ASP.NET CORE MVC MODULE 06
DEVELOPING MODELS

Summer 2021 – Web Development using ASP .Net Core MVC

MAIN SOURCES FOR THESE SLIDED

- Unless otherwise specified, the main sources for these slides are:
  - https://github.com/MicrosoftLearning/20486D-DevelopingASPNETMVCWebApplications ←for homework
  - https://docs.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-5.0 ←for "textbook"

THE MVC ARCHITECTURAL PATTERN

- See page 1630

- Models:
  - classes representing **the data** of an application.
  - often, model objects will retrieve/store data from/in a **database**.

- Views:
  - are components that **display the user interface**(UI).
  - typically, they display the **model** data.

- Controllers:
  - classes that **handle requests** from browsers.
  - may retrieve **model** data.
  - often call **view** templates to send back a response to the browser requests.

- Notes:
  - Typically, we have one **controller** class for each **model** class. (Student ← model, StudentController ← controller)
  - Each **controller** can have **multiple** views (often, each **action** has its own **view**)

EXAMPLES OF MODEL CLASSES

- For each of the following applications, think of/give examples of model classes:
- Amazon.com
- A Learning Management System (such as Canvas, Moodle, Blackboard)
- Facebook
- Redbox

IN-CLASS DEMO PROJECT - MODELS

- For today, let's start with an MVC web application.

- Then, let's add two **model** classes (add at least one **DateTime** property, maybe a **bool** too).
  - Student
  - Major

- Alternatively, we can use:
  - Actor
  - Movie

- These **model** classes are known as **POCO** classes (from Plain Old CLR Objects)
  - To add a model class, right click on the **Models** folder, then select **Add** > **Class**.

- **Note**: when these model classes will be used in conjunction with a database (we'll see Entity Framework soon), then an **Id** field (used by the database for the primary key) will be useful.

- Add **attribute**: [DataType(DataType.Date)] ← if we only need Date (no Time)!
- If time: "link" the two model classes (below is a 1 – to – many relationship) ← we'll see this again later!
  - public Major Major{get; set;}
  - public ICollection<Student> Students {get;set;}

IN-CLASS DEMO PROJECT - MODELS

```
public class Student
{
    public string FirstName { get; set; }
    public string Major { get; set; }
    public double GPA { get; set; }
    public DateTime AdmissionDate { get; set; }
    public bool IsInternational { get; set; }
}
```

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Threading.Tasks;

namespace FirstMVCApplication.Models
{
    public class Student
    {
        public string FirstName { get; set; }

        public string Major { get; set; }

        public double GPA { get; set; }

        [DataType(DataType.Date)]
        public DateTime AdmissionDate { get; set; }

        public bool IsInternational { get; set; }
    }
}
```

IN-CLASS DEMO PROJECT - CONTROLLERS

- Now let's create a **Controller** class for either one of those **model** classes.
  - In here we'll add a few **action** methods.

- In it, let's create/hard code a List of Students/Movies (later we'll get this data from a database).
  - Add a few instances to that list

- Now let's three action methods:
  - One **action** will be used to display one Student/Actor from the list, given a valid Id ← ShowDetails
    - If Id is not valid, return **NotFound**()
    - If Id is valid, pass an instance of the model to a View and display those details.
  - One **action** will be used to display a list of all Students/Actors from the list ← ListAll
  - One **action** will be used to display a form to allow the user to enter new Student/Actor. ← Add

```
return NotFound();
```

NotFoundResult ControllerBase.NotFound() (+ 1 overload)
Creates an NotFoundResult that produces a Microsoft.AspNetCore.Http.StatusCodes.Status404NotFound response.

Returns:
The created NotFoundResult for the response.

IN-CLASS DEMO PROJECT - CONTROLLERS

```
public class StudentController : Controller
{
    List<Student> allStudents = new List<Student>();
    public StudentController()
    {
        allStudents.Add(new Student() { FirstName = "Achak", AdmissionDate = DateTime.Now, GPA = 3.25, IsInternational = false, Major = "Art" });
        allStudents.Add(new Student() { FirstName = "Serena", AdmissionDate = DateTime.Now, GPA = 3.19, IsInternational = false, Major = "Computer Science" });
        allStudents.Add(new Student() { FirstName = "Vivek", AdmissionDate = DateTime.Now, GPA = 3.05, IsInternational = true, Major = "Mathematics" });
        allStudents.Add(new Student() { FirstName = "Young", AdmissionDate = DateTime.Now, GPA = 3.37, IsInternational = false, Major = "Ecology" });
    }

    public IActionResult ShowDetails(int id)
    {
        if (id < 0 || id >= allStudents.Count)
            return NotFound();
        else
            return View(allStudents[id]);
    }

    public IActionResult ListAll()
    {
        return View(allStudents);
    }

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

    public IActionResult Add(Student st)
    {
        return Content("a new student was added to the list");
    }
}
```

IN-CLASS DEMO PROJECT - VIEWS

- A view can be strongly typed (has the **@model** declaration at the top of the view page)
  - The view will receive this as an argument of the **View** method call
  - One can pass one instance of the model: @model MyApp.Models.Student
  - One can pass a collection of instances of the model: @model IEnumerable<MyApp.Models.Student>
    - Use @foreach to iterate through this collection

- A view can be dynamically typed (does not have the **@model** declaration in the view)
  - Use this if an **action does not need to pass a model** from an action to a **view**
  - Use this if an **action** need to pass more than one model classes to a **view**
    - Before using the model inside the view, make sure to check for **null**!

- Let's create a **view** for each **action**:
  - One **action** will be used to display one Student/Actor from the list, given a valid Id ← ShowDetails
  - One **action** will be used to display a list of all Students/Actors from the list ← ListAll
  - One **action** will be used to display a form to allow the user to enter new Student/Actor. ← Add

SHOWDETAILS VIEW

- Here is a first view:

ShowDetails.cshtml | Add.cshtml | ListAll.cshtml | Startup.cs | StudentController.cs

```
@model FirstMVCApplication.Models.Student

@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>ShowDetails</title>
</head>
<body>
    <h1>Showing details for a student </h1>
    <p>Name:  @Model.FirstName</p>
    <p>Major: @Model.Major</p>
    <p>Admission Date: @Model.AdmissionDate</p>
    <p>GPA: @Model.GPA</p>
    <p>Is International? @Model.IsInternational</p>
</body>
</html>
```

ShowDetails

localhost:59237/Student/ShowDetails/2

Showing details for a student

Name: Vivek

Major: Mathematics

Admission Date: 4/22/2021 4:59:10 PM

GPA: 3.05

Is International? True

localhost

localhost:59237/Student/ShowDetails/20

This localhost page can't be found

No webpage was found for the web address: **http://localhost:59237/Student/ShowDetails/20**

HTTP ERROR 404

Reload

SHOWDETAILS VIEW

- Here is a better view (using **HTML helpers**):
  - Html.**DisplayNameFor** renders the **display name** of a model class property
  - **Html.DisplayFor** renders the **value** of a model class property.

ShowDetails.cshtml | Add.cshtml | ListAll.cshtml | Startup.cs | StudentController.cs | Student.cs

```
@model FirstMVCApplication.Models.Student

@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>ShowDetails</title>
</head>
<body>
    <h1>Showing details for a student </h1>
    <p>@Html.DisplayNameFor(model => model.FirstName)  @Html.DisplayFor(model => model.FirstName)</p>
    <p>@Html.DisplayNameFor(model => model.Major)  @Html.DisplayFor(model => model.Major)</p>
    <p>@Html.DisplayNameFor(model => model.AdmissionDate)  @Html.DisplayFor(model => model.AdmissionDate)</p>
    <p>@Html.DisplayNameFor(model => model.GPA)  @Html.DisplayFor(model => model.GPA)</p>
    <p>@Html.DisplayNameFor(model => model.IsInternational)  @Html.DisplayFor(model => model.IsInternational)</p>
</body>
</html>
```

ShowDetails

localhost:59237/Student/ShowDetails/2

Showing details for a student

FirstName Vivek

Major Mathematics

AdmissionDate 4/22/2021

GPA 3.05

IsInternational ☑

SHOWDETAILS VIEW

- An even better view (using **Display Data Annotation**):
  - **FirstName** is not very user friendly
  - **Display** Data Annotation can be used to get a better display (such as **First Name**)

```
public class Student
{
    [Display(Name ="First Name")]
    public string FirstName { get; set; }

    public string Major { get; set; }

    public double GPA { get; set; }

    [Display(Name = "Admission Date")]
    [DataType(DataType.Date)]
    public DateTime AdmissionDate { get; set; }

    [Display(Name = "Is International")]
    public bool IsInternational { get; set; }
}
```

ShowDetails.cshtml | Add.cshtml | ListAll.cshtml | Star

```
@model FirstMVCApplication.Models.

@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content=
    <title>ShowDetails</title>
</head>
<body>
    <h1>Showing details for a student </h1>
    <p>@Html.DisplayNameFor(model => model.FirstName)  @Html.DisplayFor(model => model.FirstName)</p>
    <p>@Html.DisplayNameFor(model => model.Major)  @Html.DisplayFor(model => model.Major)</p>
    <p>@Html.DisplayNameFor(model => model.AdmissionDate)  @Html.DisplayFor(model => model.AdmissionDate)</p>
    <p>@Html.DisplayNameFor(model => model.GPA)  @Html.DisplayFor(model => model.GPA)</p>
    <p>@Html.DisplayNameFor(model => model.IsInternational)  @Html.DisplayFor(model => model.IsInternational)</p>
</body>
</html>
```

ShowDetails

localhost:59237/Student/ShowDetails/2

Showing details for a student

First Name Vivek

Major Mathematics

Admission Date 4/22/2021

GPA 3.05

Is International ☑

LISTALL VIEW

- To pass a collection to a strongly typed view, use **IEnumerable**
- To iterate over a collection, use **foreach**

ListAll.cshtml | ShowDetails.cshtml | Add.cshtml | Startup.cs | StudentController.cs | Student.cs

```
@model IEnumerable<FirstMVCApplication.Models.Student>
@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>ListAll</title>
</head>
<body>
    <h1>Showing ALL students </h1>
    @foreach (var student in Model)
    {
        <p>@Html.DisplayNameFor(model => model.FirstName)  @student.FirstName</p>
        <p>@Html.DisplayNameFor(model => model.Major)  @student.Major</p>
        <p>@Html.DisplayNameFor(model => model.AdmissionDate)  @student.AdmissionDate</p>
        <p>@Html.DisplayNameFor(model => model.GPA)  @student.GPA</p>
        <p>@Html.DisplayNameFor(model => model.IsInternational)  @student.IsInternational</p>
        <br />
        <br />
    }
</body>
</html>
```

ListAll | localhost:59237/Student/ListAll

Showing ALL students

First Name Achak

Major Art

Admission Date 4/22/2021 5:42:17 PM

GPA 3.25

Is International False

First Name Serena

Major Computer Science

Admission Date 4/22/2021 5:42:17 PM

GPA 3.19

Is International False

First Name Vivek

Major Mathematics

Admission Date 4/22/2021 5:42:17 PM

GPA 3.05

Is International True

First Name Young

Major Ecology

Admission Date 4/22/2021 5:42:17 PM

GPA 3.37

ACTION NAME ≠ VIEW NAME

- By default, the **action** name and the **view** name are expected to be the same.
  - To call a **view** with a name different than the **action** name, pass the name of the **view** to the **view()** method

```
public IActionResult ListAll()
{
    return View(allStudents);
}

public IActionResult DisplayAll()
{
    return View("ListAll", allStudents);
}
```

ListAll.cshtml | ShowDetails.cshtml | Add.cshtml | Startup.cs | StudentController.cs | Student.cs

```
@model IEnumerable<FirstMVCApplication.Models.Student>
@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>ListAll</title>
</head>
<body>
    <h1>Showing ALL students </h1>
    @foreach (var student in Model)
    {
        <p>@Html.DisplayNameFor(model => model.FirstName)  @student.FirstName</p>
        <p>@Html.DisplayNameFor(model => model.Major)  @student.Major</p>
        <p>@Html.DisplayNameFor(model => model.AdmissionDate)  @student.AdmissionDate</p>
        <p>@Html.DisplayNameFor(model => model.GPA)  @student.GPA</p>
        <p>@Html.DisplayNameFor(model => model.IsInternational)  @student.IsInternational</p>
        <br />
        <br />
    }
</body>
</html>
```

ListAll

localhost:59237/Student/DisplayAll

Showing ALL students

First Name Achak

Major Art

Admission Date 4/22/2021 5:48:12 PM

GPA 3.25

Is International False

First Name Serena

Major Computer Science

Admission Date 4/22/2021 5:48:12 PM

GPA 3.19

Is International False

First Name Vivek

Major Mathematics

Admission Date 4/22/2021 5:48:12 PM

- Notice the URL!!!

ACTION NAME ≠ VIEW NAME

- One can also redirect to another action
  - Call: http://localhost:59237/Student/DisplayAll

```
public IActionResult ListAll()
{
    return View(allStudents);
}

public IActionResult DisplayAll()
{
    return RedirectToAction("ListAll");
}
```

- Notice the URL!!!

ListAll.cshtml | ShowDetails.cshtml | Add.cshtml | Startup.cs | StudentController.cs | Student.cs

```
@model IEnumerable<FirstMVCApplication.Models.Student>
@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>ListAll</title>
</head>
<body>
    <h1>Showing ALL students </h1>
    @foreach (var student in Model)
    {
        <p>@Html.DisplayNameFor(model => model.FirstName)  @student.FirstName</p>
        <p>@Html.DisplayNameFor(model => model.Major)  @student.Major</p>
        <p>@Html.DisplayNameFor(model => model.AdmissionDate)  @student.AdmissionDate</p>
        <p>@Html.DisplayNameFor(model => model.GPA)  @student.GPA</p>
        <p>@Html.DisplayNameFor(model => model.IsInternational)  @student.IsInternational</p>
        <br />
        <br />
    }
</body>
</html>
```

ListAll

localhost:59237/Student/ListAll

Showing ALL students

First Name Achak

Major Art

Admission Date 4/22/2021 5:52:42 PM

GPA 3.25

Is International False

First Name Serena

Major Computer Science

Admission Date 4/22/2021 5:52:42 PM

GPA 3.19

Is International False

First Name Vivek

Major Mathematics

Admission Date 4/22/2021 5:52:42 PM

GET VS POST

- We’ve briefly seen **GET** and **POST** verbs earlier. In here we’ll see how to use them.
  - Both are used for **sending client information to web server**.
  - See here more details: https://www.w3schools.com/tags/ref_httpmethods.asp
  - GET Example: https://www.w3schools.com/html/tryit.asp?filename=tryhtml_form_get
  - POST Example: https://www.w3schools.com/html/tryit.asp?filename=tryhtml_form_post
  - Other HTTP Verbs: **PUT**, **HEAD**, **DELETE**, **PATH**, …

```html
<form action="/action_page.php" target="_blank" method="post">
  <label for="st_name">Name:</label><br>
  <input type="text" id="st_name" name="student_name" value="Alex"><br>

  <label for="major">Last name:</label><br>
  <input type="text" id="major" name="student_major" value="Art"><br><br>

   <input type="submit" value="Submit">
</form>
```

Name:
Alex
Last name:
Art
Submit

w3schools.com/action_page.php

Submitted Form Data

Your input was received as:

student_name=Alex&student_major=Art

```html
<form action="/action_page.php" target="_blank" method="get">
  <label for="st_name">Name:</label><br>
  <input type="text" id="st_name" name="student_name" value="Alex"><br>

  <label for="major">Last name:</label><br>
  <input type="text" id="major" name="student_major" value="Art"><br><br>

   <input type="submit" value="Submit">
</form>
```

Name:
Alex
Last name:
Art
Submit

w3schools.com/action_page.php?student_name=Alex&student_major=Art

Submitted Form Data

Your input was received as:

student_name=Alex&student_major=Art

GET VS POST

- **GET** requests:
  - can be cached
  - can be seen in the browser history
  - can be bookmarked
  - should not be used when dealing with sensitive data (such as username and password)
  - have length restrictions

- **POST** requests:
  - do not remain in the browser history
  - cannot be bookmarked
  - have no restrictions on data length

- **Source**: https://www.w3schools.com.cach3.com/tags/ref_httpmethods.asp.html
- See also:
  - https://medium.com/@NikiMichaelsonqiv/http-methods-get-vs-post-b3ffb60c7f55
  - https://www.c-sharpcorner.com/blogs/difference-between-get-and-post1
  - https://www.udemy.com/course/learn-html-for-beginners/learn/lecture/14509330#overview
  - https://www.completecsharptutorial.com/asp-net-mvc5/asp-net-mvc-5-httpget-and-httppost-method-with-example.php

ADD VIEW – THE GET

```
public class Student
{
    [Display(Name ="First Name")]
    public string FirstName { get; set; }

    public string Major { get; set; }

    public double GPA { get; set; }

    [Display(Name = "Admission Date")]
    [DataType(DataType.Date)]
    public DateTime AdmissionDate { get; set; }

    [Display(Name = "Is International")]
    public bool IsInternational { get; set; }
}
```

- Next, we would like to create HTML elements that allow us to add new students.
- **GET** vs **POST** (we'll see this next):
  - **GET** request: send a request ← <u>with no data</u>, we just want to get a form (to enter a new student)
  - **POST** request: send a request ← <u>with data</u> (we want to create a new student)
- We'll use the HTML helper …
  - @Html.EditorForModel()
  - Yields an input element for each property from the model.
- **Action** method, by default, accept all **HTTP verbs**, including **GET** and **POST**.
  - To restrict them to only some verbs, use following attributes: [**HttpGet**], [**HttpPost**], … (see page 1710)

StudentController.cs | Startup.cs | Student.cs

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

Add

localhost:59237/Student/Add

First Name

Major

GPA

Admission Date

mm/dd/yyyy

Is International

Startup.cs | Add.cshtml | StudentController.cs | Student.cs

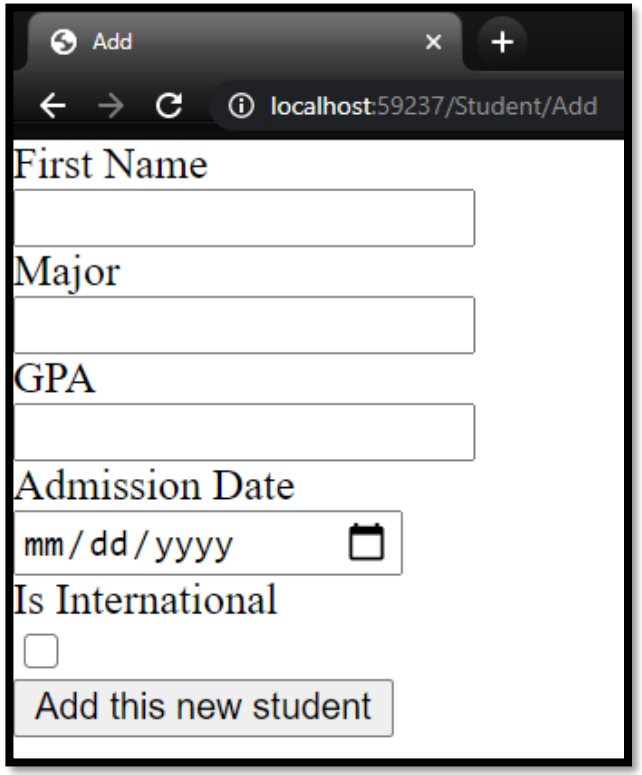
```
@model FirstMVCApplication.Models.Student

@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Add</title>
</head>
<body>
    @Html.EditorForModel()

</body>
</html>
```

ADD VIEW – THE GET

- Next, we would like to create **forms** that allow us to send/submit user data to the server.

Startup.cs | Add.cshtml | StudentController.cs | Student.cs

```
@model FirstMVCApplication.Models.Student

@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Add</title>
</head>
<body>
    @using (Html.BeginForm())
    {
        @Html.EditorForModel()
        <input type="submit" value="Add this new student" />
    }


</body>
</html>
```

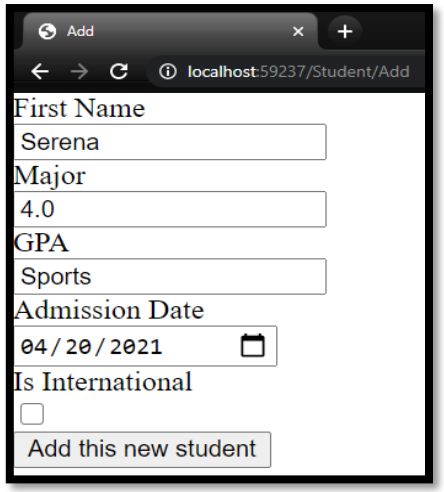


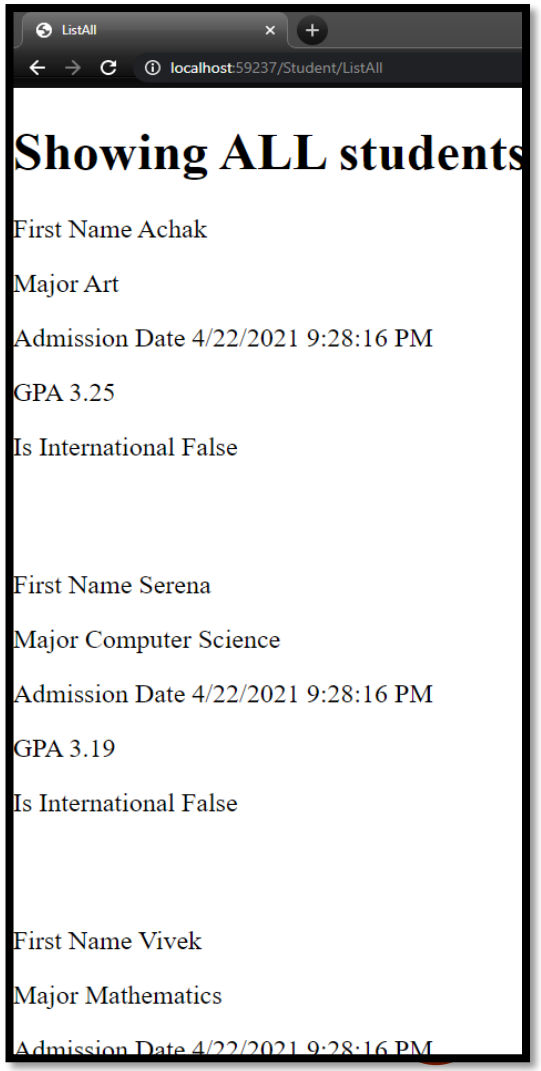
- If time: change the **HttpGet** into **HttpPost** …
  - Why doesn't it work anymore?

ADD VIEW – THE POST

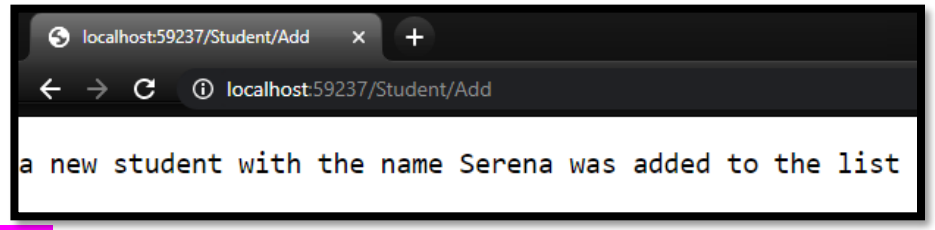
- Yes, actions can be **overloaded**!
  - We'll have an **Add** action method that used a parameter of type **Student**

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

[HttpPost]
public IActionResult Add(Student st)
{
    allStudents.Add(st);
    return Content($"a new student with the name {st.FirstName} was added to the list");
}
```





  - Test your work.
    - Add a new student.
    - Then list all students.
    - Warning … those changes don't persist!
      To test, use breakpoints … Later we'll use a DB
  - Note the URL!



- Who passed the Student object to the second **Add** method?
  - The **model binder** created an instance of a Student class, based on the data sent in the request and based on the **action** being called.

MODEL BINDING - REVIEW

- The **model binding** system, among other things:
  - **Retrieves** data from various sources such as (in order!)
    - Form fields
    - Route data
    - Query string parameters
    - Uploaded files
  - **Provides** this data to **controllers**.
  - **Converts** (when needed) string data to .NET types.

- Example:
  - **Action**:
    - [HttpGet("{id}")]
      public IActionResult SomeAction(int id, bool isPriority)
  - **Request:**
    - http://mysite.com/Home/SomeAction/70?ISPRIORITY=true
  - **Notes**:
    - 70 is converted from string into int
    - true is converted to bool
    - isPriority ← not case sensitive when doing the matching

- Source: see page 5649

MODEL BINDING - EXTRA

- Read the following resource for more info on model binding:
  - https://docs.microsoft.com/en-us/aspnet/core/mvc/models/model-binding?view=aspnetcore-5.0

- In particular:

Sources

By default, model binding gets data in the form of key-value pairs from the following sources in an HTTP request:

1. Form fields
2. The request body (For controllers that have the [ApiController] attribute.)
3. Route data
4. Query string parameters
5. Uploaded files

For each target parameter or property, the sources are scanned in the order indicated in the preceding list. There are a few exceptions:

- Route data and query string values are used only for simple types.
- Uploaded files are bound only to target types that implement `IFormFile` or `IEnumerable<IFormFile>`.

If the default source is not correct, use one of the following attributes to specify the source:

- [FromQuery] - Gets values from the query string.
- [FromRoute] - Gets values from route data.
- [FromForm] - Gets values from posted form fields.
- [FromBody] - Gets values from the request body.
- [FromHeader] - Gets values from HTTP headers.

These attributes:

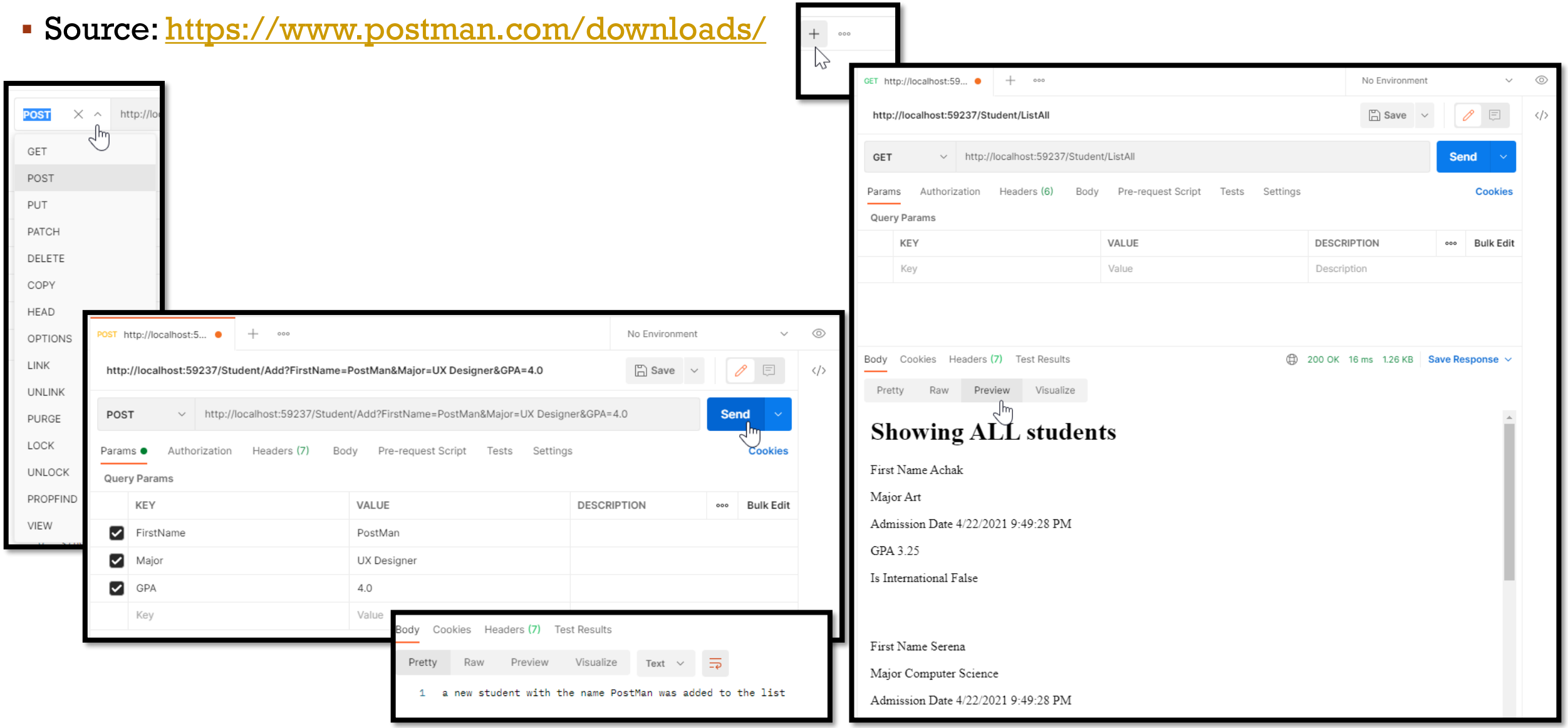
- Are added to model properties individually (not to the model class), as in the following example:

C# Copy

```
public class Instructor
{
    public int ID { get; set; }

    [FromQuery(Name = "Note")]
    public string NoteFromQueryString { get; set; }
```

POSTMAN - IF TIME

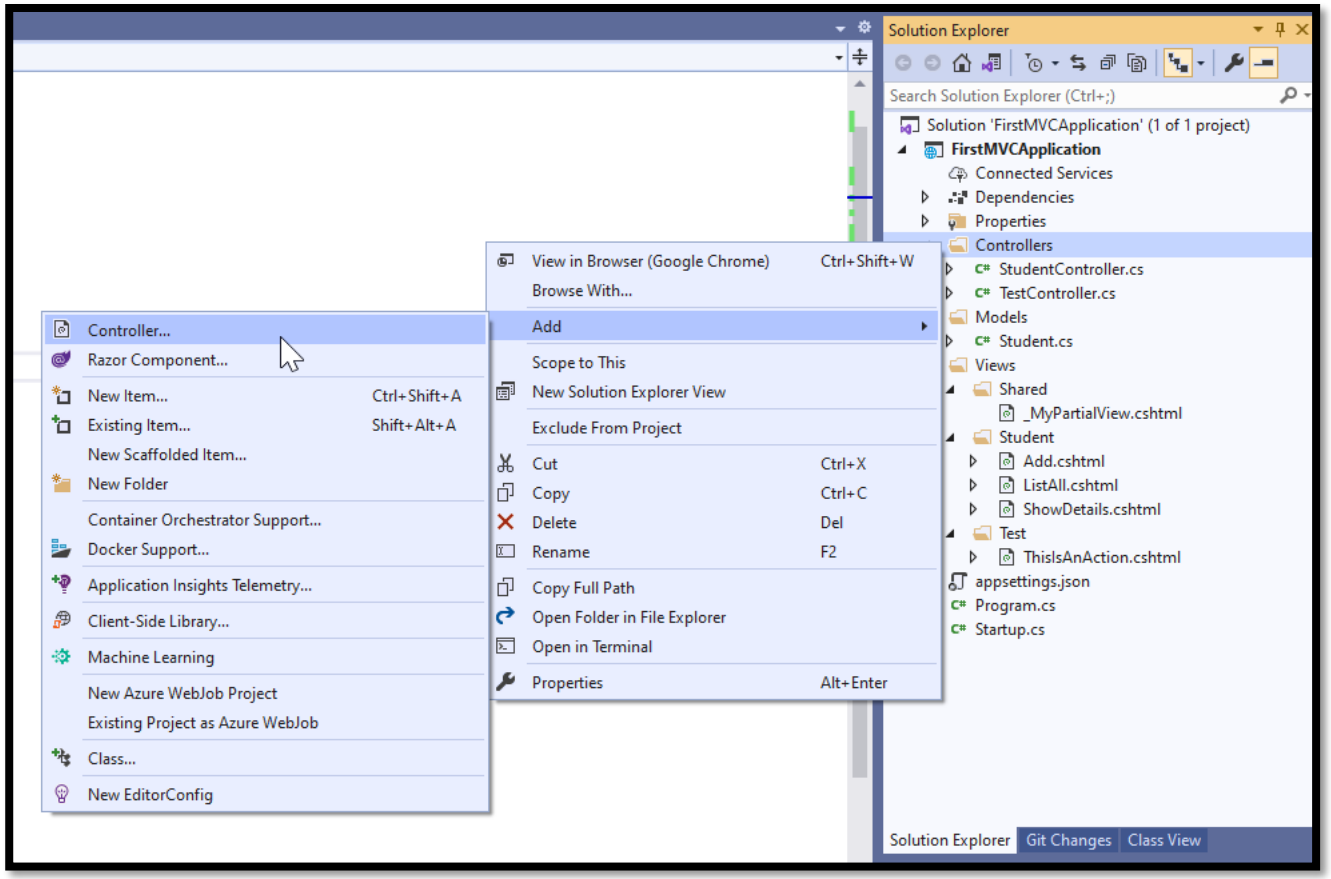
- Source: https://www.postman.com/downloads/

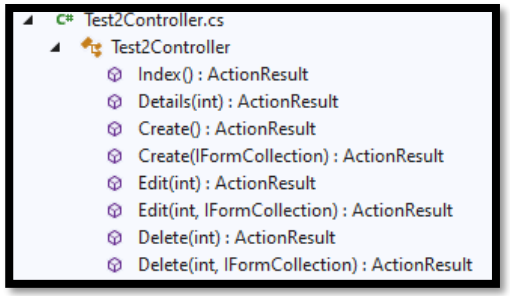


CRUD OPERATIONS

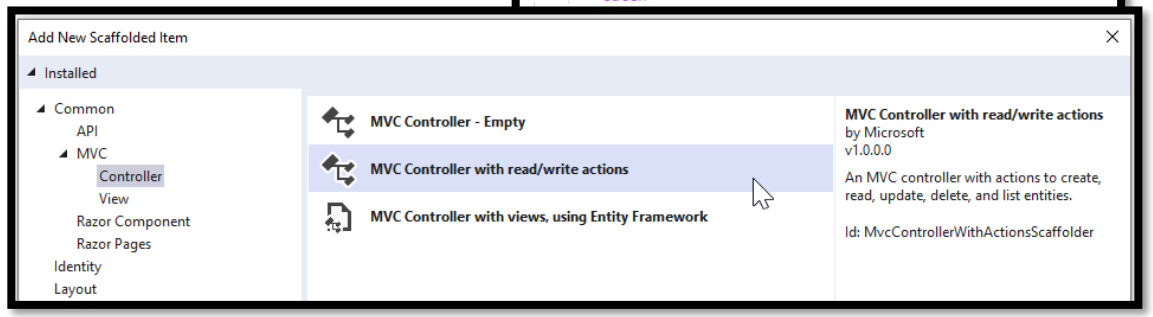
A **controller** often has CRUD operations:

- **Create operations**: Used to **create/add** new items
- **Read operations**: Used to **retrieve/search/view** existing entries
- **Update operations**: Used to **update/edit** existing entries
- **Delete operations**: Used to **delete** existing entries






```
public class Test2Controller : Controller
{
    // GET: Test2Controller
    public ActionResult Index()
    {
        return View();
    }

    // GET: Test2Controller/Details/5
    public ActionResult Details(int id)
    {
        return View();
    }

    // GET: Test2Controller/Create
    public ActionResult Create()
    {
        return View();
    }

    // POST: Test2Controller/Create
    [HttpPost]
    [ValidateAntiForgeryToken]
    public ActionResult Create(IFormCollection collection)
    {
        try
        {
            return RedirectToAction(nameof(Index));
        }
        catch
        {
            return View();
        }
    }

    // GET: Test2Controller/Edit/5
    public ActionResult Edit(int id)
    {
        return View();
    }

    // POST: Test2Controller/Edit/5
    [HttpPost]
    [ValidateAntiForgeryToken]
    public ActionResult Edit(int id, IFormCollection collection)
    {
        try
        {
            return RedirectToAction(nameof(Index));
        }
        catch
```



EDIT VIEW

- Let's add an Edit action that can allow the user to change details of a student.

```
Test2Controller.cs
  Test2Controller
    Index() : ActionResult
    Details(int) : ActionResult
    Create() : ActionResult
    Create(IFormCollection) : ActionResult
    Edit(int) : ActionResult
    Edit(int, IFormCollection) : ActionResult
    Delete(int) : ActionResult
    Delete(int, IFormCollection) : ActionResult
```

- We'll need an Edit(int id) action that responds to get request

- It will return a view containing the data we want to allow edits
  - Let's suppose that only some fields should be editable. So we'll only display those.
  - Instead of **HtmlEditorFor** helper, we'll use
    - **@Html.LabelFor(model => model.property1)** OR **<label asp-for="Property1"></label>**
    - **@Html.EditorFor(model => model.property1)** **<input asp-for="Property1" />**
      - Must use @addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers for tag helper

- The user can interact with this "view" and be able to hit a **Save** button to save these changes.

- We'll put these into a **form**:
  - @using(Html.**BeginForm**("ActionName", "ControllerNameWithoutCtrl")) { … }
  - <**form asp-controller**="ControllerNameWithoutCtrl" **asp-action**="ActionName"> … </**form**>

EDIT VIEW – THE GET

- Test2Controller.cs
  - Test2Controller
    - Index() : ActionResult
    - Details(int) : ActionResult
    - Create() : ActionResult
    - Create(IFormCollection) : ActionResult
    - Edit(int) : ActionResult
    - Edit(int, IFormCollection) : ActionResult
    - Delete(int) : ActionResult
    - Delete(int, IFormCollection) : ActionResult

- We'll need an action that accepts a valid ID (via Get requests)

Startup.cs | Add.cshtml | StudentController.cs | Student.cs | Test2Controller.cs

FirstMVCApplication

```
52
53          [HttpGet]
54          public IActionResult Edit(int id)
55          {
56              if (id < 0 || id >= allStudents.Count)
57                  return NotFound();
58              else
59                  return View(allStudents[id]);
60          }
```

- And a view that will give the user the requested d
  - We won't use this!

Add Razor View

View name: Edit

Template: Edit

Model class: Student (FirstMVCApplication.Models)

Options:

- [ ] Create as a partial view
- [ ] Reference script libraries
- [ ] Use a layout page:

(Leave empty if it is set in a Razor _viewstart file)

Add | Cancel

Edit.cshtml | Startup.cs | Add.cshtml | StudentController.cs | Student.cs | Test2Controller.cs

```
</head>
<body>

<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>
            <div class="form-group">
                <label asp-for="FirstName" class="control-label"></label>
                <input asp-for="FirstName" class="form-control" />
                <span asp-validation-for="FirstName" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Major" class="control-label"></label>
                <input asp-for="Major" class="form-control" />
                <span asp-validation-for="Major" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="GPA" class="control-label"></label>
                <input asp-for="GPA" class="form-control" />
                <span asp-validation-for="GPA" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="AdmissionDate" class="control-label"></label>
                <input asp-for="AdmissionDate" class="form-control" />
                <span asp-validation-for="AdmissionDate" class="text-danger"></span>
            </div>
            <div class="form-group form-check">
                <label class="form-check-label">
                    <input class="form-check-input" asp-for="IsInternational" /> @Html.DisplayNameFor(model => model.IsInternational)
                </label>
            </div>
            <div class="form-group">
                <input type="submit" value="Save" class="btn btn-primary" />
            </div>
        </form>
    </div>
</div>

<div>
    <a asp-action="Index">Back to List</a>
</div>

</body>
</html>
```

EDIT VIEW – THE GET

- Test2Controller.cs
  - Test2Controller
    - Index() : ActionResult
    - Details(int) : ActionResult
    - Create() : ActionResult
    - Create(IFormCollection) : ActionResult
    - Edit(int) : ActionResult
    - Edit(int, IFormCollection) : ActionResult
    - Delete(int) : ActionResult
    - Delete(int, IFormCollection) : ActionResult

- We'll need an action that accepts a valid ID (via Get requests)

Startup.cs | Add.cshtml | StudentController.cs | Student.cs | Test2Controller.cs

FirstMVCApplication

```
52
53          [HttpGet]
54          public IActionResult Edit(int id)
55          {
56              if (id < 0 || id >= allStudents.Count)
57                  return NotFound();
58              else
59                  return View(allStudents[id]);
60          }
```

- And a view that will give the user the requested data and a form that allows edit

Edit.cshtml | StudentController.cs | Student.cs | Test2Controller.cs

```
@model FirstMVCApplication.Models.Student
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers

@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Edit</title>
</head>
<body>
    <h1>Editing the student with ID: @ViewBag.StudentId</h1>

    @using (Html.BeginForm("Edit", "Student"))@*submit changes will call the Edit (Post) action!!!!!!!!!!!!!!!!!!!!!*@
    {
        @Html.LabelFor(model => model.FirstName) @Html.EditorFor(model => model.FirstName) <br /><br /> @*HTML helper*@
        @Html.LabelFor(model => model.AdmissionDate) @Html.EditorFor(model => model.AdmissionDate) <br /> @*HTML helper*@
        <label asp-for="GPA"></label> <input asp-for="GPA" />   <br /><br />                                @*tag helper*@
        <label asp-for="IsInternational"></label> <input asp-for="IsInternational" /><br /><br /><br />   @*tag helper*@

        <input type="submit" value="Save Changes" />
    }
</body>
</html>
```

Edit

localhost:59237/Student/Edit/3

Editing the student with ID: 3

First Name Young

Admission Date 04/22/2021

GPA 3.37

Is International ☐

Save Changes

localhost:59237/Student/Edit/3

localhost:59237/Student/Edit/3

Your request did not match any configured routing

EDIT VIEW – THE POST

- Test2Controller.cs
  - Test2Controller
    - Index() : ActionResult
    - Details(int) : ActionResult
    - Create() : ActionResult
    - Create(IFormCollection) : ActionResult
    - Edit(int) : ActionResult
    - Edit(int, IFormCollection) : ActionResult
    - Delete(int) : ActionResult
    - Delete(int, IFormCollection) : ActionResult

- We need a POST action to actually save those changes.
  - The previous form had a button that, when clicked, sent a **Post request**. We'll create that action now.

StudentController.cs | Student.cs | Test2Controller.cs

```
[HttpPost]
public IActionResult Edit(int id, Student st)
{
    allStudents[id].FirstName = st.FirstName;
    allStudents[id].GPA= st.GPA;
    allStudents[id].IsInternational= st.IsInternational;
    allStudents[id].AdmissionDate= st.AdmissionDate;
    return Content("Student details updated");
}
```

- Let's test it. Warning … those changes don't persist! To test, use breakpoints … Later we'll use a DB

Edit — localhost:59237/Student/Edit/3

Editing the student with ID: 3

First Name: Young
Admission Date: 04/22/2021
GPA: 3.37
Is International ☐
Save Changes

Edit — localhost:59237/Student/Edit/3

Editing the student with ID: 3

First Name: Alex
Admission Date: 04/01/2021
GPA: 2.95
Is International ☑
Save Changes

localhost:59237/Student/Edit/3

Student details updated

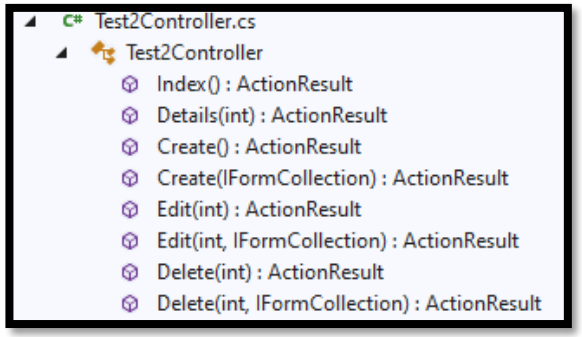
```
64  [HttpPost]
65  public IActionResult Edit(int id, Student st)
66  {
67      allStudents[id].FirstName = st.FirstName;
68      allStudents[id].GPA= st.GPA;
69      allStudents[id].IsInternational= st.IsInternational;
70      allStudents[id].AdmissionDate= st.AdmissionDate;
71      return Content($"Student details updated for {id}");
72  }
```

125 % — No issues found

Watch 1 — Search (Ctrl+E) — Search Depth: 3

Name	Value
allStudents[3]	{FirstMVCApplication.Models.Student}
AdmissionDate	{4/1/2021 12:00:00 AM}
FirstName	"Alex"
GPA	2.95
IsInternational	true
Major	"Ecology"

EDIT VIEW – THE POST



- If time, use:
  - **@Html.ActionLink**("List All Students", "ListAll")
  - **@Html.ActionLink**("List All Students", "ShowDetails", new { id = 1 })

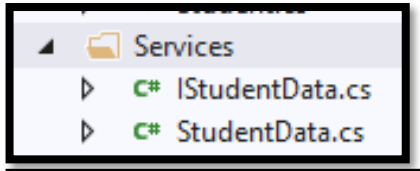
  - <button **asp-action**="DisplayAll"> No, do not delete this student</button>

LET'S CREATE A SERVICE

- We saw earlier that "those changes don't persist! To test, use breakpoints … Later we'll use a DB"
- Why?

- One "fix" is to use a **service.** (We'll see more **services** next time!)
  - This way we can create an instance that can be used for the duration of the application.
  - It is still not persistent (we'll need a database for that), but it works until the application gets restarted

- Create the class and interface
  - Add your initial data in here!




```
interface IStudentData
{
    List<Student> allStudents { get; set; }
}
```

```
public class StudentData : IStudentData
{
    public List<Student> roster { get; set; }

    public StudentData()
    {
        roster = new List<Student>();

        roster.Add(new Student() { FirstName = "Elon", LastName = "Musk", GPA = 4.1, GraduationDate = DateTime.Now, IsVeteran = false });
        roster.Add(new Student() { FirstName = "Stefon", LastName = "Diggs", GPA = 4.0, GraduationDate = DateTime.Now, IsVeteran = false });
        roster.Add(new Student() { FirstName = "Captain", LastName = "America", GPA = 3.9, GraduationDate = DateTime.Now, IsVeteran = true });
        roster.Add(new Student() { FirstName = "Thomas", LastName = "Edison", GPA = 0, GraduationDate = DateTime.Now, IsVeteran = false });
        roster.Add(new Student() { FirstName = "Nikola", LastName = "Tesla", GPA = 4.0, GraduationDate = DateTime.Now, IsVeteran = false });
    }
}
```

- Register it as a service

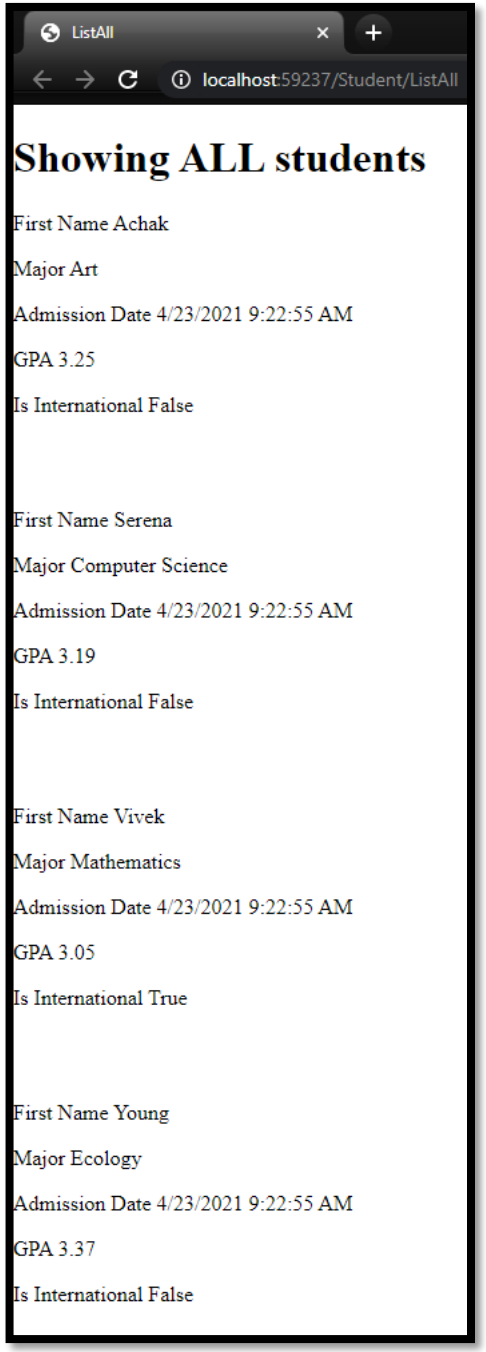
```
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc(x=>x.EnableEndpointRouting=false);
    services.AddSingleton<IStudentData, StudentData>();
}
```

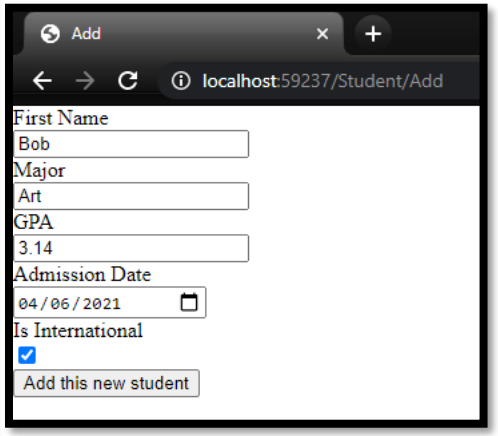
- Inject it where we need it.
  - here, injected in a **controller**
  - And use it

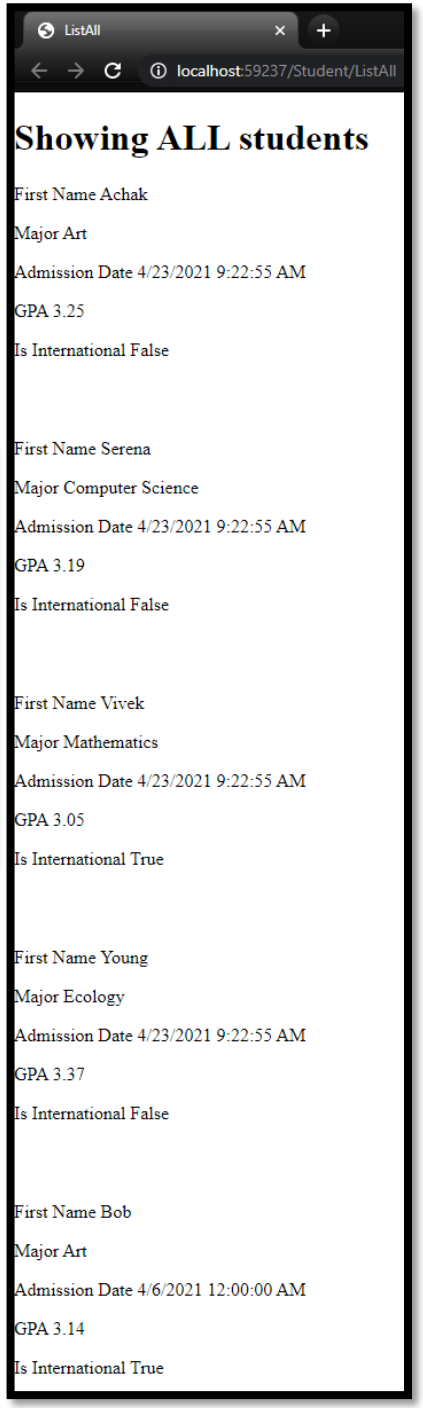
```
public class StudentController : Controller
{
    private IStudentData spring2021;

    public StudentController(IStudentData myService)
    {

        spring2021 = myService;
    }

    public IActionResult Display(int id)
    {
        if (id < 0 || id >= spring2021.roster.Count)
            return Content("invalid id!");
```

LET'S CREATE A SERVICE

- Test it
- http://localhost:59237/Student/ListAll
- http://localhost:59237/Student/Add
- Note: a better place to add testing data is in the constructor of the Service, not inside the constructor of the Controller





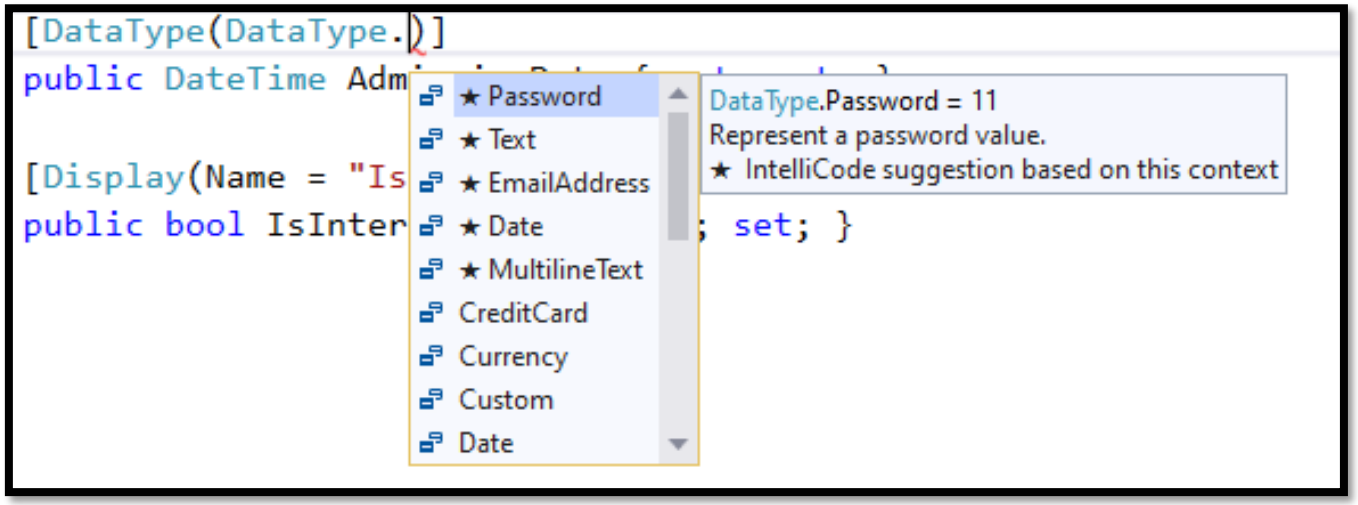


DELETE EXISTING STUDENT – IF TIME

- If time, let's implement this as well

- We saw the following data annotations earlier:
  - [Display(Name = "Admission Date")]
  - [DataType(DataType.Date)]

- Other useful examples:
  - [DataType(DataType.Password)]
  - [DataType(DataType.Multiline)]
  - [DataType(DataType.EmailAddress)]
  - …



SOME SERVER-SIDE VALIDATION

- Validation attributes allows you to specify **validation rules** for properties of **model** classes.

- Examples of built-in **validation attributes** (see page 5689 for more examples):
  - [EmailAddress] checks that the property has a **valid email format**.
  - [Phone] check that the property has a **valid telephone number format**.
  - [Range] checks that the property value is within a **specified range**.
  - [RegularExpression] checks that the property value **matches a given regular expression**.
  - [Required] checks that the **field is not null**.
  - [StringLength] checks that the **field does not exceed a given max length**. It also accepts a **MinimumLength** value
  - [Url] : Validates that the property has a URL format.

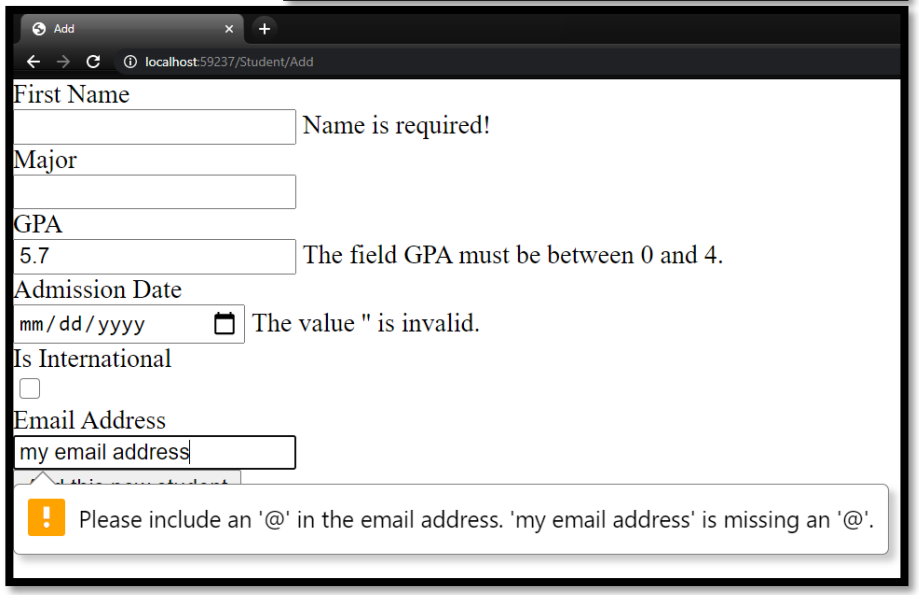
- Use **ErrorMessage** property in order too specify an error message for the user (that's different than the default error message).

- Inside the controller, use the **ModelState.IsValid** to check for the validity of data submitted by user.
  - If invalid, return back the current view …

```
public class Student
{
    [Display(Name ="First Name")]
    [Required(ErrorMessage ="Name is required!")]
    public string FirstName { get; set; }

    public string Major { get; set; }

    [Range(0,4)]
    public double GPA { get; set; }

    [Display(Name = "Admission Date")]
    [DataType(DataType.Date)]
    public DateTime AdmissionDate { get; set; }

    [Display(Name = "Is International")]
    public bool IsInternational { get; set; }

    [EmailAddress]
    [Display(Name = "Email Address")]
    public string EmailAddress { get; set; }
}
```

- See pages 5688-5715

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

[HttpPost]
public IActionResult Add(Student st)
{
    if(!ModelState.IsValid)//return back if invalid entry
    {
        return View();
    }
    _myStudentData.allStudents.Add(st);
    return Content($"a new student with the name {st.FirstName} was added to the list");
}
```



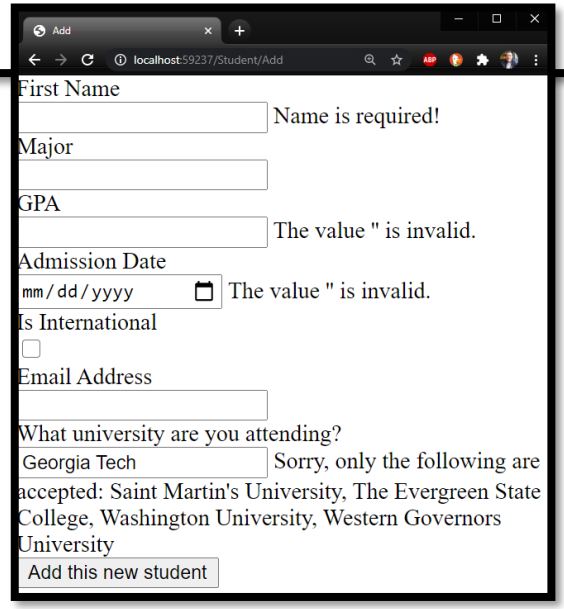
SOME SERVER-SIDE VALIDATION

- One can also create **custom validation attributes**!
- Some reasons to create and use **custom validation attributes**:
  - check the data entered against the data that is stored in a database.
  - any other scenarios that the built-in validation attributes don't handle
- Create **custom validation** attribute:
  - create a folder for it, let's call it **Validators**
  - Example: MyFirstCustomValidationAttribute
- Use **custom validation** data annotations

  [MyFirstCustomValidation]

  public string University { get; set; }
- See pages 5688-5715

```
using FirstMVCApplication.Models;
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;

namespace FirstMVCApplication.Validators
{
    public class MyFirstCustomValidationAttribute:ValidationAttribute
    {
        protected override ValidationResult IsValid(object value, ValidationContext validationContext)
        {
            List<string> AcceptedUniversities = new List<string>();
            AcceptedUniversities.Add("Saint Martin's University");
            AcceptedUniversities.Add("The Evergreen State College");
            AcceptedUniversities.Add("Washington University");
            AcceptedUniversities.Add("Western Governors University");

            Student student = (Student)validationContext.ObjectInstance;
            if (student.University!=null &&
                AcceptedUniversities.Contains(student.University.ToLower(), StringComparer.OrdinalIgnoreCase))
                return ValidationResult.Success;
            else
                return new ValidationResult("Sorry, only the following are accepted: "+
                    String.Join(", ", AcceptedUniversities.ToArray()));
        }
    }
}
```




```
public class Student
{
    [Display(Name ="First Name")]
    [Required(ErrorMessage ="Name is required!")]
    public string FirstName { get; set; }

    public string Major { get; set; }

    [Range(0,4)]
    public double GPA { get; set; }

    [Display(Name = "Admission Date")]
    [DataType(DataType.Date)]
    public DateTime AdmissionDate { get; set; }

    [Display(Name = "Is International")]
    public bool IsInternational { get; set; }

    [EmailAddress]
    [Display(Name = "Email Address")]
    public string EmailAddress { get; set; }


    [MyFirstCustomValidation]
    [Display(Name = "What university are you attending?")]
    public string University { get; set; }
}
```

VALIDATION TAG/HTML HELPERS

- There are two types of **Validation Tag Helpers**.
  - Validation **Summary Tag Helper** ← displays a summary of **all validation errors** in a single place
    - **@Html.ValidationSummary()** OR ** **

Add.cshtml | MyFirstCustomValidationAttribute.cs | Student.cs | StudentController.cs

```
@model FirstMVCApplication.Models.Student
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers

@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Add</title>
</head>
<body>
    @using (Html.BeginForm())
    {
        <div asp-validation-summary="All"></div>

        @Html.EditorForModel()
        <input type="submit" value="Add this new student" />
    }


</body>
</html>
```

Add.cshtml | MyFirstCustomValidationAttribute.cs | Student.cs | StudentController.cs

```
@model FirstMVCApplication.Models.Student

@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Add</title>
</head>
<body>
    @using (Html.BeginForm())
    {
        @Html.ValidationSummary()

        @Html.EditorForModel()
        <input type="submit" value="Add this new student" />
    }


</body>
</html>
```

Add — localhost:59237/Student/Add

- Name is required!
- The value '' is invalid.
- The value '' is invalid.
- Sorry, only the following are accepted: Saint Martin's University, The Evergreen State College, Washington University, Western Governors University

First Name [ ] Name is required!
Major [ ]
GPA [ ] The value '' is invalid.
Admission Date [mm/dd/yyyy] The value '' is invalid.
Is International [ ]
Email Address [ ]
What university are you attending? [Georgia Tech] Sorry, only the following are accepted: Saint Martin's University, The Evergreen State College, Washington University, Western Governors University
[Add this new student]

- Page: 2857+

VALIDATION TAG/HTML HELPERS

- There are two types of **Validation Tag Helpers**.
  - Validation **Message Tag Helper** ← displays a validation message for a <u>single</u> **property** on a model
    - @Html.ValidationMessageFor(model => model.PropName) OR
    - <span **asp-validation-for**="PropName">

```csharp
public class Student
{
    [Display(Name ="First Name")]
    [Required(ErrorMessage ="Name is required!")]
    public string FirstName { get; set; }

    public string Major { get; set; }

    [Range(0,4)]
    public double GPA { get; set; }

    [Display(Name = "Admission Date")]
    [DataType(DataType.Date)]
    public DateTime AdmissionDate { get; set; }

    [Display(Name = "Is International")]
    public bool IsInternational { get; set; }

    [EmailAddress]
    [Display(Name = "Email Address")]
    public string EmailAddress { get; set; }


    [MyFirstCustomValidation]
    [Display(Name = "What university are you attending?")]
    public string University { get; set; }
}
```

```csharp
[HttpGet]
public IActionResult Edit(int id)
{
    if (id < 0 || id >= _myStudentData.allStudents.Count)
        return NotFound();
    else
        ViewBag.StudentId = id;
        return View(_myStudentData.allStudents[id]);
}


[HttpPost]
public IActionResult Edit(int id, Student st)
{
    if (!ModelState.IsValid)//return back if invalid entry
    {
        return View();
    }

    _myStudentData.allStudents[id].FirstName = st.FirstName;
    _myStudentData.allStudents[id].GPA= st.GPA;
    _myStudentData.allStudents[id].IsInternational= st.IsInternational;
    _myStudentData.allStudents[id].AdmissionDate= st.AdmissionDate;
    return Content($"Student details updated for {id}");
}
```

Add.cshtml | Edit.cshtml | MyFirstCustomValidationAttribute.cs | Student.cs | StudentCon

```razor
@model FirstMVCApplication.Models.Student
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers

@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Edit</title>
</head>
<body>
    <h1>Editing the student with ID: @ViewBag.StudentId</h1>

    @using (Html.BeginForm("Edit", "Student"))
    {
        @Html.LabelFor(model => model.FirstName)
        @Html.EditorFor(model => model.FirstName)
        @Html.ValidationMessageFor(model => model.FirstName)<br /><br />

        @Html.LabelFor(model => model.AdmissionDate)
        @Html.EditorFor(model => model.AdmissionDate)
        @Html.ValidationMessageFor(model => model.AdmissionDate)<br />

        <label asp-for="GPA"></label>
        <input asp-for="GPA" />
        <span asp-validation-for="GPA"></span><br /><br />

        <label asp-for="IsInternational"></label>
        <input asp-for="IsInternational" />
        <span asp-validation-for="IsInternational"></span> <br /><br /><br />

        <input type="submit" value="Save Changes" />
    }
    </body>
</html>
```

Edit

localhost:59237/Student/Edit/2

Editing the student with ID:

First Name [ ] Name is required!

Admission Date 04/24/2021

GPA [ ] The value '' is invalid.

Is International ☑

Save Changes

- Page: 2857+

IN-CLASS DEMO – IF TIME

Demonstration: How to Bind Views to Model Classes

- Source/Steps
- https://github.com/MicrosoftLearning/20486D-DevelopingASPNETMVCWebApplications/blob/master/Instructions/20486D_MOD06_DEMO.md#demonstration-how-to-bind-views-and-model-classes

IN-CLASS DEMO – IF TIME

Demonstration: How to Use Display and Edit Data Annotations

- Source/Steps
- https://github.com/MicrosoftLearning/20486D-DevelopingASPNETMVCWebApplications/blob/master/Instructions/20486D_MOD06_DEMO.md#demonstration-how-to-use-display-and-edit-data-annotations

IN-CLASS DEMO – IF TIME

Demonstration: How to Validate User Input with Data Annotations

- Source/Steps
- https://github.com/MicrosoftLearning/20486D-DevelopingASPNETMVCWebApplications/blob/master/Instructions/20486D_MOD06_DEMO.md#demonstration-how-to-validate-user-input-with-data-annotations

IN-CLASS DEMO – IF TIME

Demonstration: How to Add Custom Validations

- Source/Steps
- https://github.com/MicrosoftLearning/20486D-DevelopingASPNETMVCWebApplications/blob/master/Instructions/20486D_MOD06_DEMO.md#demonstration-how-to-add-custom-validations

LAB/HOMEWORK: EXPLORING ASP.NET CORE MVC

- **Module 01**
  - Exercise 1: Adding a Model
  - Exercise 2: Working with Forms
  - Exercise 3: Adding Validation

- You will find the **high-level** steps on the following page:
  https://github.com/MicrosoftLearning/20486D-DevelopingASPNETMVCWebApplications/blob/master/Instructions/20486D_MOD06_LAB_MANUAL.md

- You will find the **detailed** steps on the following page:
  https://github.com/MicrosoftLearning/20486D-DevelopingASPNETMVCWebApplications/blob/master/Instructions/20486D_MOD06_LAK.md

- For your homework submit one zipped folder with your complete solution.