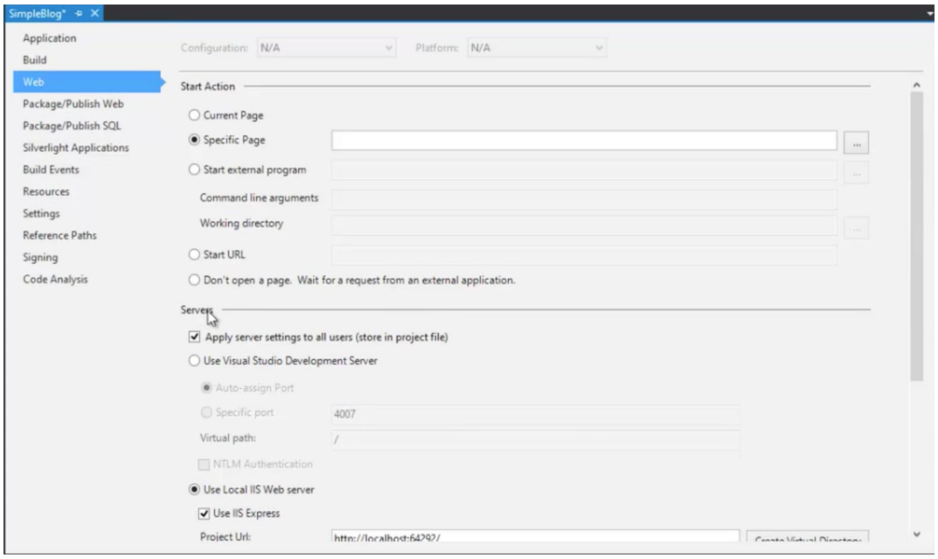
ROUTERS, AREAS AND CONTROLLERS

ASP.NET MVC C07

In this course, we are actualy write some code. I just want to show you what will be if I runned this application now. There is two ways to run our application.



Configuration: N/A Platform: N/A

Start Action

- ○ Current Page
- ● Specific Page
- ○ Start external program
- Command line arguments
- Working directory
- ○ Start URL
- ○ Don't open a page. Wait for a request from an external application.

Servers

- ☑ Apply server settings to all users (store in project file)
- ○ Use Visual Studio Development Server
  - ● Auto-assign Port
  - ○ Specific port 4007
  - Virtual path: /
  - ☐ NTLM Authentication
- ● Use Local IIS Web server
  - ☑ Use IIS Express
  - Project Url: http://localhost:64292/

Running Project - First Let's See Web server Types

If I click on SimpleBlog (Project - Not Solution) anc click Properties, Under Web section, different choises I have for for web server. Now, in order to host the application for development purposes, we need an actual web server right. We need a web server so we can actualy see the site. We have a bunch of settings under web section.

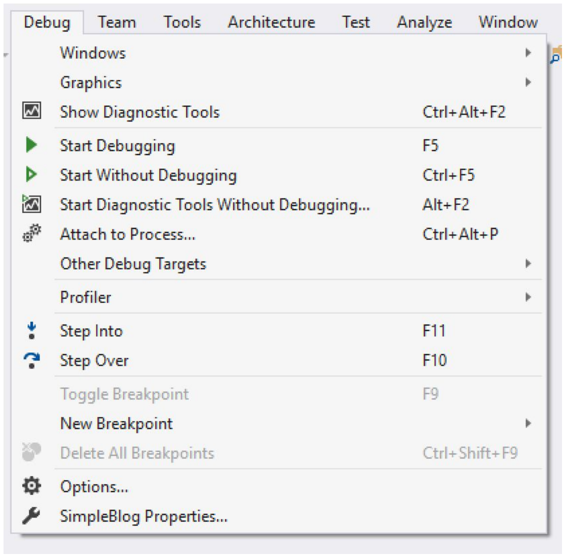
We have User Visual Studio Development Server
We have user local IIS Web Server
We have Use IIS Express
And we have User Custom Web Server

We are gonna be using IIS Express. IIS Express is a web server that functions almost exactly like IIS. Remember IIS is the web server we are goint to deploy on it. IIS Express is the development version of that server. That gives us all of the features and functionality of IIS packaged in a more developer friendly way.

We could use an actual IIS web server with unchecking Use IIS Express.

We can also use a Visual Studio Development Server. This is a legacy(inheritance) component. IIS express was introduced in last couple of years.

So we will use IIS Express.
Close our project properties.



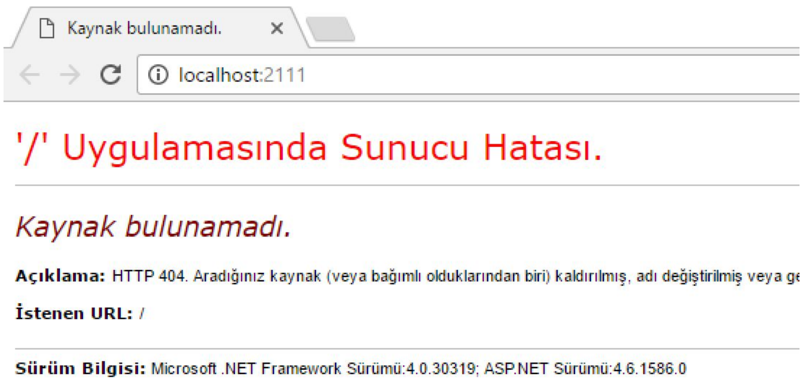
Running Project - We Have Two Options

We have two options for starting our project.

Start Debugging
Start Without Debugging

The way that we are going to working mostly with this application is “Start Without Debugging” - CTRL + F5
That will launch the web server in the background, and still allows to edit code, recompile our project while the web server is running which is nice.

If we “Start Debugging” however, we will get the normal Visual Studio Debugging experience attached to our web server. We do not really want that because we need to stop debugging when we need a modification on code.


Kaynak bulunamadı.

localhost:2111

'/' Uygulamasında Sunucu Hatası.

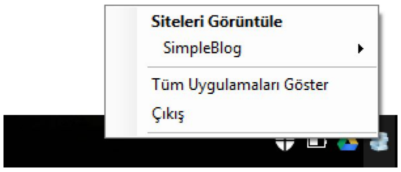
Kaynak bulunamadı.

Açıklama: HTTP 404. Aradığınız kaynak (veya bağımlı olduklarından biri) kaldırılmış, adı değiştirilmiş veya ge

İstenen URL: /

Sürüm Bilgisi: Microsoft .NET Framework Sürümü:4.0.30319; ASP.NET Sürümü:4.6.1586.0

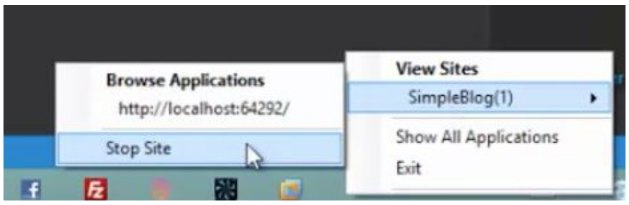
1

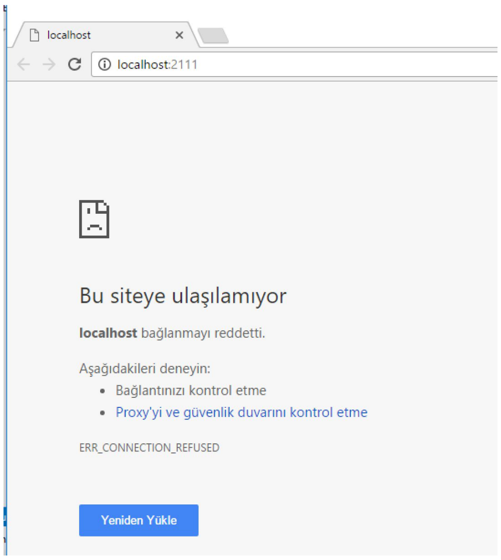


2

Start Without Debugging - CTRL F5 - Response Code 404

When we click “Start Without Debugging” It is going to do two things.
First of all, it launch the Chrome Window. Pointed to localhost:2111 It is just a random port. I see an error we will talk about that.
The next thing is IIS express launches. We see it on taskbar. If we right click on it, we see all of the sites, that running on the we server. SimpleBlog is running and also you can see its location.





