

CST8256 Web Programming Language I

Lab 4

Objective

1. Use Visual Studio SQL Server Object Explorer to manage SQL Server databases
2. Use Entity Framework Core to generate entity classes and DB Context class.
3. Generate CRUD data accessing Razor pages to an entity class

Due Date

See Brightspace posting for the due date. To earn 5 points, you are required:

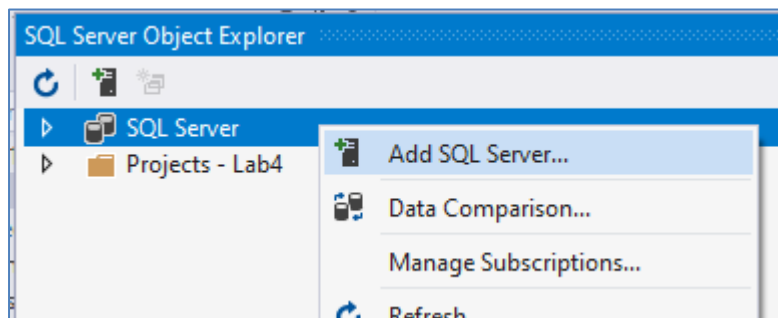
1. Complete the lab as required.
2. Zip your **web application** project folder and submit the zip file to the Brightspace before due date.
3. Demonstrate your lab work during the lab session after the due date.

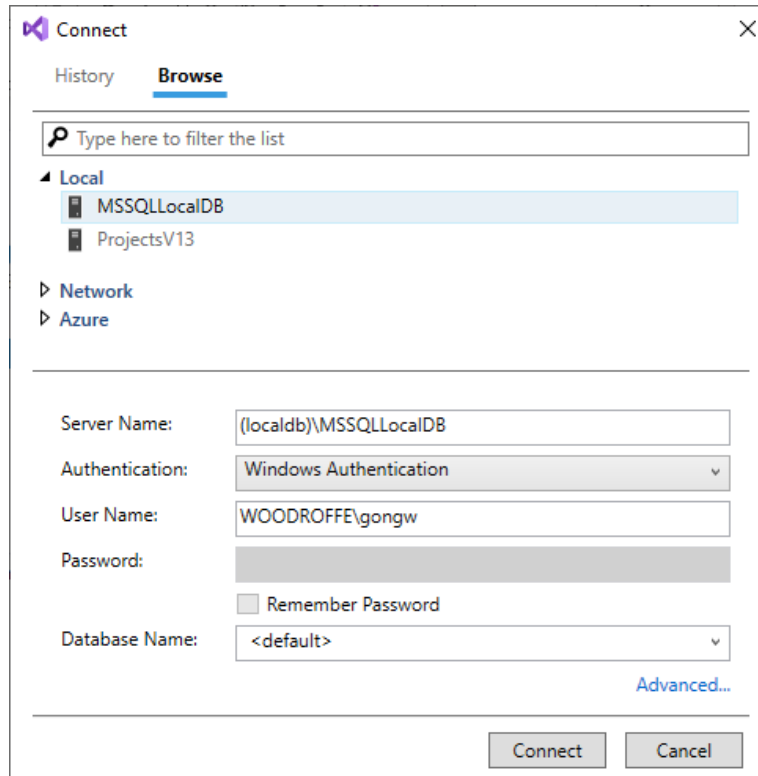
Requirements

1. **Create a new ASP.NET Razor Page web application project Lab4.**
2. **Configure and Manage SQL Server with Visual Studio**

As an IDE, Visual Studio comes with all functionality/tools for developers to work with databases from different vendors, by default, Microsoft SQL Server, of course.

Likely, you may already have one or more instances of MS SQL Server installed on your machine. To find out the details of these SQL Server instances, you can use Visual Studio's **SQL Server Object Browser** (select menu item "View > SQL Server Object Explorer")

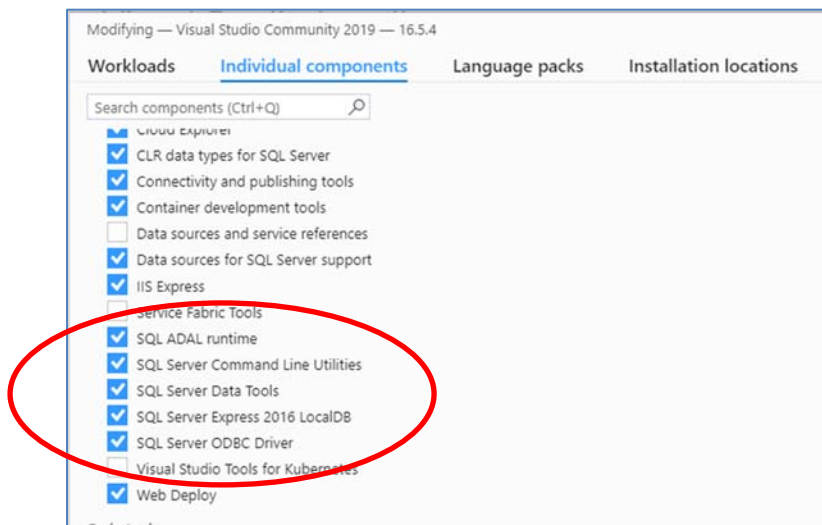




Note: In my case, there are two SQL Server instances running on my computer:

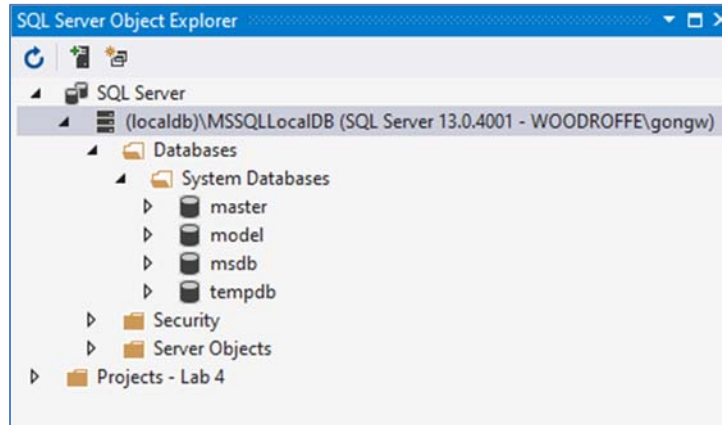
- **MSSQLLocalDB** came with Visual Studio 2022.
- **ProjectsV13** came with the previous version of Visual Studio

If you do not have any of above instances, you need to re-run Visual Studio installation program to modify features to add components as follows:

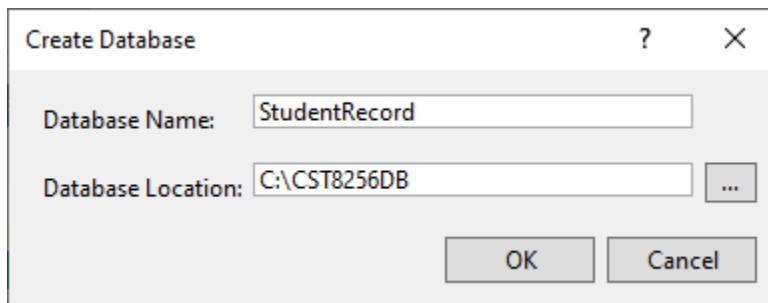
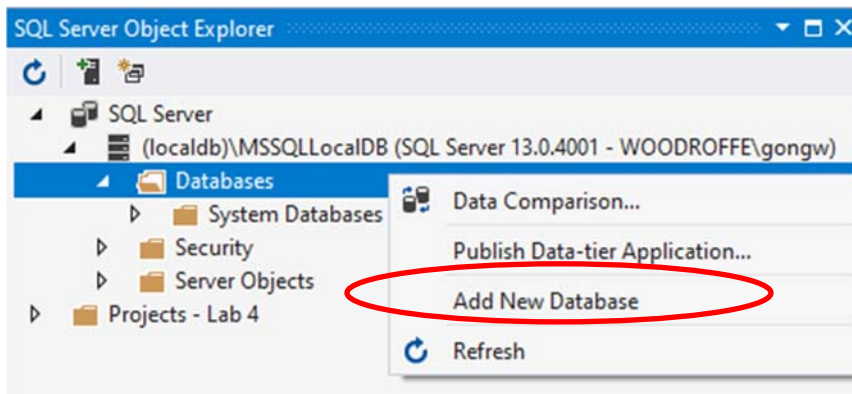


MSSQLLocalDB is a version of SQL Server for use by developers. You should use this one for this and all remaining labs in this course.

After adding SQL Servers, your SQL Server Object Explorer should look similar to the following:



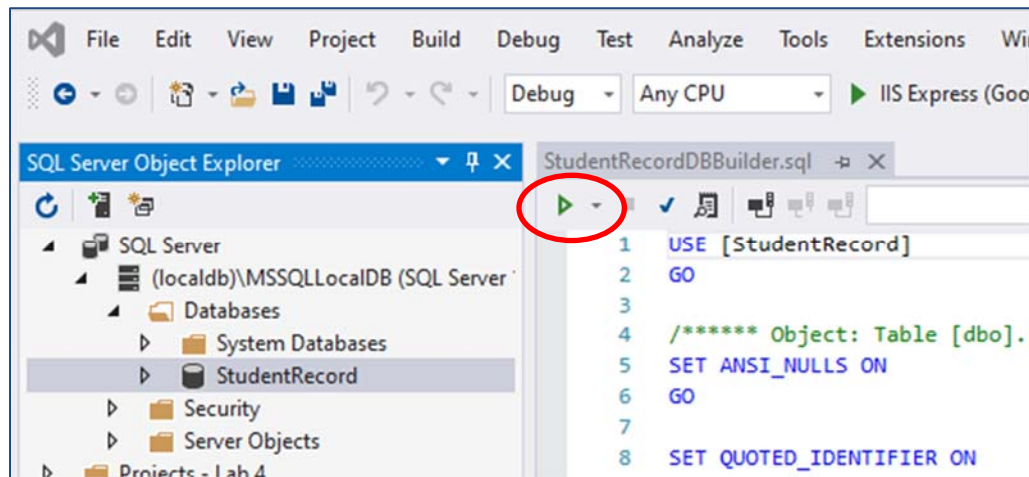
3. Create a new database StudentRecord



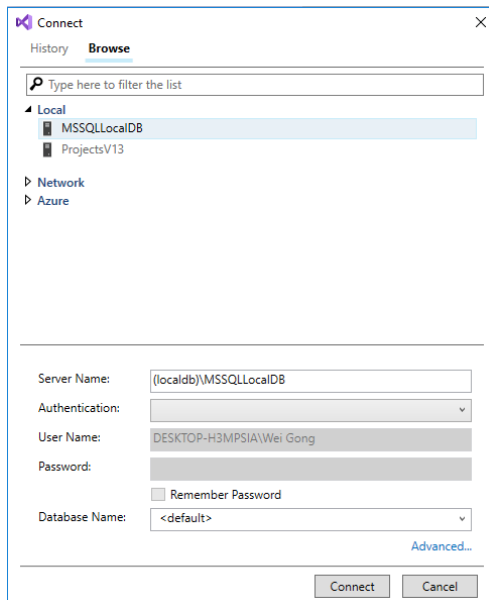
You can put the new database anywhere you want, for example, C:\CST8256DB

Once the database is created, run **StudentRecordDBBuilder.sql** (downloadable from the Brightspace) by selecting menu **File > Open ...** to open the file and then click the run button to create tables for use in lab 4.

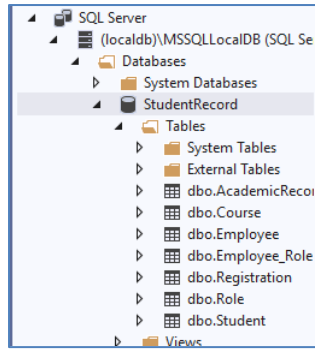
You may receive a couple of error messages when you run this SQL script the first time. They come from dropping tables not existing yet. You can ignore these error messages. Once the tables are created in the database, you will not receive these error when running the script again.



If prompted, select the database server you want to use to connect to and click **Connect** button.

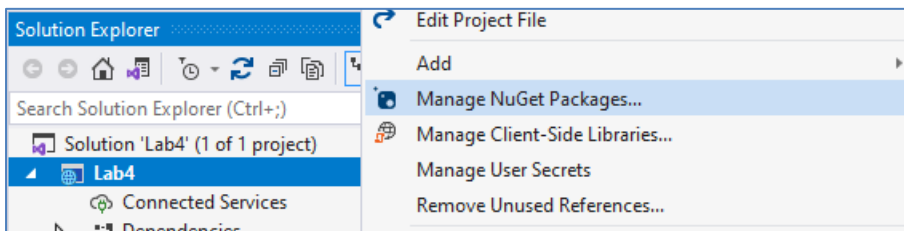


Expand the Tables folder of **StudentRecord**, you should see the following tables (not all tables will be used in this lab):



4. Install Microsoft Entity Framework Core

Entity Framework is a set of class libraries available on NuGet public domain you can install to the web application project using NuGet Package Management window.

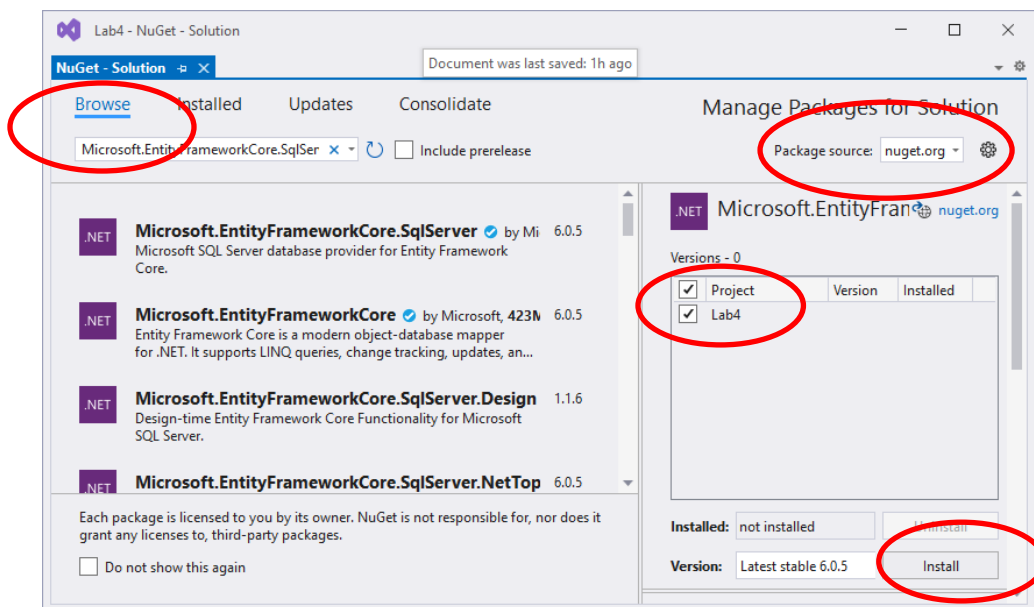


Select **Browse** tab, **nuget.org** as package source. Search and install the following three packages to the project:

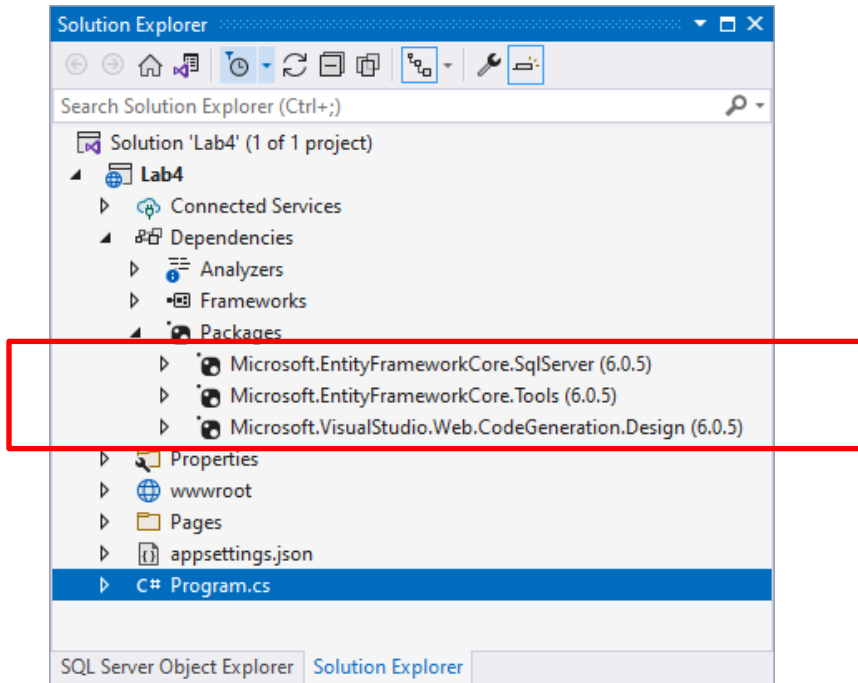
Microsoft.EntityFrameworkCore.SqlServer

Microsoft.EntityFrameworkCore.Tools

Microsoft.VisualStudio.Web.CodeGeneration.Design



To check if the required packages are installed correctly, expand the project's Dependencies you should see these packages listed under Packages.



5. Generate the code from the database StudentRecord

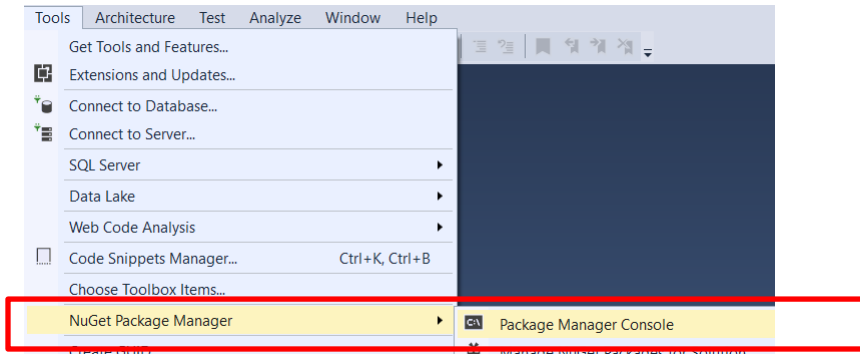
EF Tool **Scaffold-DbContext** is for generating classes from tables. It also generates a DB Context class for accessing the database.

Scaffold-DbContext runs inside Visual Studio using NuGet's Package Manager Console. It takes several parameters. The following four are the most used ones. **Connection** and **Provider** are mandatory and do not have to lead by the keywords Connection and Provider. Others are the optional.

Parameters:	
-Connection <String>	The connection string to the database.
-Provider <String>	The code generator, e.g. Microsoft.EntityFrameworkCore.SqlServer when using Microsoft SQL Server database.
-OutputDir <String>	The fold to put generated files. Paths are relative to the project fold
-Tables <String[]>	The tables (separate by comma) to generate entities for.

Create a folder **DataAccess** in project to contain the generated data access code.
This lab will only use data in **Student**, **Course** and **AcademicRecord** tables.

Start NuGet Package Manager Console:

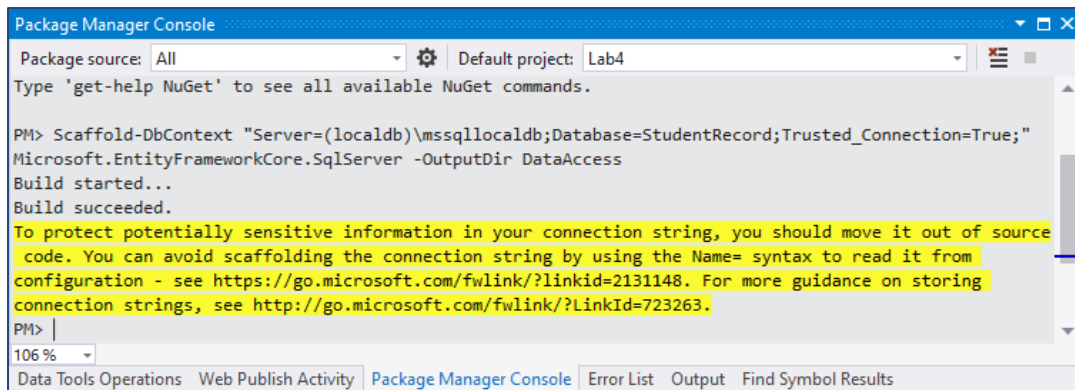


Enter the following command at the prompt:

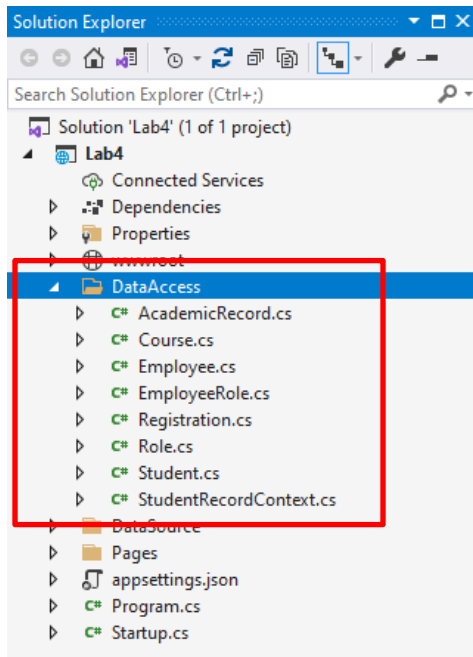
Scaffold-DbContext

```
"Server=(localdb)\mssqllocaldb;Database=StudentRecord;Trusted_Connection=True;  
" Microsoft.EntityFrameworkCore.SqlServer -OutputDir DataAccess
```

Note: DO NOT copy and paste the above line. The invisible Adobe formatting characters embedded in the line will cause the command to fail.



Entity Framework also generated a set of files in the project:



Review the contents of generated files, pay attention to the followings:

- The namespace the generated code reside in. To use the generated classes, you must include (using) this namespace at the top of your .cs and cshtml files.
- The name of the generated Context class (inherit from **DbContext** class). All instances of generated entity classes can be accessed via an instance of this context class.

6. Remove hardcoded connection string

The generated **StudentRecordcontext.cs** has the hardcoded DB connection string in it as show below:

```
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
{
    if (!optionsBuilder.IsConfigured)
    {
        #warning To protect potentially sensitive information in your connection string, you should move it out of source code. See http://go.microsoft.com/fwlink/?LinkID=527312 for more details.
        optionsBuilder.UseSqlServer(@"Server=(localdb)\mssqllocaldb;Database=StudentRecord;Trusted_Connection=True;");
    }
}
```

Instead, applications should acquire DB connection string from the application's configuration file **appsetting.json**. Add **ConnectionStrings** section with one connection string **StudentRecord** to the file as shown below:


```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "StudentRecord": "Server=(localdb)\\mssqllocaldb;Database=StudentRecord;Trusted_Connection=True;"
  }
}
```

Now, you can remove the hardcoded connection string from StudentRecordDbContext class

```
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
{
    if (!optionsBuilder.IsConfigured)
    {
        optionsBuilder.UseSqlServer();
    }
}
```

7. Register the DB Access Service with the Application

To an ASP.NET Razor Page web application, the generated DB Accessing is a service. To use this service, you must let the application know about it. This is accomplished during the application's startup phase. Add the following using statement to the top of **Program.cs** file

```
using Microsoft.EntityFrameworkCore;
using Lab4.DataAccess;
```

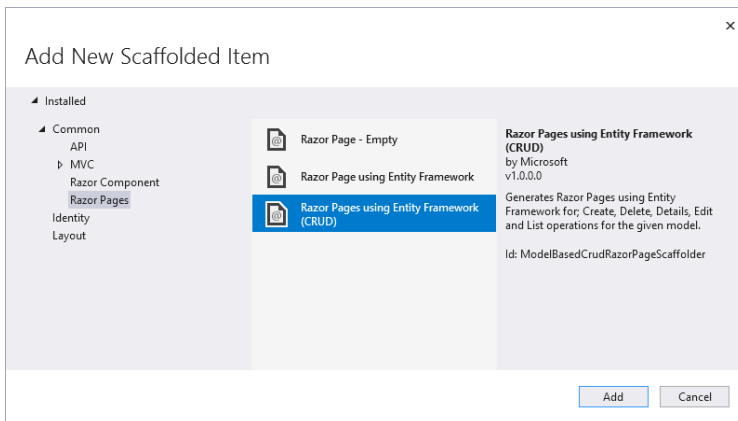
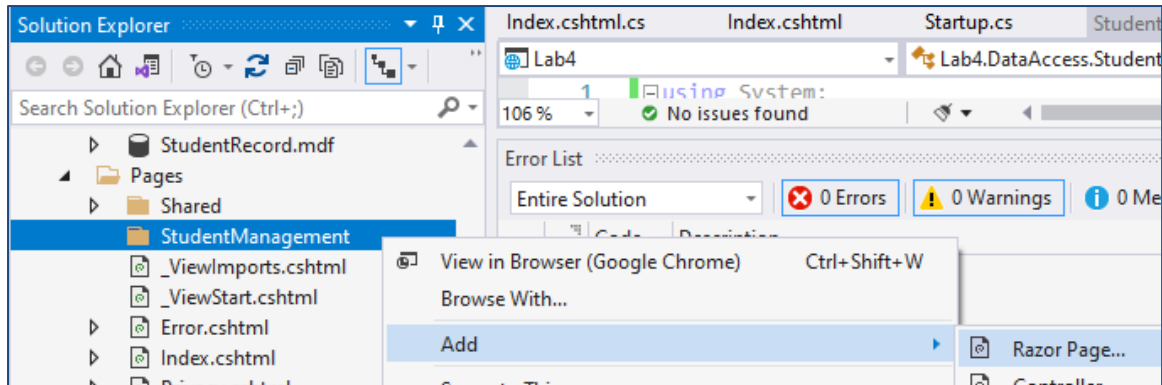
And add line 9 and line10 after builder.Services.AddRazorPages().

```
1 using Microsoft.EntityFrameworkCore;
2 using Lab4.DataAccess;
3
4 var builder = WebApplication.CreateBuilder(args);
5
6 // Add services to the container.
7 builder.Services.AddRazorPages();
8
9 string dbConnStr = builder.Configuration.GetConnectionString("StudentRecord");
10 builder.Services.AddDbContext<StudentRecordContext>(options => options.UseSqlServer(dbConnStr));
11
12 var app = builder.Build();
13
```

