Web Applications vs. Traditional Applications

by Devmountain Tutorials

WHAT'S COVERED

This section will explore the history of web technologies and the current design of web applications by discussing:

1. HISTORY OF WEB TECHNOLOGIES
2. CLOSING THE GAP BETWEEN TRADITIONAL APPLICATIONS AND WEB APPLICATIONS
3. ENGINEERING THE MODERN WEB

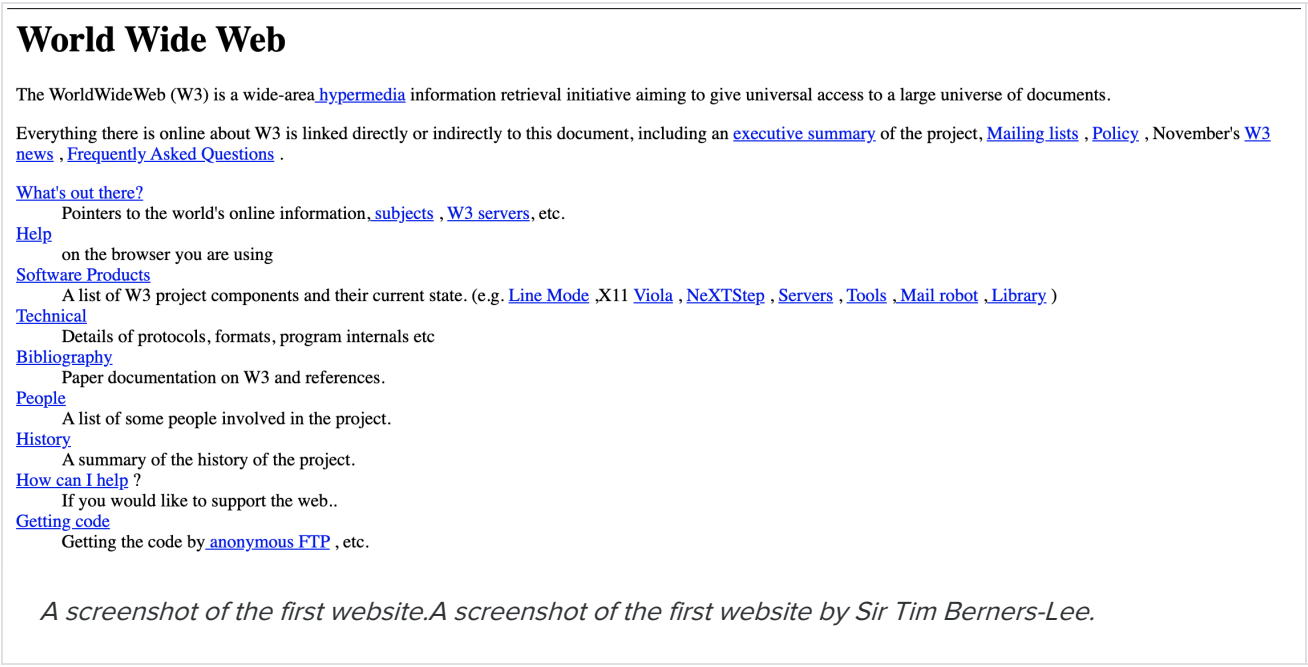
1. HISTORY OF WEB TECHNOLOGIES

Remember the comparison between Google's homepage in 2000 and their modern homepage from the beginning of Unit 1? The evolution of Google's homepage exemplifies how drastically the technological landscape of the web has changed in the past couple decades. The history of web technologies — from the rapid growth of the Internet to the technological advancements that followed — has shaped the way websites evolved from simple, early web pages to complex, modern **web applications**.

As websites grew more complex, new methodologies and best practices emerged to help engineers handle that complexity. So, in order to understand web engineering today, we need to start at its humble beginnings.

Information on this topic is sparse and contradictory so there may be a few, minor inaccuracies here and there. It is interesting to note how the documented history of the web, being so full of inaccuracies and anachronisms, mirrors the way the web and the technologies that came out of it grew themselves. A you continue reading, you'll learn how the web — which started as a surprisingly primitive technology — quickly grew into the complex amalgamation of technologies that make up the web we know (and likely take for granted) today.

