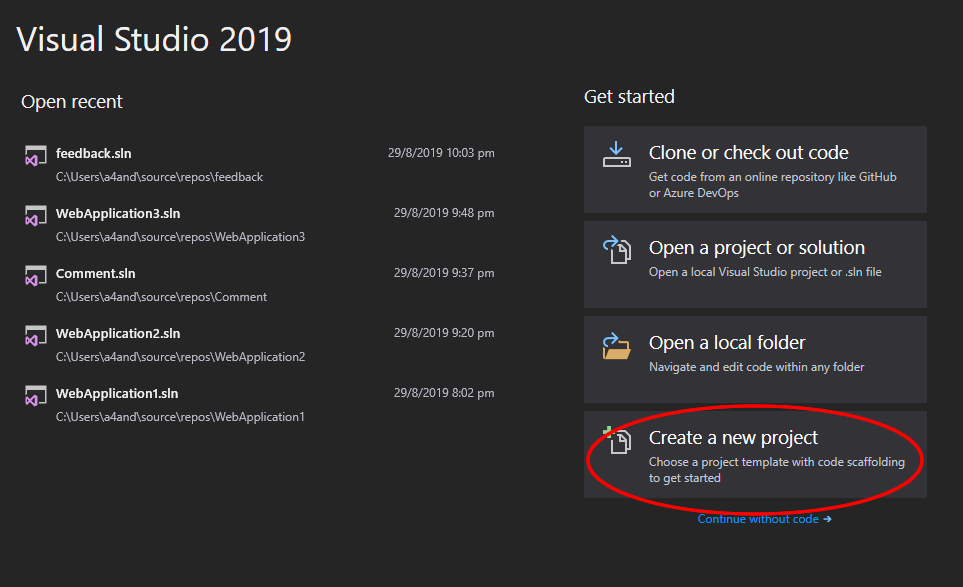
Client : Web, Mobile, Desktop, IoT etc

DB

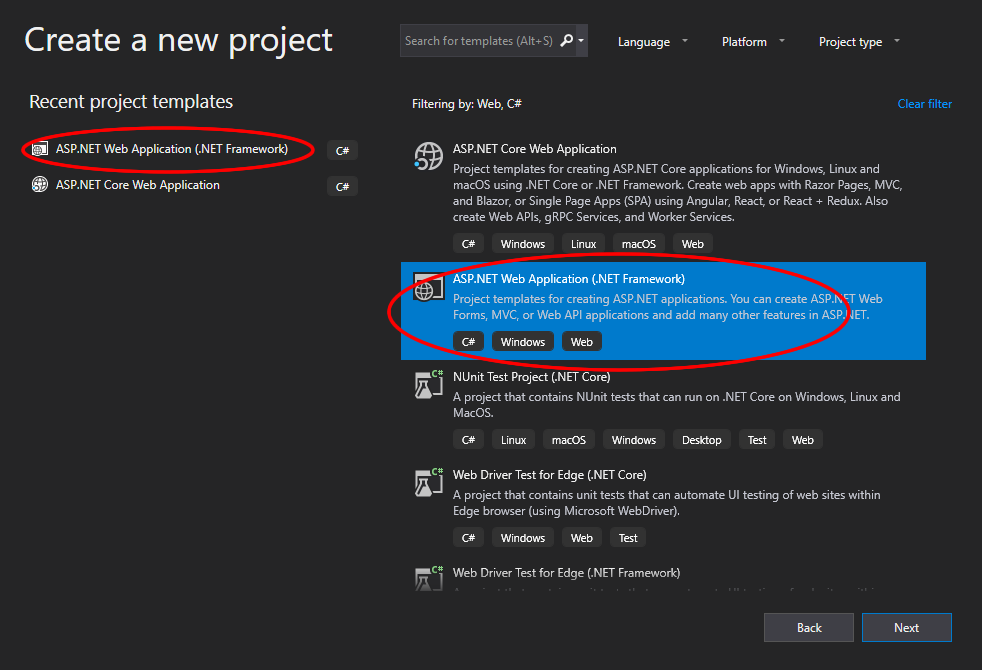
DB

**Create the Project**

1. Start your Visual Studio 2019. From the **File** menu, select **New**, then select **Project**. (Or click **New Project** on the Start page.)