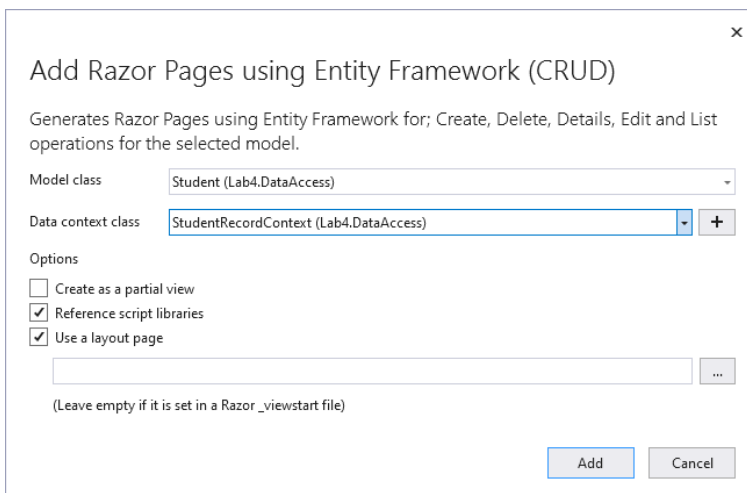
Rebuild Lab4 solution, make sure the solution build successfully.

8. Generate Razor pages for entity CRUD

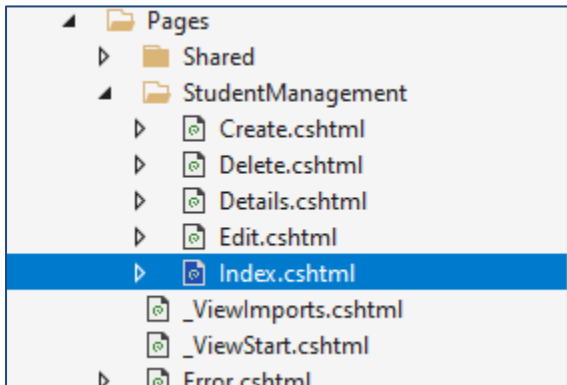
Add a **StudentManagement** folder inside Pages folder, add new Razor pages inside the folder:



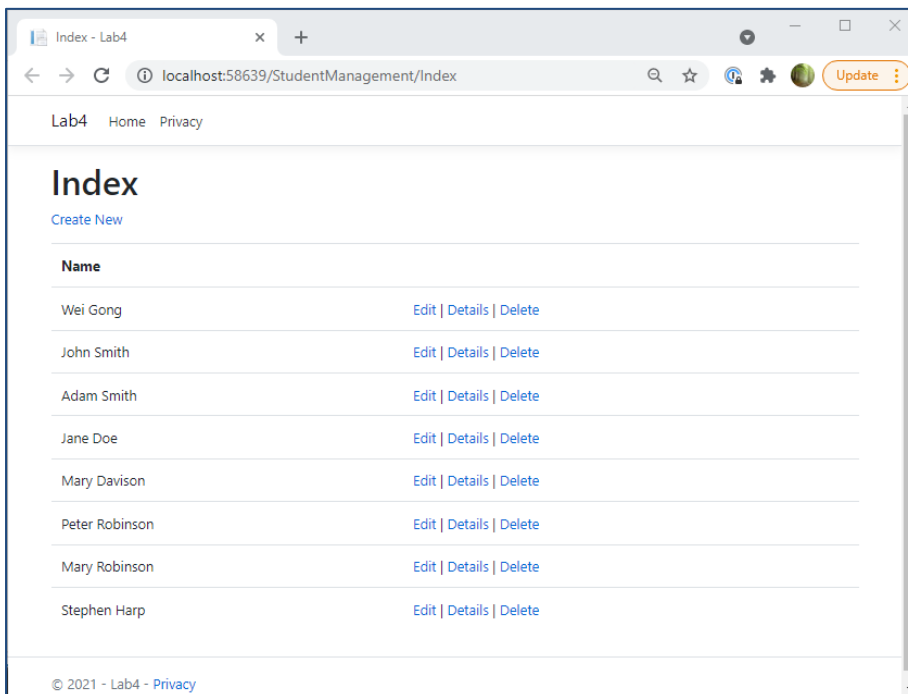
Select the EF generated Student class (inside Lab4.DataAccess namespace) as model class and the generated StudentRecordContext (also inside Lab4.DataAccess namespace) as Data context class.



