**Displaying the uploaded file**

In the previous article we have seen how to save the uploaded file into the database. In this article we will discuss how to display the uploaded image on the UI.

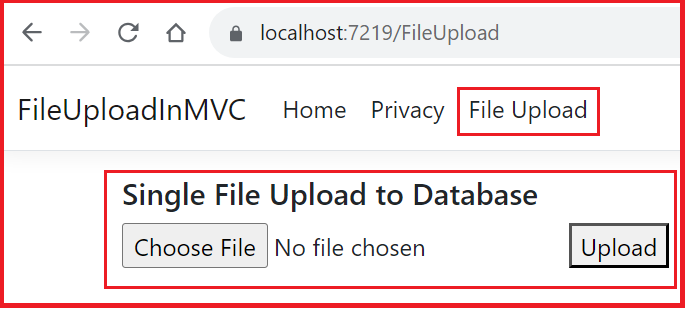
In this article we will learn how to display the file as soon as it is uploaded.

**Display the file as soon as it is uploaded**

In this article we will use the same sample application that we have been using for the previous articles.

In the previous article we made code changes to save the uploaded file to database. In this article we will continue from where we left off.

In the previous article we have added a new nav bar item “File Upload”. We have created a new controller “**FileUploadController**” and view “**Index.cshtml**” under Views --> FileUpload folder. We have added **file type** input control and the submit button to post the form as shown below. In the controller class we have added the action method “**SingleFileUploadToDatabase**” to process and save the uploaded file to database.

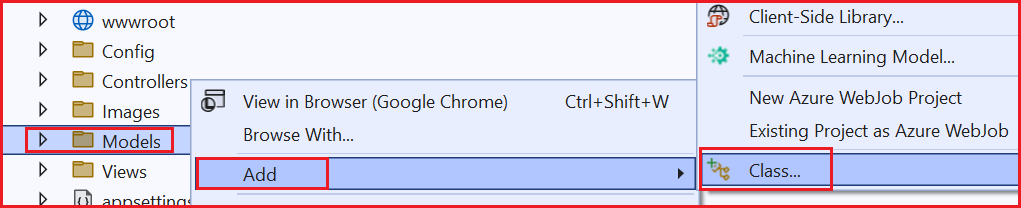


To display the file on the page, we create a new view model class which contains properties to hold the file content and its details. When the file is uploaded, in the controller action method we update the model with the uploaded file content and other details and pass the model to the view. In the view we set the source of the image control to the filecontent which is available in the model.

