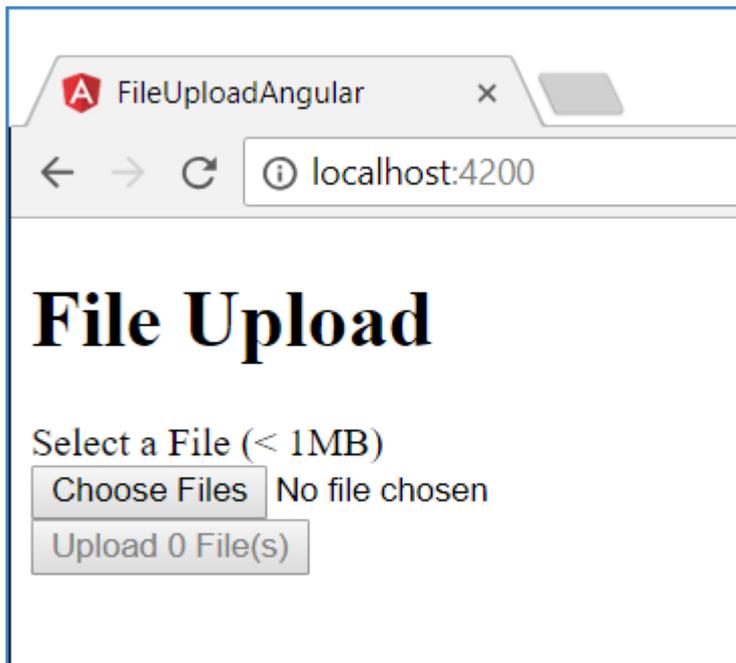


How to Upload Small Files Using Angular

Sometimes you need to upload some files to your server via an Angular application. There are a few different methods you may use to upload. In this blog post, I am going to present a method that works well for small files, up to about one to two megabytes in size. In this blog you build two projects; a .NET Core Web API project, and an Angular project. You build these two projects from scratch using the Angular CLI, .NET Core and Visual Studio Code editor.

The result from this blog post is a page that allows you to select one or more small files using an `<input type="file">` element. You then build a custom `FileToUpload` object with attributes about the file, plus the file contents. Finally, this `FileToUpload` object is sent via a Web API call to a method on the server. Once the server has this file, you may choose to save it as a file on the server, or whatever you choose.



Build .NET Core Web API

Build the .NET Core Web API application you are going to upload the files to first. Open up an instance of Visual Studio Code. From the menu, select **View | Integrated Terminal** to display a terminal window at the bottom of the editor. You should see something that looks like the following:

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

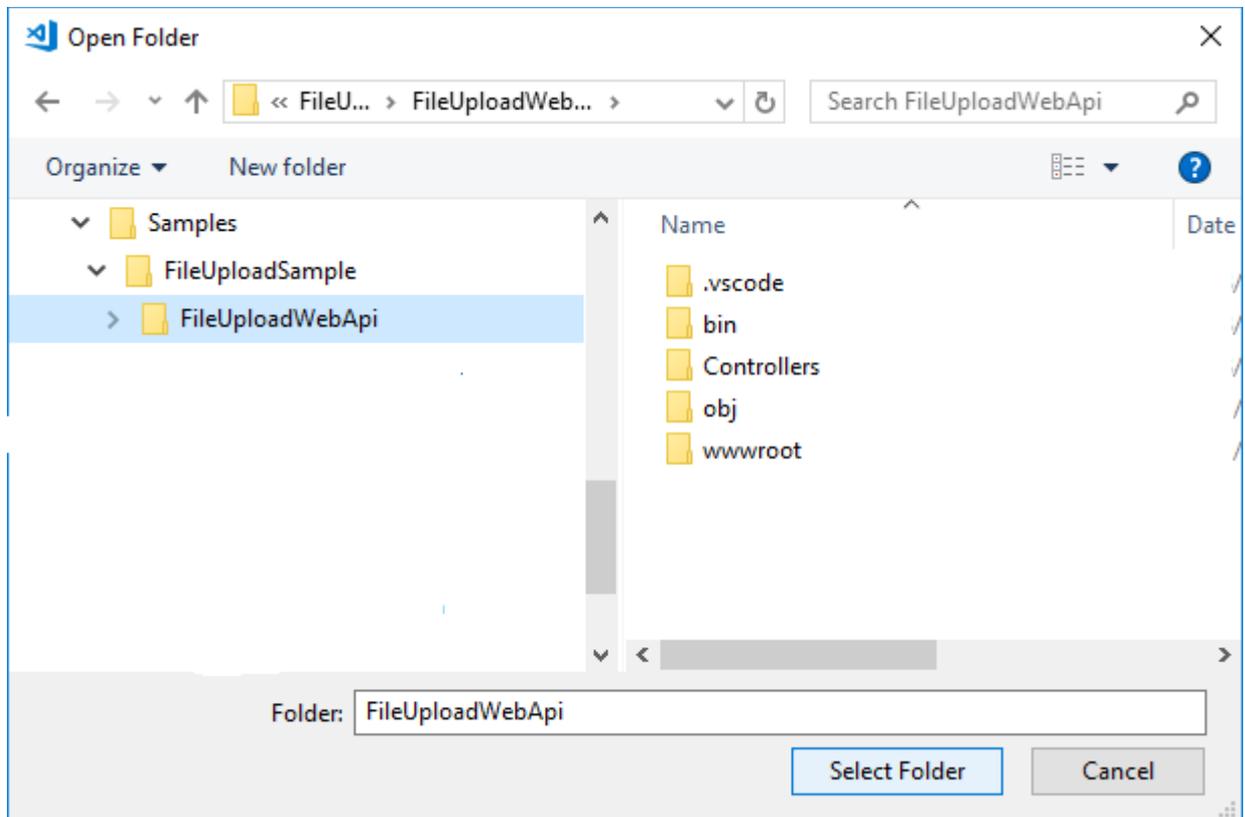
XX C:\Users\YOUR_LOGIN>
```

I am sure you have a directory somewhere on your hard drive in which you create your projects. Navigate to that folder from within the terminal window. For example, I am going to go to my **D** drive and then to the `\Samples` folder on that drive. Enter the following commands to create a .NET Core Web API project.

```
d:
cd Samples
mkdir FileUploadSample
cd FileUploadSample
mkdir FileUploadWebApi
cd FileUploadWebApi
dotnet new webapi
```

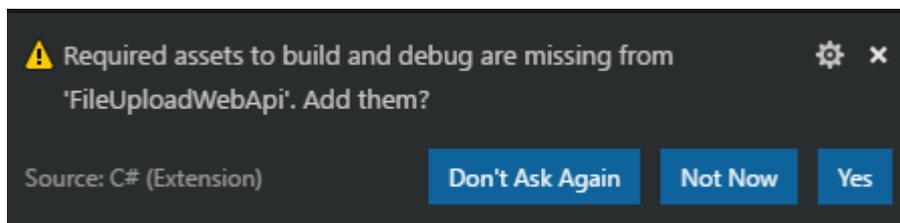
Open the Web API Folder

Now that you have built the Web API project, you need to add it to Visual Studio Code. Select **File | Open Folder...** and open the folder where you just created the **FileUploadWebApi** project. In my case this is from the folder **D:\Samples\FileUploadSample\FileUploadWebApi**.



Load Required Assets

After loading the folder, pause for a few seconds, and you should see a new prompt appear in your Code window saying that some required assets are missing. Click on the **Yes** button to add these assets to this project.



Enable Cors

The Web API project is going to run on the address **localhost:5000** by default. However, when you create a new Angular application, it will be running on **localhost:4200** by default. This means each is running on a separate domain from each other. For your Angular application to call the Web API methods, you must tell the Web API that you are allowing Cross-Origin Resource Sharing (CORS). To use CORS, you need to add a CORS package to your project. Go back to the integrated terminal and type the following command.

```
dotnet add package Microsoft.AspNetCore.Cors
```

Open the **Startup.cs** file and modify the `ConfigureServices()` method. Add the following above `services.AddMvc()` line.

