

Forum:[Hack Forums](#)

Thread: [ASP.NET Core \(5\) - Browse - Upload an Image](#)

Ciphered - June 8, 2016, 9:20 am

Hello everyone, I am just starting out in ASP.NET and I am having a lot of fun however....

I am stuck on a really small problem that I can't seem to figure out how to get past.

Basically I am creating a web application, and in this case we are creating an announcement. It has a Title, Message, Date < and those all work fine.

I would like to also upload an image along with these details.

Since the databases that I have are scaffolded(created) from my Model.cs files is there a datatype in c# that can be used for Files, images more specifically.

I've read a couple dozen articles on how to go about this and have been going for about a week or so. I just can't seem to figure this out.

Below is the code that handles these functions...

Model:

Spoiler ([Click to View](#))

[img]http://s33.postimg.org/vv5gjhze7/MODEL.png[/img]

Controller:

(I lied I can use IList<IFormFile> files as a parameter, there was a different bug that is fixed :))

Spoiler ([Click to View](#))

[img]http://s33.postimg.org/9xe1mtqpb/Controller_Metho.png[/img]

View Page:

Spoiler ([Click to View](#))

[img]http://s33.postimg.org/k8qefhien/View_Page.png[/img]

ALL help is appreciated, thanks in advanced! ?

Current Status

Thanks for all the help so far, everyone has contributed valid information. I am still in the process of compiling this knowledge, and understanding how this whole process works.

I like the idea of using my Model to store the Image Contents, I have tried this using a byte array, however, when the information is posted, the byte[] = null.

Why is this so? Am I missing something? Is there a way to send the file from my View.cshtml to a IFormFile Parameter manually without Model Binding?

[img]http://s33.postimg.org/9n2vylrwv/Image_Help0.png[/img]

Dennamanarjag - June 8, 2016, 10:00 am

There is no way to store a bitmap or something directly (assuming you're using entity framework) as far as I'm aware. You can store it as a file on the server as you're doing now, but I'd recommend just storing the image as a byte array in your model and transform it whenever needed. You could also create a separate entity like "AnnouncementImage" with a 1-1 relationship to an announcement.

Also you might want to take a look at this for easier bindings between your model and your view: mikesdotnetting.com/article/264/migrating-from-razor-web-pages-to-asp-net-mvc-5-model-binding-and-forms

Sakuto - June 8, 2016, 12:48 pm

I'm personally often just store the picture in a defined folder with the Id of the element as the name of file. You could do it here. You could then just use an `HttpPostedFileBase` as the second parameter of your Action.

Also, you should go on and move to Razor for your views rendering.

ytlayol - June 8, 2016, 3:51 pm

this is how I did it for 4.6 not sure about core

I know, I know this isn't the prettiest piece of code but hey it works besides it wasn't written to run in production

The-One - June 8, 2016, 4:25 pm

The first issue, the one addressing the incompatibility of the string type with an image, you can either encode the bytes of the image to a base-64 string, this way you can still use the string type or you could go with an `byte[]`. An image is a file and a file consists of a lot of bytes.

For the second issue you should look into [viewbags](#).

Good luck!

Corios - June 8, 2016, 5:36 pm

If your server isn't built to handle files (RAID managed), don't store them on your server. Just get something like AWS, it's pretty easy to implement and free assuming you're not going to use huge amounts of data, which you probably aren't.

An example of using it:

Code [HttpPost]

[ValidateAntiForgeryToken]

```
public async Task<IActionResult> UploadPicture(IFormFile upload, int QuestId, int CategoryId)
{
    try
    {
        string fileToUpload = string.Empty;
        string filename = string.Empty;
        Team team = _context.Teams.Include(x => x.TeamMembers).FirstOrDefault(x =>
x.TeamMembers.Any(y => y.Id == User.GetUserId()));
```

```

        if (!_context.QuestsTeamsCompleted.Any(x => x.QuestId == QuestId && x.Team.Id ==
team.Id))
        {
            if (upload.Length > 0)
            {
                filename =
ContentDispositionHeaderValue.Parse(upload.ContentDisposition).FileName.Trim("");
                await upload.SaveAsAsync(Path.Combine("uploadedImages", filename));
                fileToUpload = Path.Combine("uploadedImages", filename);
            }

            if (fileToUpload != string.Empty)
            {
                using (var client = new AmazonS3Client(AccessKey, SecretAccessKey,
Amazon.RegionEndpoint.USEast1))
                {

                    PutObjectRequest request = new PutObjectRequest()
                    {
                        BucketName = "survivorquest",
                        Key = team.Id + filename,
                        FilePath = fileToUpload,
                        CannedACL = S3CannedACL.PublicReadWrite
                    };
                    PutObjectResponse response = await client.PutObjectAsync(request);

                    TeamCompleted teamCompleted = new TeamCompleted
                    {
                        PictureOrVideoLink = BaseUrl + team.Id + filename,
                        QuestId = QuestId,
                        VideoOrPicture = "p",
                        Team = team
                    };

                    team.Points += _context.Quests.FirstOrDefault(x => x.Id == QuestId).Points;
                    _context.QuestsTeamsCompleted.Add(teamCompleted);

                    await _context.SaveChangesAsync();
                }

                System.IO.File.Delete(fileToUpload);
            }
        }
        return RedirectToAction("Detail", "Categories", new { id = CategoryId });
    }
    catch (Exception e)
    {
        return RedirectToAction("Detail", "Categories", new { id = CategoryId });
    }
}

```

Obviously, mine was for a project I wrote for some friends called "SurvivorQuest", but it's the basic idea of uploading a file to your server, uploading to their server, saving it to your database, then deleting the picture.

You can write Ajax to upload it directly to AWS and then just push the url to your server doing this.... what note that your keys will have to be loaded on the browser and anyone with fiddler could just steal your account, so your choice.