Sir Tim Berners-Lee invented the World Wide Web in 1989 while he was a researcher at the European Organization for Nuclear Research, better known as CERN. There, he published the first website — a humble collection of information on the World Wide Web project itself.



World Wide Web

The WorldWideWeb (W3) is a wide-area hypermedia information retrieval initiative aiming to give universal access to a large universe of documents.

Everything there is online about W3 is linked directly or indirectly to this document, including an executive summary of the project, Mailing lists , Policy , November's W3 news , Frequently Asked Questions .

What's out there?
Pointers to the world's online information, subjects , W3 servers, etc.
Help
on the browser you are using
Software Products
A list of W3 project components and their current state. (e.g. Line Mode ,X11 Viola , NeXTStep , Servers , Tools , Mail robot , Library)
Technical
Details of protocols, formats, program internals etc
Bibliography
Paper documentation on W3 and references.
People
A list of some people involved in the project.
History
A summary of the history of the project.
How can I help ?
If you would like to support the web..
Getting code
Getting the code by anonymous FTP , etc.

A screenshot of the first website.A screenshot of the first website by Sir Tim Berners-Lee.

DID YOU KNOW

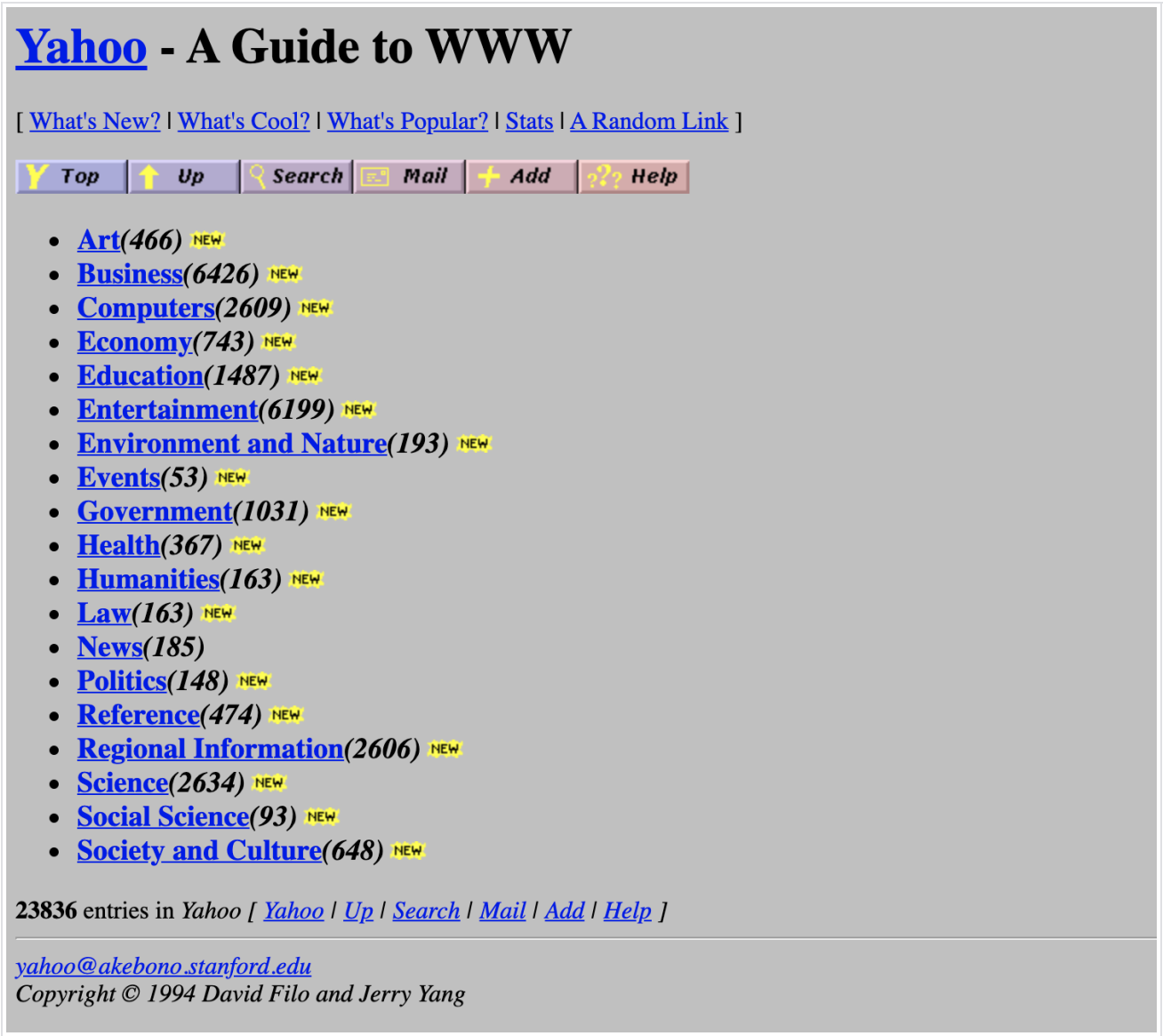
Unfortunately, the original 1990 website wasn't preserved but you can still visit the 1992 copy of the website at its original URL: http://info.cern.ch/hypertext/WWW/TheProject.html.