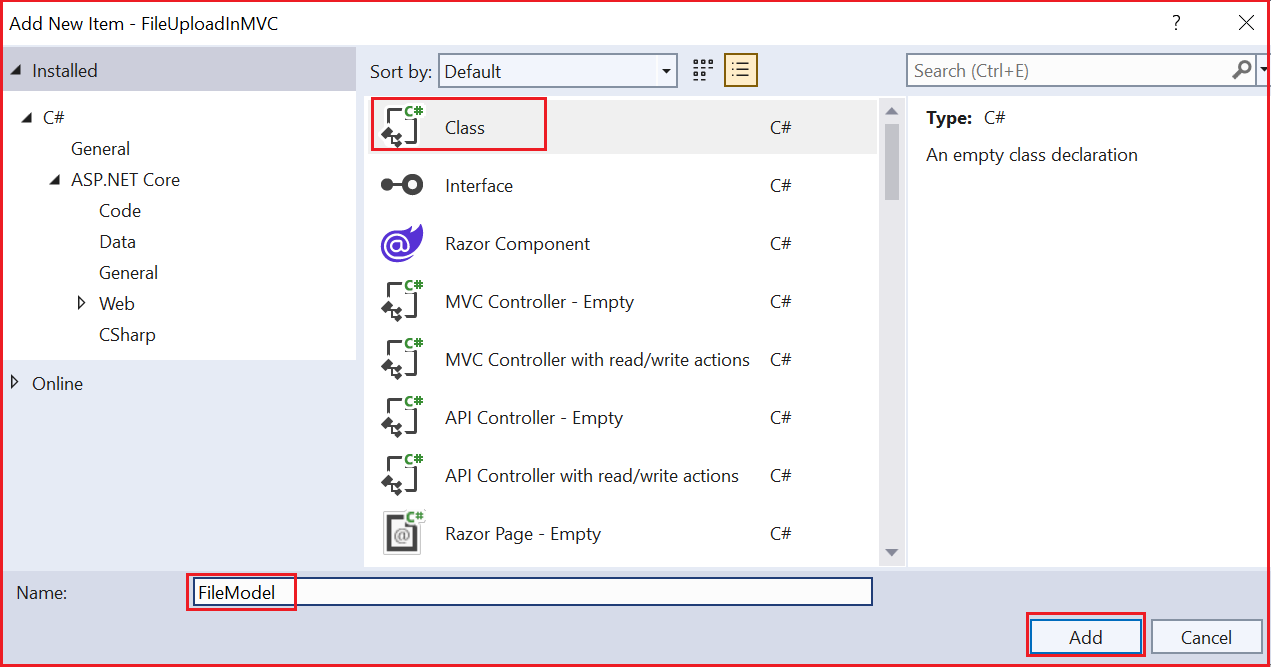
**FileModel.cs file (View Model)**

I have created a new model class “FileModel.cs” file. This is the model for the index view of “FileUpload” controller.

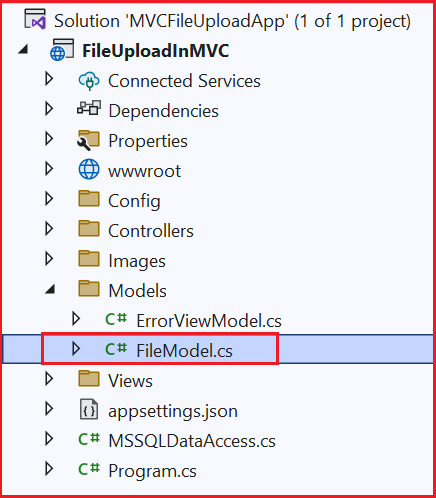
1. Right click on the “Models” folder, select “Add” and select “Class” as shown below



1. In the “Add New Item” popup, give Name as “FileModel” and click on add as shown below



1. FileModel.cs class will be created as shown below



1. I have changed the access specifier of the “FileModel.cs” class to the public. I have added three properties “fileContent” of type byte array, “fileName” of type string and “file” of type IFile as shown below

“  
 public class FileModel

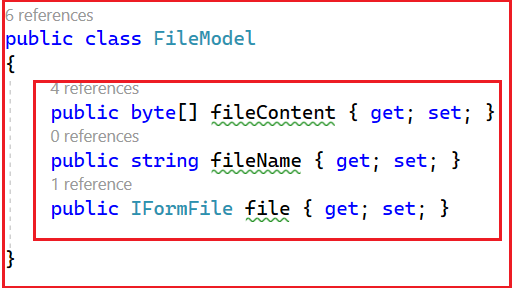
{

public byte[] fileContent { get; set; }

public string fileName { get; set; }

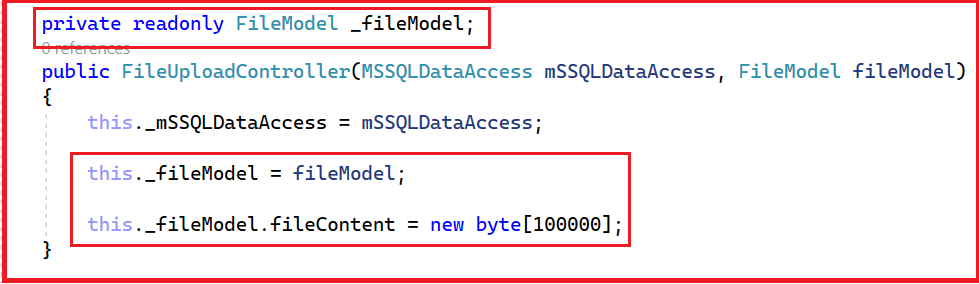
public IFormFile file { get; set; }

}  
”



