**Save uploaded file to database**

In the previous articles we have seen how to provide an option to upload files, process and save them within the solution in asp.net 6 MVC application.

In this article we will see how to save the uploaded file to a database with an example. To keep things simple, in this example we are going to use MSSQL database and “Dapper” for performing database operations.

**Dapper:**

Dapper is an open-source object-relational mapping (ORM) library for .NET and .NET Core applications.

Dapper is a Micro-ORM. It allows developers to quickly and easily access data from databases without the need to write tedious code. It supports both .NET framework and .NET core applications.

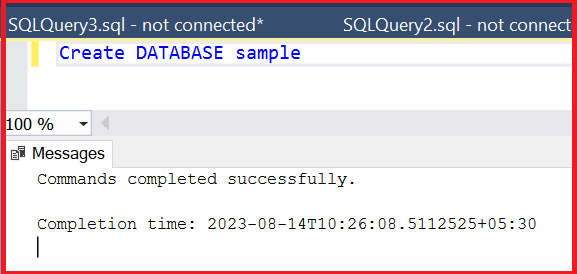
**Database changes**

For this example, I have created a database and a data table.

**Create Database**

Run below query to create the “sample” database as shown below

*“Create DATABASE sample”*



**Create table**

I have created a table “FileUpload”. It has three columns.

1. “Id” --> Data Type: **int**, **Primary key**, **Identity**, Allow Null: No – To uniquely identify the rows
2. “Name” --> Data Type: **ntext**, Allow Null: No -- To save the file name
3. “Image” --> Data Type: **varbinary(MAX)**, Allow Null: No -- To save the file content

Run the below query to create the table with the columns

*“*

*USE [sample]*

*GO*

*/\*\*\*\*\*\* Object: Table [dbo].[FileUpload] Script Date: 14-08-2023 10:35:31 \*\*\*\*\*\*/*

*SET ANSI\_NULLS ON*

*GO*

*SET QUOTED\_IDENTIFIER ON*

*GO*

*CREATE TABLE [dbo].[FileUpload](*

*[Id] [int] IDENTITY(1,1) NOT NULL,*

*[Name] [ntext] NOT NULL,*

*[Image] [varbinary](max) NOT NULL,*

*CONSTRAINT [PK\_FileUpload] PRIMARY KEY CLUSTERED*

*(*

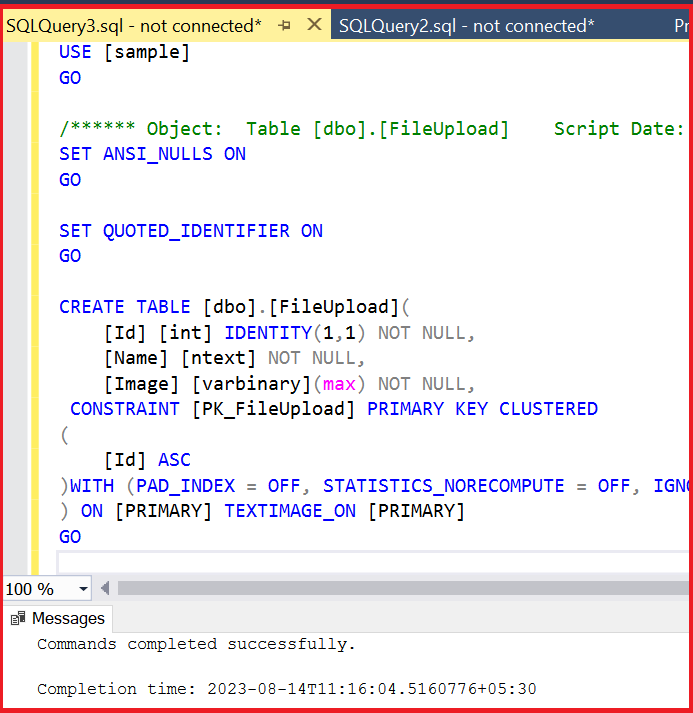
*[Id] ASC*

*)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON, OPTIMIZE\_FOR\_SEQUENTIAL\_KEY = OFF) ON [PRIMARY]*

*) ON [PRIMARY] TEXTIMAGE\_ON [PRIMARY]*

*GO*

*”*



For this example, I am using the same sample application which we had created in the previous articles.

**\_Layout.cshtml** file

I have added a new navigation bar item named “File Upload” to the “**\_Layout.cshtml**” which is within “Views --> Shared” folder as shown below

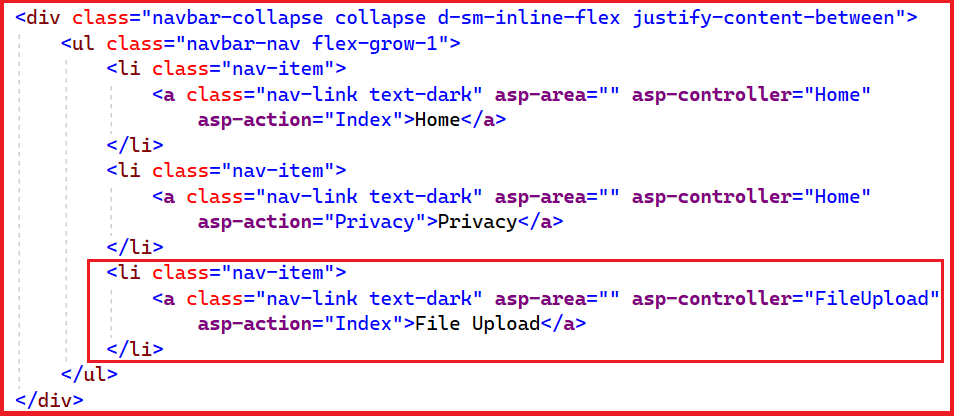
“

*<li class="nav-item">*

*<a class="nav-link text-dark" asp-area="" asp-controller="****FileUpload****"*  *asp-action="Index">File Upload</a>*

*</li>*

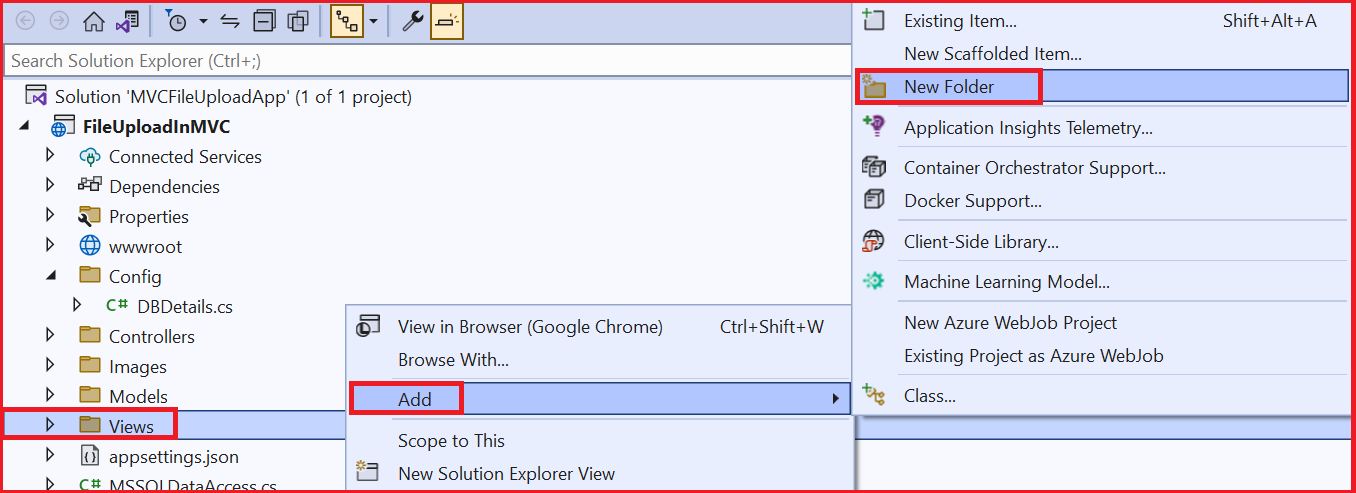
”



