Academia International College

Affiliated to Tribhuvan University

Lab No. 6

Lab Report on

<u>CRUD Operation using ASP.NET Core MVC</u>

Submitted by:

Sagar Timalsena

Roll No. 15847

6th semester

Submitted to:

Chandan Bhagat Gupta

Request

<u>Theory</u>

ASP.NET Core MVC:

- ASP.NET Core MVC is an open source web development framework from Microsoft that provides Model-View-Controller architecture.
- The MVC pattern helps you create apps that are more testable and easier to update than traditional monolithic apps.
- The Model-View-Controller (MVC) architectural pattern separates an app into three main components: Model, View, and Controller.
- Model is a class that represent the data of the app. Model objects retrieve and store model state in a database.
- View is the component that display the app's user interface (UI). Generally, this UI displays the model data.
- Controller is a class that handles browser request, retrieves model data and call view templates that return a response.

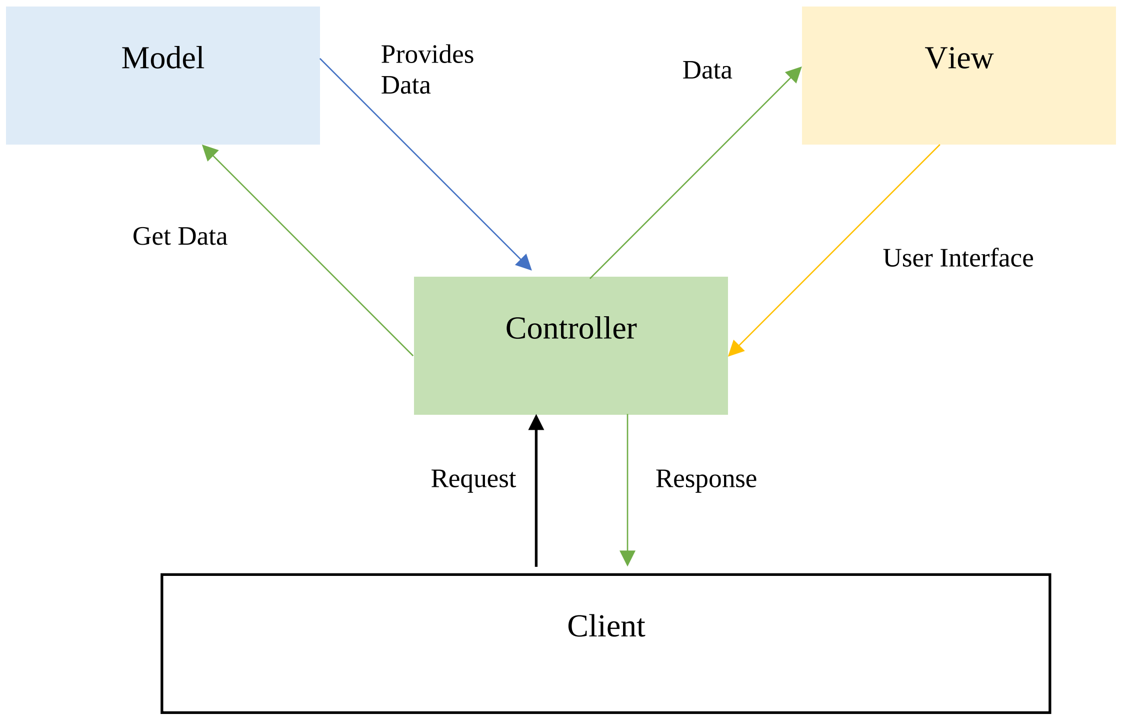


Fig: MVC Architecture

Procedure

ASP.NET Core can be used to perform CRUD operation. It has the functionality of MVC pattern which helps to manage complexity when building app, because it enables work on one aspect of the implementation at a time without impacting the code of another. We used Visual Studio 2019 to manage an ASP.NET Core MVC architecture.

