Tutorial 8

Objectives

- Entity Framework
- DbContext
- Migrations
- Database entity CRUD

Create a new Blazor WebAssembly project and enable .NET hosted.

NuGet

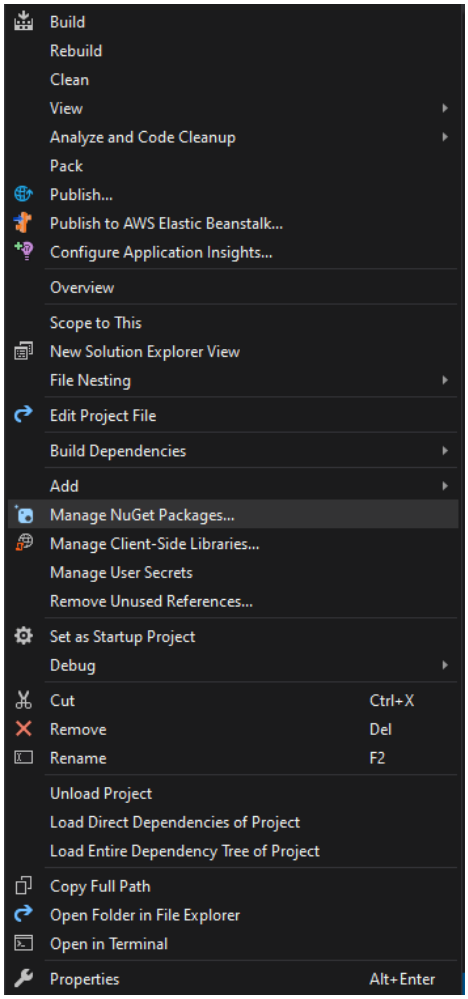
Add the following packages to the server project.

Add package to get access to DbContext:
https://www.nuget.org/packages/Microsoft.EntityFrameworkCore/

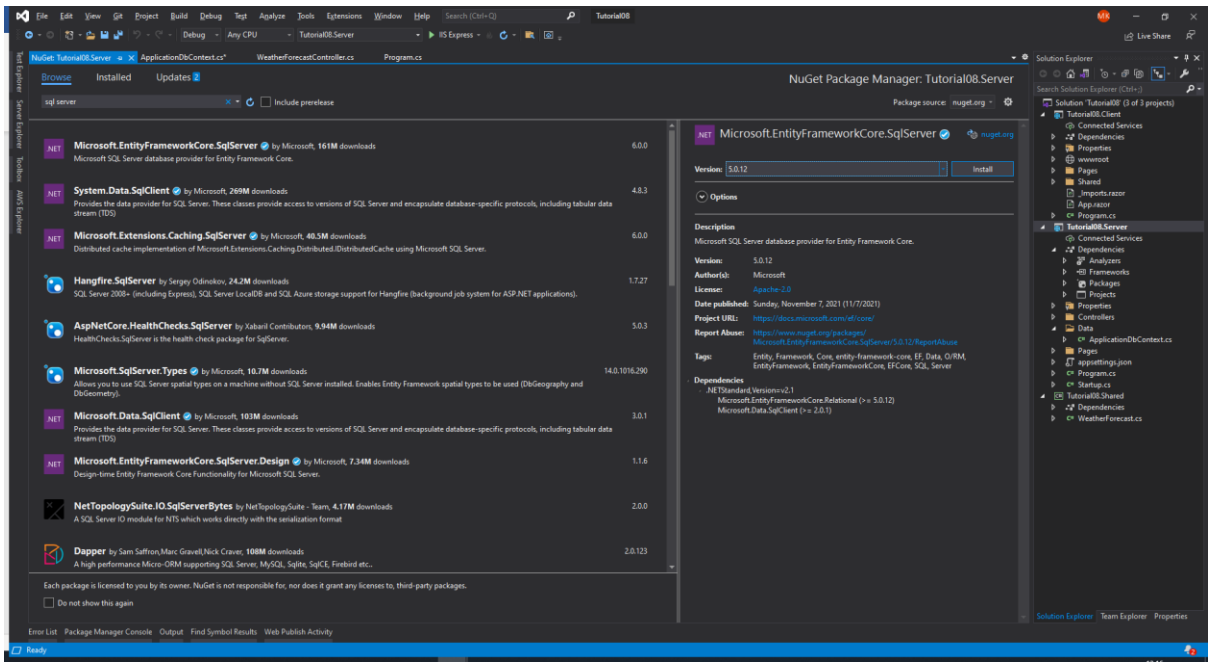
Add package to use UseSqlServer:
https://www.nuget.org/packages/Microsoft.EntityFrameworkCore.SqlServer/

