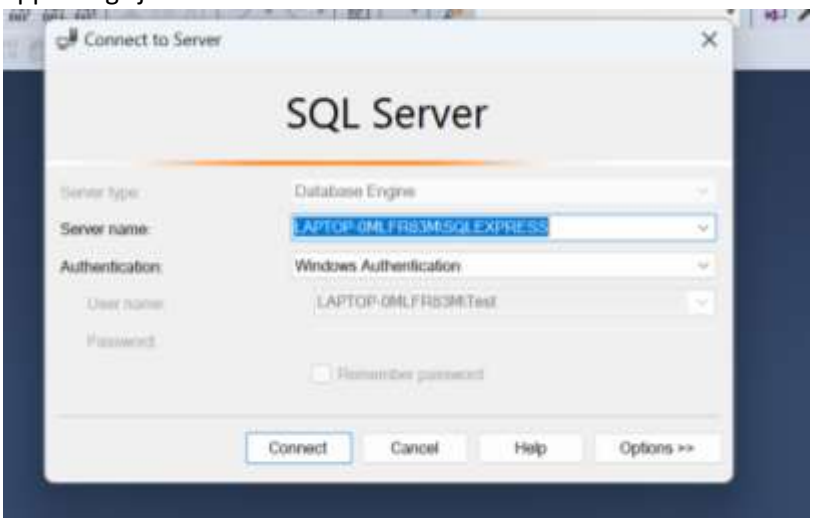
CRUD Operations on Student table

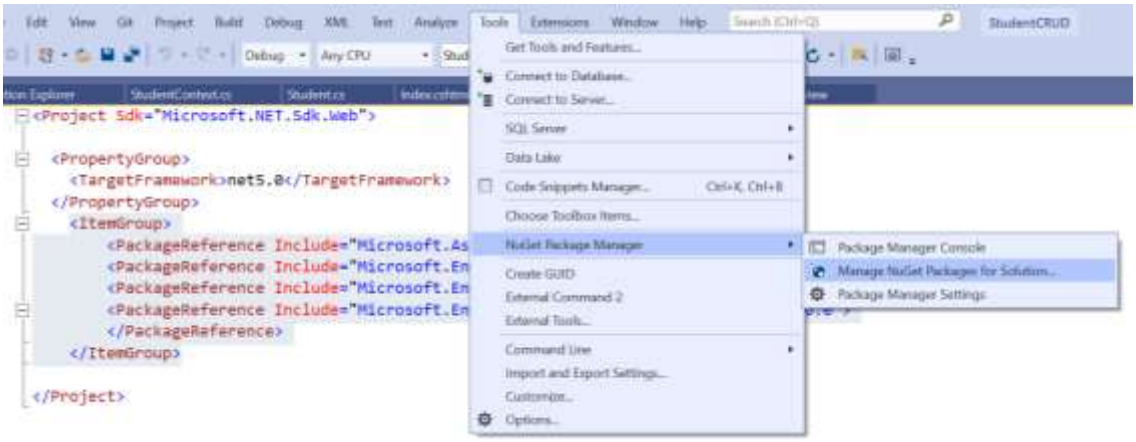
1. Download sql server express from: https://www.microsoft.com/en-us/download/details.aspx?id=101064 and install it with basic setup.
2. Once Sql server installation is complete, it will ask you to install MSSMS, this is an ide to work with MSSql.
3. Now run MSSMS to see the server's name, which is needed to setup the connection strings in appsetings.json file.



4. Create a new Asp.net core Web app in VS to work with coding part.
5. You need to include the following packages to work with EF and migration tools:

```
<ItemGroup>
            <PackageReference
Include="Microsoft.AspNetCore.Mvc.Razor.RuntimeCompilation" Version="3.0.0" />
            <PackageReference Include="Microsoft.EntityFrameworkCore"
Version="3.0.0" />
            <PackageReference Include="Microsoft.EntityFrameworkCore.SqlServer"
Version="3.0.0" />
            <PackageReference Include="Microsoft.EntityFrameworkCore.Tools"
Version="3.0.0">
            </PackageReference>
</ItemGroup>

You can either copy paste these reference to .csproj file and build the project or
install these packages via tools>Nuget Package Manager and search for the packages
and install them one by one.
```



Now you are ready to code the models and DB context class.