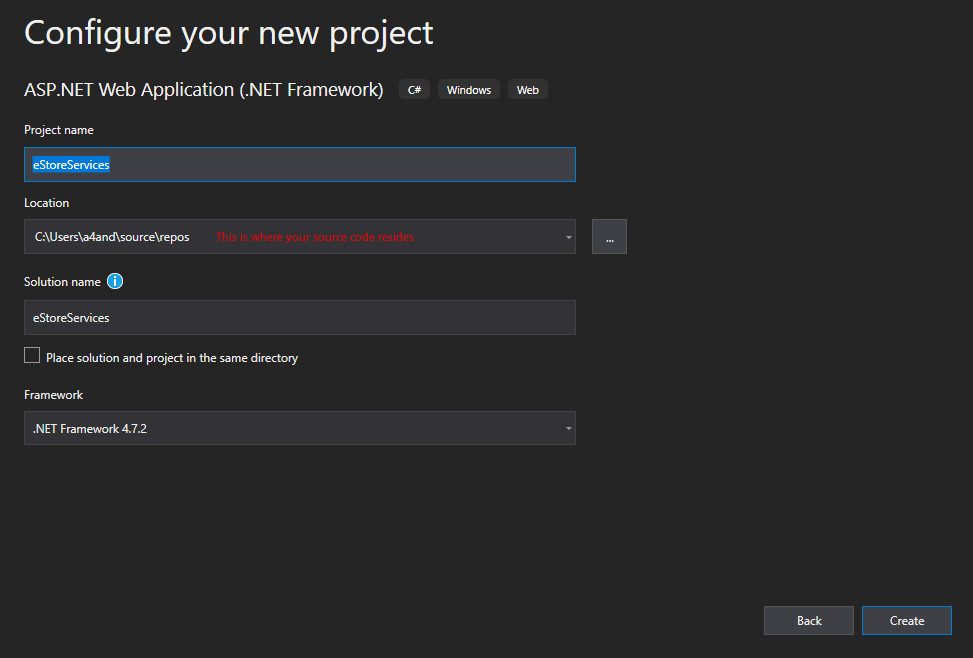


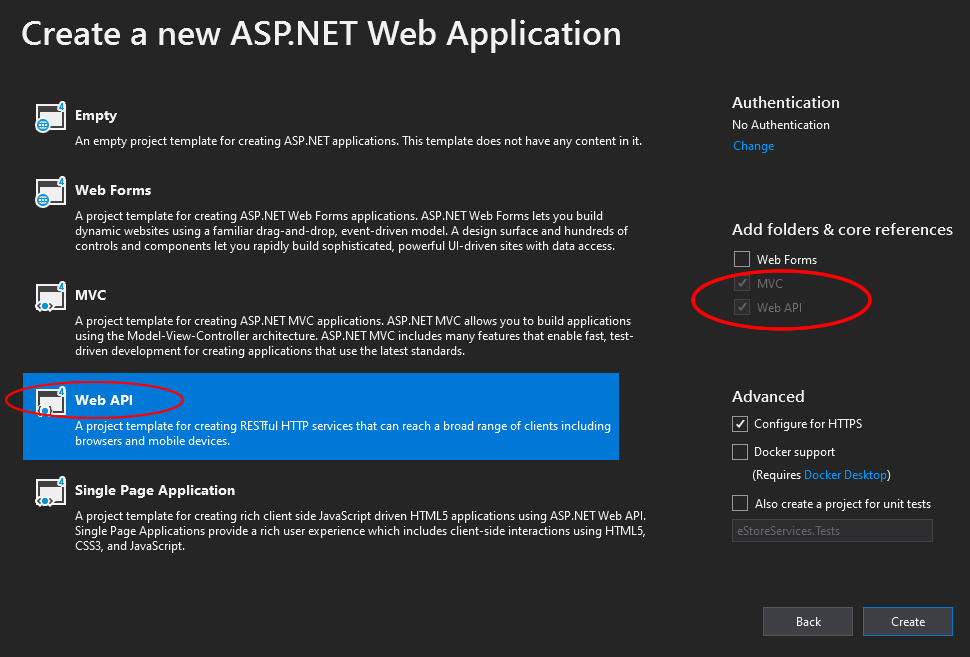
1. In the  dialog, choose according to the picture and click **NEXT**.



1. Let your project be called eStoreServices

Choose a suitable destination to store your project source codes and Click **Create**.



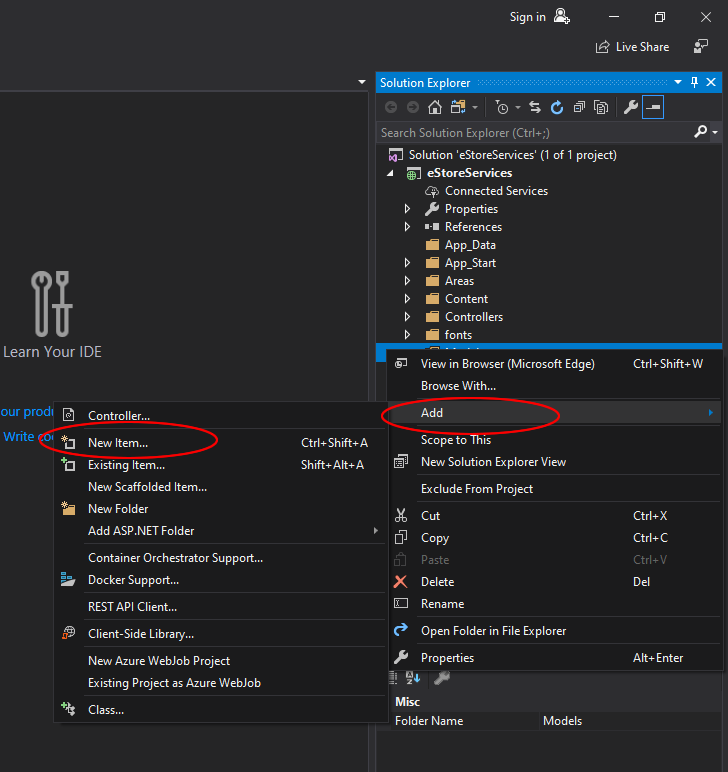
1. Choose the right type of application according to the picture.