**FileUploadController changes**

Added a new read-only property for the FileModel object. Assigning value to it in the constructor using dependency injection. After that initializing the “fileContent” property of FileModel object with a byte array of length equal to 100000 (keeping the byte array size to a large value)



“

private readonly FileModel \_fileModel;

public FileUploadController(MSSQLDataAccess mSSQLDataAccess, FileModel fileModel)

{

this.\_mSSQLDataAccess = mSSQLDataAccess;

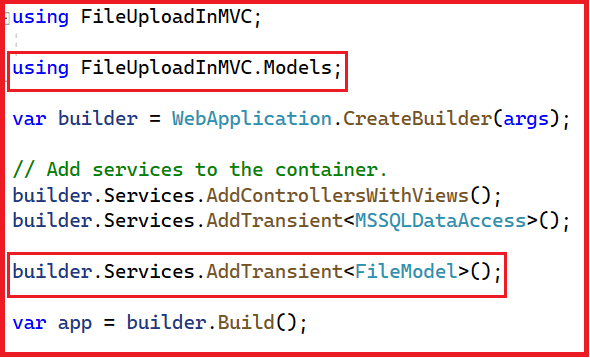
this.\_fileModel = fileModel;

this.\_fileModel.fileContent = new byte[100000];

}

”

For dependency injection to work for “FileModel” class we need to add the transient service of type “FileModel” to the “builder.Services” collection in the “Program.cs” file as shown below. This requires adding a using statement for the namespace “FileUploadInMVC.Models”



“

*using FileUploadInMVC.Models;*

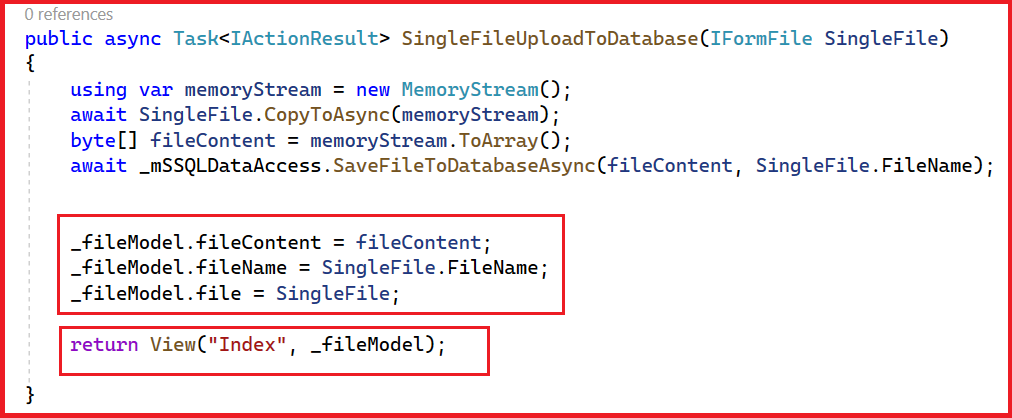
*builder.Services.AddTransient<FileModel>();*

”

Let’s update the “SingleFileUploadToDatabase” action method.

We will update the \_fileModel instance property with the uploaded file values as shown below and then we will return the updated model to the view.

Notice that the code we had written to save the uploaded file to the database remains the same.



“

*\_fileModel.fileContent = fileContent;*

*\_fileModel.fileName = SingleFile.FileName;*

*\_fileModel.file = SingleFile;*

*return View("Index", \_fileModel);*

”

Let's update the index view of “file upload” to display the uploaded image.

In the view, we will add a new image control that can display the image. We will set the source of the image control to the uploaded file content which is passed by the controller in the view model.

**Index.cshtml changes**

“

*<div>*

*<h5 class="divClass">Uploaded image</h5>*

*<img id="imgUploaded" src="@imgSrc" class="imageClass" />*

*</div>*

*<style>*

*.imageClass{*

*width:50%;*

*height:50%;*

*padding-top:5px;*

*}*

*.divClass{*

*padding-top:15px;*

*}*

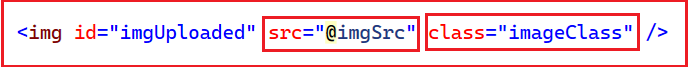
*</style>*

”



1. Added a div tag and header element for “Uploaded image”. Included a css class attribute and assigned it to “divClass” to provide a padding at the top for better UI look.
2. Added an image control <img> which can display the image. Included a css class attribute and assigned it to “imageClass”
3. Added a style tag and defined css classes “imageClass” and “divClass”

Notice that,



I have added a css class for the image control to set the width and height of the image so that the image can be displayed properly on the page.