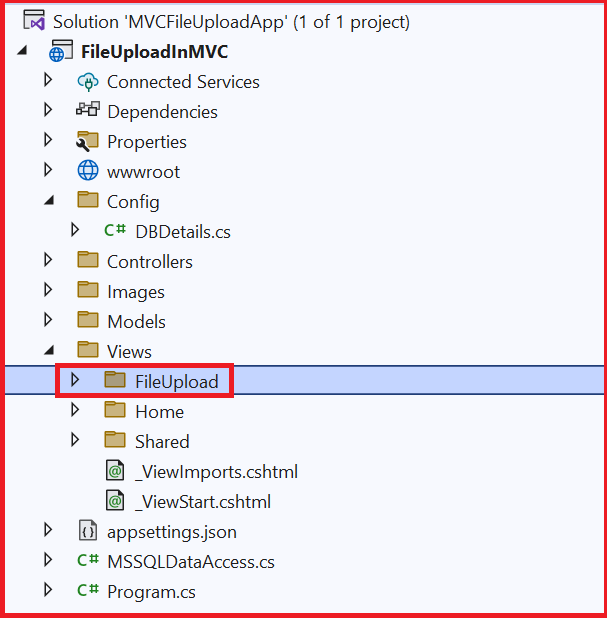
**View Changes**

I have created a new folder “FileUpload” within Views folder and added a view “index.cshtml”

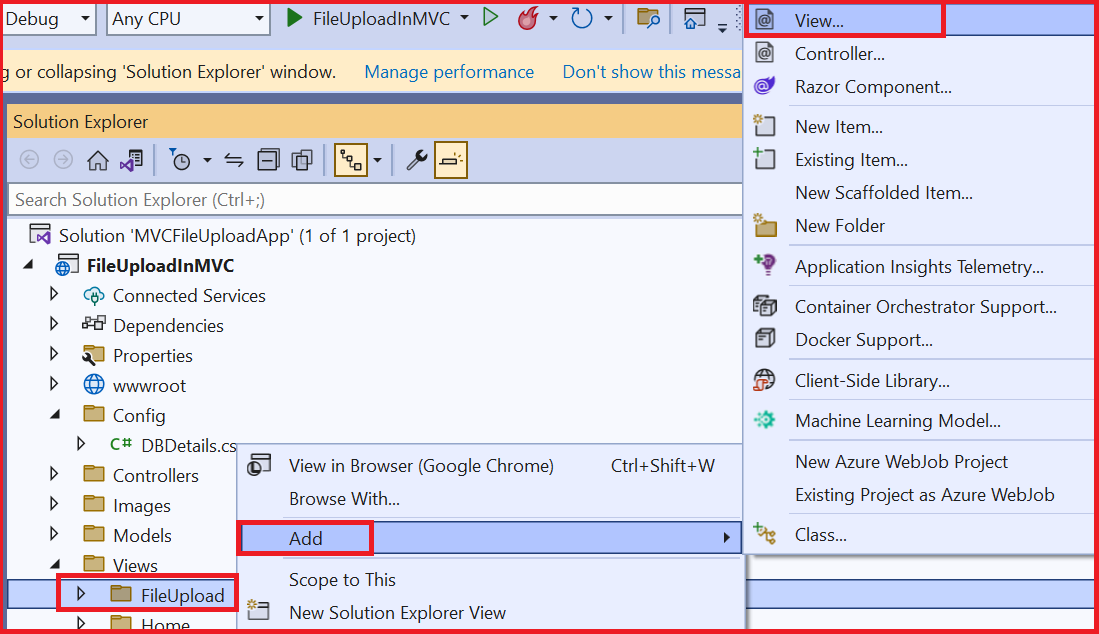
1. Right click on Views folder, select “Add” and select “New Folder”



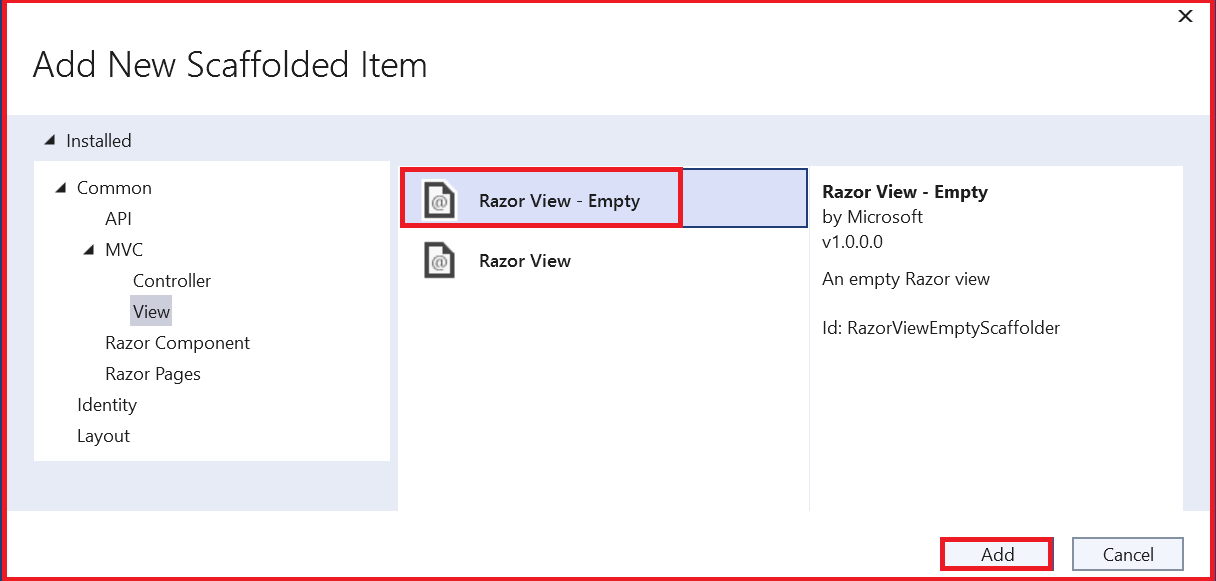
1. Give the name of the folder as “FileUpload” as shown below



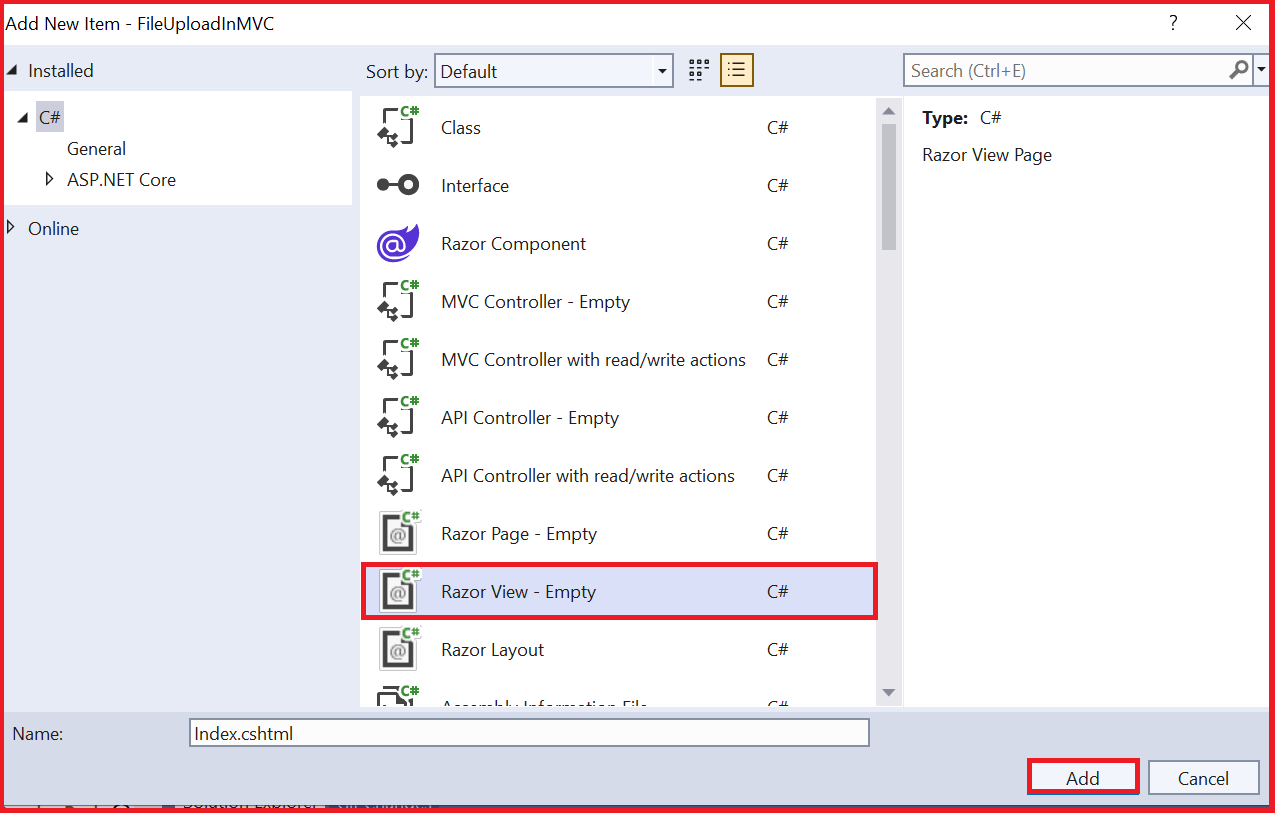
1. Right click on the “FileUpload” folder, select “Add” and select “View” as shown below



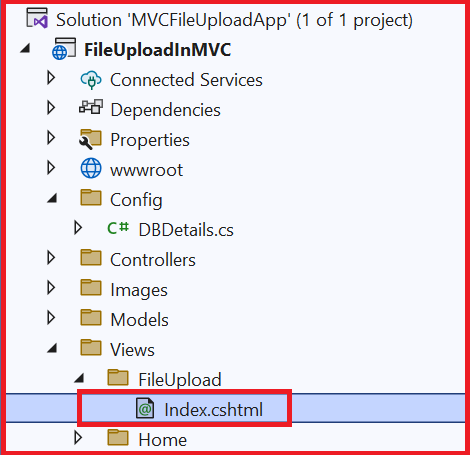
1. “Add New Scaffolded Item” popup will show up as shown below. Select “Razor View - Empty” and click “Add” as shown below



1. Give the name as “Index.cshtml” and click “Add”



1. The “Index.cshtml” file will be created as shown below



1. I am using the same view file content what we have seen in the previous article “uploading a single file” as shown below

“

*<div>*

*<h5>Single File Upload to Database</h5>*

*<form method="post" asp-controller="FileUpload" asp-action="SingleFileUploadToDatabase"*

*enctype="multipart/form-data">*

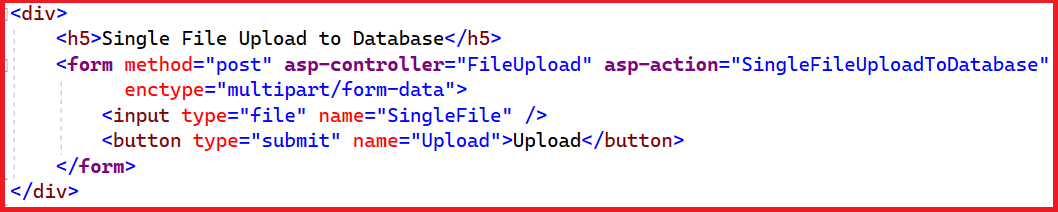
*<input type="file" name="SingleFile" />*

*<button type="submit" name="Upload">Upload</button>*

*</form>*

*</div>*

”



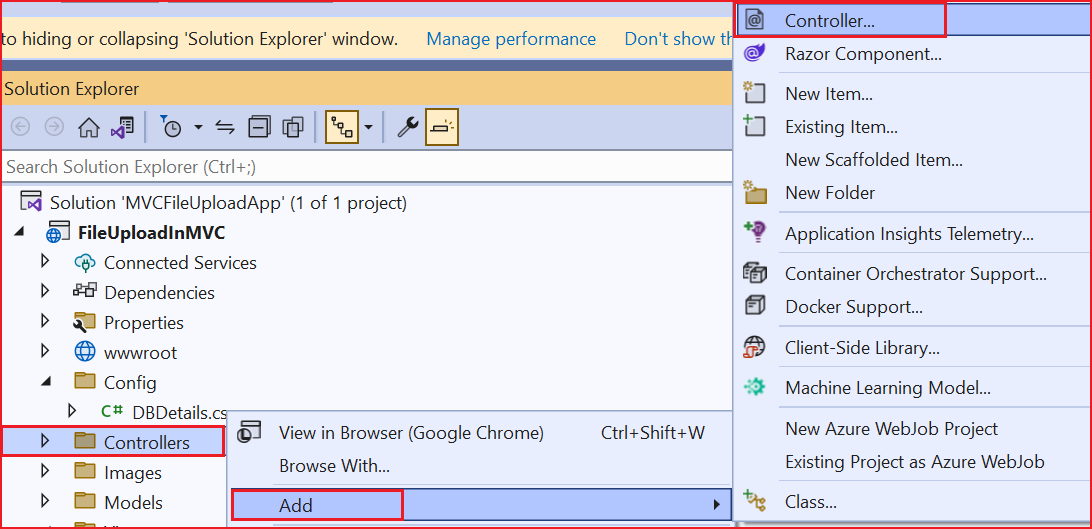
**Notice that**

1. form control has method = post, asp-controller= ”FileUpload” asp-action=” **SingleFileUploadToDatabase**” attributes. The “**SingleFileUploadToDatabase”** action method in the **FileUpload** Controller will be executed on submitting the form.
2. The name attribute of the file type input is **SingleFile**

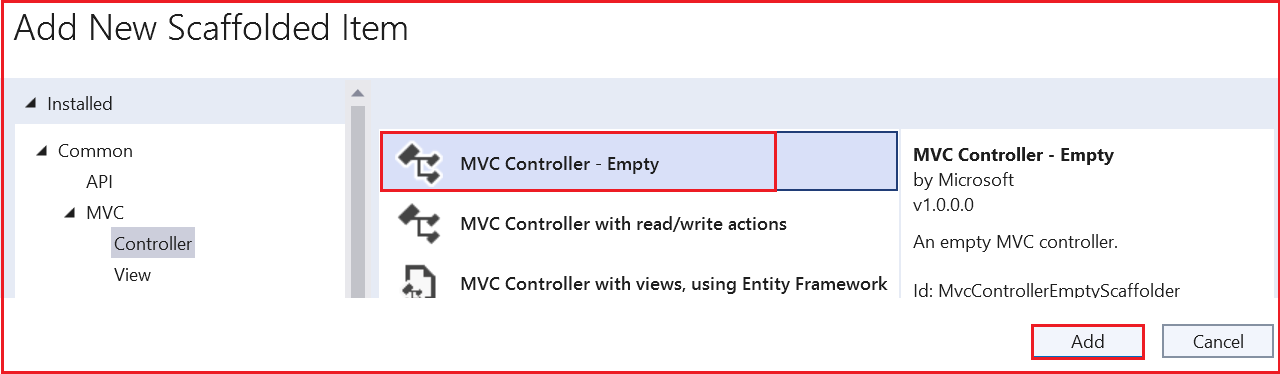
Controller changes

I have added a new controller “FileUploadController” and added an action method “SingleFileUploadToDatabase”.

