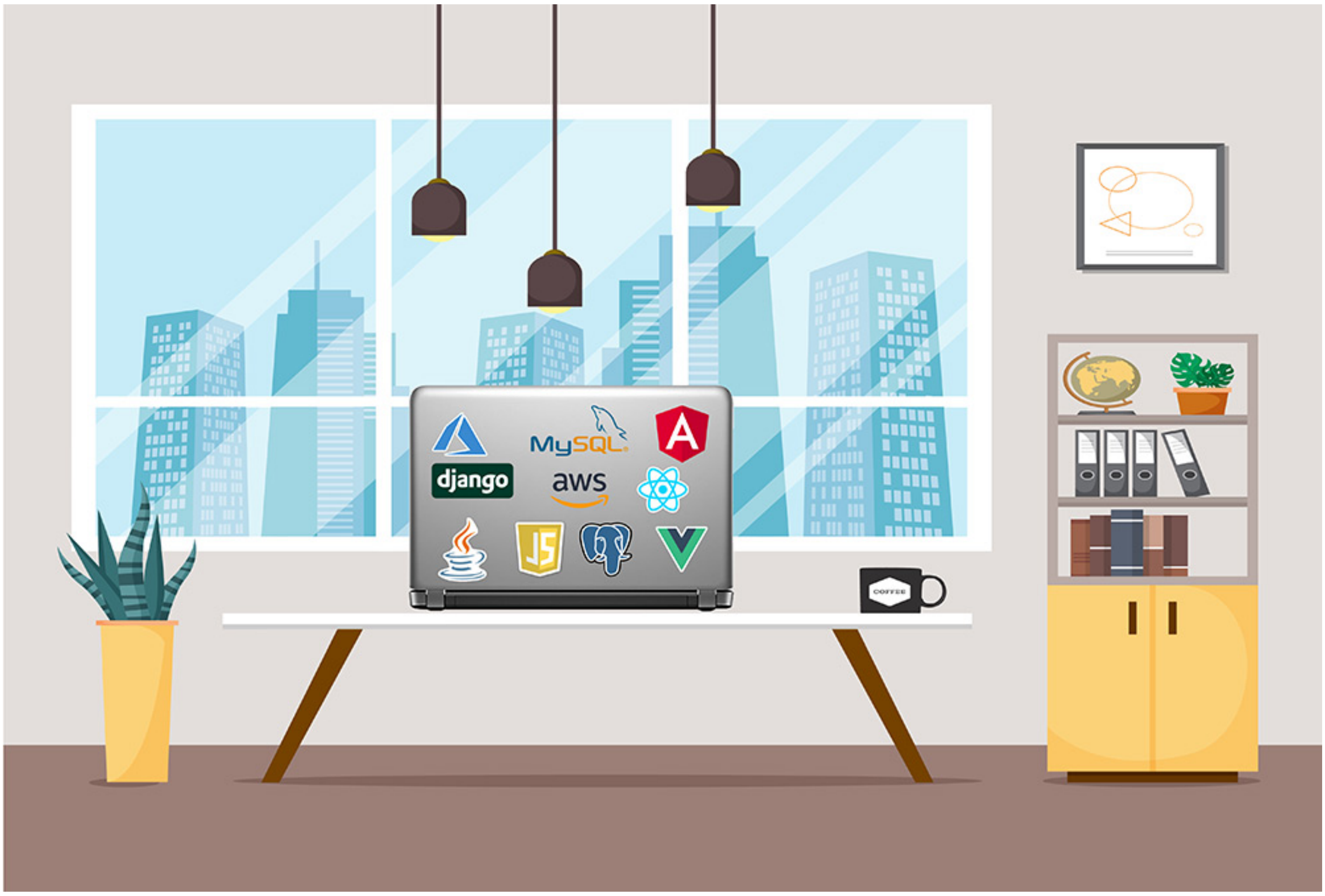
Michael's Coding Spot

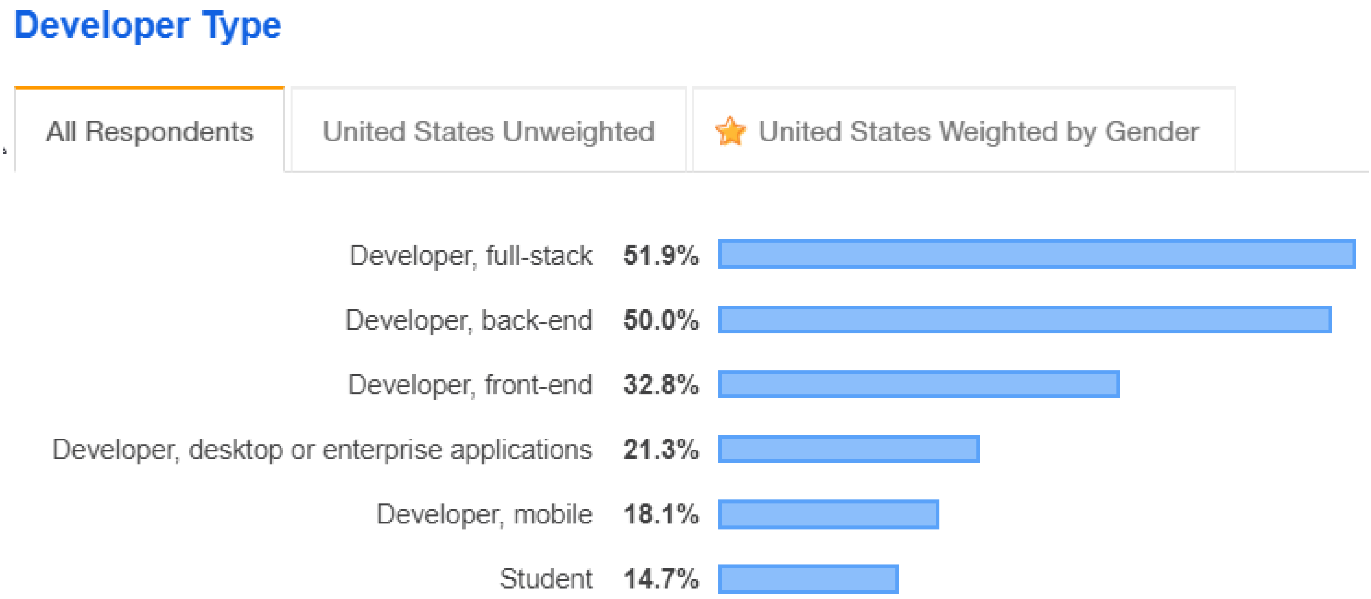
9 Must Decisions in Web Application Development

General / September 25, 2019



So you've decided to create a web application? Great, welcome to a world without any easy choices. There is a vast amount of different great technologies in every step you are going to make. And for every option, you will find a notable company that used it with great success.

The web development market is huge. It's the biggest programming space by far, with never-ending technology options. Looking at the StackOverflow survey of 2019, 52% of all developers are full-stack developers, 50% are back-end developers and 32.8% are front-end developers.



As you can probably guess, 134.8% of all developers is a pretty big market.

There are many popular technologies, databases, philosophies, design patterns and methodologies in the software world. They change drastically every few years, but somehow the old ones never seem to die. Cobol and ClearCase are great examples of something that shouldn't still exist but clearly does.

We will talk about 9 technological choices you have to make when developing a proper web application. Some will not be relevant to your product, but most will be. A lot of these choices give way to an entirely new group of choices, but that's just modern-day development.

I'm mostly a .NET developer, but I'll try to stay objective and not to express too many personal opinions.

> This article is talking about web applications, not websites. The distinction is a bit hard to put into words. Blogs and standard e-commerce sites are websites, whereas interactive websites like eBay and Facebook are web applications. Pretty much anything you can't build (easily) with WordPress is a web application.

Table of Contents

1. Client & Server Architectural Pattern

Since the internet came to life, we developed many different ways to build web applications. We had CGI, PHP, ASP, Silverlight, WebForms, and a bunch of others. In the last 10-15 years, we came to agree on 2 architectural patterns: model-view-controller(MVC) on the server-side or single-page-application (SPA) on the client side and Web API on the server side. Out of those two, the second approach (SPA + Web API) is getting the most traction in recent years.

The first decision in your web application is to choose an architectural approach.

Single Page Application (SPA) and Web API

Ever since AngularJS was released in 2010, SPA and Web API combination gradually became the most popular way to write modern web applications. And with some good reasons. With this pattern, the entire client side part of the application is loaded just once rather than loading each page from the server. The routing is done entirely on the client side. The server provides just the API for data.

There are many advantages to this approach:

- Sine the entire client is loaded once in the browser, the page navigation is immediate and the web application experience becomes more like a desktop application experience.
- Separation between client-side and server-side. There's no longer necessity for full-stack developers (that know both server-side and client-side).
- Testing both the client and server is easier because they are separate.
- Your Web API server is reusable for any type of application – web, desktop, and mobile.

And some disatvantages:

- Initial project setup is slower. Instead of creating one "new project" in your favorite MVC framework, you now have separate projects for the client-side and server-side. You'll have to deal with more technologies overall.
- Slower first-page load. In a SPA we are loading the entire client-side for the first page.
- You'll have to use bundlers like webpack for a decent develop experience. This adds some overhead like having to deal with bundler configuration. However, this is also an advantage because the bundler allows to easily add more tools to the build chain, like babel and a linter.

Model-View-Controller (MVC)

The server-side MVC pattern got popular in 2005 with the release of Ruby on Rails and Django frameworks.

In MVC, each route request goes to a **Controller** on the server. The Controller interacts with the **Model** (the data) and generates a **View** (the HTML/CSS/JavaScript client-side). This has several advantages. It creates a nice separation of concern between the client, server, and the different components. As a result, more developers can work on the same project without conflicts. It allows to reuse components. Since the Model is separate, you can replace it with a testable data set.

Some popular MVC frameworks are Ruby on Rails, ASP.NET MVC, Django, Laravel, and Spring MVC

> You can somewhat combine between MVC and SPAs. One View can be a single page application on its own. This is best done with thin SPA frameworks like Vue.js.

Others

There are a couple of other ways you can go, which aren't considered great options nowadays. They are old technologies, which were replaced for a reason. Some of those are:

- Classic ASP
- Classic PHP (without MVC)
- WebForms
- Static pages and Web API (this is still valid for static content, but not for a web application)

> You might have heard the terms LAMP, MEAN, and LYME stack somewhere. These are technology combinations. **LAMP** stands for Linux, Apache, MySQL, PHP, and **MEAN** is MongoDB, Express.js, Angular, and Node.js. These are absolutely not exclusive. For example, you can use MongoDB with any SPA framework and any programming language. And Linux is the OS for pretty much all modern servers, not just for PHP and MySQL.

2. Server-side Web API (when choosing SPA & Web API)

If you chose to go with a single page application (SPA) framework, then the next step is to choose a server-side technology. The standard means of communications are HTTP requests, so the server will provide an HTTP API. This means the server-side and client-side are decoupled. You can also consider using a RESTful API pattern.

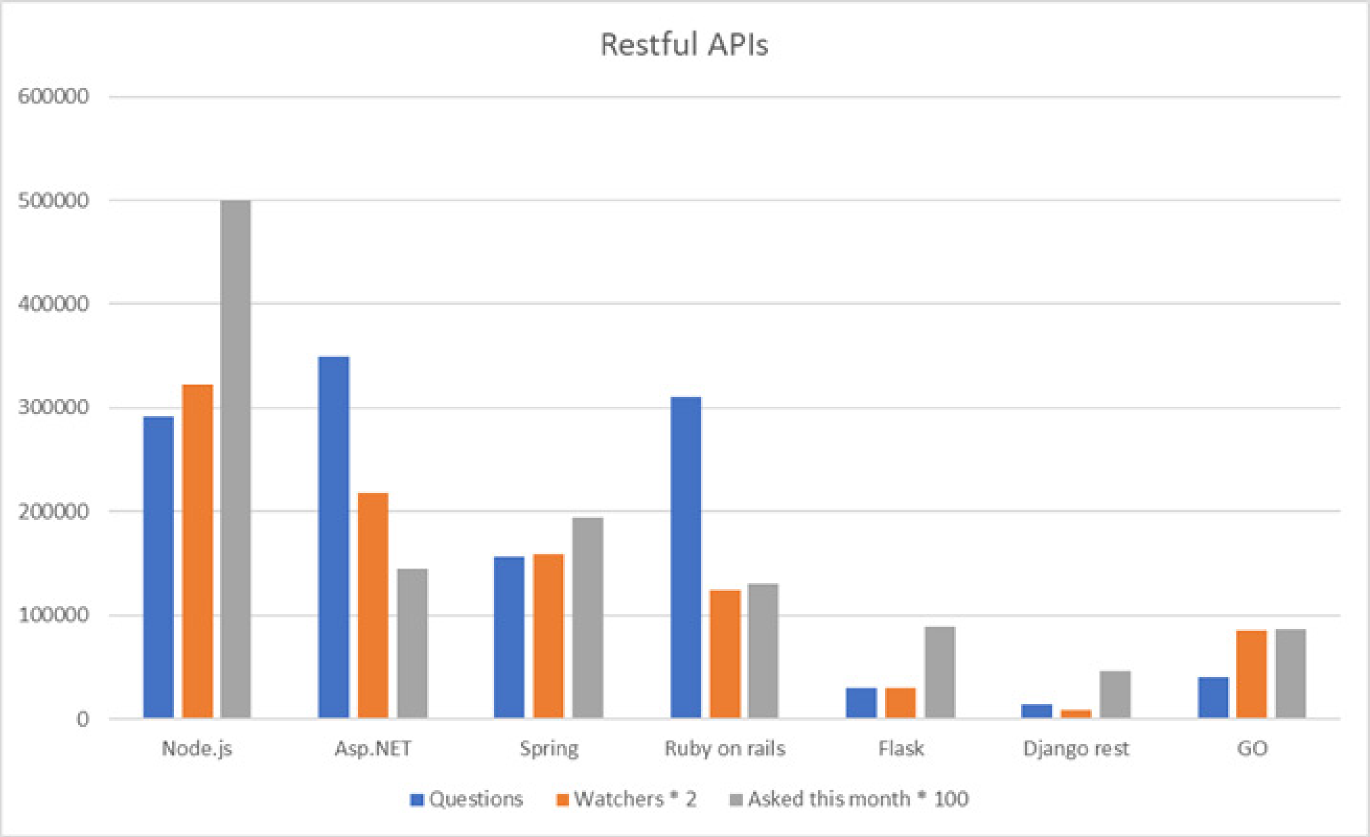
There's an abundance in great server side technologies. This is both a blessing and a curse. There are so many good choices that it becomes difficult to choose. Here are some of the more popular technologies:

- Node.js – JavaScript
- ASP.NET Web API – C#
- Java – Spring, Jersey, Apache CXF, Restlet
- Python – Flask, Django REST framework
- Go
- Ruby – Sinatra, Ruby on Rails – Rails is mostly MVC, but *Rails core* 5 supports API-only applications.

All of these frameworks are free and most are open source.

This is not an exhaustive list, but those are the most popular technologies. Choosing a popular framework is important. It probably got popular for a reason. A popular framework will have better support, better documentation, and more documented issues. Perhaps most importantly, you'll find more developers that are familiar with that framework.

Checking market popularity with Google Trends and surveys in this particular category was a bit difficult. Instead, we can see popularity by looking at Tags in StackOverflow. We can see overall usage according to the total number of questions asked. And we can see the trends according to the number of questions asked in the last month. This is not a perfect indicator of popularity, but I think it's pretty good. Here are the results:



> **Node.js**, for example, has 291K questions, 161K watchers and 5K questions asked this month

The big 4 winners are **Node.js, ASP.NET, Spring**, and **Ruby on Rails**. Flask, Django REST and GO are much less popular. However, this is not a fair comparison. **Spring, ASP.NET**, and **Ruby on Rails** are primarily MVC and not API-only, so they really have a much lower value. **Go** is a programming language, so it's overvalued as well. On the other hand, **Django-rest**, and **Flask** are server-side API technology only, so their value is "real". **Node.js** is also not an MVC framework, rather a technology to run JavaScript natively. But, it's *mostly* used for Web API with something like Express.js framework.

Keep in mind that even though Node.js is clearly the *most* popular, the other ones are still extremely popular technologies.

> **Java Jersey, Apache CXF** and **Ruby Sinatra** usage was so much lower in comparison that I didn't even include them in the chart. There are probably hundreds of other lesser known frameworks that don't appear as well.

Besides popularity, here are some more considerations when choosing:

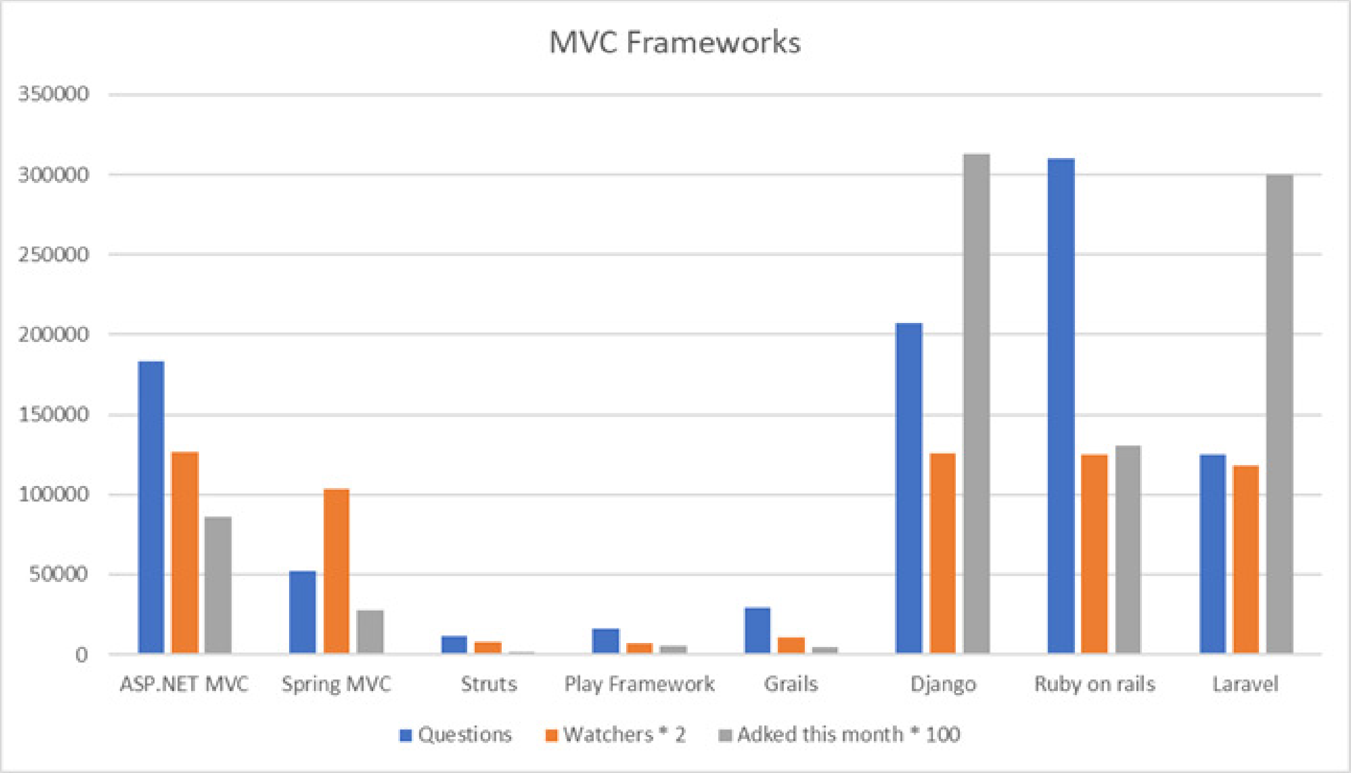
- For web applications that provide big-data analysis, consider going with a Python backend.
- Do you want to work with a strongly-typed programming language like C#, Java, and Go? Or weakly typed languages like JavaScript, Ruby, and Python? This is a big consideration. Make sure your development team is comfortable with the language.
- With Node.js, you work in the same language in client-side and server-side. I claim that's way too much JavaScript, but the world seems to think it's a good idea.
- Which development technologies are more popular in your area of the world? Prefer those.
- If you love C#, but afraid to be stuck with a Microsoft proprietary tech that's tightly coupled to Windows, then fear no more. The latest versions of ASP.NET (.NET Core) are open-source, work on Linux and have great performance on top.
- If you have a team that's already experienced with a framework or language, go with their known technology. This consideration trumps all others.

3. Server-side MVC (when choosing MVC)

Like with Web API, there's a big selection of server-side technologies that use the MVC pattern. Up to a few years ago, the MVC pattern was by far the most popular way to build web applications. The most notable frameworks are:

- C# – ASP.NET MVC
- Java – Spring MVC, Apache Struts, Play Framework
- Groovy – Grails Framework
- Python – Django
- Ruby – Ruby on Rails
- PHP – Laravel

And here's the popularity contest results according to Stack Overflow Tags:



> This is a normalized chart. **ASP.NET MVC**, for example, has 183K questions, 63K watchers and 856 questions asked this month.

These results were interesting and quite surprising for me. I expected **Ruby on Rails**, **Spring** and **ASP.NET MVC** to be on top. Instead, I found that **Django, Ruby on Rails** and **Laravel** were the most popular frameworks. **Ruby on Rails** has the most questions in all times. **Django** and **Laravel** seem to be rising in popularity with the most questions asked in the last 30 days.

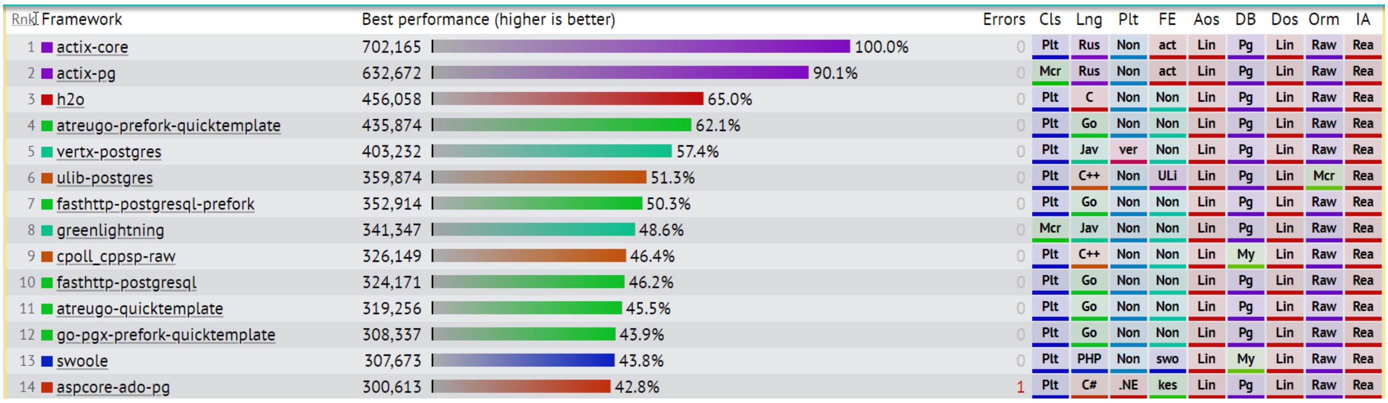
Besides popularity, the additional considerations when choosing a framework are similar to the ones for Web Api server side:

- For web applications that provide big-data and statistics, consider going with Python Django.
- The strongly typed vs weakly typed language is still a consideration.
- If you have a team that already knows and loves a framework or a language, go with the already-known known technology.

On a personal note, I'm dumbfounded that PHP is gaining popularity.

Performance Benchmarks – for Both Web API and MVC

If you're building a small business web application, performance might not matter as much. But for big applications that should serve many requests, response times are crucial. The most notable performance benchmarks comparison site is https://www.techempower.com/benchmarks/. Here, you can find a huge list of frameworks and various server configurations. Those are all benchmarked and compared into something like this:



Rnk	Framework	Best performance (higher is better)		Errors	Cls	Lng	Plt	FE	Aos	DB	Dos	Orm	IA
1	actix-core	702,165	100.0%	0	Plt	Rus	Non	act	Lin	Pg	Lin	Raw	Rea
2	actix-pg	632,672	90.1%	0	Mcr	Rus	Non	act	Lin	Pg	Lin	Raw	Rea
3	h2o	456,058	65.0%	0	Plt	C	Non	Non	Lin	Pg	Lin	Raw	Rea
4	atreugo-prefork-quicktemplate	435,874	62.1%	0	Plt	Go	Non	Non	Lin	Pg	Lin	Raw	Rea
5	vertx-postgres	403,232	57.4%	0	Plt	Jav	ver	Non	Lin	Pg	Lin	Raw	Rea
6	ulib-postgres	359,874	51.3%	0	Plt	C++	Non	ULi	Lin	Pg	Lin	Mcr	Rea
7	fasthttp-postgresql-prefork	352,914	50.3%	0	Plt	Go	Non	Non	Lin	Pg	Lin	Raw	Rea
8	greenlightning	341,347	48.6%	0	Mcr	Jav	Non	Non	Lin	Pg	Lin	Raw	Rea
9	cpoll_cppsp-raw	326,149	46.4%	0	Plt	C++	Non	Non	Lin	My	Lin	Raw	Rea
10	fasthttp-postgresql	324,171	46.2%	0	Plt	Go	Non	Non	Lin	Pg	Lin	Raw	Rea
11	atreugo-quicktemplate	319,256	45.5%	0	Plt	Go	Non	Non	Lin	Pg	Lin	Raw	Rea
12	go-pgx-prefork-quicktemplate	308,337	43.9%	0	Plt	Go	Non	Non	Lin	Pg	Lin	Raw	Rea
13	swoole	307,673	43.8%	0	Plt	PHP	Non	swo	Lin	My	Lin	Raw	Rea
14	aspcore-ado-pg	300,613	42.8%	1	Plt	C#	.NE	kes	Lin	Pg	Lin	Raw	Rea

The candidates in the above test include a server with a web framework, a database, and an ORM. In the benchmark, the framework's ORM is used to fetch all rows from a database table containing an unknown number of messages.

If we go by language, the fastest is Rust, followed by C, Go, Java, C++, PHP, C#, and Kotlin. If we go back to our "popular" frameworks and look for them, we'll find this:

1. ASP.NET Core Web API (42.8% of best result)
2. Node.js variation (17.9% of best result)
3. ASP.NET Core MVC (17.2%)
4. Spring (4.4%)
5. Laravel variation (2.9%)
6. Django (1.9%)
7. Ruby on Rails (1.3%)

By the way, the #1 performance winner Actix is a Rust language framework that I didn't include due to its very low popularity.

4. Choosing a Single Page Application (SPA) Framework

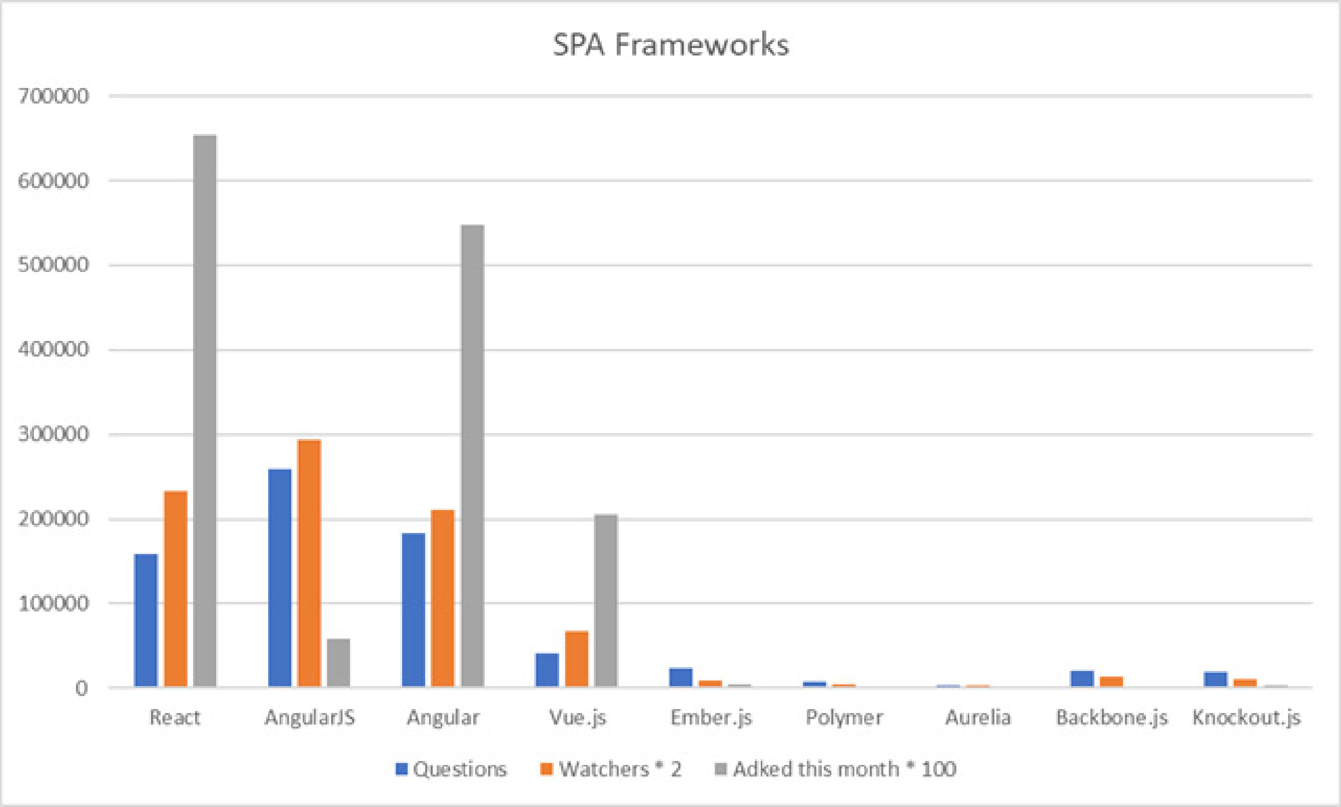
If you chose to use a Web API and SPA (not MVC), then the next decision is to choose a SPA Framework.

Just a few years ago, a new JavaScript framework sprouted about once a week. Luckily, those days are behind us and the field stabilized a bit.

The most notable SPA frameworks of all times (well, since 2010) are:

- React
- AngularJS
- Angular (2+)
- Vue.js
- Ember.js
- Polymer
- Aurelia
- Backbone.js

Let's do our usual trick with StackOverflow:



So this chart shows a few conclusions at a glance:

1. All past frameworks except for **React, AngularJS, Angular**, and **Vue.js** are dead. If you've been following web development news in the past few years, that should come as no surprise.
2. **AngularJS** has the most questions but the least new questions. Even though there's probably a huge amount of code written with AngularJS, it's a legacy framework. It's not recommended to choose it for new projects.
3. **React** and **Angular** dominate the market with **Vue.js** a distant 3rd. React in particular has the most interest.

This means your best choice in 2019 is between React, Angular, and Vue.js. These frameworks have the best community support, the most documented issues, and the best component/library support. With React and Angular having the most support.

Here are a few more points to consider:

- Angular was built as a sort of "enterprise" framework that considered everything and forces you into a particular mode of work. React and Vue.js are more separated into components and allow you to pick and choose development approaches.
- React won the "Most loved web framework" title in StackOverflow survey of 2019, with Vue.js a close second.

5. Database

Every modern web application has some sort of database. Or several databases even. In the old days, there were just relational databases and the differences were in performance and features. Today, we're in the age of "specialized databases". We need different databases for different purposes, which makes this decision pretty important. I'd say even more important than the previous decisions of server-side and client-side technology because those offer pretty much the same thing in different flavors.

It's important to understand your business needs when choosing a database. If your product needs high-performance Search capabilities, consider using Elastic Search. If you have a high-load of similar requests, whose response doesn't change very frequently, consider using Redis for its caching. If you just want to store a bunch of JSON documents without much fuss, then go with a document-store database like MongoDB.

Databases can be divided into several types:

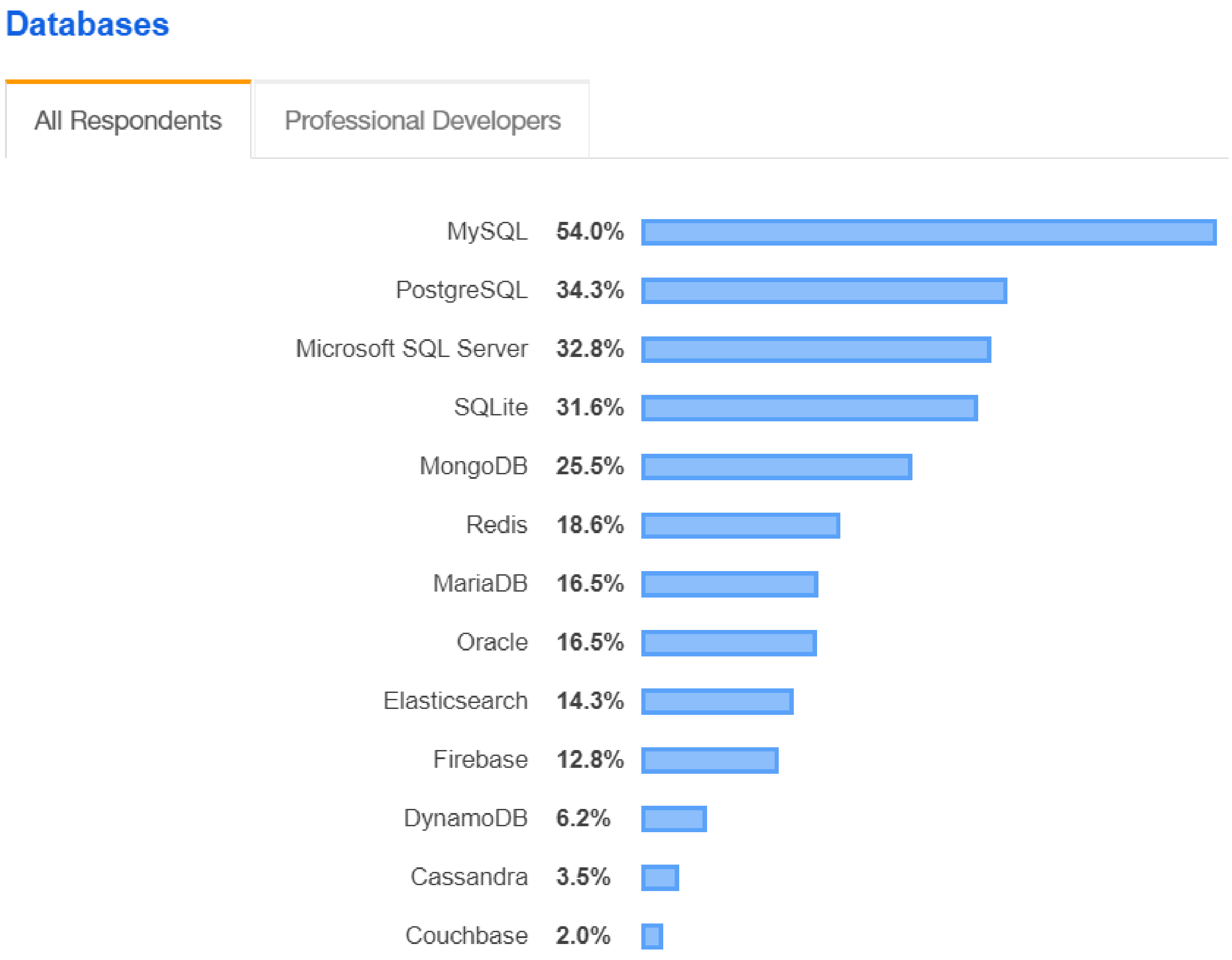
- Relational Databases – A classic table database that works with SQL queries like Microsoft SQL Server.

NoSql databases:

- Document-store databases like MongoDB
- Key-Value stores like DynamoDB and Redis
- Wide column store like Cassandra
- Graph-based database like Neo4j.

Before choosing a specific database, it's best to decide on the *type* of the database you need.

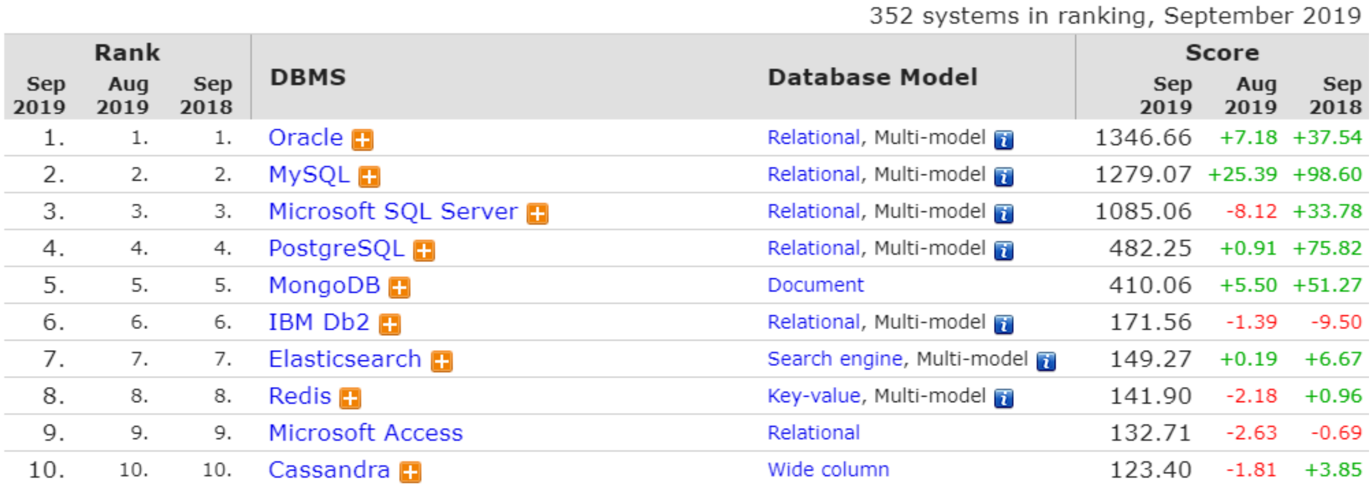
Like with other technologies, choosing a popular database is very important. You'll have more forums, bigger community and more developers familiar with the technology. Besides, they probably got popular for a reason. According to Stack Overflow 2019 survey, the most popular databases are:



MySQL is by far the most popular database. Note that the all 4 first spots are filled with relational databases. This might serve as some kind of indication that relational databases are the best choice in most applications.

Here are a few points to consider when choosing:

- Some of the commercial databases like Oracle and SQL Server can be quite pricey. Consider using one of the many open-source databases if the cost is an issue and you have a lot of data.
- Relational database stood the test of time. They are fast, reliable and have a ton of tools that work with them. You'll also find more developers familiar with the technology.
- If you're using a certain cloud provider, see which databases they support as **cloud-as-a-service**. This can reduce some initial development time when starting out. You might not actually need your cloud provider because there are services that provide independent database-as-a-service services like MongoDB Atlas.
- For small web applications, the considerations are different than for large enterprise applications. You'll need databases with the ability to scale, possibly to multiple machines (sharding).
- For **distributed databases**, consider the CAP theorem. It states that a database can provide only as much 2 out of 3 guarantees between **Consistency, Availability**, and **Partition tolerance**. Consider which two guarantees are most important to your needs and which database provides those two.
- For big applications or apps with high-frequency requests, you'll need to consider performance. One of the best places to find performance comparisons is DB-ENGINES. Here are the latest scores while writing this:



352 systems in ranking, September 2019

Rank			DBMS	Database Model	Score		
Sep 2019	Aug 2019	Sep 2018			Sep 2019	Aug 2019	Sep 2018
1.	1.	1.	Oracle	Relational, Multi-model	1346.66	+7.18	+37.54
2.	2.	2.	MySQL	Relational, Multi-model	1279.07	+25.39	+98.60
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	1085.06	-8.12	+33.78
4.	4.	4.	PostgreSQL	Relational, Multi-model	482.25	+0.91	+75.82
5.	5.	5.	MongoDB	Document	410.06	+5.50	+51.27
6.	6.	6.	IBM Db2	Relational, Multi-model	171.56	-1.39	-9.50
7.	7.	7.	Elasticsearch	Search engine, Multi-model	149.27	+0.19	+6.67
8.	8.	8.	Redis	Key-value, Multi-model	141.90	-2.18	+0.96
9.	9.	9.	Microsoft Access	Relational	132.71	-2.63	-0.69
10.	10.	10.	Cassandra	Wide column	123.40	-1.81	+3.85

Database choice is obviously a huge topic and an important one. Unlike other decisions, I suggest giving this one more weight and do additional research.

6. Deployment

In 2019, we have a big variety of **Cloud Offerings**. I believe that deployment in the cloud is the best fit in almost all cases. With a few exceptions. So before comparing cloud service providers, let's talk about why you would deploy to the cloud and what alternatives you have.

There are actually 3 options available:

1. **Cloud deployment** – Instead of bothering with setting up your servers in the basement, you can rent compute power and storage from a company like AWS or Azure. In fact, ever since AWS was launched in 2006, the software world is changing in this direction. There are many advantages to this:
   - With the Cloud, you can use dynamic scaling to cut costs. In low-pressure times, reduce the number of servers and in high-pressure times increase them. Since you pay-per-minute or per-hour, you can dramatically decrease costs.
   - The initial setup and deployment are much easier. In some cases, like with Azure App Service, deploying to a server for the first time is literally a few clicks.
   - You need to employ much fewer sys-admins (but a few DevOps engineers).
   - You no longer need to buy server machines, store them in the basement and upkeep them. It's all in the cloud man.
2. **On-premise deployment** – On-premise servers is the way organizations worked up to 2006. You had a server room and an army of sys-admins to keep them running. In 2019, we still have a lot of on-premise deployments. These companies either started on-premise and just didn't move to the cloud or they have some good reasons to stay on-premise. You should consider staying on-premise if:
   - You have some security or legal issues to place your server in the cloud. Maybe it's software in the military and government sectors.
   - You've invested so much effort in your on-premise server farm that it no longer makes sense moving to the cloud. It can happen if you're a big enough company and you've made your on-premise solution automated enough to justify the upkeep.

- You're big enough that it makes financial sense not to pay to the middle man. One example is Dropbox who aren't in the cloud. They did the math and it makes sense for them to be on-premise.

3. **Hybrid cloud solution** – The big cloud providers allow you to install a fully operational cloud server on-premise. That is, you'll have an AWS or Azure portal installed in your own data center. Pretty crazy concept and not for everyone, especially due to the cost. Both Azure Stack and AWS Outposts require you to buy new customized hardware. Google recently released Anthos, which doesn't require customized hardware.