```
services.AddCors();  
  
services.AddMvc();
```

Next, modify the `Configure()` method to look like the following:

```
public void Configure(IApplicationBuilder app,  
                    IHostingEnvironment env)  
{  
    if (env.IsDevelopment())  
    {  
        app.UseDeveloperExceptionPage();  
    }  
  
    app.UseCors(  
        options => options.WithOrigins("http://localhost:4200")  
        .AllowAnyMethod().AllowAnyHeader()  
    );  
  
    app.UseMvc();  
}
```

The code above allows you to specify a single origin from which this Web API project accepts requests. Also, that you are allowing any method and any header to be accepted from this origin.

NOTE: You must add the call to the `UseCors()` method prior to the `UseMvc()` method call.

Try it Out

It is a good idea to test out your Web API project and ensure it can accept requests. Press F5 to build the .NET Core Web API project and launch a browser. The browser will come up with a blank page. Type the following into the browser address bar.

```
http://localhost:5000/api/values
```

After hitting enter you should see a string that looks like this.

```
["value1", "value2"]
```

FileToUpload Class

A file that you are going to upload will have many attributes that you might wish to convey from the front-end to the web service. These attributes are things like the file name, the file type, the size of the file, etc. Create a C# class to accept each of these values. Create a **\Models** folder in the root of your Web API project. Add a new file named **FileToUpload.cs**. Add the following code to this new file.

```
using System;

public class FileToUpload
{
    public string FileName { get; set; }
    public string FileSize { get; set; }
    public string FileType { get; set; }
    public long LastModifiedTime { get; set; }
    public DateTime LastModifiedDate { get; set; }
    public string FileAsText { get; set; }
    public string FileAsBase64 { get; set; }
    public byte[] FileAsByteArray { get; set; }
}
```

File Upload Controller

Next you need a controller class to allow your Angular application to call a method to send the file to. Delete the **ValuesController.cs** file from the **\Controllers** folder. Now add back into the **\Controllers** folder a file named **FileUploadController.cs**. Add the following code into this file.

```
using System;
using System.IO;
using Microsoft.AspNetCore.Mvc;

namespace fileUploadSampleWebApi.Controllers
{
    [Route("api/[controller]")]
    public class FileUploadController : Controller
    {
        const string FILE_PATH = @"D:\Samples\";

        [HttpPost]
        public IActionResult Post([FromBody]FileToUpload theFile)
        {
            // Create unique file name
            var filePath = FILE_PATH +
                DateTime.Now.ToString().Replace("/", "")
                .Replace(":", "").Replace(" ", "") + "-" +
                theFile.FileName;

            // Remove file type from base64 encoding, if any
            if (theFile.FileAsBase64.Contains(","))
            {
                theFile.FileAsBase64 = theFile.FileAsBase64
                    .Substring(theFile.FileAsBase64.IndexOf(",") + 1);
            }

            // Convert base64 encoded string to binary
            theFile.FileAsByteArray =
                Convert.FromBase64String(theFile.FileAsBase64);

            // Write binary file to server path
            using (var fs = new FileStream(filePath, FileMode.CreateNew))
            {
                fs.Write(theFile.FileAsByteArray, 0,
                    theFile.FileAsByteArray.Length);
                fs.Close();
                fs.Dispose();
            }

