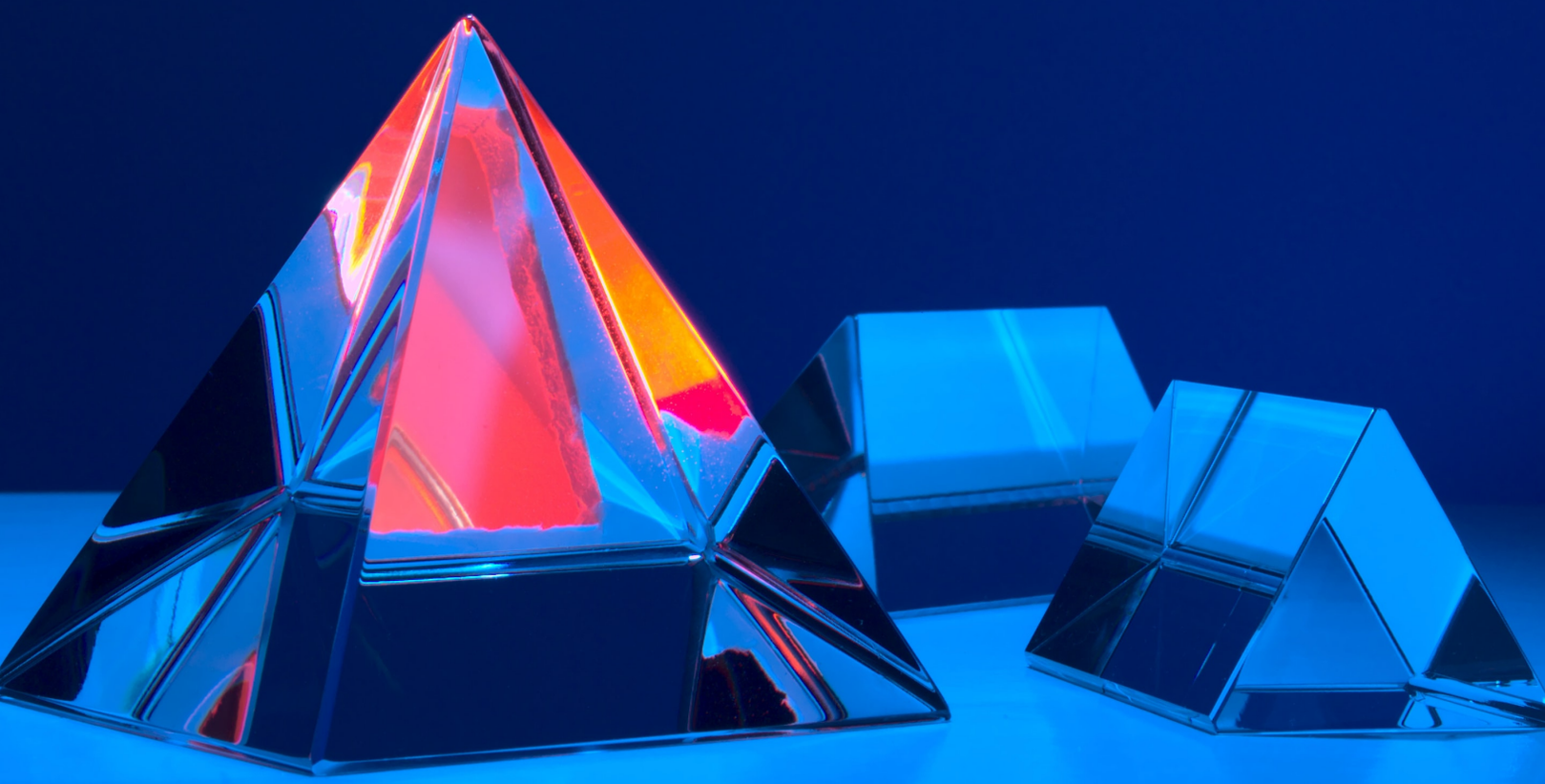
ITMAGINATION
Imagine it. Done.

THE COMPLETE GUIDE TO

JS TS node JS

IN ENTERPRISES & STARTUPS



Executive Summary

Despite its reputation, JavaScript is suitable for writing large scale applications in a variety of business environments – from startups to large corporations. To be able to do that, software engineers often use the language's superset – TypeScript.

To be able to run programs on the backend, though, Node.js is needed, and it’s exceptional at that as well. It is able to handle a large amount of simultaneous connections, even despite seemingly not being able to.

There are few benefits that all business environments appreciate, and few of them that will be appreciated in particular only by a specific one.

As such, the universal pros are:

1. (Design-time) Type Safety
2. Talent Availability
3. Faster apps written in fewer lines of code
4. One language across the stack
5. Smaller teams doing the same amount of work
6. Performance
7. Scalability

As for enterprises, they will surely fall in love with microservices, and how easy everything is to implement with JavaScript; and startup engineers will enjoy the increased iteration speeds, and the big number of packages offered by the npm (Node Package Manager).

Table Of Contents

A Short Introduction

JavaScript can be regarded as the most universal general-purpose programming language available on the market today. Of course, an obvious area where it shines the most is web front-ends. React.js, Angular, Vue, or Svelte are usually the default choices for creating unforgettable experiences. That's the most traditional use case, though far from the only one. The language has come an extremely long way from being used as only a facilitator of interactions on websites to a fully-fledged, universally useful programming language.

Now, it's really hard to use the services of a company that does not use JavaScript or TypeScript in its technology stack. It is indeed a challenge. Web backends and mobile apps are obvious use-cases. It gets even more surprising when you find out that there are libraries allowing you to do some machine learning, or work with embedded systems [sic!].

The purpose of this e-book is to introduce you to the world of JavaScript in both enterprise and startup settings.

We will be addressing the following questions, and many more, in the following pages.

Why did everybody start looking towards JavaScript more favorably?

1. What needed to change?
2. What were the key moments in its history?
3. What are some of the most popular frameworks?
4. What is the future of the ecosystem?

Let’s Start With JavaScript’s History

Over the course of the language’s lifetime there were a number of events that were revolutionary. There were moments of gradual evolution as well. Nearly all of the efforts were fueled by the army of open source contributors that worked on a solid foundation given to them by ECMA.

JavaScript's Infancy

In the Ice Age of the Internet, every website was static. Until 1995, there was no way of making websites dynamic; meaning that it was impossible to change the content of a webpage in response to different conditions.

That's why in 1995, JavaScript was created. Brendan Eich, the language's creator, was given a simple task – to make interactivity possible. Eich's creation had a clear goal, and as such fulfilled it. For a long time, that was it.

As the father of JavaScript, Brendan Eich, said:

> I was under marketing orders to make it [JavaScript] look like Java but not make it too big for its britches. It's just this sort of silly little brother language, right? The sidekick to Java.

Asynchronous actions in the browser

To put it mildly, however, reacting to the simplest of events was not at the top of people's bucket lists. At one point, people woke up, realizing that more can be done. Instead of reloading pages to show new information, they thought: "maybe

we can make the process more efficient and faster?" There had to be a way to prevent websites from reloading to get and process new data, right?

The first time when such an innovation was implemented to a wider audience was when Microsoft allowed Internet Explorer 5 to make HTTP requests. In 1999, they implemented an "object" called "XMLHTTP" which allowed for getting data from an external source **without** reloading the page.

What was the reason for its implementation? Long story short, MSFT did that just so Outlook's website fetched emails in the background. The long-term effects were much more than making one website run better. Other companies followed in Microsoft's footsteps. In 2002, Mozilla implemented the "XMLHttpRequest" to Mozilla Application Suite 1.0. The latter implementation was the one that was embraced wider, and later even used by Microsoft itself.

According to Monika Mazurczak, Front-end Developer at ITMAGINATION:

> [Microsoft's XMLHttpRequest object] was one of the two most important developments in the world of JavaScript.
>
> Asynchronous actions allowed users to do more without having to reload the page which was a huge win in-terms of UX.

There was still an additional big problem to address. There were large differences in JavaScript engine implementations, and there were even different language flavors for each browser!

Netscape had JavaScript, while IE had Jscript, for example. Even though MSFT's implementation was to some degree compatible with JavaScript (as both were based on the ECMAScript specification), the implementation was fundamentally different.

jQuery – building for different browsers without a headache

In 2006, the world witnessed the birth of a solution that aimed to address the problem of JavaScript's interoperability. Fourteen years ago, jQuery saw the light of day, abstracting the differences in implementations.

```
// If Mozilla is used
if (jQuery.browser == 'mozilla' || jQuery.browser == 'opera') {
  // Use the handy event callback
  jQuery.event.add(document, 'DOMContentLoaded', jQuery.ready)

  // If IE is used, use the excellent hack by Matthias Miller
  // http://www.outofhanwell.com/blog/index.php?title=the_window_onload_problem_revisited
} else if (jQuery.browser == 'msie') {
  // Only works if you document.write() it
  document.write(
    '<scr' + 'ipt id=__ie_init defer=true ' + 'src=javascript:void(0)></script>'
  )

  // Use the defer script hack
  var script = document.getElementById('__ie_init')
  script.onreadystatechange = function () {
    if (this.readyState == 'complete') jQuery.ready()
  }

  // Clear from memory
  script = null

  // If Safari  is used
} else if (jQuery.browser == 'safari') {
  // Continually check to see if the document.readyState is valid
  jQuery.safariTimer = setInterval(function () {
    // loaded and complete are both valid states
    if (document.readyState == 'loaded' || document.readyState == 'complete') {
      // If either one are found, remove the timer
      clearInterval(jQuery.safariTimer)
      jQuery.safariTimer = null

      // and execute any waiting functions
      jQuery.ready()
    }
  }, 10)
}

```

The image you see above was how jQuery handled the differences in implementations between browsers. What you see below is just calling the jQuery.ready() function after a page finishes loading.

Do you know what it looks like now? Thanks to the standardization of implementations, it is pretty straightforward. There are a few ways to go about this, and this is one of them:

```
1 document.addEventListener("DOMContentLoaded", function () {
2     jQuery.ready();
3 })
```

jQuery would become the most popular JavaScript library, and by a large margin. It still is the most popular library by market share to this day, where it is present on approximately 79.8% of the top 10k popular websites.

In comparison, React.js, which is widely regarded as a modern standard for front-end development, is used on 43.7% of the top 10,000 most popular pages. When it comes to the front-end side, jQuery was the final important step to get the ball rolling. Consistency across browsers allowed developers to create the apps they wanted to, but previously couldn't.

2009 – arguably the most important year for the ecosystem since the release

Three years later, in the arguably most important year for the whole JavaScript world, there were two events that changed the ecosystem.

One of these events was the initial release of Node.js, the other one being the first major revision of the language – ECMAScript 2009 (better known as ES5) saw the light of day.

The First Event – ECMAScript 2009

ES5 brought many features that were absolutely necessary. Strict mode, essential operations on arrays such as filter, map, and reduce ("The Holy Trinity of Functional Programming"), the JSON object, property getters & setters, and much more that is outside of the scope of this eBook. It was a revolution, and a welcome one.

The Second Event – The Birth Of Node.js

The creation of Node.js made JavaScript a language that is relevant because the runtime enabled us to execute scripts outside of browsers in addition to in-browser execution. Interestingly, it was not the first attempt at allowing execution of JavaScript on the server side.

Netscape's Live Wire was the first one, but since it might be the first time you have heard about it; that says something about the solution's popularity. While it does warrant an honorary mention, we won't dedicate much more space to it.

Now that JavaScript was ready for serious use, there was one more aspect that needed to be fixed before the language could see some serious enterprise adoption. The language is dynamic, without a type system.

You see, when you are thinking about writing an app that will get bigger than your standard starter projects, you will run into problems sooner or later. Strongly typed languages, such as Java or C# give you a layer of safety – they will catch some bugs themselves, without having to involve you, the developer, in any way. That's because they will ensure the correctness of the code themselves during design, and compilation.

TypeScript Comes To The Rescue

There were some attempts at making JS more predictable, however, Microsoft might have proposed the best solution to date. Typescript, JavaScript's superset, saw the light of day for the first time in 2012. Its purpose was to facilitate the development of large applications.

It has some features that made it extremely popular among the pros.

First of all, since C# and TypeScript were created by the same person, they share some similarities. Of course, don't expect the two to be twins. Second of all, migration from JavaScript to TypeScript is easy – you can migrate your codebase step by step.

Third of all, TypeScript integrates with e.g. React and Vue on the front-end seamlessly. On the backend, all major libraries work with TypeScript out of the box. Nest.js uses TS by default; while Express.js and Fastify have their additional type definitions for you to install and forget.

The World of Business and JavaScript

The world of business consists of a seemingly infinite number of companies of varying sizes and revenue. Different needs, and wants. There are beyond numerous ways of categorizing firms, however here we will use the distinction on the basis of size.

There are many definitions for a startup. Some emphasize its size, some innovation, yet the universal aspect of all startups is their rapid growth. Companies like these have one common goal in mind – rapid growth. Everything else seems to be irrelevant, as these firms can even generate losses for years (a ride hailing giant comes to mind) – it's not their goal to be profitable (yet).

Often portrayed on the other end of the business spectrum, there are big corporations. It is often thought that their goals are much different from the goals of startups. We will argue, however, that the goal is the same – creating value. It's only the path that changes. As such, there are plenty of pros that come with the use of JavaScript that companies of all sizes enjoy.

Why Use JavaScript Across Your Stack?

Talent Pool? Talent Ocean.

As it's the case with startups, large companies can also have a problem with staffing. Even though they have more resources than the small players, they often need a lot more talent, too. Luckily, our language is convenient in terms of recruitment as well.

