UNIVERSITY OF TARTU
Faculty of Science and Technology
Institute of Computer Science
Software Engineering Curriculum

Kirill Maksimov

A Qualitative Case Study on Using TypeScript as a JavaScript Alternative in Frontend Web Development in the Industry

Master's Thesis (30 ECTS)

Supervisor(s): Ishaya Peni Gambo, PhD

Tartu 2022

Author's declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Kirill Maksimov
Date: August 03, 2022

A Qualitative Case Study on Using TypeScript as a JavaScript Alternative in Frontend Web Development in the Industry

Abstract:
JavaScript was initially created to make web pages interactive. No one knew at that time that Frontend could be as huge as it is nowadays. To minimize the shortcomings of the JavaScript language, TypeScript was devised as an alternative. In this thesis we define the boundaries of the rational use of both TypeScript and JavaScript. The research aims to compare the two most commonly used programming languages on the Frontend side: JavaScript and TypeScript. The case study approach was used to gather and analyse the information. A combination of qualitative and quantitative methods was chosen for the current study. The findings of the current study provide both empirical and theoretical implications. For this work the survey was made to understand the opinion of experienced software developers concerning the language choice on the Frontend side. The information taken from the respondents is based on their JavaScript and TypeScript experience. Data collected from the survey is processed and analysed, and results are presented in a format of charts and graphs. Moreover, two application prototypes were created for the study purposes. Results showed that the difference between using TypeScript and JavaScript is not significant for the MVP. But if the MVP scales in the future, TypeScript will be the right choice. TypeScript was designed to build and maintain large-scale applications. But each tool has its limitations and TypeScript is not suitable for every case, for instance, writing extra lines of code, time consuming initial setup and issues with some third-party libraries add complexity to the project. As a result, it can be noted that TypeScript is not a panacea and it cannot be used everywhere.

Keywords: TypeScript, JavaScript, Frontend, TypeScript Compiler, TypeChecker

CERCS: P170 Computer science, numerical analysis, systems, control

Kvalitatiivne juhtumiuuring TypeScripti kasutamise kohta JavaScripti alternatiivina veebiarenduses

Annotatsioon:
JavaScript loodi algselt veebilehtede interaktiivseks muutmiseks. Sel ajal ei teadnud keegi, et Frontend võib olla nii suur, kui ta tänapäeval on. Et minimeerida puudusi JavaScripti keeles, töötati alternatiivina välja TypeScript. Selle teesi raames defineerime mõlema keele ratsionaalse kasutamise piirid. Samuti määrame ja analüüsime valdkonna arenda- jatelt kogutud andmed. Informatsioon, mis on saadud uuringu osajatelt, põhineb nende JavaScripti ja TypeScripti kasutamise kogemustel. Uuringu eesmärk on võrrelda kahte kõige sagedamini kasutatavat programmeerimiskeelt - JavaScripti ja TypeScripti. Teabe kogumiseks ja analüüsimiseks kasutati juhtuuringu meetodit, kus valiti kvalitatiivsete ja kvantitatiivsete meetodite kombinatsioon. Küsitluse eesmärk on mõista kogenud tarkvaraarendajate arvamusi Frontendi keele valikul. Küsitlusest saadud tulemusi töödeldakse ja analüüsitakse, ning tulemused esitatakse graafikute ja diagrammidega. Loodi ja analüüsiti kahte MVP-d. Tulemused näitasid, et TypeScripti ja JavaScripti kasutamise vahe ei ole MVP-de puhul oluline. Kui aga MVP-d tulevikus laienevad, siis TypeScripti kasutusele võtt on parem valik. TypeScript loodi suuremahuliste rakenduste loomiseks ja hooldatavana hoidmiseks. Igal tööriistal on oma eripära ja ei sobi igaks juhtumiks – täiendavate koodiridade kirjutamine, algseadistus on keerulisem või mõne kolmanda osapoole teegiga seotud probleemid võivad aega võtta. Seetõttu pole TypeScript imerohi ja seda ei saa igal pool kasutada. Käesoleva uuringu tulemused annavad nii empiirilised kui ka teoreetilised järeldused.

Võtmesõnad: TypeScript, JavaScript, Frontend, TypeScripti Kompilaator, TypeChecker

CERCS: P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine

List of abbreviations and terms

JS	JavaScript
TS	TypeScript
FE	Frontend
BE	Backend
LoC	Lines of code
TSC	TypeScript compiler
AST	Abstract Syntax Tree
MVP	Minimum Viable Product
WEB	World Wide Web
OWFa	Open Web Foundation Specification Agreement
IDE	Integrated Development Environment
JSON	JavaScript Object Notation
SPA	Single Page Application
NPM	Node Package Manager
ES	ECMAScript
API	Application Programming Interface

Table of Contents

List of Figures

1. Introduction

Nowadays not only JavaScript is used to create interactive web pages. Among top tier programmers, plain JavaScript is often not really popular due to weak typization and bugs that were made during the designing and developing of this programming language. The purpose of this thesis is to find out about the rational use cases of both JavaScript and TypeScript and identify usage aspects of these languages, such as the maintainability and scalability. Also one of the aims is to determine what are the limitations of JavaScript and TypeScript.

1.1 Background

This section provides background information about JavaScript and TypeScript.

1.1.1 JavaScript Fundamentals

JavaScript is probably one of the most important web programming languages today. Due to the development of the Internet, this programming language has reached heights that it was not expected. It is used by 71.5% of professional developers making it the world's most widely used programming language [1]. Primarily JavaScript was designed as a "little language" [2], because at that time the Internet did not have a suitable scripting language that could be used in the browsers. The language was needed for specific goals, it had to become easy and understandable for people far from programming, "glue programmers" [1]. Brendan Eich, the founder of JavaScript and his team was given a task to provide a working prototype of a new programming language for the Netscape browser in a short time frame. As a result JavaScript borrowed features from different programming languages, for instance, C-based syntax form Java, first-class functions from AWK and Scheme, prototype inheritance from Self, arrays as well as string and regular expressions from Python and Perl [3]. As a result JavaScript on the early stages was missing the functionality that could be found in other programming languages - modules, block-scoped variables etc. Initially JavaScript was created as a programming language for implementing simple scripting but today the situation has changed - the frontend engineering has become such a huge part of the development that it is extracted into an independent sector for study and work.

JavaScript scripts can be written in HTML and run when the web page is loaded. Scripts do not need any preparation or compilation to be executed in the browser. Thus, JavaScript differs from compilable languages. Today JavaScript can be run not only in different browsers, but also on a server side or any other device that has a special program called a JavaScript "engine" installed. We won't go into details about non-browser runtimes as we will be focusing on frontend development in this study.

The browser has its own engine, sometimes referred to as the "JavaScript Virtual Machine". Different engines have different names. For instance:

- V8 - in Chrome and Opera.
- SpiderMonkey - in Firefox.
- JSCore - in Safari.
- Chakra - in Edge.

Engines are complex, but the general base is easy to understand. Their work can be divided into three main stages:

1. The built-in engine of the browser reads/parses the text of the script.
2. Converts/compiles the script into machine language.
3. Machine code starts up and runs.

Modern browsers can read just plain JavaScript. Also, sometimes it takes time for all the browsers to keep up with the latest language's specification.

From very beginning JavaScript was a "safe" programming language. It does not provide low-level memory or CPU access, as it was originally designed for browsers that do not require this feature. The capabilities of JavaScript are highly dependent on the environment in which it runs. For instance, in Node.JS environment JavaScript supports functions for reading/writing files, making network queries and more.

Everything related to the manipulation of web pages, user interaction and the web server is available nowadays with JavaScript. But in this work we will focus on the frontend side.

For instance, in a browser, JavaScript can:

- Add new HTML code to the page, change existing nodes, change the content of the node, switch between DOM elements, select the exact element, do manipulation with the DOM, change styles of different elements on the page.

- Interact with user actions such as clicks, mouse moves, key presses and others.
- Fetch network queries to servers' API, upload and download files (AJAX and COMET technologies).
- Show modal windows.
- Work with forms.
- Work with client data, store it in the local storage, session storage or in cookies.

JavaScript capabilities in the browser are limited for the safety of the user. The purpose is to prevent an unconscionable web page from accessing personal information or damaging user data.

1.1.2 TypeScript Fundamentals

TypeScript is a programming language founded in 2012 by Microsoft due to shortcomings of JavaScript in complex applications and it was positioned as an extended tool that supports the capabilities of JavaScript. The language specifications are open and published under the Open Web Foundation Specification Agreement - OWFa 1.0. Problems with the development of complex programs in JavaScript led to the need to facilitate the development with the new programming language [4].

TypeScript is backward compatible with JavaScript. Developers of TypeScript were searching for a solution that would not interfere or conflict with the JavaScript standard cross-platform functionality and compatibility. As a result, a TypeScript compiler with a set of syntactic sugar features was developed [5]. TypeScript converts to a JavaScript code. Some features that currently are in the latest versions of Vanilla JavaScript were initially created as TypeScript syntax sugar features. In this regard, TypeScript serves as an example of what the new ECMAScript language will be like.

TypeScript program can be executed in every program which supports JavaScript. For instance, in any modern browser or it can be used in conjunction with the Node.JS server platform. The code for the compiler that translates TypeScript to JavaScript is distributed under the Apache license. Its development is carried out in a public repository through the GitHub service.

TypeScript differs from JavaScript in its ability to explicitly assign static types, support for the use of full-fledged classes (as in traditional object-oriented languages), and support for plugging in modules, which are designed to increase development speed, facilitate readability, refactor and reuse code, and help to find errors in development on compilation stage and possibly speed up the execution of programs [6]. Static typing is not mandatory

in TypeScript, it is just an option, which gives a developer a choice. A lot of different IDE support this feature.