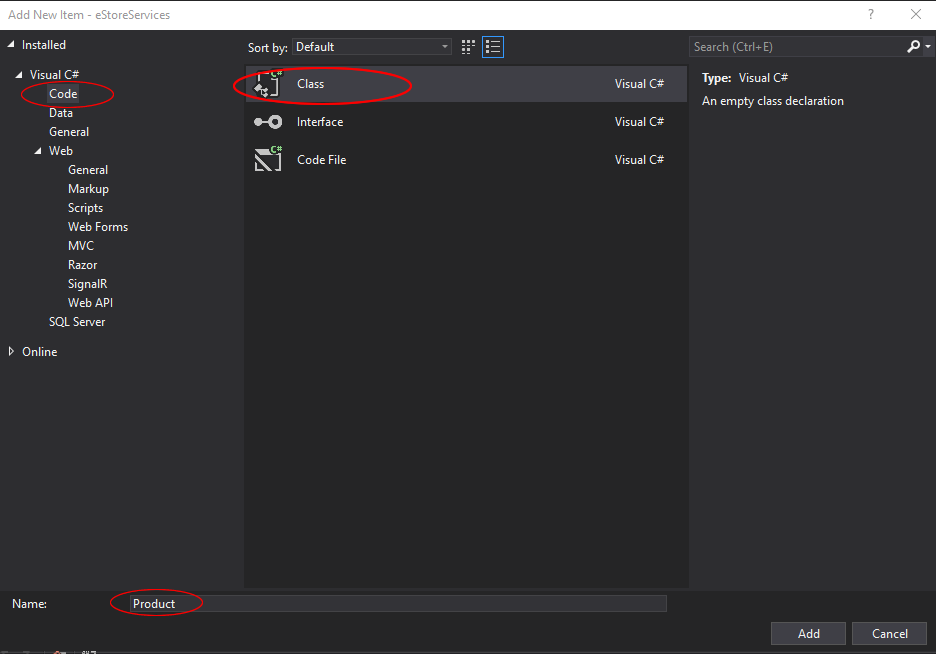
## Adding a Model

In this exercise, we will create the database by using the **Code First** approach to **Entity Framework** (EF). With **Code First**, you write C# classes that correspond to database tables, and EF creates the database. (<http://msdn.microsoft.com/en-us/library/ms178359%28v=vs.110%29.aspx#dbfmfcf>)

We start by defining our domain objects as POCOs (plain-old CLR objects). For this exercise, we will create the following the **Product** class.

1. In **Solution Explorer**, right click the **Models** folder. Select **Add**, then select **Class**. Name the class **Product**.



****

1. Replace all of the boilerplate code in **Product.cs** with the following code:

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.Linq;

using System.Web;

namespace eStoreServices.Models

{

public class Product

{

[Key]

public int BookID { get; set; }

[Required]

public string Title { get; set; }

public string Authors { get; set; }

public byte[] CoverImage { get; set; }

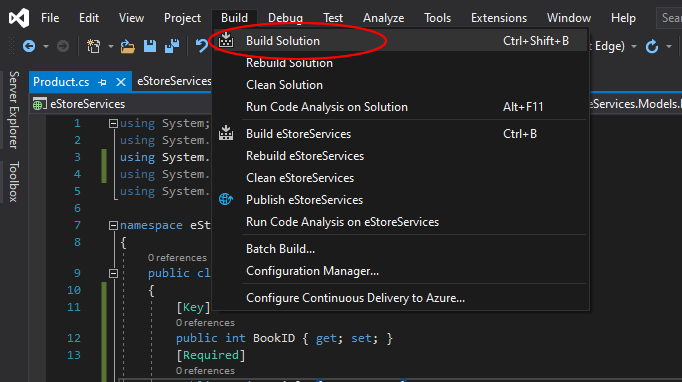
}

}

The [Key] annotation means that the BookID is the primary key

The [Required] annotation means that the field Title is mandatory

1. Compile your Product class by building Solution

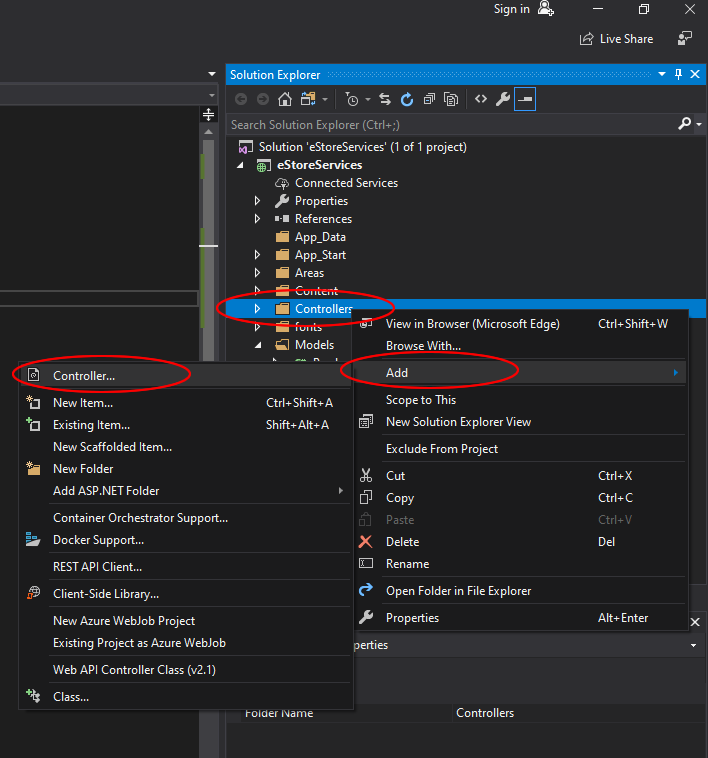


**Entity Framework** will use the model to create database table. For each model, the **BookID** property will become the primary key column of the database table.

