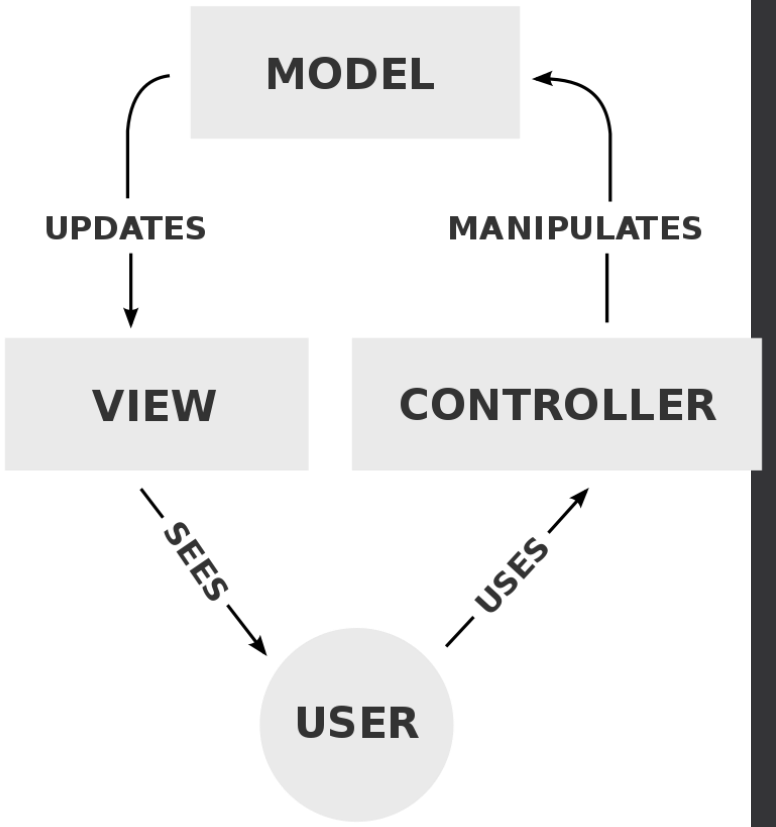
Internet Programming with MVC Design Pattern (.Net Core)

Prof. Dr. Aydın SEÇER

MVC (Model-View-Controller) Pattern

- **Model–view–controller** (usually known as MVC) is a software design pattern commonly used for developing user interfaces that divides the related program logic into three interconnected elements. (Based on **Separation of Concerns**)
- This is done to separate internal representations of information from the ways information is presented to and accepted from the user.
- The **Model** contains only the pure application data, it contains no logic describing how to present the data to a user.
- The **View** presents the model's data to the user. The view knows how to access the model's data, but it does not know what this data means or what the user can do to manipulate it.
- The **Controller** exists between the view and the model. It listens to events triggered by the view (or another external source) and executes the appropriate reaction to these events. In most cases, the reaction is to call a method on the model. Since the view and the model are connected through a notification mechanism, the result of this action is then automatically reflected in the view.



MVC Pattern Example

MODEL

```
class Student
{
    private String rollNo;
    private String name;

    public String getRollNo()
    {
        return rollNo;
    }

    public void setRollNo(String rollNo)
    {
        this.rollNo = rollNo;
    }

    public String getName()
    {
        return name;
    }

    public void setName(String name)
    {
        this.name = name;
    }
}
```

VIEW

```
class StudentView
{
    public void printStudentDetails(String studentName, String
studentRollNo)
    {
        System.out.println("Student: ");
        System.out.println("Name: " + studentName);
        System.out.println("Roll No: " + studentRollNo);
    }
}
```

CONTROLLER

```
class StudentController
{
    private Student model;
    private StudentView view;

    public StudentController(Student model, StudentView view)
    {
        this.model = model;
        this.view = view;
    }

    public void setStudentName(String name)
    {
        model.setName(name);
    }

    public String getStudentName()
    {
        return model.getName();
    }

    public void setStudentRollNo(String rollNo)
    {
        model.setRollNo(rollNo);
    }

    public String getStudentRollNo()
    {
        return model.getRollNo();
    }

    public void updateView()
    {
        view.printStudentDetails(model.getName(),
model.getRollNo());
    }
}
```

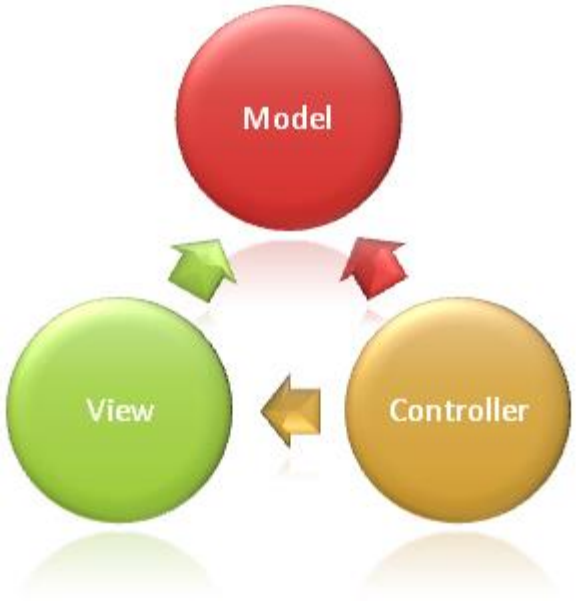
More About MVC Usage

- **First usage**: Trygve Reenskaug introduced MVC in the 1970.
- Traditionally used for desktop graphical user interfaces (GUIs),
- Then this pattern has become popular for designing web and mobile applications
- Popular programming languages like
  - JavaScript,
  - Python,
  - Perl,
  - Object Pascal/Delphi,
  - Ruby,
  - PHP,
  - Java,
  - C#,
  - Swift

have their integrated **MVC frameworks** that are used for web or mobile application development.

ASP.NET Core MVC

- The ASP.NET Core MVC framework is a lightweight**, open source**, highly testable presentation framework optimized for use with ASP.NET Core.
- Has the following Features
  - Routing
  - Model binding
  - Model validation
  - Dependency injection
  - Filters
  - Areas
  - Web APIs
  - Testability
  - Razor view engine
  - Strongly typed views
  - Tag Helpers
  - View Components



Working Environment

- In this course we use Development IDE as Visual Studio Professional 2019
- Student can use one of the following options

First option:

- Visual Studio Community (Free) : https://visualstudio.microsoft.com/tr/thank-you-downloading-visual-studio/?sku=Community&rel=16
- Visual Studio Professional (Paid)
- If we use Visual Studio we should check .Net Core SDK latest version componnent and installed.

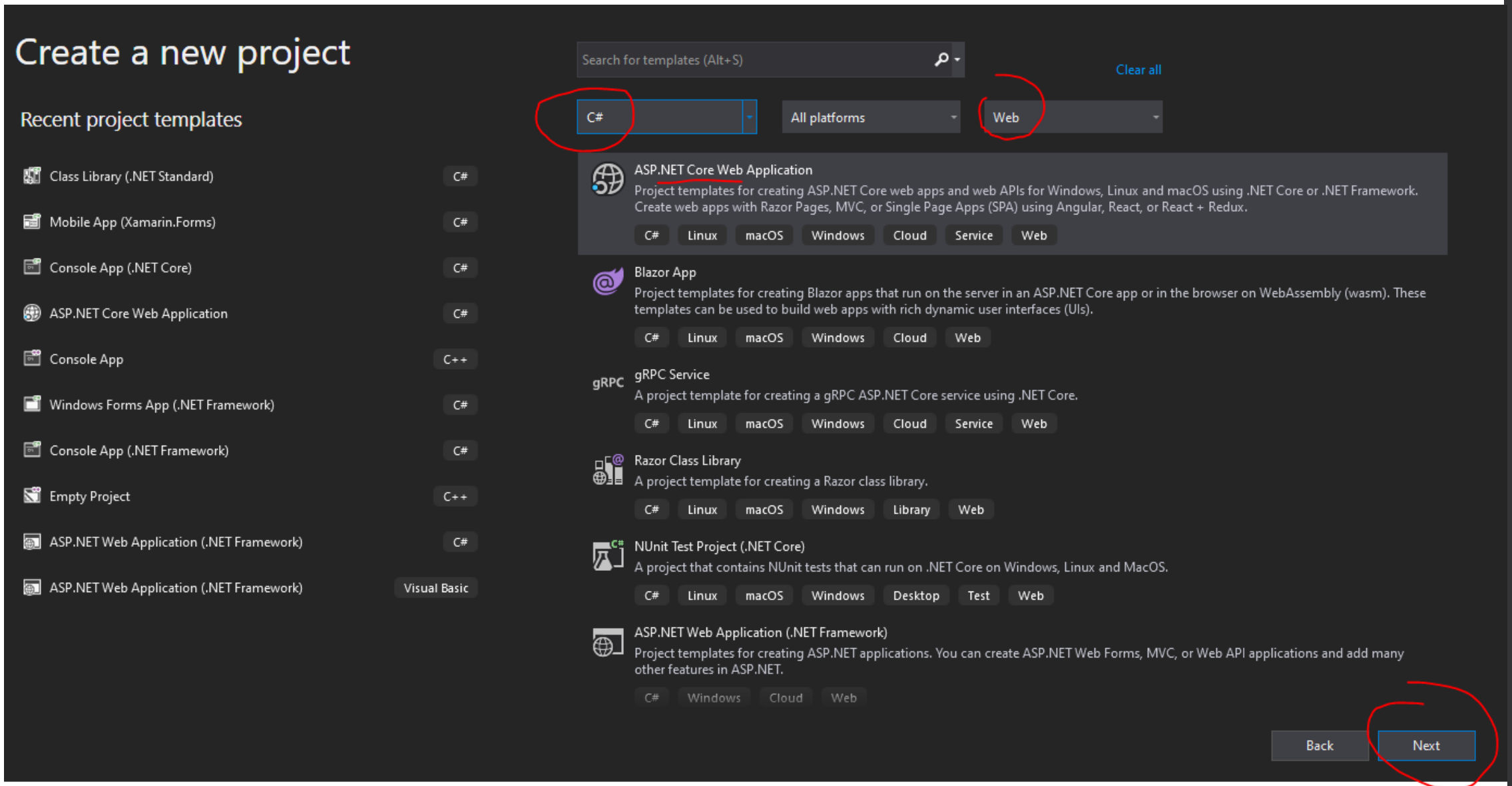
Second Option: Additionally IDE's (Optional)

- Must be Downloaded and installed .Net Core SDK 3.1 or 5.0 latest version, firstly: https://dotnet.microsoft.com/download/dotnet
- Visual Studio Code IDE (Free): https://code.visualstudio.com/docs/?dv=win
- Or JetBrains Rider (Free for Student if Registered): https://www.jetbrains.com/rider/download/download-thanks.html
- Or Visual Studio Community (Free): https://visualstudio.microsoft.com/tr/thank-you-downloading-visual-studio/?sku=Community&rel=16

Operation System:

- If you want to use Mac OS or Linux, the related IDE and Framework versions can be found on the same web sites given above.

Creating a new .Net Core Web Project



Configure your new project

ASP.NET Core Web Application C# Linux macOS Windows Cloud Service Web

Project name

FirstMVCWebApp

Location

C:\Users\asecerMonster\source\repos ...

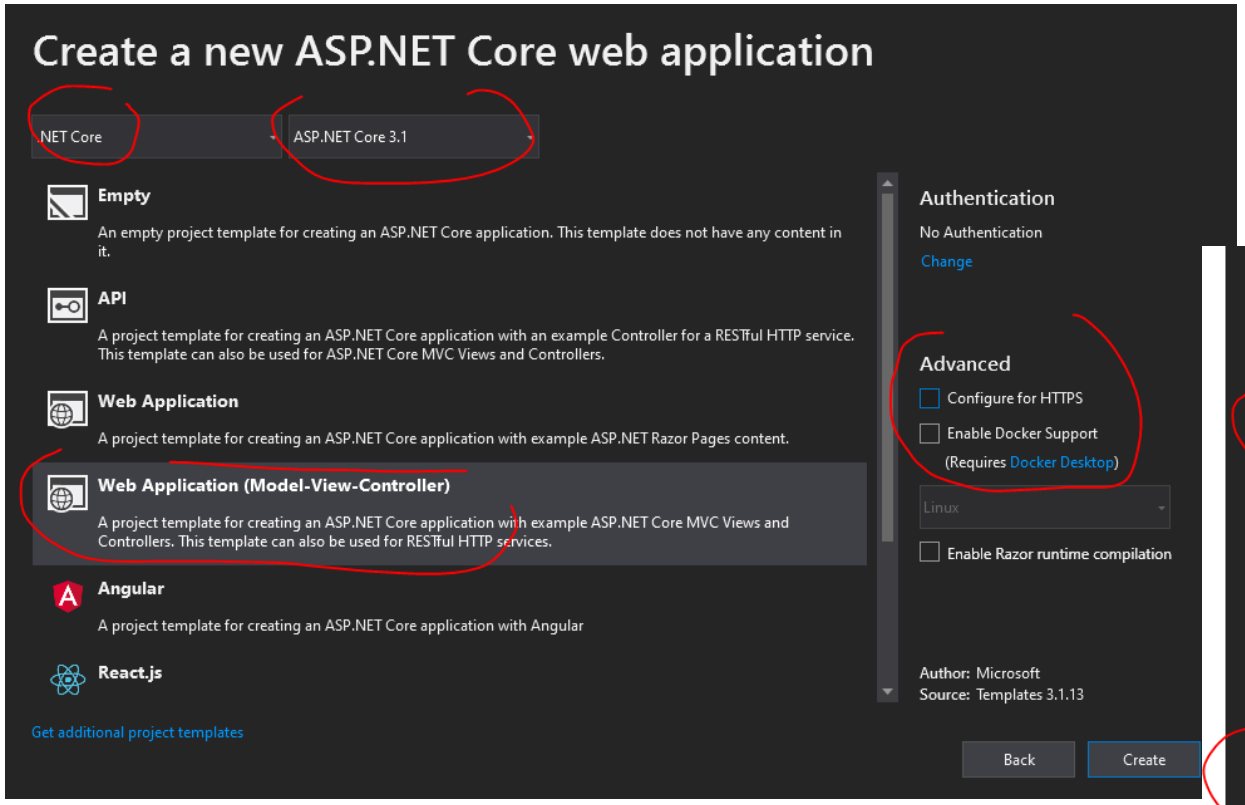
Solution name

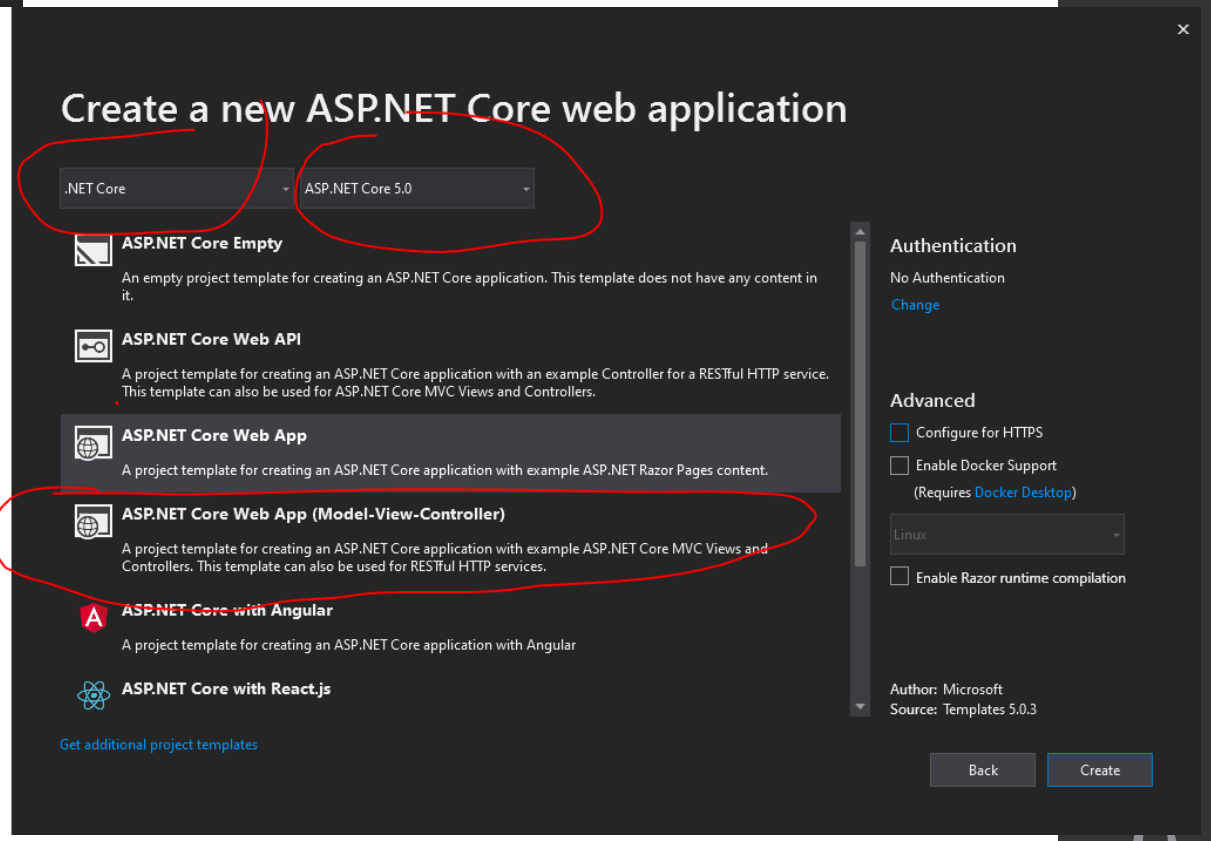
FirstMVCWebApp

Place solution and project in the same directory

Back Create

Core 3.1 or latest one Core 5.0 Frameworks

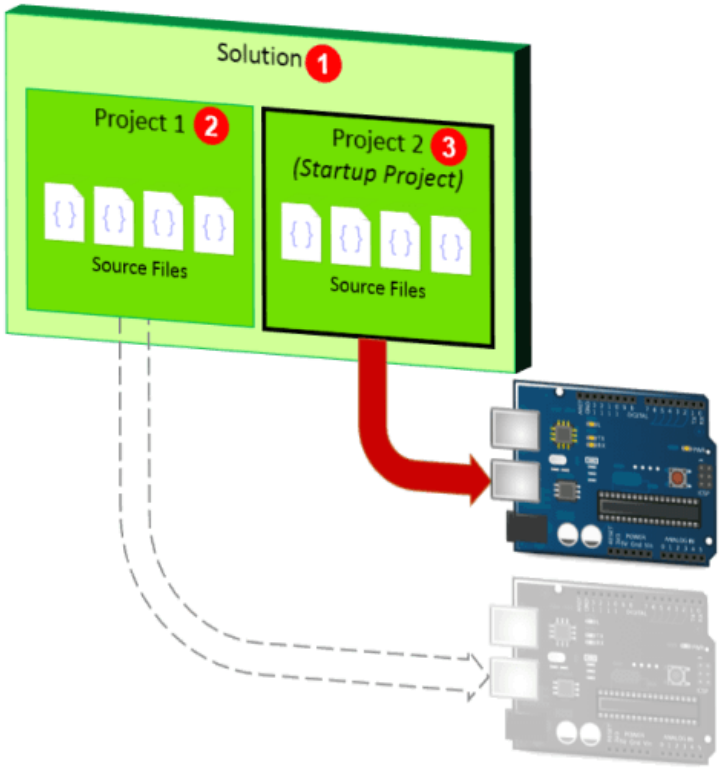


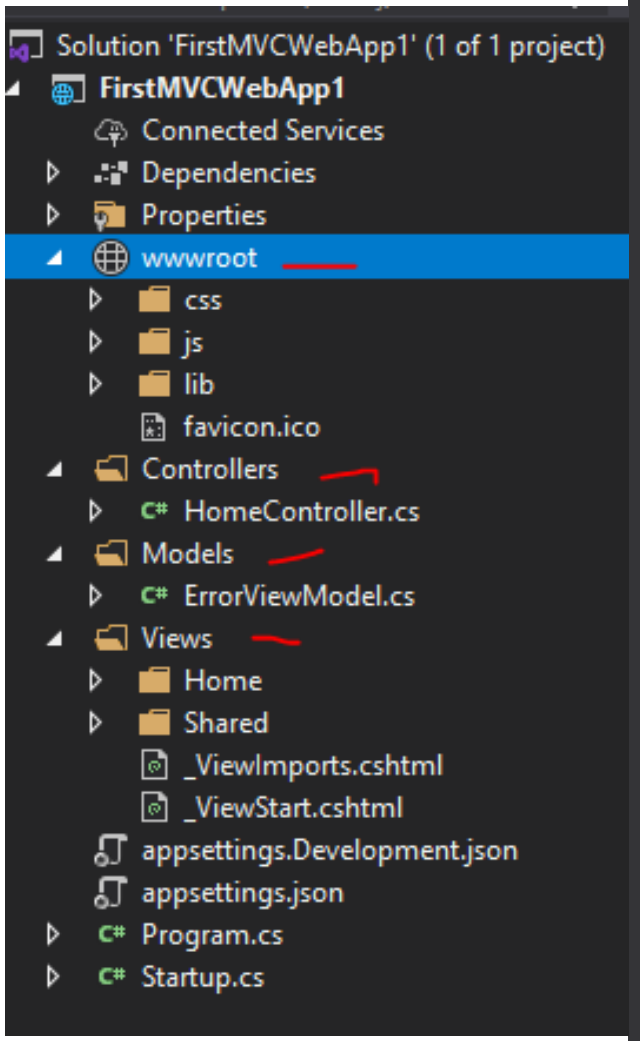


Project Base Folders

- **Controllers:** Contains clasess derived from ControllerBase class
- **Views:** Contains razor cshtml files
- **Models:** Contains domain and specific page models
- **wwwroot:** Contains static files (Javascripts, Css, Images etc…)

Note: Solution = Project (Or Projects Container)





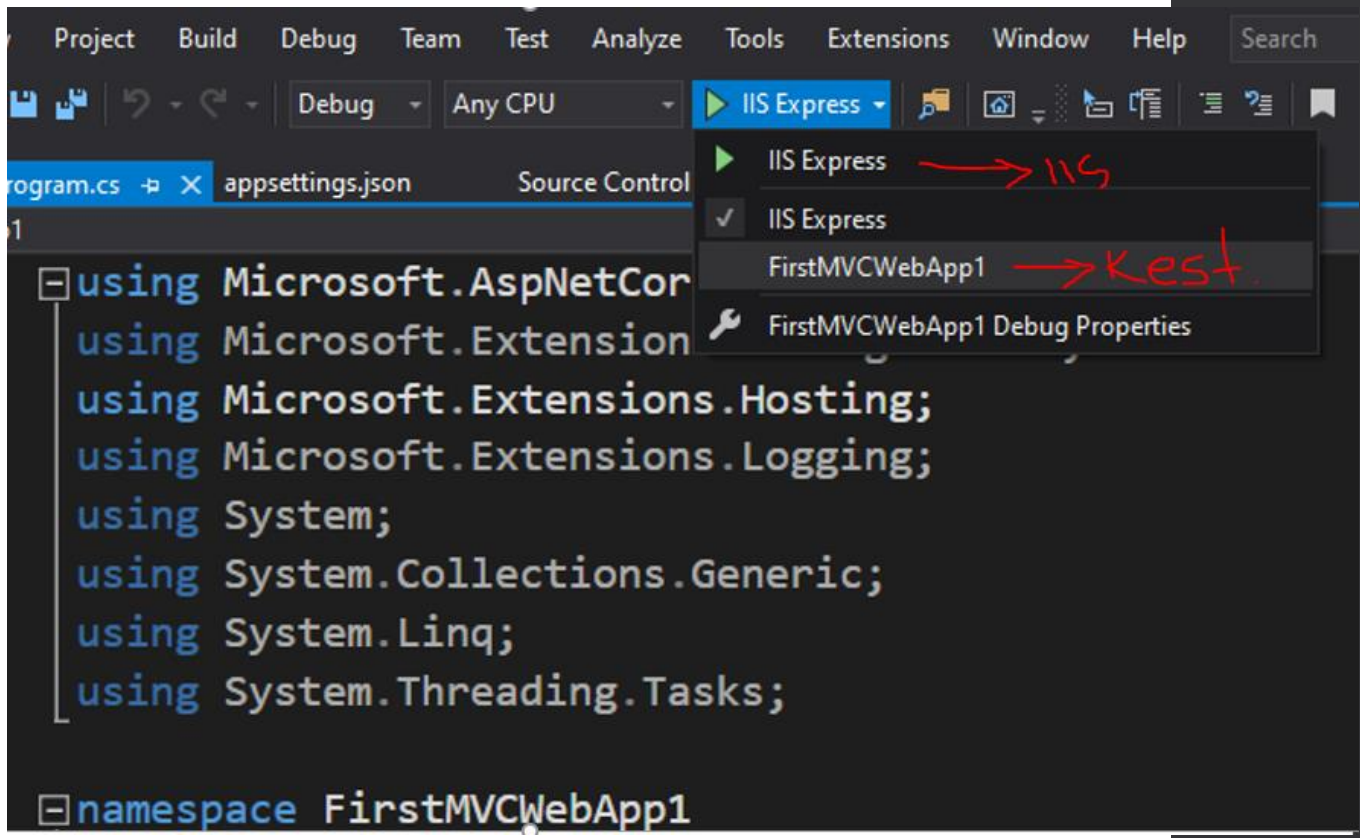
Important Config and Project Files

- **appsettings.json:** is an application configuration JSON file used to store configuration settings such as database connections strings, any application scope global variables, etc.
- **Program.cs:** is where the application starts. Program.cs class file is entry point of our application and creates an instance of IWebHost which hosts a web application.
- **Startup.cs:** ASP.NET Core application must include Startup class. It is like **Global.asax** in the traditional .NET application. As the name suggests, it is executed first when the application starts.

