Oliver Holmberg

Migrating from JavaScript to TypeScript and its advantages.

Metropolia University of Applied Sciences

Bachelor of Engineering

Degree Programme in Information and Communications Technology

Bachelor's Thesis

18 April 2023

Abstract

Author:	Oliver Holmberg
Title:	Translating from JavaScript to TypeScript and its advantages
Number of Pages:	20 pages + 1 appendices
Date:	18 April 2023
Degree:	Bachelor of Engineering
Degree Programme:	Degree Programme in Information and Communications Technology
Professional Major:	Mobile Solutions
Supervisors:	Janne Salonen

This study aims to make it a little bit clearer whether a project will benefit from having the codebase translated from JavaScript and TypeScript, and the biggest difficulties that showed up during the project that was done in conjunction with the written part of the thesis.

Keywords: TypeScript, Refactoring, Code Migration

Contents

List of Abbreviations

JS:	JavaScript, Object-oriented programming language, weakly typed, and the subject of the translation work.
TS:	TypeScript, a JS based, Microsoft developed programming language that is strongly typed. The language the previous code is being translated to.
JSX:	JavaScript XML, a JavaScript syntax extension that allows for component rendering in a similar manner to HTML, commonly used in React.
TSX:	TypeScript XML, the same thing as JSX but for TypeScript
XML:	eXtensible Markup Language, a markup language and file format.
npm:	Node Package Manager, a package manager created for JavaScript, it is being managed by npm, Inc.
IE:	Internet Explorer, an internet browser developed by Microsoft and used before the release of their more modern Microsoft Edge browser.

1 Introduction

This thesis is about translating from JavaScript to TypeScript, whether such a job is recommended, the difficulties, but also the benefits associated with that task and the result. As a basis for the thesis, a translation job was done for Musti Group Oy, where a piece of internal software was migrated from JavaScript to TypeScript where it’s packages we’re also upgraded, and the theming engine was changed.

This is largely written to help people in similar positions figure out whether making the switch from JavaScript to TypeScript is worth the effort, while also diving into exactly how much effort a project like this could take, what roadblocks should be expected depending on the scale and parts of the project and what exactly TypeScript offers that JavaScript cannot.

In addition, the thesis discusses the methodology for updating and replacing NPM and Yarn packages, due to the differences between TypeScript and JavaScript, one might have to change or modify packages to get them to work in TypeScript.

2 JavaScript’s shortcomings

JavaScript, shortened to JS, is a weakly typed programming language which is heavily utilized in web development, about 98% of all web pages use JavaScript in some capacity. However, as TypeScript is still essentially just an expansion of JavaScript, a lot of those 98% are also programmed, at least partially, TypeScript. According to the “The State of JS 2022” (1) study around 68 percent of developers prefer TypeScript as their "Flavour” of JavaScript, this was of the 69.2% of respondents who completed that part of the survey. This unfortunately doesn’t really give us any concrete evidence on whether this is the language these developers use in any professional capacity, just whether or not they would prefer to use that we’re they the person making the choice, still, it is

arguably valuable information for a team manager that is there to make the decision of what technologies the next big project would use.

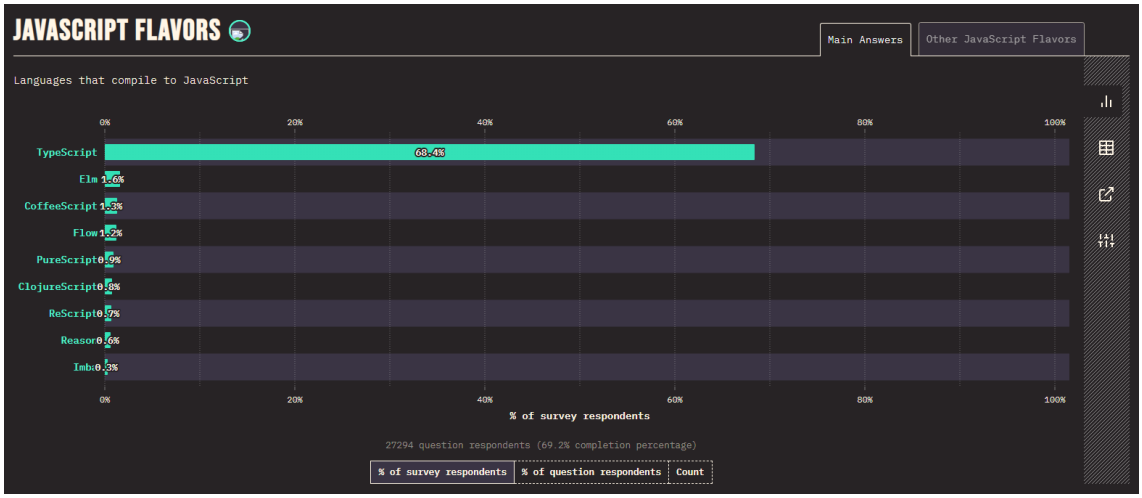

Figure 1. A screen capture from the State of JS 2022 Survey showing the percentage of developers using TypeScript and other flavours of JS when developing.