In the early 1990s, the Internet made it easier for researchers and academics to share documents, data, and software across research institutions and university campuses. For example, from a computer at Cambridge University, a user could view and download files stored in a computer at Stanford University. Due to this initial use-case (and thanks to low bandwidth and slow connection speeds) the most websites looked a lot like the world's first website — they were plain and boring, consisting only of text and hyperlinks. Users accessed websites using a handful of web browsers that were so primitive, they weren't even capable of displaying text and images on the same screen. That is, until Marc Andreessen, a college student working part-time at the National Center for Supercomputing Applications (NCSA) at University of Illinois at Urbana-Champaign got bored. In late 1992, Andreessen started working on Mosaic, a web browser that would trigger an Internet revolution and change the world forever.

Mosaic 1.0 running on a Mac OS 7 computer.

When Mosaic was published in September 1993, it made the web more accessible to non-technical users than

ever before. Before Mosaic, if researchers wanted to distribute a paper with an image like a diagram via the Internet, they couldn't embed the diagram alongside the document's text. The only thing they could do was add a link to the diagram. Then, if readers wanted to view the diagram, they'd have to click on the link, download the image, and figure out how to open it. This might not sound so bad, but computers weren't nearly as user-friendly back then as they are now. Mosaic's had the ability to display images alongside text, so users didn't have to go through the song and dance of downloading images and opening them by hand. This also gave web designers more freedom to customize the look and feel of their websites.



A screenshot of Yahoo.com from 1994. Notice the images (the yellow symbols in the navigation buttons at the top and the yellow graphics with the word "NEW" in them) and how they appear alongside text.

Just over a year after leading the Mosaic development team at NCSA, Marc Andreessen left, taking some of his NCSA colleagues with him, to create a startup named Netscape Communications Corporation (originally, Mosaic Communications Corporation). Andreessen and his colleagues brought their initial motivation for creating Mosaic — to make the web more appealing for non-technical people — over to Netscape. In 1994, Netscape officially released Netscape Navigator. It became the most popular web browser of the 1990s.

A screenshot of Mosaic Netscape 0.9 on Windows XP

At this point in web history, websites were closer to books than computer applications. The only difference between books and the first websites was that, on a website you could navigate from one page to the next by clicking a hyperlink instead of turning a physical page. Otherwise, web pages were static; once a page was finished loading, that page would never change. In contrast, computer applications were dynamic and interactive. For example, imagine any computer application that allows you to write and edit text. All text editors allow users to add text by typing on their keyboards. As they type, letters appear on the screen, changing the contents of the screen. It would be impossible to recreate this behavior using early web technologies because web pages couldn't change. If engineers wanted to close the gap between static web pages and traditional computer applications, they'd need a language that would allow them to manipulate the contents of a page and program dynamic behavior and the engineers at Netscape would be the ones to create it.