- **These files will be disscussed in detail, in the future…**

WEB App Testing Options

- **IIS Express Web Server:** IIS Express is a lightweight, self-contained version of IIS optimized for developers. IIS Express makes it easy to use the most current version of IIS to develop and test websites
- **Kestrel Web Server:** Kestrel is an open source, cross platform, light weight and a default webserver used for Asp.Net Core applications. Asp.Net Core applications run Kestrel webserver as in-process server to handle web request. Kestrel webserver is based on async I/O library called libuv primarily developed for Node.js.



Converting Empty Web Project to MVC

- Steps
  - Create a new Core 5.0 empty web Project
  - Create Controllers, Models, Views and wwwroot folders manually
  - In **Startup.cs** add following **service**

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

  - In **Startup.cs** add following **middlewares (we will discuss in future)**

```
app.UseRouting();

app.UseAuthorization();

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

Creating a Controller

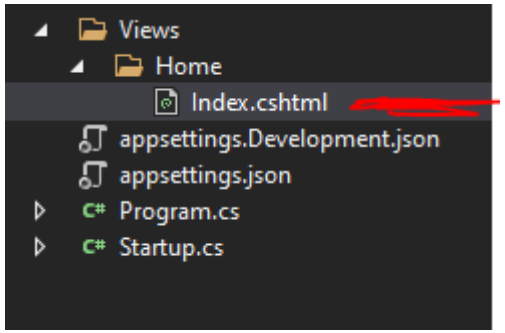
- Create a class in the Controllers folder such as:
  - HomeController.cs, the class name must be end with «Controller» always.
  - Use inheritance and inherid Base Controller Class from MVC as below
  - And add two public method as below

```
using Microsoft.AspNetCore.Mvc;

namespace MVCW01.Controllers
{
    0 references
    public class HomeController : Controller
    {

        0 references
        public string Index()
        {
            return "Index";
        }

        0 references
        public string About()
        {
            return "About";
        }
    }
}
```

Adding a View

- Create a index.cshtml file in the Views/Home folder such as:

```
<p>Hello world</p>
view
```



- Create an action method in the controller as below:

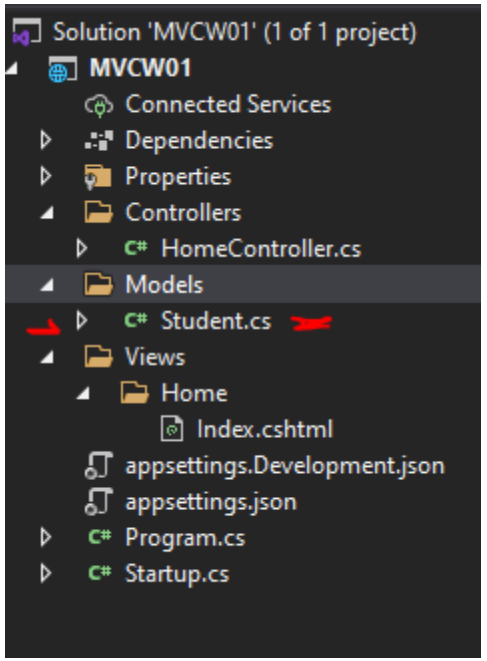
```
public class HomeController : Controller
{

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

- Run the application

Addding Model

- Add the following class in the Model Folder

```
public class Student
{
    public int StudentId { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public DateTime BirthDate { get; set; }
}
```

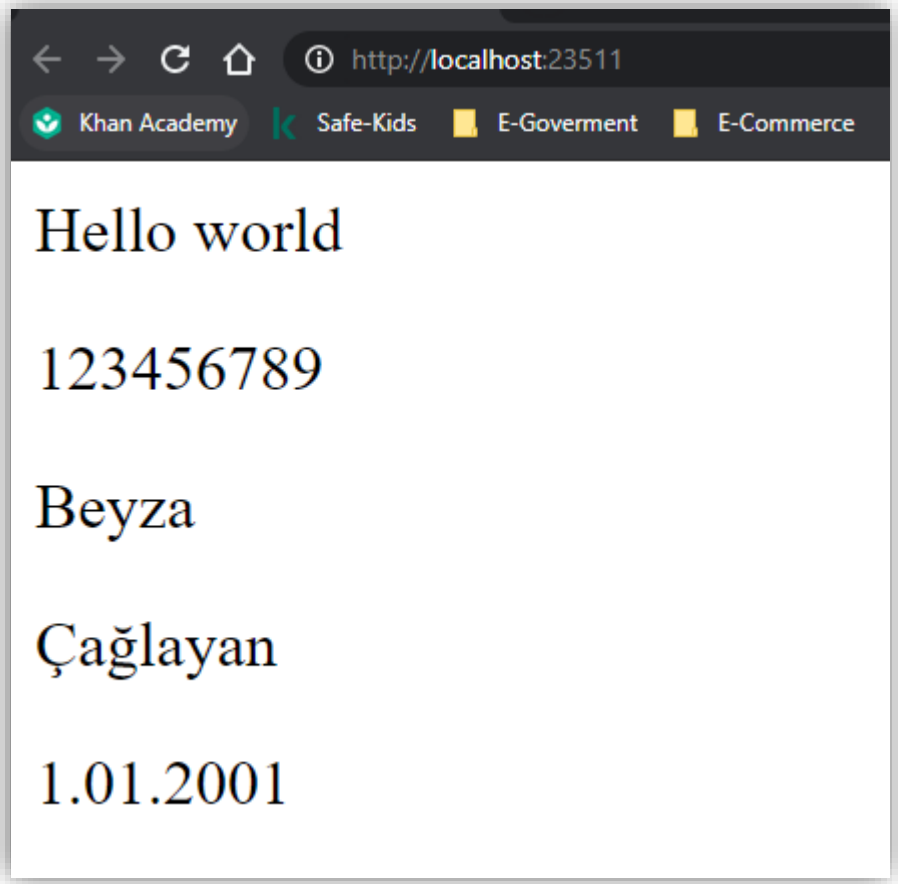


Using Model in Controller

```
public class HomeController : Controller
{
    0 references
    public IActionResult Index()
    {
        Student student = new Student
        {
            StudentId = 123456789,
            FirstName = "Beyza",
            LastName = "Çağlayan",
            BirthDate = new DateTime( year: 2001, month: 01, day: 01)


        };

        return View(student);
    }
}
```

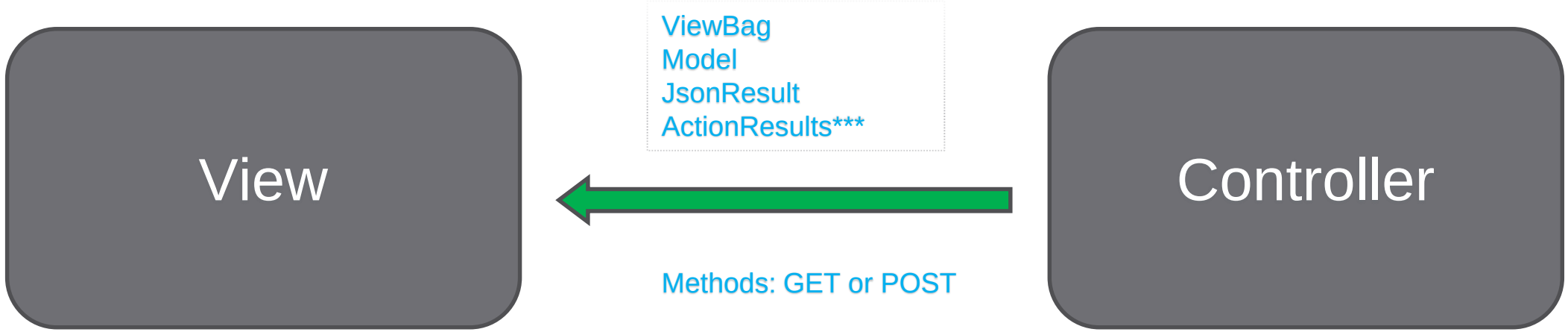
Using Model in Index.cshtml View

```
@model MVCW01.Models.Student

<p>Hello world</p>
<p>@Model.StudentId</p>
<p>@Model.FirstName</p>
<p>@Model.LastName</p>
<p>@Model.BirthDate.ToShortDateString()</p>
```



Data Trasfer-1: From Controller to View

- Transfer types and components:



Using ViewBag

Action

```
public IActionResult Index1()
{
            ViewBag.Name = "Ali";
            ViewBag.Surname = "Poyraz";
            return View();
}
```

view1

```
@{
    Layout = null;
}
<html>
<body>
<h2>ViewBag</h2>
<div>
    <table border="1">
        <thead>
        <tr style="color:red">
            <th>Name</th>
            <th>Surname</th>
        </tr>
        </thead>
        <tbody>
        <tr>
            <th>@ViewBag.Name</th>
            <th>@ViewBag.Surname</th>
        </tr>
        </tbody>
    </table>
</div>
</body>
</html>
```

view2

```
@{
    Layout = null;
    string name = ViewBag.Name;
    string surName = ViewBag.Surname;
}
<html>
<body>
    <h2>ViewBag</h2>
    <div>
        <table border="1">
            <thead>
                <tr
style="color:red">
                    <th>Name</th>
                    <th>Surname</th>
                </tr>
            </thead>
            <tbody>
                <tr>
                    <th>@name</th>
                    <th>@surName</th>
                </tr>
            </tbody>
        </table>
    </div>
</body>
</html>
```

Using Model

Action

```
public IActionResult Index2()
 {
            var student = new Student()
            {
                Name = "Ahmet",
                Surname = "Karaca"
            };
            return View(student);
 }
```

Model

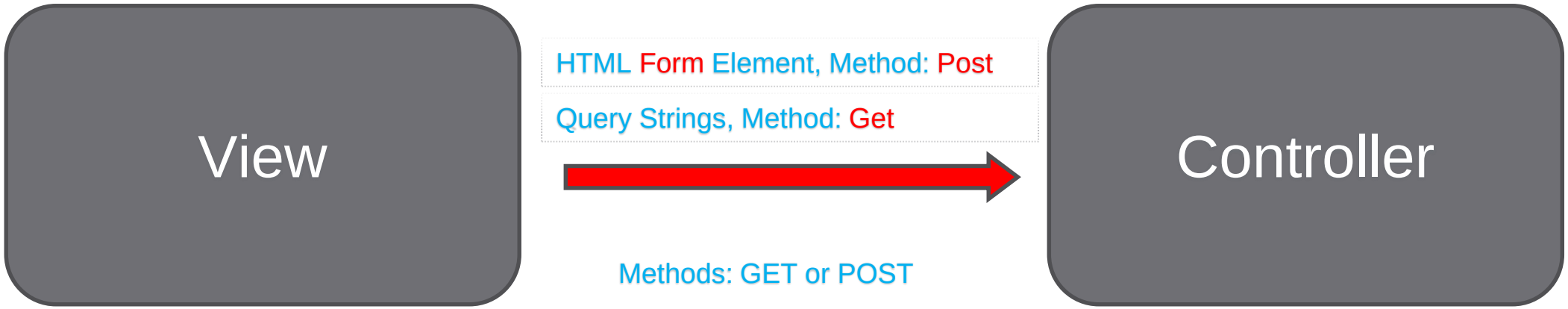
```
namespace MvcW02.Models
{
    public class Student
    {
        public string Name { get; set; }
        public string Surname { get; set; }
    }
}
```

//View

```
@model MvcW02.Models.Student

<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Index</title>
</head>
<body>
<h2>Model</h2>
<div>
    <table border="1">
        <thead>
        <tr style="color:red">
            <th>Name</th>
            <th>Surname</th>
        </tr>
        </thead>
        <tbody>
        <tr>
            <th>@Model.Name</th>
            <th>@Model.Surname</th>
        </tr>
        </tbody>
    </table>
</div>
</body>
</html>
```

Data Trasfer-2: From View to Controller

- Transfer types and components:



Using Query String

Url + Query String = https://localhost:44340/Home/Index3?x=5&y=15

```
public IActionResult Index3(int x, int y)
{
            var res = x + y;

            return View(res);
}
```

```
//View

@model int
@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport"
          content="width=device-width" />
    <title>Index3</title>
</head>
<body>
<h1>Query String</h1>
<div>
    Sum of x and y, x + y = @Model
</div>
</body>
</html>
```

Using HTML Form Element with Parameters

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

[HttpPost]
public IActionResult Register(string name, string surName)
{
    var student = new Student
    {
        Name = name,
        Surname = surName
    };
    //Db.Save(student) in future we will discuss
    ViewBag.Info = "Success: Student info saved to db";
    return View(student);
}
```

```
@{
    Layout = null;
}
<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Register</title>
</head>
<body>
    <form action="/Home/Register" method="post">
        Name <input type="text" name="name" />
        <br/>
        Sur Name <input type="text" name="surName" />
        <br/>
        <input type="submit" value="Save"/>

        <br/>
        <h3 style="color: green">@ViewBag.Info</h3>
    </form>
</body>
</html>
```

Using HTML Form Element with Model

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

[HttpPost]
public IActionResult Register(Student model)
{
    var student = model;
    //Db.Save(student) in future we will discuss
    ViewBag.Info = "Success: Student info saved to db";
    return View(student);
}
```

```
@model MvcW02.Models.Student
@{
    Layout = null;
}
<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Register</title>
</head>
<body>
    <form action="/Home/Register" method="post">
        Name <input type="text" name="name" />
        <br/>
        Sur Name <input type="text" name="surName" />
        <br/>
        <input type="submit" value="Save"/>

        <br/>
        <h3 style="color: green">@ViewBag.Info</h3>
    </form>
</body>
</html>
```

Filter Attributes, [HttpGet] and [HttpPost]

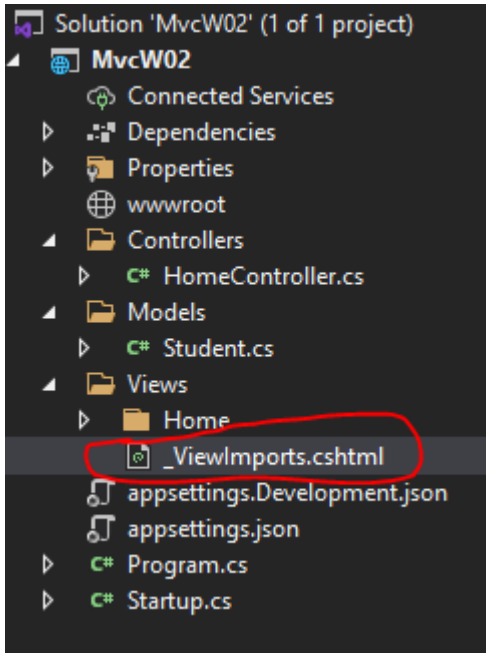
```
[HttpGet] // basically used to get data from the server
public IActionResult Register()
{
    return View();
}

[HttpPost] // basically used to send data to the server/ needs html form element
public IActionResult Register(string name, string surName)
{
    var student = new Student
    {
        Name = name,
        Surname = surName
    };
    //Db.Save(student) in future we will discuss
    ViewBag.Info = "Success: Student info saved to db";
    return View(student);
}
```

Listing Data

```
[HttpGet]
 public IActionResult GetStudentList()
 {
     var studentList = new List<Student>
     {
         new Student(){Name = "Ayşe", Surname = "Karadağ"},
         new Student(){Name = "Akif", Surname = "Eren"},

     };
     return View(studentList);
 }
```

```
@model IEnumerable<MvcW02.Models.Student>
@{
    Layout = null;
}
<html>
<body>
    <p>List </p>
    <table border="1">
        <thead>
            <tr>
                <th>Name</th>
                <th>Surname</th>
            </tr>
        </thead>
        <tbody>
            @foreach (var item in Model)
            {
                <tr>
                    <td>@item.Name</td>
                    <td>@item.Surname</td>
                </tr>
            }
        </tbody>
    </table>
</body>
</html>
```

Tag Helper




```
@namespace MvcW02
@using Models

@addTagHelper  *, Microsoft.AspNetCore.Mvc.TagHelper
```

We dont need fully qualified name for model

```
@model IEnumerable<Student>
@{
    Layout = null;
}
<html>
<body>
```

Simply Validations on Models

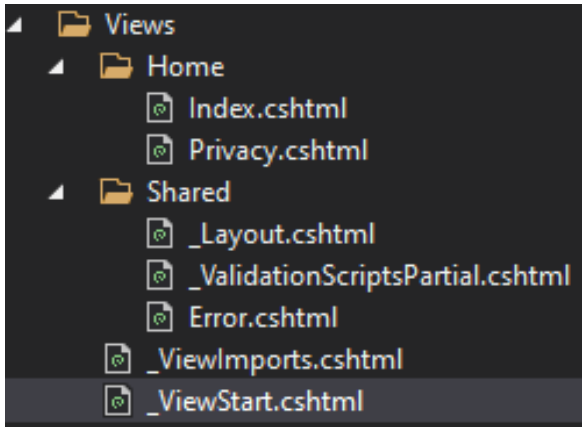
```
[HttpPost]
public IActionResult Register(Student model)
{
    var student = model;
    //Db.Save(student) in future we will discuss
    var state = ModelState.IsValid;
    if (state)
       ViewBag.Info = "Success: Student info saved to db";
    else
       ViewBag.Info = "Error";

    return View(student);
}
```

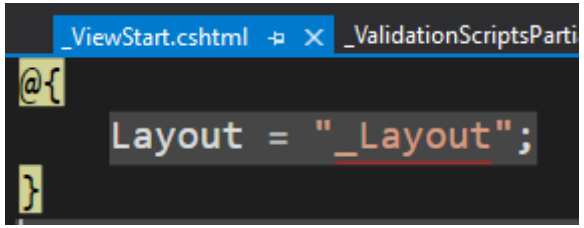
```
public class Student
{
    [Key]
    [Required]
    public int Id { get; set; }

    [Required(ErrorMessage = "Cannot be empty!")]
    [MinLength(2)]
    [MaxLength(20)]
    public string Name { get; set; }

    [
        Required,
        MinLength(2, ErrorMessage = "Minimum 2 character required!"
        MaxLength(20)
    ]
    public string Surname { get; set; }
}
```

```
<form asp-controller="Home" asp-action="Register" method="post">

        <div asp-validation-summary="All"></div>

        Name <input type="text" name="name" />
        <br />
        Sur Name <input type="text" name="surName" />
        <br />
        <input type="submit" value="Save" />

        <br />
        <h3 style="color: green">@ViewBag.Info</h3>
    </form>
```

Some Fundamental Topics and VSCode

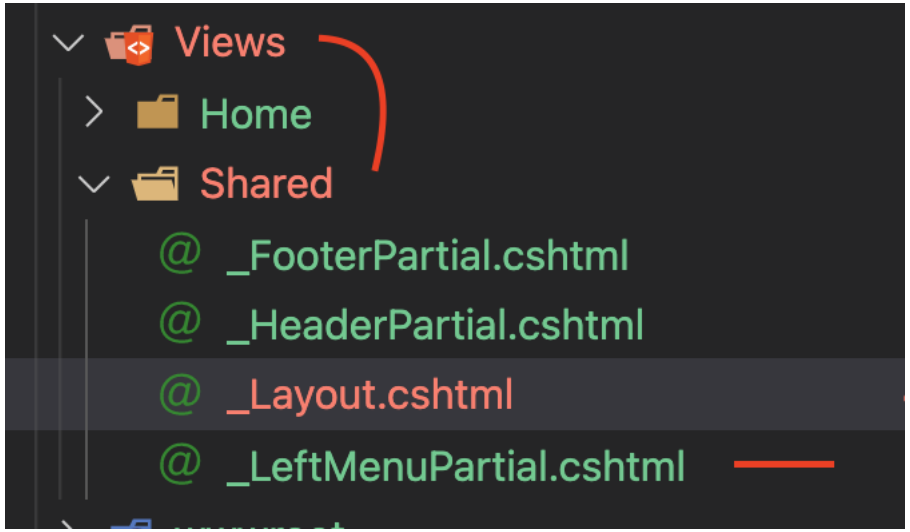
- VSCode, introduction
- Multiple Routes and Attribute Routing
- Razor Syntax
- HTML Helper, Helper Methods
- URL Helper
- Layouts (_Layout.cshtml)
- ViewStart (_ViewStart.cshtml)
- ViewImports
- Boostrap Library
- JQuery Library
- Static files and wwwroot folder
- Partial Views (@{Html.RenderPartial("_partial1.cshtml");} , @Html.Partial("_partial2.cshtml")



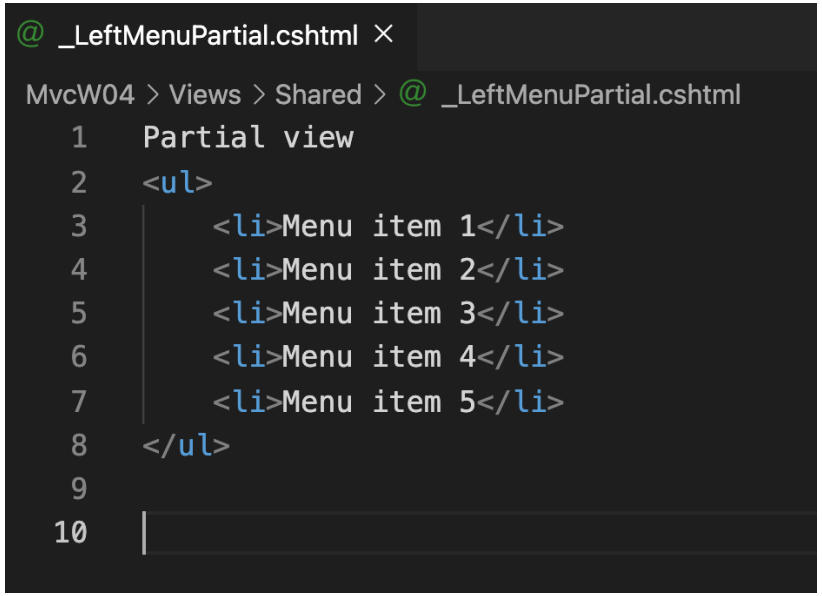


Partial Views

Location



Content




```
Using: Possible from everywhere (in cshtml)

<div style="width: 15%; background-color: red; float:
left;clear: left; height: 600px">

Left Menu

     @Html.Partial("_LeftMenuPartial")


</div>
```

Partial Views- Synchronous HTML Helper

```
<body>

@Html.Partial("_HeaderPartial")


<div class="container bg-danger">
    <main role="main" class="pb-3">
        @RenderBody()
    </main>
</div>

@Html.Partial("_FooterPartial")

</body>
```

OR

```
<body>
    @{
        Html.RenderPartial("_HeaderPartial");
    }
    <div class="container bg-danger">
        <main role="main" class="pb-3">
            @RenderBody()
        </main>
    </div>

    @{
        Html.RenderPartial("_FooterPartial");
    }
</body>
```

Partial Views- Asynchronous HTML Helper

```
<body>
    @await Html.PartialAsync("_HeaderPartial")

    <div class="container bg-danger">
        <main role="main" class="pb-3">
            @RenderBody()
        </main>
    </div>

    @await Html.PartialAsync("_FooterPartial")
</body>
```

```
<body>
    @{
        await Html.RenderPartialAsync("_HeaderPartial");
    }
    <div class="container bg-danger">
        <main role="main" class="pb-3">
            @RenderBody()
        </main>
    </div>

    @{
        await Html.RenderPartialAsync("_FooterPartial");
    }
</body>
```

Partial Views- Partial Tag Helper

Default Path:

```
<body>

    <partial name="_HeaderPartial" />

    <div class="container bg-danger">
        <main role="main" class="pb-3">
            @RenderBody()
        </main>
    </div>

    <partial name="_FooterPartial" />

</body>
```

Specific Path:

```
<body>

    <partial name="~/Views/Shared/_HeaderPartial.cshtml" />

    <div class="container bg-danger">
        <main role="main" class="pb-3">
            @RenderBody()
        </main>
    </div>

    <partial name="~/Views/Shared/_FooterPartial.cshtml" />

</body>
```

Partial Views- with Dynamic Data, (any Models)

_Layout.cshtml:

```
<body>

    <partial name="_HeaderPartial" model="@(new
List<string>(){"Button1","Button2","Button3",})" />

    <div class="container bg-danger">
        <main role="main" class="pb-3">
            @RenderBody()
        </main>
    </div>

    <partial name="_FooterPartial" />

</body>
```

_HeaderPartial.cshtml:

```
@model List<string>
<header>

<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="Home" asp-action="Privacy">Privacy</a>
</li>
          @foreach (var btn in Model)
          {
          <li class="nav-item">
          <a class="nav-link text-dark" asp-area="" asp-
controller="Home" asp-action="xx">@btn</a>
          </li>
          }

</ul>
```

Sections (Conditionally adding scripts or CSS)

Index.cshtml:

```
@{
    ViewData["Title"] = "Index";
}

@section Scripts{
    <link rel="stylesheet"
href="~/lib/bootstrap/dist/css/bootstrap.min.css" />
    <link rel="stylesheet" href="~/css/site.css" />
    <script src="~/lib/jquery/dist/jquery.min.js"></script>
    <script
src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js"></scri
pt>
    <script src="~/js/site.js" asp-append-
version="true"></script>
}
```

Layout.cshtml: required true or false???

