Summer of Tech

.Net Bootcamp Lab Instructions

Building an MVC5 Web Application

Trade Me

8-18-2016

Adapted from Wellington.Net Bootcamp by Kiwibank 8/4/2011

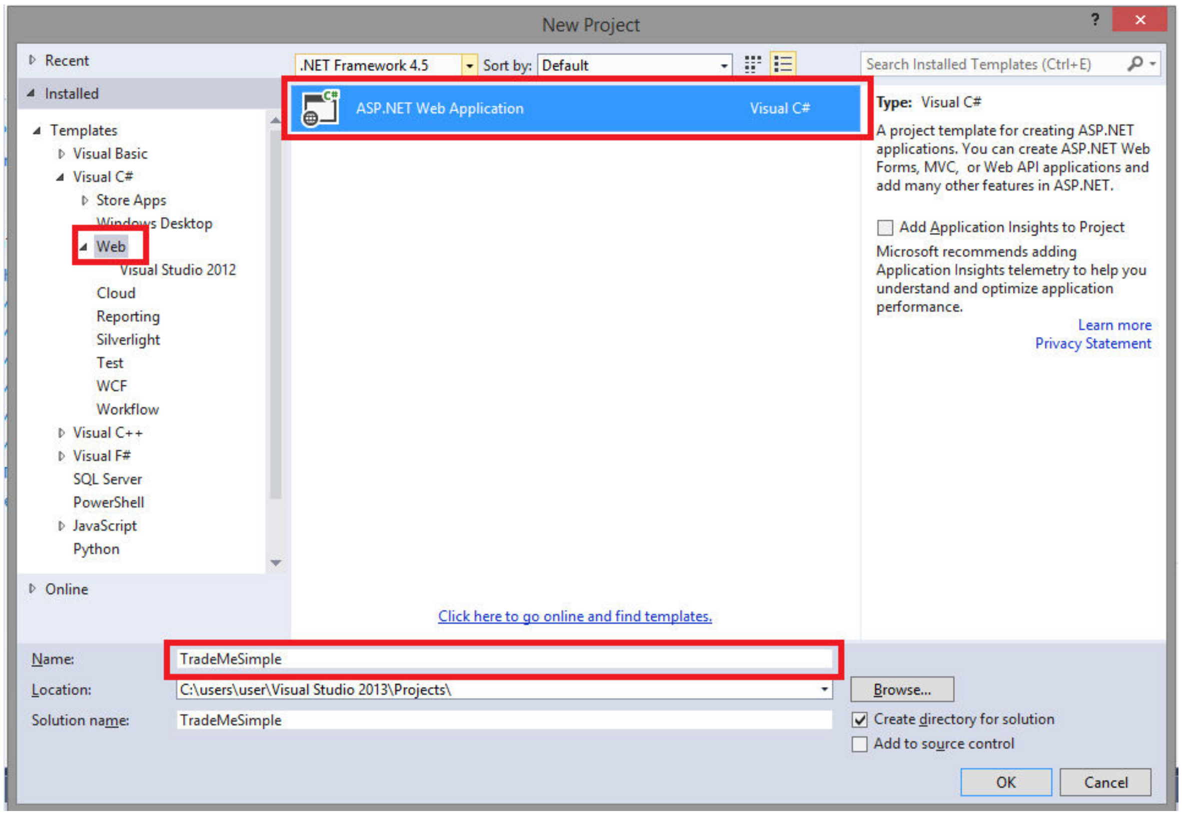
1. PREREQUISITES

Visual Studio 2013 with SQL Database Tools

2. GETTING STARTED

CREATE ASP.NET WEB APPLICATION

Where	Action	Why
Visual Studio	1. Select **File** > **New** > **Project** 2. Under Visual C# -> Web choose **ASP .Net Web Application** 3. **MAKE SURE YOU ARE SELECTING FROM THE <u>VISUAL C#</u> PROJECT LIST, NOT VISUAL BASIC .NET** 4. Name the project **TradeMeSimple** (this is important for code blocks copied later in the lab)	This causes Visual studio to generate a new solution with components required for the MVC5 framework.



Type: Visual C#

A project template for creating ASP.NET applications. You can create ASP.NET Web Forms, MVC, or Web API applications and add many other features in ASP.NET.

☐ Add Application Insights to Project

Microsoft recommends adding Application Insights telemetry to help you understand and optimize application performance.

Name: TradeMeSimple

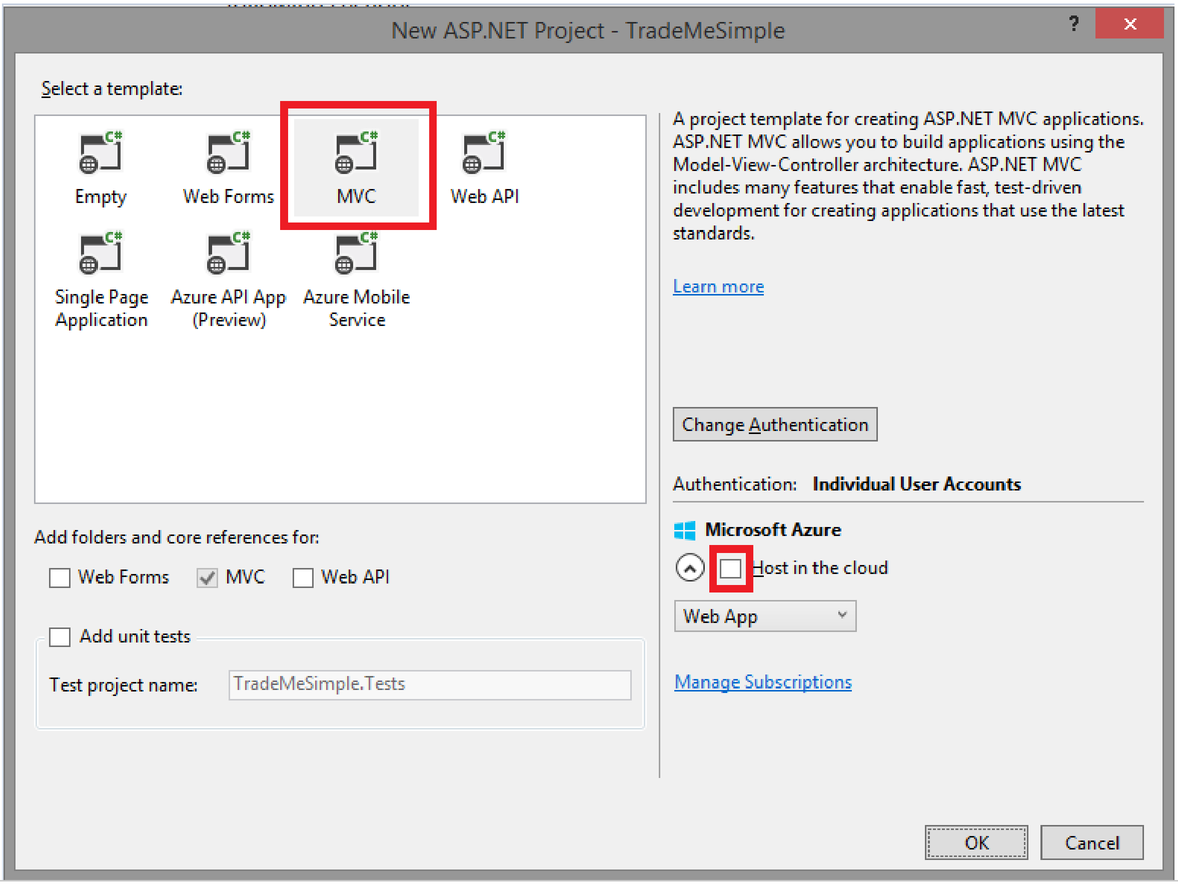
Location: C:\users\user\Visual Studio 2013\Projects\

Solution name: TradeMeSimple

☑ Create directory for solution

☐ Add to source control

	5. In the **New ASP.NET Project** window, select **MVC**	


New ASP.NET Project - TradeMeSimple

Select a template:

A project template for creating ASP.NET MVC applications. ASP.NET MVC allows you to build applications using the Model-View-Controller architecture. ASP.NET MVC includes many features that enable fast, test-driven development for creating applications that use the latest standards.

Learn more

Change Authentication

Authentication: **Individual User Accounts**

Microsoft Azure

[ ] Host in the cloud

Web App

Manage Subscriptions

Add folders and core references for:

[ ] Web Forms [✓] MVC [ ] Web API

[ ] Add unit tests

Test project name: TradeMeSimple.Tests

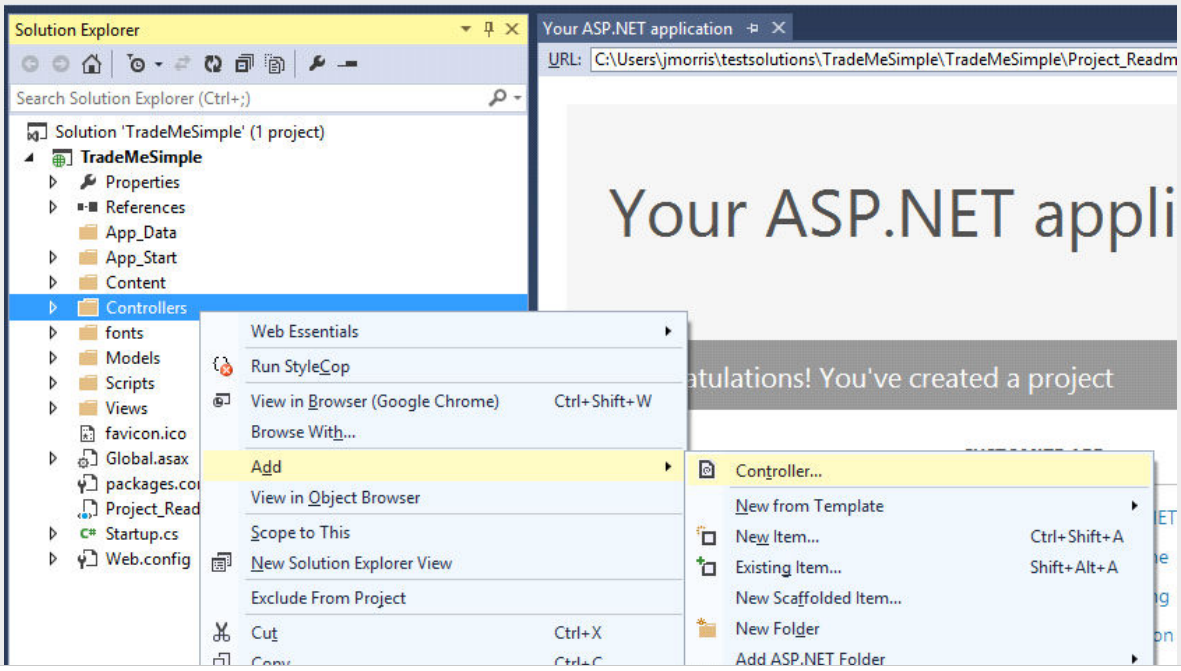
3. SEE THIS APP RUNNING

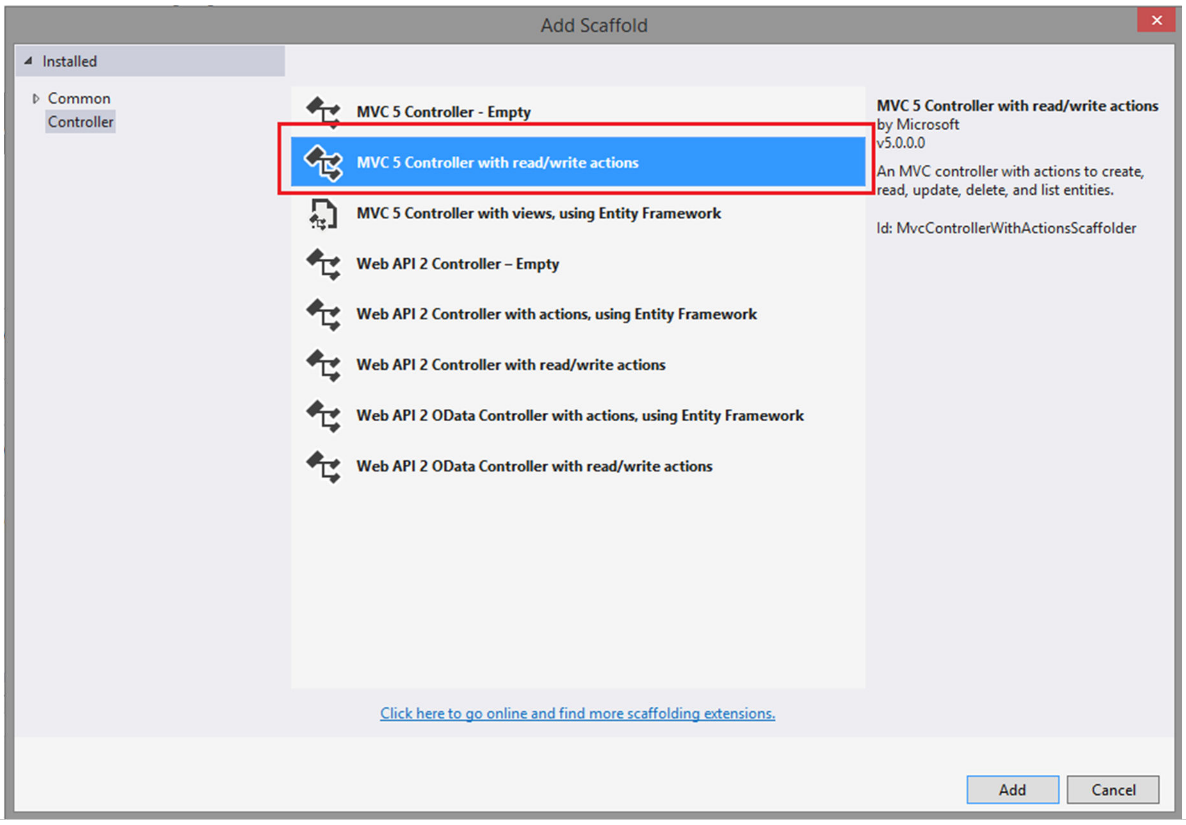
Where	Action	Why
Visual Studio	Press F5 NOTE: An ASP.Net development server will start up with your site running on a specified port. When you see http://localhost:xxxx later in this lab translate the xxxx to your specified port.	This will run the application in browser
Website	Register a new login	Investigate what functionality comes out of the box