2. CLOSING THE GAP BETWEEN TRADITIONAL APPLICATIONS AND WEB APPLICATIONS

In 1995, Netscape set their sights on adding such a programming language to Navigator. Software engineer Brendan Eich, while working for Netscape, created **JavaScript** in May. He'd famously draft a complete prototype of the language in 10 days. Later that year, Netscape included JavaScript with the newest version of Navigator. In 1996, Microsoft released their own version of JavaScript with the latest version of their browser,

Internet Explorer. However, JavaScript would not gain widespread usage and become an integral part of the web engineering toolkit until 2000.

DID YOU KNOW

JavaScript is often confused with Java, a programming language by Sun Microsystems (now Oracle). They are completely different languages. Netscape chose to name their new programming language "JavaScript" for marketing purposes — they thought if their language sounded like it was associated with Java, it would become more popular.

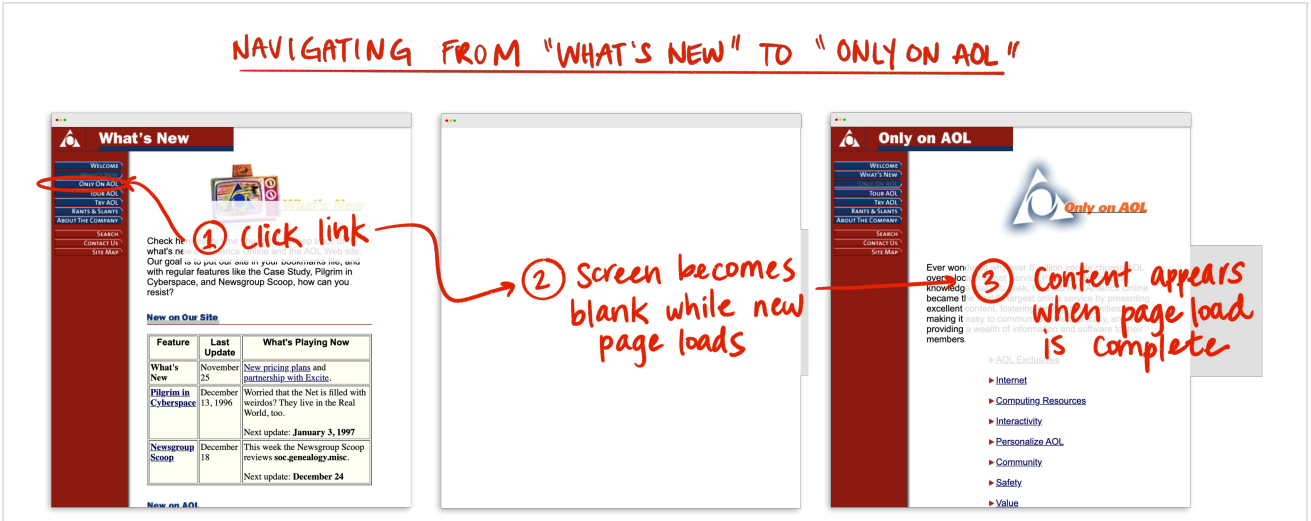
By the mid-1990s, the web was no longer a niche technology for academics and tech-savvy early-adopters. People began to see it not just as a source of entertainment but also as a new, lucrative space for commerce. With more and more websites popping up online, web developers wanted to create web pages that looked unique to make their websites visually distinct from others.

Visually distinctive web pages came at a cost, though. As web pages grew in complexity, the time it took for a page to load its contents increased, and websites became more difficult to navigate. Earlier, we mentioned that before the advent of JavaScript, web pages were static — once the browser finishes loading a **static web page**, it never changed. When you navigate from one static web page to another, your browser will treat each page as if it's unrelated to the one before it, even if both pages are from the same website. This would happen even if the web pages shared common visual elements like navigation menus or sidebars.

Even though JavaScript had been incorporated with the latest versions of major web browsers by 1996, it was still a new technology, so developers were only beginning to understand how to improve user experience through JavaScript. The problem was that every time users navigated to another web page, their browsers would have to retrieve the entire contents of that new page in full. This would happen even if the web pages shared common visual elements like navigations menus or sidebars. For example, the figure below is a side-by-side comparison between two different pages on an archived version of www.aol.com from 1996. Notice that the title and content of each page is different but the red sidebars are exactly the same.

