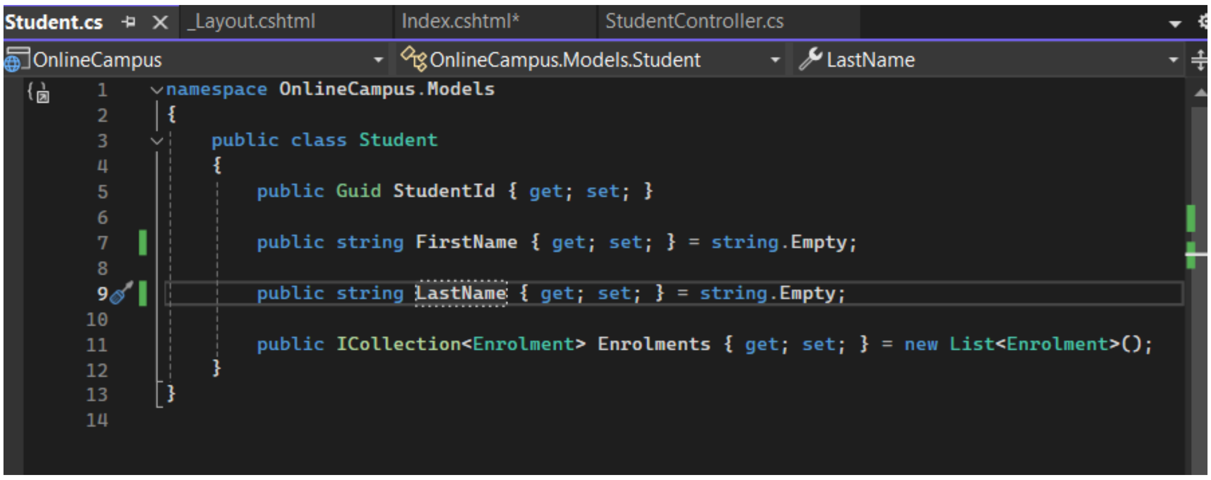
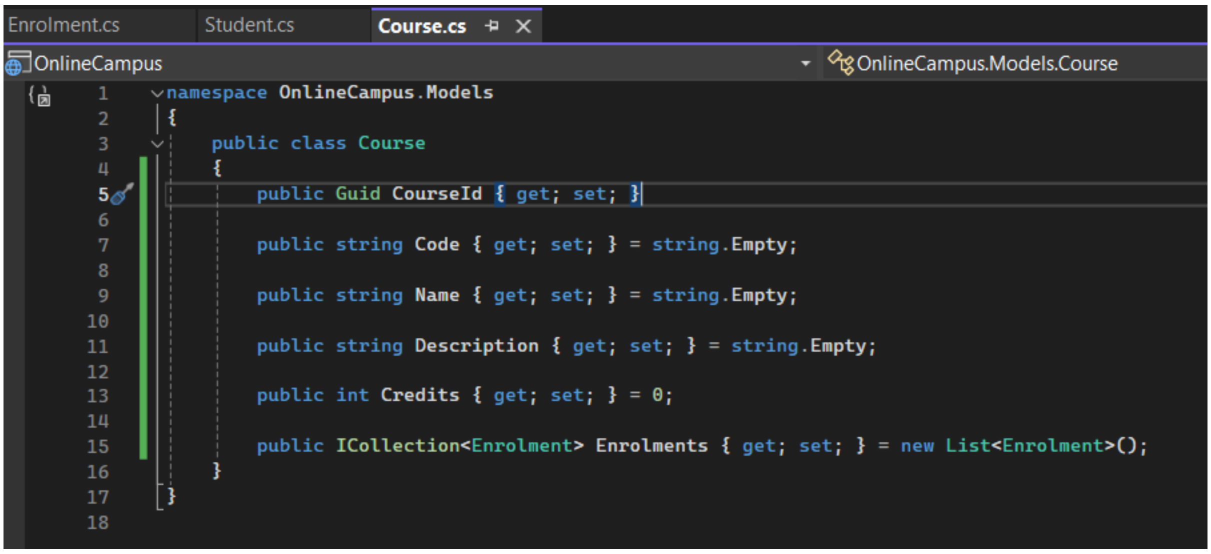
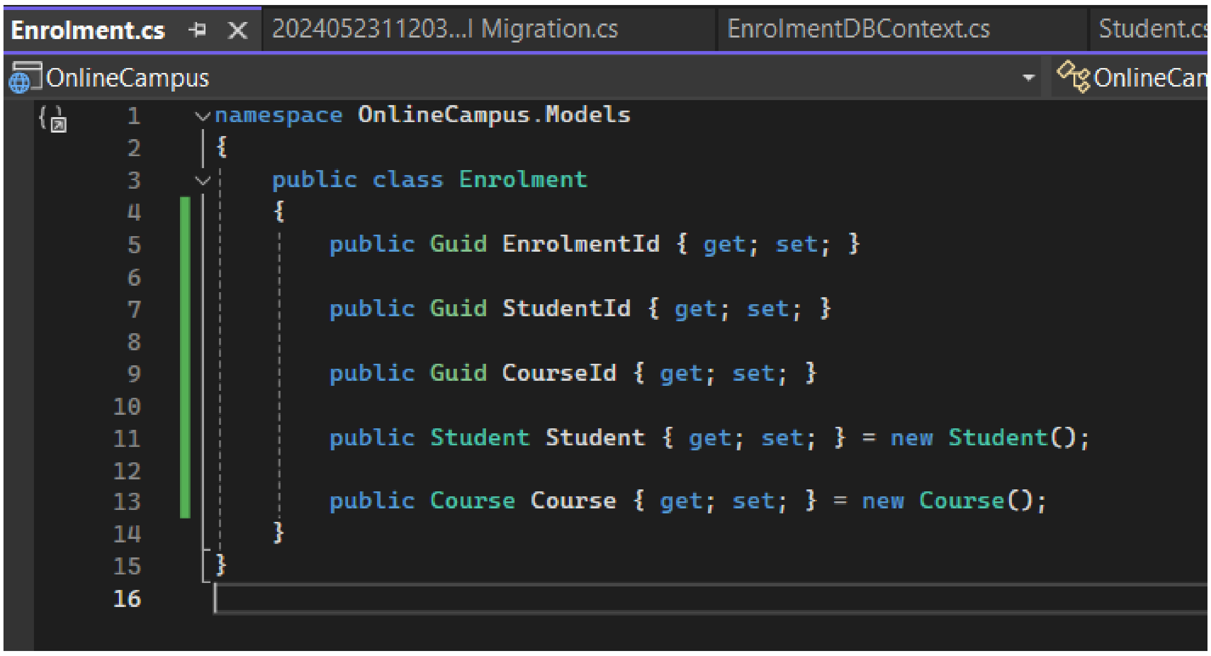
1. Create classes in the model folder for the Student, Course and Enrolment Entity Sets.
    a. Add properties to each class.



```
namespace OnlineCampus.Models
{
    public class Student
    {
        public Guid StudentId { get; set; }

        public string FirstName { get; set; } = string.Empty;

        public string LastName { get; set; } = string.Empty;

        public ICollection<Enrolment> Enrolments { get; set; } = new List<Enrolment>();
    }
}
```

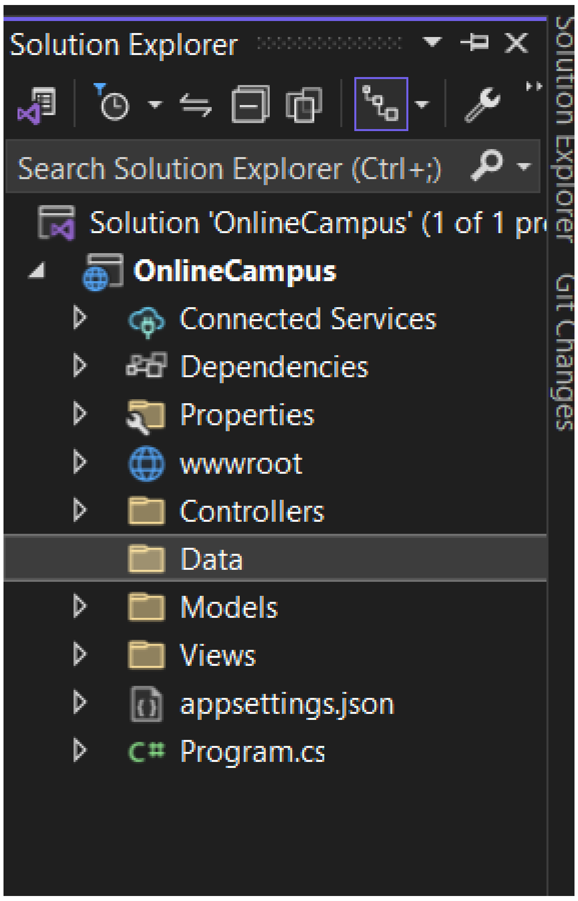


```
namespace OnlineCampus.Models
{
    public class Course
    {
        public Guid CourseId { get; set; }

        public string Code { get; set; } = string.Empty;

        public string Name { get; set; } = string.Empty;

        public string Description { get; set; } = string.Empty;

        public int Credits { get; set; } = 0;

        public ICollection<Enrolment> Enrolments { get; set; } = new List<Enrolment>();
    }
}
```

