

FIT5032 Working with File Upload

Chief Examiner: ABM Russel

POST CLASS ACTIVITIES

TOPICS

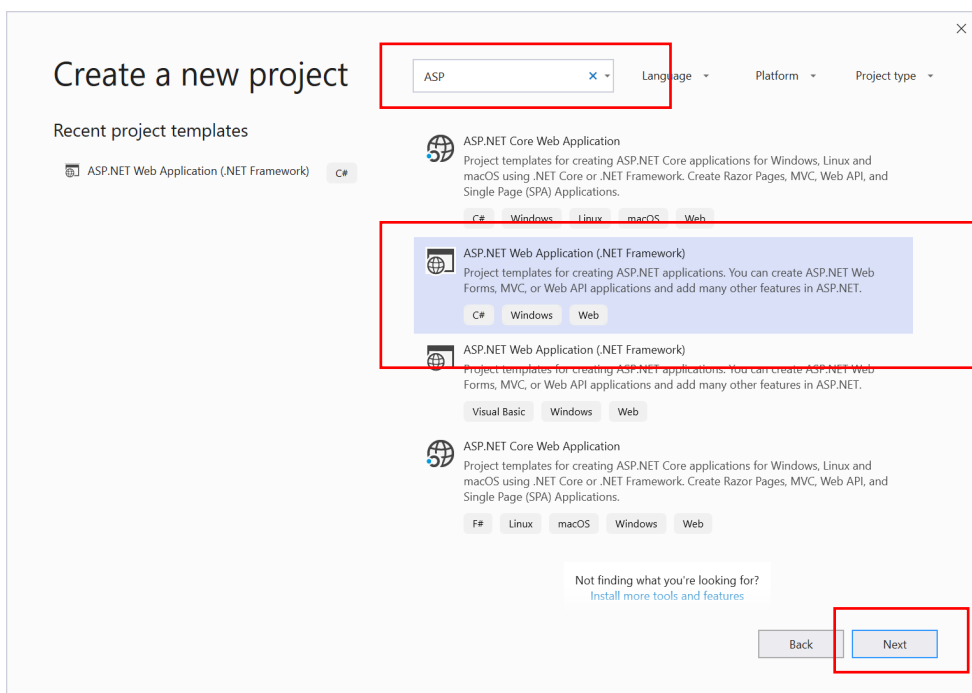
- Upload image files (e.g. png & jpeg)
- Store the path of the uploaded file on the server

It is not compulsory to complete this tutorial. It functions as a supplementary material to showcase how to upload files in an ASP.NET application.

STUDIO ACTIVITIES

Step 1

Please remember that it is possible to zoom into the images in this document.



Step 2

Configure your new project

ASP.NET Web Application (.NET Framework) C# Windows Web

Project name


Location

Solution name ⓘ


☐ Place solution and project in the same directory

Framework


Create a new ASP.NET Web Application

**Empty**


An empty project template for creating ASP.NET applications. This template does not have any content in it.

**Web Forms**


A project template for creating ASP.NET Web Forms applications. ASP.NET Web Forms lets you build dynamic websites using a familiar drag-and-drop, event-driven model. A design surface and hundreds of controls and components let you rapidly build sophisticated, powerful UI-driven sites with data access.

**MVC**

A project template for creating ASP.NET MVC applications. ASP.NET MVC allows you to build applications using the Model-View-Controller architecture. ASP.NET MVC includes many features that enable fast, test-driven development for creating applications that use the latest standards.

**Web API**

A project template for creating RESTful HTTP services that can reach a broad range of clients including browsers and mobile devices.

**Single Page Application**

A project template for creating rich client side JavaScript driven HTML5 applications using ASP.NET Web API. Single Page Applications provide a rich user experience which includes client-side interactions using

Authentication

No Authentication
[Change](#)

