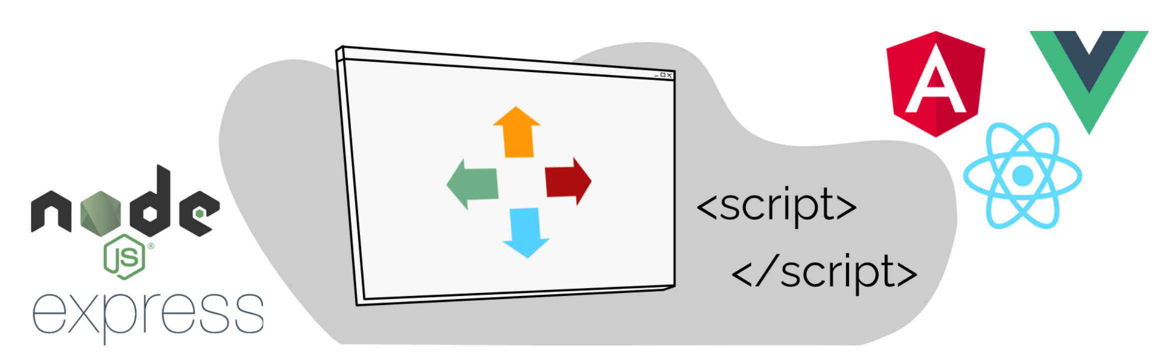


Evaluation and Comparison of Full-Stack JavaScript Technologies

Author
Nadine Weber

Supervisors
Prof. Dr. rer. nat. Tom Rüdebusch
Prof. Dr. rer. pol. Volker Sänger

Bachelor Thesis
University of Applied Sciences Offenburg
Media Department
Media and Information – Engineering and Design
Summer Semester 2022

Abstract

This thesis evaluates and compares current Full-Stack JavaScript Technologies. Through extensive research on the state of the art of JavaScript and its related frameworks, different aspects of Full-Stack Development are analysed to judge the popularity of technologies.

The language JavaScript and the idea of Full-Stack Development are presented with the functionality of different frameworks. The JavaScript runtime Node.js was examined and marked as the most influential JavaScript technology, which opened up many opportunities.

As technology stacks MERN, MEAN and MEVN were investigated, featuring the base technologies Node.js, MongoDB and Express.js. It was discovered that front-end frameworks have the most influence on which variant of Full-Stack can be chosen. Comparison criteria between the technology stacks were the learning curve, the maintainability, modularity and media integration. These criteria were extracted from research and a questionnaire conducted with students of the University of Applied Sciences Offenburg.

For the purposes of testing and experiencing a Full-Stack JavaScript application, the game *RemArrow*, based on the 1979s game *Simon*, was designed and implemented. The comparison with predefined criteria shows the result that the MERN stack with React.js is the best to learn and promises the most potential. Arising JavaScript technologies and their popularity are very dependent on the industry and skill set of the developer.

In conclusion, it can be established that the concept of Full-Stack Development is currently very interesting and more than just a trend. It has potential of becoming a new kind of web development, and part of the curriculum taught at universities. Expert knowledge is needed but there is a high demand and much potential for Full-Stack JavaScript Developers.

Preface

The following work was written as the thesis of the bachelor degree *Media and Information – Engineering and Design* at the University of Applied Sciences Offenburg. The topic was chosen because of the topicality of JavaScript frameworks and the idea to explore the state of the art of Web Development Technologies.

First of all, thank you to my amazing family – in particular my mum, my dad, my sister Melissa and my grandma – for always believing in me. You have supported me through the course of my studies and encouraged me to become the version of myself that I am today.

During my thesis I was lucky to have a group of people behind me helping me philosophise about the topic and creating new ideas. I want to especially thank Julia Marcia Mann and Stefano Gampe, who gave me amazing advice, gave me feedback on different aspects of this work and pushed me during the writing process.

I would like to thank my supervisor Prof. Dr. Tom Rüdebusch for the extensive help, his enthusiasm and guidance throughout my thesis.

Special thanks to Melissa Weber, Juliane Hoffmann, Leonie Hoffmann and Stefano Gampe for reviewing this document.

And last but not least, I want to thank my fellow students who have made this study programme an amazing journey.

Nadine Weber, Offenburg August 2022

Pledge of Integrity

I hereby declare that the thesis submitted is my own unaided work. All direct or indirect sources used are acknowledged as references. This paper was not previously presented to another examination board and has not been published.

Offenburg, August 31, 2022

Contents

List of Figures I

List of Tables V

1 Introduction 1

1.1 Motivation . . . 1
1.2 Goals . . . 1
1.3 Methodology . . . 2

2 Full-Stack JavaScript Development 3

2.1 JavaScript . . . 3
2.1.1 History . . . 3
2.1.2 Flavours . . . 4
2.1.3 Versions . . . 6
2.1.4 Popularity . . . 9
2.1.5 Document Object Model (DOM) . . . 13
2.2 JavaScript Frameworks . . . 15
2.2.1 Client-Server-Relationship . . . 15
2.2.2 Statistics & Usages . . . 16
2.2.3 Frontend . . . 21
2.2.3.1 React.js . . . 21
2.2.3.2 Angular . . . 28
2.2.3.3 Vue.js . . . 34
2.2.4 Backend . . . 39
2.2.4.1 Node.js . . . 39
2.2.4.2 Express.js . . . 43
2.2.4.3 MongoDB . . . 45
2.2.5 JavaScript Alternative: PyScript . . . 48
2.2.6 Characteristics . . . 48
2.3 Technology Stacks . . . 49
2.3.1 Fundamentals . . . 49
2.3.2 Tech Stack Variants (MEAN / MERN / MEVN) . . . 51
2.4 Full-Stack Web Development . . . 54
2.4.1 Full-Stack Developer Role and Skill Set . . . 54
2.4.2 Development Operations (DevOps) . . . 56
2.4.3 Advantages and Disadvantages of Full-Stack Development . . . 62

3 Comparison **63**
3.1 Scope Conditions . . . 63
3.2 Questionnaire . . . 64
3.2.1 Evaluation . . . 64
3.2.2 Recommendations . . . 69
3.3 Criteria . . . 70
3.3.1 Learning Curve . . . 71
3.3.2 Maintainability . . . 73
3.3.3 Modularity . . . 74
3.3.4 Media Integration . . . 75
3.4 Comparison . . . 76
3.4.1 Detailed Comparison . . . 76
3.4.2 Overview of Comparison . . . 84

4 RemArrow – Can you remember? **87**
4.1 Requirements Analysis . . . 88
4.2 Game Logic . . . 89
4.3 Design . . . 91
4.4 User Interface Design . . . 92
4.5 A Full-Stack MERN Implementation . . . 94

5 Conclusion & Outlook **109**
5.1 Conclusion . . . 109
5.2 Evaluation and Outlook . . . 110

Bibliography **111**

Glossary and Acronyms **119**

A Questionnaire **123**

B Source Code Examples **139**

C Screenshots of Prototypical Application **149**

D Poster **151**

List of Figures

2.1 JavaScript Logo [4] 3
2.2 LiveScript Logo [4] 5
2.3 CoffeeScript Logo [4] 5
2.4 TypeScript Logo [15] 5
2.5 Ecma International Logo [23] 6
2.6 Babel Logo [27] 8
2.7 Transpiler Babel transforming ES6 to ES5 syntax, adapted from [27] 8
2.8 PYPL PopularitY of Programming Language Index, June 2022 [2] 9
2.9 Ranking of Programming Languages used in GitHub Projects by the State of the Octoverse 2021 [32] 10
2.10 Top 5 Programming Languages used by Developers, adapted by Slash Data's Developer Nation Q1 2022 [33] 11
2.11 Size of Programming Language Communities in Q1 2022 by Slash Data's Developer-Nation [33] 12
2.12 Simple Button (written in HTML) shown in browser 13
2.13 Simple Button shown in DOM as source code 13
2.14 Simple Button shown in DOM tree structure 14
2.15 Frontend Development [44] 15
2.16 Backend Development [44] 15
2.17 State of JavaScript 2021 Logo [46] 16
2.18 Satisfaction to User Count Ratio by StateOfJS 2021, adapted from [17] 17
2.19 Ranking of the Usage of Frontend JavaScript Frameworks by StateOfJS 2021, adapted from StateOfJS 2021 [17] 18
2.20 Ranking of the Usage of Backend JavaScript Frameworks by State of JS 2021, adapted from StateOfJS 2021 [17] 19
2.21 Overview of JavaScript Frameworks 20
2.22 ReactJS Logo [46] 21
2.23 Virtual DOM Functionality, adapted by [72] 22
2.24 JSX [75] 23
2.25 Functional Component in React [75] 23
2.26 Class Component in React [75] 24
2.27 Adding a React Component to a website with simple code [75] 25
2.28 Create-React-App page before any modifications 25
2.29 Simple Button in React 25
2.30 React Application UI 25
2.31 Simple Button in React (App.js) 26
2.32 Button as Class Component (button.js) 26
2.33 Button included as Component (App.js) 27

2.34 AngularJS – Superheroic JavaScript MVW Framework [81] . . . 28
2.35 Angular – The modern web developer's platform [52] . . . 28
2.36 Component in Angular [84] . . . 29
2.37 Properties and Event Handling Example . . . 29
2.38 Incremental DOM Functionality, adapted by [87] . . . 30
2.39 Angular Quick Start page before any modifications . . . 31
2.40 Simple Button in Angular . . . 31
2.41 Angular Application UI . . . 31
2.42 Simple Button in Angular (App.component.html) . . . 32
2.43 Button Component (button.component.html) . . . 32
2.44 Button Component (button.component.ts) . . . 32
2.45 Button included as Component (app.component.html) . . . 33
2.46 WebAssembly Logo [91] . . . 33
2.47 VueJS Logo [92] . . . 34
2.48 Vue Single File Component (SFC) Example . . . 34
2.49 Vue Quick Start Setup Options . . . 36
2.50 Vue Quick Start page before any modifications . . . 36
2.51 Simple Button in Vue . . . 36
2.52 Vue Application UI . . . 36
2.53 Button Single File Component (CostumButton.vue) . . . 37
2.54 Button included as Component (App.vue) . . . 37
2.55 jQuery Logo [100] . . . 38
2.56 NodeJS Logo [103] . . . 39
2.57 Node.js Event Loop and Task Queue, adapted by [3] . . . 40
2.58 Node.js Installation Process including the npm package manager . . . 42
2.59 npm search of the keyword *slideshow* [116] . . . 42
2.60 Express Logo [54] . . . 43
2.61 Creating an Express Server (server.js) [54] . . . 43
2.62 URL Route Example with Express (server.js) . . . 44
2.63 MongoDB Logo [120] . . . 45
2.64 JSON Example of Database Entries . . . 45
2.65 JSON to BSON Data Structure . . . 46
2.66 MongoDB Atlas Start Page . . . 47
2.67 Stack Data Structure with Last-in/First-out (LIFO) manner . . . 49
2.68 MERN Stack consisting of MongoDB, Express.js, React.js and Node.js in comparison to the stack data structure . . . 50
2.69 Overview of Technologies used in Full-Stack Development . . . 50
2.70 Building a Tech Stack, adapted by [133] . . . 51
2.71 MERN / MEAN / MEVN Technology Stack Architecture, adapted by [135] . . . 53
2.72 Skill Set of a Full-Stack Developer . . . 55
2.73 Involvement in DevOps by Company Role, adapted from SlashData's Developer Nation Q1 2022 [33] . . . 56
2.74 Type of involvement in DevOps, adapted from SlashData's Developer Nation Q1 2022 [33] 57
2.75 DevOps Delivery Pipeline and Feedback Loop [139] . . . 58
2.76 nvm Update of Node.js to the Long-Term Support Version . . . 59
2.77 Scheme of Monolithic vs. Microservice Architecture, adapted from [153] . . . 60
2.78 Full Stack Developer [155] . . . 62

3.1 Study Programmes at the University of Applied Sciences Offenburg 65
3.2 Language Experience of Students at the University of Applied Sciences Offenburg . . 66
3.3 Knowledge of Stack Types of Students at the University of Applied Sciences Offenburg 67
3.4 Example of a Learning Curve, adapted from [158] 71
3.5 Top 5 ways in which developers learn to code, adapted by Slash Data's Developer Nation Q1 2022 [34] 72
3.6 Expected frequency of maintaining a website by Students of the University of Applied Sciences Offenburg 73
3.7 Interest in the concept of Modularity by Students at the University of Applied Sciences Offenburg 74
3.8 Learning Curves from React.js, Angular and Vue.js 79

