

Enterprise Software Architecture and Design

A Full Stack WorkShop Project

By

Jean Pierre Deffo Fotso

Software Engineer and Architect

Ingegnere in Sistema Multimediale e Telematica

Email : jeanpierre.deffofotso@blucrm.com

Tel: 3889400697

Linkedin : <https://www.linkedin.com/in/jean-pierre-deffo-fotso-602ab422/>

Twitter: <https://twitter.com/defcoq73>

Medium: <https://medium.com/@jpdeffo>

Github: <https://github.com/Defcoq>

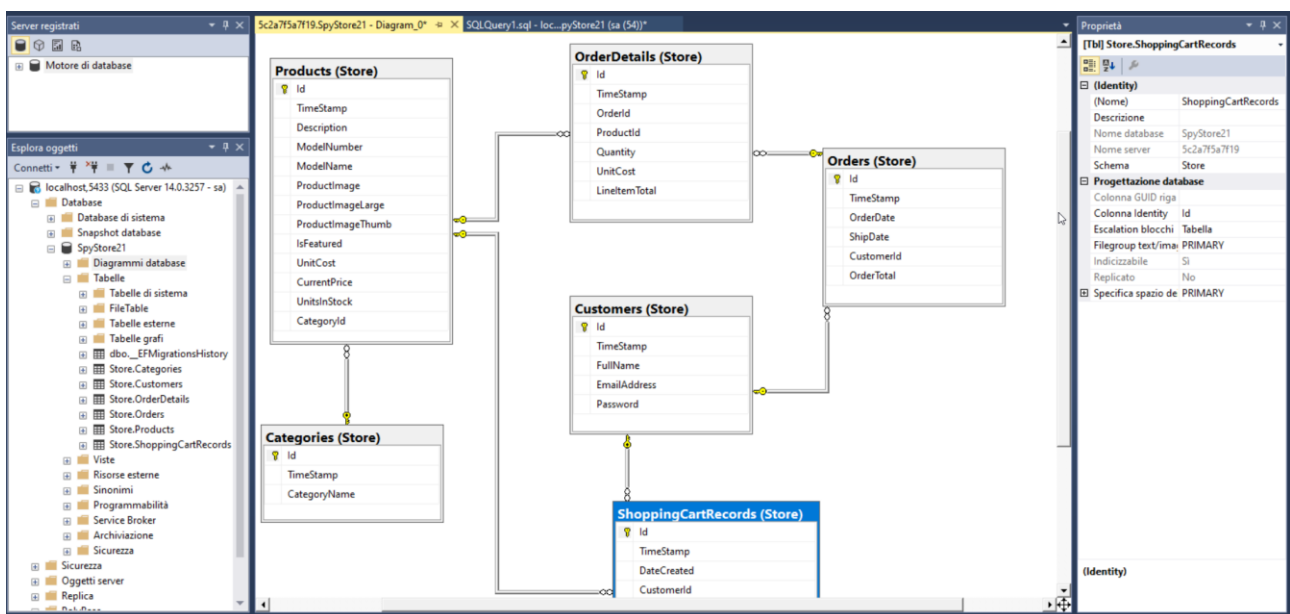
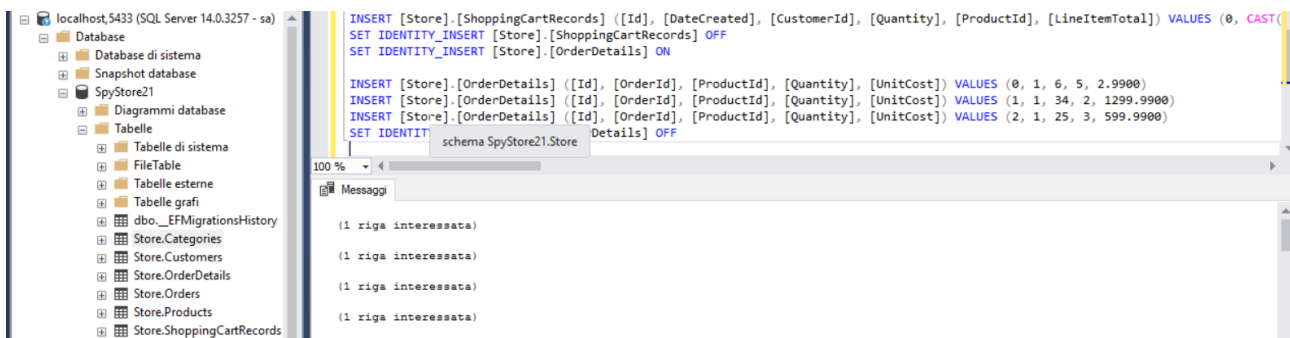
Facebook: <https://www.facebook.com/profile.php?id=100011294258548>

Sample demonstration project for a workshop session to UNISI
(Università di Siena, facoltà di Ingegneria informatica)



Nowdays, the way enterprise software solution are build is based on the use of best practices and design pattern, the choice of technologies and framework to facilitate best integration with existing software, testability and last but not last a great and easy deployment process (this is where DevOps methodology is useful). We have to not repeat ourselves by using existing libraries and framework that feel our needs and that can simplify our development time, we have to take the best tecnology base on our design and the knowledge of our domain model.

In this workshop, we will be building an **e-commerce system** with a domain model consist of few entities as you can get from the image bellow (The image is the diagram of tables and their relationship generate by Entityframework(EF) code first), the database EF generate is called “**SpyStore**”



As you can get from the image above, our domain model will be consist only of six tables.

Why full stack?

FullStack mean that our project will be consisted of two different part with different technologies and framework because a software architecture for enterprise is always base on a classic **client-server** relationship where client application (web, desktop, mobile ecc..) need always data by communicating with a server application wich have tho opportunity to talk with a database store to get data, the server side can be a REST application or a SOAP endpoint (in .Net ecosystem asmx web service and WCF base on SOAP protocol), or it can be a simple web server like an apache or IIS web site. Client side part and server side part can be very complex and all dipend of what we are trying to build. For a great design, the server side can be very very complex and the need of a Microservice architecture can be a best choice so that we can be able to use different framework and technologies in server side to get our job done. The workshop sample project we are building will have the REST endpoint and database part running inside a docker container (you can also run it without docker if we need) so our achitecture will have in mine some idea of a microservice cluster since from the beginning. So Full stak solution mean **Frontend + backend solution** together, we have many option when building a full stak solution like using “**MEAN(Mongo-Express-Angular-Node)**” Stack or **MERN(Mongo-Express-REACT-Node)** stack, in this case Mongo is using as database store, Angular or React as client side framework or platform while Node is be using as a backend framework to build server artifacts. The same thing can be build with other language and framework like java, python ecc.. but our sample project will be based on Microsoft ecosystem for server side development, we will be used **dotnet core** for his simplicity, cross platform and easy deployment nature.

Before going any more in deep to explain how the whole project is organized, you have to know the **S.O.L.I.D Design Principles** to be able to build and know how to layer an enterprise software solution, this design as you'll see is base of the OOP(Object Oriented Programming).

The S.O.L.I.D. Design Principles

The S.O.L.I.D. design principles are a collection of best practices for object-oriented design. All of the Gang of Four design patterns adhere to these principles in one form or another. The term **S.O.L.I.D.** comes from the initial letter of each of the five principles that were collected in the book ***Agile Principles, Patterns, and Practices in C#*** by **Robert C. Martin**. The following sections look at each one in turn.

Single Responsibility Principle (SRP)

The principle of SRP is closely aligned with SoC. It states that every object should only have one reason to change and a single focus of responsibility. By adhering to this principle, you avoid the problem of monolithic class design that is the software equivalent of a Swiss army knife. By having concise objects, you again increase the readability and maintenance of a system.

Open-Closed Principle (OCP)

The OCP states that classes should be open for extension and closed for modification, in that you should be able to add new features and extend a class without changing its internal behavior. The principle strives to avoid breaking the existing class and other classes that depend on it, which would create a ripple effect of bugs and errors throughout your application.

Liskov Substitution Principle (LSP)

The LSP dictates that you should be able to use any derived class in place of a parent class and have it behave in the same manner without modification. This principle is in line with OCP in that it ensures that a derived class does not affect the behavior of a parent class, or, put another way, derived classes must be substitutable for their base classes.

Interface Segregation Principle (ISP)

The ISP is all about splitting the methods of a contract into groups of responsibility and assigning interfaces to these groups to prevent a client from needing to implement one large interface and a host of methods that they do not use. The purpose behind this is so that classes wanting to use the same interfaces only need to implement a specific set of methods as opposed to a monolithic interface of methods.

Dependency Inversion Principle (DIP)

The DIP is all about isolating your classes from concrete implementations and having them depend on abstract classes or interfaces. It promotes the mantra of coding to an interface rather than an implementation, which increases flexibility within a system by ensuring you are not tightly coupled to one implementation. Dependency Injection (DI) and Inversion of Control (IoC) Closely linked to the DIP are the DI principle and the IOC principle. DI is the act of supplying a low level or dependent class via a constructor, method, or property. Used in conjunction with DI, these dependent classes can be inverted to interfaces or abstract classes that will lead to loosely coupled systems that are highly testable and easy to change. In IoC, a system's flow of control is inverted compared to procedural programming. An example of this is an IoC container, whose purpose is to inject services into client code without having the client code specifying the concrete implementation. The control in this instance that is being inverted is the act of the client obtaining the service.

All the SOLID Design is apply in the sample project (see the full source code to great detail),for example dotnet core have a built in IoC container so our service and repository will be automatically inject for us where we need without having to explicitly intanciated it, this is very great to have many implementation of the same behavior and use it in different way like in our test project as we will see in this guide.

The backend server side is Base on DDD (Domain Driven Design)

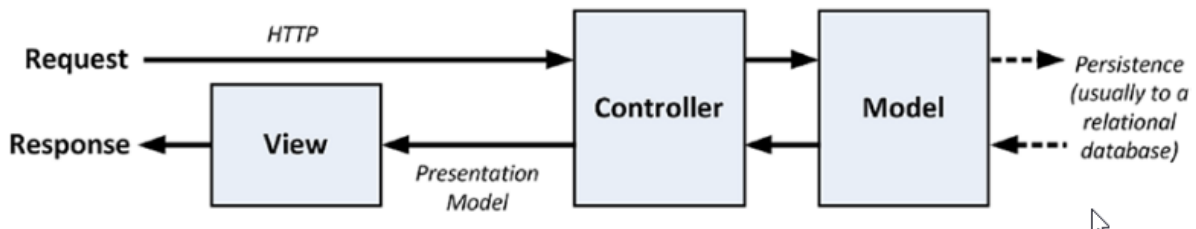
Domain-driven Design (DDD)

In a nutshell, DDD is a collection of patterns and principles that aid in your efforts to build applications that reflect an understanding of and meet the requirements of your business. Outside of that, it's a whole new way of thinking about your development methodology. DDD is about modeling the real domain by fully understanding it first and then placing all the terminology, rules, and logic into an abstract representation within your code, typically in the form of a domain model. DDD is not a framework, but it does have a set of building blocks or concepts that you can incorporate into your solution.

You can read my post here(<https://medium.com/@jpdeffo/domain-driven-design-ddd-in-microservice-architecture-for-nutshell-19c7c579009a>) medium.com chanel where I'm talking about DDD with a great dotnet core example.

Another pattern wich we need to explain before begin anything is MVC (**Model-View-Controller**) as it is use either in frondend part(Angular and React) and backend part for building REST endpoint.

Understanding the MVC Pattern



The term model-view-controller has been in use since the late 1970s and arose from the Smalltalk project at Xerox PARC, where it was conceived as a way to organize some early GUI applications. Some of the fine detail of the original MVC pattern was tied to Smalltalk-specific concepts, such as screens and tools, but the broader concepts are still applicable to applications, and they are especially well-suited to web applications.

In high-level terms, the MVC pattern means that an MVC application will be split into at least three pieces.

- **Models**, which contain or represent the data that users work with
- **Views**, which are used to render some part of the model as a user interface

- **Controllers**, which process incoming requests, perform operations on the model, and select views to render to the user

Each piece of the MVC architecture is well-defined and self-contained, which is referred to as the separation of concerns. The logic that manipulates the data in the model is contained only in the model, the logic that displays data is only in the view, and the code that handles user requests and input is contained only in the controller. With a clear division between each of the pieces, your application will be easier to maintain and extend over its lifetime, no matter how large it becomes.

Understanding Models

Models—the M in MVC—contain the data that users work with. There are two broad types of model: view models, which represent just data passed from the controller to the view, and domain models, which contain the data in a business domain, along with the operations, transformations, and rules for creating, storing, and manipulating that data, collectively referred to as the model logic.

Models are the definition of the universe your application works in. In our sample e-commerce workshop project, for

example, the model represents everything in the e-commerce that the application supports, such as customers, product, category, shopping cart and so on, as well as the operations that can be used to manipulate the data in the model, such as adding and removing product in shopping cart.

The model is also responsible for preserving the overall state and consistency of the data—for example, making sure that before add a product to shopping cart, verify that there are someone in the stock.