4. CREATE A CONTROLLER

Where	Action	Why
Visual Studio	Press SHIFT + F5	To stop the application and allow modifying the C#

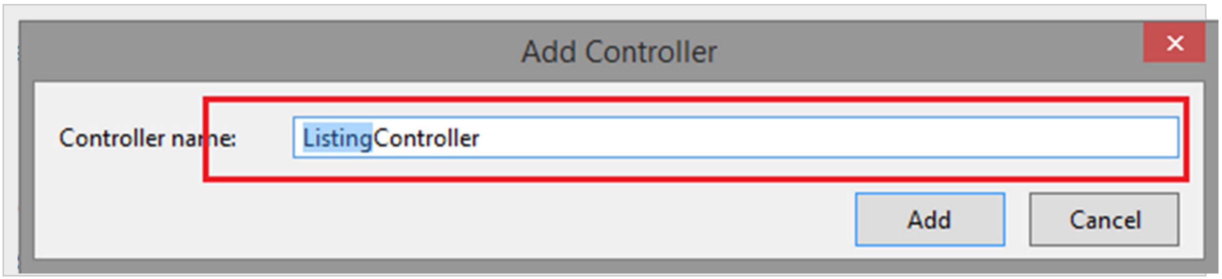
Visual Studio	Right click **Controllers > Add > Controller** • Choose **MVC5 Controller with read/write actions** • Name => ListingController	Creates a new empty controller




Add Scaffold
Installed
Common
Controller
MVC 5 Controller - Empty
MVC 5 Controller with read/write actions
MVC 5 Controller with views, using Entity Framework
Web API 2 Controller – Empty
Web API 2 Controller with actions, using Entity Framework
Web API 2 Controller with read/write actions
Web API 2 OData Controller with actions, using Entity Framework
Web API 2 OData Controller with read/write actions
Click here to go online and find more scaffolding extensions.
Add
Cancel

MVC 5 Controller with read/write actions
by Microsoft
v5.0.0.0
An MVC controller with actions to create, read, update, delete, and list entities.

Id: MvcControllerWithActionsScaffolder



Controller name: ListingController

MVC CAN GENERATE A VIEW AT RUNTIME

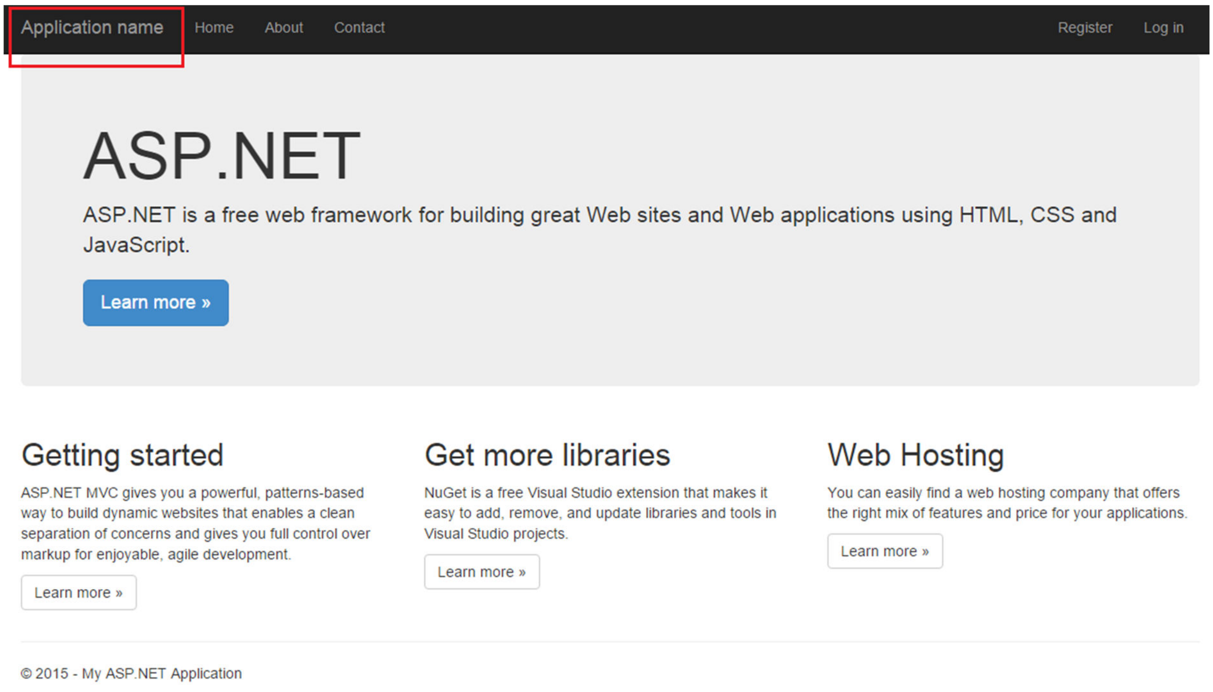
Where	Action	Why

Visual Studio	Change Index method
	`public string Index()`
	`{`
	`    return "Test";`
	`}`	

	Press F5	This will run the application in browser
Website	Browse to http://localhost:xxxx/Listing	A view appears based on the specified text

5. MODIFY THE TEMPLATE (LAYOUT PAGE)

Note Title on all pages => Application name, let’s change that



ASP.NET

ASP.NET is a free web framework for building great Web sites and Web applications using HTML, CSS and JavaScript.

Getting started

ASP.NET MVC gives you a powerful, patterns-based way to build dynamic websites that enables a clean separation of concerns and gives you full control over markup for enjoyable, agile development.

Get more libraries

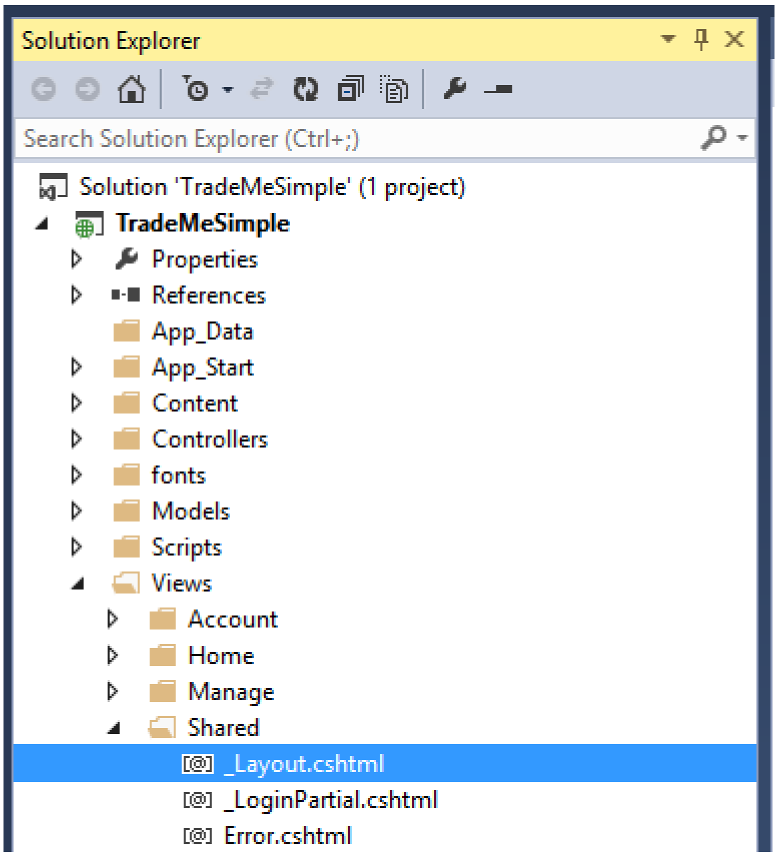
NuGet is a free Visual Studio extension that makes it easy to add, remove, and update libraries and tools in Visual Studio projects.

Web Hosting

You can easily find a web hosting company that offers the right mix of features and price for your applications.

© 2015 - My ASP.NET Application

Where	Action	Why
Visual Studio	Open **Views > Shared >** **_Layout.cshtml** (Note, when the application is running, the file tree may not be showing in Visual Studio. Click **View > Solution Explorer**. You can pin the Solution Explorer to prevent keep it visible at all times).	


Solution Explorer

Search Solution Explorer (Ctrl+;)

```
Solution 'TradeMeSimple' (1 project)
◢ TradeMeSimple
    ▷ Properties
    ▷ References
      App_Data
    ▷ App_Start
    ▷ Content
    ▷ Controllers
    ▷ fonts
    ▷ Models
    ▷ Scripts
    ◢ Views
        ▷ Account
        ▷ Home
        ▷ Manage
        ◢ Shared
              _Layout.cshtml
              _LoginPartial.cshtml
              Error.cshtml
```

MODIFY THE APP TITILE

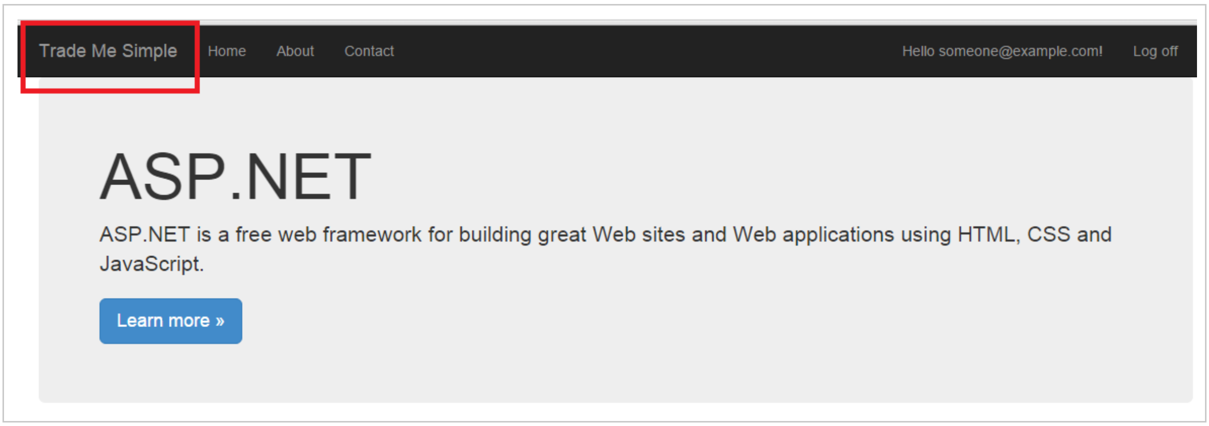
Visual studio

```
<button type="button" class="navbar-toggle" data-
toggle="collapse" data-target=".navbar-collapse">
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                </button>
                @Html.ActionLink("Application name", "Index",
"Home", new { area = "" }, new { @class = "navbar-brand" })
            </div>
```

TO

```
<button type="button" class="navbar-toggle" data-
toggle="collapse" data-target=".navbar-collapse">
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                </button>
                @Html.ActionLink("Trade Me Simple", "Index",
"Home", new { area = "" }, new { @class = "navbar-brand" })
            </div>
```

Changes the template displayed title



ASP.NET

ASP.NET is a free web framework for building great Web sites and Web applications using HTML, CSS and JavaScript.

ADD A NEW NAVIGATION LINK

Where	Action	Why
Visual Studio	Find the block containing the navigation links and add this below: `<li>@Html.ActionLink("My Listings", "Index", "Listing")</li>`	Adds navigation links to the template
	Save _Layout.cshtml	
Website	Browse to http://localhost:xxxx/ The new title and new menu item will appear. Click the new menu item to get to your Controller's Index Action.	Navigate without manually entering URLs.

6. VIEW

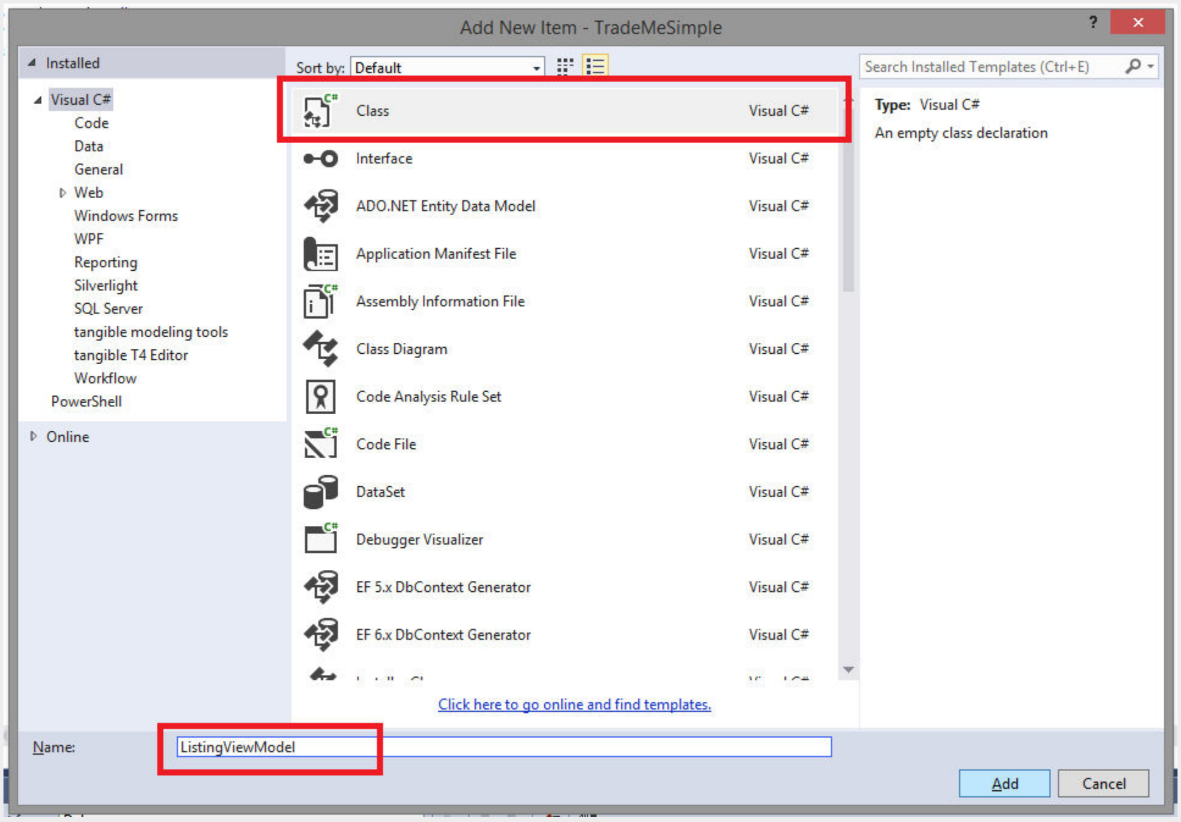
Where	Action	Why
Visual Studio	Press SHIFT + F5	To stop the application and allow modifying the C#

	Back in your ListingController, change the Index method back
	`public ActionResult Index()`
	`{`
	`    return View();`
	`}`	

	Right click Index() method and select Add View – leave all default settings	Demonstrates a simple way to create a new view
	Press F5	Runs the application
Website	Navigate to http://localhost:xxxx/Listing (Index Action is implied if there is no /ActionName in the URL)	See the new view

7. MODELS

Where	Action	Why
Visual Studio	Press SHIFT + F5	To stop the application and allow modifying the C#
	Right click on **Models** folder. Select Add -> Class Name => **ListingViewModel**	


Add New Item - TradeMeSimple

Sort by: Default

Type: Visual C#
An empty class declaration

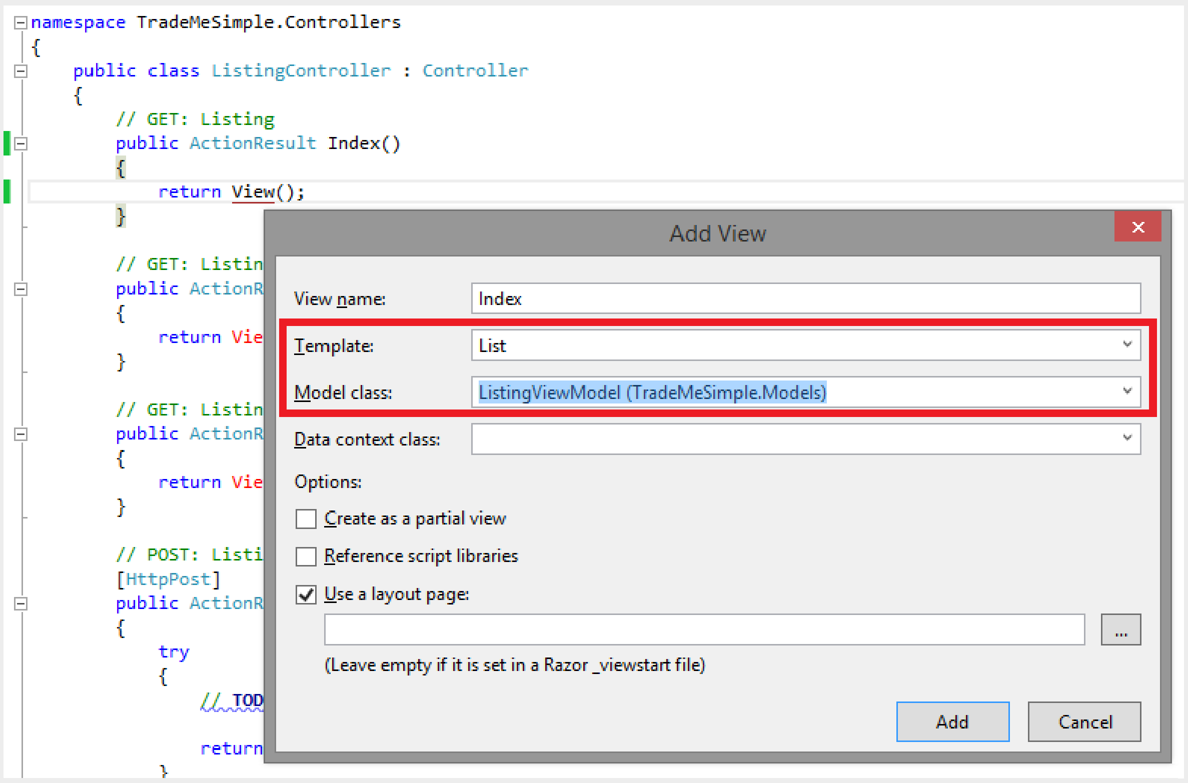
Name: ListingViewModel

Installed
Visual C#
Code
Data
General
Web
Windows Forms
WPF
Reporting
Silverlight
SQL Server
tangible modeling tools
tangible T4 Editor
Workflow
PowerShell
Online
Search Installed Templates (Ctrl+E)
Class Visual C#
Interface Visual C#
ADO.NET Entity Data Model Visual C#
Application Manifest File Visual C#
Assembly Information File Visual C#
Class Diagram Visual C#
Code Analysis Rule Set Visual C#
Code File Visual C#
DataSet Visual C#
Debugger Visualizer Visual C#
EF 5.x DbContext Generator Visual C#
EF 6.x DbContext Generator Visual C#
Click here to go online and find templates.
Add
Cancel

	Specify the model	Defines the properties stored against the model

```
    public class ListingViewModel
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public DateTime StartDate { get; set; }
        public bool Featured { get; set; }
        public int Price { get; set; }
    }
```

ADD VIEW FOR DISPLAYING MODELS		
Visual studio	In ListingController, right click **Index()** action method and select Add View, Use **List** for Template and **ListingViewModel** for the Model class. Leave the other settings default.	Generates a view capable of displaying a list of Listing models to the user.



```
namespace TradeMeSimple.Controllers
{
    public class ListingController : Controller
    {
        // GET: Listing
        public ActionResult Index()
        {
            return View();
        }

        // GET: Listin
        public ActionR
        {
            return Vie
        }

        // GET: Listin
        public ActionR
        {
            return Vie
        }

        // POST: Listi
        [HttpPost]
        public ActionR
        {
            try
            {
                // TOD

                return
            }
```

Add View

View name: Index

Template: List

Model class: ListingViewModel (TradeMeSimple.Models)

Data context class:

Options:

☐ Create as a partial view

☐ Reference script libraries

☑ Use a layout page:

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

Click "Yes" to replace the view.

	Note: if you right-click on the html/razor template which was created for you, you can use "Go to controller". Right-click on the word "View" in the controller, and you can navigate back to the view.	

	Paste the following array into the ListingController class:
	`    public class ListingController : Controller`
	`    {`
	`        private static readonly ListingViewModel[]`
	`Listings =`
	`        {`
	`            new ListingViewModel`
	`            {`
	`                Id = 1,`
	`                Name = "My Ferrari",`
	`                StartDate = DateTime.Now,`
	`                Featured = false,`
	`                Price = 50000`
	`            },`
	`            new ListingViewModel`
	`            {`
	`                Id = 2,`
	`                Name = "My House",`
	`                StartDate = DateTime.Now,`
	`                Featured = true,`
	`                Price = 500000`
	`            }`
	`        };`
	
	`You will also need`
	`using TradeMeSimple.Models;`
	`at the top of the file. This can be done for you if you`
	`click the little lightbulb in the left margin.`
		To give us some data to use, without worrying about a real database.

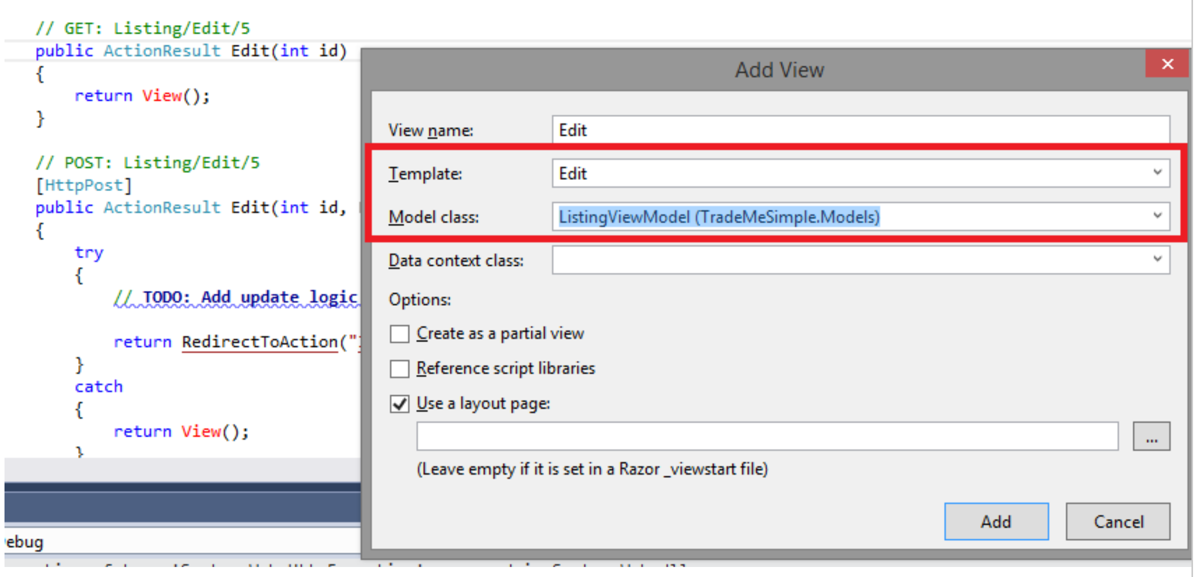
	Modify the Index action to pass the fake model data to the view
	`        public ActionResult Index()`
	`        {`
	`            return View(Listings);`
	`        }`
		Cause the model data to display in the View

	Press F5	Runs the application
Website	Navigate to My Listings	Check out the template Visual Studio created for you based on the model's properties.

ADD VIEW FOR EDITING A MODEL

Visual studio	Press SHIFT + F5	To stop the application and allow modifying the C#

	Back in your ListingController, Right click Edit(`int id`) method and select Add View. Use **Edit** for Template and **ListingViewModel** for the Model class. Leave the other settings default. **N.B. Make sure you do NOT do this on the [HttpPost] version of the Edit Action.**	



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

// POST: Listing/Edit/5
[HttpPost]
public ActionResult Edit(int id,
{
    try
    {
        // TODO: Add update logic

        return RedirectToAction("
    }
    catch
    {
        return View();
    }
```

ebug

Add View

View name: Edit

Template: Edit

Model class: ListingViewModel (TradeMeSimple.Models)

Data context class:

Options:

☐ Create as a partial view

☐ Reference script libraries

☑ Use a layout page:

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

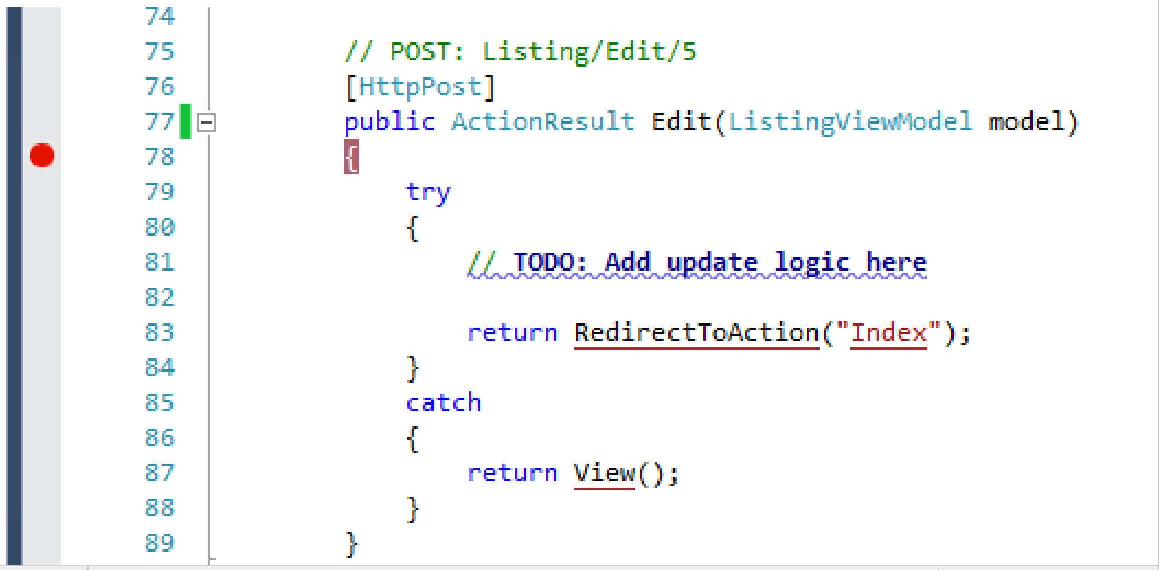
	Back in your ListingController, Modify the **Edit** action to pass the model with the selected Id to the view. The id will be automatically received from the last part of the URL path.
	`        // GET: Listing/Edit/5`
	`        public ActionResult Edit(int id)`
	`        {`
	`            var modelToEdit = Listings.Single(model =>`
	`model.Id == id);`
	`            return View(modelToEdit);`
	`        }`
		Supply the correct model to the Edit view so it can display it in its form.

	Press F5	Runs the application
Website	Navigate to My Listings and click the Edit link on one of the listings.	Check out the form Visual Studio created for you based on the model's properties.

8. DEBUGGING

Where	Action	Why
Visual Studio	Press SHIFT + F5	To stop the application and allow modifying the C#

	Back in your ListingController, modify the **[HttpPost]** version of the Edit action to receive the specific model.
	`        // POST: Listing/Edit/5`
	`        [HttpPost]`
	`        public ActionResult Edit(ListingViewModel model)`
	`        {`
	`            try`
	`            {`	

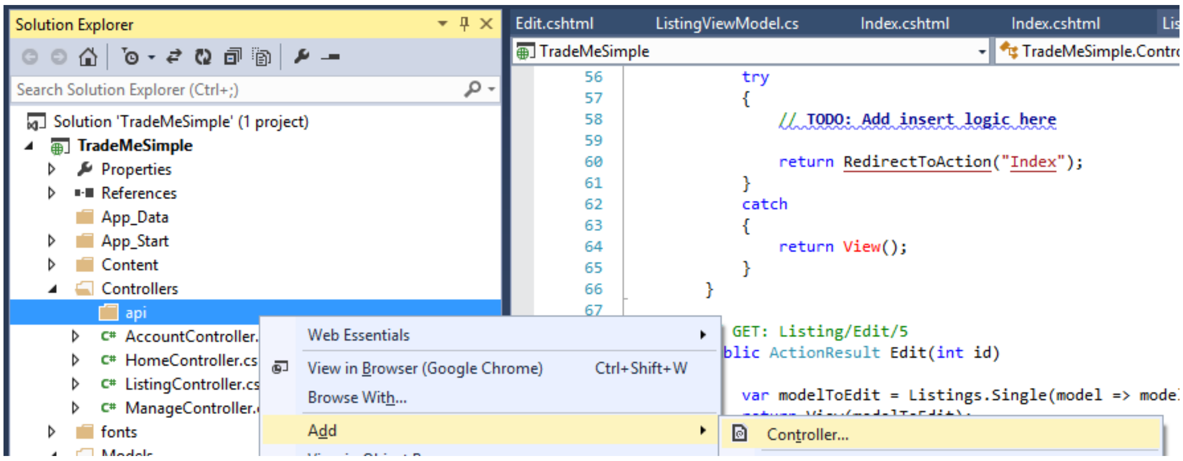
	Click in the gray strip in the left margin to put a Breakpoint in the **[HttpPost]** Edit action:	To pause the code at this point and allow debugging
		
	Press F5	Runs the application with Debugging. Note, CTRL+F5 is used to run without debugging or breakpoints.
Website	Edit one of the listings, change some data and click save.	Hit the breakpoint placed in the Edit post action.
Visual Studio	Examine Locals / Call Stack views. Hover over **model** parameter in the Edit method and you will see the content entered in the form fields has come through on the model.	Get to know debug functionality, and see how you can get hold of user input.
	Press F5 to continue running past the breakpoint, or press SHIFT + F5 to stop the application. F10 would step line-by-line.	To unpause execution

```
        // POST: Listing/Edit/5
        [HttpPost]
        public ActionResult Edit(ListingViewModel model)
        {
            try
            {
                // TODO: Add update logic here

                return RedirectToAction("Index");
            }
            catch
            {
                return View();
            }
        }
```

9. API CONTROLLERS (HTTP URLS FOR AJAX OR OTHER APPS TO ACCESS)

Where	Action	Why
Visual Studio	Press SHIFT + F5	To stop the application and allow modifying the C#
ADD A WEBAPI CONTROLLER		
Visual Studio	Create a folder called “api” in the Controllers folder.	This is the conventional place to put API controllers.
	Right-click on the api folder and add a **Web API 2 Controller with read/write actions**, named **ListingsController** (note the plural this time – important to get good URL path names for this RESTful resource).	Create a controller which handles endpoints for HTTP GET, POST, etc. API requests.

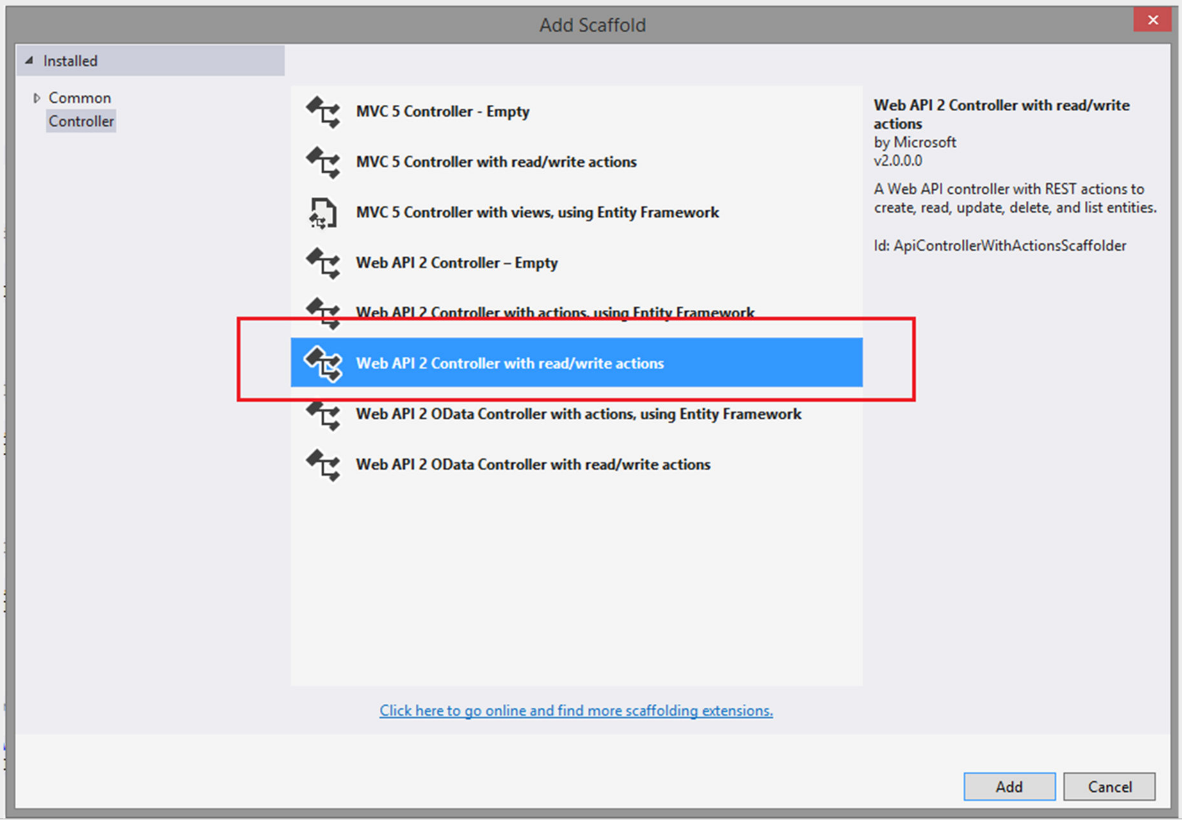


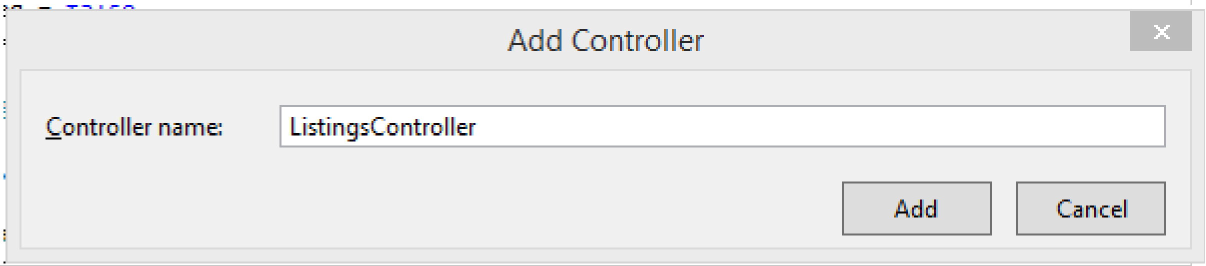
```
            try
            {
                // TODO: Add insert logic here

                return RedirectToAction("Index");
            }
            catch
            {
                return View();
            }
        }

 GET: Listing/Edit/5
blic ActionResult Edit(int id)

    var modelToEdit = Listings.Single(model => mode
```


Add Scaffold
Installed
Common
Controller
MVC 5 Controller - Empty
MVC 5 Controller with read/write actions
MVC 5 Controller with views, using Entity Framework
Web API 2 Controller – Empty
Web API 2 Controller with actions, using Entity Framework
Web API 2 Controller with read/write actions
Web API 2 OData Controller with actions, using Entity Framework
Web API 2 OData Controller with read/write actions
Web API 2 Controller with read/write actions
by Microsoft
v2.0.0.0
A Web API controller with REST actions to create, read, update, delete, and list entities.
Id: ApiControllerWithActionsScaffolder
Click here to go online and find more scaffolding extensions.
Add
Cancel



Controller name: ListingsController

	After adding the controller you will be shown a readme.txt file which gives you a few steps to take to enable WebAPI controllers. Follow all the steps, noting that for step 3, the given line of code must be added as the **first line** in the `Application_Start()` method.	Add support for WebAPI to enable the API Controllers in the project.

```
[reademe.txt]

Visual Studio has added the full set of dependencies for ASP.NET Web API 2 to project
'RubbishMVC5'.

The Global.asax.cs file in the project may require additional changes to enable ASP.NET Web
API.

1. Add the following namespace references:

    using System.Web.Http;
    using System.Web.Routing;

2. If the code does not already define an Application_Start method, add the following method:

    protected void Application_Start()
    {
    }

3. Add the following lines to the beginning of the Application_Start method:

    GlobalConfiguration.Configure(WebApiConfig.Register);
```

In the ListingsController class, add the following highlighted code:

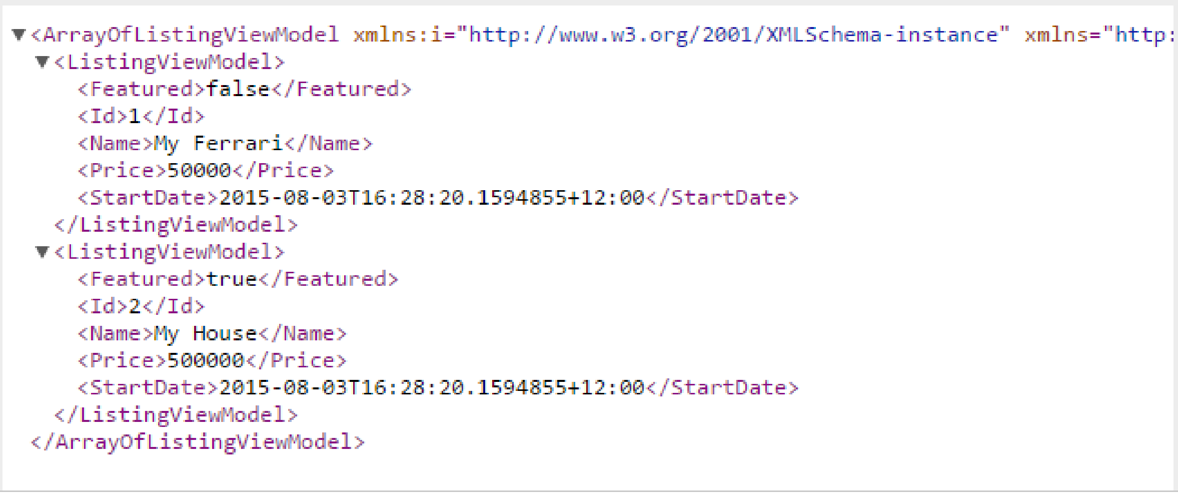
```
    public class ListingsController : ApiController
    {
        private static readonly ListingViewModel[]
Listings =
        {
            new ListingViewModel
            {
                Id = 1,
                Name = "My Ferrari",
                StartDate = DateTime.Now,
                Featured = false,
                Price = 50000
            },
            new ListingViewModel
            {
                Id = 2,
                Name = "My House",
                StartDate = DateTime.Now,
                Featured = true,
                Price = 500000
            }
        };

        // GET: api/Listing
        public IEnumerable<ListingViewModel> Get()
        {
            return Listings;
        }
```

You will also need
`using TradeMeSimple.Models;`
at the top of the file. This can be done for you if you click the little lightbulb in the left margin.

Provide some fake data so we don't have to worry about a real database, and set up the main HTTP GET method to return the model data.

Website	In Firefox or Chrome, Browse to http://localhost:xxxx/api/listings You can use the drop-down arrow on the green Play > button in the Visual Studio toolbar and choose **Browse With**, or Press F5 in Visual Studio and manually enter the URL into one of the desired browsers. You should see something like the below:	Check that the API endpoint is working, and demonstrate that WebAPI recognises when a browser is directly accessing the URL and serialises the models into XML.



```
▼<ArrayOfListingViewModel xmlns:i="http://www.w3.org/2001/XMLSchema-instance" xmlns="http:
  ▼<ListingViewModel>
     <Featured>false</Featured>
     <Id>1</Id>
     <Name>My Ferrari</Name>
     <Price>50000</Price>
     <StartDate>2015-08-03T16:28:20.1594855+12:00</StartDate>
   </ListingViewModel>
  ▼<ListingViewModel>
     <Featured>true</Featured>
     <Id>2</Id>
     <Name>My House</Name>
     <Price>500000</Price>
     <StartDate>2015-08-03T16:28:20.1594855+12:00</StartDate>
   </ListingViewModel>
 </ArrayOfListingViewModel>
```

CALL THE API ENDPOINT WITH JAVASCRIPT/AJAX

Visual studio	Press SHIFT + F5	To stop application
	Right-click on the Script folder and choose Add->Javascript File. Name the file **ListingsGetter.js** Put the following code in the file:	This javascript function runs on page load and assigns a click event to any button named “btn-listings”. The click event calls the Listings API endpoint, shows an alert and creates a browser console log entry.

```
$(function () {
    function getListings() {
        $.get('/api/listings', function (data) {
            alert('Retrieved ' + data.length + '
listings.');
            console.log(data);
        });
    }
    $('button[name="btn-listings"]').click(getListings);
});
```

	In the **Views > Listing > Index.cshtml**, add the following markup (highlighted): `<h2>Index</h2>` `<p><button name="btn-listings">Do Ajax!</button></p>` `<p>` `    @Html.ActionLink("Create New", "Create")` `</p>` And reference the javascript file (this can go at the bottom of the Index.cshtml file): `@section scripts{` `    @Scripts.Render("~/Scripts/ListingsGetter.js")` `}`	Add a button to get wired up to the javascript click event, and include the file with the javascript function into the view so it gets loaded by the browser
	Press F5	Runs the application
Website	Navigate to My Listings, and click the “Do Ajax!” button you have added. You should see an alert saying, “Retrieved 2 listings”. Press F12 to bring up the browser’s developer tools. Check out the **Network** tab and click the “Do Ajax!” button again. Examine the response body. Depending on the browser, you may also need to click the “Enable network traffic capturing” toolbar button, click or double-click the request, and/or click the **Response** sub-tab. This will allow you to examine the HTTP content which is returned by your ListingsController API. Also, check out the **Console** tab to see the log outputs. You can expand the [Obect, Object] array variable to see the values received.	Demonstrate that WebAPI automatically serializes the models to JSON when requested by JQuery. Also demonstrate that JQuery automatically deserializes the JSON back into javascript objects which you can access in the browser.
EXTRA FOR HARDENED ROOKIES		
	Get the listings retrieved via ajax to display in the page content instead of just the console. Retrieve a single listing, by ID, via ajax. Use ajax to send a listing model to an ApiController POST method and show the model has reached the server using a breakpoint.	Because you are awesome.