4.1 Game Cover of the Electronics Game Senso (German Version) [172] 87
4.2 Flowchart of Game Logic (Basic structure) 89
4.3 Flowchart of Game Logic (Initialising the Game and Creating a Sequence of Keys) . . . 90
4.4 Flowchart of Game Logic (Playing the Sequence, Checking whether the correct Key was pressed and Setting up the Highscore) 90
4.5 Name Scribbles & Game Components 91
4.6 RemArrow Colour Scheme 92
4.7 RemArrow Picture- and Word-Mark 92
4.8 RemArrow UI Design Desktop 93
4.9 RemArrow UI Design Mobile 93
4.10 Structure of the MERN Stack, adapted by [135] 94
4.11 GitLab Repository of RemArrow 96
4.12 GitLab Page Pipeline 96
4.13 Creating an Express Server (server.js) 97
4.14 Defining a Custom Command for Development (package.json) 97
4.15 Listening for Requests with an Environmental Port Number (server.js) 98
4.16 Score APIs (scores.js) 99
4.17 Adding the Routes to the Server file (server.js) 99
4.18 Testing API Responses in Postman 100
4.19 Creation of a free shared cluster in MongoDB Atlas 100
4.20 Creation of a User and Setting a Password for the Connection String 101
4.21 Network Access for all users 101
4.22 Connecting MongoDB Atlas to Express using mongoose (server.js) 102
4.23 MongoDB Atlas Database RemArrow 102
4.24 RemArrow Class Component (remarrow.js) 103
4.25 RemArrow Root File (App.js) 103
4.26 Creating Arrow Components by passing through .svg files as imgURLs (arrows.js) . . . 104
4.27 Creating Arrow Components by passing through .svg files (arrows.js) 104
4.28 Media Queries for Arrow Container Areas (App.css) 105
4.29 Arrow Styles (Hover and Active-Effect for Arrow Components) (App.css) 106
4.30 Score and Highscore State (scores.js) 106
4.31 State / API Update Functions (scores.js) 107
4.32 ComponentDidMount (scores.js) 108

A.1 Study Programmes 125

A.2 Number of Semesters . . . 125
A.3 Age Groups . . . 126
A.4 Experience with Web Development . . . 126
A.5 Web Development Fields . . . 127
A.6 Interest in Web Development Types . . . 127
A.7 Attendance of the course “Interactive Distributed Systems” . . . 128
A.8 Experience with JavaScript . . . 128
A.9 Language Experience . . . 129
A.10 Familiarity with the term Full-Stack Web Development . . . 129
A.11 Interest in Full-Stack Web Development . . . 130
A.12 Opinion on JavaScript Frameworks . . . 130
A.13 Familiarity with Full-Stack Web Development . . . 131
A.14 Knowledge of Frontend Frameworks . . . 131
A.15 Knowledge of Backend Frameworks . . . 132
A.16 Knowledge of Stack Types . . . 132
A.17 Possible topics featured in the course Interactive Distributed Systems . . . 133
A.18 Frameworks that should be used in a laboratory experiment . . . 133
A.19 Opinions on topics taught in the course Interactive Distributed Systems . . . 134
A.20 Interest in the concept of Modularity . . . 135
A.21 Expected frequency of maintaining a website . . . 135
A.22 Interest in trends . . . 136
A.23 Students way of getting to know new technologies in the field of Web Development . . . 136
A.24 Students way of learning new technologies . . . 137
A.25 Way of Learning starting to use a framework . . . 137
A.26 Comments . . . 138

B.1 Server - server.js . . . 140
B.2 Server - scores.js . . . 141
B.3 Server - scoreModel.js . . . 142
B.4 Server - scoreController.js . . . 142
B.5 Frontend - App.js . . . 143
B.6 Frontend Styles - All, Scores (App.css) . . . 144
B.7 Frontend Styles - Arrows, Buttons (App.css) . . . 145
B.8 Frontend Styles - Media Queries (App.css) . . . 146
B.9 Button Group Component (ButtonGroup.js) . . . 147
B.10 Titles Container (titles.js) . . . 147

C.1 Graphic of a possible UI Design for Desktop . . . 149
C.2 RemArrow Desktop Size . . . 149
C.3 RemArrow Tablet Size . . . 150
C.4 RemArrow Smartphone Size . . . 150

All figures, pictures, diagrams and graphics without citation were created by Nadine Weber.

List of Tables

2.1 Differences between ES5 and ES6, Table adapted from [7, 25] 7
2.2 Versions of Frontend Frameworks 21
2.3 Versions of Backend Frameworks 39
2.4 Overview of Terminologies in SQL and NoSQL Databases 45
2.5 Overview of popular Technology Stacks 52

3.1 Comparison of Frontend Frameworks 76
3.2 Comparison of Quick Start Options 77
3.3 Comparison of JavaScript Full-Stacks (MERN vs. MEAN vs. MEVN) 84

4.1 Flowchart Legend 89
4.2 Overview of Score APIs in RemArrow 98

All tables without citation were created by Nadine Weber.

Chapter 1

Introduction

1.1 Motivation

Using websites is and continues to be a big part of our digital lives. With constantly changing and diversifying web technologies and frameworks, the term *Full-Stack JavaScript* is often mentioned. This thesis will investigate the current state of *Full-Stack JavaScript Technologies* and shows a comparison between the most common stack types (in particular MEAN – MongoDB ExpressJS AngularJS NodeJS, MERN with ReactJS and MEVN with VueJS). Furthermore, an exemplary implementation of a game will be made using the MERN stack.

Especially in the field of Web Development, it is important to know the technologies, keep up with the latest trends, and try out different features. This helps in finding out what technology is best for you, your use case, and your type of company. Because there is a variety of technologies – which increase in number year by year – it is important to know whether they are just trending because they are new or if they are relevant for future developments. These developments can either be seen on the technical side of gaining experience for even better frameworks or on a personal level of growth with the broadening of your expertise.

To take a closer look and evaluate the different Full-Stack Architectures, this thesis will focus on the most important concepts and paradigms of Full-Stack JavaScript Development. Furthermore, the conception and implementation of an exemplary application will be used to draw conclusions about their state and possible opportunities.

1.2 Goals

The objective of this thesis is to give an insight into the fundamentals of Full-Stack Web Development Technologies while making a detailed comparison based on some prototypical implementation. With the high number of frameworks and technologies in mind, this thesis will show which technology is best to learn and to use in the context of university students studying web development.

In order to fully understand the meaning of Full-Stack Development, an overview of its technical terms and a general insight of its importance will be laid out. Starting with the most important terms and concepts, different perspectives will be used to look at advantages and disadvantages of this trend.

1.3 Methodology

For a comprehensive summary of the current state of used JavaScript Technologies, literary research and different empirical studies were taken in as background knowledge. In order to grant a variety of perspectives, all studies carried out for this paper feature different user groups and methods of data collection (Google Trends, GitHub Statistics, etc.). When looking at the statistics in detail, the data and methodology of the studies will be further explained in *Chapter 2.2.2* (Statistics & Usages).

Web Technologies change very fast and are often dependent on the community. Therefore, it was important to additionally look into community support in forums (i. e. StackOverFlow) or Blog Entries. In forums one can view issues being raised and find summaries of experienced developers. To ensure that these (rather non-scientific) sources were correct, double-checking and some face-to-face interviews with actual developers were carried out.

An empirical study was conducted on the campus of the University of Applied Sciences Offenburg with students in the course *Interactive Distributed Systems*. The Questionnaire was designed in the German Language as the students' native language is German. To conduct the study, the free online form creator *Google Forms* was used [1]. Its easy and fast creation was perfect to receive meaningful results very quickly without any obstacles.

This thesis deals with the fundamentals of the language JavaScript, the state of the art of JavaScript Frameworks and their comparison when used as Full-Stack Web Technologies. By understanding the meaning of Full-Stack Web Development itself, *Chapter 2* will examine the trending concept theoretically and practically with an implementation of a simple button.

After getting to know the fundamental technologies, the criteria for the subsequent comparison of JavaScript Frameworks will be conducted using a requirement analysis and the evaluation of the questionnaire. The detailed comparison in *Chapter 3* will be based on predefined criteria. These aspects will be taken out of the context, inspected in detail and put back into perspective.

Chapter 4 will focus on the description of the game *RemArrow*, that has been implemented as a Full-Stack JavaScript Implementation in form of a MERN stack. To be able to judge the functionality of the game prototype, the thesis takes a look at the comparison criteria, which all have great meaning to the use of this application.

With various aspects of Full-Stack JavaScript Development presented, a conclusion will be drawn and evaluated, and an outlook into future developments will be given.

Chapter 2

Full-Stack JavaScript Development

JavaScript and its frameworks can be a lot to take in when first starting Web Development. To be able to compare *JavaScript Technologies* and get to the bottom of the trend *Full-Stack JavaScript Development* this chapter takes a look at the fundamentals. Starting with the language JavaScript itself, this chapter continues with an explanation of the later used frameworks, the technology stacks, as well as the meaning of the role of a Full-Stack Developer.

This knowledge will be taken as the base for creating exemplary applications and deciding on appropriate comparison criteria. The understanding of these fundamental technologies and general conditions in web development will help the evaluation process.

Disclaimer: Basic knowledge of HTML5, CSS3 and JavaScript is assumed.

2.1 JavaScript

As one of the main and most used languages in Web Technologies [2], it is interesting to find out the origin of JavaScript. After taking a look into the history, the JavaScript flavours, the current state of versioning, the popularity, as well as the basics of the Document Object Model (DOM) will be examined.

2.1.1 History

> JavaScript is everywhere. *If you are able to read [a website] article or browse through your Facebook news feed, it is mostly due to this single technology.*
> – AltexSoft [3]

Figure 2.1: JavaScript Logo [4]

JavaScript (abbr. JS) was invented by *Brendan Eich* in 1995 and is commonly known as the programming **language of the web** [5]. It was introduced as a way to add programmes to web pages in the Netscape Navigator 2.0 browser [6]. The language has since been adopted by all web browsers, as well as server and embedded applications [7]. It has made modern web applications possible, with which you can interact directly without having a page reload for every action [6, 8].

As the dominant web development technology, JavaScript has been around for over 25 years, maintaining its place on top of the leader board (see Figure 2.9). It enables the use of applications in web browsers and can be used as both a front- and backend language [6]. JavaScript is used to provide various forms of **interactivity and cleverness**, especially for the **client-side scripting** of the World Wide Web (WWW) [3, 6, 9].

Another name often mentioned in the context of content, structure and layout of a website is **DHTML**. This stands for *Dynamic Hypertext Markup Language*. It describes the set of technologies otherwise known and implemented as the basic building blocks of the web with HTML, CSS and JS. JavaScript is used to add behaviour to your website and is the engine on which DHTML runs. Any dynamic manipulations of content can be described with the help of the **DOM** (more details in 2.1.5). DOM was a term frequently used in the past.

ECMAScript (abbr. ES) standardises and is the **official name** of the JavaScript language. It was standardised in 1997 with the scripting language specification by Ecma International, the *European Computer Manufacturers Association* [8, 10]. In practice, the terms ECMAScript and JavaScript can be used interchangeably. They are two names for the same language. The name JavaScript was chosen for marketing reasons. When JS was released, the language Java was gaining popularity. The goal was to use this momentum in favour of JS. As is clear now, the name JavaScript took on well, as most people use the name JavaScript instead of the standard ECMAScript. [6, 11]

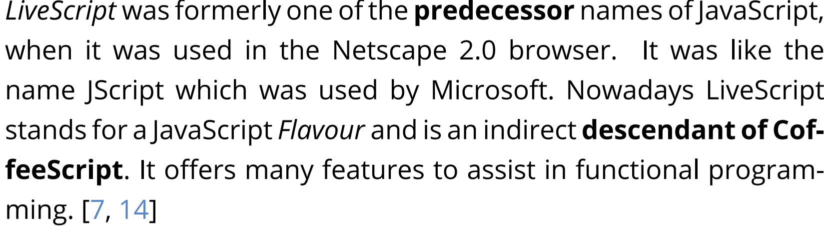
From a programmers' perspective, JavaScript is rather controversial. Although stated as a general-purpose programming language in the language specification [7], it is a scripting language with an object-oriented approach. Programmers thoughts on this are conflicted. JavaScript is liberal in the sense of it accepting almost anything – especially because it is a **weakly typed** language, where you don't have to specify which data type a variable is. Finding problems in the programmes is therefore harder, but the flexibility has advantages, i. e. using techniques where other programming languages are constrained. It needs time to learn the proper syntax but once internalised there is much space for creativity. [6]

2.1.2 Flavours

Sticking to scripting languages, worth mentioning are the languages *LiveScript*, *CoffeeScript* and *TypeScript*. They are so-called JavaScript *Flavours*. The name **flavour** is used to introduce different interpretations of the language JavaScript itself. They all compile to JavaScript but offer a more strict coding format making the JS more readable and easy to manage. Essentially, they are equivalent to JS only with stricter code conventions. This solves the problem of weakly typing and makes programming much easier with different flavours suited to your use case. [11–13]

Most JavaScript flavours are compilated, which means they have a speed advantage in comparison to JavaScript. When looking at scripting languages, there are two different types: **compiled and interpreted** languages. JavaScript is an interpreted language, such as most scripting languages. This means the code does not directly translate into machine code. Instead, it has an interpreter translating the code. This means the time it takes to translate JavaScript into machine code takes longer. In the following, different JS flavours will be examined. [11]

Figure 2.2: LiveScript Logo [4]



LiveScript was formerly one of the **predecessor** names of JavaScript, when it was used in the Netscape 2.0 browser. It was like the name JScript which was used by Microsoft. Nowadays LiveScript stands for a JavaScript *Flavour* and is an indirect **descendant of CoffeeScript**. It offers many features to assist in functional programming. [7, 14]

Figure 2.3: CoffeeScript Logo [4]

CoffeeScript is a language that compiles to JavaScript. It was the first JS flavour and was invented to show the heart of JavaScript, described as gorgeous by the CoffeeScript developers. It states to only use **the good parts of JavaScript** in a simple way [13]. The CoffeeScript Compiler produces modern JavaScript syntax, which includes asynchronous functions and JSX (JavaScript XML) amongst other things. Those terms will be explained in more detail when looking at React.js (see 2.2.3.1). Every .coffee file can be compiled into a .js file. One main difference to JS is that CoffeeScript uses indentation instead of curly brackets, similar to the syntax of *Python* or *YAML*. [13]

Figure 2.4: TypeScript Logo [15]

TypeScript (abbr. TS) is JavaScript with **syntax for types**. It is a strongly typed programming language with strict types to add additional syntax. This helps in catching errors early during the development process in the editor. As JavaScript has much flexibility the idea of adding strict types and static checking (detecting errors in code without running it) is very popular. It can be done simply by adding the line @ts-check to a JS file, offering error descriptions in the code immediately. TypeScript becomes JavaScript when removing the types. It gets transformed easily and doesn't change its behaviour. When learning TypeScript one needs to learn JavaScript, as it's based on it. TypeScript is JavaScript's runtime with a **compile-time type checker**. That means every resource aiming at the understanding of JavaScript will help learning TypeScript as well. It is by far the most used JavaScript Flavour currently. [16–18]

Further JavaScript Flavours are *Elm* [19], *PureScript* [20], *Reason* [21] and *ClojureScript* [22], to list more without naming the details. They all have different **focuses** on the strictness of types, their use-case and the way of programming (i. e. functional or object-oriented).

