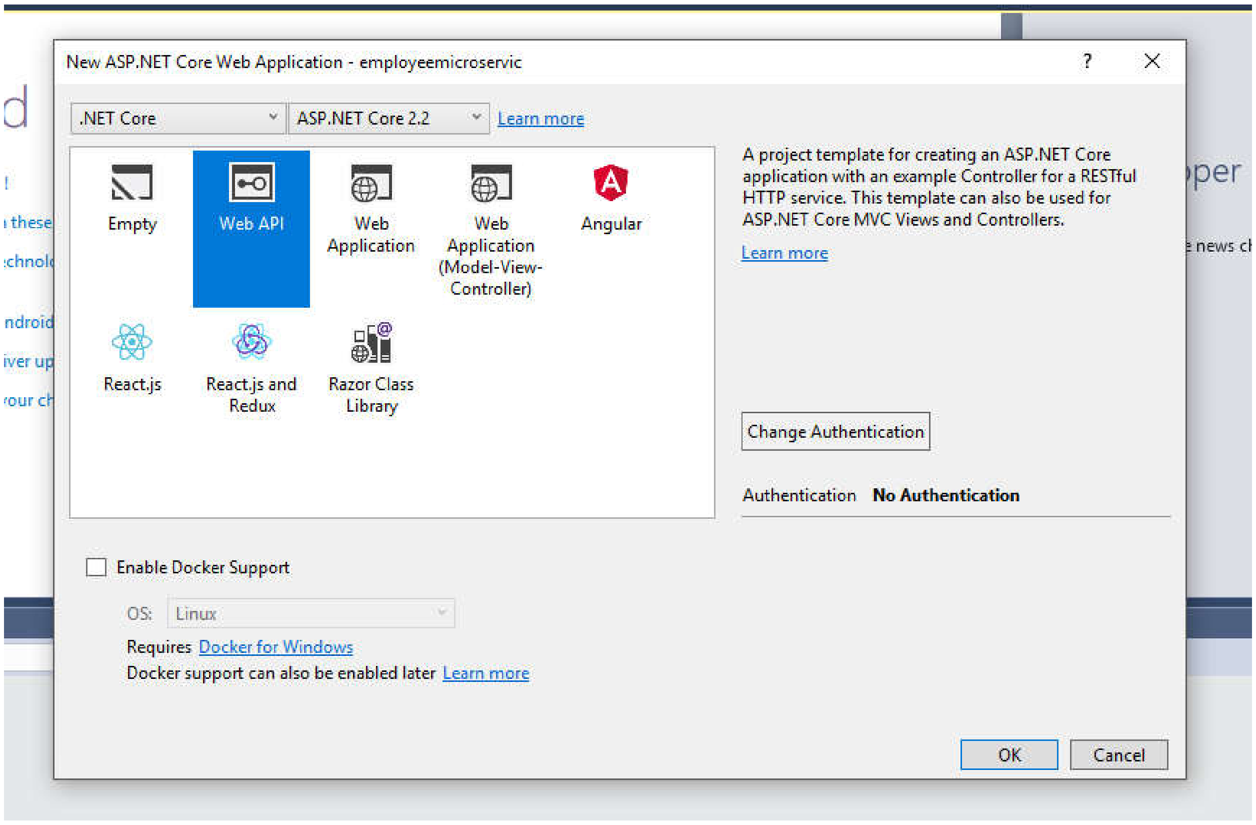
Building Web Api microservice using Sql Server database

Step 1 : open visual studio 2017 and create new web Api project



New ASP.NET Core Web Application - employeemicroservic

.NET Core | ASP.NET Core 2.2 | Learn more

A project template for creating an ASP.NET Core application with an example Controller for a RESTful HTTP service. This template can also be used for ASP.NET Core MVC Views and Controllers.

Learn more

Change Authentication

Authentication **No Authentication**

☐ Enable Docker Support

OS: Linux

Requires Docker for Windows

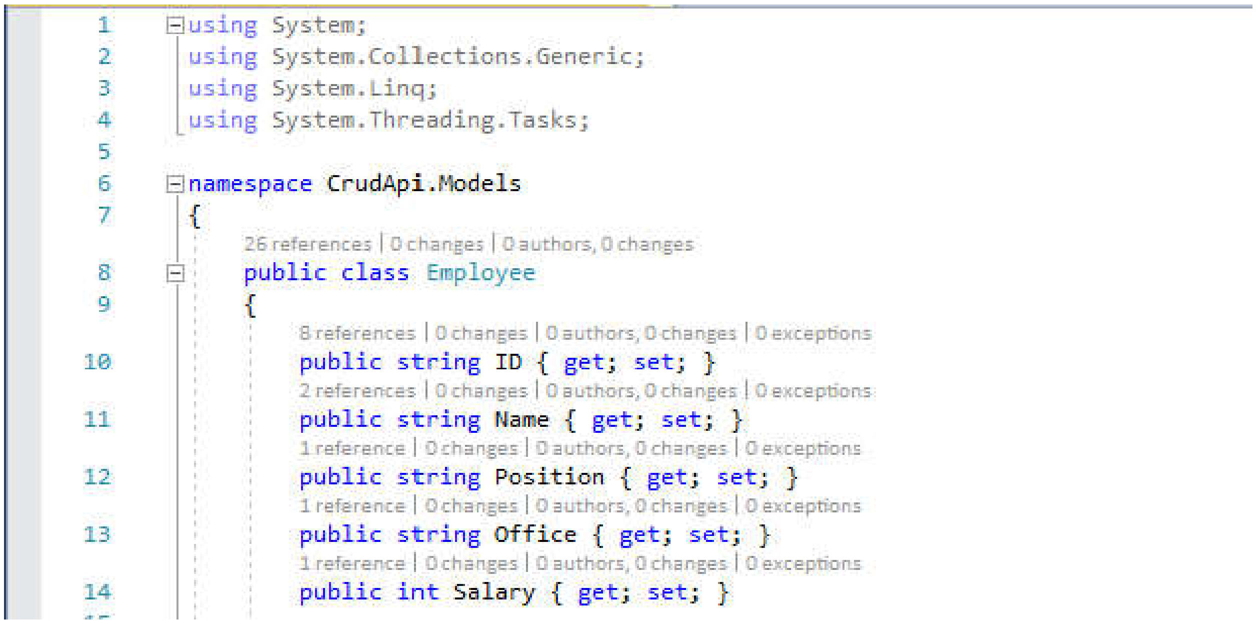
Docker support can also be enabled later Learn more