Comparing Cloud Service Providers

Cloud services are divided into 3 categories: **Infrastructure as a Service** (IaaS), **Platform as a Service (PaaS)**, and **Software as a Service (SaaS)**. To deploy our web application, we're interested in **IaaS**. The three dominant IaaS cloud providers are: Amazon's AWS, Microsoft Azure, and Google Cloud Platform. Besides, it's worth mentioning IBM Cloud, DigitalOcean, Oracle Cloud Infrastructure Compute, Red Hat Cloud, and Alibaba Cloud.

> Besides AWS, Azure, and GCP, the only real competitor in terms capabilities is Alibaba. Unfortunately for them, major western enterprises are not so willing to work with a Chinese company.

The 3 big cloud providers have somewhat similar offerings. They all offer services like Scalable Virtual Machines, Load balancers, Kubernetes orchestrators, serverless offerings, storage as a service, database as a service, private networks & isolation, big data services, machine learning services, and many more.

Competition is fierce. Prices are somewhat similar and when one of the providers comes up with a new popular product, the other providers will adjust and offer similar products.

7. Authentication & Authorization

In practically all web applications we have a **Sign-in** mechanism. When signed in, you are identified in the web application and can: Leave comments, buy products, change personal settings and use whatever functionality the application offers. The ability of your application to verify that you are who you say you are is called **Authentication**. Whereas **Authorization** means permissions mechanism for different users.

There are a couple of ways you can go with authentication and authorization:

1. **The manual solution** – Most web frameworks have support for authentication and authorization (usually with JWT tokens). If not, there's always a free 3rd party library available. This means you can manually implement basic auth mechanisms.
2. **External Identity Server** – There are several open-source and commercial identity providers that implement the OpenID Connect standard. This means that client-server communication will involve a dedicated identity server. The advantages are that these servers have already implemented a bunch of auth features for you. These might include:
   - Single sign-on and sign-out with different application types

- Built-in support for external identity providers (Google, Facebook,…)
- Role-based permissions (authorization)
- Multi-factor authentication
- Compliant with standards like ISO, SOC2, HIPAA,…
- Built-in analytics and logs

Your cloud provider probably has an identity server, like AWS Cognito or Azure Active Directory B2C.

Notable open-source solutions are: IdentityServer, MITREid Connect, Ipsilon.

Notable commercial solutions: Auth0, Okta, OneLogin

Make sure to do some price calculations before committing to a commercial solution like Auth0. They tend to get kind of pricey and using an open-source implementation is also a good option.

8. Logging

Server-side logging is pretty important for any type of software, and web applications are not an exception. Whether to implement logging or not is not really a decision – implement logging. The decision is where to send these logs and hot to consume them.

Here are some logging targets to consider:

- A database. Logging to a database has many advantages

  - You can retrieve the logs from anywhere, without access to the production machine.
  - It's easy to aggregate logs when you have multiple servers.
  - There's no chance the logs will be deleted from a local machine.
  - You can easily search and extract statistics from the logs. This is especially useful if you're using Structured Logging.

  There's a myriad of choices for a database to store your logs. We can categorize them as follows:

  - **Relational Databases** are always an option. They're easy to set up, can be queried with SQL and most engineers are already familiar with them.
  - **NoSQL Databases** like CouchDB. These are perfect for structured logs that are stored in JSON format.
  - **Time-series Databases** like InfluxDB are optimized to store time-based events. This means your logging performance will be better and your logs will take less storage space. This is a good choice for intense high-load logging.
  - **Searchable Solutions** like Logstash + Elastic Search + Kibana (The "Elastic Stack") provide a full service for your logs. They will store, index, add search capabilities and even visualize your logs data. They work best with structured logging.
- **Error/Performance monitoring tools** can also act as logging targets. This is very beneficial since they can display errors alongside log messages that were in the same Http request context. A few choices are elmah.io, AWS Cloudwatch and Azure Application Insights.

- Logging to File is still a good logging target. It doesn't have to be exclusive, you can log both to file and a database for example.

Once you have logging in place, I suggest doing a test run on retrieving them or searching them. It's going to be a shame to wait until you have a bug in production only to find out that you have some kind of problem in your logging.

9. Payment Processor

In many web applications, you are going to be charging your customers. That is, receive credit card, PayPal or Bitcoin for services (well, not Bitcoin). Even though theoretically, you can implement payment processing yourself, it's not recommended. You'll have to deal with PCI compliance, security issues, different laws in different countries (like GDPR), frauds, refunds, invoices, exchange rates, and a million other things.

There are several big commercial players in this field like PayPal, Stripe, 2checkout, and BlueSnap. Those have a rich API and you can integrate the payment in your own site.

Here are some things to consider when choosing a payment processing company:

- **Security and PCI Compliance** – Make sure your chosen payment processor has PCI compliance. Any company that handles credit card has to uphold this standard. All the major companies will be PCI compliant.
- **Fees** – The standard fees in the industry are 2.9% of the transaction + 30 cents. Maybe some companies offer a cheaper rate.
- **Exchange Rates** – If you have international customers, they will pay in their own currency. Check which exchange rates the payment processor charges.
- **Ease of API** – One of the most important considerations is the ease of API. Be sure you will have a lot of interaction with the payment processor. You'll want hooks on transaction events, modification of invoices, adding discounts or more fees, refunds, and so on. Stripe, in particular, is known for it's excellent API.
- **Popularity** – Big companies will have more forums and internet discussions. With PayPal or Stripe, that's not going to be a problem. With big companies, you'll also find more developers familiar with the framework.
- **Data portability** – If you ever wanted to change payment processors, the current processor needs to allow that option. PayPal, for example, just won't give you your customer's credit card data. Whereas Stripe allows to export card data.

Summary

As you can see, web development in 2019 is not getting any closer to a consensus. If anything, we have more technologies to choose from and as much controversy as ever. I guess this is good news for us developers since we can afford to have specialties and get paid extra for our respective narrow fields.

In almost all decisions, all the choices are pretty good. That means you won't make a terrible mistake by choosing the one over the other. On the other hand, by choosing one you're usually stuck with them. Suppose you chose to go with AWS and then decided to move to Azure. That's not going to be that easy.

There were a lot more "Must" decisions I wanted to include like **Error Monitoring, Application Performance Management tools, ORMs, Mobile support + PWAs**, and **Localization**. Then, there are a bunch of client-side decisions like free or paid UI Controls, bundlers, linters, and so on. This article got a little bigger than I planned, so I'll leave those to another post.

Thanks for reading, and subscribe to more articles if you enjoyed this one. Cheers.

← Previous Post Next Post →

27 thoughts on "9 Must Decisions in Web Application Development"

MICHAEL.MICHAEL AIRD
SEPTEMBER 25, 2019 AT 4:51 PM

One comment: Knockout shouldn't be included in the list of SPA frameworks. I love Knockout and we use it in our application but it's not a SPA framework, it's a 2-way data-binding framework. It does that really well but it doesn't include any of the other things you'd expect in a SPA framework (routing, state management, etc)

MICHAEL SHPILT
SEPTEMBER 25, 2019 AT 6:31 PM

Thanks for the tip. I'll update the post

KALMEN
SEPTEMBER 25, 2019 AT 11:53 PM

I love to know if Embarcardero Delphi is part of the test in your performance benchmarking for the programming langauges?

MICHAEL SHPILT
SEPTEMBER 26, 2019 AT 3:58 PM

No, I didn't include it because I thought Embarcardero Delphi was primarily for desktop & mobile apps

SONNY TORRES
SEPTEMBER 26, 2019 AT 9:27 AM

Wow man, this seems very comprehensive from a high level and very succinct. So much knowledge packed into this. Ive been writing python scripts for the past 2 years and want to get into Software Engineering. Thank you for this great post.

MICHAEL SHPILT
SEPTEMBER 26, 2019 AT 3:55 PM

Thanks. Python is one of the hottest techs right now, so you're in a good spot, good luck!

MAURICIO F MARTINEZ
SEPTEMBER 26, 2019 AT 11:17 PM

Thank you for sharing your knowledge Michael. This is a great post.

MICHAEL SHPILT
SEPTEMBER 27, 2019 AT 10:29 AM

Thanks, Mauricio

群众演员甲
SEPTEMBER 27, 2019 AT 5:27 AM

Well-structured post, all of these 9 points are what I concerned.
Really hopeful, thank you!
Just to mentioned for the deployment, I experienced by myself that use Docker is extremely easy, simple and convenient.

MICHAEL SHPILT
SEPTEMBER 27, 2019 AT 10:29 AM

Thanks for the tip!

ULA
SEPTEMBER 27, 2019 AT 6:01 PM

Great article! Web development is truly a challenge both for clients and developers. Here we share some thoughts on Angular and React frameworks. A useful guide which one is better:
https://www.polidea.com/blog/angular-vs-react-how-to-choose-the-best-framework-guide-for-beginners/

IVAN
SEPTEMBER 28, 2019 AT 11:28 PM

In part "2. Server-side Web API (when choosing SPA & Web API) " you say "On the other hand, Node.js, Django-rest, and Flask are server-side API technology only.", this is not true in case of Node. Node is JavaScript runtime, but more broadly it is a platform to run JS outside browser and its use is MUCH broader than just server-side API (it is more appropriate to compare it to JVM or CLR). The broadest use-case is probably build tools for anything Web related with modules like Webpack. If you want to look at API frameworks specifically – that would be Express (which you have mentioned in MEAN stack).

MICHAEL SHPILT
SEPTEMBER 29, 2019 AT 10:04 AM

Thanks for the correction, Ivan. I'll update the article when I get the chance

GULSAR
SEPTEMBER 30, 2019 AT 10:13 AM

Excellent article. Lot of information is packed in this article. It is treasure trove of information for anyone who is looking to get an understanding of modern web-application.Thanks a lot for writing such informative article.

JESSICA WATSON
SEPTEMBER 30, 2019 AT 3:36 PM

Great write-up Michael.
Really appreciate the way you have written and explained. Worth reading it.
Thanks for sharing it with us.
Good work..!!

MICHAEL SHPILT
OCTOBER 7, 2019 AT 9:33 PM

Thanks Jessica

TMPRESTON
OCTOBER 2, 2019 AT 11:16 PM

Great write up. I'd also add Seq to the searchable log store. Accepts structured logs from variety of logging frameworks and simpler to install/maintain.

MICHAEL SHPILT
OCTOBER 7, 2019 AT 9:34 PM

Thanks. Didn’t know about Seq, I’ll check it out

LADEBUG
OCTOBER 8, 2019 AT 6:26 PM

Wow, Michael, what a detailed and sufficient guide for the professionals, thank you for your work!!

As for wet-behind-the-ears programmers and also for product owners who consider to develop a web app for their idea/business, it’d be nice to start with the basics. Here https://litslink.com/blog/web-application-architecture I found the essencials of web app development such as components, types, the diagram of the architecture, best practices and more.

MICHAEL SHPILT
OCTOBER 8, 2019 AT 7:11 PM

Thanks for the feedback

RUKKY KOFI
OCTOBER 26, 2019 AT 6:28 AM

Hey Michael, I really enjoyed this article and I’m looking forward to a second one. It’s hard to get to know some of these things if you’re not working as part of a team where these practices are standardized. I do hope you are going to work on the second one. That last paragraph at the end mentioned some things I’ve never even heard of before and I’ve been doing the web development thing for little over a year now. I’m particularly interested in learning about Application Performance Management tools and Localization.

MICHAEL SHPILT
OCTOBER 26, 2019 AT 4:49 PM

Thanks for the feedback Rukky. I’ll release an article on APMs (though specifically for .NET) in the near future (in February 2020, got long term plans 🙂

STURLA
OCTOBER 27, 2019 AT 9:45 AM

No mention of WASM and in particular Blazor since you are in .net land mostly? Thats totally what Im going for in my next “file new project” for a SPA without any javascript 😉

MICHAEL SHPILT
OCTOBER 28, 2019 AT 3:27 PM

Good luck! Yeah, I kind of ignored Blazor since it’s not even GA yet.

MARKETING KBK
FEBRUARY 10, 2020 AT 2:48 PM

Great job! Hats off to your clarity, your article was very informative and it was very usefull for developers. Are you looking for any web application development services kindly visit KBKBUSINESSSOLUTIONS.com to build successfull web projects.

AVANTIKA
MARCH 17, 2020 AT 9:09 AM

It was a well structured and well-crafted post. As a fresher, i can tell that all your suggestions will be useful. Thanks for sharing.

BOTREETECHNOLOGIES
MAY 6, 2020 AT 3:10 PM

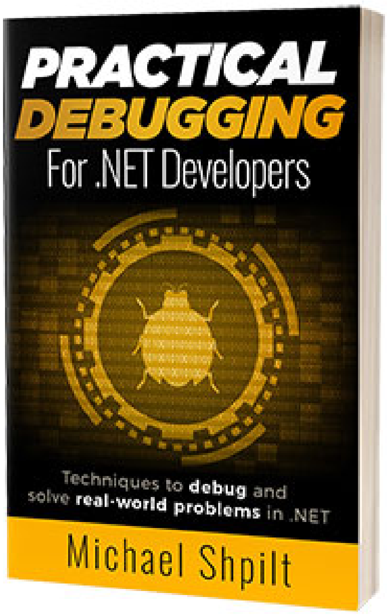
One of the very best articles on web application development. I have read different content pieces on the similar topic but matches the quality of this one. Thank you for sharing your valuable insights. It really helped me out.

Comments are closed.

Welcome to my blog! I'm a software developer, C# enthusiast, author, and a blogger. I write about C#, .NET, memory management, performance, and solving difficult problems in .NET.

More about me →

I just released my new book **Practical Debugging for .NET Developers**



Find out more here

Recent Posts

- The Best C# .NET Web Application Tech Stack: Deploying to Azure
- The Best C# .NET Web Application Tech Stack: Choosing The Back End
- The Best C# .NET Web Application Tech Stack: Choosing The Front End
- Creating a Database in C#: Interviewing the CEO of RavenDB Oren Eini
- Optimizing CPU-Bound and Memory-Bound .NET Applications: 11 Best Practices
- 6 Essential Tools to Detect and Fix Performance Issues in .NET
- Best Practices to Measure Execution Time in JavaScript
- 6 Hidden Productivity Gems in Resharper and Rider

Debugging Tutorial

Privacy Policy