Side-by-side comparison of two AOL webpages

Even though the sidebars are exactly alike, web browsers would still load each page as if it were completely new. Every time browsers pulled in “new” pages of content, all existing content disappeared, and users would have to wait for the next page’s content to finish loading. This would happen for every single user action, even if that action changed only a small part of the page. Even worse, internet speeds were painfully slow which compounded the negative impact of this inefficient process on user experience.



Navigating from "What's New" to "Only on AOL"

A more efficient loading strategy was needed — if engineers could designate when and how often parts of a page should load, they’d be able to program the browser to load common elements like navigation menus once. That way, the browser would only have to retrieve and reload the content that had changed between

one page and the next. Today, this technique is known as **AJAX**, which stands for **Asynchronous JavaScript and XML**. On paper, AJAX seems like an obvious solution. In fact, JavaScript already supported the ability to program loading different parts of a web page at different times by as early as the late 1990s.

Unfortunately, these technologies were fairly obscure and would remain underutilized until the early 2000s. That's when large companies like Microsoft and Google used AJAX to build some of the first large scale web applications. In 2000, Microsoft published Outlook Web Access which allowed users to read and write emails right from within their web browser. In 2004, Google followed suit with a web-based email application of their own, Gmail. In 2005, web interface developer Jesse James Garrett published a blog post titled **Ajax: A New Approach to Web Applications**. Garrett's article was the first time the acronym "AJAX" appeared publicly. In it, he described how the content loading and manipulation technique worked, listing websites using AJAX that were popular at the time, and outlined how AJAX might be used in the future. His article spread awareness of AJAX across the wider web engineering community. It was no longer a technique exclusive to engineers at well-funded tech companies. Finally, professional and amateur web engineers alike had the tools and knowledge to create websites that didn't just resemble traditional computer applications but could even replace them.

The spirit of rapid innovation and experimentation on the web didn't stop with the development of AJAX; it continues today. It's hard to believe that the web started as a small collection of websites owned by research institutions and the Internet was simply used to exchange documents from one research institution to another. It might be even harder to believe that, in general, the World Wide Web runs on the same technology today as it did in 1989. Today, the Internet is still just a computer network used to transfer data from one place to the next. Its applications are far more diverse though. The web isn't just a platform for academics and their research papers any more — it's now used for social networking, shopping, video gaming, streaming movies, and so much more. It's amazing that a platform is able to support such a huge variety of users and activities.

This is the last section on the history of the web; after this paragraph, we'll depart the past and talk about the present state of web engineering and development. Before you go, we'd like to leave you with one last observation — that the web was never designed to stream movies, support social networking, or become a place for ecommerce. Tim Berners-Lee's motivation for inventing the World Wide Web wasn't to create an all-in-one platform generic and powerful enough to become the birthplace of completely new industries. He was just frustrated with the difficult task of finding information stored on different computers and wanted to invent a system that would make it easier to locate certain resources. The web, as we know it, only exists thanks to a series of coincidences: Marc Andreessen created Mosaic, the mother of all web browsers, as a lark because he was bored at work; Netscape invented JavaScript just to compete with Microsoft's Internet Explorer; Brendan Eich accidentally made JavaScript flexible enough to withstand and support years' worth of language updates and revisions. It's amazing, and probably a little amusing, that the web even works at all.

Traditional applications and Web applications
Here, we use the term *traditional application* to refer to desktop applications — which run on operating systems like Windows or MacOS — and mobile applications — which run on Android and iOS. In contrast, *web applications* run in web browsers.

JavaScript
The first programming language of the web. Web developers use JavaScript to program dynamic behavior on a website, which made it possible for developers to create web applications.

Static Web Page
Web pages that loaded and never changed.

Asynchronous JavaScript and XML (AJAX)
AJAX is not a specific technology. Rather, it's a technique that developers use to decrease loading times and improve user experience. It is difficult to pinpoint the origin of the term but it was first publicly used by Jesse James Garrett in his article, **Ajax: A New Approach to Web Applications**.

3. ENGINEERING THE MODERN WEB

Web engineering has become less distinct from other forms of software engineering over time. Humans aren't the only ones going online; it seems like everything is connected to the Internet nowadays. Think of any type of product — cars, TVs, doorbells, clothing, even light bulbs — and you'll find a version of that product that's capable of using an Internet connection. It's not difficult to imagine a future where every object we interact with is networked somehow.

That's all to say how difficult it can be to pick apart web engineering practices that are exclusive to web engineers. Due to technology's increasing reliance on the Internet, the methods and best practices that were initially exclusive to web engineers can be found in many genres of software engineering and vice versa. However, there is one technology at the center of the web ecosystem that's exclusive to web engineering — the web browser. Even though there are many different web browsers, they're all expected to comply with the standards and specifications of the web. The advantage of web applications over traditional applications is that the relatively uniform behavior of web browsers makes it easy to create **cross-platform applications**, or applications that will work on different systems.

For example, let's say you want to create a cross-platform mobile application — an application that will work on Android phones and iOS phones. To develop an Android application, you'd have to use language compatible with Android devices like Kotlin. iOS isn't compatible with Kotlin though; instead, most iOS apps are written using Swift. Besides having to translate from one language to another, there are enough differences between both systems that you'd essentially have to create two mobile applications — one for Android and another for iOS. It's much easier to write a cross-platform web application because they're designed to run within web browsers. Although there are different types of web browsers, they all have the same core features and understand the same coding languages. The advantage of web applications is that, in general, developers only have to build one application instead of building multiple applications to account for different platforms.

Unfortunately, web browsers aren't completely uniform. According to Mozilla Developer Network's 2019 Developer Needs Assessment, the most frustrating part of web development is having to support different browsers, avoiding or removing a feature that doesn't work across all browsers, and encountering situations where the look and feel of a website isn't consistent across different browsers. A couple paragraphs ago, we mentioned that all web browsers understand the same programming languages. However, since web browsers are developed and maintained by different companies, each company is responsible for maintaining and updating their own browser. As a result, each browser has its own set of quirks and bugs that developers need to take into account in order to write code that is compatible with and consistent across all browsers. Luckily, there are many tools that will automate the process of making code compatible with different browsers. For example, some of these tools allow developers to take advantage of new language features, even when they're so new that browsers haven't implemented them yet, by transforming written code into

code that most browsers will be able to run. Still, these tools aren't fail-proof 100% of the time; at the end of the day, web browsers are the products of third-parties. Unless companies like Apple, Microsoft, and Google agree to implement the same updates and features at the same time, browser compatibility issues will likely continue to be the bugbear of all web developers.

There are many wildly successful applications on the web now, so it seems that dealing with browser compatibility issues is a small price to pay for cross-platform capabilities. Several prominent web applications you might recognize are Google Maps, Gmail, Google Drive, Facebook, Spotify, Slack, Discord, and YouTube. The next section will take a deeper dive into how applications like the ones we just mentioned are architected. This will give you an idea of how the once-humble website can become an advanced application by leveraging the Internet, web browsers, and other technologies.

TERM TO KNOW

Cross-platform applications
Applications that are compatible with different systems.

TERMS TO KNOW

Asynchronous JavaScript and XML (AJAX)
AJAX is not a specific technology. Rather, it's a technique that developers use to decrease loading times and improve user experience. It is difficult to pinpoint the origin of the term but it was first publicly used by Jesse James Garrett in his article, **Ajax: A New Approach to Web Applications**.

Cross-platform applications
Applications that are compatible with different systems.

JavaScript
The first programming language of the web. Web developers use JavaScript to program dynamic behavior on a website, which made it possible for developers to create web applications.

Static Web Page
Web pages that loaded and never changed.

Traditional applications and Web applications
Here, we use the term traditional application to refer to desktop applications — which run on operating systems like Windows or MacOS — and mobile applications — which run on Android and iOS. In contrast, web applications run in web browsers.