Lab 6 – Web API

# Description

This lab allows you to start implementing other features of MVC: Web API

# Estimated Time

This lab will take an estimated 2 hours to complete

# Notes

* Application Demo can be found at: <http://afrasialab6.azurewebsites.net/>
* I intentionally blocked all actions other than GET, but your service should have a working version for all the verbs.
* See “Brightspace -> Course Content -> Extra Materials -> Azure Usage” for information about deploying Azure Web Apps, Databases and Storage Accounts.
* Check out the sample code for more information: <https://github.com/aarad-ac/WebApiCore>

# Create a new MVC Core project called ‘Lab6’

1. Create new ASP.NET Core Web Application called Lab6.
   1. Select the ‘Empty’ ASP.NET Core Template
   2. Uncheck the HTTPS box
   3. Refer to previous labs if you forgot how to do this
2. 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. “Swashbuckle.AspNetCore” >= v:6.1.4
3. Double click on Lab6 project (you can also navigate to the directory containing your project and open Lab6.csproj in a text editor)
   1. Add the following lines to it:

<PropertyGroup>

<GenerateDocumentationFile>true</GenerateDocumentationFile>

<NoWarn>$(NoWarn);1591</NoWarn>

</PropertyGroup>

# Configure your new Web Application

1. At the root of the project create a folder called ‘Data’
2. Under Data folder, create a class ‘DbInitializer.cs’:
   1. Replace the class with:

public static class DbInitializer

{

public static void Initialize(StudentDbContext context)

{

context.Database.Migrate();

if (context.Students.Any())

{

return; // DB has been seeded

}

var students = new Student[]

{

new Student{FirstName="Carson",LastName="Alexander",Program="ICT"},

new Student{FirstName="Meredith",LastName="Alonso",Program="ICT"},

new Student{FirstName="Arturo",LastName="Anand",Program="ICT"},

new Student{FirstName="Gytis",LastName="Barzdukas",Program="ICT"},

};

foreach (Student c in students)

{

context.Students.Add(c);

}

context.SaveChanges();

}

}

Note: you are getting errors with ‘StudentDbContext’ and ‘Student’ at this point. Ignore it until you finish creating the model.

1. Replace body of Main in ‘Program.cs’ with:

var host = CreateHostBuilder(args).Build();

using (var scope = host.Services.CreateScope())

{

var services = scope.ServiceProvider;

try

{

var context = services.GetRequiredService<StudentDbContext >();

DbInitializer.Initialize(context);

}

catch (Exception ex)

{

var logger = services.GetRequiredService<ILogger<Program>>();

logger.LogError(ex, "An error occurred while seeding the database.");

}

}

host.Run();

1. See the Azure SQL document in Brightspace -> extra materials to set up your db and find your database connection string
2. 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.

"ConnectionStrings": {

"DefaultConnection": "AZURE SQL 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("DefaultConnection");

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

services.AddControllers();

services.AddSwaggerGen(c =>

{

c.SwaggerDoc("v1", new OpenApiInfo

{

Version = "v1",

Title = "Students API",

Description = "A simple example ASP.NET Core Web API",

});

var xmlFile = $"{Assembly.GetExecutingAssembly().GetName().Name}.xml";

var xmlPath = Path.Combine(AppContext.BaseDirectory, xmlFile);

c.IncludeXmlComments(xmlPath);

});

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

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

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

}

app.UseSwagger();

app.UseSwaggerUI(c =>

{

c.SwaggerEndpoint("/swagger/v1/swagger.json", "My Students API V1");

c.RoutePrefix = string.Empty;

});

app.UseRouting();

app.UseEndpoints(endpoints =>

{

endpoints.MapControllers();

});

# Create the ‘Controllers’ and ‘Models’ folders

1. Create a folder in your project called ‘Controllers’
2. At the root of the project create a folder called ‘Models’

# Create the Model

1. At the root of the ‘Models’ folder create a file called ‘StudentBase.cs’
   1. This should define a model that has the following fields. Use appropriate attributes for the fields. You should already know how to do this. You are on your own.
      1. FirstName
      2. LastName
      3. Program
2. At the root of the ‘Models’ folder create a file called ‘Student.cs’
   1. Student.cs should inherit StudentBase
   2. It should define an ID field of type Guid, that gets autogenerated by database. See the sample code for an example. You are on your own.
3. At the root of the ‘Data’ folder create a file called ‘StudentDbContext.cs’
   1. You are on your own
4. Add migration for initial creation of database and apply it to database. If you forgot how to do this, refer to lab 5.
5. At this point all the compile errors you previously ignored should be resolved.

# Create the Controller’s Actions

1. Create StudentsController under Controllers folder
2. Add the following lines to StudentsController, to the beginning of the controller class.

private readonly StudentsDbContext \_context;

public StudentsController(StudentsDbContext context)

{

\_context = context;

}

1. In your StudentsController create the ‘Get’ action

/// <summary>

/// Get collection of Students.

/// </summary>

/// <returns>A collection of Students</returns>

/// <response code="200">Returns a collection of Students</response>

/// <response code="500">Internal error</response>

[HttpGet]

[ProducesResponseType(StatusCodes.Status200OK)]

[ProducesResponseType(StatusCodes.Status500InternalServerError)]

public async Task<ActionResult<IEnumerable<Student>>> Get()

{

return Ok(await \_context.Students.ToListAsync());

}

1. Follow the sample code I posted for Cars, and create actions for:
   1. GetById
   2. Create
   3. Delete
2. Create an action for:
   1. Update
   2. When you Update a resource, we try to update the resource and if the resource does not exist, we return a 404 response.
3. Make sure you put the right attributes, routing, and docstring
4. Push your code to Github

# Publish

1. Refer to previous labs for publishing instructions. Note that there is one additional step to the process. On the “API Management” set up step, check the “Skip this step” and hit finish.

# Deliverable

Deploy your website to Windows Azure and push the code to GitHub. Submit the link to Brightspace.

Answer the following questions with your submission to Brightspace (as text submission):

1. Define each HTTP Status Code that you used in this project and explain where/when you should use them.
2. Does the service you created in this assignment conform to all REST principals? Explain why.

# Grading Scheme

|  |  |
| --- | --- |
| model | 6 |
| context | 2 |
| migration | 1 |
| Controller | 15 |
| dbinitializer | 2 |
| q1 | 12 |
| q2 | 5 |
| gitignore | 2 |
| working assignment | 5 |
| total | 50 |