ASP.NET Core web app development contains following steps:

Creating a Web Application in Visual Studio

1. Start Visual Studio and select "Create a new project". In the Create a new project dialog, select ASP.NET Core Web Application and click Next button.
2. In "Configure your new project" dialog, enter the project name. For e.g. **Lab6**. It is important to name the project with capitalization because it represents the namespace of entire project.
3. Select Create button.
4. In "Create an new ASP.NET Core web application" dialog, select: .NET Core and ASP.NET Core 5.0 in the dropdowns.
    - ASP.NET Core Web App (Model-View-Controller).
    - Create.

The above steps creates an ASP.NET Core MVC web app in Visual Studio which contains different in-built files which helps to configure and run default web pages of ASP.NET Core web app.

Building a Web Application in Visual Studio

1. Install required dependencies using NuGet Package.
    - Microsoft.EntityFrameworkCore
    - Microsoft.EntityFrameworkCore.SqlServer
    - Microsoft.EntityFrameworkCore.Tools System.ConfigurationManager

2. Configure ConnectionStrings to identify the server instance, connect SQL Server instance in "appsettings.json".

```
{
    "Logging": {
    "LogLevel": {
        "Default": "Information",        Request
        "Microsoft": "Warning",
        "Microsoft.Hosting.Lifetime": "Information"
        }
    },
    "AllowedHosts": "*",
    "ConnectionStrings": {
        "DevConnection":
        "Server=(localdb)\\mssqllocaldb;Database=StudentInfo;Trusted_Conn
        ection=True;MultipleActiveResultSets=True"
    }

}
```

3. Add a Model. Model is a class that represent the data of the app. Model objects retrieve and store model state in a database.

    - Right-click on the Model Folder, select Add > Class and name the class as Student.cs.

4. Update the Student.cs file using System; using System.Collections.Generic; using System.ComponentModel.DataAnnotations; using System.Linq; using System.Threading.Tasks;

```
namespace Lab6.Models
{
    public class Student
    {
        [Key]
        public int StudentId { get; set; }
        [Required]
        public int SymbolNumber { get; set; }
        [Required]
        public string FullName { get; set; }
        [Required]
        public string Address { get; set; }

    }
}
```

5. Create a DBContext file. DBContext specifies the entities to include in the data model. The context object allows querying and saving data.

    - Right-click on the Model Folder, select Add > Class and name the class as StudentContext.cs.

6. Update the StudentContext.cs file.

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

namespace Lab6.Models
{
    public class StudentContext: DbContext
    {
        public StudentContext(DbContextOptions<StudentContext>
        options):base(options)
        {
                }

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

7. Registers the database context in Startup.ConfigureServices of the Startup.cs file.

```
public void ConfigureServices(IServiceCollection services)
{
        services.AddControllersWithViews();

        services.AddDbContext<StudentContext>(options =>
options.UseSqlServer(Configuration.GetConnectionString("DevConnection")));
}
```

8. Initial migration.
    - From the Tools menu, select NuGet Package Manager > Package Manager Console.
    - In the Package Manager Console (PMC), enter the following commands:

      ```
      Add-Migration InitialCreate Update-Database
      ```

9. Add a Controller. Controller is a class that handles browser request, retrieves model data and call view templates that return a response.
    - Right-click on the Controller Folder and select Add > Controller.
    - In the Add Scaffold dialog box, select “MVC Controller – Empty”.
    - In the Add New Item, enter Controller name and select Add. For e.g. **StudentController.cs.**

10. Add action methods for Create, Read, Update and Delete.

```
using Microsoft.AspNetCore.Mvc;
using System; using
System.Collections.Generic; using
```

