Lab 6 – Web API

Tyler King 040979598

# 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.

# Create a new MVC Core project called ‘Lab6’

1. Create new ASP.NET Core Empty Application called Lab6.
   1. Uncheck the HTTPS box
   2. 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: 6.0.3 (or newer)
   2. "Microsoft.EntityFrameworkCore.Sqlite” >= v: 6.0.3 (or newer)
   3. "Microsoft.EntityFrameworkCore.Tools" >= v: 6.0.3 (or newer)
   4. “Microsoft.VisualStudio.Web.CodeGeneration.Design” >= v:6.0.2(or newer)
   5. “Microsoft.EntityFrameworkCore.SqlServer” >= v: 6.0.3 (or newer)
   6. “Swashbuckle.AspNetCore” >= v:6.3.0 (or newer)
   7. “Swashbuckle.AspNetCore.Annotations” >= v:6.3.0 (or newer)

# Configure your new Web Application

1. Open your project file, and comment out (or delete) the line: <Nullable>enable</Nullable>
2. At the root of the project create a folder called ‘Data’
3. 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. Clear the content of the Program.cs file and replace it with the code at: <https://gist.github.com/aarad-ac/dc15a10e0d797cb9e64ef8ee99bf1e2a>
2. Then replace /\*LabName\*/ with Lab6, and replace /\*DBContext\*/ with StudentDbContext
3. Ignore the syntax errors around ` StudentDbContext` and `Startup` but fix the rest of them by adding the right ‘using’ statements (you are on your own)
4. See the Azure SQL document in Brightspace -> extra materials to set up your db and find your database connection string
5. 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. In solution Explorer, right click on the project file, and add a class called Startup.cs
2. Delete the content of Startup.cs and replace it with the code from: <https://gist.github.com/aarad-ac/795e94e13926a6a2ebb665b774b3444a>
3. Modify ‘Startup.cs’. replace /\* Define services here \*/ with the following code. Ignore errors around `StudentDbContext` and fix the rest by adding the right ‘using’ statements (you are on your own):

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

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

services.AddControllers();

services.AddSwaggerGen(c =>

{

c.EnableAnnotations();

});

1. Modify ‘Startup.cs’. Replace /\* Define routing here \*/ with the appropriate code (you are on your own)
2. Replace the following lines:

else

{

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

}

app.UseStaticFiles();

with these lines:

app.UseSwagger();

app.UseSwaggerUI(c =>

{

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

c.RoutePrefix = string.Empty;

});

# 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 ‘Student.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. It should define an ID field of type Guid, that gets autogenerated by database. Refer to the Entity Framework lecture. You are on your own.
         1. Annotate the ID with this: [SwaggerSchema(ReadOnly = true)]
      2. FirstName
      3. LastName
      4. Program
2. At the root of the ‘Data’ folder create a file called ‘StudentDbContext.cs’
   1. You are on your own
3. Add migration for initial creation of database and apply it to database. If you forgot how to do this, refer to lab 5.
4. At this point all the compile errors you previously ignored should be resolved.

# Scaffold the Controller’s Actions

1. Right click on the Controllers folder, then select:
   1. Add > New scaffolded item..
   2. API Controller with actions using Entity Framework
   3. Scaffold Student model
2. In the Scaffolded controller, modify the action responding to PUT to return a Student when the request is successful
3. In your StudentsController annotate every action with the right ProducesResponseType annotations and explain why you added that annotation in a comment. For instance in ‘Get’ action

[HttpGet]

[ProducesResponseType(StatusCodes.Status200OK)] // returned when we return list of Students successfully

[ProducesResponseType(StatusCodes.Status500InternalServerError)] // returned when there is an error in processing the request

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

{

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

}

# 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 what each code means.
2. Does the service you created in this assignment conform to all REST principals? Explain why.
   1. The service I’ve created in this assignment conforms to rest principals as it’s comprised of client (browser), server (azure sql), and api (controller class methods). These all conform to interact using rest principles.
   2. The client and server are separated with different functions as the client displays info received from the server who’s only concern is storing, altering, and moving data all without sessions (stateless).
   3. Uniform interface as a principal is followed as only the URI of the initial page is needed to navigate, resources carry the info needed on how to process it, interface identifies which resource in involved in a interaction from client to server. Resources are modified through representations in HTTP standards (i.e GET).
   4. Data is cached and if the cache doesn’t have that info its retrieved from server (i.e default get request)
   5. System is layered as what the client sees is operated on a separate database from where student info is stored (Azure page hosting -> Azure SQL)

# Grading Scheme

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