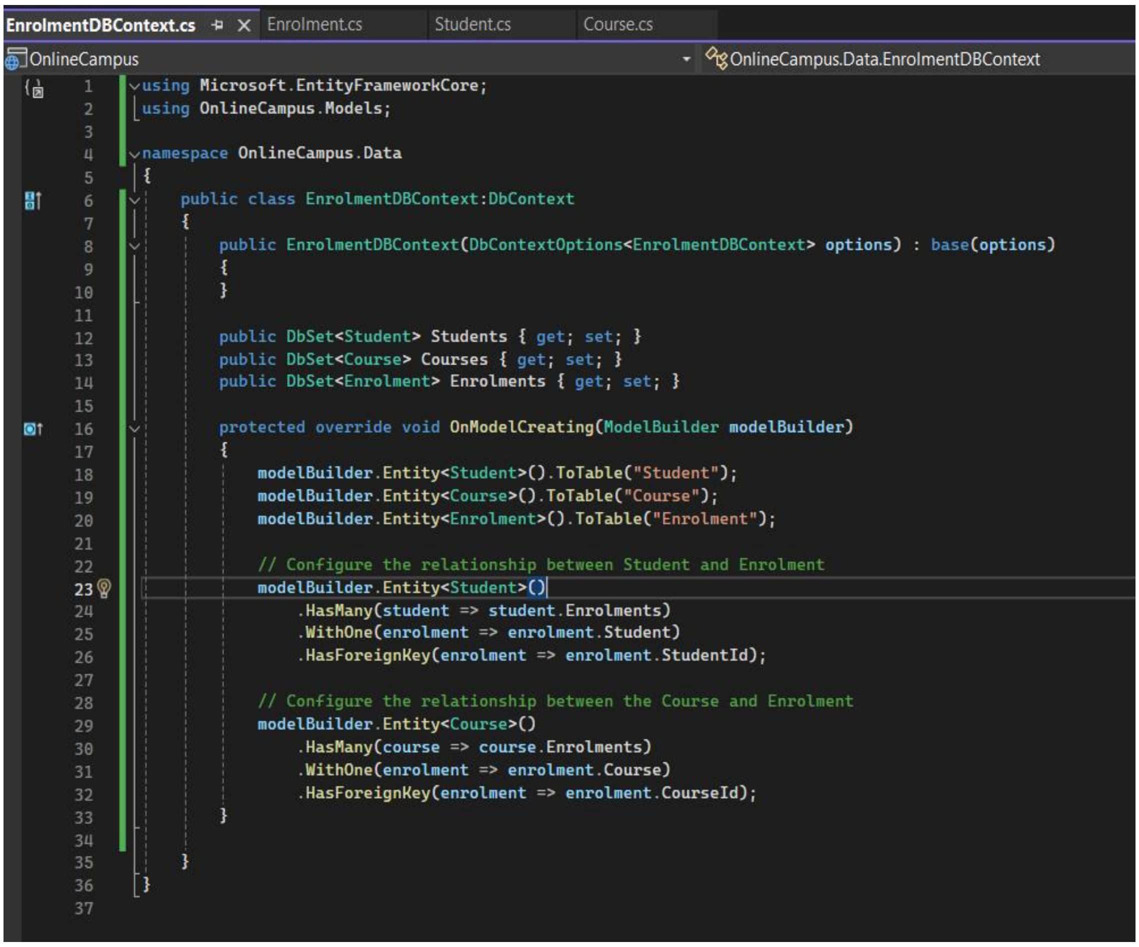


```
namespace OnlineCampus.Models
{
    public class Enrolment
    {
        public Guid EnrolmentId { get; set; }

        public Guid StudentId { get; set; }

        public Guid CourseId { get; set; }

        public Student Student { get; set; } = new Student();

        public Course Course { get; set; } = new Course();
    }
}
```

2. Set up the Database Context.
    a. Install EntityFrameworkCore and EntityFrameworkCore .SqlServer.
    b. Create a Data Folder

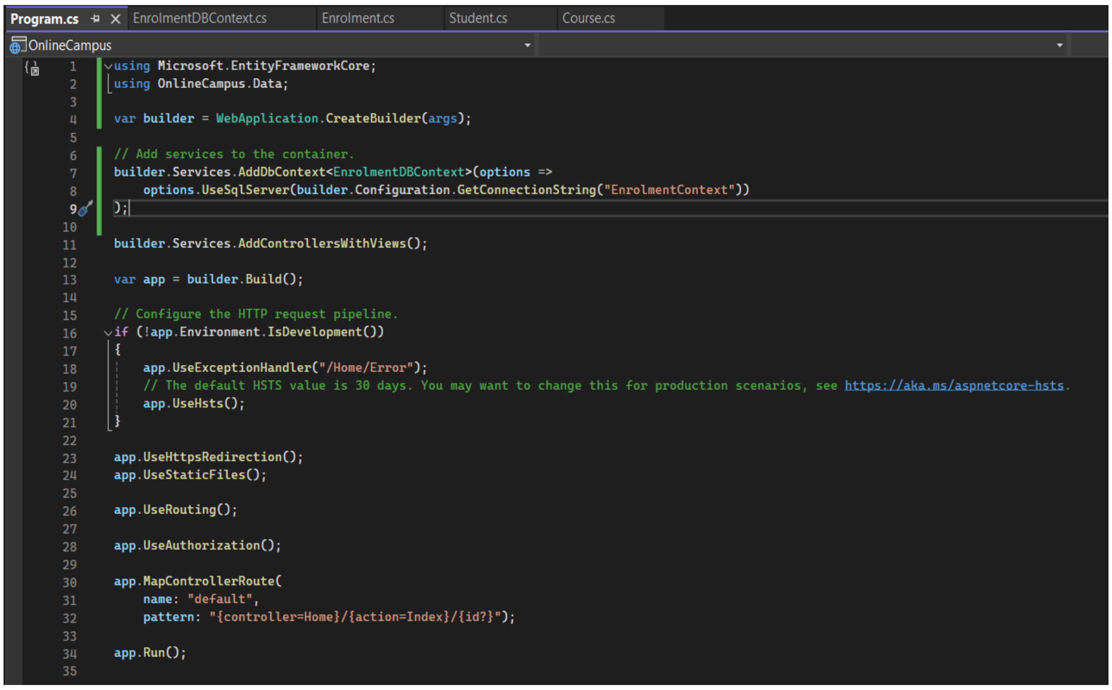


c. Inside the Data Folder, create a EnrolmentDBContext class:



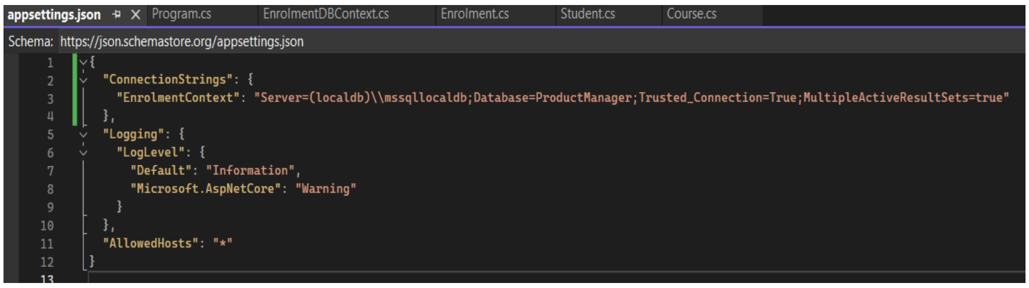
```csharp
using Microsoft.EntityFrameworkCore;
using OnlineCampus.Models;

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

        public DbSet<Student> Students { get; set; }
        public DbSet<Course> Courses { get; set; }
        public DbSet<Enrolment> Enrolments { get; set; }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder.Entity<Student>().ToTable("Student");
            modelBuilder.Entity<Course>().ToTable("Course");
            modelBuilder.Entity<Enrolment>().ToTable("Enrolment");

            // Configure the relationship between Student and Enrolment
            modelBuilder.Entity<Student>()
                .HasMany(student => student.Enrolments)
                .WithOne(enrolment => enrolment.Student)
                .HasForeignKey(enrolment => enrolment.StudentId);

            // Configure the relationship between the Course and Enrolment
            modelBuilder.Entity<Course>()
                .HasMany(course => course.Enrolments)
                .WithOne(enrolment => enrolment.Course)
                .HasForeignKey(enrolment => enrolment.CourseId);
        }

    }
}
```

3. Configure the Database (Practice not recommended for security reasons)
   a. In the Program.cs file, inject the database context.