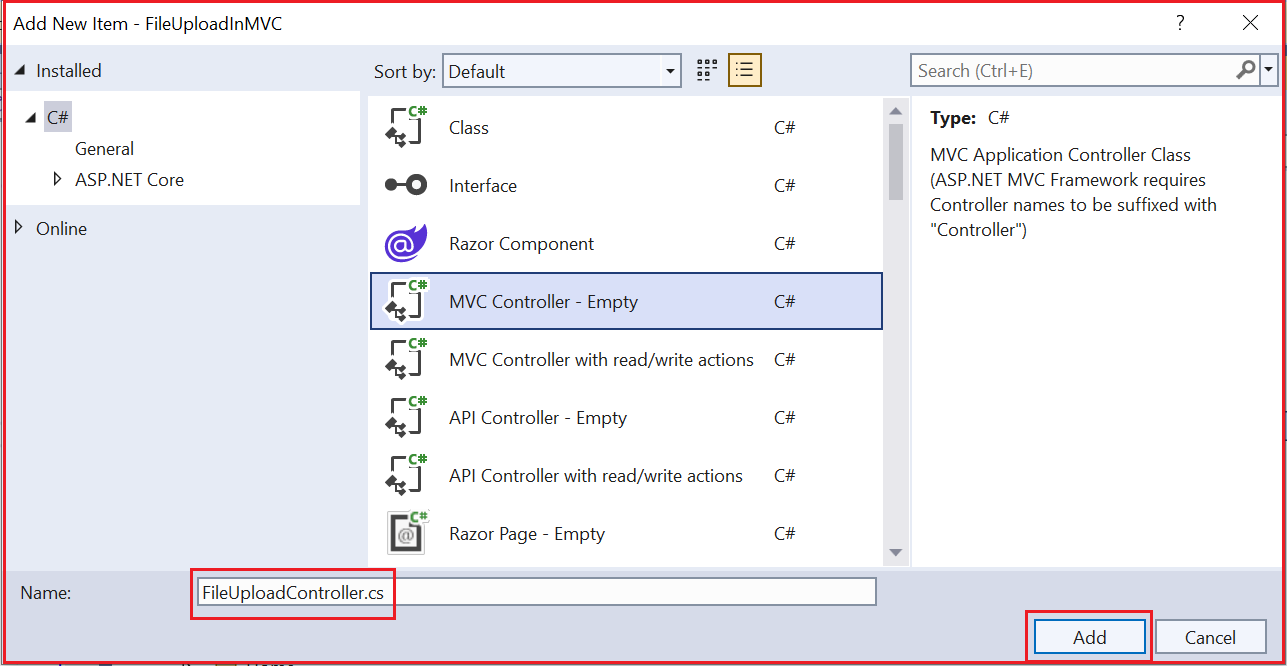
1. Right click on the “Controllers” folder, select “Add” and select “Controller” as shown below



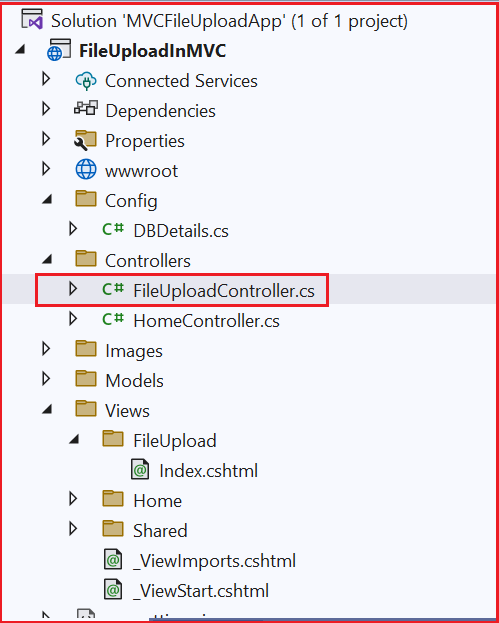
1. “Add New Scaffolded Item” popup will show up. “Select MVC Controller – Empty" and select “Add” as shown below



1. Give the controller's name as “FileUploadController” in the “Add New Item” popup and click on “Add” as shown below



1. “FileUploadController” will be created as shown below



**Action method**

I have added the action method “SingleFileUploadToDatabase” to FileUploadController class as shown below

“

*public async Task<IActionResult> SingleFileUploadToDatabase(IFormFile SingleFile)*

*{*

*using var memoryStream = new MemoryStream();*

*await SingleFile.CopyToAsync(memoryStream);*

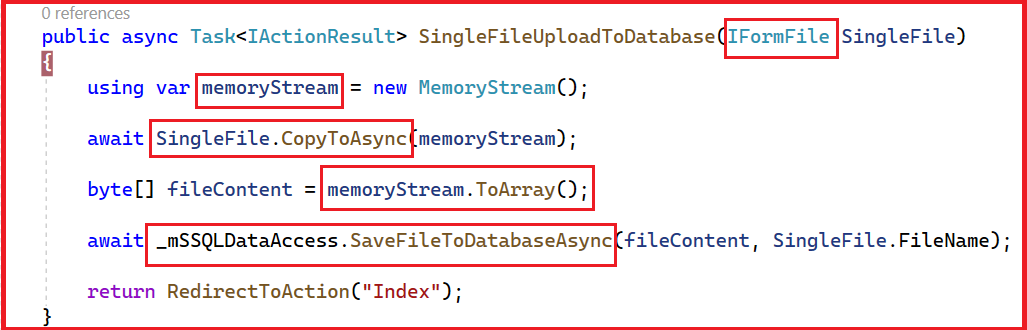
*byte[] fileContent = memoryStream.ToArray();*

*await \_mSSQLDataAccess.SaveFileToDatabaseAsync(fileContent, SingleFile.FileName);*

*return RedirectToAction("Index");*

*}*

”



1. Like the previous article we have a parameter of type “IFormFile” for the action method, which contains the file content and other details.
2. We are creating a memory stream. Then using the “CopyToAsync” method of “IFormFile” object to copy the contents of the file to memory stream
3. Then we are converting the memory stream to byte array
4. Then we pass byte array which contains the file content and the uploaded file name to the method “**SaveFileToDatabaseAsync**” which is defined in the “MSSQLDataAccess” class file to save the contents of the file to the database.

I have added a private readonly field “\_mSSQLDataAccess” of type MSSQLDataAccess. I am getting the instance of “MSSQLDataAccess” class in the constructor through built in dependency injection and assigning it to “\_mSSQLDataAccess” field as shown below

“

*private readonly MSSQLDataAccess \_mSSQLDataAccess;*

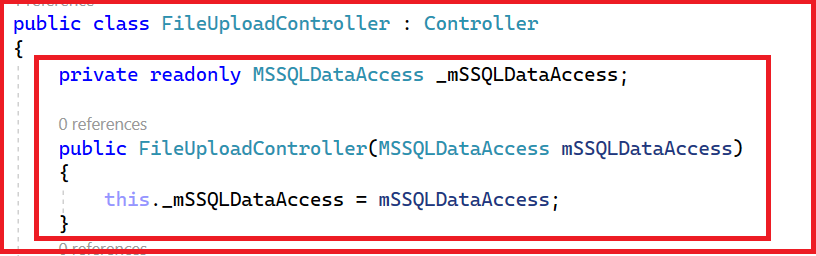
*public FileUploadController(MSSQLDataAccess mSSQLDataAccess)*

*{*

*this.\_mSSQLDataAccess = mSSQLDataAccess;*

*}*

”



For dependency injection to work for “MSSQLDataAccess” class we need to add the transient service of type “MSSQLDataAccess” to the “builder.Services” collection in the “Program.cs” file as shown below

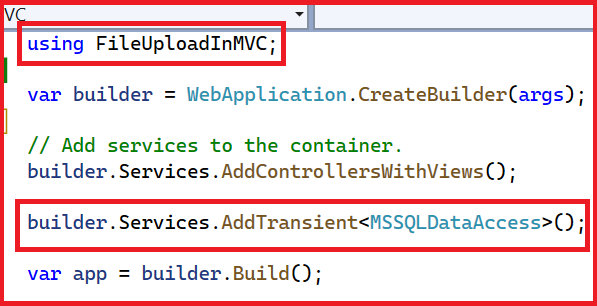
**Program.cs**

“

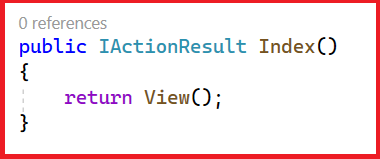
*using FileUploadInMVC;*

*builder.Services.AddTransient<MSSQLDataAccess>();*

”



The index action method in the “FileUploadController” just returns the ViewResult object as shown below



“

public IActionResult Index()

{

return View();

}

”

The complete “FileUploadController” file is as shown below



**appsettings.json**

Add the database connection string to the appSettings.json file as shown below

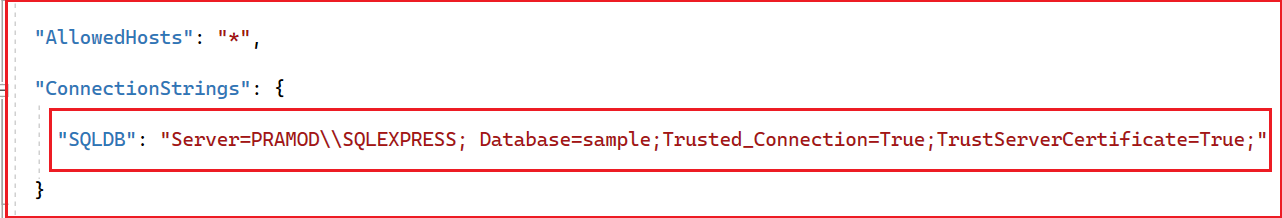
“

*"ConnectionStrings": {*

*"SQLDB": "Server=PRAMOD\\SQLEXPRESS; Database=sample;Trusted\_Connection=True;TrustServerCertificate=True;"*

*}*

”

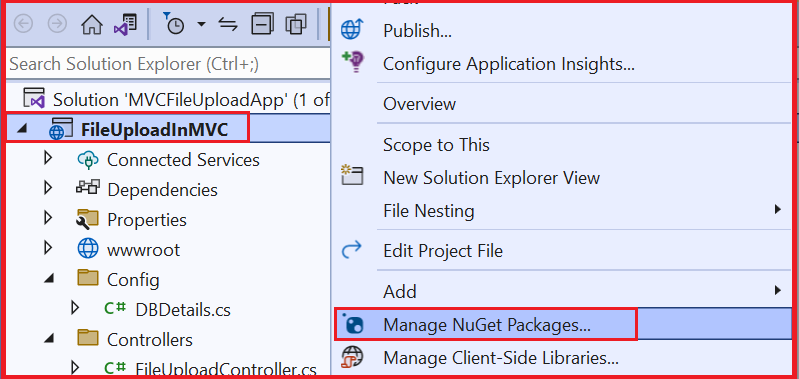


For performing database operations, I have created a class “MSSQLDataAccess”. This class acts as a data layer which executes all the sql queries on the database using dapper.

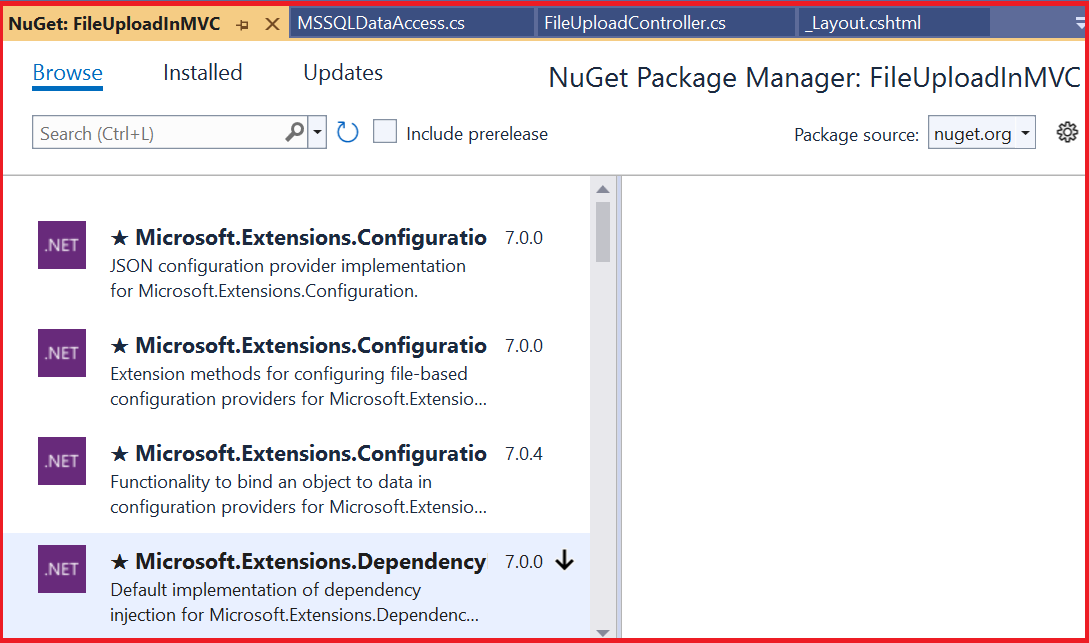
Installing NuGet packages.

To be able to perform database operations using Dapper we need to install the NuGet packages “Dapper” and “System.Data.SqlClient”

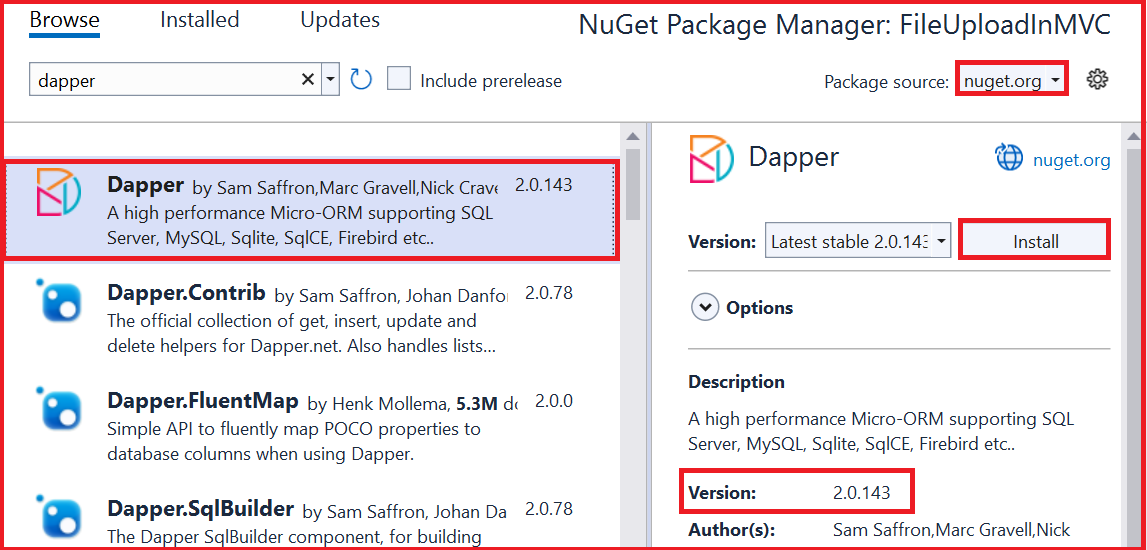
1. Right click on the project and select “Manage NuGet Packages”