1. Add a new folder named Models in the project and add two classes inside this folder, one for the table and another for the Dbcontext.

```
namespace StudentCRUD.Models
{
    public class Student
    {
        public int ID { get; set; }
        public string Name { get; set; }
        public int RollNo { get; set; }
        public string ClassName { get; set; }
    }
}
```

```
namespace StudentCRUD.Models
{
    public class StudentContext:DbContext
    {
        public StudentContext(DbContextOptions<StudentContext> options) : base(options)
        {

        }

        public DbSet<Student> Students { get; set; }
    }
}
```

2. Now, we need to write a connection string in appsettings.json file for database creation and connection.

```
"ConnectionStrings": {

    "DefaultConnection": "Server=LAPTOP-
0MLFR83M\\SQLEXPRESS;Database=StudentDb;Trusted_Connection=True;MultipleActiveResultSets=
True"

  },
```

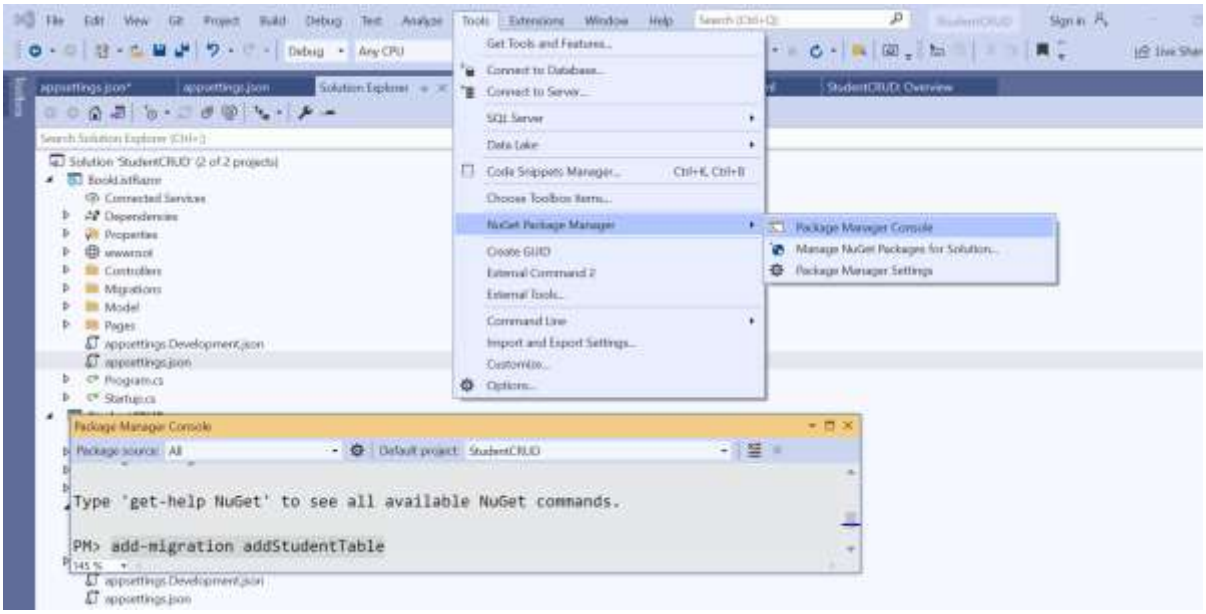
In my laptop the server name is: LAPTOP-0MLFR83M\\SQLEXPRESS, and it can vary in your case. You need to write the right server name to connect to the database.

3. Goto Startup.cs file and add the new database services as below:

```
public void ConfigureServices(IServiceCollection services)
            {
           services.AddDbContext<StudentContext>(option =>
           option.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));

            services.AddRazorPages();
                  }
```

4. After this, we need to go to Nuget Manager Console and run the following command: *add-migration addStudent*



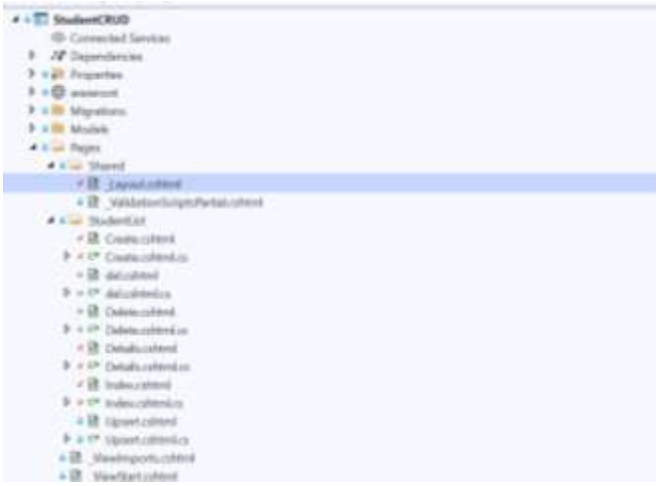
After successful run of this migration, it will add a new folder named Migration in the project and the Sql script is written in 20230616014249_addStudent.cs file, you can see the code there.