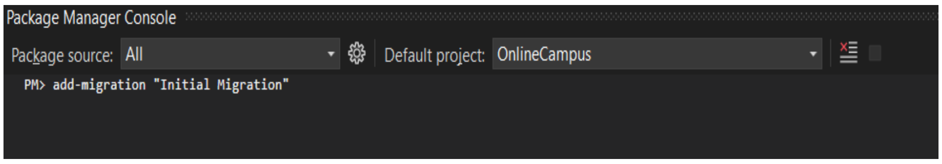
```csharp
using Microsoft.EntityFrameworkCore;
using OnlineCampus.Data;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddDbContext<EnrolmentDBContext>(options =>
    options.UseSqlServer(builder.Configuration.GetConnectionString("EnrolmentContext"))
);

builder.Services.AddControllersWithViews();

var app = builder.Build();

// Configure the HTTP request pipeline.
if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Home/Error");
    // The default HSTS value is 30 days. You may want to change this for production scenarios, see https://aka.ms/aspnetcore-hsts.
    app.UseHsts();
}

app.UseHttpsRedirection();
app.UseStaticFiles();

app.UseRouting();

app.UseAuthorization();

app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}");

app.Run();
```

   b. In the appsetting.json file, define the connection string.

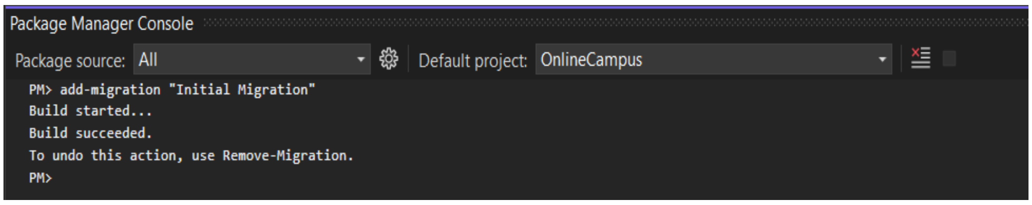


```json
{
  "ConnectionStrings": {
    "EnrolmentContext": "Server=(localdb)\\mssqllocaldb;Database=ProductManager;Trusted_Connection=True;MultipleActiveResultSets=true"
  },
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*"
}
```

4. Migrate the Database
    a. Install EntityFrameworkCore.Tools.
    b. In the Package Manager Console, add the initial migration:

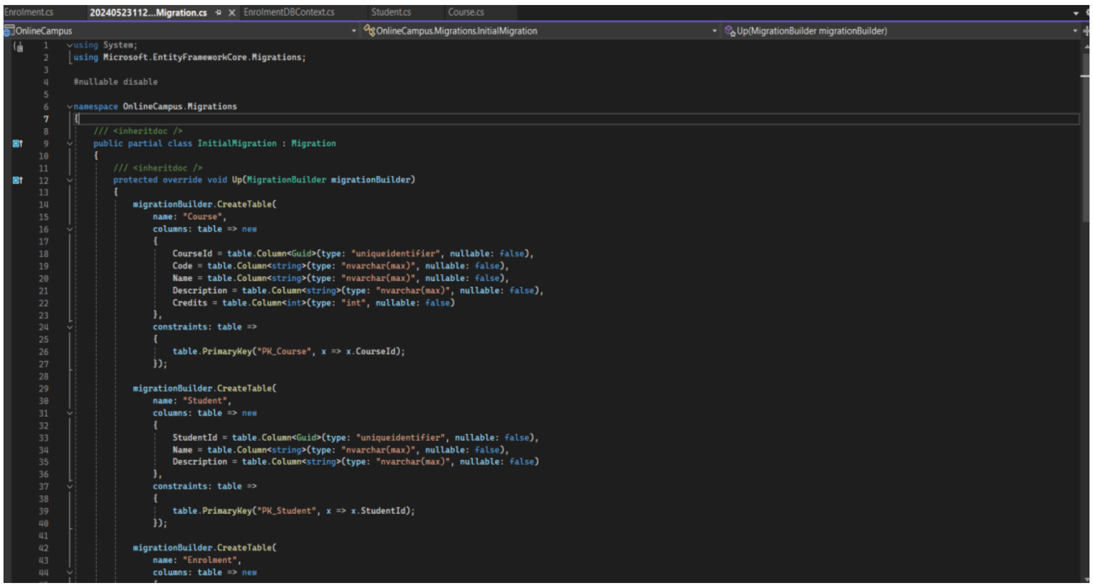


```
PM> add-migration "Initial Migration"
```



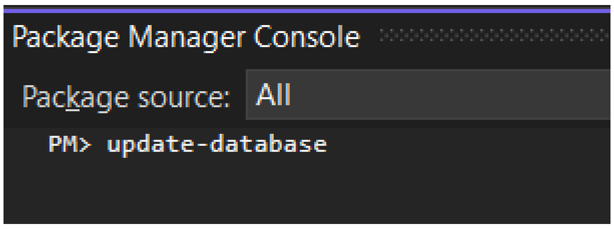
```
PM> add-migration "Initial Migration"
Build started...
Build succeeded.
To undo this action, use Remove-Migration.
PM>
```

    c. This generates the following migration file, which contains detailed information regarding the table configurations. These can also be changed before the next step for further refinement, for instance, the nvarchar data type of CourseCode is of max length and we have established that it should be of 6 characters:

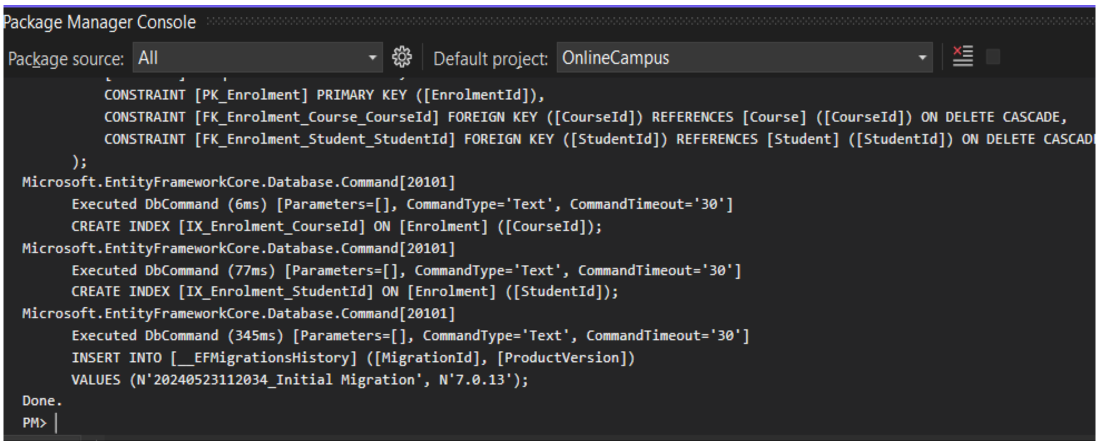

Enrolment.cs
20240523112...Migration.cs
EnrolmentDBContext.cs
Student.cs
Course.cs
OnlineCampus
OnlineCampus.Migrations.InitialMigration
Up(MigrationBuilder migrationBuilder)

```
using System;
using Microsoft.EntityFrameworkCore.Migrations;

#nullable disable

namespace OnlineCampus.Migrations
{
    /// <inheritdoc />
    public partial class InitialMigration : Migration
    {
        /// <inheritdoc />
        protected override void Up(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.CreateTable(
                name: "Course",
                columns: table => new
                {
                    CourseId = table.Column<Guid>(type: "uniqueidentifier", nullable: false),
                    Code = table.Column<string>(type: "nvarchar(max)", nullable: false),
                    Name = table.Column<string>(type: "nvarchar(max)", nullable: false),
                    Description = table.Column<string>(type: "nvarchar(max)", nullable: false),
                    Credits = table.Column<int>(type: "int", nullable: false)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_Course", x => x.CourseId);
                });

            migrationBuilder.CreateTable(
                name: "Student",
                columns: table => new
                {
                    StudentId = table.Column<Guid>(type: "uniqueidentifier", nullable: false),
                    Name = table.Column<string>(type: "nvarchar(max)", nullable: false),
                    Description = table.Column<string>(type: "nvarchar(max)", nullable: false)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_Student", x => x.StudentId);
                });

            migrationBuilder.CreateTable(
                name: "Enrolment",
                columns: table => new
```

    d. In the Package Manage Console, Update the database:

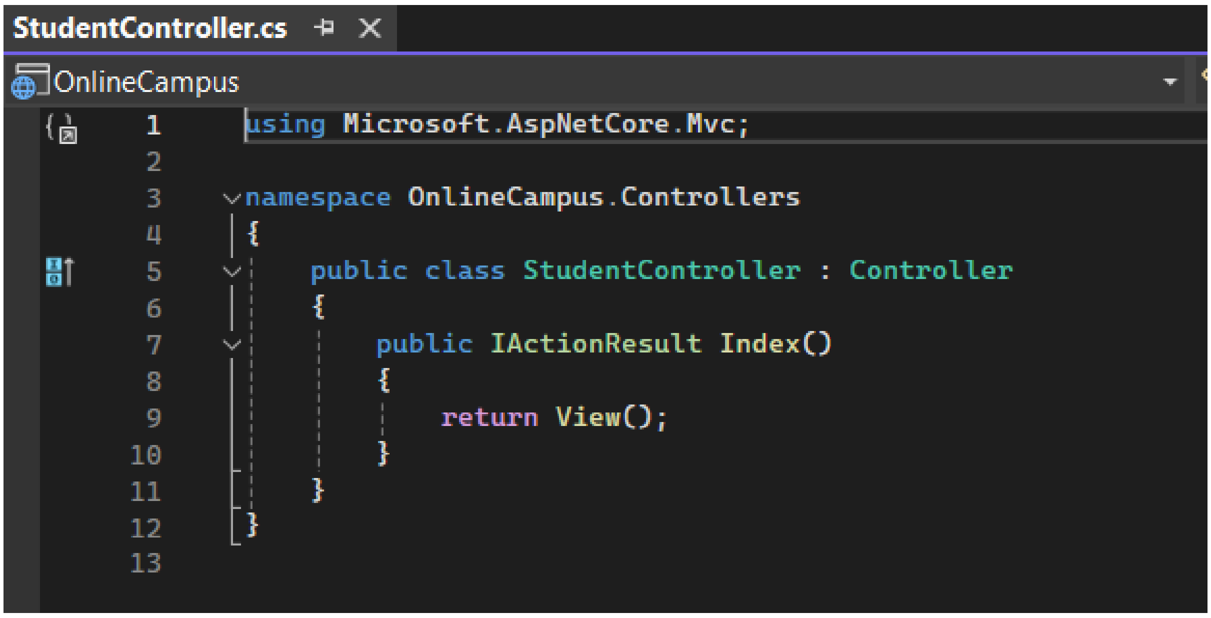


```
PM> update-database
```

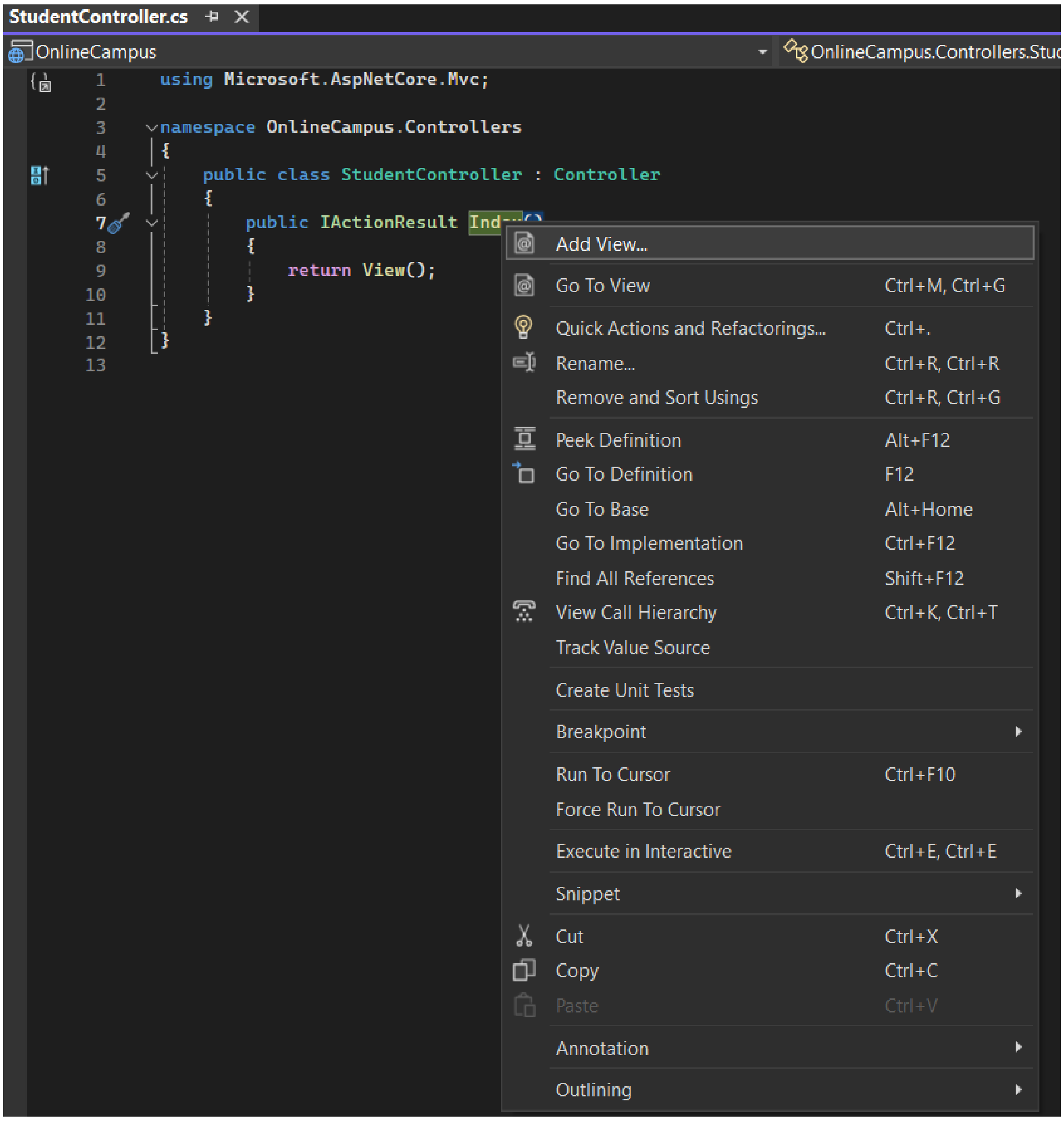


```
            CONSTRAINT [PK_Enrolment] PRIMARY KEY ([EnrolmentId]),
            CONSTRAINT [FK_Enrolment_Course_CourseId] FOREIGN KEY ([CourseId]) REFERENCES [Course] ([CourseId]) ON DELETE CASCADE,
            CONSTRAINT [FK_Enrolment_Student_StudentId] FOREIGN KEY ([StudentId]) REFERENCES [Student] ([StudentId]) ON DELETE CASCAD
        );
Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (6ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
      CREATE INDEX [IX_Enrolment_CourseId] ON [Enrolment] ([CourseId]);
Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (77ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
      CREATE INDEX [IX_Enrolment_StudentId] ON [Enrolment] ([StudentId]);
Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (345ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
      INSERT INTO [__EFMigrationsHistory] ([MigrationId], [ProductVersion])
      VALUES (N'20240523112034_Initial Migration', N'7.0.13');
Done.
PM>
```

