Evaluation Guide:

ASP.NET Web CMS and Experience Platforms

INGENIUX

Content beyond boundaries

Contents

So, it's time for a new Web Content Management platform (CMS), and you want it to be ASP.NET-based. There are significant technology, platform, deployment and operational differences between the leading ASP.NET CMS platforms, so how do you know which one is right for your company?

As you evaluate ASP.NET Web CMS solutions for your company, it's important to understand four key differences and how they impact your online capabilities and web experience road map.

4 Key Differences Between ASP.NET CMS Platforms

1 Architecture: Decoupled vs. Tightly Coupled

If you want flexibility in how you build your front-end or want to leverage the content management capabilities for external applications, then you want a CMS with a decoupled architecture.

With a decoupled architecture, the application for managing content is separate from the application delivering content. The CMS does not dictate the stack, or set of technologies required to deliver the content and website, allowing for a more flexible content delivery model. In a decoupled architecture model, content can be delivered anywhere and in any format. In contrast, in a tightly coupled architecture, web content, customer data, analytics data, website presentation, and content delivery take place in a single database and application framework. In other words, content management and content delivery are the same application.

So why does this matter?

As you build your website, you want to ensure your content displays correctly on any device or channel, so a separation of content from delivery is critical. But you may also be developing customer-facing web applications that have content you'd like to manage in your CMS as well. In this case, your web application is separate from your CMS, so you need to know you can manage content in the CMS and deliver it to the web application front-end using some type of content delivery service.

In a tightly coupled architecture, this is impossible to do without having everything within the CMS. In other words, you would need to build your web apps on the CMS platform using its templates and server technology.

In addition, from an environment management perspective, you will need to set up multiple instances of your CMS (application and database) for development, staging, and production, requiring replication services to move application and content changes between environments. This model typically requires additional licenses for both the CMS and the database and can be challenging to manage.

In a CMS with a decoupled architecture, you can set up multiple deployment options based on your specific needs. These include:

- Dynamic delivery using ASP.NET
- Multi-format delivery using mixed or different server technologies
- Web Services delivery using a REST or SOAP-based API
- Device targeted delivery using a mobile detection system
- Push-based delivery such as XML, JSON, or into an external database where it is consumed by a remote application
- Plain old HTML delivery for static web content.

Your environment is easier to manage as well with a decoupled architecture. Publishing targets configured within the CMS deploy content to end-points such as staging or production web services. Bi-directional syncing keeps information up-to-date and repository services manage versions for each publishing target. This provides a very light application footprint for content delivery.

A decoupled architecture is also much easier to scale. With decoupled you can easily deploy to public clouds, content delivery networks, or platform-as-a-service solutions such as Microsoft Azure websites that provide built-in auto scaling. And because a decoupled CMS does not require a database on the web server, it is much faster, immune from SQL injection and other denial of service threats.

2 Development Model: MVC vs. Web Forms

Let's talk about how your developers code your front-end experience by examining the two typical approaches in ASP.NET: Web Forms and MVC.

ASP.NET Web Forms is the traditional development model for ASP.NET. Microsoft created Web Forms to help developers more familiar with client server development (e.g., WinForms) quickly migrate to building web pages through a visual RAD interface.

The problem with Web Forms is that it supports a tightly coupled architecture, one where the interface is integrated with the application functionality (code behind). This means the application code and the interface are not easily reusable. With Web Forms, you are also locked into the controls available for the CMS platform, dependent on their quality, upgrade cycle and HTML output standards, or you must spend a lot of time creating your own customizations that then require maintenance.

MVC (model view controller) provides a different architectural pattern for development. MVC is the more modern approach to ASP.NET development and is the future of ASP.NET with full support from Microsoft.

With MVC there is a clear separation of presentation from application logic, enabling the reuse of both. With MVC, a request is first sent to the Controller which then decides which Model (application logic/validations) and UI (View) to put together to create the appropriate front-end interface.

Another difference between Web Forms and MVC is that you can use Microsoft Web Pages (Razor) in MVC. Razor is a lightweight view engine that Microsoft recommends you use. With Web Forms, your only template option is via an aspx page.

You can see the importance of an MVC development model if you need to support cross channel, multi-device websites and web applications. Using MVC, you can create

multiple interfaces, and MVC knows which one to use based on the initiating request. In the case of Web Forms, you must build all your different interfaces and supporting application code separately. If all your CMS supports is Web Forms then you also have no way to send content to external applications - all your web apps need to be built directly on the CMS.

Finally, you want to make sure that if the CMS comes with a page builder application (an app that lets you quickly design your web pages), it supports an MVC model. Otherwise, any advantages you think you are getting from the page builder are quickly lost.

3 Structured Content vs. Blob Content

In a multichannel, multi-device world, the ability to reuse content is critical. Content reuse takes a variety of forms. It could be the updating of content on multiple websites or mobile devices, supporting multilingual requirements, or it could involve content for both digital and print. The idea of creating and managing separate versions of content for each channel/device is wasted time and effort.

To reuse content, you must structure it. Also known as intelligent content, structured content simply means content is stored in a way that defines and describes it.

The opposite end of structured content is Blob (binary large object) content. Many CMS platforms continue to store content in Blob format. Essentially, you have this large WYSIWYG editing environment where you write the entire content of your page, including images, multimedia and maybe some documents, and it's all stored as a Blob in the database. How do you know what this content is? How do you pull it apart to display it differently for mobile versus the website? How do you automatically resize images for mobile? The questions are enormous, and the answer is: you can't.

Traditional Web content like what we've described above is HTML and only describes what the content looks like. Structured (intelligent) content is XML, JSON (Java Script Object Notation), or XHTML with additional tag sets that describe what the content means.

Structuring intelligent content requires using human readable tags that applications also understand and know how to process. You typically apply business logic to content processing at the presentation layer. For instance, a style sheet or ASP.NET view would know how to present a <Title> tag as an H1 for a web page; and how to apply a separate set of mark up for a print doc or specific mobile device.

You can also apply logic to structured content at the application layer. Render content dynamically using audience segments, visitor behavior, device types, business rules, and other factors. In this case, structured content supports personalization across multiple channels. What does this mean for your selection of a CMS? Many CMS platforms continue to store content in Blob format. Others offer a combination of both. But if you really care about structured content (and you should if you want to create the best customer experience without a huge amount of wasted effort), you will want a CMS that creates your content using an XML schema and stores the content using a very granular set of tags, content, and meta data. Moreover, content should be fully separated from the presentation, use taxonomy or categories to define topics, and chunk the content either in elements (structure within a page) or components (XML fragments you assemble to create a dynamic page).

4 Database: NoSQL vs. SQL Server

Nearly all ASP.NET CMS platforms use Microsoft SQL Server for a database engine. In some cases, they might use Oracle or MySQL. These are relational database platforms that store web content, customer data and analytics data using a standard model and schema. In a relational database, content is typically stored as Blobs.

In an agile decoupled content management and deployment model, a relational database presents challenges. A Blob does not provide the structured (or intelligent) content model needed. What's required is a flexible content or data model, offering a richer content structure, and that's where NoSQL can help.

NoSQL databases – also described as "not only SQL" - are schema free, meaning content can have any structure and that structure can change over time. In NoSQL, content is stored as documents or JSON objects that have rich metadata and use search-based indexing allowing content to be easily queried and retrieved using metadata.

There are many advantages to a NoSQL database for content management. NoSQL is:

- Designed to manage content and provide a much more intelligent content model for storage and reuse.
- Schema-free so content definitions can change without database upgrades or "joins."
- Less expensive and easier to administrate.
- Easier to cluster, scale and geographically distribute through the provision of automatic "sharding" of information.
- Built for cloud-based deployments.

Bonus: Don't Forget the Support Model

When companies are looking for a new CMS, the technology always seems to take precedence. But there's something else that is very important and deserves more than a simple check mark and cursory response on an RFP form: the Support Model.

Building a new website or web application takes a lot of work, and it's fraught with challenges. Yes, you will have issues, and you will need your CMS provider to help you. Hence the support they provide.

But all support models are not alike. While many may tout a platform that is agile, they don't also provide agile support. When you are under the gun and need to get help, you don't have time to look at how many incidents you are allowed, or if you've used up all the critical, or priority, tickets. And there will be times when you want a real voice on the phone.

Be very careful to understand the support model a CMS provider offers and if it meets your needs. A new team will need more support than normal. A team working with a very new development approach (maybe you are switching from Web Forms to MVC) will require more support. If you are moving to structured content and struggling to figure out how to implement your content model in your CMS, you'll need support.

There is a wide range of reasons you'll want quick and easy access to your CMS provider's support team. Ask about turnaround times, number of priority tickets allowed, and access to phone support, community forums, and more.

If content is critical to your business, you want to have an unlimited technical support plan with both phone and online service, a Service Level Agreement (SLA) with guaranteed response times (ideally one hour or less for critical issues), and access to developer support. If you are using a hosted or SaaS solution, it is essential to have 24/7 support and monitoring.

About Ingeniux

Ingeniux is the leading provider of web content management and digital experience software. We enable organizations to orchestrate the entire customer experience from acquisition through to sales to support and service, across any device, application, or website.

We build content management software with an unparalleled focus on the content itself. The Ingeniux CMS is designed to manage and deliver modern websites, customer support portals, online communities, and other customer touchpoints.

We believe in intelligent “structured” content. We design our software to enable content reuse, enable true mobile and multi-channel content delivery, and insightful content discovery. Our unique content-as-a-service capabilities deliver content into web and mobile applications, and other key channels.

Ingeniux software is available as a fully managed software service or an on-premise application. Ingeniux delivers unparalleled service and support to customers worldwide.

To learn more, visit us at **http://www.ingeniux.com**.

INGENIUX

1601 2nd Avenue, Suite 1010
Seattle, WA 98101

info@ingeniux.com
877 445 8228