            return Ok();
        }
    }
}
```

Be sure to change the FILE_PATH constant from "D:\Samples" to a valid path on your machine that the web project has permissions to write to.

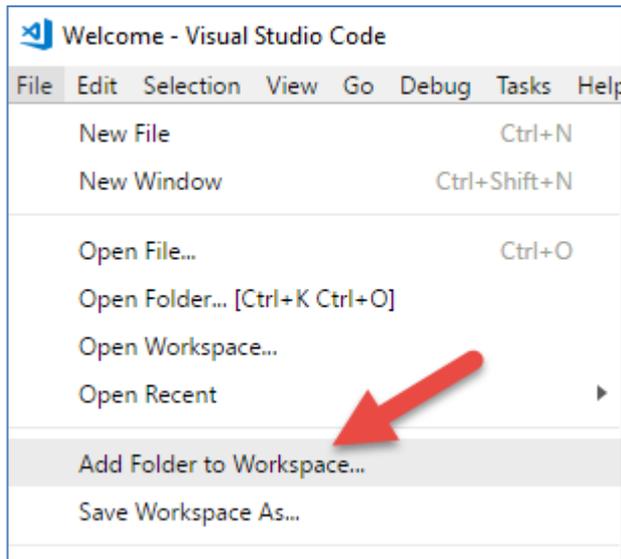
Build Angular Upload Project

It is now time to build the Angular project. From within VS Code open the integrated terminal window. Navigate back to the folder where you created this project. So, for my project I want to go to the **D:\Samples\FileUploadSample** folder. Enter the following command to create a new Angular application.

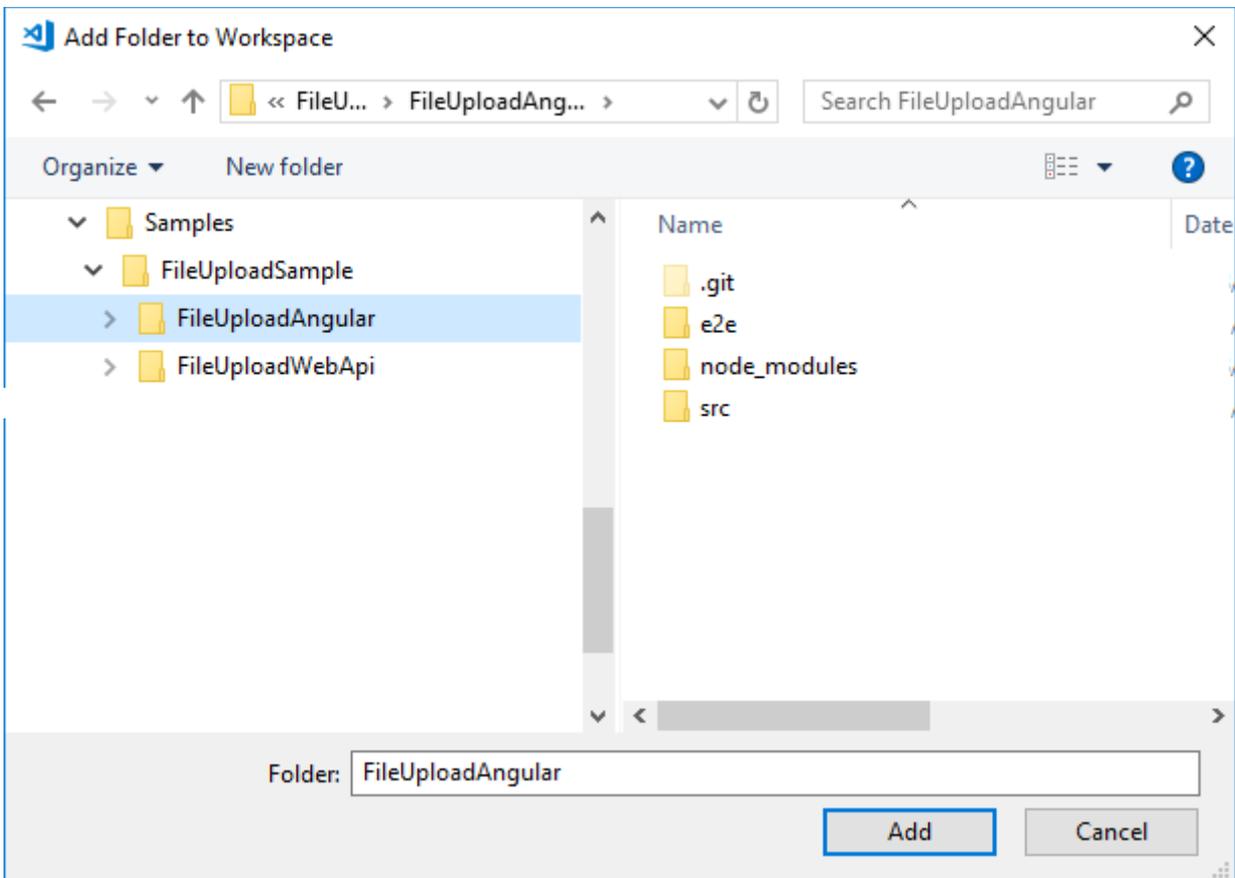
```
ng new FileUploadAngular
```

Add Web API Project to Workspace

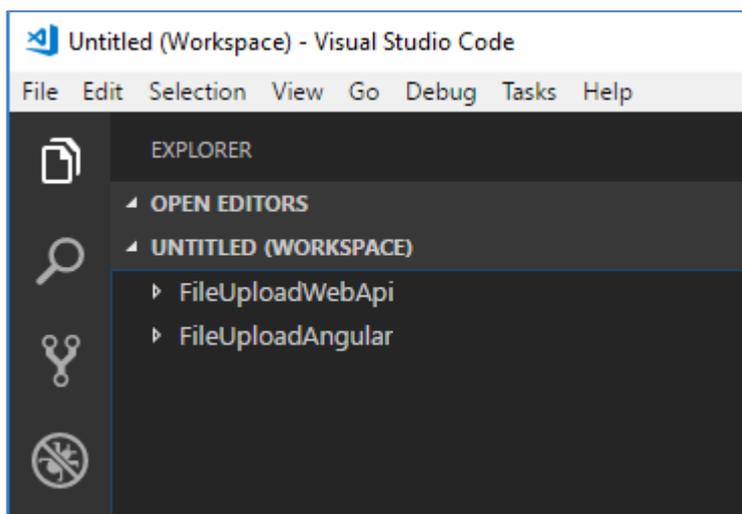
Once the project is created, add the newly created folder named FileUploadAngular to Visual Studio Code. Select the **File | Add Folder to Workspace...** menu item as shown below.



Choose the **FileUploadAngular** folder as shown in the following screen shot.



You should now see two projects within VS Code as shown in the following screen shot. This is called a Workspace.



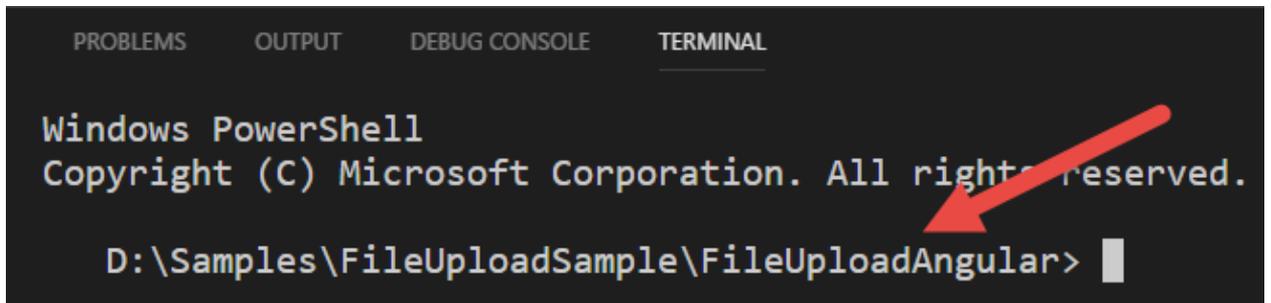
Save the Workspace

Click **File | Save Workspace As...** and give it the name **FileUploadSampleApp**. Click the Save button to store this new workspace file on disk. From now on, you

may always open this application by double-clicking on the **FileUploadSampleApp.code-workspace** file.

Create FileToUpload Class

Just like you created a FileToUpload class in C# to hold the various attributes about a file, you want to do the same thing in Angular. Go back into the terminal window and navigate to the **FileUploadAngular** folder.



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

D:\Samples\FileUploadSample\FileUploadAngular>
```

