Capturing Image From Web Cam In ASP.NET Core MVC

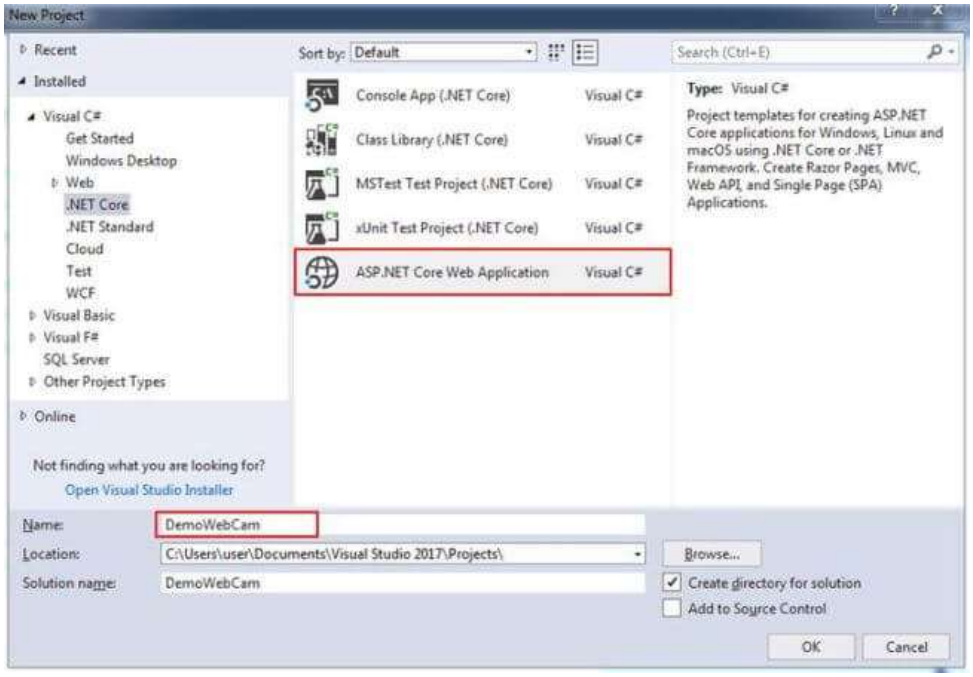
In this article, we are going to learn how to capture a simple image and store it in a folder and database in simple steps. For doing this, we are going to use "*WebcamJS*" JavaScript library created by- Joseph Huckaby also there is a link to the source code of this library https://github.com/jhuckaby/webcamjs.
In this modern era of smartphones, everyone has a cool camera which has also been used by many apps and web applications to capture profile pictures as we are moving towards new technologies, i.e., ASP.NET Core MVC. Let's have a look at how we can achieve this functionally in it.

Agenda

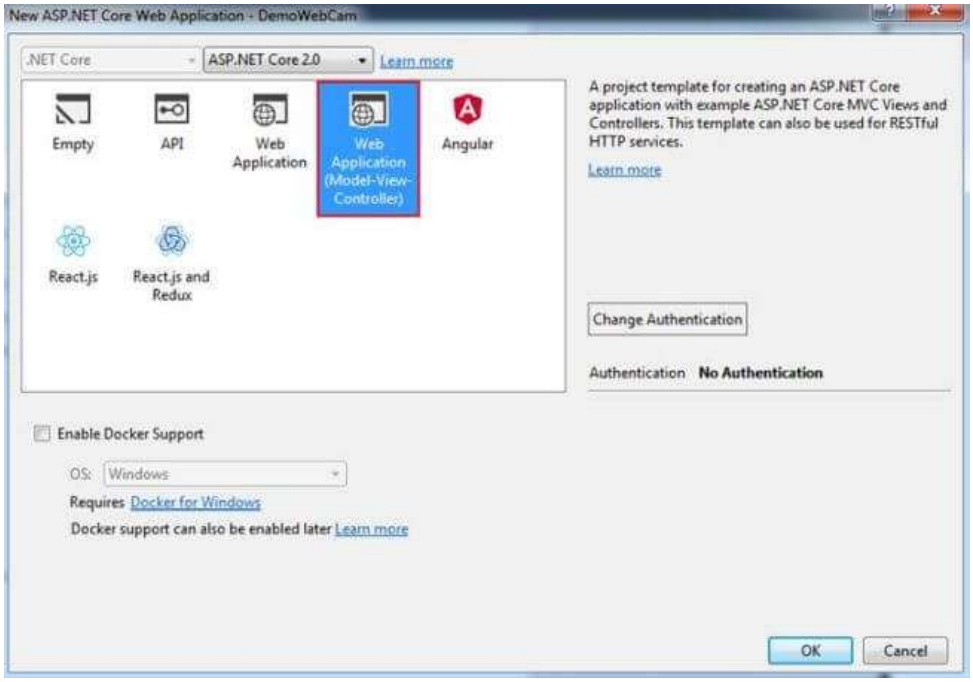
- Create ASP.NET Core MVC application.
- Download and Adding webcam.js related reference files to project.
- Adding Controller (camera controller).
- Adding Capture view.
- Adding Action Method Capture for Handling Ajax Post request
- Capture Image in Debug Mode
- Storing Captured Image in a folder
- Storing Captured Image in Database
- Complete Code Snippet of the camera controller
- Complete Code Snippet of Capture.cshtml
- Finally Get The Output.