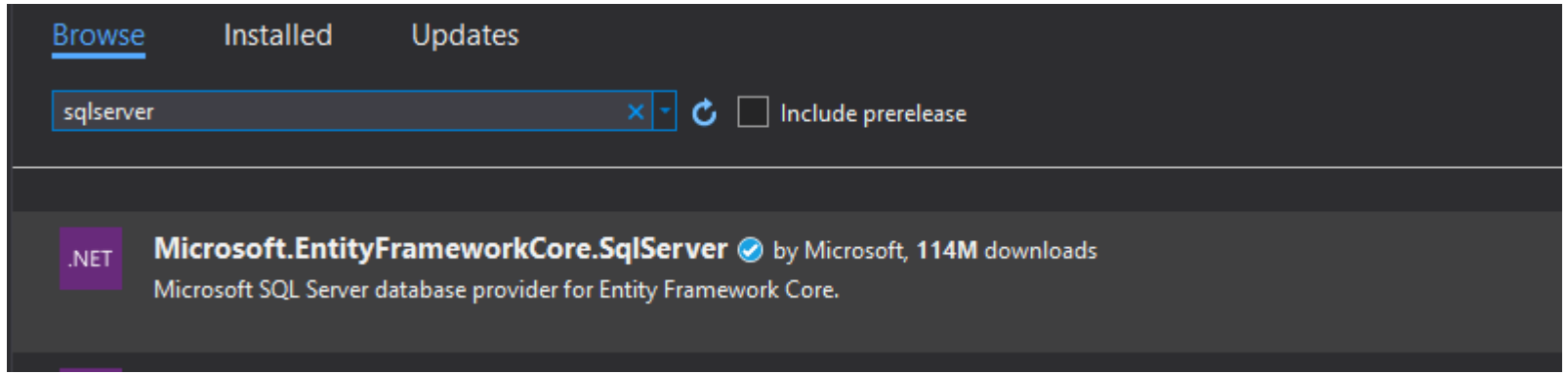
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width,
initial-scale=1.0" />
    <title>@ViewData["Title"] - MvcW04</title>

    @await RenderSectionAsync("Scripts", required: true)

</head>
```

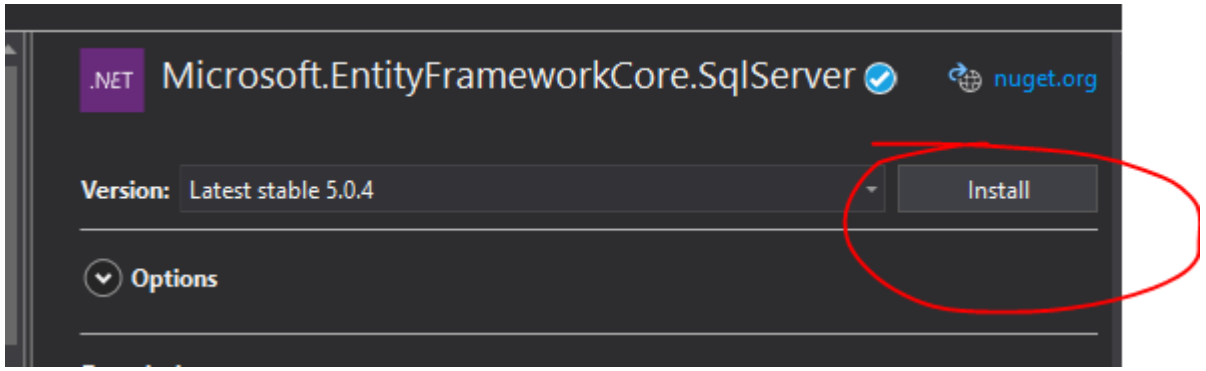
Working With Entity Framework Core Technology

- **School** Database

  Schema

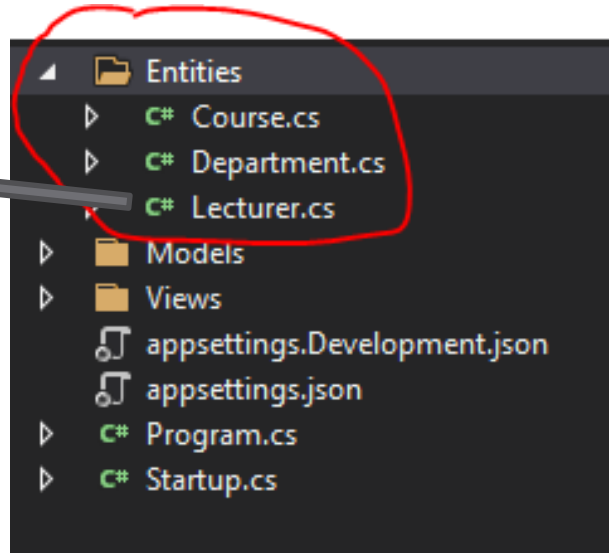
Entity FrameworkCore SQL Nuget Package





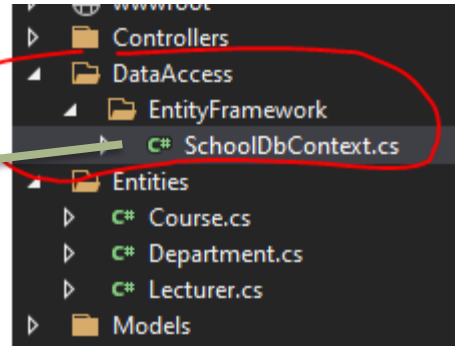
Creating Entity Layer for Db Schema

```
namespace MvcW04.Entities
{
    0 references
    public class Lecturer
    {
        [Key]
        0 references
        public int Id { get; set; }

        [ForeignKey(name:"DepartmentId")]
        0 references
        public int DepartmentId { get; set; }

        [ForeignKey(name:"DepartmentId")]
        0 references
        public  Department Department { get; set; }
        0 references
        public string Name { get; set; }
    }
}
```



Creating DataAccess Layer for Db Schema

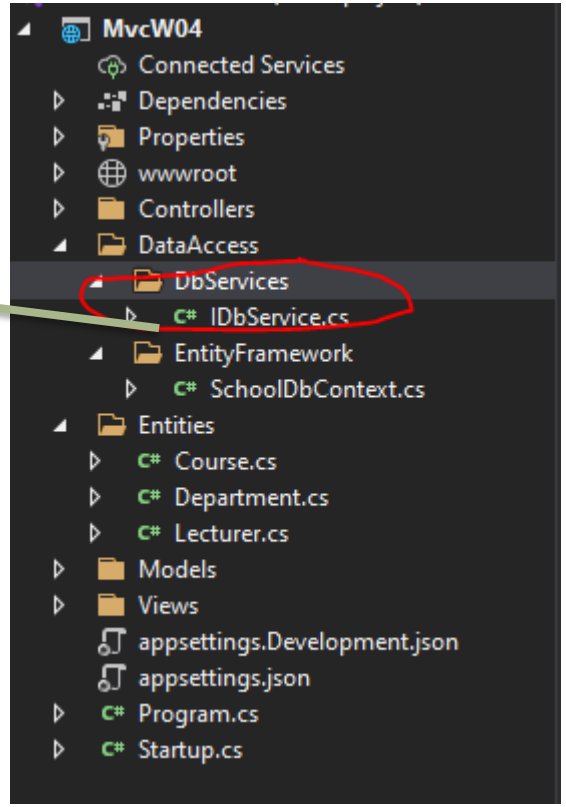
Inheriting DbContext Class, Creating SchoolDbContext



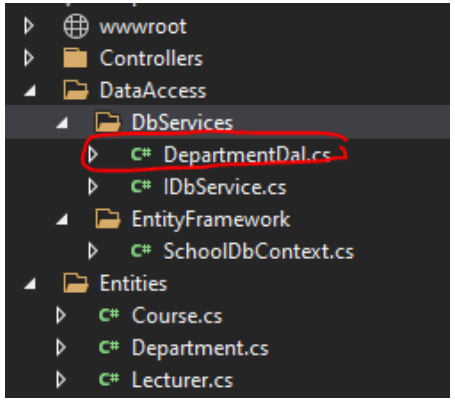

```
namespace MvcW04.DataAccess.EntityFramework
{
    public class SchoolDbContext:DbContext
    {
        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
        {
            optionsBuilder.UseSqlServer(
                @"data source=.;initial catalog=School;
                    Trusted_Connection=True;Max Pool Size=32767;
                    MultipleActiveResultSets=True;");
        }
        public DbSet<Department> Departments { get; set; }

        public DbSet<Lecturer> Lecturers { get; set; }

        public DbSet<Course> Courses { get; set; }
    }
}
```

IDbService Generic interface for CRUD

```
namespace MvcW04.DataAccess.DbServices
{
    public interface IDbService<T> where T: class, new ()
    {
        T Get(int id);
        T Add(T entity);
        void Remove(T entity);
        void Update(T entity);
        List<T> GetList();
    }
}
```



Dal Services: DepartmentDal




```
public class DepartmentDal:IDbService<Department>
    {
        public Department Get(int id)
        {
            using (var context = new SchoolDbContext())
            {
                var record =
context.Set<Department>().FirstOrDefault(p=>p.Id==id);
                return record;
            }
        }
        public void Update(Department entity)
        {
            using (var context = new SchoolDbContext())
            {
                var added = context.Entry(entity);
                added.State = EntityState.Modified;
                context.SaveChanges();
            }
        }
```

```
        public List<Department> GetList()
        {
            using (var context = new SchoolDbContext())
            {
                return context.Set<Department>().ToList();
            }
        }

        public Department Add(Department entity)
        {
            using (var context = new SchoolDbContext())
            {
                var added = context.Entry(entity);
                added.State = EntityState.Added;
                context.SaveChanges();
                return entity;
            }
        }
        public void Delete(Department entity)
        {
            using (var context = new SchoolDbContext())
            {
                var updated = context.Entry(entity);
                updated.State = EntityState.Modified;
                context.SaveChanges();
            }
        }
    }
```

Using Dal Service in Controller as Concrete without Dependency Injection

```
public class HomeController : Controller
    {
        private readonly DepartmentDal _departmentDal;

        public HomeController()
        {
            _departmentDal = new DepartmentDal();
        }

        public IActionResult Index()
        {
            var departmentList = _departmentDal.GetList();

            return View();
        }
    }
```

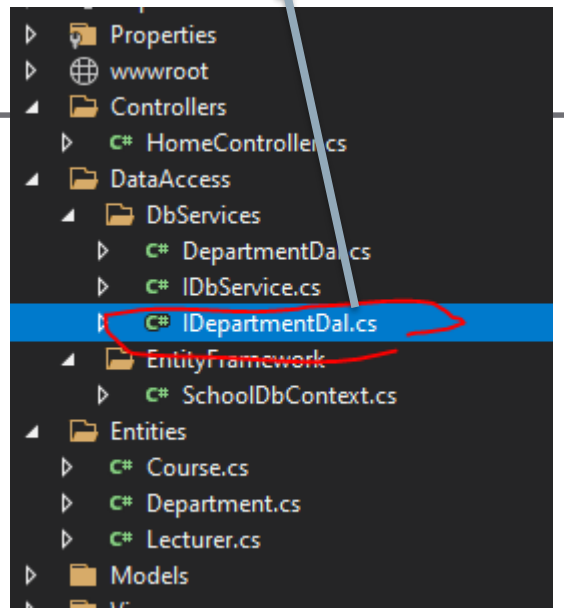
Using Dal Service in Controller as Abstract with Dependency Injection (Controller level)

Startup.cs:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddSingleton<IDepartmentDal,DepartmentDal>();

    services.AddControllersWithViews();
}
```

```
public class HomeController : Controller
{
    private readonly IDepartmentDal _departmentDal;
    public HomeController(IDepartmentDal departmentDal)
    {
        _departmentDal = departmentDal;
    }
    public IActionResult Index()
    {
        var departmentList = _departmentDal.GetList();
        return View();
    }
}
```

```
namespace MvcW04.DataAccess.DbServices
{
    public interface IDepartmentDal:IDbService<Department>
    {
    }
}
```



Using Dal Service in Controller as Abstract with Dependency Injection (Action level)

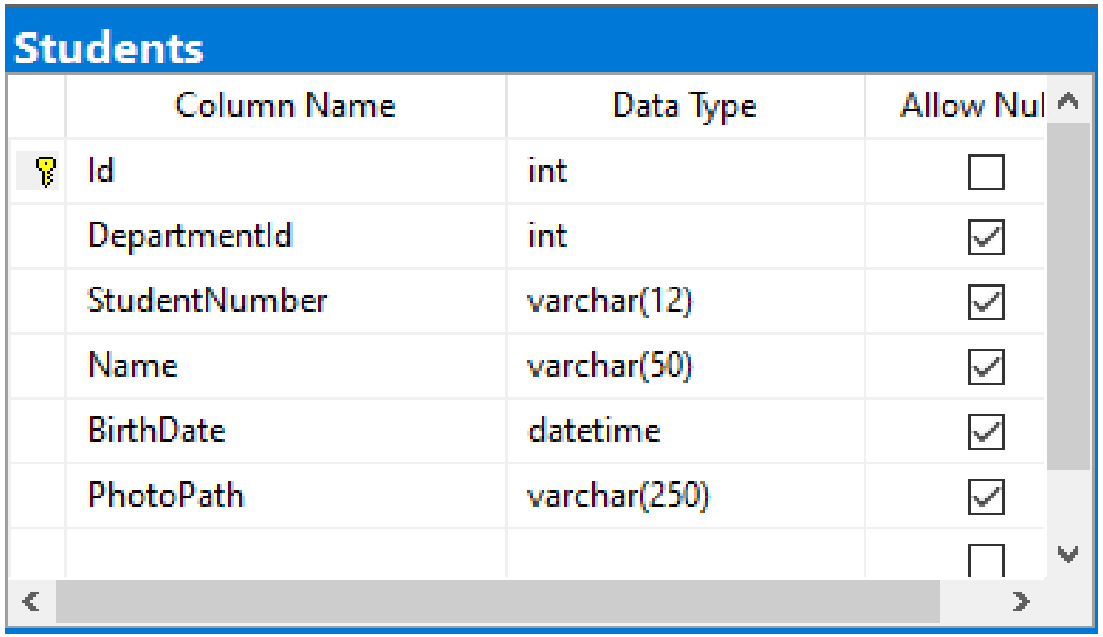
```
public class HomeController : Controller
    {

        public IActionResult Index([FromServices] IDepartmentDal departmentDal)
        {
            var departmentList = departmentDal.GetList();
            return View();
        }
    }
```

CRUD Operations for School Database

CREATE
READ
UPDATE
DELETE

CRUD Operations for Students Table

Database Table



Entity

```
public class Student
{
    [Key]
    public int Id { get; set; }
    [ForeignKey("DepartmentId")]
    public int DepartmentId { get; set; }
    [ForeignKey("DepartmentId")]
    public Department Department { get; set; }
    public string StudentNumber { get; set; }
    public string Name { get; set; }
    public DateTime? BirthDate { get; set; }
    public string PhotoPath { get; set; }
}
```

IStudent interface and StudentDal class

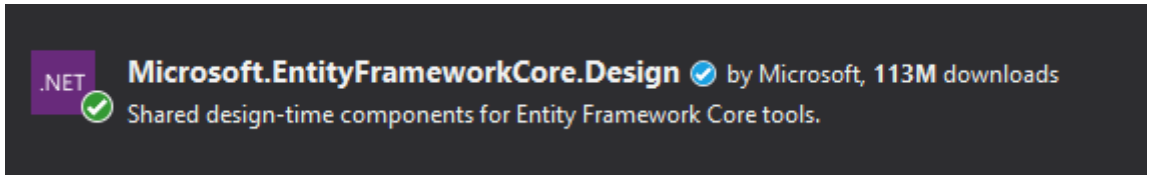
```
public interface IStudentDal : IDbService<Student>
{
}
```

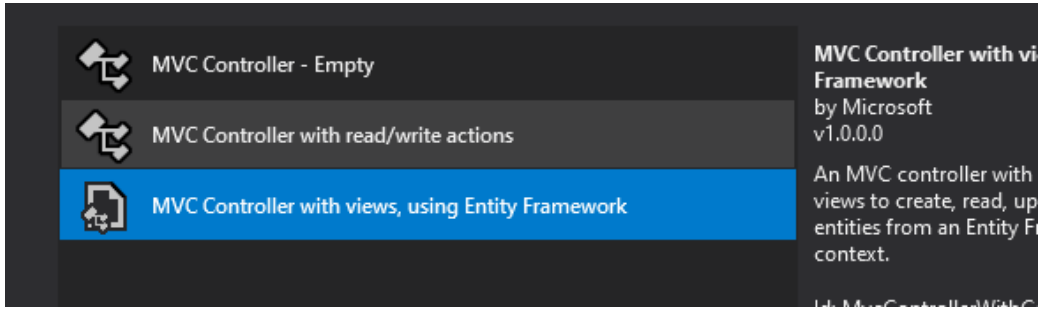
```
public class StudentDal:IStudentDal
{
    public Student Get(int id)
    {
        using (SchoolContext context = new SchoolContext())
        {
            return context.Set<Student>().FirstOrDefault(p => p.Id == id);
        }
    }

    public Student Add(Student entity)
    {
        using (SchoolContext context = new SchoolContext())
        {

            var added = context.Entry(entity);
            added.State = EntityState.Added;
            context.SaveChanges();

            return entity;//for tracking generated pk.
        }
    }
:
:
```

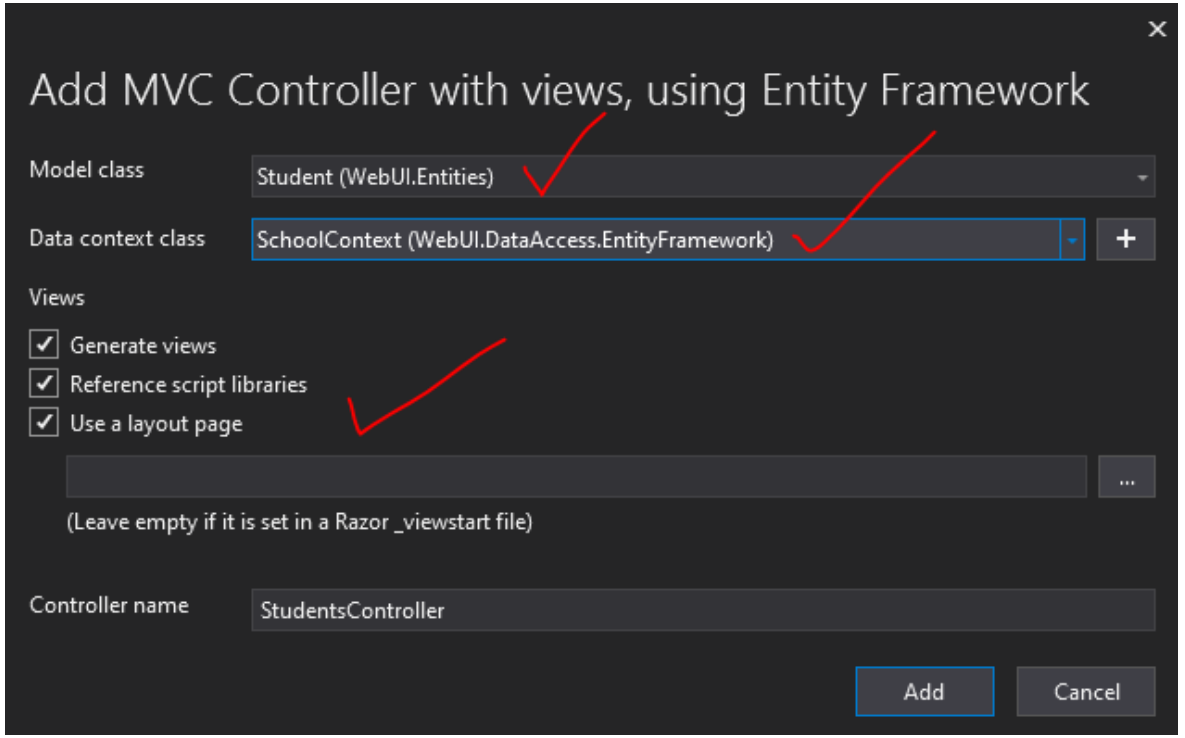
```
        public void Remove(Student entity)
        {
            using (SchoolContext context = new SchoolContext())
            {

                var added = context.Entry(entity);
                added.State = EntityState.Deleted;
                context.SaveChanges();
            }
        }

        public void Update(Student entity)
        {
            using (SchoolContext context = new SchoolContext())
            {

                var added = context.Entry(entity);
                added.State = EntityState.Modified;
                context.SaveChanges();
            }
        }

        public List<Student> GetList()
        {
            using (SchoolContext context = new SchoolContext())
            {

                return context.Set<Student>().ToList();
            }
        }
    }
```

Add Student Controller (Auto Generated)

- Nuget installation
- Add Controller
- Options







Dependency Injection

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

    services.AddSingleton<IDepartmentDal, DepartmentDal>();

    services.AddSingleton<IStudentDal, StudentDal>();
}
```

```
public class StudentsController : Controller
{
    private readonly IStudentDal _studentDal;
    private readonly IDepartmentDal _departmentDal;

    0 references
    public StudentsController(IStudentDal studentDal, IDepartmentDal departmentDal)
    {
        _studentDal = studentDal;
        _departmentDal = departmentDal;
    }
```

Some Modifications on Interfaces, Classes an Controllers

```
//IDbService Modification
2 references
public interface IDbService<T> where T : class, new()
{
    4 references
    T Get(int id);

    3 references
    T Add(T entity);

    3 references
    void Remove(T entity);

    2 references
    void Update(T entity);

                    //Linq filter as predicate param
    4 references
    List<T> GetList(Expression<Func<T, bool>> filter = null);
}
```

```
//StudentDal Modification
3 references
public List<Student> GetList(Expression<Func<Student, bool>> filter = null)
{
    using (SchoolContext context = new SchoolContext())
    {

        return filter == null ?
            context.Set<Student>().Include("Department").ToList() :
            context.Set<Student>().Include("Department").Where(filter).ToList();

    }
}
```

```
//DepartmentDal modification
0 references
public List<Department> GetList(Expression<Func<Department, bool>> filter = null)
{
    using (SchoolContext context = new SchoolContext())
    {

        return filter == null ?
            context.Set<Department>().ToList() :
            context.Set<Department>().Where(filter).ToList();
    }
}
```

Students Controller

```
public class StudentsController : Controller
{
    private readonly IStudentDal _studentDal;
    private readonly IDepartmentDal _departmentDal;

    0 references
    public StudentsController(IStudentDal studentDal, IDepartmentDal departmentDal)
    {
        _studentDal = studentDal;
        _departmentDal = departmentDal;
    }

    [HttpGet]
    1 reference
    public IActionResult Index()
    {
        var studentList = _studentDal.GetList();
        return View(studentList);
    }

    [HttpGet]
    0 references
    public IActionResult Details(int id)...

    [HttpGet]
    0 references
    public IActionResult Create()
    {
        ViewData["DepartmentId"] = new SelectList(items: _departmentDal.GetList(), dataValueField: "Id", dataTextField: "Name");
        return View();
    }
```

```
[HttpPost]
[ValidateAntiForgeryToken]
0 references
public IActionResult Create(Student student)
{
    if (ModelState.IsValid)
    {
        _studentDal.Add(student);
        return RedirectToAction(nameof(Index));
    }
    ViewData["DepartmentId"] = new SelectList(items: _studentDal.GetList(), dataValueField: "Id", dataTextField: "Name", student.DepartmentId);
    return View(student);
}

[HttpGet]
0 references
public IActionResult Edit(int id)
{
    var student = _studentDal.Get(id);
    if (student == null)
    {
        return NotFound();
    }
    ViewData["DepartmentId"] = new SelectList(items: _departmentDal.GetList(), dataValueField: "Id", dataTextField: "Name", student.DepartmentId);
    return View(student);
}

[HttpPost]
[ValidateAntiForgeryToken]
0 references
public IActionResult Edit(Student student)
{

    if (ModelState.IsValid)
    {
        _studentDal.Update(student);
        return RedirectToAction(nameof(Index));
    }
    ViewData["DepartmentId"] = new SelectList(items: _departmentDal.GetList(), dataValueField: "Id", dataTextField: "Id", student.DepartmentId);
    return View(student);
}
```

Auto Generated Views

```
@model IEnumerable<WebUI.Entities.Student>

@{
    ViewData["Title"] = "Index";
}

<h1>Students</h1>

<p>
    <a asp-action="Create">Create New</a>
</p>

<table class="table">
    <thead>…</thead>
    <tbody>
        @foreach (var item :Student in Model)
        {
            <tr>
                <td>
                    @Html.DisplayFor( expression: modelItem :IEnumerable<Student> => item.Department.Name)
                </td>
                <td>
                    @Html.DisplayFor( expression: modelItem :IEnumerable<Student> => item.StudentNumber)
                </td>
                <td>
                    @Html.DisplayFor( expression: modelItem :IEnumerable<Student> => item.Name)
                </td>
                <td>
                    @Html.DisplayFor( expression: modelItem :IEnumerable<Student> => item.BirthDate)
                </td>
                <td>
                    @Html.DisplayFor( expression: modelItem :IEnumerable<Student> => item.PhotoPath)
                </td>
                <td>
                    <a asp-action="Edit" asp-route-id="@item.Id">Edit</a> |
                    <a asp-action="Details" asp-route-id="@item.Id">Details</a> |
                    <a asp-action="Delete" asp-route-id="@item.Id">Delete</a>
                </td>
            </tr>
        }
    </tbody>
</table>
```

Create

Date Picker Created Automatically and date format sepecified by browser defaults please choose Browser lang as Turkish.

