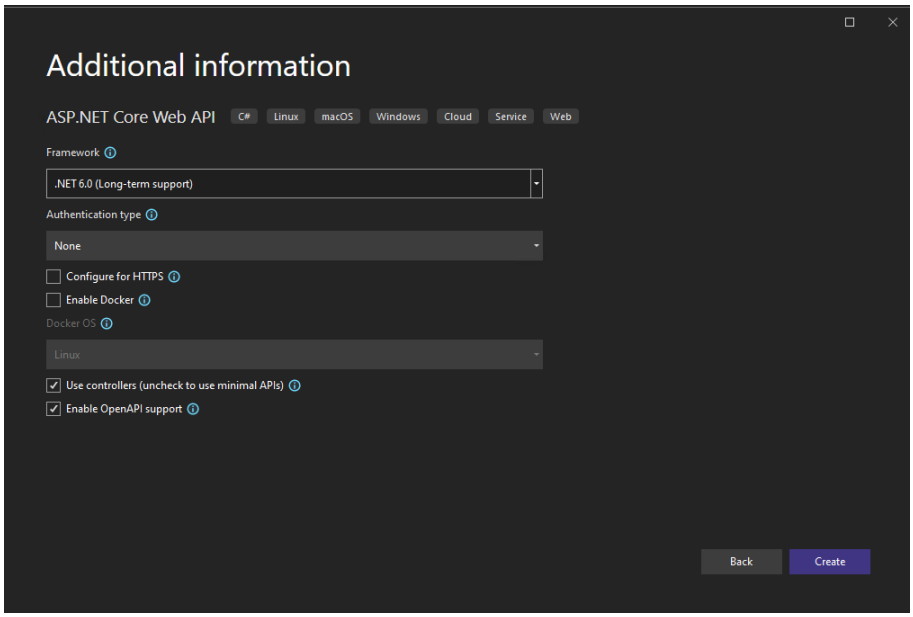
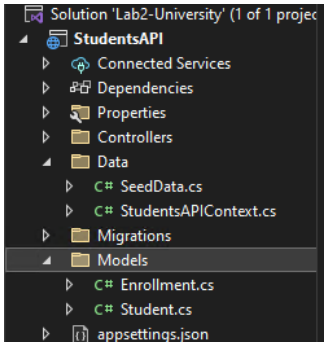
Lab 2.1 Create a Docker container-based Web API with ASP.NET Core

- Create a web API project.
- Add a model class and a database context.
- Scaffold a controller with CRUD methods.
- Configure routing, URL paths, and return values.
- Add Docker Support.
- Run SQL Server as a Docker Container.
- Configure connection string to database.

Create a web project:



Add a model class `Student` and `Enrollment`



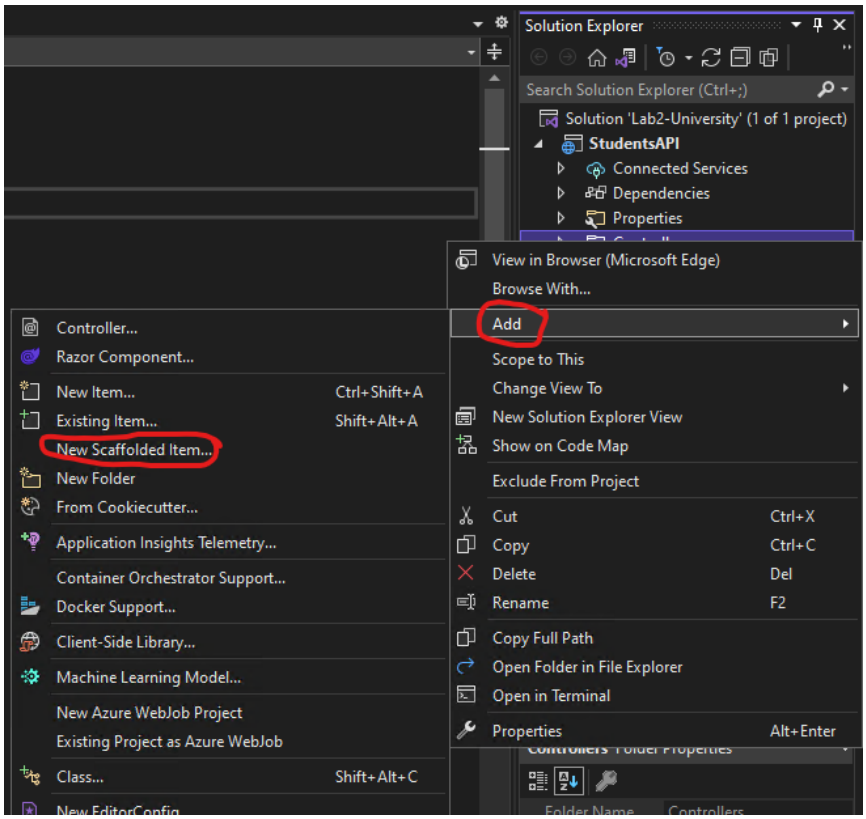

```
namespace StudentsAPI.Models
{
    public class Student
    {
        public int ID { get; set; }
        public string LastName { get; set; }
        public string FirstMidName { get; set; }
        public DateTime EnrollmentDate { get; set; }
        public ICollection<Enrollment> Enrollments { get; set; }
    }
    public enum Grade
    {
        A, B, C, D, F
    }
    public class Enrollment
    {
        public int EnrollmentID { get; set; }
        public int CourseID { get; set; }
        public string Title { get; set; }
        public int Credits { get; set; }
        public int StudentID { get; set; }
```