I have set the source of the image to a variable “@imgSrc”. The image to be displayed will be passed as a model property from the controller. I have defined the “FileModel” as model for the view as shown below.

“

@model FileModel

@{

string imgSrc;

if(@Model?.fileContent != null)

{

var base64 = Convert.ToBase64String(@Model?.fileContent);

imgSrc = String.Format("data:image/gif;base64,{0}", base64);

}

else

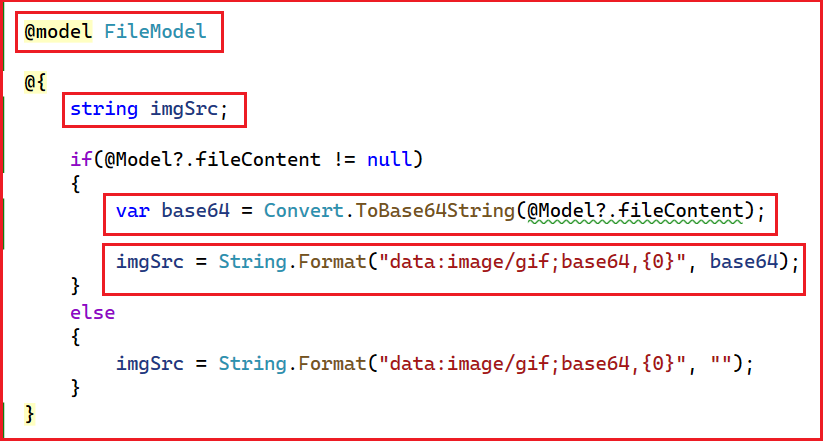
{

imgSrc = String.Format("data:image/gif;base64,{0}", "");

}

}

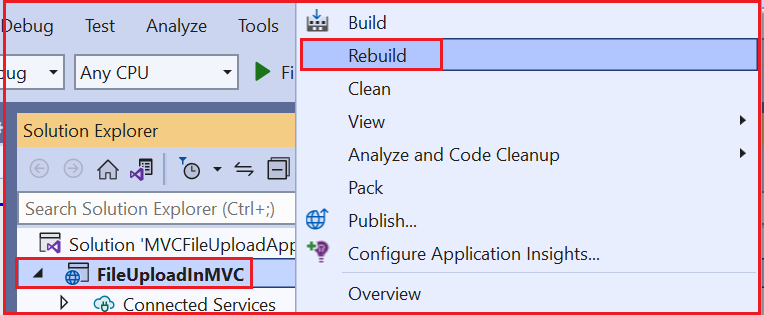
”



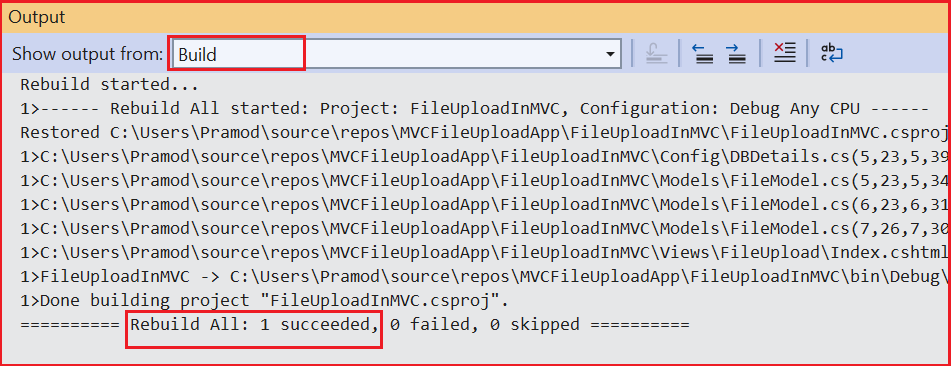
Notice I have defined a string variable “**imgSrc**”. I have added a conditional if else statement to check whether the Model or the Model.FileContent is null. If either of them is null, then else statement will be executed, and no image will be displayed. If both are not null, then we convert the file content which is in byte array format to base 64 string. Then the formatted string which contains the base 64 format of the file content will be assigned to “imgSrc” which is the source of the image control.