```
@model WebUI.Entities.Student
@{
    ViewData["Title"] = "Create";
}
<h1>Create</h1>
<h4>Student</h4>
<hr />
<div class="row">
    <div class="col-md-4">
        <form asp-action="Create">
            <div asp-validation-summary="ModelOnly" class="text-danger"></div>
            <div class="form-group">
                <label asp-for="DepartmentId" class="control-label"></label>
                <select asp-for="DepartmentId" class="form-control" asp-items="ViewBag.DepartmentId"></select>
            </div>
            <div class="form-group">
                <label asp-for="StudentNumber" class="control-label"></label>
                <input asp-for="StudentNumber" class="form-control" required="required" />
                <span asp-validation-for="StudentNumber" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Name" class="control-label"></label>
                <input asp-for="Name" class="form-control" />
                <span asp-validation-for="Name" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="BirthDate" class="control-label"></label>
                <input asp-for="BirthDate" class="form-control" />
                <span asp-validation-for="BirthDate" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="PhotoPath" class="control-label"></label>
                <input asp-for="PhotoPath" class="form-control" />
                <span asp-validation-for="PhotoPath" class="text-danger"></span>
            </div>
            <div class="form-group">
                <input type="submit" value="Create" class="btn btn-primary" />
            </div>
        </form>
    </div>
</div>
```

Edit

```
@model WebUI.Entities.Student

<h1>Edit</h1>
<h4>Student</h4>
<hr />
<div class="row">
    <div class="col-md-4">
        <form asp-action="Edit">
            <div asp-validation-summary="ModelOnly" class="text-danger"></div>
            <input type="hidden" asp-for="Id" />
            <div class="form-group">
                <label asp-for="DepartmentId" class="control-label"></label>
                <select asp-for="DepartmentId" class="form-control" asp-items="ViewBag.DepartmentId"></select>
                <span asp-validation-for="DepartmentId" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="StudentNumber" class="control-label"></label>
                <input asp-for="StudentNumber" class="form-control" />
                <span asp-validation-for="StudentNumber" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Name" class="control-label"></label>
                <input asp-for="Name" class="form-control" />
                <span asp-validation-for="Name" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="BirthDate" class="control-label"></label>
                <input asp-for="BirthDate" class="form-control" />
                <span asp-validation-for="BirthDate" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="PhotoPath" class="control-label"></label>
                <input asp-for="PhotoPath" class="form-control" />
                <span asp-validation-for="PhotoPath" class="text-danger"></span>
            </div>
            <div class="form-group">
                <input type="submit" value="Save" class="btn btn-primary" />
            </div>
        </form>
    </div>
</div>
```

Client-side Validations

```cshtml
<div class="row">
    <div class="col-md-4">
        <form asp-action="Edit">
            <div asp-validation-summary="ModelOnly" class="text-danger"></div>
            <input type="hidden" asp-for="Id" />
            <div class="form-group">
                <label asp-for="DepartmentId" class="control-label"></label>
                <select asp-for="DepartmentId" class="form-control" asp-items="ViewBag.DepartmentId"></select>
                <span asp-validation-for="DepartmentId" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="StudentNumber" class="control-label"></label>
                <input asp-for="StudentNumber" class="form-control" required="required"/>
                <span asp-validation-for="StudentNumber" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Name" class="control-label"></label>
                <input asp-for="Name" class="form-control" />
                <span asp-validation-for="Name" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="BirthDate" class="control-label"></label>
                <input asp-for="BirthDate" class="form-control" />
                <span asp-validation-for="BirthDate" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="PhotoPath" class="control-label"></label>
                <input asp-for="PhotoPath" class="form-control" />
                <span asp-validation-for="PhotoPath" class="text-danger"></span>
            </div>
            <div class="form-group">
                <input type="submit" value="Save" class="btn btn-primary" />
            </div>
        </form>
    </div>
</div>

@section Scripts {
    @{await Html.RenderPartialAsync("_ValidationScriptsPartial");}
}
```

```
alidationScriptsPartial.cshtml  Edit.cshtml*  Student.cs  Create.cshtml*  StudentDal.cs  IStudentDal.cs  _He
1   <script src="~/lib/jquery-validation/dist/jquery.validate.min.js"></script>
2   <script src="~/lib/jquery-validation-unobtrusive/jquery.validate.unobtrusive.min.js"></script>
3
```

Model Validation

```
29 references
public class Student
{
    [Key]
    7 references
    public int Id { get; set; }
    [ForeignKey(name: "DepartmentId")]
    8 references
    public int DepartmentId { get; set; }
    [ForeignKey(name: "DepartmentId")]
    6 references
    public Department Department { get; set; }
    12 references
    public string StudentNumber { get; set; }

    [Required(ErrorMessage = "Name is ???")]
    12 references
    public string Name { get; set; }
    12 references
    public DateTime? BirthDate { get; set; }
    12 references
    public string PhotoPath { get; set; }
}
```

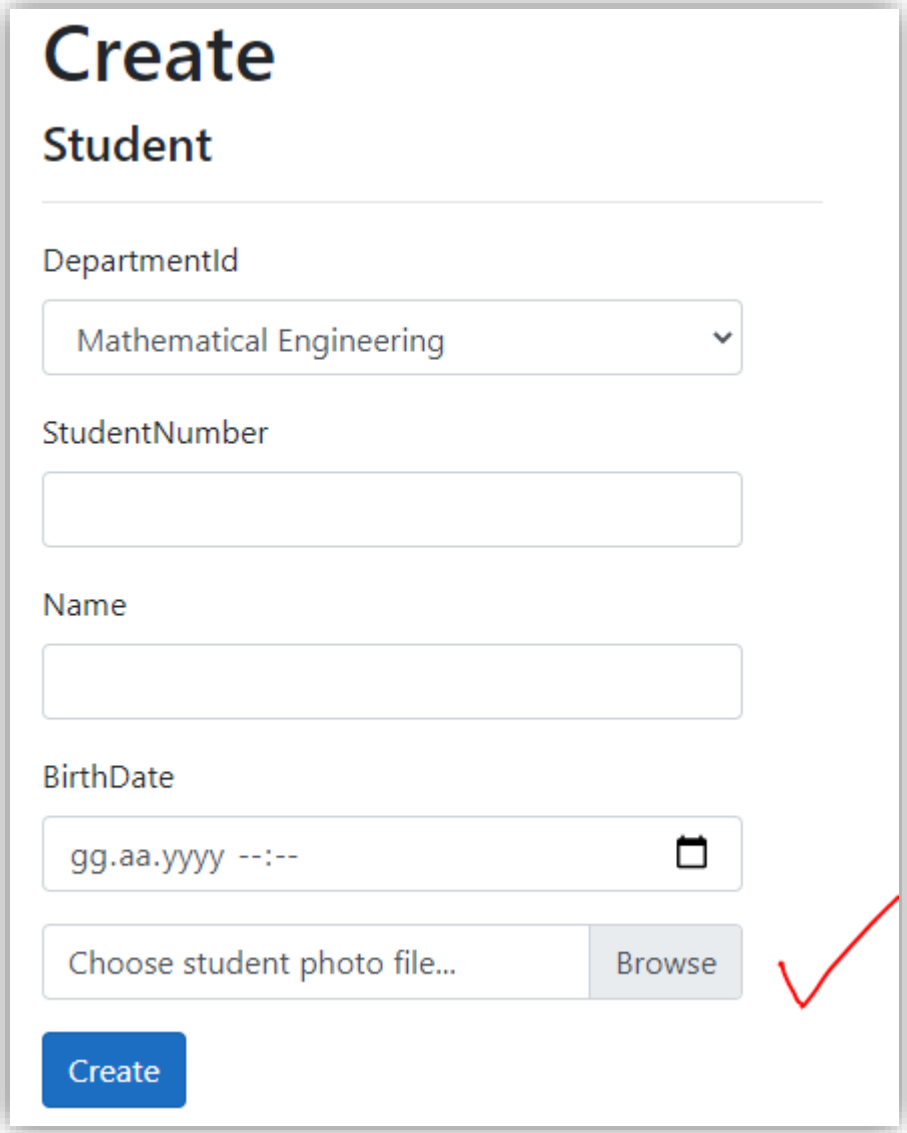
```
[HttpPost]
[ValidateAntiForgeryToken]
0 references
public IActionResult Edit(Student student)
{

    if (ModelState.IsValid)
    {
        _studentDal.Update(student);
        return RedirectToAction(nameof(Index));
    }
    ViewData["DepartmentId"] = new SelectList(items: _depart
    return View(student);
}
```

```
<form asp-action="Edit">
    <div asp-validation-summary="ModelOnly" class="text-danger"></div>
    <input type="hidden" asp-for="Id" />
    <div class="form-group">
        <label asp-for="DepartmentId" class="control-label"></label>
        <select asp-for="DepartmentId" class="form-control" asp-items="ViewBag.DepartmentId"></select>
        <span asp-validation-for="DepartmentId" class="text-danger"></span>
    </div>
```

Uploading Files (multipart/form-data)

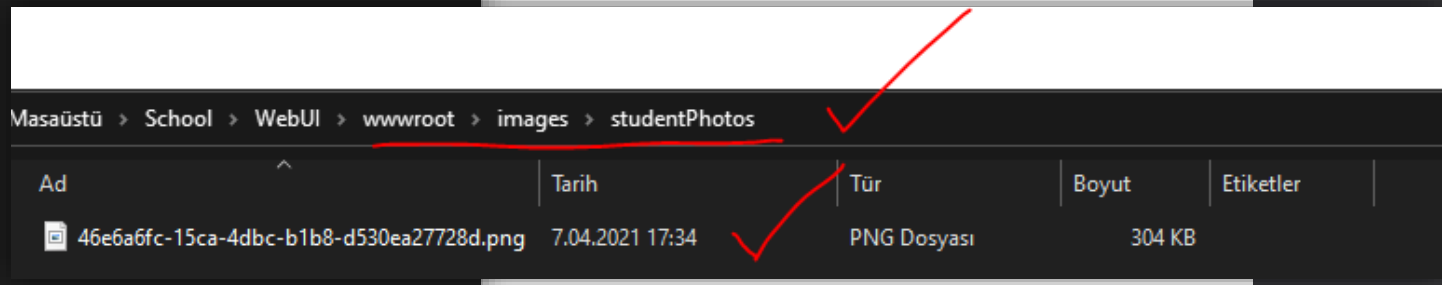
```
<form asp-action="Edit" method="post"  enctype="multipart/form-data">
```

```
<form asp-action="Create" method="post" enctype="multipart/form-data">
    <div asp-validation-summary="ModelOnly" class="text-danger"></div>
    <div class="form-group">…</div>
    <div class="form-group">…</div>
    <div class="form-group">…</div>
    <div class="form-group">…</div>


    <div class="input-group mb-3">
        <div class="custom-file">
            <input type="file" class="custom-file-input" name="file">
            <label class="custom-file-label" for="">Choose student photo file...</label>
        </div>
    </div>

    <div class="form-group">
        <input type="submit" value="Create" class="btn btn-primary" />
    </div>
</form>
```



```
//upload and return full photo path
1 reference
private string UploadPhoto(IFormFile file)
{
    var path = $"wwwroot/images/studentPhotos";
    var fullPath = string.Empty;

    if (!Directory.Exists(path))
        Directory.CreateDirectory(path);

    var permittedExtensions = new string[] { "jpg", "jpeg", "bmp", "png" };

    try
    {

        var fileNameGuid = Guid.NewGuid().ToString();
        fullPath = Path.Combine(path, $"{fileNameGuid}{Path.GetExtension(file.FileName)}");

        if (file.Length > 0)
        {
            using (MemoryStream memoryStream = new MemoryStream())
            {
                file.CopyTo(memoryStream);

                using (var fileStream = new FileStream(fullPath, FileMode.OpenOrCreate))
                {
                    memoryStream.Seek(offset:0, loc:SeekOrigin.Begin);

                    memoryStream.CopyTo(fileStream);

                    fileStream.Flush();
                }
            }
        }
    }
    catch (Exception ex) { }

    return fullPath.Replace(oldValue:"wwwroot/",newValue:"");
}
```

```
[HttpPost]
[ValidateAntiForgeryToken]
0 references
public IActionResult Create(Student student)
{
    var file = Request.Form.Files[0];//uploaded file

    student.PhotoPath = UploadPhoto(file);//upload file


    if (ModelState.IsValid)
    {
        _studentDal.Add(student);
        return RedirectToAction(nameof(Index));
    }
    ViewData["DepartmentId"] = new SelectList(items:_studentDal
    return View(student);
}
```



Caching Architecture

- In –Memory Cache
- Distributed Cache

In –Memory Cache (IMemoryCache Interface)

- In Startup.cs

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

    services.AddControllersWithViews();

    services.AddSingleton<IDepartmentDal, DepartmentDal>();

    services.AddSingleton<IStudentDal, StudentDal>();
}
```

DI for IMemoryCache in Controller

```
private IMemoryCache _memoryCache;

0 references | 0 changes | 0 authors, 0 changes
public StudentsController(IStudentDal studentDal, IDepartmentDal departmentDal, IMemoryCache memoryCache)
{
    _studentDal = studentDal;
    _departmentDal = departmentDal;
    _memoryCache = memoryCache;
}
```

Caching Student List

- Memory Cache is a key-value dictionary

```
[HttpGet]
3 references | asecer79, 6 days ago | 1 author, 1 change
public IActionResult Index()
{
    //read from memory first (from cache)
    var studentListCache = _memoryCache.Get<List<Student>>( key: "studentList");

    if (studentListCache != null)//from cache
    {
        //if cache exists
        return View(studentListCache);
    }
    else //from db, if cache do not exists
    {
        var studentListFromDb = _studentDal.GetList();

        //write memory
        _memoryCache.Set("studentList", studentListFromDb);
        return View(studentListFromDb);
    }


}
```

Cache Consistency- Memory Cache

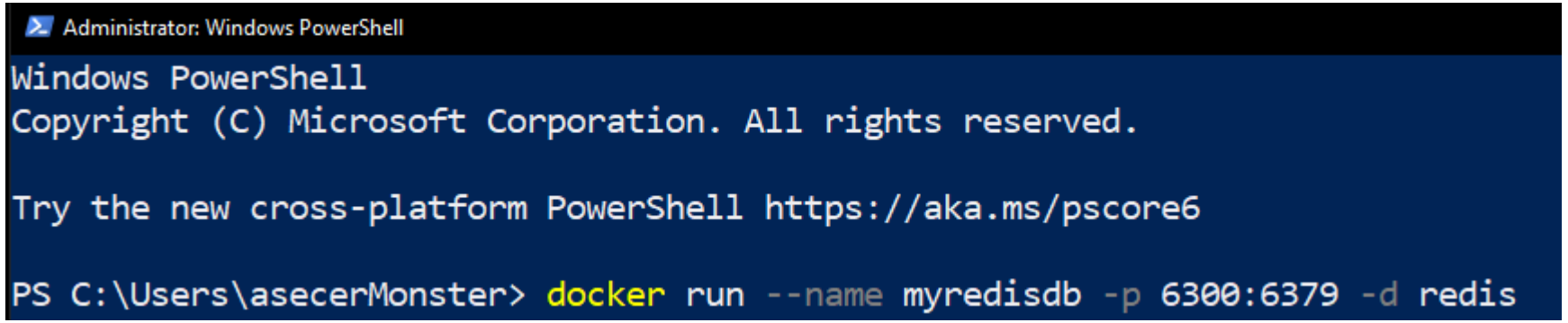
- After deleted, updated or created new entities, the existing cache must be updated or destructed

```
[HttpPost]
[ValidateAntiForgeryToken]
0 references | asecer79, 6 days ago | 1 author, 1 change
public IActionResult Create(Student student)
{
    var file = Request.Form.Files[0];

    student.PhotoPath = UploadPhoto(file);

    //after inserting, old cache must be deleted from memory otherwise dirty data will
    _memoryCache.Remove(key: "studentList");

    if (ModelState.IsValid)
    {
        _studentDal.Add(student);

        return RedirectToAction(nameof(Index));
    }

    ViewData["DepartmentId"] = new SelectList(items: _departmentDal.GetList(), dataValueFiel
    return View(student);
}
```

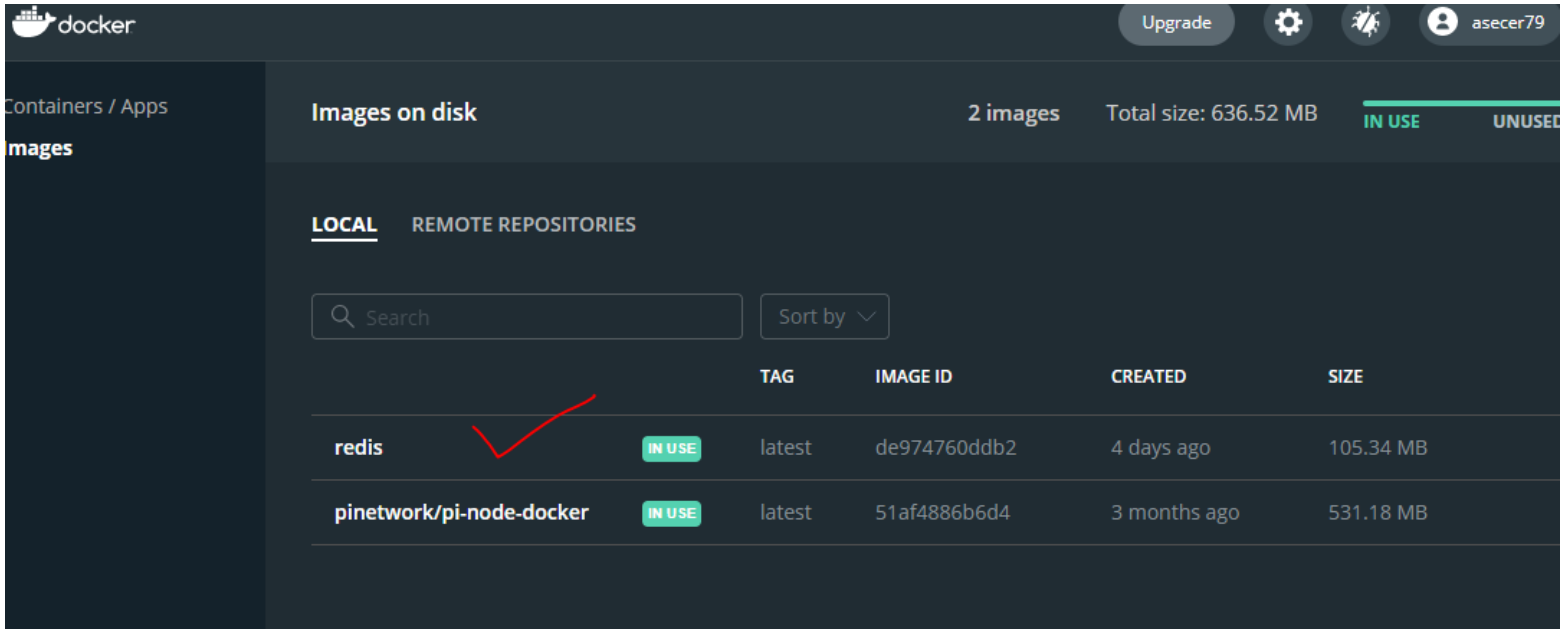
Distributed Cache (Redis)

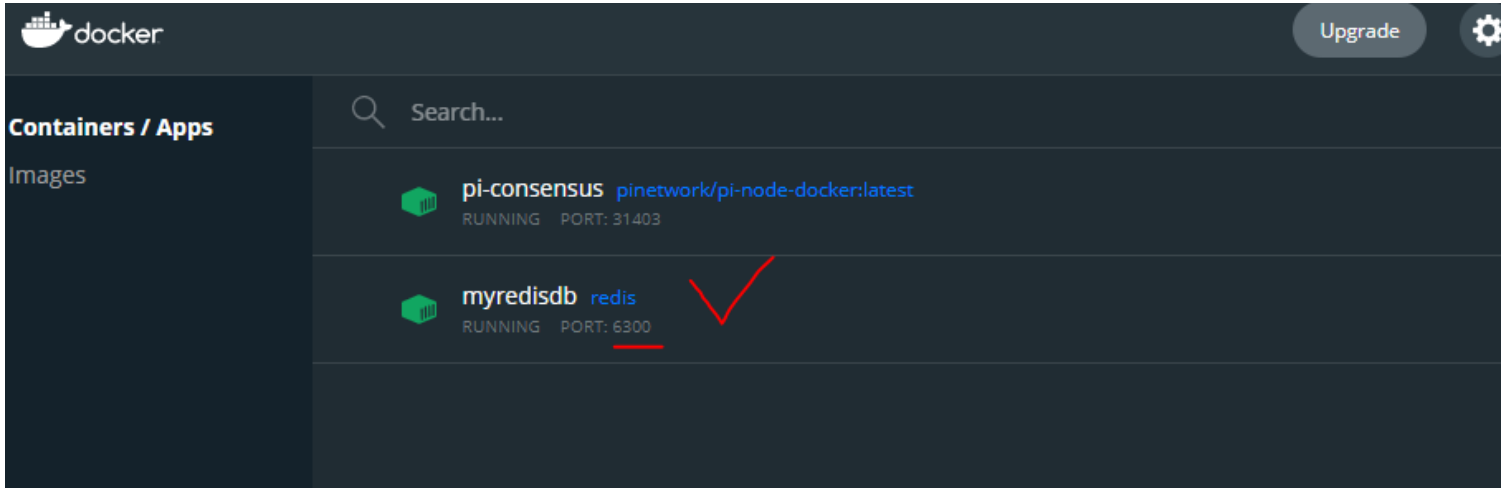
- **Option 1:**
- Use commercial version of redis for Windows:
  - https://www.memurai.com/get-memurai

- **Optiopn 2: (Recommended)**
  - Use Docker image on a Linux container (Recommended operating system is Linux)
  - After installation docker (https://www.docker.com/get-started)
    - In Windows Power Shell (or Linux bash): write following command



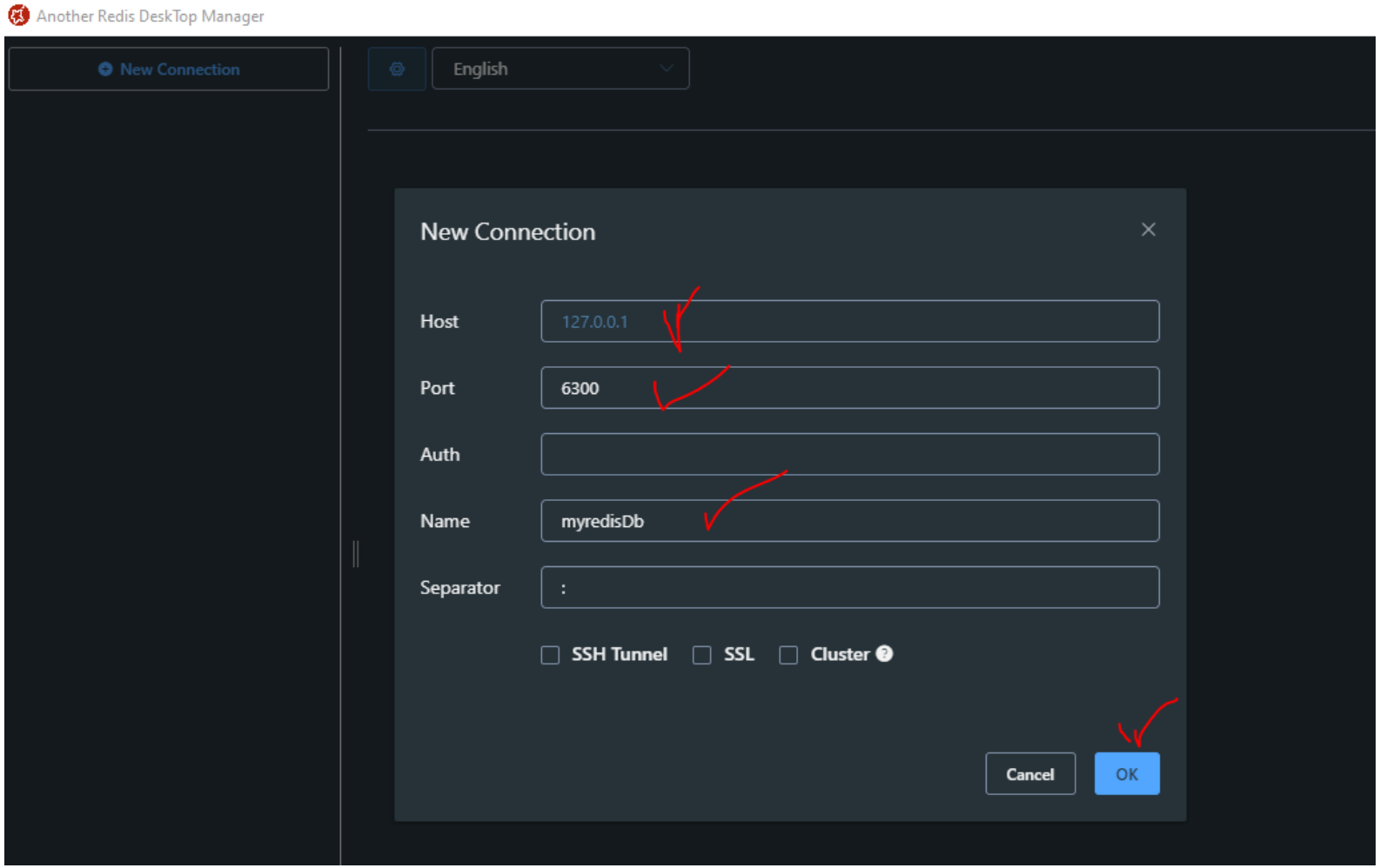
Docker Desktop Manager



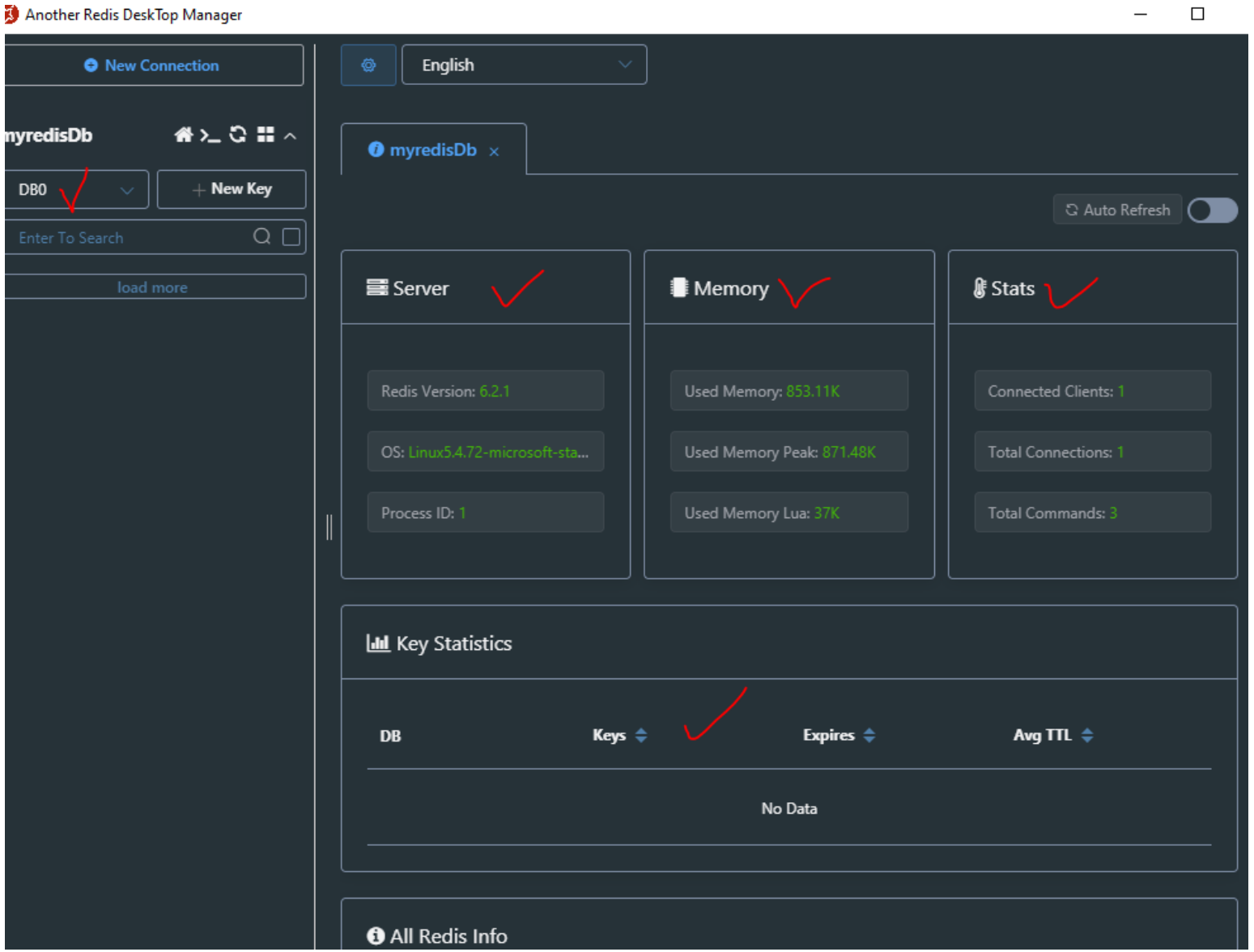


Another Redis Desktop Manager

- Download this tool for free to manage and view your redis databases
  - https://appimage.github.io/Another_Redis_Desktop_Manager/

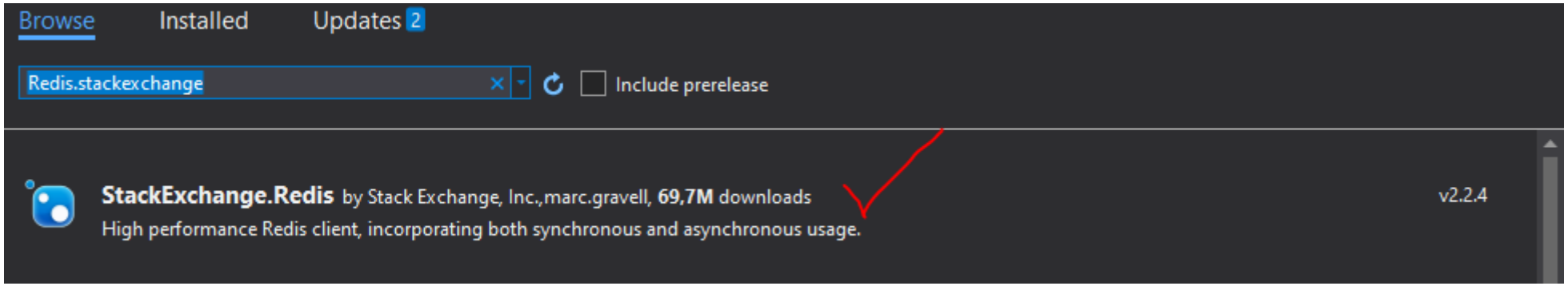


You can view and reports all keys and values



Using Redis in .Net Core App

- Install the nuget package:



Simple Usage of Redis for Students Controller

```
readonly ConnectionMultiplexer _connectionMultiplexer =
    ConnectionMultiplexer.Connect(configuration: "127.0.0.1:6300");
private readonly IDatabase _redisDatabase;


0 references | asecer79, 7 minutes ago | 1 author, 2 changes
public StudentsController(IStudentDal studentDal, IDepartmentDal departmentDal)
{
    _studentDal = studentDal;
    _departmentDal = departmentDal;

    //redis has 16 database we chose the db: 0 (optional)
    _redisDatabase = _connectionMultiplexer.GetDatabase(db: 0);
}
```

Caching Student List with Redis

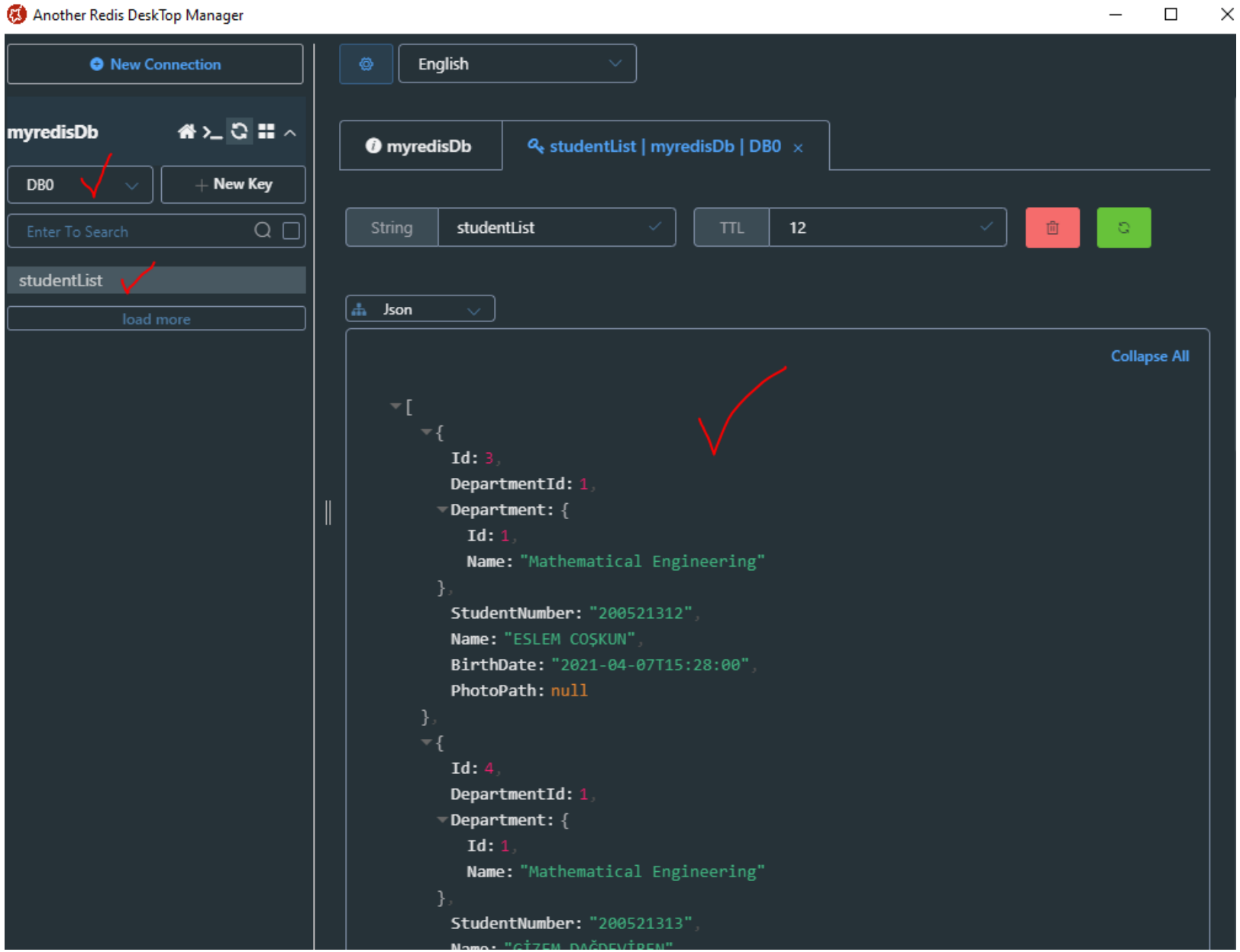
```
[HttpGet]
3 references | asecer79, 11 minutes ago | 1 author, 2 changes
public IActionResult Index()
{
    //if redis key exists?
    bool keyUsed = _redisDatabase.KeyExists("studentList");

    if (keyUsed)
    {
        var rawStringFromRedis = _redisDatabase.StringGet(key:"studentList");

        var studentListFromRedis = JsonSerializer.Deserialize<List<Student>>(rawStringFromRedis);

        return View(studentListFromRedis);

    }
    else
    {
        var studentListFromDb = _studentDal.GetList();

        //set redis cache
        var jsonString = JsonSerializer.Serialize(studentListFromDb);
        _redisDatabase.StringSet(key:"studentList", jsonString, expiry:TimeSpan.FromSeconds(25));

        return View(studentListFromDb);
    }

}
```

Cache Consistency- Redis Cache

- After deleted, updated or created new entities, the existing cache must be updated or destructed

```
[HttpPost]
[ValidateAntiForgeryToken]
0 references | asecer79, 6 days ago | 1 author, 1 change
public IActionResult Create(Student student)
{
    var file = Request.Form.Files[0];

    student.PhotoPath = UploadPhoto(file);

    bool keyUsed = _redisDatabase.KeyExists("studentList");
    if (keyUsed)
        _redisDatabase.KeyDelete("studentList");

    if (ModelState.IsValid)
    {
        _studentDal.Add(student);

        return RedirectToAction(nameof(Index));
    }

    ViewData["DepartmentId"] = new SelectList(items: _departmentDal
    return View(student);
}
```

Monitoring Cache from Redis Desktop Man.



DI Implementation of Memory and Distributed Cache

```
public interface ICacheHelper
{
    3 references | asecer79, 2 days ago | 1 author, 1 change
    bool KeyExists(string key);
    3 references | asecer79, 2 days ago | 1 author, 1 change
    T Get <T> (string Key);

    3 references | asecer79, 2 days ago | 1 author, 1 change
    void Set<T>(string key, T data);

}
```

MemoryCacheHelper.cs class

```
public class MemoryCacheHelper: ICacheHelper
{
    private readonly IMemoryCache _memoryCache;

    0 references | asecer79, 2 days ago | 1 author, 1 change
    public MemoryCacheHelper(IMemoryCache memoryCache)
    {
        _memoryCache = memoryCache;
    }

    3 references | asecer79, 2 days ago | 1 author, 1 change
    public bool KeyExists(string key)
    {
        return _memoryCache.Get(key)!=null;
    }

    3 references | asecer79, 2 days ago | 1 author, 1 change
    public T Get<T>(string Key)
    {
        var cache = _memoryCache.Get<T>(Key);
        return cache;
    }

    3 references | asecer79, 2 days ago | 1 author, 1 change
    public void Set<T>(string key, T data)
    {
        _memoryCache.Set(key, data);
    }

}
```

RedisCacheHelper.cs class

```
public class RedisCacheHelper: ICacheHelper
{
    private readonly ConnectionMultiplexer _connectionMultiplexer;
    private IConfiguration Configuration;
    private readonly IDatabase db;
    0 references | 0 changes | 0 authors, 0 changes
    public RedisCacheHelper(IConfiguration configuration)
    {
        Configuration = configuration;
        _connectionMultiplexer = ConnectionMultiplexer.Connect(configuration.GetSection(key: "RedisConnString").Value);
        db = _connectionMultiplexer.GetDatabase(db: 0);
    }
    3 references | 0 changes | 0 authors, 0 changes
    public bool KeyExists(string key)
    {
        return db.KeyExists(key);
    }
    3 references | 0 changes | 0 authors, 0 changes
    public T Get<T>(string Key)
    {
        var cache = JsonSerializer.Deserialize<T>(json: db.StringGet(Key));
        return cache;
    }
    3 references | 0 changes | 0 authors, 0 changes
    public void Set<T>(string key, T data)
    {
        db.StringSet(key, JsonSerializer.Serialize(data));
    }
}
```

```
//appsettings.json
{
  "RedisConnString": "127.0.0.1:6379",


  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "AllowedHosts": "*"
}
```

DI - Registering services in Startup

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

    //one of the following must be uncommented if you want to use it. Both must not be selected...
    services.AddSingleton<ICacheHelper, RedisCacheHelper>();
   // services.AddSingleton<ICacheHelper, MemoryCacheHelper>();
```

Single Interface → Multiple Cache Options

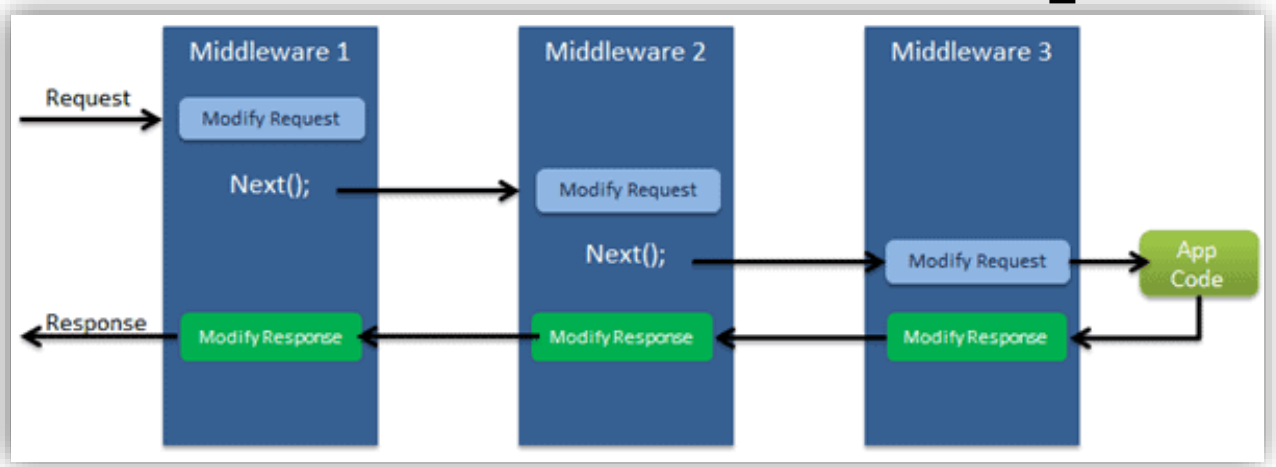
```
private readonly ICacheHelper _cacheHelper;

0 references | asecer79, 2 days ago | 1 author, 1 change
public CustomersController(ICacheHelper cacheHelper)
{
    _cacheHelper = cacheHelper;
}

[HttpGet]
[Route(template: "getList")]
0 references | asecer79, 2 days ago | 1 author, 1 change
public IActionResult getList()
{


    string key = "customers_cache";

    if (_cacheHelper.KeyExists("customers_cache"))
    {
        return Ok(_cacheHelper.Get<List<Customer>>(key));
    }

    Thread.Sleep(millisecondsTimeout: 2000);

    var data :List<Customer> = new DatabaseTest().GetList();


    _cacheHelper.Set(key: "customers_cache", data);

    return Ok(data);
}
```

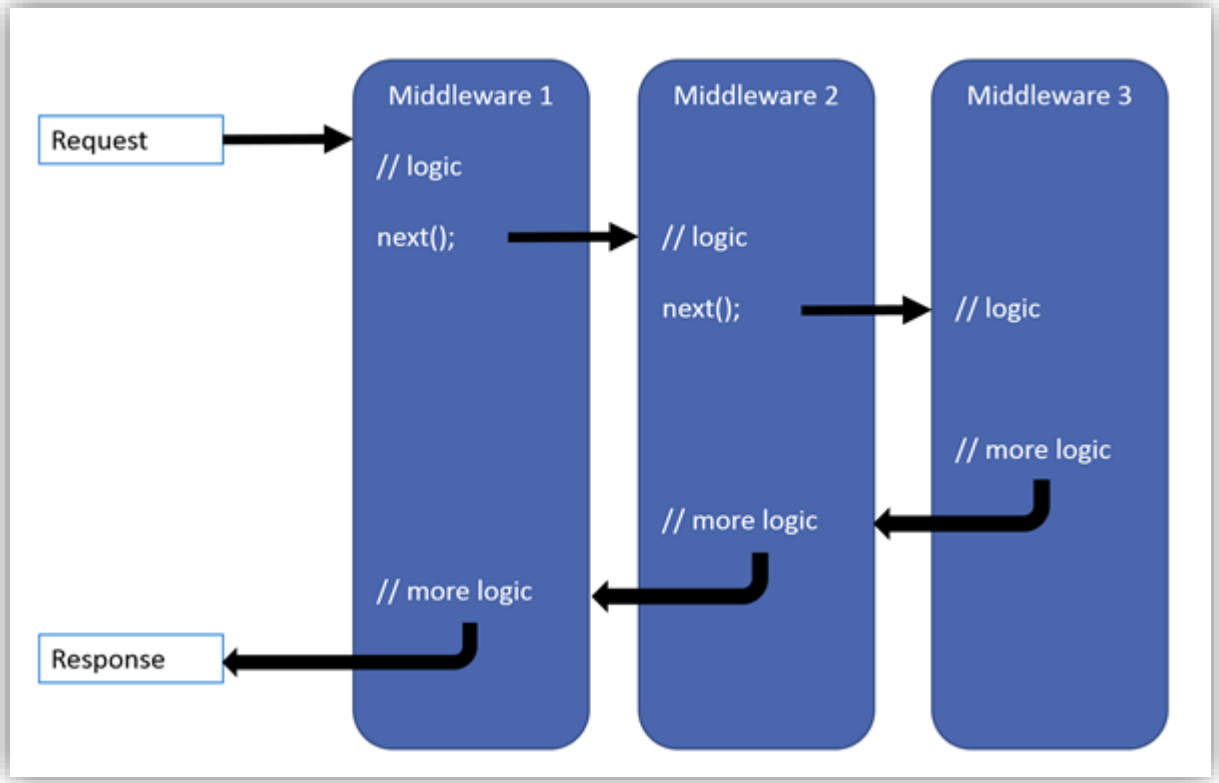
Middlewares

ASP.NET Core - Middleware

- A middleware is nothing but a component (class) which is executed on every request in ASP.NET Core application
- In the classic ASP.NET, HttpHandlers and HttpModules were part of request pipeline.
- Middleware is similar to HttpHandlers and HttpModules where both needs to be configured and executed in each request.
- Typically, there will be multiple middleware in ASP.NET Core web application.
- It can be either framework provided middleware, added via NuGet or your own custom middleware.
- We can set the order of middleware execution in the request pipeline.
- Each middleware **adds or modifies http request** and optionally passes control to the next middleware component.

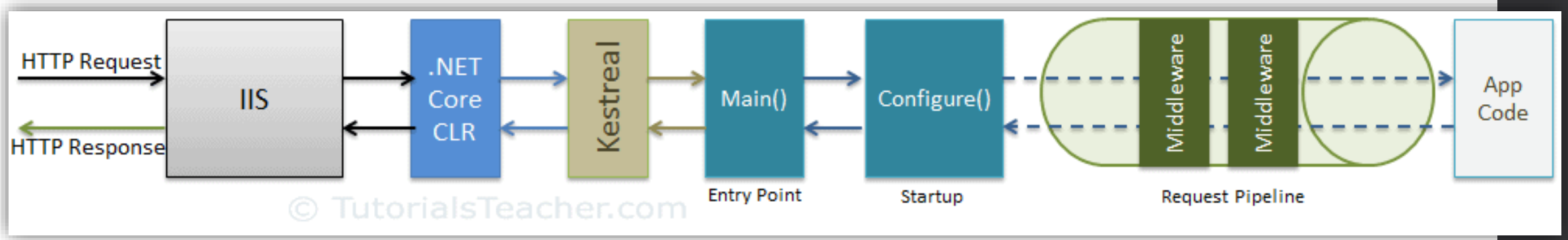
Execution of middleware components.





App **pipeline** to handle requests and responses

- **ASP.NET Core Request Processing**



Configure Middleware: Single Middleware

- We can configure middleware in the **Configure** method of the **Startup** class using **IApplicationBuilder** instance.
- The following example adds a single middleware using Run method which returns a string "Hello World!" on each request

```
public class Startup
{
    public Startup()
    {
    }
    public void Configure(IApplicationBuilder app, IHostingEnvironment env, ILoggerFactory loggerFactory)
    {
        //configure middleware using IApplicationBuilder here..

        app.Run(async (context) =>
        {
            await context.Response.WriteAsync("Hello World!");

        });

        // other code removed for clarity..
    }
}
```

Understand Run Method

- Run() is an extension method on IApplicationBuilder instance which adds a terminal middleware to the application's request pipeline.
- Run () Signature:
  - public **static void** Run(this **IApplicationBuilder** app, **RequestDelegate** handler)
- RequestDelegate Signature:
  - public **delegate Task** RequestDelegate(**HttpContext** context);
- As you can see above, the Run method accepts a method as a parameter whose signature should match with RequestDelegate.
- Therefore, the method should accept the HttpContext parameter and return Task.
- So, you can either specify a lambda expression or specify a function in the Run method.

Write Middleware Explicitly

```
public class Startup
{
    public Startup()
    {
    }

    public void Configure(IApplicationBuilder app, IHostingEnvironment env)
    {
        app.Run(MyMiddleware);//passging delegate
    }

    private Task MyMiddleware(HttpContext context)
    {
        return context.Response.WriteAsync("Hello World! ");
    }
}
```

With Lambda (Delegate)

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{

    app.Run(async context => await context.Response.WriteAsync("Hello World!"));

    //or

    app.Run(async (context) => { await context.Response.WriteAsync("Hello World!"); });
}
```

Warning!

- Mostly there will be multiple middleware components in ASP.NET Core application which will be executed sequentially.
- The Run method adds a terminal middleware so it cannot call next middleware as it would be the last middleware in a sequence.
- The following will always execute the first Run method and will never reach the next middlewares
- If we need single middleware, that used with Run method, must be put in Configure method as last middleware.
- Otherwise, other middlewares will be unreachable.

```
app.Run(handler: async context =>
        {
            Debug.WriteLine(message: "Request 1");
            await context.Response.WriteAsync(text: "Request1");

});

app.UseRouting();

app.UseAuthorization();
```

```
app.UseHttpsRedirection();

app.UseStaticFiles();

app.UseRouting();

app.UseAuthorization();

app.Run(handler: async context =>
        {
            Debug.WriteLine(message: "Request 1");
            await context.Response.WriteAsync(text: "Request1");

        });
```

Configure Multiple Middleware

- To configure multiple middleware, use Use() extension method.
- It is similar to Run() method except that it includes next parameter to invoke next middleware in the sequence.