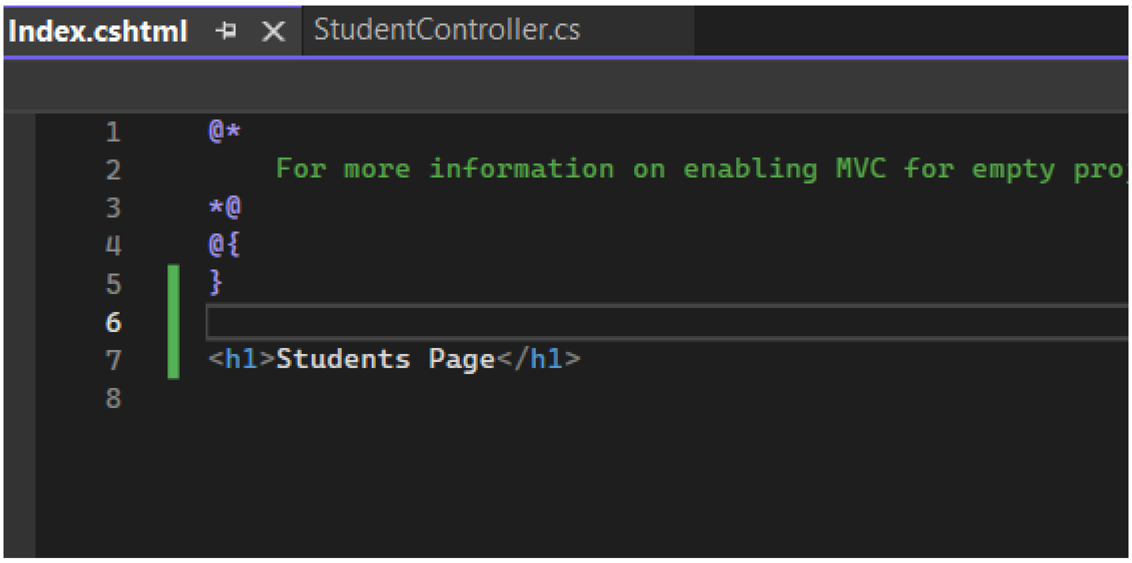
5. Create a StudentController in the Controller folder:



```
using Microsoft.AspNetCore.Mvc;

namespace OnlineCampus.Controllers
{
    public class StudentController : Controller
    {
        public IActionResult Index()
        {
            return View();
        }
    }
}
```

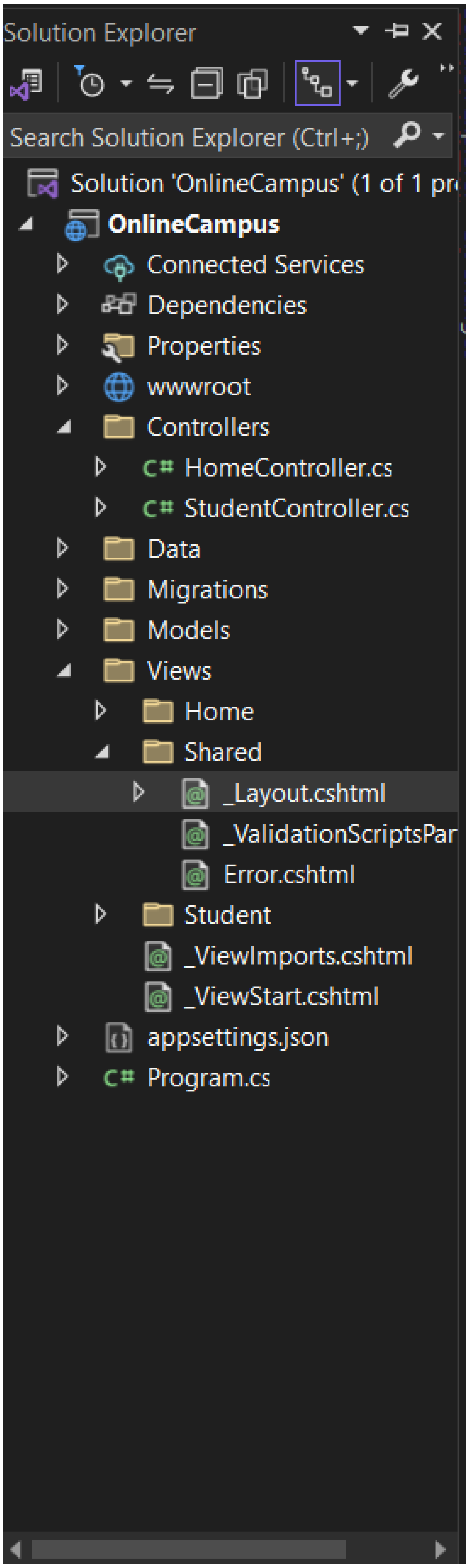


```
using Microsoft.AspNetCore.Mvc;

namespace OnlineCampus.Controllers
{
    public class StudentController : Controller
    {
        public IActionResult Ind
        {
            return View();
        }
    }
}
```



```
@*
    For more information on enabling MVC for empty pro
*@
@{
}

<h1>Students Page</h1>
```

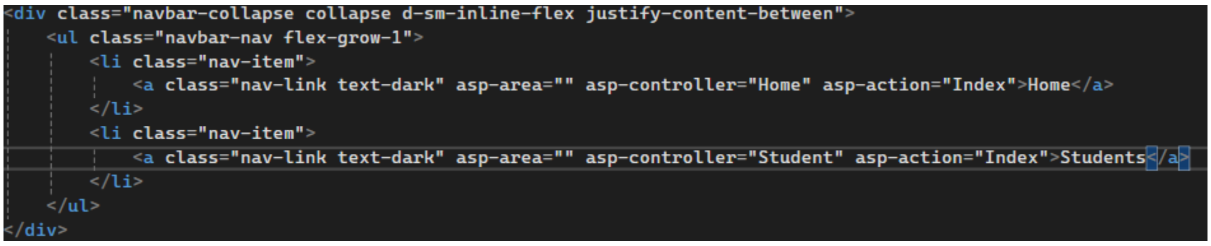
6. In the Layout file, add a link to the Student Index View:



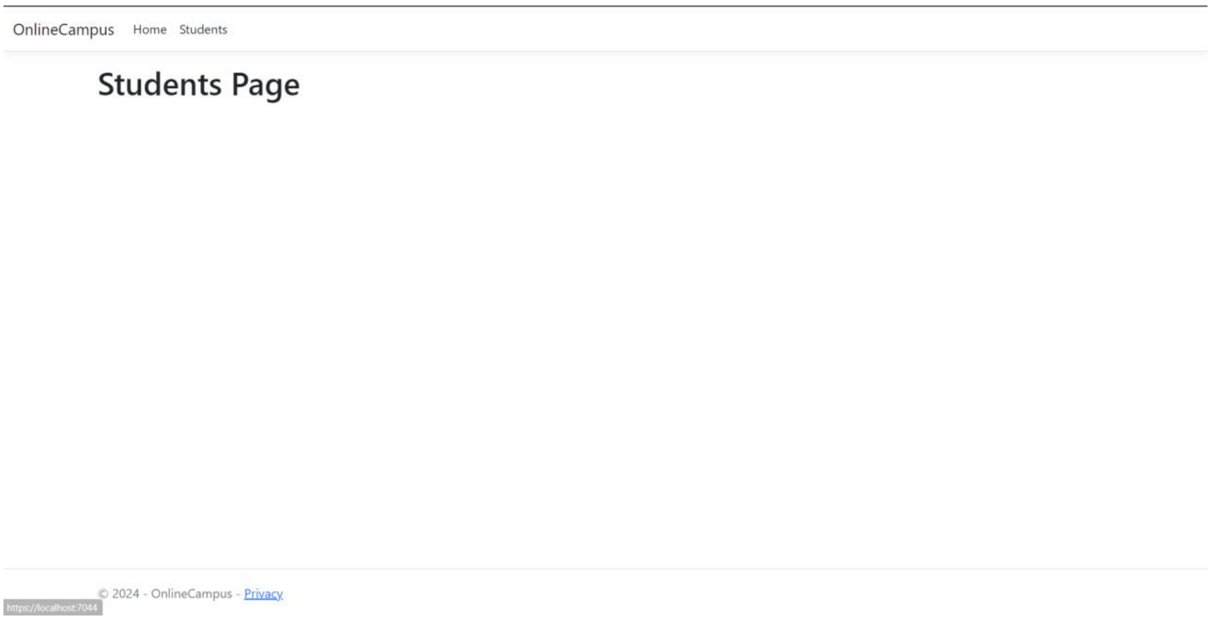
Solution Explorer

Search Solution Explorer (Ctrl+;)

```
Solution 'OnlineCampus' (1 of 1 pr
◢ OnlineCampus
    ▷ Connected Services
    ▷ Dependencies
    ▷ Properties
    ▷ wwwroot
    ◢ Controllers
        ▷ C# HomeController.cs
        ▷ C# StudentController.cs
    ▷ Data
    ▷ Migrations
    ▷ Models
    ◢ Views
        ▷ Home
        ◢ Shared
            ▷ _Layout.cshtml
              _ValidationScriptsPar
              Error.cshtml
        ▷ Student
          _ViewImports.cshtml
          _ViewStart.cshtml
    ▷ appsettings.json
    ▷ C# Program.cs
```


```
<div class="navbar-collapse collapse d-sm-inline-flex justify-content-between">
    <ul class="navbar-nav flex-grow-1">
        <li class="nav-item">
            <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Index">Home</a>
        </li>
        <li class="nav-item">
            <a class="nav-link text-dark" asp-area="" asp-controller="Student" asp-action="Index">Students</a>
        </li>
    </ul>
</div>
```