Bu siteye ulaşılamıyor

localhost bağlanmayı reddetti.

Aşağıdakileri deneyin:

- Bağlantınızı kontrol etme
- Proxy'yi ve güvenlik duvarını kontrol etme

ERR_CONNECTION_REFUSED

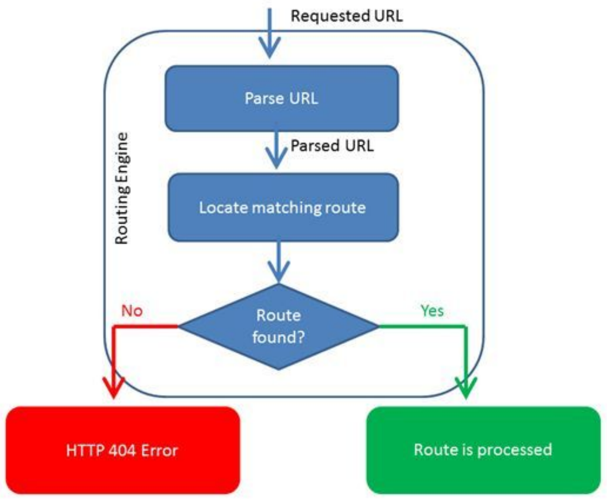
Stopping The Web Site

Now, we know the IIS express is running so we can refresh server. If we did not have IIS running, Let’s stop the web site. We can not stop on Visual Studio.

To stop it, right click on IIS Express on system trash, we can say Exit, or we can select a specific site for stop.

If I stopped the site, we do not get a server error, we will get a browser error saying could not connect anything. So because, there is noting to connect.

So, back into Visual Studio and CTRL F5 for “Start Without Debugging” and now our web server is back running, IIS Express is running our web application and we are getting a 404. Exactly we do not have anything inside our project that make anything other than 404.



Routing - One of the most fundemantialy important aspect of ASP.NET MVC

During project development, our IIS Express will be running all the time.

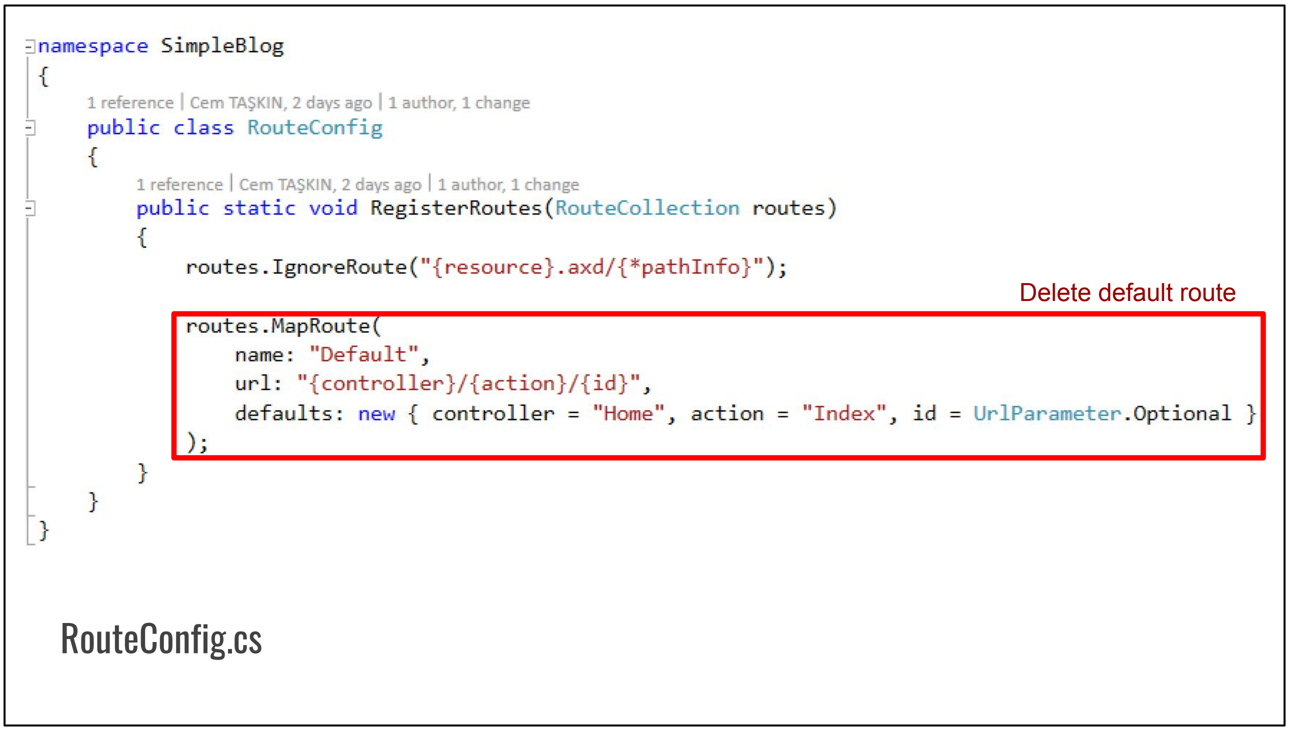
So let's talk about one of the most fundemantaly important aspects of ASP.NET MVC. That is routing.

Unlike php, or web forms, web forms now support routing, but typically web forms in past does not have routing, php does not have routing, coldfusion does not have routing.

With ASP.NET MVC routing comes a part of framework, it is actually a part of ASP.NET itself now.

And what routing means is, it is the way to describe to the web server what path goes to what controller.

Our request from our browser goes into the web server. The web servers spuls up of ASP.NET, ASP.NET spuls up our global.asax. global.asax spolls up ASP.NET MVC. Then tries to figure out, ok we are going to particular URL, what controller needs to be invoked. And that's what routing is.



```
namespace SimpleBlog
{
    1 reference | Cem TAŞKIN, 2 days ago | 1 author, 1 change
    public class RouteConfig
    {
        1 reference | Cem TAŞKIN, 2 days ago | 1 author, 1 change
        public static void RegisterRoutes(RouteCollection routes)
        {
            routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

            routes.MapRoute(
                name: "Default",
                url: "{controller}/{action}/{id}",
                defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional }
            );
        }
    }
}
```

If we go to our RouteConfig, we see cup of routes written for us.

The first one we have is routes.IgnoreRoute route anything.axd.
axd is a special extensions for files that do not actually exits on the filesystem that are used for debugging purposes. We are not be using an axd file but i am leave that ignore.