OK | Cancel

NB : This time we don’t need to check the Enable Docker Support checkbox. We are going to have the normal web api project and add the docker file explicitly after some steps here…

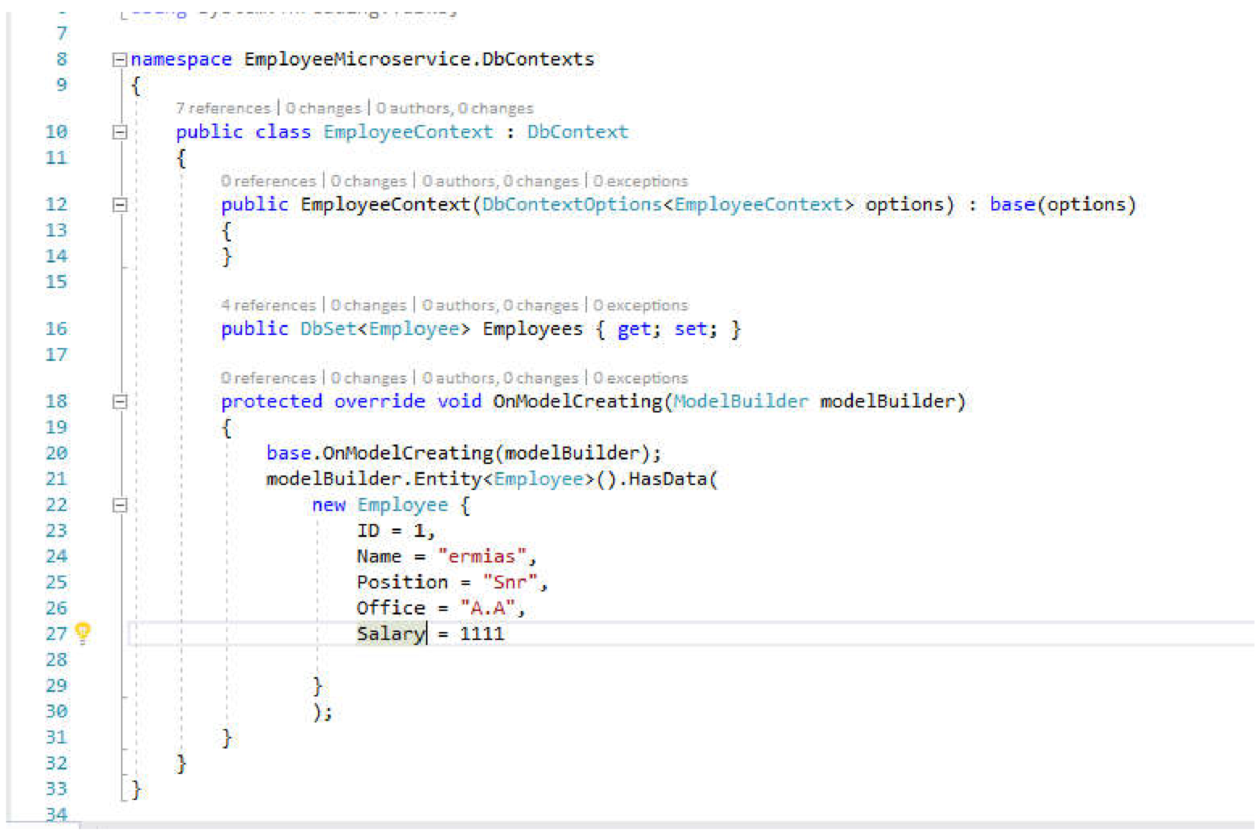
Step 2 : Add all of the folders(models, controller, Repository and DbContext) that we added on the last demo we have done last week.

Step 3 : create employee schema model for the data inside the Models Folder that we have created before.