General TypeScript features that are commonly used [7]:

- Static typing.
- Generic types.
- Interfaces and abstract classes.
- Optional arguments for functions.
- Public, private, protect for methods of classes.
- TSC is able to compile TypeScript code to JavaScript ES5 or even ES3.

1.1.3 TypeScript as a JavaScript Alternative

JavaScript is not suitable for all cases; various projects require different programming languages and technologies. Nowadays, many new languages could be transpiled into JavaScript before they run in the browser. Modern tools can convert JavaScript alternatives' code very fast and transparent, effectively allowing developers to write code in another language, automatically converting it to JavaScript under the hood.

Examples of such programming languages:

- CoffeeScript adds syntactic sugar to JavaScript, it provides the possibility to use arrow functions and some other syntactic constructions. It introduces a shorter syntax that allows you to write clean and concise code. Previously it used JavaScript engine, namely Narwhal, then it switched to Node.JS due to the popularity of this engine. The philosophy of CoffeeScript is close to Ruby programming language - to write less code, which looks short and concise.
- Dart is completely different from CoffeeScript. It was originally proposed by Google as a replacement for JavaScript, but it did not become really popular at the frontend web development.
- TypeScript focuses on adding strong typing to make it easier to develop and maintain large and complex systems. It was developed by Microsoft. It is backward compatible with JavaScript.

TypeScript is a language for application-scale JavaScript. TypeScript adds optional types to JavaScript that support tools for large-scale JavaScript applications for any browser, for any host, on any OS. TypeScript compiles to readable, standards-based JavaScript [8]

1.1.4 TypeScript Compiler

Understanding of TypeScript compiler and the whole process of running TypeScript applications will help to clarify the questions considered in this work.

TypeScript compiler, also known as TSC, compiles or better to say, transpiles TypeScript code to JavaScript. This section is describing the steps from transpiling a TypeScript code to the last step, which is running the program in the browser. This thesis is focusing on frontend development, we will skip the Node.JS server side applications part, but the principles are remaining the same in both frontend and backend:

1. Parsing the TS code.
2. Transforming the TS code into abstract syntax tree.
3. Checking the type safety.
4. Converting the code to JS source code.
5. Running the application.

Let's dive a little deeper and explain all the steps. TSC is a command-line utility that parses the TypeScript code and transforms it into an abstract syntax tree (AST) [5]. Then the TypeScript AST is checked by TSC' TypeChecker that the code is type safe. Then TypeScript AST is converted into a JavaScript source code. And after these manipulations the code is run as the usual JavaScript application does.

TSC needs a JSON configuration file to search for .ts files and transpile them into JavaScript code.

First of all, the root-level options can be configured in the "compilerOptions" section of the configuration file. Different options could be chosen, for instance, for the output of the transpilation - ES3 (which is a default option), ES5, ES6 or ES2020 JavaScript specification code.

The root-level options are responsible for how the application is represented to TSC, for instance, what TypeScript files should be compiled. The compiler options also specifies where to output the JavaScript files in the project directory. Also among different settings of root-level options, files for compilation can be included and excluded.

Moreover TypeScript compiler can set a module system. Some of the environments that run JavaScript code are using different module systems. For instance, Node.JS is using CommonJS, some others use AMD, which stands for Asynchronous Module Definition. It

is rather widespread in web to get JavaScript modules asynchronously when needed. TSC is converting "import" statements into the required format [5].

1.2 Motivation and Problem Statement

The motivation for this thesis is mainly based on the goal of increasing the understanding of the issue of choosing an appropriate frontend programming language for certain projects. At first glance, the question seems to be well studied and there are many articles on the Internet about the advantages or disadvantages of programming languages. But in practice this question comes up very often when there is a need to choose between the different programming languages on the frontend side of the application. As a result, companies sometimes spend time and resources studying an issue that can be resolved much faster than is usually the case. Not many valuable articles are available on the web resources on the rationale usage of TypeScript as an alternative to JavaScript on the frontend side of the projects and prototypes. This study will allow us to delve deeper into this issue and may serve as a basis for future researches.

This thesis presents three research results on using JavaScript and its main alternative - TypeScript, for creating FE web applications. First, the author conducts a survey; the objective is to understand web developers' professional opinion about the usage of JavaScript and TypeScript in modern software development. It will help to collect the feedback from engineers about using these two technologies. The experience gained by different software engineers in the development process can be considered when understanding the requirements for TypeScript as a JavaScript alternative for MVPs and prototypes. Second, comparing JavaScript to TypeScript in terms of scalability. Third, review the limitations of both JavaScript and TypeScript.

This work will help not only to analyze the current situation in the industry, but it will also help many developers to understand why a certain technology is chosen for a specific case, what are the good reasons for choosing TypeScript as a JavaScript alternative, and why it is worth being guided when there is a programming language choice. The results will be analyzed by the author and presented in this thesis in the form of a review of JavaScript and TypeScript for software engineers.

1.3 Research Questions

There is no single correct answer to the question of choosing a programming language. It is hard to foresee the changing requirements of the project over time. But based on

the available data which was gathered from the survey, it is possible to determine the rationality of using different technologies. TypeScript is a kind of add-on over JavaScript with additional features and functionality that was created for complex scalable projects. Core JavaScript was created as a language that is easy to use and learn, its task was to create the most simple interactivity of web pages. When choosing a language, it is advisable to determine for which purposes this language is suitable for and for which is not. The boundary is not clear and is left to the discretion of each developer. However, the experience gained by software web developers can help to define these boundaries. The first research question is asking if the TypeScript is a rational choice for creating an MVP. The second research question is focusing to better understand the aspect of maintainability of TypeScript comparing to JavaScript. And the third research question is covering the aspect about limitations of TypeScript.

RQ1: Is it rational to use TypeScript over JavaScript for the MVP on the Frontend side?
RQ2: How TypeScript differs from JavaScript considering applications' scalability?
RQ3: What are the limitations of using TypeScript on the frontend side?

1.4 Research Goals

Each tool has its own specific area of appliance, a certain kind of task for which it was created. If we are talking about programming languages, then they are a kind of tool for creating programs, in our case, we consider the client part of the application as a program. JavaScript is good for the purpose for which it was created - to make web pages interactive. At the time of the creation of this language, no one could have thought that the frontend side of various applications could grow so much. TypeScript was created in order to minimize the shortcomings that JavaScript has. It was developed much later and already met many requirements that the original language did not have. The main one is strong typing. As we said earlier, every tool is good for a certain task. In this case, we have two instruments that are extremely similar to each other. Which means that the end product is the same - JavaScript code. But are they really similar? How strong are the disadvantages of JavaScript when building simple applications, prototypes or MVPs and huge scalable applications?

All these questions cannot be answered easily, but some recommendations can be made based on the opinion of engineers who have had experience with these two different languages. After analyzing gathered information from the interviews, it will be possible to define conclusions about the most rational usage of a particular technology. When developing a new project, the question always arises about the choice of technologies that will be used. And often developers do not have an exhaustive answer on what is better

to use in a particular case. In this case, materials in the form of a guide or instruction are often helpful. It can be helpful to create guidelines for determining the rationality of language use across different projects.

The purpose of the study is to provide software developers with information about TypeScript and JavaScript rational use cases in a clear and understandable format.

1.5 Contributions

This study presents the results of comparing TypeScript and JavaScript for some use cases. The information was gathered by conducting a survey. Also the samples of the application prototypes were created for the study purposes. To summarise, the contributions of this work can be divided into 3 main parts:

1. Practically, we created two prototype applications for static code analysis [1]. These frontend applications are MVPs which were built using different programming languages - TypeScript and JavaScript. The functionality of these applications is the same. The MVPs were analyzed using the Sonarqube static code analyzer. The obtained information was analyzed and presented in the current work.
2. Methodologically, both the static code analysis and the survey were carried out simultaneously in order to study the issue of choosing a programming language. The results from both methods were combined and taken into account when writing the conclusions of this work.
3. This study can be used as a manual for the software developers who face an issue of the programming language choice. Also, the analyzed results of the survey can be used by various researchers, including students of different specialties. They may expand this study in their future works.

1.6 Structure of the Thesis

The structure of this thesis is as follows: the purpose of the thesis is explained in Chapter 1 along with the Technical Background, Problem Statement, Research Questions and Contributions. Moreover, in the first Chapter of this thesis the information about the author's motivation could be found as well as the historical background and the foundations of the JavaScript programming language and the prerequisites of the creation of the TypeScript programming language.

[1] https://gitlab.com/maximov.epam/group-mates

Literature review on using TypeScript as a JavaScript alternative is presented in Chapter 2. Also in this chapter the aspects of using the TypeScript as a scalable programming language can be found. Moreover the limitations of both JavaScript and TypeScript were mentioned during the review of the literature. Furthermore some peculiarities about the developing of MVPs can be found in the Chapter 2.

Third Chapter is describing the methodology of this work. Also it covers the overview of the demographics of the participants of the survey, overview of the questions which were asked during the survey and also it describes the applications prototypes that were created for the study purposes based on the use of JavaScript and TypeScript.

Results of the survey are presented in the format of diagrams and charts in the Chapter 4. Also, the analysis of the presented results could be found in this Chapter.

The fifth Chapter is the Discussion. The exploration of the results can be found in this Chapter. Also the author provides the list of limitations of the current study.

The conclusion can be found in the sixth Chapter. The author summarizes the results of the study and outlines future work.

2. Literature Review

This section covers literature review on using TypeScript as a JavaScript alternative in the frontend side. This section is divided into three parts. First part in this section is covering the questions of using TypeScript for creating the application prototypes. This part includes the analysis of the other researches regarding the code quality of both TypeScript and JavaScript. The second part investigates the other researches that has an information about TypeScript scalability. And the last part of this section is focusing on the works which covers the limitations of both TypeScript and JavaScript.