Build a new class that represents a file you wish to upload.

```
ng g class file-upload/fileToUpload
```

Open the generated **file-to-upload.ts** file and add the following code.

```
export class FileToUpload {
  fileName: string = "";
  fileSize: number = 0;
  fileType: string = "";
  lastModifiedTime: number = 0;
  lastModifiedDate: Date = null;
  fileAsBase64: string = "";
  fileAsText: string = "";
}
```

Create File Upload Service

As with any Angular application, you should always separate any logic that communicates with a Web API into an Angular Service class. You can create a new service using the Angular CLI. In the Integrated Terminal window, enter the following command to create a FileUploadService class

```
ng g s file-upload/fileUpload -m app.module
```

Open the newly generated **file-upload.service.ts** file and add the following import statements.

```
import { HttpClient, HttpHeaders } from '@angular/common/http';
import { Observable } from 'rxjs/Observable';
import { FileToUpload } from '../file-to-upload';
```

Add two constants just below the import statement. The first constant is the path to the FileUpload controller you create previously. The second constant is a header used to post JSON data to the Web API.

```
const API_URL = "http://localhost:5000/api/FileUpload/";
const httpOptions = {
  headers: new HttpHeaders({
    'Content-Type': 'application/json'
  })
};
```

The HttpClient class needs to be injected into this service class in order to post the file information to the Web API method. Modify the constructor of the FileUploadService class to look like the following.

```
constructor(private http: HttpClient) { }
```

Add a method named uploadFile() to which you pass an instance of the FileToUpload class. This class is built using properties from a File object retrieved from the user via an <input type="file"> element. Call the post() method on the HttpClient class passing in the URL where the controller is located, the file object to post and the http header constant.

```
uploadFile(theFile: FileToUpload) : Observable<any> {
  return this.http.post<FileToUpload>(
    API_URL, theFile, httpOptions);
}
```

File Upload Component

It is now time to build the HTML page that prompts the user for a file from their local drive they wish to upload to the web server. Create an Angular component using the Angular CLI. This builds a .css, .html and a .ts file from which you build the file upload page. In the terminal window, enter the following command.

```
ng g c fileUpload
```

Open the **file-upload.component.ts** file and add two import statements for the FileUploadService and the FileToUpload class.

```
import { FileUploadService } from './file-upload.service';
import { FileToUpload } from './file-to-upload';
```

You should limit the maximum size of file a user may attempt to upload. The files are going to be base64 encoded prior to sending. When you base64 encode something, it does grow larger than the original size. So, it is best to limit the size of the file to upload. Add a constant just below the import statements and set a limit of one megabyte. Feel free to play with different file sizes to see what works with your situation.

```
// Maximum file size allowed to be uploaded = 1MB
const MAX_SIZE: number = 1048576;
```

You need to add two public properties to `FileUploadComponent` class. The first property, *theFile*, holds an instance of a file object returned from the file upload object. There is no equivalent of the file object from HTML in Angular, so you must use the *any* data type. The second property, *messages*, is used to display a set of messages to the user when something happens.

```
theFile: any = null;
messages: string[] = [];
```

The `FileUploadComponent` class creates an instance of a `FileToUpload` class from the information contained in the *theFile* property. It then passes this object to the `FileUploadService` to upload the file to the server. Inject the `FileUploadService` into the constructor of the `FileUploadComponent`.

```
constructor(private uploadService: FileUploadService) { }
```

Get File Information from Change Event

When the user selects a file from their file system, the change event is fired on the file input type. You are going to respond to this change event and call a method named `onFileChange()` in your component. Write this code as shown below.

```
onFileChange(event) {
  this.theFile = null;

  // See if any file(s) have been selected from input
  if (event.target.files && event.target.files.length > 0) {
    // Don't allow file sizes over 1MB
    if (event.target.files[0].size < MAX_SIZE) {
      // Set theFile property
      this.theFile = event.target.files[0];
    }
    else {
      // Display error message
      this.messages.push("File: " + event.target.files[0].name
        + " is too large to upload.");
    }
  }
}
```

In the `onFileChange()` method an argument, named *event*, is passed in by the file input type. You check the `target.files` property of this argument to see if the user selected a file. If a file is selected, check the `size` property to make sure it is less than the size you placed into the `MAX_SIZE` constant. If the file meets the size requirement, you retrieve the first element in the `files` array and assign it to the *theFile* property. If the file exceeds the size requirement, push a message onto the `messages` array to inform the user of the file name that is in error.

Read and Upload File

Add a private method named `readAndUploadFile()` to the `FileUploadComponent` class as shown below.

```
private readAndUploadFile(theFile: any) {
  let file = new FileToUpload();

  // Set File Information
  file.fileName = theFile.name;
  file.fileSize = theFile.size;
  file.fileType = theFile.type;
  file.lastModifiedTime = theFile.lastModified;
  file.lastModifiedDate = theFile.lastModifiedDate;

  // Use FileReader() object to get file to upload
  // NOTE: FileReader only works with newer browsers
  let reader = new FileReader();

  // Setup onload event for reader
  reader.onload = () => {
    // Store base64 encoded representation of file
    file.fileAsBase64 = reader.result;

    // POST to server
    this.uploadService.uploadFile(file)
      .subscribe(resp => { this.messages.push("Upload complete");
    });
  }
  // Read the file
  reader.readAsDataURL(theFile);
}
```