Apart from being loved by large organizations' software development teams, JavaScript is also the leading language in terms of developer affinity. Its simplicity, low barriers of entry, general availability of learning resources, and its universality make it a popular choice among programmers, both experienced and fresh.

Furthermore, JavaScript is the leader in GitHub's "State of the Octoverse" study, being the most popular language since 2014 (the latest release was in 2020). TypeScript placed 4th, ahead of of C#, PHP or Ruby, and behind Java, Python, and of course, JavaScript.

In StackOverflow's Summary from the same year, JavaScript remained the most popular language for the eighth year in a row [sic!]. When it comes to affinity, TypeScript places second, behind Rust, and ahead of Python.

Those are just some indicators of how deep the talent pool can be. One "catch" is that the bulk of JavaScript's popularity on GitHub could be attributed to frontend engineers, ITMAGINATION's Chief Innovation Officer, Marcin Dąbrowski adds.

A Deep Candidate Pool

When recruiting JavaScript developers, you can expect a wide pool of them. There are approximately 13.8 million of them, which is the highest number in the world. The number is increasing, as well. Last October (October 2020), the number was at 12.4 million.

Because of the language's gentle learning curve, the tempo is unlikely to slow down. Unless a major world catastrophe occurs, things won't change – which is why planning for the future with JavaScript at the center is neither short-sighted nor irresponsible. It's a safe investment.

Apps That Run Faster, Built Quicker With Fewer Lines Of Code

You might think that this header is a lie. I would not blame you. It is completely true, however – and there are documented cases that prove this statement is true.

In 2013, PayPal wrote an article about their journey with Node.js. They managed to rebuild their app in JavaScript twice as fast, with fewer people. On top of that, there were 33% fewer lines of code, and it consisted of 40% fewer files (!).

Not only that – the app could serve twice the amount of requests versus the Java based application. Interestingly, the Node.js app utilized only one of the cores,

while Java used... 5 of them [sic!]. If you are not impressed by now, pages were served 200ms faster.

Node.js in general does not require many resources to run and is quite fast.

Artur Łabudziński, Senior Node.js & JavaScript Software Developer at ITMAGINATION, shared how he is impressed by the language's performance and Google's great work on their V8 engine – the foundation of Node.js.

"[Talking about the reasons for JavaScript's popularity in cloud & backend use cases] To me, because of its speed. Because of its optimization, because Google works really hard on V8, which is the engine that runs JavaScript.

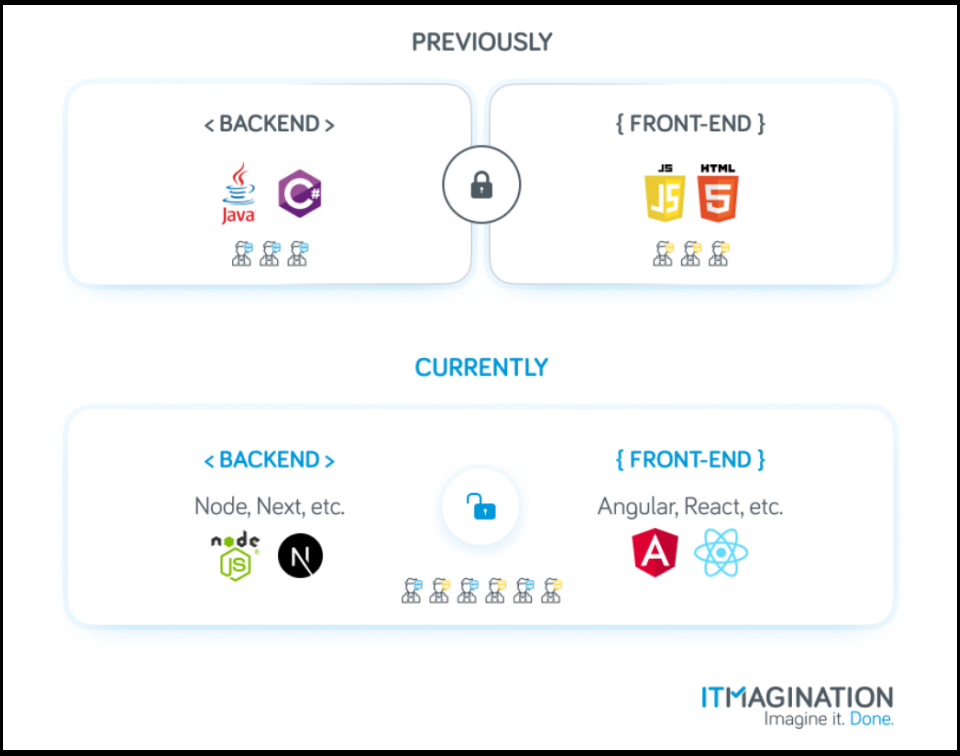
They optimized it, wrote a few compilers, which is why it starts fast and also can simultaneously optimize itself because of JIT, which stands for "Just In Time compilation".

Thanks to this, we now have a decent programming language that runs everywhere. And Node.js, an environment for JavaScript that runs on V8 with a connection to the Operating System that allows us to write both backends and apps that are more serious, and do more than just showing a website.]

Since 2013, there were a lot of improvements made to the language, and runtime, which most likely further improved performance.

The Same Language On The Backend And The Front-end

Many companies use frameworks such as Express.js, Fastify or Nest.js on the server-side, and React, Angular or Vue on the client-side. They don't stop there either. React Native/NativeScript often serve them when creating mobile apps. Electron allows them to create desktop apps. Even though they have the resources to have dedicated teams working in native languages for all platforms, they choose not to. Why?



To begin with, if all software engineers specialize in the same language, then everybody understands each other. Traditionally this wouldn't be the case with one team developing in C#/Java and the other in JavaScript. This created artificial divides that are difficult to overcome. At PayPal, back in 2013, front-end and backend teams were unified "into one team which allow[ed] [them] to understand and react to [their] users' needs at any level in the technology stack." Teams can share their experiences, often 1:1, which leads to faster bug fixes, for one.

Additionally, there is the economic benefit of smaller spending since the teams are smaller.

