Lab 5 – Static Content, Migrations, and File Uploads

# Description

This lab allows you to start implementing other features of MVC: Static Content, Migrations, and File Uploads

# Estimated Time

This lab will take an estimated 3 hours to complete

# Notes

* Be sure Visual Studio is up to date.
* When using the <input> tag be sure to set the type to collect the proper values
* Demo can be found at: <http://afrasialab5.azurewebsites.net/>
  + Note, create, and delete buttons are intentionally disabled. Your submission should be functional though.
* Sample code can be found at: <https://github.com/aarad-ac/AzureStorageAndMigrations>

# Create a new MVC Core project called ‘Lab5’

1. Open Visual Studio 2019
2. Create new empty ASP.NET Core Web Application called Lab5.
   1. Refer to Lab 4 if you forgot how to do this
3. Install the following NuGet packages (refer to lab 4 if you forgot how to do this):
   1. “Microsoft.EntityFrameworkCore" v: 5.0.6
   2. "Microsoft.EntityFrameworkCore.Sqlite” v: 5.0.6
   3. "Microsoft.EntityFrameworkCore.Tools" v: 5.0.6
   4. “Microsoft.VisualStudio.Web.CodeGeneration.Design” v: 5.0.2
   5. “Microsoft.EntityFrameworkCore.SqlServer” v: 5.0.6
   6. “Azure.Storage.Blobs” v: 12.9.1

# Configure your new Web Application

1. See the Azure SQL document in Brightspace -> extra materials to set up your db and find your database connection string
2. See the Azure Storage document in Brightspace -> extra materials to set up your blob storage and find your database connection string
3. Modify both appsettings.Development.json and appsettings.json and add the following lines right before “Logging”, replacing Red text with your appropriate connection strings. Do not remove the quotes. Replace password placeholders with the right password where needed.

"ConnectionStrings": {

"DefaultDBConnection": "AZURE SQL CONNECTION STRING GOES HERE",

"AzureBlobStorage": "AZURE STORAGE CONNECTION STRING GOES HERE"

},

1. Modify ‘Startup.cs’. Add the following constructor to the class:

public IConfiguration Configuration { get; }

public Startup(IConfiguration configuration)

{

Configuration = configuration;

}

1. Modify ‘Startup.cs’. Add the following lines to the method ‘ConfigureServices(IServiceCollection services)’”

var connection = Configuration.GetConnectionString("DefaultDBConnection");

services.AddDbContext<AnswerImageDataContext>(options => options.UseSqlServer(connection));

var blobConnection = Configuration.GetConnectionString("AzureBlobStorage");

services.AddSingleton(new BlobServiceClient(blobConnection));

services.AddControllersWithViews();

Note: you are getting errors with ‘AnswerImageDataContext’ at this point. Ignore it until you finish creating the models.

1. Modify ‘Startup.cs’. Replace the contents of ‘Configure(IApplicationBuilder app, IWebHostEnvironment env)’ with:

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

}

else

{

app.UseExceptionHandler("/Home/Error");

}

app.UseStaticFiles();

app.UseRouting();

app.UseEndpoints(endpoints =>

{

endpoints.MapControllerRoute(

name: "default",

pattern: "{controller=Home}/{action=Index}/{id?}");

});

# Create the ‘Controllers’, ‘wwwroot’, ‘Views’ and ‘Models’ folders

1. At the root create wwwroot folder and add a css file there. You can use the one I posted with the sample code
2. Create a folder in your project called ‘Views’
3. At the root of this new ‘Views’ folder, create a file called ‘\_ViewImports.cshtml’
4. Add the following lines of code to the file ‘\_ViewImports.cshtml’

@using Lab5

@addTagHelper "\*, Microsoft.AspNetCore.Mvc.TagHelpers"

1. Under views, create your layout in the folder called shared
2. At the root of this new ‘Views’ folder, create a file called ‘\_ViewStart.cshtml’
3. Add the following lines of code to the file ‘\_ViewStart.cshtml’

@{

Layout = "\_Layout";

}

1. At the root of the ‘Views’ folder created a folder called ‘Home’
2. At the root of the ‘Views’ folder created a folder called ‘AnswerImages’
3. At the root of the project create a folder called ‘Models’
4. Create a view called Error.cshtml under Views/Shared. It should contain some error message like this example:

@{

ViewData["Title"] = "Error";

}

<h1 class="text-danger">Error.</h1>

<h2 class="text-danger">An error occurred while processing your request.</h2>

1. Create a folder in your project called ‘Controllers’
2. Under ‘Controllers’, create a new Controller called ‘Home’
3. Under ‘Controllers’, create a new Controller called ‘AnswerImages’