You are going to pass the *theFile* property to this method. I know you are thinking this is not necessary, however, later when I show you how to handle multiple file uploads, you will see why this is a good practice. Create a new instance of a *FileToUpload* class, and set the properties from the file object retrieved from the file input type. Next, you are going to use the *FileReader* object from HTML 5. Note, this object only works with more modern browsers. Create a new instance of a *FileReader*, and setup an *onload()* event which is called after the file has been loaded by the *readAsDataURL()* method. The *readAsDataURL()* reads the contents and returns the contents as a base64 encoded string. Within the *onload()* event you get the file contents in the *result* property of the reader. Place the contents into the *fileAsBase64* property of the *FileToUpload* object. Call the *uploadFile()* method on the *FileUploadService* class, passing in this *FileToUpload* object. Upon successfully uploading the file, push the message "Upload Complete" onto the messages array to have it displayed to the user.

The *readAndUploadFile()* method is called in respond to the Upload File button's click event.

```
uploadFile(): void {
  this.readAndUploadFile(this.theFile);
}
```

The File Upload HTML

Open the **file-upload.component.html** file and delete all HTML within that file. Add the following HTML to this file.

```
<h1>
  File Upload
</h1>

<label>Select a File (&lt; 1MB)</label>
<br/>
<input type="file" (change)="onFileChange($event)" />
<br/>
<button (click)="uploadFile()" [disabled]="!theFile">
  Upload File
</button>
<br/>
<br/>

<!-- ** BEGIN: INFORMATION MESSAGE AREA ** -->
<div *ngIf="messages.length > 0">
  <span *ngFor="let msg of messages">
    {{msg}}
    <br />
  </span>
</div>
<!-- ** END: INFORMATION MESSAGE AREA ** -->
```

In the `<input type="file" ...>` you see the call to the `onFileChange()` event you wrote. The Upload File button is disabled until the *theFile* property is not null. When it is clicked upon, the `uploadFile()` method is called to start the upload process. At the end of this file is the location where you write all messages contained within the `messages` array.

Modify App Module

Open the **app.module.ts** file and add to the `imports` property a reference to the `HttpClientModule`. After typing the comma, and **HttpClientModule**, hit the Tab key, or use the light bulb in Code, to add the appropriate import statement to this file.

```
imports: [
  BrowserModule,
  HttpClientModule
],
```

Modify App Component HTML

Open the **app.component.html** file and delete all the code within this file. Add the following code to display your file upload component.

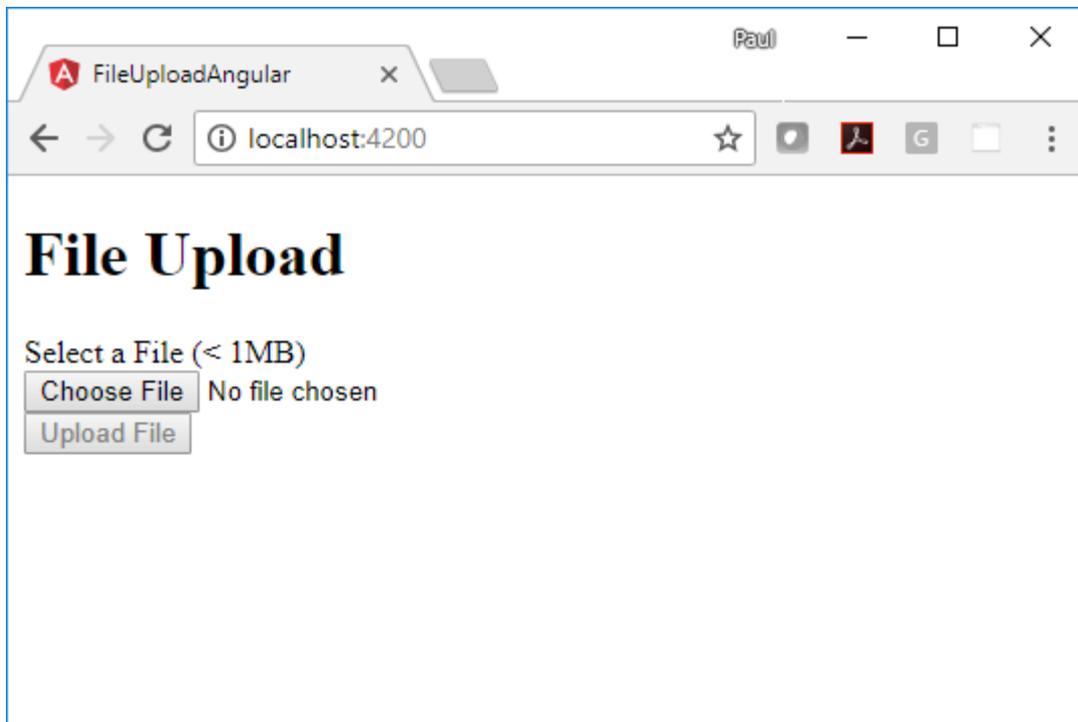
```
<app-file-upload></app-file-upload>
```