With these *Scripting* Variants in mind, the standard ECMAScript will be focused on.

2.1.3 Versions

Most Programming Languages and Web Technologies are constantly changing and developing. As already described in the History of JavaScript (2.1.1) the standardisation of JavaScript is **EcmaScript** (abbr. ES). As of today there are six versions of EcmaScript and therefore six major improvements to JavaScript. [7]

Figure 2.5: Ecma International Logo [23]

ECMAScript has been adopted as the scripting language which is supported in all browsers in overlap with the World Wide Web [7]. Here is a short overview of EcmaScript versions categorised by their main revisions [8]:

- The *Original JavaScript*: ES1 ES2 ES3 (1997-1999)
- The *First Main Revision*: ES5 (2009)
- The *Second Revision*: ES6 (2015)
- *Yearly Additions* (2016, 2017, 2018)

The first edition of the Ecma Standard 252 (ECMA-262) was adopted by the Ecma General Assembly of June 1997 [7]. The European Computer Manufacturers Association (abbreviated and officially trademarked Ecma) has a detailed documentation of the changes. Since ES3, ECMAScript 1 - 6 are fully supported in all modern browsers . After ES6, the versions are named by their release year. The latest version, being the fourteenth edition of ECMAScript (ECMAScript 2023), **was approved** in July 2022. [7, 10]

Changes between the **first and second edition** are editorial in nature. The **third edition** was given improvements with powerful regular expressions, better string handling, new control statement and minor changes in anticipation of future language growth. [7]

When **ES4** was in preparation, issues were being raised, splitting the committee into two different camps. Brendan Eich (the inventor of JavaScript) puts it this way:

> *" It's no secret that the JavaScript standards body, Ecma's Technical Committee 39, has been split for over a year, with some members favoring ES4, a major fourth edition to ECMA-262, and others advocating ES3.1 based on the existing ECMA-262 Edition 3 (ES3) specification. Now, I'm happy to report, the split is over. "* [24]

Due to conflicts in the technical committee of ECMAScript whether to advance the ES3 version with an incremental update (ES3.1) or to switch approaches with ES4, the fourth edition never was released or even completed. Instead, they renamed ES4 to ES5 and worked together for this version in particular. The major work of ES3 was incorporated into the development of the sixth edition, which was unofficially code-named '**Harmony**'. It was made clear that communication in their collaboration was important. In correspondence, focusing on the goals and finding harmony was the best solution to proceed. [7, 24]

The **fifth edition**, which was released in 2009, was the first major revision and was a huge success. It featured many functions that are used frequently today, such as working with multiple line strings and array functions (e.g. map). 2013, ES5 was fully supported in all modern browsers (i. e. Chrome, IE, Edge, Firefox, Safari, Opera). Many improvements were made throughout the versions, adapting to new technologies and growth of data handling. A big leap to ES6 was made. This development needs a closer inspection and will be analysed in the following. [7, 8]

ECMAScript Version 6

The ES6 version was the second major revision and an enhancement, because it offers the opportunity for **object-oriented** programming. JavaScript wasn't originally object-oriented. The concept of classes and objects were introduced with ES6. [7, 11]

EcmaInternational states that the ES6 edition is the culmination of a fifteen-year long effort. It was adopted in December 2009, featuring language specifications that had become common in browser implementations,**support** for features, alongside with better support for large applications. Furthermore, the use of ECMAScript as a compilation target for other languages (2.1.2) was included. Some other advancements were the inspection of objects, support for JSON object encoding format, enhanced error checking and programme security. [7]

Between ES5 and ES6 there are many **differences**. These are shown in Table 2.1. The most basic ones are more specific variable definitions (var, let and const), arrow functions (a short syntax for writing function expressions) and the for/of Loop (to iterate through iterable objects). Functions like map, sets and classes for JavaScript objects show a more programmatic approach to ES6. As most technologies evolve to become more complex and bigger, these changes were adapted very quickly and many frameworks use the ES6 syntax by default.

	ES5	**ES6**
Edition	ECMA-262 5th edition	ECMA-262 6th edition
Definition	Fifth edition of ECMAScript	Sixth edition of ECMAScript
Release Date	December 2009	June 2015
Data Type Support	Primitive data types (string, number, boolean, null and undefined	New Primitive data type 'symbol' for supporting unique values
Defining Variables	By using the word **var**	By using the words **var, let and const**
Performance	Lower Performance due to missing features	Higher Performance due to new features and the shorthand storage implementation
Support	Lesser support than ES6 but still very good	Wide range of community support
Object Manipulation	Time-consuming	More smoothly (due to destructuring and speed operators)
Arrow Functions	Not featured (Both function and return keywords are used to define a functio)	New Feature (function keywords is not required to define a function)
Loops	For-Loop	For...of-Loop to perform an iteration over the values of iterable objects

Table 2.1: Differences between ES5 and ES6, Table adapted from [7, 25]

Many runtime environments or older browsers still do not support the entirety of the ES6 syntax. In this case, something is needed to convert modern JavaScript into old JavaScript – to support backwards compatibility. This conversion is made by a so-called **transpiler**. A transpiler is a source-to-source compiler, that takes a source code file and converts it to another source code file, which is in another or different version of the same language [26]. Some JavaScript Flavours have built-in support for transpilers to run the transpile command without new software additions. A few transpilers are *Babel* [27], *Bublé* [28] or *Traceur Compiler* [29].

An example of a **transpilation with Babel** is shown in Figure 2.7 using the map function. The map function creates a new array by calling a function for every element of an already existing array. While ES6 shows the lightweight syntax with an arrow function, Babel transpiles – meaning it transforms the syntax – into an inner function call which is commonly known in JavaScript. After the transpilation, all environments using the programme are supported and should know what to do. [13, 27]

Using Babel as the exemplary **transpiler**, there needs to be an understanding of source-to-source compilers themselves to be able to know how to configure them correctly. Without commanding a compiler, it will do nothing. Babel supports TypeScript and JSX syntax presets. This leads to an easier handling of the compiler. [27, 30]

Figure 2.6: Babel Logo [27]

```
1  Babel Input: ES2015 arrow function
2  [1, 2, 3].map( n => n + 1 );
3
4  Babel Output: ES5 equivalent
5  [1, 2, 3].map( function(n) {
6          return n + 1;
7      }
8  );
```

Figure 2.7: Transpiler Babel transforming ES6 to ES5 syntax, adapted from [27]

Because most functions of the newer versions tend to be very specific, the latest features from ECMAScript 2016-2022 will be addressed in the section about React.js (2.2.3.1). [7]

2.1.4 Popularity

> *Even the simplest Website can benefit from a little interactivity here and there – making it better, more responsive, or easier to use.*
> – Langridge [31]

In our internet bubble, we see many websites which either have a good or bad user experience. Interactivity upgrades help most websites to be more usable in principle and more fun to use, instead of just reading plain text. JavaScript is an important part of the work of web developers to improve their sites. Judging the popularity, usage and importance of the language itself can be done with the help of some facts and figures.

When thinking of developing, most people start by searching for keywords in search engines (cf. A.24). One simple search and you get thousands of results in less than a few seconds. The **PYPL PopularitY of Programming Language Index** takes exactly this type of data to estimate popularity. It was created by analysing how often languages and language tutorials are searched on the search engine Google. The more search queries, the more popular the language is assumed to be, remaining a product of the collective wisdom of the web developer community. The raw data in this index was sourced from Google Trends. [2]

As shown in Figure 2.8 JavaScript was the 3rd most searched language in June 2022, trending with 9.21 % shares and +0.4 % uprising compared to Mai 2022. This supports the idea of JavaScript being interesting for users. [2]

Worldwide, Jun 2022 compared to a year ago:

Rank	Change	Language	Share	Trend
1		Python	27.61 %	-2.8 %
2		Java	17.64 %	-0.7 %
3		JavaScript	9.21 %	+0.4 %
4		C#	7.79 %	+0.8 %
5		C/C++	7.01 %	+0.4 %
6		PHP	5.27 %	-1.0 %
7		R	4.26 %	+0.5 %
8	↑↑↑	TypeScript	2.43 %	+0.7 %
9	↓	Objective-C	2.21 %	+0.1 %
10	↓	Swift	2.17 %	+0.4 %
11	↑↑	Matlab	1.71 %	+0.2 %
12	↓↓	Kotlin	1.57 %	-0.2 %
13	↓	Go	1.48 %	+0.0 %
14	↑↑	Rust	1.29 %	+0.4 %
15		Ruby	1.1 %	-0.0 %

Figure 2.8: PYPL PopularitY of Programming Language Index, June 2022 [2]

Another great indicator of popularity is the ranking of the Programming Languages used in version control systems (i. e. GitHub), for they are used by most developers (see more in 2.4.2 DevOps). The **State of the Octoverse** by GitHub shows all development activities on GitHub. Their research combines telemetry from 4 million repositories and surveys from more than 12,000 developers. With this data, they show current trends and can predict future trends more specifically. For ranking Programming Languages, they view the usage in repositories. [32]

When looking at the statistic by Octoverse, the increase in rank of **TypeScript** is worth mentioning. TypeScript alone has improved its rank from 10th to 4th place in only 5 years. In some statistics TypeScript is listed separately, although it is a JavaScript Flavour and can be seen as part of JavaScript. This may be justified by TypeScript's growing popularity due to the feature of showing errors at compile time. Additionally through complex applications being more manageable with the strongly typed language and the trend of using a universal language. The competitive JavaScript products *TypeScript* and *PyScript* will be investigated in more detail when looking at frameworks. [2]

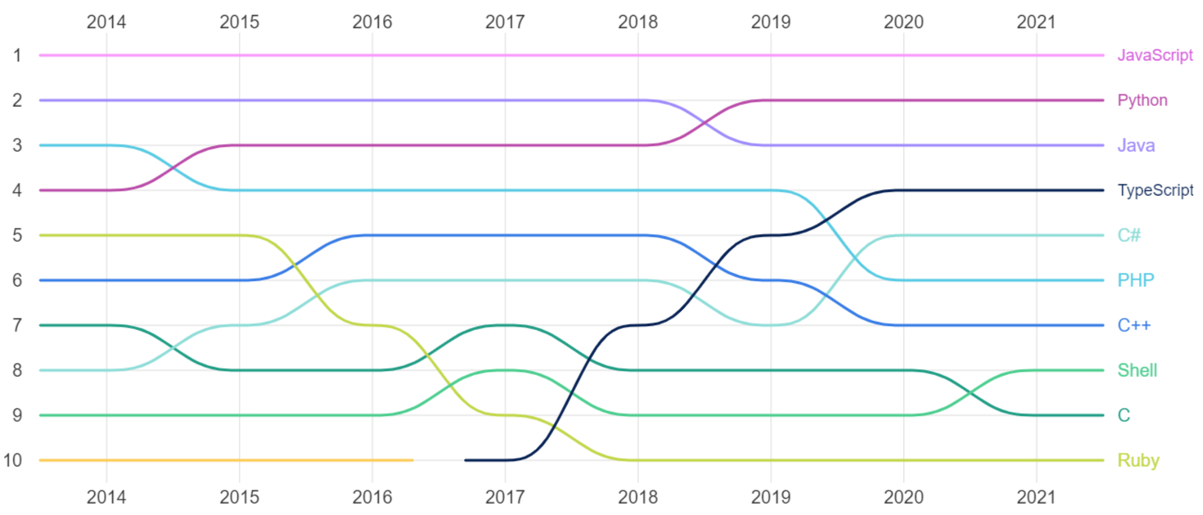


Figure 2.9: Ranking of Programming Languages used in GitHub Projects by the State of the Octoverse 2021 [32]
(JavaScript in pink, Java in purple and TypeScript in dark blue colour)

