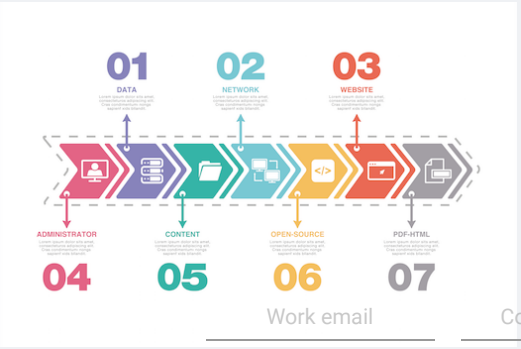

History of Content Management Systems and Rise of Headless CMS

Dec 18th 2018, Brent Heslop

To understand how content management systems (CMS) first came on the scene and why there are different types, let's look back at how content has evolved on the web.

DOWNLOAD The Ultimate Guide to CMS NOW.

Web 1.0 Managing Static Web Content

Web 1.0 is the term used to refer to the first stage of development on the World Wide Web that was characterized by simple static websites. The history of content management systems began in 1989 when Tim Berners-Lee proposed an internet-based hypertext system HTML and wrote the browser and server software in late 1990. HTML came from SGML, which stands for the Standard Generalized Markup Language, and was created at IBM by Charles F. Goldfarb, Ed Mosher, and Ray Lorie in the 1970s. The first websites were simple HTML text files. You used an FTP program to copy the files to a directory under a running web server. In 1993, Mosaic browsers began supporting images that could appear along with text, and static brochure-like sites shared company and product information.

In the early 1990s, the first step to managing content on a web page came with Server Side Includes (SSI). Server Side Includes let you keep portions of your site separate from the main content, such as the site menu or a footer. Around the same time, the Common Gateway Interface came on the scene that let you create interactive forms.

As early as 1990, Tim Berners-Lee said the separation of document structure from the document's layout had been a goal of HTML. In 1994, Håkon Wium Lie worked at CERN and using the Web for publishing was growing. However, it wasn't possible to style documents, such as displaying a newspaper-style multi-column layout in a Web page. Lie saw the need for a style sheet language for the Web. Later Lie was joined by Bert Bos who was building a customizable browser with style sheets. By 1995 the World Wide Web Consortium (W3C) was up and running, and Lie and Bos worked together at the W3C on the first style sheets recommendations.

In August 1996, the first commercial browser to support CSS was Microsoft's Internet Explorer 3. The next browser to support CSS was Netscape Communicator, version 4.0. Netscape's initial implementation to support CSS was more of an attempt to stop Microsoft from claiming to be more standards-compliant than Netscape. Unfortunately, the Netscape browser would frequently crash when the page included Cascading Style Sheets. The battle for controlling standards between Netscape and Microsoft came to be known as the browser wars.

In 1996, ColdFusion added a full scripting language called CFML. Processing forms with ColdFusion or using

the Common Gateway Interface and programming languages like Perl and Python became the norm. From 1995 to 1997, server-side scripting was the rage. During this same time, Personal Home Page (PHP) and Active Server Pages (ASP) came into play with server-side scripting for generating content sent from the server to the Web browser. Similar to ASP and PHP, JavaServer Pages (JSP) arrived on the scene later in 1999 and was built around the Java programming language and was also fairly popular.

In 1997, Microsoft introduced iframes that let you split the HTML browser window into segments, with each frame showing a different document that could be used to display content from other sites, and was popular for presenting ads and banners. The iframe tag brought with it security, navigation, and search engine optimization issues that eventually were addressed.

The DOM and Dynamic HTML Revolution

The turning point came in 1997 as dynamic content came into its own with the introduction of the Document Object Model (DOM). The DOM defines the logical structure of documents that lets you identify and programmatically control parts of a document. The DOM is an application programming interface (API) for HTML and XML documents. For example, the DOM lets you access and manipulate the styles of HTML elements like the entire body (body) or a division (div) on a page.

Dynamic HTML using Asynchronous JavaScript and XML, commonly called Ajax, was a revolutionary breakthrough letting developers request and receive data to update a Web page without reloading the page.

