**Object-Oriented Design II**

SSE 554

Dr. MacNeil

Spring 2017

**Project II**

Brent Bitler

Matthew Robison

Table of Contents

[Table of Contents 2](#_Toc475792016)

[Table of Figures 4](#_Toc475792017)

[Introduction 5](#_Toc475792018)

[Purpose 5](#_Toc475792019)

[ASP.net MVC Breakdown 6](#_Toc475792020)

[Overview 6](#_Toc475792021)

[Quick Start App 7](#_Toc475792022)

[Overview 7](#_Toc475792023)

[App & Request Life Cycle 9](#_Toc475792024)

[Overview 9](#_Toc475792025)

[Application Life Cycle 9](#_Toc475792026)

[Request Life Cycle 9](#_Toc475792027)

[Routing & Controllers 11](#_Toc475792028)

[Overview 11](#_Toc475792029)

[Actions 14](#_Toc475792030)

[Filters 16](#_Toc475792031)

[Selectors 17](#_Toc475792032)

[ActionName 17](#_Toc475792033)

[NonAction 17](#_Toc475792034)

[ActionVerbs 18](#_Toc475792035)

[Views 20](#_Toc475792036)

[Application Creation 22](#_Toc475792037)

[Data Model 22](#_Toc475792038)

[Helpers 22](#_Toc475792039)

[Model Binding 22](#_Toc475792040)

[Databases 22](#_Toc475792041)

[Validation 22](#_Toc475792042)

[Caching 22](#_Toc475792043)

[Unit Tests 22](#_Toc475792044)

[Appendix 23](#_Toc475792045)

[HomeController.cs 23](#_Toc475792046)

[RouteConfig.cs 23](#_Toc475792047)

[ProcessController.cs 24](#_Toc475792048)

[UserController.cs 24](#_Toc475792049)

[AdminController.cs 25](#_Toc475792050)

[HomeController.cs (with Redirection) 25](#_Toc475792051)

[HomeController.cs (with GetCurrentTime) 25](#_Toc475792052)

[HomeController.cs (with Action and NonAction Selectors) 26](#_Toc475792053)

[UserController.cs (with ActionVerbs) 27](#_Toc475792054)

[ourView.cshtml 27](#_Toc475792055)

[Bibliography 28](#_Toc475792056)

[Activity Log - Bitler 29](#_Toc475792057)

[Activity Log - Robison 30](#_Toc475792058)

Table of Figures

Figure 1. MVC desgign flow 6

Figure 2. Initial debug with no code implemented 7

Figure 3. Initial default controller added with test output text 7

Figure 4. 404 error resolved 8

Figure 5. MVC Request Life Cycle flow chart 10

Figure 6. RouteConfig class with RegisterRoutes method 11

Figure 7. Home/Index provides same URL as before 12

Figure 8. Process page output 12

Figure 9. Global.asax Application\_Start method 13

Figure 10. User route added with search action 13

Figure 11. UserController class 13

Figure 12. Action functionality output 13

Figure 13. AdminController returns string of admins 14

Figure 14. Output of admin controller access 15

Figure 15. RedirectToAction method utilization 15

Figure 16. Two action filters applied for caching purposes 16

Figure 17. Current time cached for 10 seconds until page refresh 16

Figure 18. Current time utilizing ActionName 17

Figure 19. NonAction in use 18

Figure 20. Output of NonAction 18

Figure 21. 404 error when attempting to accessing timeAsString as an action 18

Figure 22. ActionVerb in use 19

Figure 23. ActionVerb output with HTTP Get 19

Figure 24. Action results to return a View 20

Figure 25. Server Error due to view not created 20

Figure 26. Successful creation of our first view 21

Introduction

## Purpose

For Project II, the purpose was to choose a topic new to us from the text, demonstrate our capabilities through software design, coding, and finally unit testing while using some form of source control. For Project II, we have decided to investigate ASP.net MVC, its functionality, and implementation by creating several mini sample webpages demonstrating our capabilities per the project requirements on the syllabus. This report will chronicle our design strategy and timeline. The features of the project requirements are outlined in the table below.

Table 1. Project II Composition

|  |  |
| --- | --- |
| Team Composition | Brent Bitler, Matthew Robison |
| Topic | ASP.net MVC |
| Distributed Version Control System | GitHub |
| Programming Language | C# |
| IDE | Visual Studio 2015 |
| TDD Tool | Built-in VS unit test tool |

ASP.net MVC Breakdown

## Overview

ASP.net MVC is a framework designed by Microsoft to facilitate website development, utilizing MVC (Model View Controller) methodologies. The benefit here lies in keeping tasks decoupled and provides a clean separation between the three ideologies. The design pattern of MVC is illustrated in figure 1. It is not a standalone venture, but rather was built on top of ASP.net so developers like us are still able to utilize pre-existing ASP.net functionality.

MODEL

CONTROLLER

VIEW

Figure 1. MVC desgign flow

The model is the collection of classes in control of the data aspect of the program. The view, as its name hints towards, helps manage the user experience through the UI (User Interface) and is in HTML (Hypertext Mark-up Language). The controller is the collection of classes that deals with user interaction and application conditionals and logic. The rest of this report will chronicle the implementation of an ASP.net MVC application from start to finish.

Quick Start App

## Overview

The first step of the application process is to utilize the Visual Studio IDE to begin creating our webpage. We began by selecting an empty ASP.net 4.5.2 template and addings references for MVC. This will provide us with a blank slate for development with minimal predefined content. Here is the starting point for our program after a quick launch. Obviously, we expect a connection error as we have no defined any code thus far.

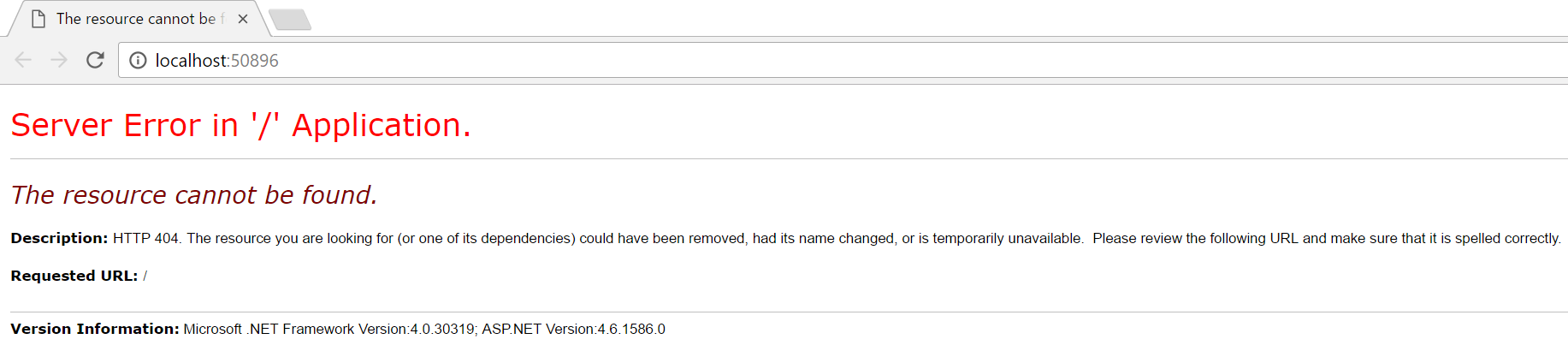


Figure 2. Initial debug with no code implemented

This will change shortly after adding a controller to the application as shown in figure 3 with a string test output.

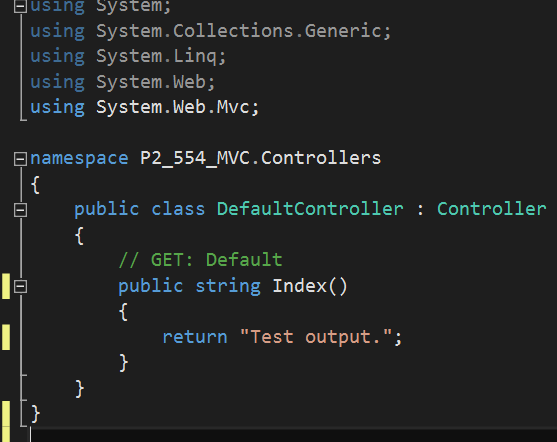


Figure 3. Initial default controller added with test output text

Now if we run the application we have an established connection with the resulting output shown in figure 4.

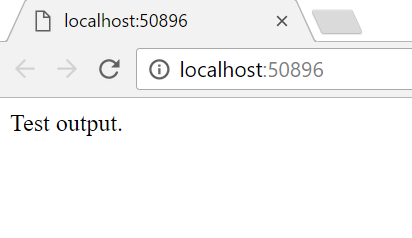


Figure 4. 404 error resolved

App & Request Life Cycle

## Overview

A life cycle, in general, is a grouping of steps are tasks utilized to handle some sort of request to change the state of an application. The input is a request while the output is a response. In general, MVC consists of two life cycles [1]:

* Application life cycle
* Request life cycle

## Application Life Cycle

The application life cycle is simple the time the application process begins running until the time it stops. Within this timeframe, numerous request life cycles are able to take place to facilitate the application life cycle.

## Request Life Cycle

The other half of this spectrum is the request life cycle, which is the series of events that occurs when an HTTP request is processed by the application. The request is then routed thorugh the URL Routing Module, which is a .NET component able to latch into the application. From here, the MVC framework converts the routing data into a controller able to process requests. At this point, an action invoker searches for and chooses the correct Action method to invoke the controller [1].

This can be seen as the rebound point for the flow diagram seen in figure 5, which fires off Result Execution. Depending on the type of output it will either be a View engine and subsequently render the view, otherwise the action result will execute on its own. This is what capitulates a real response to the initial HTTP request.

Routing

View Engine

Result Execution

Controller Initialization

Action Execution

Result Execution

RESP

REQ

Figure 5. MVC Request Life Cycle flow chart

Routing & Controllers

## Overview

This is the mechanism of tunneling an HTTP request to a controller, handled in the System.Web.Routing assembly. A default template is loading to an MVC project in the form of a Global.asax file. This file utilizes an implementation of the RouteConfig class, with a very useful RegisterRoutes method as seen in figure 6.

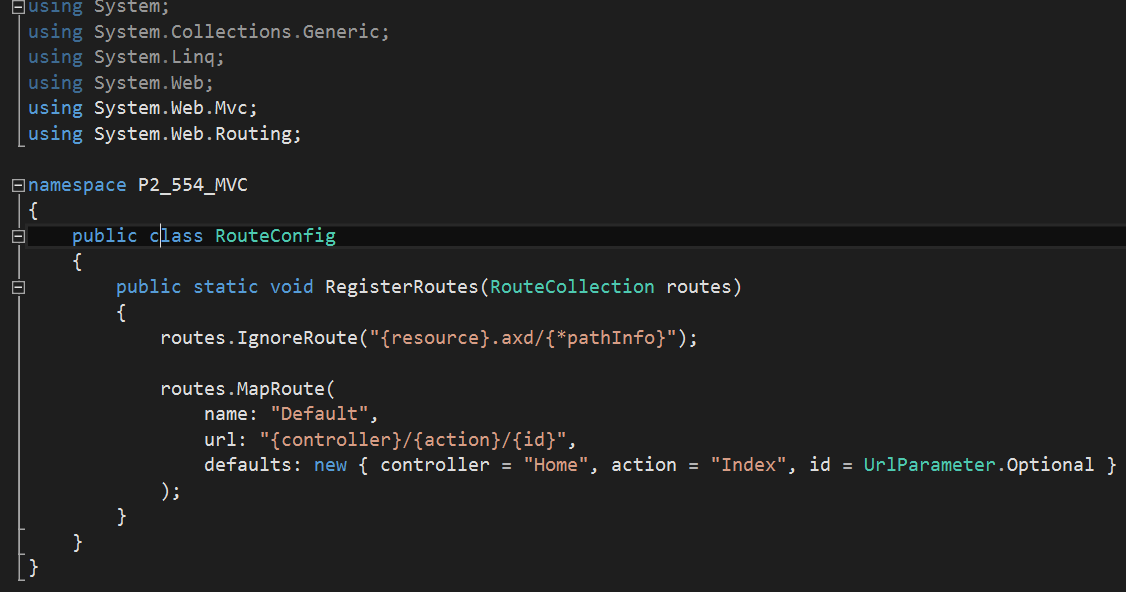


Figure 6. RouteConfig class with RegisterRoutes method

Routes will be defined by us, which then have URLs mapped to the designated controller action. URLs are defined by the notation {controller name}/{action name}/{id name}. The Visual Studio IDE generates a random port number for the application. In our instance, 50896 was utilized. Any time a browser requests <http://(ourwebpage)/Home> it returns the output from the Index method of HomeController. Utilizing the same URL with /Home or /Home/Index will yield the same output as before with this test output. This is shown in figure 7.

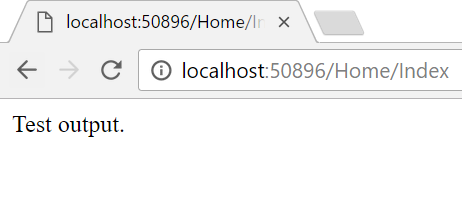


Figure 7. Home/Index provides same URL as before

Additionally, we can create a custom process controller with a custom convention as opposed to this predefined nomenclature by adding a new controller. The default action will be List instead of Index. The result from the ProcessController is displayed in figure 8.

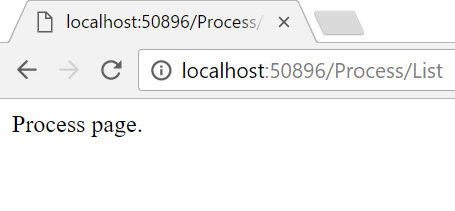


Figure 8. Process page output

Let us change some code to better explain controllers in more detail. Figure 9 shows the creation of a UserController with updated Global.asax code functionality while figure 10 illustrates the updated implemention of the RouteConfig class, so someone browsing can input the URL “User/Brent” with Brent as the parameter name and not an Action method. The UserController class reflects this functionality in figure 11. Because the parameter is added to the Action method, the MVC framework is built to search for the value matching the parameter name. The Server.HtmlEncode method converts any malicious code in plain text. The resulting code when run after requesting <http://localhost:50896/User/Brent> returns the result in figure 12.

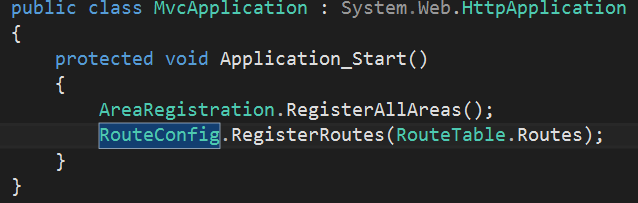


Figure 9. Global.asax Application\_Start method

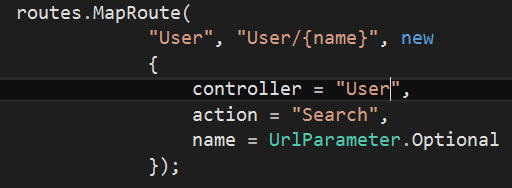


Figure 10. User route added with search action

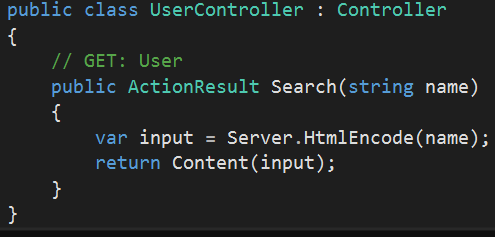


Figure 11. UserController class

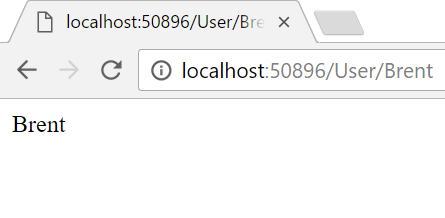


Figure 12. Action functionality output

Actions

Action methods are in charge of pushing requests creating responses to them in the form of ActionResults. A good note here is action methods have to be instantiated and cannot be static, but they can return anything such as a string, integer, Boolean, etc. When a URL is received, the UrlRoutingModule moves it along to the MVC route handler based on the information provided in the routing table. The MVC route handler then moves it along to the MVC handler, which is an HTTP handler. The MVC handler instantiates the controller by utilizing the RouteData for the controller value through virtue of the IController Interface. The method that executes creates an action invoker, which looks in the RouteData and locates the action parameter sent from the routing engine.

The UserController was already established previously. We will continue this mini example by creating an AdminController. Through adding the list of administrators to the Controller as seen in figure 13, by accessing the URL <http://localhost:50896/Admin/GetAllAdmins>, the method returns the list of admins as shown in figure 14. Furthermore, we are able to utilize the RedirectToAction method with two inputs action name and controller name to redirect from a controller to another as shown in figure 15, yielding the same output, again, as figure 14.

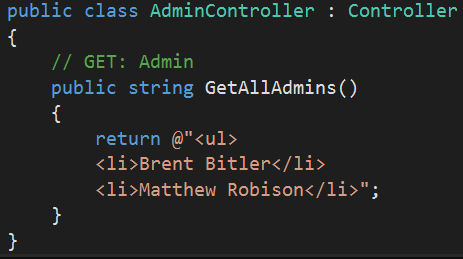


Figure 13. AdminController returns string of admins

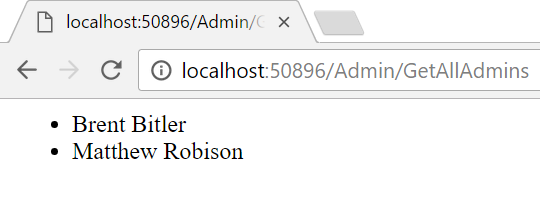


Figure 14. Output of admin controller access

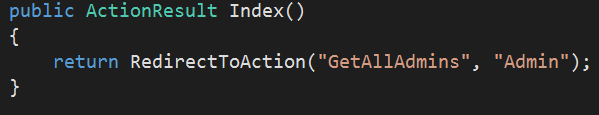


Figure 15. RedirectToAction method utilization

Filters

As we now know, controllers help define action methods with a one-to-one correspondence with user interactions; however, filters allow us as programmers to execute logic before and/or after an action method is executed. There are four unique types of filters and are executed based on the priority ordering below [2]:

1. Authorization filter – work with authentication and authorization
2. Action filter – wrap the action method execution
3. Result filter – wrap execution of the ActionResult object
4. Exception filter – executes if there is an unhandled exception thrown, useful for logging or displaying error pages

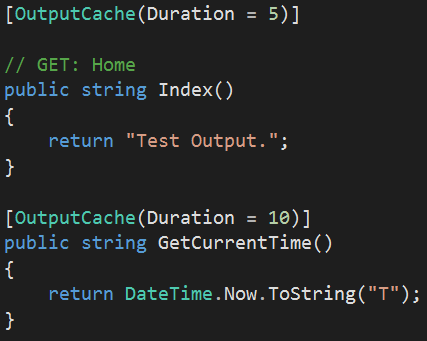
Here, we will implement two action filters. The first will be applied to the Index() action and will cache its return value for 5 seconds. By adding the GetCurrentTime method as seen in figure 16 with a duration of 10 seconds, when we navigate to <http://localhost:50896/Home/GetCurrentTime>, the value displayed will hold until 10 seconds have passed and the page is refreshed as shown in figure 17. 

Figure 16. Two action filters applied for caching purposes

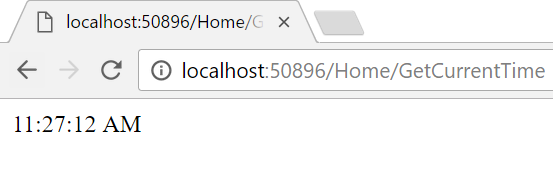


Figure 17. Current time cached for 10 seconds until page refresh

Selectors

Selectors help dictate what particular action method is invoked following a request and can be broken down into three categories:

1. ActionName
2. NonAction
3. ActionVerbs

## ActionName

The action name class allows us as software developers to utilize an action name different from the method name. We can continue utilizing the previous example to drive this point home by inserting the line [ActionName(“currTime”)] directly before the GetCurrentTime method. Doing so in coordination to navigating to the URL <http://localhost:50896/Home/currTime> will yield the following output page as shown in figure 18.

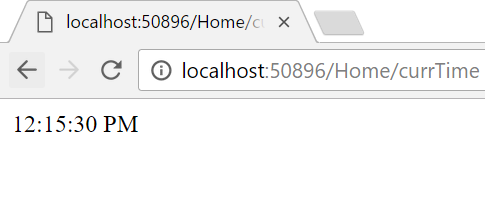


Figure 18. Current time utilizing ActionName

## NonAction

NonAction is fairly self-explanatory. This built in attribute is useful when one does not desire a method to be treated as if it were an action method and an example is shown in figure 19 with its output in figure 20 by navigating to <http://localhost:50896/Home/currTime> again. However, using timeAsString as an action will yield a 404 error as shown in figure 21.

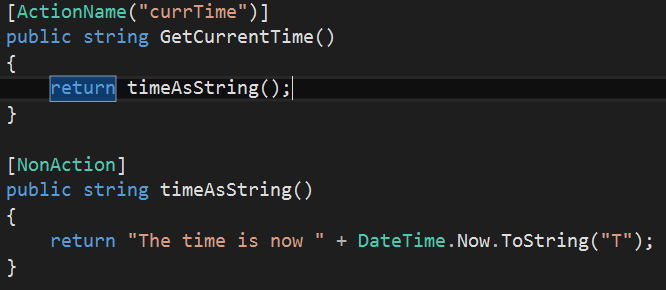


Figure 19. NonAction in use

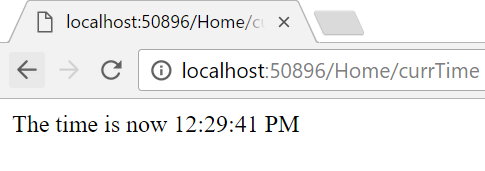


Figure 20. Output of NonAction

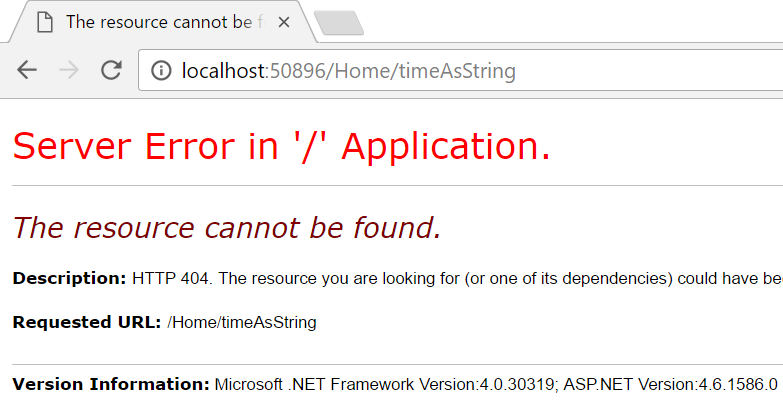


Figure 21. 404 error when attempting to accessing timeAsString as an action

## ActionVerbs

ActionVerbs help restrict a specific action to certain HttpVerbs. This allows us as developers to define two action methods with the same exact name, but one action will be responsible for responding to an HTTP Get requiest while the other will be responsible for respoding an HTTP Post request. This demonstrated with code in figure 22 and the corresponding output in figure 23 through updates to our previously defined UserController class.

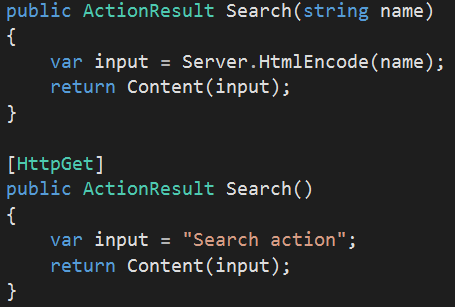


Figure 22. ActionVerb in use

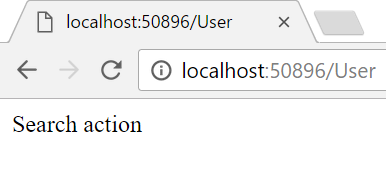


Figure 23. ActionVerb output with HTTP Get

Views

Views are the closest thing to a page in ASP.net MVC. When adding three action methods to our updated HomeController class as shown in figure 24 and running the application, we receive a much more descriptive error than before as seen in figure 25. The server error mentions how it is unable to find the ‘ourView’ view, so this must be added programmatically.

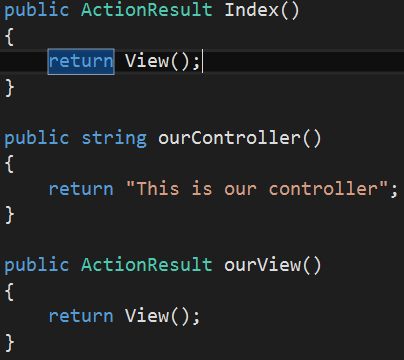


Figure 24. Action results to return a View

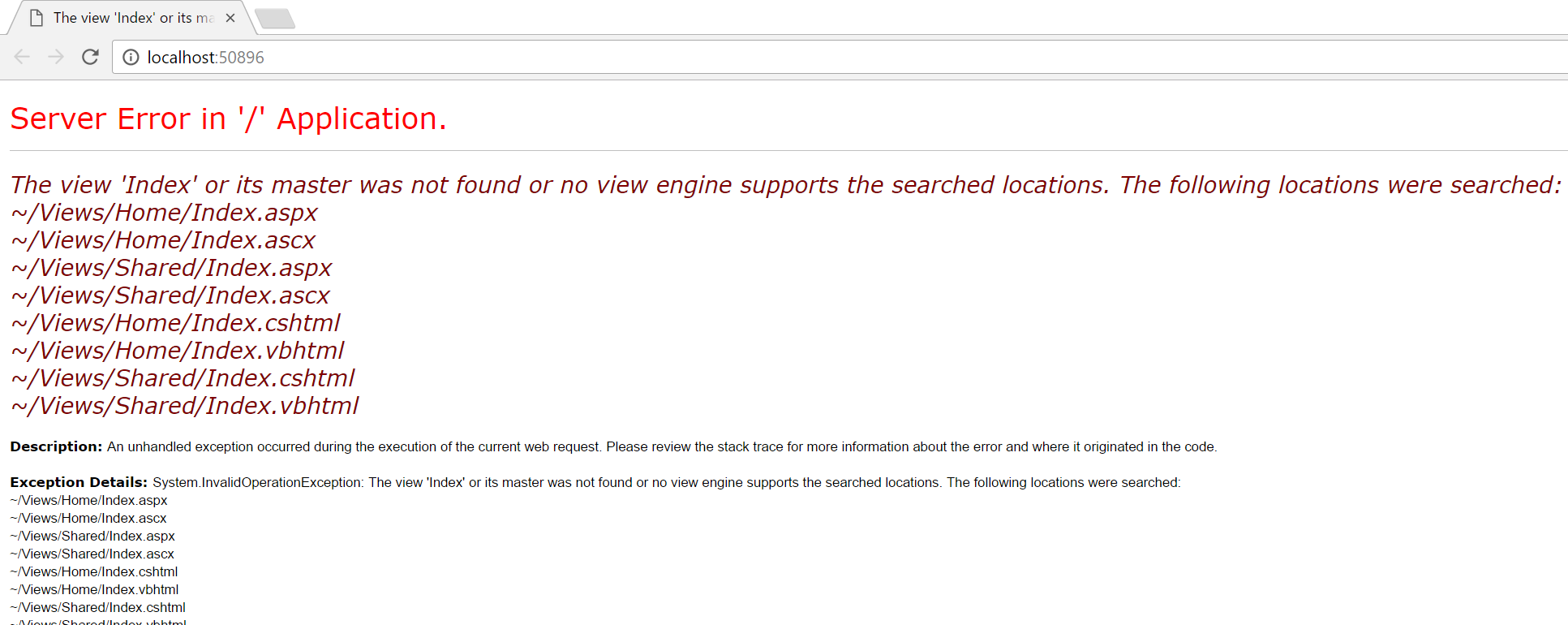


Figure 25. Server Error due to view not created

After creating our view and navigating to the same page, we are left with the output shown in figure 26, illustrating a successful creation of a view.

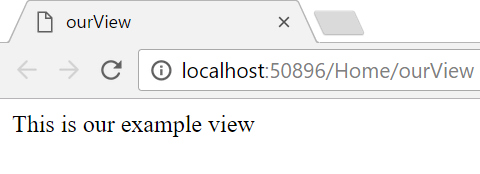


Figure 26. Successful creation of our first view

Application Creation

## Data Model

Now that we have gone over a myriad of the groundwork topics for working with MVC, we are ready to begin our full-fledged application. Here, we will begin looking at the inner workings of data models, which help store data that is gathered based on commands from a controller. We created a Driver controller to have a means for interacting with a database of rallycar drivers. This will initially have the capability to create a new driver, update their basic information such as age, ID number for the database, registration date and their name in case spelling mistakes are made during the creation of a new driver entry by the database manager.

Several ActionResults were put in place for the aforementioned features. Additionally we utilized the HttPost ActionVerb for the creation of a new driver entry as seen in figure 27. We also created a Driver class with getters and setters for their basic information as seen in figure 28. Finally, we created the GetDriverList method to return the list of four drivers and updated the index action method to iterate over the members of the Driver list in order of ID number and return the Drivers view. We added a view in the Driver model class. This code for this segment is rather lengthy and therefore left in the appendix. This HTML code is what sets up the page’s formatting.

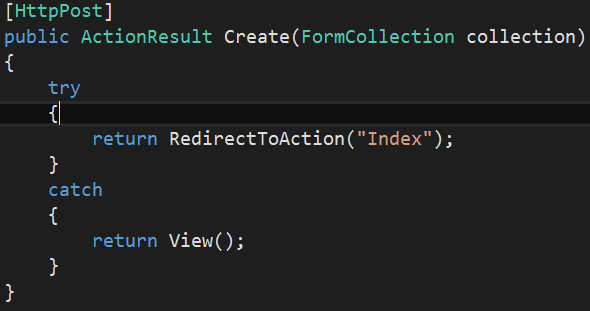


Figure 27. Create ActionResult

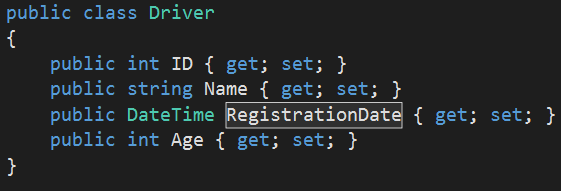


Figure 28. Driver class

## Helpers

HTML is extremely prevalent when it comes to working with views in any capacity. Thankfully, ASP.net has a built in HtmlHelper class to provide developers like us the capability to create HTML controls programmatically, which are generated at runtime. We’ve set up the Edit action to setup the list of drivers and return the view. After creating the view for this action, several helper methods are utilized from HTML.BeginForm to write an opening Form Tag and makes sure the method is going to be “Post” when Save is selected. This helps keep the code legible and functional. These methods can again be seen in view’s HTML code located in the appendix of this report.

## Model Binding

Model binding is the means of transitioning from the View’s values to the Model class once the Action method has been executed in the controller class. The model binder is what handles this. Post and Get utilize default binders to achieve this functionality behind the scenes. By adding a view to the Create method and utilizing the Create template, when the user inputs values on this view, the information is accessible in FormCollection.

The Create method is shown in figure 29. We are retrieving the values posted by the HTML view and sending them to the Driver properties by assignment. Type casting is handled when formats are not completely aligned, through a process known as manual binding. The beauty of the HTML views is when they are generated, proper names are also generated based on the property names in the Driver Model so the model binding works flawlessly. This effort is illustrated by the line of code in figure 30. The resulting capability of creating a new driver is shown in figure 31.

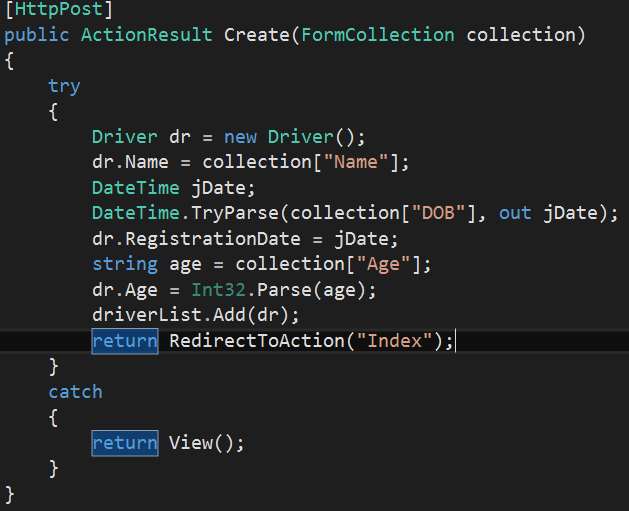


Figure 29. Create method with model binding



Figure 30. Model binding in action

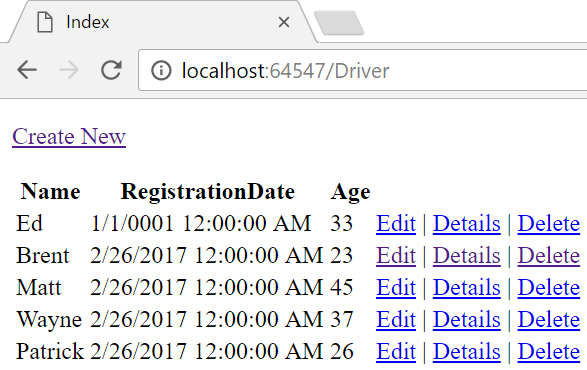


Figure 31. Create method successfully executed with model binding

## Databases

Instead of relying on hard coding member’s information in, the next step is to transition to utilizing a database backend to apply such robust changes.

## Validation

## Caching

## Unit Tests

Appendix

## HomeController.cs

namespace P2\_554\_MVC.Controllers

{

public class HomeController : Controller

{

// GET: Home

public string Index()

{

return "Test Output.";

}

}

}

## RouteConfig.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

using System.Web.Routing;

namespace P2\_554\_MVC

{

public class RouteConfig

{

public static void RegisterRoutes(RouteCollection routes)

{

routes.IgnoreRoute("{resource}.axd/{\*pathInfo}");

routes.MapRoute(

"Process", "Process/{action}/{id}",

defaults: new

{

controller = "Process",

action = "List",

id =

UrlParameter.Optional

});

routes.MapRoute(

"User", "User/{name}", new

{

controller = "User",

action = "Search",

name = UrlParameter.Optional

});

routes.MapRoute(

name: "Default", url: "{controller}/{action}/{id}", defaults: new

{

controller = "Home",

action = "Index",

id = UrlParameter.Optional

});

}

}

}

## ProcessController.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

namespace P2\_554\_MVC.Controllers

{

public class ProcessController : Controller

{

// GET: Process

public string List()

{

return "Process page.";

}

}

}

## UserController.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

namespace P2\_554\_MVC.Controllers

{

public class UserController : Controller

{

// GET: User

public ActionResult Search(string name)

{

var input = Server.HtmlEncode(name);

return Content(input);

}

}

}

## AdminController.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

namespace P2\_554\_MVC.Controllers

{

public class AdminController : Controller

{

// GET: Admin

public string GetAllAdmins()

{

return @"<ul>

<li>Brent Bitler</li>

<li>Matthew Robison</li>";

}

}

}

## HomeController.cs (with Redirection)

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

namespace P2\_554\_MVC.Controllers

{

public class HomeController : Controller

{

// GET: Home

public ActionResult Index()

{

return RedirectToAction("GetAllAdmins", "Admin");

}

}

}

## HomeController.cs (with GetCurrentTime)

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

namespace P2\_554\_MVC.Controllers

{

public class HomeController : Controller

{

[OutputCache(Duration = 5)]

// GET: Home

public string Index()

{

return "Test Output.";

}

[OutputCache(Duration = 10)]

public string GetCurrentTime()

{

return DateTime.Now.ToString("T");

}

}

}

## HomeController.cs (with Action and NonAction Selectors)

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

namespace P2\_554\_MVC.Controllers

{

public class HomeController : Controller

{

// GET: Home

public string Index()

{

return "Test Output.";

}

[ActionName("currTime")]

public string GetCurrentTime()

{

return timeAsString();

}

[NonAction]

public string timeAsString()

{

return "The time is now " + DateTime.Now.ToString("T");

}

}

}

## UserController.cs (with ActionVerbs)

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

namespace P2\_554\_MVC.Controllers

{

public class UserController : Controller

{

// GET: User

public ActionResult Search(string name)

{

var input = Server.HtmlEncode(name);

return Content(input);

}

[HttpGet]

public ActionResult Search()

{

var input = "Search action";

return Content(input);

}

}

}

## ourView.cshtml

@{

Layout = null;

}

<!DOCTYPE html>

<html>

<head>

<meta name="viewport" content="width=device-width" />

<title>ourView</title>

</head>

<body>

<div>

This is our example view

</div>

</body>

</html>

Bibliography

[1] Engelbrecht, Andries P. *Computational Intelligence: An Introduction, (2nd ed.)*. Wiley, 2007.

[2] https://msdn.microsoft.com/en-us/library/gg416513(VS.98).aspx

Activity Log - Bitler

|  |  |  |
| --- | --- | --- |
| Date | Time (mins) | Description |
| 1/9/17 | 60 | Looked through Naïve GA |
| 1/10/17 | 60 | Looked through and worked Naïve GA |
| 1/11/17 |  |  |
| 1/12/17 | 120 | Research GA’s online for topic ideas |
| 1/13/17 | 180 | Read Ch 9 - GA |
| 1/14/17 | 90 | Read Ch 9 - GA |
| 1/15/17 |  |  |
| 1/16/17 | 120 | Begin constructing class structure of string unscramble program |
| 1/17/17 | 120 | Begin chromosome class |
| 1/18/17 | 120 | Continue working chromosome class, begin developing population class |
| 1/19/17 |  |  |
| 1/20/17 |  |  |
| 1/21/17 | 60 | Finish chromosome class |
| 1/22/17 | 240 | Continue working population class |
| 1/23/17 | 120 | Connector class |
| 1/24/17 | 30 | Come up with initial variable settings |
| 1/25/17 |  |  |
| 1/26/17 | 240 | Begin generating report |
| 1/27/17 | 180 | Put together test data |
| 1/28/17 | 360 | Conclude paper, graphs, etc |
| 1/29/17 | 180 | Final review of paper |
| 1/30/17 | 15 | Submit paper |
|  |  |  |
| TOTAL | 2295 |  |

Activity Log - Robison