

# Choosing an IT Stack for Web Application Development

## Executive Summary

For building web applications there are three primary IT stacks in use today. **I suggest choosing the Microsoft .Net platform to build web applications whose architecture is most influenced by W3C standards.** The World Wide Web Consortium (W3C) is the main international standards organization for the World Wide Web (W3) which was founded and is headed by Tim Berners-Lee, who is largely credited for creating the World Wide Web.

Microsoft completely re-architected their development environment for the web releasing the first version of the Microsoft .Net environment in 2002. It is the most modern, web-centric of the three major IT stacks and it has extensions through the .Net framework libraries to Microsoft's server and client components.

Microsoft's has continued its history of being the leader in Rapid Application Development (RAD) by making the .Net platform the simplest environment for programmers. It could be argued that its' RAD capabilities are what won it the OS war, but that is another discussion.

Unfortunately with much of Microsoft's technology it has continued its history of being Windows centric. I would encourage ignoring the vast majority of Microsoft technologies that are operating system (OS) or IE (Internet Explorer) specific. **Fortunately, there are many Microsoft technologies that are not specific to a user's OS or type of browser; ASP.Net AJAX being a great example of this.**

By choosing the .Net environment and using the W3C as your primary architectural guidance you can get the best of both worlds; the best development platform and the best architectural guidance.

## IT Stacks

An IT stack consists of the server-side processing of web applications and includes: a server operating system, a web server that is installed on top of the OS to deliver web pages, a server programming language, a relation database management system (RDMS), and an Integrated Development Environment (IDE).

IT Stacks	Operating System	Application Server	Server Programming Language	Database Software	Programming IDE
J2EE	Unix (Linux, Solaris, HP-UX, or AIX)	BEA, JBOSS, Oracle, WebShere (IBM), Sun	Java, JSP	typically Oracle	Jbuilder, Eclipse, Jdeveloper, NetBeans
LAMP	Linux	Apache	PHP, Python, Perl	My SQL	Eclipse
.Net	Windows Server	IIS	C#, VB.Net, ASP.Net	SQL Server	Visual Studio

Clearly if starting from scratch, choosing a proprietary Unix OS is not a reasonable option. For similar reasons, when possible, choosing a proprietary Java application server should also be avoided. Currently Java is in flux as Sun has just

recently released it as open source. There is also a common argument that Java is bloated and not optimized for the web. For companies that currently use J2EE it is a fine platform, but if you have the ability to start from scratch there are better options. **For these strong reasons I'm going to label J2EE as a largely legacy platform and I would suggest focusing on .Net and LAMP.**

While some argue that there are other good options like Ruby on Rails (RoR). I cannot respect the argument that dynamic programming languages (languages that do not compile) can actually be an enterprise platform. Ruby on Rails is a framework built on a single JavaScript file (prototype.js), it is not a platform. An application built entirely with a dynamic programming language can scale up (more users), but struggles to scale out (more features). This is where .Net succeeds.

I will discuss AJAX frameworks and I have high complements for prototype.js and several other AJAX frameworks, and encourage their use, within the .Net or LAMP platforms. While there are many plug-ins and related technologies that go along with selecting a web application development platform choosing an AJAX or JavaScript framework is very important and is largely platform agnostic.

Only three databases are gaining market share and I would strongly suggest considering only these three databases: Oracle, SQL Server, and MySQL. Oracle offers the best performance and is the most expensive. SQL Server offers similar performance to Oracle in most circumstances, is the simplest to administer (following Microsoft's strong RAD philosophy), and is inexpensive compared to Oracle. If you do not have a highly data intensive application then MySQL is a great solution as it offers reasonable performance and the software is free. Although I would encourage you **to not focus on solely software costs as engineering and service costs are usually large multiples of the actual software costs.**

## Why being RAD is so important

**Building web applications is mostly an engineering expense, not a software expense.** This is where Rapid Application Development (RAD) pays off. In a world where the waterfall model of application development has been discredited in favor of Agile Methodologies RAD development is even more crucial to success.