With all these changes in place, let’s rebuild the application and run.

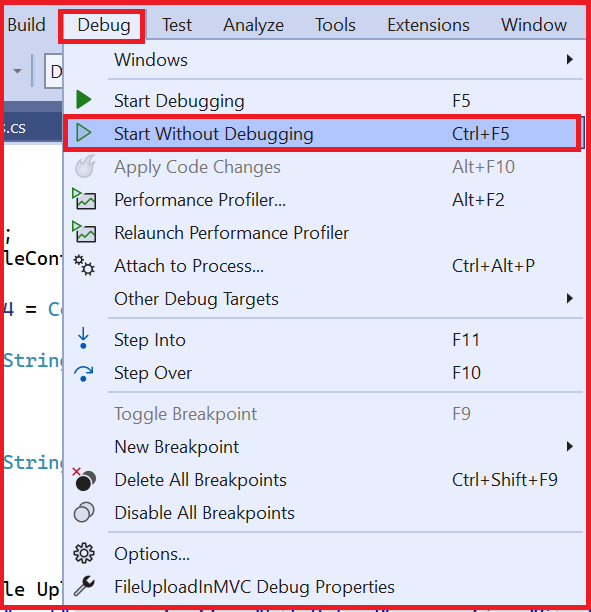
1. Right click on the project and select “Rebuild”



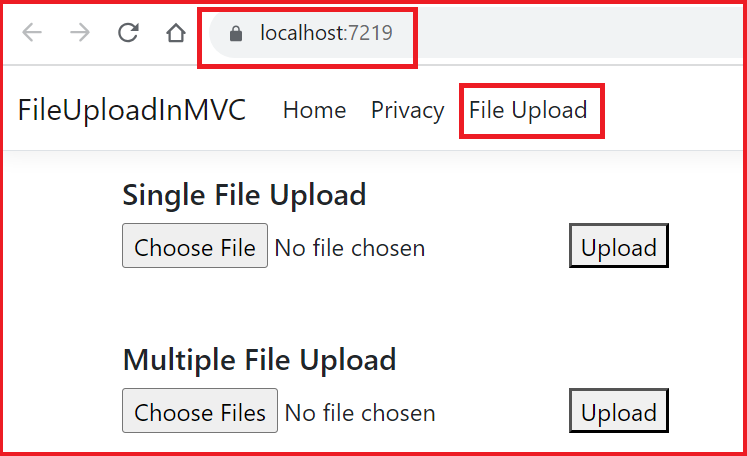
1. We can check the status of the build from the output window



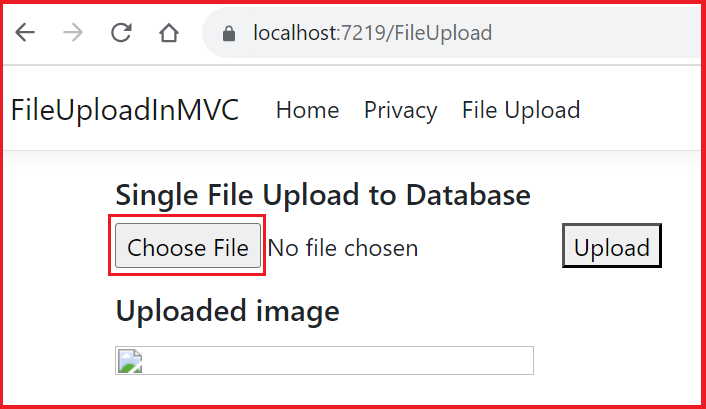
1. Run the application by selecting “Start Without Debugging” from “Debug” section