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    app.Use(async (context, next) =>
    {
        await context.Response.WriteAsync(text:"Hello World From 1st Middleware!\n");
        await next();
    });

    app.Use(async (context, next) =>
    {
        await context.Response.WriteAsync(text:"Hello World From 2nd Middleware!\n");
        await next();
    });

    app.Use(async (context, next) =>
    {
        await context.Response.WriteAsync(text:"Hello World From 3rd Middleware!\n");
        await next();
    });

    app.Run(handler:async (context) =>
    {
        await context.Response.WriteAsync(text:"Hello World From 4th Middleware!\n");
    });
}
```

Built-in Middlewares Via NuGet

- We can add server side features we need in our application by installing different plug-ins via NuGet.
- There are many middleware plug-ins available which can be used in our application.

Middleware	Description
Authentication	Adds authentication support.
CORS	Configures Cross-Origin Resource Sharing.
Routing	Adds routing capabilities for MVC or web form
Session	Adds support for user session.
StaticFiles	Adds support for serving static files and directory browsing.
Diagnostics	Adds support for reporting and handling exceptions and errors.

Diagnostics Middlewares

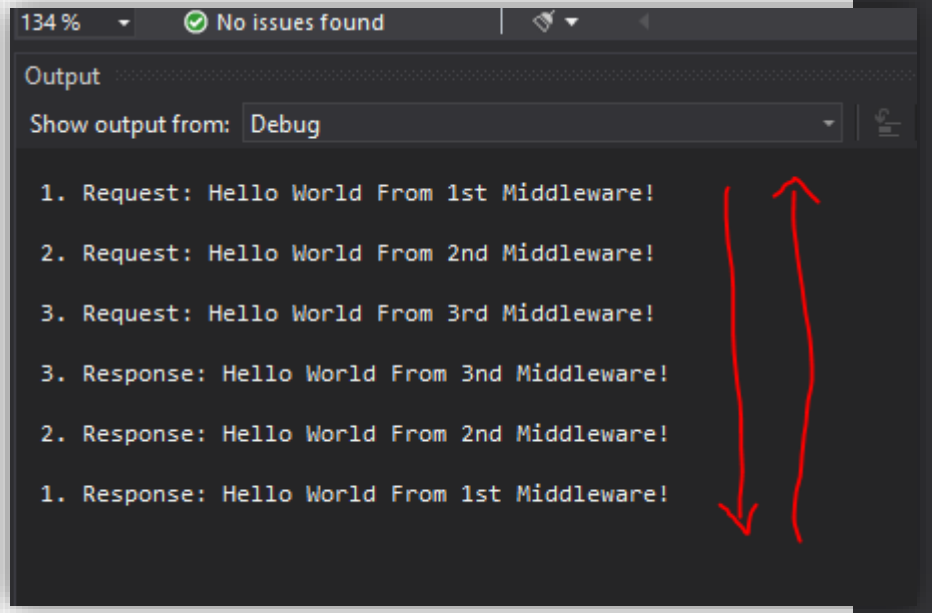
- Diagnostics middleware is used for reporting and handling exceptions and errors in ASP.NET Core, and diagnosing Entity Framework Core migrations errors.

Middleware	Extension Method	Description
DeveloperExceptionPageMiddleware	UseDeveloperExceptionPage()	Captures synchronous and asynchronous exceptions from the pipeline and generates HTML error responses.
ExceptionHandlerMiddleware	UseExceptionHandler()	Catch exceptions, log them and re-execute in an alternate pipeline.
StatusCodePagesMiddleware	UseStatusCodePages()	Check for responses with status codes between 400 and 599.
WelcomePageMiddleware	UseWelcomePage()	Display Welcome page for the root path.

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    app.UseWelcomePage();
}
```

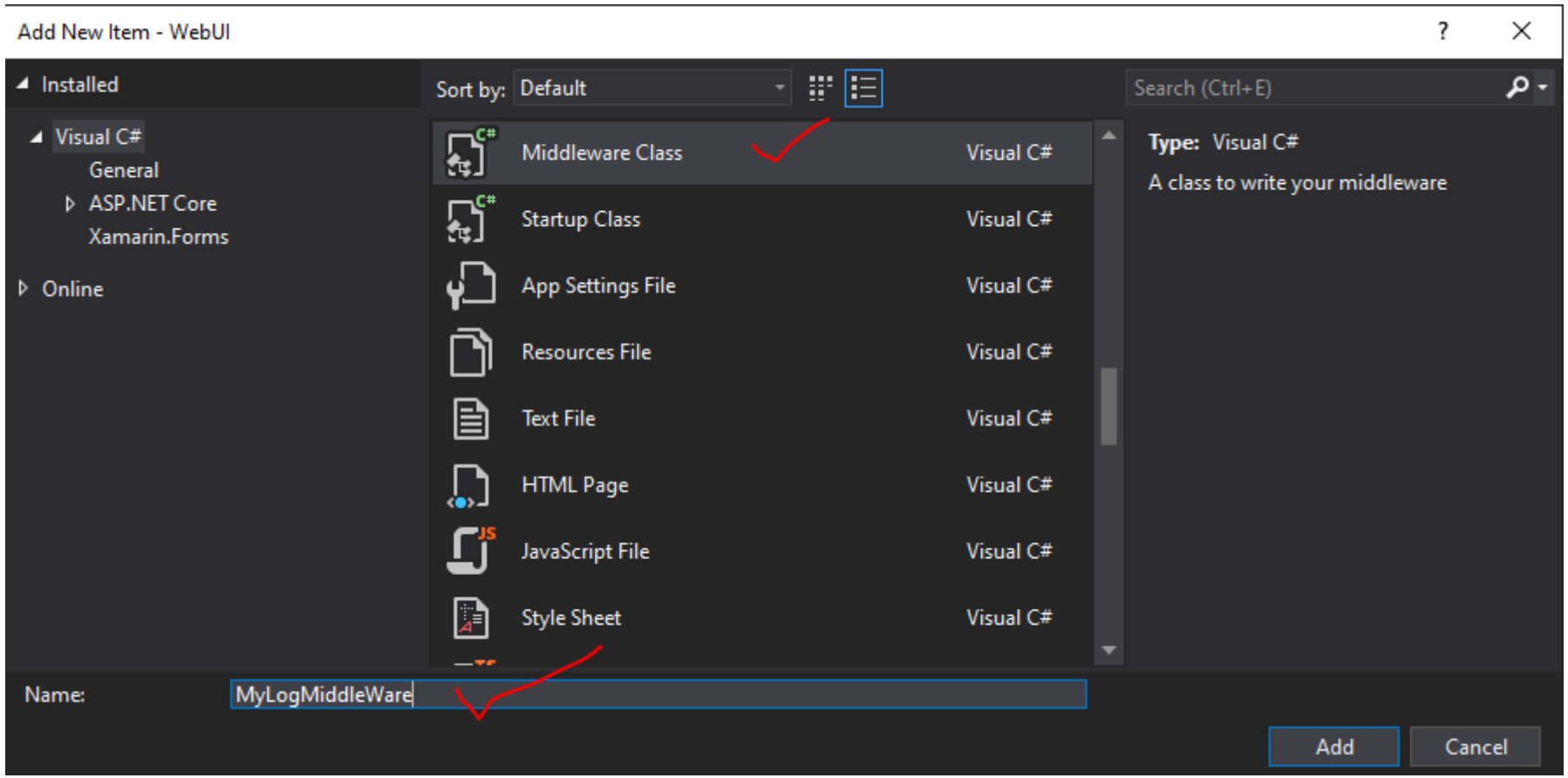
Multiple Middleware-Request and Response

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    app.Use(async (context, next) =>
    {
        Debug.WriteLine(message:"1.Request: Hello World From 1st Middleware!\n"); //request
        await next();
        Debug.WriteLine(message:"1. Response: Hello World From 1st Middleware!\n"); //response
    });

    app.Use(async (context, next) =>
    {
        Debug.WriteLine(message:"2. Request: Hello World From 2nd Middleware!\n");//request
        await next();
        Debug.WriteLine(message:"2. Request: Hello World From 2nd Middleware!\n");
    });

    app.Use(async (context, next) =>
    {
        Debug.WriteLine(message:"3. Request: Hello World From 3rd Middleware!\n");//request
        await next();
        Debug.WriteLine(message:"3. Request: Hello World From 2nd Middleware!\n");//response
    });

}
```



Custom Middleware in.NET Core Application

- The custom middleware component is like any other .NET class with Invoke() method.
- However, in order to execute next middleware in a sequence, it should have RequestDelegate type parameter in the constructor.



Custom Middleware Class

```
public class MyLogMiddleware
{
    private readonly RequestDelegate _next;

    0 references | 0 changes | 0 authors, 0 changes
    public MyLogMiddleware(RequestDelegate next)
    {
        _next = next;
    }

    0 references | 0 changes | 0 authors, 0 changes
    public async Task Invoke(HttpContext httpContext)
    {
        //do some logic here (ie save log datafile for all client request)
        Debug.WriteLine( message: $"{httpContext.Request.Path.Value}{httpContext.Request.QueryString}");
        await _next(httpContext);
        Debug.WriteLine( message: $"{httpContext.Response.StatusCode}");

    }
}
```

Custom Middleware Extesion Method

```
public static class MyLogMiddleWareExtensions
{
    1 reference | 0 changes | 0 authors, 0 changes
    public static IApplicationBuilder UseMyLogMiddleWare(this IApplicationBuilder builder)
    {
        return builder.UseMiddleware<MyLogMiddleware>();
    }
}
```

- Using Custom Middleware

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{

    app.UseMyLogMiddleWare();

}
```

Map Method

- To apply middleware for only a specific path
- The Map extension method is used to match request delegates based on a request's path.
- Map simply accepts a path and a function that configures a separate middleware pipeline.

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    app.Use(async (context, next) =>
    {
        Debug.WriteLine(message: "Use middleware executed.");
        await next.Invoke();
    });

    //if path /example condition satisfied this middleware will be executed
    app.Map(pathMatch: "/example", configuration: internalApp :IApplicationBuilder =>
        internalApp.Use(async (context, next) =>
        {
            Debug.WriteLine(message: "/example middleware executed.");
            await next();
        }));
}
```

MapWhen Method

- is more efficient when condition is complex…

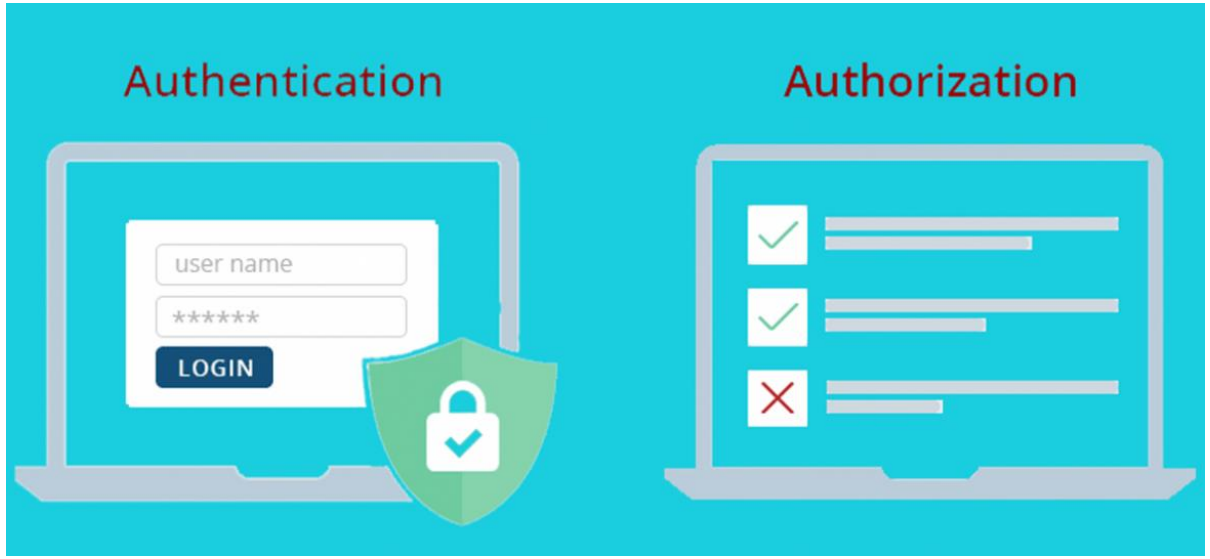
```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    app.Use(async (context, next) =>
    {
        Debug.WriteLine(message: "Use middleware Executed.");
        await next.Invoke();
    });

    //if conditions are satisfied this middleware will be executed
    app.MapWhen(predicate: x :HttpContext => x.Request.Method == "GET", configuration: internalApp :IApplicationBuilder =>
    {
        internalApp.Run(handler: async context => await context.Response.WriteAsync(text: "MapWhen Middleware for GET action"));
    });
}
```

Security

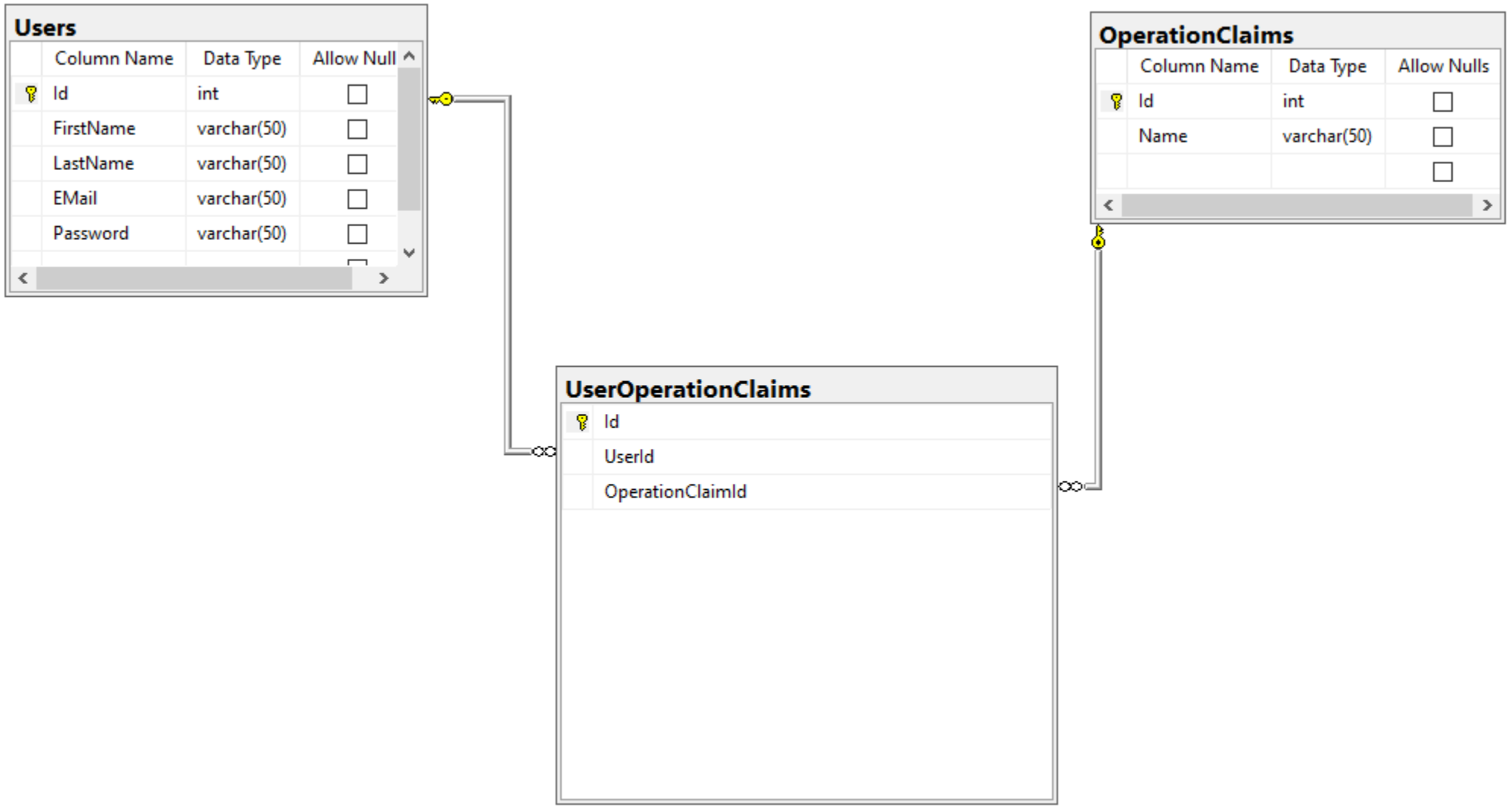
- Authentication
- Authorization

Authentication vs Authorization

- **Authentication** means confirming your own identity, whereas **authorization** means being allowed access to the system.
- In even more simpler terms **authentication** is the process of verifying oneself, while **authorization** is the process of verifying what you have access to specific menu or operations.



Role Based Membership Table Architecture



Security& Membership Entities

```
public class User
{
    [Key]
    0 references | 0 changes | 0 authors, 0 changes
    public int Id { get; set; }
    0 references | 0 changes | 0 authors, 0 changes
    public string FirstName { get; set; }
    0 references | 0 changes | 0 authors, 0 changes
    public string LastName { get; set; }
    0 references | 0 changes | 0 authors, 0 changes
    public string EMail { get; set; }
    0 references | 0 changes | 0 authors, 0 changes
    public string Password { get; set; }
    0 references | 0 changes | 0 authors, 0 changes
    public ICollection<OperationClaim> OperationClaims { get; set; }

}
```

```
public class OperationClaim
{
    0 references | 0 changes | 0 authors, 0 changes
    public int Id { get; set; }
    0 references | 0 changes | 0 authors, 0 changes
    public string Name { get; set; }

}
```

```
public class UserOperationClaim
{
    [Key]
    0 references | 0 changes | 0 authors, 0 changes
    public int Id { get; set; }

    [ForeignKey(name: "UserId")]
    0 references | 0 changes | 0 authors, 0 changes
    public int UserId { get; set; }

    [ForeignKey(name: "UserId")]
    0 references | 0 changes | 0 authors, 0 changes
    public User User { get; set; }

    [ForeignKey(name: "OperationClaimId")]
    0 references | 0 changes | 0 authors, 0 changes
    public int OperationClaimId { get; set; }

    [ForeignKey(name: "OperationClaimId")]
    0 references | 0 changes | 0 authors, 0 changes
    public OperationClaim OperationClaim { get; set; }

}
```

Adding Entity Sets to Repository

```
public class SchoolContext : DbContext
{
    0 references | asecer79, 26 days ago | 1 author, 1 change
    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        optionsBuilder.UseSqlServer( connectionString: "data source=.; initial catalog =School;Trusted_Connection=True;");
    }


    0 references | 0 changes | 0 authors, 0 changes
    public DbSet<User> Users { get; set; }
    0 references | 0 changes | 0 authors, 0 changes
    public DbSet<OperationClaim> OperationClaims { get; set; }
    0 references | 0 changes | 0 authors, 0 changes
    public DbSet<UserOperationClaim> UserOperationClaims { get; set; }
    0 references | asecer79, 26 days ago | 1 author, 1 change
    public DbSet<Course> Courses { get; set; }
    2 references | asecer79, 26 days ago | 1 author, 1 change
    public DbSet<Department> Departments { get; set; }
    0 references | asecer79, 26 days ago | 1 author, 1 change
    public DbSet<Lecturer> Lecturers { get; set; }
    0 references | asecer79, 26 days ago | 1 author, 1 change
    public DbSet<Semester> Semester { get; set; }
    0 references | asecer79, 26 days ago | 1 author, 1 change
    public DbSet<Student> Students { get; set; }
}
```

DbService Interface and Dal Class

```
public interface IUserDal:IDbService<User>
{

}
```

```
public class UserDal:IUserDal
{
    9 references | 0 changes | 0 authors, 0 changes
    public User Get(int id)
    {
        using (SchoolContext context = new SchoolContext())
        {
            return context.Set<User>().FirstOrDefault(p :User => p.Id == id);
        }
    }
    5 references | 0 changes | 0 authors, 0 changes
    public User Add(User entity)...
    5 references | 0 changes | 0 authors, 0 changes
    public void Remove(User entity)...
    4 references | 0 changes | 0 authors, 0 changes
    public void Update(User entity)...
    9 references | 0 changes | 0 authors, 0 changes
    public List<User> GetList(Expression<Func<User, bool>> filter = null)...
}
```

Additional Specialized Operations in IUserDal

```
public interface IUserDal:IDbService<User>
{
    2 references | 0 changes | 0 authors, 0 changes
    public User GetUserByEmailAndPassword(string email,string password);
    2 references | 0 changes | 0 authors, 0 changes
    public List<OperationClaim> GetUserOperationClaims(int userId);
}
```

```
public class UserDal : IUserDal
{
    2 references | 0 changes | 0 authors, 0 changes
    public User GetUserByEmailAndPassword(string email, string password)
    {
        using (SchoolContext context = new SchoolContext())
        {
            return context.Set<User>().FirstOrDefault(p:User => p.EMail == email && p.Password == password);
        }
    }
    2 references | 0 changes | 0 authors, 0 changes
    public List<OperationClaim> GetUserOperationClaims(int userId)
    {
        using (SchoolContext context = new SchoolContext())
        {
            var claims:List<OperationClaim> = (from p:UserOperationClaim in context.UserOperationClaims
                where p.UserId == userId
                select new OperationClaim()
                {
                    Name = p.OperationClaim.Name,
                    Id = p.OperationClaimId
                }).ToList();

            return claims;
        }
    }
    9 references | 0 changes | 0 authors, 0 changes
    public User Get(int id)...
```

Initial Settings in Startup.cs

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    else
    {
        app.UseExceptionHandler("/Home/Error");

        app.UseHsts();
    }
    app.UseHttpsRedirection();

    app.UseStaticFiles();

    app.UseCookiePolicy();

    app.UseRouting();

    app.UseAuthentication();

    app.UseAuthorization();

    app.UseEndpoints(endpoints :IEndpointRouteBuilder =>
```

Authorization Levels

Action Level Authorization

```
[Authorize] //only authentication is enough
[HttpGet]
0 references | asecer79, 13 days ago | 1 author, 2 changes
public IActionResult Create()
{
    ViewBag.DepartmentId = new
        SelectList(items:_departmentDal.GetList(),
            dataValueField:"Id", dataTextField:"Name");
    return View();
}
```

```
[Authorize(Roles="admin")] //authentication + userClaim = admin  requires
[HttpGet]
0 references | asecer79, 13 days ago | 1 author, 2 changes
public IActionResult Create()
{
    ViewBag.DepartmentId = new
        SelectList(items:_departmentDal.GetList(),
            dataValueField:"Id", dataTextField:"Name");
    return View();
}
```

```
[Authorize(Roles="admin, teacher")] //authentication + userClaim = admin +  userClaim = teacher require
[HttpGet]
0 references | asecer79, 13 days ago | 1 author, 2 cha
class System.String
Represents text as a sequence of UTF-16 code units.
public IActionResult Create()
{
    ViewBag.DepartmentId = new
        SelectList(items:_departmentDal.GetList(),
            dataValueField:"Id", dataTextField:"Name");
    return View();
}
```

Controller Level Authorization

```
[Authorize] // only checks authentication (optional according to needs)
[Authorize (Roles = "admin,user")] //checks both authentication and userClaims (roles) (optional according to needs)
1 reference | asecer79, 13 days ago | 1 author, 7 changes
public class StudentsController : Controller
{
    private readonly IStudentDal _studentDal;
    private readonly IDepartmentDal _departmentDal;
    private readonly ICacheService _cacheService;

    0 references | asecer79, 19 days ago | 1 author, 1 change
    public StudentsController(IStudentDal studentDal, IDepartmentDal departmentDal, ICacheService cacheService)
    {
        _studentDal = studentDal;
        _departmentDal = departmentDal;
        _cacheService = cacheService;
    }
```

Global Level Authorization (Optional)

```
public void ConfigureServices(IServiceCollection services)
{

    services.AddMemoryCache();
    services.AddSingleton<ICacheService, MemoryCacheService>();
    //services.AddSingleton<ICacheService, RedisCacheService>();
    services.AddHttpContextAccessor();
    services.AddSingleton<AuthHelper>();


    services.AddControllersWithViews(configure: context :MvcOptions =>
    {   //all actions and controllers require authorization (global filter)
        context.Filters.Add(item: new AuthorizeFilter());
    });
```

Cookie Based Authentication

- Configure appsettings.json as:

```json
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "CookieAuthOptions": {
    "Name": "_asecerToken",
    "LoginPath": "/Auth/Login",
    "LogoutPath": "/Auth/Logout",
    "AccessDeniedPath": "/Auth/Err",
    "SlidingExpiration": true,
    "TimeOut": 3600 //as seconds
  },
  "AllowedHosts": "*"
}
```

for simplicity create a class for settings as:

```
public class CookieAuthOptions
{
    1 reference | 0 changes | 0 authors, 0 changes
    public string Name { get; set; }
    1 reference | 0 changes | 0 authors, 0 changes
    public string LoginPath { get; set; }
    1 reference | 0 changes | 0 authors, 0 changes
    public string LogoutPath { get; set; }
    1 reference | 0 changes | 0 authors, 0 changes
    public string AccessDeniedPath { get; set; }
    1 reference | 0 changes | 0 authors, 0 changes
    public bool SlidingExpiration { get; set; }
    1 reference | 0 changes | 0 authors, 0 changes
    public int TimeOut { get; set; }

}
```

Configuring Startup.cs for Cookie Schema

```
public void ConfigureServices(IServiceCollection services)
{

    services.AddMemoryCache();
    services.AddSingleton<ICacheService, MemoryCacheService>();
    //services.AddSingleton<ICacheService, RedisCacheService>();
    services.AddHttpContextAccessor();
    services.AddControllersWithViews(configure: context :MvcOptions =>...);

    services.AddSingleton<AuthHelper>();

    var cookieAuthOptions = Configuration.GetSection(key: "CookieAuthOptions").Get<CookieAuthOptions>();

    services.AddAuthentication(CookieAuthenticationDefaults.AuthenticationScheme).AddCookie(options =>
    {
        options.Cookie.Name = cookieAuthOptions.Name;
        options.LoginPath = cookieAuthOptions.LoginPath;
        options.LogoutPath = cookieAuthOptions.LogoutPath;
        options.AccessDeniedPath = cookieAuthOptions.AccessDeniedPath;
        options.SlidingExpiration = cookieAuthOptions.SlidingExpiration;
        options.ExpireTimeSpan = TimeSpan.FromSeconds(cookieAuthOptions.TimeOut);
        // options.Cookie.SameSite = SameSiteMode.Lax;
    });
```

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    else
    {
        app.UseExceptionHandler("/Home/Error");

        app.UseHsts();
    }
    app.UseHttpsRedirection();

    app.UseStaticFiles();

    app.UseCookiePolicy();

    app.UseRouting();
```

Writing Our Custom AuthHelper Class

- Dependency Injections

```
public class AuthHelper
{
    private readonly IUserDal _userDal;
    private readonly IConfiguration _configuration;
    private readonly IHttpContextAccessor _httpContextAccessor;
    0 references | 0 changes | 0 authors, 0 changes
    public AuthHelper(IConfiguration configuration, IHttpContextAccessor contextAccessor, IUserDal userDal)
    {
        _configuration = configuration;
        _httpContextAccessor = contextAccessor;
        _userDal = userDal;
    }

    1 reference | asecer79, 13 days ago | 1 author, 1 change
    private IEnumerable<Claim> GetClaims(User user)...
    1 reference | 0 changes | 0 authors, 0 changes
    public async Task<bool> SecureSignIn(string userName,string password)...
    1 reference | asecer79, 13 days ago | 1 author, 1 change
    public async Task SingOut()...
}
```

GetClaims Method

- Query User claims from db and convert them to system security claims.

```
private IEnumerable<Claim> GetClaims(User user)
{
    var claims = new List<Claim> //Claim = System.Security.Claim
    {
        new Claim(type:ClaimTypes.Email, value:user.EMail),
        new Claim(type:ClaimTypes.Name, value:user.EMail),//username
        new Claim(type:ClaimTypes.NameIdentifier, value:$"{user.FirstName} {user.LastName}")
    };

    //add multiple roles from db
    var operationClaims:List<OperationClaim> = _userDal.GetUserOperationClaims(user.Id);

    foreach (var claim in operationClaims)
    {
        claims.Add(item:new Claim(type:ClaimTypes.Role, value:claim.Name));
    }

    return claims;
}
```

SecureSignIn Method

```
public async Task<bool> SecureSignIn(string userName,string password)
{
    var user = _userDal.GetUserByEmailAndPassword( email: userName, password);

    if (user != null)
    {

        var claims :IEnumerable<Claim> = GetClaims(user);

        var identity = new ClaimsIdentity(claims, CookieAuthenticationDefaults.AuthenticationScheme);

        var principal = new ClaimsPrincipal(identity);

        await _httpContextAccessor.HttpContext.SignInAsync(CookieAuthenticationDefaults.AuthenticationScheme,
            principal); // Task

        return true;
    }

    return false;
}
```

SingOut() Method

```
public async Task SingOut()
{
    await _httpContextAccessor.HttpContext.SignOutAsync(CookieAuthenticationDefaults.AuthenticationScheme);

}
```

Creating AuthController for Login Processes

- Do not forget [AllowAnonymous] attribute.
- Otherwise system will face infinity loops !!!!