What this thesis will partially focus on is the failings of weakly typed programming languages and how these can be abused in JavaScript to do so called “hacks”. The issue with using “hacks” in JavaScript is that the parts of the program using code like this usually works shoddily and oftentimes the author of the code isn’t even entirely sure why or how the code works. JavaScript having dynamic typing, often changes the types of data on the fly. As an example.

```
23 = "23"
```

Will return the “true” value, since in JavaScript a number value can equal a string. When the JavaScript compiler is asked whether or not the number 23 is the same as the string 23, the compiler responds that it is indeed the same value, even if their types are disparate. This in turn becomes a huge hurdle when doing TypeScript translation because you now need to make sure that data maintains a consistent typing throughout its lifecycle, and you need to decipher what the original author even meant for the program to send and receive.

The weird ways in which JavaScript behaves with types was largely the content of a 2012 talk by Gary Bernhardt called Wat.

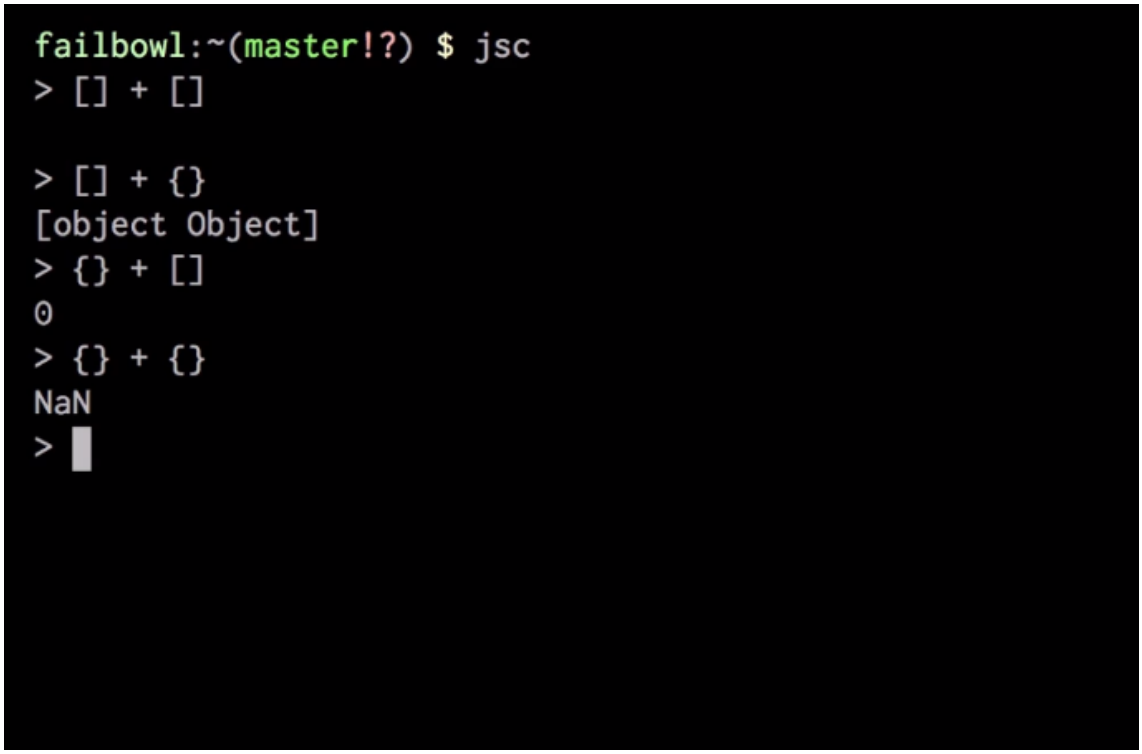

Figure 2. A Screen capture of Gary Bernhardt's talk, Wat.

> “Now what would Array plus Object be? This should obviously be a type error because those are completely disparate types.”
> Gary Bernhardt, 2012.

In Figure 2 from Gary Bernhardt’s talk, Wat (2). We see a few great examples of how JavaScript handles types in the context of addition. In the first line you can see two empty arrays’ being added to each other, becoming an empty string, changing types. On the second line an empty array and an empty object are summed up to a singular object. And on the third line, reversing the second line, an object is summed up with an array, which confusingly results in a zero, changing types radically. The fourth line gives us the only correct result, wherein two empty objects summed up together results in, NaN, not a number, which is technically correct.

This isn’t necessarily bad if the developer is aware of what they are doing when they might make mathematical operations between two disparate types. But this often also leads to a lot of confusion and mistakes in new developers, especially ones trying to make sense of an already existing codebase as new hires. This is something that TypeScript gets rid of entirely.

