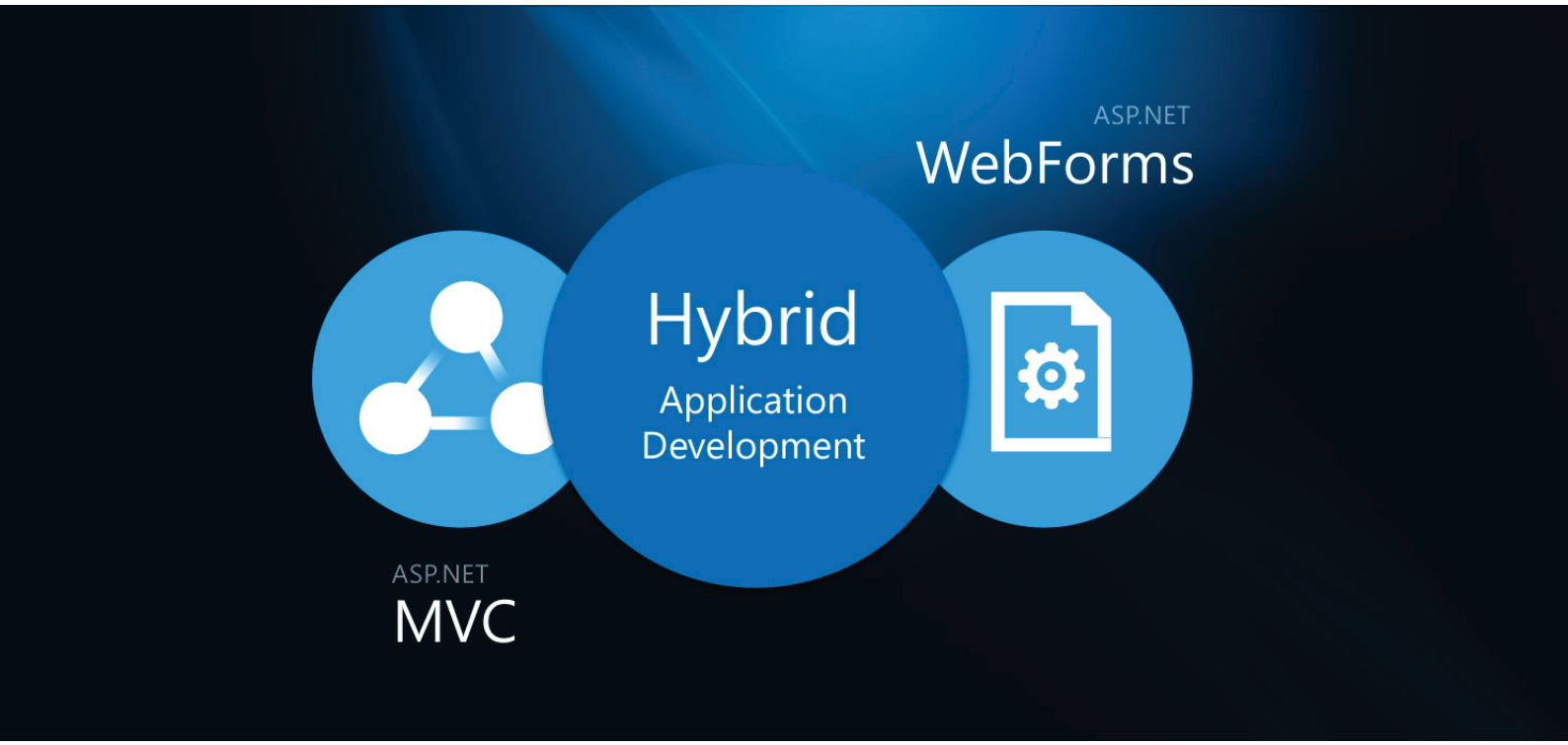
Telerik Sitefinity




# ASP.NET MVC in Sitefinity

**Ivan Osmak, Sitefinity Product Manager**

telerik
*deliver more than expected*

# What is ASP.NET MVC?

The ASP.NET MVC web application framework was introduced by Microsoft in March 2009. As of this writing, Microsoft is preparing to release the fourth iteration of this framework. ASP.NET MVC was created in response to growing criticism of ASP.NET WebForms, as well as evolving trends in web development. The web development community criticized ASP.NET WebForms on three major points:

1. Problems with testability
2. Lack of control over the markup
3. Modifying the stateless nature of the web (HTTP) through ViewState

Criticisms aside, ASP.NET WebForms has been and is ***STILL*** a tremendously successful web framework that will be heavily used in the years to come. In fact, one could argue that Microsoft was using ASP.NET MVC to address a growing demographic, rather than fundamentally shifting how .NET web applications are created.

# The MVC pattern and its advantages

The ASP.NET MVC pattern is Microsoft's take on a long-standing and well-known MVC design pattern. MVC stands for Model-View-Controller; in this design pattern there are three building blocks of an application and they are used to create clean separation between an application's layers.

The MVC pattern's goal is to separate presentation (View), logic (Controller) and data (Model). Seasoned developers will recognize the value of this proposal, but it's important to acknowledge that this is entirely possible with ASP.NET WebForms. An ASP.NET WebForms application can enjoy similar benefits if sound decisions are made when planning the application's architecture and implementation. However, because these layers are mandated by the MVC pattern it's harder to make a mistake.

## Testability

In ASP.NET WebForms the building block is a Control class (even Page class inherits Control class). Testing of types that derive from Control class is essentially impossible; this is due to the fact that the instance of Control class requires an instance of HttpContext class, which Microsoft has made sealed and almost impossible to mock.

With advent of ASP.NET MVC, Microsoft has remedied this problem by introducing a new class called HttpContextBase which is abstract and therefore trivial to mock. This fact, together with the built-in separation of concerns makes ASP.NET MVC building blocks much easier to test.

## Control over the markup output

Over the past ten years, we have witnessed the growing importance of web standards, proper markup and styling. With the explosion in popularity of JavaScript, client side frameworks and AJAX, the ability to precisely control the markup of web pages has become extraordinarily important. Total control over the markup in ASP.NET WebForms is impossible – and with a good reason.


Understanding why ASP.NET WebForms does not provide greater levels of control over the markup requires us to go back, more than ten years ago, to the release of ASP.NET 1.0. At that time, the main problems of the industry were quite different: poor support of web standards, almost every browser interpreted markup, CSS and JavaScript differently and an army of developers trained to develop in stateful environment (desktop). Microsoft offered a solution to these critical problems through its ASP.NET WebForms framework. The framework took care of the browser incompatibilities and mimicked state (on a purely stateless HTTP protocol) through the ViewState feature – thus making it simpler for thousands of developers to embrace the ever growing need for web development. The abstraction – which we have all appreciated ten years ago and which made ASP.NET WebForms phenomenally popular – is what many criticize today. With browsers becoming highly standardized, developers having ten or so years' experience on the web – we need the control back.

ASP.NET MVC framework, together with its brilliant Razor rendering engine was the answer to this growing criticism. With ASP.NET MVC, it is the developer that completely controls the markup as the abstractions provided by the Control class are gone. There is no more “runat=server” attribute, as the server is not involved anymore. With developers starting to appreciate the stateless ways of HTTP protocol, the ViewState is gone as well.

# Comparing ASP.NET MVC and ASP.NET WebForms

While it is quite simple to technically juxtapose the two frameworks, it's important to examine the practicalities of everyday life. Namely, the great advantage of ASP.NET WebForms lays in the enormous infrastructure and ecosystem built around it. In fact, ASP.NET WebForms could be compared to

petroleum powered cars, while ASP.NET MVC to electric cars. With all the advantages of electric cars, it's practical to consider the numerous gas stations or mechanics that do not support this technology. Furthermore, in software there is no silver bullet. While, ASP.NET MVC makes a lot of sense for web pages, it's possible that web applications – especially ones not relying on JavaScript and AJAX – could still benefit greatly from WebForms.

In the end, design patterns have been around for quite a while in software development. These patterns transcend environments and programming languages and an experienced developer will decide what to use based on what needs to be built – not on what's "cooler".

# How can one use ASP.NET MVC within Sitefinity?

Since v3.0, all the way to v5.0 Sitefinity has been exclusively ASP.NET WebForms application. With v 5.1, Sitefinity begins to natively support ASP.NET MVC. This news is bound to produce mixed and strong reactions, so it's perhaps best to offer the following statement first:

> Sitefinity 5.1 was not rewritten to ASP.NET MVC. Sitefinity will not ditch support for ASP.NET WebForms or decrease investment in it. The implementation of ASP.NET MVC in Sitefinity is not a "look alike", but rather the actual ASP.NET MVC where appropriate extension points of the framework were taken advantage of.

## The three modes of ASP.NET MVC

Within Sitefinity, there are three ways in which ASP.NET MVC can be used. We call these three modes: hybrid, pure and classic.

The first two modes are Sitefinity specific and bring ASP.NET MVC with a Sitefinity twist. Namely, Controllers are interpreted as widgets, meaning that you can build Sitefinity widgets from Controllers and actually have more than one Controller within a page. The classic mode, as the name suggests, is the mode in which a page is represented by one controller (in essence, bypassing Sitefinity page routes) – as it is outside of the context of Sitefinity.

You might be tempted to think that the first two modes are "hacks", or at best anti-patterns, however that is not the case. In the following paragraphs implementations of all modes will be explained in detail,

but before we move on – let's quickly explore how and why this kind of revolutionary support is possible within Sitefinity.

## Flexible architecture of Sitefinity

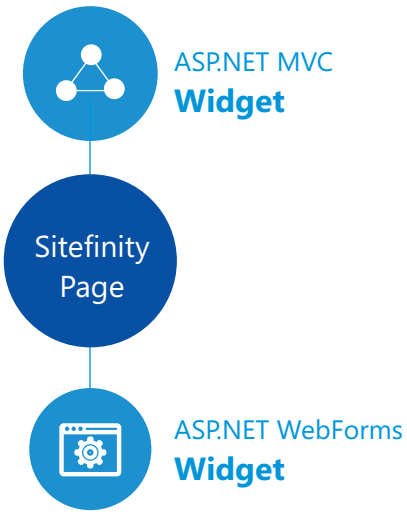
A bit more than a year ago, when Sitefinity 4.0 was released –completely rewritten from the previous 3.X versions – many architectural decisions were taken that made our support for ASP.NET MVC today possible.

First and foremost, Sitefinity is using the Routing Engine. Microsoft initially developed this engine for ASP.NET MVC and later extracted it so ASP.NET WebForms could also take advantage of it. However, this Routing Engine has remained paramount with ASP.NET MVC and is an absolute requirement for any type of real (not hacked or faked) support of ASP.NET MVC. Because Sitefinity already used this Routing Engine natively, no changes were needed to meet this requirement.
Secondly, in Sitefinity 4.1, Sitefinity natively implemented Virtual Path Provider approach for serving pages and content. Coincidently, this is the very same way in which ASP.NET MVC works, with its native implementation – VirtualPathProviderViewEngine class.

Experienced developers will see how Sitefinity provides the two key pieces for ASP.NET MVC support, even before it was officially announced. Namely, Sitefinity is able to work with routes and has extensive extension points around handling HTTP requests through its native implementation of Routing Engine. On the other hand, through native implementation of Virtual Path Provider, Sitefinity is able to control how, what and when is being served to the client – in the very same way ASP.NET does.

## Hybrid – mixing ASP.NET WebForms and ASP.NET MVC within a page

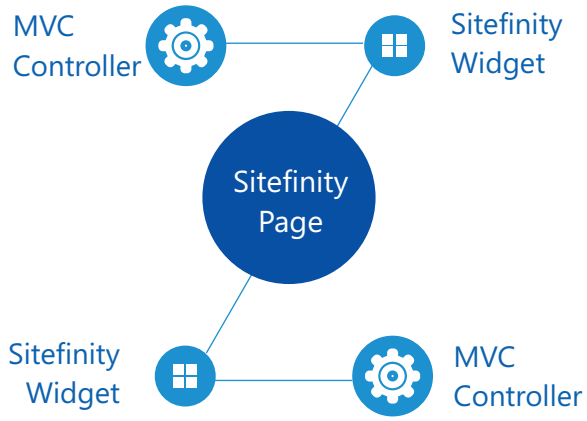
The hybrid mode allows widgets based on Controls (ASP.NET WebForms) and Controllers (ASP.NET MVC) coexist on the same page.
When using hybrid mode, a controller, model and view are created just as they would be created using classic ASP.NET MVC. The only difference is that the Controller class is decorated with the ControllerToolboxItem attribute. When started, Sitefinity will automatically recognize the controller as a toolbox item and it will add it to the toolbox, as a widget. Sitefinity will represent this widget in the Page Editor through a Control called MvcControllerProxy and it will display the content of the controller during Render method. However, this is valid only for the edit



mode. On the actual page, due to our usage of Virtual Path Provider, the contents of the View will be rendered directly into the page, without the proxy control.

> **Note:** HTML specification does not allow nested FORM tags. As in ASP.NET WebForms, all page content must be enclosed inside of a FORM tag, thus a FORM tag cannot be used inside your MVC based widgets when working in hybrid mode. There is a simple solution to this, instead of using the Html.BeginForm extension method, use the Html.BeginFormSitefinity extension method. This method will recognize if the page is in the hybrid mode and will provide a working solution through JavaScript solution, while if the page is in pure mode will output standard FORM tag. If you're building a page that heavily uses forms, we suggest – for the sake of clarity - creating that particular page in "pure" mode.

## Pure – using multiple Controllers within a page




In the pure mode, Sitefinity will not allow adding Widgets based on ASP.NET WebForms to the page, but it will, however, allow composing the page through multiple Controllers – each representing one Widget. As a result, the rendered page will not have any ASP.NET WebForms artifacts (such as <form runat="server" or ViewState).

It is important to note here, that this is not achieved through Partial Views, but rather full blown Controller-View-Model building blocks. Developers with experience in ASP.NET MVC may find this odd, as in ASP.NET MVC a page is represented by a Controller. In Sitefinity, we have moved this abstraction a bit lower.

Namely, Sitefinity's native route controls the pages and is responsible for instantiating the Widgets on it. Having this lower level abstraction, we are able to instantiate multiple controllers, execute them and have their output "baked" into the page by the time it reaches the Virtual Path Provider. If you reflect System.Web.Mvc assembly, you will quickly find out that this is exactly what ASP.NET MVC does – except it does it only for one controller. In the end, the only real difference is that Sitefinity provides a bit more advanced route registration system (and does that also through UI – page management) and a bit more advanced page composition mechanism through our implementation of Virtual Path Provider.

## Classic – using routes and controllers only

Classic mode of using ASP.NET MVC with Sitefinity is simply an opened up possibility to use RouteTable to register routes and implement Controllers, just as one would do in the stand alone ASP.NET MVC application. More information on this mode can be taken directly from the ASP.NET MVC website.

There is only one thing to keep in mind here. The route that serves Sitefinity pages is registered before the ASP.NET MVC routes and will therefore have the priority should there be a collision between the two routes. This decision has been made consciously – with the thinking that developers will understand the collision problem much more easily than would end users.

# Which mode should I use?

There is no "best" mode that we can recommend. We have provided three modes as each of them provides certain advantages and depending on the situation each of the modes may prove to be the best.

Hybrid mode provides the most flexibility. At the moment, Sitefinity does not provide pure MVC versions of all the widgets; therefore should one decide to use MVC and yet have all the benefits of built-in Sitefinity widgets one should go with the hybrid mode. Over the time, as Sitefinity provides more and more pure MVC widgets, pages can be simply converted to the pure mode.

If one is building a simple website or a very specific one, which is not covered by built in functionality – and at the same time fancies MVC – a pure mode should be chosen.

The classic mode should generally be used only for existing or new applications that need to be integrated with Sitefinity. One has to keep in mind that by using the classic mode, page management is delegated to ASP.NET MVC and therefore, features such as pages, page editor, and layout editor will be completely circumvented.

Finally, it is very important to note that modes can be mixed. Some pages can work in the pure ASP.NET MVC mode, some in hybrid mode, some using only ASP.NET WebForms and still some routes can point to the classic implementation of the ASP.NET MVC. The final decision is really left up to the developer – with no "best practice" being pushed by Sitefinity.

# Is Sitefinity abandoning ASP.NET WebForms?

Short and long answer is: no.

We do see benefits in using ASP.NET MVC in certain parts of application, just as we see the value in ASP.NET WebForms.

In addition to this, Sitefinity will continue to deliver widgets based on ASP.NET WebForms, just as their ASP.NET MVC counterparts will be developed.

We are fully aware of the enormous amount of code and functionality being implemented on the ASP.NET WebForms framework for Sitefinity and we deem it impossible to simply declare this code as not supported or in maintenance state.

All this being said, there are no short, mid or long term plans to prefer either of the frameworks over the other one.

# How do I choose between ASP.NET MVC and ASP.NET WebForms?

Very often, when we provide alternate ways of achieving something we are asked what the "best practice" is. We cannot answer this question. It is like to ask us do we recommend using a car or an airplane: to pick up a bottle of milk, we'd suggest a car; to visit a different continent, we'd recommend an airplane.

The decision will and should depend on many different factors. How much does one value benefits of ASP.NET MVC as opposed to benefits of ASP.NET WebForms. If majority of your team is experienced in ASP.NET WebForms, the proposition to move towards ASP.NET MVC may not be as convincing. How much of the client side development you expect to do; perhaps the learning curve of ASP.NET MVC may pay off.

Once again, we do not have a strong and definitive answer. Our mission is to empower people to build web sites and web applications productively. The more of choice we offer, the better we fulfill our mission.

# About the Author

Ivan Osmak is the Product Manager of Sitefinity and is responsible for aligning the nine teams of Sitefinity to push in same direction, ensure conceptual integrity of the product and finally make all that happen on time. Being a developer with twelve years of experience, he also implements an occasional feature on the side, such as Module Builder or Mobile & Responsive Design.

Ivan comes originally from Croatia, however for the last 15 or so years has lived on various places in United States, Austria and finally settled down in Sofia, Bulgaria, where he has been the member of Sitefinity team, in various roles, for last six years.

# About Sitefinity

Sitefinity is a modern online business platform which adapts to any business requirement and works equally well for online marketers, developers and IT managers. Sitefinity is used for all kinds of online applications from commercial websites to community portals and intranets, and scales effortlessly no matter the size of your project. Best of all, Sitefinity offers a simple, easy-to-use interface with a virtually flat learning curve that makes it a delight to use.

Sitefinity powers more than 10,000 websites worldwide with a particularly strong presence in financial services, government and education. Global brands such as Kraft Foods, Dannon, Magna, BASF, Toyota, Chevron and Yale School of Management rely on Sitefinity every day

# Sitefinity Around the World

**NORTH AMERICA**
+1-888-365-277

**BULGARIA**
+359-2-8099850

**UNITED KINGDOM**
+44-20-7291-0580

**GERMANY**
+49-89-2441642-70

**AUSTRALIA**
+61-2-8090-1465

telerik
*deliver more than expected*