Add package to work with Migrations:
https://www.nuget.org/packages/Microsoft.EntityFrameworkCore.Tools/

Go right click on the server project and select Manage NuGet Packages.



When installing packages make sure to select any 5.x version.



DbContext

Create a new folder in the server project and call it Data. In this folder create a class called ApplicationDbContext and make the class inherit from DbContext.

We will follow the Entity Framework documentation: https://docs.microsoft.com/en-us/ef/core/dbcontext-configuration/ to set up database context.

The ApplicationDbContext class must expose a public constructor with a DbContextOptions<ApplicationDbContext> parameter.

A database set within DbContext behaves as a table in a database. In order to create table "WeatherForecasts", we have to add a new DbSet called "WeatherForecasts".

```
public class ApplicationDbContext : DbContext
{
    0 references
    public DbSet<WeatherForecast> WeatherForecasts { get; set; }
    0 references
    public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
    : base(options)
    {
    }
}
```

In appsettings.json add the following connection string.

```
appsettings.json   ApplicationDbContext.cs   WeatherForecastController.cs   Program.cs
son.schemastore.org/appsettings.json
{
  "ConnectionStrings": {
    "DefaultConnection": "Server=(localdb)\\mssqllocaldb;Database=Tutorial08"
  },
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "AllowedHosts": "*"
}
```

Finally register ApplicationDbContext for dependency injection in Startup.cs.

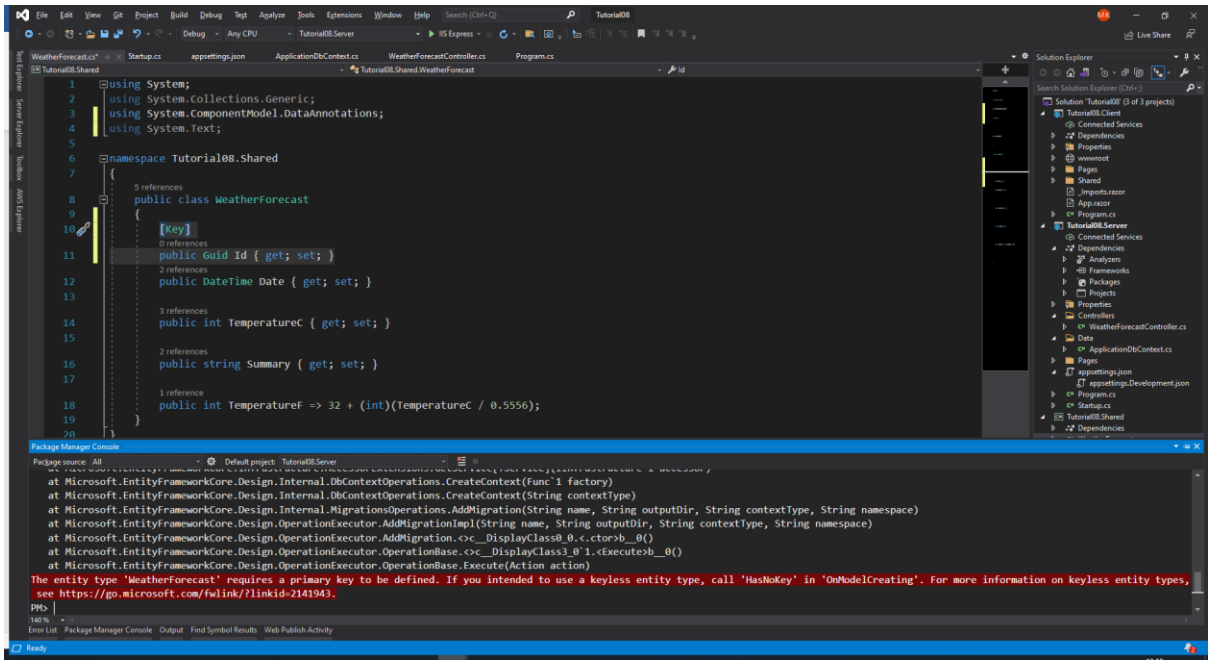
```
2 references
public IConfiguration Configuration { get; }

// This method gets called by the runtime. Use this method to add services to the container.
// For more information on how to configure your application, visit https://go.microsoft.com/fwlink/?LinkID=398940
0 references
public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<ApplicationDbContext>(options =>
        options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection"))
    );
    services.AddControllersWithViews();
    services.AddRazorPages();
}
```