Creating ASP.NET Core MVC application

In this part, we are going to create an ASP.NET CORE MVC application and name it as "*DemoWebCam*".



After naming it, just click on the OK button. A new dialog will then pop up for selecting a template. Select the Web Application (Model-View-Controller) template and click the OK button.



After creating the application, it's time to download and add webcam.js and related files to the project.

Download and Adding webcam.js related reference files to project

In this part, we are going to download the webcam.js script from this path *https://github.com/jhuckaby/webcamjs* and add it to our "*wwwroot*" folder such that we can reference it on the View.

Note

In ASP.NET Core, whatever static files you want to access on the browser, those must be kept in the "*wwwroot*" folder.

After adding the files, let's move to add a Controller.

Adding Controller (CameraController)

In this part, we are going to add a controller with the name “Camera” (“CameraController”).

CodeSnippet

```
1. using System;
2. using System.Collections.Generic;
3. using System.Linq;
4. using System.Threading.Tasks;
5. using Microsoft.AspNetCore.Mvc;
6.
7. namespace DemoWebCam.Controllers
8. {
9.     public class CameraController : Controller
10.    {
11.        public IActionResult Capture()
12.        {
13.            return View();
14.        }
15.    }
16. }
```

After adding a Camera Controller, next, we are going to change the name of the Index action method to capture.

Now, let's add a View to this Capture action method.

Adding Capture View

In this part, we are going to add Capture View. To add a View, just right click inside the “Capture” action method, a new dialog will pop up for configuring the View. In that, we are going to just enter the View name as “Capture” and uncheck the layout checkbox.

After adding the View, let's add some controls and scripts to it for capturing it and submitting it to the Controller.

Code Snippet of Capture View

```
1. <div class="col-md-2"></div>
2.     <div class="col-md-4">
3.         <div class="panel panel-default">
4.             <div class="panel-heading">Camera</div>
5.             <div class="panel-body">
6.                 <div id="my_camera"></div>
```

```
            <!-- A button for taking snaps -->
            <form>
                <input type="button" class="btn btn-
success" value="Take Snapshot" onClick="take_snapshot()">
            </form>

        </div>
    </div>
</div>
<div class="col-md-4">
    <div class="panel panel-default">
        <div class="panel-heading">Captured Photo</div>
        <div class="panel-body">
            <div id="results">Your captured image will appear here...
</div>
        </div>
        <br />
        <br />
    </div>
</div>
```

After adding the controls, now, let's add scripts to it for capturing a picture.

This script is for displaying the webcam and has a method to capture images.

```
<!-- Configure a few settings and attach camera -->
<script language="JavaScript">
    Webcam.set({
        width: 320,
        height: 240,
        image_format: 'jpeg',
        jpeg_quality: 90
    });
    Webcam.attach('#my_camera');
</script>
<!-- Code to handle taking the snapshot and displaying it locally --
>
<script language="JavaScript">
    function take_snapshot() {
        // take snapshot and get image data
        Webcam.snap(function (data_uri) {
            // display results in page
            document.getElementById('results').innerHTML =
                '<img src="' +
                data_uri +
                '"/>';
```

```
21.
22.             Webcam.upload(data_uri,
23.                 '/Camera/Capture',
24.                 function (code, text) {
25.                     alert('Photo Captured');
26.                 });
27.
28.         });
29.     }
30. </script>
```

Now, let's run the application to confirm that your capture View is working accurately.

Adding Action Method Capture for Handling Ajax Post request

For capturing an image, we are going to add Capture Action Method which will handle post request.

After that, we are going to read values from Request (HttpContext.Request.Form.Files) next we are going to create a unique name to this image which we have captured and store it in "CameraPhotos" folder which we have created in the "wwwroot" folder.

Code Snippet of Capture Action Method

```
1. [HttpPost]
2. public IActionResult Capture(string name)
3. {
4.     var files = HttpContext.Request.Form.Files;
5.     if (files != null)
6.     {
7.         foreach (var file in files)
8.         {
9.             if (file.Length > 0)
10.             {
11.                 // Getting Filename
12.                 var fileName = file.FileName;
13.                 // Unique filename "Guid"
14.                 var myUniqueFileName = Convert.ToString(Guid.NewGuid());
15.                 // Getting Extension
16.                 var fileExtension = Path.GetExtension(fileName);
17.                 // Concating filename + fileExtension (unique filename)
18.                 var newFileName = string.Concat(myUniqueFileName, fileExtension);
19.                 //  Generating Path to store photo
20.                 var filepath = Path.Combine(_environment.WebRootPath, "CameraPhotos") + $@"\
    {newFileName}";
```

```
21.
22.             if (!string.IsNullOrEmpty(filepath))
23.             {
24.                 // Storing Image in Folder
25.                 StoreInFolder(file, filepath);
26.             }
27.
28.             var imageBytes = System.IO.File.ReadAllBytes(filepath);
29.             if (imageBytes != null)
30.             {
31.                 // Storing Image in Folder
32.                 StoreInDatabase(imageBytes);
33.             }
34.
35.         }
36.     }
37.     return Json(true);
38. }
39. else
40. {
41.     return Json(false);
42. }
43. }
```

Now, let us complete with adding capture Action Method to handle the post Request.

Let's run the application and have a look at it in debug mode.

Capture Image in Debug Mode at Client Side

When we click on the Capture button, it calls "*take_snapshot*" function of JavaScript in which we get the base64 format of a captured image which is shown in detail below.

After capturing, the base64 string is uploaded to Capture Action Method.

The Capture Action Method after receiving the request.

Snapshot after capturing the image successfully.

Storing Captured Image in folder

```
1. /// <summary>
2.  /// Saving captured image into Folder.
3.  /// </summary>
4.  /// <param name="file"></param>
5.  /// <param name="fileName"></param>
```

```
private void StoreInFolder(IFormFile file, string fileName)
{
    using (FileStream fs = System.IO.File.Create(fileName))
    {
        file.CopyTo(fs);
        fs.Flush();
    }
}
```

Debug Mode while storing the image in folder.

Folder view where Images are stored after capturing

After storing the image in the folder, next, we are going to store the image in the database.

Storing Captured Image in Database

In this part, we are going see step by step how to store Base64String in the database.

ImageStore Table

After having a look at table design next we are going to create a Model with the same column name and datatypes in the Models folder.

ImageStore Model

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
using System.Linq;
using System.Threading.Tasks;

namespace DemoWebCam.Models
{
    [Table("ImageStore")]
    public class ImageStore
    {
        [Key]
        public int ImageId { get; set; }
        public string ImageBase64String { get; set; }
        public DateTime? CreateDate { get; set; }
    }
}
```

After adding Model, Next, we are going to Add DatabaseContext to store Image into the database.

Adding DatabaseContext

In this part, we are going to add DatabaseContext class in “EntityStore” folder.

After adding DatabaseContext class we are going to inherit DatabaseContext class from DbContext Class and declare entity (“ImageStore”) in it.

Code Snippet

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using DemoWebCam.Models;
using Microsoft.EntityFrameworkCore;

namespace DemoWebCam.EntityStore
{
    public class DatabaseContext : DbContext
    {
        public DatabaseContext(DbContextOptions<DatabaseContext> options) : base(options)
        {

        }
        public DbSet<ImageStore> ImageStore { get; set; }
    }
}
```

After adding DatabaseContext Class, next we are going to Configure Connection string in appsettings.json.

View of appsettings.json

```
{
  "Logging": {
    "IncludeScopes": false,
    "LogLevel": {
      "Default": "Warning"
    }
  },
  "ConnectionStrings": {
    "DefaultConnection": "Data Source=SAI-
   PC\\SQLEXPRESS; UID=sa; Password=Pass$123;Database=TESTDB;"
  }
}
```

Finally, we are in the last part for storing the image into the database. We just need to configure service in Startup.cs class for injecting connection string.

Startup.cs class

Now we are ready to store images into the database.

Below is code which we are going to use for storing the image into the database.

In this part, we have created a separate method "*StoreInDatabase*" which takes image bytes as input. After taking bytes as input, next for storing in the database we are going to convert these bytes to base64string.

Code Snippet for storing Base64String into database

```
/// <summary>
/// Saving captured image into database.
/// </summary>
/// <param name="imageBytes"></param>
 private void StoreInDatabase(byte[] imageBytes)
{
    try
    {
        if (imageBytes != null)
        {
            string base64String = Convert.ToBase64String(imageBytes, 0, imageBytes.Length);
            string imageUrl = string.Concat("", base64String);
            ImageStore imageStore = new ImageStore()
            {
                CreateDate = DateTime.Now,
                ImageBase64String = imageUrl,
                ImageId = 0
            };
            _context.ImageStore.Add(imageStore);
            _context.SaveChanges();
        }
    }
    catch (Exception)
    {
        throw;
    }}
```

Debug Mode while storing Image in the database.

After Storing Image in Database

After storing the image in database, next, let's see the complete code snippet view of Controller and View.

Complete Code Snippet of CameraController

```
using System;
```

```
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Threading.Tasks;
using DemoWebCam.EntityStore;
using DemoWebCam.Models;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;

namespace DemoWebCam.Controllers
{
    public class CameraController : Controller
    {
        private readonly DatabaseContext _context;
        private readonly IHostingEnvironment _environment;
        public CameraController(IHostingEnvironment hostingEnvironment, DatabaseContext cont
        {
            _environment = hostingEnvironment;
            _context = context;
        }

        [HttpGet]
        public IActionResult Capture()
        {
            return View();
        }


        [HttpPost]
        public IActionResult Capture(string name)
        {
            var files = HttpContext.Request.Form.Files;
            if (files != null)
            {
                foreach (var file in files)
                {
                    if (file.Length > 0)
                    {
                        // Getting Filename
                        var fileName = file.FileName;
                        // Unique filename "Guid"
                        var myUniqueFileName = Convert.ToString(Guid.NewGuid());
                        // Getting Extension
                        var fileExtension = Path.GetExtension(fileName);
```

```
47.                 // Concating filename + fileExtension (unique filename)
48.                 var newFileName = string.Concat(myUniqueFileName, fileExtension);
49.                 //  Generating Path to store photo
50.                 var filepath = Path.Combine(_environment.WebRootPath, "CameraPhotos") + $@
    {newFileName}";
51.
52.                 if (!string.IsNullOrEmpty(filepath))
53.                 {
54.                     // Storing Image in Folder
55.                     StoreInFolder(file, filepath);
56.                 }
57.
58.                 var imageBytes = System.IO.File.ReadAllBytes(filepath);
59.                 if (imageBytes != null)
60.                 {
61.                     // Storing Image in Folder
62.                     StoreInDatabase(imageBytes);
63.                 }
64.
65.             }
66.         }
67.         return Json(true);
68.     }
69.     else
70.     {
71.         return Json(false);
72.     }
73.
74. }
75.
76. /// <summary>
77. /// Saving captured image into Folder.
78. /// </summary>
79. /// <param name="file"></param>
80. /// <param name="fileName"></param>
81. private void StoreInFolder(IFormFile file, string fileName)
82. {
83.     using (FileStream fs = System.IO.File.Create(fileName))
84.     {
85.         file.CopyTo(fs);
86.         fs.Flush();
87.     }
88. }
89.
90. /// <summary>
```

```
        /// Saving captured image into database.
        /// </summary>
        /// <param name="imageBytes"></param>
        private void StoreInDatabase(byte[] imageBytes)
        {
            try
            {
                if (imageBytes != null)
                {
                    string base64String = Convert.ToBase64String(imageBytes, 0, imageBytes.Length);
                    string imageUrl = string.Concat("", base64String);

                    ImageStore imageStore = new ImageStore()
                    {
                        CreateDate = DateTime.Now,
                        ImageBase64String = imageUrl,
                        ImageId = 0
                    };

                    _context.ImageStore.Add(imageStore);
                    _context.SaveChanges();
                }
            }
            catch (Exception)
            {
                throw;
            }
        }
    }
}
```

Complete Code Snippet of Capture.cshtml View

```
@{
    Layout = null;
}

<!DOCTYPE html>
<html lang="en">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-
    8">
    <title>WebcamJS Test Page</title>
    <link href="~/lib/bootstrap/dist/css/bootstrap.css" rel="stylesheet" />
    <style type="text/css">

```

```
13.         body {
14.             font-family: Helvetica, sans-serif;
15.         }
16.
17.         h2, h3 {
18.             margin-top: 0;
19.         }
20.
21.         form {
22.             margin-top: 15px;
23.         }
24.
25.             form > input {
26.                 margin-right: 15px;
27.             }
28.
29.
30.         #buttonhide {
31.             background: transparent;
32.             border: none !important;
33.             font-size: 0;
34.         }
35.     </style>
36.
37. </head>
38. <body class="container">
39.     <br />
40.     <div class="col-md-2"></div>
41.     <div class="col-md-4">
42.         <div class="panel panel-default">
43.             <div class="panel-heading">Camera</div>
44.             <div class="panel-body">
45.                 <div id="my_camera"></div>
46.                 <!-- A button for taking snaps -->
47.                 <form>
48.                     <input type="button" class="btn btn-
    success" value="Take Snapshot" onClick="take_snapshot()">
49.                 </form>
50.
51.             </div>
52.         </div>
53.     </div>
54.     <div class="col-md-4">
55.         <div class="panel panel-default">
56.             <div class="panel-heading">Captured Photo</div>
```

```
57.          <div class="panel-body">
58.              <div id="results">Your captured image will appear here...
    </div>
59.          </div>
60.          <br />
61.          <br />
62.      </div>
63.  </div>
64.
65.  <div class="col-md-2">    </div>
66.  <!-- First, include the Webcam.js JavaScript Library -->
67.  <script src="~/webcamjs/webcam.js"></script>
68.  <!-- Configure a few settings and attach camera -->
69.  <script language="JavaScript">
70.      Webcam.set({
71.          width: 320,
72.          height: 240,
73.          image_format: 'jpeg',
74.          jpeg_quality: 90
75.      });
76.      Webcam.attach('#my_camera');
77.  </script>
78.
79.  <!-- Code to handle taking the snapshot and displaying it locally --
    >
80.  <script language="JavaScript">
81.      function take_snapshot() {
82.          // take snapshot and get image data
83.          Webcam.snap(function (data_uri) {
84.              // display results in page
85.              document.getElementById('results').innerHTML =
86.                  '<img src="' +
87.                  data_uri +
88.                  '"/>';
89.
90.              Webcam.upload(data_uri,
91.                  '/Camera/Capture',
92.                  function (code, text) {
93.                      alert('Photo Captured');
94.                  });
95.          });
96.      }
97.  </script>
98.
99. </body>
```

```
100. </html>
```

After having a complete view of the Controller and View Code snippet, let's run the application to do final testing.

Finally Output

In this part, we have the capture image which is stored in the folder as well as the database successfully.

Displays alert after capturing Image.

Conclusion

In this article we have learned how to capturing image from Web Cam in ASP.Net Core MVC using webcam.js, and how to store the image in a folder and database in a step by step way.