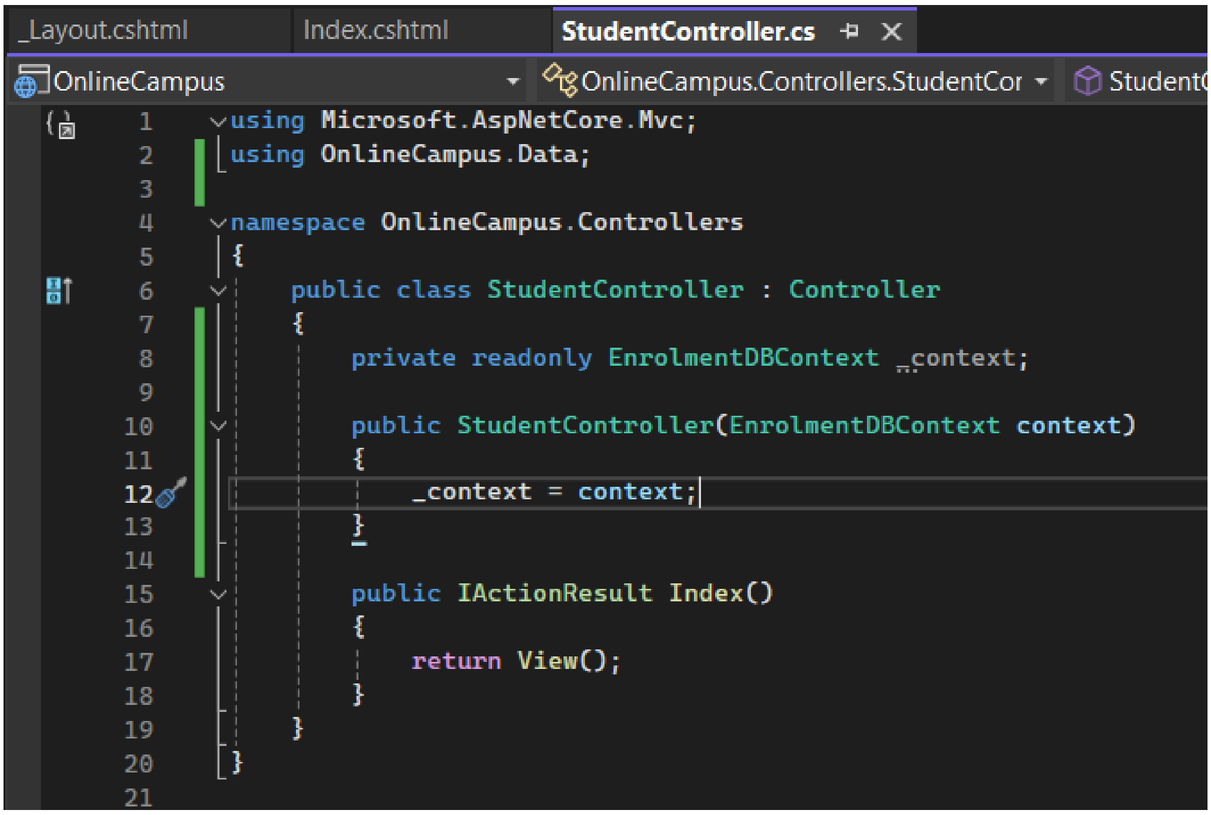
7. Run the application and check the Students Page.



Students Page

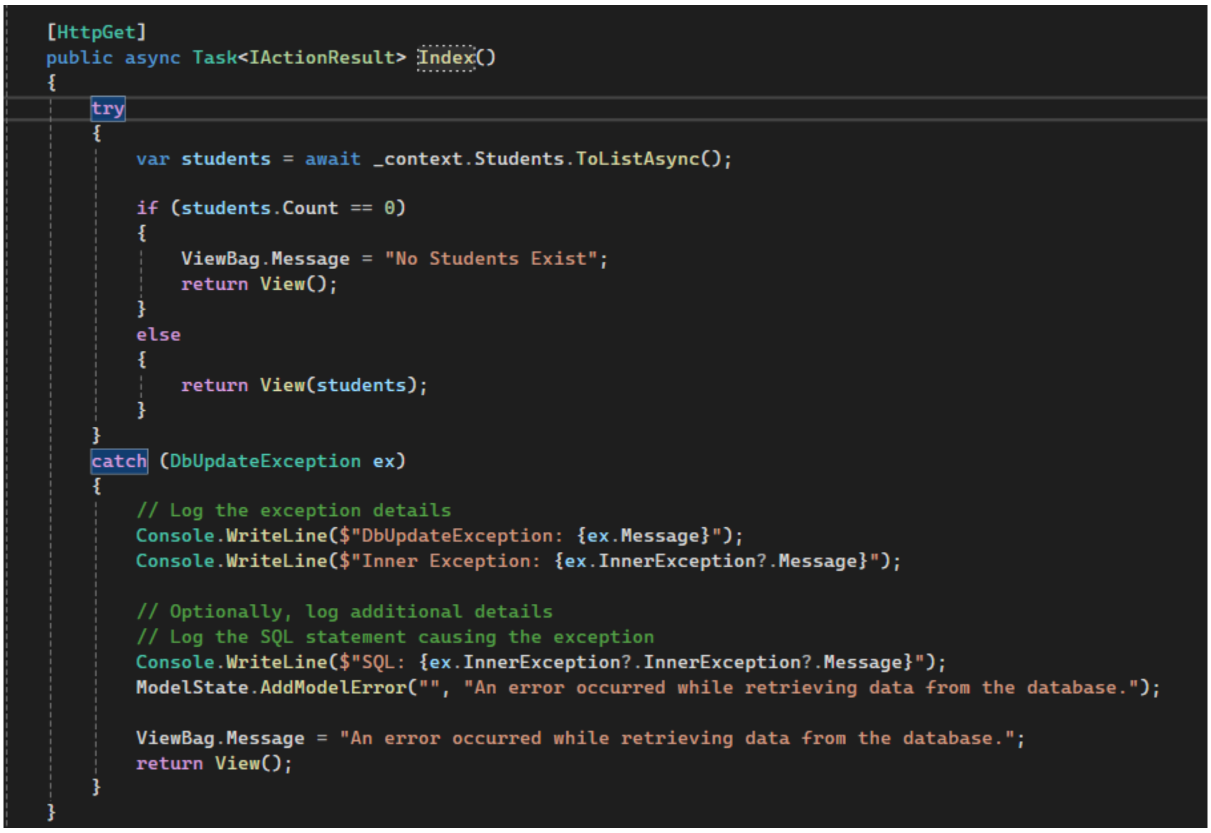
© 2024 - OnlineCampus - Privacy

8. Inject the Database into the StudentController:



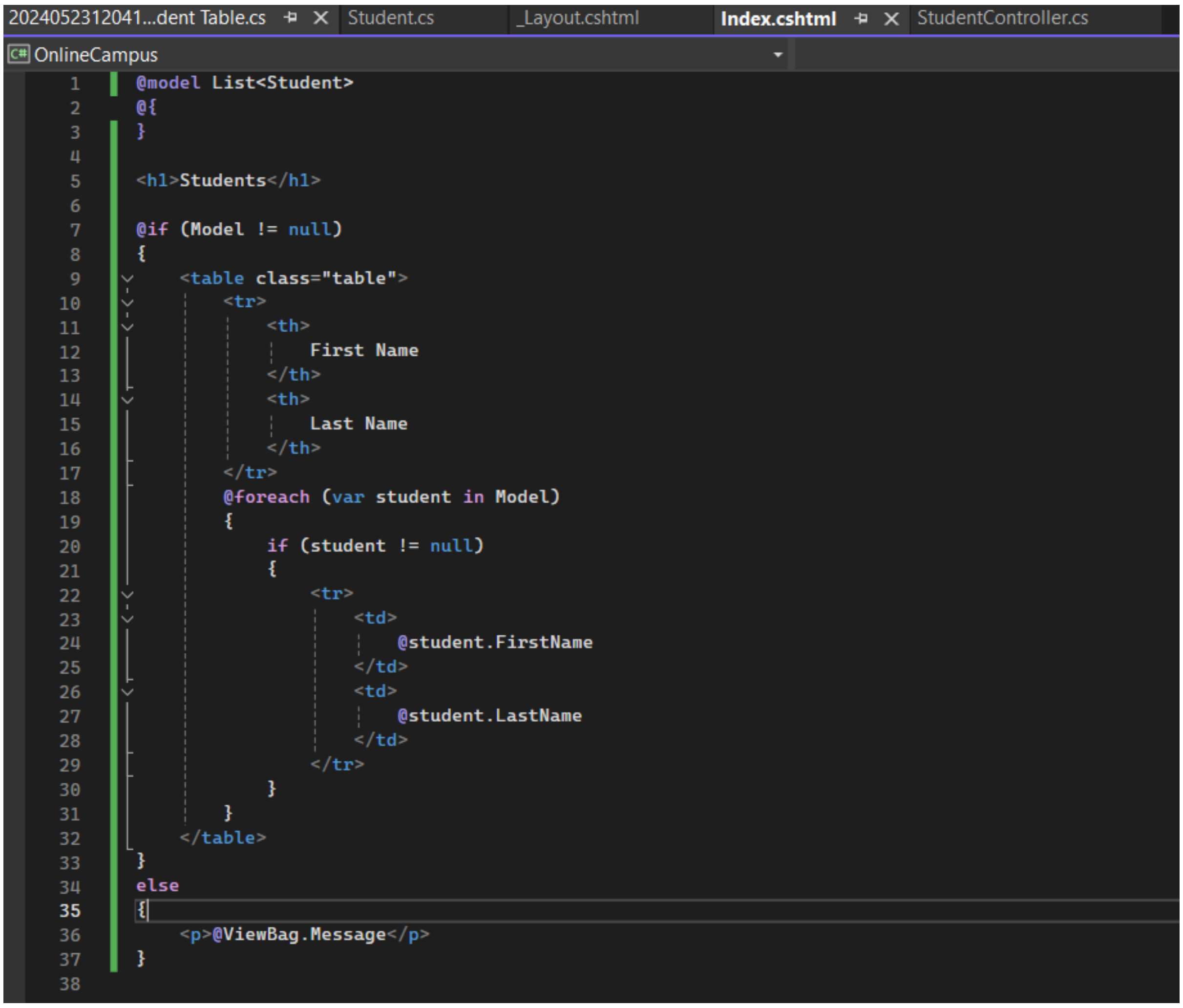
```
using Microsoft.AspNetCore.Mvc;
using OnlineCampus.Data;

namespace OnlineCampus.Controllers
{
    public class StudentController : Controller
    {
        private readonly EnrolmentDBContext _context;

        public StudentController(EnrolmentDBContext context)
        {
            _context = context;
        }

        public IActionResult Index()
        {
            return View();
        }
    }
}
```

9. The Index view will be used to display a list students from the students table.
    a. If the students table is empty, we create a message “No Students Exist” or we return the View with the returned values of students.



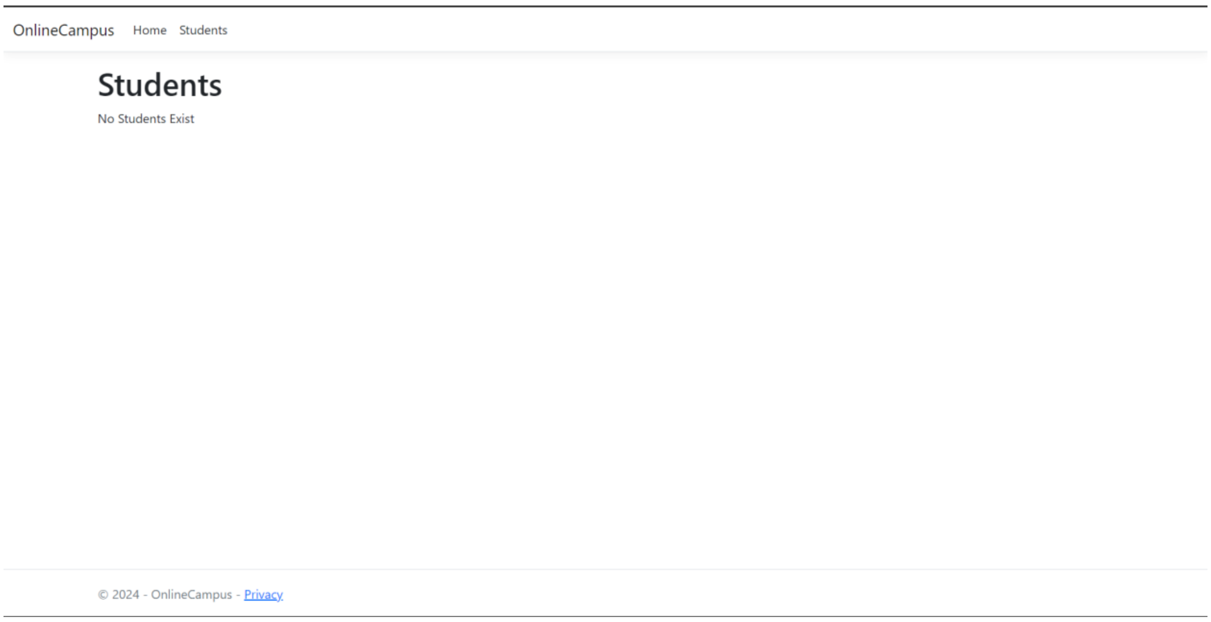
```
[HttpGet]
public async Task<IActionResult> Index()
{
    try
    {
        var students = await _context.Students.ToListAsync();

        if (students.Count == 0)
        {
            ViewBag.Message = "No Students Exist";
            return View();
        }
        else
        {
            return View(students);
        }
    }
    catch (DbUpdateException ex)
    {
        // Log the exception details
        Console.WriteLine($"DbUpdateException: {ex.Message}");
        Console.WriteLine($"Inner Exception: {ex.InnerException?.Message}");

        // Optionally, log additional details
        // Log the SQL statement causing the exception
        Console.WriteLine($"SQL: {ex.InnerException?.InnerException?.Message}");
        ModelState.AddModelError("", "An error occurred while retrieving data from the database.");

        ViewBag.Message = "An error occurred while retrieving data from the database.";
        return View();
    }
}
```

b. Update the View



```
@model List<Student>
@{
}

<h1>Students</h1>

@if (Model != null)
{
    <table class="table">
        <tr>
            <th>
                First Name
            </th>
            <th>
                Last Name
            </th>
        </tr>
        @foreach (var student in Model)
        {
            if (student != null)
            {
                <tr>
                    <td>
                        @student.FirstName
                    </td>
                    <td>
                        @student.LastName
                    </td>
                </tr>
            }
        }
    </table>
}
else
{
    <p>@ViewBag.Message</p>
}
```

10. Test the application:



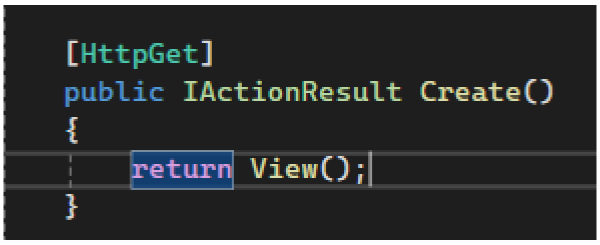
Students

No Students Exist

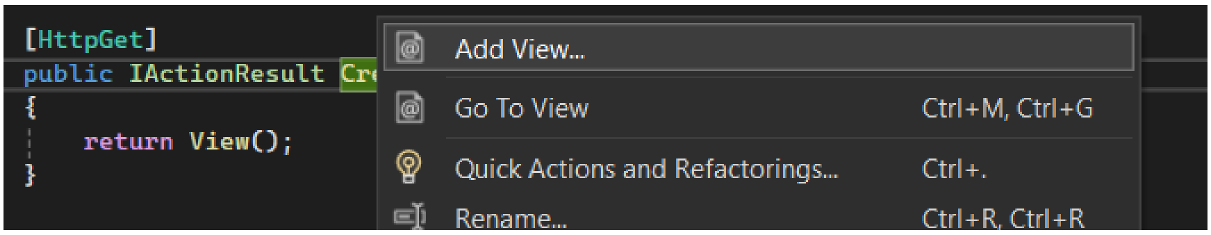
© 2024 - OnlineCampus - Privacy

11. Now we need to complete the CRUD (Create, Read, Update, Delete) functionality for the student related data. So far, we must create the Read functionality. In the next section we will create the Create functionality.

12. In the Students Controller, add a Create action, that will direct the user to a form in order to Add a student.

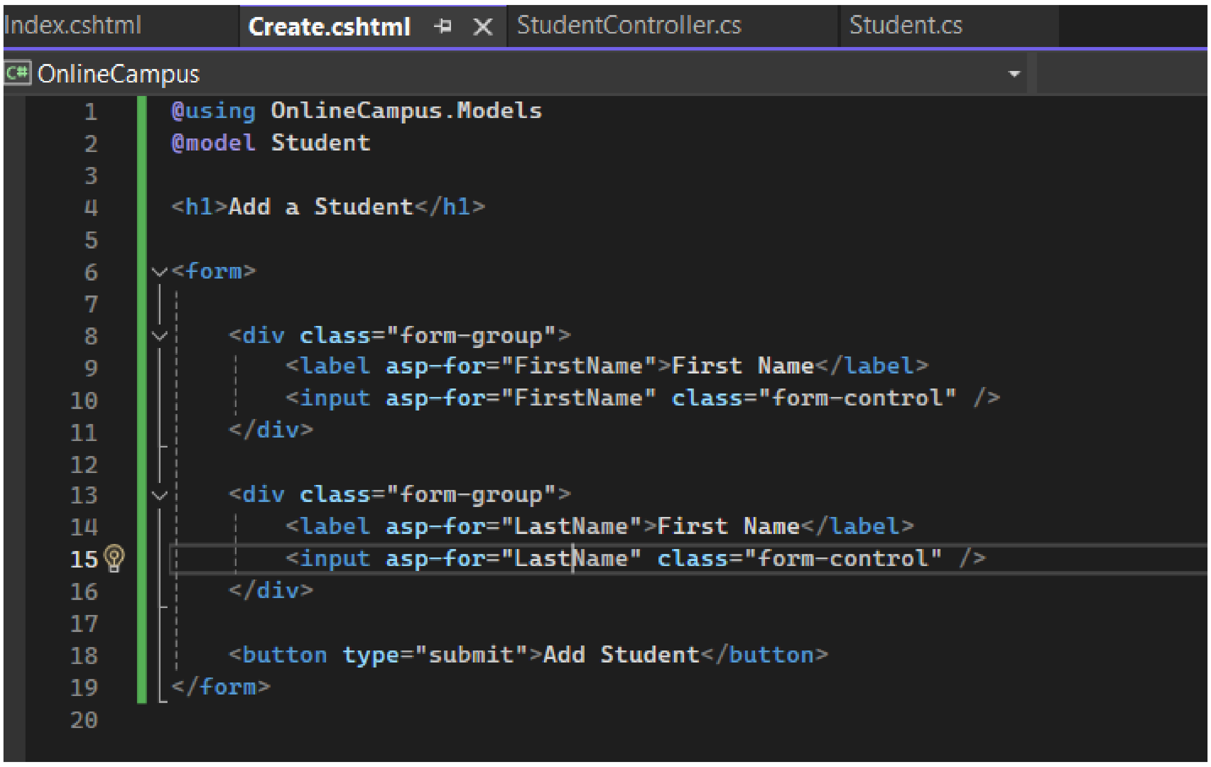


```
[HttpGet]
public IActionResult Create()
{
    return View();
}
```



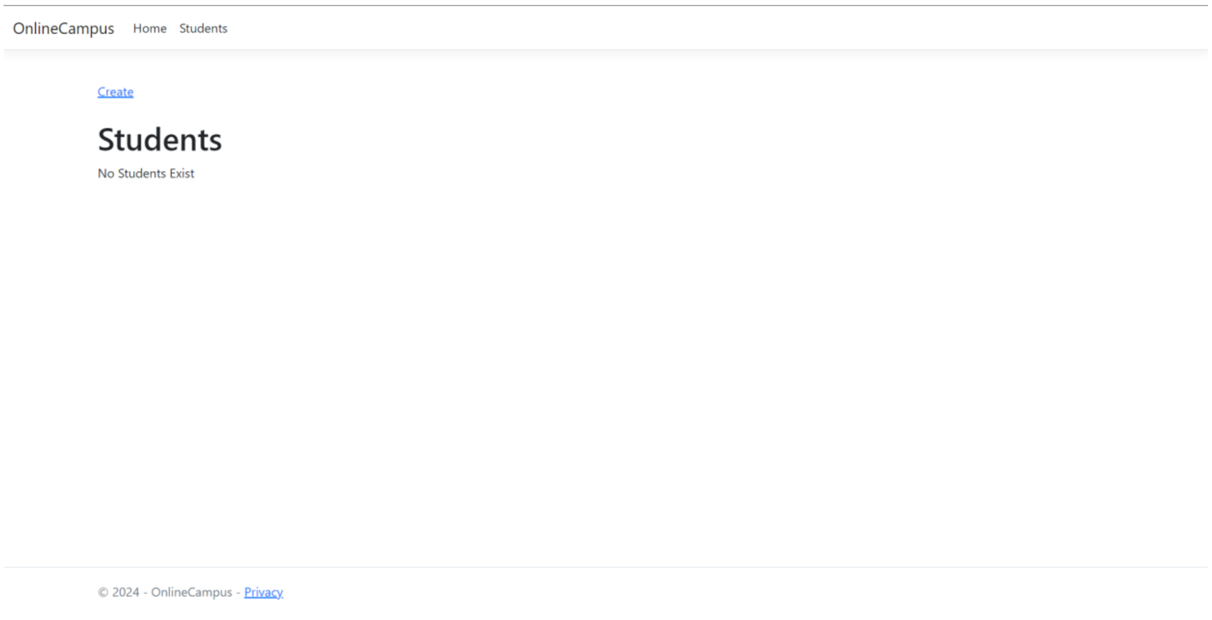
```
[HttpGet]
public IActionResult Cre
{
    return View();
}
```

Add View...
Go To View Ctrl+M, Ctrl+G
Quick Actions and Refactorings... Ctrl+.
Rename... Ctrl+R, Ctrl+R



```
@using OnlineCampus.Models
@model Student

<h1>Add a Student</h1>

<form>

    <div class="form-group">
        <label asp-for="FirstName">First Name</label>
        <input asp-for="FirstName" class="form-control" />
    </div>

    <div class="form-group">
        <label asp-for="LastName">First Name</label>
        <input asp-for="LastName" class="form-control" />
    </div>

    <button type="submit">Add Student</button>
</form>
```

13. Run the application

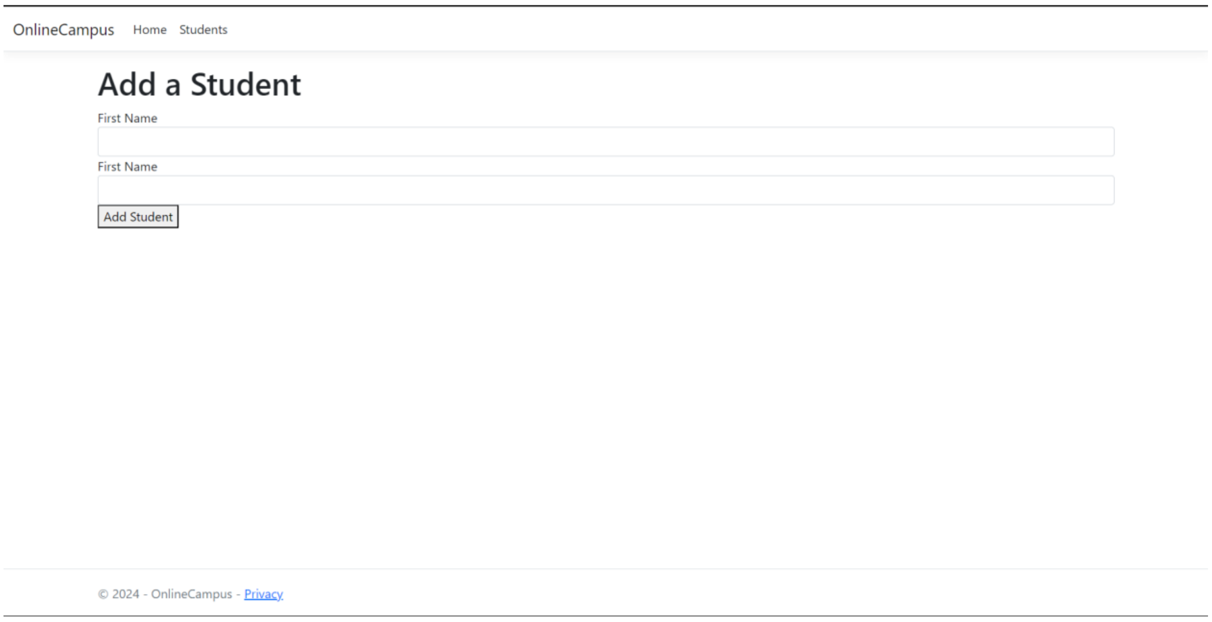

OnlineCampus Home Students

Create

Students

No Students Exist

© 2024 - OnlineCampus - Privacy


OnlineCampus Home Students

Add a Student

First Name

First Name

Add Student

© 2024 - OnlineCampus - Privacy

14. Now, we must enable the user to add the student once they submit the form.

GET Method	Post Method
Try-Catch	Try-Catch
Logging	Logging
No Tracking	Tracking
Sort, Filter	Concurrency
Page, Group	Overposting
	Anti-Forgery Token