Migrations

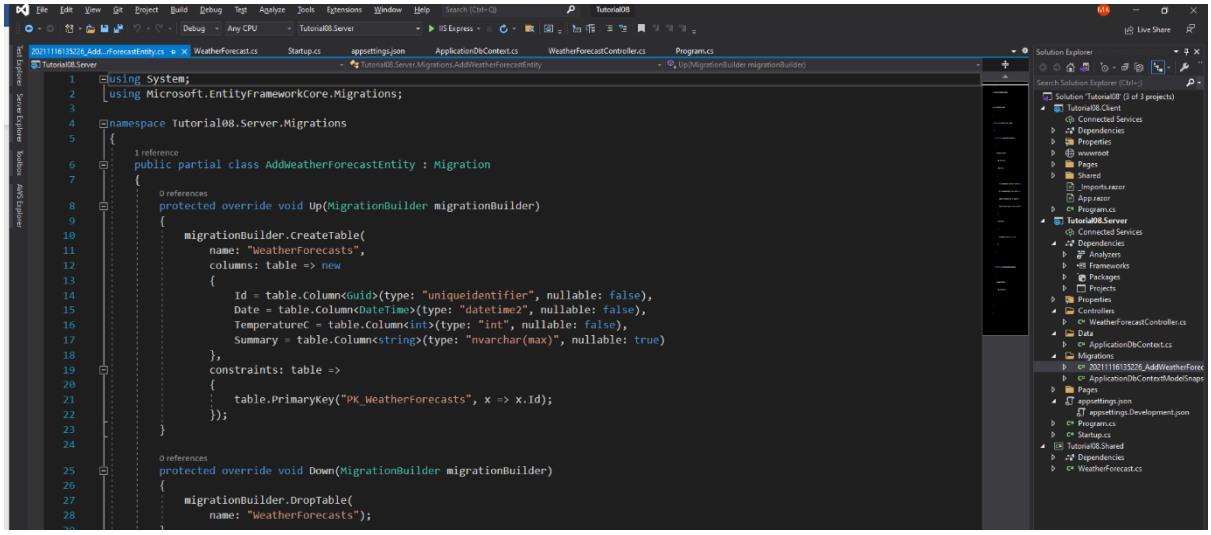
Migrations define which changes will be applied against the database. In this case we are going to add a new table. These are the steps for making a migration:

- Open Package Manager Console (Tools > NuGet Package Manager > Package Manager Console)
- Execute: add-migration AddWeatherForecastEntity
    - New migration for creating a table is generated
- Execute: update-database
    - This applies and updates the database

We should see an error like on the screenshot below. That's because our model does not have a primary key so we need to add a property Id that will act as a primary key.



