ASP.NET Web App Remote Methods - This version for course "Cloud Computing for Software Developers"

MVC is all about the "C" - the M and the V are optional extras.

Client Webserver M

**C** EF DB

Render

V

"C" is for "Controller"

It is the Centre of MVC operation.

Clients do NOT visit pages, they visit Methods in controllers  
In the following example, the base website is "https://cars.example.com/"

|  |  |
| --- | --- |
| URL Address | Where it really goes |
| https://cars.example.com/  *acts the same as* https://cars.example.com/home  https://cars.example.com/home/index  BUT NOT  https://cars.example.com/index "index" is now a method that needs to be the child of a controller | method "Index" in class "HomeController" |
| https://cars.example.com/modifications  *acts the same as*  https://cars.example.com/modifications/index | method "Index" in class "ModificationsController" |
| https://cars.example.com/modifications/edit | method "Edit" in class "ModificationsController" |
| https://cars.example.com/paintjobs/racing | method "Racing" in class "PaintjobsController" |

A "web page" is the text output from a Method in a Controller

A "View" is a helper template of reusable text to save typing in our C# code

MVC gives you a lot of complex relationships where the Microsoft MVC team have already done a lot of the programming work and we make use of some of that work by following naming conventions and matching names.

For example:

https://cars.example.com/paintjobs/hotrod

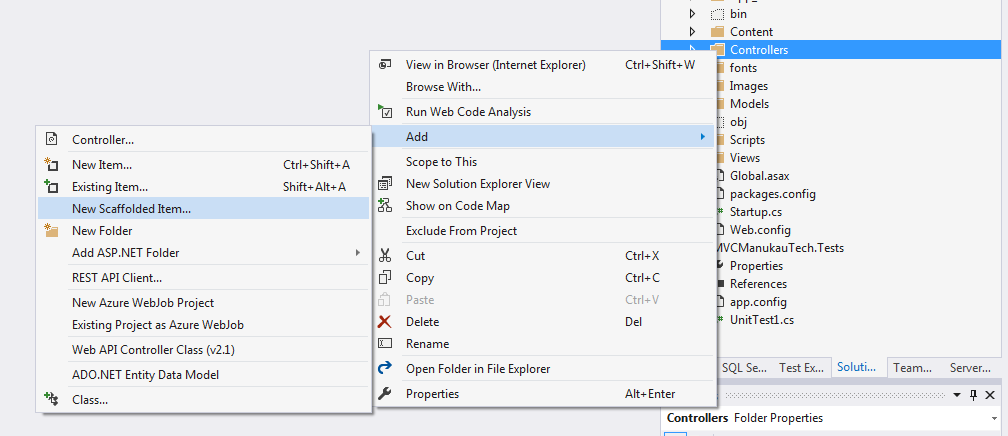
The real address is method "Hotrod" in class "PaintjobsController"

The View is "Hotrod.cshtml" in project sub-folder Views\Paintjobs  
This Controller and this View connect to each other and relate   
by matching the keywords "Paintjobs" and "Hotrod".

**Starting with C for Controller**

Start a new ASP.NET Core Web Application - Select the blank website option.

In the "Solution Explorer" panel, usually on the right, right-click on the folder "Controllers"



Select "Add" --> "New Scaffolded Item"  
The screenshot above shows going direct to "Controller" as an option, but we only get this when the system remembers we have done this before.  
Then from the Popup Dialog, select **MVC Empty Controller**.  
On the prompt for a name, enter **CalcController**

We will start by adding and testing

public string Add(double a, double b)

{

double c = a + b;

return c.ToString();

}

to test this, run the app and edit the address adding: /Calc/Add?a=4&b=7

http://localhost:44384/Calc/Add?a=4&b=7

we should see ... 11

--------------------------

**Exercise - Temperature converter.**  
Add a new Method to Controller Calc.  
Name this Controller Method: TempConvert.

The first line of code to create this method will look like:

public string TempConvert(double f)

Input parameter f - is a value of temperature in degrees fahrenheit

Output value c - converted to degrees Celcius.  
Conversion formula is:  
c = (f - 32) \* 100 / 180

--------------------------

**Exercise - Cost Estimate**

We can estimate the cost of building a house in Tribnia based on its area in square metres.

But we also need to include a cost for building on a hill. The steeper the hill, the greater the house cost.

Use the formula:

estimated\_cost = (area \* 2300) + (hill\_angle \* 2000)

But wait, there's more!

The Tribnia Council has a rule that it is unsafe to build on a hill with an angle of more than 40 degrees.  
We therefore need to build into this algorithm:  
if the hill angle is greater than 40 degrees then we need to indicate to the caller (also known as consumer) that there is a problem.

We can try adding a little HTML like this:

string response = "<b>ERROR: </b> This hill is too steep to build on.";

Try this in ASP.NET CORE 2.0 and HTML rendering does not happen.  
We see the raw tags on the browser screen.

To get this working we need to bring in methods and objects that send the protocol "headers" to the browser.  
These are like instructions to the browser on the actions to take with this data.

public IActionResult Quad(double a, double b, double c)

{

Response.ContentType = "text/html";

//coding of the supplied formula goes here

string response =   
 "<b>ERROR: </b> This hill is too steep to build on.";

return Content(response);

}

--------------------------

Your exercises - with a little help including on whiteboard in class. Do responses including info like the previous example.

**JSON quick intro (lecturer-led)**

JSON gives more detail and we consider it to be more modern than CSV.

Begin with the introduction from W3Schools. Read the first 4 pages:  
<https://www.w3schools.com/js/js_json_intro.asp>

With JSON we use {} for objects - often these are 1 row of data from a database table.  
We use square brackets for arrays.  
The above "Mean" example done again as "JSONMean" with JSON has this test input

http://localhost:7151/Calc/JSONMean?json=[8,3,4,7,20]

To work with JSON in C#  
At the top of your controller add this statement to the other "using" statements:  
using Newtonsoft.Json;

Ref is

<http://www.newtonsoft.com/json>

<http://www.newtonsoft.com/json/help/html/SerializingJSON.htm>

**Useful methods are**

To convert from JSON string to an array, list, collection or other object  
JsonConvert.DeserializeObject<T>()

eg

Product deserializedProduct = JsonConvert.DeserializeObject<Product>(JSONString);

where T is the type of object you want to convert to. You will need this for your "Mean" exercise.

The opposite is "SerializeObject" for converting from your object, eg database table data, to JSON to send across the internet to another business

by example of "product" as the object

string output = JsonConvert.SerializeObject(product);

**We can also work with JSON in JavaScript - with different method names.**

From string to object

var myJSON = '{ "name":"John", "age":31, "city":"New York" }';  
var myObj = JSON.parse(myJSON);

From object to string

var myObj = { "name":"John", "age":31, "city":"New York" };  
var myJSON = JSON.stringify(myObj);

**Summary of JSON methods**

|  |  |  |
| --- | --- | --- |
| **Action** | **C#** | **JavaScript** |
| string to object | JsonConvert.DeserializeObject<T>() | JSON.parse() |
| object to string | JsonConvert.SerializeObject(); | JSON.stringify(); |