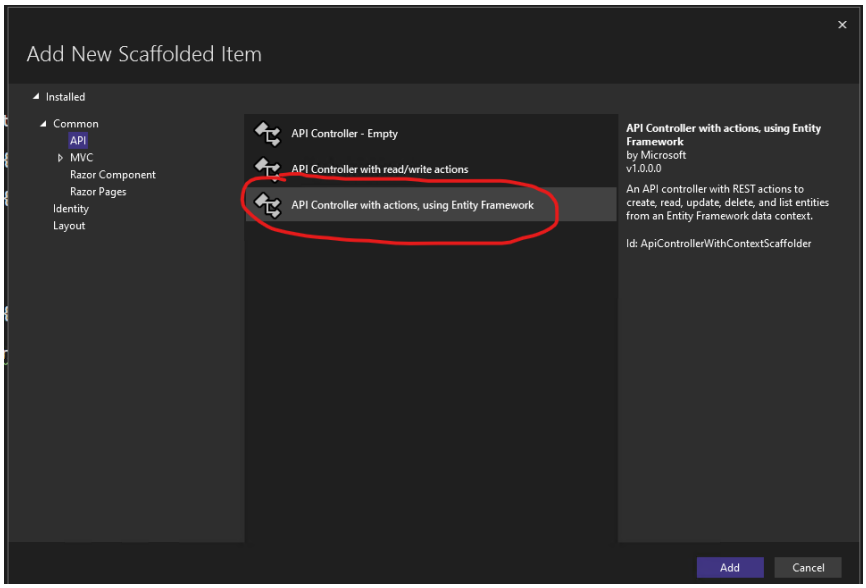
```
        public Grade? Grade { get; set; }
        public Student Student { get; set; }
    }
}
```

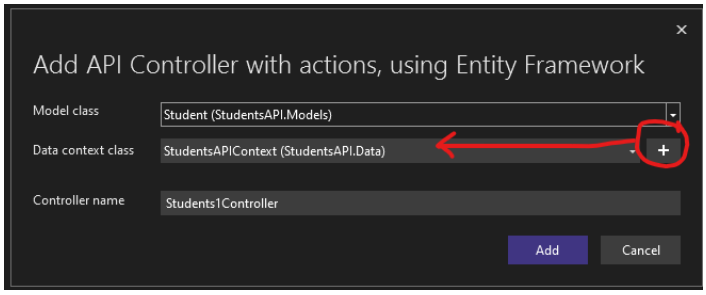
Add NuGet packages: `Microsoft.EntityFrameworkCore.Design;`

`Microsoft.EntityFrameworkCore.SqlServer`

Use the scaffolding tool to produce Create, Read, Update, and Delete (CRUD) pages for the student model.