2.1 Use of TypeScript and JavaScript for creating MVP

MVP is a version of a new product, which allows a team to collect the maximum amount of validated learning about customers with the least effort [9]. The research showed that there were no direct comparison of JavaScript to TypeScript in terms of developing the MVP or prototype. But some works raise the question of TypeScript use and how it increases the code quality of the projects. The github open access projects were analysed [10], [11]. These works showed that TypeScript is a better choice for the large-scale applications and it can help to decrease the amount of potential bugs in the application.

2.2 Scaling Frontend Applications

Different articles, papers and studies present the researches which describe scalability of different projects with the using of JavaScript and TypeScript. Some of the materials are focusing on the limitations of the languages, for instance, papers [12] and [13]. Moreover, a lot of materials are covering the optimization aspects [14]. Other researches compare the scalabitity capabilities of JavaScript to TypeScript [11]. In this section we cover all of the scalability aspects of both TypeScript and JavaScript.

2.2.1 Specifics of Frontend Applications' Scalability

Different definitions of scaling programs were found during the literature review, all of them can be divided into two groups - first is talking about increasing of the number of the system's elements and the second says about handling the raising workload without

adding the new elements.

Among the first group of definitions the most exhaustive represents scalability as an ability of a system to accommodate an increasing number of elements or objects, to process growing volumes of work gracefully, and/or to be susceptible to enlargement [15].

And when we describe the scalability considering handling the raising workload, the most comprehensive definition is that scalability is the ability to handle increased workload without adding resources to a system [16].

Frontend applications have some scalability specifics - usually they do not face a high workload. User should not wait long for the processed data and the best practice is to process operations on the backend side. Heavy calculations and the work with data processing usually bases on the backend side.

But when we talk about the frontend applications - how can they scale if usually there is no huge data workload? Most of the time when developers discuss the frontend scalability they are talking about the adding of the new functionality or implementing the new features to the existing application, so we can say - adding a new elements to the system. A lot of aspects should be noticed while adding a new feature - architectural structure, understandability of the code, consistency and many others.

Some books offers the guides and instructions how to write the proper frontend application that can be extended with the different functionality in the future easily [17]. But all the books that describe the scalability of TypeScript mention that with the using of TypeScript programming language the extension of the current functionality of the application is much easier than scale applications that are based on JavaScript [18]. Further these aspects of JavaScript and TypeScript programming languages will be covered.

Frontend applications' architecture is the core that defines how easy is this program to scale. Complex tools like different frontend frameworks were created to solve the problem of architecturing the applications' structure. Scale in frontend is not a traditional scaling issue in computing, when the system has to be able to work with a higher workload. Scaling on the frontend side is completely different to backend side, because the only hardware that handles everything on the client side is client's hardware.

The size of the client side code is a significant aspect to bear in mind when deal with the frontend application. It reflects the time that user is waiting for starting to interact with the page.

To summarize, some specific aspects of the frontend applications' scalability could be mentioned:

1. Frontend applications are build for various users, so the requirements are different: clients use applications on the different machines with a different browsers.
2. Frontend applications always scale in terms of new features. Architectural aspect is a base that is significant in this case. Furthermore, with the increasing number of new implemented features, the complexity of these features are raising as well, which leads to the new code and as a result to scaling the application.
3. It is crucial to bear in mind that the loading time and the size of the bundle is always important on the frontend side. Moreover the user data that is loaded for the application should also be optimized.
4. When all the tools are added to the project for development and production purposes the application should remain configurable for maintaining and further development. Different configuration files adds complexity to the overall code base. When frontend application scales, configurations of the program should be understandable.

2.2.2 TypeScript and JavaScript Scalability Comparison

TypeScript and JavaScript differs just with the type checking mechanism, so the scaling considering the workload is the same. Because in the end the JavaScript code will be run in the browser or Node.JS environment. On the other hand TypeScript gives an ability to write the good self-documented code that increases the maintainability of the applications. Interfaces and types can be used to understand the complex data structures. Also "tsconfig.json" helps to configure the projects. As it was mentioned before - TypeScript is a superset of features on JavaScript programming language and the main goal of creating this language was to help the large JavaScript applications scale.

When the frontend applications are scaling - the number of unit tests are increasing. With the TypeChecker which is used by TypeScript compiler some unit tests could be removed form the project. There is no need to check the type of the data provided, because these errors will occur on the compilation step.

Considering the architectural aspect - both TypeScript and JavaScript frontend applications are using frameworks and libraries usually to develop complex large-scale applications. The components' structure is organised by the framework or the library often. Still there is a flexibility to organise the structure of your Single Page Applications or the traditional ones. If we talk about the server side rendering application based apps, for instance, on Next.JS there is a slightly possibility to change the code's structure.

2.3 Limitations of TypeScript Use on the Frontend Side

This section is covering the question of the limitations of the TypeScript programming language. The focus is made on the frontend use of the TS. Generally the TypeScript limitations could be divided into three main groups:

1. Optional type checking.
2. Different approach of writing the TS code.
3. Issues with the third-party libraries.
4. Additional configurations.
5. Additional lines of code.

TypeScript does depend on the team of developers who are writing the code. The Type-Checking is optional and plain JavaScript code is a valid code in TS projects. Usually before starting the project developers make the agreement about what should be covered by types in .ts files [19]. The manner of writing code is quite different from one project to another because sometimes software developers do not follow the best practices and they can skip defining the types or start to define the types everywhere where it is just possible to do, which is considered not the best practice.

Furthermore, software developers who are new to TypeScript sometimes start to disable the TypeScript checks for some exact lines of code or even disable the whole file in order to "fix" the issue and in some cases developers just can change the TypeScript configuration file to get rid of the warning or error that is highlighted by TypeScript. All of the approaches that are mentioned above are considered as a bad practices and should not been followed during the development process.

Some sources [20] represents the use of third-party libraries as an issue in some cases. For instance, if there are no types package for the TypeScript project. It takes a long time to fix some others code in the external library. The main goal of the external library is to decrease the time spent on the feature development but with TypeScript sometimes the initial implementation and integration of third-libraries takes a longer time.

Additional configurations should be made in the project that is using TypeScript [21]. Also, the configuration of some tools become more complicated in case of TypeScript use, for instance, new extensions must be provided to the Webpack bundler because it needs to go thorough all of the files in the project.

The number of lines of code when writing two applications with the same functionality

on JavaScript and on TypeScript will be different. TypeScript doesn't require creating the types and interfaces, but without defining the types of the variables and, for instance, the arguments for the functions, it doesn't make sense to include into the application complicated additional configurations. So, usually TypeScript applications have more lines of code. This code is definitely one of the TypeScript advantages because the code become "self-documented" and these TypeScript features can help to understand complex data structures and implement new features. But on the other hand it will take time initially to write all these additional lines of code.

3. Methodology

The case study method was chosen to answer the declared Research Questions. A combination of qualitative and quantitative methods were used for the current study. First of all, the survey was conducted with the software developers with different background and various programming levels. Respondents who were participating in the interviews, are working in different companies. The interviews include a variety of questions for in-depth study of the questions raised.

The survey includes interviews with 11 web software developers. Questions made for the interview aimed to understand the opinion of software developers and their concerns about TypeScript as a JavaScript alternative on the Frontend side.

Questions were made for developers of various levels and they were repeated for all the respondents who were interviewed to identify general trends. To extract the developers' concerns regarding the posed Research Questions, the author conducted semi-structured interviews with the specific set of questions. Most of the questions are open. The author of this study provided for the respondents the ability to share their opinion about questions posed.

The questionnaire can be seen in the appendix part of the work. The answers of the respondents were analyzed by the author and are presented in the current study.

The interviews took place online via various communication platforms. The questions are divided into 4 separate groups: The first group of questions presents introductory questions and demographics questions. Their purpose is to prepare the interviewed developer for a dialogue and understand the overall experience in programming field and the background of the respondent. The second group contains the questions about the requirements to the programming language that is used on the Frontend side of the application. The third group of questions, namely Advantages, clarifies the strengths of both TypeScript and JavaScript programming languages. It determines whether the developer had a choice of technologies and programming languages when working with the project. The fourth group defines the aspects of the TypeScript language which can be considered as its weaknesses. This group also compares disadvantages of JavaScript and TypeScript. More detailed overview of the

questions from the conducted survey is presented below.

There were several requirements for the potential respondents of this survey:

1. Experience in the field as a Frontend Developer or the Full-Stack Developer.
2. Experience with either JavaScript or TypeScript programming language.
3. At least one year of work as a professional Software Developer.

Also two applications were written with the same functionality but based on different programming languages - TypeScript and JavaScript. These applications were written for the study purposes to understand better the difference between the use of the considered programming languages for the purposes of writing the applications' protptypes and MVPs. The static code analysis was made to find out which programming language suits better for the protptypes and MVPs. Moreover the survey contains the questions regarding the developers' opinions about the use of TypeScript for the small prototypes or for the MVPs. The results are represented in the "Analysis and results section" of the current work.

3.1 Overview of the Participants of the Survey

The interviews took place from the 1st of March 2022 to the 30st of April 2022. During this time eleven interviews were taken. Only developers with experience in frontend web development or Full-Stack web development were interviewed. The emphasis was made on the experienced developers, who were engaged in frontend development. After the interviews were done, all participants were divided into three groups:

1. Senior group [1] [2] [3] [4] [5]. Very experienced developers who have used TypeScript in different projects, who also have experience in creating projects with the use of TypeScript from scratch, who know the aspects and details of this programming language. Years of experience range is from 4 to 15 years. Average years of experience are 9,6. There are very different titles of the respondents in this group - Architect, Team Lead, Full-Stack Senior Developers and Frontend Senior Developer. Average TypeScript experience for the group is 3,6 years.

