ASP.NET CORE MVC MODULE 07 ENTITY FRAMEWORK CORE

Summer 2021 – Web Development using ASP .Net Core MVC

MAIN SOURCE + READ THE REST

- I **strongly** encourage you to read the rest of the site below if you intend to use EF Core:
  - https://www.learnentityframeworkcore.com/

Learn Entity Framework Core

Your guide to using the latest version of Microsoft's Object Relational Mapper

- In particular you'll learn about:
  - **Relationships** (one to many, many to many, and one to one)
  - **Conventions** in Entity Framework Core (remember "convention over configuration" …)
  - **Configuration** (to override conventions)
  - **Connection strings**
  - **Concurrency**
  - **Migrations** (make changes to your model and then propagate those changes to your database schema)
  - Executing Raw SQL Queries
  - **Loading Related Data** In Entity Framework Core (example: Lazy Loading)

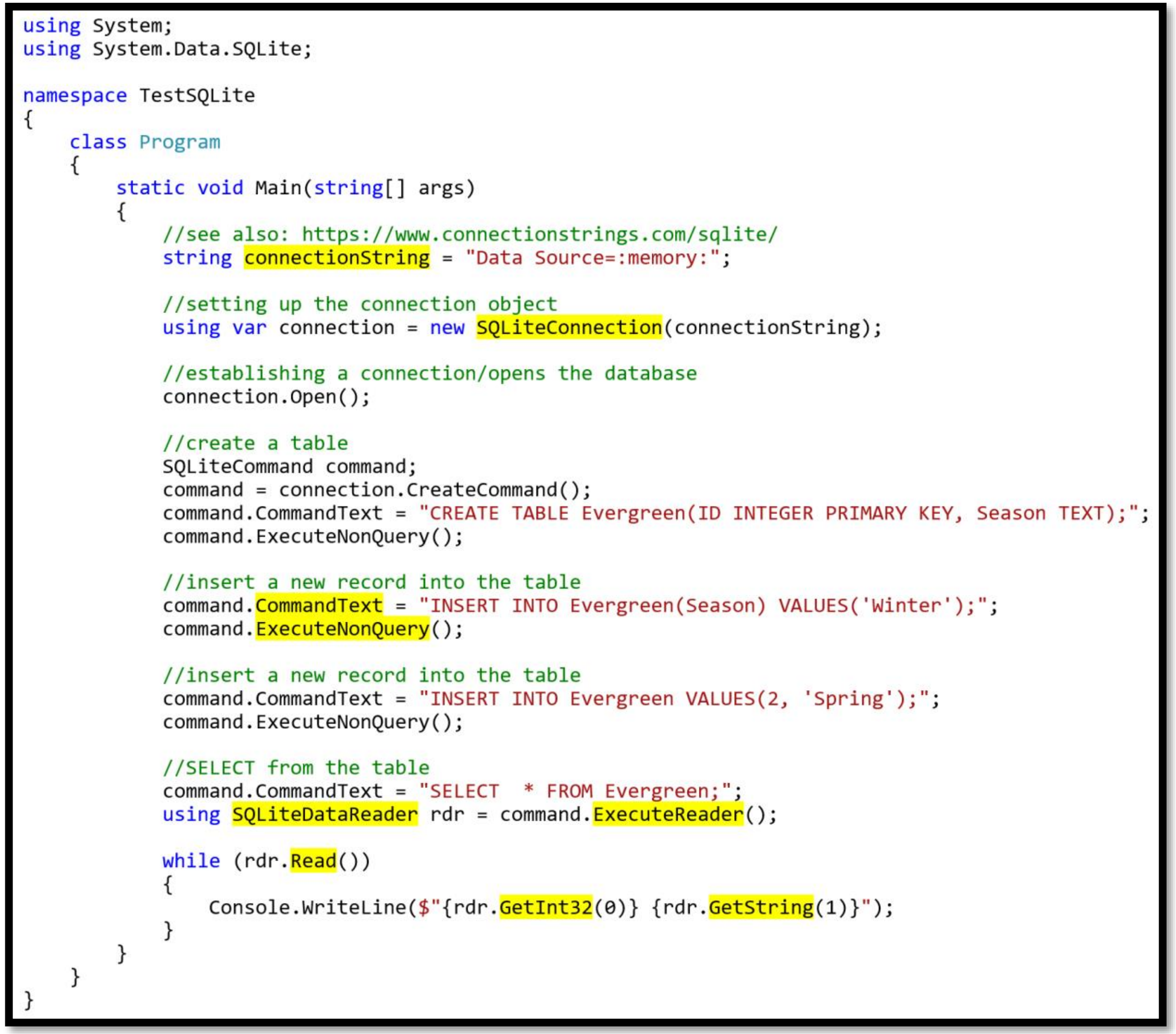
LET'S START – BRIEF ONLY

- There are **multiple ways for working with Databases in C#.**
  - One simple way (but far from ideal) is the following (the steps below are for a SQLite database!):

- You'll need: `using System.Data.Sqlite;`
  - For this, you'll need to use **NuGet** to install it
  - Open Packet Manager Console
  - Run the following command:
    - Install-Package System.Data.SQLite.Core

- You'll need to create a **SQLiteConnection** object
  - Then use **ExecuteNonQuery()** method when you do not expect any data to be returned.
    - For example: create table, insert a new record, …
  - Use the **SQLiteDataReader** class when you need to retrieve data from a database.
    - For example: select

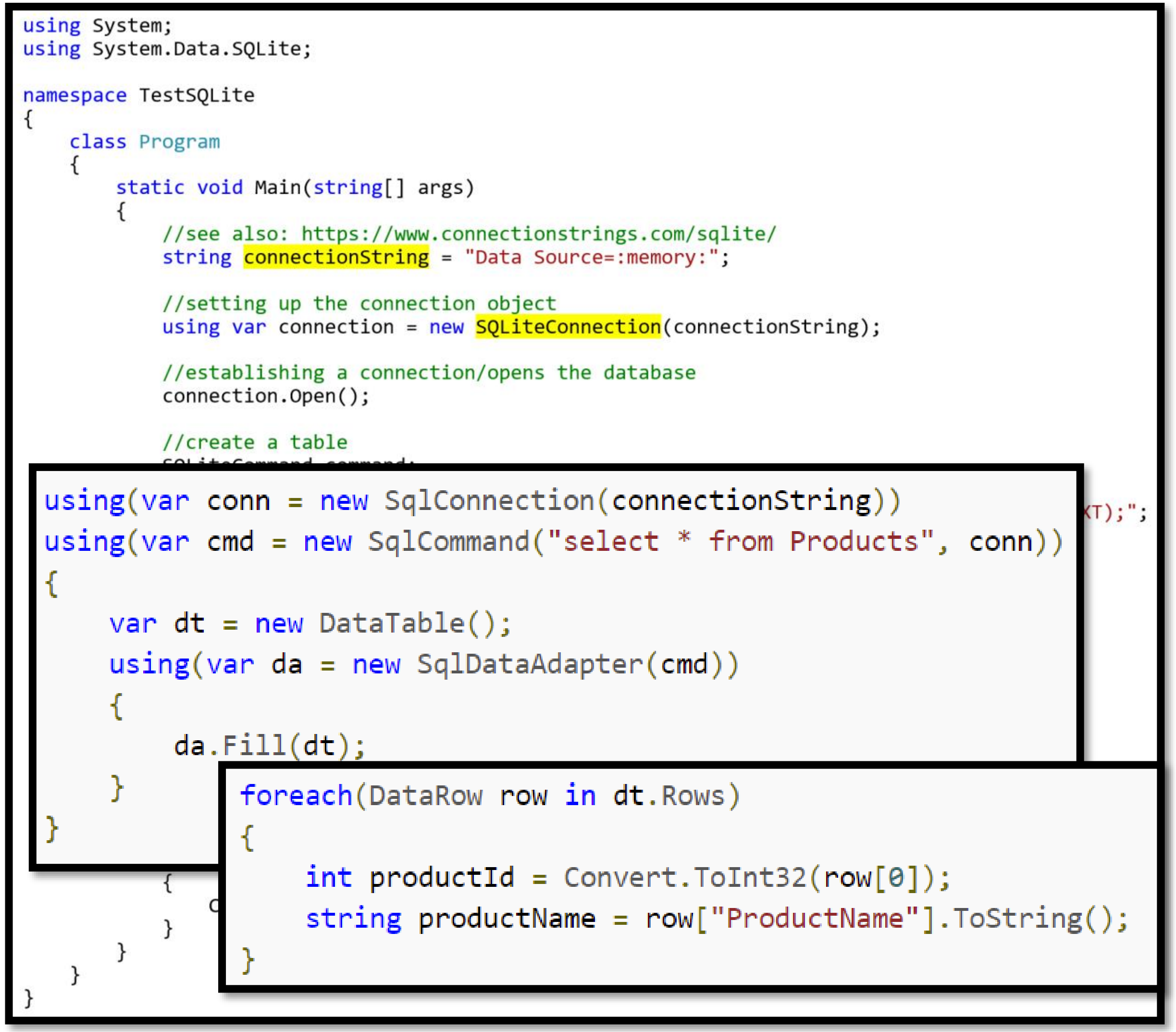
- Connection Strings Reference page:
  - https://www.connectionstrings.com/



```
using System;
using System.Data.SQLite;

namespace TestSQLite
{
    class Program
    {
        static void Main(string[] args)
        {
            //see also: https://www.connectionstrings.com/sqlite/
            string connectionString = "Data Source=:memory:";

            //setting up the connection object
            using var connection = new SQLiteConnection(connectionString);

            //establishing a connection/opens the database
            connection.Open();

            //create a table
            SQLiteCommand command;
            command = connection.CreateCommand();
            command.CommandText = "CREATE TABLE Evergreen(ID INTEGER PRIMARY KEY, Season TEXT);";
            command.ExecuteNonQuery();

            //insert a new record into the table
            command.CommandText = "INSERT INTO Evergreen(Season) VALUES('Winter');";
            command.ExecuteNonQuery();

            //insert a new record into the table
            command.CommandText = "INSERT INTO Evergreen VALUES(2, 'Spring');";
            command.ExecuteNonQuery();

            //SELECT from the table
            command.CommandText = "SELECT  * FROM Evergreen;";
            using SQLiteDataReader rdr = command.ExecuteReader();

            while (rdr.Read())
            {
                Console.WriteLine($"{rdr.GetInt32(0)} {rdr.GetString(1)}");
            }
        }
    }
}
```

LET'S START – BRIEF ONLY

- There are **multiple ways for working with Databases in C#.**
  - One simple way (but far from ideal) is the following (the steps below are for a SQLite database!):



```
using System;
using System.Data.SQLite;

namespace TestSQLite
{
    class Program
    {
        static void Main(string[] args)
        {
            //see also: https://www.connectionstrings.com/sqlite/
            string connectionString = "Data Source=:memory:";

            //setting up the connection object
            using var connection = new SQLiteConnection(connectionString);

            //establishing a connection/opens the database
            connection.Open();

            //create a table
```

```
using(var conn = new SqlConnection(connectionString))
using(var cmd = new SqlCommand("select * from Products", conn))
{
    var dt = new DataTable();
    using(var da = new SqlDataAdapter(cmd))
    {
        da.Fill(dt);
    }
}
```

```
foreach(DataRow row in dt.Rows)
{
    int productId = Convert.ToInt32(row[0]);
    string productName = row["ProductName"].ToString();
}
```

- This approach is called **weakly-typed**
- It's prone to errors.
- What happens if you have wrong SQL syntax?
  - Will your code compile?
- What happens if the conversion fails?
- You may also need data conversion …

- Source:
  - https://www.learnentityframeworkcore.com/

LET'S START – BRIEF ONLY

- There are **multiple ways for working with Databases in C#.**
  - A better way is to use an ORM (Object-Relational Mapper), such as **Entity Framework (EF) Core**:
    - Examples of ORM frameworks: **Entity Framework**, **Hibernate** and **Django**

- This approach is called **strongly-typed**
  - You'll be able to get **IntelliSense support**
  - It allows you to work with data in an **object-oriented** way

- It maps **objects** to **tables**
- It generates SQL and executes it against the database for you
- You won't need to know **SQL** to work with **SQL** from **C#**

```
foreach(var record in dbContext.Albums)
     Console.WriteLine($"{record.AlbumId},
                  {record.Title},
                  {record.ArtistID}");
```

- **Entity Framework Core** is a layer between your **code** and a **database**
  - To connect to various types of databases, it uses various libraries:
    - Microsoft SQL Provider ← to connect to SQL Databases (SQL Server, or Azure SQL Database)
    - SQLite Provider ← to connect to an SQLite database
    - Memory Provider ← mimics a database in memory, great for testing
    - Other providers ← provided by other vendors

- Source: https://www.learnentityframeworkcore.com/

EF CORE – CLASSES WE'LL USE – QUICK OVERVIEW

- Main source: https://www.learnentityframeworkcore.com/

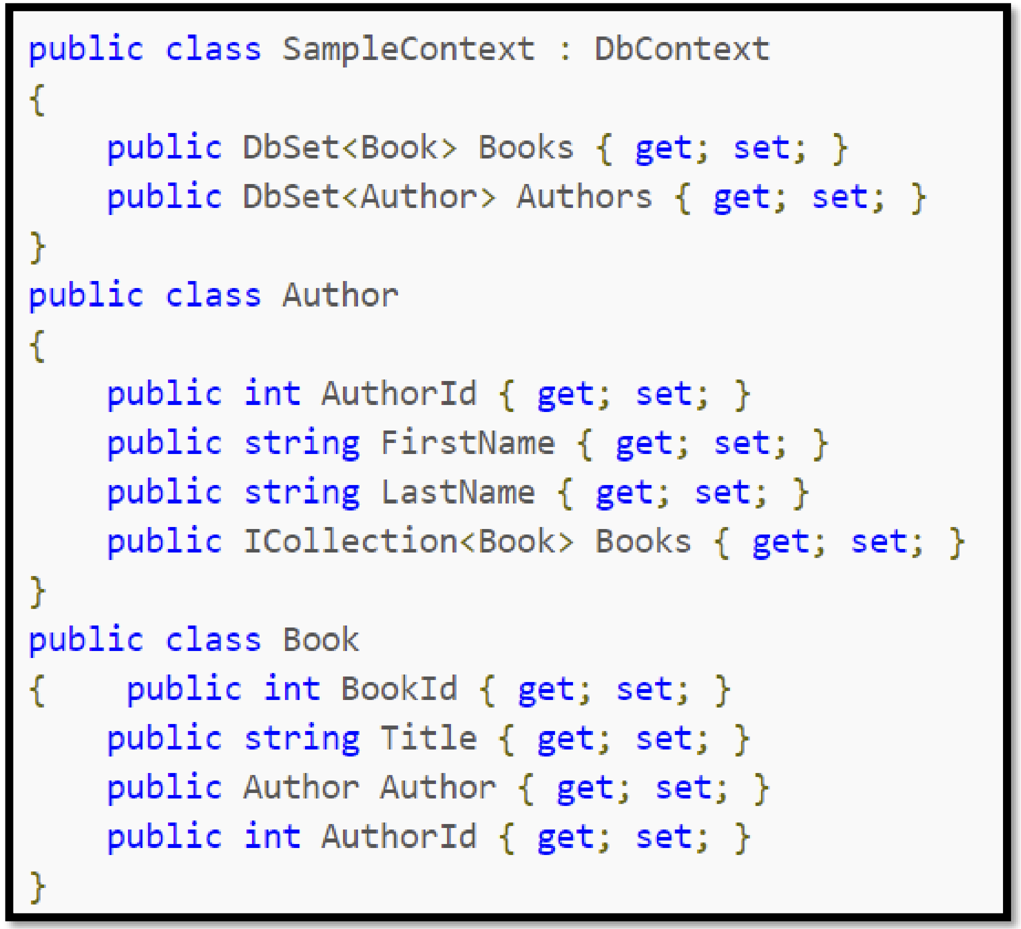
- **DbContext** – a base class responsible with:
  - **Database Connections** (open, close, manage connections to a database)
  - **Data operations** (adding data, modifying data, deleting data, data querying)
  - **Change Tracking** (keeps track of changes you do in your application – so you can save them to the database)
  - **Data Mapping** (maps **properties** from **entities** to **columns** in **tables**)
  - **Transaction management** (when **SaveChanges** is called, a transaction is created for all pending changes. If an error occurs when the changes are applied to the database, they are all rolled back)
  - …

- **DbSet**<TEntity> - a class that represents a collection for a given entity ("think of it as your table")
  - It is the gateway to database operations against an entity.
  - DbSet<TEntity> classes are added as **properties** to the **DbContext**
  - DbSet<TEntity> classes are mapped by default to database tables that have the name of the DbSet<TEntity> property.

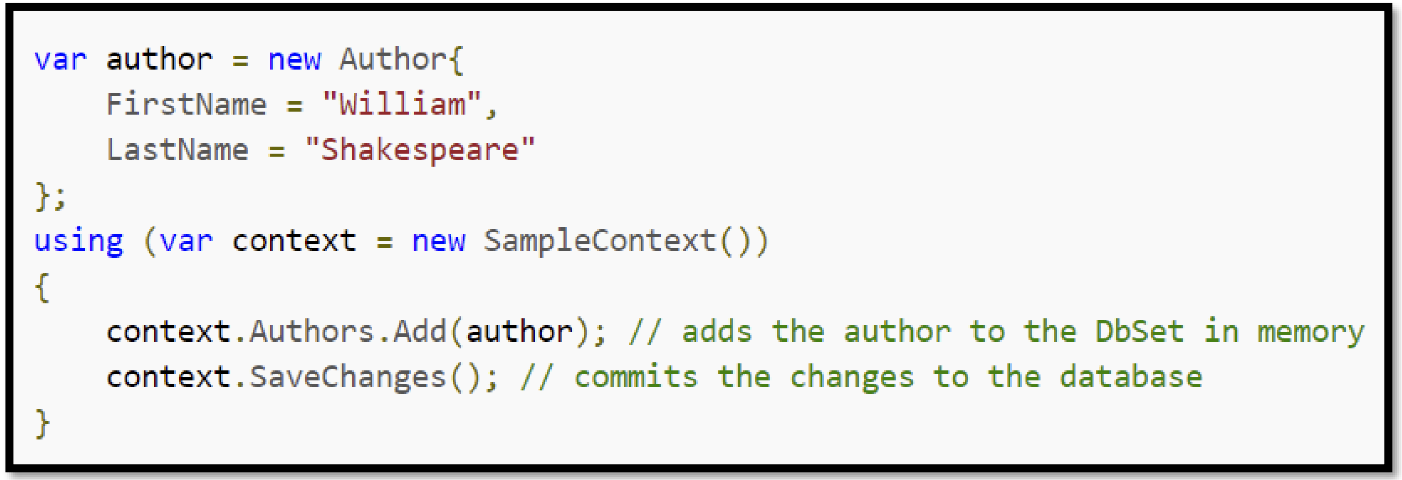
EF CORE – QUICK EXAMPLE – DETAILS BELOW

- Also check out:
  - https://www.learnentityframeworkcore.com/
  - https://www.learnentityframeworkcore.com/dbset

- A quick example to see how **DbContext** and **DbSet** relate:



```
public class SampleContext : DbContext
{
    public DbSet<Book> Books { get; set; }
    public DbSet<Author> Authors { get; set; }
}
public class Author
{
    public int AuthorId { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public ICollection<Book> Books { get; set; }
}
public class Book
{    public int BookId { get; set; }
    public string Title { get; set; }
    public Author Author { get; set; }
    public int AuthorId { get; set; }
}
```

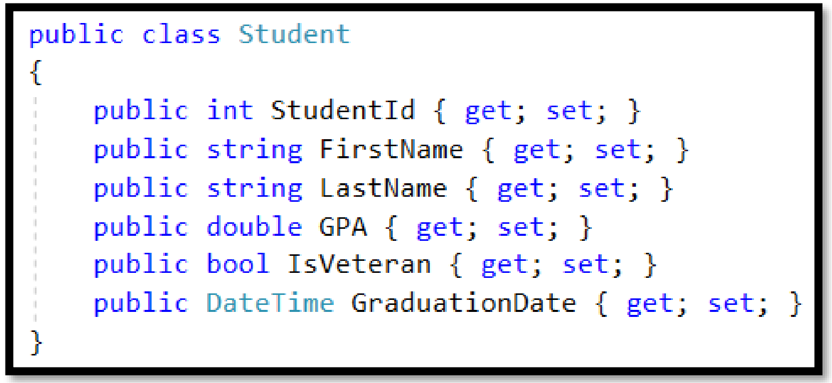


```
var author = new Author{
    FirstName = "William",
    LastName = "Shakespeare"
};
using (var context = new SampleContext())
{
    context.Authors.Add(author); // adds the author to the DbSet in memory
    context.SaveChanges(); // commits the changes to the database
}
```

LET'S DIVE IN … AN EXAMPLE (NON-MVC)

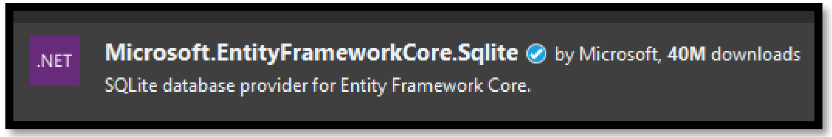
- Create a new **Console Application** (.Net core!)
  - We'll work with a local database (let's say **myDatabase**.db)

- First, I would like to create an **Entity** class, say Student
  - If time, we can add more Entities …



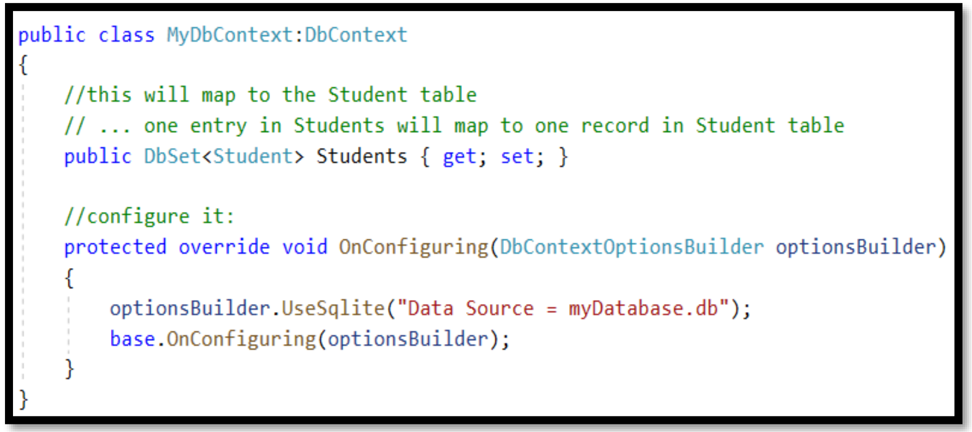
```
public class Student
{
    public int StudentId { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public double GPA { get; set; }
    public bool IsVeteran { get; set; }
    public DateTime GraduationDate { get; set; }
}
```

- Add the **Microsoft.EntityFrameworkCore.Sqlite** NuGet package
  - Go to: Tools > NuGet Package Manager > Manage NugGet Packages …



.NET **Microsoft.EntityFrameworkCore.Sqlite** ✓ by Microsoft, **40M** downloads
SQLite database provider for Entity Framework Core.

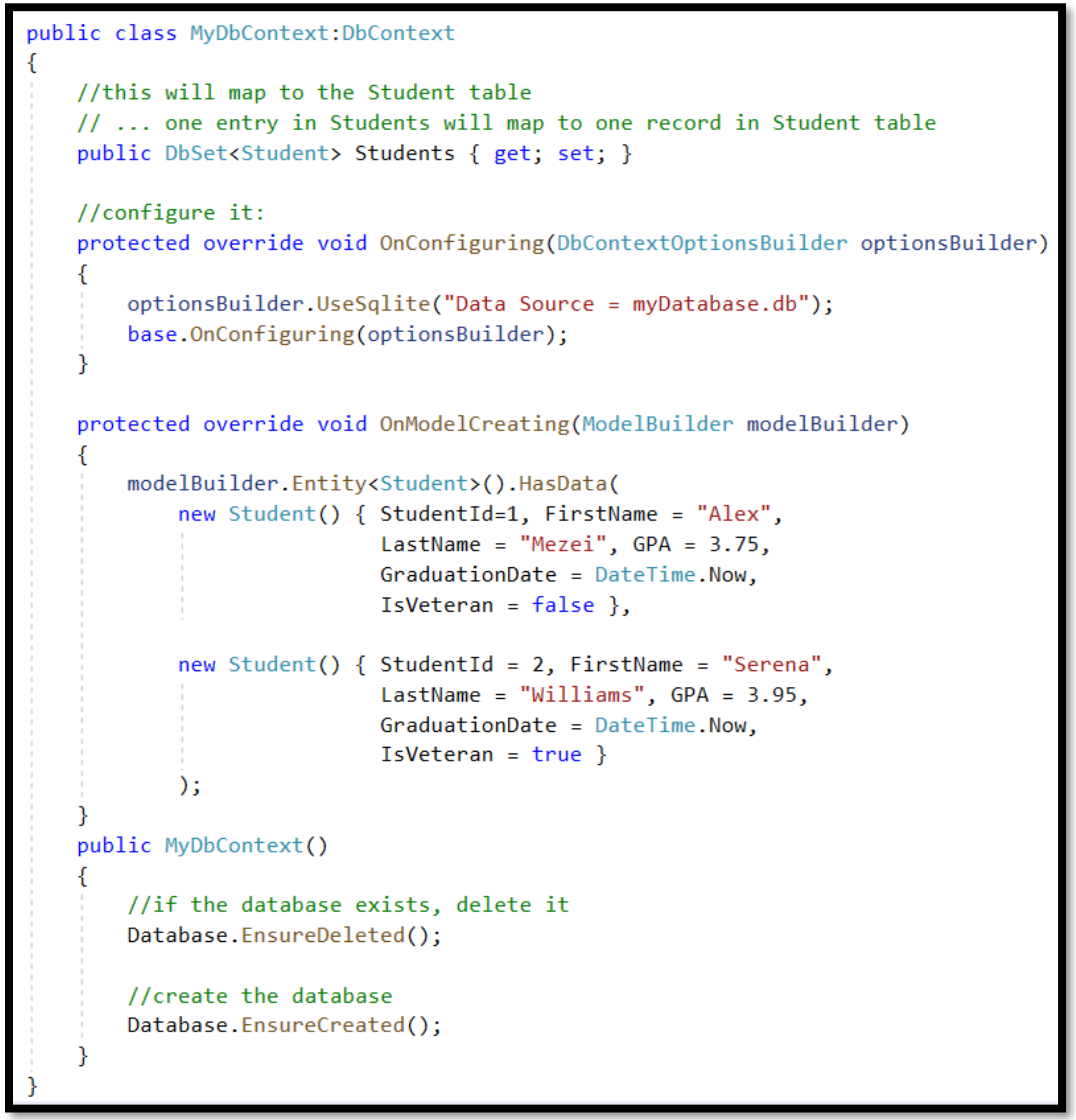
- Create your derived **DbContext** class:
  - It will abstract for you the work with the SQL database …
  - Include: `using Microsoft.EntityFrameworkCore;`
  - Use **DbSet** to map a tables (or more!) to an entities (to a C# class)
  - Then configure it to work with your Database



```
public class MyDbContext:DbContext
{
    //this will map to the Student table
    // ... one entry in Students will map to one record in Student table
    public DbSet<Student> Students { get; set; }

    //configure it:
    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        optionsBuilder.UseSqlite("Data Source = myDatabase.db");
        base.OnConfiguring(optionsBuilder);
    }
}
```

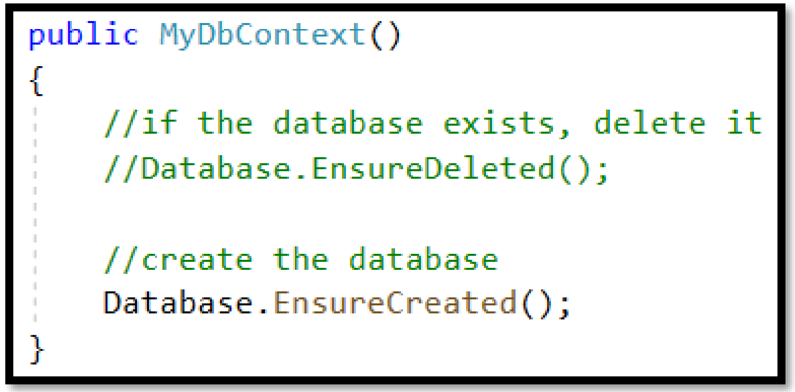
LET'S DIVE IN … AN EXAMPLE (NON-MVC)

- First, let's add some data (and make sure we recreate the same database each time)



```
public class MyDbContext:DbContext
{
    //this will map to the Student table
    // ... one entry in Students will map to one record in Student table
    public DbSet<Student> Students { get; set; }

    //configure it:
    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        optionsBuilder.UseSqlite("Data Source = myDatabase.db");
        base.OnConfiguring(optionsBuilder);
    }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Student>().HasData(
            new Student() { StudentId=1, FirstName = "Alex",
                            LastName = "Mezei", GPA = 3.75,
                            GraduationDate = DateTime.Now,
                            IsVeteran = false },

            new Student() { StudentId = 2, FirstName = "Serena",
                            LastName = "Williams", GPA = 3.95,
                            GraduationDate = DateTime.Now,
                            IsVeteran = true }
            );
    }
    public MyDbContext()
    {
        //if the database exists, delete it
        Database.EnsureDeleted();

        //create the database
        Database.EnsureCreated();
    }
}
```

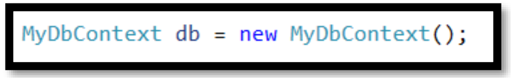
- Better yet, use (what's the difference?):



```
public MyDbContext()
{
    //if the database exists, delete it
    //Database.EnsureDeleted();

    //create the database
    Database.EnsureCreated();
}
```

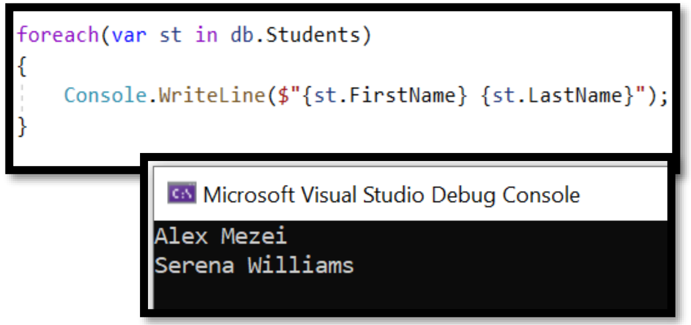
LET'S DIVE IN … AN EXAMPLE (NON-MVC)

- In Main, create an instance of your derived **DbContext** class.



```
MyDbContext db = new MyDbContext();
```

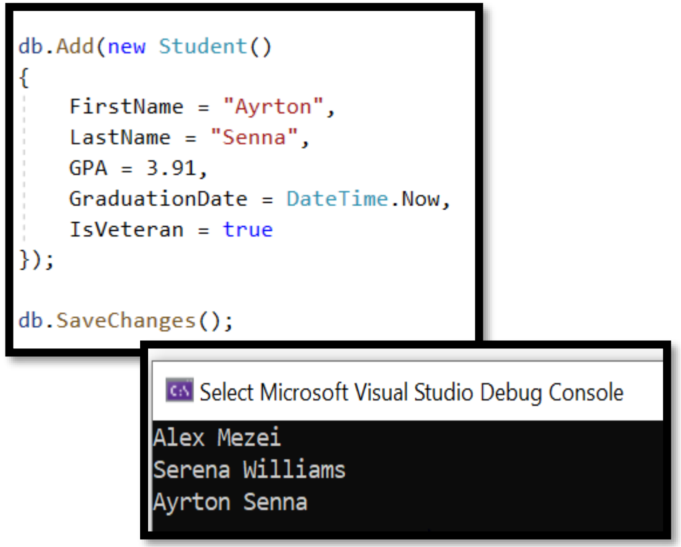
- Then use it query/get all the first and last names from the Students table.



```
foreach(var st in db.Students)
{
    Console.WriteLine($"{st.FirstName} {st.LastName}");
}
```

```
Alex Mezei
Serena Williams
```

- How would you add a new entry to the database?
  - Important: until you call the SaveChanges method, your changes will not be saved into the database!

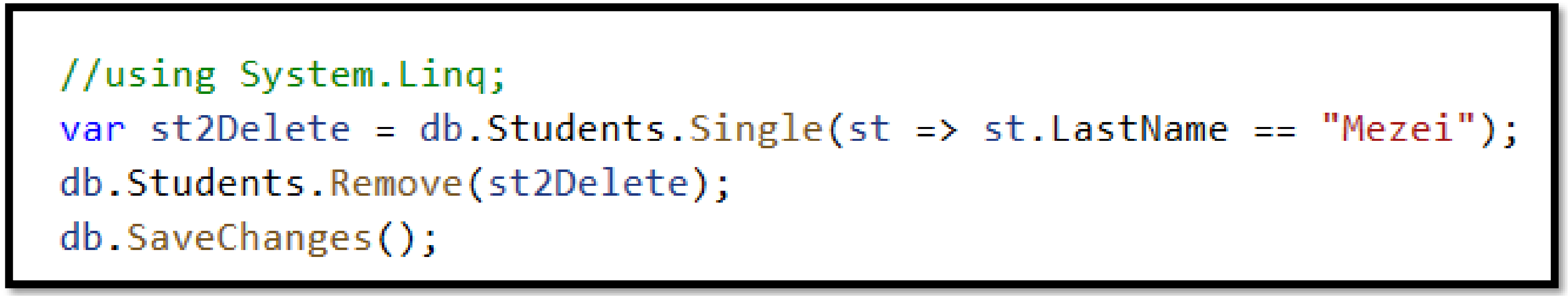


```
db.Add(new Student()
{
    FirstName = "Ayrton",
    LastName = "Senna",
    GPA = 3.91,
    GraduationDate = DateTime.Now,
    IsVeteran = true
});

db.SaveChanges();
```

```
Alex Mezei
Serena Williams
Ayrton Senna
```

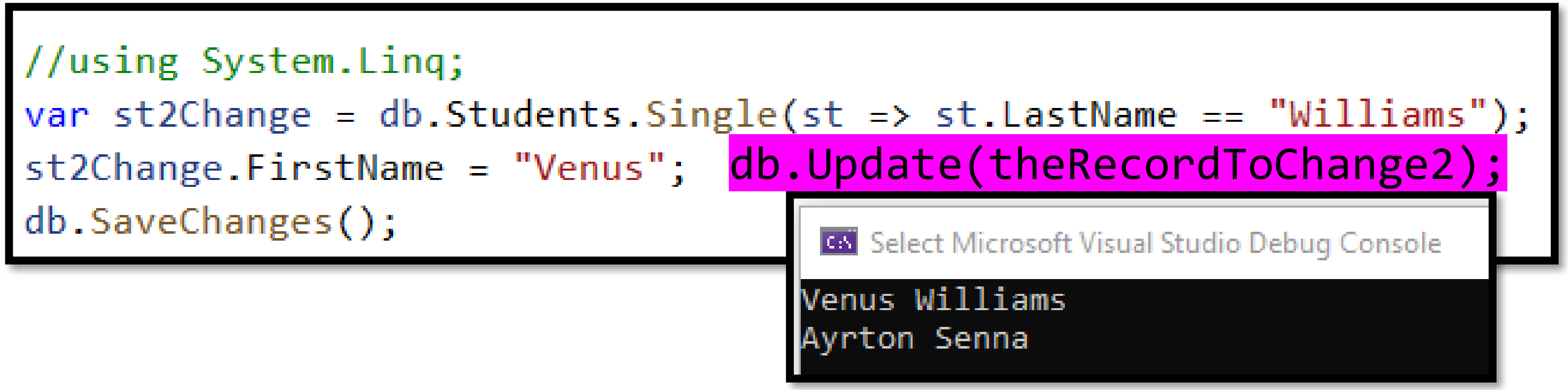
LET'S DIVE IN … AN EXAMPLE (NON-MVC)

- Let's **delete** a record



```
//using System.Linq;
var st2Delete = db.Students.Single(st => st.LastName == "Mezei");
db.Students.Remove(st2Delete);
db.SaveChanges();
```

- Let's **change** an existing record



```
//using System.Linq;
var st2Change = db.Students.Single(st => st.LastName == "Williams");
st2Change.FirstName = "Venus";  db.Update(theRecordToChange2);
db.SaveChanges();
```

Select Microsoft Visual Studio Debug Console

```
Venus Williams
Ayrton Senna
```

SOME CONVENTIONS IN ENTITY FRAMEWORK CORE

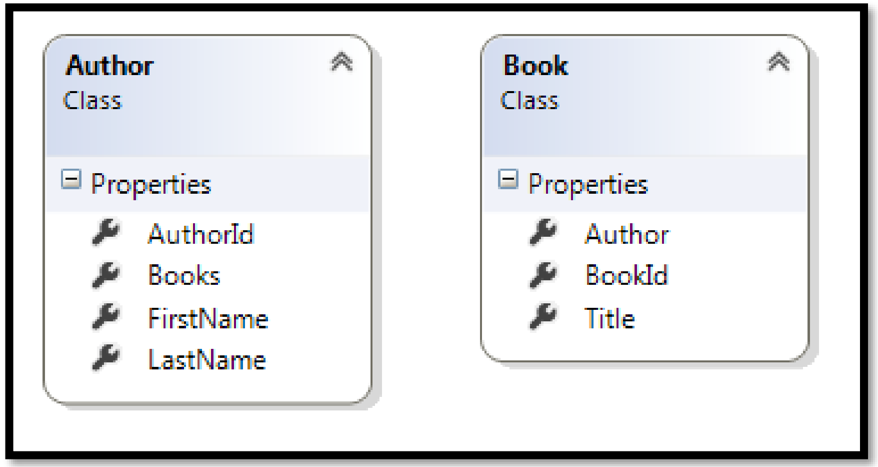
- Sources (images are from these sources!):
  - https://www.learnentityframeworkcore.com/conventions
  - https://www.learnentityframeworkcore.com/model

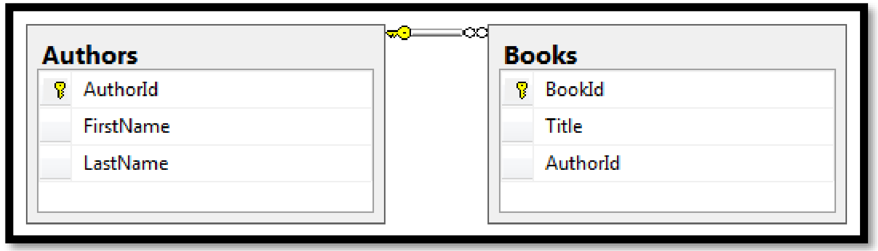
- Here you have an example with two models (mapped to two **related** tables)

- Primary key:
  - "If a property is named **ID** or <entity name>ID (not case-sensitive), it will be configured as the **primary key**."
  - Note: AuthorID and BookId
  - Alternative names for each table's primary key: Id

- Foreign Key:
  - "The convention for a foreign key is that it must have the same data type as the principal entity's primary key property and the name must follow one of these patterns:
    - <navigation property name><principal primary key property name>Id
    - <principal class name><primary key property name>Id
    - <principal primary key property name>Id"
  - Alternative names for AuthorId from Books table: AuthorAuthorId, …

- "EF Core will map entity **properties** to database **columns** with the same name."





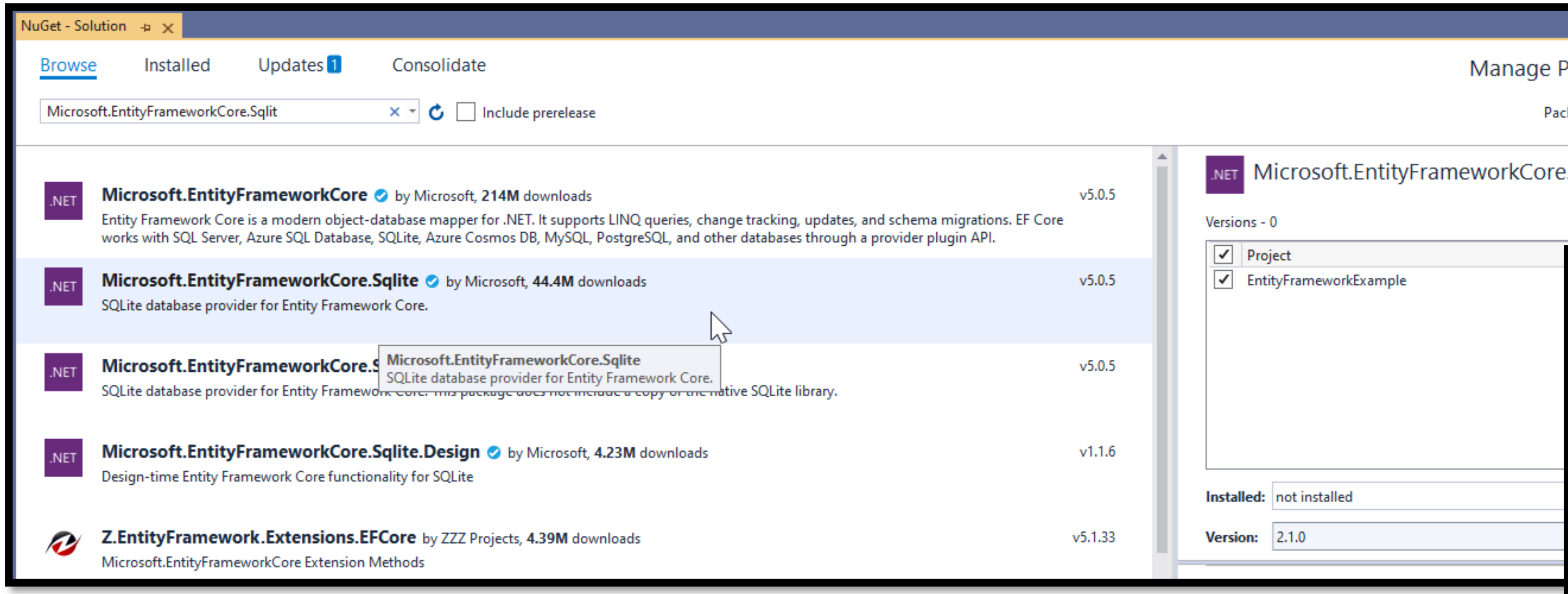
IN-CLASS DEMO

LET'S DIVE IN AGAIN … AN MVC EXAMPLE

Demonstration: How to Use Entity Framework Core

- Source/Steps
- https://github.com/MicrosoftLearning/20486D-DevelopingASPNETMVCWebApplications/blob/master/Instructions/20486D_MOD07_DEMO.md#lesson-2-working-with-entity-framework-core

SCREENSHOTS …WHAT WE'LL DO



Microsoft.EntityFrameworkCore.Sqlit

☐ Include prerelease

Package source: nuget.org

Microsoft.EntityFrameworkCore by Microsoft, **214M** downloads v5.0.5
Entity Framework Core is a modern object-database mapper for .NET. It supports LINQ queries, change tracking, updates, and schema migrations. EF Core works with SQL Server, Azure SQL Database, SQLite, Azure Cosmos DB, MySQL, PostgreSQL, and other databases through a provider plugin API.

Microsoft.EntityFrameworkCore.Sqlite by Microsoft, **44.4M** downloads v5.0.5
SQLite database provider for Entity Framework Core.

Microsoft.EntityFrameworkCore.S[...] v5.0.5
SQLite database provider for Entity Framework Core. This package does not include a copy of the native SQLite library.

Microsoft.EntityFrameworkCore.Sqlite
SQLite database provider for Entity Framework Core.

Microsoft.EntityFrameworkCore.Sqlite.Design by Microsoft, **4.23M** downloads v1.1.6
Design-time Entity Framework Core functionality for SQLite

Z.EntityFramework.Extensions.EFCore by ZZZ Projects, **4.39M** downloads v5.1.33
Microsoft.EntityFrameworkCore Extension Methods

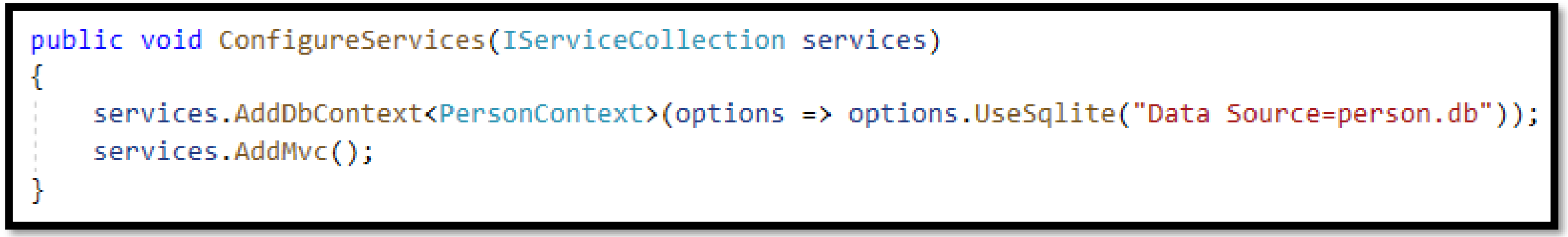
Microsoft.EntityFrameworkCore.Sqlite nuget.org

Versions - 0

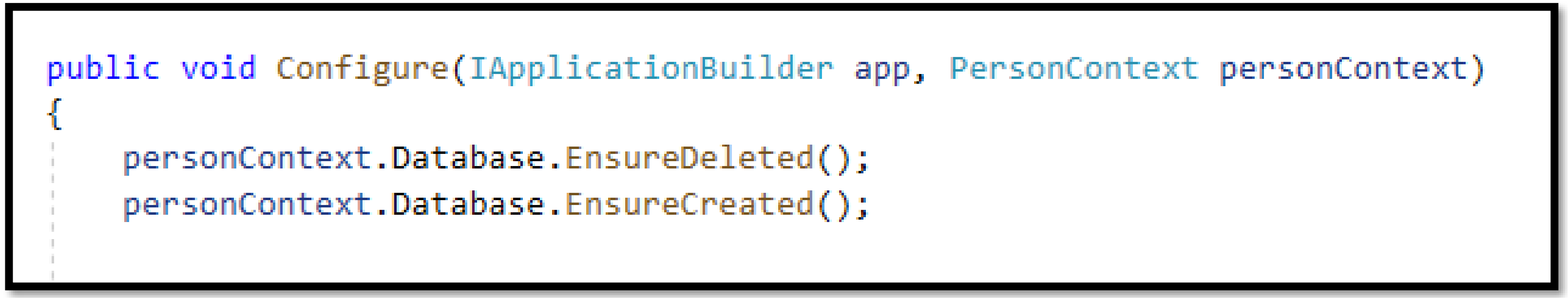
☑ Project
☑ EntityFrameworkExample

Installed: not installed

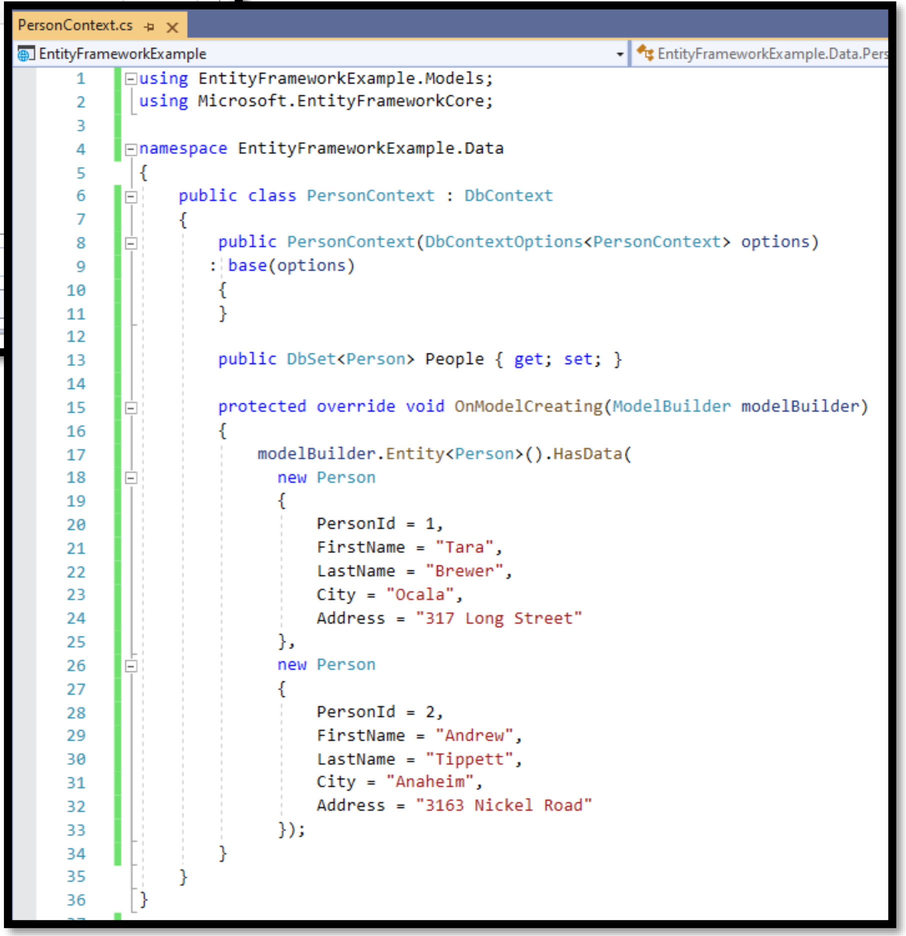
Version: 2.1.0



```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<PersonContext>(options => options.UseSqlite("Data Source=person.db"));
    services.AddMvc();
}
```

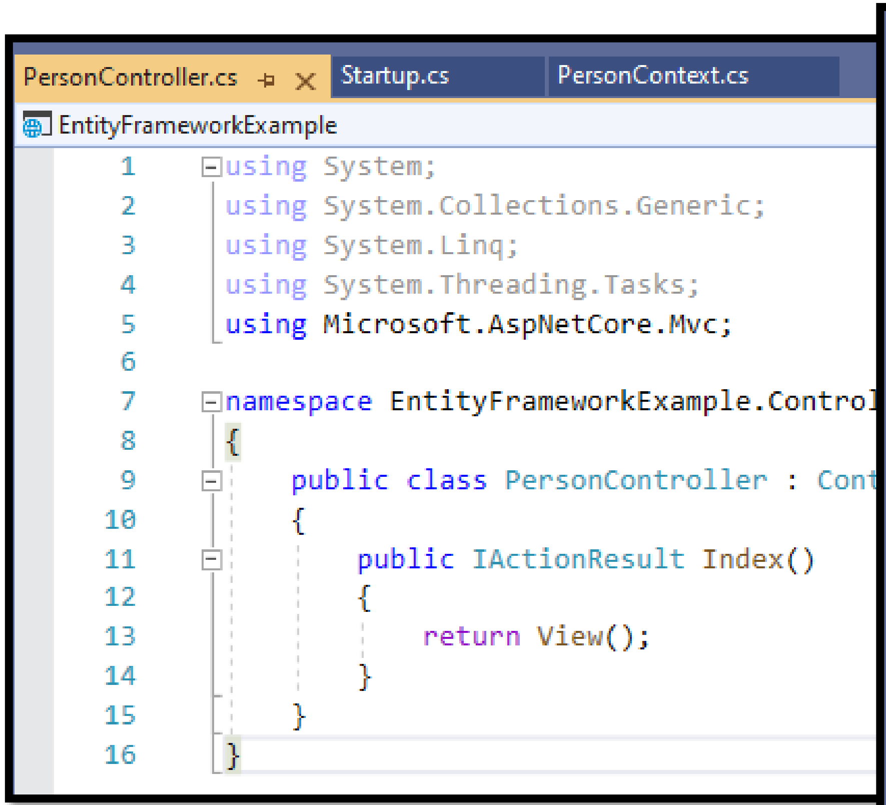


```
public void Configure(IApplicationBuilder app, PersonContext personContext)
{
    personContext.Database.EnsureDeleted();
    personContext.Database.EnsureCreated();
```

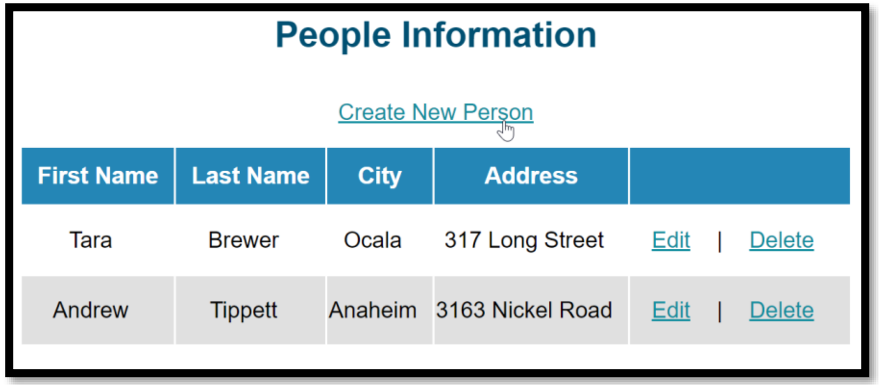


```
using EntityFrameworkExample.Models;
using Microsoft.EntityFrameworkCore;

namespace EntityFrameworkExample.Data
{
    public class PersonContext : DbContext
    {
        public PersonContext(DbContextOptions<PersonContext> options)
            : base(options)
        {
        }

        public DbSet<Person> People { get; set; }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder.Entity<Person>().HasData(
                new Person
                {
                    PersonId = 1,
                    FirstName = "Tara",
                    LastName = "Brewer",
                    City = "Ocala",
                    Address = "317 Long Street"
                },
                new Person
                {
                    PersonId = 2,
                    FirstName = "Andrew",
                    LastName = "Tippett",
                    City = "Anaheim",
                    Address = "3163 Nickel Road"
                });
        }
    }
}
```

SCREENSHOTS …WHAT WE'LL DO



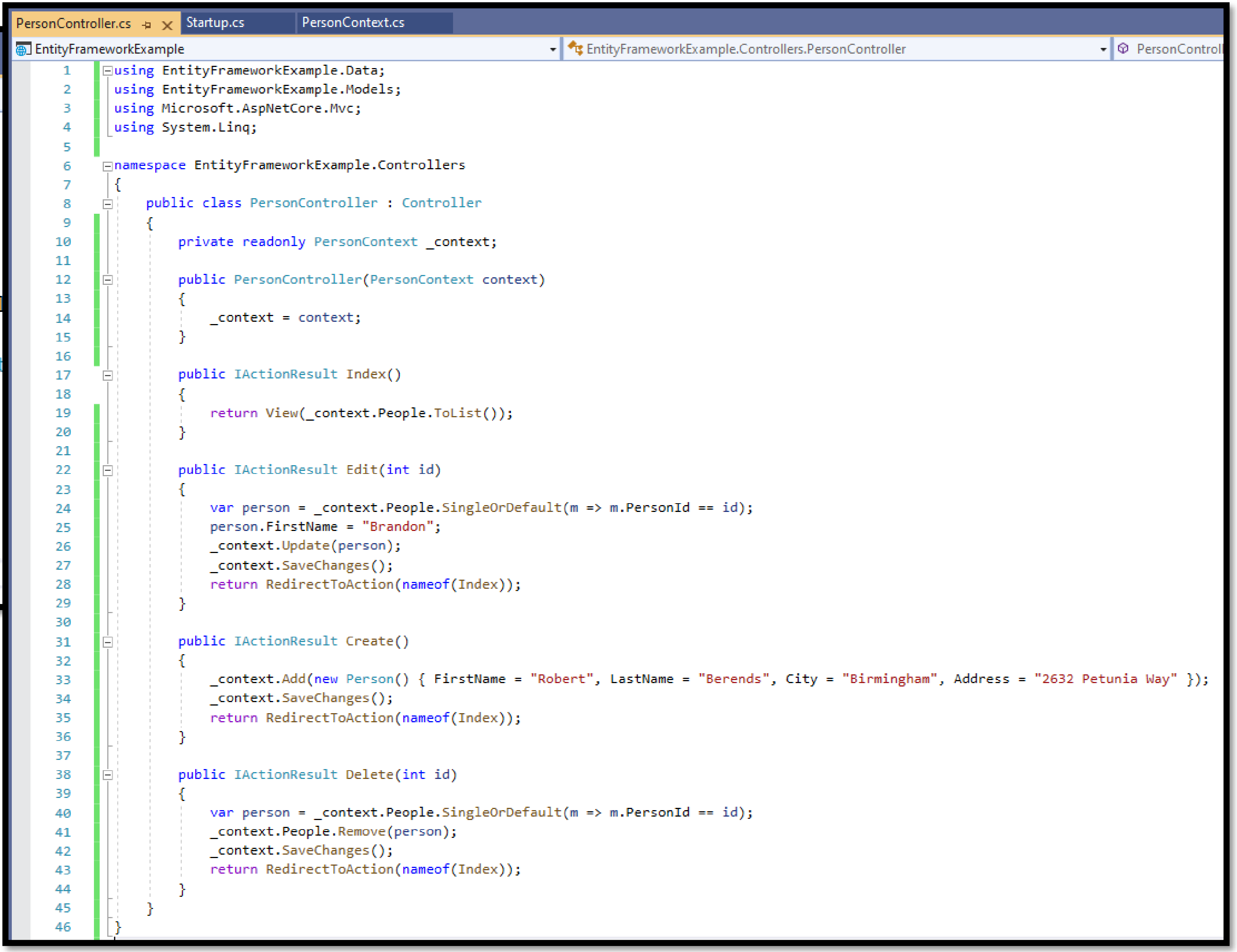
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;

namespace EntityFrameworkExample.Control
{
    public class PersonController : Cont
    {
        public IActionResult Index()
        {
            return View();
        }
    }
}
```



People Information

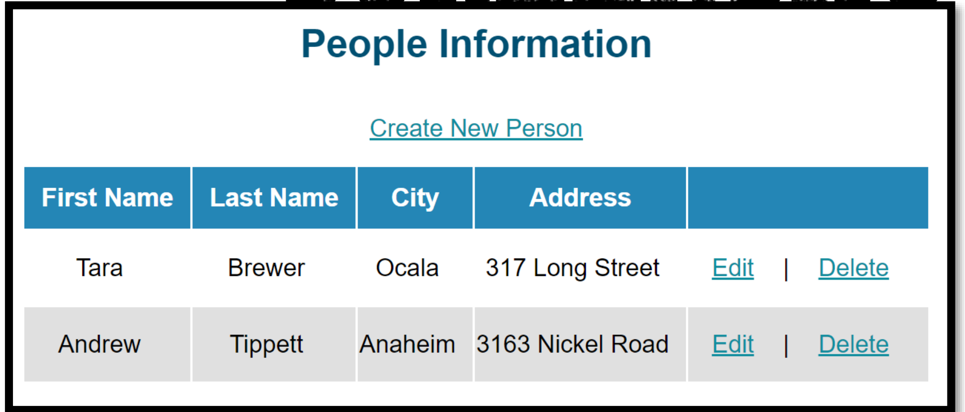
Create New Person

First Name	Last Name	City	Address	
Tara	Brewer	Ocala	317 Long Street	Edit | Delete
Andrew	Tippett	Anaheim	3163 Nickel Road	Edit | Delete



```
using EntityFrameworkExample.Data;
using EntityFrameworkExample.Models;
using Microsoft.AspNetCore.Mvc;
using System.Linq;

namespace EntityFrameworkExample.Controllers
{
    public class PersonController : Controller
    {
        private readonly PersonContext _context;

        public PersonController(PersonContext context)
        {
            _context = context;
        }

        public IActionResult Index()
        {
            return View(_context.People.ToList());
        }

        public IActionResult Edit(int id)
        {
            var person = _context.People.SingleOrDefault(m => m.PersonId == id);
            person.FirstName = "Brandon";
            _context.Update(person);
            _context.SaveChanges();
            return RedirectToAction(nameof(Index));
        }

        public IActionResult Create()
        {
            _context.Add(new Person() { FirstName = "Robert", LastName = "Berends", City = "Birmingham", Address = "2632 Petunia Way" });
            _context.SaveChanges();
            return RedirectToAction(nameof(Index));
        }

        public IActionResult Delete(int id)
        {
            var person = _context.People.SingleOrDefault(m => m.PersonId == id);
            _context.People.Remove(person);
            _context.SaveChanges();
            return RedirectToAction(nameof(Index));
        }
    }
}
```

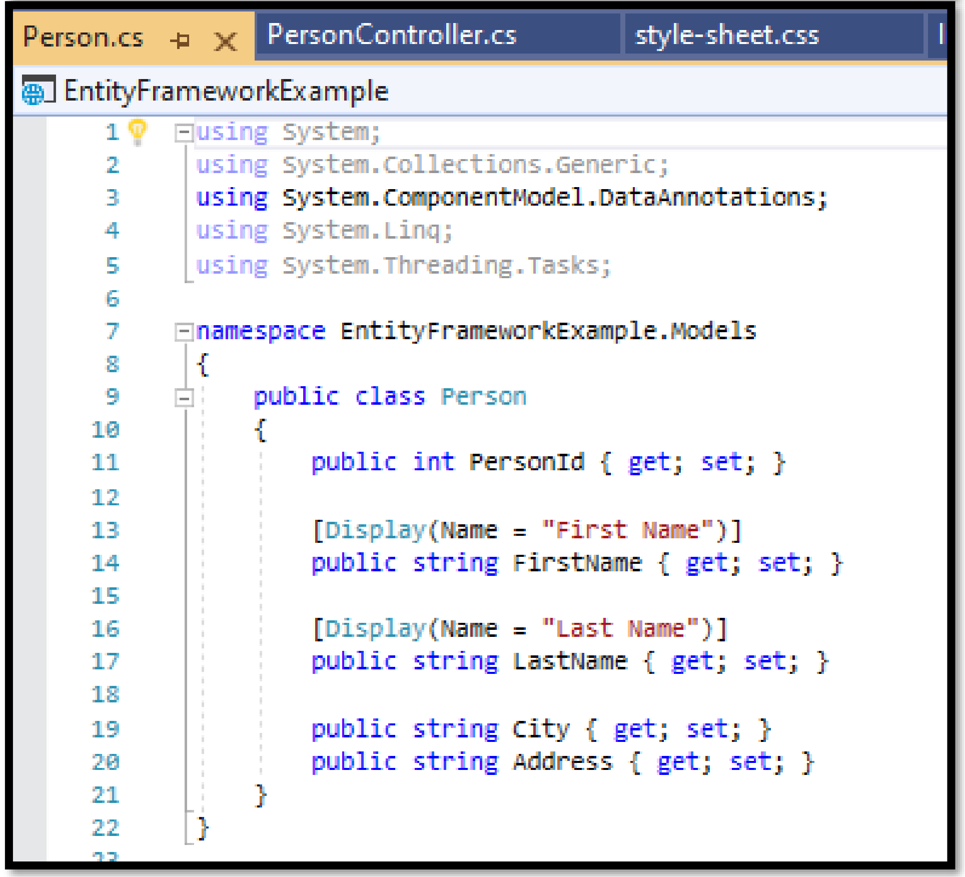
SCREENSHOTS … WHAT WE'LL DO



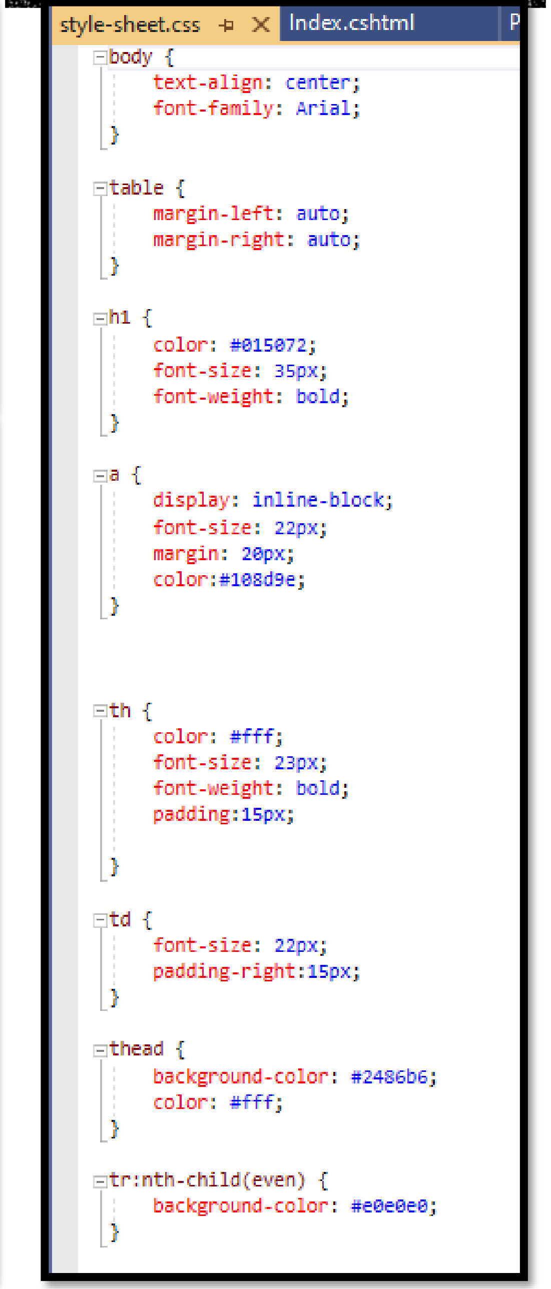
People Information

Create New Person

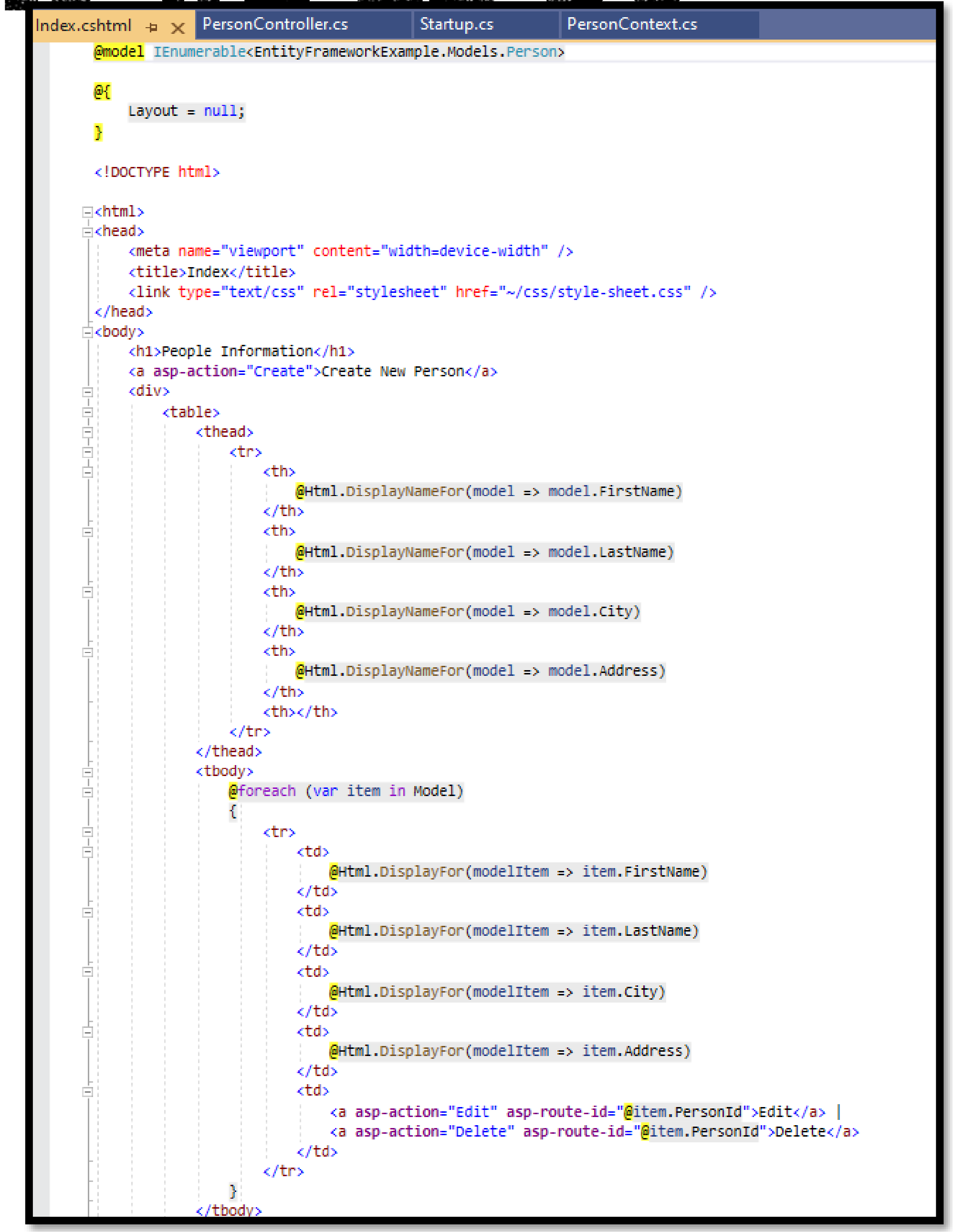
First Name	Last Name	City	Address	
Tara	Brewer	Ocala	317 Long Street	Edit | Delete
Andrew	Tippett	Anaheim	3163 Nickel Road	Edit | Delete