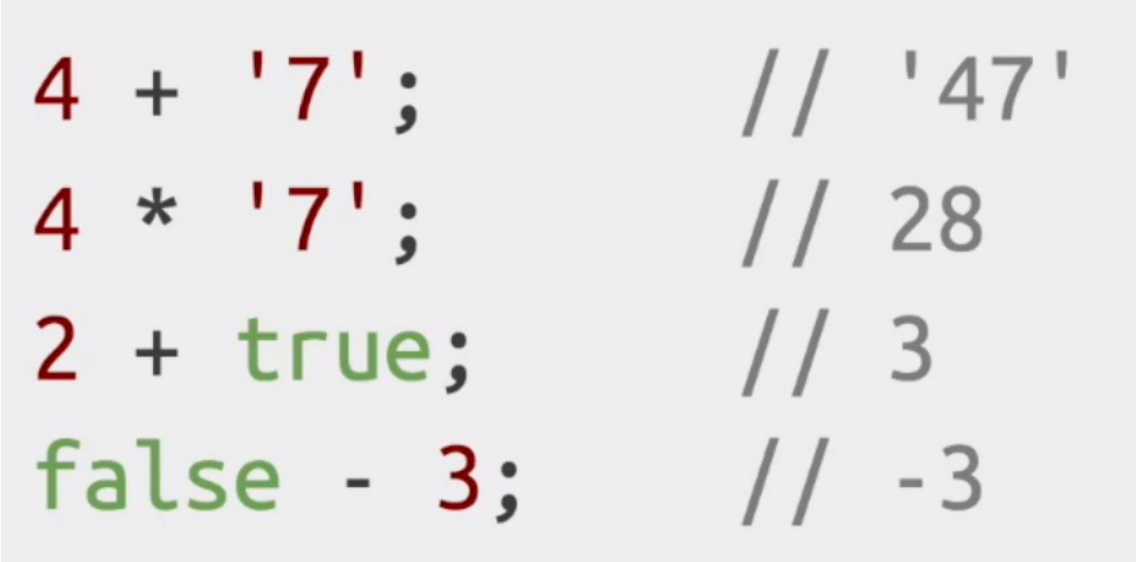

Figure 3. A screen capture from the video Typing: Static vs. Dynamic, Weak vs. Strong / Intro to JavaScript ES6 Programming, lesson 16 by Codexpanse (3) showcasing some more weird typing quirks in JavaScript.

Another example is shown in Figure 3, where adding up the number 4 and the string 7 result in the string 47 essentially dynamically typing the number 4 into a string and adding them using the rules that would be applied when doing addition on two strings. When multiplied they instead become the number 28, showcasing some of the inconsistencies in the way JavaScript dynamically declares types on variables. The values true and false are treated as Booleans until they’re used in an arithmetic operation where they are typed into their numeric values 1 for true and 0 for false. This would not pass in TS code as they would be type errors due to their disparate typing, a string could not be used in an arithmetic operation in TypeScript.

2.1 JavaScript History

JavaScript, originally Mocha, then LiveScript released in 1995 (4), it was developed in ten days for Netscape by Brendan Eich, with the goal of making the web pages of that day more dynamic. It released as a part of the Netscape Navigator software.

The name change from LiveScript to JavaScript happened a little while after release so it could be marketed as a companion language to Java, the hottest programming language at the time. Aside from some similarities in syntax, JavaScript didn't have a lot in similar with Java (5), and was employed in an entirely different use case.

In the year 1996 Microsoft also wanted to get in on the action with their web browser Internet Explorer and released an almost identical scripting language called Jscript to run on their own browser.

After this there was a push to standardize JavaScript and in June of the year 1997, JS was standardised by Ecma International under the title ECMA-262 (6). The objective of the standard is to ensure interoperability of JavaScript functionality throughout browsers. These days there are three scripting engines in wide use, SpiderMonkey, as utilized by the Firefox Browser and all its forks. Chrome V8, in use by all chromium-based browsers. And JavaScriptCore, which is being used exclusively by Apples Safari browser.

2.2 What JavaScript is well suited for?

If you're in the process of starting a new project and you're wondering whether to pick TS or JS, the only time JS would be recommended over TS is if you need to hack together a personal program that does one very simple thing and questions of security, maintainability and readability are not primary concerns.

Obviously, it is perfectly possible to write a secure, easy to maintain and easy to read program with JavaScript as well, it just takes more work than TypeScript in

the long run. JavaScript is good for smaller programs whereas TypeScript excels in larger applications that have a longer forecasted lifespan and a wider user base.

Because TypeScript is essentially just JavaScript and more, there is usually not a compelling reason not to just go for TypeScript when starting a new project, especially as it is a very easy language to learn and use for people already familiar with JavaScript.

JS is however a bit faster on compile time, requires less code and is a bit easier to learn, as learning TS asks you to first learn JS (7). Another compelling reason to pick JS over TS is framework compatibility, if your framework of choice doesn't support TS and has no compelling alternatives then the choice is made significantly easier, but this is less and less of an issue over time as most modern frameworks support TS. Overall JS is an easier, lower maintenance language and works well for small internal and personal projects or mock-ups. But due to concerns of project life cycle and security it isn't recommended for larger projects.

3 What TypeScript brings to the table.

So, let's get to looking at what TypeScript actually is, and what new features it brings to JavaScript, and why you might want to consider switching over to TypeScript in future projects, or even migrating current projects from JavaScript over to TypeScript.

3.1 What is TypeScript?

TypeScript is an open-source programming language developed by Microsoft, and is a superset of JavaScript (8), meaning essentially, it functions as JavaScript does with optional static typing. Theoretically speaking existing JavaScript applications are entirely valid TypeScript applications and could be ran as is. By just changing the file extensions. In practice this is rarely the case

as most applications written in JavaScript use packages and “hacks” as mentioned previously, and when changing the file extension the developer will most likely be faced with a good chunk of errors on compile time.
Some errors could simply be that TypeScript asks you to be a little bit clearer about your intentions, others could be proper bugs that have made it under the radar in the ambiguity that JavaScript affords.

As a result, a program written in TS will be more stable, secure (8), easier to understand for new team members, and more pleasant to maintain.
More stable because it stops a programmer from making short sighted fixes to issues that can easily cause lead to far bigger issues down the line. And easier on new developers as from the eyes of a junior dev that has just started on a codebase it is far more pleasant when, let’s say variables with the names X and Y also have a typing right next to them in the form of X: string and Y: number. This also makes maintenance of the codebase easier as you can more easily follow the path that data takes.

Programming in TS takes a little bit more time, and when initially starting out will probably lead into a lot of errors when done as a migration job and not writing TS from the start. As an example, the translation job that was done for Musti Group Oy, resulted in over 1200 errors when changing the file extensions from JS to TS/TSX (TypeScript XML). It is most likely easier to go through these errors one file at a time rather than changing all the file extensions at once, this is thankfully, very easy. As JS and TS can be ran in the same program interchangeably.

A raw and simplified list of the most important features that TypeScript adds to JavaScript is, type annotation, compile time error checking, type inference, type erasure, interfaces, enumerated types, generics, namespaces, tuples and async (asynchronous) functions.

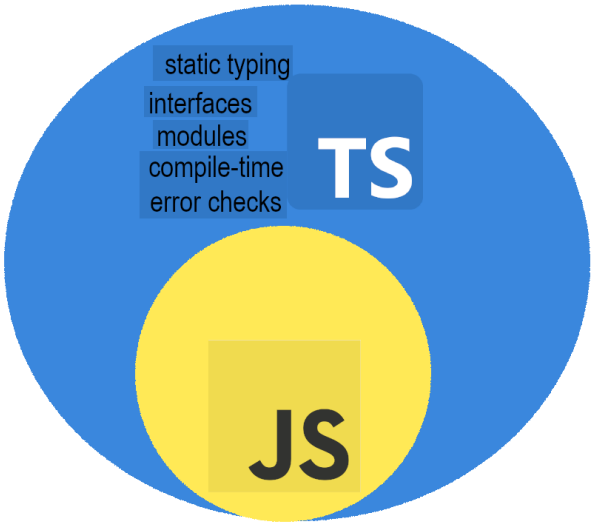

Figure 4. A visualization of Typescript's relation to JavaScript. TS being an extension of JS.

3.2 History of Typescript

TypeScript was initially released under version number 0.8 in October of 2012 after two years of development at Microsoft, but at the time of release was only supported by Microsoft Visual Studio 2012. The reason for Microsoft wanting to develop TypeScript comes from how lacking JavaScript was for developers working on larger scale projects. (9)

3.3 The difficulties in translating from JavaScript to TypeScript

As previously mentioned in chapter 3.1, the migration job is in theory very simple but in practice often takes quite a bit of work. Most of the time this is not that difficult but still takes up a long time. The project at Musti Group Oy lasted from November 2022 to April 2023, this was a very large project in the tens of

thousands of lines of code and primarily done by a single developer with help from a more experienced team which obviously delayed the project quite a lot.

The translation job was done file by file, after initially trying to run the application after changing every file extension to TS/TSX the developer was faced with over 1200 errors in the program code and it was decided that going file by file would be the easier way, it is entirely possible to run the program as a mix of TypeScript and JavaScript and this lead to the ability to fix problems as they crop up in a file, one by one, instead of looking at a very daunting list of hundreds or thousands of errors. The initial stages of the translation job mostly related to fixing issues that arose when updating from React Router v4 to React Router v6 and various other small issues, this part of the project took approximately two months to finish, a lot of help was offered by two experienced consultants and a junior dev that had worked at the company for a little under a year prior. After that the objective was to remove Rebass from use and replace it with googles MUI library, this took about a month. Following this was when the translation work started to earnestly focus on the needs of TypeScript, during all of the previous stages everything needed to be explicitly typed in order to get it to run through TypeScript, but due to a lack of foresight and experience a lot of things we're left typed as any, the catchall typing that treats your variables a lot like JavaScript, especially anything a little more complex that needed more complicated typing to account for arrays and several different types within a function would pretty much be left untouched, and running entirely as it would in JavaScript, this was partially also necessitated by the fact that the app was running both JavaScript and TypeScript at the time, due to the choice of translating the application one file at a time.

3.3.1 JavaScript to TypeScript translator software.

There's quite a lot of translator software available online that promises to do the job for you. These are rather functional if the application runs in a single file and is relatively simple, even very advanced software like ChatGPT can't do translations when your software starts implementing packages and runs in

several files. There is also the question of data privacy, especially if this is being done for an internal software and not a private small project, feeding translator software or AI, internal and private program code should not even be considered out of a worry for privacy and security.

3.4 The primary differences in JavaScript and TypeScript code.

Let’s look at how a very basic JavaScript application would differ from the same application but created in TypeScript.