[1] https://youtu.be/CKxl1roaiC0
[2] https://youtu.be/pYal3nwKuF8
[3] https://youtu.be/KLImFVpqX_s
[4] https://youtu.be/G5qtdzUVycA
[5] https://youtu.be/4UfWpcs2N6Y

2. Middle group [6] [7] [8] [9]. Respondents in this group have one or two projects with the use of TypeScript. Software developers can also have an experience of starting a project from scratch but not more than once. Years of experience vary from 1 to 5 years. Average years of experience are 2,5. Average TypeScript experience for the group is 1,8 years.
3. Junior group [10] [11]. This group included developers who do not have experience in TypeScript in production, but have some frontend experience and have used JavaScript before. Both Juniors have 2 years of experience in Web Development but they didn't use TypeScript before.

The interviews that are hosted on the Youtube platform and they can be seen just by links provided above. Interviews do not have the open access and they can not be found by search in Youtube.

As a result, the interviews were taken from five Senior developers, four Middles and two Juniors. Representatives of 9 different companies were interviewed - Finnair, Paf, Itransition Group, EPAM Systems, Pipedrive, Digimevo, Bondora, Paxful and R8Tech. Companies vary in size from 19 to about 30,000 employees. Companies are both IT oriented and non-IT companies. Main offices of these companies are located in 5 different countries.

The interviews were conducted for the study purposes. All the respondents gave a permission to process the data gathered during the interviews for the study purposes.

3.2 Overview of Questions

This section represents the questions overview from the questionnaire. All the questions can be divided into 4 groups:

1. Demographics. This group mostly includes questions about interviewees overall experience, their experience exactly with TypeScript and some information about their current positions and responsibilities.
2. Requirements. Questions about frontend language requirements were asked. The main focus was on TypeScript and JavaScript.

[6] https://youtu.be/VT3UVXFolgc
[7] https://youtu.be/pKf5FwdrVxI
[8] https://youtu.be/JAuhteKkD1A
[9] https://youtu.be/-gcBpWtYG2M
[10] https://youtu.be/VCI00AjJGD8
[11] https://youtu.be/nI9tPdNF4RM

3. Advantages. The benefits of the TypeScript language were discussed. Sometimes a comparison with JavaScript has been made.
4. Limitations. In this section the interviewees provided the information about TypeScript's disadvantages.

3.3 TypeScript and JavaScript MVPs

Two simple frontend prototype applications were created to answer the question about rationality of the use of TypeScript for the MVPs. These are the Single Page Applications (SPA) created based on React and Redux. There is no backend in the application. The data is stored in the LocalStorage of the browser with the help of third-party library, namely Redux-persist. After refreshing the page, the data can still be seen. The application is not production ready solution, it is just a prototype made for the study purposes, sensitive data will not be stored in the LocalStorage. The code can be found in the open access remote repository [12].

The "README.md" file contains the information about:

1. Project's description
2. Technologies used in the projects
3. Manual for launching the project
4. Overview of the main functionality
5. How to use the project

The main functionality of these applications include:

1. Logging in with the default administrator credentials
2. Logging in with the added user credentials
3. Logging out
4. Adding a new university member
5. Deleting a university member
6. Validating the fields
7. Presenting the data in the table
8. Checking the session time
9. Showing the modal when session is going to expire
10. Automatically exiting the current session in case of no activity
11. Continuing the current session from a modal

[12] https://gitlab.com/maximov.epam/group-mates

12. Exiting the current session from a modal
13. Routing

The simple login/logout functionality was implemented by adding a hard coded initial administrator credentials on the frontend. The author reminds that the application is made just for the study purposes. After logging in the administrator can add new university members. The collected data can be seen in the main table on the right side of the screen. Members can be deleted by clicking the delete icon in the row of the member.

After adding a new Group member, he/she is able to log in with his/her username and password. After deleting the Group member he/she is not able to log in to the system anymore.

If the current session has no activity for two minutes, the pop up is arising on the screen. It has two options - keep current session alive or log out. In case there is a pop up on the screen and still there is no user's activity the application will log out automatically. Also the user is able to log out manually by clicking the button at the top right corner of the screen.

The functionality of the application was developed with two different programming languages for the purpose of their analysis and research. Technologies that were used:

1. ReactJS
2. Redux
3. Redux-Saga
4. React-router-dom
5. Redux-persist
6. React final form
7. MomentJS
8. Material-UI
9. Lodash
10. Prettier
11. Sonarqube

A code static analysis of two applications with similar functionality was carried out. The tool used for the code analysis is Sonarqube. It was chosen as a common solution for the code static analysis of the applications that are written in different languages. Sonarqube supports 17 different languages including JavaScript and TypeScript. One of applications was written in TypeScript and the second one with the use of JavaScript.

The general code static analysis was taken into account. The number of potential bugs, the number of "code smells" cases, the number of vulnerabilities and the total number of LoC were analysed.

4. Analysis and Result

This section represents the results of the study, which were extracted from the collected data. We also discuss and compare the results gained from the survey and static code analysis of the created MVPs. The results are presented in the form of diagrams, graphs and pictures with processed information. Results of the Survey section contains data grouped by question category. The interviews were conducted remotely, and the results were summarized after collecting the required amount of data.

4.1 Code Static Analysis Results

To answer the Research Question 1, two MVPs were created using both TypeScript and JavaScript programming language [1]. Static code analysis showed that the difference between MVPs using TypeScript and JavaScript programming languages is insignificant.

Figure 1 represents the results of static analysis of JavaScript code for the written MVP. The analysis showed that only one potential error was found. Number of vulnerabilities - 0, in this section the project was rated "A". Code smells rate is "A" and Sonarqube found only 8 cases. Total lines of code are 603.

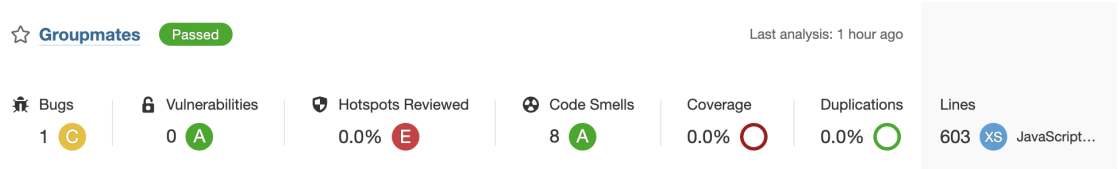

Figure 1. General static code analysis overview made with Sonarqube for JavaScript MVP

An MVP written using TypeScript was also analyzed and the results are shown on the Figure 2. The results are very close to the output that was received for the code static analysis of the application written with JavaScript.

The results of static code analysis between an MVP written in the JavaScript programming language and an MVP using TypeScript are slightly different and this difference is not really crucial. Due to the small amount of code, the statistics show approximately the same

[1] https://gitlab.com/maximov.epam/group-mates

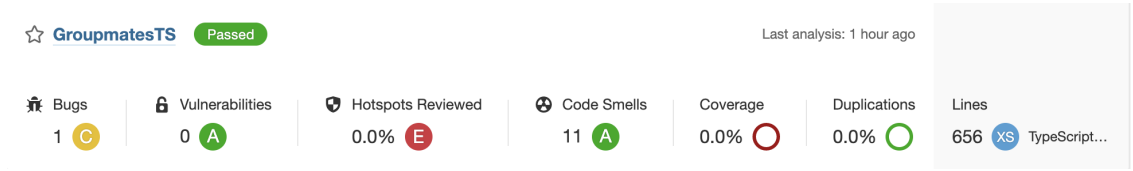

Figure 2. General static code analysis overview made with Sonarqube for TypeScript MVP

results. Moreover, the potential error is the same in both projects. This is because the underlying logic is similar in both applications.

TypeScript definitely makes a project more complex when setting up the initial environment. But as it was mentioned before it can be simplified in a different ways - for instance, a ready-to-use template can be chosen for the new application. The second point that is worth to mention - usually the initial set up is done by skilled person in the company, because the initial base of the project is quite important.

The Research Question 1 was asking if it is worth to use TypeScript for the application prototypes - considering all of the above facts, it can be concluded that if you need to prototype an application, there is no need to use exactly TypeScript, but if this application scales, TypeScript will be a good basis for implementing new features and scaling the application.

4.2 Results of the Survey

This section represents the analysed results gathered from the respondents. The further analysis is based on that data.

All the questions from the questionnaire can be divided into 4 groups:

1. Demographics
2. Requirements
3. Advantages
4. Limitations

Each group contains the questions which were formulated by the author of this work. A certain group of questions is aimed to clarify a specific research question of this study, except the Demographics part, which is used to understand better the background of the Respondents of this survey.

The goal of the survey is to gather the information from the highly qualified software

developers. This information is needed to provide the answers to the questions which are declared in this research.

4.2.1 Demographics

The Demographics section represents the results regarding the overall experience of the Respondents. Moreover, it shows their experience with TypeScript specifically. After analyzing the demographics results all the respondents were divided into 3 groups: Seniors, Middles and Juniors. Definitions of each group could be found at "Overview of the participants of the survey" section.