The model in an application built using the MVC pattern should

- Contain the domain data
- Contain the logic for creating, managing, and modifying the domain data

- Provide a clean API that exposes the model data and operations on it

The model should not

- Expose details of how the model data is obtained or managed (in other words, details of the data storage mechanism should not be exposed to controllers and views)

- Contain logic that transforms the model based on user interaction (because that is the controller's job)

- Contain logic for displaying data to the user (that is the view's job)

The benefits of ensuring that the model is isolated from the controller and views are that you can test your logic more easily (In .net ecosystem, we can use unittest project as we are going to see in this guide) and that enhancing and maintaining the overall application is simpler and easier.

Understanding Controllers

Controllers are the connective tissue in the MVC pattern, acting as conduits between the data model and views. Controllers define actions that provide the business logic that operates on the data model and that provide the data that views display to the user.

A controller built using the MVC pattern should

- Contain the actions required to update the model based on user interaction

The controller should not

- Contain logic that manages the appearance of data (that is the job of the view)

- Contain logic that manages the persistence of data (that is the job of the model)

Understanding Views

Views contain the logic required to display data to the user or to capture data from the user so that it can be processed by a controller action.

Views should

- Contain the logic and markup required to present data to the user

Views should not

- Contain complex logic (this is better placed in a controller)

- Contain logic that creates, stores, or manipulates the domain model

Views can contain logic, but it should be simple and used sparingly.

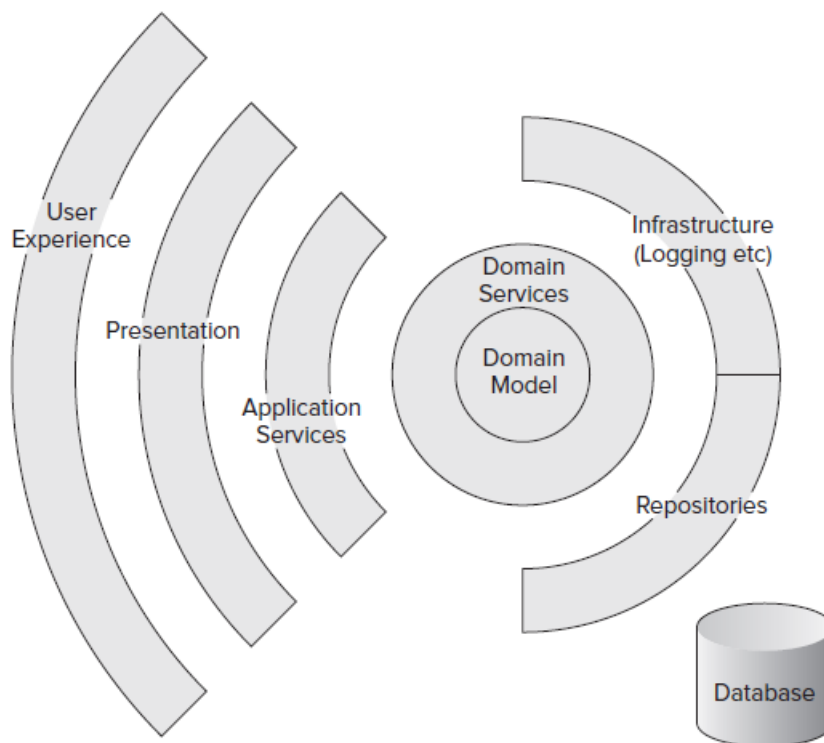
Putting anything but the simplest method calls or expressions in a view makes the overall application harder to test and maintain.

Layering Your application and separating Your Concerns

You cannot build a maintainable and scalable application on poor foundations. Planning a good architecture is critical to the success of an application. Before examining a structured approach to designing your application, you must learn why you need to think about the logical structure of your application and the problems you will encounter if you do not start with a good architectural footing.

Layering an application

is a form of separation of concerns and can be achieved via namespaces, folders, or with separate projects as you can see in the image below where we divide our application in different project so that the SoCis apply, and in our e-commerce project, we divide the solution in different sub project we can be run side by side our standalone



Fontend Framework and Technologies For the workshop Project

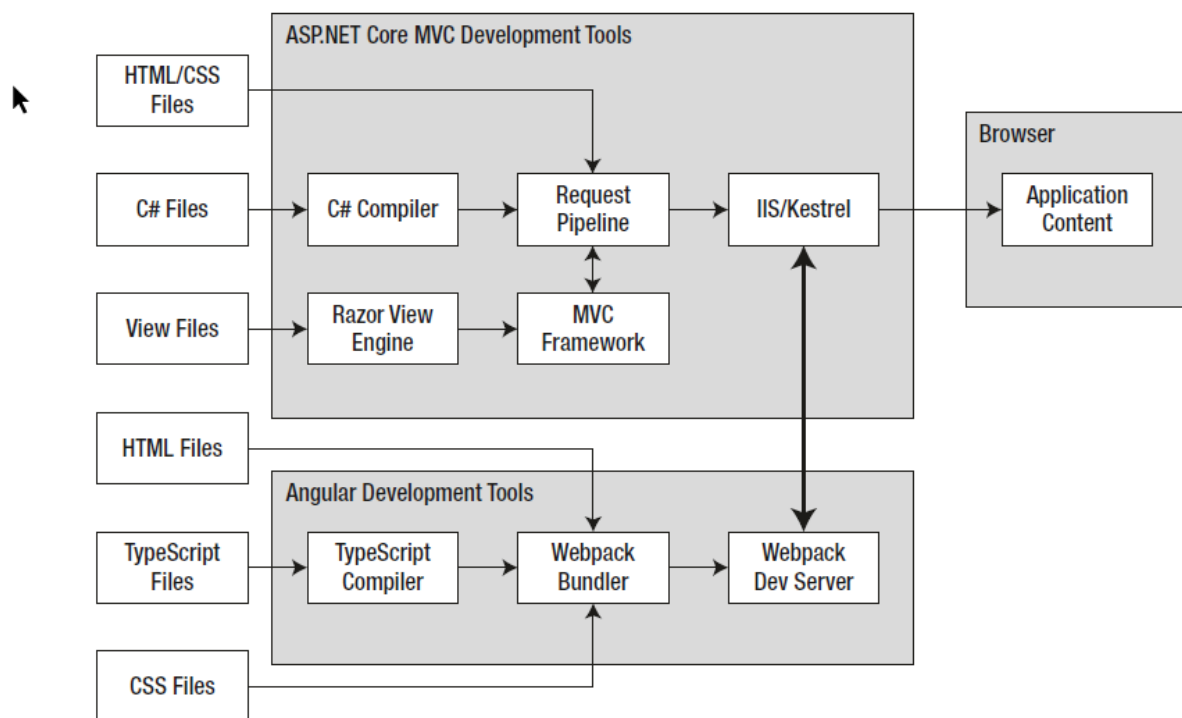
The workshop is composed of 3 clients side application building with 3 different framework, all three client are web base because nowday, 90% of entreprise software have a web client interface because of the fact that for this type of client application, we usually not need to install

anything on the client machine, and client side framework like Angular can be also use to build both desktop app and mobile app.

Angular Client Frontend

In this part of the workshop, we will be demonstrating and talking about :

- What and why Angular for frontend developement
- Angular as MVC framework
- Angular is no just a framework or library but it's a wall platform
- Angular component, service
- Angular DI (Dipendence injection)
- Angular with typescript integration, what is typescript and why typescript is usefull for angular development?
- Angular developement tool (nodejs, webpark systemjs ecc..)
- PWA (Progressive Web App): what is ? and why it's very important today
- Angular Routing
- What is a single page application
- Angular and visual studio toolchain together



React Client Frontend

React like Angular for this workshop is based on Node.js and TypeScript to facilitate the development so we will demonstrate and talk about:

- What and why React ?
- React Component
- React Virtual DOM
- How to create a React project from command line
- Manage state with React
- React Routing
- TypeScript, Node.js, SPA , PWA as we have talked with Angular etc...

Asp.net Core MVC web Client Frontend (based on Razor view engine)

Here we will demonstrate a server side rendering client view using natively in Microsoft ecosystem

- What and why to use Asp.net MVC with Razor view engine ?
- Razor is not Blazor what is the difference?
- Razor View component
- Razor Taghelper
- How asp.net core MVC web app is built ?
- What are controller, action method.
- Asp.net MVC Routing system
- Convention over configuration in asp.net core MVC

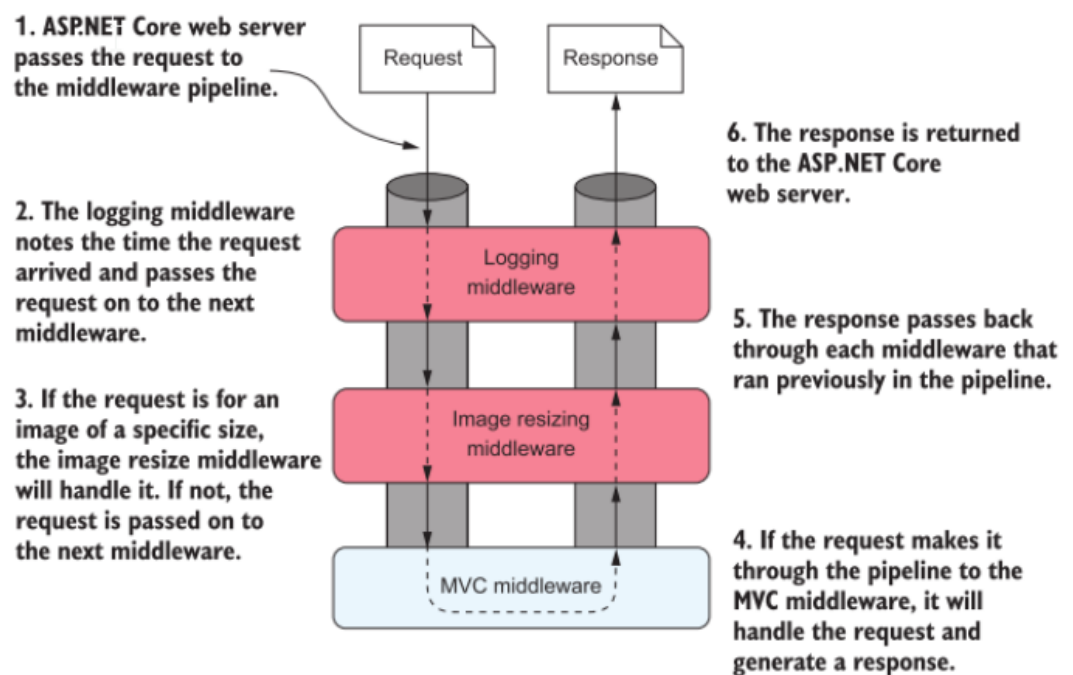
Server Side Framework and Technologies

The server side backend part is a huge topic because many players come in place. In the workshop sample project, we are using Microsoft .NET Core framework and tools to build all our backend application server, so we will talk and demonstrate this topic:

Asp.net core MVC

- What and why asp.net core framework?

- Advantage and disadvantage of asp.net core framework
- Cross platform solution
- Easy deployment by using container base system
- Asp.net core MVC dependence injection
- The lightweight web server : Kestrel vs IIS
- Asp.net core as a modular framework: the concept of asp.net core MVC middleway



- Asp.net core Routing
- Aspnet core Controller and action method
- Aspnet core filter
- Aspetnet core Model Binding
- Authentication and Authorisation in asp.net core with asp.net core identity

Asp.net core MVC ORM(Object Relational Mapping) : Entity Framework (EF)

- Why the need of an ORM?

- What EF and why we need it to save and retrieve data?
- Convention over configuration to mapped our domain model entities to database table with with asp.net core EF Fluent API
- Code First and Database First approach
- The role of LINQ (Language integrate Query) in dot.net core

Asp.net core Unittesting

- What it's unittest and why is so important (TDD)?
- How to use visual studio or dotnet command line tool to unittest our project

Deployment options (DevOps)

- How to deploy asp.net core project in production
- Docker container explain
- Microservice explain
- Orchestrator solution explain

Tools we need to run the sample applications:

To run the workshop sample application project, you need to install in your development enviroment the following tools:

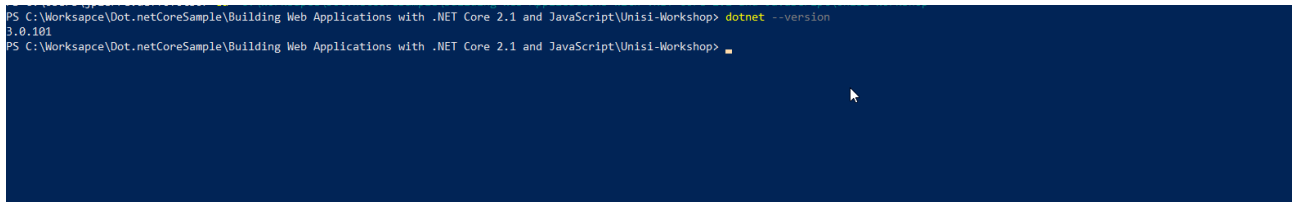
Installing .NET Core

The .NET Core Software Development Kit includes the runtime and development tools needed to start the development project and perform database operations. To install the .NET Core SDK on Windows, download the installer from <https://dotnet.microsoft.com/download/thank-you/dotnet-sdk-3.0.100-windows-x64-installer> . This URL is for the 64-bit .NET Core SDK version 3.0.0 (you can also instal the last major release),

which is the version that I use throughout this workshop and that you should install to ensure that you get the expected results from the examples. Rather than type in a complex URL, you can go to <https://www.microsoft.com/net/download/core> and select the 64-bit installer for the .NET Core SDK. (Microsoft also publishes a runtime-only installer, but this does not contain the tools that are required for this workshop.)