Add the `Enrollment to StudentsAPIContext`

```
    public class StudentsAPIContext : DbContext
    {
        public StudentsAPIContext (DbContextOptions<StudentsAPIContext> options)
            : base(options)
        {
        }
        public DbSet<StudentsAPI.Models.Student> Student { get; set; }
        public DbSet<StudentsAPI.Models.Enrollment> Enrollment { get; set; }
    }
```

Seed the database

Create a new class named *SeedData* in the *Data* folder. Replace the generated code with the following:

```
using StudentsAPI.Models;
using Microsoft.EntityFrameworkCore;

namespace StudentsAPI.Data
{
    public class SeedData
    {
        public static void Initialize(IServiceProvider serviceProvider)
        {
            using (var context = new StudentsAPIContext(
                serviceProvider.GetRequiredService<DbContextOptions<StudentsAPIContext>>()))
            {
                context.Database.EnsureCreated();
                // Look for any Student.
                if (context.Student.Any()){
                    return;   // DB has been seeded
                }
                context.Student.AddRange(
                    new Student { FirstMidName = "Carson", LastName = "Alexander", EnrollmentDate = DateTime.Parse("2005-09-01") },
                    new Student { FirstMidName = "Meredith", LastName = "Alonso", EnrollmentDate = DateTime.Parse("2002-09-01") },
                    new Student { FirstMidName = "Arturo", LastName = "Anand", EnrollmentDate = DateTime.Parse("2003-09-01") },
                    new Student { FirstMidName = "Gytis", LastName = "Barzdukas", EnrollmentDate = DateTime.Parse("2002-09-01") },
                    new Student { FirstMidName = "Yan", LastName = "Li", EnrollmentDate = DateTime.Parse("2002-09-01") },
                    new Student { FirstMidName = "Peggy", LastName = "Justice", EnrollmentDate = DateTime.Parse("2001-09-01") },
                    new Student { FirstMidName = "Laura", LastName = "Norman", EnrollmentDate = DateTime.Parse("2003-09-01") },
                    new Student { FirstMidName = "Nino", LastName = "Olivetto", EnrollmentDate = DateTime.Parse("2005-09-01") }
                );
                context.SaveChanges();
                context.Enrollment.AddRange(
                    new Enrollment { StudentID = 1, CourseID = 1050, Title = "Chemistry", Credits = 3, Grade = Grade.A },
                    new Enrollment { StudentID = 1, CourseID = 4022, Title = "Microeconomics", Credits = 3, Grade = Grade.C },
                    new Enrollment { StudentID = 1, CourseID = 4041, Title = "Macroeconomics", Credits = 3, Grade = Grade.B },
                    new Enrollment { StudentID = 2, CourseID = 1045, Title = "Calculus", Credits = 4, Grade = Grade.B },
                    new Enrollment { StudentID = 2, CourseID = 3141, Title = "Trigonometry", Credits = 4, Grade = Grade.F },
                    new Enrollment { StudentID = 2, CourseID = 2021, Title = "Composition", Credits = 3, Grade = Grade.F },
                    new Enrollment { StudentID = 3, CourseID = 1050, Title = "Chemistry", Credits = 3 },
                    new Enrollment { StudentID = 4, CourseID = 1050, Title = "Chemistry", Credits = 3 },
                    new Enrollment { StudentID = 4, CourseID = 4022, Title = "Microeconomics", Credits = 3, Grade = Grade.F },
                    new Enrollment { StudentID = 5, CourseID = 4041, Title = "Macroeconomics", Credits = 3, Grade = Grade.C },
                    new Enrollment { StudentID = 6, CourseID = 1045, Title = "Calculus", Credits = 4 },
                    new Enrollment { StudentID = 7, CourseID = 3141, Title = "Trigonometry", Credits = 4, Grade = Grade.A }
                );
                context.SaveChanges();
            }
        }
    }
}
```

Add the seed initializer:

Replace the contents of *Program.cs* with the following code. The new code is highlighted.

```
using Microsoft.EntityFrameworkCore;
using StudentsAPI.Data;
using System.Text.Json.Serialization;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddDbContext<StudentsAPIContext>(options =>
    options.UseSqlServer(builder.Configuration.GetConnectionString("StudentsAPIContext")));

// Add services to the container.
//builder.Services.AddControllers();

builder.Services.AddControllers().AddJsonOptions(x =>
                x.JsonSerializerOptions.ReferenceHandler = ReferenceHandler.IgnoreCycles);
// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

var app = builder.Build();

using (var scope = app.Services.CreateScope())
{
    var services = scope.ServiceProvider;
    SeedData.Initialize(services);
}

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseAuthorization();

app.MapControllers();

app.Run();
```

Replace the `[HttpGet("{id}")]` of *StudentsController.cs* with the following code. The new code is highlighted.

```
        // GET: api/Students/5
        [HttpGet("{id}")]
        public async Task<ActionResult<Student>> GetStudent(int id)
        {
            //var student = await _context.Student.FindAsync(id);
            var student = await _context.Student
                .Include(s => s.Enrollments)
                .AsNoTracking()
                .FirstOrDefaultAsync(m => m.ID == id);

            if (student == null)
            {
                return NotFound();
            }

            return student;
        }
```

Test web API:

StudentsAPI 1.0 OAS3

http://localhost:5193/swagger/v1/swagger.json

Students

GET /api/Students

POST /api/Students

GET /api/Students/{id}

Parameters | Cancel

Name	Description
id * required integer($int32) (path)	1

Execute | Clear

Responses

Curl

```
curl -X 'GET' \
  'http://localhost:5193/api/Students/1' \
  -H 'accept: text/plain'
```

Request URL

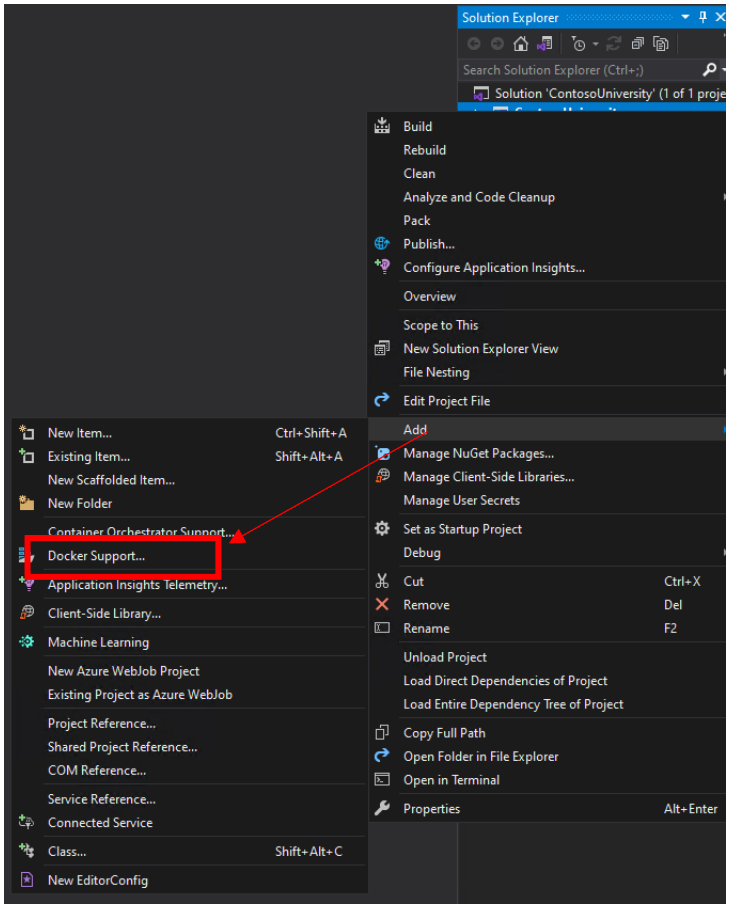
```
http://localhost:5193/api/Students/1
```

Server response

Code	Details
200	Response body


```
{
  "id": 1,
  "lastName": "Alexander",
  "firstMidName": "Carson",
  "enrollmentDate": "2005-09-01T00:00:00",
  "enrollments": [
    {
      "enrollmentID": 1,
      "courseID": 1050,
      "title": "Chemistry",
      "credits": 3,
      "studentID": 1,
      "grade": 0,
      "student": null
    },
    {
      "enrollmentID": 2,
      "courseID": 4022,
      "title": "Microeconomics",
      "credits": 3,
      "studentID": 1,
      "grade": 2,
      "student": null
    },
    {
```

Add Docker Support



Docker file

```
FROM mcr.microsoft.com/dotnet/aspnet:6.0 AS base
WORKDIR /app
EXPOSE 80

FROM mcr.microsoft.com/dotnet/sdk:6.0 AS build
WORKDIR /src
COPY ["StudentsAPI/StudentsAPI.csproj", "StudentsAPI/"]
RUN dotnet restore "StudentsAPI/StudentsAPI.csproj"
COPY . .
WORKDIR "/src/StudentsAPI"
RUN dotnet build "StudentsAPI.csproj" -c Release -o /app/build

FROM build AS publish
RUN dotnet publish "StudentsAPI.csproj" -c Release -o /app/publish

FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "StudentsAPI.dll"]
```

How to run SQL Server as a Docker Container

You'll need Docker Desktop installed and be running in the Linux mode to follow these instructions.

We'll use the docker run command, with -d to run in the background, --name to give our container a meaningful name, -p to expose port 1433, and setting two environment variables with -e to set the SA password and accept the EULA. The image we'll run is mcr.microsoft.com/mssql/server:2019-latest

```
docker run -d -e "ACCEPT_EULA=Y" -e "SA_PASSWORD=My!P@ssw0rd1" -p 1433:1433 --name Universitydb
mcr.microsoft.com/mssql/server:2019-latest
```

Our Docker SQL container is visible on localhost, so our connection string will look like this

```
Server=localhost;User Id=sa;Password=My!P@ssw0rd1;
```

Using volumes

Now, if we stop our database with `docker stop Universitydb` and then restart it with `docker start Universitydb`, the table and data in the database will be retained. But if we remove the container with `docker rm -f Universitydb` then any data is lost. What if we want to keep our data, but not the container?

We can do that with volumes. Let's add one more parameter to our docker run command. This is `-v customervol:/var/opt/mssql`, which will "mount" a "volume" to our container. This volume is called `customervol` and will be created automatically if it doesn't exist. We'll then mount that to the location `/var/opt/mssql` inside our container. This is where SQL Server writes our data, so this will put that data into the volume.

```
docker run -d -e "ACCEPT_EULA=Y" -e "SA_PASSWORD=My!P@ssw0rd1" -p 1433:1433 --name Universitydb -v
universitydbvol:/var/opt/mssql mcr.microsoft.com/mssql/server:2019-latest
```

Now we can connect and run the same times as before, but this time if we remove the container with `docker rm -f Universitydb` our volume will still exist. We can see the volumes we have with docker volume ls and delete the volume with `docker volume rm Universitydb` if we want to get rid of it.

How to Communicate Between Docker Containers

If you are running more than one container, you can let your containers communicate with each other by attaching them to the same network.

Docker creates virtual networks which let your containers talk to each other. In a network, a container has an IP address, and optionally a hostname.

You can create different types of networks depending on what you would like to do. We'll cover the easiest options:

- The **default bridge network**, which allows simple container-to-container communication by IP address, and is created by default.
- A **user-defined bridge network**, which you create yourself, and allows your containers to communicate with each other, by using their container name as a hostname.

```
C: >docker network ls

NETWORK ID     NAME      DRIVER    SCOPE

a07ee2e707d9   bridge    bridge    local

2da0102bd2ad   host      host      local

c89679c0de6e   none      null      local
```

`C: >docker network inspect` Display detailed information on one or more networks

Default bridge network

When Docker starts up, it will create a default network called… `bridge`.

From that point onwards, all containers are added into to the bridge network, unless you say otherwise

In a **bridge network, each container is assigned its own IP address**. So containers can communicate with each other by IP.

```
# Windows CMD

C:\>docker inspect Universitydb | findstr IPAddress

        "SecondaryIPAddresses": null,

        "IPAddress": "172.17.0.3",

                "IPAddress": "172.17.0.3",
```

appsettings.json

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "StudentsAPIContext": "Server=172.17.0.3;Database=ContosoUniversity1;User Id=sa;Password=My!P@ssw0rd1"
  }
}
```