Visual Studio generates Razor pages inside **StudentManagement** folder for fully functional CRUD accessing of database's Student table.

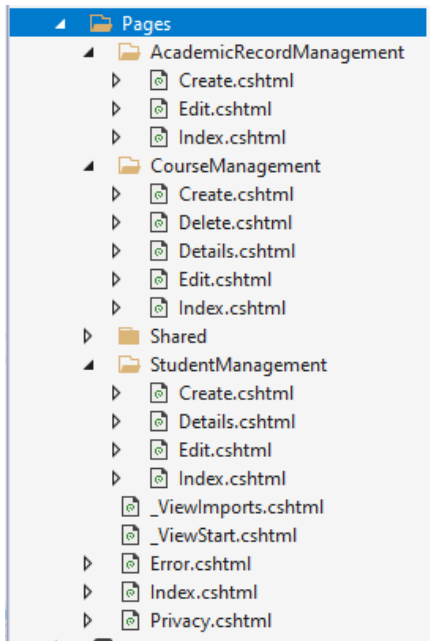


View the **StudentManagement/Index.cshtml** in browser:

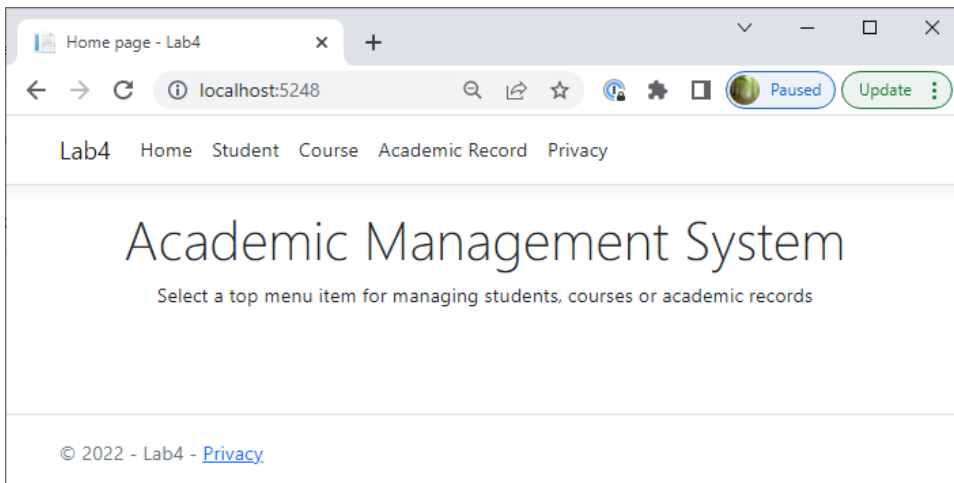


Try the links on the page for create new student, editing existing student, etc. Review and familiarize yourself with the generated code.

Similarly, generate the CRUD Razor pages for **Course** and **AcademicRecord**, in their respective folders. After completion, the Pages folder of the lab's solution should look like:



Add menu items **Student**, **Course**, and **Academic Record** linking to `_Layout.cshtml` page in their corresponding folders. The landing page of the application should look like:

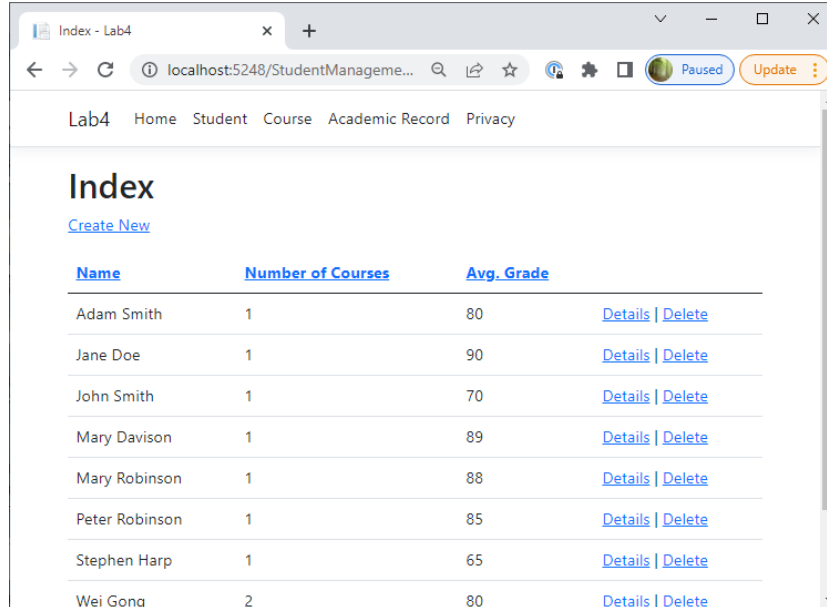


9. Razor pages for managing students

- **StudentManagement/Index**

Usually, the index page for each entity should provide some high-level summary about the entity in concern. Modify the generated **StudentManagement/Index** page such that it also shows the number of courses each student took and the average grade the student achieved. The column head should be clickable for sorting the students by the value of the clicked column, respectively.