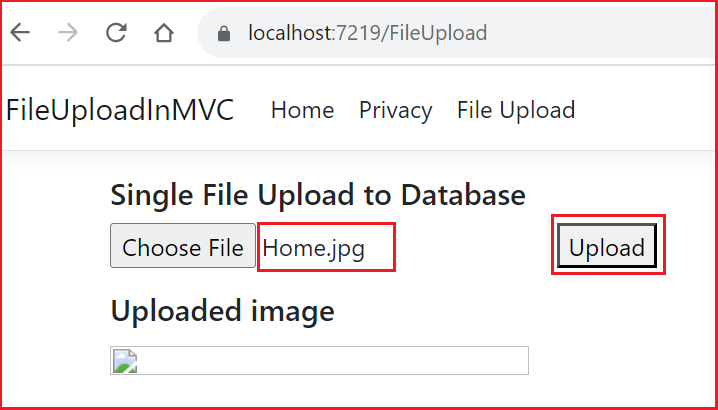
1. The application will display the “index” view of the Home Controller. Click on the “File Upload” nav bar item as highlighted below to navigate to index view of “File upload” controller.



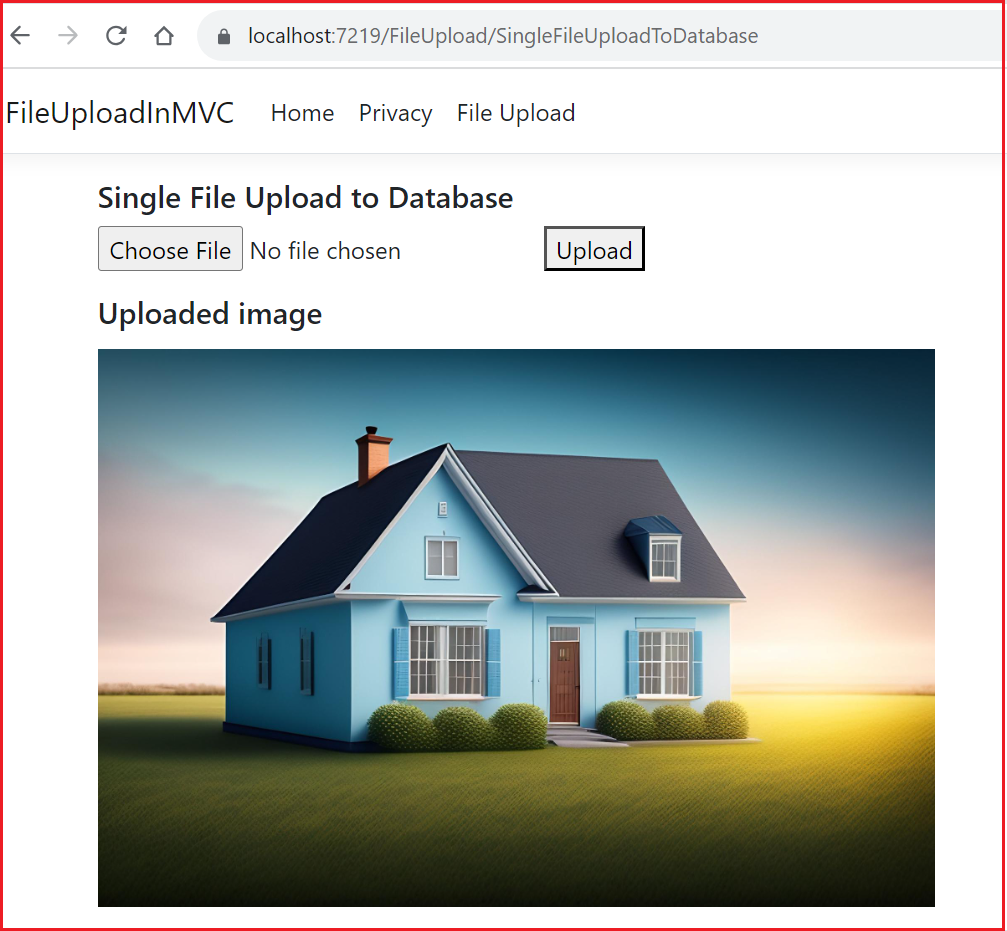
1. Index view of the “FileUpload” controller will be displayed as shown below. Click on “Choose File” to select a file to be uploaded from the explorer.