```csharp
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Threading.Tasks;

namespace EntityFrameworkExample.Models
{
    public class Person
    {
        public int PersonId { get; set; }

        [Display(Name = "First Name")]
        public string FirstName { get; set; }

        [Display(Name = "Last Name")]
        public string LastName { get; set; }

        public string City { get; set; }
        public string Address { get; set; }
    }
}
```



```css
body {
    text-align: center;
    font-family: Arial;
}

table {
    margin-left: auto;
    margin-right: auto;
}

h1 {
    color: #015072;
    font-size: 35px;
    font-weight: bold;
}

a {
    display: inline-block;
    font-size: 22px;
    margin: 20px;
    color:#108d9e;
}



th {
    color: #fff;
    font-size: 23px;
    font-weight: bold;
    padding:15px;

}

td {
    font-size: 22px;
    padding-right:15px;
}

thead {
    background-color: #2486b6;
    color: #fff;
}

tr:nth-child(even) {
    background-color: #e0e0e0;
}
```



```html
@model IEnumerable<EntityFrameworkExample.Models.Person>

@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Index</title>
    <link type="text/css" rel="stylesheet" href="~/css/style-sheet.css" />
</head>
<body>
    <h1>People Information</h1>
    <a asp-action="Create">Create New Person</a>
    <div>
        <table>
            <thead>
                <tr>
                    <th>
                        @Html.DisplayNameFor(model => model.FirstName)
                    </th>
                    <th>
                        @Html.DisplayNameFor(model => model.LastName)
                    </th>
                    <th>
                        @Html.DisplayNameFor(model => model.City)
                    </th>
                    <th>
                        @Html.DisplayNameFor(model => model.Address)
                    </th>
                    <th></th>
                </tr>
            </thead>
            <tbody>
                @foreach (var item in Model)
                {
                    <tr>
                        <td>
                            @Html.DisplayFor(modelItem => item.FirstName)
                        </td>
                        <td>
                            @Html.DisplayFor(modelItem => item.LastName)
                        </td>
                        <td>
                            @Html.DisplayFor(modelItem => item.City)
                        </td>
                        <td>
                            @Html.DisplayFor(modelItem => item.Address)
                        </td>
                        <td>
                            <a asp-action="Edit" asp-route-id="@item.PersonId">Edit</a> |
                            <a asp-action="Delete" asp-route-id="@item.PersonId">Delete</a>
                        </td>
                    </tr>
                }
            </tbody>
```

HOW DO YOU USE ENTITY FRAMEWORK IN AN MVC APPLICATION?

- You’ll need to create **entity classes** (**model** classes that will be mapped to tables in the database)

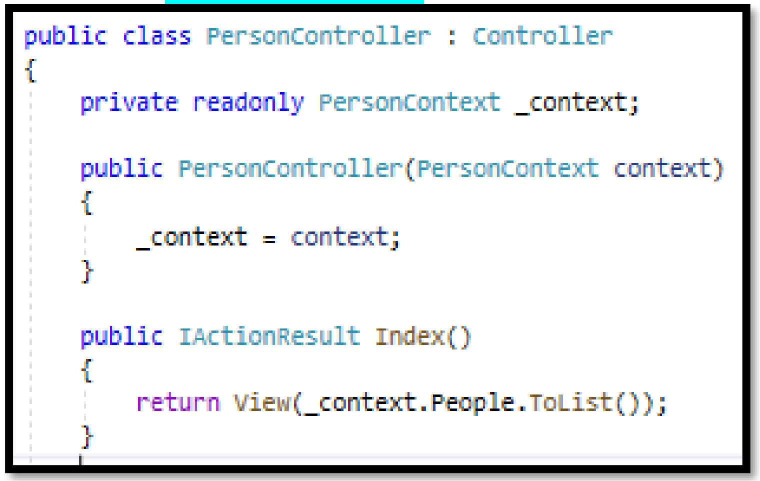
- You’ll need to install the **Microsoft.EntityFrameworkCore.Sqlite** NuGet package

- You’ll need to create a **DBContext** derived class
  - In it, you’ll need **DbSet** properties for each entity

- To use it as a service throughout your application, you’ll need to configure and register this class
  - In ConfigureServices method add something similar to:
  - services.AddDbContext<HrContext>(options => options.UseSqlite("Data Source=example.db"));

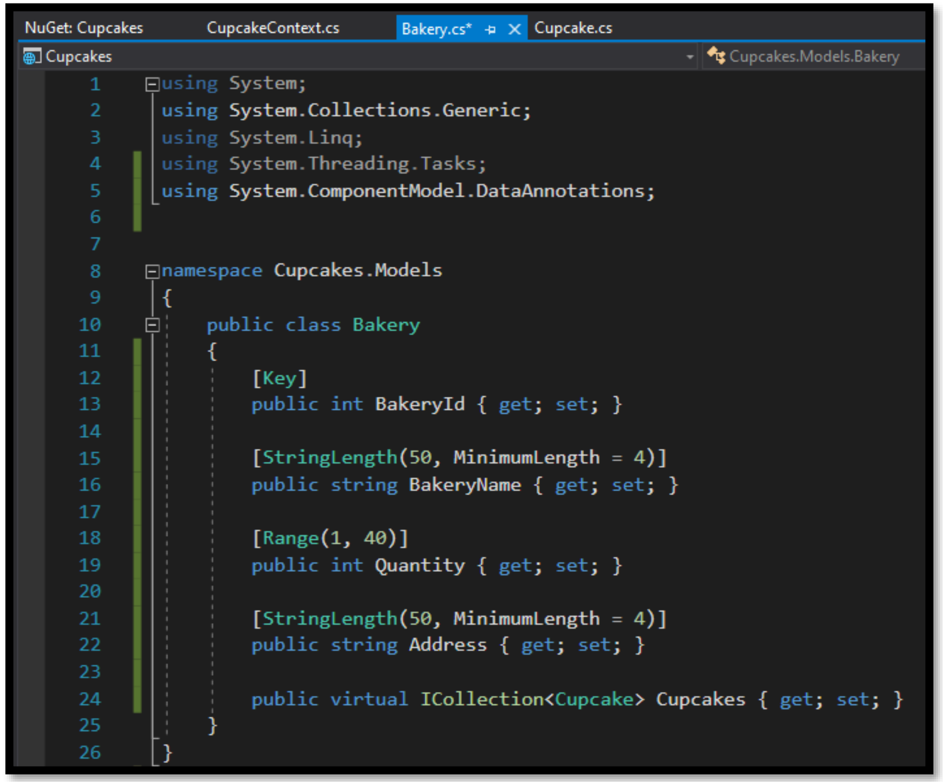
- Then inject it (like any other service) where you need it
  - For a controller class, create a private field, and set up the constructor

- And use it.
  - Remember to call SaveChanges() if you make changes to the Database data

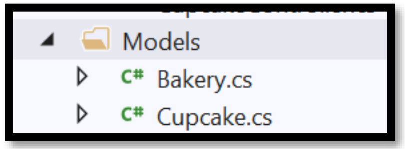


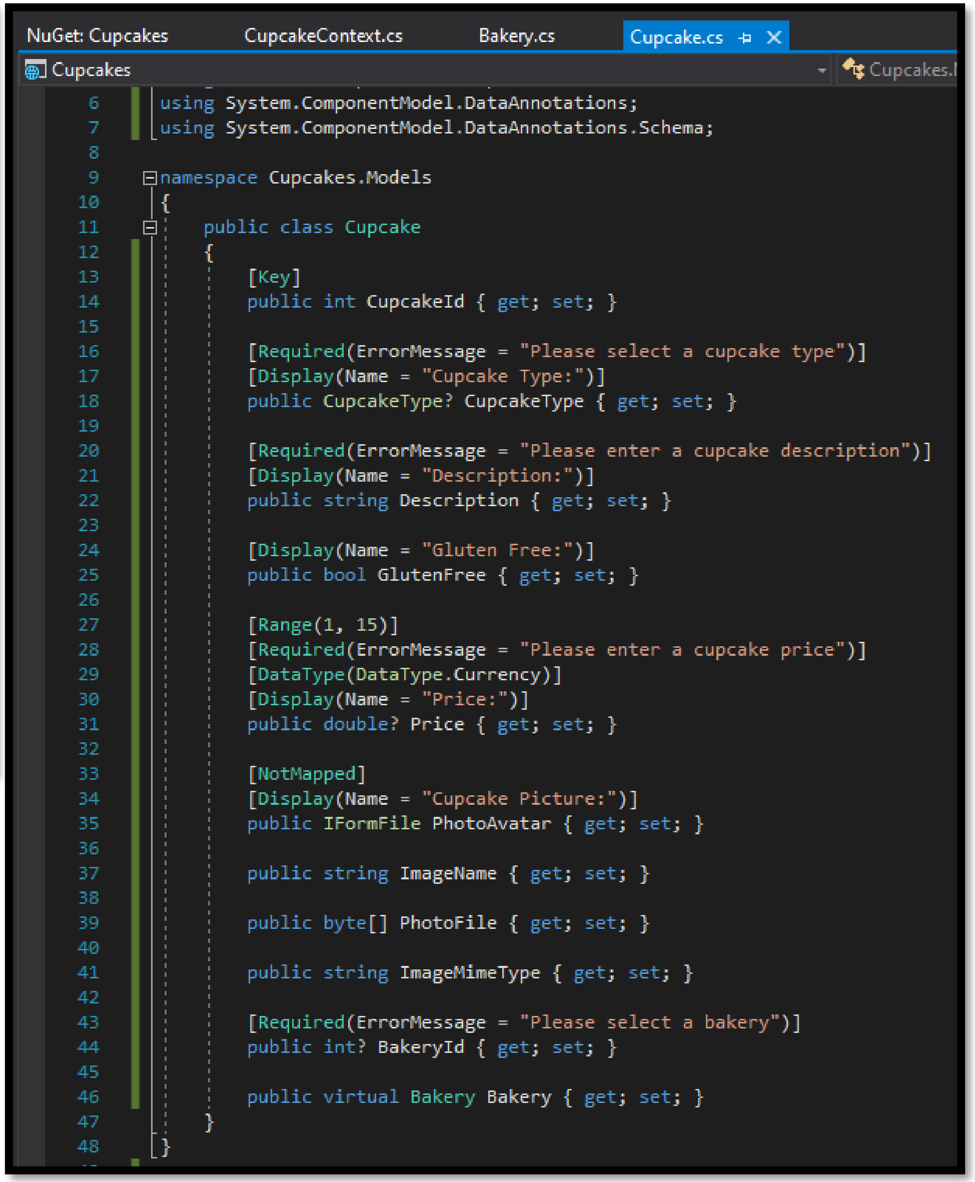
```
public class PersonController : Controller
{
    private readonly PersonContext _context;

    public PersonController(PersonContext context)
    {
        _context = context;
    }

    public IActionResult Index()
    {
        return View(_context.People.ToList());
    }
```

THIS IS YOUR HOMEWORK – ENTITIES – PREVIEW



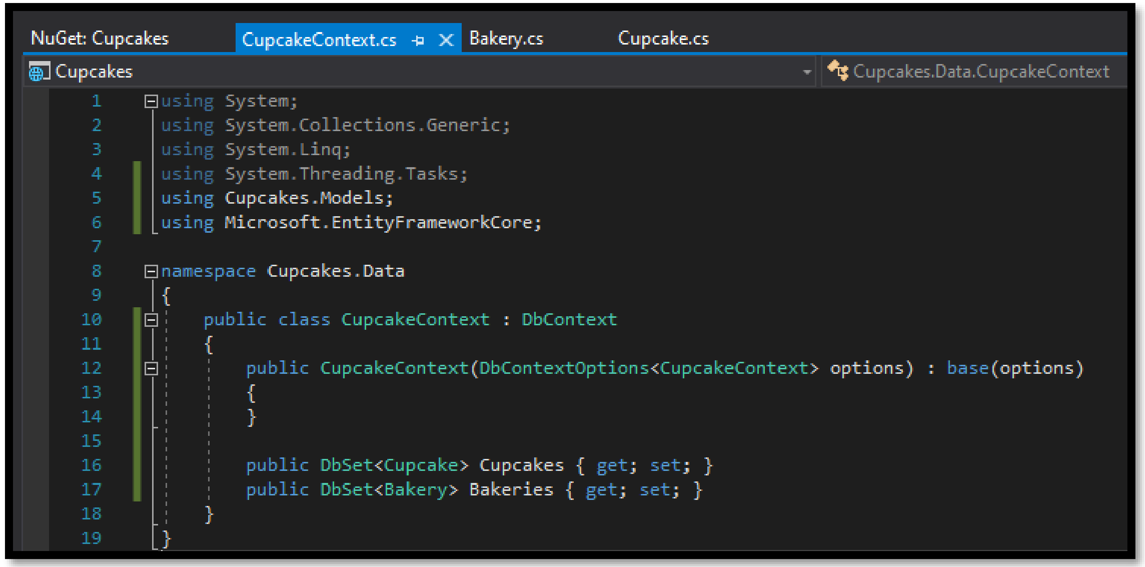
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.ComponentModel.DataAnnotations;


namespace Cupcakes.Models
{
    public class Bakery
    {
        [Key]
        public int BakeryId { get; set; }

        [StringLength(50, MinimumLength = 4)]
        public string BakeryName { get; set; }

        [Range(1, 40)]
        public int Quantity { get; set; }

        [StringLength(50, MinimumLength = 4)]
        public string Address { get; set; }

        public virtual ICollection<Cupcake> Cupcakes { get; set; }
    }
}
```



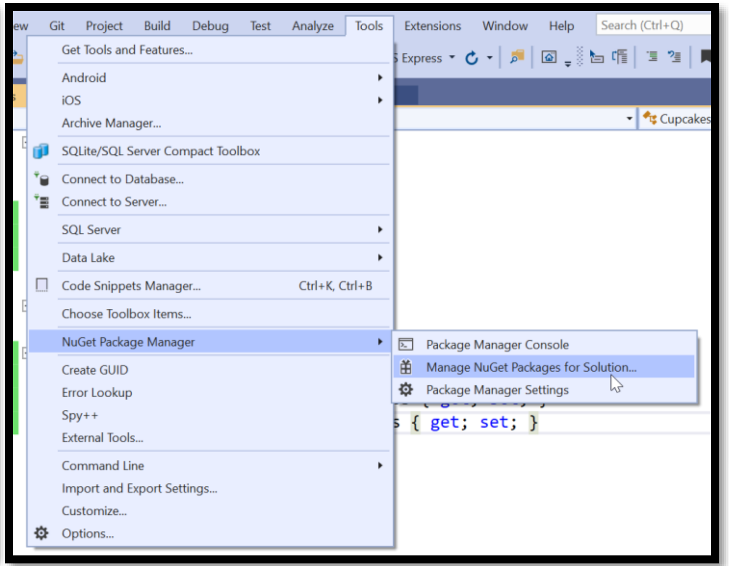


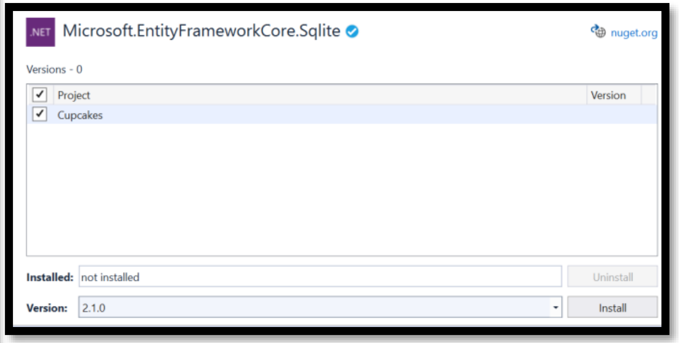
```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace Cupcakes.Models
{
    public class Cupcake
    {
        [Key]
        public int CupcakeId { get; set; }

        [Required(ErrorMessage = "Please select a cupcake type")]
        [Display(Name = "Cupcake Type:")]
        public CupcakeType? CupcakeType { get; set; }

        [Required(ErrorMessage = "Please enter a cupcake description")]
        [Display(Name = "Description:")]
        public string Description { get; set; }

        [Display(Name = "Gluten Free:")]
        public bool GlutenFree { get; set; }

        [Range(1, 15)]
        [Required(ErrorMessage = "Please enter a cupcake price")]
        [DataType(DataType.Currency)]
        [Display(Name = "Price:")]
        public double? Price { get; set; }

        [NotMapped]
        [Display(Name = "Cupcake Picture:")]
        public IFormFile PhotoAvatar { get; set; }

        public string ImageName { get; set; }

        public byte[] PhotoFile { get; set; }

        public string ImageMimeType { get; set; }

        [Required(ErrorMessage = "Please select a bakery")]
        public int? BakeryId { get; set; }

        public virtual Bakery Bakery { get; set; }
    }
}
```

THIS IS YOUR HOMEWORK – ENTITY FRAMEWORK & NUGET – PREVIEW



```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Cupcakes.Models;
using Microsoft.EntityFrameworkCore;

namespace Cupcakes.Data
{
    public class CupcakeContext : DbContext
    {
        public CupcakeContext(DbContextOptions<CupcakeContext> options) : base(options)
        {
        }

        public DbSet<Cupcake> Cupcakes { get; set; }
        public DbSet<Bakery> Bakeries { get; set; }
    }
}
```





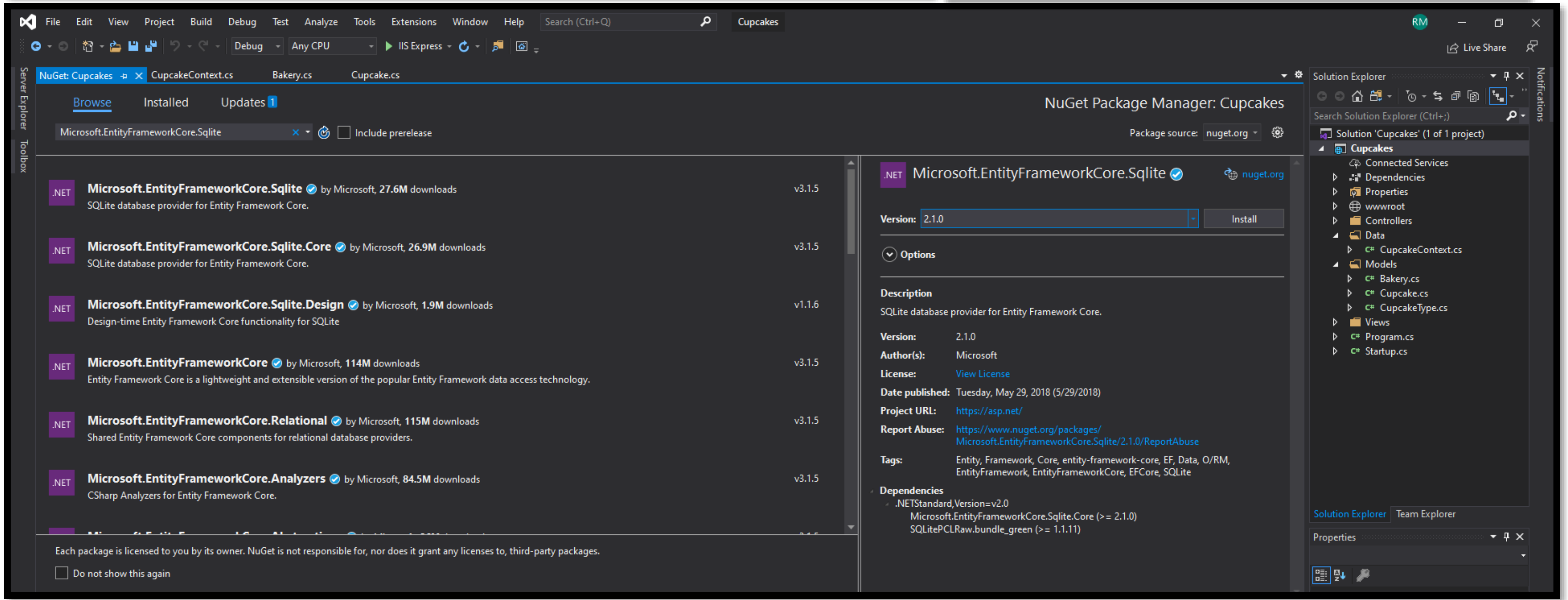
Microsoft.EntityFrameworkCore.Sqlite

Versions - 0

	Project	Version
☑	Cupcakes	

Installed: not installed

Version: 2.1.0



NuGet Package Manager: Cupcakes

Microsoft.EntityFrameworkCore.Sqlite ☐ Include prerelease

Package source: nuget.org

- **Microsoft.EntityFrameworkCore.Sqlite** by Microsoft, 27.6M downloads — v3.1.5
  SQLite database provider for Entity Framework Core.
- **Microsoft.EntityFrameworkCore.Sqlite.Core** by Microsoft, 26.9M downloads — v3.1.5
  SQLite database provider for Entity Framework Core.
- **Microsoft.EntityFrameworkCore.Sqlite.Design** by Microsoft, 1.9M downloads — v1.1.6
  Design-time Entity Framework Core functionality for SQLite
- **Microsoft.EntityFrameworkCore** by Microsoft, 114M downloads — v3.1.5
  Entity Framework Core is a lightweight and extensible version of the popular Entity Framework data access technology.
- **Microsoft.EntityFrameworkCore.Relational** by Microsoft, 115M downloads — v3.1.5
  Shared Entity Framework Core components for relational database providers.
- **Microsoft.EntityFrameworkCore.Analyzers** by Microsoft, 84.5M downloads — v3.1.5
  CSharp Analyzers for Entity Framework Core.

Each package is licensed to you by its owner. NuGet is not responsible for, nor does it grant any licenses to, third-party packages.

☐ Do not show this again

.NET Microsoft.EntityFrameworkCore.Sqlite — nuget.org

Version: 2.1.0

Description

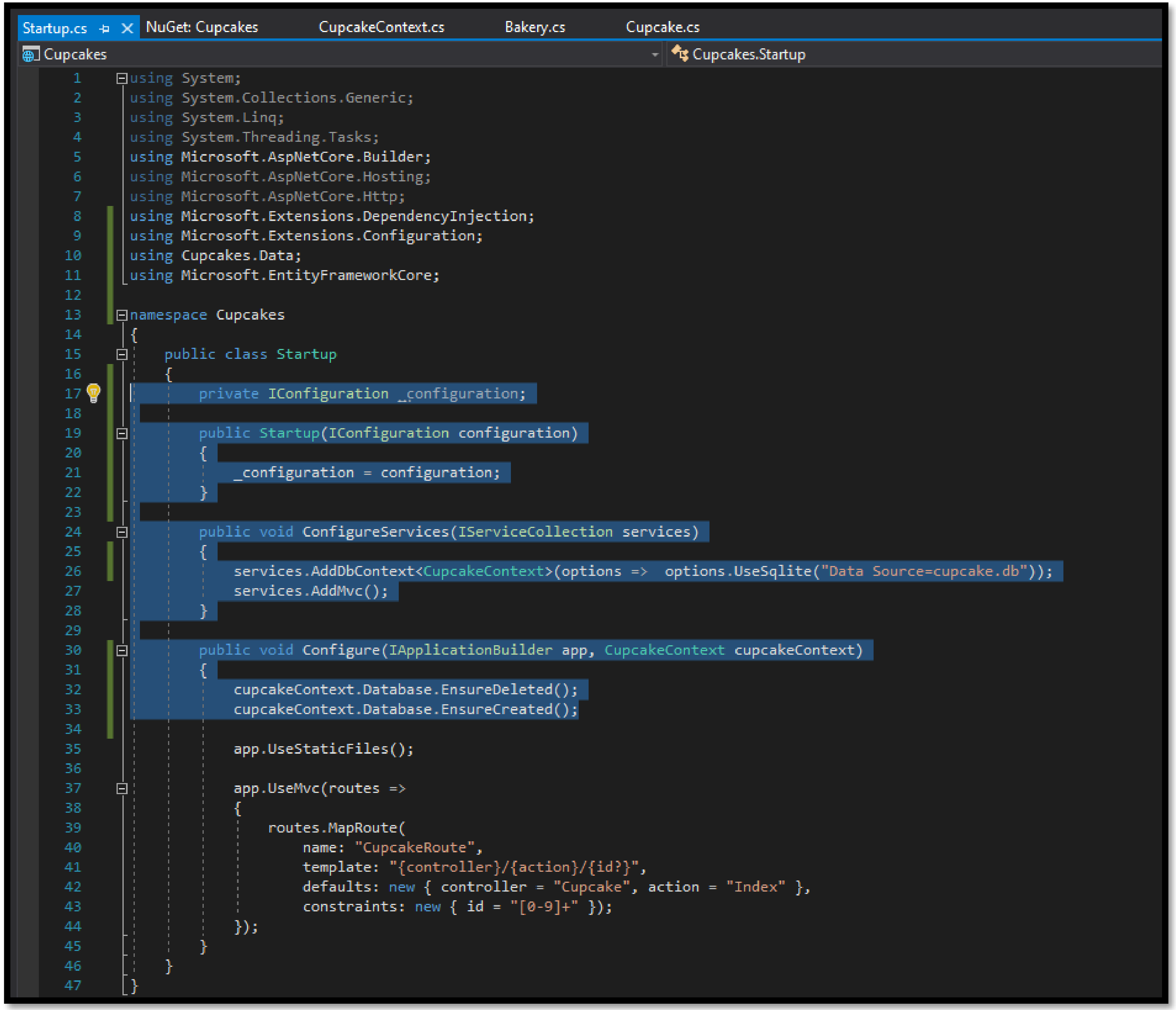
SQLite database provider for Entity Framework Core.

Version: 2.1.0
Author(s): Microsoft
License: View License
Date published: Tuesday, May 29, 2018 (5/29/2018)
Project URL: https://asp.net/
Report Abuse: https://www.nuget.org/packages/Microsoft.EntityFrameworkCore.Sqlite/2.1.0/ReportAbuse
Tags: Entity, Framework, Core, entity-framework-core, EF, Data, O/RM, EntityFramework, EntityFrameworkCore, EFCore, SQLite

Dependencies

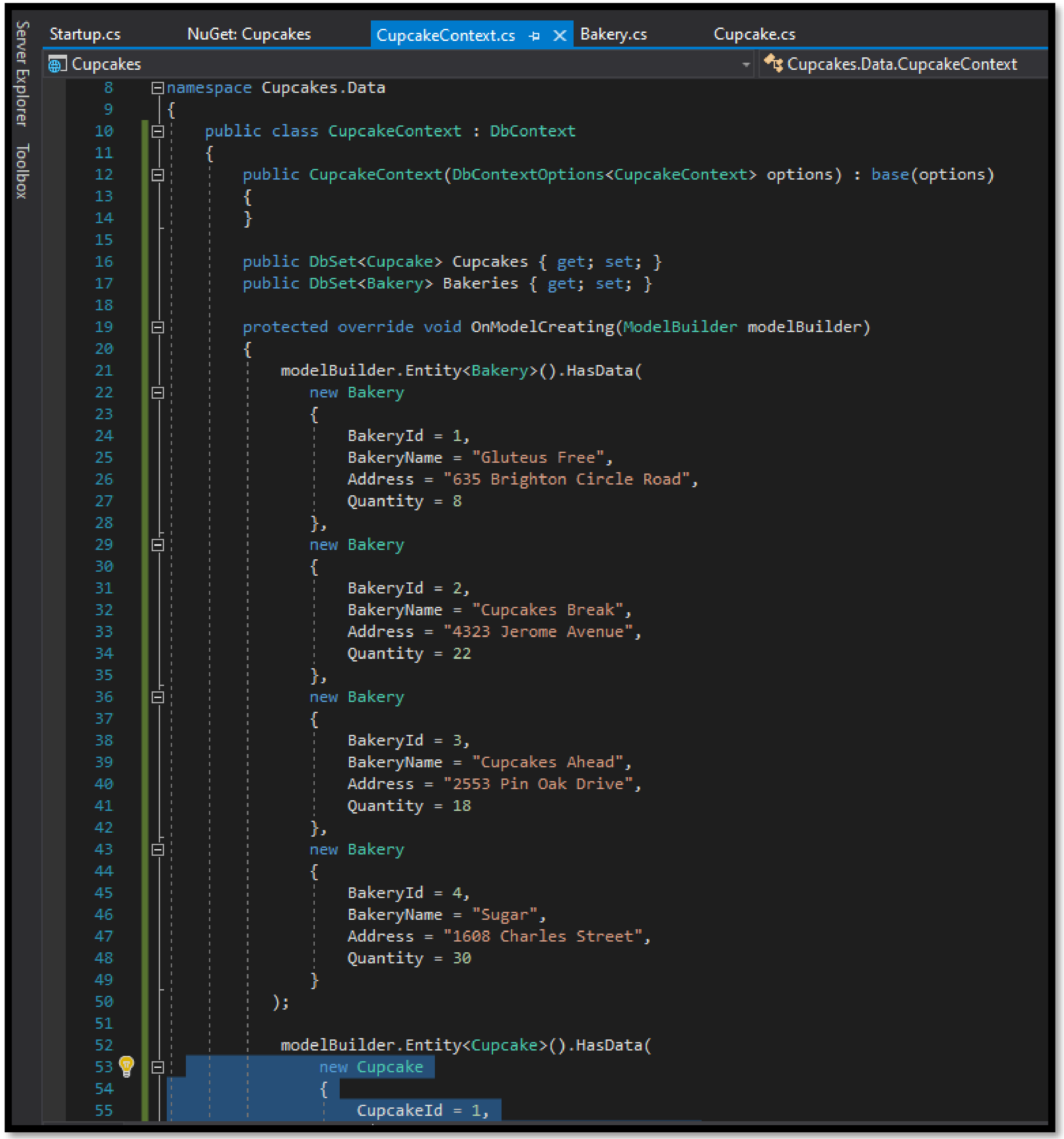
.NETStandard,Version=v2.0
Microsoft.EntityFrameworkCore.Sqlite.Core (>= 2.1.0)
SQLitePCLRaw.bundle_green (>= 1.1.11)

THIS IS YOUR HOMEWORK – CONFIG+INJECTION – PREVIEW



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Http;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Configuration;
using Cupcakes.Data;
using Microsoft.EntityFrameworkCore;

namespace Cupcakes
{
    public class Startup
    {
        private IConfiguration _configuration;

        public Startup(IConfiguration configuration)
        {
            _configuration = configuration;
        }

        public void ConfigureServices(IServiceCollection services)
        {
            services.AddDbContext<CupcakeContext>(options =>  options.UseSqlite("Data Source=cupcake.db"));
            services.AddMvc();
        }

        public void Configure(IApplicationBuilder app, CupcakeContext cupcakeContext)
        {
            cupcakeContext.Database.EnsureDeleted();
            cupcakeContext.Database.EnsureCreated();

            app.UseStaticFiles();

            app.UseMvc(routes =>
            {
                routes.MapRoute(
                    name: "CupcakeRoute",
                    template: "{controller}/{action}/{id?}",
                    defaults: new { controller = "Cupcake", action = "Index" },
                    constraints: new { id = "[0-9]+" });
            });
        }
    }
}
```

THIS IS YOUR HOMEWORK– ADD INITIAL DATA – PREVIEW



```
namespace Cupcakes.Data
{
    public class CupcakeContext : DbContext
    {
        public CupcakeContext(DbContextOptions<CupcakeContext> options) : base(options)
        {
        }

        public DbSet<Cupcake> Cupcakes { get; set; }
        public DbSet<Bakery> Bakeries { get; set; }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder.Entity<Bakery>().HasData(
               new Bakery
               {
                   BakeryId = 1,
                   BakeryName = "Gluteus Free",
                   Address = "635 Brighton Circle Road",
                   Quantity = 8
               },
               new Bakery
               {
                   BakeryId = 2,
                   BakeryName = "Cupcakes Break",
                   Address = "4323 Jerome Avenue",
                   Quantity = 22
               },
               new Bakery
               {
                   BakeryId = 3,
                   BakeryName = "Cupcakes Ahead",
                   Address = "2553 Pin Oak Drive",
                   Quantity = 18
               },
               new Bakery
               {
                   BakeryId = 4,
                   BakeryName = "Sugar",
                   Address = "1608 Charles Street",
                   Quantity = 30
               }
            );

            modelBuilder.Entity<Cupcake>().HasData(
                new Cupcake
                {
                    CupcakeId = 1,
```

THIS IS YOUR HOMEWORK– INJECT AND USE EF – PREVIEW

- Now, we can inject the **DbContext** derived instance to any place we need (to all controllers, for example) and use it to interact with a database (add/edit/delete…)

OTHER OPTIONAL EF RESOURCES

- https://www.youtube.com/watch?v=gPGVklH1bg4
- https://www.codeproject.com/Articles/1158937/SQLite-with-Csharp-Net-and-Entity-Framework
- https://www.learnentityframeworkcore.com/

OTHER OPTIONAL RESOURCES

- https://www.youtube.com/watch?v=ayp3tHEkRc0
- **Using SQLite in C# - Building Simple, Powerful, Portable Databases for Your Application**

- **https://www.youtube.com/watch?v=S9HrLdSrVho**
- **Entity Framework - Part 0 - Introduction**

THE REPOSITORY PATTERN

- We can inject the **dbcontext** and use it to interact with a database (add/edit/delete…)

- One (also the homework) can add one more level of abstraction, a **repository** …
  - Particularly useful for **unit testing** (test with in-memory data) and in general, when we want to accomplish **separation of concerns**

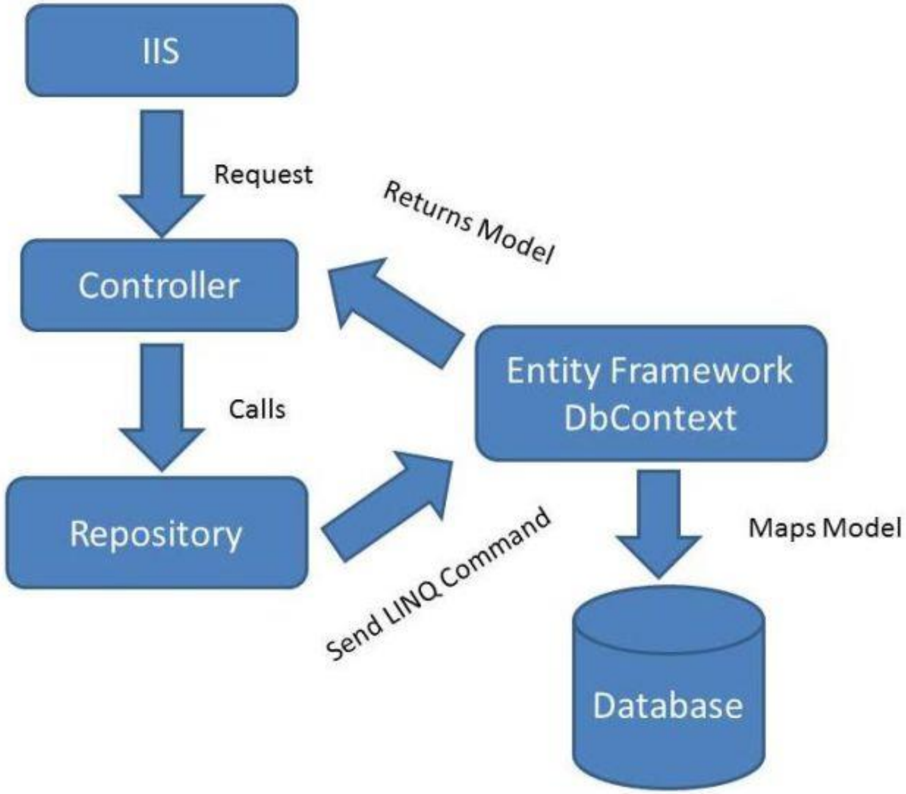
- Steps to use a repository (in a controller):

1. Define an **interface** for the repository class
   - declare here the methods you want to be available to the controller
   - Examples: add, delete, edit, …

2. Define the **class** that implements the above interface
   - This class must implement all the data access methods declared above.
   - It will make use of the **DbContext** derived class
   - It's called the Repository class

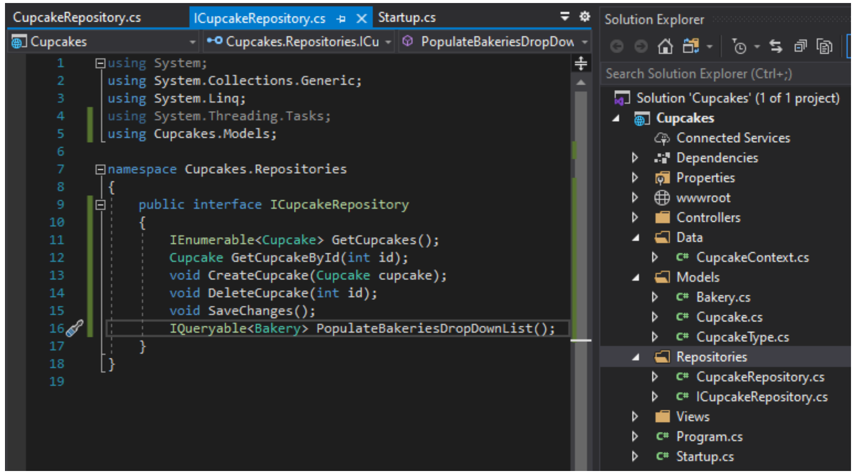
3. Use the interface and class defined above, to declare a **service**.

4. Use dependency injection to inject the repository class to a **controller**.
   - You can modify controller class to use the repository class, not the **DbContext** class.

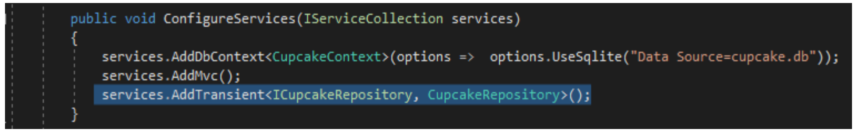


- Image source:
  - https://www.c-sharpcorner.com/UploadFile/3d39b4/crud-using-the-repository-pattern-in-mvc/

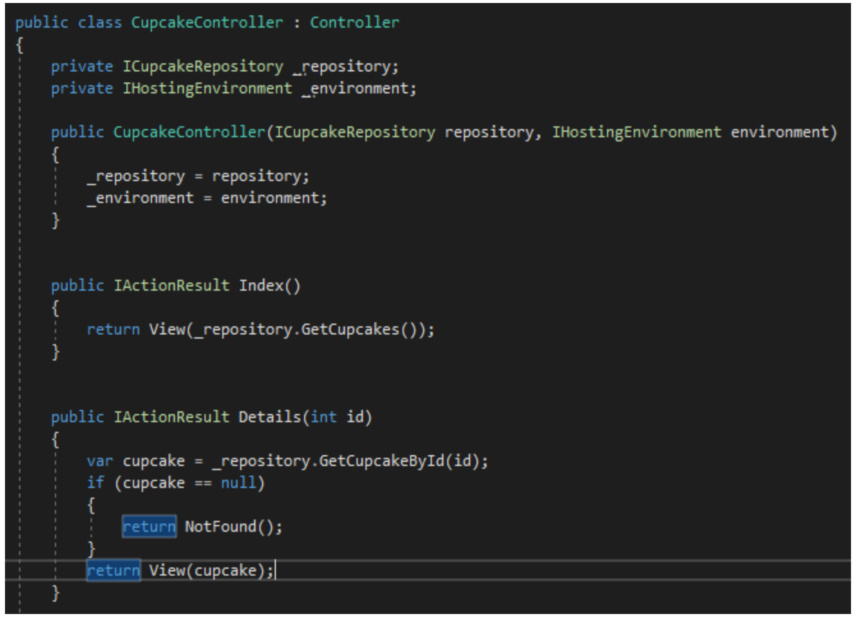
THIS IS YOUR HOMEWORK – SKIP IF NO TIME – REPOSITORY



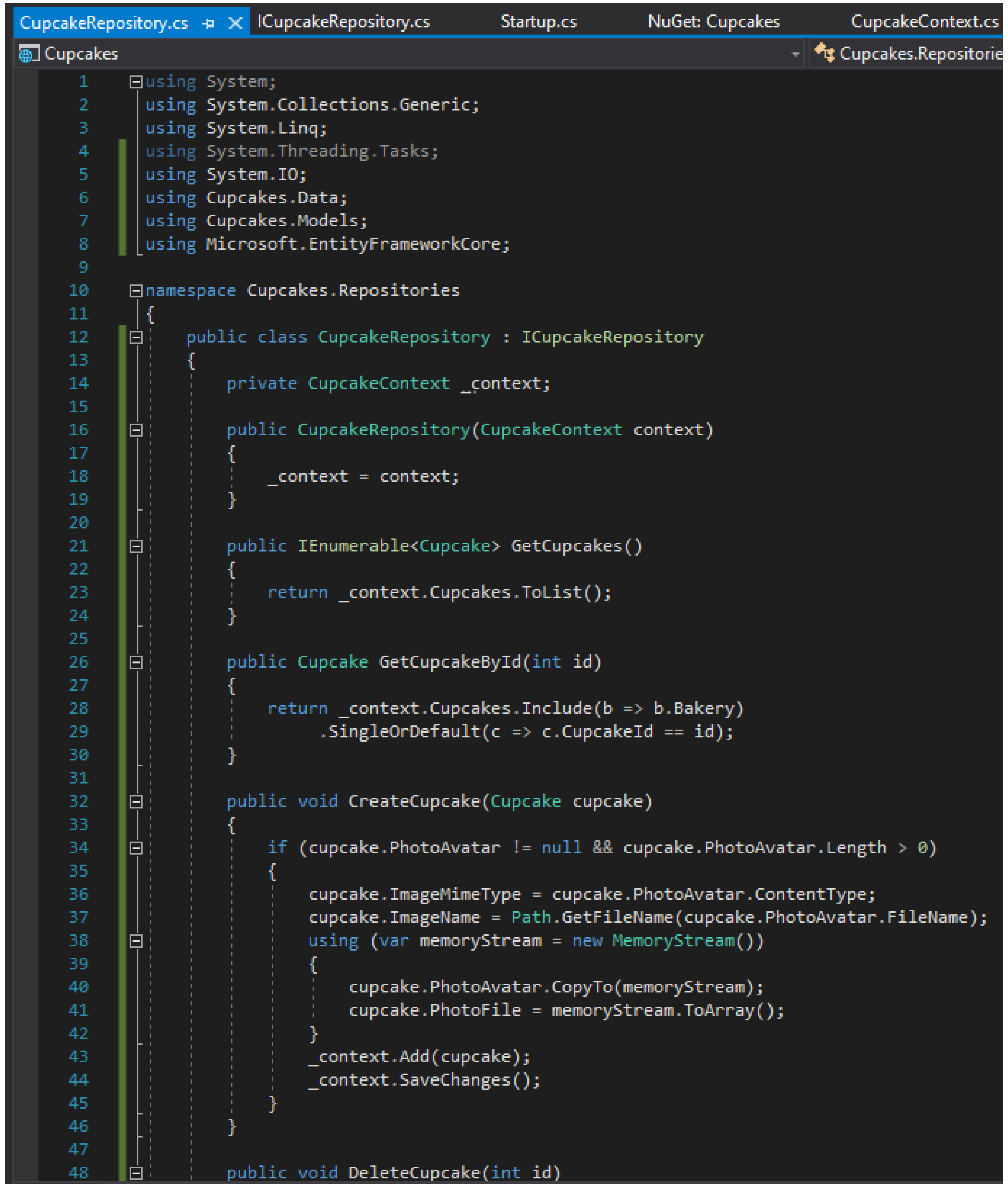
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Cupcakes.Models;

namespace Cupcakes.Repositories
{
    public interface ICupcakeRepository
    {
        IEnumerable<Cupcake> GetCupcakes();
        Cupcake GetCupcakeById(int id);
        void CreateCupcake(Cupcake cupcake);
        void DeleteCupcake(int id);
        void SaveChanges();
        IQueryable<Bakery> PopulateBakeriesDropDownList();
    }
}
```



```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<CupcakeContext>(options =>  options.UseSqlite("Data Source=cupcake.db"));
    services.AddMvc();
    services.AddTransient<ICupcakeRepository, CupcakeRepository>();
}
```



```
public class CupcakeController : Controller
{
    private ICupcakeRepository _repository;
    private IHostingEnvironment _environment;

    public CupcakeController(ICupcakeRepository repository, IHostingEnvironment environment)
    {
        _repository = repository;
        _environment = environment;
    }


    public IActionResult Index()
    {
        return View(_repository.GetCupcakes());
    }


    public IActionResult Details(int id)
    {
        var cupcake = _repository.GetCupcakeById(id);
        if (cupcake == null)
        {
            return NotFound();
        }
        return View(cupcake);
    }
```



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.IO;
using Cupcakes.Data;
using Cupcakes.Models;
using Microsoft.EntityFrameworkCore;

namespace Cupcakes.Repositories
{
    public class CupcakeRepository : ICupcakeRepository
    {
        private CupcakeContext _context;

        public CupcakeRepository(CupcakeContext context)
        {
            _context = context;
        }

        public IEnumerable<Cupcake> GetCupcakes()
        {
            return _context.Cupcakes.ToList();
        }

        public Cupcake GetCupcakeById(int id)
        {
            return _context.Cupcakes.Include(b => b.Bakery)
                .SingleOrDefault(c => c.CupcakeId == id);
        }

        public void CreateCupcake(Cupcake cupcake)
        {
            if (cupcake.PhotoAvatar != null && cupcake.PhotoAvatar.Length > 0)
            {
                cupcake.ImageMimeType = cupcake.PhotoAvatar.ContentType;
                cupcake.ImageName = Path.GetFileName(cupcake.PhotoAvatar.FileName);
                using (var memoryStream = new MemoryStream())
                {
                    cupcake.PhotoAvatar.CopyTo(memoryStream);
                    cupcake.PhotoFile = memoryStream.ToArray();
                }
                _context.Add(cupcake);
                _context.SaveChanges();
            }
        }

        public void DeleteCupcake(int id)
```

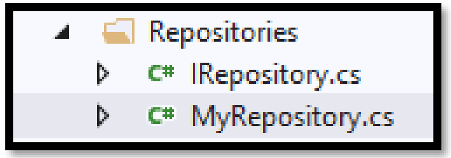
IN-CLASS DEMO

Demonstration: How to Apply the Repository Pattern - Connect to MS SQL Server

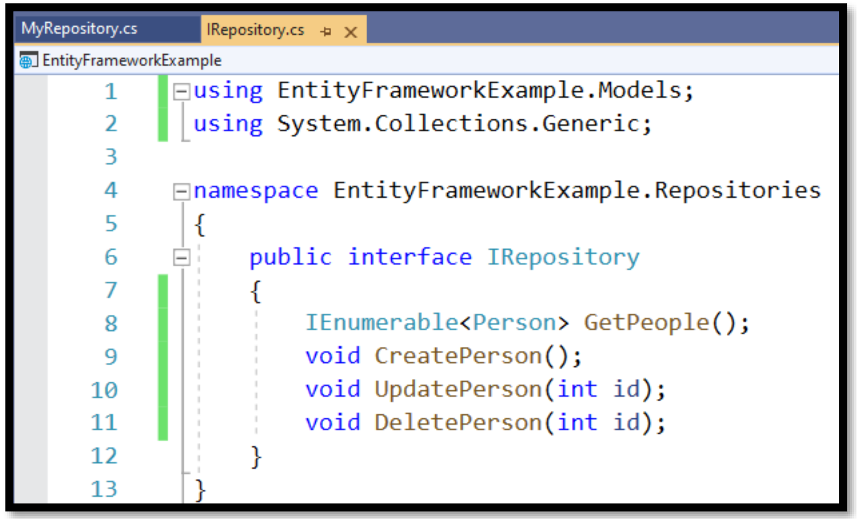
- Source/Steps
  - https://github.com/MicrosoftLearning/20486D-DevelopingASPNETMVCWebApplications/blob/master/Instructions/20486D_MOD07_DEMO.md#lesson-2-working-with-entity-framework-core

SCREENSHOTS …WHAT WE’LL DO

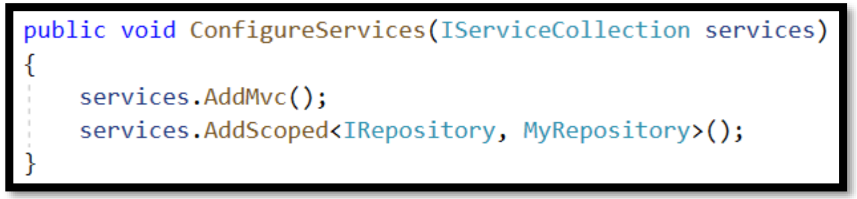
- What are we doing in here?



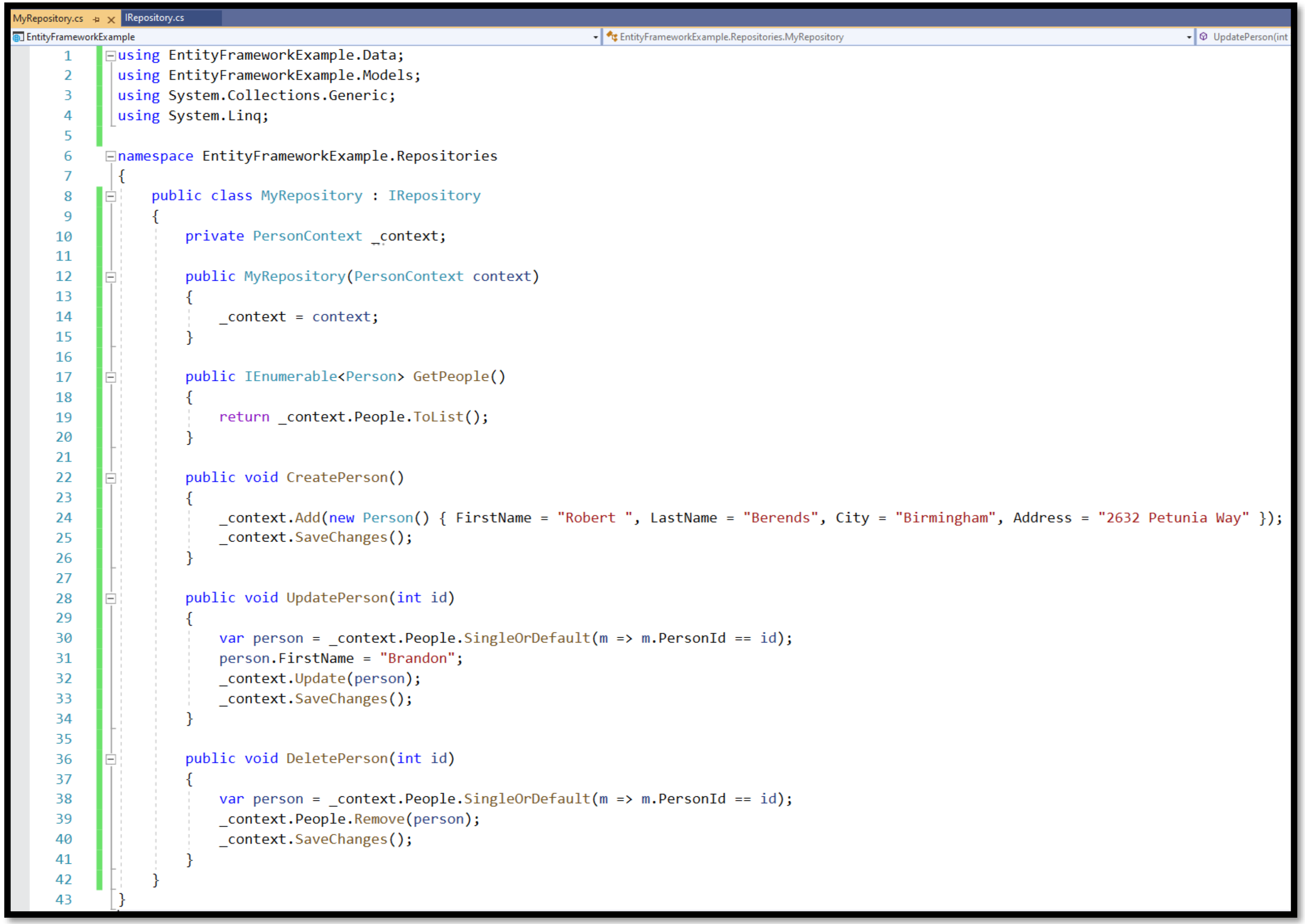
Repositories
IRepository.cs
MyRepository.cs



```
using EntityFrameworkExample.Models;
using System.Collections.Generic;

namespace EntityFrameworkExample.Repositories
{
    public interface IRepository
    {
        IEnumerable<Person> GetPeople();
        void CreatePerson();
        void UpdatePerson(int id);
        void DeletePerson(int id);
    }
}
```



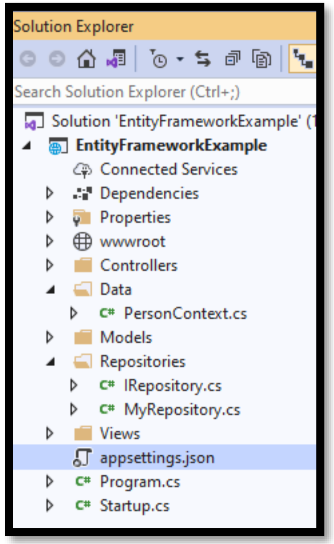
```
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc();
    services.AddScoped<IRepository, MyRepository>();
}
```



```
using EntityFrameworkExample.Data;
using EntityFrameworkExample.Models;
using System.Collections.Generic;
using System.Linq;

namespace EntityFrameworkExample.Repositories
{
    public class MyRepository : IRepository
    {
        private PersonContext _context;

        public MyRepository(PersonContext context)
        {
            _context = context;
        }

        public IEnumerable<Person> GetPeople()
        {
            return _context.People.ToList();
        }

        public void CreatePerson()
        {
            _context.Add(new Person() { FirstName = "Robert ", LastName = "Berends", City = "Birmingham", Address = "2632 Petunia Way" });
            _context.SaveChanges();
        }

        public void UpdatePerson(int id)
        {
            var person = _context.People.SingleOrDefault(m => m.PersonId == id);
            person.FirstName = "Brandon";
            _context.Update(person);
            _context.SaveChanges();
        }

        public void DeletePerson(int id)
        {
            var person = _context.People.SingleOrDefault(m => m.PersonId == id);
            _context.People.Remove(person);
            _context.SaveChanges();
        }
    }
}
```

SCREENSHOTS …WHAT WE'LL DO

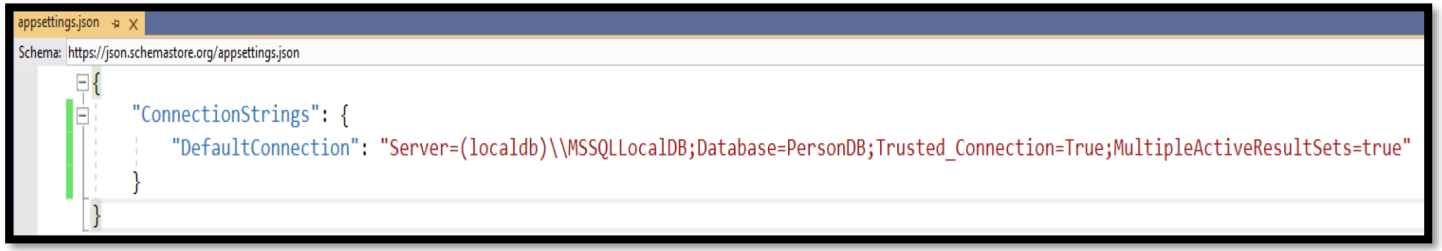
- What are we doing in here?



Solution Explorer

Search Solution Explorer (Ctrl+;)

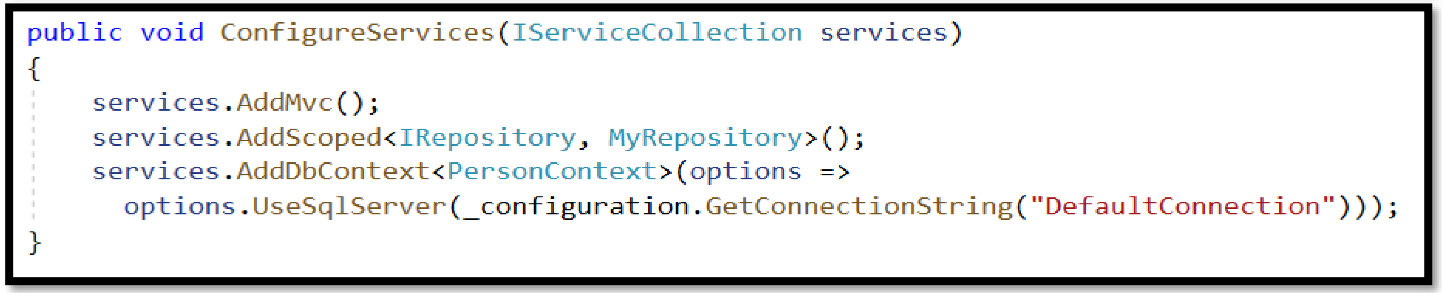
- Solution 'EntityFrameworkExample' (
  - **EntityFrameworkExample**
    - Connected Services
    - Dependencies
    - Properties
    - wwwroot
    - Controllers
    - Data
      - PersonContext.cs
    - Models
    - Repositories
      - IRepository.cs
      - MyRepository.cs
    - Views
    - appsettings.json
    - Program.cs
    - Startup.cs



appsettings.json

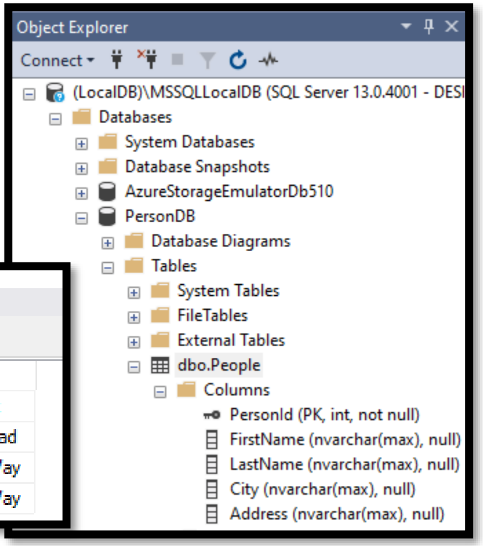
Schema: https://json.schemastore.org/appsettings.json

```json
{
  "ConnectionStrings": {
    "DefaultConnection": "Server=(localdb)\\MSSQLLocalDB;Database=PersonDB;Trusted_Connection=True;MultipleActiveResultSets=true"
  }
}
```



```csharp
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc();
    services.AddScoped<IRepository, MyRepository>();
    services.AddDbContext<PersonContext>(options =>
      options.UseSqlServer(_configuration.GetConnectionString("DefaultConnection")));
}
```

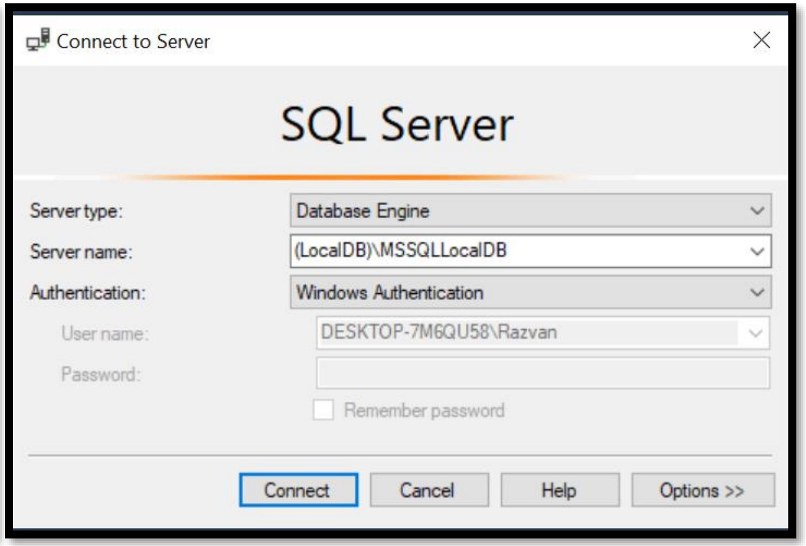
https://www.connectionstrings.com/sql-server/



Object Explorer

Connect ▾

- (LocalDB)\MSSQLLocalDB (SQL Server 13.0.4001 - DESI
  - Databases
    - System Databases
    - Database Snapshots
    - AzureStorageEmulatorDb510
    - PersonDB
      - Database Diagrams
      - Tables
        - System Tables
        - FileTables
        - External Tables
        - dbo.People
          - Columns
            - PersonId (PK, int, not null)
            - FirstName (nvarchar(max), null)
            - LastName (nvarchar(max), null)
            - City (nvarchar(max), null)
            - Address (nvarchar(max), null)



Connect to Server

SQL Server

Server type: Database Engine

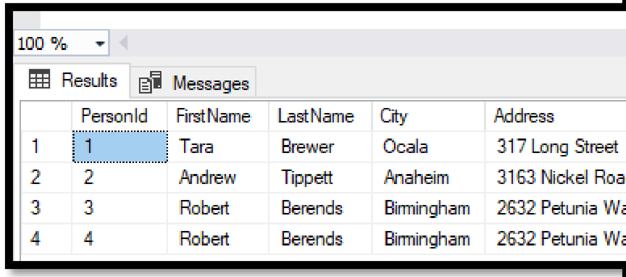
Server name: (LocalDB)\MSSQLLocalDB

Authentication: Windows Authentication

User name: DESKTOP-7M6QU58\Razvan

Password:

□ Remember password



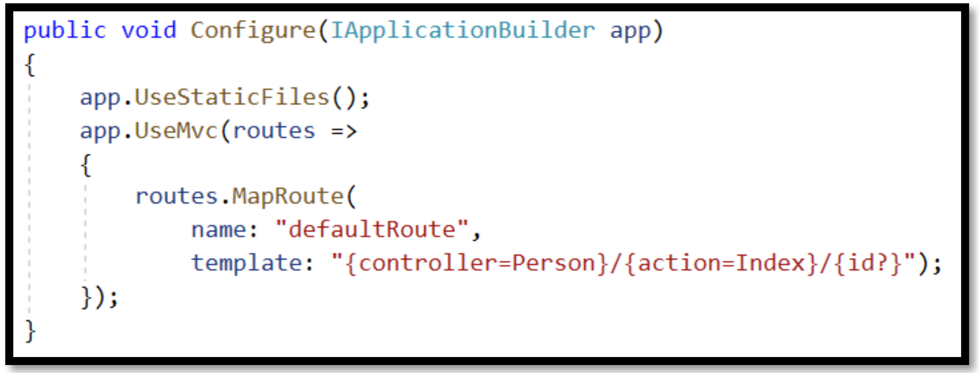
100 %

Results | Messages

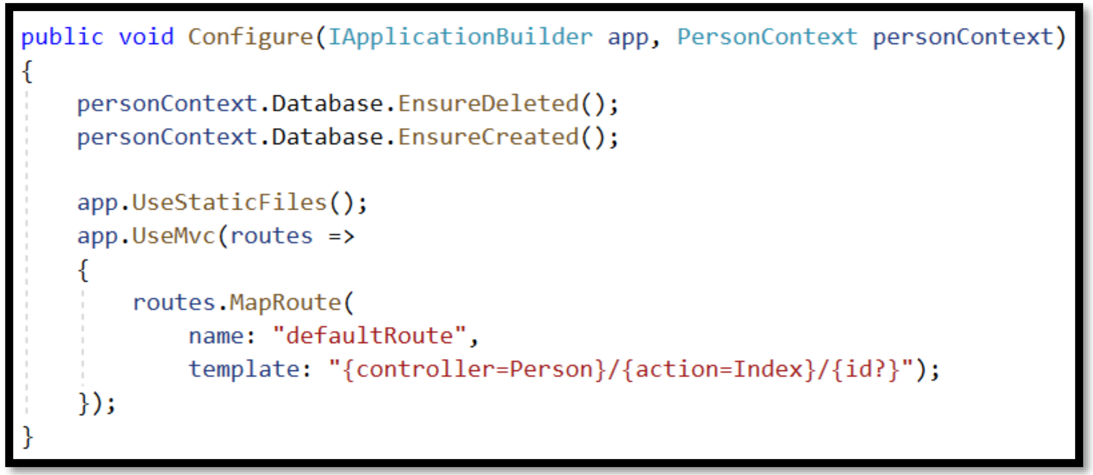
	PersonId	FirstName	LastName	City	Address
1	1	Tara	Brewer	Ocala	317 Long Street
2	2	Andrew	Tippett	Anaheim	3163 Nickel Road
3	3	Robert	Berends	Birmingham	2632 Petunia Way
4	4	Robert	Berends	Birmingham	2632 Petunia Way

SCREENSHOTS …WHAT WE'LL DO

- What are we doing in here?
- Injecting a service in the **middleware**

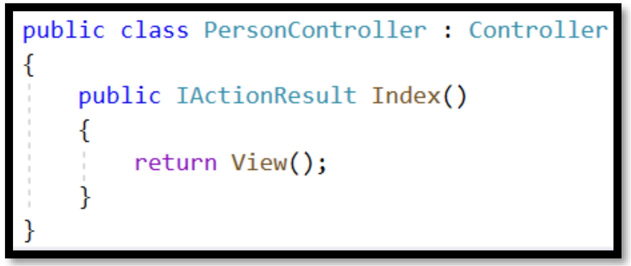


```
public void Configure(IApplicationBuilder app)
{
    app.UseStaticFiles();
    app.UseMvc(routes =>
    {
        routes.MapRoute(
            name: "defaultRoute",
            template: "{controller=Person}/{action=Index}/{id?}");
    });
}
```

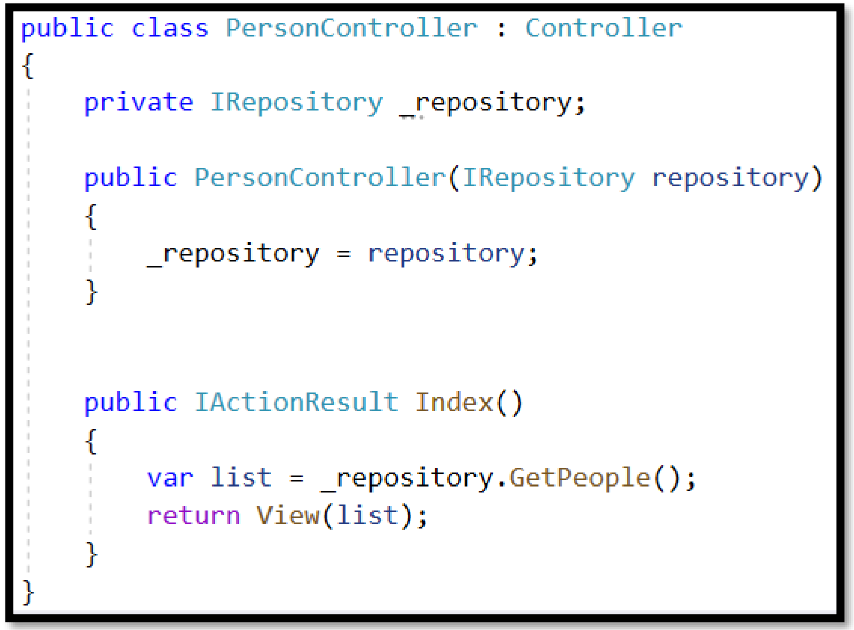


```
public void Configure(IApplicationBuilder app, PersonContext personContext)
{
    personContext.Database.EnsureDeleted();
    personContext.Database.EnsureCreated();

    app.UseStaticFiles();
    app.UseMvc(routes =>
    {
        routes.MapRoute(
            name: "defaultRoute",
            template: "{controller=Person}/{action=Index}/{id?}");
    });
}
```

- Injecting a service in a **controller**



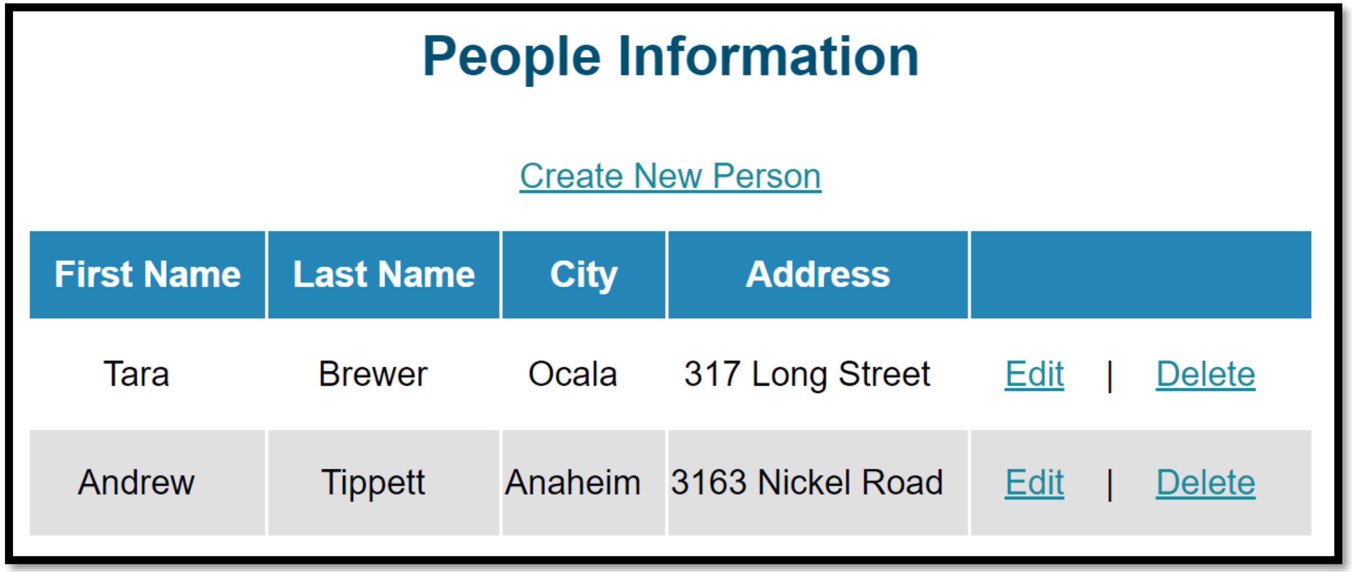
```
public class PersonController : Controller
{
    public IActionResult Index()
    {
        return View();
    }
}
```



```
public class PersonController : Controller
{
    private IRepository _repository;

    public PersonController(IRepository repository)
    {
        _repository = repository;
    }


    public IActionResult Index()
    {
        var list = _repository.GetPeople();
        return View(list);
    }
}
```

SCREENSHOTS …WHAT WE'LL DO

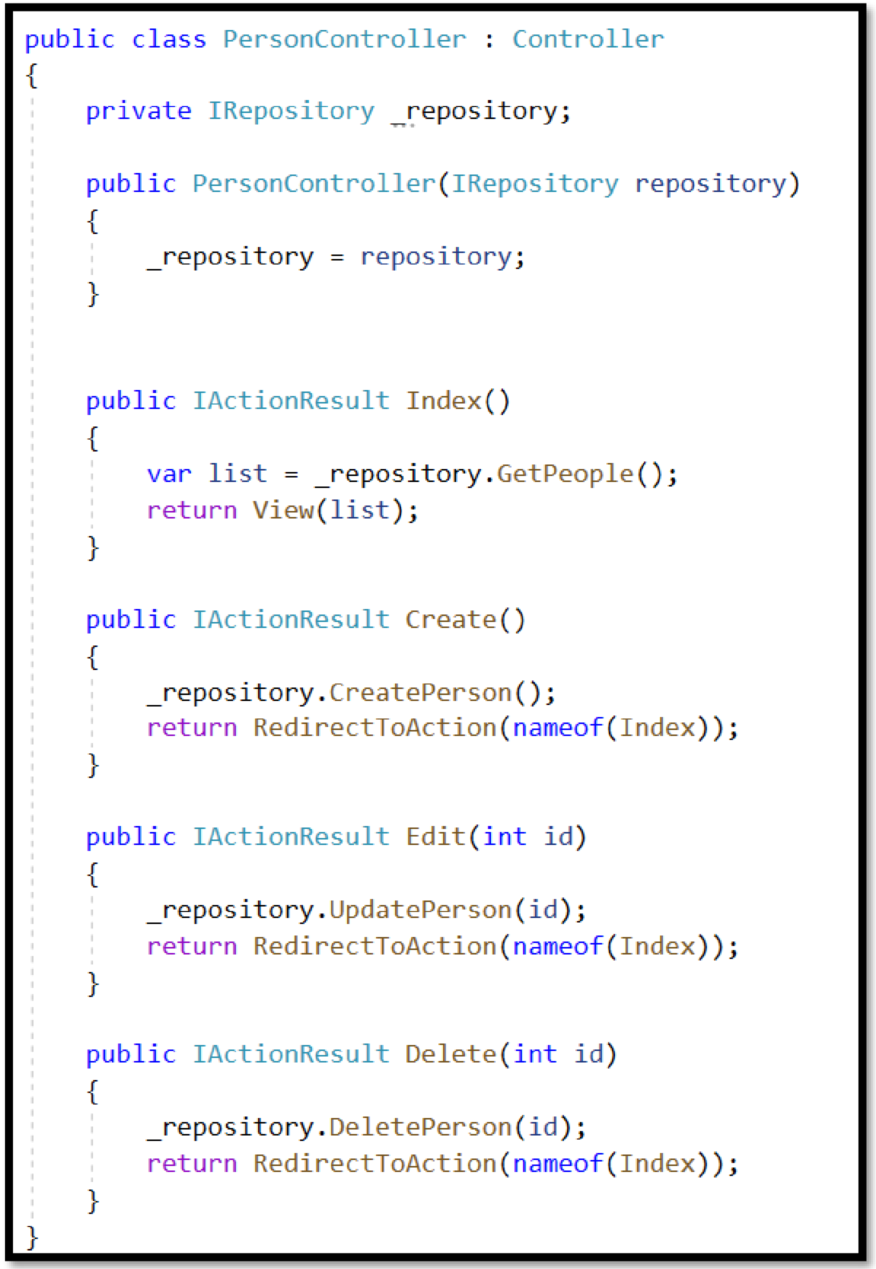
- What are we doing in here?



People Information

Create New Person

First Name	Last Name	City	Address	
Tara	Brewer	Ocala	317 Long Street	Edit | Delete
Andrew	Tippett	Anaheim	3163 Nickel Road	Edit | Delete



```
public class PersonController : Controller
{
    private IRepository _repository;

    public PersonController(IRepository repository)
    {
        _repository = repository;
    }


    public IActionResult Index()
    {
        var list = _repository.GetPeople();
        return View(list);
    }

    public IActionResult Create()
    {
        _repository.CreatePerson();
        return RedirectToAction(nameof(Index));
    }

    public IActionResult Edit(int id)
    {
        _repository.UpdatePerson(id);
        return RedirectToAction(nameof(Index));
    }

    public IActionResult Delete(int id)
    {
        _repository.DeletePerson(id);
        return RedirectToAction(nameof(Index));
    }
}
```

ANOTHER EXAMPLE

- See page 1691+
- “Part 5, work with a database in an ASP.NET Core MVC app”

IF TIME …

- **LINQ** ← **Language Integrated Query**
  - Writing a query using C#
  - LINQ can be used to extract data from databases, enumerable objects, XML documents, etc.
  - Example (see also: https://www.youtube.com/watch?v=PpqdsJDvcxY&list=PLdo4fOcmZ0oX7uTkjYwvCJDG2qhcSzwZ6):

```
var studentsList = from st in db.Students
              where st.IsVeteran == true
              orderby st.LastName
              select st;
```

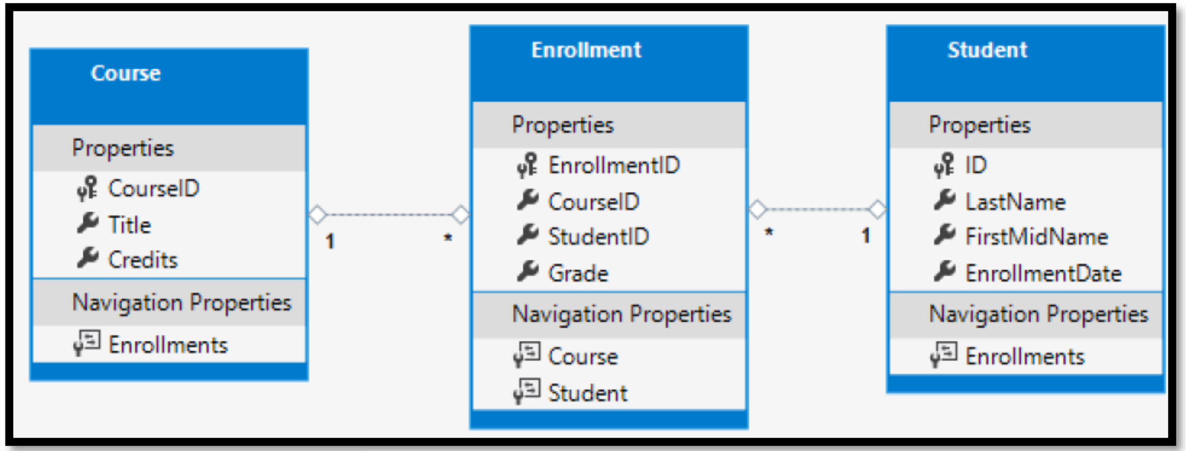
- **Fluent API**
  - Alternative to LINQ
  - Example:

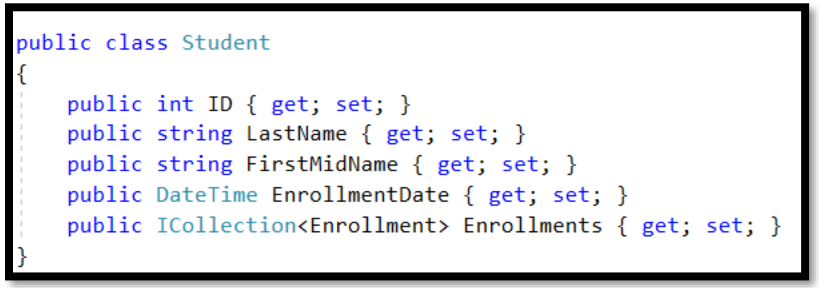
```
var studentsList = db.Students
                    .Where(st => st.IsVeteran == true)
                    .OrderBy(st => st.LastName)
```

RELATED DATA - LINKING ENTITIES – IF TIME

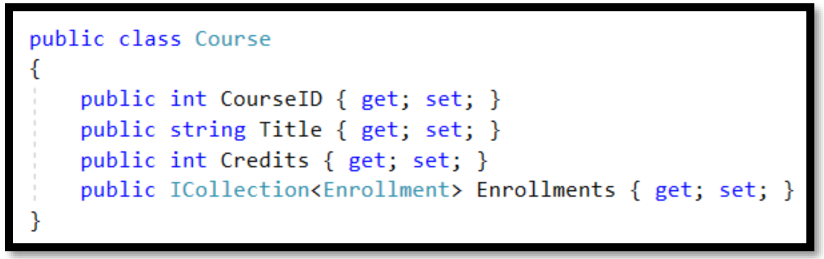
- one can link an **entity** to other **entities** using by **navigation properties**.
  - When an entity is related to another entity, one should add a **navigation property** to represent the association.
    - When the multiplicity of the association is one or zero-or-one, the navigation property is represented by a **reference** object.
    - When the multiplicity of the association is many, the navigation property is represented by a **collection**.

- The following are from page 3854:

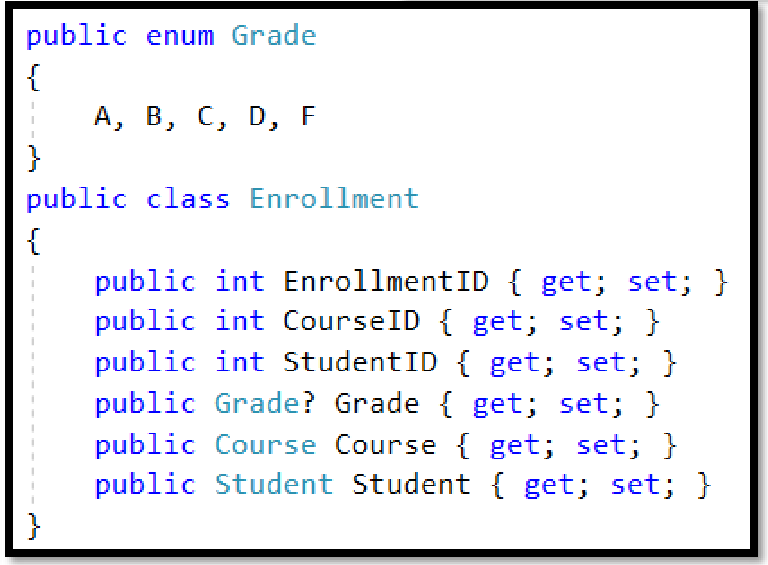




```
public class Student
{
    public int ID { get; set; }
    public string LastName { get; set; }
    public string FirstMidName { get; set; }
    public DateTime EnrollmentDate { get; set; }
    public ICollection<Enrollment> Enrollments { get; set; }
}
```



```
public class Course
{
    public int CourseID { get; set; }
    public string Title { get; set; }
    public int Credits { 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 int StudentID { get; set; }
    public Grade? Grade { get; set; }
    public Course Course { get; set; }
    public Student Student { get; set; }
}
```

LOADING RELATED DATA – IF TIME

- Sources:
  - page 4004+: “Part 6, Razor Pages with EF Core in ASP.NET Core - Read Related Data”
  - https://docs.microsoft.com/en-us/ef/core/querying/related-data/
  - https://docs.microsoft.com/en-us/ef/core/querying/related-data/eager
  - https://docs.microsoft.com/en-us/ef/core/querying/related-data/explicit
  - https://docs.microsoft.com/en-us/ef/core/querying/related-data/lazy

- **Eager loading**: the related data is loaded from the DB as part of the **initial query**.
  - all child entities will be loaded using a **single database call** (source)
  - Use the **Include** method. To include related data from multiple relationships, use the **Include** method several times in the same query.
  - If you need to include more levels of related data, use the **ThenInclude** method

- **Explicit loading**: the related data is explicitly loaded from the DB **at a later time**.
  - if Lazy Loading is turned off, one can still use explicit loading (source)
  - the related data is loaded explicitly from the database after the original query is completed.
    - **Explicit** loading is similar to **lazy** loading, except that: you explicitly retrieve the related data in code; it doesn't happen automatically when you access a navigation property (source: https://stackoverflow.com/questions/34627865/eager-lazy-and-explicit-loading-in-ef6).
  - use the **Entry** method of the Entity Framework context class
  - use the **Load** method for the related entities
  - One can also use the explicit loading ORM pattern in conjunction with LINQ.. For this you need to first call **Query** method

- **Lazy loading**: the related data is transparently loaded from the DB **when the navigation property is accessed**.
  - the default behavior (source)
  - the related data is loaded from the database as you access the navigation property
  - change the navigation property to be **overridden**: use the **virtual** keyword.
  - you should also **turn on the creation of lazy-loading proxies**: call the **UseLazyLoadingProxies** method.
  - UseLazyLoadingProxies method is distributed with the **Microsoft.EntityFrameworkCore.Proxies** NuGet package

IF TIME …

- **Migrations** (making changes to **models** and/or to the **database**)

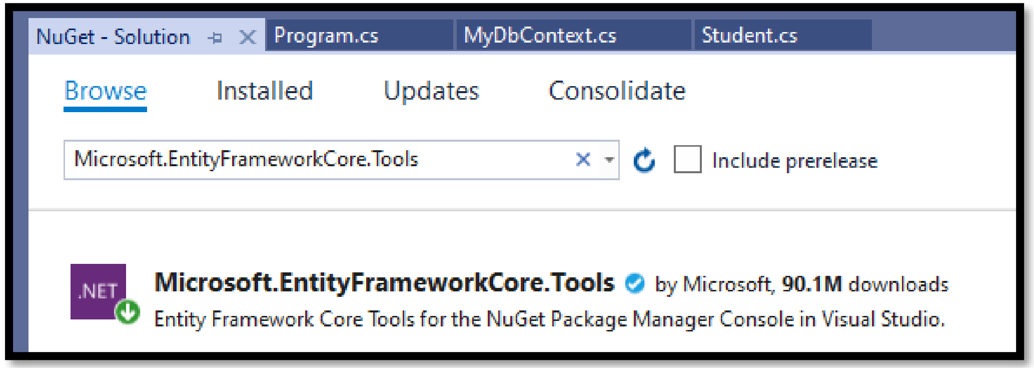
- Sources:
  - Page 4094+
  - Page 4199+
  - https://www.learnentityframeworkcore.com/migrations

- **Add-Migration** command ← generates code to create the initial database schema.
  - The schema is based on the model specified in the **DbContext** derived class

- **Update-Database** command ← runs the Up method in the Migrations/<time-stamp>_InitialCreate.cs file.
  - The Up method creates the database

MIGRATIONS

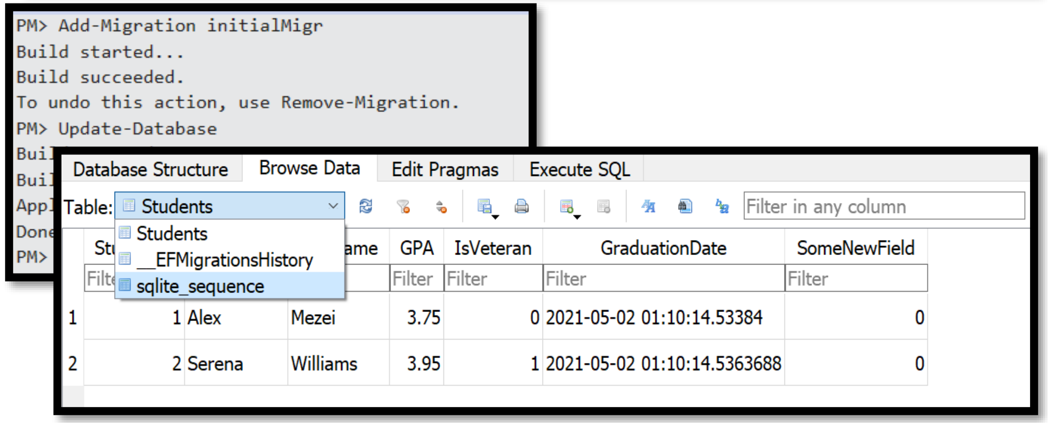
- Install: **Microsoft.EntityFrameworkCore.Tools**



Microsoft.EntityFrameworkCore.Tools

Microsoft.EntityFrameworkCore.Tools by Microsoft, **90.1M** downloads
Entity Framework Core Tools for the NuGet Package Manager Console in Visual Studio.

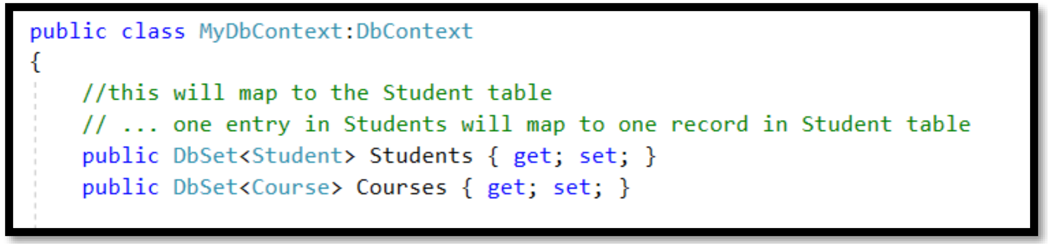
- Delete the database.
- Comment out EnsureDeleted, EnsureCreated
  - **Add-Migration** somename
  - **Update-Database**



