# Upload Files in .Net Core Web Api

In many projects, we need to provide a possibility to our customers to upload different files in the app and to use them as well. So, this would be the topic for this blog post. We are going to upload files to the server (.NET Core Web API part) and then to use those files in our Angular client app.

For this post, we will stick to image files, but the logic is reusable for other file types as well.

We have created the starter project to work with through this blog post and it can be downloaded from (link). We strongly recommend you to download this project because it would be much easier to follow along with this blog post. In this project, we create a new user and show all the created users as well. Through this post, we are going to modify the create logic by adding an upload functionality and our new user will be created together with an image path related to it.

If you want to download our finished project, you can do that from (link).

This complete blog post is going to be divided into the following sections:

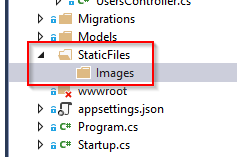
* Controller and Action Logic - .NET Core Part
* Upload File - Angular Part
* Using Uploaded File in Our Application

## Controller and Action Logic - .NET Core Part

After we have downloaded our starter project, we are going to open the UploadFilesServer project.

This project is created on top of the SQL database, so to create that database, we need to run the update-database command in a Package Manager. By doing this, our migrations will be executed and the database and table will be created.

The next step is to create a new folder StaticFiles and inside a new folder Images in the Solution Explorer window:



To continue, let’s create a new API Controller file in the Controllers folder and name it UploadController.

Let’s modify that file by adding a new action that will be responsible for the upload logic:

[HttpPost("upload"), DisableRequestSizeLimit]

public IActionResult Upload()

{

try

{

var file = Request.Form.Files[0];

var folderName = Path.Combine("StaticFiles", "Images");

var pathToSave = Path.Combine(Directory.GetCurrentDirectory(), folderName);

var pathToDb = Path.Combine(folderName, file.FileName);

if (file.Length > 0)

{

var fileName = ContentDispositionHeaderValue.Parse(file.ContentDisposition).FileName.Trim('"');

var fullPath = Path.Combine(pathToSave, fileName);

using (var stream = new FileStream(fullPath, FileMode.Create))

{

file.CopyTo(stream);

}

return Ok(new { pathToDb });

}

else

{

return BadRequest();

}

}

catch (Exception ex)

{

return StatusCode(500, "Internal server error");

}

}

We are using a POST action for the upload-related logic and disabling the request size limit as well. The logic inside this action is pretty straightforward. We extract the file from the request and providing the path where the file will be stored. Moreover, if the file is really there, we just take its name and provide a full path on the server to store file to and create a path to store in the database. This database path is going to be returned as a result of this action.

### Serving Static Files

Usually, all the files in the wwwroot folder are servable for the client applications. We provide that by adding app.UseStaticFiles() in the Startup class in the Configure method. Of course, our uploaded images will be stored in the StaticFiles folder, and due to that, we need to make it servable as well. To do that, let’s modify the Configure method in the Startup.cs class:

public void Configure(IApplicationBuilder app, IHostingEnvironment env)

{

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

}

else

{

app.UseHsts();

}

app.UseHttpsRedirection();

app.UseCors("CorsPolicy");

app.UseStaticFiles();

app.UseStaticFiles(new StaticFileOptions()

{

FileProvider = new PhysicalFileProvider(Path.Combine(Directory.GetCurrentDirectory(), @"StaticFiles")),

RequestPath = new PathString("/StaticFiles")

});

app.UseMvc();

}

And that is it. We have prepared our server side and it is time to jump right to the client side code.