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace CrudApi.Models
{
    26 references | 0 changes | 0 authors, 0 changes
    public class Employee
    {
        8 references | 0 changes | 0 authors, 0 changes | 0 exceptions
        public string ID { get; set; }
        2 references | 0 changes | 0 authors, 0 changes | 0 exceptions
        public string Name { get; set; }
        1 reference | 0 changes | 0 authors, 0 changes | 0 exceptions
        public string Position { get; set; }
        1 reference | 0 changes | 0 authors, 0 changes | 0 exceptions
        public string Office { get; set; }
        1 reference | 0 changes | 0 authors, 0 changes | 0 exceptions
        public int Salary { get; set; }
```

Step 4 : Add EmployeeContext class to the DbContexts folder

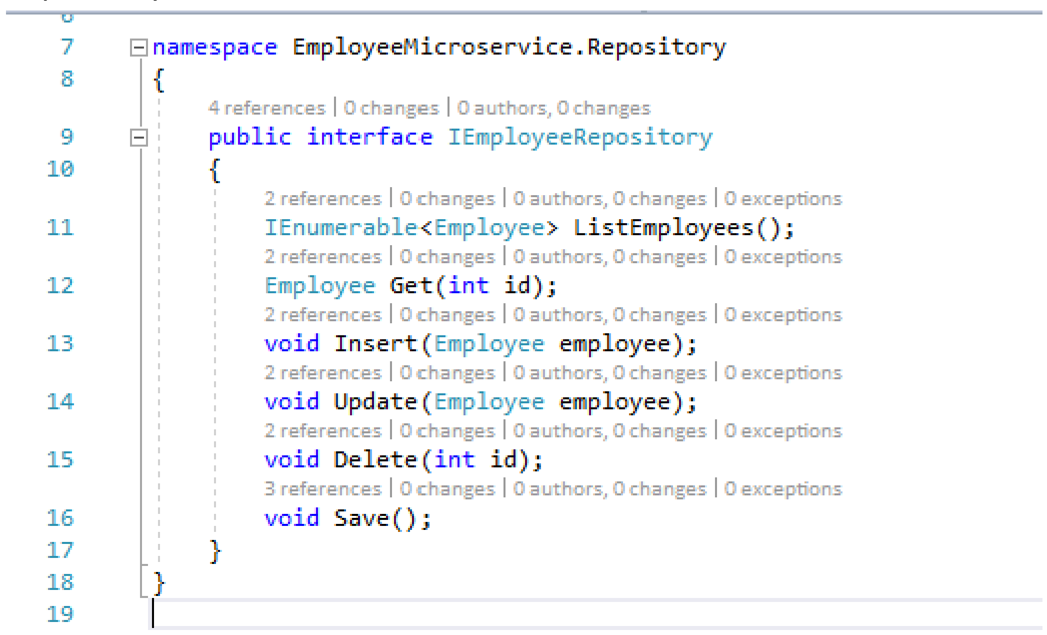


```
namespace EmployeeMicroservice.DbContexts
{
    7 references | 0 changes | 0 authors, 0 changes
    public class EmployeeContext : DbContext
    {
        0 references | 0 changes | 0 authors, 0 changes | 0 exceptions
        public EmployeeContext(DbContextOptions<EmployeeContext> options) : base(options)
        {
        }

        4 references | 0 changes | 0 authors, 0 changes | 0 exceptions
        public DbSet<Employee> Employees { get; set; }

        0 references | 0 changes | 0 authors, 0 changes | 0 exceptions
        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            base.OnModelCreating(modelBuilder);
            modelBuilder.Entity<Employee>().HasData(
                new Employee {
                    ID = 1,
                    Name = "ermias",
                    Position = "Snr",
                    Office = "A.A",
                    Salary = 1111

                }
                );
        }
    }
}
```

NB: we use this class to get in touch with the Database. We can perform CRUD operations on the Table Employees by first importing this class.

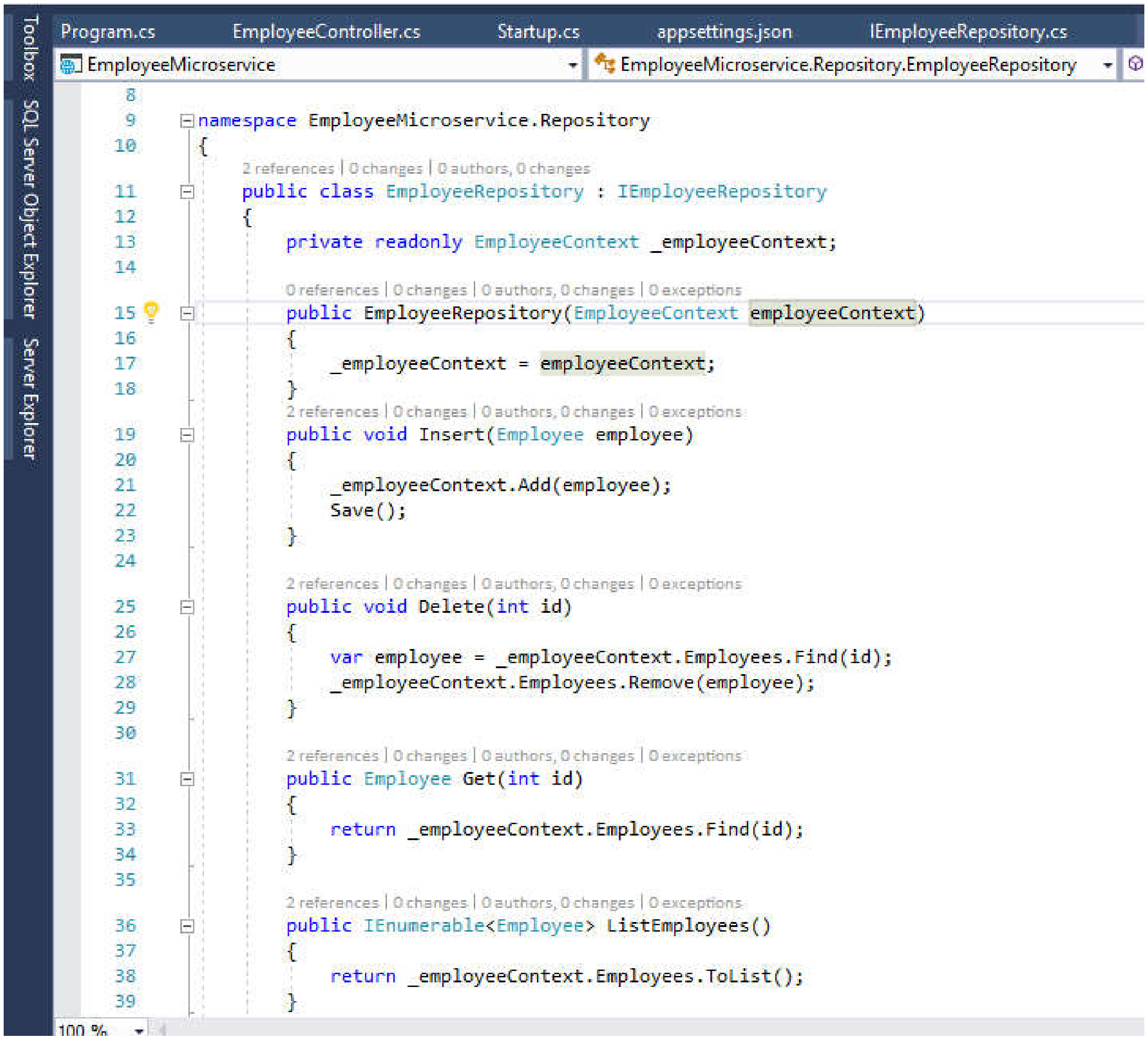
(when we add migration the table that we stated as DbSet will be created based on the Employee schema that we created before).

Step 5 : create classes EmployeeRepository and IEmployeeRepository inside Repository folder.



```
namespace EmployeeMicroservice.Repository
{
    4 references | 0 changes | 0 authors, 0 changes
    public interface IEmployeeRepository
    {
        2 references | 0 changes | 0 authors, 0 changes | 0 exceptions
        IEnumerable<Employee> ListEmployees();
        2 references | 0 changes | 0 authors, 0 changes | 0 exceptions
        Employee Get(int id);
        2 references | 0 changes | 0 authors, 0 changes | 0 exceptions
        void Insert(Employee employee);
        2 references | 0 changes | 0 authors, 0 changes | 0 exceptions
        void Update(Employee employee);
        2 references | 0 changes | 0 authors, 0 changes | 0 exceptions
        void Delete(int id);
        3 references | 0 changes | 0 authors, 0 changes | 0 exceptions
        void Save();
    }
}
```

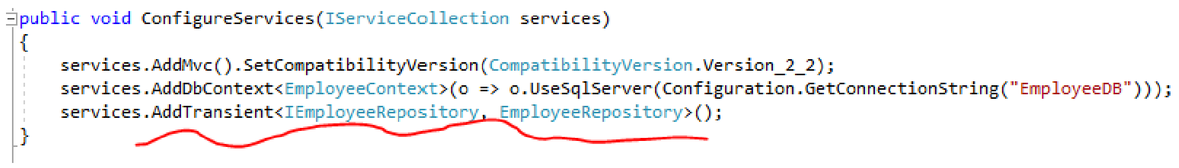
EmployeeRepository.cs



```
namespace EmployeeMicroservice.Repository
{
    2 references | 0 changes | 0 authors, 0 changes
    public class EmployeeRepository : IEmployeeRepository
    {
        private readonly EmployeeContext _employeeContext;

        0 references | 0 changes | 0 authors, 0 changes | 0 exceptions
        public EmployeeRepository(EmployeeContext employeeContext)
        {
            _employeeContext = employeeContext;
        }
        2 references | 0 changes | 0 authors, 0 changes | 0 exceptions
        public void Insert(Employee employee)
        {
            _employeeContext.Add(employee);
            Save();
        }

        2 references | 0 changes | 0 authors, 0 changes | 0 exceptions
        public void Delete(int id)
        {
            var employee = _employeeContext.Employees.Find(id);
            _employeeContext.Employees.Remove(employee);
        }

        2 references | 0 changes | 0 authors, 0 changes | 0 exceptions
        public Employee Get(int id)
        {
            return _employeeContext.Employees.Find(id);
        }

        2 references | 0 changes | 0 authors, 0 changes | 0 exceptions
        public IEnumerable<Employee> ListEmployees()
        {
            return _employeeContext.Employees.ToList();
        }
```

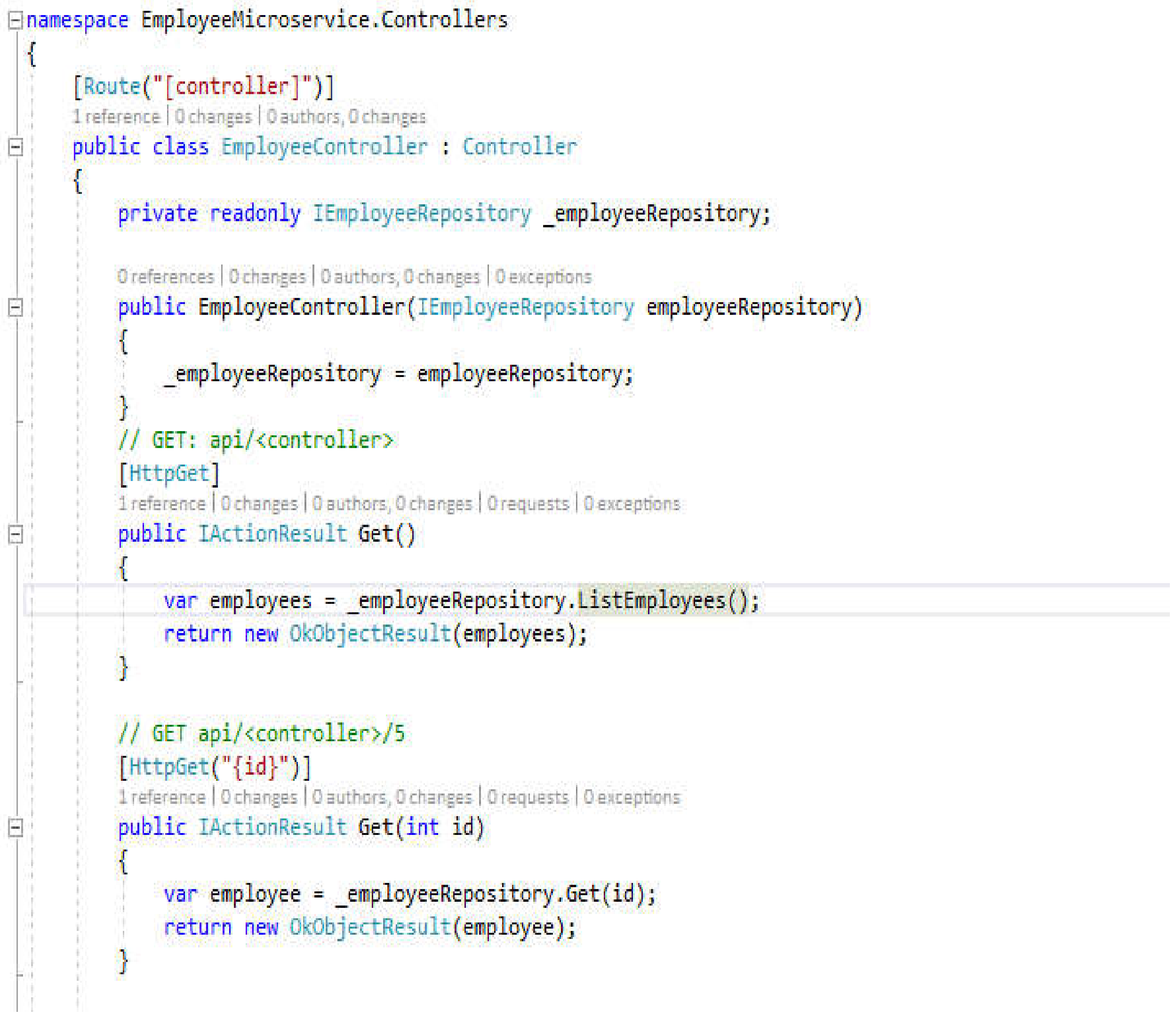
```
        3 references | 0 changes | 0 authors, 0 changes | 0 exceptions
        public void Save()
        {
            _employeeContext.SaveChanges();
        }

        2 references | 0 changes | 0 authors, 0 changes | 0 exceptions
        public void Update(Employee employee)
        {
            _employeeContext.Entry(employee).State = EntityState.Modified;
            Save();
        }
    }
}
```

Step 6 : Add the repository classes to the ConfigurationServices method in the Startup class.

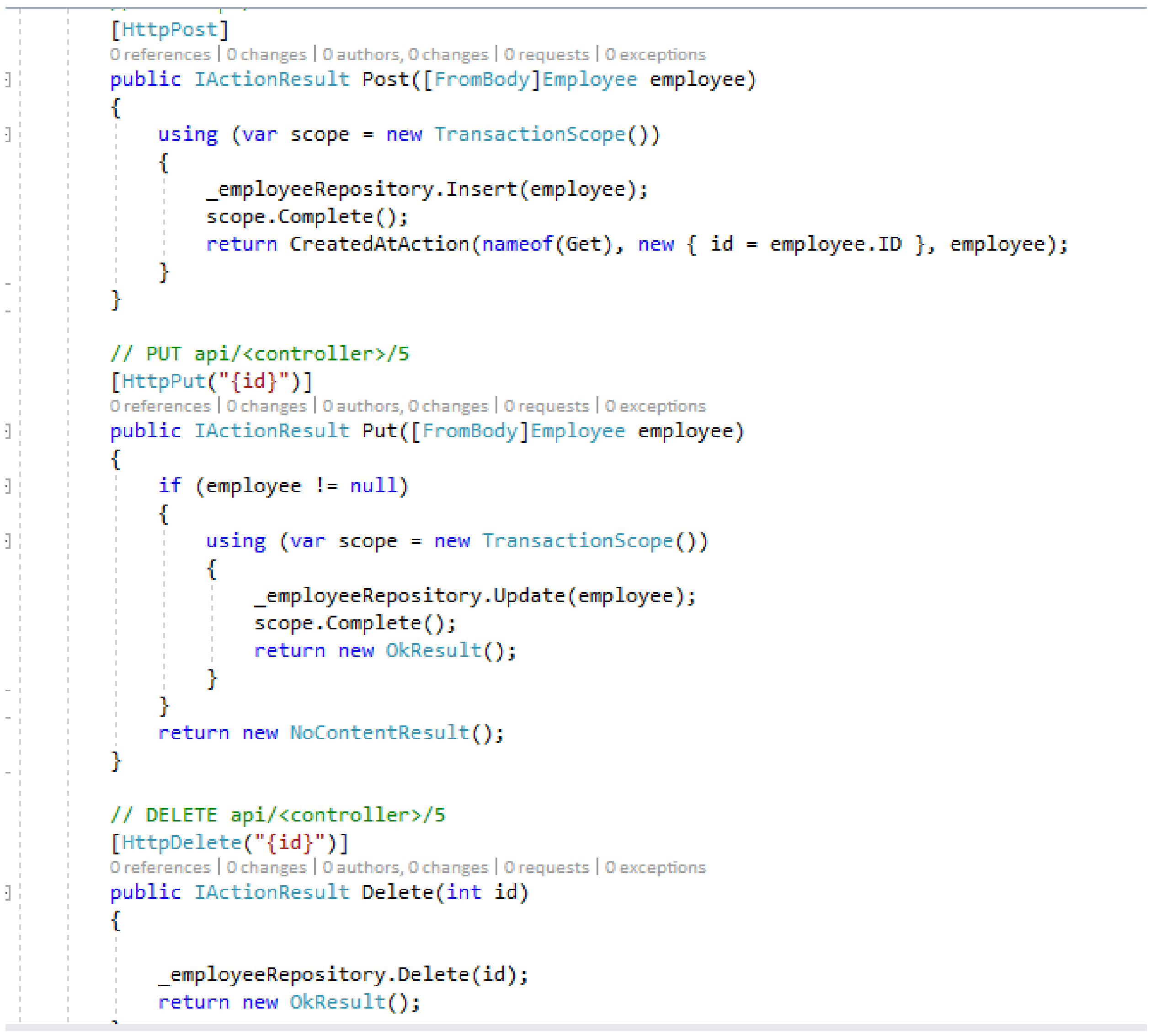