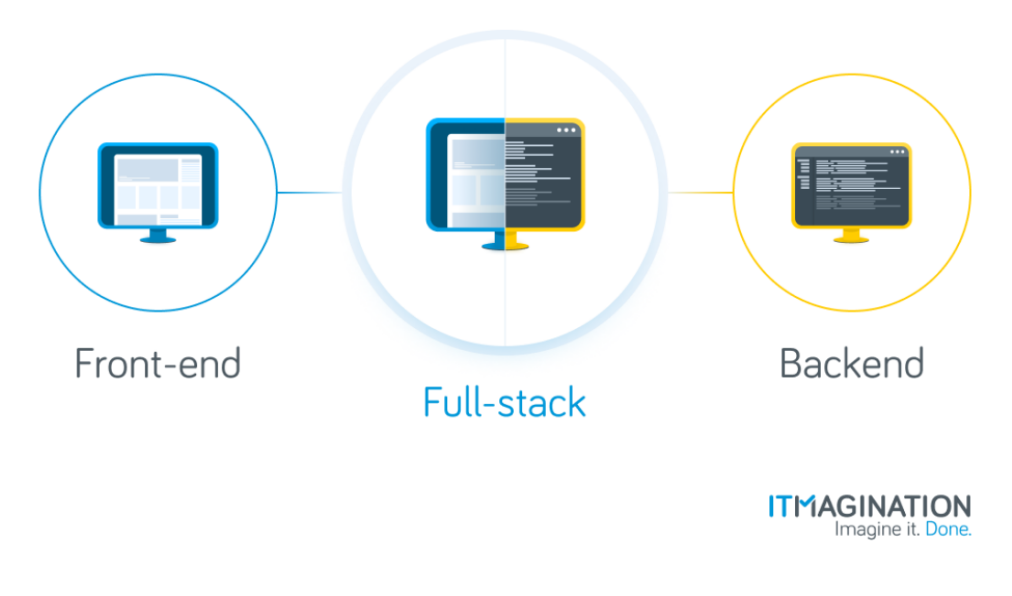
This is a crucial aspect for smaller businesses. Maintaining two separate teams is costly. Startups have many assets – though an excess of money is usually not one of them. As such, many new, rapidly growing companies opt-in to use Node.js on the back-end and on the front-end. The choice of Node.js is not subpar by any means at all.

It also is the most natural choice browsers can execute JavaScript only – technically you can use other languages to add interactivity to web pages, though it's a bit too early for that yet. As such, you have to use JavaScript on the back-end as well as on the front-end if you want to use the same language everywhere.

This way, teams are smaller, more agile, and communicate better.

Fewer Developers Required - PayPal's Success In Reducing The Number Of Developers To Ship

With their transition, they made sure that there is only one team writing code in one language only. Back then, full-stack JavaScript developers were truly a rare find. There were more popular stacks, such as Java & Java Server Faces (currently Jakarta Server Faces) or .NET + Angular.js.



Nowadays, developers working on both sides of an application are more common, and they enjoy certain popularity. They can save big companies a lot of money while the app maintains the same level of quality.

As a result, there are fewer team members and overall a smaller number of teams required.

Better Performance

Node.js is fast. How fast? In the TechEmpower benchmark, JavaScript took second place with "just-js." Of course, the project itself is a bit of a curiosity, though of course the app will work. It's faster than drogon written in C++, and ntex or actix written in Rust [sic!]. es4x took 18th place, while Fastify took 58th place. Both of them are JavaScript frameworks – the latter being the most popular out of all three JS frameworks we mentioned with e.g. Microsoft picking it up.

Effortless Scalability

There are three main ways to scale an app:

- **Cloning**

It's when you have each clone of the same app handle some of the work. It's the easiest approach, and it's highly effective

- **Decomposing**

It's the practice of breaking down an app to have multiple, smaller, apps that do the same job. The term is commonly associated with microservices.

- **Splitting**

It's when you want each clone of the app to handle only a part of the data you split the app, then. E.g. each clone would take care of users based on their location, or language.

Back to Node.js – it is the king of scalability. Authors of the runtime heavily emphasize the capability of apps to grow. It's the core feature of the piece of code, and even one of its heavily promoted features. On the project's about page, in the first sentence, we can read that " [...] Node.js is designed to build scalable network applications."

One of its built-in modules, "cluster", provides a ready-to-go way of utilizing all of your machine's processor cores. Yes, even despite the fact that JavaScript does not have a way to use more cores than 1! Utilizing all of the resources of a single server is therefore easy as pie.

The way it works is that it will split the main application among several related processes that will communicate with the main one using inter-process communication. If you are curious about how the messaging works exactly, head over to Node's documentation.

Not only cloning is easy – after all JavaScript is one of the hegemons of microservices. For all of the intricacies of scaling Node.js apps, this article is a great resource.

TypeScript

Most of "serious" apps, libraries, and frameworks are written in **TypeScript** (TS) – JavaScript's superset which provides design-time type safety. Design-time means as much as type safety during code editing.

To give a simple example of problems that could have occurred, should TypeScript have not been created:

If in C#, you write

```
Console.WriteLine(11 + "1");

Console.WriteLine(11 - "1");
```

Nothing will compile, and you won't be able to evaluate the expression, because you get an error. Using a plus is fair game in C#, since it's string concatenation. Using a minus will get you in trouble. Fair enough.

In Python we cannot use plus or a minus.

```
print("11" + 1)

print("11" - 1)
```

This simple code will not run correctly, because we will get a "TypeError."

Now in JavaScript, things get interesting. What we get from

```
console.log(11 + "1");

console.log(11 - "1");
```

is 111 and 10 respectively. No errors, no problems.

Even though these examples are trivial to say at the very least, they illustrate a bigger issue, and give a taste of what could happen should Microsoft have not created their language (TypeScript) in 2012. Luckily, they did, which made wrapping heads around large codebases much easier. What would TypeScript do with the snippet (please notice how it's the same as the JavaScript one)?

```
console.log(11 + "1");

console.log(11 - "1");
```

The answer is: nothing at all. We would get an error when compiling our code, and our editor would complain as well. The wider community also fell in love, quickly converting numerous projects to TypeScript, and adding type definitions for libraries and frameworks that did not transition. As a result, you can enjoy type safety 99% of the time. Artur Łabudziński, Senior Node.js & JavaScript Developer at ITMAGINATION recommends using TypeScript in even the smallest of projects in this recent video interview where we discussed the current state of full-stack JavaScript Development.

In the case of TypeScript, the benefits largely outweigh any possible negatives. Simplicity, speed, cost-efficiency. All while maintaining (design-time) type safety. To be clear, what are the possible drawbacks of adopting the superset?

We are mainly talking about time expenditure. Sometimes it will be necessary to declare type definitions for packages that do not have those. (Partial) Design-type safety can be achieved in JavaScript as well, using JSDoc comments, though it won't

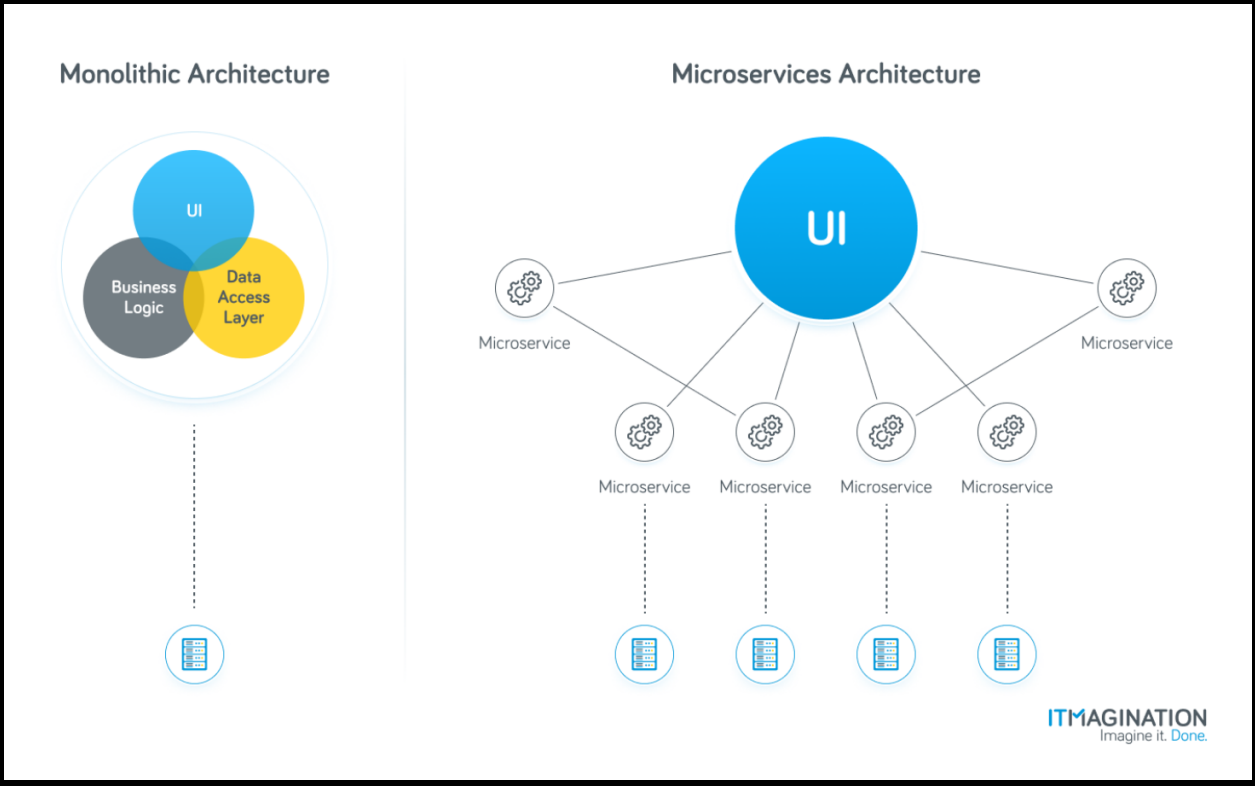
be as good as TypeScript one. Lastly, your .ts files will need to be transpiled to JavaScript. Albeit it does not take long, it is slightly inconvenient.

JavaScript’s Benefits For Enterprise Corporate Settings

In the software engineering world, there comes to mind one aspect that is liked more in large organizations rather than by the small ones. Microservices are an excellent way to scale apps, though they are a bit hard to maintain and test.

Microservices

Up until recently, the general consensus was to build apps using a monolithic architecture. To put it simply, all parts of the app were closely dependent on each other, leading to an important problem. If one part of the app failed, all of the app failed.



That is one of the primary reasons enterprise teams started embracing the microservices architecture. All parts of the app were separate, thus if one feature stopped working, the app remained functional. There are two primary ways the separation is achieved – containers and serverless functions.

Microservices – Containers

When teams are looking to make their apps more independent from the environment from which they are working on, they turn to containers.

They still require some consideration as to what and where they run, though not that much. In general, containerization is a reliable way to ensure we remove the very well-known problem with software engineering - “but it runs on my computer”.

Unsurprisingly, it is where JavaScript & Node.js rule as well. Node.js requires little to run – little RAM and configuration, among others. Developers also have a web server in the standard library, giving developers a lot of room to maneuver without needing to refer to 3rd party code. Even if you decide to use a library, such as Express.js, they are usually just thin layers on top of the built-in Node's Http library.

Create. Build. Use PM2 for managing the production app. Done.

Microservices – Serverless Computing

Serverless functions are another step in abstracting the "where" away. Solutions, such as Azure Functions or AWS Lambdas are a bit of a radical step in separating features and functions. They are also traditionally preferred by enterprise users. In fact, in 2020, around 80% of AWS users from bigger companies use Lambdas.

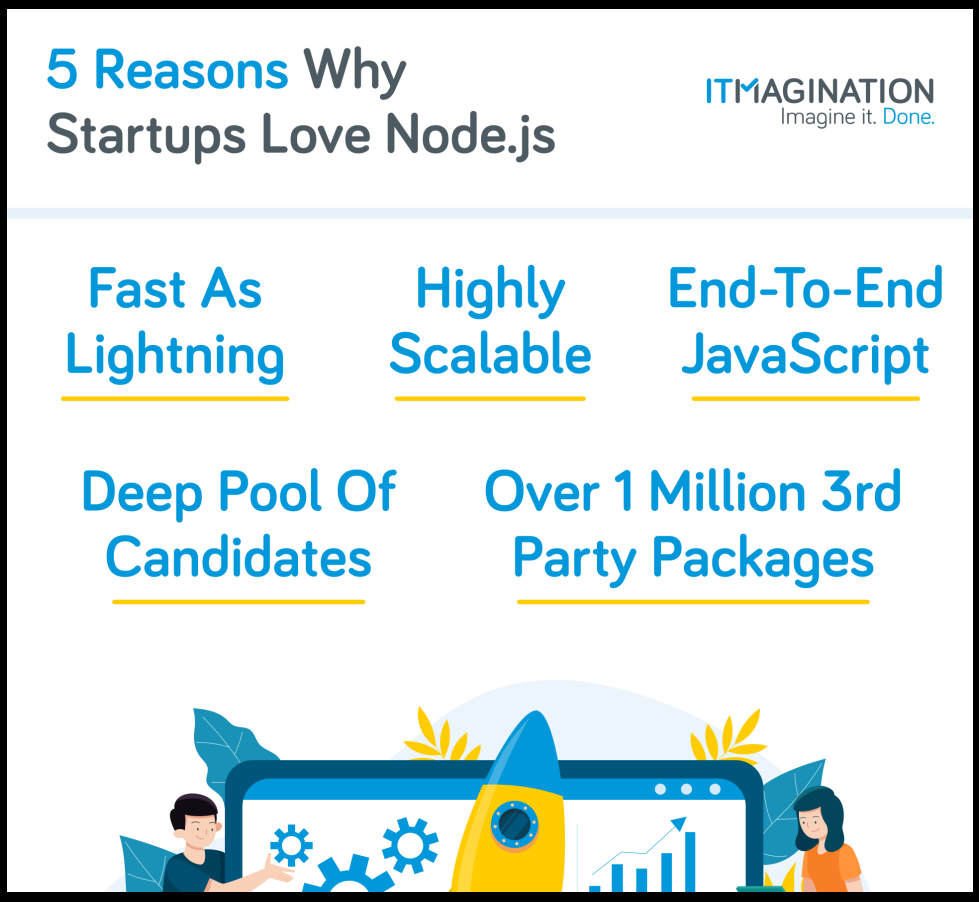
The biggest users embrace them heavily, as debugging can be an issue that is not worth it for the smaller teams, but fault tolerance, isolation, and is heavily desired by the bigger teams. They scale better, are quicker to deploy, and we can avoid any lock-ins.

An important metric here is the cold start, which is what happens when your functions are idle for a longer while. As it turns out, the only language that can compete with JavaScript is Python.

To end with, serverless functions are about the most cost efficient from all microservices solutions, as one pays only as much as one uses instead of paying up front for a month.

JavaScript's Benefits For Startups

Software engineering teams at startups need a language and an environment that will enable them to iterate quickly while keeping the costs down.



No better environment comes to mind than JavaScript & Node.js. As such, the list of rapidly growing companies that rely on some of the frameworks & solutions to create value. Electron is one example of work efficiency (not without drawbacks).

Expo enables teams to reuse code between native mobile applications and web applications (Twitter).

What are the pros in faster and smaller environments, exactly, however?

Fast Iteration Speed & Time To Market

The time it takes to develop is a crucial metric for both big and small, though players that only begin to set up a foothold in a new market benefit from this key metric more than the giants.

In the world of rapid growth, and startups, there is a key term – Minimum Viable Product or MVP. What is it? It is the creation of services or products with basic features with the purpose of gaining customers' attention.

As such, the more advanced features are supposed to be developed with time, and are therefore omitted. On top of that, it might happen that the product-market fit isn't quite there, which

Rapidly growing companies are increasingly often looking to JavaScript with TypeScript. Not only are they generally unproblematic with the correct testing approach - they also enable companies to write code faster than with other popular languages.

We already mentioned how PayPal marveled at the development speed of their team migrating their backend from Java to JavaScript. In short, not only did it take them less time to write the same functions – they did that in fewer lines of code. It

happened all the way back before TypeScript. Imagine the results should the same switch occur now!

Additionally, deployment to production is simple, due to the fact you don't need to compile the codebase. You also don't need to set up a separate web server, such as Apache. You upload the code, you spin up PM2 (production process manager; helps manage apps), and you're pretty much good to go.

Abundance Of Third Party Packages

There is a game in the JS community. Pick a word and check if an npm package exists with a given name. Of course, not quite literally – the point being their breadth of packages is unprecedented, and extraordinary.

For startups, this means there is a safety net in terms of ready-made solutions. Components, libraries, packages, modules – whatever you need is there. In fact, in April 2020, there were more than 1.3 million of them! The package manager makes the installation a breeze as well – both in local scenarios and in e.g. containers. In case you don't fancy this particular solution – no worries. You have got yarn, and pnpm at your disposal.

Industry-standard JavaScript Frameworks

Now that you know why startups are choosing JavaScript, it's time to show you our choice of production-ready, and battle-tested web frameworks.

The JavaScript ecosystem is famous for having a package for every occasion, which is why we will continue to show our choices of libraries & frameworks for all different purposes. Only the ones, however, which are ready for serious use, and the ones that you are safe to bet on.

Front-end

React

The most popular out of the three frameworks. It is also regarded as the most "flexible" one. In the past, it used to be only about the view layer of the app. Currently, with the planned introduction of Server Components it might become a light full-stack framework.

React Pros

+ The most popular

+ An ocean of third-party extensions for every purpose

+ Loved by the dev community

+ Business logic is easy to reuse in React Native apps

React Cons

- Can be a bit heavy

- Mixing JavaScript, HTML, and CSS directly together

- Not too fast

- A limited list of first-party plugins

When To Use React

Software engineers from companies of all sizes choose React, because of its extensive community support, and the depth of the talent pool. You can't go wrong with picking React.

React GitHub Repository

https://github.com/facebook/react

Vue

Vue is the most community-run framework out of the three we list in this section. It is the creation of Evan You, the main decision-maker, and the author of other libraries & frameworks, such as Vite or Vitepress.

Associated with the lowest learning curve out of the three most popular frameworks, it gained traction in China, after which it gained popularity in the West. You can think of it as being an option in between React and Angular.

Vue Pros

+ Can do what you need it to do; it is the most flexible out of the three

+ An ecosystem of first-party frameworks extending Vue's functionality

Vue Cons

- The least popular in startup & enterprise settings in the West

- Handles introductions of new major versions poorly

When To Use Vue

Vue tends to be used together with PHP backends quite often. The talent pool is smaller than in the case of React or Angular developers, however you still won't have much of a problem finding developers who are proficient with Vue.

Go for Vue if you know you won't want to reuse code between the web app and a native mobile app, as the options for Evan You's framework are somewhat poor and/or not well-maintained (Weex, Vue Native, Nativescript-Vue).

Vue GitHub Repository

https://github.com/vuejs/vue

Angular

The last front-end framework in this section. It is the heaviest and has the most features built-in. With Google's backing, it is the most popular in big companies. Its use in the biggest of projects is heavily emphasized with every aspect of the framework written in TypeScript.

An interesting fact is that a lot of internal Google projects use AngularDart, a port of the framework written in Dart.

Angular Pros

+ The most regularly released out of the three

+ Entirely written in TypeScript

+ Heavily Opinionated

+ A CLI generator

Angular Cons

- Heavily Opinionated

- Heavy

- Slow

- Not universally liked among developers

When To Use Angular

Angular is the heaviest among the three mainstream frameworks, and has the most specific use case. You want to use it almost only if you know your front-end is going to get **really** large.

Angular GitHub Repository

https://github.com/angular/angular

Hybrid

Next.js

Next.js, one of the most popular React frameworks that got its popularity for making React development effortless. There are numerous built-in features such as hybrid static & server rendering, static HTML export, API routes (enables you to build an API with Next.js), and many, many more.

Next.js Pros

+ No configuration needed

+ Fast time-to-market

+ Much better SEO than with plain React

+ A great solution for both startups and enterprises

Next.js Cons

- Opinionated routing — you're forced to use page-based routing

- Few third-party plugins

When To Use Next.js

Next.js provides an opinionated setup for developing React apps that is used by the biggest thanks to its simple hybrid rendering. It is often used by those looking to set up a quick MVP because of its smart defaults. This showcases the ability of the framework to be a smart choice for all React development purposes.

Next.js GitHub Repository

https://github.com/vercel/next.js

Nuxt.js

Nuxt.js to Vue.js is what Next.js is to React.js. It simplifies, and turbocharges the framework bringing some much appreciated features to the framework letting you focus more on development.

Nuxt.js Pros

+ No config needed

+ Fast time to market

+ Much better SEO than with plain Vue.js

+ A great choice for startups and enterprises alike

+ Tight collaboration with the creator of Vue.js

Nuxt.js Cons

- Opinionated routing – you're forced to use page-based routing

- Few third-party plugins

When To Use Nuxt.js

Nuxt.js is "The Intuitive Vue Framework." It makes use of the same page-based routing, and is about as opinionated as its React counterpart.

As such, the same use case applies. If you are developing a Vue.js app, and want to focus only on that, you might appreciate the set of smart defaults and the config..

Nuxt.js GitHub Repository

https://github.com/nuxt/nuxt.js

Backend

Nest.js

Nest.js is a framework that goes best with Angular, and will feel familiar for all Angular developers. Not only is it similar in all that it's built entirely in TypeScript. It is also built with big applications in mind, and has an enormous number of first-party modules.

The authors of the framework did prepare numerous guides on how to implement some of the most important features, however. They should serve as a facilitator of adoption for startup engineers, and new learners alike.

Nest.js Pros

+ Heavily opinionated

+ Designed to be reliable

+ Feels great with combination with Angular

Nest.js Cons

- Heavily opinionated

- Few third-party plugins

- Not as popular among developers

When To Use Nest.js

Nest.js is best used when paired with Angular. As a consequence, it's best to use it in projects of a bigger caliber. Using it in smaller projects is usually overkill.

Nest.js GitHub Repository

https://github.com/nestjs/nest

Express

Express is perhaps the most popular backend framework there is, in the world of Node.js. It is, almost always, the first framework that every JavaScript developer learns on their journey, and often is enough to quickly get your MVP up and running.

Unfortunately, it is not recommended using it in bigger projects, as the framework is not ideal for that purpose. For one, it currently does not support async/await, which is a must to handle many requests at the same time.

To make matters worse, the last release of the library (v.4.17.1) was... May 26th, 2019 (!), and Express v5 has been in alpha since 2014 (!). The library is a good starting point for your own internal library, or for an MVP. For other uses, the other options on the list are better options for you.

Express Pros

+ Heavily unopinionated

+ Likely the most popular backend framework in the Node.js world

Express Cons

- Heavily unopinionated

- Likely dead or on life support

- No async/await

When To Use Express

Use when developing your hobby projects, or as a starting point for your internal library.

Express GitHub Repository

https://github.com/expressjs/express

Fastify

Just as the name suggests, the main focus of this library is on speed. It also claims to be focused on developer experience while lowering the cost of the infrastructure. It definitely delivers on the promises, being the fastest out of all major backend Node.js frameworks.

Fastify Pros

+ Built-in Validators

+ Lightning fast

+ A rich plugin ecosystem

+ Opinionated with a way out

Fastify Cons

- Not as popular

- Not many companies use it

When To Use Fastify

When speed is a number one concern for you, and you stick to JavaScript only, the library is for you. It is the fastest mainstream JavaScript framework, and has a rich ecosystem of plugins. In case you want to use TypeScript, you still can.

Fastify GitHub Repository

https://github.com/fastify/fastify

Unit Testing

Jasmine

A “[s]imple JavaScript testing framework for browsers and node.js” is what we will start off with. It runs everywhere that JavaScript runs, making it incredibly flexible. What’s more is that the library has no external dependencies, making it low overhead.

Jasmine Pros

+ No external dependencies

+ Whatever you’re working on — you can test it with Jasmine

+ Built-in assertion library

Jasmine Cons

- Plenty of configuration required

- Slightly unintuitive reports

- Does not have in-built support for rerunning failed tests

When To Use Jasmine

Since it is the most universal position on the list, when in doubt, use Jasmine.

Jasmine GitHub Repository

https://github.com/jasmine/jasmine

Jest

Jest is a library “with a focus on simplicity.” As such, it is zero-config, and provides helpful error messages, when your tests go wrong.

Supported by Facebook, it enjoys great popularity among JavaScript developers, and works out of the box with the most popular libraries. When compared to Jasmine, however, the support for libraries is not that great.

Jest Pros

+ Great documentation

+ Simple

+ Visual regression tests

+ Easy coverage report

Jest Cons

- Can be difficult to get used to it, learning it for the first time

- Does not support as many libraries as Jasmine

When To Use Jest

If you want to have an easy time testing, go for Jest. It used to come as a default option with React apps created with *create-react-app*.

Jest GitHub Repository

https://github.com/facebook/jest

Mocha

Mocha is a library that is often called “Mocha/Chai” as it is used with Chai, an assertion library, the most often. Another library emphasizing its simplicity, it did not fare very well after the introduction of Jest. It is even perhaps a bit unfair to compare the two together – after all Mocha is a test runner, and as such, it requires other libraries to work.

Originally designed to support Node.js testing, its capabilities are now much wider, supporting Node.js, and the browser code as well. It can also perform a plethora of testing, since it can perform integration, and end-to-end tests as well. We included it here, in this section because it is its most common use case.

Mocha Pros

+ Simple

+ Easy support for generators

+ Easy asynchronous testing

+ You may choose between a wide variety of assertion libraries

Mocha Cons

- Reliance on third-party packages (it is not a con in itself, since it was not designed to be a test runner only)

- Intimidating for beginners

When To Use Mocha

Slightly less popular than the aforementioned options, it is also the most flexible, since it is *technically* a test runner only, and as such, you may pair with the assertion library of your choice.

Mocha GitHub Repository

https://github.com/mochajs/mocha

End-to-end testing

Cypress

An evolved library for evolved testing for anything that runs in a browser. It simplified how you performed end-to-end testing – previously you had to use many different frameworks. With cypress all you require is just the base framework – everything is bundled together!

Cypress Pros

+ Doesn't use Selenium

+ Uses JavaScript for defining tests

+ Easy to use

+ Fast to configure

Cypress Cons

- Doesn't use Selenium

- Does not provide support for Safari; runs tests in Chrome only

When To Use Cypress

When in doubt – use Cypress. It's a solid option, with many benefits. It's a safe bet for all of your end-to-end testing purposes.

Cypress GitHub Repository

https://github.com/cypress-io/cypress

Nightwatch.js

Another one of the easy frameworks. It leverages the W3C WebDriver API to perform all predefined commands. It has a clean syntax which feels intuitive, therefore it feels easy to pick up.

There is an option to use Selenium, though by default it uses the standard WebDriver API. A huge advantage of the framework is that it can mimic Firefox & Chrome.

Nightwatch.js Pros

+ Can use different browsers

+ An integrated solution

+ Selenium as an opt-in

Nightwatch.js Cons

- Does not provide support for Safari

- Not as popular

When To Use Nightwatch.js

Out of the two, go for Nightwatch if you are designing your app in the Behavior-Driven Design way.

Nightwatch.js GitHub Repository

https://github.com/nightwatchjs/nightwatch

Future Directions & Conclusions

After a long and rocky history, JavaScript grew up from being an ugly duckling to a beautiful swan.

Throughout the years, there were two parties which made it perhaps the most dominating general--purpose programming language. Those two parties were open-source contributors, and ECMA. The resilience, and dedication of developers has lifted the language out of a big slump, bridging and fixing its gaps. ECMA gave them a solid foundation to build on, augmenting the language with requested features.

What does the future of JavaScript hold?

The future of JavaScript is not exactly clear. It seems as if the case for its popularity depends mostly on the WebAssembly project's future. Long story short, it is possible for some time to compile languages, such as C++ or Rust to the code that browsers can use. Its use has some severe limitations, however, as e.g. you cannot

access the DOM (page's structure) and you are restricted to relatively new browsers only.

It might turn out that JavaScript will stop being the only programming language being able to execute in browsers' engines. If this happens, the superiority of Eich's creation might be challenged, although there will be one aspect of it that will remain true – the simplicity of the language.

Is there a time where I don't want to be using JavaScript?

Yes. JavaScript won't always be a good choice. For instance, if you run a startup developing software for embedded devices or machine learning, then you might want to reconsider the choice of JavaScript for your core product. Here, even after all these years C, and C++ are key. Similarly, in the world of game development, you want to see games being developed in any dynamic language. These are all highly specialized areas, though, therefore if you belong to the ~95% of businesspeople, the language created by Brendan Eich is always going to be a good choice for businesses of all sizes.

About ITMAGINATION

We help our clients innovate by providing professional custom software design & development services, building data solutions, and extending their IT team's capacity. Our team works based on Scrum & Agile principles.

We have over **13 years of experience** in delivering business value across various industries and have a portfolio of over **500 successful projects delivered** to more than **250 clients around the globe.**

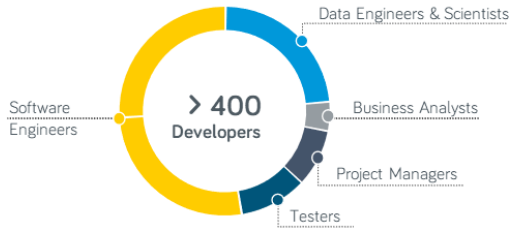
Established in 2008, we are one of the fastest-growing technology services companies in the CEE region, featured in both Deloitte's Fast 50 CE and the Financial Times' FT 1000.

Who we are

We help our clients innovate by providing professional custom software design & development services, building data solutions, and extending their IT team's capacity. Our team works based on Scrum & Agile principles.

Established in 2008, we are one of the fastest-growing technology services companies in the CEE region, featured in both Deloitte's Fast 50 CE and the Financial Times' FT 1000.

We have over **13 years of experience** in delivering business value across various industries and have a portfolio of over **500 successful projects delivered** to more than **250 clients around the globe.**



Our Fields Of Expertise

ITMAGINATION
Imagine it. Done.

Custom Software Development

- Product Design (UI & UX)
- Web Applications
- Mobile Applications
- SDLC & DevOps

Data Solutions Development

- Data Preparation & Management
- Data Processing & Big Data
- Data Analytics & Business Intelligence
- Data Science & Machine Learning

Cloud-Native Solutions Development

- Cloud-Based Application Development
- Big Data & Cloud Data Platforms
- Cloud Migration & System Integration

Our Collaboration Models

End-To-End Project Delivery

Extended Delivery Center (EDC)

Global Projects

> 70%

of our projects serve the international market.

We work with

the world's leading brands across numerous industries; including Banking & Fintech, Telecom, FMCG, Retail, Insurance, Healthcare, Construction, and many more.

Kabbage · Greenlight · orange · Danone · AXA · Skanska · pwc · JLL · FIS · Crédit Agricole · T · L'Oréal · Aviva · U.S. Xpress · Apptical · OLX · BNP Paribas · Santander · eurofins · Mars · ERGO · DSI Underground · Generali · Tikkurila

Selected Technologies

✓ Backend	Microsoft .NET	.NET Core	Java	spring	node	RabbitMQ	ActiveMQ
✓ Frontend	React	ANGULAR	HTML CSS	JS	TypeScript	VueJS	
✓ Mobile	android	Kotlin	iOS	Swift	React Native	Xamarin	
✓ Databases	SQL Server	PostgreSQL	elastic	mongoDB.			
✓ Dashboards & Reporting	Power BI	+ableau	Grafana				
✓ Cloud Partners	Microsoft Azure	aws	Google Cloud				
✓ DevOps		docker	OPENSHIFT	CHEF		ANSIBLE	Terraform
✓ Big Data & Data Science	hadoop	R	Spark	python			
✓ QA & Automated Testing	Protractor	JMeter	cypress.io	appium	Jest		

Why Choose Polish IT Talent

Largest Talent Pool in Central/Eastern Europe

Home to over 300,000 professional full-time developers – accounting for roughly 1/4 of the entire CEE region.

Lower Rates Without Sacrificing Skill

The cost of living is still lower in Poland, which can help lower your development costs by up to 50%, by choosing to hire Polish talent.

Long Tradition of Engineering

From communist times to the modern day, Poland's education system has always emphasized STEM fields. Computer Science is the most popular degree at Polish Universities.

IP & Data Protection

Poland is part of the EU, which guarantees strict adherence to Data Protection and protection of Intellectual Property Rights.

Seamless Communication

Poland ranks #11 in the world for English proficiency according to the English Proficiency Index, placing Poland just behind Germany and ahead of Belgium.

Critical Thinking

Polish engineers feel comfortable providing valuable input to the development of projects and challenging the status quo which helps deliver a higher quality end-product.

Extended Delivery Center Collaboration Model

Skilled interdisciplinary teams from ITMAGINATION's software house ready to extend your development capacities and cover entire projects

Rapid Scaling

- Interdisciplinary teams aligned with your tech-stack
- Our bench increases flexibility and saves time
- Mid-term and long-term capacity plan
- Entrust us with a project or part of your roadmap

Know-how

- Due diligent candidate verification
- Processes and management advisory
- Team composition and seniority
- HR and people's development is on us

Commitment to work in partnership with your teams in an agile manner, from planning to maintenance, ensuring that your goals are ours.

Agile

- Sprints ensure progress tracking and feedback
- From day-to-day meetings up to Steering Committees
- Your stakeholders have full ownership of the value delivered
- Our teams embrace SCRUM principles

Partnership

- Your business goals are our goals
- Open communication
- Your tools and SDLC, your IP, your ownership
- Kick-off within 4 weeks

Extended Delivery Center Benefits

- Access Poland's Talent Pool
- Scale Quickly & Safely
- Elasticity
- Outstanding Competencies
- Long-term Planning
- Reduce Operational Risk
- Technological Consultancy
- Ease of Mind Focus on Core Business
- Optimize Your Time & Costs

Thank you for checking out our eBook, get in touch!

ITMAGINATION
Imagine it. Done.

Paweł Borowski
Board Member
pawel.borowski@itmagination.com

Marcin Dąbrowski
Chief Innovation Officer
marcin.dabrowski@itmagination.com

Marketing Team

Aleksander Jess **(Author)**
Content Marketer
aleksander.jess@itmagination.com

Hisham Itani **(Editor)**
Head of Marketing
hisham.itani@itmagination.com