As applications become easier to build they can be implemented quicker and more inexpensively. However, that is not to say building web applications is easy. As applications get more feature rich and replace desktop applications they get more complicated. This is not 1995; we know how to build web applications. Now it is a matter of time, money, and prioritizing the features.

Much is bantered about different IT stacks having better performance. I've read separately from different reputable sources that each IT stack offers the best performance. **The performance of a web application depends on the architects and engineers designing the system and configuring the platform; not the platform itself.**

There are only two caveats where platform decision alone can hinder performance. One is selecting a dynamic programming language to perform tasks requiring high performance. The second is trying to use MySQL in extremely high performance web applications. In some circumstances MySQL should be replaced by Oracle in the LAMP stack. SQL Server can handle 99% of the applications that Oracle can handle and MySQL can handle the loads of most applications. Otherwise **good application performance can be achieved using .Net, LAMP, and J2EE.**

## 10 Reasons why .Net is the Best Choice

1. The best RAD environment with Visual Studio
  - a. Best IDE debugger
  - b. Better object oriented programming environment than LAMP
  - c. The simplest development platform for Web applications and Mobile software applications
2. The lowest cost from a full Software Development Life Cycle (SDLC) perspective which includes maintenance costs
  - a. The productivity gains should surpasses additional software costs as software costs are relatively small compared to engineering costs
3. Most straight-forward, least fragmented roadmap
4. Tightest integration between stack components (simplifies configuration and administration)
5. Easiest programmer's to find
  - a. Java programmers are also plentiful, but they are also more expensive
6. The .Net framework is compatible with many languages including: PowerShell (a command line shell scripting language), Ruby, Python, Mono (open source), and C++ (through un-managed code)
7. Great documentation (a common complaint with LAMP)
8. The .Net framework libraries offer the simplest, most feature rich plug-in functionality
9. Up to date, web-centric architecture
10. The .NET executable format (Common Language Infrastructure, CLI), foundation classes (the Base Class Library), C#, and .Net runtime are ISO and ECMA standardized

## MSFT's Express versions

**Microsoft offers free versions of its Visual Studio and SQL Server software** called Visual Studio Express and SQL Server Express. SQL Server Express is a full RDMS, offering support for triggers, stored procedures, etc. SQL Server Express is comparable to MySQL. Although, as you would expect Microsoft puts some limitations on SQL Server Express, such as limiting the size of the database to 4GB.

**Microsoft also has a low cost Windows Server version** to support these Express editions, called **Windows Server 2003 Web Edition**. The only SQL Server version the Web Edition supports is SQL Server Express. Windows Server Web Edition doesn't support many of the more advanced server features such as Active Directory, clustering, or more than 2GB RAM. For a basic website these features are probably not a requirement. You probably won't need these features from day one, but if you expect to need them by day 60 or 90 it is sensible to initially deploy on Windows Server 2003 Standard Edition and SQL Server 2005 Standard Edition.

## Beware of Vendor Religion

**There are clear advantages of staying on only one platform** like: having to maintain only one environment, allowing your programmer team to focus on one platform, and having one vendor neck to squeeze when problems arise.

Unfortunately there is a tendency for IT people to exhibit near religious devotion to one vendor or all open source software. Most open source software is pretty disappointing, but many are very good and some are even better than the competing proprietary offerings. Whether the vendor is Microsoft, Oracle, Sun, IBM, Red Hat, or any other large software company capable of building out its own development platform, each company is going to offer a relatively decent percentage of software that is either pretty disappointing or entirely misses a large demographic.

For example, Microsoft's **Visual Studio is the clear leader** and Office 2007 is an impressive platform upgrade. However, you would expect more from Vista after five plus years in development. Microsoft's SharePoint is great for large enterprises, but **Wordpress and MediaWiki is a far better document management system for small enterprises**. For large enterprises with dynamic, technology savy employees MediaWiki may still be a better or complimentary solution.