Also remove the Edit links, there is nothing to edit in our case.



Index - Lab4

localhost:5248/StudentManageme...

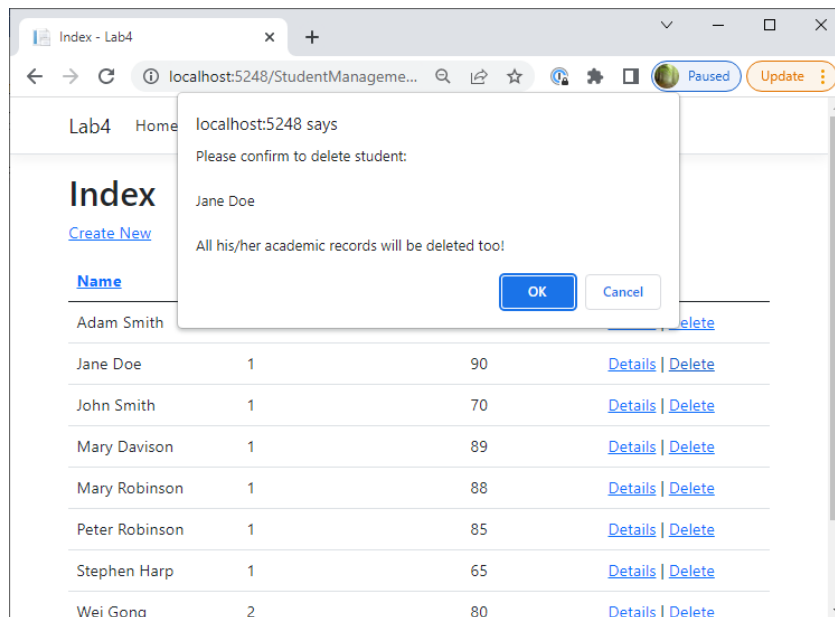
Lab4 Home Student Course Academic Record Privacy

Index

[Create New](#)

Name	Number of Courses	Avg. Grade	
Adam Smith	1	80	Details Delete
Jane Doe	1	90	Details Delete
John Smith	1	70	Details Delete
Mary Davison	1	89	Details Delete
Mary Robinson	1	88	Details Delete
Peter Robinson	1	85	Details Delete
Stephen Harp	1	65	Details Delete
Wei Gong	2	80	Details Delete

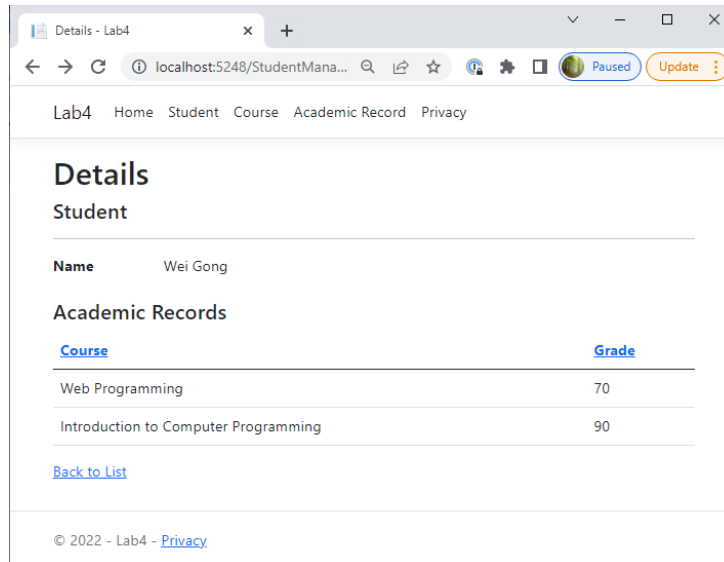
The generated **Delete** link leads the user to the **Delete** page to get the confirmation from the user. To avoid unnecessary roundtrips to the server-side, the confirmation prompts should happen at the client-side. Change the behavior of the Delete link such that when the user clicks the link, a client-side prompt dialog box is shown to get the confirmation from the user, as shown below:



If the user confirms the deletion, the student and all his/her academic records will be deleted (**Note:** you must delete the associated academic records first, and then delete the student. Otherwise, a DB referential integrity error will happen)

- **StudentManagement/Detail**

Besides the entity's immediate properties, usually more details of the related entities should be shown on this page. Modify **StudentManagement/Detail** page to show the student's academic records as well as the name of the student, as follows:



- **StudentManagement/Create**

When saving a new entity into DB, the generated code does not check for the DB's primary key constrain. Add the validation to the page's model class (StudentManagement/Create.cshtml.cs file) such that an error message is displayed if the entered student id already exists in the database Student table as shown below:

