ASP.NET: How to delay the popup of Windows Authentication dialog.

1. Make all the pages ananymous accessable and windows integrated authentication.

2. Make a bridge.aspx that is only windows integrated authentication.

3. When login:

    1. If windows login information is available, like in IE, then uses the information to auto-login

    2. If windows login information is not availbale, like in Firefox, redirect to bridge.aspx to request for the windows login information, then in bridge.aspx to redirect to the page that user request. During the redirecting, the login page will be activated again, yet this time the windows login information is available now.

Challenge/Response authentication.

## [**ASP.NET MVC Pro’s and Con’s**](http://www.ytechie.com/2008/10/aspnet-mvc-pros-and-cons.html)

20 Oct, 2008  [asp.net](http://www.ytechie.com/category/aspnet), [mvc](http://www.ytechie.com/category/mvc)

In our current iteration of improving our software development strategy, ASP.NET WebForms simply doesn’t fit in with the new demands of being unit testable and flexible. It comes as no surprise that ASP.NET MVC has been getting all the headlines lately. Like many others, I’ve gotten the itch to try it out. In this post I’ll talk about what I believe are the main pro’s and con’s of the new style for building web applications.



**PRO – No ViewState or "surprise crap"**

In traditional ASP.NET WebForms, the luxury of pretending to behave like a Windows Form comes at a price. The ViewState is a reliable way of storing all of the state information for the form. Unfortunately, due to the limitations of the web, this data needs to be a giant string inside of a hidden form field. This ends up adding a substantial number of bytes to the page, making it slower and requiring extra bandwidth. Of course the ViewState is controllable, much like the dinosaurs in Jurassic Park.

Not only is the ViewState gone, but "mystery" generated HTML is also gone. You have strict control over the HTML. This gives you great power, but with great power comes great responsibility. Use it wisely, and you will have elegant XHTML output with no surprises. You need to really know your HTML, which in today’s web world is a prerequisite anyway.

**PRO – Faster server-side**

It’s hard to get any real performance data about MVC, but it’s been suggested that it’s potentially 8000 times faster. Supposedly it’s due to less processing since it simply processes a "template" instead of having to build a complicated control tree. Even if it’s twice as fast, or even marginally faster, that would be significant for popular sites, or give at least a slight boost to smaller sites.

**PRO – Simplified model for multiple related views**

One thing that I found much easier to do with MVC was to have multiple versions of a page that displayed the same data, but in a slightly different format. For example, on my [RSS package tracking](http://www.simpletracking.com/) website, you can look at your tracking information in a full-featured desktop browser, a mobile browser, or an RSS reader. The data being displayed is always the same, but the actual rendered output was different. If I later want to make an iPhone specific version for example, I would simply make a new view, and use an existing controller action.

**PRO – Unit testable**

One of the biggest challenges with WebForms was that testing was difficult. Not only was the actual output not easy to test, but the code-behind would tend to be a place that would contain important code that would never get unit tested. With both MVC and WebForms, it’s best to keep your logic away from the page, but it’s not always easy or ideal. MVC makes it simple to unit test the logic that is specific to a page. To do so, you simply test the actions in your controllers, which are regular, easy to test classes.

**CON – Difficult to convert an existing site**

MVC is not only a framework in this case, but a style. It is possible to convert specific pages as needed, but the cost is high. The problem is that MVC is radically different, so the benefit of converting probably isn’t worth it for most of your *existing* pages.

If you decide to convert your site to MVC, you may also run into issues trying to maintain existing page addresses. The specific issue I’ve ran into is that routes cannot have a trailing slash. If you want to maintain existing URL’s that have trailing slashes, there is no way to have the built-in routing generate URL’s with a trailing slash. You may end up losing one of the big advantages that MVC has to offer.

**CON – Not best for SEO out of the box**

I’ve mentioned some of the [SEO issues](http://www.ytechie.com/2008/10/aspnet-mvc-what-about-seo.html) before, and all but the trailing slash issue have a reasonable workaround. The routing engine likes to allow multiple addresses to render the same page, instead of enforcing a single address for each page. Luckily, [as Scott Hanselman mentions](http://www.hanselman.com/blog/ASPNETMVCAndTheNewIIS7RewriteModule.aspx), you can use a URL rewrite engine to bend it to your will. I highly recommend spending some time writing intelligent rules that perform the necessary 301 redirects, because you don’t want to take chances with SEO (Search Engine Optimization).

**CON – Challenges if you’re not running IIS7**

It’s clear that the last couple of versions of IIS have been major improvements over their predecessors. IIS7 takes .NET integration to an entirely new level. There is already [a good page that covers the challenges you’ll face if you’re not running IIS6](http://weblogs.asp.net/omarzabir/archive/2008/06/30/deploy-asp-net-mvc-on-iis-6-solve-404-compression-and-performance-problems.aspx). I’ll just list them here for brevity:

* .NET needs to handle all page requests to ensure that the MVC pages will be processed. This leads to bad performance of static files.
* HTTP Compression through IIS6 doesn’t work, because the MVC pages are dynamic.
* The homepage gives a 404 when hosted on the root of a domain.

**Summary**

If I needed to build a new site from scratch, and was able to use IIS7, it would be extremely likely that I would choose ASP.NET MVC. It’s a joy to work with (possibly because it’s "new"), and just makes sense. If I needed to work with an existing site, I would certainly have to consider the pro’s and con’s I mentioned above. ASP.NET MVC gives us an amazing new tool in our huge Microsoft toolbox.