For this example, we are using a very simple snippet of code that is made to solve the famous Fizz Buzz programming test, the code we are using in this example is from Zenware’s FizzBuzz github repository that offers a Fizz Buzz solution for most programming languages. (10)

First in JavaScript

```
function fizz_buzz(num) {
    if (num % 15 === 0) {
        console.log("FizzBuzz");
    } else if (num % 5 === 0) {
        console.log("Buzz");
    } else if (num % 3 === 0) {
        console.log("Fizz");
    } else {
        console.log(num);
    }
}

for (var i = 1; i <= 100; i++) {
    fizz_buzz(i);
}
```

What the code does is generate 100 lines of console logs that output Fizz when the number of the line is a multiple of three, buzz in the case of multiples of five and fizzbuzz for multiples of three and five.

When translated into TypeScript, the program code doesn’t change a whole lot.

```
function fizzbuzz(num: number): string | number {
  if (num % 15 === 0) return 'FizzBuzz';
  if (num % 5 === 0) return 'Buzz';
  if (num % 3 === 0) return 'Fizz';
  return num;
}

for (let i: number = 1; i <= 100; i++) console.log(fizzbuzz(i));
```

We get rid of the else ifs as these are not necessary in TypeScript code and we assign a type declaration to the num variable, and a string or number type declaration to fizzbuzz in the first line as we create the function. The math operations are done in an identical manner to how it was done in JavaScript.

Assigning a type declaration to a variable in TypeScript is very simple and is done by adding a colon to the end of the variable name, after which we give it a type declaration, for example.

```
num: number
str: string
numstr: string | number
```

The difficulty in type declaration comes with larger programs and variables that may need to change types during their route. In this case, if we had for example a date variable that needed to be converted into numbers, for example when trying to output a JSON string. We would use conversion functions baked into TS like parseInt and toString.

```
let date: Date = new Date();
date.parseInt()
jsonString + Date.toString()
```

This is a very crude example of what you might have to do and in practice going through this many type conversions is often overkill, but it is shown here as an example of how type conversions might be done in actual code.

3.5 Methodology and how to start out.

When starting out the migration process from JS to TS the project at Musti started out with translating smaller parts of the application like features and in-house package solutions, at the start of the project it is also necessary to create a tsconfig.json file (11) an example of one would be,

```
{
  "compilerOptions": {
    "outDir": "./built",
    "allowJs": true,
    "target": "es5"
  },
  "include": ["./src/**/*"]
}
```

What this example does is, on the first line "outDir": "./built", specifies an out directory to the emitted .js files as TypeScript compiles into .js. In the example the directory would look like.

```
$ tsc
built
└── index.js
example
├── index.ts
└── tsconfig.json
```

The second line "allowJs": true, specifies that the project will accept .js files as inputs, this is crucial when migrating a project as you are most likely to want to do it one file at a time, this lets you run .ts and .js files with eachother.

The third line "target": "es5" lets you set a target for translating newer JavaScript constructs to an older standard, in this case ECMAScript 5.

The fourth line "include": ["./src/**/*"] specifies which parts of the program are included in the tsconfig compilation, using the “./src/**/*” folder in this case includes the whole program. You may exclude and include specific parts of the program if you don’t wish to translate the whole application to TS, which may in some edge cases be necessary. Usually in this situation it would be better to use the “exclude” tag to target specifically which parts of the project are not intended to be checked by the TS compiler.

There are a lot of configuration options for the tsconfig.json file so looking through the documentation (11) is well advised. Important options to keep in mind are for example jsx, that lets you control how jsx elements are handled in the compilation phase. strictNullChecks, which in best practice is set to true in order to enforce that no null values slip through the cracks.

During the migration process there tsconfig.json file will most likely grow in size quite a lot as new configuration options are required for certain parts of the program, but a simple tsconfig file as at the start of chapter 3.5 should be sufficient to start with, and while most IDE's (Integrated Development Environment) autogenerate the tsconfig file, it is still recommended to go through the autogenerated configuration options in order to get a grasp on what exactly is happening during compile time.

After setting up the tsconfig file you are ready to start the migration process. For someone not yet well versed in TypeScript it's easiest to start out with shorter files that are easier to translate to get used to the quirks and differences in TypeScript compared to JavaScript, it's also recommended to set up a separate file for type declarations, especially for complicated functions with several data types.

Most simple files will however consist almost entirely of just simple typing of the variables and functions contained within that file, front-end related files will be among the easiest files to translate directly, as they are often typing related errors. It is also important to remember that if your file contains any JSX code, as most front-end related files tend to have, to have that file ending with a .tsx extension, instead of the traditionally used .ts file extension.

4 Interview with a programmer.

For this thesis three anonymous interviews were conducted, the interviews we're rather open ended with very broad and simple questions, as such they

varied in length greatly. Below is a paraphrased and condensed version of the conducted interviews, the interviews have been included in Appendix 1.

When asked about how much previous experience the interviewees had with working on TypeScript applications the answers didn't vary greatly, all the developers had a few years of on and off experience and a good understanding of TypeScript as a whole, one of the developers used almost exclusively TypeScript in a professional capacity.

On the topic of whether they would rather use JavaScript or TypeScript for projects the answers still stayed very similar each other, it was generally accepted that TypeScript is better for larger projects with several developers due to its easy readability and error checking during compile time, but also due to the fact that TypeScript is a newer programming language with more interest that the majority of the team would rather program in, even if some people would prefer JavaScript. JavaScript was still generally preferred for simple applications and scripts. One of the developers preferred programming most everything in TypeScript, so that they could directly impart any experience gained on private projects to their professional life.