SlashData's **DeveloperNation**, (formerly known as Developer Economics) is the largest and most global independent research on developers including web developers, game developers and data scientists, amongst others. The data is owned exclusively by SlashData and no vendor, community or other partner. Their report is based on a large-scale, online developer survey. Twice a year they conduct a survey with approximately 40,000 developers in over 150 countries in more than eight languages. The 22th edition of 2022 was designed, produced, and carried out by **SlashData** over a period of ten weeks. This period was between December 2021 and February 2022. It reached more than 20,000 developers in 166 countries. For every survey, they dive into key developer trends in one quartal and show insights into leading research technologies in fields such as web, mobile, desktop, cloud, IoT, games, AR/VR and machine learning development. The main areas which they investigate include motivation, geography, area of interest and usage of programming languages. The geographic reach of this survey reflects the global scale of the developer economy. The following statistics are part of the **key developer trends for Q1 2022**. [33–35]

Comparing DeveloperNation to other statistics, the indices available from players like PYPL [2], Tiobe [36], Redmonk [37], Stack Overflow's yearly Developer Survey [38], or GitHub's State of the Octoverse [32] give great insights, but mostly offer **relative comparisons** due to their respective communities. DeveloperNation claims to be reflective of the global scale of developer economy, combining the experience and areas of involvement of developers. Therefore it offers a broad spectrum of impressions from different fields. [33]

In the 22nd edition of DeveloperNation it was confirmed, that JavaScript remains the **most popular programming language** for the tenth survey in a row with nearly 17.4 million developers using it. DeveloperNation explicitly sums up TypeScript and CoffeeScript under JavaScript (see Figure 2.10) [34]. Diving into developer demographics on a global scale, almost 65 % of developers are under the age of 35. The age group between 25 and 34 is the largest with 31,9 %. The results stem from a broad age spectrum. From young coders who are under the age of 18 to experienced ones over the age of 55. 79 % of respondents were male and 19 % female, excluding other options and those who did not specify their gender. [33]

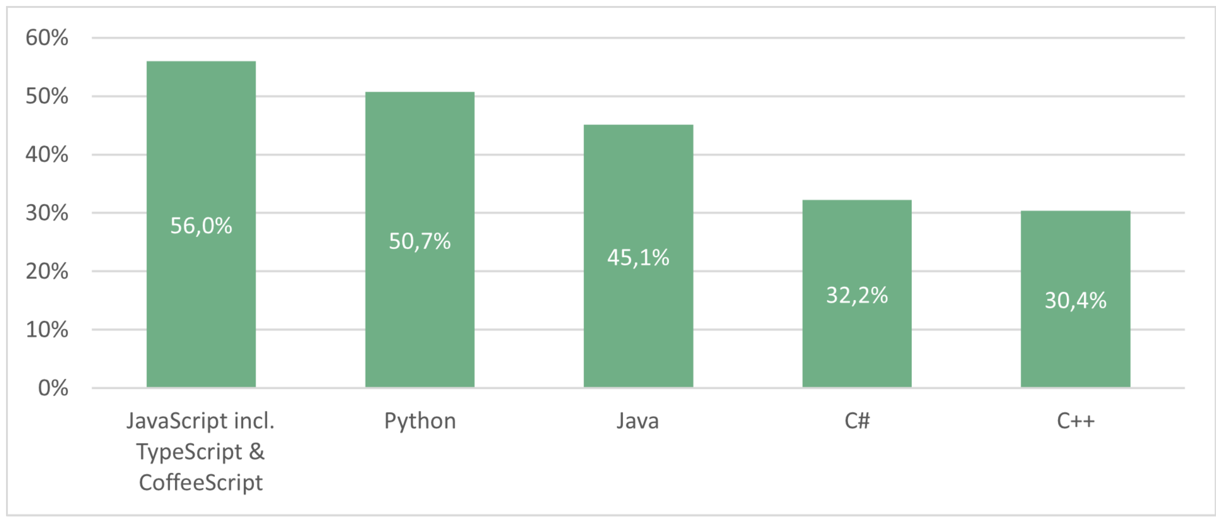


Figure 2.10: Top 5 Programming Languages used by Developers, adapted by Slash Data's Developer Nation Q1 2022 [33]

5.2 million developers joined the JavaScript community in the span from 2021 to 2022, being the highest growth in absolute terms across all languages with 17.4 million active developers in total (cf. 2.11). Even in software sectors where JavaScript is not among developers' top choices – like data science or embedded development who for instance prefer Python – about a fourth of developers use it in their projects. Reasons why JavaScript is so popular, explained by SlashData, are the attractiveness as an entry-level language but also as an addition to a skill set of a developer. [33]

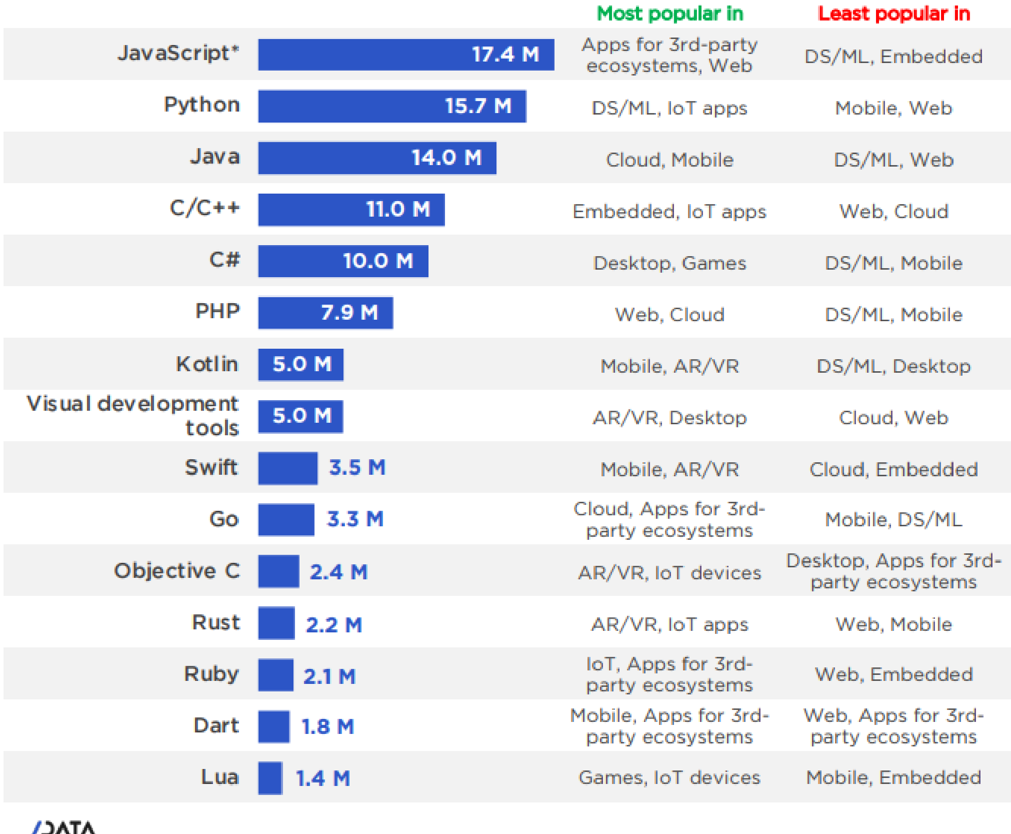


Figure 2.11: Size of Programming Language Communities in Q1 2022 by Slash Data's DeveloperNation [33]

Summarising the features and characteristics which made JS popular include [5]:

- *You don't have to acquire or download JavaScript.*
- *JavaScript already runs in your browser, on your computer, on your tablet, and on your smartphone.*
- *JavaScript is free to use for everyone.*

In the past years, JavaScript took another turn with Node.js which was introduced in 2009 (see 2.2.4.1) and the resulted possibility of using the language on the server-side. Besides being known as the language embedded in web browsers, it has been widely adopted for **server and embedded applications** since. This is where Full-Stack JavaScript Web Development was made possible and the common stack types, which will be later discussed, were born. [3, 7, 39]

JavaScript reaches popularity from different perspectives. There are a lot of use cases and developments, that will be brought up in this thesis.

2.1.5 Document Object Model (DOM)

When speaking of the structure of a website, there are different types of representations that need to be thought of. To get into the structural thinking, the basic principle is: “A web page is a document ” [31]. When a HTML document is loaded into a web browser, it becomes a document object [40]. Every element thereby becomes an object. To view the website and its document structure, there are three ways: Firstly and most commonly through a browser window, secondly through HTML source code and thirdly through the *Document Object Model* (DOM). [31]

According to the W3 Consortium [40], the Document Object Model defines the logical structure of documents and the way a document is *accessed* and *manipulated*. The document structure therefore shows the set of objects. This logical structure appears as tree-like, showing the relationship between them. The object-oriented sense is one of the reasons why the name *Document Object Model* was chosen. Every tree structure represents the elements of the web page, their nodes being the actual elements, attributes and text. [41]

Let’s take a look at an *example* which will be used for comparison throughout the chapters:
There is a simple button on a web page (see Figure 2.12). The web page has a heading ”Simple Button” and the Button with the call to action ”Press Me!”. When you hover over the button, a small black shadow is displayed. By clicking on the button, an alert box appears with the message ”Hello!”.

Simple Button

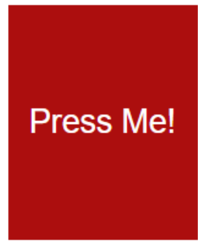


Figure 2.12: Simple Button (written in HTML) shown in browser

The DOM represents this button in the browser like this:

```
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <style>...</style>
5          <title>Button Simple</title>
6      </head>
7      <body>
8          <h1>Simple Button</h1>
9          <div id="button_div">
10             <button id="my_button" type="submit"
                   onClick="alert('Hello!')">Press Me!</button>
11         </div>
12     </body>
13 </html>
```

Figure 2.13: Simple Button shown in DOM as source code

The same structure can be shown as a simple tree structure, which is a common representation of data objects and their connections. Each HTML tag and its content is represented by a box. For instance, the body contains a heading (<h1>) and a box (<div>) with the actual button inside. The inscription of the button is the box on the bottom right with the text "Press Me". Identifiers and Values are *leaves* or *nodes* without children.

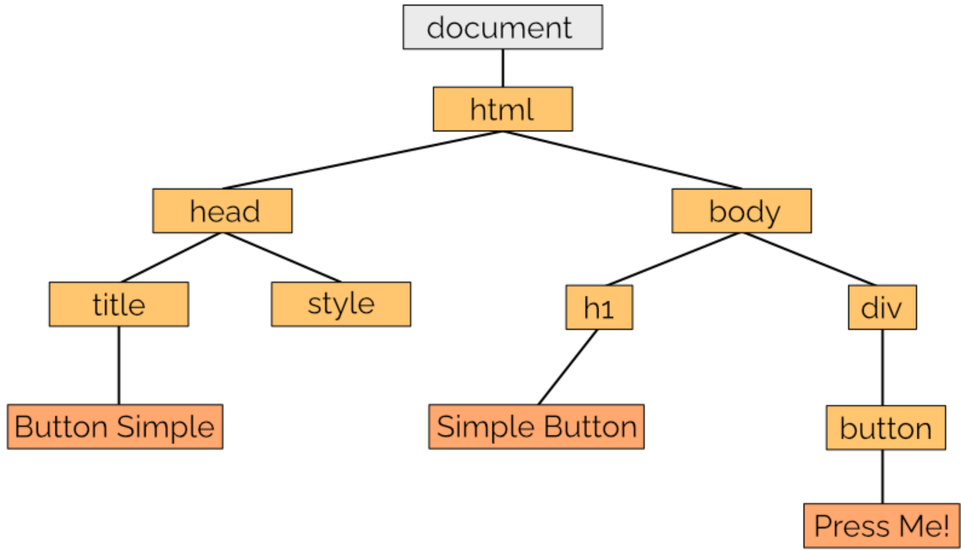


Figure 2.14: Simple Button shown in DOM tree structure

Working with the DOM helps **navigating** in the structure, **adding, modifying or deleting elements** very easy. Anything found in an HTML or XML document can be accessed, changed, deleted, or added (using the Document Object Model) [40]. It is the Application Programming Interface (API) for HTML and XML documents [41]. Every document object can be accessed with window.document or document. This feature is also shown in the tree structure, with document being the root node. JavaScript then works with the Document Object Model to **attach actions to different events** (for instance mouseovers, keydowns, drags and clicks) [31]. It is not only useful for representing, but also for working with CSS and JS to manipulate these elements. Once changed and saved, the page on the screen is updated and the changes are seen *live* [6].

When working with the DOM a certain level of caution needs to be at hand. If the HTML is not valid and semantic (also referred to as well-formed), the DOM doesn't represent the objects in the correct sense resulting in **nesting problems**. This leads to **false structure and manipulation issues**. For example a code problem can be that two leaves have two roots. This phenomenon is not possible in nature and in DOM-World. JavaScript Consoles and the console.log function in web browsers are therefore very useful for error correction to show whether something is wrong with the DOM. [31]

2.2 JavaScript Frameworks

> *One language, different technologies.*
> – AltexSoft [3]

Frameworks are a huge part in the lives of web developers. Every year their collection increases [17]. To get to know the existing and most relevant frameworks, this section will focus on some statistics and usages by the industry, deciding on which frameworks to look into. The frameworks with most relevance will be compared on a basic level through a simple red button example (as introduced in 2.12). Before getting to know frontend- and backend frameworks, the Client-Server-Relationship will be explained.

2.2.1 Client-Server-Relationship

Generally, a web application consists of three parts: the **Frontend, the Backend and the Database**. A Web Developer usually focuses on the frontend *or* the backend and is therefore called a Frontend- or vice versa a Backend-Developer. Both ends need to work inherently together, are equally demanding and contain different challenges. Full-Stack Development is essentially combining the technologies: Working on frontend and backend simultaneously (cf. 2.4). When talking about Full-Stack JavaScript Development, JavaScript is the language that is used on both client- and server-side. [42]

The frontend is everything that you perceive directly on the user interface (abbr. UI) and interact with. It is on the client-side. The Backend consists of APIs which are the connection between what the user types in, and what is saved in the database. It is mostly on the server-side. The database is the place where all data gets stored, typically managed through a database management system (DBMS). [42, 43]

Rendering can be done on the client- or server-side, which is known as client-side-rendering (CSR) or server-side rendering (SSR). There are also server-less applications with everything being stored in a cloud, or static site generation. These approaches will not further be discussed.

Figure 2.15: Frontend Development [44]

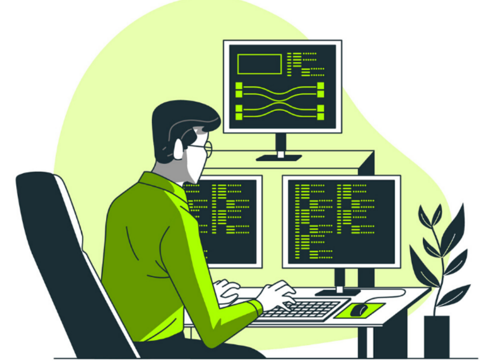


Figure 2.16: Backend Development [44]

2.2.2 Statistics & Usages

> *The choice of programming language matters deeply to developers because they want to keep their skills up to date and marketable. Languages are a beloved subject of debate.*
> – SlashData's DeveloperNation, 22th Edition [33]

Much like the choice of programming languages, the **choice of frameworks** is very much dependent on the skill level of the developer and the market requirements. Frameworks provide a standard development environment, helping to minimise time expense of development. Every year the number of frameworks grow, showing a diverse spectrum of possibilities. To know which frameworks to look into, a variety of statistics sorted by different communities and fields are helpful. These frameworks will be examined next. [6, 35]

Before diving into the statistics, a distinction between **frameworks and libraries** must be made. *Libraries* provide functionality in form of methods to use. In the case of Full-Stack JavaScript Development, one adds JavaScript functionality to projects to create some form of interactivity or modularity in reusing code. Very similar, a *Framework* helps to create entire applications by providing most functionality itself without specifically declaring which functionality to use. A *framework* for that reason is a more specific form of a library. It adds some of the possible functionality for usage. Because web development in itself is already rather complicated, adding a layer of abstraction helps keep an overview and makes the programming journey more pleasant. The borders between the two terms *library and framework* tend to become blurred and most statistics group both of them together, naming the two terms as synonyms. This thesis uses the names interchangeably as well. [6, 45]

Starting with the Web Community, one of the JavaScript-focused surveys is **The State of JavaScript** (StateOfJS), run by Sascha Greif and a team of open-source contributors and consultants. The 2021 State of JS survey ran from January 13 to February 2, 2022, with 16,085 responses. The average age group was 25-34 years and most developers (23.2 %) had 6-10 years of experience. 55.6 % of respondents had a higher education degree in a development related study field. [17]

Figure 2.17: State of JavaScript 2021 Logo [46]

Due to the global pandemic of Covid-19 and underestimated work on improvements, the 2021 survey was released in 2022. Improvements included amongst other things the correspondence between salary and experience levels, to get a deeper understanding of the **demographics**. Their goal was to identify upcoming trends in the web development ecosystem, in order to help developers make choices on which technologies to use. [47, 48]

Looking at the JavaScript frameworks, the StateOfJS2021 has the widest overview with over eleven **Front- and Backend Frameworks** listed in their survey [17]. Amongst the *Frontend-Frameworks* were React.js [46], Svelte [49], Vue.js [50], Preact [51] and Angular [52]. The *Backend-Frameworks* included Next.js [53], Express.js [54], Nest.js [55], Nuxt.js [56], Strapi [57] and Gatsby [58]. These are the frameworks which will continuously be named throughout different statistics and be investigated in this chapter. Some frameworks like Next.js and React.js build upon each other. This needs to be put into context for their usage.

Keeping an eye on the **Satisfaction-to-User-Count-Ratio** by StateOfJS, the Frontend-Framework **React.js** [46] has scored one of the best results. 83.66 % of regular users are satisfied with using React. The StateOfJS presents its results divided into four quadrants. As seen in Figure 2.18 React falls in quadrant 2, as a technology with high usage and high satisfaction. Thus it's a safe technology to adopt. Alongside, **Express.js** [54] is the dominant Backend-Framework with 87.5 % satisfaction rate and 10,775 users. Viewing the highest user counts, the next technology is on the opposing side in quadrant 3. **Angular** [52] shows a rather low satisfaction rate with 44.74 % combined with a low usage count of less than half of the respondents using it. According to StateOfJS, technologies with low usage and low satisfaction rates are harder to recommend. [17]

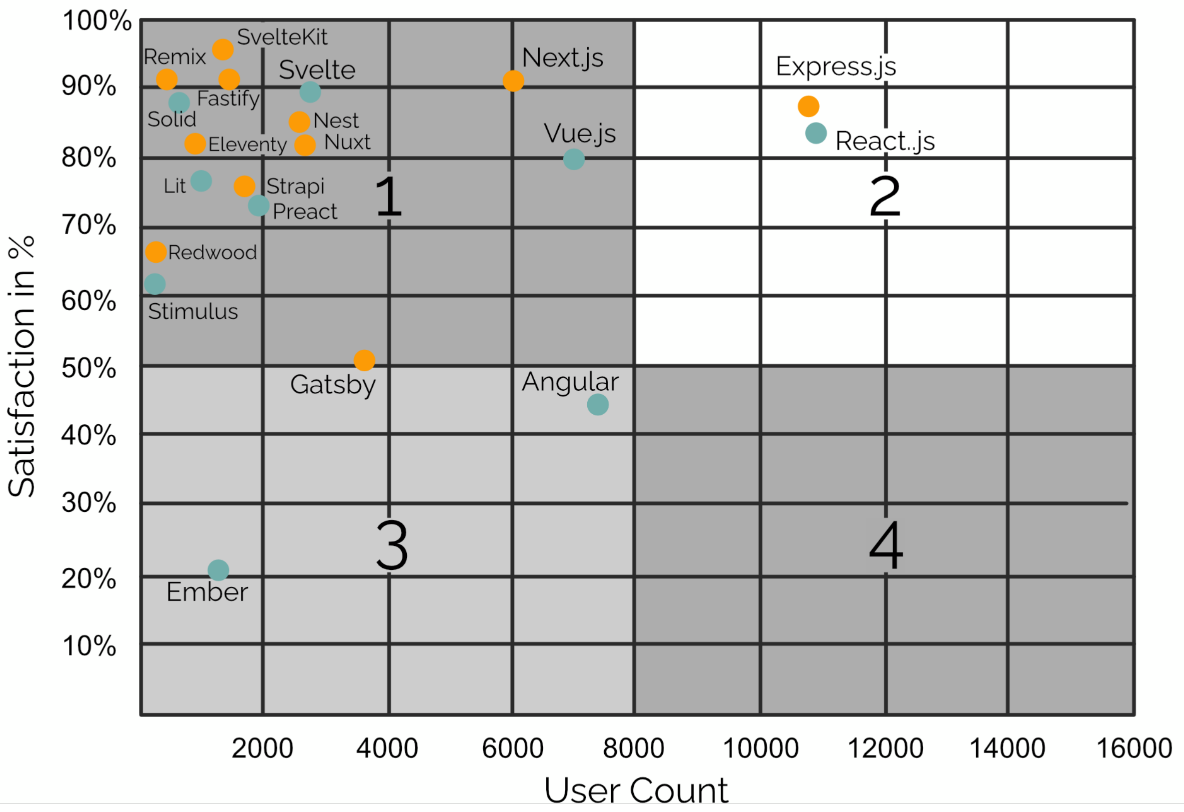


Figure 2.18: Satisfaction to User Count Ratio by StateOfJS 2021,
adapted from [17]

Clarification of quadrants
1: Low usage, high satisfaction. Technologies worth keeping an eye on.
2: High usage, high satisfaction. Safe technologies to adopt.
3: Low usage, low satisfaction. Technologies that are harder to currently recommend.
4: High usage, low satisfaction. Reassess these technologies if you're currently using them.
(Blue – Frontend Frameworks / Libraries; Orange – Backend Frameworks / Libraries)

Worth keeping an eye on are technologies with low usage and high satisfaction. The lower usage might be explained due to the beginning phase of their launch and the still dominant other technologies in use. **Vue.js** [50] scores the best user count numbers under this category. **Next.js** [53] is close in range with 91.21 % satisfaction from users. The popularity of a framework thereby is not only defined by the satisfaction of a single user, but the dimension of how many users perceive it as satisfactory. Some smaller, yet lesser known frameworks like Svelte [49] or Nuxt [56] have a hard time standing a chance against a framework most developers are familiar with and use. These smaller frameworks are mostly tailored to specific needs, making them less interesting for a mass of developers who need to keep up their working solutions with a broad framework. [17]

Exploring the **Usage of Frontend Frameworks** in more detail, the StateOfJS 2021 shows (in Figure 2.19) that React kept its place as most used for the last five years with a usage of 80 %. Directly following with 54 % usage is Angular and Vue.js closely with a usage of 51 % usage. These three frameworks are also the frameworks that developers are most *aware* of in StateOfJS with 100 % in the *awareness* statistic. In the *interest and satisfaction* statistic, **Svelte** [49] and Solid [59] have both risen to the top position. React and Vue.js held their position but Angular is radically dropping with only 45 % users being satisfied using it. As Angular is still very much in use, this shows a trend for the future, which makes it harder to recommend at the moment. [17]

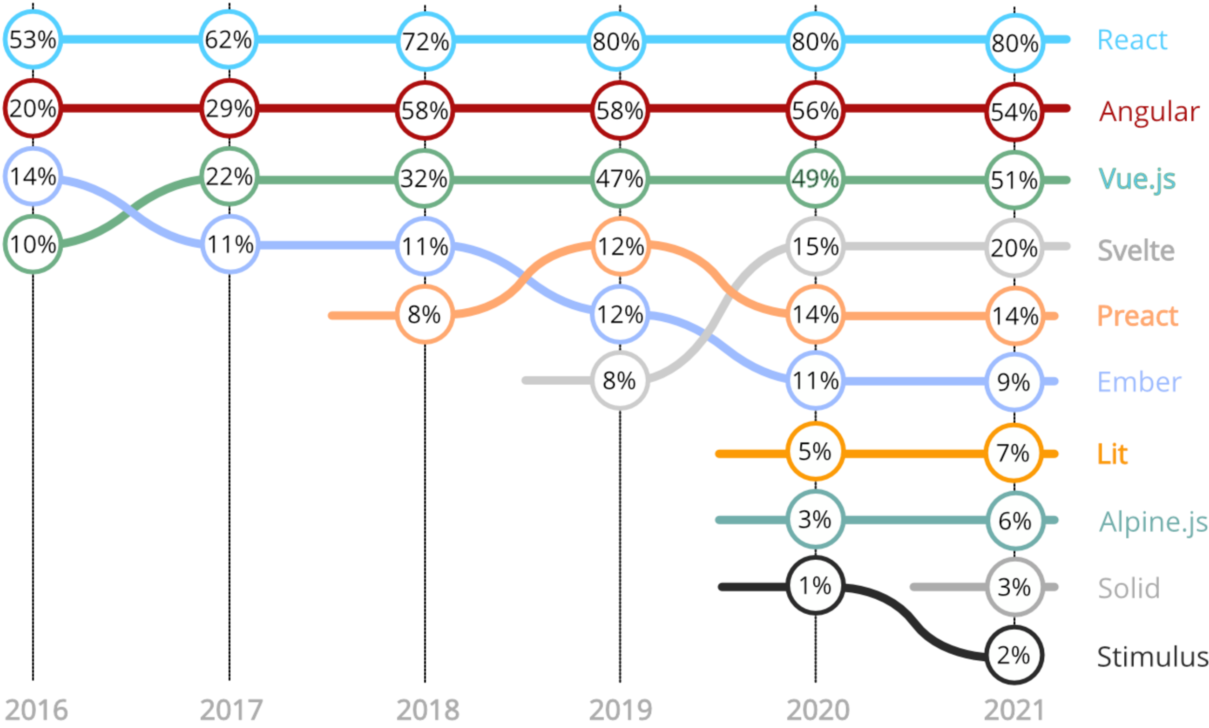


Figure 2.19: Ranking of the Usage of Frontend JavaScript Frameworks by StateOfJS 2021, adapted from StateOfJS 2021 [17]

Since JavaScript was introduced as a Backend Language with Node.js, there has been a major increase of developed **Backend Frameworks**. Further details on Node.js can be found in *Section* 2.2.4.1. As shown in Figure 2.20, Express.js has a very powerful position with 81 % of *usage*. In comparison, Next.js only follows with 45 % usage. 73.5 % of developers would use Express.js again. The satisfaction levels of Next.js and Express.js are both over 80 %, showing that they are very popular. In aspects of *awareness* Express.js, Next.js and Gatsby have the highest positions. Surprisingly, the rather new framework **SvelteKit** is only being used by 10 % but has an impressive satisfaction level of 96 %. Especially Svelte and SvelteKit thereby show very good prospects for future developments. [17]

Leaving the overview of the StateOfJS, it is time to focus on a statistic that focuses on a popular used address for code problems – a forum. Many developers start searching for code solutions on search engines, which eventually stack up to a collection of coding questions and answers as seen at **Stack Overflow** [60]. Taking a look into the collective knowledge of over 100 million people every month, Stack Overflow grants insights into problems and usages of different technologies. It thereby hopes to improve their community. [60]

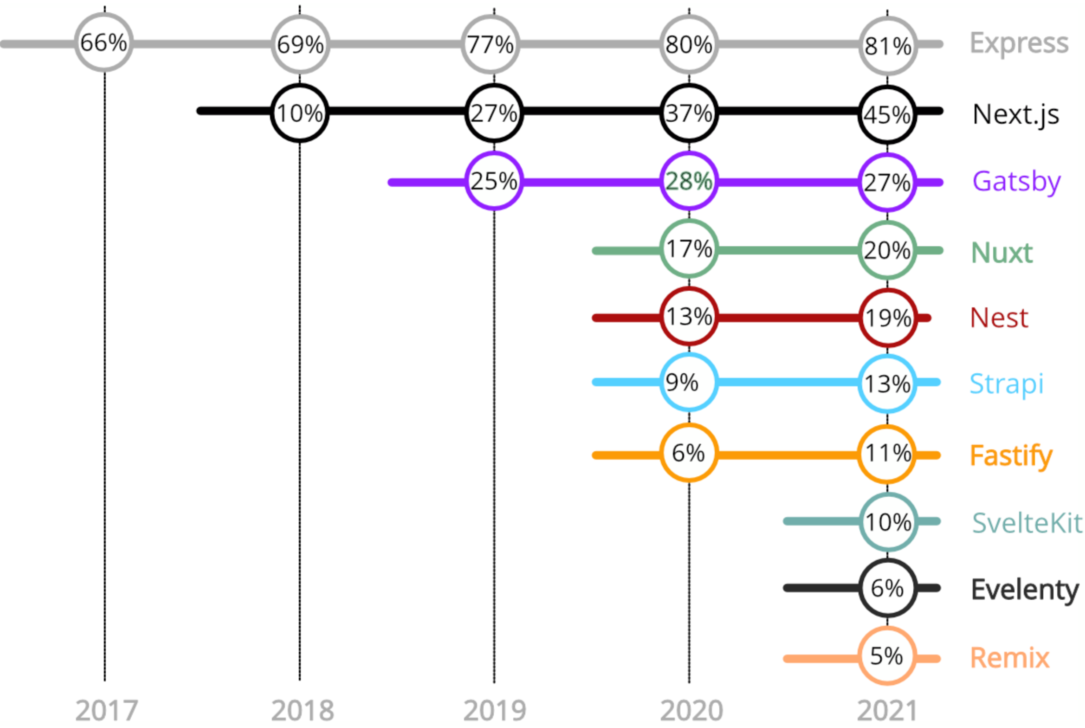


Figure 2.20: Ranking of the Usage of Backend JavaScript Frameworks by State of JS 2021, adapted from StateOfJS 2021 [17]

The 2022 **Developer Survey by Stack Overflow** took place from May 11, 2021 until June 1, 2022 with 73,268 software developers responding from 180 countries. The participants were mainly developers by profession (73.03 %) and students learning to code (8.61 %). This shows an impression of the current state in the job market and how students get an introduction to coding. With 18.88 % responses from the USA, 5.84 % from the United Kingdom of Great Britain and Northern Ireland, and 3.47 % from Canada, most English speaking countries are represented. Albeit with a focus on the USA, 7.52 % of the responses were from Germany. [61]

Continuing to look at all frameworks, React.js is the **most wanted web framework** in the fifth year (22.54 %). React is also *desired* by one out of four developers. The newcomer Phoenix is the most *loved* framework of the year 2022, closely overtaking Svelte's spot. Angular stays in the third year as most *threaded*, making it very unpopular amongst Stack Overflow Users. As many coding problems are exchanged, this might be an indication that Angular is harder to understand for those learning to code. As Angular upheld its percentage of usage, it is certainly interesting to investigate further. Especially, because **Angular** is used by more professional developers (23.06 %). [61]

Concerning popularity, it is a bad sign if a technology has no **support and/or community** (in a forum or blog) whom might answer coding questions. A developer then needs to focus on details, and might loose hours over a simple coding problem. Especially for beginners in coding, this is a crucial point when starting to use a framework. That is why the usage in Stack Overflow Survey showed useful insights. Furthermore, the majority of 46.82 % of the survey respondents identified as Full-Stack Developers. Thus providing this survey as a viable basis. [38]

On the backend side, **Express** scores with 22.99 % of usage in fourth place of **most common web technologies**. React.js has 47.12 %, Angular 20.39 % and Vue.js 18.82 % usage. Especially React.js developers show a diverse range of using interconnected frameworks and technologies, most notably with Node.js. [38]

Looking at the direction that JavaScript Development is following, it is incredible how many frameworks exist. The number of frameworks currently is so high, that 38 % of JavaScript Developers agree in the StateOfJS that the **ecosystem** is changing to fast. Furthermore 37.3 % of JavaScript Developers agree that building JavaScript Apps is overly complex right now. Compared to the last five years, using frameworks is an improved way of building JS apps. But the diversity and number of possibilities might lead a beginner developer to feel overwhelmed. The more important is it, to investigate important JavaScript technologies of today. [17]

As analysed with the statistics, **React.js, Angular and Vue.js** have shown their popularity in various ways. They, alongside Express.js as a backend framework, offer interesting uses for the frontend. They will be examined in great detail to understand how these technologies work and what benefits they bring in *Chapter 3*. To be able to judge the frameworks in later comparison, they will each be presented showing an overview of their characteristics.

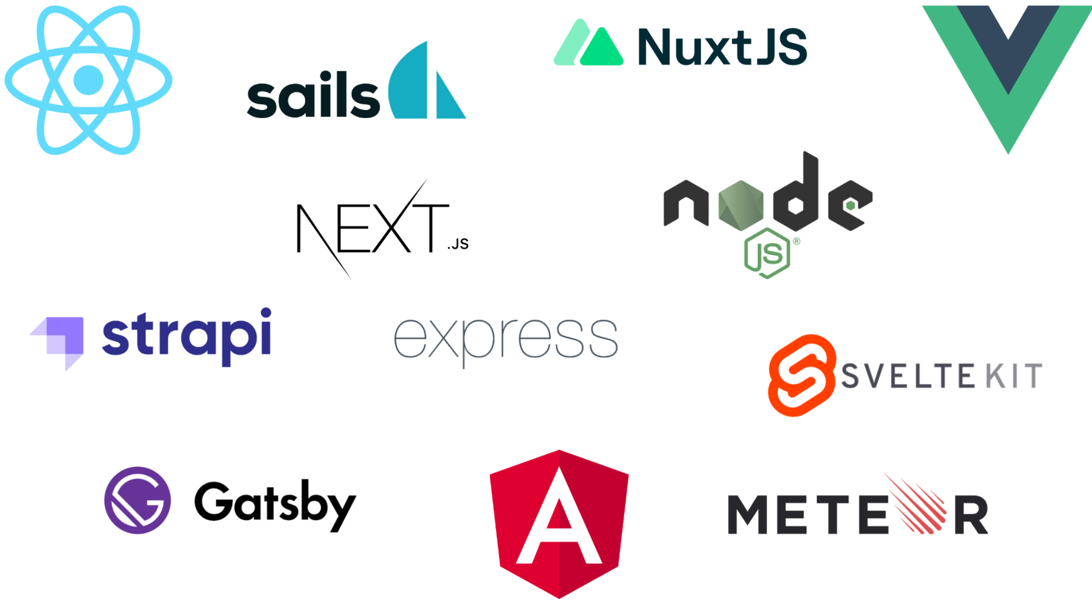


Figure 2.21: Overview of JavaScript Frameworks

2.2.3 Frontend

Frontend is what a user sees. The main goal is to give the user a smooth experience without any problems. The design therefore needs to be user-friendly. Usually Frontend-Developers work closely together with designers to create a good user experience. [62]

To help compare the frameworks, the thesis follows a simple **button example**. The detailed introduction of the button example can be found in *Section 2.1.5*. To shortly recap: The User Interface contains a heading and a simple red button, with the following properties:

- Heading "Simple Button" (Black colour)
- Button (Red colour) with the inscription "Press Me!" (White colour)
- Hover-Reaction: Black shadow around the corners (CSS-Box-Shadow)
- Click-Reaction: Alert-Box with the message "Hello"

At the time of this thesis' publication, the frontend frameworks are available in the following versions:

Frontend Framework	Version	Last Release	Initial Release	Source Lang	LTS
React.js [63]	18.2.0	June 2022	2013	JS	–
Angular [52]	14.0.4	June 2022	2016	TS	13.0.0
Vue.js [50]	3.23.33	April 2022	2014	JS / TS	3.2.37

Table 2.2: Versions of Frontend Frameworks

Note about Versioning: LTS stands for the **Long-Term Support** version of a software, indicating that the version receives support for a longer period of time with security releases and bug fixes. It is the most stable and supported version, recommended by software suppliers. [64]

2.2.3.1 React.js

A JavaScript library for building user interfaces. [46]

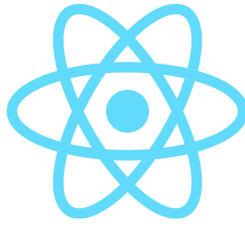


Figure 2.22: ReactJS Logo [46]

ReactJS is a JavaScript library developed by Meta (formerly Facebook) [65], which was released in May 2013. It is used to build interactive user interfaces and has grown to be the most popular Frontend Technology. It is used by prominent social media websites like Facebook [66] and Instagram. [67].

React.js claims to make it painless to create interactive UIs. They use a **component-based** approach with the components each having their own state, making them reusable for a UI. React is also able to render on the server using Node.js (see 2.2.4.1). A variation of React.js is *ReactNative*, which enables the creation of mobile apps platform-independently. [46, 68]

ReactJS states to be a JS library. **Libraries** provide functionality for working in JavaScript. As JavaScript with ES6 already includes much functionality (as seen in *Table* 2.1), the term library is aiming towards adding functionality such as component-based programming with JSX. In comparison, **Frameworks** help to create entire applications with the framework providing a great deal of functionality itself. With React the user may use as much as he needs. This reaches from simple interactivity to a complex React-powered app. [45, 69]

The essentials behind React

A dominant feature of React is, that it uses a **Virtual DOM** to keep track of changes that are made to components. Instead of re-rendering the DOM as a whole, it only updates the virtual DOM. This means whenever a change is made, the current DOM is compared to its original data structure through a diffing algorithm, which computes the differences. As shown in Figure 2.23, the detected changes then get updated in the real DOM. The DOM changes get batched up and applied in one go, it is more efficient than doing each change individually. This reduces expensive re-rendering when data changes because it only renders the components that really have changed. The user interface is then automatically updated any time its state changes. This is also called a **reactive** interface. [70, 71]

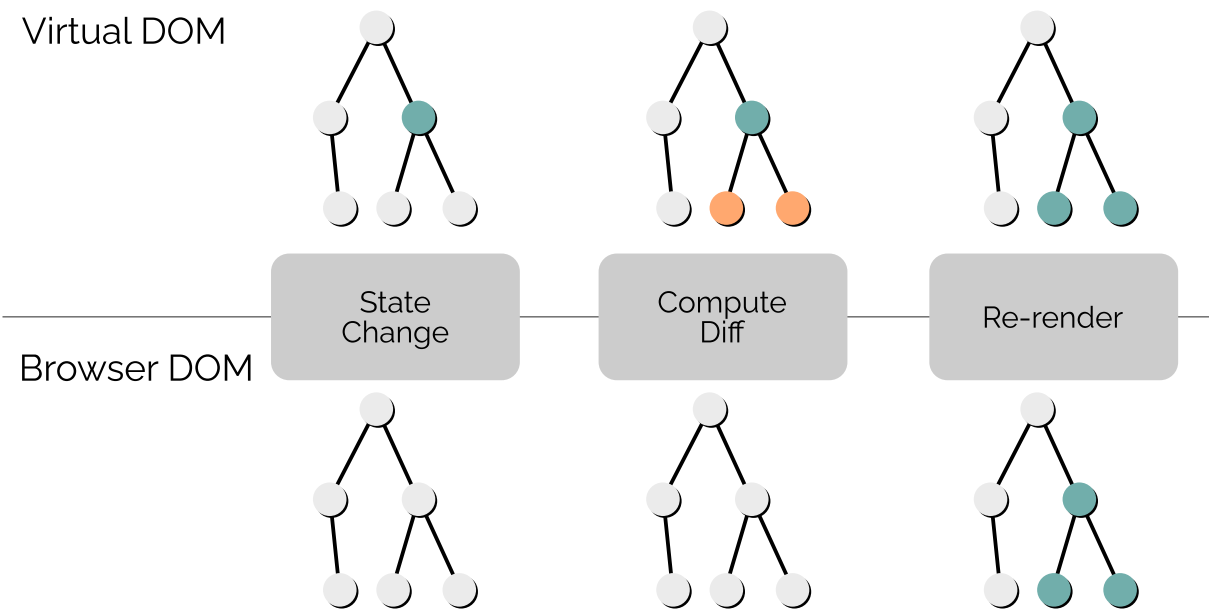


Figure 2.23: Virtual DOM Functionality, adapted by [72]

As JavaScript is an interpreted language, it tends to be slower than a compiled language. By now there are **just-in-time compilers**, which support faster development via line-by-line translation. This minimises the speed difference to compiled languages. [73]

React contains the building blocks of elements and components. Components are composed through elements. The **component-based** approach offers the opportunity, that every user interface element can be declared as a component. Code that is used multiple times is very convenient to declare as a component. For instance a simple view containing four buttons could be implemented with a button component. Just as in object-oriented projects, each button has certain attributes: colour, title, shape and behaviour. These would be attributes written in the *Button Component*. In this modular system it is very easy to reuse components and to create complex apps from small reusable pieces. Having these encapsulated components instead of templates enables React to pass data through the app, therefore keeping the state out of the DOM. [69]

JSX stands for JavaScript XML and is a **syntax extension** to JavaScript. It offers the possibility of using HTML elements in JavaScript. That being the case it is possible to write HTML in the same file, which already contains JavaScript code. The DOM of the website (cf. 2.1.5) get composed through JS elements, in particular the React classes. The HTML elements written in React therefore need to be converted into JavaScript elements, which is done by writing in JSX. It furthermore improves the readability of the code and makes it more understandable at first sight. If any JavaScript expression needs to be rendered, it has to be put into curly brackets. In this short example in 2.24, the name is the variable being inserted into the greeting "Hello, name!", hence the name being in curly brackets. [74]

```
1  const name = "John Doe";
2  const element = <h1> Hello, {name}! </h1>;
```

Figure 2.24: JSX [75]

JSX requires transpilation, because the browser doesn't understand JSX. That is the reason why a **compiler** is needed. Reactjs.org especially mentions Babel for ES6 and JSX code to be properly highlighted. Because JSX is closer to JavaScript than to HTML, React DOM uses **camel-case property naming convention** instead of HTML attribute names (e.g. class (html) is renamed to className (jsx)). JSX is optional and not required to use in React. It can be used by other libraries as well. JavaScript developers have much freedom when writing their code. They can use the language with HTML, XML, JSX and AJAX. [11, 13, 69]

A React app is composed of elements and components. To create a React **element** the method createElement( ) is used. The JSX gets compile to this method which returns plain JavaScript objects. These objects can then be manipulated in the DOM. To change the properties of the element, they can be accessed through a either a document-wide id or a class. Accordingly, the object can be get through element = document.getElementById(id). This in no change to vanilla JavaScript. IDs act as unique identifiers, to identify which elements have been changed, updated or deleted. Instead of re-rendering all the components, the id helps the diffing algorithm to only re-render the components that have changed their state. [70, 76]

A component can be declared by extending the React.Component class. It implements a **render ( )**-method, which is required to define a component. The method takes input data and returns what is displayed. It is written in a syntax similar to XML. It only returns one element, being a JSX element, a boolean or null, or another React component. This is generally a simple <div>. [46]

The simplest React component is a JavaScript function, also called **Functional Component** (Figure 2.25). It receives data input through a parameter called property ('prop'), and returns a React element. They return the JSX code which is rendered to the DOM tree. The functional components can be called through the function name with parenthesis (e.g. HelloWorld( )) or using the name of the functional component itself (e.g. <HelloWorld />) [75]

```
1  function Greeting(props){
2      return (
3          <h1>Hello, {props.name}</h1>
4      )
5  }
```

Figure 2.25: Functional Component in React [75]

Beside a functional component, a **Class Component** can be implemented (Figure 2.26). It requires the keyword class instead of function and the inheritance of the React.Component class. Using class components adds additional features such as local state and lifecycle methods. This structure is a ES6 class and is faster and simpler than a functional component. The keyword **this** needs to be used to refer to the correct DOM element. Components bring different advantages. They are small reusable pieces of code, each having their own independent structure. Creating a UI becomes much simpler, especially when exchanging components or re-implementing features. [70, 77]

```
1  class Greeting extends React.Component {
2      render(){
3          return (
4              <h1>Hello, {this.props.name}</h1>
5          )
6      }
7  }
```

Figure 2.26: Class Component in React [75]

To start using components a bottom-up strategy may be used, working the way up to the view hierarchy. At the top of the hierarchy is the single App component. A view therefore gets built gradually. Splitting up components into smaller components. The naming conventions of components are advised to be generic enough, so that they are applicable to more use cases than the context that they are build in. React recommends to build components, when a part of UI is used several times or if its complex enough to be on its own. [70]

Props, States and Hooks

Input data can be accessed by render( ) through JSX attributes, the so-called **props**. They are essentially how data is passed into an element and how to build a user-defined component. An example of the prop usages are already seen in the functional and class component examples. Thereby the attribute 'name' is passed over. To modify props, another concept is used.

React has the strict rule: All React components must act like pure functions with respect to their props. As an interactive UI changes over time, the prop can't change itself. To update the UI, the components ideally should update themselves. For that reason, a component should maintain an internal state. When it has, it is called a **stateful component**. The state allows the components to change the output. If the state data changes, the render( )-method will be invoked resulting in an updated DOM. States are like props, but they are private and fully controlled by the component. Components on that account are similar to JavaScript functions. They receive inputs through props and return React elements describing what should appear on the screen. Through props and states, an applications can exchange data an render the changes.

Because functional components don't have access to the props which are passed to them, they can be extended through the concept of **Hooks**, for instance useState( ) and useEffect( ). It is possible through **useState( )** to create the illusion of a state for the functional component. Examples of States will be shown in 4.5..

A handy feature that helps developing in React are the **React Developer Tools**, that are an extension of Chrome Developer Tools. They are a toolset used to develop and debug React apps. It additionally adds a DevTool to the Browser, making it possible to inspect the React component hierarchy alongside with the JSX code. Just as changing values in css the inspected selected props and states can be edited. [71, 78]

As indicated in 2.1.2 (Flavours), the usage of TypeScript is great for keeping an overview of the code with the help of strong types. It helps using the correct props by having information about what a component contains. TypeScript also helps with documentation and other developers understanding the code. TypeScript is already integrated in the 'Create React App' command npx create-react-app my-app --template typescript. It supports JSX and models the patterns like useState correctly. [18]

These are the basic building blocks of React. Further specifics and use cases will be explored in Chapter 3 (Comparison) and 4.5 (Full-Stack MERN Implementation).

Using React

React can be added to a project in various ways. The simplest way is to add a <div> with a unique id inside the body of an HTML page. A component then might be added through the script tag, as seen in Figure 2.27. Because this way requires an internet connection, it is simpler to integrate React by downloading the React package itself. Another popular way to create a React project is via command line with npx create-react-app my-app. It offers a full project setup ready to be used. Prerequisites are the installation of Node. [69]

```
1  <script src="https://unpkg.com/react@18/umd/react.development.js"
       crossorigin></script>
2  <script src="https://unpkg.com/react-dom@18/umd/react-dom.development.js"
       crossorigin></script>
3
4  <!-- Loading the React component -->
5  <script src="button.js"></script>
```

Figure 2.27: Adding a React Component to a website with simple code [75]

Button Example

As an exemplary creation of a React app, the simple button example of 2.1.5 was implemented. To create a React project, the setup was done with the command npx create-react-app my-app. The completely configured React project then was available to be modified by the developer. The application itself can be run by switching to the public folder of the project and executing the command npm run start. Further information about npm is in *Chapter* 2.2.4.1.

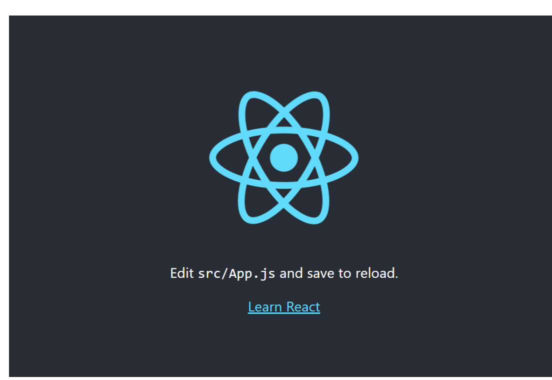


Figure 2.28: Create-React-App page before any modifications

Simple Button

Figure 2.29: Simple Button in React

Figure 2.30: React Application UI

As JSX combines HTML and JS, the code itself required some changes compared to Vanilla JavaScript. The file shown in Figure 2.31 is App.js, the main file where all components are composed to a full web application. It is in the source (src) folder of the project. The lines previously written in the body of the HTML page, now have to be surrounded by a <div> to only return one HTML element. The function call of 'hello( )' is a JSX function in curly brackets without parenthesis.

```
1  import './App.css';
2
3  function App() {
4    return (
5      <div className="App">
6        <h1>Simple Button</h1>
7            <div id="button_div" onClick={hello}>
8            <button id="my_button">Press Me!</button>
9        </div>
10     </div>
11   );
12 }
13
14 export default App;
15
16 function hello() {
17   alert('Hello!');
18 }
```

Figure 2.31: Simple Button in React (App.js)

To use the component-based approach, the Button was implemented as the component in button.js shown in 2.32. The class therefore needed to extend the React.Component class and return the UI elements in a container for the component to render only one JSX element.

```
1  import React, {Component} from 'react'
2
3  class Button extends Component {     // Extending the Component Class
4      render(){
5          return (                     // Returning only one JSX element
6              <div className="buttonContainer">
7                  <h1>Simple Button</h1>
8                  <div id="button_div" onClick={hello}>
9                      <button id="my_button">Press Me!</button>
10                 </div>
11             </div>
12         )
13     }
14 }
15
16 function hello() {
17     alert('Hello!');
18 }
19
20 export default Button;
```

Figure 2.32: Button as Class Component (button.js)

This leaves the file App.js only with a few lines (shown in 2.33). The component needs to be imported and added as the newly defined JSX tag <Button />. Having the button as a component makes it very easy to add another button to the application or reuse it in another application.

```
import './App.css';
import Button from './components/button'     // Import of Component

function App() {
  return (
    <div className="App">
      <Button />                              // Button Component
    </div>
  );
}

export default App;
```

Figure 2.33: Button included as Component (App.js)

Toolchain Recommendations

A JavaScript toolchain usually consists of a package manager like npm, a bundler like webpack (cf. 3.3.3) and a compiler such as Babel. To ensure the best developer experience – especially with large applications – ReactJS recommends elements to use in toolchains. The are presented to help with tasks like using third-party libraries from npm, detecting common mistakes early and optimising the output for production. [70]

An already discovered toolchain is using the Create React App command which is setting up a new-single page application. For server-rendered websites, React recommends using Next.js or for static oriented websites Gatsby. [70]

Mobile App Development: React Native

> *React Native is like React, but it uses native components instead of web components as building blocks.*
> – Reactnative.org [79]

As formerly mentioned, the framework **React Native** is very similar to React, and is also a viable option of a framework. The code still is written in JavaScript and JSX, but is rendered in native code. A developer as a consequence doesn't need to learn another technology to develop mobile applications. Native code is platform-independent, meaning the app uses the same native platform APIs other apps do. Therefore when using native components, Android as well as iOS apps can be created. To be able to create platform-specific versions of components on a single codebase, which can be shared across platforms, makes it very powerful. With its functionality, the satisfactory-to-usage-ratio in the StateOfJS (cf. Figure 2.18) shows that it is a JavaScript technology worth keeping an eye on. When speaking of powerful mobile and desktop frameworks, Electron [80] is a worthy ally, being slightly more popular. [17, 68, 79]

Through being developed by the large company Meta, React really had a **big breakthrough** being supported by many communities and tooling. Websites like Flipboard, Netflix and Airbnb also use React and are pushing React to become more popular as well.