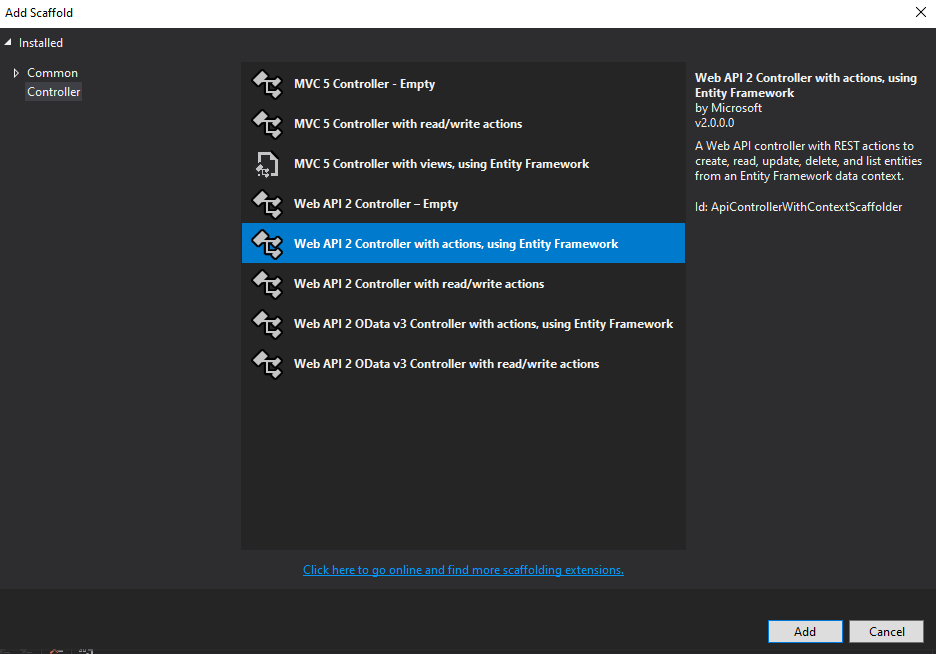
**Adding a Controller**

In this section, we will add **Web API** controllers that support CRUD operations (create, read, update, and delete). The controllers will use **Entity Framework** to communicate with the database layer.

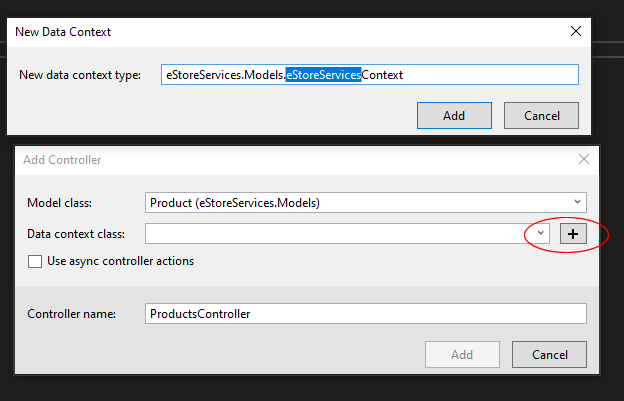
1. In **Solution Explorer**, right-click the **Controllers** folder. Select **Add**, then select **Controller**.



1. In the **Add Scaffold** dialog, select **Web API 2 Controller with actions, using Entity Framework**. Click **Add**.

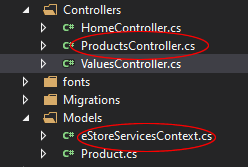


1. In the **Add Controller** dialog, do the following:
2. In the **Model class** dropdown, select the Product class. (If you do not see it listed in the dropdown, **make sure that you built the project**.)
3. Leave the controller name as **ProductsController**.
4. Click plus **(+)** button next to **Data Context Class**.



1. The dialog adds two classes to your project:

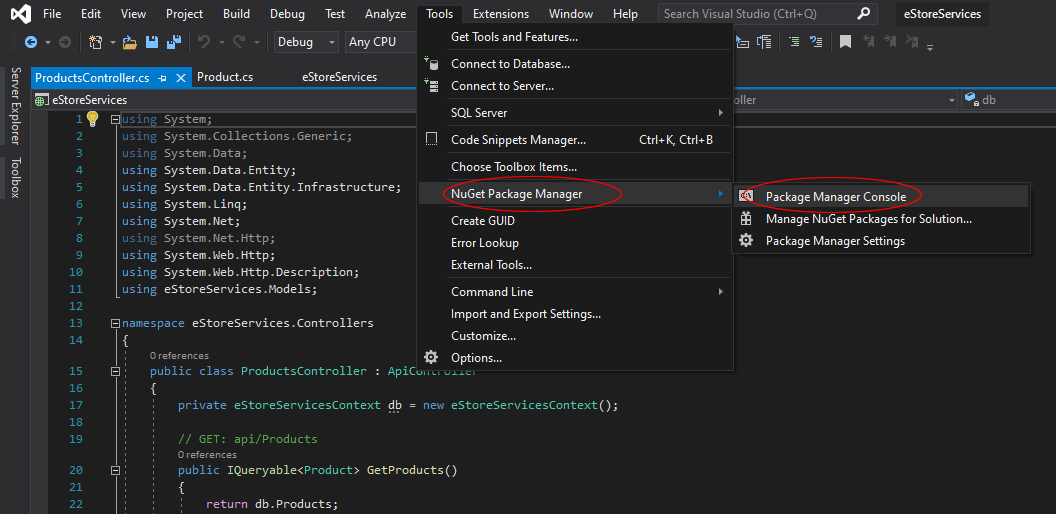
* **ProductsController** defines a Web API controller. The controller implements the REST API that clients use to perform CRUD operations on the products in the database table.
* **eStoreServicesContext** manages entity objects during run time, which includes populating objects with data from a database, change tracking, and persisting data to the database. It inherits from DbContext.



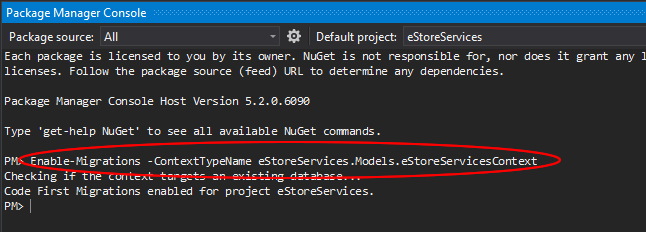
### Seeding the Database

In this section, you will use **Code First Migrations in Entity Framework** (EF) to seed the database with initial data.