5. Finally, to create the database and table we need to invoke the *update-database* command on the console.

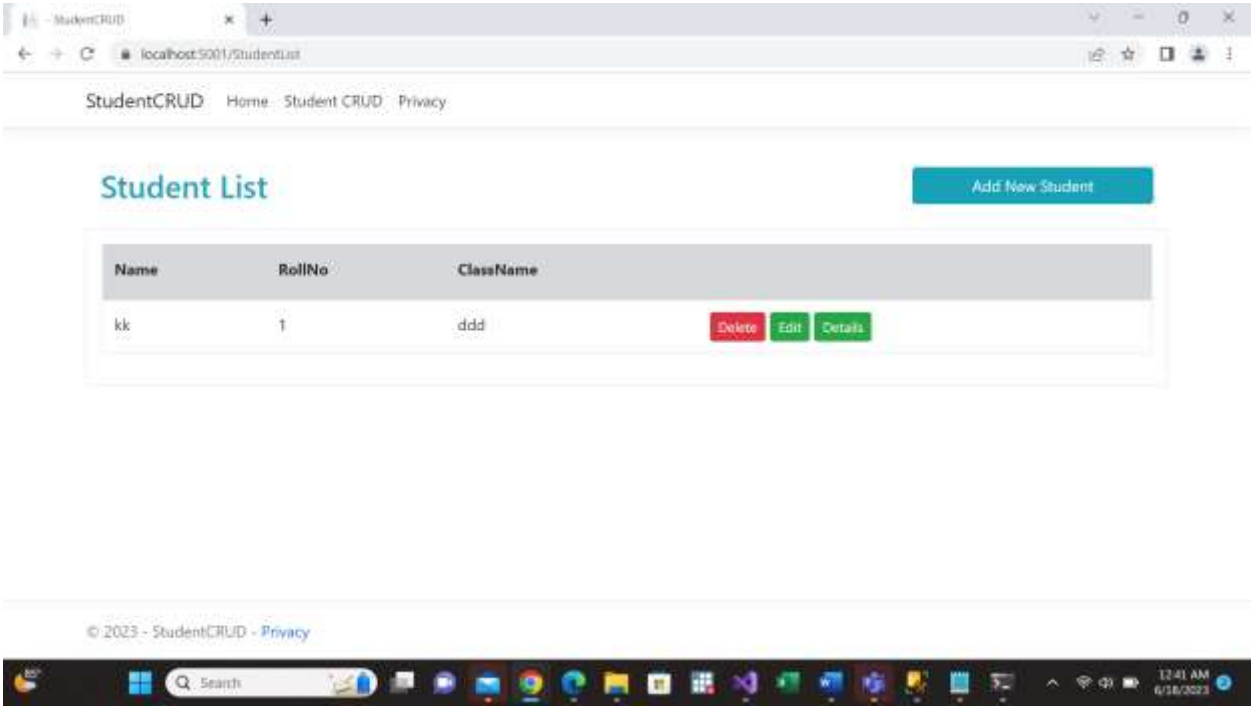
Goto MSSMS and refresh the databases, you can see the newly created DB and the Table named Students.

Now, we can write codes for CRUD operation on Students Table:

For that, I created a separate folder within Pages folder to place all the views and models for CRUD operation. The folder name is StudentList, the file structure is as below:



First, I created Index razor page, by right click on the StudentList folder and choosing to Add a Razor page. This way, we get both Index.cshtml, and Index.cshtml.cs files created with minimum razor template. In the Index view, we want to see the following UI.



We will see the code and discuss all the methods and views.

A **page handler** is a method that runs in response to a request. It's also worth bearing in mind that page handlers should generally not be performing business logic directly. Instead, they should call appropriate services in the application model to handle requests. If a page handler receives a request to add a product to a user's cart, it shouldn't directly manipulate the database or recalculate cart totals, for example. Instead, it should make a call to another class to handle the details. This approach of separating concerns ensures your code stays testable and maintainable as it grows.

Model binding

The process of extracting values from a request and converting them to .NET types is called **model binding**. ASP.NET Core can bind two different targets in Razor Pages:

1. Method arguments—If a page handler has method arguments, the values from the request are used to create the required parameters.
2. Properties marked with a [BindProperty] attribute—Any properties marked with the attribute will be bound. By default, this attribute does nothing for GET requests.

What is routing?

Routing in ASP.NET Core is the process of mapping an incoming HTTP request to a specific handler. In Razor Pages, the handler is a page handler method in a Razor Page. In MVC, the handler is an action method in a controller. You can use routing to control the URLs you expose in your application. You can also use routing to enable powerful features like mapping multiple URLs to the same Razor Page and automatically extracting data from a request's URL.

TIP Routing is not case sensitive, so the request URL does not need to have the same URL casing as the route template to match.

e.g.

Requests to the URL /products/view match the route template "Products/View", which in turn corresponds to the View.cshtml Razor Page. The RoutingMiddleware selects the View.cshtml Razor Page as the endpoint for the request, and the EndpointMiddleware executes the page's handler once the request reaches it in the middleware pipeline.

DEFINITION The query string is part of a URL that contains additional data that doesn't fit in the path. It isn't used by the routing infrastructure for identifying which action to execute, but it can be used for model binding.