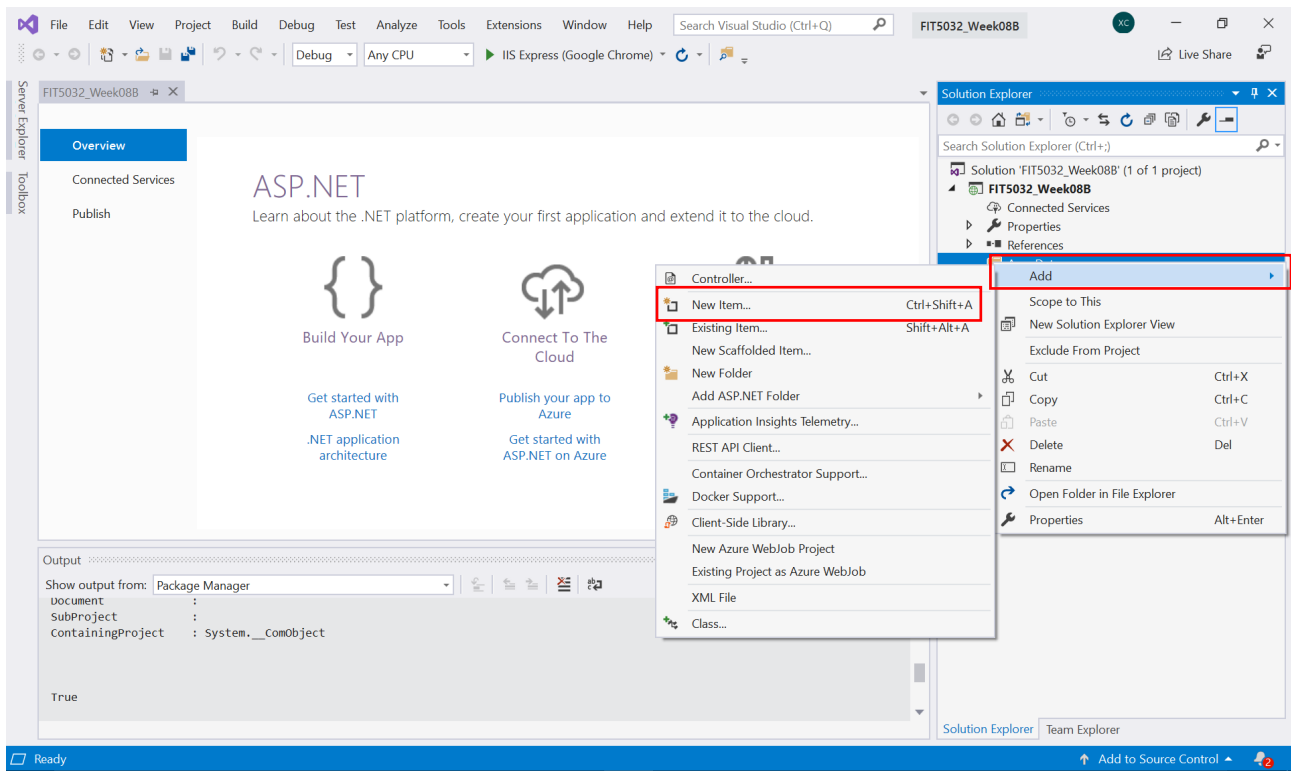
Add folders & core references

☐ Web Forms
☒ MVC
☐ Web API

Advanced

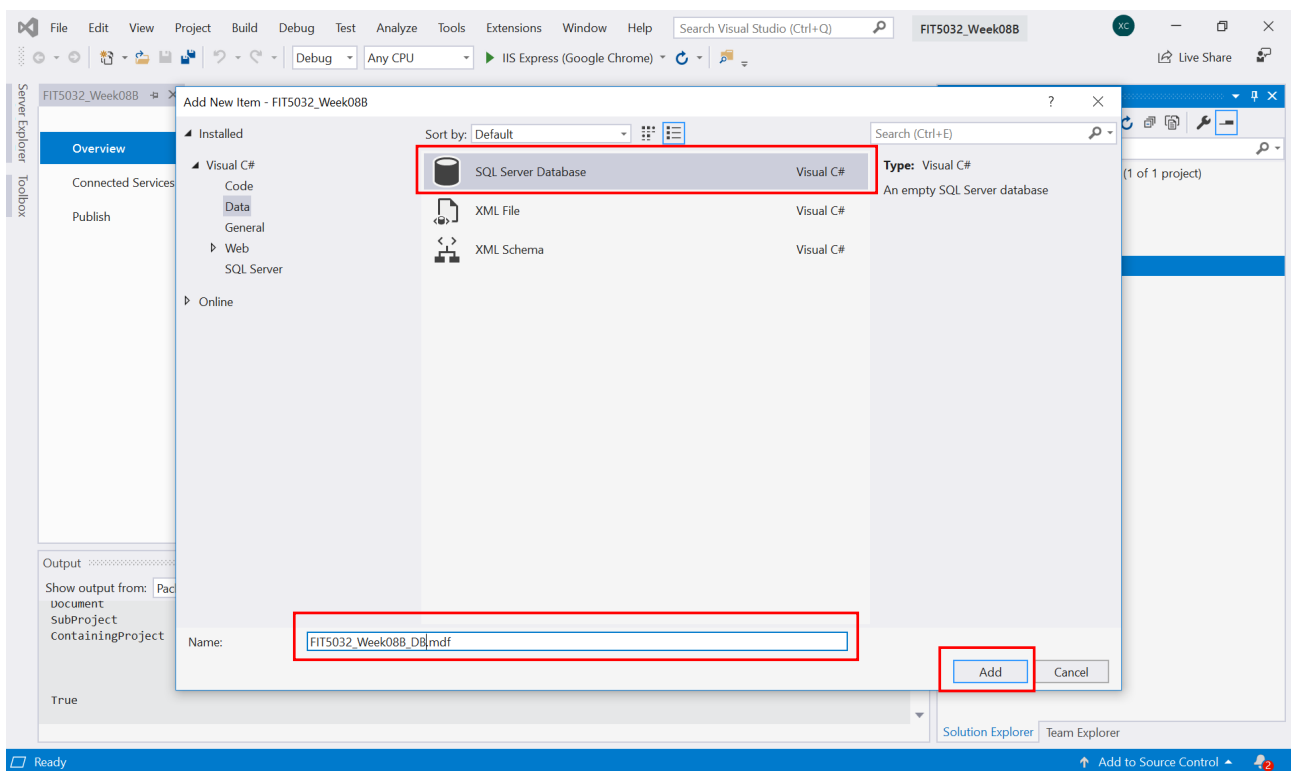
☒ Configure for HTTPS
☐ Docker support
(Requires [Docker Desktop](#))
☐ Also create a project for unit tests

Step 3



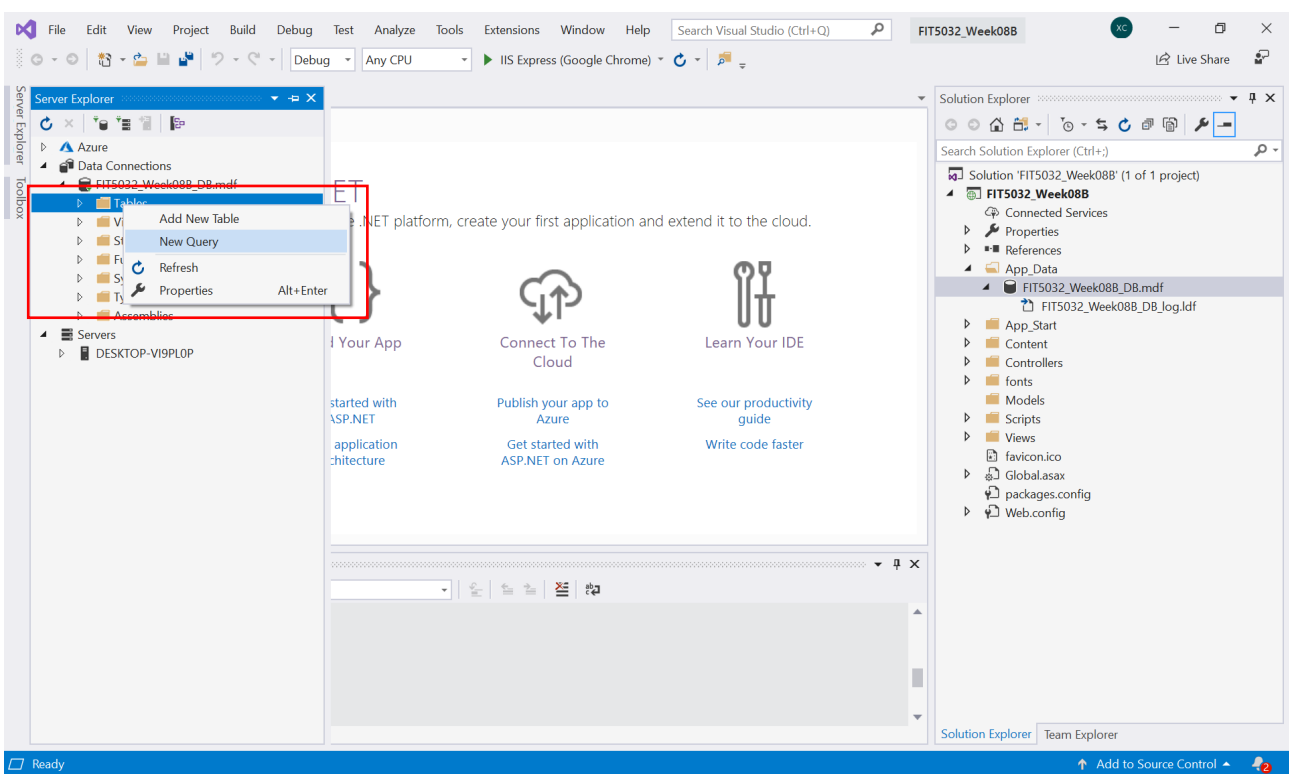
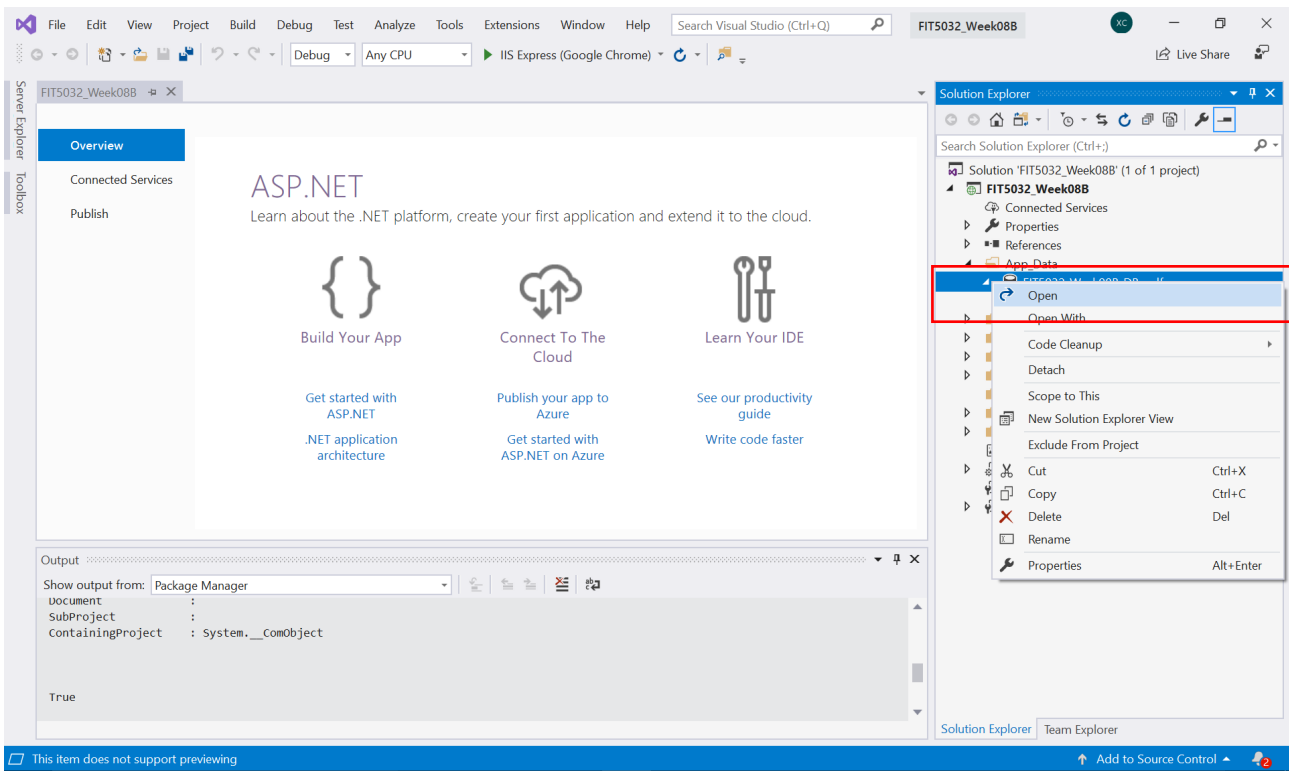
Step 4

I will use a database first approach to create a simple database to store my images.



Step 5

Here, there are several approaches that can be used. I can choose to store the images in my database as either a Binary Large Object (BLOB) or the path to the image itself. However, the more common approach is to **store the path to the image itself on the server**. You can choose to use a BLOB instead if you want to however, it is not the preferred approach. If you have a file server, you can dedicate the server just for this purpose.



The DDL to create the table is provided as follows.

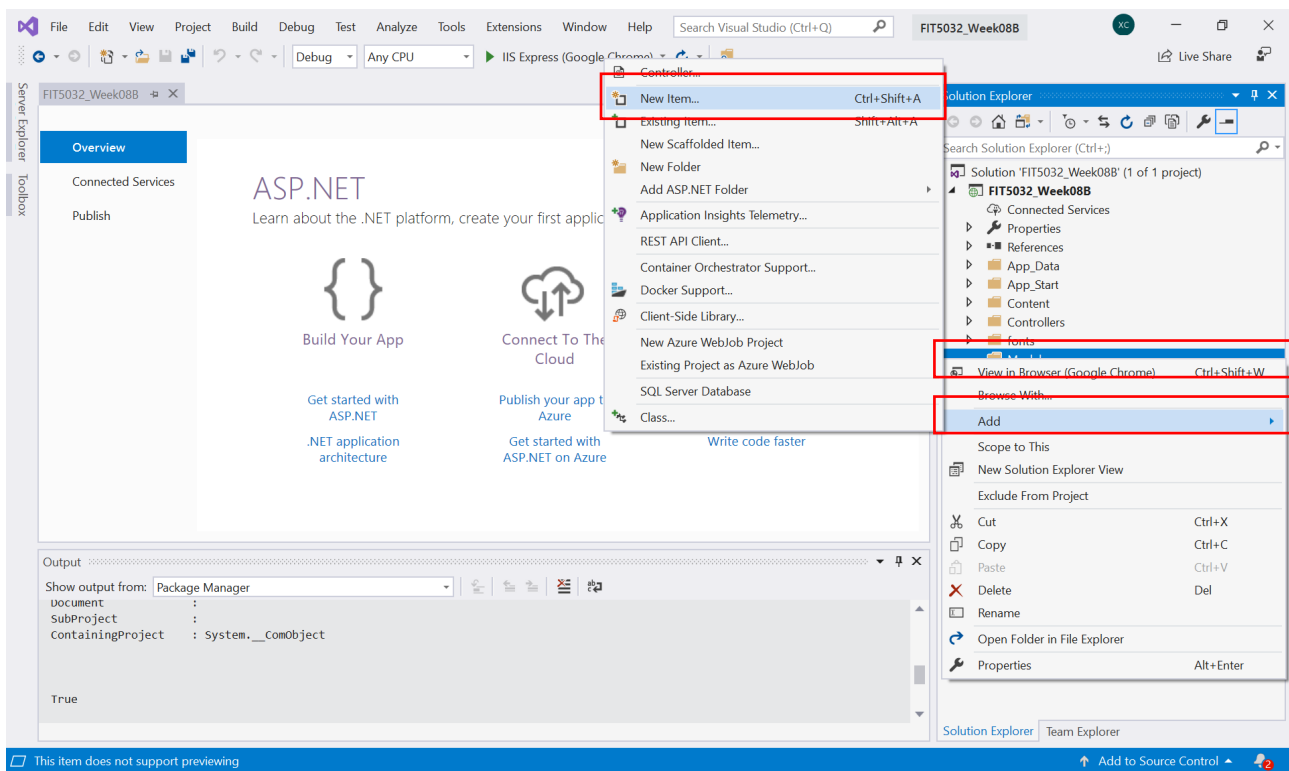
```
CREATE TABLE [dbo].[Images]
(
    [Id] INT NOT NULL PRIMARY KEY IDENTITY(1,1),
    [Path] VARCHAR(50) NOT NULL,
    [Name] VARCHAR(50) NOT NULL
)
```

Please remember to execute the SQL query before the following steps.

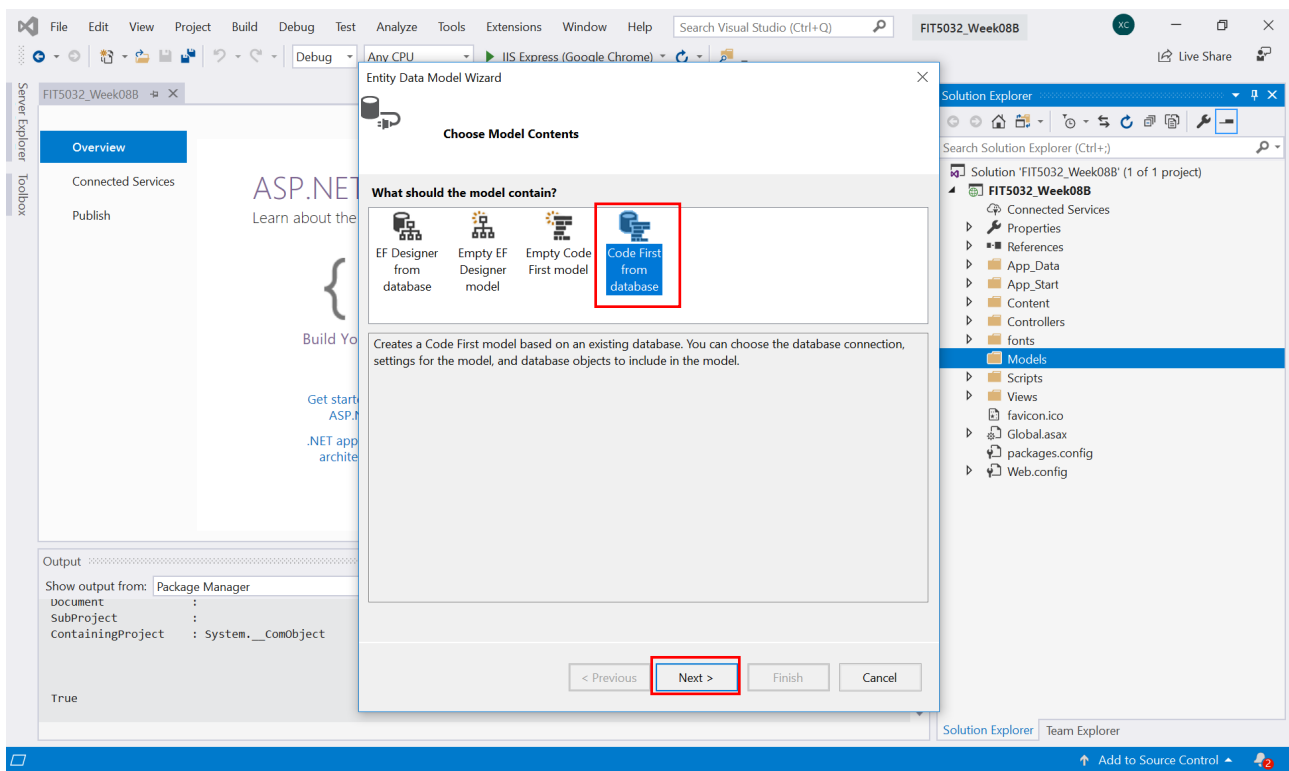
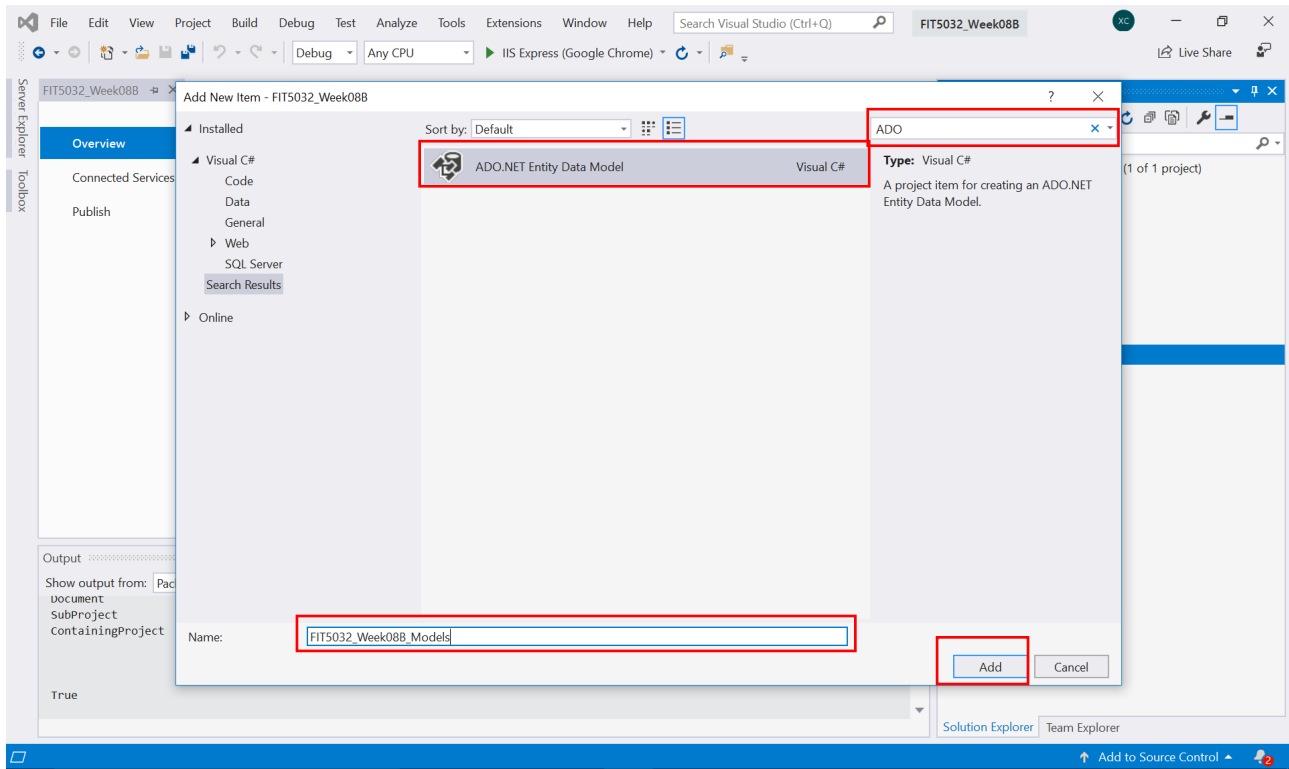
Step 6

After that, you can use the automatic generation feature to create the models, controllers, and views. However, you will need to make changes so that it is working as intended.

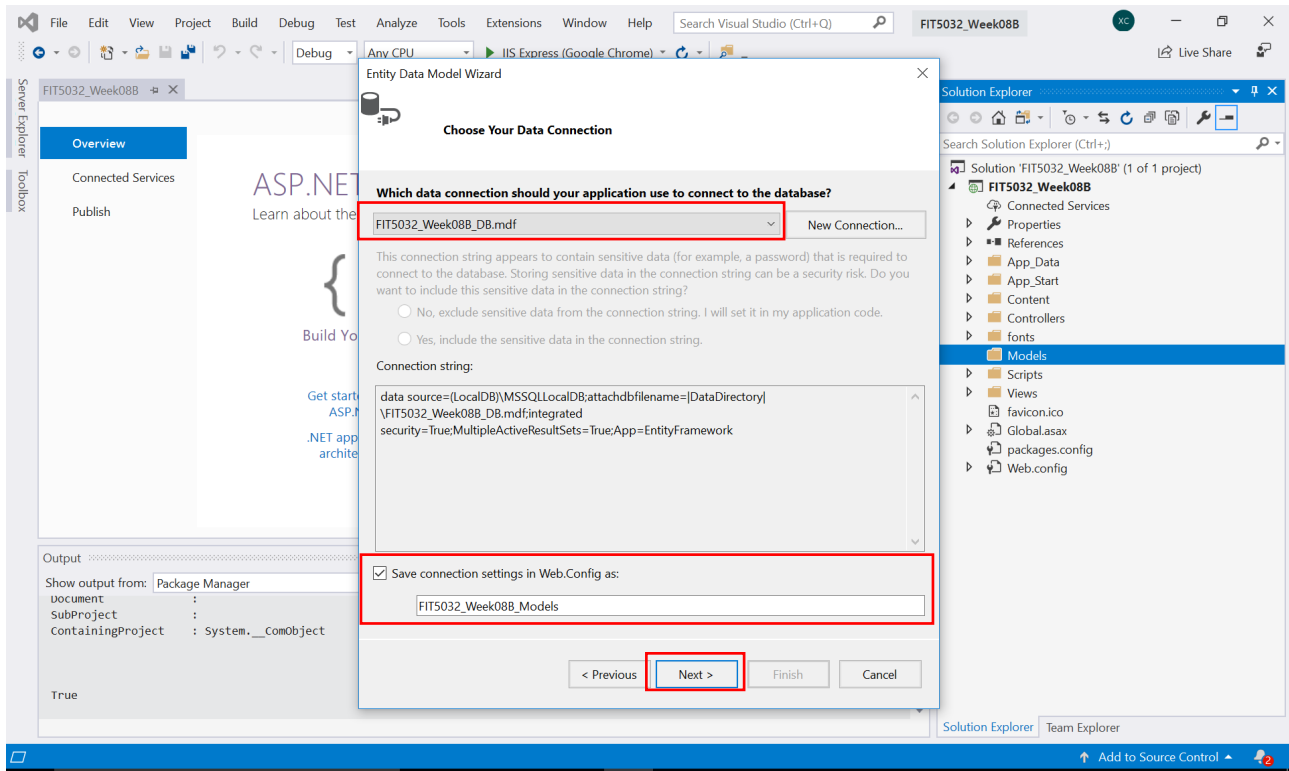
Right Click Models → Add → New Item



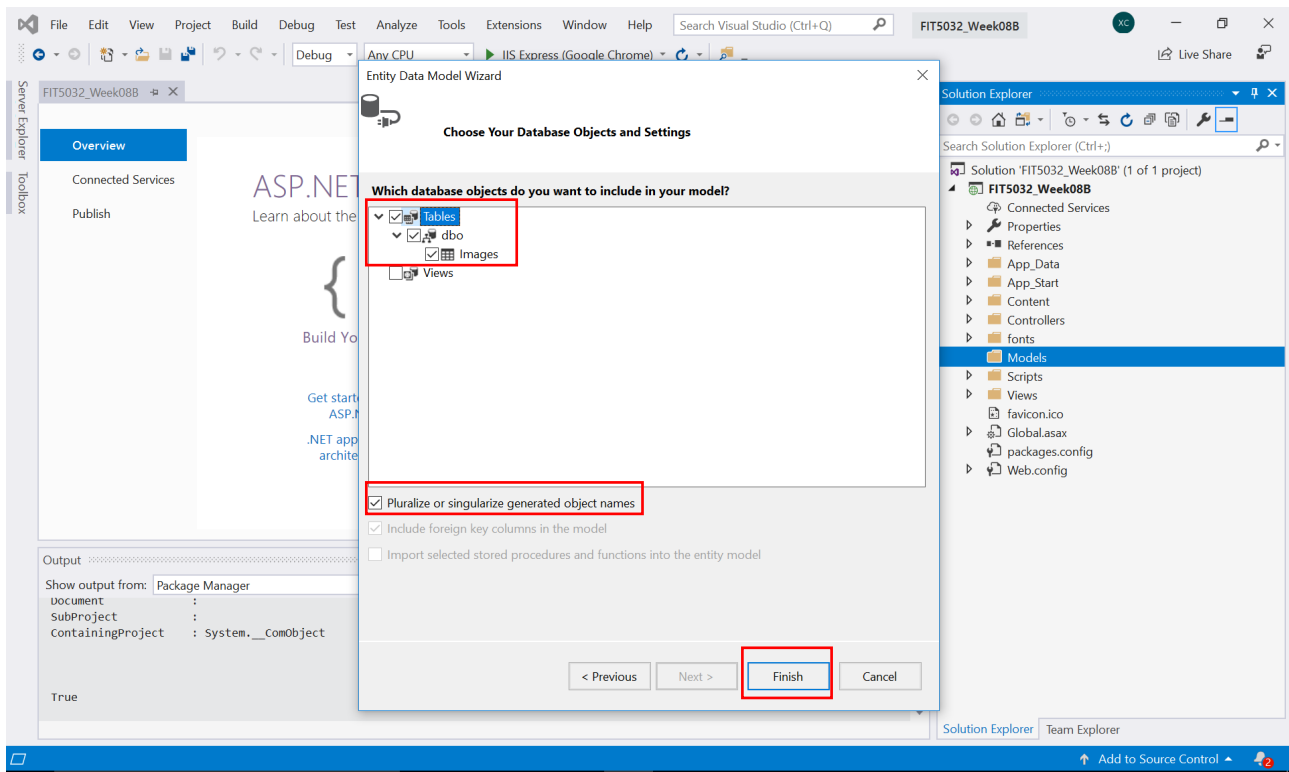
Step 7



Step 8

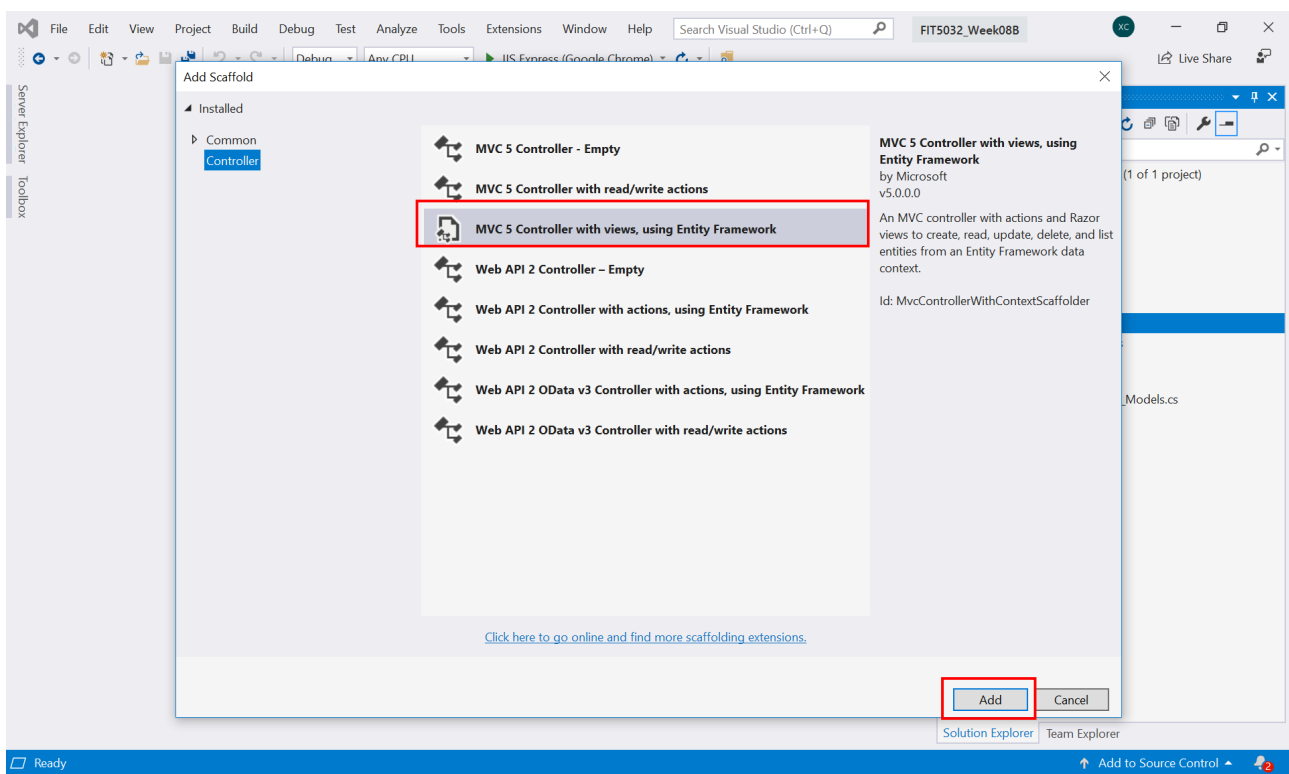
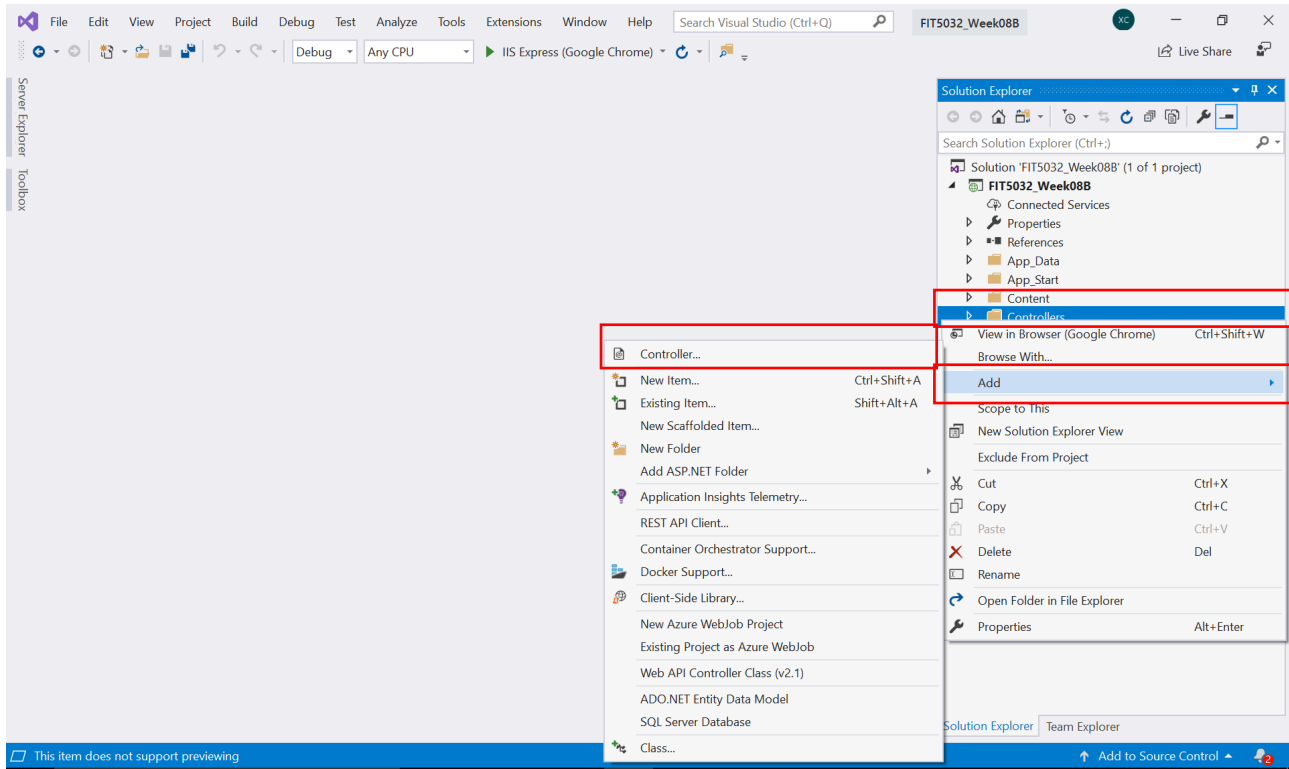


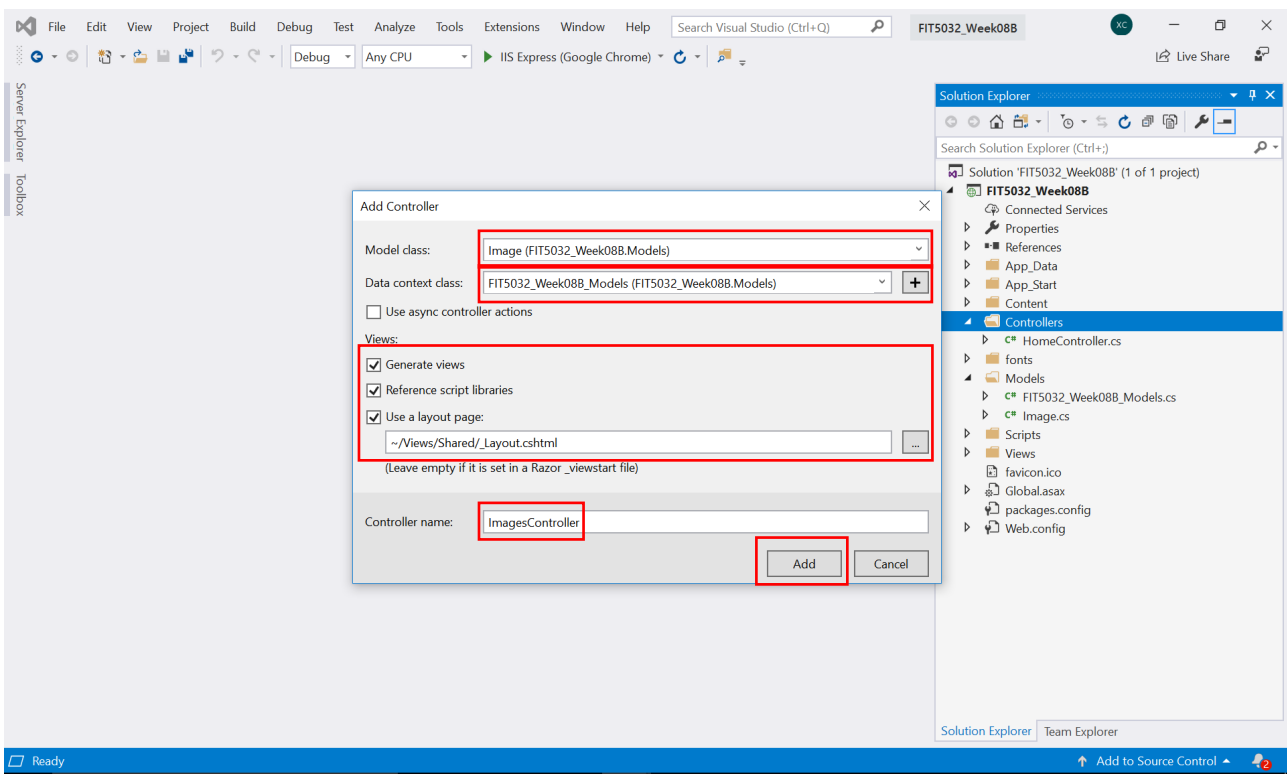
Step 9



Step 10

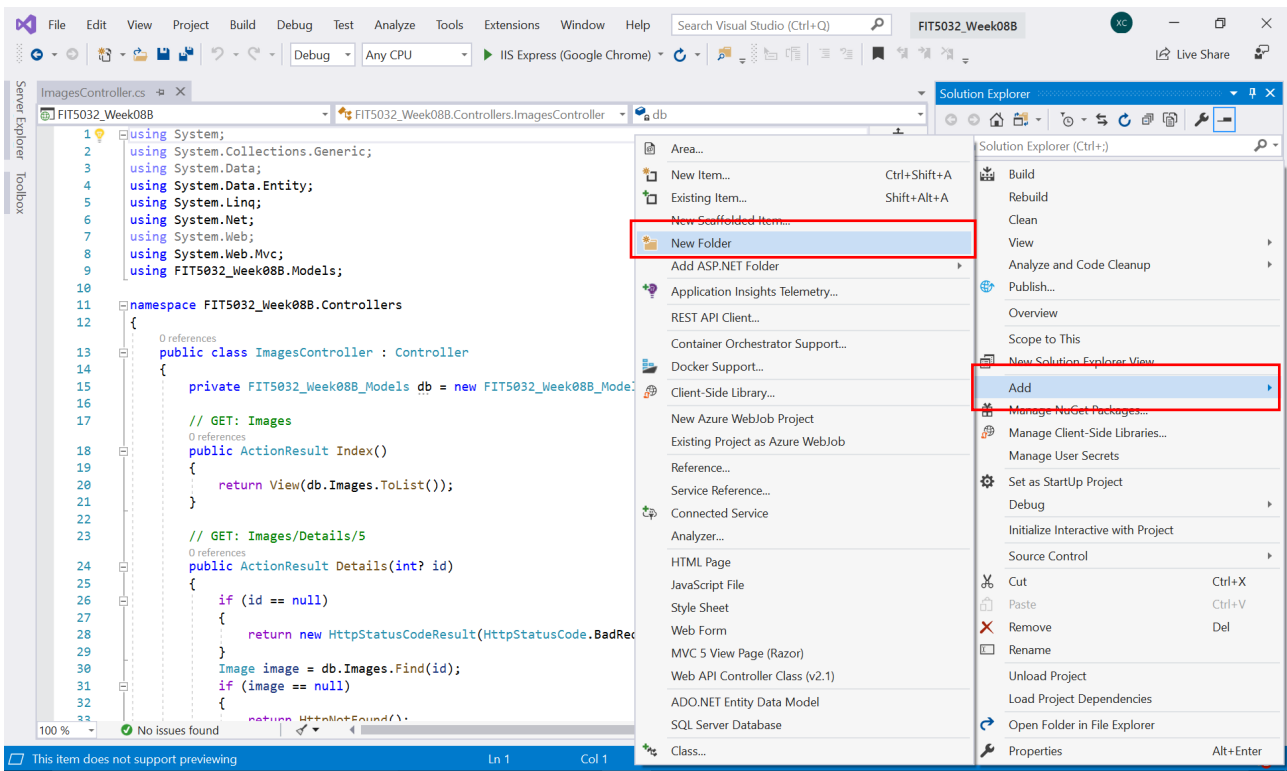
Please remember to Rebuild your project after Step 9.

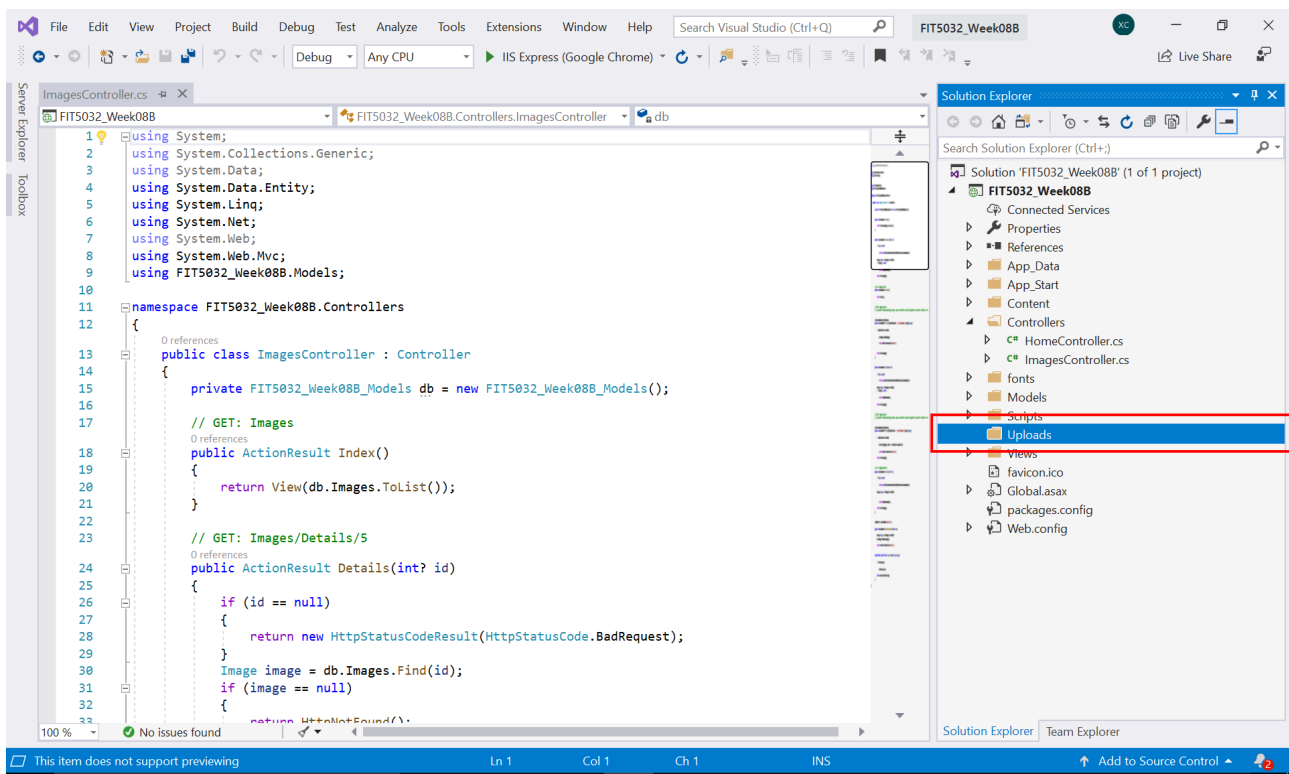




Step 11

Create a folder called "Uploads", this folder will be used to store all the uploaded images.





Step 12

You will need to make the following changes in **ImagesController.cs**.

```
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Create([Bind(Include = "Id,Name")] Image image, HttpPostedFileBase
postedFile)
{
    ModelState.Clear();
    var myUniqueFileName = string.Format(@"{0}", Guid.NewGuid());
    image.Path = myUniqueFileName;
    TryValidateModel(image);

    if (ModelState.IsValid)
    {
        string serverPath = Server.MapPath("~/Uploads/");
        string fileExtension = Path.GetExtension(postedFile.FileName);
        string filePath = image.Path + fileExtension;
        image.Path = filePath;
        postedFile.SaveAs(serverPath + image.Path);

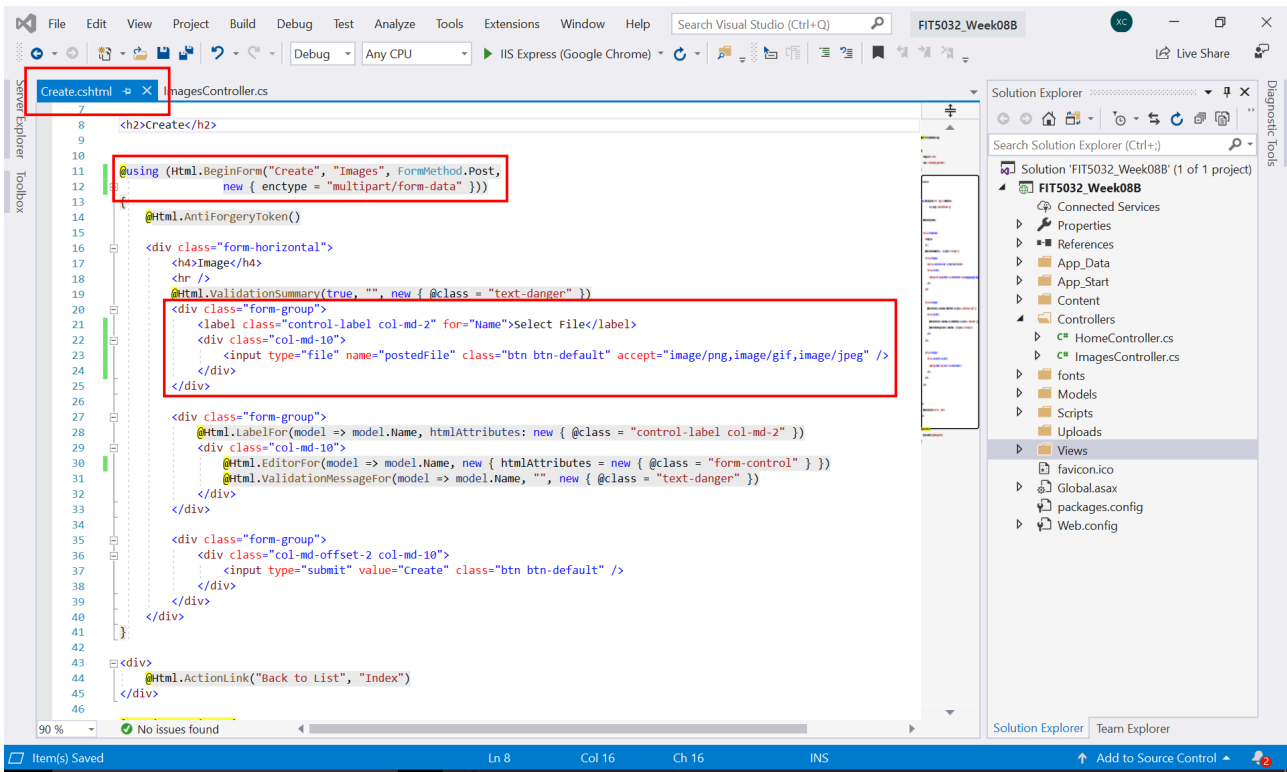
        db.Images.Add(image);
        db.SaveChanges();
        return RedirectToAction("Index");
    }

    return View(image);
}
```

Step 13

You will also need to make changes to the **Create.cshtml of the Images View**.

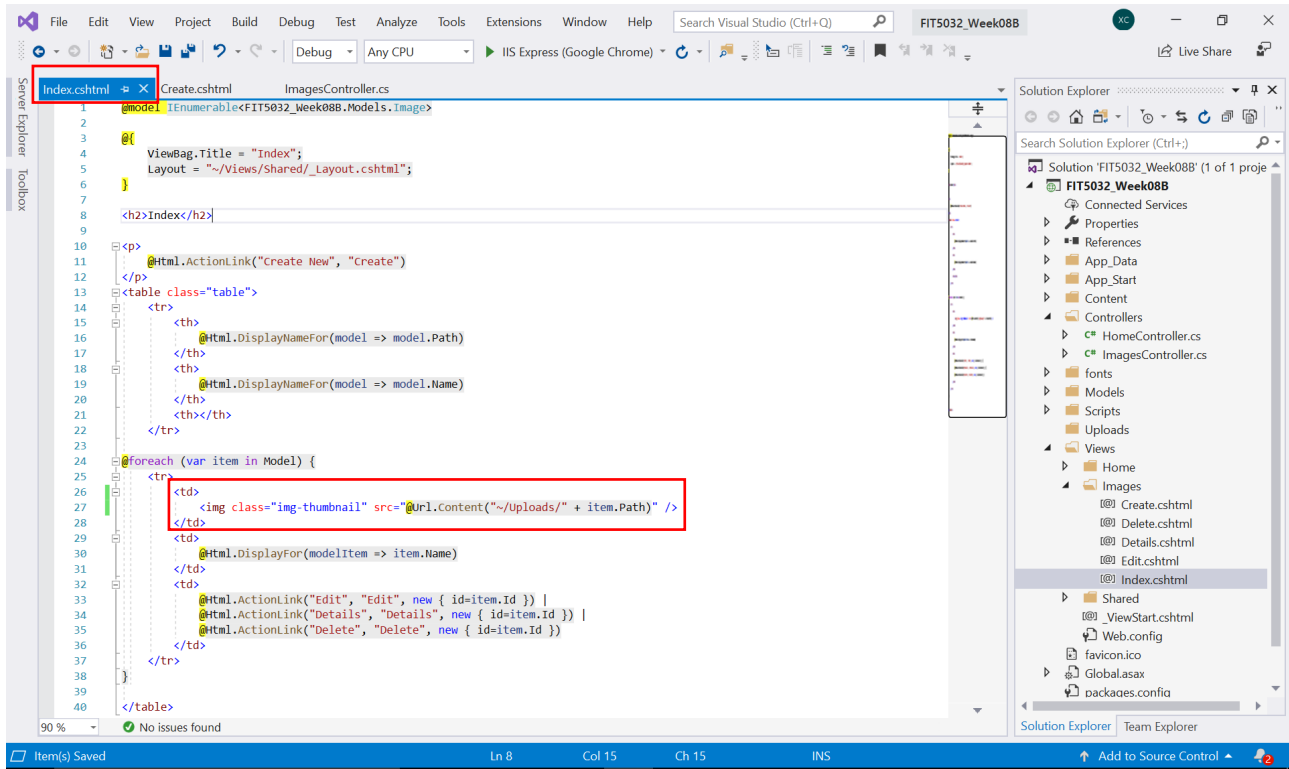
File uploading requires the form to have an enctype thus the form needs to be changed, and we remove the old path input and introduce an input for a file instead (The codes for them are in the screenshot). Type it out based on the screenshot below.



```
7 <h2>Create</h2>
8
9
10
11 @using (Html.BeginForm("Create", "Images", FormMethod.Post,
12     new { enctype = "multipart/form-data" }))
13
14 @Html.AntiForgeryToken()
15
16 <div class="form-horizontal">
17     <h4>Image</h4>
18     <hr />
19     @Html.ValidationSummary(true, "", new { @class = "text-danger" })
20     <div class="form-group">
21         <label class="control-label col-md-2" for="Name">Select File</label>
22         <div class="col-md-10">
23             <input type="file" name="postedFile" class="btn btn-default" accept="image/png,image/gif,image/jpeg" />
24         </div>
25     </div>
26
27     <div class="form-group">
28         @Html.LabelFor(model => model.Name, htmlAttributes: new { @class = "control-label col-md-2" })
29         <div class="col-md-10">
30             @Html.EditorFor(model => model.Name, new { htmlAttributes = new { @class = "form-control" } })
31             @Html.ValidationMessageFor(model => model.Name, "", new { @class = "text-danger" })
32         </div>
33     </div>
34
35     <div class="form-group">
36         <div class="col-md-offset-2 col-md-10">
37             <input type="submit" value="Create" class="btn btn-default" />
38         </div>
39     </div>
40
41 </div>
42
43 <div>
44     @Html.ActionLink("Back to List", "Index")
45 </div>
46
```

Step 14

At the **Index.cshtml** of the **Images View**, please type out the changes.



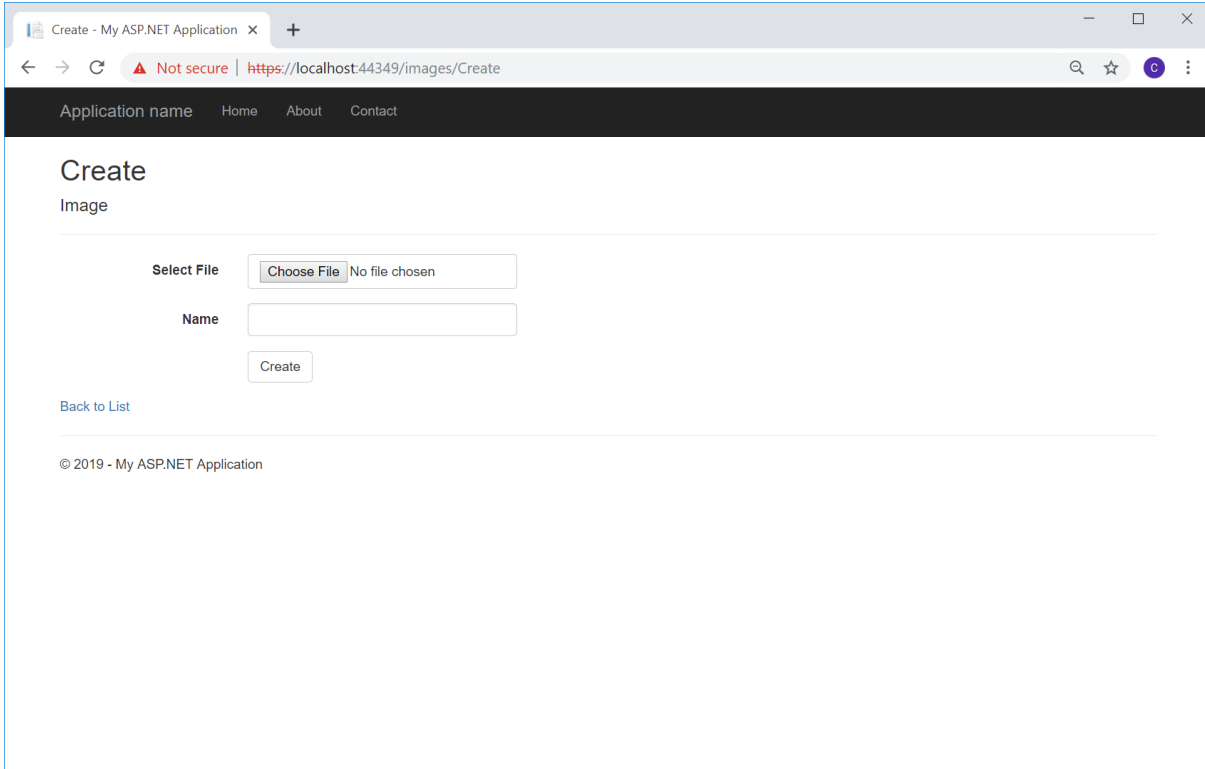
```
1  @model IEnumerable<FIT5032_Week08B.Models.Image>
2
3  @{
4      ViewBag.Title = "Index";
5      Layout = "~/Views/Shared/_Layout.cshtml";
6  }
7
8  <h2>Index</h2>
9
10 <p>
11     @Html.ActionLink("Create New", "Create")
12 </p>
13 <table class="table">
14 <tr>
15 <th>
16     @Html.DisplayNameFor(model => model.Path)
17 </th>
18 <th>
19     @Html.DisplayNameFor(model => model.Name)
20 </th>
21 </tr>
22 <tr>
23 <td>
24     @foreach (var item in Model) {
25 <tr>
26 <td>
27     
28 </td>
29 <td>
30     @Html.DisplayFor(modelItem => item.Name)
31 </td>
32 <td>
33     @Html.ActionLink("Edit", "Edit", new { id=item.Id }) |
34     @Html.ActionLink("Details", "Details", new { id=item.Id }) |
35     @Html.ActionLink("Delete", "Delete", new { id=item.Id })
36 </td>
37 </tr>
38 </tr>
39 </table>
40
```

Step 15

You can also introduce a hyperlink at the shared layout so that the user can navigate to the Images View.

Upon the completion of this, you can try running the solution. You should see something like the images below.

At the **/Images/Create**:



Create - My ASP.NET Application

Application name Home About Contact

Create

Image

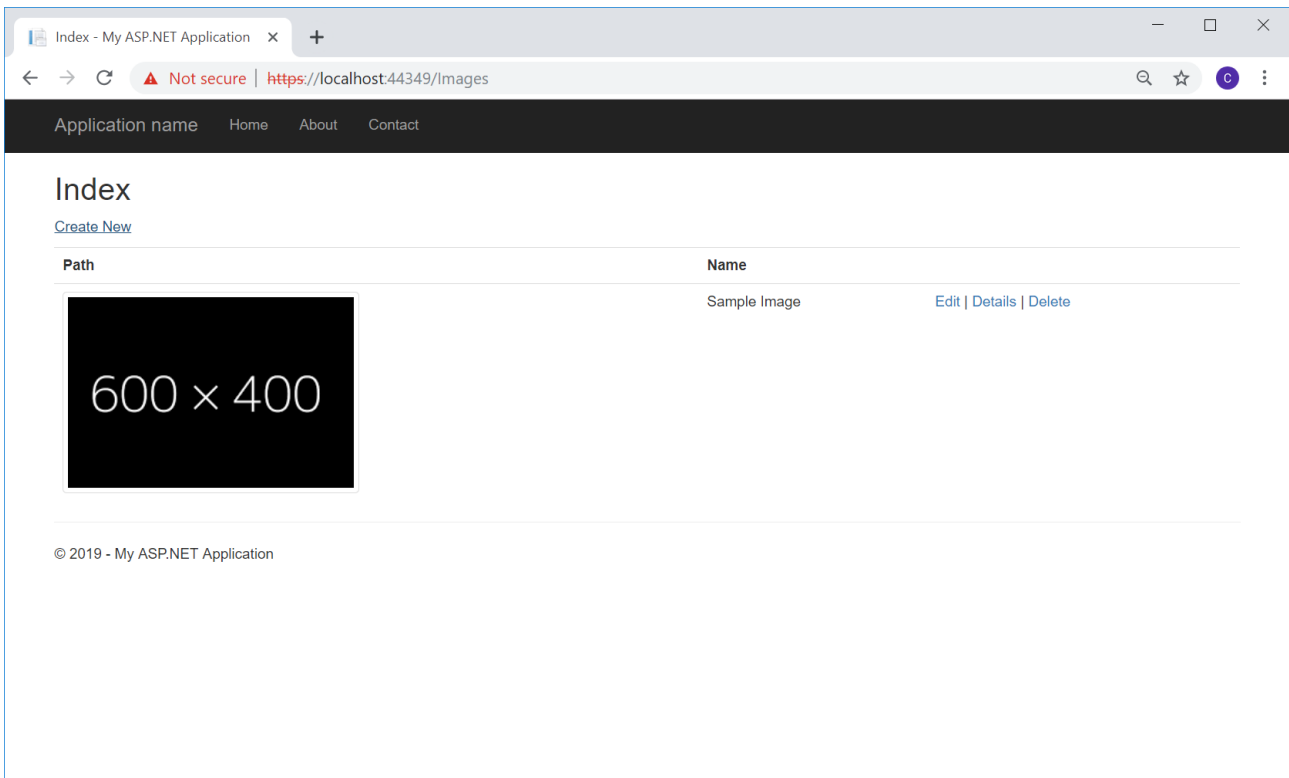
Select File No file chosen

Name

[Back to List](#)

© 2019 - My ASP.NET Application

At the **/Images**:




Index - My ASP.NET Application

Application name Home About Contact

Index

[Create New](#)

Path	Name	
	Sample Image	Edit Details Delete

© 2019 - My ASP.NET Application

EXPLANATION

Here, I took a very simplistic approach to achieve this goal, and there are quite of numbers of issues not addressed. For example, you should introduce more validations which will validate the size of the image as well.

You can also figure out how to fix the "edit" feature so that it will be working as intended. If you want to, you can also further improve upon this project. If you want to know how to improve this project, please ask your tutor how to do so.

CONCLUSION

Upon the completion of this supplementary material, you should be able to understand how upload of image is handled in a simple manner.

Another approach would be to use a BLOB to store the image itself in the database. However, with this approach, it would be harder to see the images as it needs to be queried instead.