Try it Out

If your Web API project is not still running, go ahead and start it now by pressing F5. Go to the Integrated Terminal window and start your Angular application using the following command.

```
npm start
```

Open your browser and enter **localhost:4200** into the address bar. You should see a web page the looks like the following:



Click on the Choose File button and select a file that is less than 1 megabyte in size. After you select a file, the name of that file appears to the right of this button, and the Upload File button becomes enabled. Click the Upload File button and after just a second or two, an "Upload Complete" message should be displayed. Check the folder where you specified to write the files and you should see a file name that starts with today's date, followed by the file name you selected.

Upload Multiple Files

If you wish to have the user select and upload multiple files, this is also possible. Again, each file must be under 1 megabyte in size, but, you may select as many files as you wish. To accomplish this, make the following changes in your code. Open the **file-upload.component.html** file and modify the HTML shown in bold below.

```
<label>Select a File(s) (&lt; 1MB)</label>
<br/>
<input type="file" (change)="onFileChange($event)"
      multiple="multiple" />
<br/>
<button (click)="uploadFile()" [disabled]="!theFiles.length">
  Upload {{theFiles.length}} File(s)
</button>
```

Open the **file-upload.component.ts** file and locate the following line of code.

```
theFile: any = null;
```

Modify this from a single object to an array of file objects.

```
theFiles: any[] = [];
```

Modify the `onFileChange()` method so it looks like the code below.

```
onFileChange(event) {
  this.theFiles = [];

  // See if any file(s) have been selected from input
  if (event.target.files && event.target.files.length > 0) {
    for (let index = 0; index < event.target.files.length; index++)
    {
      let file = event.target.files[index];
      // Don't allow file sizes over 1MB
      if (file.size < MAX_SIZE) {
        // Add file to list of files
        this.theFiles.push(file);
      }
      else {
        this.messages.push("File: " + file.name
          + " is too large to upload.");
      }
    }
  }
}
```

Finally, modify the `uploadFile()` method to loop through the list of file objects selected. Each time through the loop, pass the current instance of the file object retrieved from the file input type to the `readAndUploadFile()` method. You now understand why it was important for you to pass a file object to the `readAndUploadFile()` method.

```
uploadFile(): void {
  for (let index = 0; index < this.theFiles.length; index++) {
    this.readAndUploadFile(this.theFiles[index]);
  }
}
```

Try it Out

Save all your changes. Go back to the browser and you may now select multiple files. Select a few files, click the Upload Files button and ensure that all files are uploaded to the appropriate folder.

Summary

In this blog post you learned to upload small files from your client web application to a server using Angular and a .NET Core Web API project. On the server-side you need to make sure you enable CORS to pass data from one domain to another. On the client-side you are using the `FileReader` class to read data from the user's file system. This object is only available in more modern browsers. This technique should only be used for small files. Don't use this technique for large files as it would take too long to read the data from disk, and too long to send to the server. During this time, you can't provide feedback to the user and you would not be able to display a percentage uploaded. There are several good open-source libraries for uploading large files to the server that do provide feedback as the upload process is happening.

Getting the Sample Code

You may download the sample code by navigating to www.pdsa.com/downloads. Select "PDSA/Fairway Blog" from the Category drop-down. Then select "Upload Small Files using Angular" from the Item drop-down.