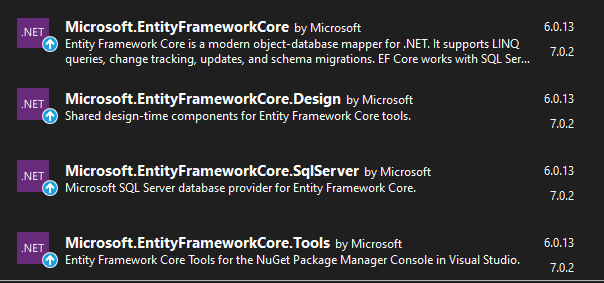
Right-click from project > Manage Nuget Packages > Install the following

Be sure to install the highest 6.0 version, do not install 7.0 versions



Create your models Model > Add Class

This is your object with its properties. Ex: Employee has an Id, Salary and Age

this is generally just a class with properties, defining fields for database

Create Folder DAL (Data Access Layer)

Create a DBContext class in this folder.

Using Microsoft.EntityFrameworkCore to be able to inherit DbContext ( : DBContext)

Using [projectname].Models (critical)

Should have 2 constructors, empty constructor must be listed first

sample of the second constructor:

       public SchoolContext(DbContextOptions options) : base(options)

        {

        }

Configuration for db goes here (boilerplate)

private static IConfigurationRoot \_configuration;

        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

        {

            if (!optionsBuilder.IsConfigured)

            {

                var builder = new ConfigurationBuilder()

                    .SetBasePath(Directory.GetCurrentDirectory())

                    .AddJsonFile("appsettings.json", optional: true, reloadOnChange: true);

                \_configuration = builder.Build();

                string cnstr = \_configuration.GetConnectionString("SchoolDb");

                optionsBuilder.UseSqlServer(cnstr);

            }

        }

    }

Add Connection string to appsettings file (don’t forget to add a comma) if there is json already in the file. The comma goes right after previous content

EmployeeDB = connection string name matches config from db Context file

EmployeeDb = (catalog) Name of db table you wish to create

{

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft.AspNetCore": "Warning"

}

},

"AllowedHosts": "\*",

"ConnectionStrings": {

"SchoolDb": "Data Source=KTG16\\SQLEXPRESS;Initial Catalog=SchoolDb;Trusted\_Connection=True;TrustServerCertificate=True;"

}

}

This might be needed in two places.

time to set up the database for the migration:

Go into the appropriate context file (separate folder and context file for each database)

Table(s) Name (1 set per table you are creating)

public DbSet<name of model class with table schema> DatabaseName {get; set;}

Build solution and fix any errors

Package Manager Console

add-migration initial

update-database

From SSMS Run SQL statement to add data to the database

INSERT INTO [dbo].[Students]

           ([LastName]   ,[FirstMidName]    ,[EnrollmentDate])

     VALUES

           ('Tremaine','Scott','01/01/2003'),

  ('Smith','John','12/13/2014'),

('Frank','Mary','06/07/2008'),

  ('Stevens','Steve','10/10/2010')

INSERT INTO [dbo].[Courses]

           ([CourseID],[Title],[Credits])

     VALUES

           (1001,'CSharp',4),

(2001,'SQL Server',4),

(3001,'HTML',2)

INSERT INTO [dbo].[Enrollments]

           ([CourseID],[StudentID],[Grade])

     VALUES

           (1001,1,1),

(2001,1,0),

(3001,1,3),

(1001,2,3),

(2001,2,2),

(3001,2,0),

(1001,3,4),

(2001,3,4),

(2001,4,4)

Add View – this is what the user sees

Can use empty razor or razor template depending on what you need to display

update the edit/delete/details links with the primary key for whatever model for the view

Now that the models and views are set up, it’s time to provide the application access to the database.

We do that in the controller. We will need to add a using state to the dataAccessLayer in each one of the controllers in our application.

Update Controller – When this action happens, what view do you want the user to see?

Think of the dbcontext as the connection between the application and the database

add database context reference

using Solution.<DataAccessLayerFolderName/ObjectName>;

create a new database object for the controller object:

private <name of Context object> db = new <objectname>();

lActionResult (Index or whatever the view file for that object is called)

view(db.DatabaseName.ToList()). (just a list)

details(

DETAILS view:

right click your custom vies folder, create a view called "details" or whatever,

template is details – source will be whatever model you'd like to use to show details.

A details page is only displaying one record. But you could add additional html code to the details page to display data related to that record. For example, for a course, you could display all the students enrolled in that course like this:

<dd>

            <table class="table">

                <tr>

                    <th>Student First Name</th>

                    <th>Student Last Name</th>

                    <th>Grade</th>

                </tr>

                @foreach (var item in Model.Enrollments)

                {

                    <tr>

                        <td>

                            @Html.DisplayFor(modelItem => item.Student.FirstMidName)

                        </td>

                        <td>

                            @Html.DisplayFor(modelItem => item.Student.LastName)

                        </td>

                        <td>

                            @Html.DisplayFor(modelItem => item.Grade)

                        </td>

                    </tr>

                }

            </table>

        </dd>

Update Navigation under Shared Layout

If you want your page to be a separate link on the navigation bar be sure to add it here