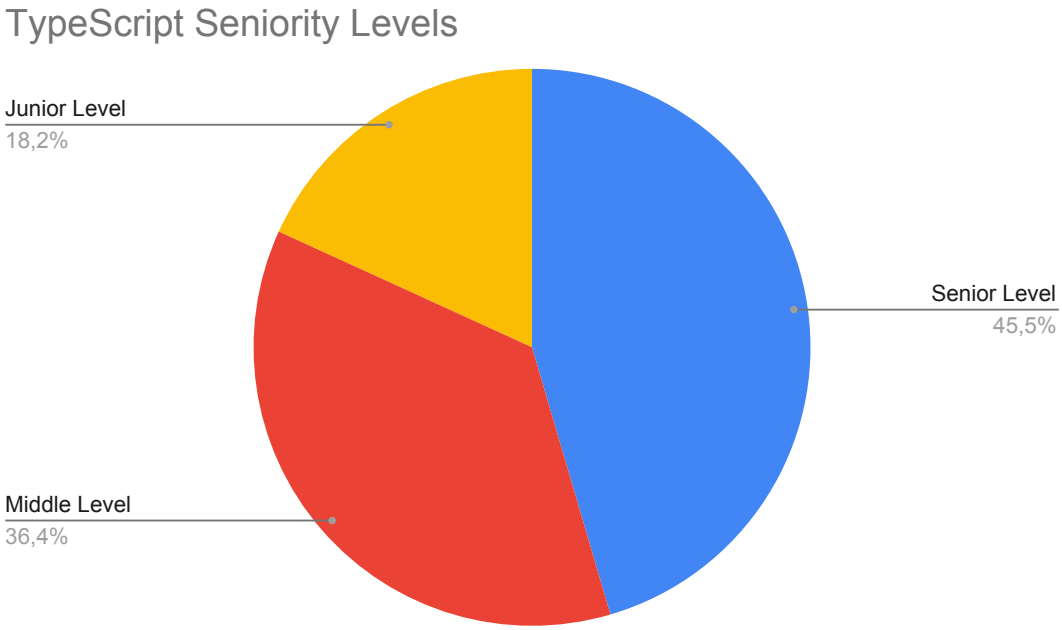

Figure 3. Respondents divided by Seniority Groups

Figure 3 represents the TypeScript Seniority Levels diagram. It presents the respondents divided by their work experience and experience in TypeScript. The explanation of the Seniority Levels can be found in the "Overview of the Participants" section of the current study.

Figure 4 shows the differences between the respondents in years of experience in software development. Two respondents have significantly more experience than the others. The maximum rate is 15 years and the lowest rate is 1 year. The average experience of software developers from the senior level group is 9.6 years, from the intermediate level group - 2.5 years, from the junior level group - 2 years. The total work experience of all of the respondents is 5.6 years.

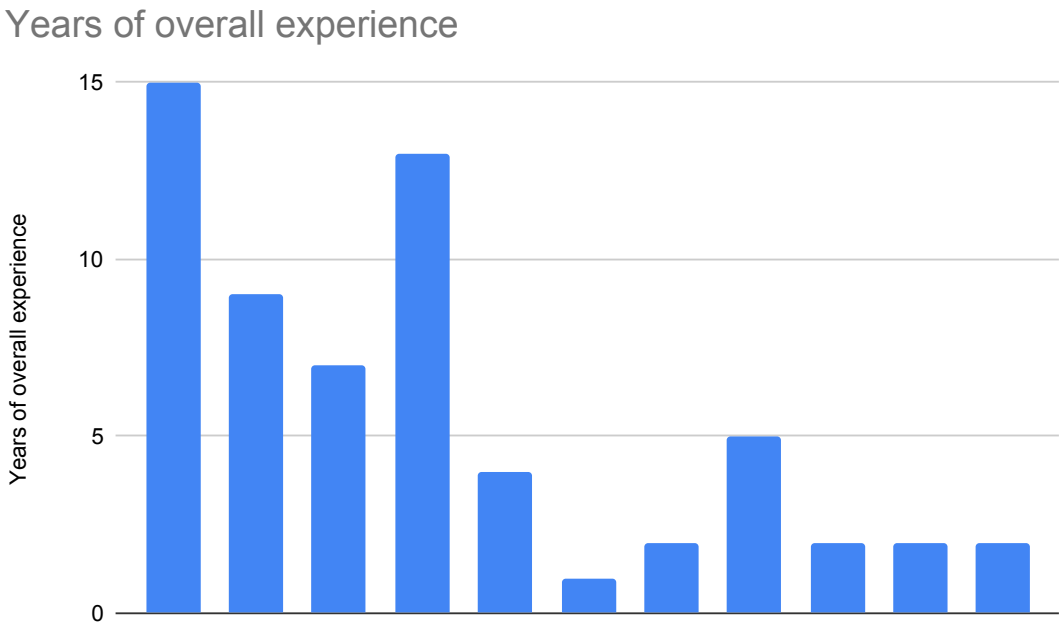

Figure 4. Overall experience of respondents by years

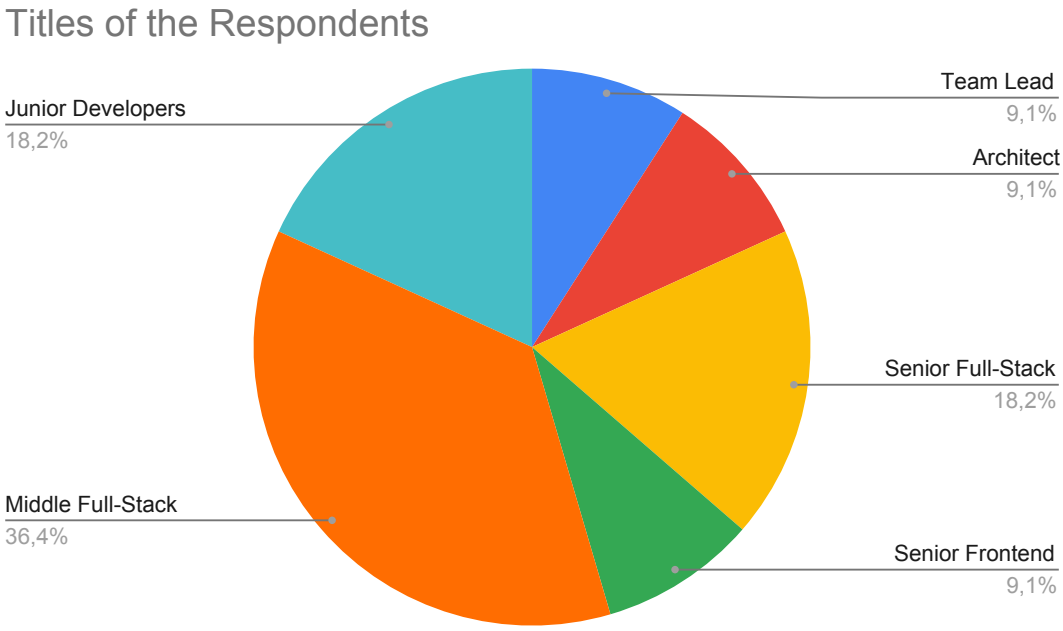

Figure 5. Titles of the Respondents in the companies where they are employed

Figure 5 represents the Titles of the respondents. It can be noted that the focus was made on experienced Software Developers while conducting a survey. Most of the respondents are in either high technical positions or even management positions in the companies they work for.

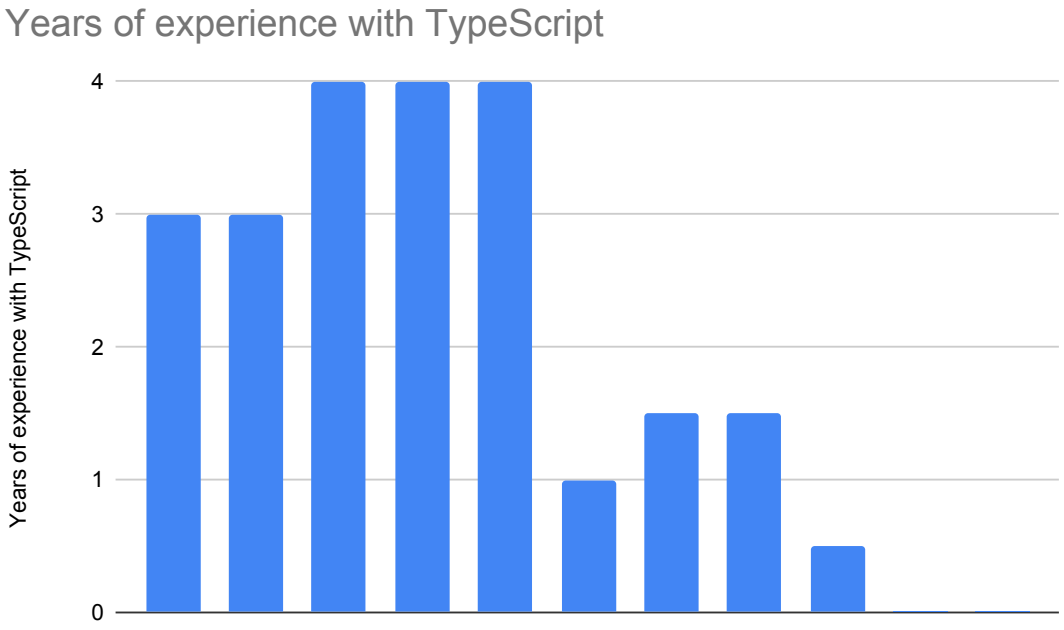

Figure 6. Overall TypeScript experience of respondents by years

Figure 6 represents the experience of the respondents in production of the applications with the TypeScript programming language. The graph clearly shows why the respondents were divided into 3 groups according to the experience of using TypeScript. Two respondents had no or little experience with TypeScript and were assigned to the Junior Level Group. Four respondents with less than two years of work experience are in the Middle level group, and another five were selected in the Senior level group. The average work experience of Senior Developers is 3.6, Middle Developers is 1.125 and Juniors is 0 years. Total experience with TypeScript in production is 2 years.

Figure 7 shows the number of projects of the respondents in the TypeScript programming language. 9 respondents out of 11 had at least 1 TypeScript project. One respondent, who is a Lead Software Developer, was involved as a participant to 30 different TypeScript projects. It should be noted that every respondent has an experience with either JavaScript or TypeScript programming language.

The applications that are shown on the Figure 7 did not include "pet" projects or the projects which were made for the self-education purposes. Just the applications that were professionally developed in a team were considered in this statistics. The average number

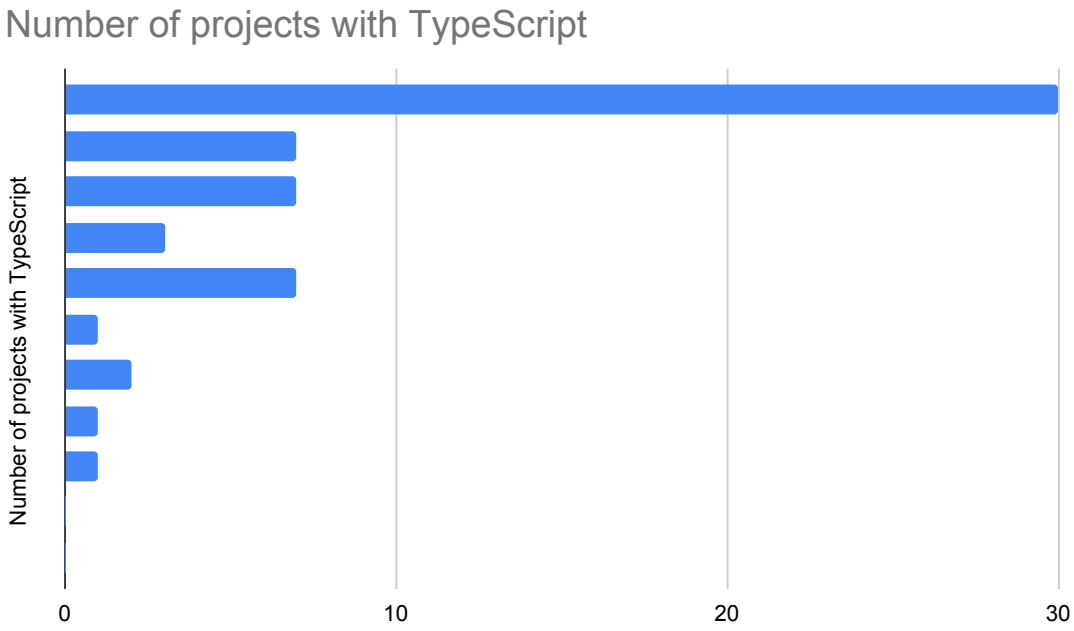

Figure 7. Number of TypeScript projects

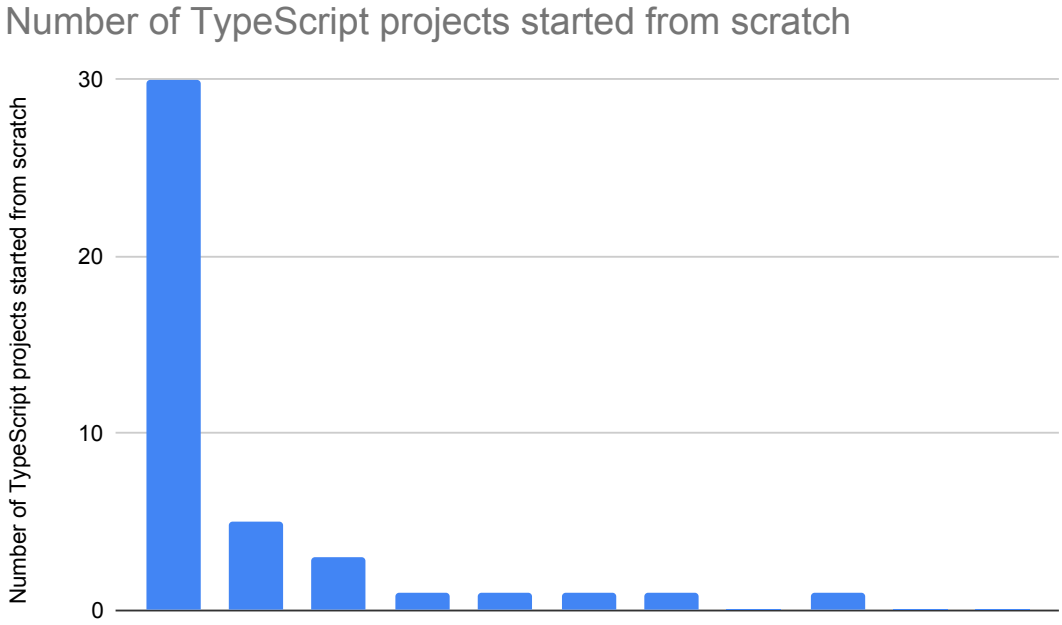

Figure 8. Number of TypeScript projects started from scratch

of the projects are 5.3.

Figure 8 represents the number of the projects which were started by the Respondents from scratch in production with the use of TypeScript. The diagram shows that one of the respondents who is the Team Lead in the Finnair company participated in the biggest number of projects from the very beginning. 3 respondents out of 11 don't have such an experience and 5 others have started from scratch just 1 TypeScript project.

4.2.2 Requirements

Requirements section contains the answers of the software developers regarding choosing programming language for frontend application. Engineers provided their reasons, thoughts and opinions about using JavaScript and TypeScript for the different use cases :

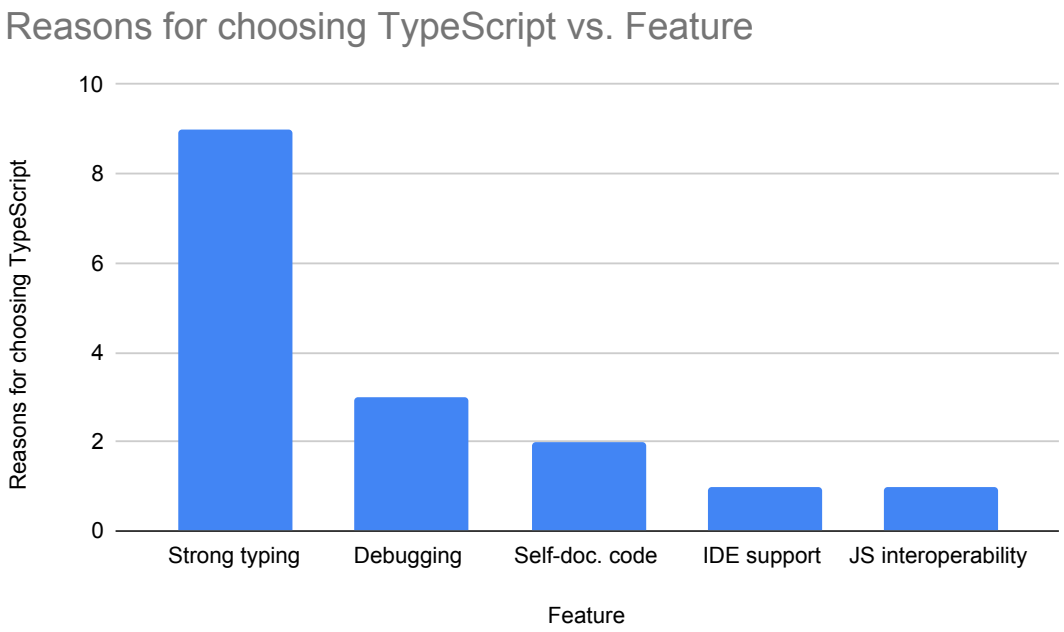

Figure 9. Reasons for choosing TypeScript

Figure 9 shows the reasons for choosing TypeScript in the application on the frontend side. It can be seen that the main reason for choosing TypeScript as a main programming language for the frontend applications is strong typing - 9 Software Developers mentioned it as a reason to choose TS instead of JS. 3 engineers mentioned debugging as a crucial feature. And 1 vote was for "self-documented" code, IDE support and interoperability with JavaScript.

Figure 10 presents preference of the programming language which they the respondents will choose in case of starting the new MVP or prototype.

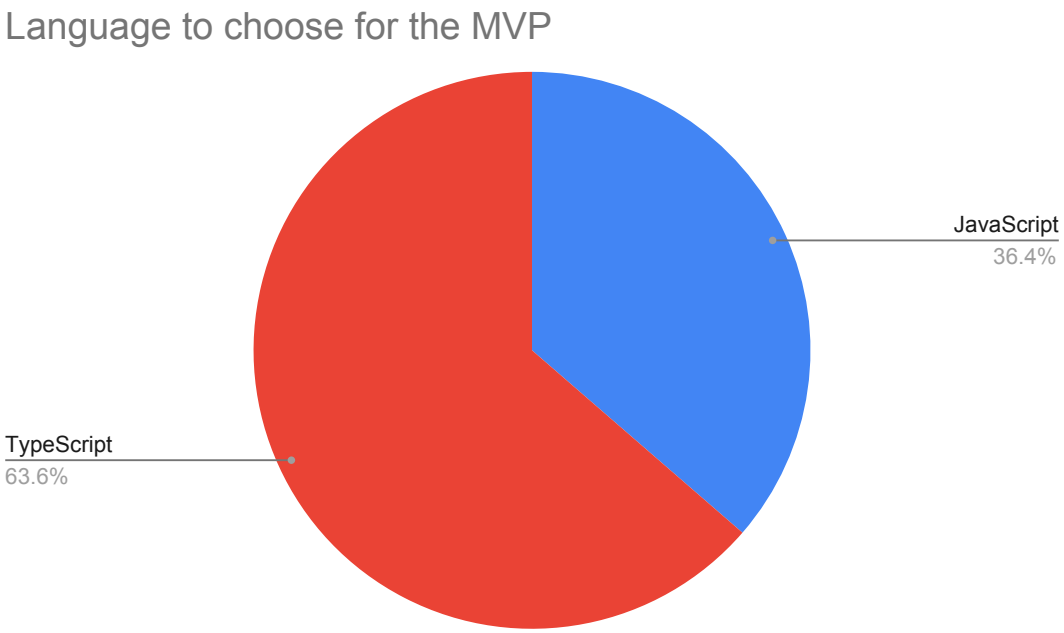

Figure 10. Preference on the programming language for the MVP

It is needed to note that 2 Junior Level developers chose JavaScript just for a reason that they had experience with it before and they didn't have a previous experience with TypeScript.