So for specific applications each platform will have an advantage and being somewhat platform agnostic allows you to implement the best solution. **It is important to choose a primary platform and have a roadmap for that platform. However, religiously sticking to that platform for every application will, at times, cause you to implement far less than optimal solutions.**

I would guess that over 95% of large to mid-size enterprises, and even many smaller organizations, have applications running on Microsoft, LAMP, and J2EE; but sensibly many of these applications's hosting may be outsourced. Contrary to prevailing wisdom open source and .Net can play nice together, and momentum is growing. There is an open source .net programming language, started by Novell, called Mono.net. While I wouldn't suggest going this route it is a demonstration of **Microsoft's clear advantages (Visual Studio, the .Net framework and tight Windows server/client integration with things like Active Directory)** and **LAMP's clear advantages (low cost with solid performance)**. As eWeek's July 9 2006 cover article *eWEEK Labs Bakeoff: Open Source Versus .Net Stacks* determined "Open source and .Net better learn to play nice."

## Client-side Technologies

So far we have discussed the server-side components to a web application. However, the client-side is just as important. The web server sends HTML and code to the client and this code must adhere to web standards. Web standards are implemented at the client-side. The server code can be implemented in many different languages and follow many different architectural philosophies, but when server processing ends the output must be web (browser) compliant. This means HTML and ECMAScript (JavaScript).

### A Short List of Client-Side Technologies:

- Code
  - JavaScript / JScript
- Markup Language
  - HTML (HyperText Markup Language)
- Data Interchange Languages
  - XML (Extensible Markup Language)
  - JSON (JavaScript Object Notation)
- Data Definition (MetaData) Languages
  - XML Schema
  - RSS (Really Simple Syndication)
  - RDF (Resource Description Framework)
  - SOAP (Simple Object Access Protocol)
- Styles
  - CSS (Cascading Style Sheets): used to format HTML
  - XSLT (Extensible Stylesheet Language Transformations): used to format XML

There are many technologies beyond this short list and you will not use all of these technologies, but you will definitely use JavaScript and HTML. You should use CSS; XML and/or JSON; and XML Schema and/or RDF.

## AJAX Frameworks

There are many AJAX frameworks available. All of these frameworks can be used no matter what IT stack you use. If you need an extremely lightweight framework I would suggest using jQuery or Prototype. Both of these offer the core functionality you will need as well as some nice to have extras.

If you do not need an extremely lightweight framework (and most web applications will not) I suggest using ASP.Net AJAX, Script.aculo.us or the Dojo Toolkit. On the .Net platform I suggest using ASP.Net AJAX because of the server-side integration (this framework is OS and browser agnostic at the client-side). If using the LAMP platform I would suggest using Script.aculo.us or the Dojo Toolkit, although these libraries can just as easily be used with the .Net platform.

As someone who implemented my own AJAX JavaScript code from 2002 through 2006 I can safely say that using any of these five frameworks will save you considerable time while building a better, more browser agnostic website. There are other freely available AJAX frameworks such as the Yahoo! UI Library and Rico. These also are not bad choices, but I prefer ASP.Net AJAX, Script.aculo.us, and the Dojo Toolkit. Proprietary packages are also available from companies like JackBe and Tibco. However, with a great free (and somewhat open source) option from Microsoft plus other great open source options with strong support I would pass on the proprietary offerings.

### A Short List of AJAX Frameworks:

1. ASP.Net Ajax
  - Great integration between server and client (the server component can only be implemented using .Net)
  - Integrated into Visual Studio 2008 (now in Beta)
  - Best documentation and sample code
  - Offers a Control Toolkit add-on that offers dozens of out of the box controls
  - The client-side was released under a shared source license, the server-side piece source code is available but retains a Microsoft license, the control toolkit is open source
2. jQuery
  - Lightweight (only 1 JavaScript file)
  - support for CSS 1-3 and basic XPath syntax
3. Prototype.js
  - Lightweight (only 1 JavaScript file)
4. Dojo Toolkit
  - Run by the Dojo Foundation whose members include: IBM, Sun Microsystems, JotSpot, SitePen, Renkoo, AOL, TurboAjax, and OpenLaszlo
5. Script.aculo.us
  - This framework is implemented on top of Prototype
  - Ruby on Rails is implemented on top of Script.aculo.us
  - Possible community fragmentation due to Ruby on Rails inventor David Hansson from 37signals enforcing some trademarks