1. From the **Tools** menu, select **NuGet Package Manager**, then select **Package Manager Console**. In the **Package Manager Console** window, enter the following command:

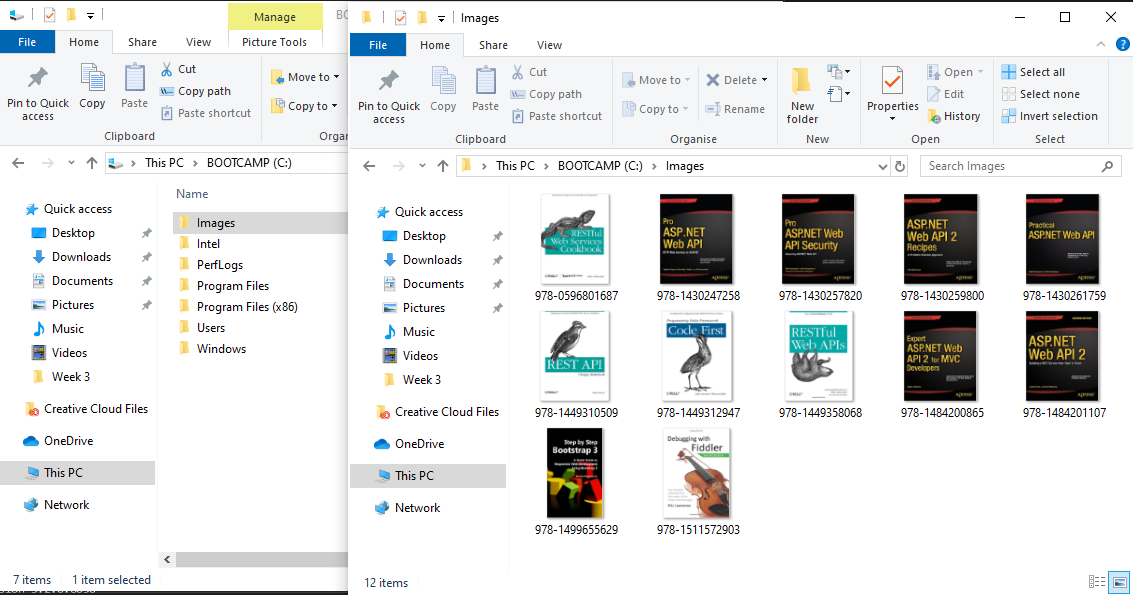


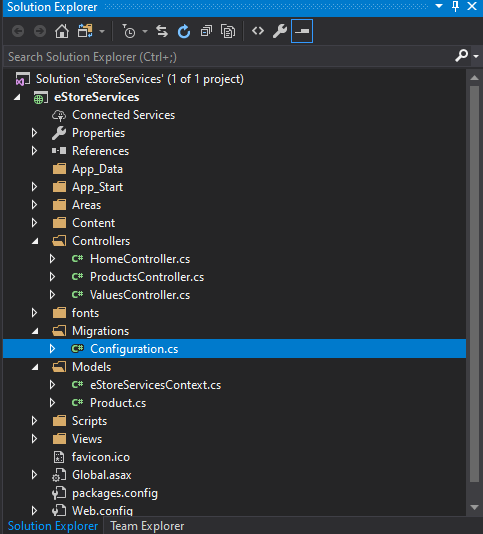
**Enable-Migrations -ContextTypeName eStoreServices.Models.eStoreServicesContext**



This command adds a folder named **Migrations** to your project, and a code file named **Configuration.cs** in the **Migrations** folder.

1. Each of the product in this exercise contains a cover image. We are going to store the image as a binary data in the database, in C# as **Byte[]**.
2. Download the zipped **Images.zip** from **LMS** and extract into the C Drive as shown in the picture. Make sure that the path is exactly C:\Images



1. Open the **Configuration.cs** file. Add the following **using** statement.  
   

**using eStoreServices.Models;**

**using System.IO;**

1. Then add the following the **ReadFile** method just below the constructor method.

**//Open file into a filestream and read data in a byte array.**

**byte[] ReadFile(string sPath)**

**{**

**//Initialize byte array with a null value initially.**

**byte[] data = null;**

**//Use FileInfo object to get file size.**

**FileInfo fInfo = new FileInfo(sPath);**

**long numBytes = fInfo.Length;**

**//Open FileStream to read file**

**FileStream fStream = new FileStream(sPath, FileMode.Open,**

**FileAccess.Read);**

**//Use BinaryReader to read file stream into byte array.**

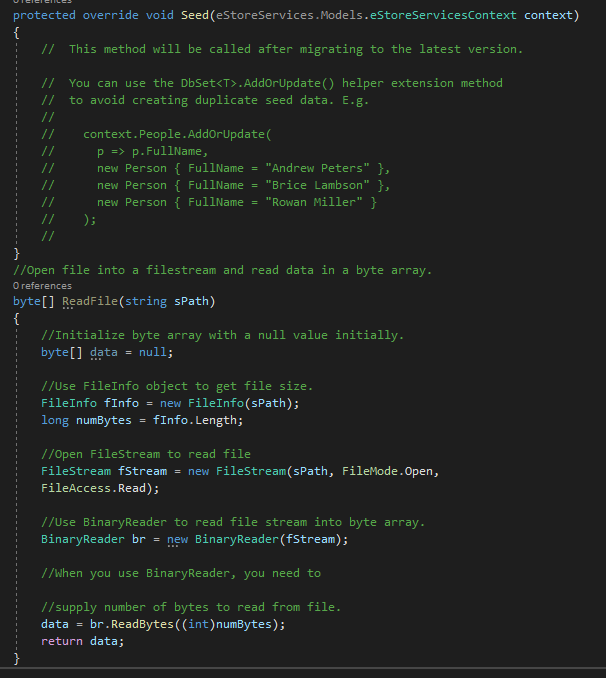
**BinaryReader br = new BinaryReader(fStream);**

**//When you use BinaryReader, you need to**

**//supply number of bytes to read from file.**

**data = breakbeats((int)numBytes);**

**return data;**

**}**

The above codes will attempt to read the image files and convert them to Base64 string. In time to come, we will be building a Web Client that will use javascript to read an image and convert it to a Base64 String and send to the web api.