Half of the Middle level developers chose TypeScript and another 50% gave their preference to JavaScript.

4 out of 5 Senior level developers said that they will choose TypeScript for any project they start, no matter what it will be. Even for MVP or for a prototype TypeScript should be chosen in their opinion. 1 Senior level developer said that in his current company there is a strict rule regarding language choice and a small services should be done with the using of JavaScript. In all other cases developers should choose TypeScript. It is an interesting fact - he also added that after some time he noticed that JavaScript is not chosen anymore for new small projects - developers prefer to be consistent and they choose TypeScript in every case. To summarise - all the software developers from the Senior group chose TypeScript.

About the difference between TypeScript and the JavaScript community, all of the respondents who had experience with TypeScript responded that the JavaScript and TypeScript communities are almost the same, and there is a lot of information on the Internet about interesting TypeScript and JavaScript solutions that can be useful for the developers. Respondents with no experience with TypeScript did not provide their answers. Diagram is nor provided for this question from the survey.

About how hard is it to find the TypeScript Software Developer in the market 4 respondents (Senior) and 1 respondent (Middle) said that finding a TypeScript Developer is definitely difficult. And they also mentioned that it doesn't really matter how much current knowledge of TypeScript the candidate has, it's more important for the candidate to has the overall experience with JavaScript and frontend in general, because switching to TypeScript from JavaScript is not a long process for one person.

It depends on a person and his background how much time will it take to start coding with TypeScript, but it's definitely harder to switch from JavaScript to TypeScript for the entire company. 1 Respondent said that moving to TypeScript from JavaScript for an whole organization with 5 different IT departments takes from 1 to 2 years. Because the developers should write consistent code across the organization. Respondents who did not participate in the recruitment of new team members did not answer. Diagram is not provided for this question from the survey.

The last question in the "Requirements" group was about whether TypeScript complicates projects. All respondents who had experience using TypeScript noted that at the initial stage, initial setup with the TypeScript makes the project more complicated - additional configurations are required to use this programming language. It was also noted that when applying the template, this configuration does not complicate the project. With further web development, the use of TypeScript does not complicate the project, but facilitates the work with different data structures. The diagram is not provided for this question.

4.2.3 Advantages

Advantages sections introduces the strengths of TypeScript and JavaScript and shows and explains aspects of the rational use cases for these programming languages. In addition, the questions in the Advantages group provide information about the benefits of TypeScript over JavaScript.

The first question in the Advantages section from the questionnaire is about comparing TypeScript scalability to JavaScript scalability. All of the developers who participated in the survey and had experience with TypeScript said that scaling a frontend application with TypeScript is easier than scaling a JavaScript project. Like any strongly typed programming language, is better suited for the scaling purposes. Types and interfaces are the sort of additional documentation that gives an understanding of the complex data structures in a project. Thus, working with data structures becomes easier and more visual, since different IDEs support the TypeScript programming language and usually provide hints of data types used. All these TypeScript features increase scalability.

The second question in this group asked what are the advantages of TypeScript in terms of maintainability. All developers who have worked with TypeScript note that it is easier to maintain a project in the TypeScript programming language than in JavaScript, due to fewer potential errors and more explicit work with data structures. It is needed to mention that one project which is based on TypeScript can differ to the others which also use TypeScript programming language. The quality of code does not depend on the programming language choice, but it should be noted that TypeScript provides more different options for the explicit work with complex data types rather than JavaScript.

4.2.4 Limitations

Limitations section contains questions and answers from respondents about the shortcomings of both TypeScript and JavaScript. Based on the experience of the software developers, the questions about limitations of both JavaScript and TypeScript were answered.

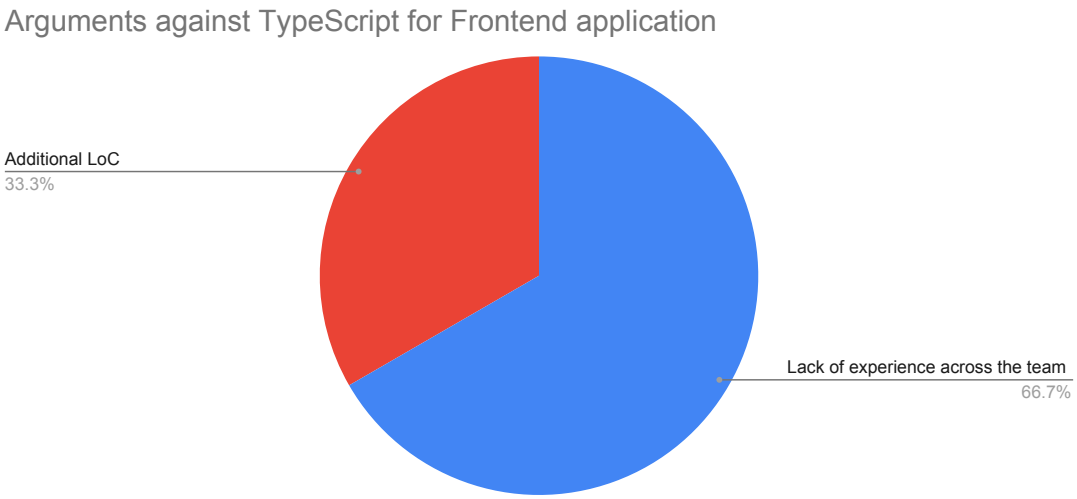

Figure 11. Arguments against TS for the Frontend Applications

Figure 11 shows that 6 out of 9 software developers who had experience with TypeScript noted that the main argument against using TypeScript in a new project is the lack of experience with this technology among team members. 3 engineers mentioned the need to write additional lines of code as a disadvantage of TypeScript and the reason against this programming language.

Figure 12 shows that 4 developers from a Senior level group said that if the library does not have the @types NPM package, it becomes a big issue for a team of software developers. It's time consuming to deal with such a libraries.

2 Senior software developers said that TypeScript can add additional complexity to the

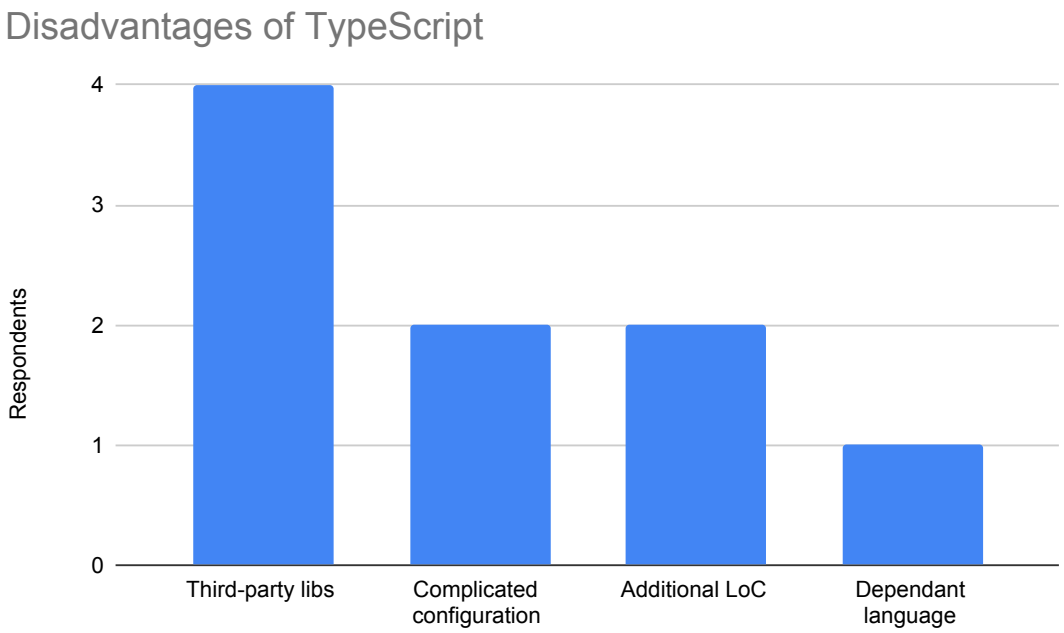

Figure 12. Disadvantages of TypeScript

configuration of the projects. For instance, for Webpack the additional manipulations in the configuration file are required. Furthermore, 1 software developer mentioned that software developers have to set up a lot of different plugins when dealing with TypeScript. For instance, for the Jest testing library the plugin should be installed. Moreover, when developers try to upgrade the version of different plugins in the project there is a huge possibility that something will crash in this case.

1 respondent from a Senior level group mentioned that the main disadvantage of TypeScript programming language is that it's not an independent language. It's still a superset of features over JavaScript and, for instance, on frontend when debugging a problem, in the browser the line of code of the final bundle is shown when dealing with SPA. That means that it is making the debugging process more complex, because as a result the JS code is used but not the TS.

Considering Research Question 2 and the differences between TypeScript and JavaScript regarding scalability, the answer was predictable - TypeScript was originally built for highly scalable applications, and the features that TypeScript provides to software developers help them to work with complex data structures and implement new features.

About the Research Question 3 - of course, TypeScript has its limitations. First, using TypeScript complicates the project at an early stage. Writing all the configurations for new projects takes time. The help of a ready-to-use template can make it easier to start

a project, but if we compare it with JavaScript, it becomes clear that TypeScript needs more customization. Furthermore, the work with third-party libraries is more complex with TypeScript.

5. Discussion

In this study we conducted the semi-structured survey with the software developers of various programming levels and the gathered information was processed and analysed to answer the questions about the rational use of TypeScript. Also, the scalability of TypeScript comparing to JavaScript was covered. Moreover, the limitations of both TS and JS were determined. Furthermore, the two application prototypes were created for the analysis purposes.

The MVPs and application prototypes differ from one to another. The functionality of MVPs can vary. To provide the answer to the question about the rational use of TypeScript and JavaScript for the MVP, it is crucial to understand the requirements for the certain application prototype. But in general, if we are discussing the prototype of the application, that means that this application should be a light weight, initial version of a bigger application.

So, we can conclude that when writing the first version of an application, it usually does not require working with complex data structures. So, for "pet" projects and simple services, both languages are a good choice with their own advantages and disadvantages. But if this prototype is scalable in the future, TypeScript should be chosen as the programming language.

Considering maintainability the answer is clear - TypeScript provides more instruments than JavaScript for better maintaining. All the developers who had experience with both programming languages noted that TypeScript is a better choice for further maintaining of projects.

Every programming language has it's own limitations and TypeScript is not an exception. TypeScript adds complexity to the initial setup. But after all, if we consider the modern trends of web frontend development it can be noted that pros of TypeScript overweight its cons, because this language was developed for creating the large-scale applications. To summarize all the information above about the limitations of TypeScript and JavaScript it can be mentioned that small MVP or application prototype can be created based on either TypeScript or JavaScript. Both approaches are fine in this case, but for the large-scale

applications, definitely TypeScript should be chosen.

5.1 Limitations

This work contains materials to cover the aspect of TypeScript's scalability, limitations and rationality of its use. However, this study also has some limitations which are listed below:

1. The survey was not large-scale. Only 11 respondents took part in the interviews. It may be possible to conduct an interviews with the larger amount of software engineers. Various developers can have different points of view concerning the programming languages used on the frontend side of the application.
2. In this study, the definition of rational use is probably not fully covered. It may be better to explain what exactly is the rational use for programming language.
3. The application prototypes made for this work were created for educational purposes only. Perhaps, real-life examples would be a more reliable option for analyzing MVP applications.
4. Just two applications were created for the code static analysis. It might be better to analyze a bigger amount of various MVPs, probably some MVPs that have an open access could be analysed for the purposes of this study.
5. This study is limited by the analysis of the two programming languages - TypeScript and JavaScript. While there are a lot of different less popular alternatives of languages which converts as a result to JavaScript.
6. To answer the question of the rationality of using a programming language, it should also be considered that indicators other than static code analysis and survey must be analysed. The ideal solution would be to analyze the real experience of companies that creates MVP applications in a real market. Moreover, analyze their costs for developing these applications in JavaScript and in TypeScript.
7. Advantages and disadvantages of TypeScript could be shown not only by code static analysis, but also it could be possible to analyze the code samples from the real applications and provide the example of strengths and weaknesses of both TypeScript and JavaScript.

6. Conclusion and Future Work

This study presents aspects of using the TypeScript programming language as an alternative to JavaScript for frontend applications. The survey was conducted with software developers who have different backgrounds in programming. Collected information about using TypeScript on the frontend side was processed and analysed. In addition, two application prototypes were created for analysis purposes.

RQ1: Static code analysis showed that there is no fundamental difference between TypeScript-based MVP and JavaScript-based MVP. The various rates and indicators of the analysis performed shows that potential errors and bugs are usually the same, and there is a slight difference in the results of static code analysis for these two application prototypes. But if the application scales in the future, TypeScript will be a good base in this case, since it provides the functions which are necessary to extend the functionality of the application.

RQ2: In terms of maintainability - TypeScript is better suited for large-scale applications because JavaScript was originally created for a different purpose. TypeScript is a superset of JavaScript designed to make the modern applications to be well-scaled. Dealing with complex data structures is explicit in TypeScript and implementing new features with TypeChecking mechanism can help programmers to avoid the potential bugs.

RQ3: The survey showed that the main TypeScript's limitation is the use of the third-party libraries that are not adapted for the use with TypeScript. Moreover, writing all the additional TypeScript configurations can take some developers' time. Furthermore, additional lines of code must be written in the TypeScript projects.

6.1 Future Work

From the results of the study and the limitations that were previously discussed the following enhancements may be considered for future research:

1. One possible future direction for the research is to include other JavaScript alternatives into consideration and compare them with JavaScript and TypeScript.
2. The code static analysis could be made for the real MVPs.

3. The survey could be expanded with more respondents.
4. The definition of the rational use should be better explained and clarified.

Bibliography

[1] A. Wirfs-Brock and B. Eich, "Javascript: The first 20 years," *Proceedings of the ACM on Programming Languages*, vol. 4, no. HOPL, pp. 1–189, 2020.

[2] J. Bentley, "Programming pearls: Little languages," *Communications of the ACM*, vol. 29, no. 8, pp. 711–721, 1986.

[3] A. Rauschmayer, *Speaking JavaScript: an in-depth guide for programmers*. " O'Reilly Media, Inc.", 2014.

[4] G. Bierman, M. Abadi, and M. Torgersen, "Understanding typescript," in *European Conference on Object-Oriented Programming*, Springer, 2014, pp. 257–281.

[5] B. Cherny, *Programming TypeScript: making your JavaScript applications scale*. O'Reilly Media, 2019.

[6] R. H. Jansen, *Learning TypeScript*. Packt Publishing Ltd, 2015.

[7] A. Freeman, "Understanding typescript," in *Essential TypeScript 4*, Springer, 2021, pp. 35–41.

[8] S. bin Uzayr, *TypeScript for Beginners: The Ultimate Guide*. CRC Press, 2022.

[9] V. Lenarduzzi and D. Taibi, "Mvp explained: A systematic mapping study on the definitions of minimal viable product," *2016 42th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pp. 112–119, 2016.

[10] L. Fischer and S. Hanenberg, "An empirical investigation of the effects of type systems and code completion on api usability using typescript and javascript in ms visual studio," *ACM SIGPLAN Notices*, vol. 51, no. 2, pp. 154–167, 2015.

[11] J. Bogner and M. Merkel, "To type or not to type? a systematic comparison of the software quality of javascript and typescript applications on github," *arXiv preprint arXiv:2203.11115*, 2022.

[12] D. Johannes, F. Khomh, and G. Antoniol, "A large-scale empirical study of code smells in javascript projects," *Software Quality Journal*, vol. 27, no. 3, pp. 1271–1314, 2019.

[13] N. Nikiforakis, L. Invernizzi, A. Kapravelos, *et al.*, "You are what you include: Large-scale evaluation of remote javascript inclusions," in *Proceedings of the 2012 ACM conference on Computer and communications security*, 2012, pp. 736–747.

[14] M. Patrou, K. B. Kent, and D. Sheppard, “Optimization of javascript large-scale urban simulations,” in *International Conference on Network-Based Information Systems*, Springer, 2020, pp. 20–31.

[15] A. B. Bondi, “Characteristics of scalability and their impact on performance,” in *Proceedings of the 2nd international workshop on Software and performance*, 2000, pp. 195–203.

[16] C. B. Weinstock and J. B. Goodenough, “On system scalability,” carnegie-mellon univ pittsburgh pa software engineering inst, Tech. Rep., 2006.

[17] E. Elliott, *Programming JavaScript applications: Robust web architecture with node, HTML5, and modern JS libraries*. " O’Reilly Media, Inc.", 2014.

[18] N. Black, “Boris cherny on typescript,” *IEEE Software*, vol. 37, no. 2, pp. 98–100, 2020.

[19] E. K. Kristensen and A. Møller, “Type test scripts for typescript testing,” *Proceedings of the ACM on Programming Languages*, vol. 1, no. OOPSLA, pp. 1–25, 2017.

[20] S. Fenton, Fenton, and Spearing, *Pro TypeScript*. Springer, 2014.

[21] A. Freeman, “Creating the project,” in *Essential Angular for ASP. NET Core MVC 3*, Springer, 2019, pp. 15–40.

Appendices

Appendix 1 - Non-exclusive licence for reproduction and publication of a graduation thesis

I Kirill Maksimov

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright, "A Qualitative Case Study on Using TypeScript as a JavaScript Alternative in Frontend Web Development in the Industry", supervised by Ishaya Peni Gambo, PhD
2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.
3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.
4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Kirill Maksimov
03/08/2022

Appendix 2 - Survey

Questionnaire

1. Demographics Questions
    - What is your current position?
    - How many years of web development experience do you have in total?
    - How many years of experience do you have with TypeScript?
    - How many projects with TypeScript on frontend did you have in production?
    - How many projects with the use of TypeScript did you start from scratch?
2. Requirements Questions
    - What were the reasons for choosing TypeScript?
    - If you need to choose a programming language for the new frontend application, what would it be?
    - How would you estimate the TypeScript community? Did you face any issues when looking for a solution for the existing issues?
    - How difficult is it to find the TypeScript Developer in the market?
    - What do you think about TypeScript configurations?
3. Advantages Questions
    - How would you estimate scalability of TypeScript compared to JavaScript?
    - What are the advantages of TypeScript considering maintainability?
    - What are the other advantages of TypeScript compared to JavaScript?
4. Limitations Questions
    - Based on your experience, what are usually the arguments against TypeScript when choosing the programming language?
    - Based on your experience, what were the limitations of using TypeScript?