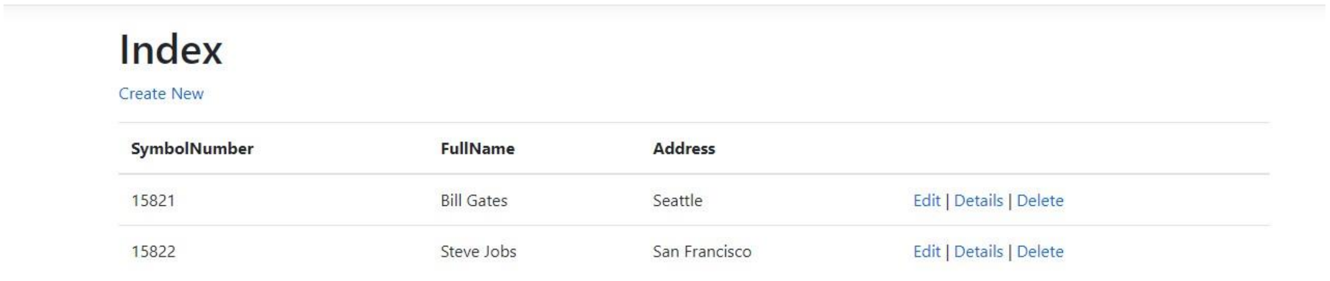
```
System.Linq; using
System.Threading.Tasks; using
Microsoft.EntityFrameworkCore;
using Lab6.Models;

namespace Lab6.Controllers
{
        public class StudentController : Controller
        {
                private readonly StudentContext _db;

                public StudentController(StudentContext db)
                {
                        _db = db;
                }
                public IActionResult Index()
                {
                        var displaydata = _db.Student.ToList();
        return View(displaydata);
                }

                public IActionResult Create()
                {
                        return View();
                }


                [HttpPost]
                public async Task<IActionResult> Create(Student std)
                {
                        if(ModelState.IsValid)
                        {
                                _db.Add(std);
                await _db.SaveChangesAsync();
        return RedirectToAction("Index");
                        }
                        return View(std);
                }

                public async Task<IActionResult> Edit(int? id)
                {
                        if(id == null)
                        {
                                return RedirectToAction("Index");
                        }

                        var getStudentDetails = await _db.Student.FindAsync(id);
                return View(getStudentDetails);
                }

                [HttpPost]
                public async Task<IActionResult> Edit(Student std)
                {
                        if(ModelState.IsValid)
                        {
                                _db.Update(std);
                await _db.SaveChangesAsync();
        return RedirectToAction("Index");                       }

                        return View(std);
                }
        public async Task<IActionResult> Details(int? id)
        {
            if (id == null)
            {
                return RedirectToAction("Index");
            }

            var getStudentDetails = await _db.Student.FindAsync(id);
    return View(getStudentDetails);
        }

        public async Task<IActionResult> Delete(int? id)
        {
            if (id == null)
            {
                return RedirectToAction("Index");
            }

            var getStudentDetails = await _db.Student.FindAsync(id);
    return View(getStudentDetails);
        }

        [HttpPost]
        public async Task<IActionResult> Delete(int id)
        {
            var   getStudentDetails   =   await   _db.Student.FindAsync(id);
     _db.Student.Remove(getStudentDetails);                              await
_db.SaveChangesAsync();

            return RedirectToAction("Index");
```

```
                    }
    }
    }
```

11. Create View Template for Create, Read, Update and Delete.
    - Right-click on the corresponding Action Method for e.g. Index and select Add View > Razor View
    - In MVC View dialog, choose corresponding Template, Model Class and Data Context.
    - Repeat same process for each methods.

<u>Output</u>

1. Index Page

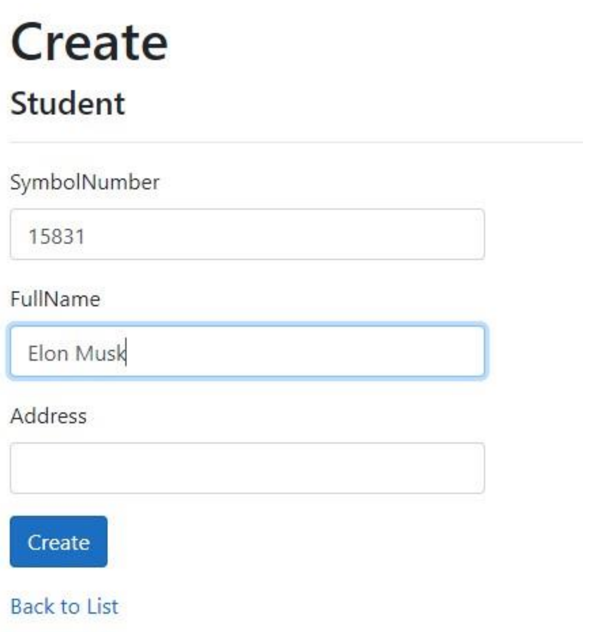


Index

Create New

SymbolNumber	FullName	Address	
15821	Bill Gates	Seattle	Edit | Details | Delete
15822	Steve Jobs	San Francisco	Edit | Details | Delete

2. Create Page



Create

Student

SymbolNumber

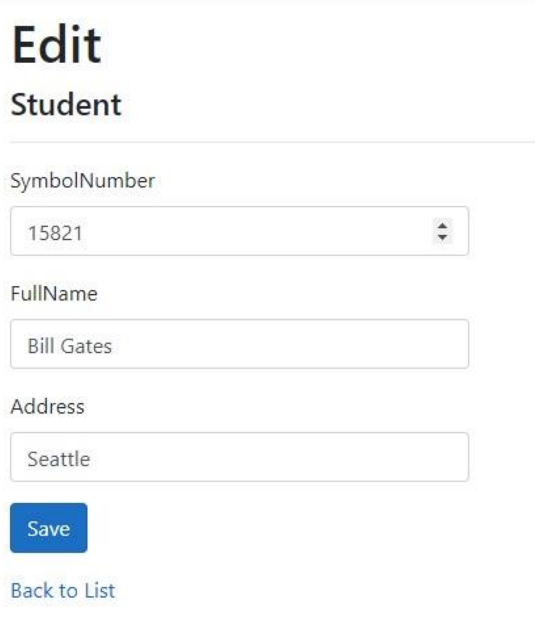
15831

FullName

Elon Musk

Address

3. Edit Page



Edit

Student

SymbolNumber

15821

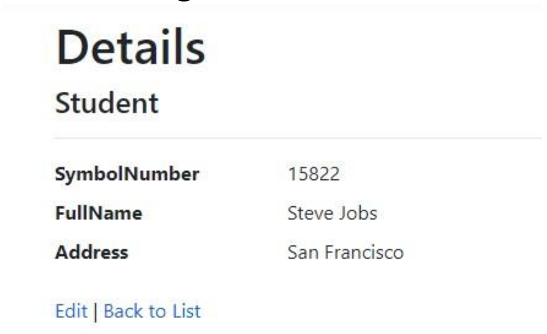
FullName

Bill Gates

Address

Seattle

4. Details Page



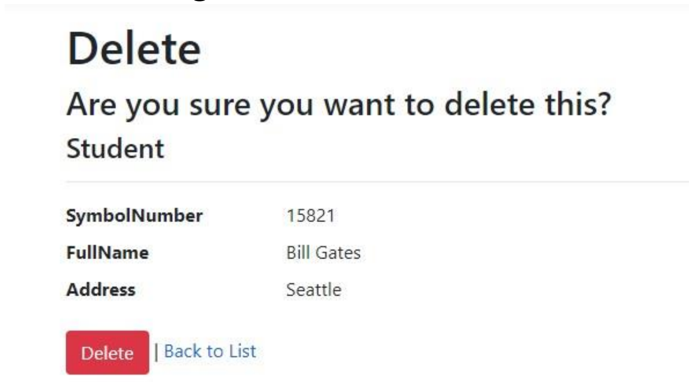
Details

Student

SymbolNumber	15822
FullName	Steve Jobs
Address	San Francisco

Edit | Back to List

5. Delete Page



Delete

Are you sure you want to delete this?

Student

SymbolNumber	15821
FullName	Bill Gates
Address	Seattle

Conclusion

Hence, we implemented ASP.NET Core MVC to perform CRUD operation. In this project, Models are added for managing Students in a database. These model classes are used with Entity Framework Core (EF Core) to work with a database. ASP.NET Core is built with dependency injection (DI). Services, such as the database context, must be registered with DI in Startup.

GitHub Repository

All the above codes used in this report are uploaded in GitHub Repository
https://github.com/Sagar1555/ncclab