```
PM> Add-Migration initialMigr
Build started...
Build succeeded.
To undo this action, use Remove-Migration.
PM> Update-Database
Bui
Bui
App
Done
PM>
```

	Stu		ame	GPA	IsVeteran	GraduationDate	SomeNewField
	Filte			Filter	Filter	Filter	Filter
1	1	Alex	Mezei	3.75	0	2021-05-02 01:10:14.53384	0
2	2	Serena	Williams	3.95	1	2021-05-02 01:10:14.5363688	0

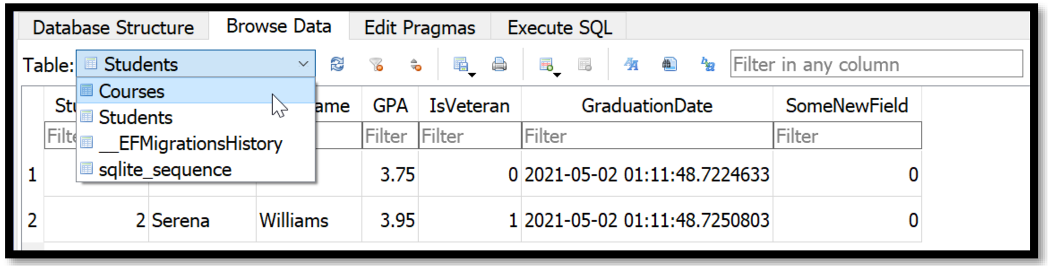
- Let's add a new **Course** class.
  - Then add a new entity to the DbContext class



```
public class MyDbContext:DbContext
{
    //this will map to the Student table
    // ... one entry in Students will map to one record in Student table
    public DbSet<Student> Students { get; set; }
    public DbSet<Course> Courses { get; set; }
```

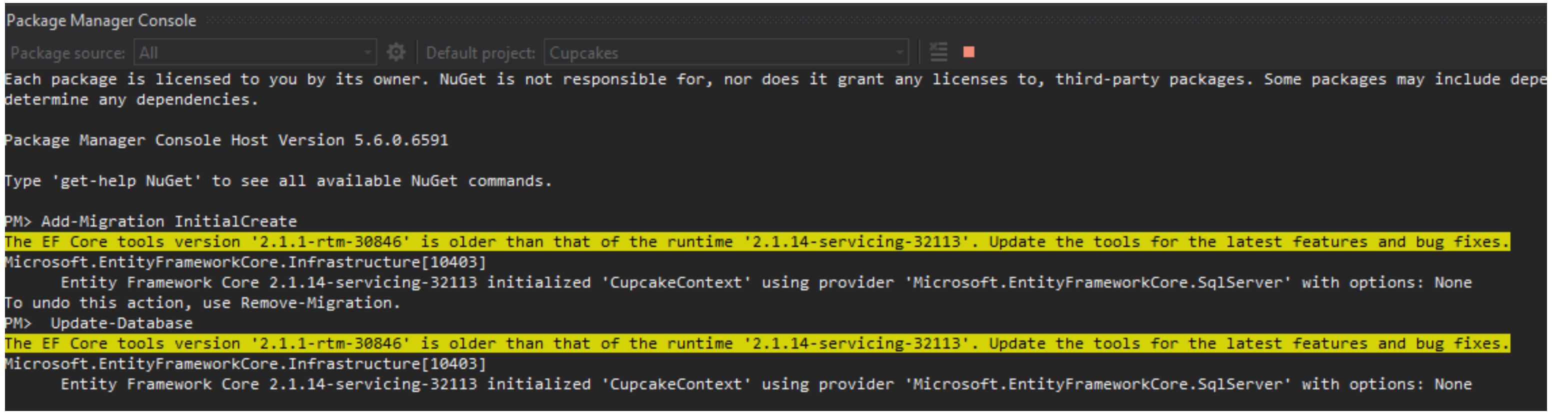
- **Add-Migration** somename2
- **Update-Database**

- The database was updated …

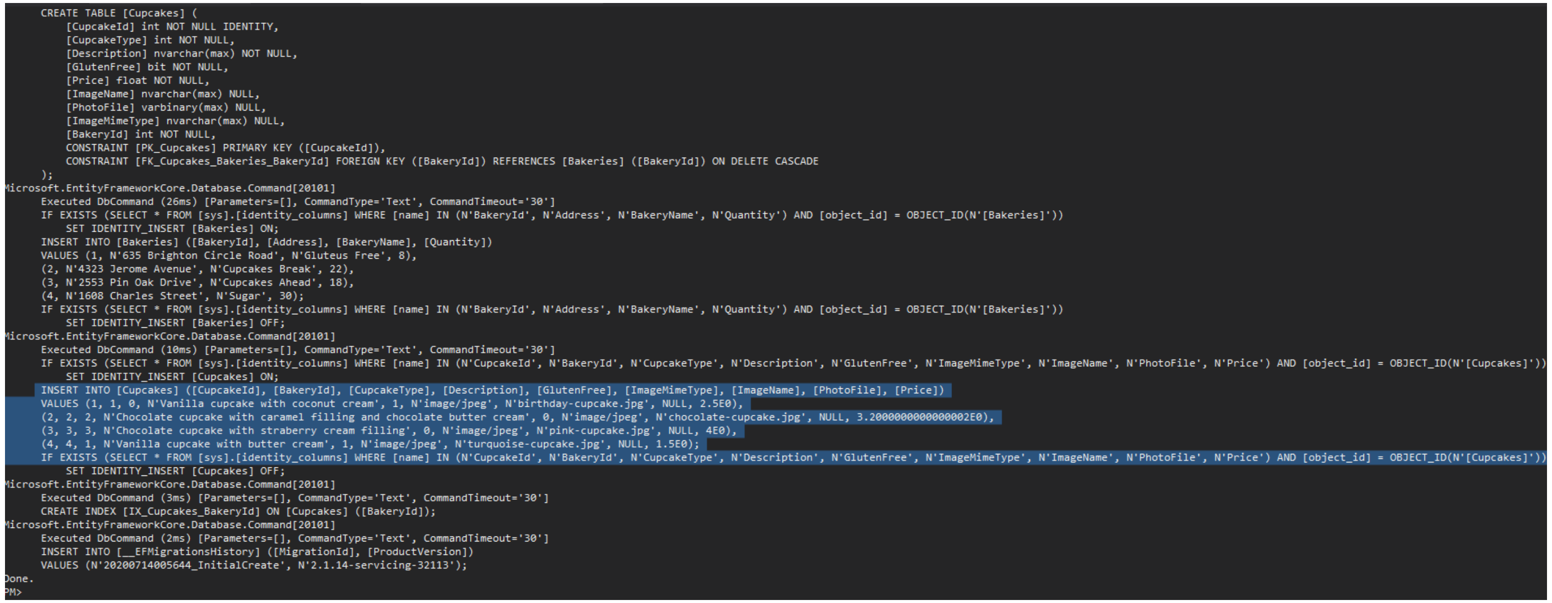


	Stu		ame	GPA	IsVeteran	GraduationDate	SomeNewField
	Filte			Filter	Filter	Filter	Filter
1				3.75	0	2021-05-02 01:11:48.7224633	0
2	2	Serena	Williams	3.95	1	2021-05-02 01:11:48.7250803	0

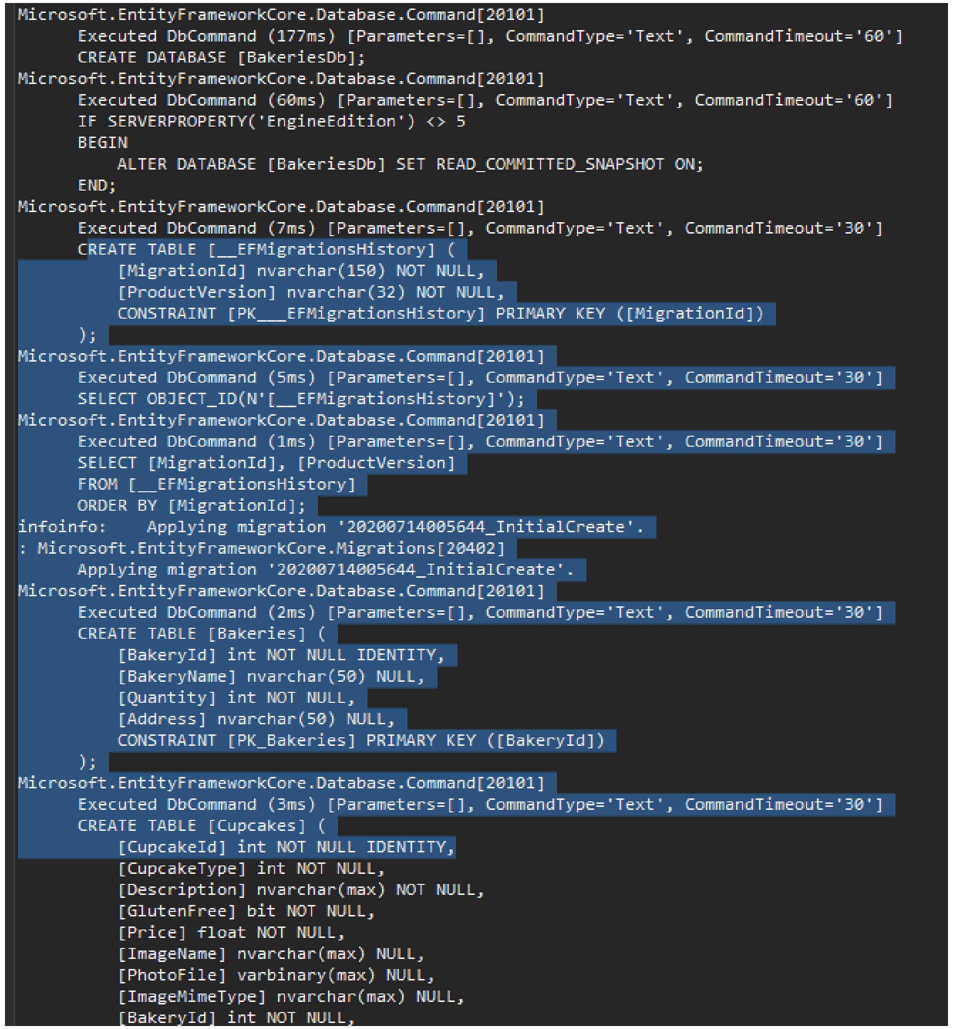
THIS IS YOUR HOMEWORK – SKIP IF NO TIME – USING MIGRATION



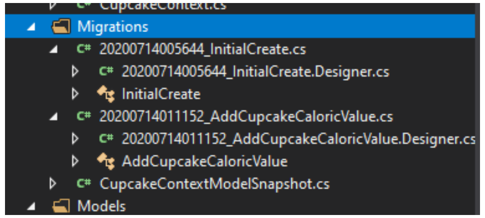
```
Each package is licensed to you by its owner. NuGet is not responsible for, nor does it grant any licenses to, third-party packages. Some packages may include depe
determine any dependencies.

Package Manager Console Host Version 5.6.0.6591

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

PM> Add-Migration InitialCreate
The EF Core tools version '2.1.1-rtm-30846' is older than that of the runtime '2.1.14-servicing-32113'. Update the tools for the latest features and bug fixes.
Microsoft.EntityFrameworkCore.Infrastructure[10403]
      Entity Framework Core 2.1.14-servicing-32113 initialized 'CupcakeContext' using provider 'Microsoft.EntityFrameworkCore.SqlServer' with options: None
To undo this action, use Remove-Migration.
PM>  Update-Database
The EF Core tools version '2.1.1-rtm-30846' is older than that of the runtime '2.1.14-servicing-32113'. Update the tools for the latest features and bug fixes.
Microsoft.EntityFrameworkCore.Infrastructure[10403]
      Entity Framework Core 2.1.14-servicing-32113 initialized 'CupcakeContext' using provider 'Microsoft.EntityFrameworkCore.SqlServer' with options: None
```

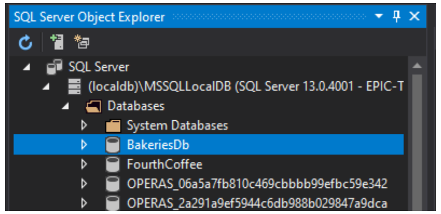


```
      CREATE TABLE [Cupcakes] (
          [CupcakeId] int NOT NULL IDENTITY,
          [CupcakeType] int NOT NULL,
          [Description] nvarchar(max) NOT NULL,
          [GlutenFree] bit NOT NULL,
          [Price] float NOT NULL,
          [ImageName] nvarchar(max) NULL,
          [PhotoFile] varbinary(max) NULL,
          [ImageMimeType] nvarchar(max) NULL,
          [BakeryId] int NOT NULL,
          CONSTRAINT [PK_Cupcakes] PRIMARY KEY ([CupcakeId]),
          CONSTRAINT [FK_Cupcakes_Bakeries_BakeryId] FOREIGN KEY ([BakeryId]) REFERENCES [Bakeries] ([BakeryId]) ON DELETE CASCADE
      );
Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (26ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
      IF EXISTS (SELECT * FROM [sys].[identity_columns] WHERE [name] IN (N'BakeryId', N'Address', N'BakeryName', N'Quantity') AND [object_id] = OBJECT_ID(N'[Bakeries]'))
          SET IDENTITY_INSERT [Bakeries] ON;
      INSERT INTO [Bakeries] ([BakeryId], [Address], [BakeryName], [Quantity])
      VALUES (1, N'635 Brighton Circle Road', N'Gluteus Free', 8),
      (2, N'4323 Jerome Avenue', N'Cupcakes Break', 22),
      (3, N'2553 Pin Oak Drive', N'Cupcakes Ahead', 18),
      (4, N'1608 Charles Street', N'Sugar', 30);
      IF EXISTS (SELECT * FROM [sys].[identity_columns] WHERE [name] IN (N'BakeryId', N'Address', N'BakeryName', N'Quantity') AND [object_id] = OBJECT_ID(N'[Bakeries]'))
          SET IDENTITY_INSERT [Bakeries] OFF;
Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (10ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
      IF EXISTS (SELECT * FROM [sys].[identity_columns] WHERE [name] IN (N'CupcakeId', N'BakeryId', N'CupcakeType', N'Description', N'GlutenFree', N'ImageMimeType', N'ImageName', N'PhotoFile', N'Price') AND [object_id] = OBJECT_ID(N'[Cupcakes]'))
          SET IDENTITY_INSERT [Cupcakes] ON;
      INSERT INTO [Cupcakes] ([CupcakeId], [BakeryId], [CupcakeType], [Description], [GlutenFree], [ImageMimeType], [ImageName], [PhotoFile], [Price])
      VALUES (1, 1, 0, N'Vanilla cupcake with coconut cream', 1, N'image/jpeg', N'birthday-cupcake.jpg', NULL, 2.5E0),
      (2, 2, 2, N'Chocolate cupcake with caramel filling and chocolate butter cream', 0, N'image/jpeg', N'chocolate-cupcake.jpg', NULL, 3.2000000000000002E0),
      (3, 3, 3, N'Chocolate cupcake with straberry cream filling', 0, N'image/jpeg', N'pink-cupcake.jpg', NULL, 4E0),
      (4, 4, 1, N'Vanilla cupcake with butter cream', 1, N'image/jpeg', N'turquoise-cupcake.jpg', NULL, 1.5E0);
      IF EXISTS (SELECT * FROM [sys].[identity_columns] WHERE [name] IN (N'CupcakeId', N'BakeryId', N'CupcakeType', N'Description', N'GlutenFree', N'ImageMimeType', N'ImageName', N'PhotoFile', N'Price') AND [object_id] = OBJECT_ID(N'[Cupcakes]'))
          SET IDENTITY_INSERT [Cupcakes] OFF;
Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (3ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
      CREATE INDEX [IX_Cupcakes_BakeryId] ON [Cupcakes] ([BakeryId]);
Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (2ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
      INSERT INTO [__EFMigrationsHistory] ([MigrationId], [ProductVersion])
      VALUES (N'20200714005644_InitialCreate', N'2.1.14-servicing-32113');
Done.
PM>
```



```
Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (177ms) [Parameters=[], CommandType='Text', CommandTimeout='60']
      CREATE DATABASE [BakeriesDb];
Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (60ms) [Parameters=[], CommandType='Text', CommandTimeout='60']
      IF SERVERPROPERTY('EngineEdition') <> 5
      BEGIN
          ALTER DATABASE [BakeriesDb] SET READ_COMMITTED_SNAPSHOT ON;
      END;
Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (7ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
      CREATE TABLE [__EFMigrationsHistory] (
          [MigrationId] nvarchar(150) NOT NULL,
          [ProductVersion] nvarchar(32) NOT NULL,
          CONSTRAINT [PK___EFMigrationsHistory] PRIMARY KEY ([MigrationId])
      );
Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (5ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
      SELECT OBJECT_ID(N'[__EFMigrationsHistory]');
Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (1ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
      SELECT [MigrationId], [ProductVersion]
      FROM [__EFMigrationsHistory]
      ORDER BY [MigrationId];
infoinfo:    Applying migration '20200714005644_InitialCreate'.
: Microsoft.EntityFrameworkCore.Migrations[20402]
      Applying migration '20200714005644_InitialCreate'.
Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (2ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
      CREATE TABLE [Bakeries] (
          [BakeryId] int NOT NULL IDENTITY,
          [BakeryName] nvarchar(50) NULL,
          [Quantity] int NOT NULL,
          [Address] nvarchar(50) NULL,
          CONSTRAINT [PK_Bakeries] PRIMARY KEY ([BakeryId])
      );
Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (3ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
      CREATE TABLE [Cupcakes] (
          [CupcakeId] int NOT NULL IDENTITY,
          [CupcakeType] int NOT NULL,
          [Description] nvarchar(max) NOT NULL,
          [GlutenFree] bit NOT NULL,
          [Price] float NOT NULL,
          [ImageName] nvarchar(max) NULL,
          [PhotoFile] varbinary(max) NULL,
          [ImageMimeType] nvarchar(max) NULL,
          [BakeryId] int NOT NULL,
```


CupcakeContext.cs
Migrations
20200714005644_InitialCreate.cs
20200714005644_InitialCreate.Designer.cs
InitialCreate
20200714011152_AddCupcakeCaloricValue.cs
20200714011152_AddCupcakeCaloricValue.Designer.cs
AddCupcakeCaloricValue
CupcakeContextModelSnapshot.cs
Models


SQL Server Object Explorer
SQL Server
(localdb)\MSSQLLocalDB (SQL Server 13.0.4001 - EPIC-T
Databases
System Databases
BakeriesDb
FourthCoffee
OPERAS_06a5a7fb810c469cbbbb99efbc59e342
OPERAS_2a291a9ef5944c6db988b029847a9dca

LAB/HOMEWORK: USING ENTITY FRAMEWORK CORE IN ASP.NET CORE

- **Module 07**
  - Exercise 1: Adding Entity Framework Core
  - Exercise 2: Use Entity Framework Core to Retrieve and Store Data
  - Exercise 3: Use Entity Framework Core to Connect to Microsoft SQL Server

If you run into HTTP 500 error, add the following to the Configure method in Startup.cs:

```
cupcakeContext.Database.EnsureCreated();
```

- You will find the **high-level** steps on the following page:

  https://github.com/MicrosoftLearning/20486D-DevelopingASPNETMVCWebApplications/blob/master/Instructions/20486D_MOD07_LAB_MANUAL.md

- You will find the **detailed** steps on the following page:

  https://github.com/MicrosoftLearning/20486D-DevelopingASPNETMVCWebApplications/blob/master/Instructions/20486D_MOD07_LAK.md

- For your homework submit one zipped folder with your complete solution.