1. Replace the code in the **Seed** method. This will create and add 12 records to the database table, including reading the image file and storing the binary into the table.

protected override void Seed(eStoreServices.Models.eStoreServicesContext context)

{

context.Products.AddOrUpdate(x => x.BookID,

new Product() { BookID = 1, Title = "ASP.NET Web API 2 Recipes: A Problem-Solution Approach 1st ed. Edition", Authors = "Filip Wojcieszyn", CoverImage = ReadFile("C:\\Images\\978-1430259800.jpg") },

new Product() { BookID = 2, Title = "ASP.NET Web API 2: Building a REST Service from Start to Finish 2nd ed. Edition", Authors = "Jamie Kurtz", CoverImage = ReadFile("C:\\Images\\978-1484201107.jpg")},

new Product() { BookID = 3, Title = "Expert ASP.NET Web API 2 for MVC Developers 1st ed. Edition", Authors = "Adam Freeman", CoverImage = ReadFile("C:\\Images\\978-1484200865.jpg")},

new Product() { BookID = 4, Title = "Pro ASP.NET Web API Security: Securing ASP.NET Web API (Expert's Voice in .NET) 1st ed. Edition", Authors = "Badrinarayanan Lakshhmiraghavan", CoverImage = ReadFile("C:\\Images\\978-1430257820.jpg")},

new Product() { BookID = 5, Title = "RESTful Web Services Cookbook: Solutions for Improving Scalability and Simplicity 1st Edition", Authors = "Subbu Allamaraju", CoverImage = ReadFile("C:\\Images\\978-0596801687.jpg") },

new Product() { BookID = 6, Title = "Programming Entity Framework: Code First 1st Edition", Authors = "Julia Lerman, Rowan Miller", CoverImage = ReadFile("C:\\Images\\978-1449312947.jpg") },

new Product() { BookID = 7, Title = "Practical ASP.NET Web API 1st ed. Edition", Authors = "Badrinarayanan Lakshhmiraghavan", CoverImage = ReadFile("C:\\Images\\978-1430261759.jpg") },

new Product() { BookID = 8, Title = "Pro ASP.NET Web API: HTTP Web Services in ASP.NET (Expert's Voice in .NET) 1st ed. Edition", Authors = "Ali Uurlu, Alexander Zeitler, Ali Kheyrollahi", CoverImage = ReadFile("C:\\Images\\978-1430247258.jpg")},

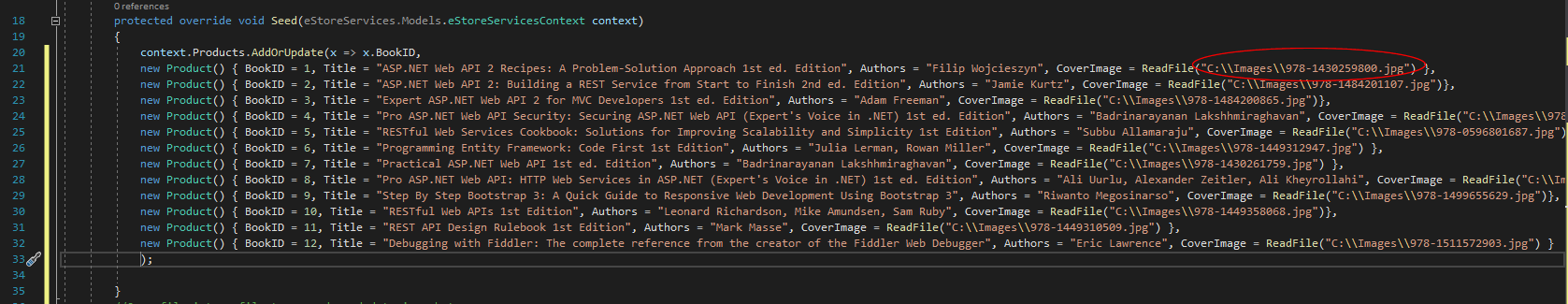
new Product() { BookID = 9, Title = "Step By Step Bootstrap 3: A Quick Guide to Responsive Web Development Using Bootstrap 3", Authors = "Riwanto Megosinarso", CoverImage = ReadFile("C:\\Images\\978-1499655629.jpg")},

new Product() { BookID = 10, Title = "RESTful Web APIs 1st Edition", Authors = "Leonard Richardson, Mike Amundsen, Sam Ruby", CoverImage = ReadFile("C:\\Images\\978-1449358068.jpg")},

new Product() { BookID = 11, Title = "REST API Design Rulebook 1st Edition", Authors = "Mark Masse", CoverImage = ReadFile("C:\\Images\\978-1449310509.jpg") },

new Product() { BookID = 12, Title = "Debugging with Fiddler: The complete reference from the creator of the Fiddler Web Debugger", Authors = "Eric Lawrence", CoverImage = ReadFile("C:\\Images\\978-1511572903.jpg") }

);

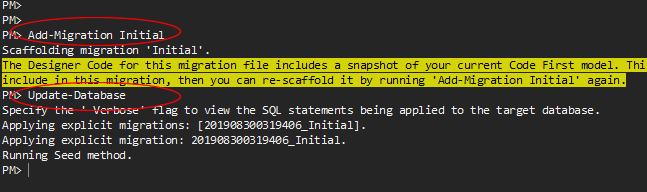
}****

1. In the **Package Manager Console** window, type the following commands:

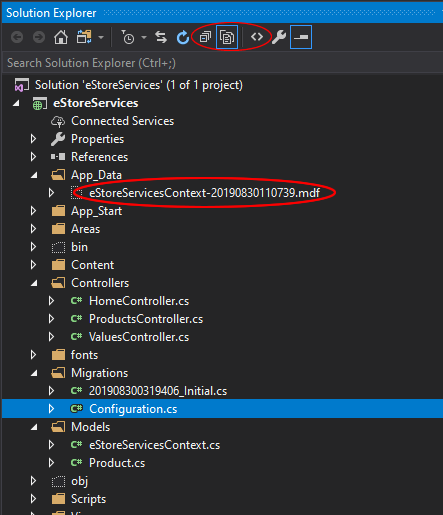
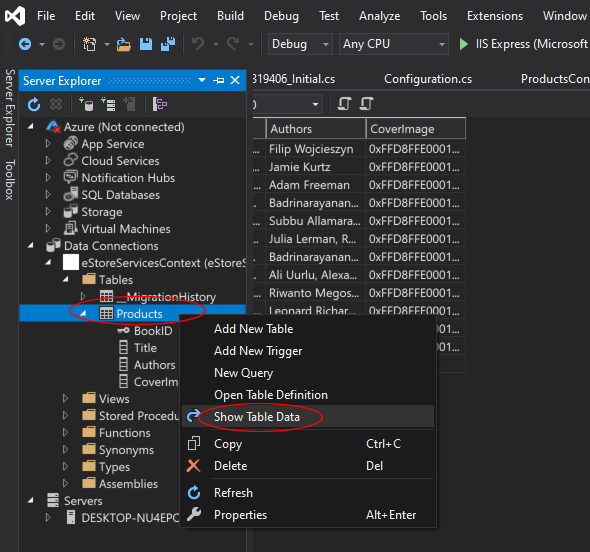
**Add-Migration Initial**

**Update-Database**

The first command generates code that creates the database (refer to the content of **\****numbers indicating the datetime***\_Initial.cs** file in the **Migrations** folder), and the second command executes that code. The database is created locally with **SQL Express**, using [**LocalDB**](http://msdn.microsoft.com/en-us/library/hh510202.aspx).



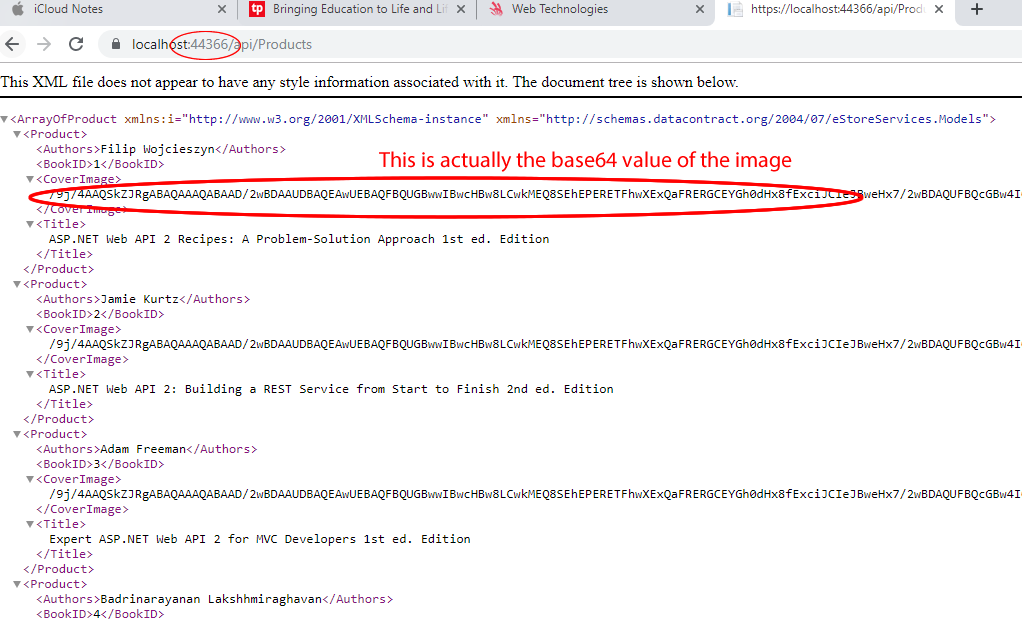
1. The database table **Product** will have 12 records/rows. Remember the primary key is **BookID**. You can inspect the table by following the below steps.

**

### Task 2: Explore the Web API

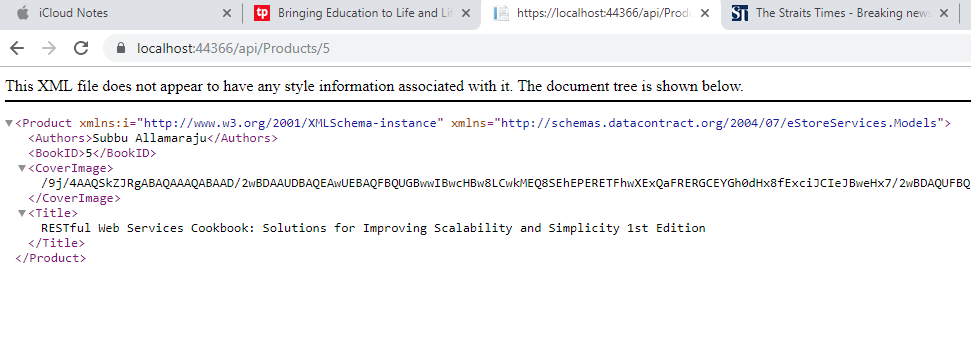
1. Press **F5** to run the application in debug mode. **Visual Studio** starts **IIS Express** and runs your web app. **Visual Studio** then launches the browser you have chosen and opens the app’s home page.

When **IIS Express** runs a web project, it assigns a **port number**. In the image below, the port number is 44366. When you run the application, **you will see a different port number**.

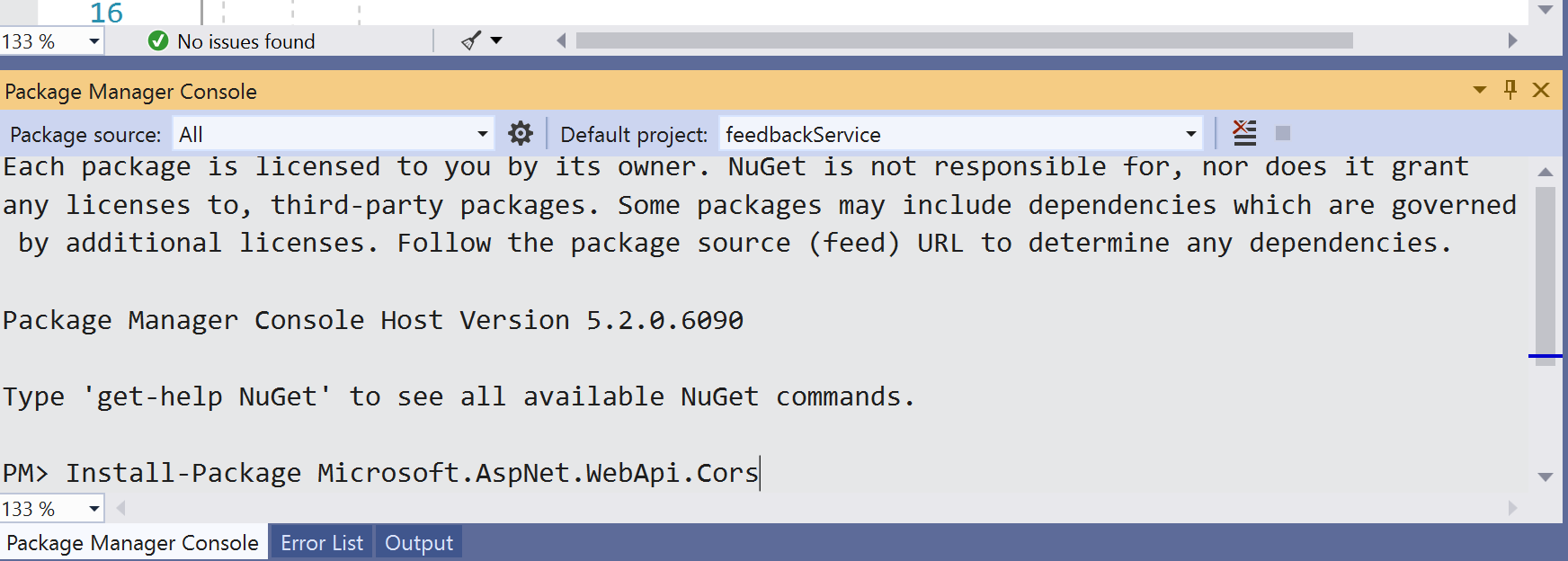


The **API** enables CRUD operations on the database. The following table summarizes the API.

|  |  |
| --- | --- |
| **GET: /api/products** | **Get all products.** |
| **GET: /api/products/{id}** | **Get a product by ID.** |
| **POST: /api/products** | **Create a new product.** |
| **PUT: /api/products/{id}** | **Update an existing product.** |
| **DELETE: /api/products/{id}** | **Delete a product.** |

Try with the different entries in the URL and use different browsers.  


In order for our web api to be able to be consumed by any web client from any domain (Our case will be a web client running locally from our windows file system), we need to set our web api to implement cross origin resource sharing (CORS)

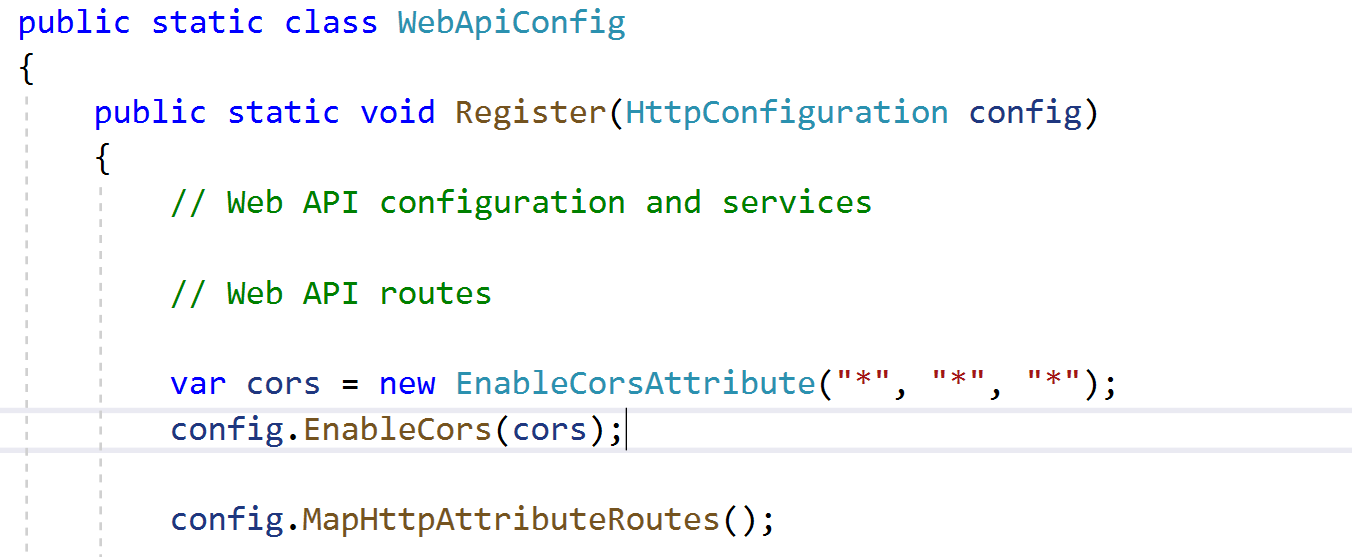


Activate the Package Manager Console and execute the command:  
Install-Package Microsoft.AspNet.WebApi.Cors

Add the following codes to WebApiConfig.cs

var cors = new EnableCorsAttribute("\*", "\*", "\*");

config.EnableCors(cors);



Now your web api is freely available to any web client from any domain.

In the next lesson we will be exploring the use of POSTMAN to test out the different WEB API end-points

\*\* End \*\*