```
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_2);
    services.AddDbContext<EmployeeContext>(o => o.UseSqlServer(Configuration.GetConnectionString("EmployeeDB")));
    services.AddTransient<IEmployeeRepository, EmployeeRepository>();
}
```

Step 7 : add EmployeeController class in the controller folder to control the incoming requests.

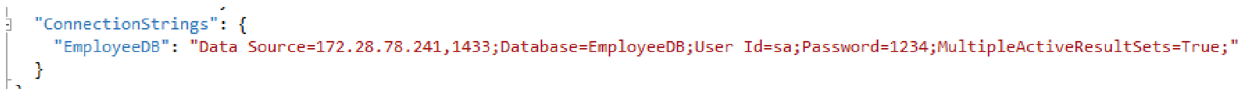


```
namespace EmployeeMicroservice.Controllers
{
    [Route("[controller]")]
    1 reference | 0 changes | 0 authors, 0 changes
    public class EmployeeController : Controller
    {
        private readonly IEmployeeRepository _employeeRepository;

        0 references | 0 changes | 0 authors, 0 changes | 0 exceptions
        public EmployeeController(IEmployeeRepository employeeRepository)
        {
            _employeeRepository = employeeRepository;
        }
        // GET: api/<controller>
        [HttpGet]
        1 reference | 0 changes | 0 authors, 0 changes | 0 requests | 0 exceptions
        public IActionResult Get()
        {
            var employees = _employeeRepository.ListEmployees();
            return new OkObjectResult(employees);
        }

        // GET api/<controller>/5
        [HttpGet("{id}")]
        1 reference | 0 changes | 0 authors, 0 changes | 0 requests | 0 exceptions
        public IActionResult Get(int id)
        {
            var employee = _employeeRepository.Get(id);
            return new OkObjectResult(employee);
        }
```



```
[HttpPost]
0 references | 0 changes | 0 authors, 0 changes | 0 requests | 0 exceptions
public IActionResult Post([FromBody]Employee employee)
{
    using (var scope = new TransactionScope())
    {
        _employeeRepository.Insert(employee);
        scope.Complete();
        return CreatedAtAction(nameof(Get), new { id = employee.ID }, employee);
    }
}

// PUT api/<controller>/5
[HttpPut("{id}")]
0 references | 0 changes | 0 authors, 0 changes | 0 requests | 0 exceptions
public IActionResult Put([FromBody]Employee employee)
{
    if (employee != null)
    {
        using (var scope = new TransactionScope())
        {
            _employeeRepository.Update(employee);
            scope.Complete();
            return new OkResult();
        }
    }
    return new NoContentResult();
}

// DELETE api/<controller>/5
[HttpDelete("{id}")]
0 references | 0 changes | 0 authors, 0 changes | 0 requests | 0 exceptions
public IActionResult Delete(int id)
{

    _employeeRepository.Delete(id);
    return new OkResult();
```

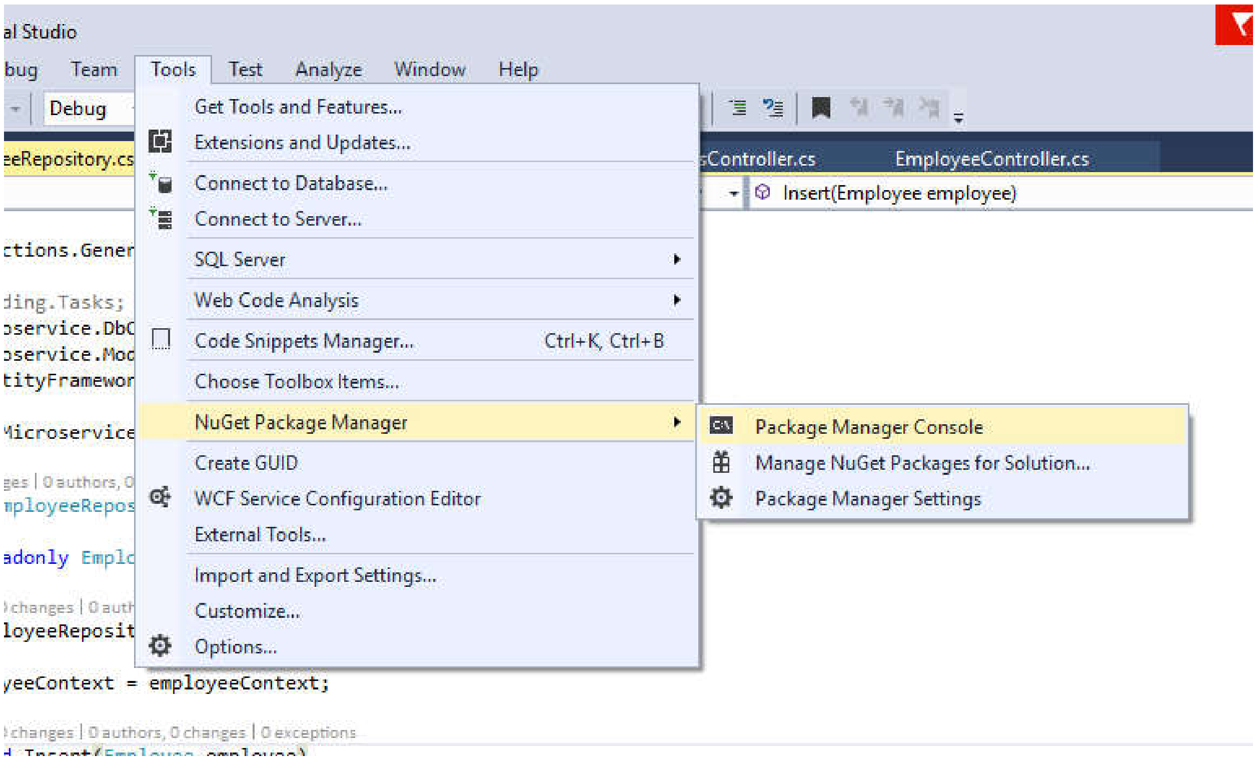
Step 8 : Open appsettings.json and add the appropriate connection string



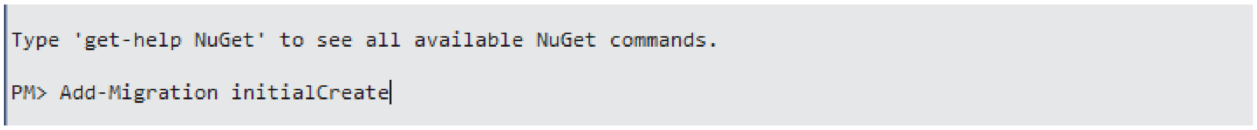
```
"ConnectionStrings": {
  "EmployeeDB": "Data Source=172.28.78.241,1433;Database=EmployeeDB;User Id=sa;Password=1234;MultipleActiveResultSets=True;"
}
```

(a connection string with data source -> the ip of the pc and port number of the sql server, database -> name of the database, Username and password of the sql server)

Step 9 : Save all classes and open the package manager console

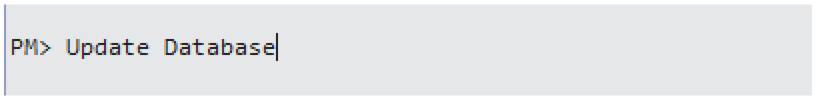


Step 10 : Add migration and name it initialCreate



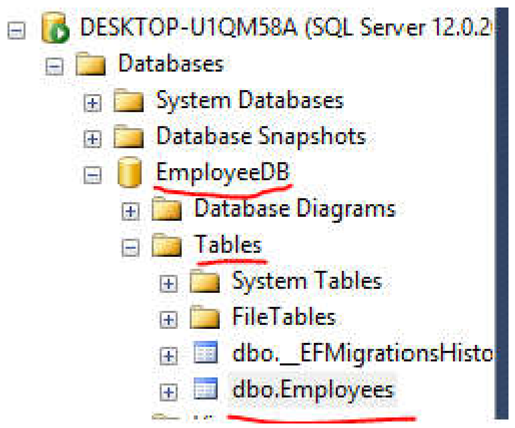
```
Type 'get-help NuGet' to see all available NuGet commands.