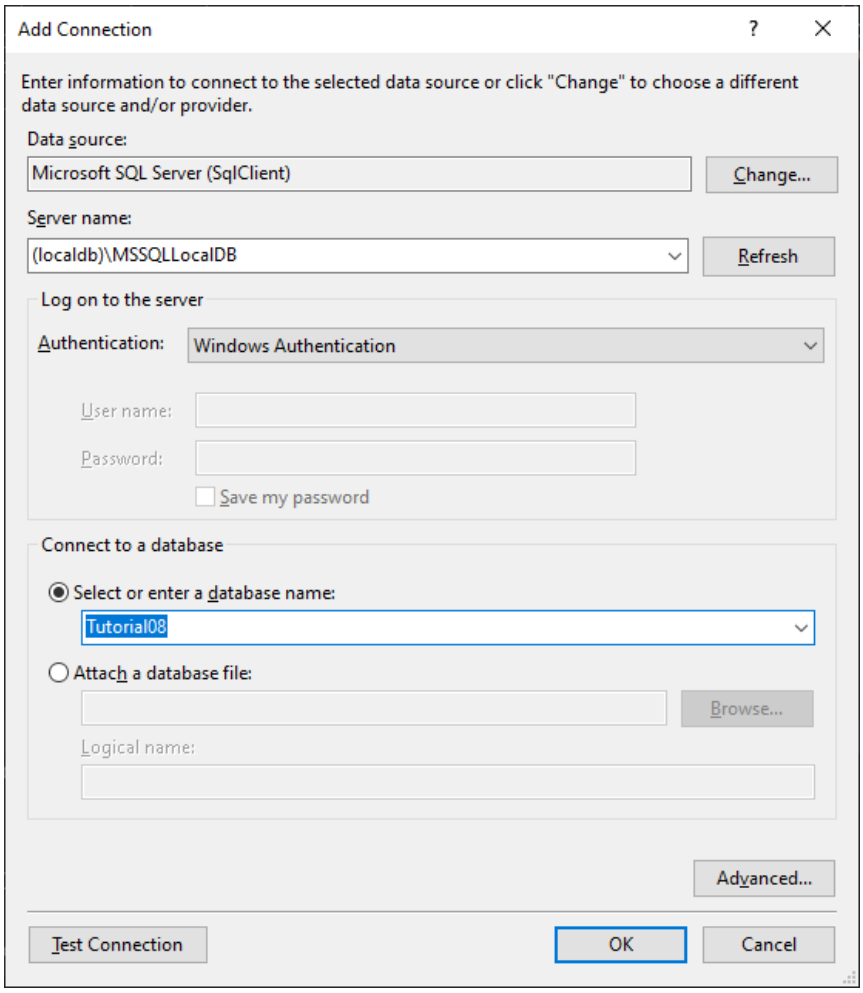
If we add migration again this time we should see a new migration class created. This class contains all the changes that will be applied and we can see that when we execute update-database this will create a new table.

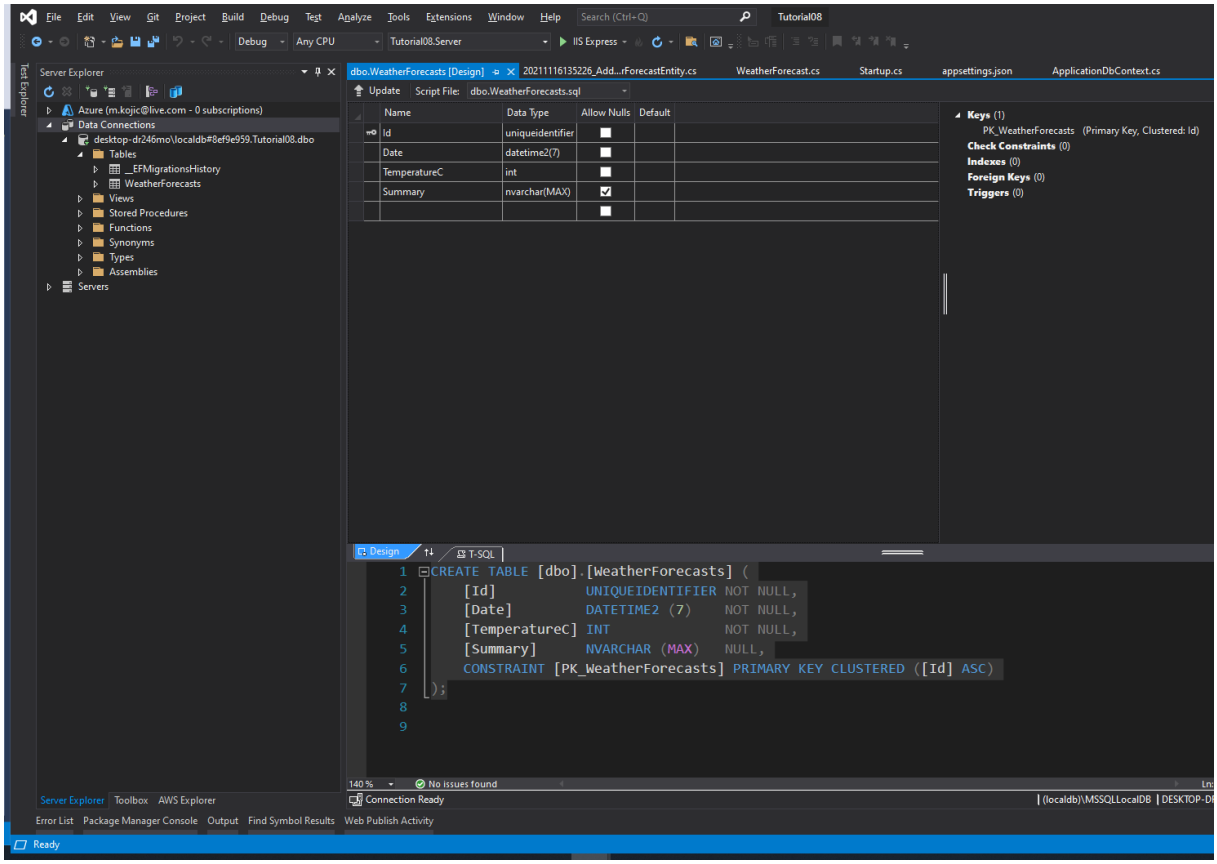


SQL Server Object Explorer

Go to Server Explorer and click on Connect to Database.

Provide the server name and select the database from the list like on the screenshot below.





We can see that migration is applied and table WeatherForecasts is created.

CRUD for database records

We can reuse WeatherForecastController and add some endpoint. In the next tutorial we will learn how we can pass parameters with our requests but for now we just want to have something we can use to test CRUD.

We can use Dependency Injection to provide a dependency to ApplicationDbContext in WeatherForecastController.

```
private readonly ILogger<WeatherForecastController> _logger;
private readonly ApplicationDbContext _applicationDbContext;

0 references
public WeatherForecastController(ILogger<WeatherForecastController> logger, ApplicationDbContext applicationDbContext)
{
    _logger = logger;
    _applicationDbContext = applicationDbContext;
}
```

Update API endpoints

Add endpoints for Create, Edit, Delete and update the one for listing items.

```
stEntity.cs   WeatherForecastController.cs   WeatherForecast.cs   Startup.cs   appsettings.json   ApplicationDbContext.cs   Program.cs
Tutorial08.Server.Controllers.WeatherForecastController   Get()

[HttpGet]
0 references
public IEnumerable<WeatherForecast> Get()
{
    return _applicationDbContext.WeatherForecasts.ToArray();
}

[HttpGet("create")]
0 references
public ActionResult Create()
{
    var rng = new Random();
    _applicationDbContext.WeatherForecasts.Add(
        new WeatherForecast
        {
            Id = Guid.NewGuid(),
            Date = DateTime.Now.AddDays(rng.Next(1,5)),
            TemperatureC = rng.Next(-20, 55),
            Summary = Summaries[rng.Next(Summaries.Length)]
        });

    _applicationDbContext.SaveChanges();
    return Ok();
}

[HttpGet("edit")]
0 references
public ActionResult Edit()
{
    _applicationDbContext.WeatherForecasts.First().Summary = "EDITED!";
    _applicationDbContext.SaveChanges();
    return Ok();
}

[HttpGet("delete")]
0 references
public ActionResult Delete()
{
    _applicationDbContext.WeatherForecasts.Remove(_applicationDbContext.WeatherForecasts.First());
    _applicationDbContext.SaveChanges();
    return Ok();
}
```

Test the CRUD operations by navigating in your browser to:

- https://localhost:<PORT>/WeatherForecast/create
- https://localhost:<PORT>/WeatherForecast/edit
- https://localhost:<PORT>/WeatherForecast/delete
- https://localhost:<PORT>/WeatherForecast/