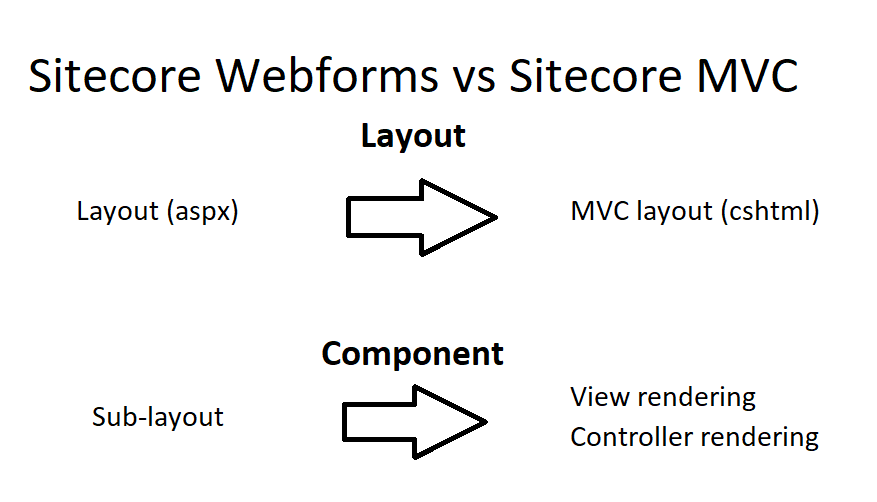
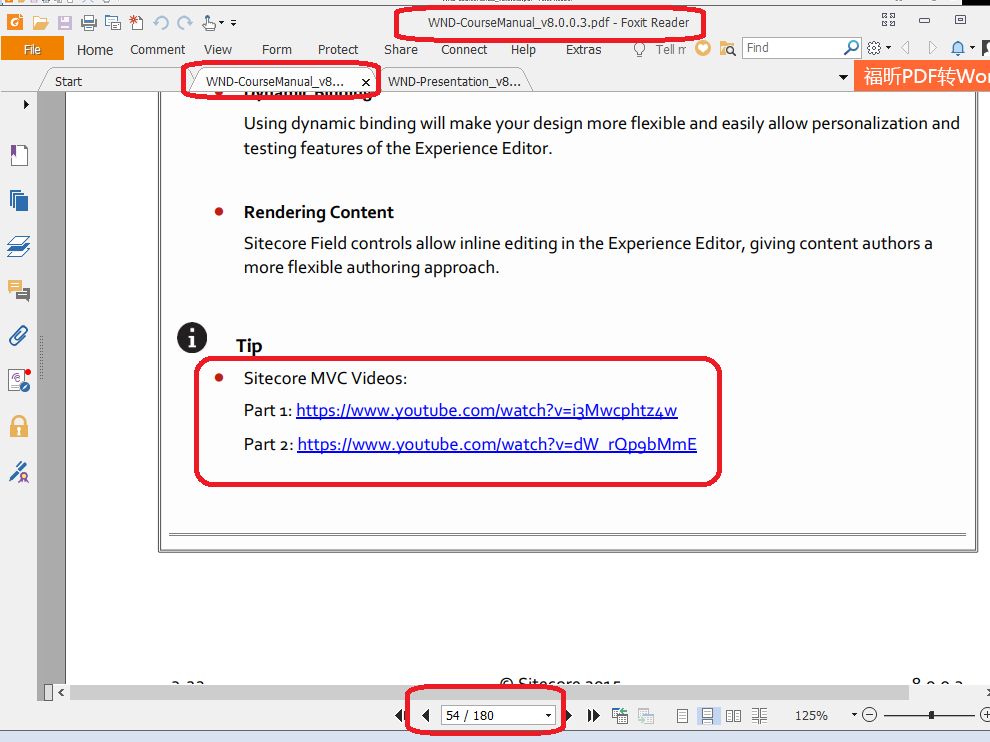
**What is the differences?**



Instead of using aspx layout (and Sub-layout component) of Sitecore Webforms, Sitecore MVC use MVC layout (.cshtml extension) with View rendering and Controller rendering for render the page. We can’t use Sub-layout in Sitecore MVC.

Before we start, watch 2 below video (just watch, don’t follow):

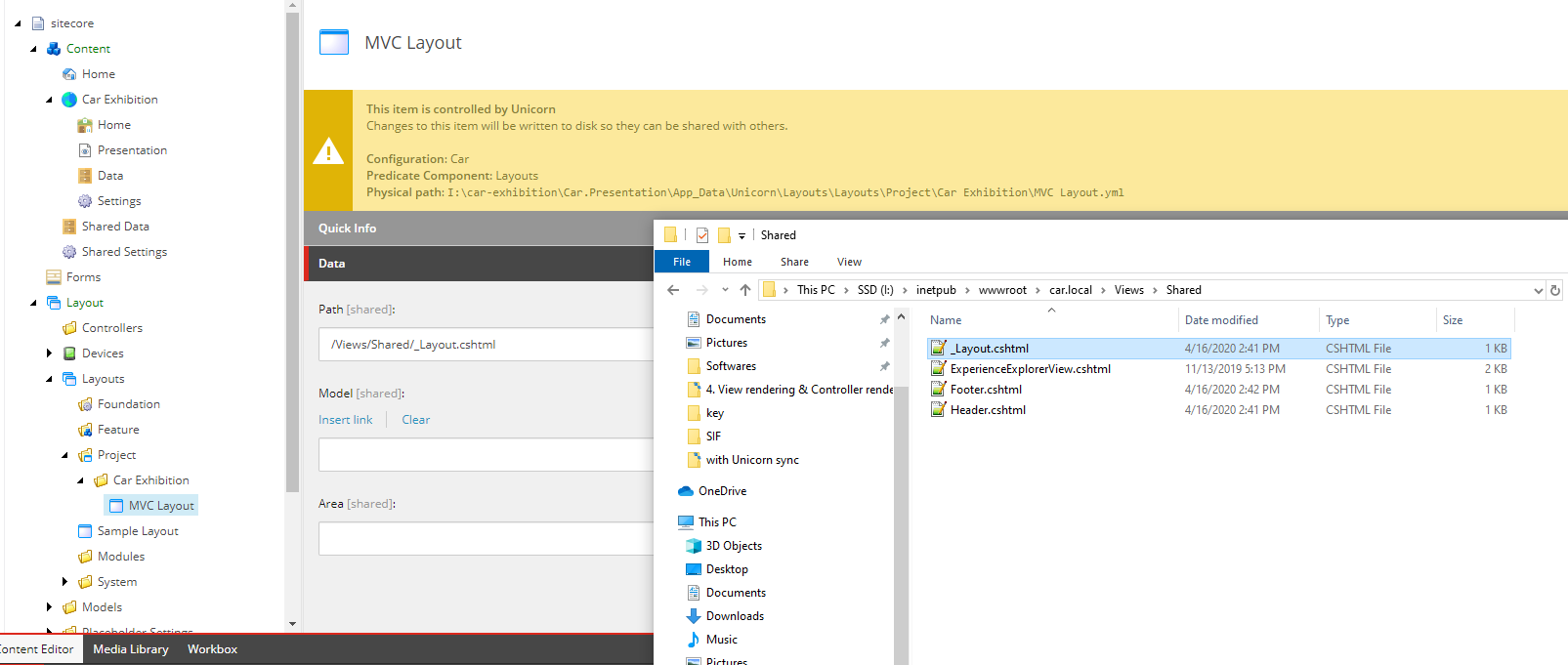


**----------------------------------------------------------**

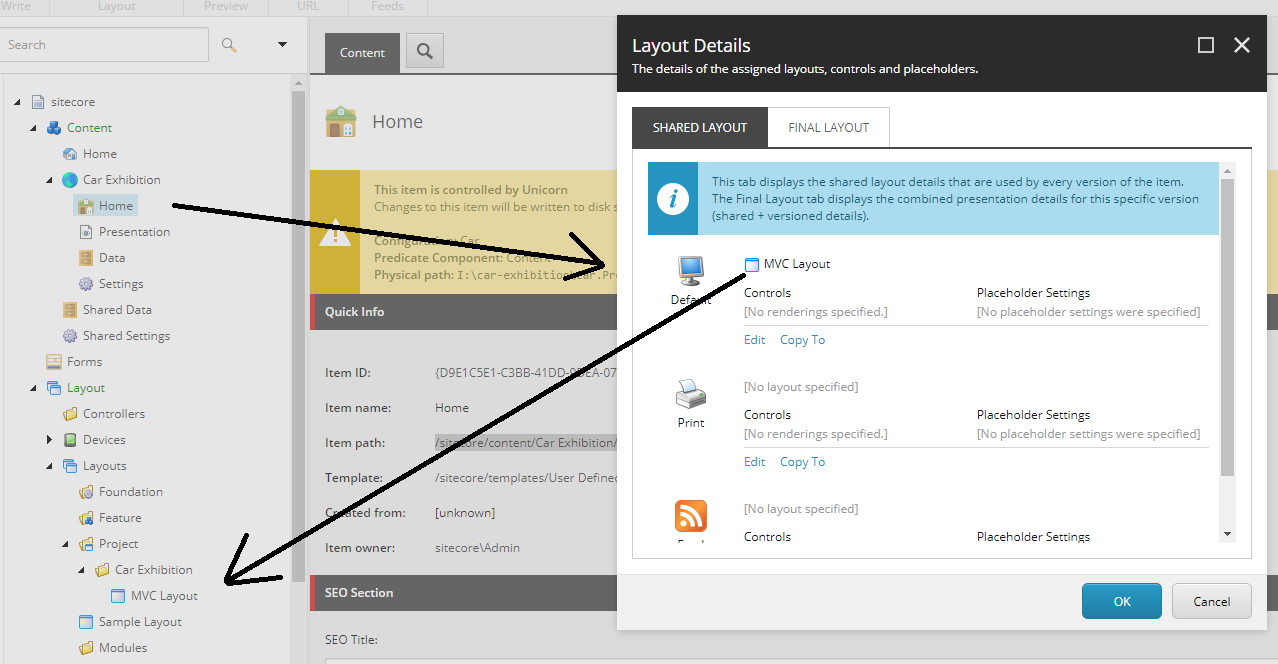
**Let’s start now**

**Dynamic binding & Static binding**

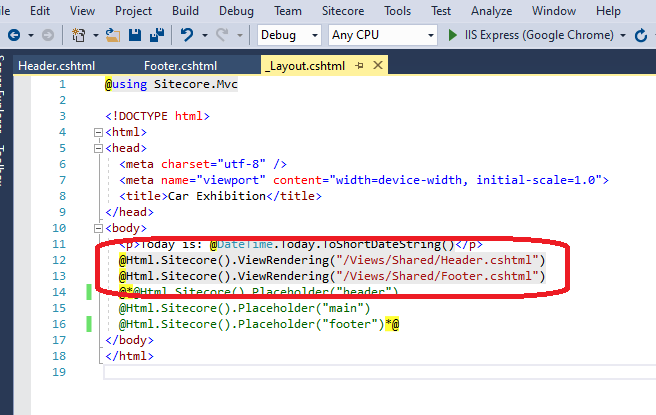
* Go to CMS, publish the site and refresh the page
* Create a new MVC Layout point to the layout page in your webroot



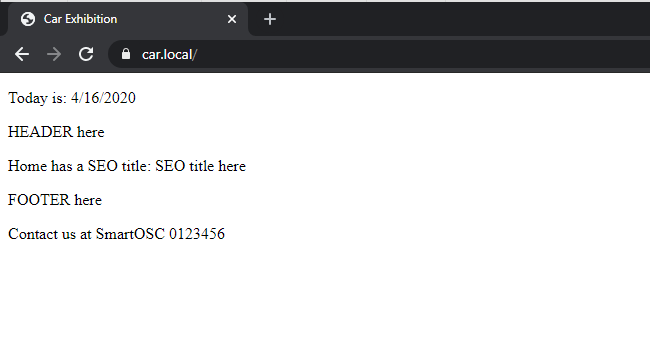
Set that layout to the Home item at /sitecore/content/Car Exhibition/Home



* Take a look at the Header.cshtml, Footer.cshtml.
* We can render them in the layout page by call these helpers (prerequisite: @using Sitecore.Mvc)



Go to <your\_site>/. It will look like the picture below:



It’s called ***static binding***. We can use this method to add the component that we need to have in the whole page (Ex: header, footer …) to reduce the working time.

In the other hand, we can render that component by dynamic binding method: add them to a placeholder.

(same with adding component to a page in Webforms)

**View rendering and Controller rendering**

“view rendering and controller rendering: they are called component”

Follow my clip to create some content for your page:

Watch the video: [Car - Content Create.mp4](3.%20Car%20-%20Content%20Create.mp4)

**View rendering**

What is a view rendering?

A view rendering consists of a view that takes a RenderingModel by default. The model is assembled by the Sitecore MVC pipeline, and you can create your own, custom models. Because the pipeline expects an Initialize() method on your model, you can either inherit from RenderingModel or implement IRenderingModel.

Pros

* Simple to use – especially if you do not use a custom model.

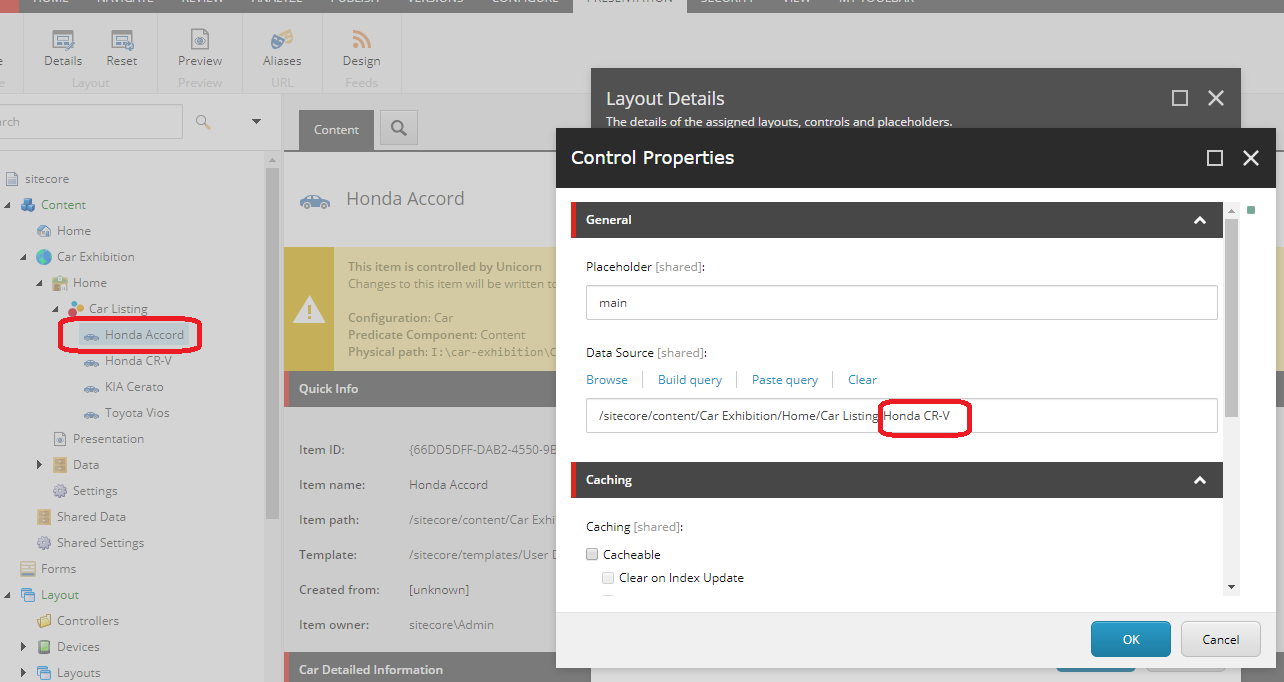
Most of the time, you only want to display some content from the context or data source item without requiring any business logic – a simple ‘page’ will often consist of a title and some text, which you can do very easily with Sitecore’s RenderingModel and the @Html.Sitecore() helper

**Ex: Create View rendering to render the content:**

Solution 1: the component load data from a specific item

Watch the video: [View rendering.mp4](4.%20View%20rendering.mp4)

* My Honda Accord page can display data from Honda CR-V. You can see the picture below for more information

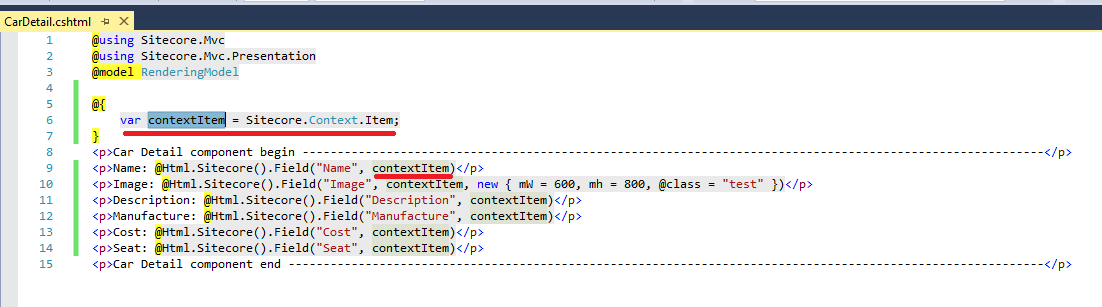


* By this way, a page can load data from other page to display.

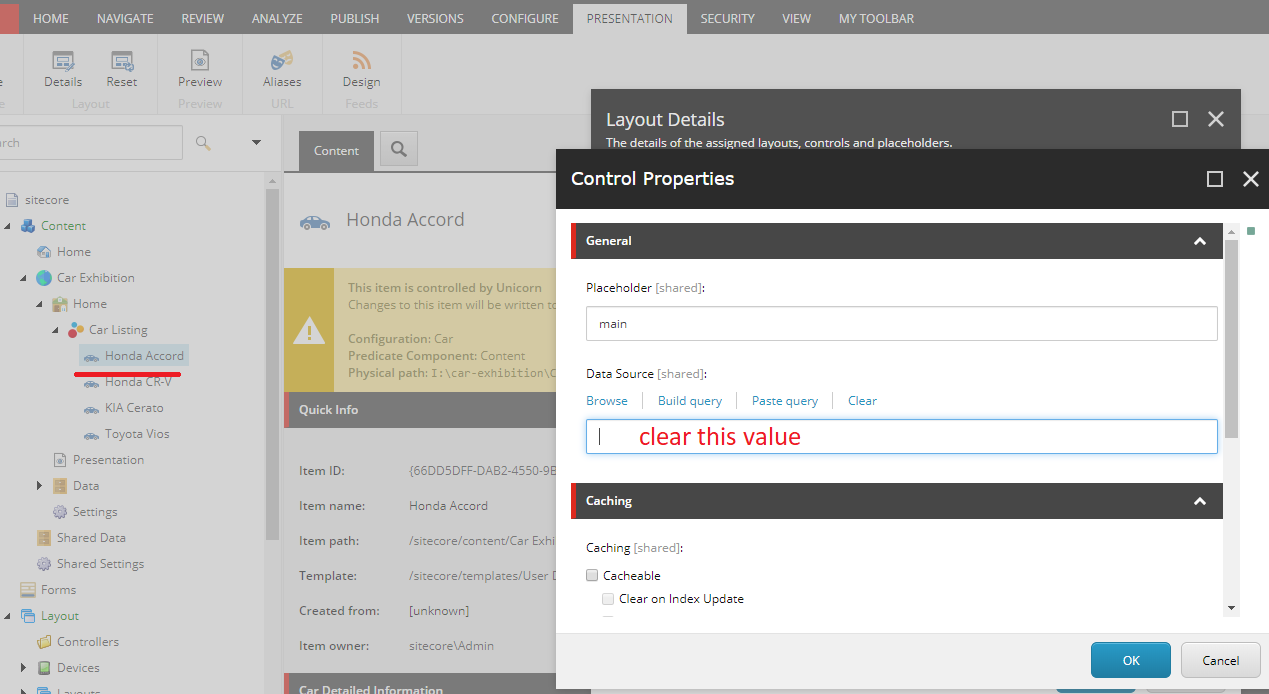
Okay. So how to display current item’s data for this component => let’s move to Solution 2

Solution 2: the component load data from context item by default

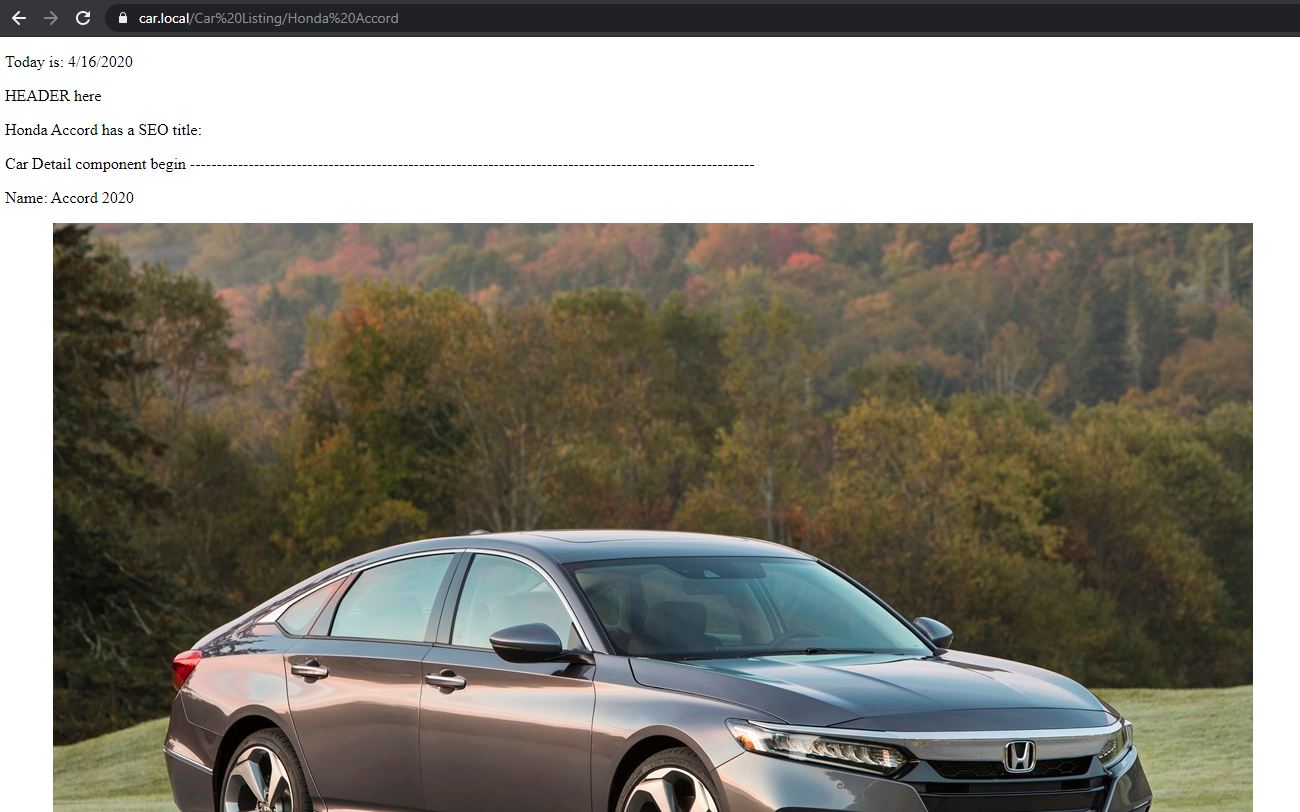
Step 1: edit the “CarDetail.cshtml”



Step 2: we do not need the Data Source value anymore, leave it empty now because we already load data from the current item



Let’s see the result. They are the same:



***Note***: when you create view rendering component =>

**Datasource location**: where to component start searching for datasource item

**Datasource template**: the type of template that component searching for

**Controller rendering**

What is a controller rendering?

Rather than specifying a view, the component definition item for a controller rendering specifies a controller name and an action name. When the rendering is added to a placeholder, this action is executed and the view is returned.

Pros

* You can use dependency injection and common
* Create your own actions.

Keeps your business logic separate from your model. It is the most important thing. Let the View actually be a view, do not bring business logic into view.

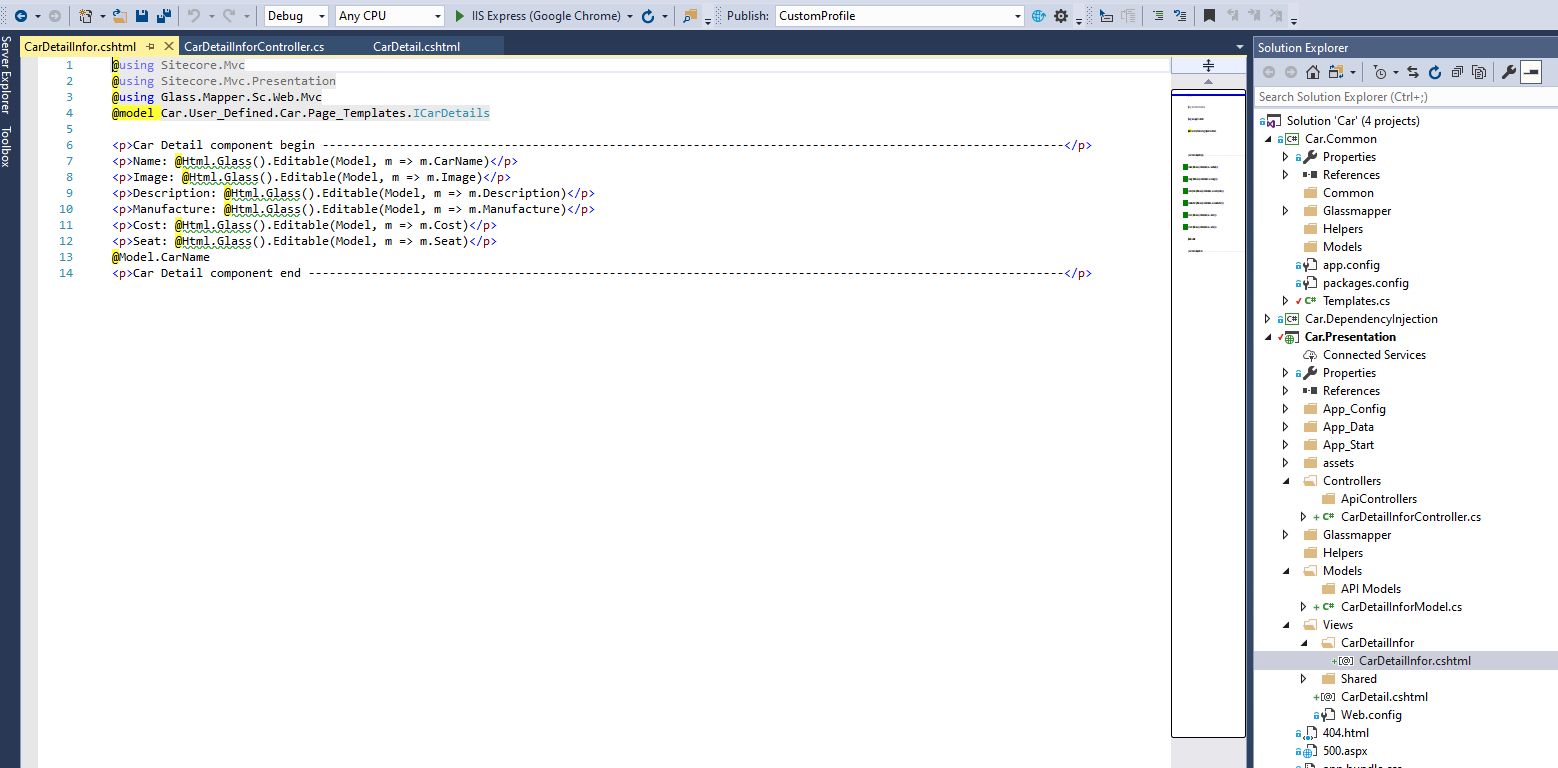
So, how to use it?

Step 1: create the model

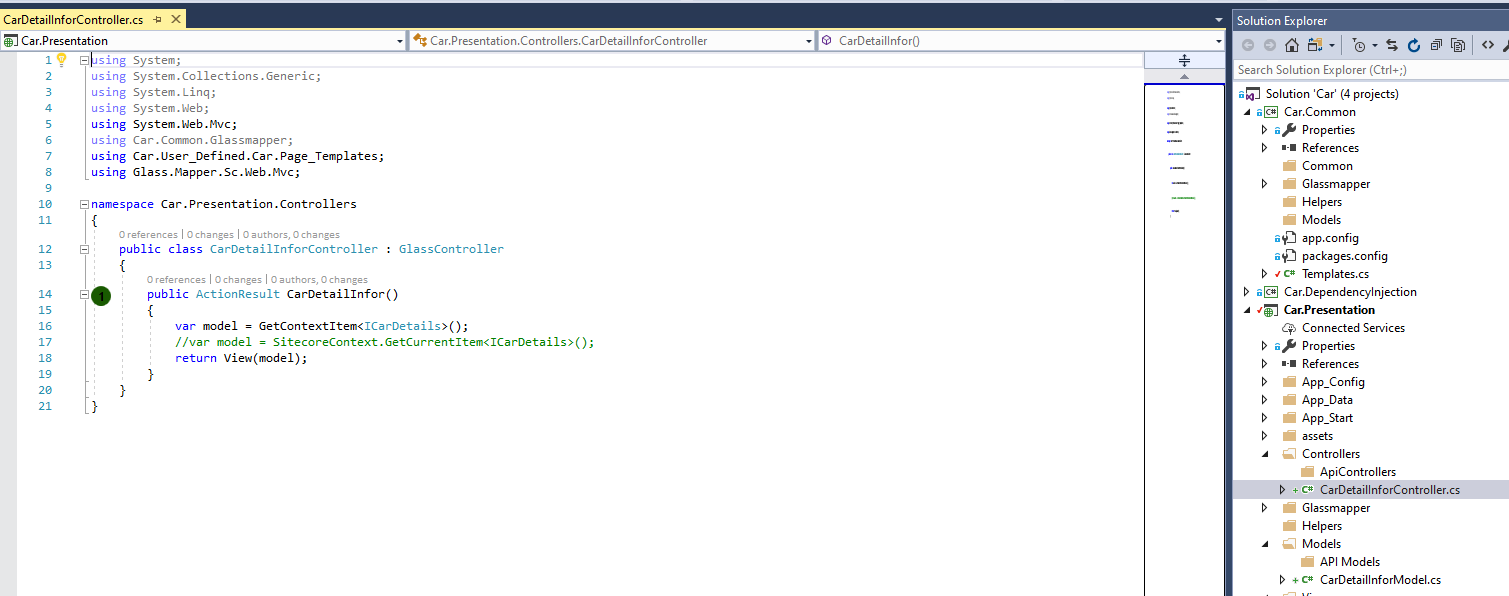
Model mapping with Glassmapper:

Watch the video: [Model mapping with Glassmapper.mp4](5.%20Model%20mapping%20with%20Glassmapper.mp4)

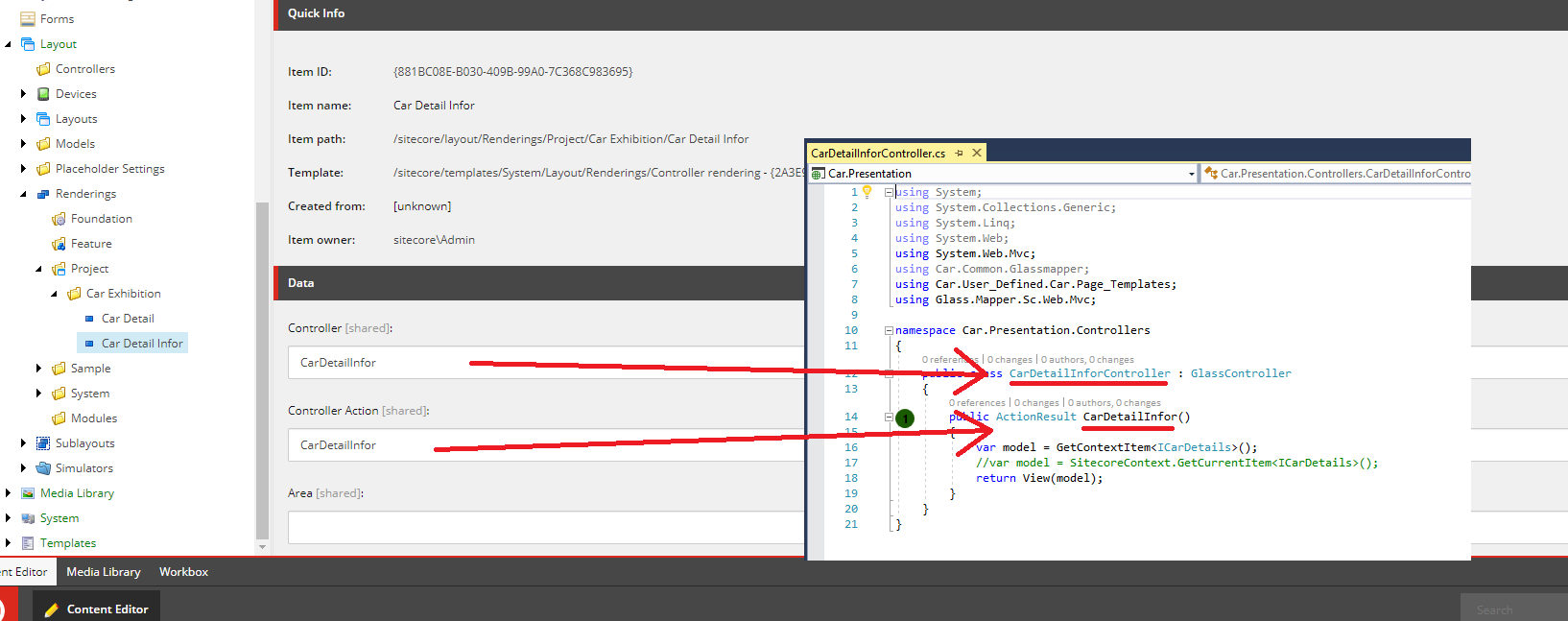
Step 2: create a view named “CarDetailInfor.cshtml” in the Views/CarDetailInfor



Step 3: create a controller for that view named “CarDetailInforController.cs” at Controllers folder



Step 3: create a controller rendering component to use these above things



Add this component into a page to see how does it work.

Note: about the model ICarDetails, instead of mapping by field name, we can do it by field Id.

***With mapping by field name:***

Pros: the code is more meaning and easy to read, we can see this one (in Templates.cs) mapped to another one in the Content tree’s Templates folder.

***With mapping by field Id:***

Pros: the user can change the field name and our logic still work well. So we don’t care about the name.

See the below picture to more detailed information:

