**Object-Oriented Design II**

SSE 554

Dr. MacNeil

Spring 2017

**Project II**

Brent Bitler

Matthew Robison

Table of Contents

[Table of Contents 2](#_Toc475742539)

[Table of Figures 4](#_Toc475742540)

[Introduction 5](#_Toc475742541)

[Purpose 5](#_Toc475742542)

[ASP.net MVC Breakdown 6](#_Toc475742543)

[Overview 6](#_Toc475742544)

[Quick Start App 7](#_Toc475742545)

[Overview 7](#_Toc475742546)

[App & Request Life Cycle 9](#_Toc475742547)

[Overview 9](#_Toc475742548)

[Application Life Cycle 9](#_Toc475742549)

[Request Life Cycle 9](#_Toc475742550)

[Routing & Controllers 11](#_Toc475742551)

[Overview 11](#_Toc475742552)

[Actions 15](#_Toc475742553)

[Filters 17](#_Toc475742554)

[Selectors 18](#_Toc475742555)

[Views 19](#_Toc475742556)

[Application Creation 20](#_Toc475742557)

[Data Model 20](#_Toc475742558)

[Helpers 20](#_Toc475742559)

[Model Binding 20](#_Toc475742560)

[Databases 20](#_Toc475742561)

[Validation 20](#_Toc475742562)

[Security 20](#_Toc475742563)

[Caching 20](#_Toc475742564)

[Unit Tests 20](#_Toc475742565)

[Appendix 21](#_Toc475742566)

[Bibliography 22](#_Toc475742567)

[Activity Log - Bitler 23](#_Toc475742568)

[Activity Log - Robison 24](#_Toc475742569)

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 14

Figure 13. AdminController returns string of admins 15

Figure 14. Output of admin controller access 16

Figure 15. RedirectToAction method utilization 16

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 . 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 . 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.

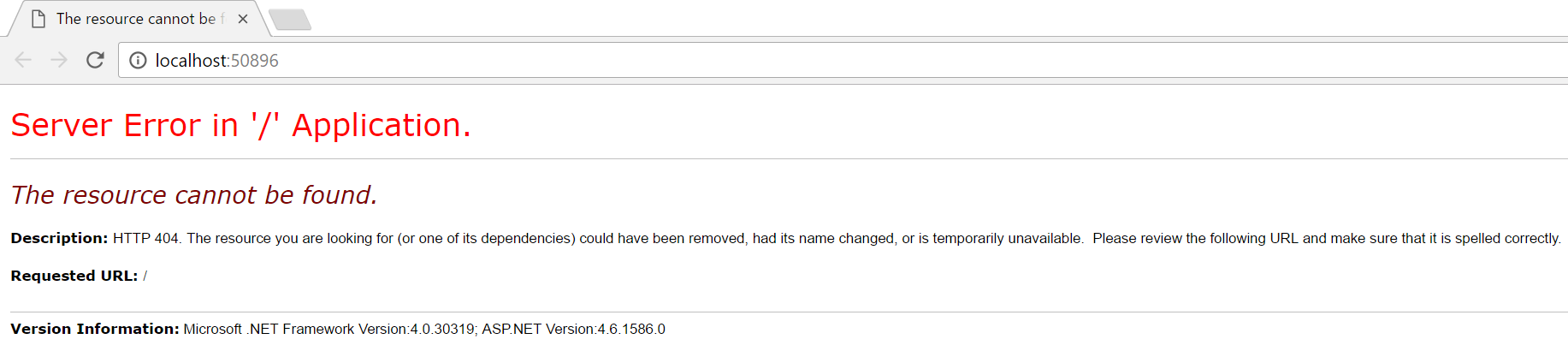


Figure . Initial debug with no code implemented

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

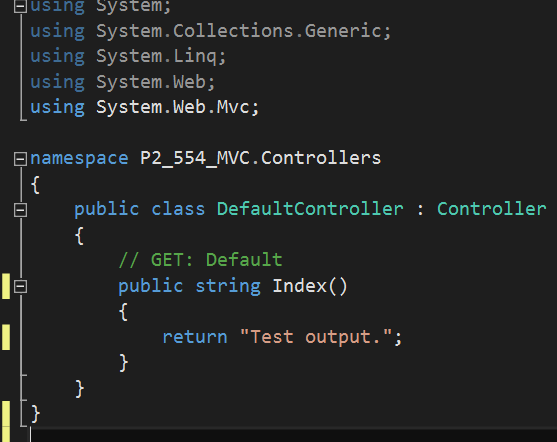


Figure . Initial default controller added with test output text

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

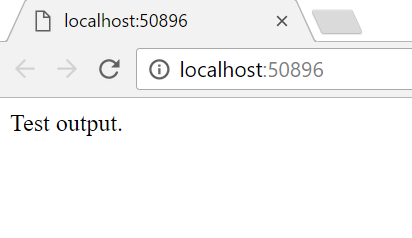


Figure . 404 error resolved

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:

* 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.

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 . 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.

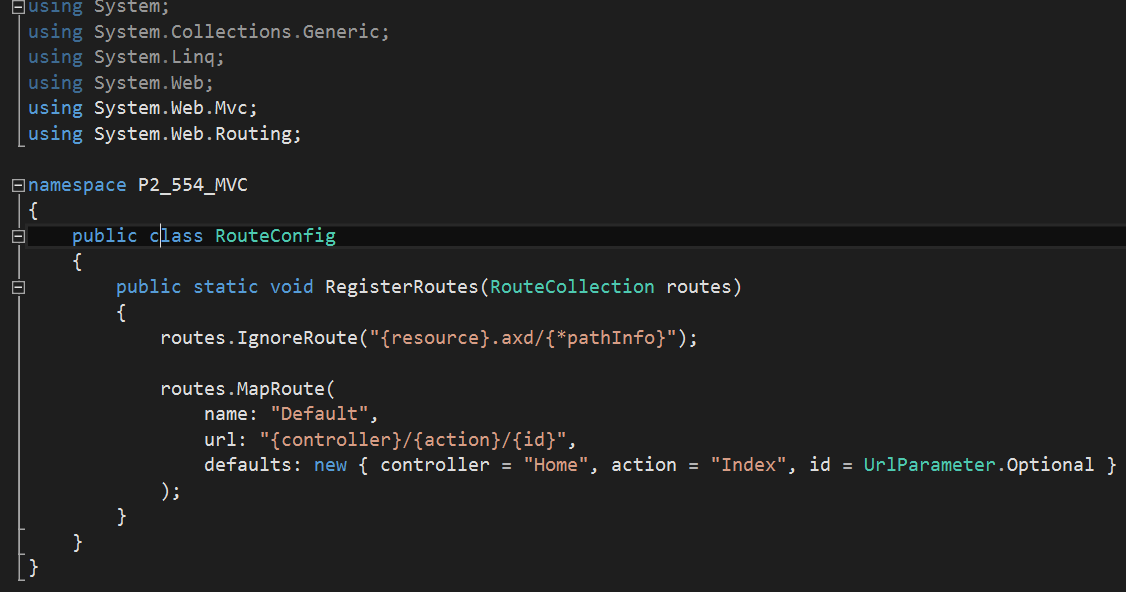


Figure . RouteConfig class with RegisterRoutes method

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.

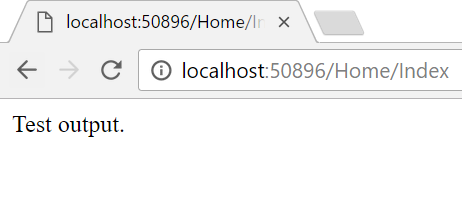


Figure . Home/Index provides same URL as before

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 resul from the ProcessController is displayed in figure 8.

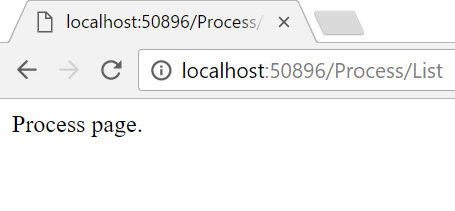


Figure . Process page output

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.

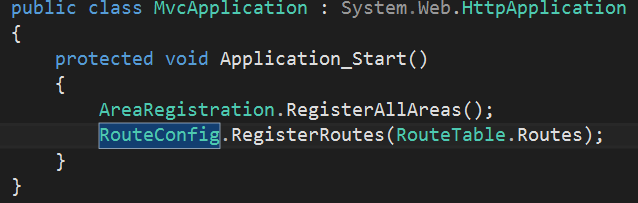


Figure 9. Global.asax Application\_Start method

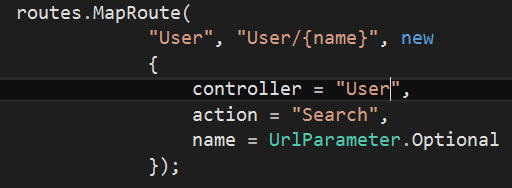


Figure 10. User route added with search action

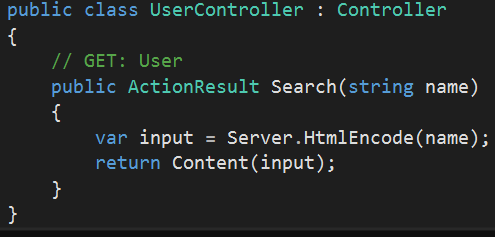


Figure 11. UserController class

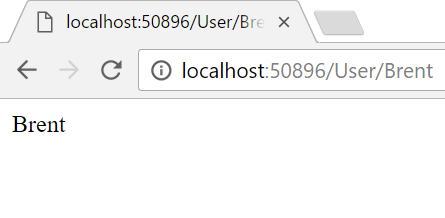


Figure 12. Action functionality output

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.

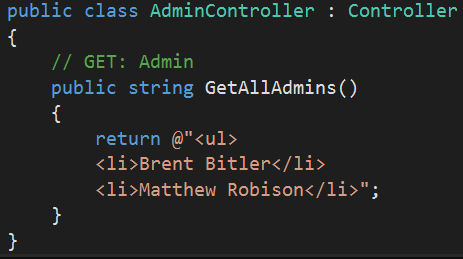


Figure . AdminController returns string of admins

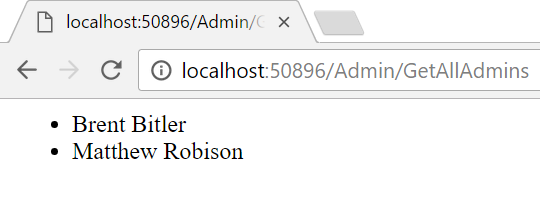


Figure . Output of admin controller access

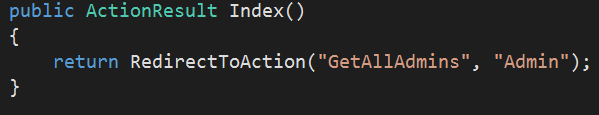


Figure . RedirectToAction method utilization

Filters

Selectors

Views

Application Creation

## Data Model

## Helpers

## Model Binding

## Databases

## Validation

## Security

## Caching

## Unit Tests

Appendix

Bibliography

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

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