The final question of the interview was a very open ended one, the question was broadly, what about TypeScript do they like and dislike, this question gave a much broader and interesting range of answers.

One developer was of the opinion they would pick JavaScript over TypeScript, as his iterative and on the fly programming style worked better with JavaScript, in TypeScript the developer would often get stuck on small and insignificant errors during compile time that could have been fixed later down the road as they showed up during testing. They were not a fan of having to sink in time for typing all the different functions and properties of a program, where most of the typing would very much just be done with on the fly, it's the last 5% of the typing that takes up several hours of time. Another developer thought the compile time error checking was very useful and helped a lot with stopping faulty logic and

bad typing before it every became a problem, stopping the application from having deep seated logic and typing issues that may only be noticed years later. What was also universally disliked was working with packages in TypeScript as some of them don't come with bundled typings, or don't have any typings to load in at all. This would be far less of a problem if the software was built for TypeScript from the beginning. Another interesting viewpoint that was brought up was the fact that since JavaScript has been the de facto programming language of the internet for over 25 years now it has a lot more material written about it, so fixing a really specific and tricky bug in JavaScript is often quite a bit simpler than it is in TypeScript, since you've got far more written material to assist you.

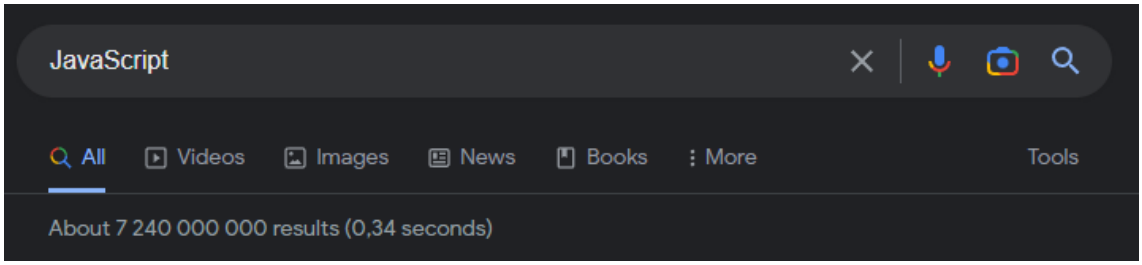

Figure 5. Google Search results for the search term "JavaScript", showing over 7 billion results.

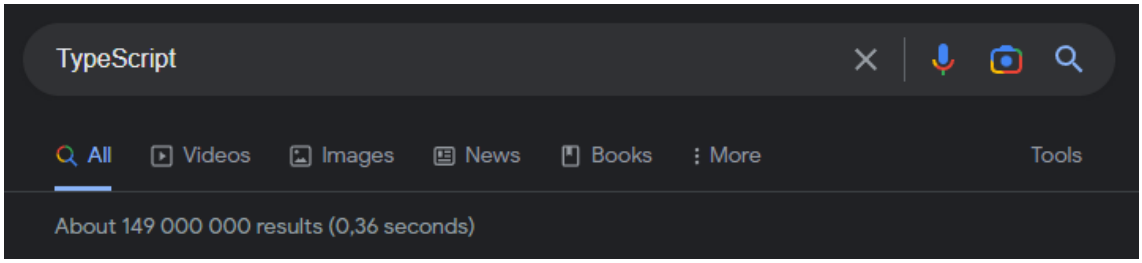

Figure 6. Google Search results for the search term "TypeScript", showing roughly 149 million results.

In conclusion, if you're a very experienced developer well versed in JavaScript, TypeScript might just feel like a chore to implement, but It'll be a great help to any new junior or intern developers starting on your team that haven't been properly introduced to the software yet. It will also make the intro to a codebase far easier to explain and understand. And if you're looking for help on Google

for a specific issue you're having and you're not finding what you're looking for, it might be worth checking results for a similar error in JavaScript to see if the resolution in TypeScript might be similar. As an example looking for something like "IE11 Page does not render TypeScript" gives you about 335 000 results, whereas replacing TypeScript with JavaScript at the end of the search term gives 954 000 results.

5 Should I, my team or my company consider translating existing software from JavaScript to TypeScript.

You might be thinking of a project that would benefit greatly from strict typing as you're reading this, maybe a very simple piece of software that has a lot of Junior developers attached to it, that would benefit from the added readability that TS brings to the table. Or a huge and sprawling application that has a lot of data flowing through it from several different back-end servers that would greatly benefit from having that flow of data made a little bit more readable and secure with typing. Here's what one should consider before settling on whether a migration from JS to TS is necessary or beneficial in the first place, as there are a few things in your program that could make the work a lot more difficult than initially expected.

5.1 Issues most likely to crop-up when starting off the project.

The issues that showed up most prominently at the start of the project that was done for Musti were mostly things that were indirectly related to the TypeScript translation, for example when using React it was pretty much necessary to update React in order to translate to TypeScript, this was partially due to the amount of packages that we're already present in the project, the versions of these packages that came bundled with type declarations for TS also required newer versions of React in order to function.

Most large-scale applications usually use some NPM or Yarn packages to cut down on the amount of coding required by the application, a lot of smaller

packages, but for example crucial parts of the program like MSAL authentication for logging into the software. Some packages come preloaded with TypeScript declarations but in some cases, this required an update of the package for it to have those declarations, as they weren't present in the old version of the package, which then also lead to them needing a newer version of React to function.