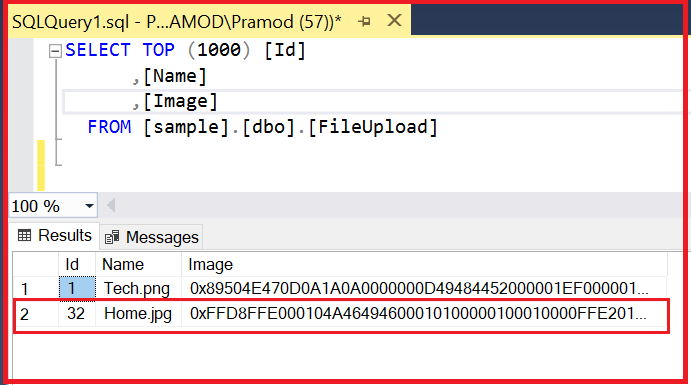
1. After selecting the file, its name will be displayed as highlighted below. Click on “Upload” button to upload the file.



1. The uploaded image will be displayed as shown below

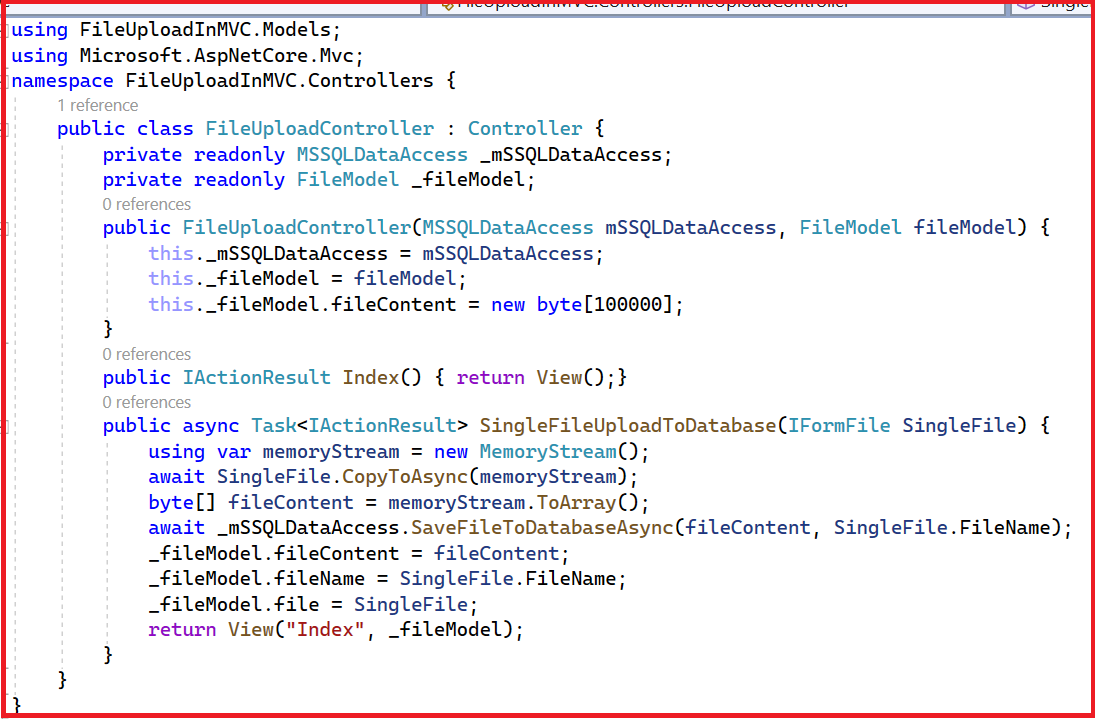


1. The code to save the file to database is still in place, so let’s verify if the file got saved in database.



In this article we learnt how to display the file immediately after it is uploaded. Let me know your thoughts on this in the comments section. In the next article we will discuss how to display the file while visiting a page.

Complete “FileUploadController.cs” file now looks like below



Complete index.cshtml of file upload controller now looks like below