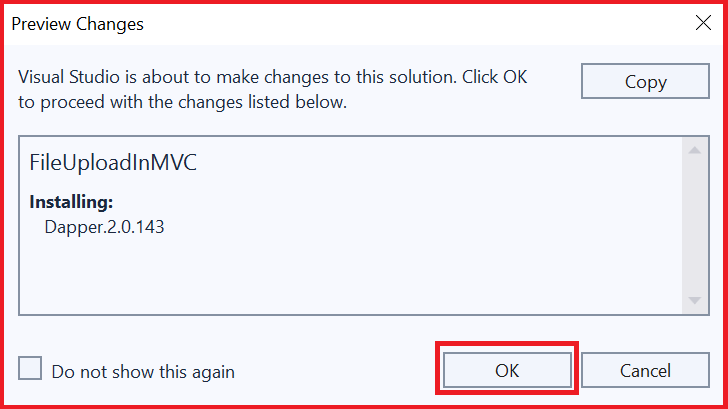
1. NuGet package manager window will open as shown below



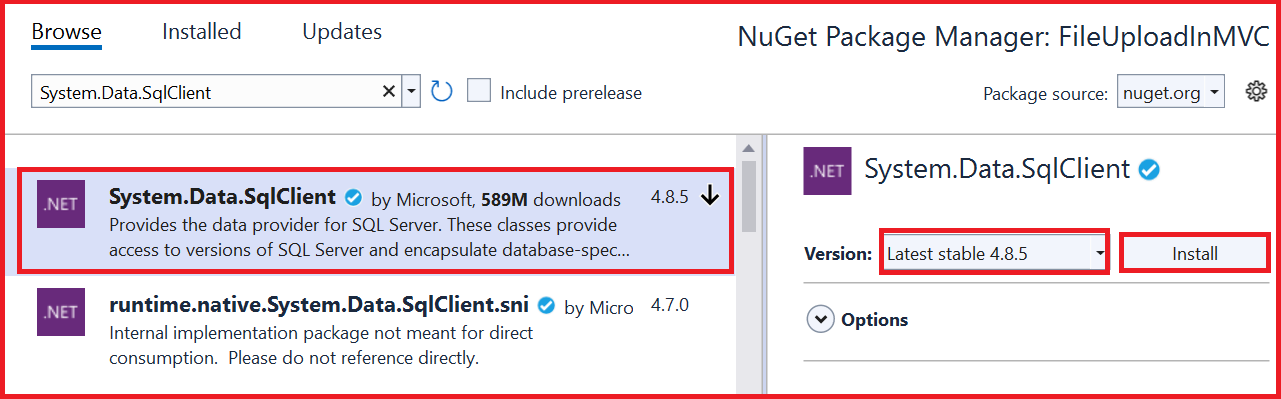
1. Under “Browse” section, search for dapper. Select the one by “Sam Saffron” I am installing the latest version 2.0.143 at the time of writing this article. Click on Install as highlighted below.



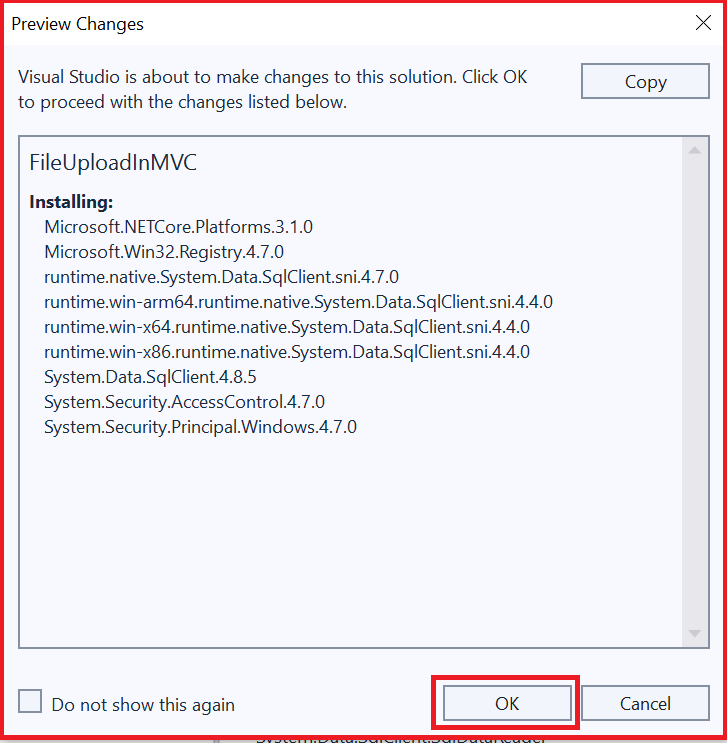
1. “Preview Changes” screen will show up. Click on OK to install



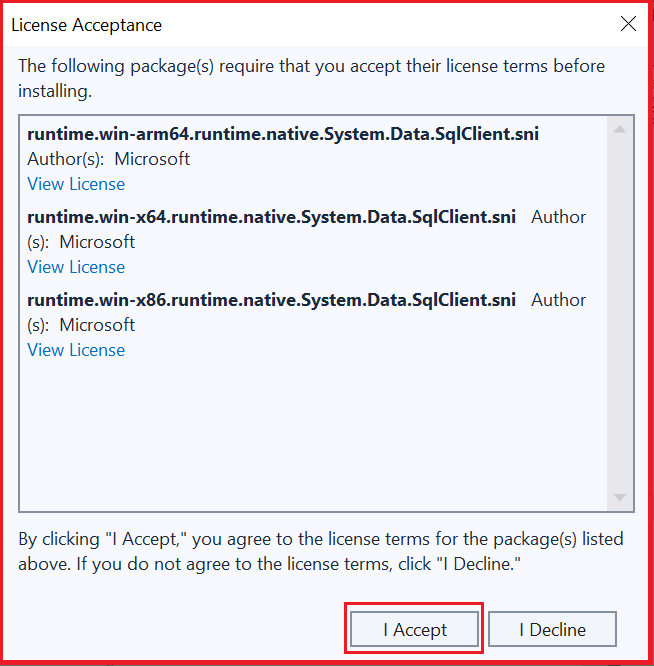
1. Similarly install “System.Data.SqlClient” by Microsoft. Installing the latest version “4.8.5” at the time of writing this article



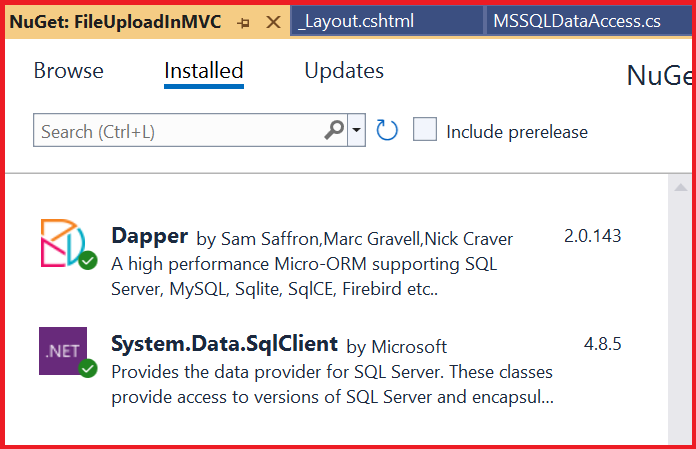
1. “Preview Changes” screen will show up. Click on OK to install



1. "License Acceptance" screen will show up, click “I Accept” as shown below to install the package

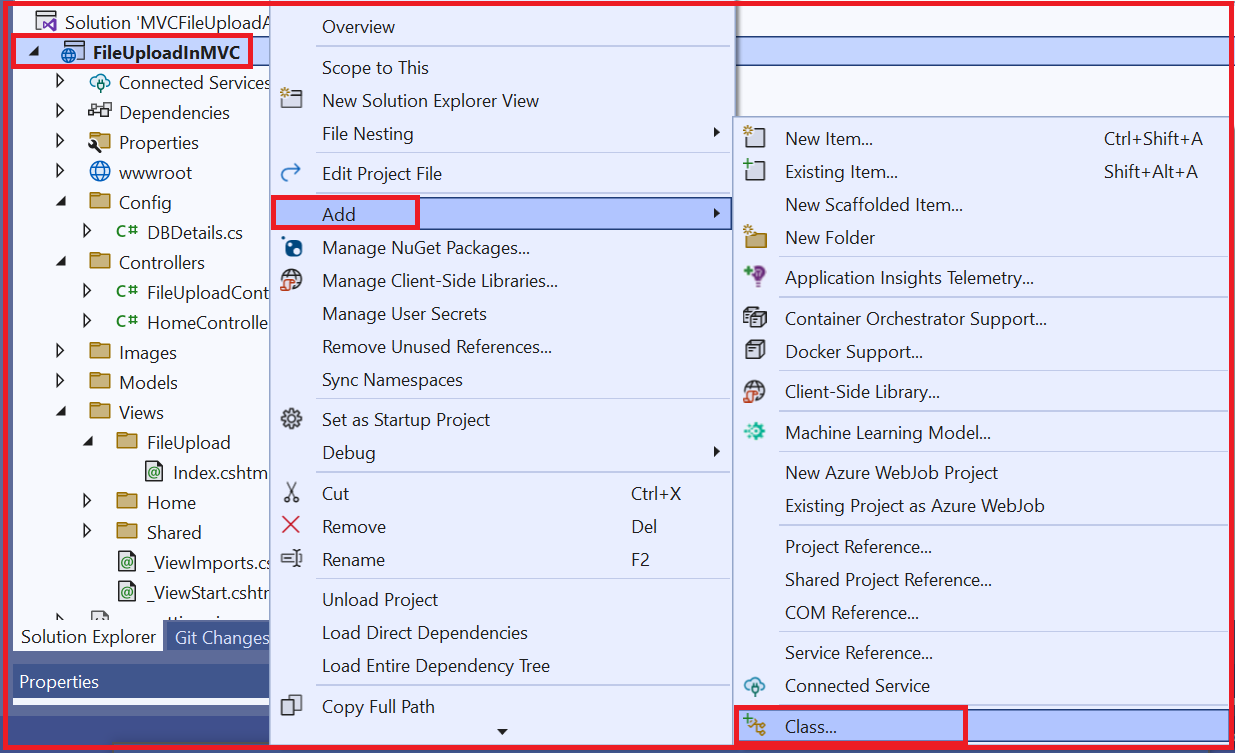


After installing the packages, we can verify from the “Installed” section as shown below



**MSSQLDataAccess**

Add a new class file “MSSQLDataAccess” to the project by right clicking on the project, select “Add” and select “class”



In the “MSSQLDataAccess” file,

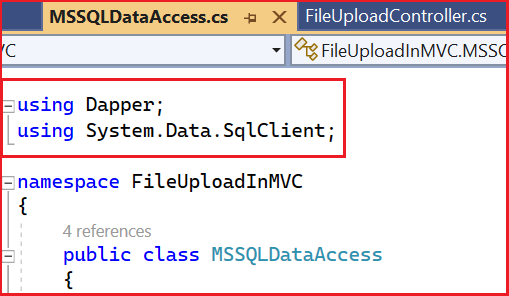
1. we have added a using statement for including “Dapper” namespace and “System.Data.SqlClient” namespace as shown below

“

*using Dapper;*

*using System.Data.SqlClient;*

”



1. We have added a constructor to get the IConfiguration object using built in dependency injection. We need “IConfiguration” object to fetch the database connection string from appsettings.json file

“

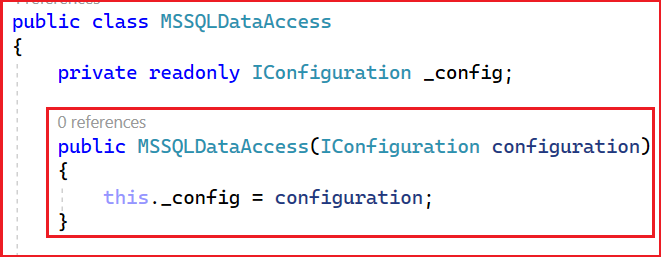
*public MSSQLDataAccess(IConfiguration configuration)*

*{*

*this.\_config = configuration;*

*}*

”



1. We have added the method “SaveFileToDatabaseAsync” which takes in byte array parameter for file content and string parameter for file name.

“

*public async Task<bool>* ***SaveFileToDatabaseAsync****(byte[] fileContent, string fileName)*

*{*

*bool result = false;*

*try*

*{*

*using var conn = new SqlConnection(\_config.GetConnectionString("SQLDB"));*

*var query = "Insert into FileUpload ([Name],[Image]) values (@name,@image)";*

*var parameters = new { name = fileName, image = fileContent };*

*var numOfRows = await conn.ExecuteAsync(query, parameters);*

*if (numOfRows > 0)*

*{*

*result = true;*

*}*

*return result;*

*}*

*catch (Exception)*

*{*

*return result;*

*}*

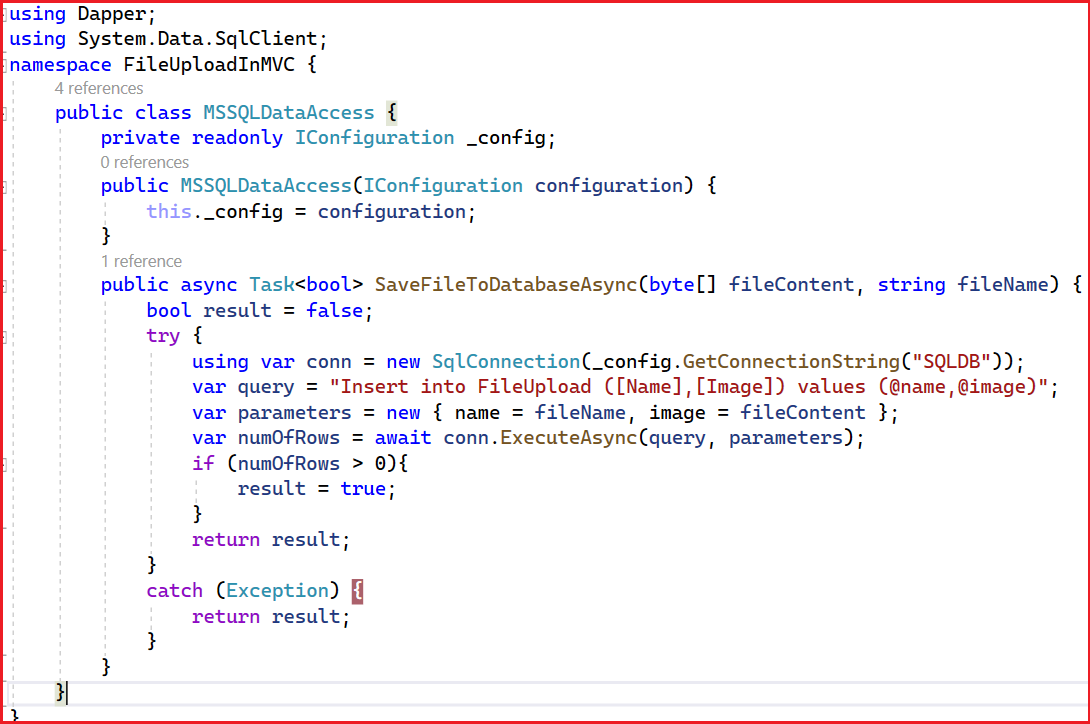
*}*

”

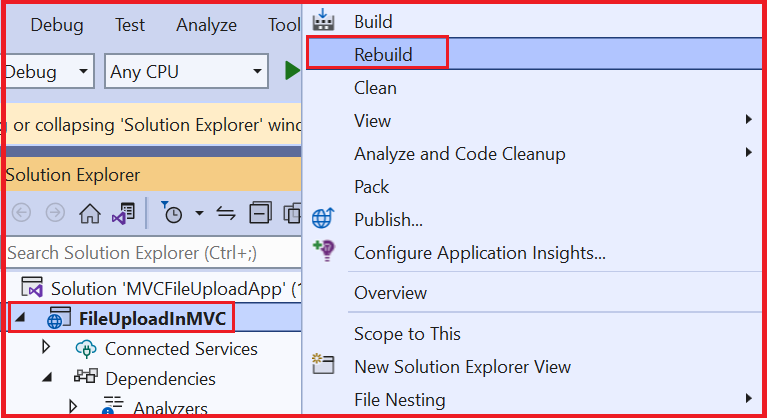


1. We are creating a new SQL connection by passing the connection string of the database.
2. We fetch the connection string from appsettings.json file using IConfiguration object
3. We have framed the SQL insert query with parameters.
4. We have created an anonymous object which contains values for the parameters
5. Then we are executing the query by passing query and the anonymous object which contains the parameters using ExecuteAsync method of the SQL connection object

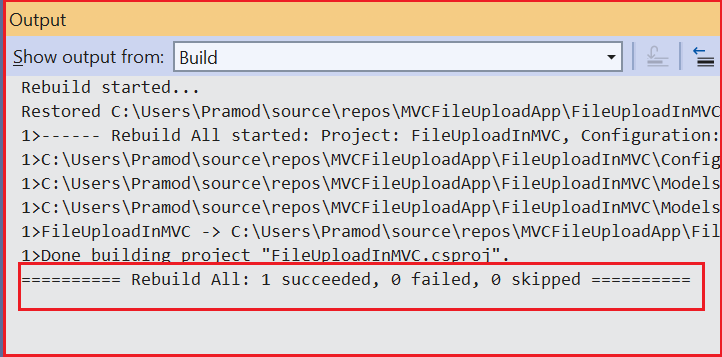
The complete MSSQLDataAccess.cs file is as shown below



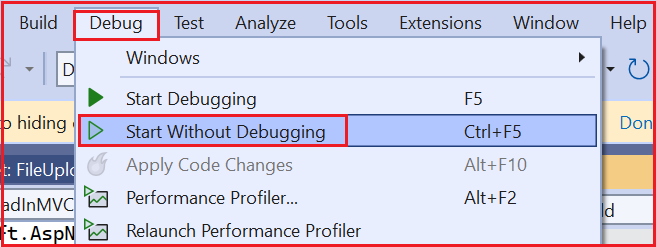
With all these changes in place, right click on the project and select “Rebuild” as shown below



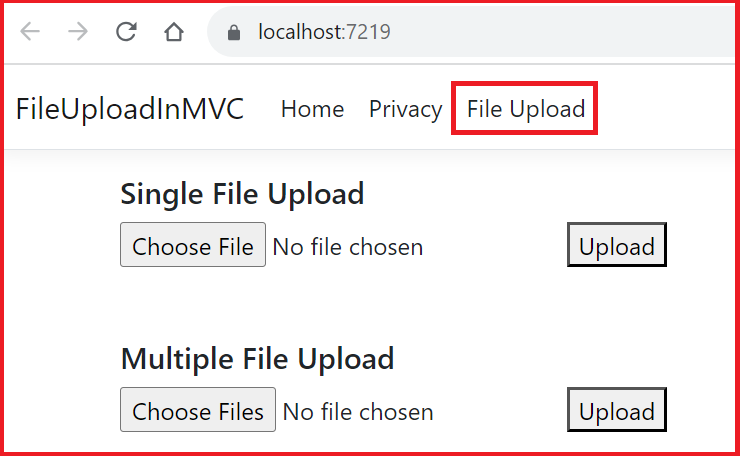
We can check the status of the Rebuild in the output (Build) window as shown below



After the Rebuild is successful, launch the application by selecting “Start Without Debugging” under “Debug” section as shown below

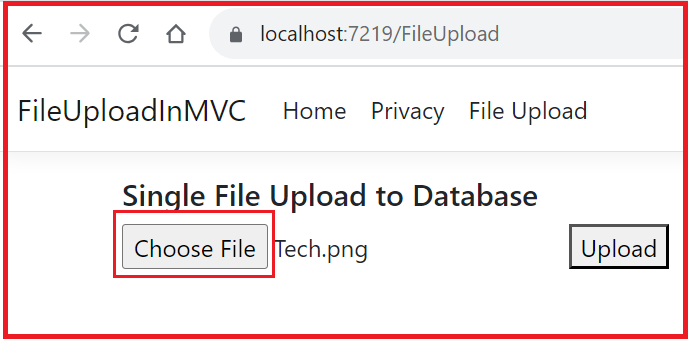


Once the application is launched, it will load the Index view of the HomeController as shown below

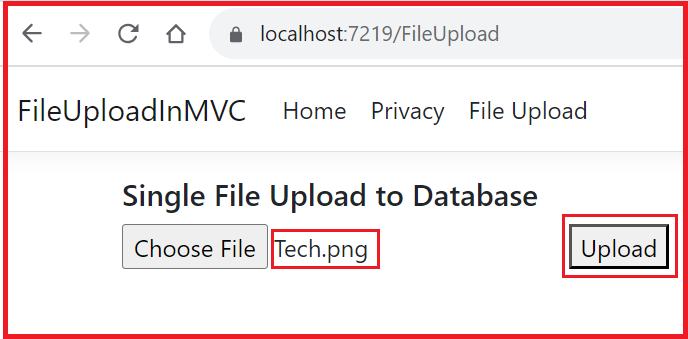


Notice that now there is a new navbar item “File Upload” which we have added in this article as highlighted in the above image.

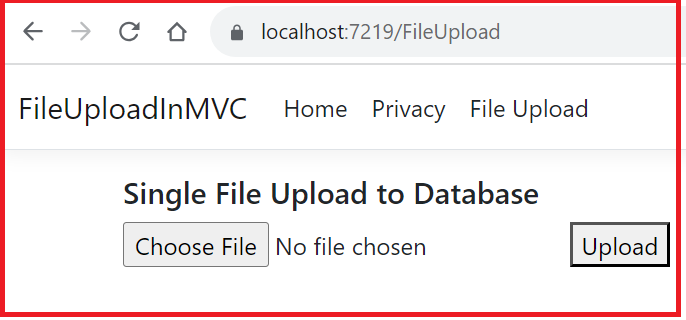
Click on the “FileUpload” navbar item to navigate to Index view of FileUpload controller.



Click on “Choose File” button and select a file to upload. Then click on the “Upload” button as shown below

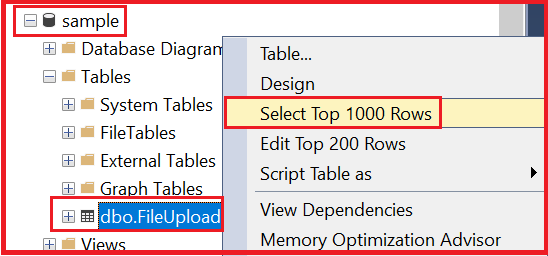


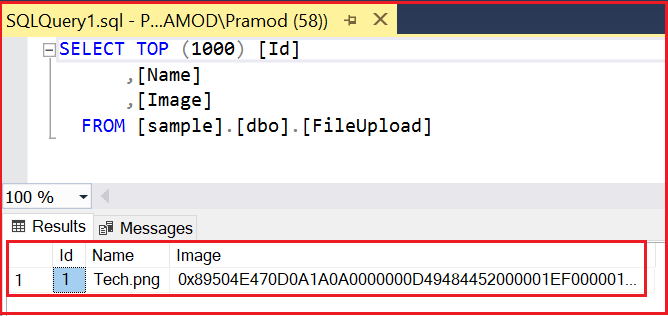
On click of “Upload” button, the file will be processed and saved in the database, and we redirect the application to index view as shown below



Now let’s check the “FileUpload” table in “sample” database to see whether the record is available

1. Right click on the table and chose “Select Top 1000 Rows” as shown below





Notice that the file has been successfully saved in the database as shown above.

In this article we learnt how to save the uploaded file to the database. Please let me know your thoughts in the comments section. In the next article I will discuss how to read the file from database and display it on the view.