C# web pages

and C# database programming

Rich Hildred for

PROG1760 #1 (12F)

Advanced Programming w Visual Studio C#.NET

Study Notes

Introduction

C# web programming can be performed in 3 basic ways:

1. Web Pages (with Razor syntax)
2. MVC (Model View Controller)
3. Web Forms (traditional ASP.NET)

These notes are for Web Pages (with Razor syntax). Instead of a .html file these web pages are contained in a .cshtml file that is a mixture of regular html and C# code that is turned into html on a web server running IIS and the .NET framework or Apache and Mono. This method of web programming is basic html5 css3 and javascript with one extra ingredient. The extra ingredient is the ability to have C# code mixed in with html that is run at the server to produce an html5 response that browsers can render.

In order to edit and test .cshtml pages in Visual Studio either Visual Studio 2012 must be used, or the Microsoft Web Platform Installer can be used to add MVC3 to Visual Studio 2010. Unfortunately Visual Studio 2008 can not be used as it works with the .NET 2.x framework, and .cshtml is dependent on the .NET 4.x framework. One can get the Microsoft Web Platform installer at: http://www.microsoft.com/web/downloads/platform.aspx.

A .cshtml page is different from a .html page in one way. A .cshtml page has C# code that is preceded by an "@" symbol. The code can be a single expression like "@DateTime.Now" in Figure 1

```
<!DOCTYPE html>
<html>
<head>
    <title>Date Time example</title>
</head>
<body>
    <h1>
        Date and Time</h1>
    <p>
        The time now is @DateTime.Now</p>
</body>
</html>
```

Figure 1

Or the C# code can be placed within a block as in Figure 2:

```
@{
// enter some C# code here
}
```

Figure 2

There is a longer example of C# code in a block in Figure 9.

To Start a .cshtml page

1) If you already have a web site open it if need be by going to File/Open/Website and selecting the root directory of the site and proceed to step 6.
2) In Visual Studio go to File/New/Website.
3) In the New Web Site dialog, select Visual C# and then select ASP.net empty web site Visual C#.
4) Name your web site in the web location panel at the bottom of the dialog
5) Press ok and in your solution explorer a project should appear with a web.config file in it

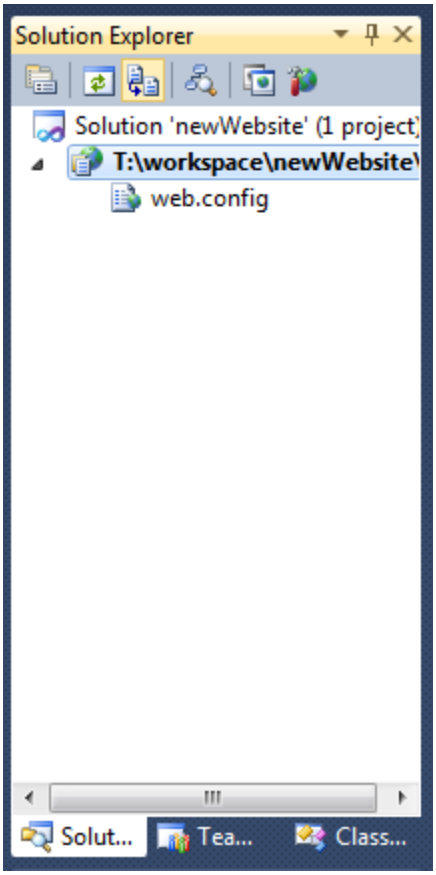


Figure 3

6) On the project (second line in figure 3) click the right mouse button and in the context menu select Add New Item
7) In the Add New Item dialog select Web Page(Razor)
8) In the name panel at the bottom of the Add New Item dialog name your file something other than WebPage.cshtml

The ability to place C# code in a web page can be useful in itself for tasks like managing a catalogue, or some sort of data visualization task, but to write a web application it is also necessary to be able to act on data input from the internet.

Input From the Internet

To understand how to get input from the Internet we need to understand a little about the http protocol. The http protocol is made up of requests (blue lines) from web browsers and responses from web servers (purple lines). The requests and responses are streams of text data that flow over the same TCP/IP connection. The requests and responses consist of specific headers that describe content in a body that follows the headers and 1 blank line.

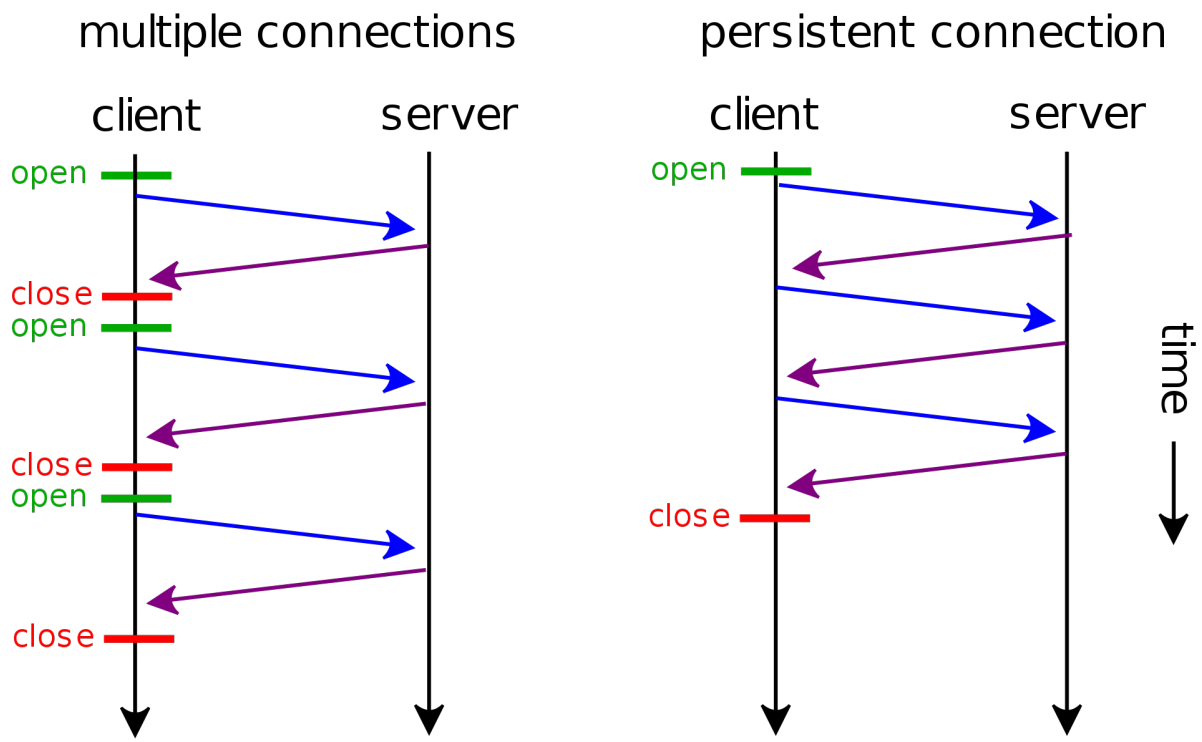


Figure 4[i]

This is an http GET request (blue line in Figure 4)

```
GET /webInput/add2Numbers.cshtml HTTP/1.1
Host: localhost:50401
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:20.0) Gecko/20100101
Firefox/20.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Cache-Control: max-age=0
```

Figure 5

This is the http response to the same request (purple line in Figure 4):

```
HTTP/1.1 200 OK
Server: ASP.NET Development Server/10.0.0.0
Date: Mon, 29 Apr 2013 00:40:06 GMT
X-AspNet-Version: 4.0.30319
X-AspNetWebPages-Version: 1.0
X-SourceFiles: =?UTF-
8?B?VDpcd29ya3NwYWNlXHdlYklucHV0XGFkZDJOdW1iZXJzLmNzaHRtbA==?=
Cache-Control: private
Content-Type: text/html; charset=utf-8
Content-Length: 569
Connection: Close

<!DOCTYPE html>
<html>
<head>
    <title>Add 2 Numbers</title>
</head>
<body>
    <h1>
        Add 2 Numbers</h1>
    <form action="" method="post">
        <p>
            <label for="textInput1">
                Enter First Number</label><br />
            <input type="text" name="textInput1" value="" /></p>
        <p>
            <label for="textInput2">
                Enter Second Number</label><br />
            <input type="text" name="textInput2" value="" /></p>
        <input type="submit" value="Add" />
    </form>
</body>
</html>
```

Figure 6

The World Wide Web is built on these requests and responses. When the browser has rendered the form described in the html that is the body of the response in Figure 6 it will look something like the image in Figure 7.

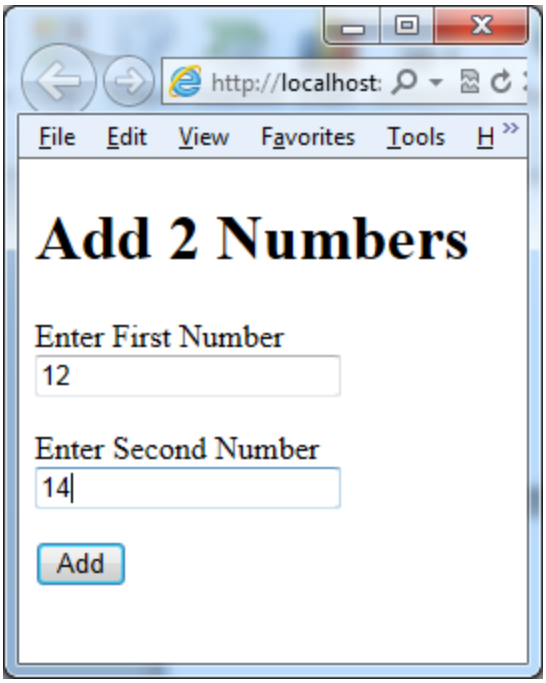


Figure 7

When the user clicks on the add button in their browser, a new post request is generated. Notice the numbers 12 and 14 in the browser and in the request. In the request 12 and 14 are the values of name-value pairs that are named by the input name attributes in Figure 6.

```
POST /webInput/add2Numbers.cshtml HTTP/1.1
Host: localhost:50401
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:20.0) Gecko/20100101 Firefox/20.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://localhost:50401/webInput/add2Numbers.cshtml
Connection: keep-alive
Content-Length:  27
Content-Type:  application/x-www-form-urlencoded

textInput1=12&textInput2=14
```

Figure 8

It is up to our .cshtml page to produce the response body for both the GET and the POST requests. In our C# code the textInput1 and textInput2 form values are contained in Request[“textInput1”] and Request[“textInput2”] respectively. In the case of the first get request there is no form input, so we must test those fields to see if they are null. In the 2nd post request the data is filled in so we can display the sum of the results like:

```
<p>
            Sum of @nInput1 + @nInput2 = @nResult</p>
```

The complete page is listed in Figure 9.

```
<!DOCTYPE html>
<html>
<head>
    <title>Add 2 Numbers</title>
</head>
<body>
    <h1>
        Add 2 Numbers</h1>
    <form action="" method="post">
    @{
        try
        {
            String sTextInput1 = "";
            if (Request["textInput1"] != null)
            {
                sTextInput1 = Request["textInput1"];
            }
            String sTextInput2 = "";
            if (Request["textInput2"] != null)
            {
                sTextInput2 = Request["textInput2"];
            }
        <p>
            <label for="textInput1">
                Enter First Number</label><br />
            <input type="text" name="textInput1" value="@sTextInput1" /></p>
        <p>
            <label for="textInput2">
                Enter Second Number</label><br />
            <input type="text" name="textInput2" value="@sTextInput2" /></p>
        <input type="submit" value="Add" />
            if (sTextInput1 != "" && sTextInput2 != "")
            {
                Decimal nInput1 = Decimal.Parse(sTextInput1);
                Decimal nInput2 = Decimal.Parse(sTextInput2);
                Decimal nResult = nInput1 + nInput2;
        <p>
            Sum of @nInput1 + @nInput2 = @nResult</p>
            }
        }
        catch (Exception e)
        {
        <p>
            Exception @e.Message</p>
        }
    }
    </form>
</body>
</html>
```

Figure 9

With this approach we can capture all sorts of inputs from the internet. The other interesting thing from this approach is the way in which we handle bad data using structured exception handling. We parse

the nInput1 and nInput2 Decimals inside a try block. That way if the user enters “asdf” in the textInput1 input, we can catch the format exception and display a message that is informative to our intended audience. At first our intended audience is us, so we just display e.Message, which describes the error to a programmer. The exciting thing about the structured exception handling is that it lets us handle the error at the correct level of abstraction. This leads us to another exciting point. In our .cshtml files we can take advantage of the whole C# .NET ecosystem, including adding our own classes.

Adding and Instantiating Classes in the .cshtml Web Pages

When we used the Add Item dialog there were many choices such as “Style Sheet”, “XML File” and “HTML Page”. One of the choices is to add a Class. To Add a class:

1) Right mouse click over the project and from the context menu select Add New Item
2) From the Add New Item dialog select Class
3) If this is the first class that you are adding to the web site you will get a dialog asking you if you want to place the file in the App_Code folder. Click Yes, and the empty class will be created for you to fill in.

Being able to add classes to work behind web pages is exciting for a number of reasons. Like css separates presentation from the content of an html file, classes take that one step further by being able to have multiple .cshtml views of the same business/domain model classes. The class in Figure 10 is pretty basic, but illustrates the point.

```
using System;
/// <summary>
/// Demo is a do nothing class that demonstrates throwing an exception that can be
handled at the correct level of abstraction
/// </summary>
public class Demo
{
    public int nDemo = 15;
    /// <summary>
    /// just throws an exception
    /// </summary>
    /// <returns>nothing ... only throws</returns>
    public int throws()
    {
        throw new Exception("I threw from the throw method");
    }
}
```

Figure 10

When the Demo class is called from the following .cshtml file in Figure 11:

```
<!DOCTYPE html>
<html>
    <head>
        <title>get member from demo object</title>
    </head>
    <body>
        <h1>get member from demo object</h1>
        @{
            try
            {
                Demo dNew = new Demo();
                <p>Demo object says @dNew.nDemo</p>
                int nThrows = @dNew.throws();
                <p>Demo object says you shouldn't see this</p>
            }
            catch (Exception e)
            {
                <p>Demo object is broken @e.Message</p>
            }
        }
    </body>
</html>
```

Figure 11

The following output in Figure 12 is generated:

Figure 12

Placing business logic in classes allows us to instantiate objects that have domain behaviour like oMyaccount.credit or oMyAccount.debit as well as persist and retrieve themselves from the database per Michael Fowler's Active Record pattern, "An object that wraps a row in a database table or view, encapsulates the database access, and adds domain logic on that data."[ii] We can also instantiate C# ADO.NET objects to manipulate SQL data in a more traditional way.

Accessing MySql from .cshtml Web Pages

ADO.NET is the basic way to access a database from C# as described in Murach. The basic pattern is to:

1. Add a reference to the provider that you are using … for mysql we browse to mysql.data.dll
2. Instantiate a connection object depending on the provider that you are using
3. Instantiate a command object from the sql String that you want to run and the connection object from step 1
4. Execute the command into a data reader if the command is a select, or to return the number of rows affected if the command is not a select

The following table names the specific classes for some of the ADO.NET providers that are used in this process.

SQL Server	OLE DB	ODBC	Oracle	MySQL
SQLConnection	OleDbConnection	OdbcConnection	OracleConnection	MySqlConnection
SQLCommand	OleDbCommand	OdbcCommand	OracleCommand	MySqlCommand
SqlDataReader	OleDbDataReader	OdbcDataReader	OracleDataReader	MySqlDataReader

The following example in Figure 13 illustrates this pattern, by first connecting with the connection string `"user=root;database=sample;convert zero datetime=true"`. In this case we have added "convert zero datetime=true to our connection string so that we can display rows with null MySql DATETIME values as C# DateTime objects. Next we create a command from the String `"SELECT type, class, numberofcrew, commisioned, name FROM ships"` and the connection object that we just created. Finally we execute the query into a MySqlDataReader and loop through the results with a results.Read().

```
@using System;
@using System.Data;
@using MySql.Data;
@using MySql.Data.MySqlClient;
<!DOCTYPE html>
<html>
    <head>
        <title>A List of Ships from the database</title>
    </head>
    <body>
    <h1>A List of Ships from the database</h1>
        @{
            MySqlConnection conn = new MySqlConnection("user=root;database=sample");
            try
            {
                conn.Open();
                String sSQL = "SELECT type, class, numberofcrew, commisioned, name FROM
ships";
                MySqlCommand cmd = new MySqlCommand(sSQL, conn);
                MySqlDataReader results = cmd.ExecuteReader();
                <table>
                @while (results.Read())
                {

<tr><td>@results[0]</td><td>@results[1]</td><td>@results[2]</td><td>@results[3]</td><td>@
results[4]</td></tr>
                }
                </table>
            }
            catch(Exception e)
            {
                <p>Error message was @e.Message</p>
            }
            finally
            {
                conn.Close();
            }
        }
    </body>
</html>
```

Figure 13

Parameters and SQL Injection Attacks from Web Input

If we want to be able to search our ships database the naïve approach would be to use the input technique from Figure 9. We could do something like in Figure 14:

```
String sQuery = "";
if(Request["query"] != null)
{
    sQuery = Request["query"];
}
String sSQL = "SELECT type, class, numberofcrew, commisioned, name FROM ships WHERE name
LIKE CONCAT('%', '" + sQuery + "', '%')";
```

Figure 14

Unfortunately this is a really bad idea, because if someone searches for a ship named `"');DROP TABLE test;#"` and there is actually a test table in the database they can cause the table to be dropped. This is called a SQL injection attack, and is certain to happen, given enough time. The proper way to prevent this kind of attack is to use parameters like in Figure 15.

```
String sQuery = "";
if(Request["query"] != null)
{
    sQuery = Request["query"];
}
conn.Open();
String sSQL = "SELECT type, class, numberofcrew, commisioned, name FROM ships WHERE name
LIKE CONCAT('%', @query, '%')";
MySqlCommand cmd = new MySqlCommand(sSQL, conn);
cmd.Parameters.AddWithValue("query", sQuery);
```

Figure 15

This important technique must also be used for inserting data from web input as in Figure 16.

```
String sSQL = "INSERT INTO ships(type, class, numberOfCrew, commisioned, name)
VALUES(@type, @class, @numberOfCrew, @commisioned, @name)";
MySqlCommand cmd = new MySqlCommand(sSQL, conn);

cmd.Parameters.AddWithValue("type", Request["type"]);
cmd.Parameters.AddWithValue("class", Request["class"]);
cmd.Parameters.AddWithValue("numberOfCrew", Int32.Parse(Request["numberOfCrew"]));
cmd.Parameters.AddWithValue("commisioned", DateTime.Parse(Request["commisioned"]));
cmd.Parameters.AddWithValue("name", Request["name"]);
int nRows = cmd.ExecuteNonQuery();
<p>@nRows rows inserted into the database</p>
```

Figure 16

When working with web input, an annoying artifact of the request/response architecture is that the input doesn't display in the same page as the results. Consider the Add 2 Numbers dialog from Figure 6. If you can see the results of the addition the input fields no longer display the values being added.

Using AJAX to Update Part of a Page

If we could update just the part of the page that displays the results of our input, the input would still be visible along with the results. Asynchronous Javascript And XML is a technique for updating only a part of a page. Although it is possible to use this technique with pure javascript, browser inconsistencies make using a library like jquery a less frustrating approach. We can get jquery when we load our page from the googleapis page by including this script tag in our script:

```
<script
src="//ajax.googleapis.com/ajax/libs/jquery/2.0.0/jquery.min.js"></script>
```

By adding a div for the results to go in, and a div for the submit button we can progressively enhance the web page if javascript is enabled to only update the results div when the submit button is clicked. If javascript isn't enabled the page will behave like it did before. Other than the div and the ids our .cshtml page is largely unchanged. The post request that we get is also unchanged from Figure 8.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Add 2 Numbers</title>
    </head>
    <body>
        <h1>Add 2 Numbers</h1>
        <form action="" method="post">
            <p><label for="textInput1">Enter First Number</label><br /><input type="text"
name="textInput1" id="textInput1" /></p>
            <p><label for="textInput2">Enter Second Number</label><br /><input
type="text" name="textInput2" id="textInput2" /></p>
            <input id="submitButton" type="submit" value="Add" />
        </form>
        <div id="results">
            @{
                String sTextInput1 = Request["textInput1"];
                String sTextInput2 = Request["textInput2"];
                if(sTextInput1 != null && sTextInput2 != null)
                {
                    try
                    {
                        Decimal n1 = Decimal.Parse(sTextInput1);
                        Decimal n2 = Decimal.Parse(sTextInput2);
                        Decimal nResult = n1 + n2;

                        <p>The Sum of @n1 + @n2 = @nResult</p>
                    }
                    catch(Exception e)
                    {
                        <p>Exception @e.Message</p>
                    }
                }
            }
        </div>
    </body>
    <script type="text/javascript"
src="//ajax.googleapis.com/ajax/libs/jquery/2.0.0/jquery.min.js"></script>
    <script type="text/javascript">
        jQuery("#submitButton").click(function () {
            jQuery("#results").html("<img src=\"images/turningArrow.gif\" />");
            var oData = { textInput1: jQuery("#textInput1").val(),
                textInput2: jQuery("#textInput2").val()
            };
            jQuery("#results").load("Add2NumbersAjax.cshtml #results", oData);
            return false;
        });
    </script>
</html>
```

The jQuery() selector method also has an alias of $(). You will see many examples where $() is used for jQuery. I have chosen to use the jQuery syntax, because $ is also used in Murach HTML5 as an alias for document.getElementById(). Hopefully explicitly using jQuery reduces any confusion that this causes.

Uploading Images to a web site

Often we need visitors to our site to at least be able to upload an avatar to our site. Usually it is good for customers as well to be able to post new pictures to our site. This is a common use case, so for some time now there has been a way to do this with http. If we send an http request with content type = “multipart/form-data” we can include an image in that content.

```
POST /fileSamples/DragAndDrop.cshtml HTTP/1.1
Host: localhost:49406
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:20.0) Gecko/20100101
Firefox/20.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://localhost:49406/fileSamples/DragAndDrop.cshtml
Content-Length: 775899
Content-Type: multipart/form-data; boundary=--------------------------
244291341994
Connection: keep-alive
Pragma: no-cache
Cache-Control: no-cache


-----------------------------244291341994 Content-Disposition: form-data;
name="fileToUpload"; filename="Jellyfish.jpg" Content-Type: image/jpeg
ÿØÿà►┼JFIF►┌┐┌►`►`►►ÿá … rest of image omitted
```

A request like this can be constructed with a form or using html5 javascript. In the case of a form, the form tag must have the attribute enctype="multipart/form-data". There is a “file” type of input so that the user can select the file to upload, then when a regular submit type button is pressed the multipart/form-data request will be sent to the server. This is a form that we can use to send the proper type of image upload request to a server:

```
<form action="" method="post" enctype="multipart/form-data">
<fieldset>
    <legend>Upload Image</legend>
    <p>
    <label for="fileToUpload">File To Upload</label><br />
    <input type="file" name="fileToUpload" />
    </p>
    <input type="submit" value="Upload" />
</fieldset>
</form>
```

Your server code receives the request like any other web input POST request, but by the time your code gets the request the image is ready for further processing.

```
if(IsPost)
{
    image = WebImage.GetImageFromRequest();
    if(image != null)
    {
        String sPath = Path.GetFileName(image.FileName);
        sImagePath = "images/" + sPath;
        image.Save(sImagePath);
    }
}
```

Note* The default maximum image upload size is 4,096, 000 bytes (~4 mb). You can change this by editing your web.config file as follows:

```
<configuration>
  <system.web>
    <compilation debug="true" targetFramework="4.0"/>
    <httpRuntime maxRequestLength="8000" />
  </system.web>
</configuration>
```

Using Ajax To Drag and Drop an Image into Part of a page:

You can also use Ajax to drag and drop an image in Firefox and Chrome. The code is the same on the server, so it is easy to offer drag and drop on browsers that support it and more conventional form generated requests for internet explorer. After adding the dataTransfer property to jQuery, we need to define 2 new event listeners. The following dragover event listener just stops event propagation and sets the effect to show that a file is being copied on to the server.

```
jQuery.event.props.push('dataTransfer');
jQuery("#dropTarget").bind('dragover', function (e) {
    e.stopPropagation();
    e.preventDefault();
    e.dataTransfer.dropEffect = 'copy';
    // Explicitly show this is a copy.
});
```

The interesting part is the drop event listener. In the drop event listener we go through a number of stages.

1) Turn off event propagation
2) Get the file that was dropped into a FormData object
3) Construct a request and send to the server
4) Listen for the result and display the image if it worked.

```
        jQuery("#dropTarget").bind('drop', function (evt) {
            // step 1 disable events
            evt.stopPropagation();
            evt.preventDefault();

            // step 2 get the file that was dropped into a formdata object
            if (evt.dataTransfer.files === undefined) {
                alert("Internet Explorer doesn't support file drop yet ... Please use the
Upload mechanism, below");
                return;
            }
            var files = evt.dataTransfer.files;
            // files is a FileList of File objects. We are only uploading the first one
here.
            myFileObject = files[0];
            var formData = new FormData();
            // Append our file to the formData object
            // Notice the first argument "file" and keep it in mind
            formData.append('fileToUpload', myFileObject);
            // Step 3 Create our XMLHttpRequest Object - note that we use an
XMLHttpRequest so that we can send FormData with it
            var xhr = new XMLHttpRequest();
            // Open our connection using the POST method - note that the url is supplied
to the javascript in the c# code

            xhr.open("POST", "@Request.Url.ToString()");

            xhr.send(formData);

            // Step 4 listen for the result and display the image if it worked
            xhr.onreadystatechange = function () {
                if (xhr.readyState == 4 && xhr.status == 200) {
                    // if we have the image on the server then we can put it in the
displayImage div (which has to exist)
                    document.getElementById("displayImage").innerHTML = "<img
src=\"images/" + myFileObject.name + "\" />";
                }
            }

        });
```

Using this approach we can do image upload to a single server in a directory that we have control over. This approach doesn't work for deployment on the cloud though as it doesn't scale particularly well when an application's load is spread across multiple cloud based servers. On the cloud the above approach works as the way to get the image scaled and onto one server as a temporary file. Once the image is on one server, another step is required to copy it on to the content delivery network (CDN) where it can be available to multiple servers serving the same site. The CDN that we will be using is called Cloudinary. We will learn more about Cloudinary when we talk about deployment. Before we can talk about deployment on the cloud, it is good to understand how a web application can also consume services provided by other web applications. An example of a web application that you will likely want to consume in your .cshtml is Google Custom Search Engine.

Using a WebClient object to Access a Google Custom Search Engine

When we discussed parameters and SQL Injection attacks, we used an example that searched the name field in a table to search that table using the LIKE SQL operator. Unfortunately, even with parameters to prevent a SQL injection attack this approach is flawed. If the data we are looking for is in another field this approach doesn't work. The approach also doesn't work as expected if you enter 2 query parameters separated with a space. The expected result is that if one or both of the entered query parameters is found it will be displayed. We expect that because that is how Google search works. Unfortunately our little SQL search doesn't work that way. We can however access Google search programmatically and have it only search its index for our site. This can be done with a Google Custom Search Engine. To set up a Google Custom Search Engine:

1) Use your google id to set yourself up as a google web master and submit your site's url and preferably a site map for indexing by Google (this process is documented quite well in the google help). It may take some time before Google gets to indexing your site.
2) Go to http://www.google.com/cse and follow the instructions to create a new search engine
3) Get your search engine's id by clicking on settings in the left, selecting the basics tab and clicking on the "Search Engine Id" button.
4) Get yourself an api key by going to https://code.google.com/apis/console. You may need to create a new project, and click to toggle on Google Custom Search under services. You API key is found by selecting API Access on the left side of the screen under Simple API Access.
5) Use your search engine id and api key to make a query string like:

```
String sUrl = "https://www.googleapis.com/customsearch/v1?key={0}&cx={1}&alt=json&q={2}";
```

There is a complete example of Google custom search with WebClient access of a Youtube video uploads as well as CSS menu code to tie them altogether with a RenderPage directive at:

https://github.com/rhildred/csharpexamples/tree/master/googleCustomSearchEngineVideosRenderPage

The idea of this larger example is that it shows using .cshtml and css to make a site with multiple pages, as well as showing the use of WebClient and Json.Decode. There is also an XML example of doing something similar at:

https://github.com/rhildred/csharpexamples/tree/master/xDocumentExample.

Now that we have a bit more of a multipage site example we can look at one way of deploying on the internet.

Deployment

For your site to be viewable on the Internet, you will need to put it on a host that is accessible from the internet. The host that runs your web site will also need to support the resources it uses like database and server side code in a .cs or .cshtml file. A host that provides these resources, up to a point for free is

http://appharbor.com. Appharbor also offers enhanced services for when your application moves into production and requires more resources. There are also Virtual Private Servers and more traditional hosting companies like GoDaddy.com that can host .cshtml files.

Appharbor uses a program called git to synchronize changes between your personal development computer and the host that is visible from the internet. Git is a popular tool for concurrent programming among multiple developers. It was developed and designed by Linus Torvalds for Linux development. In the AppHarbor context of putting data on a server Git is used somewhat like the ftp program that is described in Murach HTML5. Git, however, has much more functionality to deal with synchronization among multiple users. Git's synchronization is like optimistic locking in databases. It looks to see if you were working on the latest copy, before accepting changes. If you weren't working on the latest copy, git has functionality to help resolve the issue. You can get git from http://sourceforge.net/projects/mingwbuilds/files/external-binary-packages/. In this package git comes with some other handy unix like tools like wget.

If you are working on your own project in AppHarbor you shouldn't have any concurrency/synchronization issues so your workflow should be like this:

To Start a Blank Project:

1) In your browser surf to https://appharbor.com/, sign in and click on "Your Applications."
2) In the "Create New Application" dialog in the center of the screen supply a unique name for your application.
3) Click the "Create New" button
4) In the project panel, on the left hand side of your screen click on the Repository URL button.
5) If a url is displayed copy that URL, otherwise it will say Repository URL copied to clipboard
6) In your file explorer, go to the directory where you installed msys, and click on msys.bat
7) In the Msys shell, type "cd /e/workspace" where "e" is the drive letter that you are working from and "workspace" is where you are keeping your web sites. Press enter.
   *Note – in the MSys shell you can type part of a path and then press the tab key and shell will try to complete your path.
8) Type "git clone https://rhildred@appharbor.com/richssite2.git" where the url is supplied from the url copied in step 5. Press enter.
9) You will be prompted for your appharbor password. Note that there is no feedback when you type it in and press enter. You should get a warning that you have cloned and empty repository.
10) In visual studio start a new solution. For the location enter the directory from step 7. For the name you will use the application name from step 2.
11) Create some content (a simple hello world index.html file will do)
12) In your msys window change to the application name directory
13) Type "git add richssite2.sln" and "git add richssite2" using your application name and pressing enter after each step to add the files to revision control
14) Type 'git commit –am "initial commit" ' where the message "initial commit" refers to what you just did. Press Enter.

15) Type “git push origin master” to put the code on appharbor. Press Enter. You will need to enter your password when prompted. Remember that there is no feedback for password typing.
16) Go to appharbor. If you click on the name of your application you should be looking at a screen that says your application is active. There is a link on the top right that you can follow to view your app.

To Add Database to your project:

1) On App Harbor under your application click “addons”, and scroll down to mysql. Click on See More and click to select the free instance Yocto
2) Click on Configuration variables to get the mysql connection string. Click the copy to clipboard icon to the right to get your connection string.
3) In Visual studio paste the connection string into the web.Config in the following manner:

```
<?xml version="1.0"?>

<!--
  For more information on how to configure your ASP.NET application, please visit
  http://go.microsoft.com/fwlink/?LinkId=169433
  server=4f20c4c6-7196-4b1e-a9c0-a1ba00fc5d53.mysql.sequelizer.com;
  database=db4f20c4c671964b1ea9c0a1ba00fc5d53;
  uid=ljqunwpggfltrnyh;
  pwd=yJNZnSEbYP8PJRuTtjizAtVvFAkgNF6NpCgJa37RXL6Y34NFZPyEvEYfVrtTFxvs
  -->

<configuration>
    <system.web>
        <compilation debug="true" targetFramework="4.0" />
    </system.web>
    <connectionStrings>
      <remove name="MyDb"/>
      <add name="MyDB"
           connectionString="server=4f20c4c6-7196-4b1e-a9c0-
a1ba00fc5d53.mysql.sequelizer.com;database=db4f20c4c671964b1ea9c0a1ba00fc5d53;uid=ljqunwp
ggfltrnyh;pwd=yJNZnSEbYP8PJRuTtjizAtVvFAkgNF6NpCgJa37RXL6Y34NFZPyEvEYfVrtTFxvs"
           providerName="MySql.Data.MySqlClient"/>
    </connectionStrings>


</configuration>
```

You can also use the connection string with mysql workbench to put data in the database.

To Add Mysql and Razor to your Project

You will need some references to extra code. The references that you need are:

1) MySql.Data
2) MySql.Data.Entities
3) Mysql.Web
4) Microsoft.aspnet.razor

While you are working on this you may also want to get the image Content Delivery network

5) CloudinaryDotNet

These packages can be retrieved by going to the Nuget package manager. In Visual Studio 2010 this is available by right mousing over your project and selecting “Add Library Package Manager.”

To edit the new database:

1) Open MySQLWorkBench.exe, Click on New Connection
2) In the host field of the Setup New Connection dialog, paste the server from your connection string in your web.Config, UserName=uid, DefaultSchema=database and click on “Store In Vault” to enter the connection string pwd value for your password.
3) Click on Test Connection to make sure that you have done this correctly.
4) Use the SQL Workbench window to edit the new database

For instance for our example you will need one table to hold the numbers that we will be entering from the web.

CREATE TABLE numbers(id INTEGER PRIMARY KEY AUTO_INCREMENT,

theNumber INTEGER)

AppHarbor hosts the mysql database that we are using, both when we push to production and on our local machine. AppHarbor supports a few different databases like that but it also supports an image content delivery network called Cloudinary.

Image Upload and Retrieval from Cloudinary

Cloudinary is a content delivery network (CDN) for uploading and retrieval of images. Like Mysql you can use Cloudinary in both your test and production environments. You are already experienced in using a CDN for retrieval. When you add this to an html file:

```
<script type="text/javascript"
src="//ajax.googleapis.com/ajax/libs/jquery/2.0.0/jquery.min.js"></script>
```

You are actually getting your jquery on the fly from Google’s CDN. Cloudinary is similar in that you just use a URL that they provide to get your content. You can also upload images to Cloudinary. We already learned how to upload and process images to a web site. With Cloudinary we reuse that technique to get the image in a temporary location and then we add a few lines of code to copy it to the CDN.

As with MySQL the first step to using Cloudinary is to add it to your project. You will see Cloudinary under addons, where you saw Mysql on appharbor. Once you have added it, you will need to get your CLOUDINARY_URL from the Configuration Variables pane also like we did for MySQL. To add an image to cloudinary you can do the following:

```
Cloudinary cloudinary = new Cloudinary("cloudinary://283344254821121:INXwnv9AJtOP9PUdmR-
xxVa2DTk@hfm9olpom");
CloudinaryDotNet.Actions.ImageUploadParams uploadParams = new
CloudinaryDotNet.Actions.ImageUploadParams()
{
       File = new CloudinaryDotNet.Actions.FileDescription(sImagePath)
};

CloudinaryDotNet.Actions.ImageUploadResult uploadResult =
cloudinary.Upload(uploadParams);
```

You will still need to get the URL for the new resource using the uploadResult like this:

```
string url = cloudinary.Api.UrlImgUp.BuildUrl(String.Format("{0}.{1}",
uploadResult.PublicId, uploadResult.Format));
File.Delete(sImagePath);
```

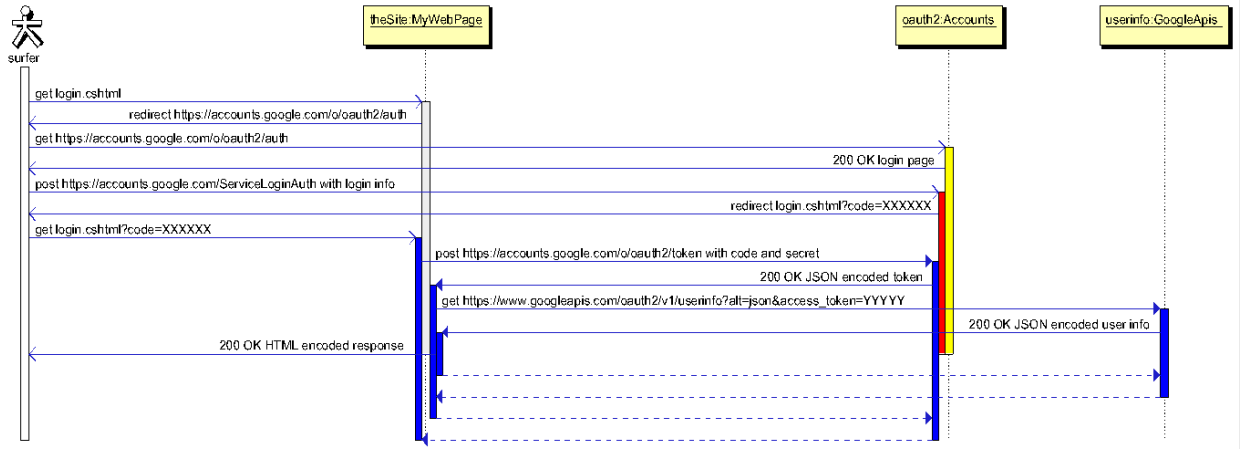
Finally we also deleted the temporary file. There is a more complete example at https://github.com/rhildred/csharpexamples/tree/master/imageFileUploadToCloudinary. You probably don't want users to be able to upload images to your account without first identifying themselves. Next we will look at a way of identifying users that will be needed if users will be involved in providing content for your web application.

Identifying Users with a Federated Login

There is a login/sign up helper that comes with the Razor engine much like the Image helper and Json helper that we already have used in our code. This login helper has a couple of problems though.

1) The user has to remember a different password for your site.
2) The user's password is sent in the clear to the .cshtml code where it is verified.

While there are plenty of sites that operate this way, it makes more sense to leave the identification of people to the social networks that they belong to like Linked In, Twitter, Facebook or Google Plus. Fortunately these networks all support a mechanism for doing this with a protocol called Oauth2. The Oauth2 protocol is a way that a user can use a trusted website like Google to login to a site that Google knows about, but that the user doesn't yet trust. The Oauth2 protocol works like this:



Oauth2 works over https: using Google's certificate so that no information leaks out except for the code, which you need to combine with your secret to access the user info, which is protected by https. There is a code example of this flow at https://github.com/rhildred/csharpexamples/tree/master/federatedLogin. With the code in these examples it is possible to make almost anything, but by their nature these examples are short. The next example pulls the federated login together with the image upload and javascript to edit wikitext to make a simple (as possible) blogging page for a website.

In summary, web programming with .cshtml seems like a nice way to introduce the remaining more advanced techniques of file input/output, XML, LINQ and deployment. Java Server pages (.jsp) and .php are also very similar, and a lot of the same techniques apply. C Sharp html pages (.cshtml) is also a powerful method in it's own right for implementing web applications.

---

[i] Helix84 HTTP persistent connection, client-server connection schema, http://commons.wikimedia.org/wiki/File:HTTP_persistent_connection.svg retrieved (April 28)

[ii] Fowler, Martin "Active Record", http://www.martinfowler.com/eaaCatalog/activeRecord.html retrieved (April 29th)