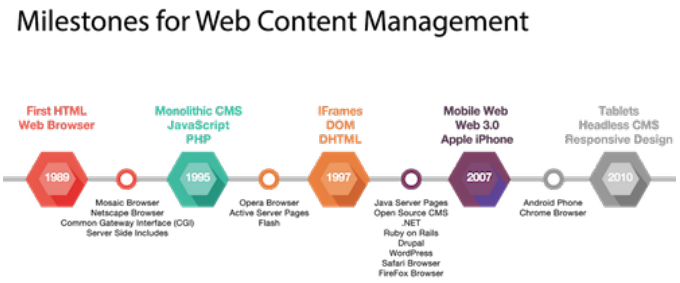

Figure 1. A Timeline of Milestones for Web Content Management

Web 2.0 and the Role of a CMS

Dynamic content delivery brought with it new ways to present and interact with content on the Web, with an emphasis on sites being more social. The term Web 2.0 helped define what is also called the participative or participatory and social web. Web 2.0 also refers to the surge in user-generated content and the ease of use to make websites work with other products and systems.

As the web moved from being static brochure sites to interactive sites with dynamic content, the desire for collaboration and fresh, relevant content grew, and the need to manage content came to the forefront. Websites needed to be updated daily, with different people wanting to add and edit content. For example, then Marketing Department wants to update promotional material, Human Resources needs to post new jobs, the Public Relations Department needs to post press releases, the Docs Department needs to publish product documentation, the Support Department wants to interact with customers online, and so on. The role of a content management system was to provide the capability for multiple users with different permission levels to manage content for a website or a section of the content.

The Core Components of a CMS

There are two core elements of any content management system (CMS): The Content Management Application (CMA) and the Content Delivery Application (CDA). A CMA for website content allows for the administration of users and groups so that they can create, edit, and remove site content. The CMA also includes the front-end user interface that allows a person to add, modify, and remove content from a Web site without requiring knowledge of HTML, Cascading Style Sheets (CSS), or programming languages, thus eliminating the involvement of a developer. The Content Delivery Application (CDA) compiles that information and updates the website.

The Rise of the Monolithic CMS

It was apparent that a system was needed that would allow individuals and groups to manage and deliver content to the web. A monolithic CMS is a system that incorporates everything required for managing and publishing content to the Web. This type of CMS is a coupled system, meaning that it is an all-in-one content-management solution. We will take a more in-depth look into the difference between coupled and decoupled systems later in this article.

Founded in 1985, FileNet is considered to be the first system that was a real content management system. In 1995 FileNet introduced a complete integrated document management suite of programs with document imaging, document management, and workflow. Vignette came on the scene in late 1995 with the goal of making web publishing more accessible and more personalized, and is commonly credited for originating the term “content management system.” A year later Vignette introduced StoryBuilder. Many enterprise CMSs began to appear around this time including, Interwoven (1995), Documentum (1996), FatWire (1996), FutureTense (1996), Inso (1996), and EPiServer (1997).

Open Source CMS and Frameworks

By the early 2000s, content management systems dominated the web. Open source content management systems and frameworks began to appear. A framework is a programming library of pre-written code, such as the then-popular Zend framework written in the PHP programming language. OpenCMS, PHP-Nuke, Mambo, WordPress, Drupal, Plone, and Joomla all offered free alternatives for content management. WordPress gained popularity as an open-source solution focusing on blog content delivery and letting third-party developers add customizations and extensions. In 2006, Alfresco offered an open-source alternative to enterprise content management.

The Website-Building Platform Surge

Starting in 2003, easy to use website-building CMS sites offered premade templates for people who had no coding experience, such as WordPress (2003), SquareSpace (2003), followed later by Weebly (2006), and Wix (2006). While not pure content management systems, these building platforms provided a path to building a small, low-cost website that required no knowledge of HTML, CSS, and coding.

Web APIs, XML, and JSON

A large part of Web 2.0 was making websites work with other products and systems. A Web API is a Programming Interface that allows access to a system, such as a website through standard HTTP request methods. The data is typically wrapped in a standard format, such as XML or JSON to make it easy to read and

work with.

XML stands for eXtensible Markup Language that is a data format. Like HTML, XML is a descendant of SGML, the Standard Generalized Markup Language. XML allows for transporting data through feeds and API calls because it's a platform-independent format.

JSON stands for JavaScript Object Notation that is a format of storing serialized data with key-value pairs and transmitting that data between a server and a web application. JSON feeds can be loaded asynchronously much more easily than XML and RSS feeds. Some sites, such as Twitter provide RSS feeds, which are easy to use on the server-side but frustrating on the client-side, since you cannot load an RSS feed with AJAX unless you are requesting it from the same domain on which it is hosted. JSON also gained preference over XML since it has a smaller footprint, is easier to use, and works great with JavaScript-enabled browsers since JavaScript automatically recognizes JSON.

SOAP and REST

To communicate object information back and forth for social and e-commerce sites, developers often use machine-based interactions, such as REST and SOAP. SOAP stands for Simple Object Access Protocol. REST stands for REpresentational State Transfer. REST is an architectural style, whereas SOAP is a protocol. An architectural style specifies guidelines that a developer must follow to be considered a RESTful API, including that it supports a client-server model, be stateless, cacheable, have a uniform interface, and be a layered system. A layered system is one where you can keep data on different systems, so your APIs can be on one server, data on a second server, and use a third server to authenticate requests.

Developed in the early 1990s, SOAP did not come into the mainstream until the early 2000s. SOAP is a standardized, extensible, XML-based messaging protocol that is language-, platform-, and transport-independent with built-in error handling. SOAP uses Web Services Description Language (WSDL), which is a service description language. SOAP uses Web Services Description Language (WSDL), which is a service description language used to provide web services over the Internet. The WSDL specifies the available functions, so a client program can connect and discover the functions offered by the web services.

SOAP is not as popular today and is being replaced with new APIs, such as REST and GraphQL. SOAP works well in distributed enterprise environments and is still used for B2B applications because you can define a "data contract" with it. However, in the web world, 70% of public APIs are RESTful APIs. When a RESTful API is called, the server will transfer to the client a representation of the state of the requested resource. REST uses multiple standards like HTTP, JSON, URL, and XML. A REST API uses a Web Application Description Language (WADL), and it doesn’t require the extensive processing SOAP does, so it is faster. It is also easier to use and more efficient and flexible than SOAP. RESTful web APIs are typically loosely based on HTTP methods to access resources via URL-encoded parameters and the use of JSON or XML to transmit data. JSON ensures reliable, fast, and easy data exchanges, so it is the most common data exchange format for working with RESTful APIs.

Going Mobile with Web 3.0

In the late 1990s and early 2000s Nokia Symbian, Palm, and Blackberry mobile devices provided access to the Web. However, it wasn’t until the introduction of the iPhone in 2007 and the Android smartphone in 2008 that mobile phones really had an impact on delivering web content. In 2010 smart tablets came on the scene. REST APIs and JSON data format were vital to delivering content to mobile devices. This megatrend of delivering content to mobile devices ushered in the mobile web era, which has also been called Web 3.0 to identify the shift from computers and laptops to mobile content delivery. By the beginning of 2014, mobile internet use exceeded desktop use in the U.S.

This rise in content consumption by mobile devices presented a problem for the monolithic CMS that was explicitly created for delivering Web content to desktops and laptops. There was no way to deliver content for both desktop and mobile devices reliably. To address the rise of mobile web usage, developers began creating both desktop and mobile versions of their websites, with mobile designs offering stripped-down versions of select desktop website pages.

The mobile sites were on a separate subdomain and called mobile or “m.dot” sites since the subdomains would end in “.m.” One problem that arose is that Google does not provide indexing of m.dot sites. Instead, Google only annotates the m.dot URLs to say the main website is mobile-friendly.

In 2010, Ethan Marcotte introduced the term “responsive design” that promoted a shift in thinking from the fixed design for desktop websites to responsive, fluid, adaptable layouts. To deliver on the promise of responsive design, the W3C created media queries as part of the CSS3 specification. A media query allows developers to ascertain the type of device and inspect the physical characteristics of the device, such as the screen size. For example, using CSS you can use the @media rule to determine what screen size is being used and include a block of CSS properties for that device.

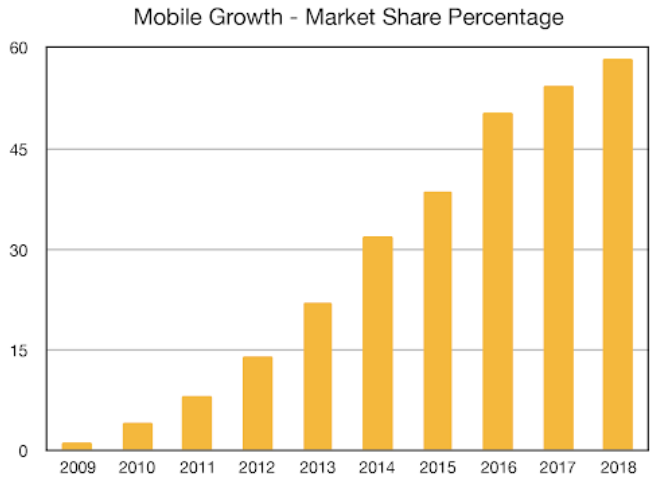

Figure 2: Worldwide Mobile Growth
Source: StatCounter Global Stats (http://gs.statcounter.com/)

The Paradigm Shift to Omnichannel

The word “omni” means “all things” in Latin, so omnichannel refers to all possible channels. Just as the mobile channel was disruptive to the delivery of web content, new channels, such as smartwatches, gaming consoles, voice-activated devices like Amazon’s Echo (Alexa) and Google Home are continually appearing that present content delivery problems for the traditional CMS. The paradigm shifts—from delivering content for a few channels to true omnichannel content delivery that is flexible enough to support whatever tomorrow’s channels may come on the scene—demand a better solution, which was the decoupled and headless CMS solution.

The Decoupled and Headless API-First CMS Solution

A decoupled system consists of two or more systems that can transact without being connected, similar to the separation of an HTML (content) file from a CSS (formatting) and a JavaScript (programming) file. A decoupled CMS allows developers to make changes to the presentation (formatting) and behavior (programming) layer without affecting the content of the site.

The term decoupled and headless are frequently used interchangeably, but there is a difference. A headless

CMS does not have a front-end system or presentation environment. A headless CMS is API-first, which means it integrates content management tools via API. Separating formatting from content allows you to publish content to any device or channel. A decoupled CMS typically includes a front-end formatting system of templates. A headless CMS separates managing content from presenting formatted content; so in other words, it removes the interdependency of presentation and behavior layers from the content. Moving from a coupled system to a decoupled headless CMS opens up a new world of managing content.

The Content Hub Architecture

Key to the success of working with a headless CMS is the content hub architecture. A content hub centralizes all your content in one place using an API to deliver content anywhere. This content-centric approach accelerates and simplifies content management, letting your developers use the best-of-breed tools to create digital experience platforms (DXP) with omnichannel content delivery to help create more personalized customer journeys and more impactful digital experiences.

Figure 3: The Content Hub Architecture

Integration and Frameworks

Not only does the content hub architecture help you with omnichannel content delivery, but using a content hub also gives you more freedom for integration. Using a headless CMS following the content hub architecture lets you choose the best of existing tools or services, such as marketing automation tools, analytics, a personalization engine, translation services, video delivery services, e-commerce platform, and AI extensions.

> The world of technology is constantly changing at a rapid rate. There is always a new way of capturing and delivering customer data better, faster, and cheaper. Integrations with a headless CMS with a content hub architecture makes it much easier to be agile and switch to new tools and services without disrupting your

content or content delivery.

Figure 4. Integration and the Content Hub

Security and CMS Solutions

Most headless CMS offerings fall in the Content as a Service (CaaS) category, meaning the service is centralized and hosted on the Cloud. As with any CMS, you put your trust into your CMS vendor. This trust applies to any third-party applications that you integrate into your CMS as well. The benefit of using a reputable company with supported integrations that you can trust minimizes risk and ensures a safer more secure site.

Scalability

Using a traditional CMS to handle increases in traffic is a typical solution to add multiple servers running the CMS. This is time-consuming and expensive. A headless CMS has the ability to scale and additionally avoid database bottlenecks that you are likely to encounter using a traditional CMS. Scaling is much easier to do with a headless CMS since most headless CMS offerings are Cloud-hosted, so it is possible to automatically adjust your Cloud infrastructure to match demand.

Another important technology for being able to deliver content fast and on a global scale is the Content Delivery Network (CDN). A CDN is a network of servers spread around the globe. Static assets and dynamic content of your website are cached and saved on all the CDN's servers. When a person requests a page, the website retrieves cached content from the nearest CDN server and delivers it to the client. Having a CDN-enabled headless CMS vastly improves the performance of delivering content around the world.

The Digital Experience Platform

Gartner defines a digital experience platform (DXP) "as an integrated set of technologies, based on a common platform, that provides a broad range of audiences with consistent, secure and personalized access to information and applications across many digital touchpoints. Organizations use DXPs to build, deploy and continually improve websites, portals, mobile and other digital experiences." The headless CMS approach is quickly becoming a crucial component of the new generation of Digital Experience Platforms (DXPs). DXPs go significantly beyond web content management to create rich, engaging experiences for audiences addressing a

multitude of channels. This ties in nicely with the content-hub architecture to enable any type of integration needed to deliver content to any channel.

The Importance of Personalization

Personalization is key to building an effective Digital Experience Platform. Personalization means understanding your visitors' interests and tailoring content to fit their needs and preferences, providing them with an experience they find relevant. The more relevant a person finds your message the more you increase customer loyalty and revenue. Personalization is a mission-critical marketing activity. Using a headless CMS, personal data is made available via APIs, web services, and open data standards, so you are not tied down by data stored in a pre-built system.

Personalization tools and services, such as Optimizely, Monetate, One Spot, Evergage, Salesforce Commerce Cloud, and Adobe Target all help you track and act on a visitor's behavior, location, profile, and other attributes to create a dynamically personalized, highly relevant experience. By creating a more meaningful experience for your visitors you're also generating better business results.

Artificial Intelligence and Machine Learning

In July 2018 at a Town Hall meeting in San Francisco, Google CEO Sundar Pichai called artificial intelligence "one of the most important things that humanity is working on," saying that it is "more profound than electricity or fire." Artificial Intelligence (AI) and Machine Learning (ML) are ushering in the next era of digital transformation. All major tech companies are following this mega-trend. Google offers TensorFlow; IBM, Watson and AI OpenScale; Adobe, Sensi; SalesForce, Einstein; Amazon Lex and Amazon Rekognition; Microsoft Azure Cognitive Services; and Facebook announced it is expanding its AI-research division to roughly 170 scientists and engineers.

AI and machine learning are already having a significant impact on content management. Integrating your content hub with AI and machine-learning tools and services can help you discover hidden opportunities, speed up processes, and most importantly, offer relevant digital experiences to customers. Personalization engines are using AI and machine learning to deliver smarter, customized, and predictive customer experiences. In addition to the personalization services mentioned in the previous section, some examples of using content management with AI and machine language services and tools such as SEO optimization with CanIRank, MarketBrew, and BrightEdge; content creation and text analysis with MonkeyLearn, Acrolinx, Automated Insights, and Narrative Science; and translation services with KantanMT, and SYSTRAN.

The Headless CMS Solution

There will always be disruptive technologies that will change the CMS playing field. There is no doubt that AI and machine learning are going to play a huge role in the future of content management. The primary goal is to build the best digital experience platform with omnichannel delivery that is secure, scalable, and as future-proof as possible, such as Contentstack. By allowing you to integrate with new technologies and applications as they come on the scene, a headless CMS is likely to be the longest-lasting solution in the history of content management systems.