# Create the Model

1. At the root of the ‘Models’ folder, create a file called ‘AnswerImage.cs’
2. Add the following Properties to the ‘AnswerImage.cs’ file
   1. int AnswerImageId
   2. string FileName
   3. string Url
   4. Question Question
      1. Note that the type is a Question. Question should be an enum that has two values “Earth” and “Computer”. If you need a sample code refer to EntityFramework sample and check how we define driver’s license.
   5. Add proper attributes to the entity properties you created above. Based on your knowledge of ASP.NET so far, you should be able to identify minimum requirements by now. The attributes should help with prober validation of model. You are on your own for this.
3. At the root of the ‘Data’ folder create a file called ‘AnswerImageDataContext.cs’
   1. Use the example code located in Week 9 slides or lab 4 for guidance.
   2. Make sure in your context you include the Constructor and the DBSet to hold your AnswerImage objects.
4. Add a migration for initial creation of database
   1. In the ‘Package Manager Console’ type: Add-Migration InitialCreate
   2. Hit ‘Enter’
5. Apply the migration to DB
   1. In the ‘Package Manager Console’ type: Update-Database
   2. Hit ‘Enter’
6. Validate your database is created, and the AnswerImage table has the desired columns
   1. You can do this on Azure or from Visual Studio
   2. If DB is not created, start to troubleshoot

# Create the Controller’s Actions and Views

1. In your HomeController create the ‘Index’ action
   1. This should navigate to Index view where you introduce yourself
2. Add the following lines to AnswerImagesController, to the beginning of the controller class.

private readonly AnswerImageDataContext \_context;

private readonly BlobServiceClient \_blobServiceClient;

private readonly string earthContainerName = "earthimages";

private readonly string computerContainerName = "computerimages";

public AnswerImagesController(AnswerImageDataContext context, BlobServiceClient blobServiceClient)

{

\_context = context;

\_blobServiceClient = blobServiceClient;

}

1. In your AnswerImagesController create the ‘Index’ action
   1. Index should show all the files that are already uploaded to Azure Storage
   2. Checkout the sample I posted. This is a simple query you already practiced a few times. You are on your own.
   3. Add the corresponding view for it
2. In your AnswerImagesController create the ‘Upload’ (both GET and POST) actions
   1. Create a GET action:

public IActionResult Upload()

{

return View();

}

* 1. Create a POST action called Upload with the following signature
     1. public async Task<IActionResult> Upload(IFormFile answerImage)
  2. You need to handle the upload of the images here. You are on your own. Make sure to refer to my sample code whenever needed
  3. Create the corresponding view to AnswerImagesController, upload action
     1. Note that you will need to give the user an option to choose which question they are answering. The image should be uploaded to the container corresponding to the user’s choice (I can easily check that in your index page from the image URL, so make sure to do it right).

1. In your AnswerImagesController create the ‘Delete’ (both GET and POST) actions
   1. Refer to sample code, understand what it is doing, and copy the code to your project. Adjust variable names where needed.
   2. Note that, you are already recording what question the users are answering, so you need to delete the image from the containers based on the that field in the db

# Publish

1. During the publish process, you need to configure your Azure SQL
2. On the publish tab, and under ‘Service Dependencies’ section, find the ‘SQL Server Database’, and click ‘Configure’ in front of it.
3. On the next page, select ‘Azure SQL DataBase’
4. Click next on the following page, too
5. On ‘Save Connection String in’, select ‘None’
6. On the ‘Configure Azure SQL Database’, enter your Azure SQL username and password, and click next
7. Uncheck all check marks, and click finish
8. You are back to Publish tab
9. Make sure ‘Storage’ appears under ‘Service Dependencies’, you do not need to configure it
10. Click publish button
11. Note that if after publishing your app cannot reach the database, follow these steps:
    1. Go to Azure Portal
    2. Find your Azure SQL instance
    3. On the Overview tab, click on ‘Set Server Firewall’
    4. On the Firewall Settings page, change ‘Allow Azure services and resources to access this server’ from ‘No’ to ‘Yes’
    5. Click ‘save’

# Push to Git

1. Push your code to git
2. Note: No PASSWORDs should be pushed to Git. You can add your appsettings to git ignore, so they never get pushed to git.
   1. Just create a file called’.gitignore’
   2. And then add the following lines to it:

appsettings.json

appsettings.Development.json

# Deliverable

1. Answer the following questions by pictures:
   1. What will the computers look like in 20 years?
   2. What will the earth look like in 20 years?
2. Upload your answers to your website. Now, there should be two photos in your AnswerImages/Index view. Delete any extra photos you might have there.
3. Submit to brightspace:
   1. Link to your Git repo.
   2. Link to your published website

# Marking Scheme:

|  |  |
| --- | --- |
| program | 1 |
| Startup | 1 |
| viewimports | 1 |
| viewstart | 1 |
| layout | 1 |
| home/index | 1 |
| Error Page | 1 |
| structure | 1 |
| model | 6 |
| context | 2 |
| migration | 1 |
| HomeController | 1 |
| AnswerImagesController | 7 |
| AnswerImage/Index | 2 |
| Upload | 4 |
| Delete | 3 |
| images uploaded | 1 |
| gitignore | 2 |
| working assignment | 3 |