PM> Add-Migration initialCreate
```

Step 11 : update database to commit the changes that we have made



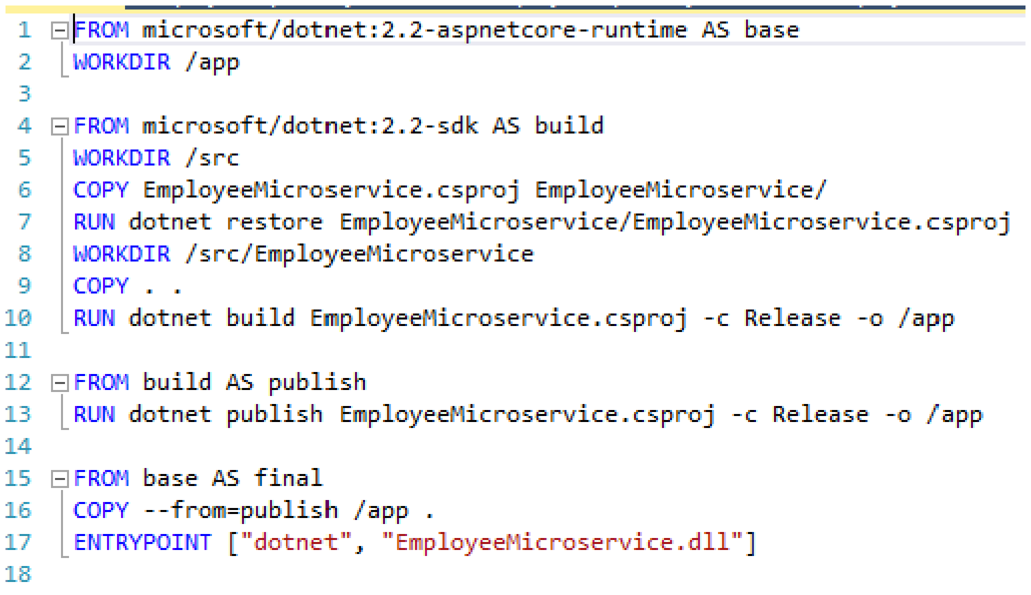
```
PM> Update Database
```

Now we can go and check on Sql server if the table is created and the first sample data is added.



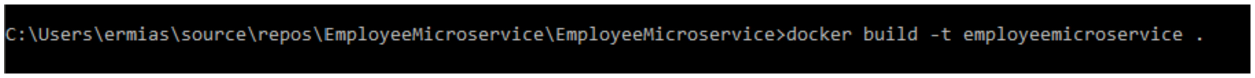
Make it a microservice which can be runnable in docker

➔ Add the following docker file to the project directory



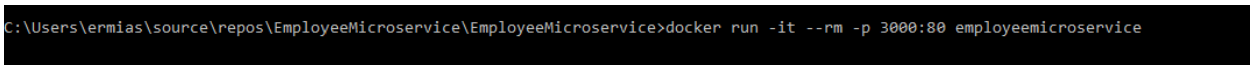
```
FROM microsoft/dotnet:2.2-aspnetcore-runtime AS base
WORKDIR /app

FROM microsoft/dotnet:2.2-sdk AS build
WORKDIR /src
COPY EmployeeMicroservice.csproj EmployeeMicroservice/
RUN dotnet restore EmployeeMicroservice/EmployeeMicroservice.csproj
WORKDIR /src/EmployeeMicroservice
COPY . .
RUN dotnet build EmployeeMicroservice.csproj -c Release -o /app

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

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

➔ Open cmd and browse to the project directory then build a new image of our new emploeemicroservice.



```
C:\Users\ermias\source\repos\EmployeeMicroservice\EmployeeMicroservice>docker build -t employeemicroservice .
```

➔ After all this we are finally able to run our microservice from docker.



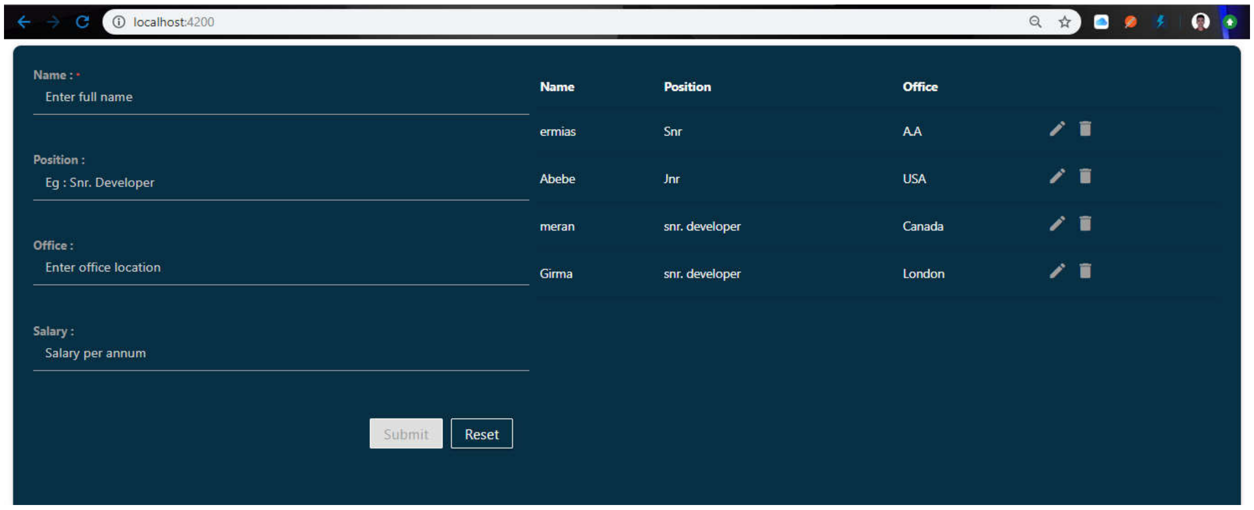
```
C:\Users\ermias\source\repos\EmployeeMicroservice\EmployeeMicroservice>docker run -it --rm -p 3000:80 employeemicroservice
```

Connect the new web api microservice with the CRUD UI that we have done before using Angular 6

➔ Change the port number to 3000

```
readonly baseURL = 'http://localhost:3000/employee';
```

➔ Rebuild and run the Angular image, then open a browse and browse to " localhost:4200 "



Name	Position	Office
ermias	Snr	AA
Abebe	Jnr	USA
meran	snr. developer	Canada
Girma	snr. developer	London

:: well! We are getting the data that we have on our database.