Run the installer; once the install process is complete, open a new PowerShell command prompt and run the command and you'll get the following result :

```
dotnet --version
```

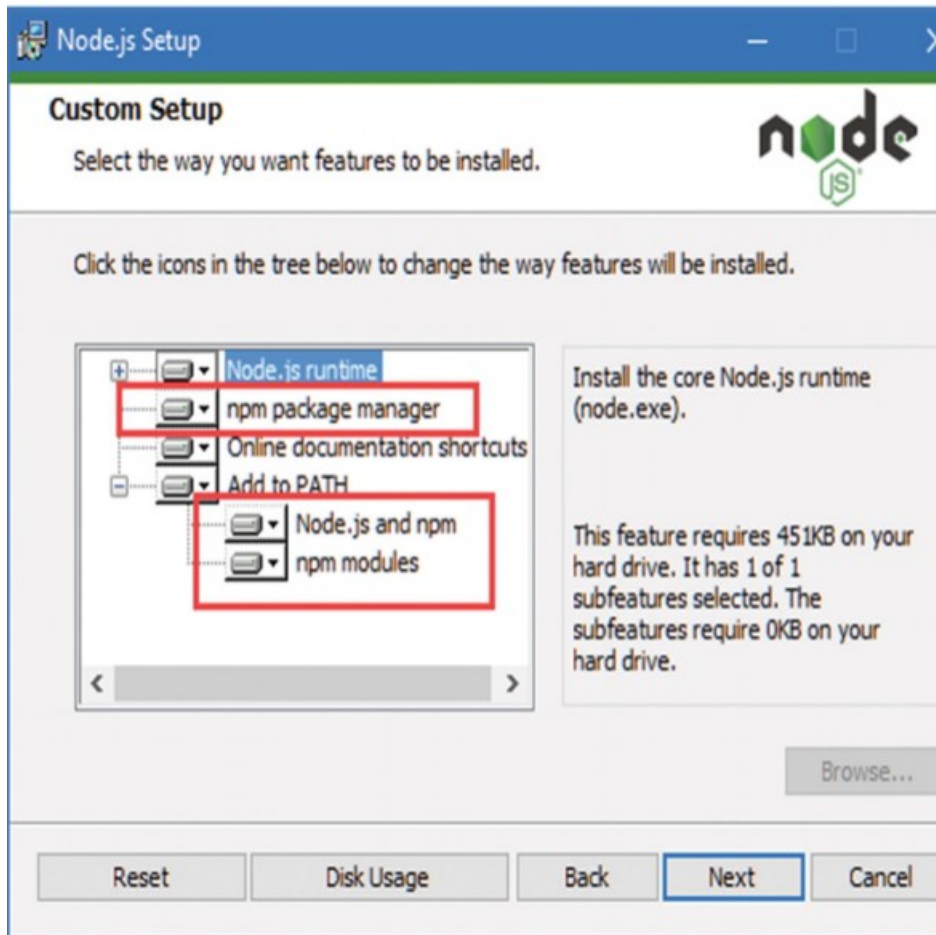


```
PS C:\Workspace\Dot.netCoreSample\Building Web Applications with .NET Core 2.1 and JavaScript\Unisi-Workshop> dotnet --version
3.0.101
PS C:\Workspace\Dot.netCoreSample\Building Web Applications with .NET Core 2.1 and JavaScript\Unisi-Workshop>
```

Installing Node.js

Node.js is a runtime for server-side JavaScript applications and has become a popular platform for development tools. In this workshop, Node.js is used by the Angular and react build tools to compile and prepare the code that ASP.NET Core MVC will send to the browser.

It is important that you download the same version of Node.js that I use throughout this sample project. Although Node.js is relatively stable, there are still breaking API changes from time to time, and they may stop the examples from working. To install Node.js, download and run the installer from <https://nodejs.org/dist/v12.4.0/node-v12.4.0-x64.msi> . This is the installer for version 12.4.0. You may prefer more recent releases for your projects, but you should stick with the 12.4.0 release for the rest of this book. Run the installer and ensure that the “npm package manager” option and the two Add to PATH options are selected, as shown in the figure below:



The NPM package manager is used to download and install Node packages. Adding Node.js to the PATH ensures that you can use the Node.js runtime at the command prompt just by typing node. Once installation is complete, open a new command prompt and run the command shown below:

```
node -v
```

```
PS C:\Workspacce\Dot.netCoreSample\Building Web Applications with .NET Core 2.1 and JavaScript\Unisi-Workshop> node -v
v10.15.3
PS C:\Workspacce\Dot.netCoreSample\Building Web Applications with .NET Core 2.1 and JavaScript\Unisi-Workshop>
```

Installing Git

Download and run the installer from <https://git-scm.com/downloads> . When the installation is complete, open a new command prompt and run the command in Listing below to check that Git is installed and working properly.

```
git --version
```

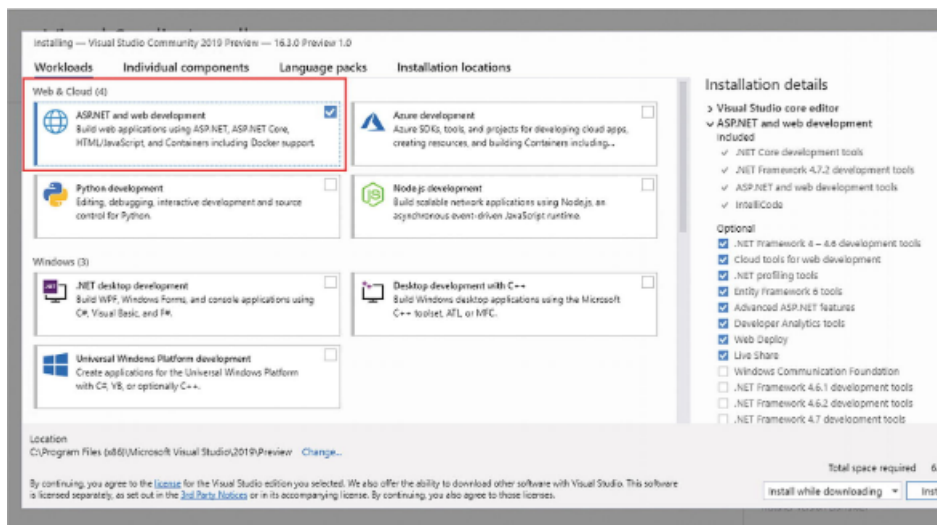
```
PS C:\Workspac\Dot.netCoreSample\Building Web Applications with .NET Core 2.1 and JavaScript\Unisi-Workshop> git --version
git version 2.17.1.windows.2
PS C:\Workspac\Dot.netCoreSample\Building Web Applications with .NET Core 2.1 and JavaScript\Unisi-Workshop>
```

Installing Visual Studio 2019

Visual Studio is the traditional development environment for ASP.NET Core and Entity Framework Core projects. It offers a full-featured development experience, but it can be resource hungry. Consider using Visual Studio Code, described in the next section, if you want a lighter-weight development experience.

Download the installer from <https://www.visualstudio.com/vs>. There are different editions of Visual Studio available, but the free Community edition is sufficient for the examples in this demo.

Run the installer and ensure that the **“ASP.NET and web development”** workload is selected, as shown in Figure below. This workload contains all the Visual Studio features required for this workshop.



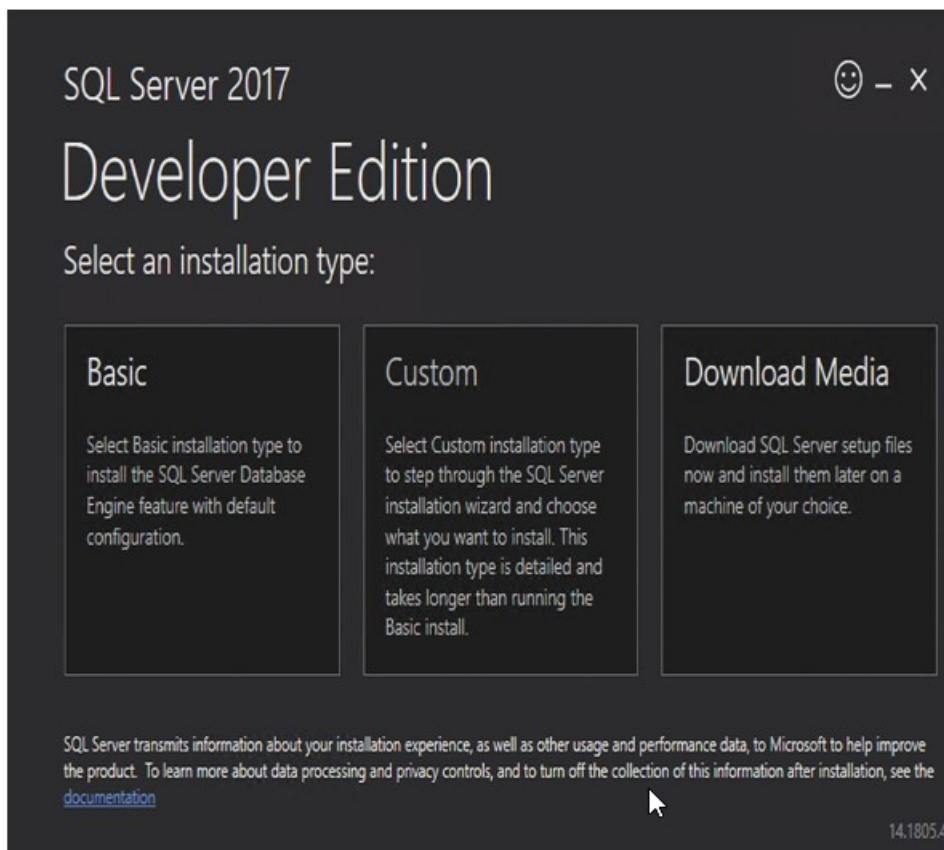
Click the Install button to begin the process of downloading and installing the Visual Studio features.

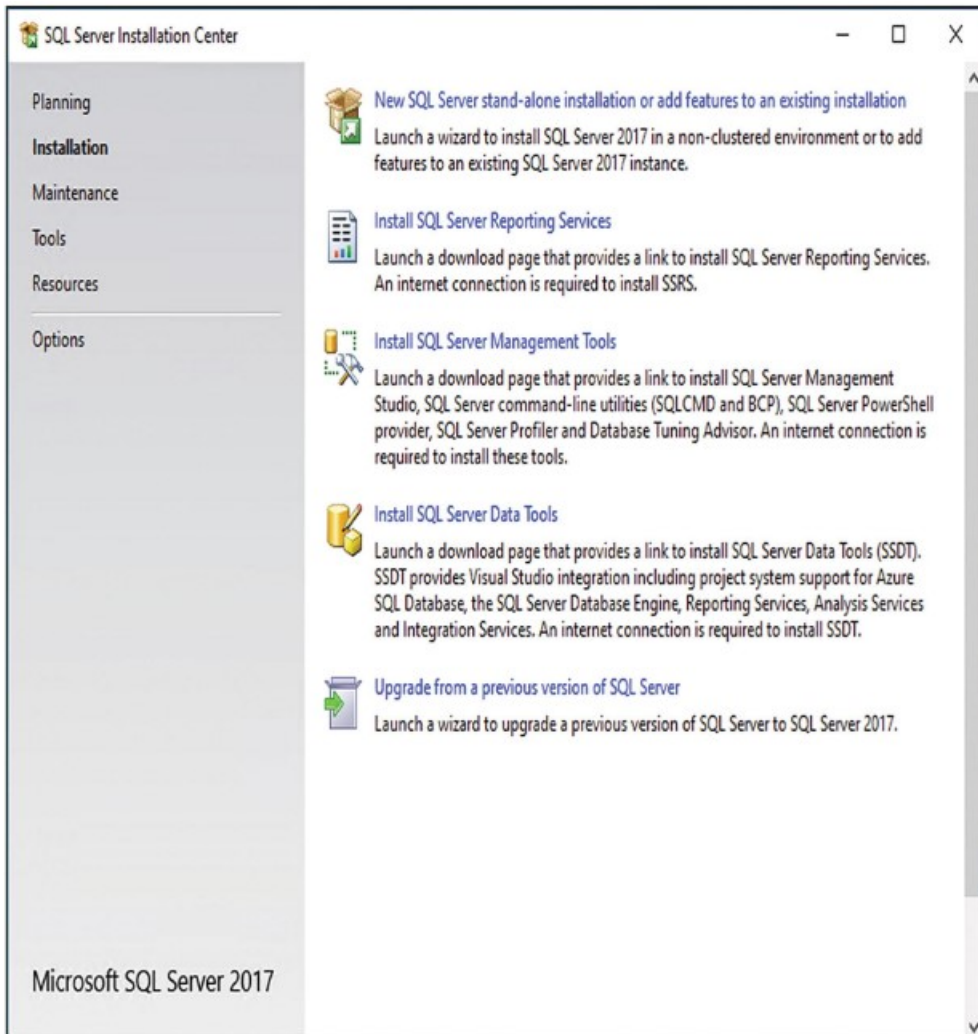
Installing Visual Studio Code

Visual Studio Code is a light-weight editor that doesn't have all the features of the full Visual Studio product; however, it works across platforms and is perfectly capable of handling ASP.NET Core and Angular development and can be used for the examples in this book.

To install Visual Studio code, visit <http://code.visualstudio.com> and click the download link for Windows. Run the installer and then start Visual Studio Code.

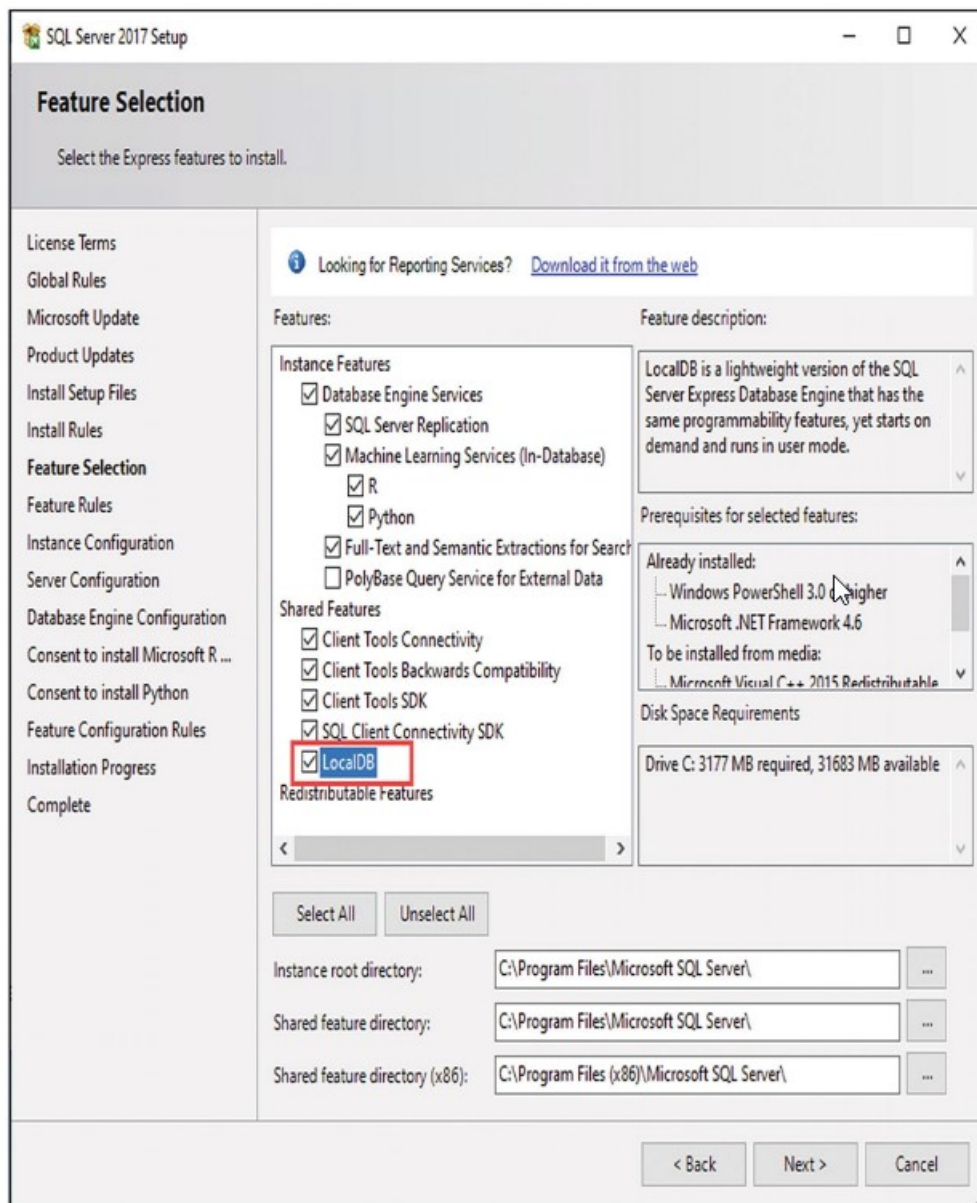
The database examples in this workshop also can be run inside a LocalDB (we will run it on a docker container), which is a zero-configuration version of SQL Server and which can be installed as part of the SQL Server Express edition, which is available for use without charge from <https://www.microsoft.com/en-in/sql-server/sql-server-downloads> . Download and run the Express edition installer and select the Custom option, as shown in Figure:



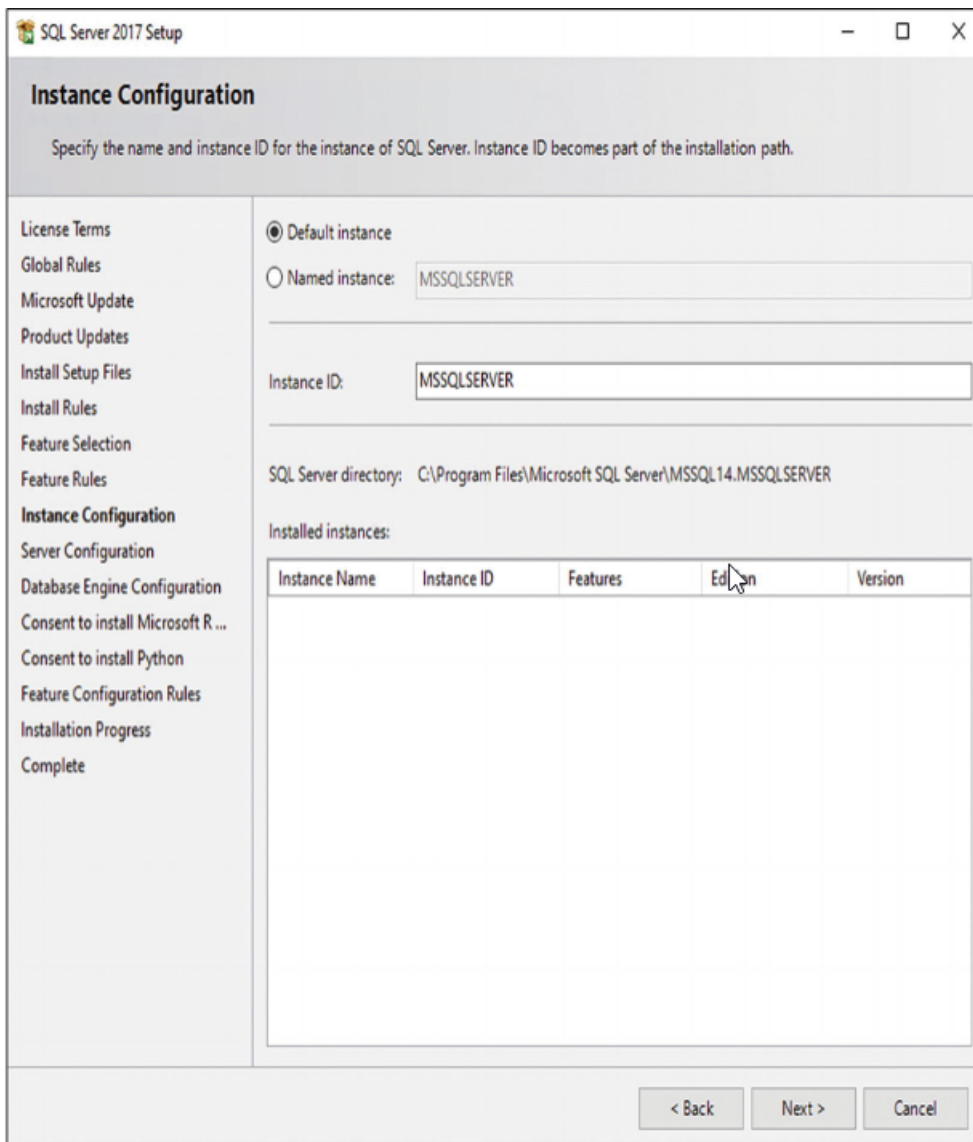


This will also install the sql server Management tool so you can view all the database object generate by entity framework and you can also run custom Tansact SQL query to view data.

Work through the installation process, selecting the default options as they are presented. When you reach the Feature Selection page, ensure that the LocalDB option is selected, as shown in Figure



On the Instance Configuration page, select the “Default instance” option, as shown in Figure



Continue to work through the installation process, selecting the default values. Once the installation is complete, install the latest cumulative update for SQL Server. At the time of writing, the latest update is available at <https://support.microsoft.com/en-gb/help/4498951/cumulative-update-15-for-sql-server-2017> , although newer updates—or newer versions of SQL Server—may have been released by the time you read this guide.

Installing Docker Desktop

Docker is an operating system (OS) virtualisation base on container technology, it's a great way to perfectly deploy and run our application, you can read more about Virtual Machine and Container base solution from my medium.com channel where I wrote many topic about container and microservices (<https://medium.com/@jpdeffo>).

To install docker, follow the this link : <https://docs.docker.com/docker-for-windows/install/> (you have to enable HyperV on windows)

Installing Angular Globally

To install angular globally on your local machine development, you need NPM (Node Packet Manangement), you have installed Node before so it's come natively with npm tool, so open a powershell command line tool and fire the command bellow to install Angular packages to your local machine:

```
npm install --global @angular/cli@8.2.4
```

Installing React Globally

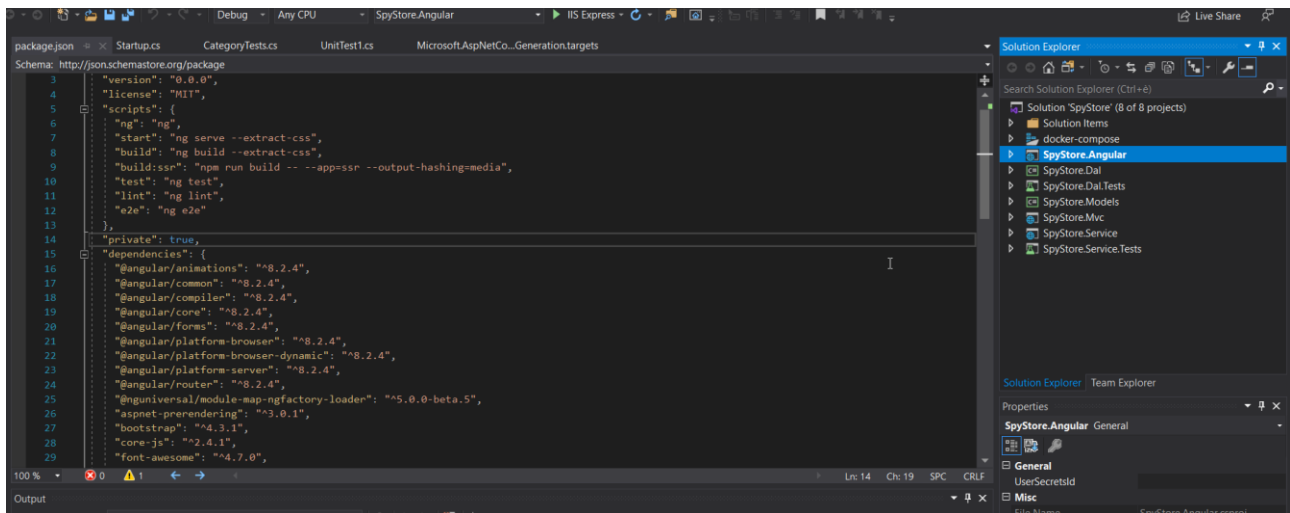
React as angular, provide a popular command line tool base on node.js to simplify React development process called **CreateReactApp**. You can install globally this tool in your local machine by running the command bellow from powershell:

```
npm install -g create-react-app
```

Sample e-commerce project organisation

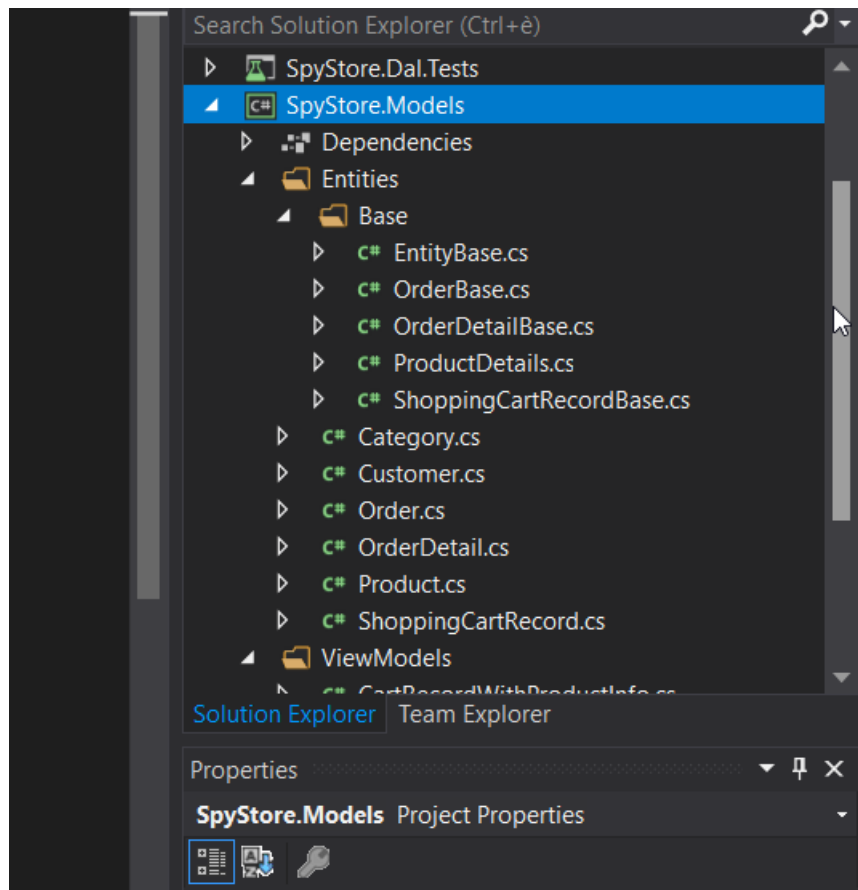
The following two images show the organisation folder and visual studio solution for the project, we are going to describe line by line what does every project

SpyStore.Angular	18/01/2020 16:49	Cartella di file	
SpyStore.Dal	23/12/2019 17:50	Cartella di file	
SpyStore.Dal.Tests	18/01/2020 16:26	Cartella di file	
SpyStore.Models	23/12/2019 17:48	Cartella di file	
SpyStore.Mvc	28/12/2019 22:56	Cartella di file	
SpyStore.Service	15/01/2020 21:01	Cartella di file	
SpyStore.Service.Tests	15/01/2020 20:35	Cartella di file	
spystore-react	18/01/2020 19:01	Cartella di file	
.dockerignore	25/12/2019 16:39	File DOCKERIGNO...	1 KB
.editorconfig	25/12/2019 14:45	File EDITORCONFIG	1 KB
CreateSolutionsAndProject.cmd	25/12/2019 21:53	Script di comandi ...	3 KB
databasDiagram.png	19/12/2019 11:36	File PNG	132 KB
dockercommand.txt	19/12/2019 11:18	Documento di testo	1 KB
docker-compose.dcpproj	25/12/2019 16:42	File DCPROJ	1 KB
docker-compose.override.yml	25/12/2019 16:42	File YML	1 KB
docker-compose.yml	25/12/2019 17:00	File YML	1 KB
Global.json	24/12/2019 15:32	JSON File	1 KB
scriptDatiDiEsempioSpyStore.sql	15/01/2020 21:17	Microsoft SQL Ser...	59 KB
scriptSpySporeConImaginiFinti.sql	15/01/2020 22:13	Microsoft SQL Ser...	58 KB
SpyStore.sln	29/12/2019 11:38	Visual Studio Solut...	10 KB
SpyStoreBluermReadme.docx	19/01/2020 18:51	Documento di Mic...	1.927 KB
utilityCommand.txt	19/12/2019 17:29	Documento di testo	2 KB



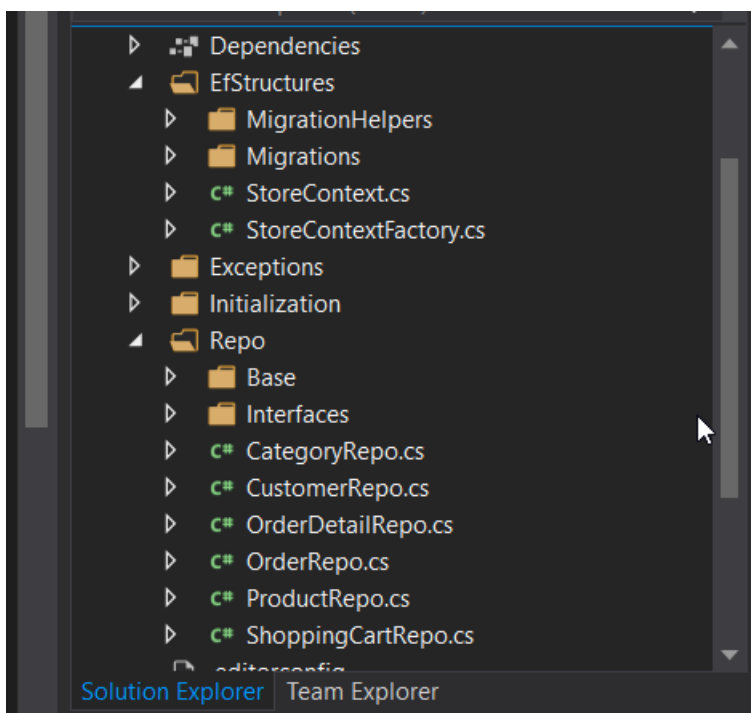
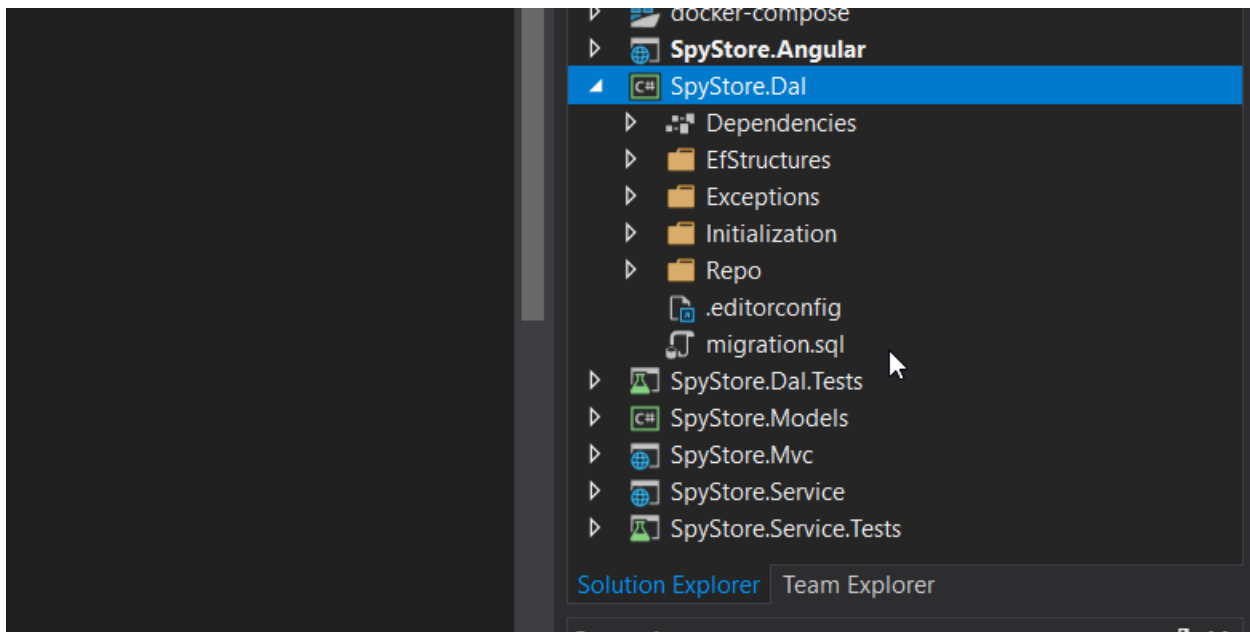
1. SpyStore.Models

This is a c# class project library that contain the domain model entities, this project is free because it not have any reference to other project so it can be use every where with any database provider (oracle,mysql, potsgress), it can also contain viewmodel (VM) entities, viewmodel are entities use to share data with client side application, we can seen it as a data transfer object (DTO).



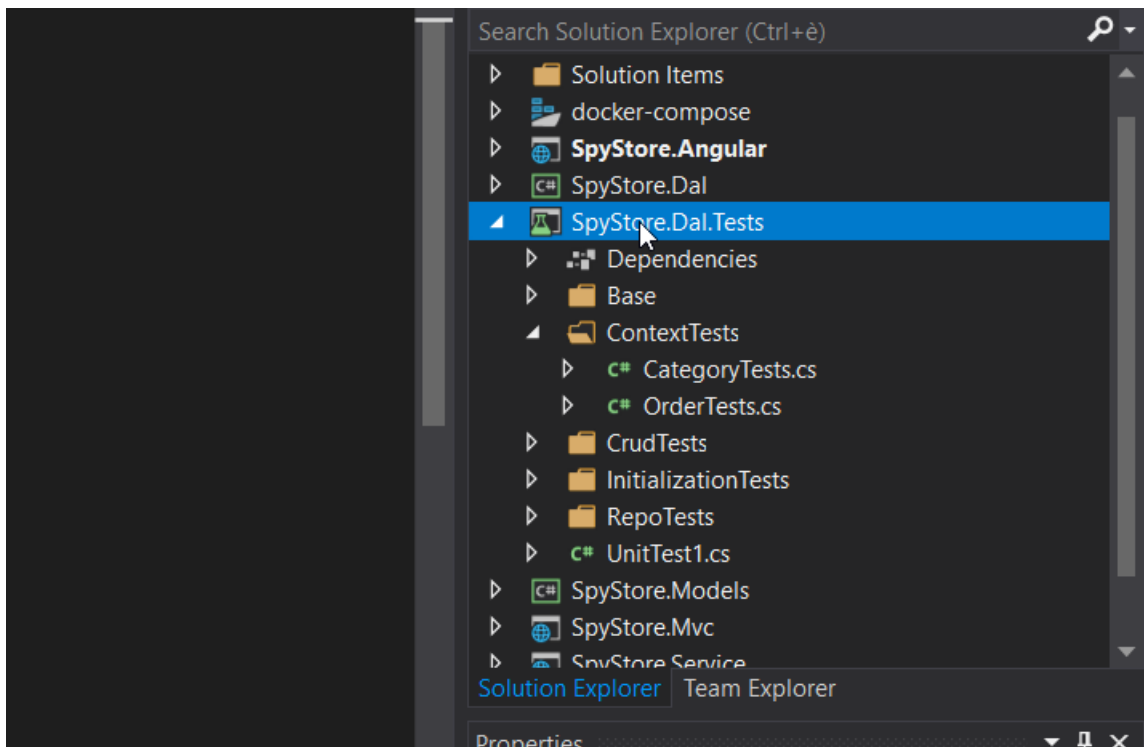
2. SpyStore.Dal

This is also a c# class library with Domain Access Layer (Dal), here we have classes that contain code to communicate with the underline database, here all entity framework package are declare and use, we can also define here the repository object. Repository pattern hold the in memory object with data coming from database, in a real project, the repository can be another c# class library in is on. The DAL contain a reference to Model project.



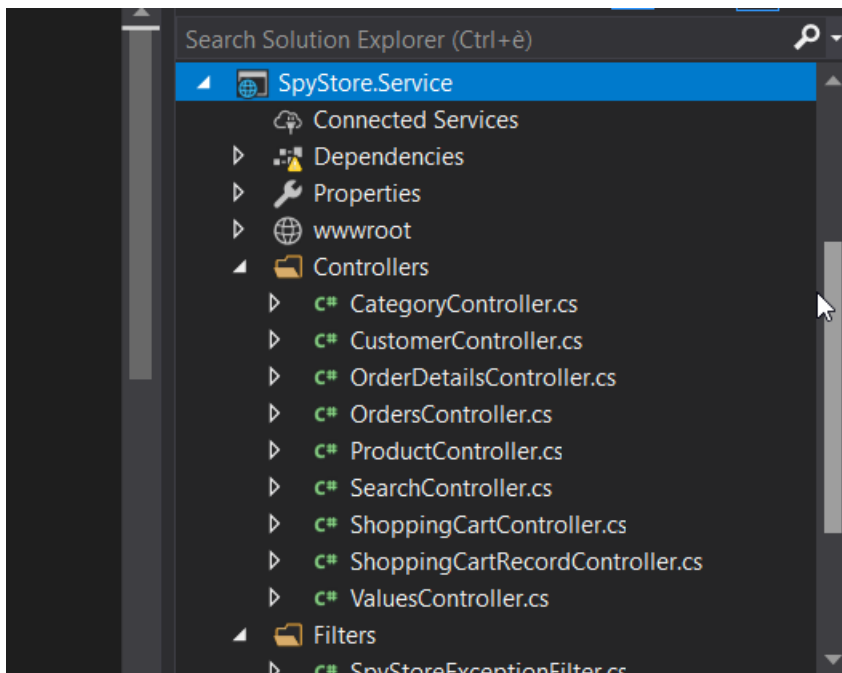
3. SpyStore.Dal.Tests

Another c# class library where we can unittest our DAL, we can use it to make CRUD operation directly to the DB.



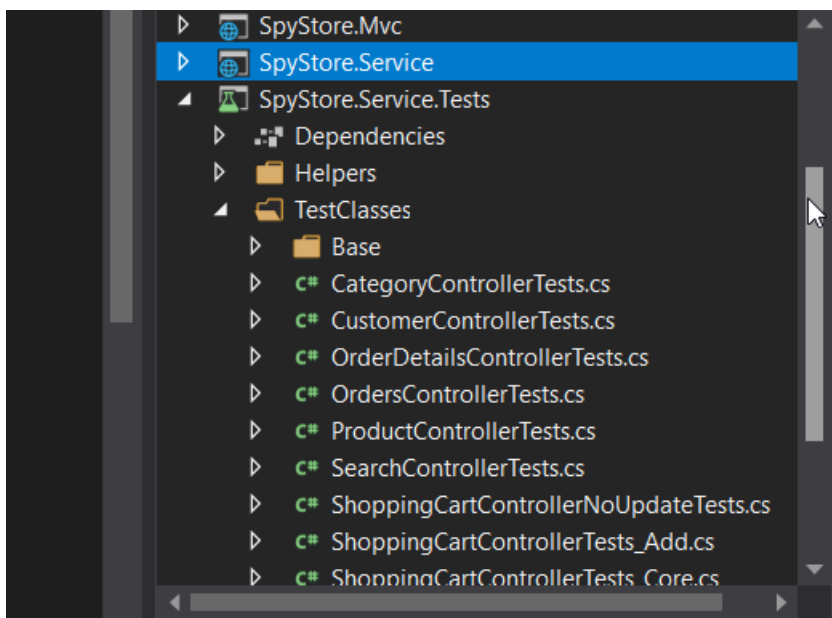
4. SpyStore.Service

An asp.net core REST MVC project where we define all the endpoints to communicate with our backend application from the outside, we can run this project in **ISS (Internet Information Service)** the classic web server in windows operating system, or we can run it on **Kestrel** the lightweight server used in asp.net core. We decide to **containerize** (running in docker container) this project so that all client side application can easily communicate with it without to launch it from visual studio or from command line with “**dotnet run**” command. This project contains a reference to Model and DAL project



5. SpyStore.Service.Tests

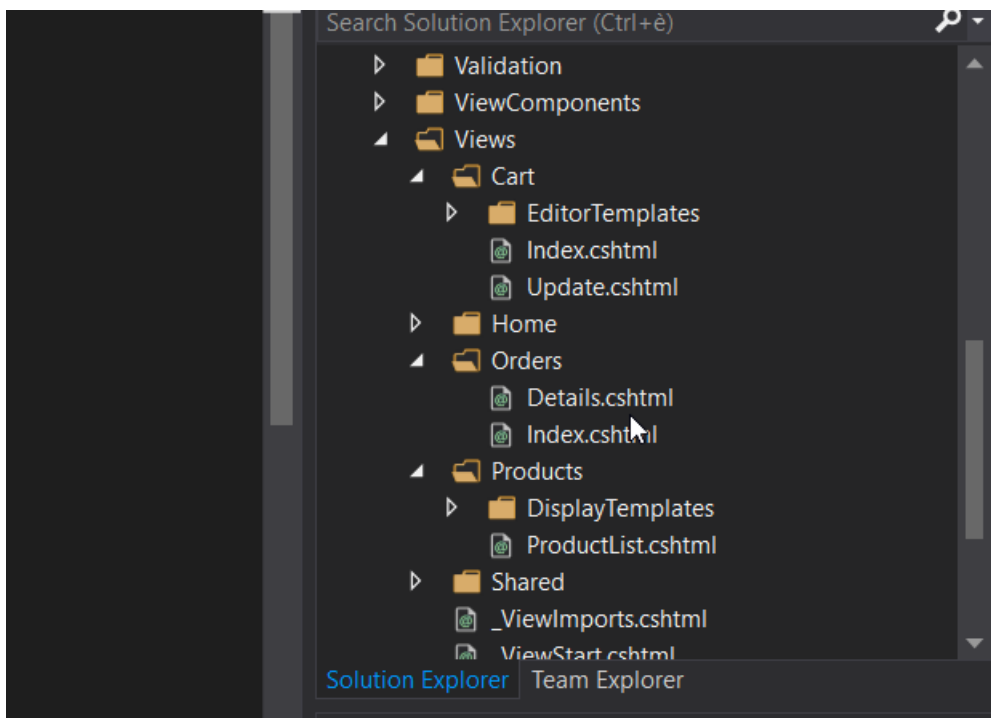
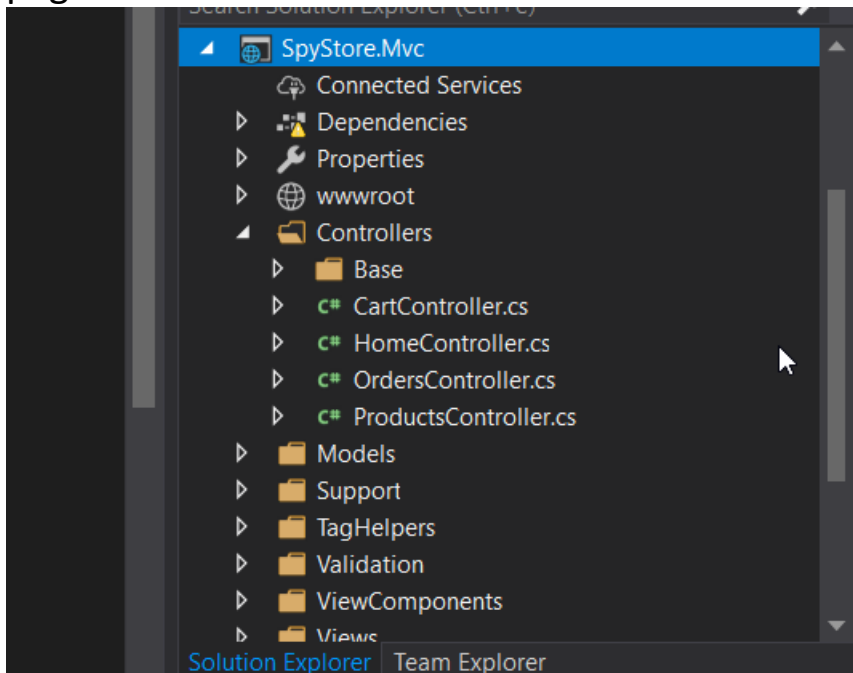
This is another c# unittests project where we can test our REST endpoint so that we can be sure that CRUD operation can directly be done from the REST interface.



6. SpyStore.MVC

This an asp.net core MVC web project, our first client application, it have a reference to model project, an communicate with the REST endpoint

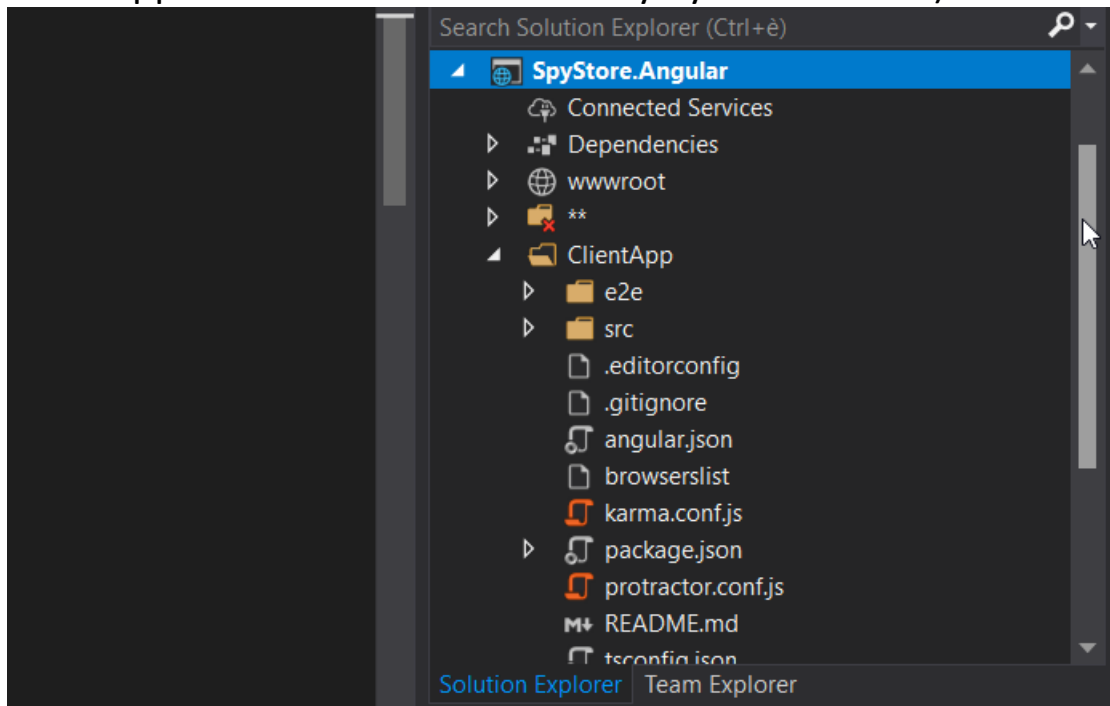
through c# httpClient class. Contain all the razor view to show the html page to the final user



7. SpyStore.Angular

This is a visual studio web site with angular project. Angular project can be generate directly from command line, but visual studio make it easy to create and angular project wich can have a MVC c#

backend endpoint, in this project we will define angular Artifact like component, service and so on (all angular files are contain in ClientApp folder create automatically by visual studio).



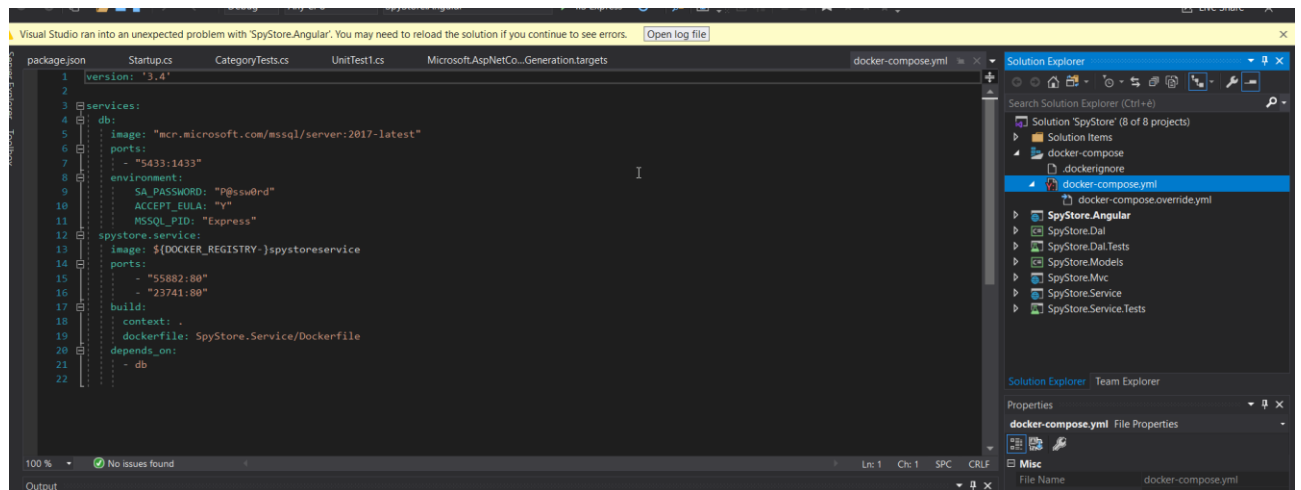
8. SpyStore-React

Like angular project, this is not part of visual studio solution because we have decide to create this project from React command line tool

Nome	Ultima modifica	Tipo	Dimensione
.git	18/01/2020 18:48	Cartella di file	
node_modules	18/01/2020 19:01	Cartella di file	
public	18/01/2020 19:01	Cartella di file	
src	18/01/2020 19:01	Cartella di file	
.env	31/12/2019 12:52	File ENV	1 KB
.gitignore	26/10/1985 10:15	Documento di testo	1 KB
dockerfile	15/01/2020 10:02	File	1 KB
nginx.conf	15/01/2020 10:02	File CONF	4 KB
package.json	18/01/2020 18:39	JSON File	2 KB
package-lock.json	18/01/2020 18:40	JSON File	613 KB
README.md	15/01/2020 10:02	Markdown File	3 KB
spystore.react.nswag	18/01/2020 17:38	File NSWAG	3 KB
tsconfig.json	15/01/2020 10:02	JSON File	1 KB

9. Docker.Compose

The last project in visual studio solution is the docker compose project where we define yaml file to run the sql server inside a docker container, and the REST endpoint, docker compose is the tool use to orchestrate a container base solution (microservices)



How to Run the project ?