```
[AllowAnonymous]
1 reference | asecer79, 13 days ago | 1 author, 1 change
public class AuthController : Controller
{
    private readonly AuthHelper _authHelper;

    0 references | asecer79, 13 days ago | 1 author, 1 change
    public AuthController(AuthHelper authHelper)...

    [AllowAnonymous]
    [HttpGet]
    0 references | asecer79, 13 days ago | 1 author, 1 change
    public IActionResult Login()...

    [AllowAnonymous]
    [HttpPost]
    0 references | asecer79, 13 days ago | 1 author, 1 change
    public async Task<IActionResult> Login(string userName,string password)...

    [HttpGet]
    0 references | asecer79, 13 days ago | 1 author, 1 change
    public async Task<IActionResult> LogOut()...

    [AllowAnonymous]
    [HttpGet]
    0 references | asecer79, 13 days ago | 1 author, 1 change
    public string Err()...
}
```

Login Actions

```
[AllowAnonymous]
[HttpGet]
0 references | asecer79, 13 days ago | 1 author, 1 change
public IActionResult Login()
{
    return View();
}

[AllowAnonymous]
[HttpPost]
0 references | asecer79, 13 days ago | 1 author, 1 change
public async Task<IActionResult> Login(string userName,string password)
{
    var isSuccess :bool = await _authHelper.SecureSignIn(userName,password);
    if (isSuccess)
        return RedirectToAction("Index", controllerName: "Home");

    return View();
}
```

LogOut and Err Actions

```
[HttpGet]
0 references | asecer79, 13 days ago | 1 author, 1 change
public async Task<IActionResult> LogOut()
{
  var data :IEnumerable<Claim> =  HttpContext.User.Claims;
   await _authHelper.SingOut();
    return RedirectToAction("Login");
}

[AllowAnonymous]
[HttpGet]
0 references | asecer79, 13 days ago | 1 author, 1 change
public string Err()
{
    return "UnAuthorized";
}
```

Login View

```
@{
    ViewData["Title"] = "Login";
}

<h1>Login</h1>


<form asp-action="Login" method="post">
    user <input type="" name="userName" />
    <hr />
    pwd <input type=""  name="password"/>
    <hr />
    <input type="submit" value="Login" />

</form>
```

Test

MONSTERHOME.School - dbo.Users | MONSTERHOME.School - Diagram_2* | MONSTE

	Id	FirstName	LastName	EMail	Password
▶	1	Aydin	Secer	asecer@yildiz.e...	123
	NULL	NULL	NULL	NULL	NULL

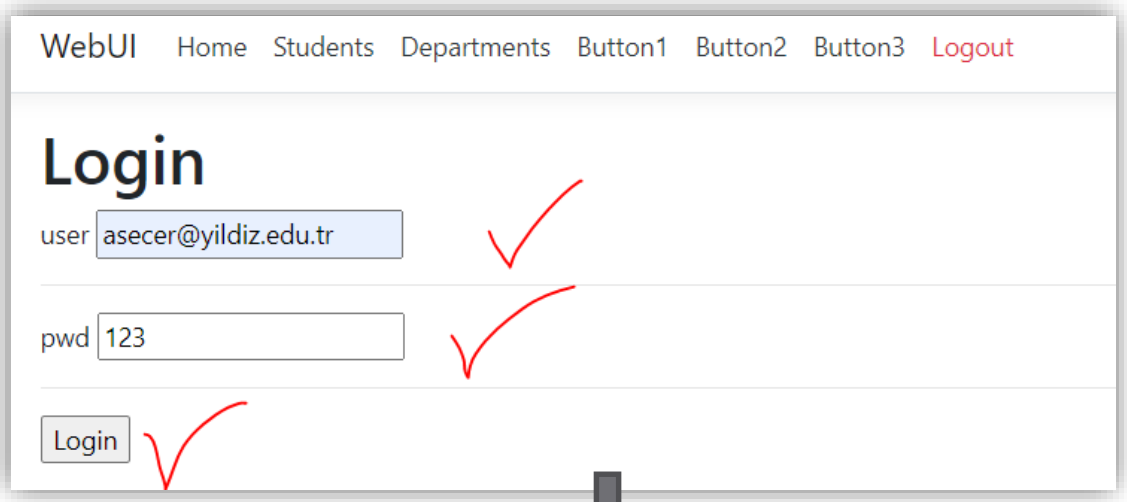
MONSTERHOME.Sch....OperationClaims

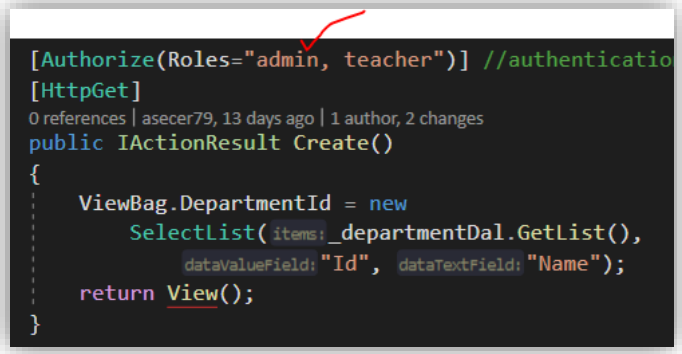
	Id	Name
▶	1	admin
	2	user
	3	student
	4	teacher
*	NULL	NULL

MONSTERHOME.Sch...OperationClaims | MONSTERHOME.

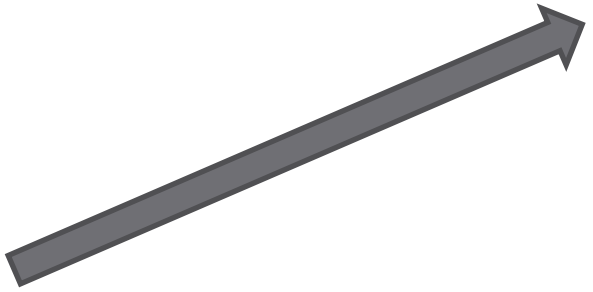
	Id	UserId	OperationClai...
▶	1	1	1
	2	1	2
	3	1	3
*	NULL	NULL	NULL

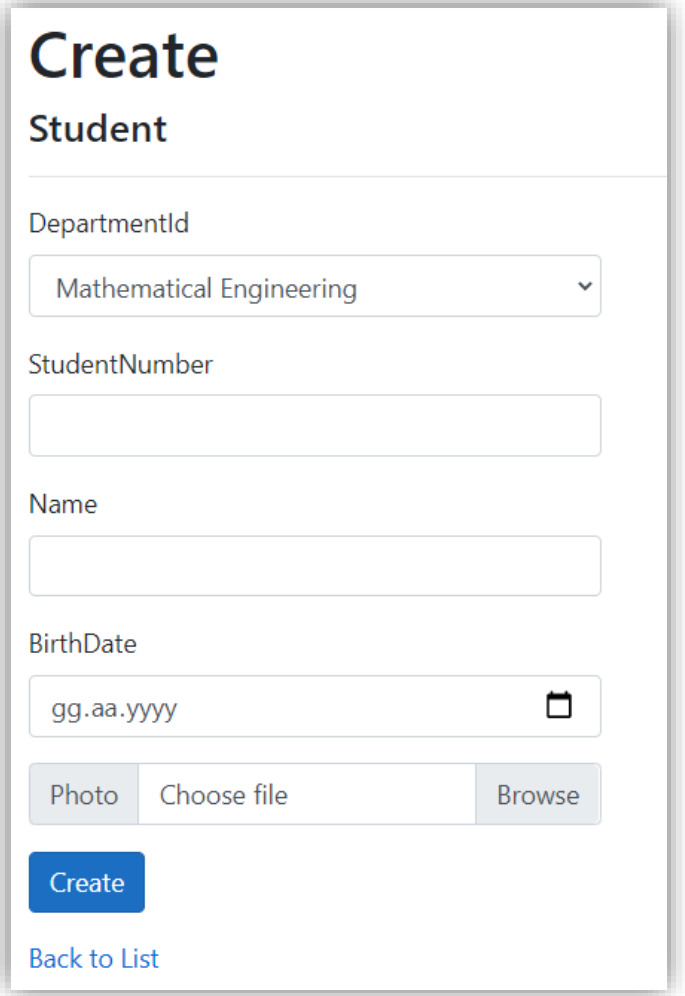
Test





```
[Authorize(Roles="admin, teacher")] //authenticatio
[HttpGet]
0 references | asecer79, 13 days ago | 1 author, 2 changes
public IActionResult Create()
{
    ViewBag.DepartmentId = new
        SelectList(items: _departmentDal.GetList(),
            dataValueField: "Id", dataTextField: "Name");
    return View();
}
```



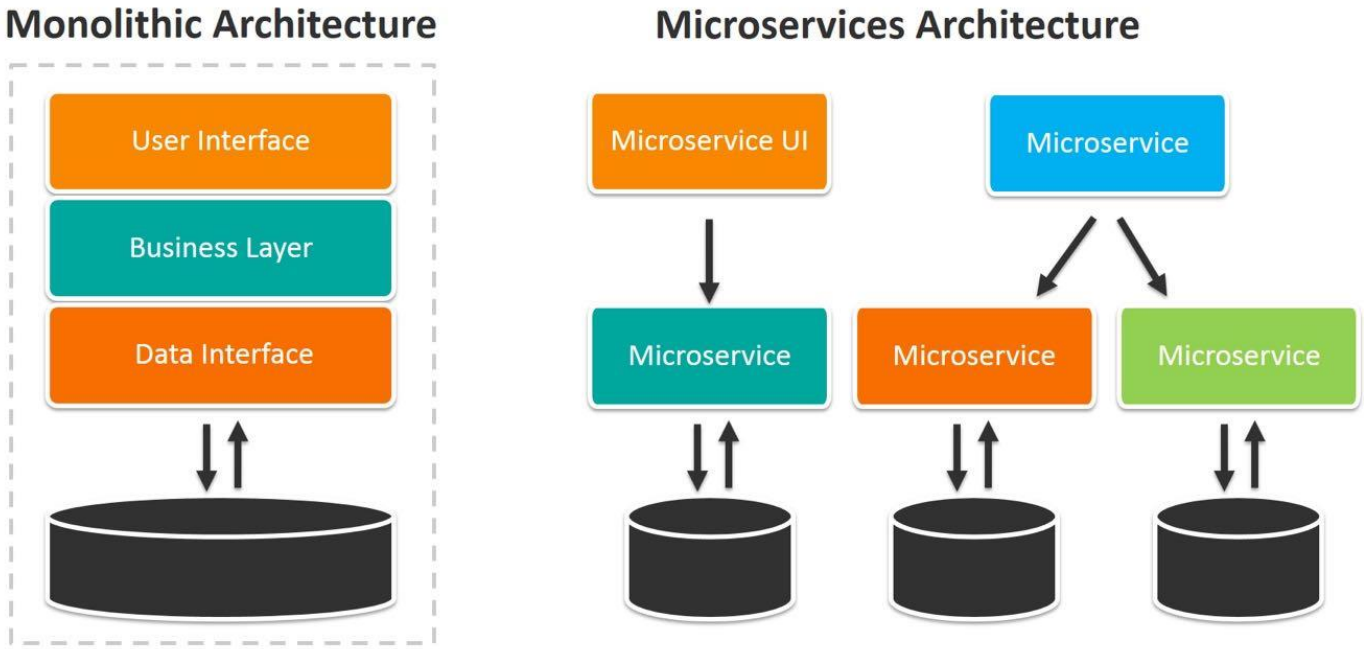


t

Web Application Architectures

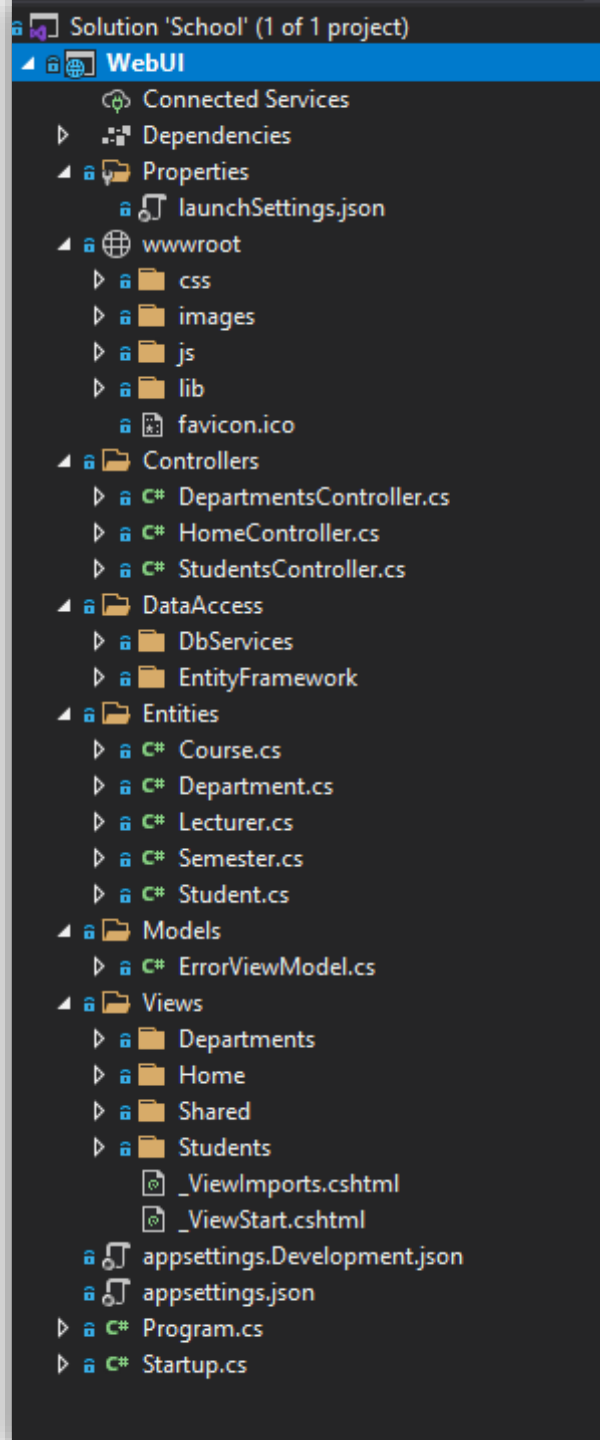
Monolithic Approach

- Monolithic application is one that is entirely self-contained, in terms of its behavior.
- It may interact with other services or data stores in the course of performing its operations, but the core of its behavior runs within its own process and the entire application is typically deployed as a single unit.
- If such an application needs to scale horizontally, typically the entire application is duplicated across multiple servers or virtual machines.



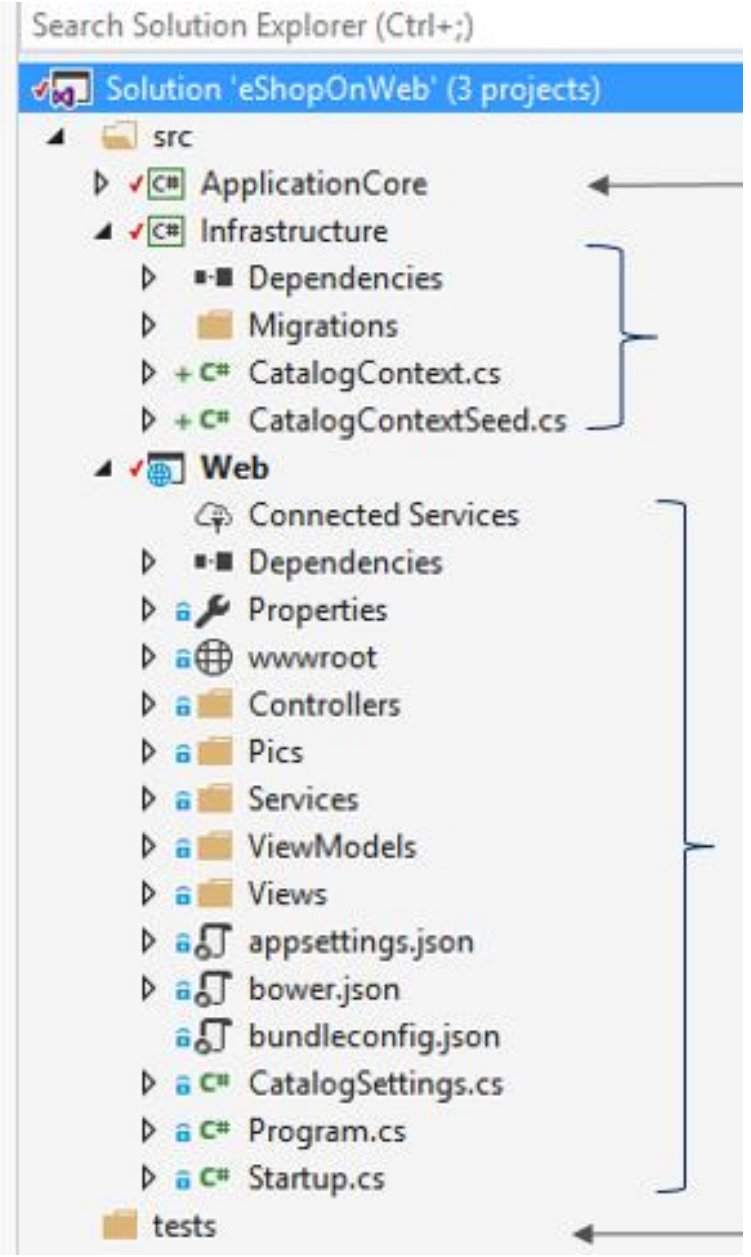
Monolithic: All-in-one applications

- In a single project scenario, separation of concerns is achieved through the use of folders.
- The default template includes separate folders for MVC pattern responsibilities of Models, Views, and Controllers, as well as additional folders for Data and Services.
- In this arrangement, presentation details should be limited as much as possible to the Views folder, and data access implementation details should be limited to classes kept in the Data folder.
- Business logic should reside in services and classes within the Models folder.
- Although simple, the single-project monolithic solution has some disadvantages.
- As the project's size and complexity grows, the number of files and folders will continue to grow as well.
- User interface (UI) concerns (models, views, controllers) reside in multiple folders, which aren't grouped together alphabetically.
- This issue only gets worse when additional UI-level constructs, such as Filters or ModelBinders, are added in their own folders.
- Business logic is scattered between the Models and Services folders, and there's no clear indication of which classes in which folders should depend on which others.
- This lack of organization at the project level frequently leads to spaghetti code.

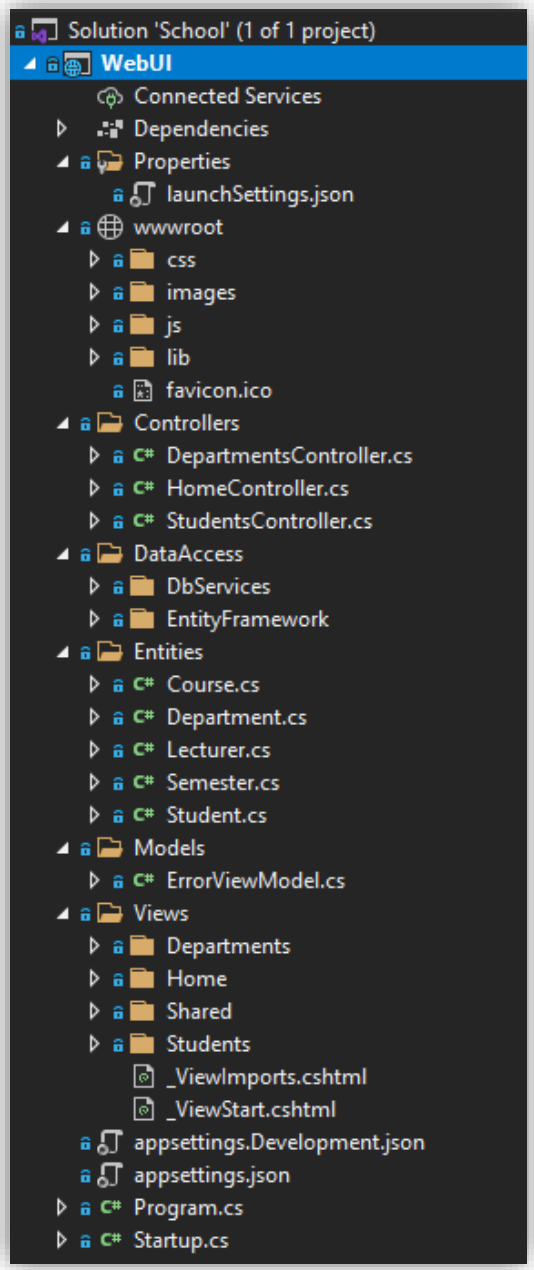


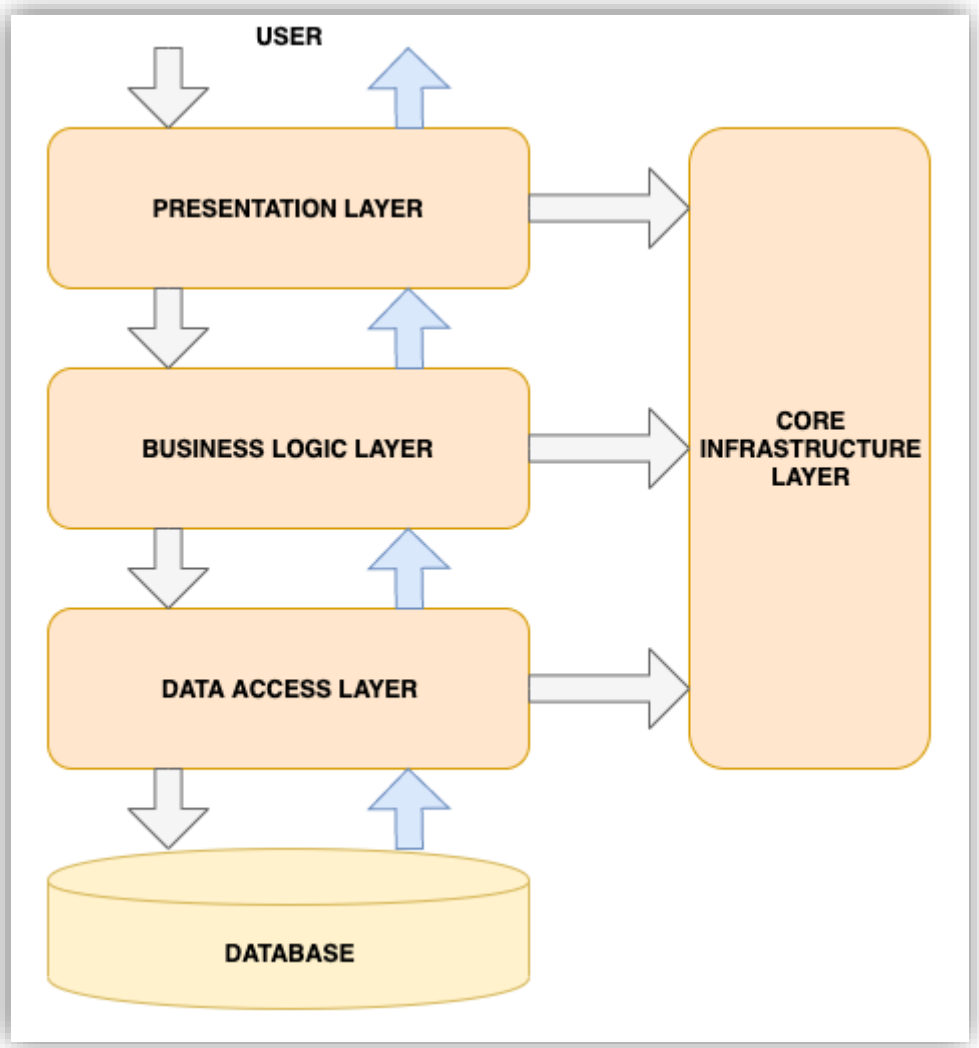
Monolithic: "N-Layer" Architecture Applications

- The layers are frequently abbreviated as UI, BLL (Business Logic Layer), and DAL (Data Access Layer).
- Using this architecture, users make requests through the UI layer, which interacts only with the BLL.
- The BLL, in turn, can call the DAL for data access requests.
- The UI layer shouldn't make any requests to the DAL directly, nor should it interact with persistence directly through other means.
- Likewise, the BLL should only interact with persistence by going through the DAL.
- In this way, each layer has its own well-known responsibility.
- One disadvantage of this traditional layering approach is that compile-time dependencies run from the top to the bottom.
- That is, the UI layer depends on the BLL, which depends on the DAL.
- This means that the BLL, which usually holds the most important logic in the application, is dependent on data access implementation details (and often on the existence of a database).
- Testing business logic in such an architecture is often difficult, requiring a test database.
- The dependency inversion principle can be used to address this issue.



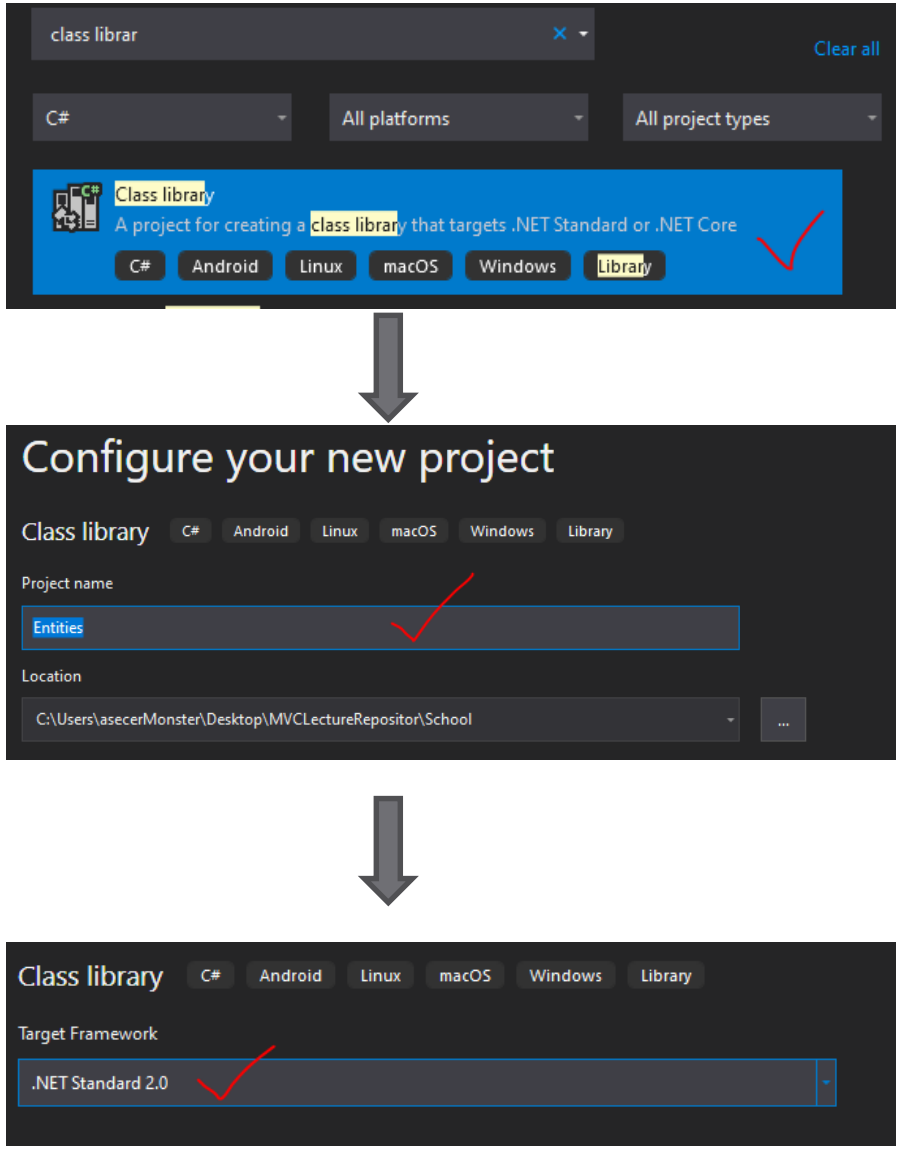
Converting the School Project to N-Layer Arch..



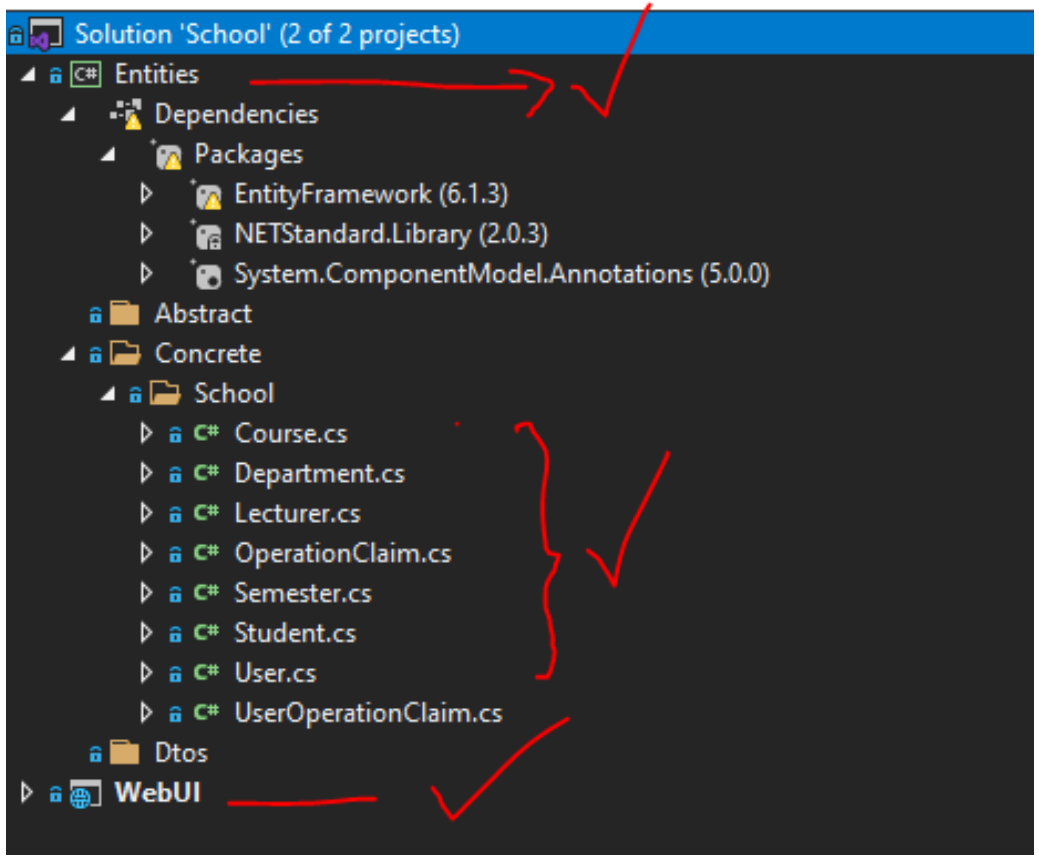


‘Entities’ Layer

- Add a new class library named ‘**Entities**’ Project as .Net Standard
- Move all entity classes into this Project
- Reorganize references and namespaces
- Reorganize if you have DI operations.
- Add Project reference to dependent Projects



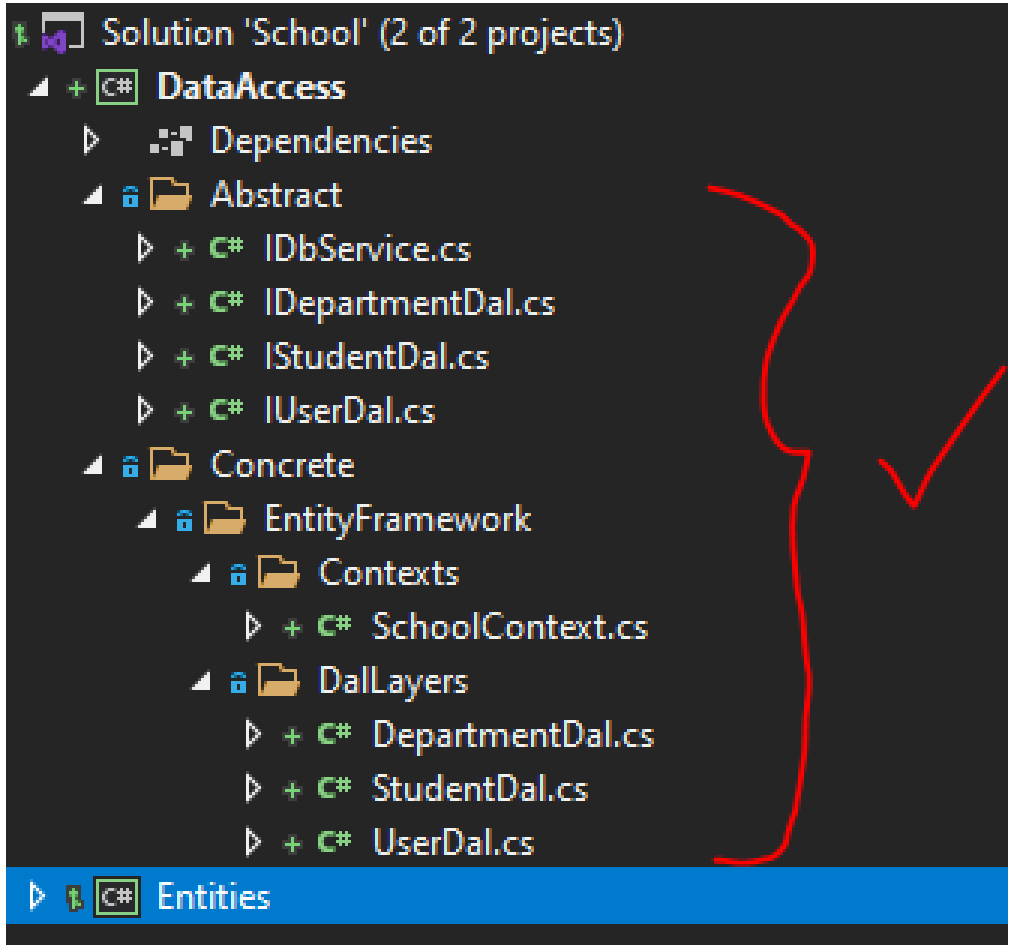
Project view of Entity Layer




```
using Entities.Concrete.School;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using System;
using System.Collections.Generic;
using System.IO;
using WebUI.CacheServices.Abstract;
using WebUI.DataAccess.DbServices.Abstract;


namespace WebUI.Controllers
{

    [Authorize(Roles = "admin")]
    1 reference | asecer79, 13 days ago | 1 author, 7 changes
    public class StudentsController : Controller
    {
        private readonly IStudentDal _studentDal;
        private readonly IDepartmentDal _departmentDal;
```

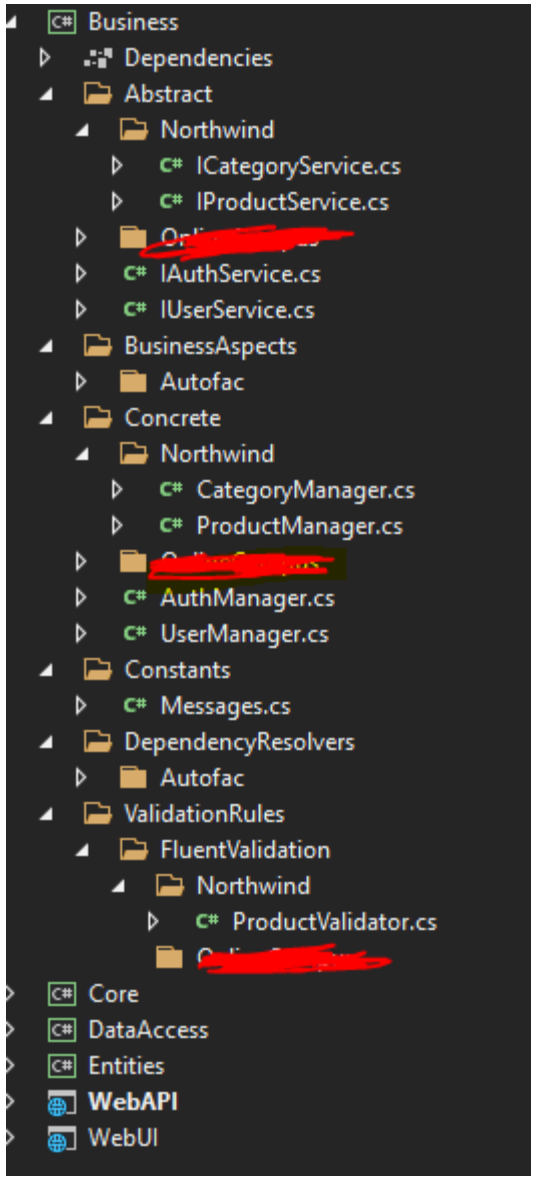
‘DataAccess’ Layer

- Do the same steps as Entities



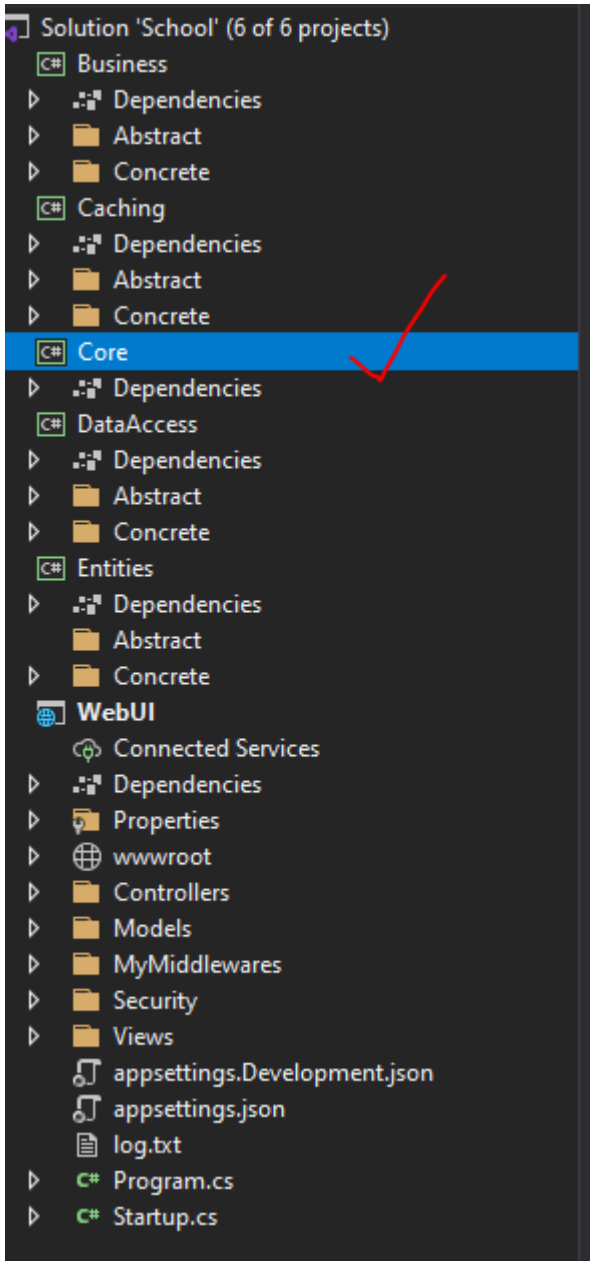
Business Layer

- --



Core Layer

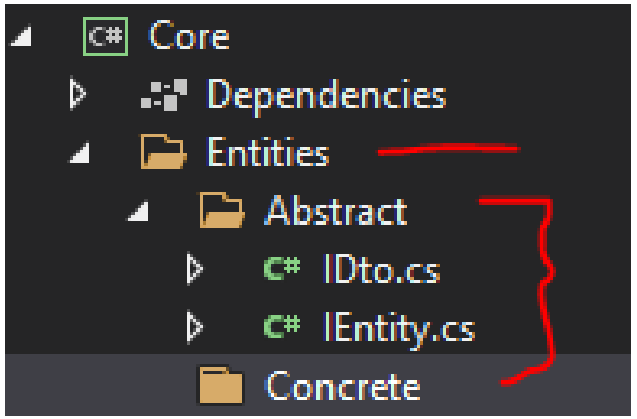
- The top of common approaches for application



Repository Pattern Implementation in Core Layer

Entites (Ientity and IDto interfaces

```
namespace Core.Entities.Abstract
{
    0 references
    public interface IEntity
    {
        // common operations for all entities
        //
    }
}
```




```
namespace Core.Entities.Abstract
{
    0 references
    public interface IDto
    {
        //data transfer objects common functionality operations
    }
}
```

Implementing Interfaces in Entity Layer

```
public class Student : IEntity
{
    7 references
    [Key] public int Id { get; set; }
```

```
public class User : IEntity
{
    [Key]
    2 references
    public int Id { get; set; }

    1 reference
    public string FirstName { get; set; }
```

.
.
.
.
.

```
public class Lecturer : IEntity
{
    [Key]
    0 references
    public int Id { get; set; }

    [ForeignKey(name: "DepartmentId")]
    0 references
    public int DepartmentId { get; set; }
```

DataAccess common operations in Core Layer

IEntityRepository<T> Generic Global Interface

```
namespace Core.DataAccess
{
    1 reference
    public interface IEntityRepository<T> where T : class, IEntity, new()
    {
        1 reference
        T Get(Expression<Func<T, bool>> filter);
        1 reference
        IList<T> GetList(Expression<Func<T, bool>> filter = null);
        1 reference
        T Add(T entity);
        1 reference
        void Update(T entity);
        1 reference
        void Delete(T entity);
    }
}
```

EfEntityRepositoryBase Class

```
public class EfEntityRepositoryBase<TEntity, TContext> : IEntityRepository<TEntity> where TEntity : class, IEntity, new()
where TContext : DbContext, new()
{
    1 reference
    public TEntity Get(Expression<Func<TEntity, bool>> filter)
    {
        using (var context = new TContext())
        {
            var record :TEntity = context.Set<TEntity>().FirstOrDefault(filter);
            return record;
        }
    }

    1 reference
    public IList<TEntity> GetList(Expression<Func<TEntity, bool>> filter = null)
    {
        using (var context = new TContext())
        {
            return filter != null
                ? context.Set<TEntity>().Where(filter).ToList()
                : context.Set<TEntity>().ToList();
        }
    }

    1 reference
    public TEntity Add(TEntity entity)
    {
        using (var context = new TContext())
        {
            var added :EntityEntry<TEntity> = context.Entry(entity);
            added.State = EntityState.Added;
            context.SaveChanges();
            return entity;
        }
    }

    1 reference
    public void Update(TEntity entity)
    {
        using (var context = new TContext())
        {
            var updated :EntityEntry<TEntity> = context.Entry(entity);
            updated.State = EntityState.Modified;
            context.SaveChanges();
        }
    }

    1 reference
    public void Delete(TEntity entity)
    {
        using (var context = new TContext())
        {
            var deleted :EntityEntry<TEntity> = context.Entry(entity);
            deleted.State = EntityState.Deleted;
            context.SaveChanges();
        }
    }
}
```

Implementing Interfaces in DataAccess Layer

```
namespace DataAccess.Abstract
{
    4 references
    public interface IStudentDal : IEntityRepository<Student>
    {



        //Extra methods here that does not exists in main repository
        //for example
        0 references
        List<Student> GetListIncludedDepartment(Expression<Func<Student, bool>> filter = null);
        1 reference
        Student GetIncludedDepartment(int id);
    }
}
```

Implementig Repository Base Class Dal Layers

```
namespace DataAccess.Concrete.EntityFramework.DalLayers
{
    1 reference
    public class StudentDal: EfEntityRepositoryBase<Student, SchoolContext>, IStudentDal
    {

        1 reference
        public Student GetIncludedDepartment(int id)
        {
            using (SchoolContext context = new SchoolContext())
            {
                return context.Set<Student>().Include("Department").FirstOrDefault(p :Student => p.Id == id);
            }
        }

        1 reference
        public List<Student> GetListIncludedDepartment(Expression<Func<Student, bool>> filter = null)
        {
            using (SchoolContext context = new SchoolContext())
            {

                return filter == null
                    ? context.Set<Student>().Include("Department").ToList()
                    : context.Set<Student>().Include("Department").Where(filter).ToList();
            }
        }
    }
}
```

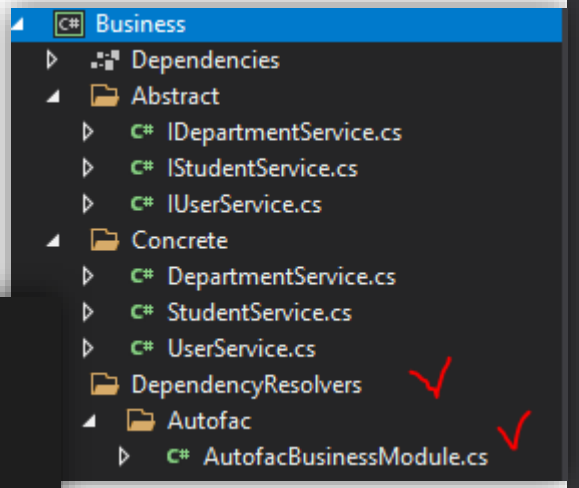
Modifying Business Layer

```
namespace Business.Abstract
{
    4 references
    public interface IStudentService
    {
        6 references
        Student Get(int id);
        2 references
        Student Add(Student entity);
        2 references
        void Remove(Student entity);
        2 references
        void Update(Student entity);
        2 references
        List<Student> GetList(Expression<Func<Student, bool>> filter = null);

    }
}
```

```
public class StudentService:IStudentService
{
    private readonly IStudentDal _studentDal;

    0 references
    public StudentService(IStudentDal studentDal)
    {
        _studentDal = studentDal;
    }

    6 references
    public Student Get(int id)
    {
        return _studentDal.GetIncludedDepartment(id);
    }

    2 references
    public Student Add(Student entity)
    {
        return _studentDal.Add(entity);
    }

    2 references
    public void Remove(Student entity)
    {
        _studentDal.Delete(entity);
    }

    2 references
    public void Update(Student entity)
    {
        _studentDal.Update(entity);
    }

    2 references
    public List<Student> GetList(Expression<Func<Student, bool>> filter = null)
    {
        return _studentDal.GetListIncludedDepartment(filter);
    }
}
```

Dependency Resolver in Business Layer (**Autofac**)

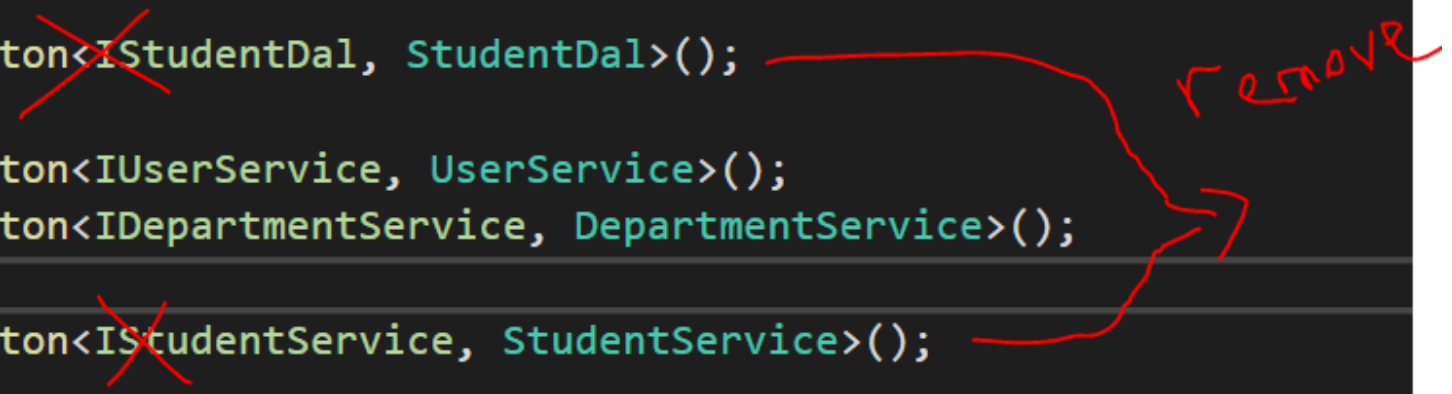
- Install **Autofac** Nuget in WEB UI and Business Layer




```
namespace Business.DependencyResolvers.Autofac
{
    0 references
    public class AutofacBusinessModule : Module
    {
        0 references
        protected override void Load(ContainerBuilder builder)
        {
            builder.RegisterType<StudentService>().As<IStudentService>();
            builder.RegisterType<StudentDal>().As<IStudentDal>();
        }
    }
}
```

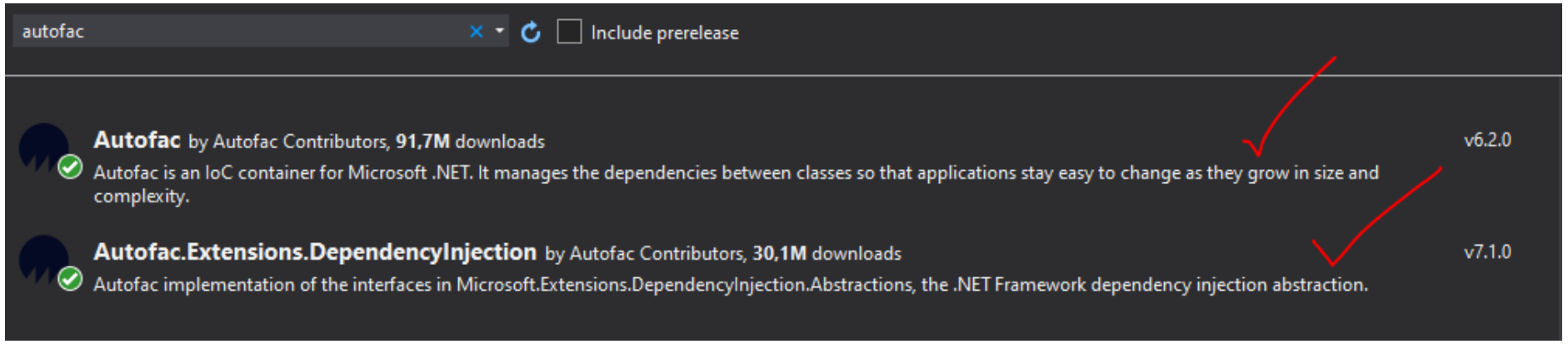
Modify Web UI Startup.cs

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

    services.AddHttpContextAccessor();

    services.AddSingleton<ICacheService, MemoryCacheService>();


    services.AddSingleton<IUserDal, UserDal>();
    services.AddSingleton<IDepartmentDal, DepartmentDal>();

    services.AddSingleton<IStudentDal, StudentDal>();

    services.AddSingleton<IUserService, UserService>();
    services.AddSingleton<IDepartmentService, DepartmentService>();

    services.AddSingleton<IStudentService, StudentService>();


    services.AddSingleton<AuthHelper>();
```



Modify Modify Web UI Program.cs

- Install following nugets on WEB UI




```
public static IHostBuilder CreateHostBuilder(string[] args) =>
    Host.CreateDefaultBuilder(args)

        .UseServiceProviderFactory(new AutofacServiceProviderFactory())
        .ConfigureContainer<ContainerBuilder>(builder =>
        {
            builder.RegisterModule(new AutofacBusinessModule());
        })
        .ConfigureWebHostDefaults(webBuilder =>
        {
            webBuilder.UseStartup<Startup>();
        }); // IHostBuilder
```

Fluent Validation in Business Layer

--

- --

--

- --

--

- --

--

- --

--

- --

--

- --

--

- --

--

- --

--

- --

--

- --

--

- --

--

- --

--

- --

--

- --

--

- --

--

- --

--

- --

--

- --

--

- --

--

- --

• • •