Some packages had separate TS declarations that needed to be added through yarn or npm (package managers commonly used in JavaScript and React development). And other packages were so out of date they didn't have any TS declarations to begin with, these packages then needed alternative packages to solve the problems that cropped up when deleting them, or completely new code to replace the functionality that the packages provided.

This took up a very significant portion of the work that was done for the project, overall, the time investment on this part was approximately 2 months of work for one intern developer with a lot of help from a junior dev and occasional help from two very experienced consultants.

What was initially penned as a three-month project ended up taking in excess of five months to complete

If you are looking to refactor code into TypeScript it's likely you're looking at a project that needs some other parts of it brought up to standard as well, alongside the react update and fixing the packages, there was also a switch from Rebass theming to Googles MUI theming, the theming change is quite unlikely to be an issue in most cases as a vast majority of apps that have been

sufficiently maintained should be using a well-documented and up to date theming package.

5.1.1 Updating npm/Yarn packages.

npm (Node Package Manager) is the default package manager for JavaScript when using the Node.js backend runtime environment. Yarn is an alternative to npm developed by Meta (formerly Facebook) with more of a focus on functioning in large codebases, Yarn was used in the project made for Musti.

When being faced with an error code in your IDE, if the error originates from a package you can often fix that by simply checking versions and their compatibility with other packages in your software, these packages are often left on the version that was used while originally programming the software which can lead to a lot of outdated packages if you haven't actively maintained the software in a few years, as was in the case of Musti's in house application. However, choosing the right package isn't always as simple as updating to the most recent version, as some packages can have dependencies to each other, and if one package has had a drastic change in the program code, could lead to the other package not functioning properly, or at all.

This is why updating and checking package interoperability can be quite an arduous process and take a good chunk of time from the development budget, directly relating to the number of packages used in the program code in the first place. If the application is relatively free of third-party packages and is mostly in-house code, translating the program should be a bit faster, especially so if the people who have written those original components are still in the house.

5.1.2 Internet Explorer 11 and ECMAScript 5 compatibility.

This is very much an edge case as most applications have entirely ditched support for Internet Explorer along with Microsoft officially ending the lifecycle of the product. But getting modern TypeScript to function on Internet Explorer

(henceforth referred to as IE) requires quite a bit of work and fiddling about, this is also a very different process in most every case depending largely on what packages and runtimes you are using with TypeScript. The solution in our case was to use appropriate polyfills in the right places, there's a wide selection of different polyfills available for different tasks and it shouldn't be too much work figuring out the correct one, GitHub Issues and Stack Overflow are usually loaded with people in similar predicaments that have managed to fix it. In other cases, the solution can be as simple as fixing the target of your application in the tsconfig.json file. Or as complicated as using webpack or babel to transpile all your code into a palatable version for IE11, other solutions include but are not limited to.

Downgrading react-scripts to version 3.1.0, since the versions after this cause a lot of compatibility issues with IE11. Installing and importing polyfills, the core-js or the react-app-polyfill package being the most common ones. Also check your browser list in the package.json file. You may also be targeting the wrong ECMAScript version, IE11 has absolutely no support for anything past ES5, and even targeting ES5 you may still get some issues as IE11 only has a 97% compatibility rating for ES5.

When debugging for an IE11 based application you may run into a ton of other issues as well, based entirely on the fact that you are running Internet Explorer, enough issues to potentially write an essay on that alone. The best practice in this case would be to simply stop supporting Internet Explorer and update the internal browser of whatever integrated systems you may be using. You should only start doing work for supporting IE11 if you absolutely must. Microsoft has stopped supporting IE11, maybe it's time for you to do the same.

5.2 Resources recommended and required for a project of similar scale.

It should be expected that a project of such scale, in Musti's case around 10,000+ lines of code, will take a good amount of time. For this project using junior developers or interns isn't necessarily a bad idea, as was done at Musti,

there’s still a need to have consultants/senior developers on hand for the newer programmers to ask for help whenever they are faced with a more difficult task, which will no doubt crop up quite a lot during a larger project.

Other resources required are rather obvious things like sufficiently powerful computers for the developers, web development isn’t particularly taxing on computers, but the compilation time can vary greatly depending on the power of the computer being utilized by the developers.

The most important part of the computer when working with TypeScript in for example Visual Studio Code, would be the amount of RAM in the computer. 8GB is a good minimum for comfortable working in rather large codebase, but the larger the program, the more you would want. 16GB of DDR4 (Double Data-Rate) or DDR5 is pretty much seen as the current standard as of 2023.

The developers working on the translation process don’t necessarily need to be versed in TypeScript before the project starts, but a good understanding of JavaScript is somewhat important, learning TypeScript for someone who already knows JavaScript is a rather trivial thing as it just adds stuff on top of JS. It would still be very useful to have at least one or two people already well versed in TypeScript working on the project, as having someone to consult with whenever you feel a little bit unsure of some larger piece of code you’re working on is a lot better than just searching up an explanation online and hoping the person who wrote it knows more than you do.