Download or clone the project from this github account from here (<https://github.com/Defcoq/fullstack-dotnet-core-with-angular-and-react.git>) or just create a local directory into your file system, open a powershell command prompt and navigate to the new directory you have just create and fire the command bellow from powershell:

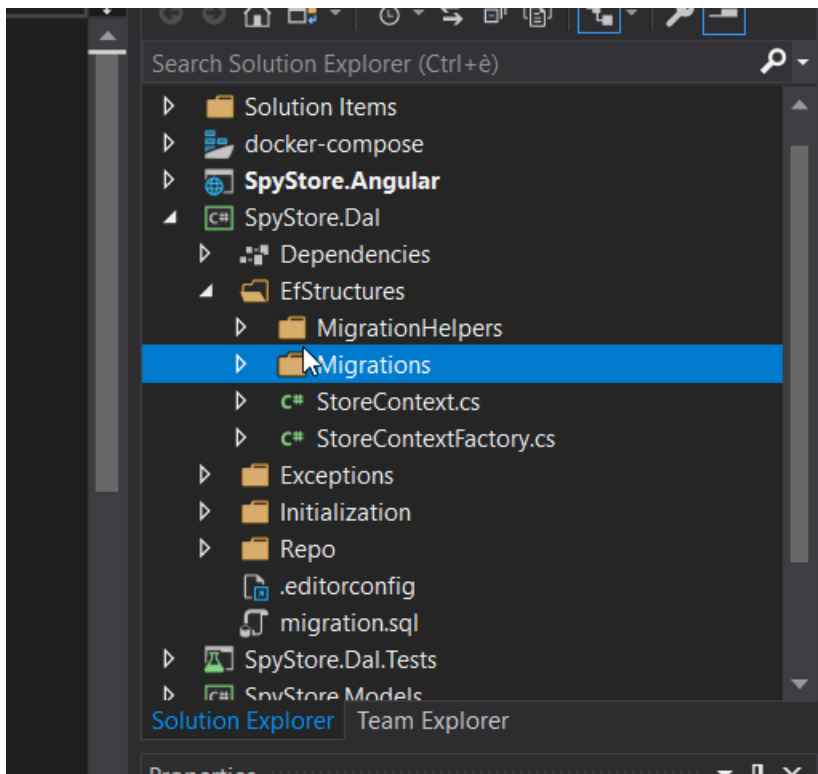
```
git clone https://github.com/Defcoq/fullstack-dotnet-core-with-angular-and-react.git
```

Now the whole project will be download from the github repository and paste inside the new directory you have created, just double click on "SpyStore.sln" the visual studio solution to open the whole project inside visual studio.

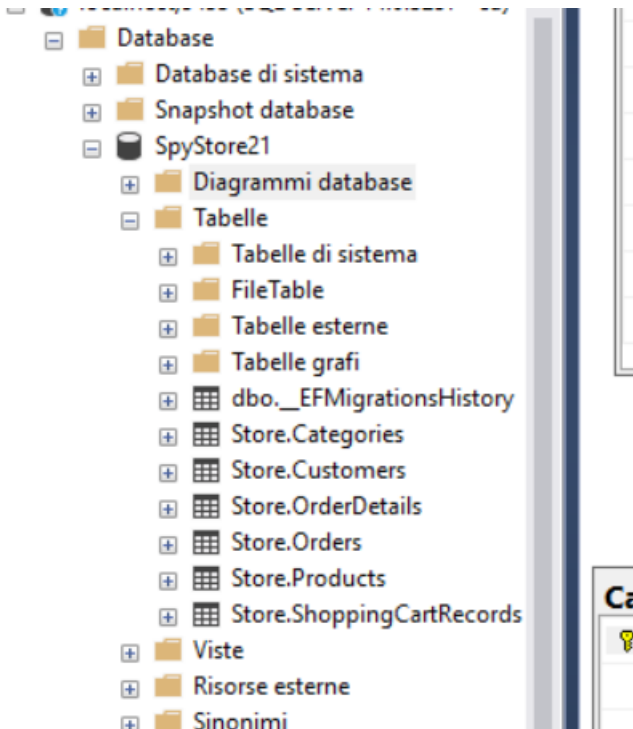
.git	19/01/2020 21:59	Cartella di file	
.vs	19/12/2019 12:35	Cartella di file	
bin	25/12/2019 16:55	Cartella di file	
obj	19/01/2020 20:53	Cartella di file	
SpyStore.Angular	18/01/2020 16:49	Cartella di file	
SpyStore.Dal	23/12/2019 17:50	Cartella di file	
SpyStore.Dal.Tests	19/01/2020 20:53	Cartella di file	
SpyStore.Models	23/12/2019 17:48	Cartella di file	
SpyStore.Mvc	28/12/2019 22:56	Cartella di file	
SpyStore.Service	15/01/2020 21:01	Cartella di file	
SpyStore.Service.Tests	15/01/2020 20:35	Cartella di file	
spystore-react	18/01/2020 19:01	Cartella di file	
.dockerignore	19/01/2020 21:58	File DOCKERIGNO...	1 KB
.editorconfig	25/12/2019 14:45	File EDITORCONFIG	1 KB
.gitignore	19/01/2020 21:59	Documento di testo	1 KB
CreateSolutionsAndProject.cmd	25/12/2019 21:53	Script di comandi ...	3 KB
databasDiagram.png	19/12/2019 11:36	File PNG	132 KB
dockercommad.txt	19/12/2019 11:18	Documento di testo	1 KB
docker-compose.dcpoj	25/12/2019 16:42	File DCPROJ	1 KB
docker-compose.override.yml	25/12/2019 16:42	File YML	1 KB
docker-compose.yml	25/12/2019 17:00	File YML	1 KB
Global.json	24/12/2019 15:32	JSON File	1 KB
scriptDatiDiEsempioSpyStore.sql	15/01/2020 21:17	Microsoft SQL Ser...	59 KB
scriptSpySporeConImaginiFinti.sql	15/01/2020 22:13	Microsoft SQL Ser...	58 KB
SpyStore.sln	29/12/2019 11:38	Visual Studio Solut...	10 KB
SpyStoreBlucrmReadme.docx	19/01/2020 22:02	Documento di Mic...	2.416 KB
utilityCommand.txt	19/12/2019 17:29	Documento di testo	2 KB

1. Run the sql server database

You have two options to run the database server, you can do it in the sql server **localdb** you have installed in the first part of this guide or directly inside a **docker container**. If you want to create the database in **localdb**, all you need to do is to delete the Migration folder inside “SpySore.Dal” project



To do this open the file “**Migration.txt**” in notepad , also open a powershell command line tool and navigate to the “SpyStore.Dal” folder project and fire line by line all the command inside the “Migration.txt” file, after doing that, entity framework will use the StoreContext to create database table in the localdb sql server instance, you can open the create database by firing sql server management studio and you’ll see all the tables create by entity framework.



Inside sql management studio, you can also run the “scriptSpySporeConImaginiFinti.sql” script to populate the db with seed data.

To run the database inside the container, you need to pull the sql server linux db from docker public registry by open a powershell command line tool, in the solution open “DalDocker.cmd” from notepad, and copy and paste the two first line to powershell

```

1 docker pull mcr.microsoft.com/mssql/server:2017-latest
2 docker run -e "ACCEPT_EULA=Y" -e "SA_PASSWORD=P@ssw0rd" -e "MSSQL_PID=Express" -p 5433:1433 --name SpyStore21 -d mcr.microsoft.com/mssql/server:2017-latest
3

```

The first command will pull the sql server db from docker public registry, it will take some times, while the second command is used to run a docker container contains or sql server from the image download from the registry, this container is expose to the port 5433 on our local machine while is bind to port 1433 inside docker container. You can stop or start the container as you’ll like, first run the following docker command to list all the running container:

```
docker container ls -a
```

to stop or start a container

```
docker container start <container id>
```



```
docker container stop <container id>
```

the container_id is taken from the docker container ls -a command in the first column.

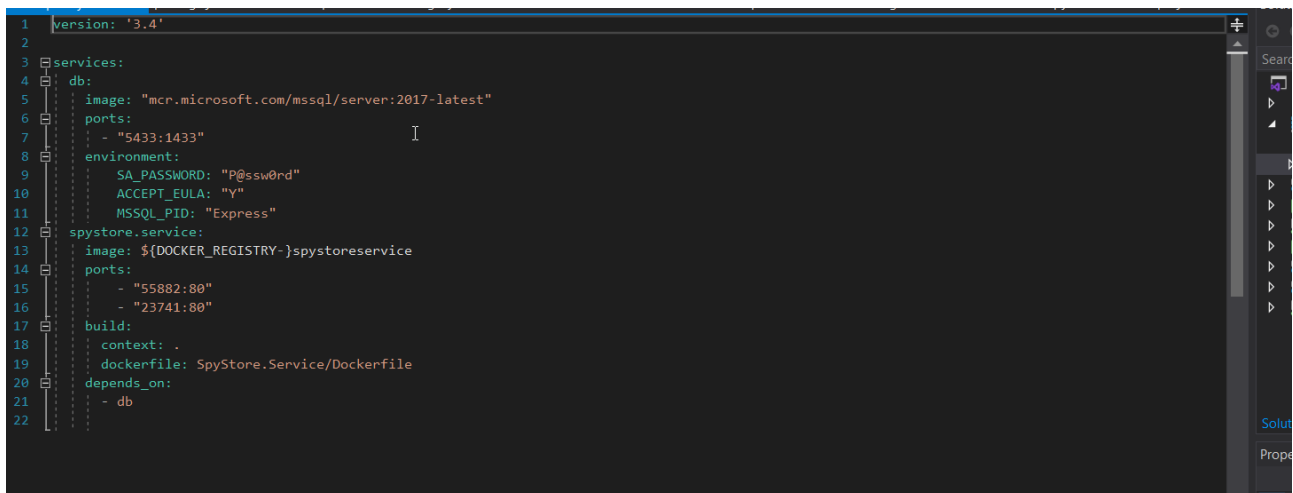
To definitively remove the container, just stop the running container with the command above and run

```
docker container rm <container id>
```

you can also use sql management studio to view the tables inside the docker container, just connect to **"localhost,5433"** address with the sql server **"sa"** user with **"P@ssw0rd"** as password like you have just use in the docker command to run the sql server container.

2. Run the docker compose project

The fast way to go is to launch the two fundamentals microservices : **"SpyStore.Service"** REST endpoint and **"sql server database"** from the docker compose project so that the REST endpoint and sql server database will be fire once and run inside two different containers, sql server will continue to run on port 5433 from our local machine, and to port 1433 inside the container while the REST endpoint will be run on port 55882 from our local machine while is exposing to port 80 inside the container as you can get from the docker compose configuration yaml file:



```
1 version: '3.4'
2
3 services:
4   db:
5     image: "mcr.microsoft.com/mssql/server:2017-latest"
6     ports:
7       - "5433:1433"
8     environment:
9       SA_PASSWORD: "P@ssw0rd"
10      ACCEPT_EULA: "Y"
11      MSSQL_PID: "Express"
12   spystore.service:
13     image: ${DOCKER_REGISTRY-}spystoreservice
14     ports:
15       - "55882:80"
16       - "23741:80"
17     build:
18       context: .
19       dockerfile: SpyStore.Service/Dockerfile
20     depends_on:
21       - db
22
```

To run the two container, open a powershell command line tool and run the docker compose command bellow inside the solution folder:

```
docker-compose up
```

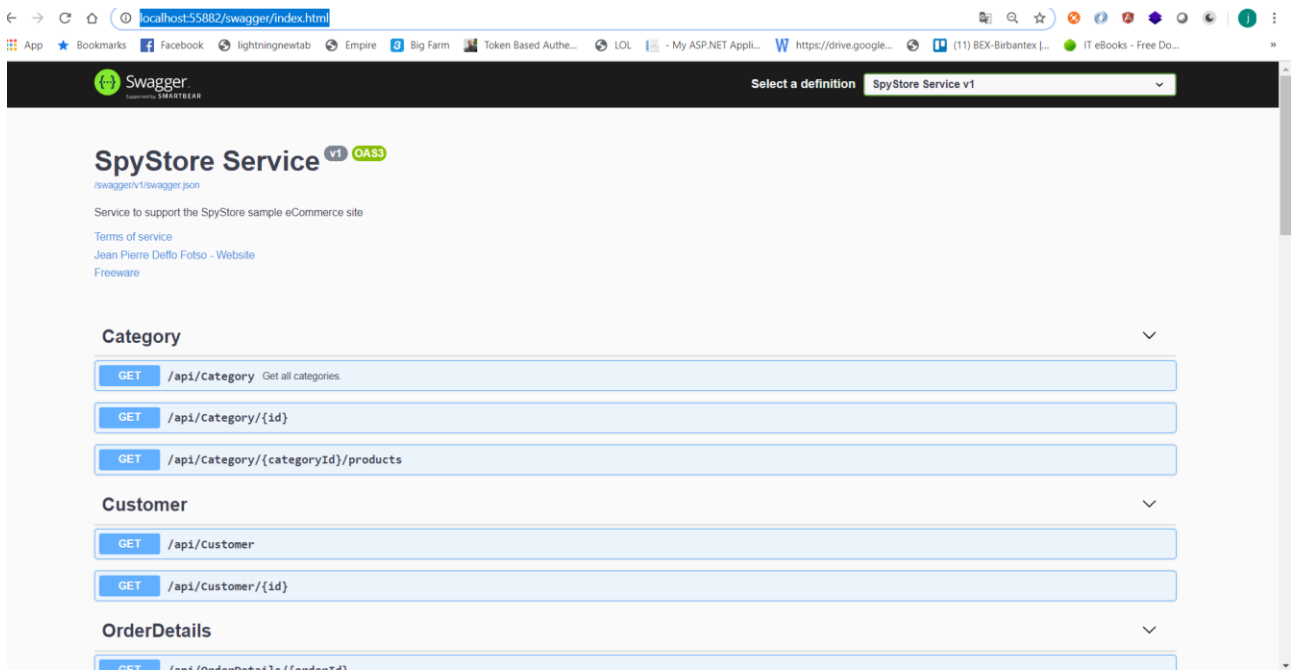
this will launch the two container inside docker..

```
PS C:\Worksapce\Dot.netCoreSample\Building Web Applications with .NET Core 2.1 and JavaScript\Unisi-Workshop\spystore-re
act> docker-compose up
Creating network "unisi-workshop_default" with the default driver
Creating unisi-workshop_db_1 ... done
Creating unisi-workshop_spystore.service_1 ... done
Attaching to unisi-workshop_db_1, unisi-workshop_spystore.service_1
db_1          | SQL Server 2019 will run as non-root by default.
db_1          | This container is running as user root.
db_1          | To learn more visit https://go.microsoft.com/fwlink/?linkid=2099216.
spystore.service_1 | warn: Microsoft.AspNetCore.DataProtection.Repositories.FileSystemXmlRepository[60]
spystore.service_1 |   Storing keys in a directory '/root/.aspnet/DataProtection-Keys' that may not be persisted ou
tside of the container. Protected data will be unavailable when container is destroyed.
spystore.service_1 | info: Microsoft.AspNetCore.DataProtection.KeyManagement.XmlKeyManager[0]
spystore.service_1 |   User profile is available. Using '/root/.aspnet/DataProtection-Keys' as key repository; keys
will not be encrypted at rest.
spystore.service_1 | info: Microsoft.AspNetCore.DataProtection.KeyManagement.XmlKeyManager[58]
spystore.service_1 |   Creating key {5c4e5383-2590-4547-8393-55dd7c80cd5a} with creation date 2020-01-19 09:26:38Z,
activation date 2020-01-19 09:26:38Z, and expiration date 2020-04-18 09:26:38Z.
spystore.service_1 | warn: Microsoft.AspNetCore.DataProtection.KeyManagement.XmlKeyManager[35]
spystore.service_1 |   No XML encryptor configured. Key {5c4e5383-2590-4547-8393-55dd7c80cd5a} may be persisted to
storage in unencrypted form.
spystore.service_1 | info: Microsoft.AspNetCore.DataProtection.Repositories.FileSystemXmlRepository[39]
spystore.service_1 |   Writing data to file '/root/.aspnet/DataProtection-Keys/key-5c4e5383-2590-4547-8393-55dd7c80
cd5a.xml'.
spystore.service_1 | warn: Microsoft.EntityFrameworkCore.Model.Validation[30000]
spystore.service_1 |   No type was specified for the decimal column 'LineItemTotal' on entity type 'ShoppingCartRec
ord'. This will cause values to be silently truncated if they do not fit in the default precision and scale. Explicitly
specify the SQL server column type that can accommodate all the values using 'HasColumnType()'.
spystore.service_1 | warn: Microsoft.EntityFrameworkCore.Model.Validation[30000]
spystore.service_1 |   No type was specified for the decimal column 'CurrentPrice' on entity type 'CartRecordWithPr
oductInfo'. This will cause values to be silently truncated if they do not fit in the default precision and scale. Expli
citly specify the SQL server column type that can accommodate all the values using 'HasColumnType()'.
spystore.service_1 | warn: Microsoft.EntityFrameworkCore.Model.Validation[30000]
spystore.service_1 |   No type was specified for the decimal column 'LineItemTotal' on entity type 'CartRecordWithP
roductInfo'. This will cause values to be silently truncated if they do not fit in the default precision and scale. Expli
citly specify the SQL server column type that can accommodate all the values using 'HasColumnType()'.
spystore.service_1 | warn: Microsoft.EntityFrameworkCore.Model.Validation[30000]
spystore.service_1 |   No type was specified for the decimal column 'CurrentPrice' on entity type 'OrderDetailWithP
roductInfo'. This will cause values to be silently truncated if they do not fit in the default precision and scale. Expli
citly specify the SQL server column type that can accommodate all the values using 'HasColumnType()'.
```

To check if the REST endpoint is up and running, open a browser windows and navigate to the following URL:

<http://localhost:55882/swagger/index.html>

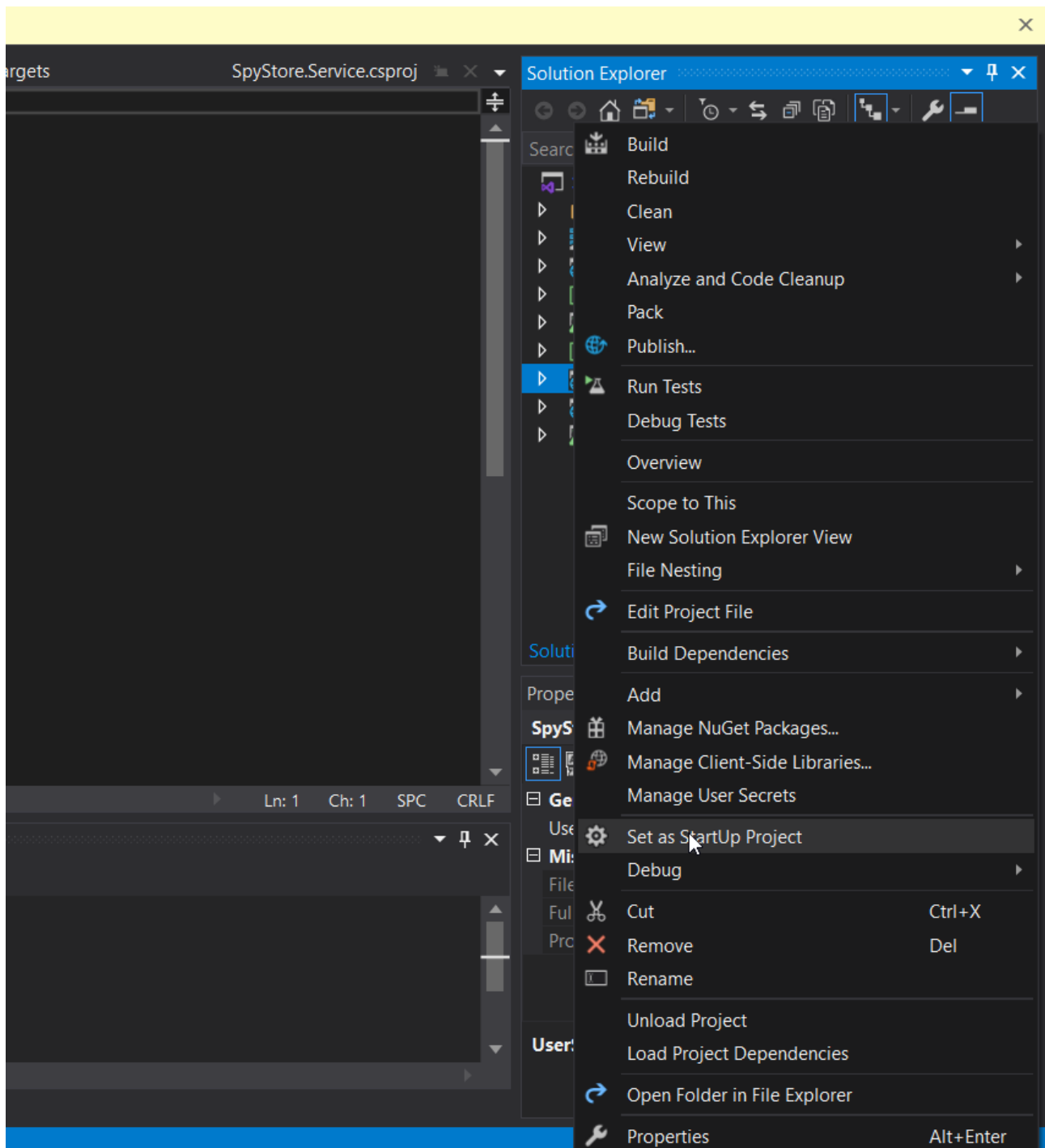
you'll see the swagger interface where you can test all the REST API endpoint directly from the swagger web page.



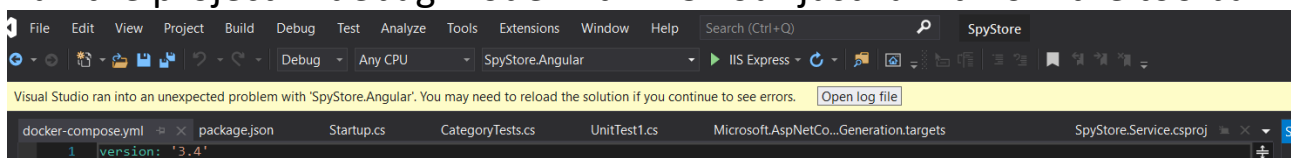
For the sql server container, you can continue to use sql server management studio to view table data, you can also run the script to seed data as we have already done above.

Now we are ready to run the MVC Client project to see the result:

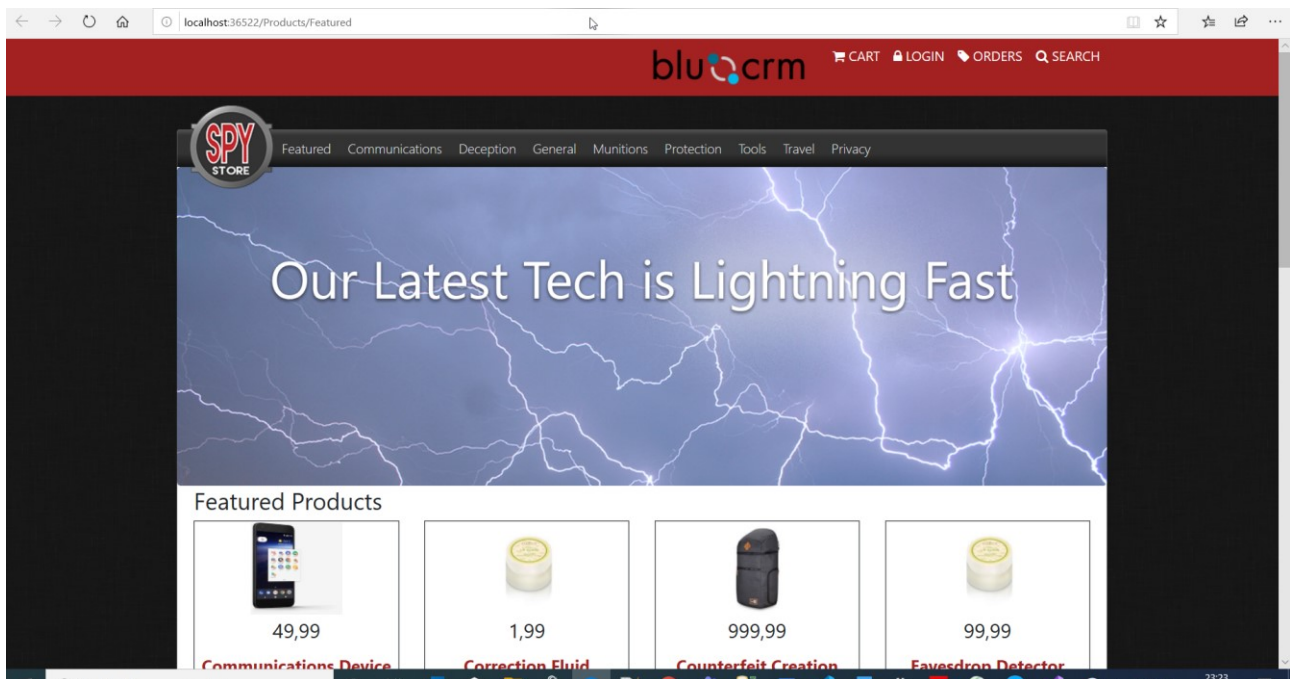
from visual studio solution, right click on “SpyStore.Mvc” project and set it as start up project



Run the project in debug mode with “F5” or just run it from the toolbar



You have done, you'll see the web page below:



You can now stop the application, if you'll like to run Angular project within visual studio, just set **"SpyStore.Angular"** as startup project as you have done before with **"SpyStore.MVC"**, and run the project, you'll see the exactly same result.

You can also run the angular project from the command line tool, just fire powershell, and navigate to the ClientApp directory inside the **"SpyStore.Angular"** first you need to install all the client side package by running the command bellow inside powershell:

```
npm install
```

and after package installation, just fire the command bellow:

```
ng serve -o
```

you'll see the exact angular web interface as you launch it from visual studio.

Finally we can run the react project, to do so, open a powershell command and navigate inside the react folder **"spystore-react"** Run the following npm command to download the client siode package:

```
npm install
```

and after you can launch the whole react client side web page by just running

```
npm start
```

you'll see the same application but this time it will be build by using React base technology.

Build the application from scratch

If you would like to build the application yourself, you can just create a new blank solution from visual studio and add project by project manually, but the sample provide a powershell script to automate the whole solution creation.

Open powershell and run the following powershell bat file

"CreateSolutionAndProjects.cmd", it will create the whole solution

You can also run the "DalNuGetPackages.cmd" to add different nuget package to all project.

You can now begin to insert different code base on our github repository project by beginning from the model project follow by DAL project and service project.. and finally you can build individually each client project