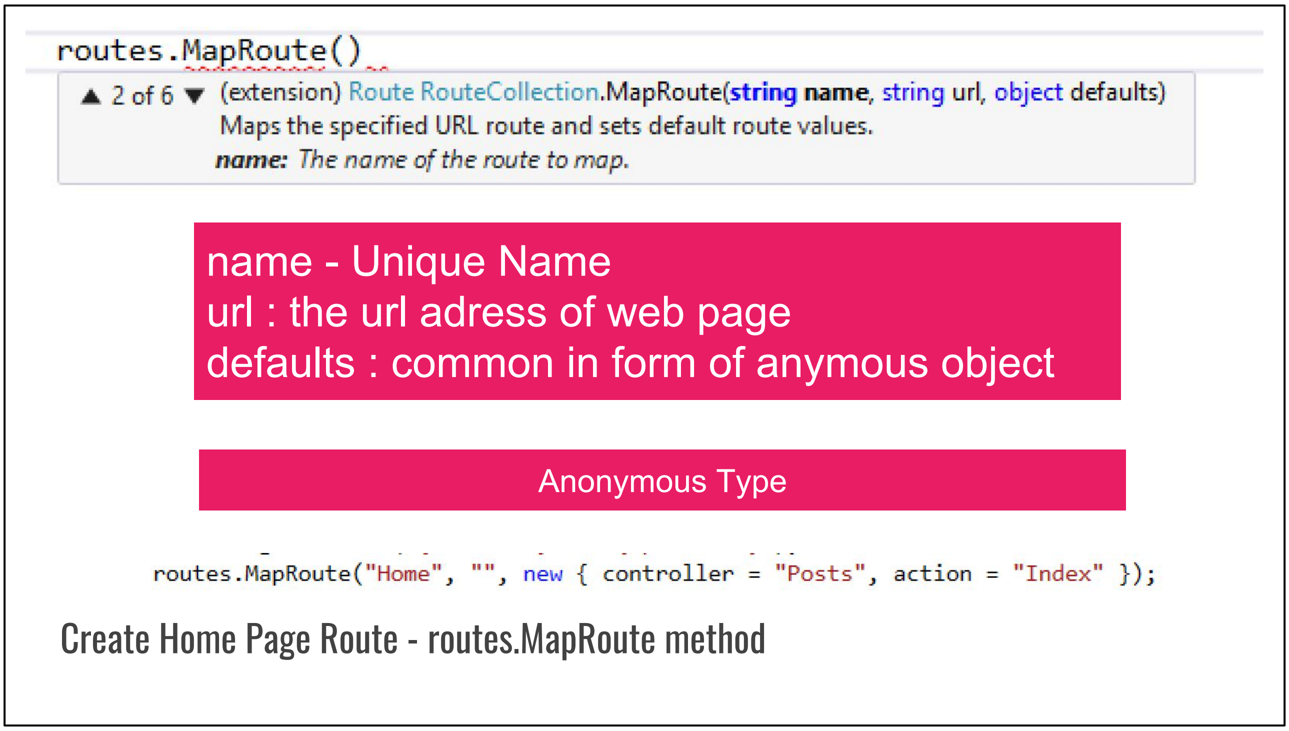
Next up, we have a informance default route. We delete the default route.
We are going to talk about default route, when we go ahead an create an area.

For the front end, i am not going to use default route. Instead of i am going to is every single individual url that type into our application for example we gonne have … / posts, or .../login or ../admin all the stuff i am going to make explicit for the frontend.

The reason i am gonna do that has two cause.

First cause, i really wanna explain how to write your own routes.
Second cause when you do things on the frontend and you do not use the default route, you get more control over the way your URLs look. The way your URLs work is important for SEO purposes as well as if you had a RESTFUL web service.

So every single page that can be loaded by the application for the frontend, not for the admin, will have an explicit entiry in our register routes method that describes the ASP.NET MVC how to get the controller that it needs.



```
routes.MapRoute()
▲ 2 of 6 ▼ (extension) Route RouteCollection.MapRoute(string name, string url, object defaults)
           Maps the specified URL route and sets default route values.
           name: The name of the route to map.
```

name - Unique Name
url : the url adress of web page
defaults : common in form of anymous object

Anonymous Type

```
routes.MapRoute("Home", "", new { controller = "Posts", action = "Index" });
```

Create Home Page Route - routes.MapRoute method

We will start with home page. That is a suitable place to start.

routes.MapRoute () , i am going to look at this method. This method has quite a few different overloads.But at base, I take the name and the URL. They can also take default programmers. They can also take constraints. They can take namespaces or pretty much any combinations.

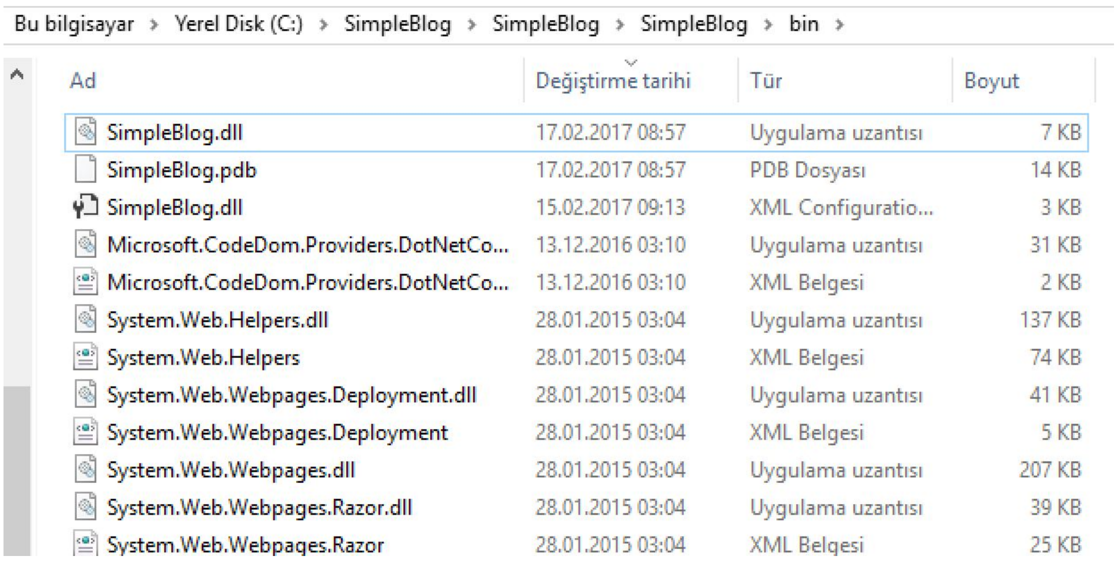
Every route needs a name. And every road name needs to be unique. I am goint to call this route “Home”. The cool thing about this route is that by naming it “Home”, whenever I wanna generate a link to this route, I just have to tell the name “Home” and it knows where to go. Even If I changed the actual URL of this route points to down the route, gives us a lot of flexibility.

The next step is to find the url. The home route has an empty url. Because, it is goint to be the route that is executed when we do not   anything following the web site. This is the home page. Therefore the url is blank.

Next step, we over defaults. Defaults are, incredibly important especially when we are defining customer routes. The defaults are common in form of anonymous objects.

Anonymous type : we use new { } , we define properties despite of defining in a class.

We use an anonymous type for defining controller and action. We have not talk about actions yet. When we create correspoding controller for Home, we talk in detail. When creating Home route, I say controller will be Posts and action will be Index. We have not talked about actions yet.



Bu bilgisayar > Yerel Disk (C:) > SimpleBlog > SimpleBlog > SimpleBlog > bin >

Ad	Değiştirme tarihi	Tür	Boyut
SimpleBlog.dll	17.02.2017 08:57	Uygulama uzantısı	7 KB
SimpleBlog.pdb	17.02.2017 08:57	PDB Dosyası	14 KB
SimpleBlog.dll	15.02.2017 09:13	XML Configuratio...	3 KB
Microsoft.CodeDom.Providers.DotNetCo...	13.12.2016 03:10	Uygulama uzantısı	31 KB
Microsoft.CodeDom.Providers.DotNetCo...	13.12.2016 03:10	XML Belgesi	2 KB
System.Web.Helpers.dll	28.01.2015 03:04	Uygulama uzantısı	137 KB
System.Web.Helpers	28.01.2015 03:04	XML Belgesi	74 KB
System.Web.Webpages.Deployment.dll	28.01.2015 03:04	Uygulama uzantısı	41 KB
System.Web.Webpages.Deployment	28.01.2015 03:04	XML Belgesi	5 KB
System.Web.Webpages.dll	28.01.2015 03:04	Uygulama uzantısı	207 KB
System.Web.Webpages.Razor.dll	28.01.2015 03:04	Uygulama uzantısı	39 KB
System.Web.Webpages.Razor	28.01.2015 03:04	XML Belgesi	25 KB

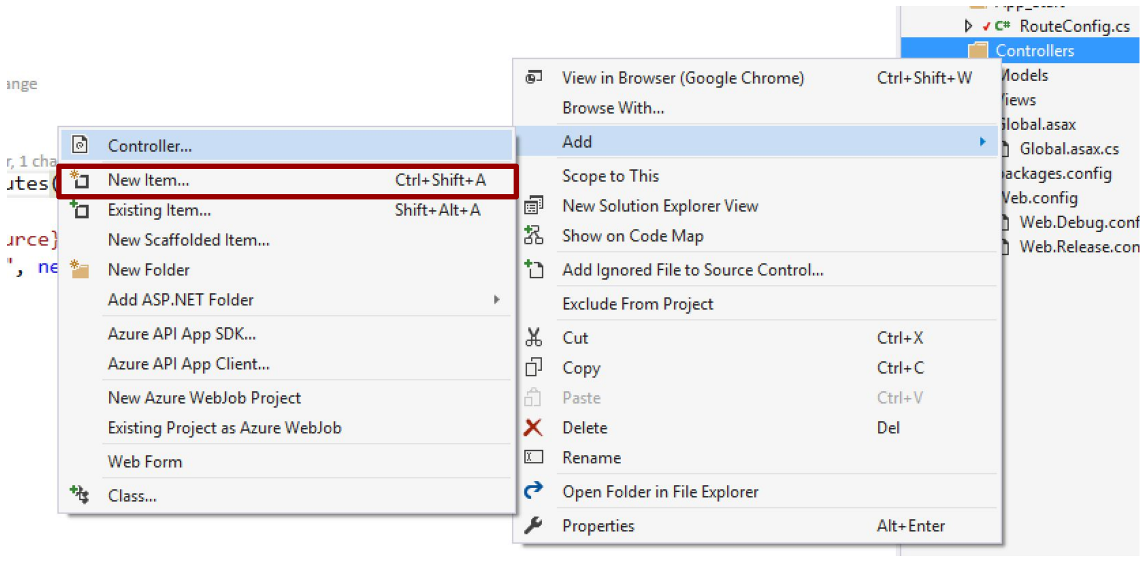
ASP.NET IS COMPILED

Ok, here something is really important to understand about ASP.NET MVC. The changes I have wrote can't take effect on the website. That is because, ASP.NET MVC is compiled. It is actually compiled into DLL's located in the bin folder of our project. As you see there is file named SimpleBlog.dll. This is the compiled assembly. There is result of compiling this project goes into this assembly. This assembly is picked up by ASP.NET.

That means that for you, whenever you edit a cs file, whenever you edit any code, any backend code, exceptions on this rule we will get into views, ever you edit any controller or model code, you must recompile your application code. You must C# to take all over the code and turn it into a DLL.

When it happens, IIS Express, will know the DLL was updated and it restars our application for us. So the easiest way to recompile we can press CTRL F5. And easier way however, if our web server is overly running we can simply call Build - Build Soluiton. By calling this command we see “Build Success” on the bottom left our window. This means, C# turns all cs files into the DLL and placed it into the bin folder.

IIS Express knows that file is chaged. Restart the application with the new code. And it is now ready to be use.



Our Home Page Route was defined. We must Create Posts Controller

By defining the Home page route, we want to route all request for the homepage to the Posts controller Index action. Actually we do not have a Posts controller or an Index action.

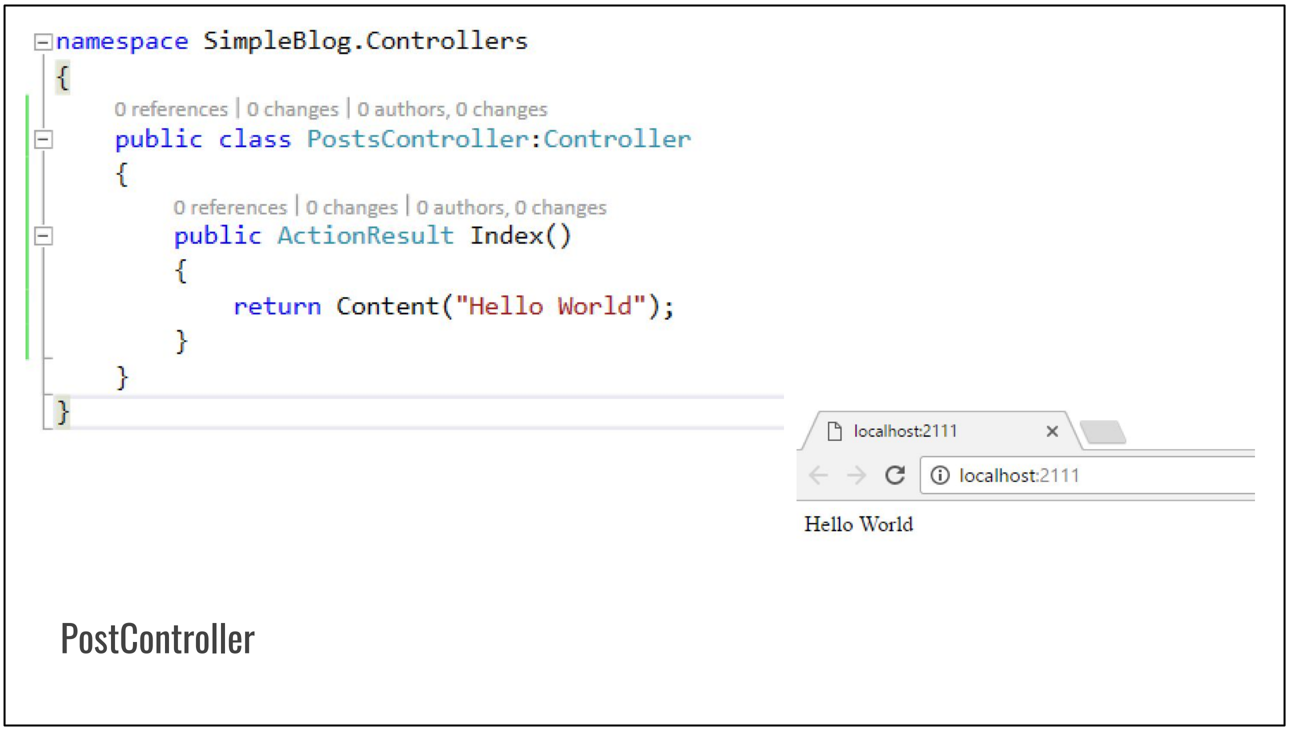
So let,s go and crete. Remember, I can place this controller anywhere that I want in my project, but I will place into Controlles folder. Because it is a convention.

Right click Controllers -> Add -> Controller. We do not actually do that. We click New Item.

I am goint to add a Class. I did not choose Controller because I want to show what this template has in code.

Let’s build the control from strach.

What do you thing the name of the class will be. The name of the class will be PostsController. It is a convention.



```
namespace SimpleBlog.Controllers
{
    0 references | 0 changes | 0 authors, 0 changes
    public class PostsController:Controller
    {
        0 references | 0 changes | 0 authors, 0 changes
        public ActionResult Index()
        {
            return Content("Hello World");
        }
    }
}
```

```
Hello World
```

PostController

After adding the new class, it is not yet a controller. To make it a controller I have to override it from Controller base class.
When we write this we get an error. We must import System.Web.MVC.

That is actually all to make a new controller. I rebuild my project. When we refresh our web site, we see the same thing. That's because, I have not created the Index action.

So what is a action ?

Controller, you can think about a class. And a action, you can thing about as a method. Remember, classes can have multiple methods. Actions are what we can route to.

So, how to make an aciton. We make a method, but we must follow a special signature.

Must be public. It is going to return a ActionResult. The name of the method is going to be the name of the action. In this case, we do not need a parameter.
It is not ready now. If I rebuild the project I get an error. Saying that our action does not return a result.
So how we return an action result.

ActionResult is just a class. Custom actionResult could be defined. ASP.NET is very flexible for returning an action result.

But the base Controller Class actually gives us a lot of build in methods, that constructs special aciton results. There a few them but we talk later about these classes. The one we will use now is Content. Example of other action results are Redirect, RedirectTo, File, View.

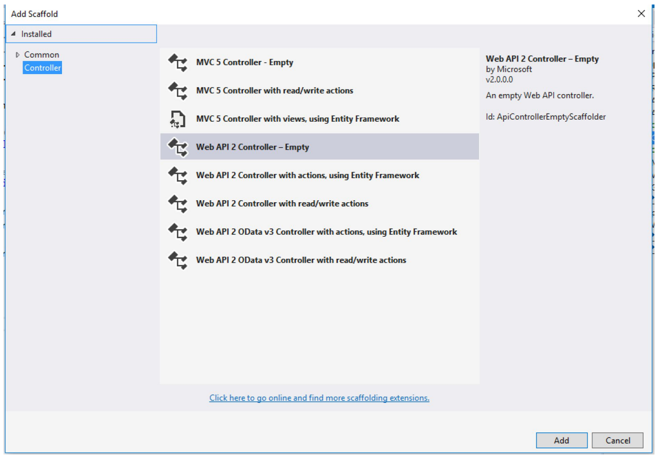
The Content Method, needs a String in its constructor. We return a Hello World.

When we rebuild our project, we see an Hello World in Browser.

So, that is our posts controller, and you can see it is our homepage of our application. We are going to need a couple of more controllers. I am just going to add them and just returning basic content for the so purpose of getting some more examples of routes into place. So we can make some practice.

```
routes.MapRoute("Login", "login", new { controller = "Auth", action = "Login" });
```

Root Name : Login
Root Url : login
Defaults : contorller : Auth, Action : login


Add Scaffold

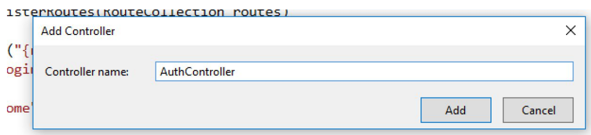
Installed
- Common
- Controller

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 v3 Controller with actions, using Entity Framework
Web API 2 OData v3 Controller with read/write actions

Web API 2 Controller – Empty
by Microsoft
v2.0.0.0
An empty Web API controller.
Id: ApiControllerEmptyScaffolder

Click here to go online and find more scaffolding extensions.

Add | Cancel


Add Controller

Controller name: AuthController

Add | Cancel

Login Route

So just write a login route.

I will leave the home root at the bottom.

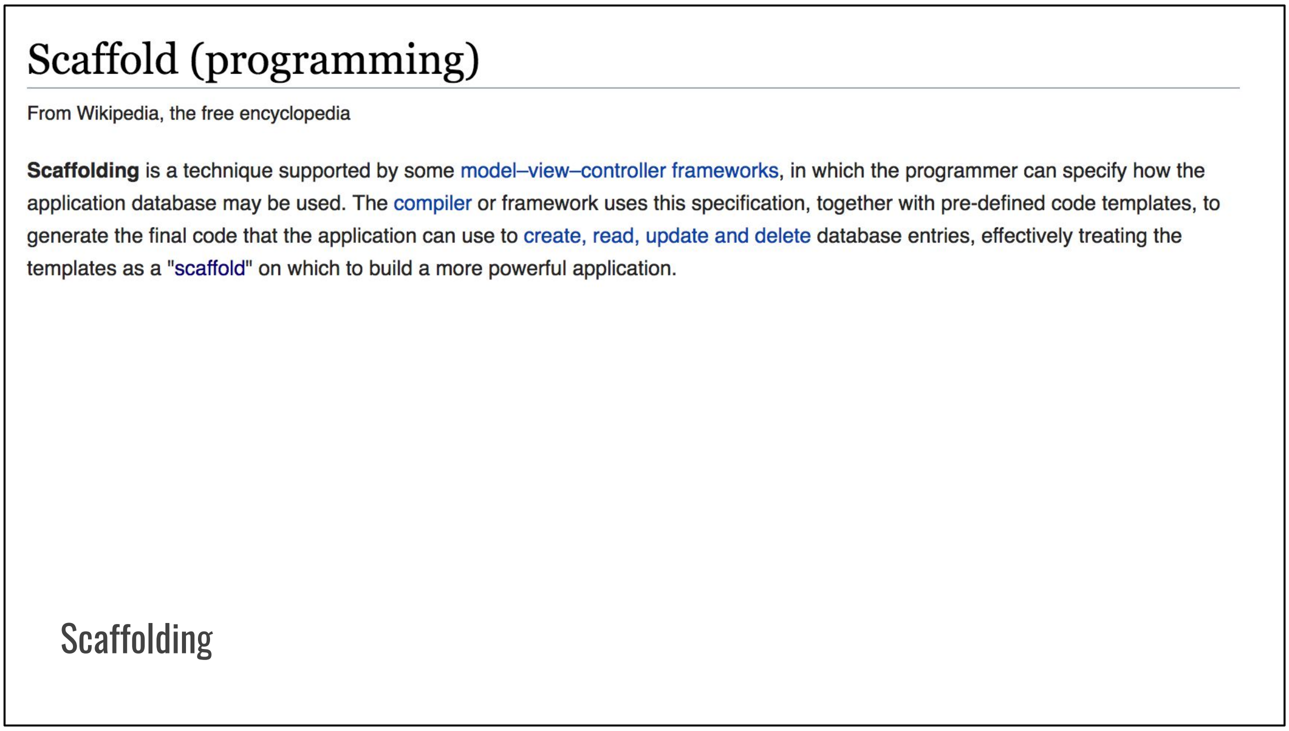
You may be looking at the url. This url matches - localhost:6442/login

When we browse this url, we took 404. Because I did not recompile the project. And also have not added the contoller and action yet.

That is the login route

So let’s create the controller. This time I rigth click on controllers, add and I am going to say add new controller.

Add Scaffold window opened.

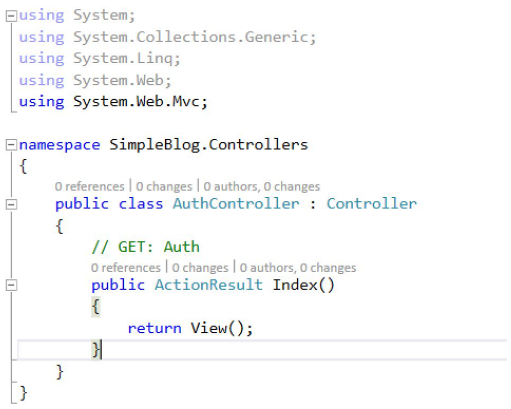

Scaffold (programming)

From Wikipedia, the free encyclopedia

Scaffolding is a technique supported by some model–view–controller frameworks, in which the programmer can specify how the application database may be used. The compiler or framework uses this specification, together with pre-defined code templates, to generate the final code that the application can use to create, read, update and delete database entries, effectively treating the templates as a "scaffold" on which to build a more powerful application.

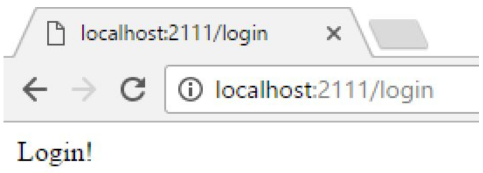
Scaffolding

Alternatively, I can add empty API controller, or I can add a controller with a bunch of staff for reading and writing data. We will never gonna use of these.
We always use Empty MVC controller.

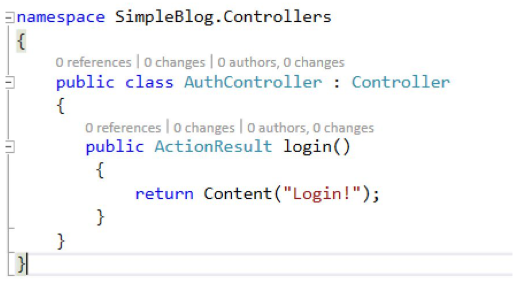


```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace SimpleBlog.Controllers
{
    0 references | 0 changes | 0 authors, 0 changes
    public class AuthController : Controller
    {
        // GET: Auth
        0 references | 0 changes | 0 authors, 0 changes
        public ActionResult Index()
        {
            return View();
        }
    }
}
```



```
localhost:2111/login
localhost:2111/login
Login!
```



```
namespace SimpleBlog.Controllers
{
    0 references | 0 changes | 0 authors, 0 changes
    public class AuthController : Controller
    {
        0 references | 0 changes | 0 authors, 0 changes
        public ActionResult login()
         {
            return Content("Login!");
         }
    }
}
```

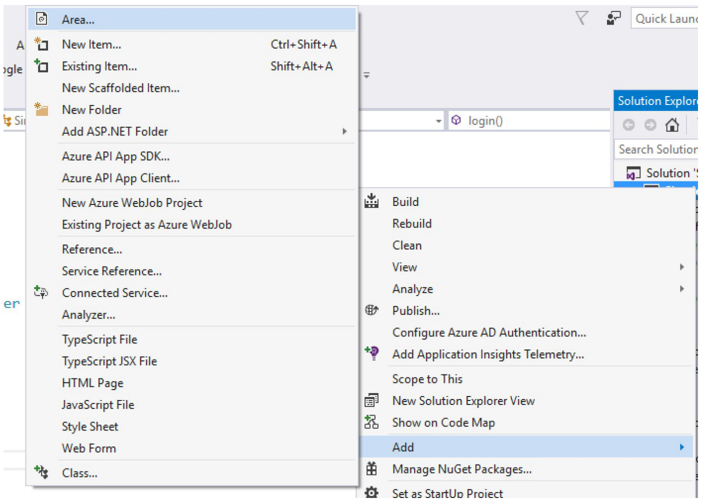
Auth Controller

What we get ?

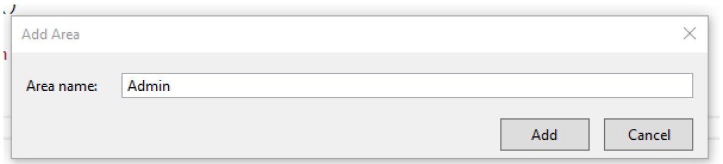
We get exactly what we did in PostsController.
We have a class inherits from Controller. We have a using statement up here.
And we have one action called Index automatically in our class with the return of view.

I am going to delete that Index action and write login action.

Our login action returns Login! text only.



Create an Area



Area name: Admin

Let’s go ahead and create and area. You can think area like a namespace.

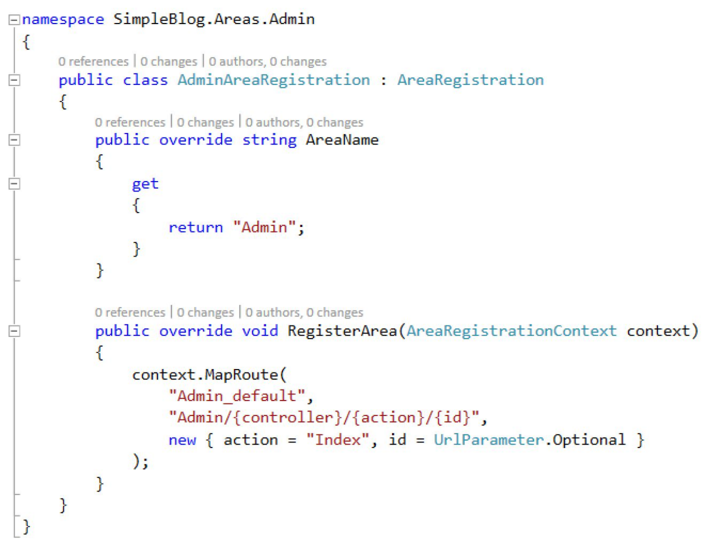
It is a way to well group its own collection of views, controllers and potentially models as well.

It is a way to group both on the file system and in the URL.

So if I right click on SimpleBlog, I can go to Add and I can say Area.

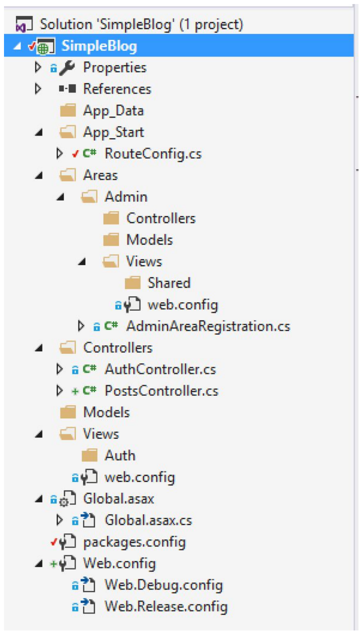
Then we must enter a name. The name of the Area gonna be Admin.

Because, this is going to be, this is gonna contain all the controllers that we use to add posts, modify posts and users. So I am goint to add this Area.



```
namespace SimpleBlog.Areas.Admin
{
    0 references | 0 changes | 0 authors, 0 changes
    public class AdminAreaRegistration : AreaRegistration
    {
        0 references | 0 changes | 0 authors, 0 changes
        public override string AreaName
        {
            get
            {
                return "Admin";
            }
        }

        0 references | 0 changes | 0 authors, 0 changes
        public override void RegisterArea(AreaRegistrationContext context)
        {
            context.MapRoute(
                "Admin_default",
                "Admin/{controller}/{action}/{id}",
                new { action = "Index", id = UrlParameter.Optional }
            );
        }
    }
}
```



```
Solution 'SimpleBlog' (1 project)
  SimpleBlog
    Properties
    References
    App_Data
    App_Start
      RouteConfig.cs
    Areas
      Admin
        Controllers
        Models
        Views
          Shared
          web.config
        AdminAreaRegistration.cs
    Controllers
      AuthController.cs
      PostsController.cs
    Models
    Views
      Auth
      web.config
    Global.asax
      Global.asax.cs
    packages.config
    Web.config
      Web.Debug.config
      Web.Release.config
```

After Adding Area

A couple of things happened.

First of all an Areas folder is appeared. Under this Admin folder. That corresponds to the name of the Area. Next step, we have another group of Controllers, Models, Views folder. And finally we have an AreaAdminRegistration.cs file.

Area Registrations will automatically be discovered by ASP.NET MVC. Effectively allow me to namespace a group of Controllers and Views into a special name such as Admin.

I can seperate all my admin code from my front end code.

If you think application as a House, area is the Room.

The first thing i will do about the Area is changing the AreName “Admin” to “admin”.

We see here RegisterArea method. This is where we register routes that are specific to this area.
I am actually leave this default here.

This happens to be very similar to the default route I deleted earlier out of our frontend, but in this case I am gonna leave it.

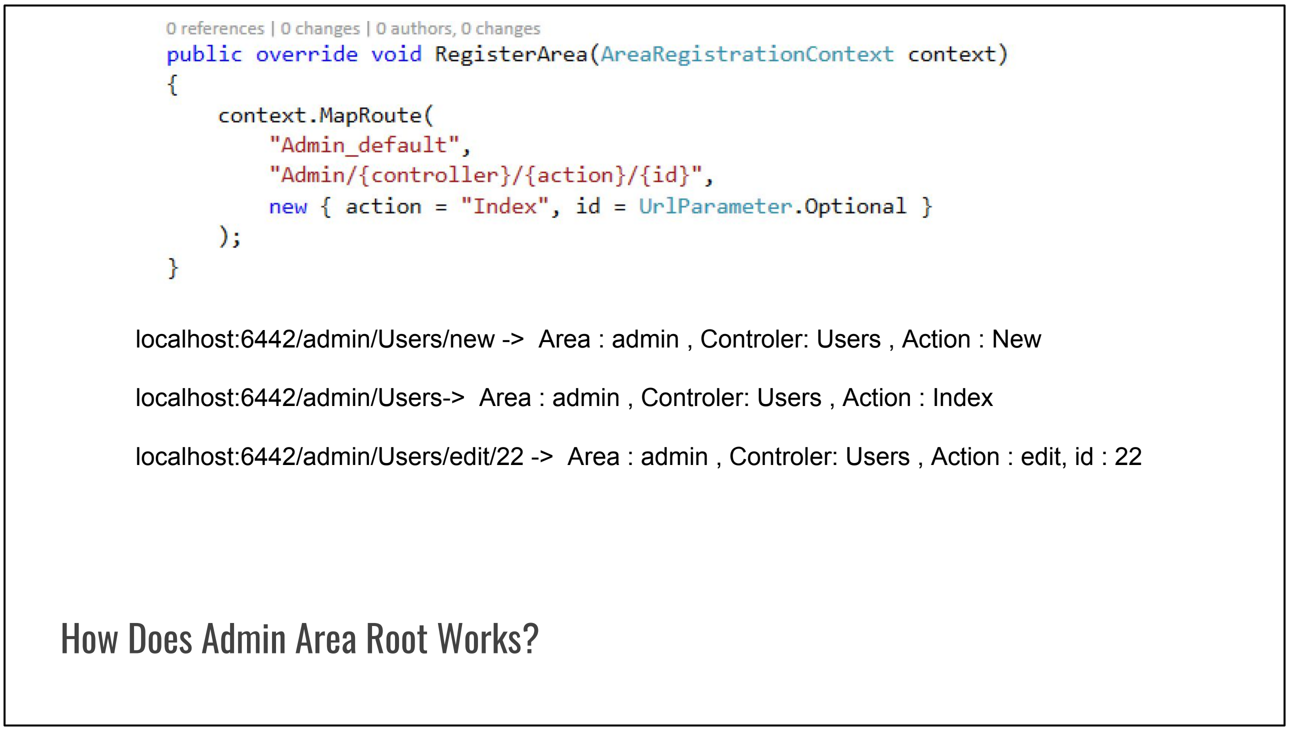
The interesthing thing about this route is that “Admin/” first of all. That comes from our area. The reason why our areas namespace this simply all the urls starts with admin.

Now we see some interesting syntax. “admin/{controller}/{action}/{id}”

These are programmeders. ASP.NET Routing Engine can extract from the URL, in order to create data that can be passed into our application.

But you know this, it is interesting. That we have a variable controller, variable action, and variable id. You also know defaults. action equals index.

So let’s how this root works.



```
0 references | 0 changes | 0 authors, 0 changes
public override void RegisterArea(AreaRegistrationContext context)
{
    context.MapRoute(
        "Admin_default",
        "Admin/{controller}/{action}/{id}",
        new { action = "Index", id = UrlParameter.Optional }
    );
}
```

localhost:6442/admin/Users/new -> Area : admin , Controler: Users , Action : New

localhost:6442/admin/Users-> Area : admin , Controler: Users , Action : Index

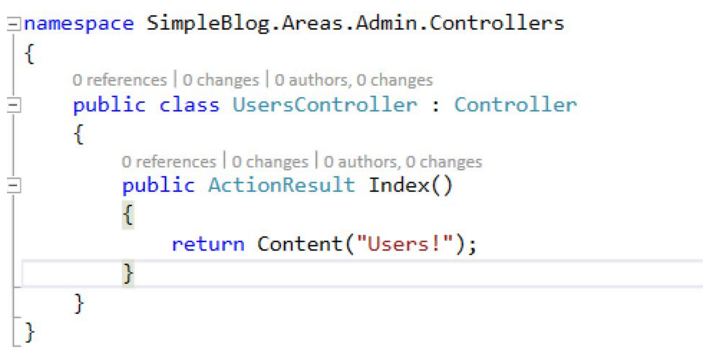
localhost:6442/admin/Users/edit/22 -> Area : admin , Controler: Users , Action : edit, id : 22

How Does Admin Area Root Works?

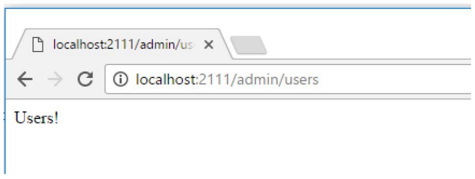
I want to show examples, how this root works. It matches values with pattern.

With /admin/users the defaults declared in route for action will choose the action as Index.

In the pattern, next step we have an id. ID is an arbitrary value that will passed to action as a parameter.



```
namespace SimpleBlog.Areas.Admin.Controllers
{
    0 references | 0 changes | 0 authors, 0 changes
    public class UsersController : Controller
    {
        0 references | 0 changes | 0 authors, 0 changes
        public ActionResult Index()
        {
            return Content("Users!");
        }
    }
}
```

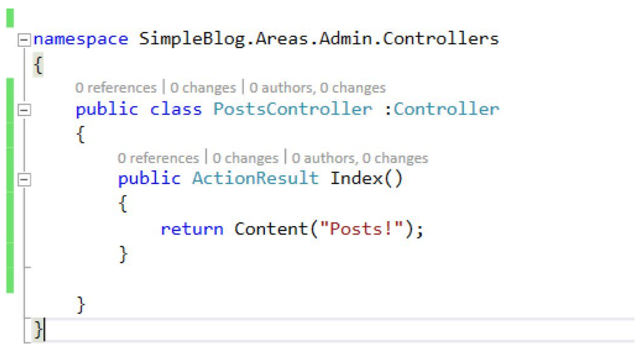


```
localhost:2111/admin/us
localhost:2111/admin/users
Users!
```

Create Users And Posts Controller In Admin Area.

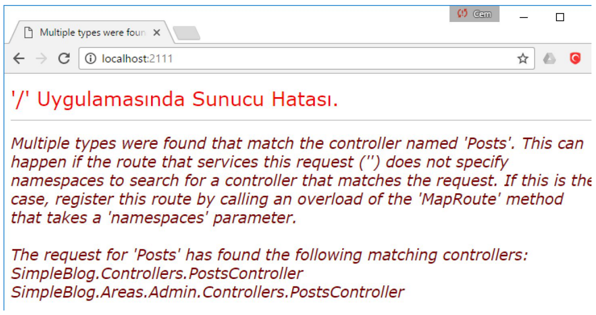
I add a UsersController in admin area. That can means, I can browse localhost:6442/admin/Users

The alternatively, what is the URL that can access this ? The first things is getting it from default, the second is explicitly get it. The alternative url is localhost:6442/admin/Users/Index



```
namespace SimpleBlog.Areas.Admin.Controllers
{
    0 references | 0 changes | 0 authors, 0 changes
    public class PostsController :Controller
    {
        0 references | 0 changes | 0 authors, 0 changes
        public ActionResult Index()
        {
            return Content("Posts!");
        }

    }
}
```



'/' Uygulamasında Sunucu Hatası.

Multiple types were found that match the controller named 'Posts'. This can happen if the route that services this request ('') does not specify namespaces to search for a controller that matches the request. If this is the case, register this route by calling an overload of the 'MapRoute' method that takes a 'namespaces' parameter.

The request for 'Posts' has found the following matching controllers:
SimpleBlog.Controllers.PostsController
SimpleBlog.Areas.Admin.Controllers.PostsController

We will create a new **Important** Controller - > PostsController

After adding this controller, we have a problem.

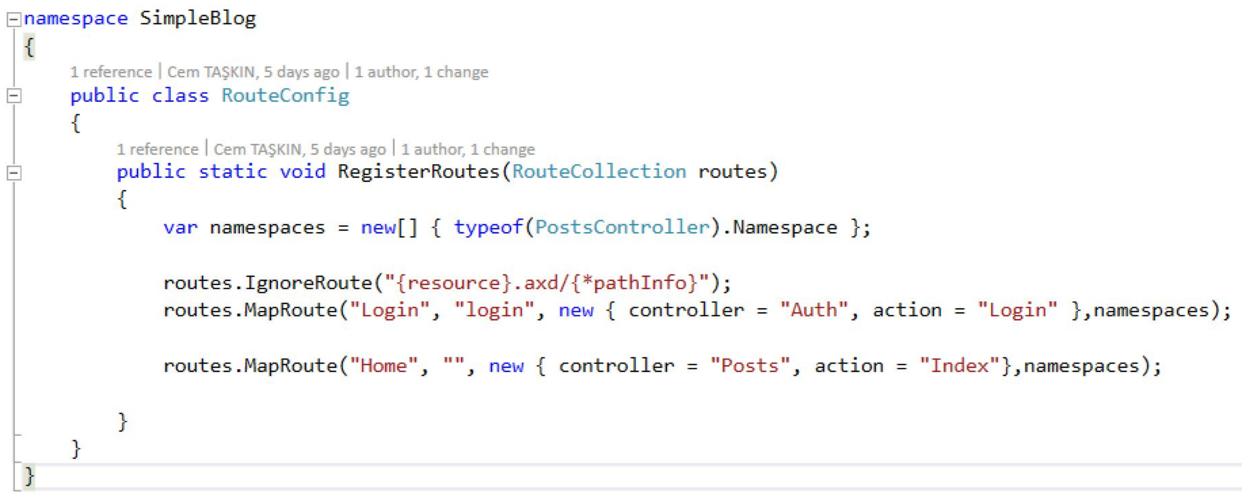
If we are try to go our home page,we see the problem.

Fortunately, ASP.NET is not good at error messages for the years.

I will go ahead and try to fix it. The problem is, you do not think of Controllers has having to exits in a particular folder. Instead, ASP.NET effectively flattens every single Controller it finds in our project into a single list. Regardless of its namespace.

What is that means.
It means, If we had two posts controllers, even if they are in different namespaces, different folders, or different areas, this will conflict ASP.NET have no idea which one you actually mean.

There is a very simple way to fix it. Build in the actual framework itself.



```
namespace SimpleBlog
{
    1 reference | Cem TAŞKIN, 5 days ago | 1 author, 1 change
    public class RouteConfig
    {
        1 reference | Cem TAŞKIN, 5 days ago | 1 author, 1 change
        public static void RegisterRoutes(RouteCollection routes)
        {
            var namespaces = new[] { typeof(PostsController).Namespace };

            routes.IgnoreRoute("{resource}.axd/{*pathInfo}");
            routes.MapRoute("Login", "login", new { controller = "Auth", action = "Login" },namespaces);

            routes.MapRoute("Home", "", new { controller = "Posts", action = "Index"},namespaces);

        }
    }
}
```

How to Solve Conflict? Answer is RouteConfig.

I am goint back to my RouteConfig file. I am goint to view again my MapRoute method.

In the signature of our method, has a string array namespaces [] argument.

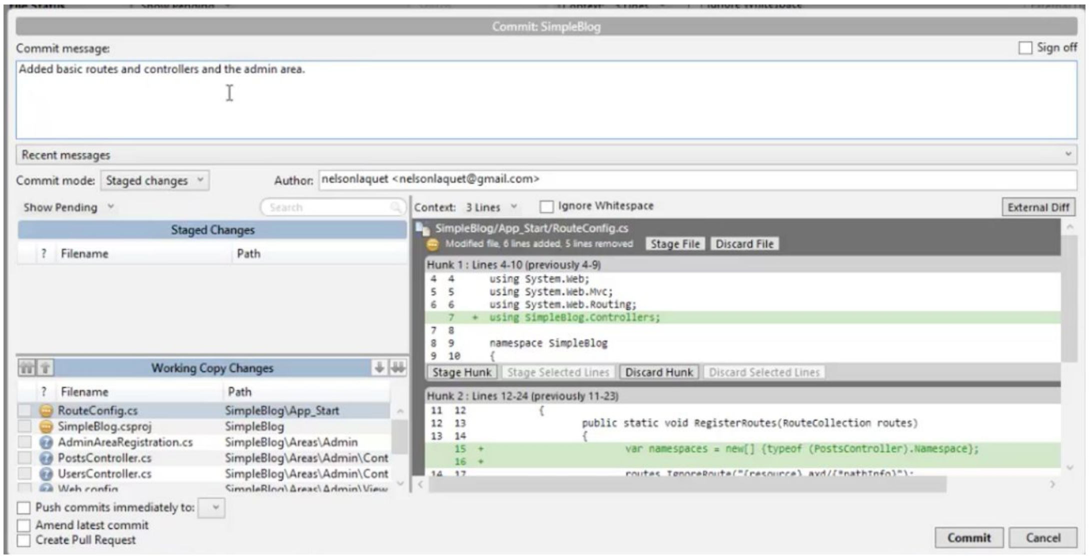
If we pass an array of strings representes namespaces, only this one route will go to, make ASP.NET MVC not confuse about which one of the Posts controllers were attending to go into.

If we go to /admin/posts we do not get the error. Because it is more specific to admin area. Because of that we see that properly works.

I create a new variable. Its name is namepsaces. It is an array and only consits an element. The element, that is has is the namespace of our PostsController, which is in Controller folder of the root folder. Not the Controller located in our Area.

Now, we pass this variable to our MapRoute method as a parameter for declaring namespaces.

So this is the solution of our conflict. ASP.NET is currently know which PostsContorller will reply the request.



Commit message:

```
Added basic routes and controllers and the admin area.
```

Recent messages

Commit mode: Staged changes

Author: nelsonlaquet <nelsonlaquet@gmail.com>

Show Pending

Context: 3 Lines   ☐ Ignore Whitespace

Staged Changes

?	Filename	Path

Working Copy Changes

?	Filename	Path
	RouteConfig.cs	SimpleBlog\App_Start
	SimpleBlog.csproj	SimpleBlog
	AdminAreaRegistration.cs	SimpleBlog\Areas\Admin
	PostsController.cs	SimpleBlog\Areas\Admin\Cont
	UsersController.cs	SimpleBlog\Areas\Admin\Cont
	Web.config	SimpleBlog\Areas\Admin\View

SimpleBlog/App_Start/RouteConfig.cs
Modified file, 6 lines added, 5 lines removed

```
Hunk 1 : Lines 4-10 (previously 4-9)
4  4       using System.Web;
5  5       using System.Web.Mvc;
6  6       using System.Web.Routing;
   7    +  using SimpleBlog.Controllers;
7  8
8  9       namespace SimpleBlog
9  10      {

Hunk 2 : Lines 12-24 (previously 11-23)
11  12              {
12  13                      public static void RegisterRoutes(RouteCollection routes)
13  14                      {
    15   +                          var namespaces = new[] {typeof (PostsController).Namespace};
    16   +
14  17                              routes.IgnoreRoute("{resource}.axd/{*pathInfo}");
```

☐ Push commits immediately to:
☐ Amend latest commit
☐ Create Pull Request

The last thing is about SourceTree

I will commit all the changes and my comment wil be

“Added basic route and controller and the admin area”