6 In Conclusion

So, is TypeScript the right flavour of JavaScript for you and your team? Hopefully this thesis can help people come to a conclusion, it’s a fantastic expansion on top of JavaScript’s base features that comes with a ton of security, maintainability and ease of use features. But it’s also something that takes a bit more time to initially work on, and even with a team that is well

versed in how TypeScript functions, JavaScript is the faster way to hack something together.

So, in case what you're working on is a small internal or personal use tool like a web scraper or equivalently simple application then JavaScript will let you program that in a shorter amount of time for pretty much the same results. But if it's a huge application, especially one that is intended to be used by the broader public, outside of the immediate vicinity of just you and your team, then TypeScript offers a safer alternative.

If you're thinking about a migration process it's very important to look towards the future of your application, what is the expected lifecycle of the software. If you're planning on possibly getting rid of the software and going with an alternative solution in the near future, doing the migration process from JavaScript to TypeScript is likely not the ideal way to go about the maintaining of the software, and you should instead investigate the possibility of making the existing JavaScript code as stable and secure as you possibly can with a smaller team. If the software your team is working on doesn't have an end-of-life date set to it yet, doing the translation work may be very beneficial. Especially if the software in question is something intended to be used by a larger user base, one that might try to intentionally or out of mistake break things for their own gain or amusement.

References

1 The State of JS 2022. Online. State of JS. <https://2022.stateofjs.com/en-US/> Read 3.4.2023.

2 Gary Bernhardt. 2012. Wat. Online. Destroy All Software. <https://www.destroyallsoftware.com/talks/wat> Watched 20.3.2023.

3 Codexpanse. 2017. Typing: Static vs. Dynamic, Weak vs. Strong / Intro to JavaScript ES6 Programming, lesson 16. Online. YouTube. <https://youtu.be/C5fr0LZLMAs> Watched 20.3.2023.

4 NETSCAPE AND SUN ANNOUNCE JAVASCRIPT, THE OPEN, CROSS-PLATFORM OBJECT SCRIPTING LANGUAGE FOR ENTERPRISE NETWORKS AND THE INTERNET. 1995. Online. Netscape. <https://web.archive.org/web/20070916144913/https://wp.netscape.com/newsref/pr/newsrelease67.html> Read 20.3.2023

5 Fireship. 2019. The Weird History of JavaScript. Online. YouTube. <https://youtu.be/Sh6lK57Cuk4> Watched 20.3.2023.

6 JavaScript History. Online. W3Schools. <https://www.w3schools.com/js/js_history.asp> Read 20.3.2023.

7 Abhimanyu Krishnan. TypeScript vs. JavaScript: Which is best in 2023. 2023. Online. Hackr.io <https://hackr.io/blog/typescript-vs-javascript> Read 27.3.2023.

8 TypeScript Doesn't Suck; You Just Don't Care About Security. 2021. Online. Security Journey. <https://www.securityjourney.com/post/typescript-doesnt-suck-you-just-dont-care-about-security> Read 29.3.2023.

9 Microsoft TypeScript, the JavaScript we need or a solution looking for a problem. 2012. Online. ArsTechnica. <https://arstechnica.com/information-technology/2012/10/microsoft-typescript-the-javascript-we-need-or-a-solution-looking-for-a-problem/> Read 20.3.2023

10 FizzBuzz. 2022. Online. Zenware <https://github.com/zenware/FizzBuzz> Read 16.4.2023

11 Migrating from JavaScript. 2023. Online. Microsoft. <https://www.typescriptlang.org/docs/handbook/migrating-from-javascript.html> Read 20.3.2023.

TypeScript Interviews.

The following interviews have been translated and paraphrased to maintain a shorter length and anonymity.

Q1. How much Experience do you have with TypeScript?

A1. A couple of projects, some of them, however, are pretty old and TypeScript was in a rather primitive state at the time.

A2. 3 years.

A3. 1 year in professional and private capacity.

Q2. Do you prefer using TypeScript or JavaScript, and what would you personally use either language for?

A1. For a solo project JS, and for a larger team effort TS. Partially because TS is a fresher programming language and people usually prefer to use it over JS. Like choosing React over Angular because people are more familiar with it.

A2. All projects that are done with a team, or into a larger codebase, TypeScript. But for simpler software I prefer JS.

A3. TypeScript whenever possible, helps me in my professional life when I also work with it for private projects.

Q3. What are to you the biggest benefits and disadvantages when using TypeScript?

A1. My personal programming style works better with JavaScript as I’m more used to it. When the code is in an early stage, TypeScript doesn’t let you play around with it as much. My programming style is very iterative. Even though most of the type declarations happen on the fly but just a few trickier type

declarations can take a large amount of time. Usually in TypeScript you get stuck on small things.

Compile time error checking is useful and it’s nice that IDE directly relays errors in the program code.

TypeScript functions are when compared to for example Python very large and complicated, as an example when you’re feeding the result of a function to another function the typings can get really complicated.

TypeScript really shows its value best in larger projects with a team that has varying degrees of skill and experience.

A2. Errors and faulty logic get caught very quickly, but using packages from yarn and npm can get tricky if they don’t have included typings.

A3. The code structure feels simpler and it’s easier to write your code in a way that makes sense and looks legible.

Sometimes when looking for answers for a bug it’s easier to look up the problem in JavaScript code instead of TypeScript, as there’s a lot more results and fixes that have cropped up over the years.