- Short Recap of React -

- React is a JavaScript library.
- It uses a component-based approach.
- It has a Virtual DOM.
- It introduced the usage of JSX.

2.2.3.2 Angular

Angular is the name for the Angular of today and tomorrow.
AngularJS is the name for all 1.x versions of Angular.
– Angular.io [52]

Angular is a frontend framework developed by Google. Before continuing its description, a distinction to AngularJS needs to be made:

Figure 2.34: AngularJS – Superheroic JavaScript MVW Framework [81]

Figure 2.35: Angular – The modern web developer's platform [52]

AngularJS and *Angular* are different in their core. AngularJS was released in 2010 by Google, and is the older technology of the two. It is written in JavaScript. Angular (without JS) is a complete **rewrite** of Google's original technology, was released in September 2016 and is based on TypeScript (see 2.1.2). AngularJS is the name for all v1.x versions of Angular, while everything from version 2 onwards – up until the current version 9 – falls under Angular. Angular therefore is a much larger framework than AngularJS and applications needed to be completely rewritten. [3, 82]

Announced in 2018, Angular had its official EOL in January 2022 extended by the global pandemic. EOL stands for **End of Life** and means that a version (of software) is no longer supported by its vendor with official fixes, security releases or enhancements. When a version has no support, it might maintain deprecated code which is a key for security vulnerabilities and a risk for hacking. The AngularJS team for that reason encourages developers to **upgrade** their applications to Angular to benefit from productivity, scalability and improved performance. They especially made a description on how to change the technologies in projects. This thesis focuses on Angular in particular to compare the latest Technologies. [52, 64, 83]

Angular is an **application design framework and development platform** for creating efficient and sophisticated single-page apps [84]. It includes a component-based framework, a collection of integrated libraries and a suite of developer tools. Angular.io describes its usage on a scale from single-developer projects to enterprise-level applications. On a website with a list of websites made with Angular (and Angular.js), they feature the GitHub Community Forum, Overleaf, Adobe Fonts or PayPal Community [85]. [52, 86]

JavaScript considers types only when actually running the programme, and even then often tries to implicitly convert values to the type it expects [6]. As **TypeScript** is a JS flavour also referred to as a *superset* of JS, the languages do not entirely deviate from another but have a significant difference in their usage. Adding to the description of TypeScript in 2.1.2, TypeScript improves the developer experience with a great IDE support including error handling. Accordingly it offers cleaner, less-error code. Because TypeScript compiles to plain JavaScript, backwards compatibility is given. [82]

The essentials behind Angular

Just like React, Angular follows a **component-based approach** with components as the building blocks of the application. Components thereby include a TypeScript class with a @Component() decorator, an HTML template and styles. The decorator specifies the CSS selector, which is a template that instructs Angular what to render. HTML elements with this selector become instances of the component. It is combined with an optimal set of CSS styles. The template thereby can render multiple HTML elements, instead of just one JSX element like React. Components usually are written in camelCase, selectors and attributes in snake-case. [84]

```
import { Component } from '@angular/core';

@Component({
  selector: 'hello-world',      // Multiple lines in template with '...'
  template: `
    <h2>Hello, {{name}}</h2>    // JS variable in HTML element {{...}}
    <p>My first component!</p>
  `
})

export class HelloWorldComponent {
  // Component's behaviour
}

<hello-world></hello-world>     // Usage of the Component
```

Figure 2.36: Component in Angular [84]

Compared to React, Angular Components are composed through a **template**. They aren't limited to return only one DOM element (e.g. a <div>). Templates declare how the component renders. The styles thereby are defined inline or by file path in the template of the component. Instead of JSX, Angular extends HTML with additional syntax, in particular double curly brackets. They instruct Angular to interpolate the contents within them. The components updates whenever their state changes. Data can be passed to components as **properties**. They are indicated with square brackets, binding the property or attribute to a corresponding value in the components class. Event listeners are declared by writing the name of the event in parentheses and the method in quotes. An exemplary implementation can be seen in 2.37. The method which gets called by clicking on the button gets defined within the component class. [84]

```
<button
  type="button"
  [id]="myId"
  (click)="sayMessage()">    // Event in parentheses, Method in Quotes
  Press me!
</button>
```

Figure 2.37: Properties and Event Handling Example

Special functionality by Angular are **directives**. They are used to add additional behaviour to elements, for managing forms, styles and overall what the user sees. There are different types of directives: Directives used within a component template, attribute directives changing appearance and structural directives changing the structure of the DOM. They thereby listen and modify the behaviour of the HTML elements. Furthermore, they cleanly separate the logic and presentation of the application. [84]

Extending the basic HTML functionality, included directives are an own RouterModule and adding two-way data binding to a form element. They are also available as first-party libraries. The names of directives always include the prefix ng. This is because the name Angular comes from the **angular brackets** by the HTML tags < >. To make it memorable, the developers left a piece of the naming convention of A**ng**ular to components and directives with the prefix ng. [84]

Another feature are **dependency injections**, also called **services**. They let the developer declare dependencies in a central location without taking care of their instantiation. This makes testing very easy and the code very flexible because it is reusable. Angular strongly recommends using it as best practice, because it helps during development. A provided example on the official website is to use a logging function, which logs the number of a count to the console. The injection is indicated through @Injectable (providedIn: ' root ') and import of the Injectable class. Another class component then gets the Logger class assigned through the constructor of its own class, therefore using the logging function. A dependency to the logger class was injected. [84]

To render the UI, Angular provides **Ivy**, a compilation and rendering pipeline. Before there was the view engine. In version 9, Ivy is the default. It enables improved debugging, ahead-of-time (AOT) compilations and lazy loading of components. The concept is different from other concepts like the Virtual DOM. It also gets called **Incremental DOM**. The incremental DOM is used internally by Google. Every component gets compiled into a series of constructions. These instructions – contained by the templates – create the DOM and update it in-place when data changes. The instructions aren't just interpreted, they are the rendering engine themselves. In other words, the components are translated into regular HTML and JavaScript. [84, 87]

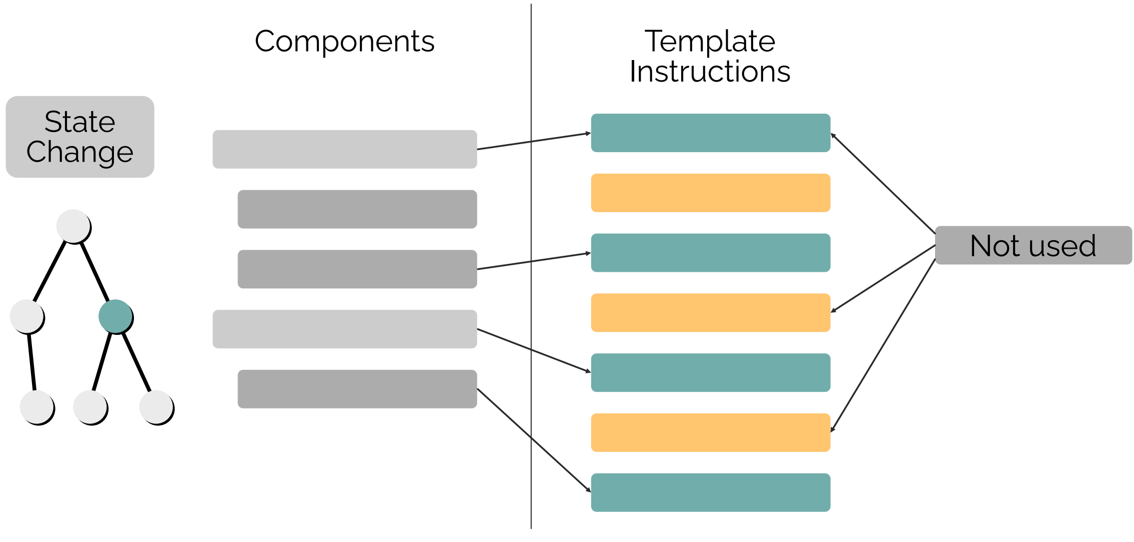


Figure 2.38: Incremental DOM Functionality, adapted by [87]

Using the incremental DOM concept, applications perform well on mobile devices, because the bundle size and memory footprints are reduced. This is done through the components reference instructions. Unused code is removed and not included in the bundle. This process is also referred to **tree shaking**. If an instruction isn't referenced, it will not be used. As shown in 2.38, the yellow template instructions would not be executed. At compile time, this saves a lot of time. [87, 88]

Using Angular

Angular offers an own implementation of a command-line interface, named **Angular CLI**. It is used as a tool to initialise, develop, scaffold and maintain Angular applications from the command line. It can be installed through npm install @angular/cli and accessed by entering ng in the command line. Offering around fifteen commands like serve for building an application with automatic rebuilding on file changes, generate for generating files based on a specific schematic or test for running unit tests, it proves to be very useful. Other useful commands are add for adding an external library or cache for cache statistics. The workflow using the Angular CLI is improved. [84]

As Angular is a **large framework**, installing the CLI really helps keeping the projects in line. Because it is a that large, it cannot simple be integrated into HTML page with a link. A simple Angular project can be created via ng new my-app. After a setup were the developer can decide on different setup scenarios, the CLI installs the necessary npm packages and other dependencies. In this workspace, the simple Welcome app is ready to be run with the command npm run start which executes the script ng serve --open. [84]

An additional complete **hands-on exercise** called 'Tour of Heroes' is provided on the Angular website. It guides through building a complete application, following the development process with CLI and important concepts. [84]

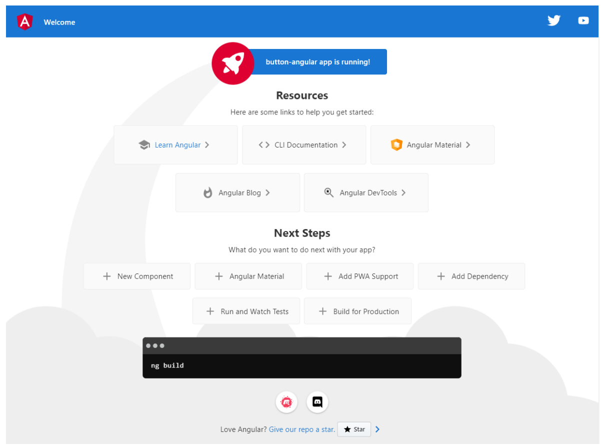


Figure 2.39: Angular Quick Start page before any modifications

Simple Button

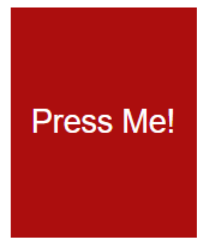


Figure 2.40: Simple Button in Angular

Figure 2.41: Angular Application UI

Button Example

To create the Angular project, the command ng new my-app was executed. As already described in the section before, the Angular CLI creates a new **workspace** with a Welcome app. The code features lots of comments, which is supporting the developer to find his way into the application. The main file is app.component.html with <app-root> as first component to load and the container for all other components. Adding the Vanilla JavaScript code to this file, is a possible solution to implement the Button example shown in 2.42. The next step was to implement the button as a component.

```
<div class="content" role="main">
  <h1>Simple Button</h1>
    <div id="button_div">
        <button id="my_button" type="submit"
            onclick="alert('Hello!')">Press Me!</button>
    </div>
</div>
```

Figure 2.42: Simple Button in Angular (App.component.html)

Building a component gets eased by the command ng generate component button in the Angular CLI. It generates starter files for the component and automatically updates the app.modules.ts file which features all declared components. Included are an HTML template determining the UI, a TypeScript file with components class handling the data and functionality, a CSS file defining the look & feel and a test file for possible later unit tests. The HTML file in 2.43 contains the template of the component. In this button example, the surrounding <div> with role 'main' needed to be removed. Instead of the onClick listener, the name of the click event in parentheses call the listener.

```
<h1>Simple Button</h1>
<div id="button_div">
    <button
        id="my_button"
        type="button"
        (click)="hello()">
        Press Me!
    </button>
</div>
```

Figure 2.43: Button Component (button.component.html)

The function which gets invoked when clicking on the button is defined in the components class in the TypeScript file 2.44. The style of the component was added through a styleURL and the file which was already included through the generate command.

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-button',
  templateUrl: './button.component.html',
  styleUrls: ['./button.component.css']
})

export class ButtonComponent {
  hello() {
    alert("Hello!")
  }
}
```

Figure 2.44: Button Component (button.component.ts)

The component was provided the selector ‘app-button’, which is how the component is included in the UI (cf. 2.45). This enables the easy reuse in other applications.

```
1 | <app-button></app-button>
```

Figure 2.45: Button included as Component (app.component.html)

Special Features

Angular offers lots of functionality and special features. The built-in support for Routing or HTTP Requests makes projects consistent and with being TypeScript-based very well manageable. Angular hence shows particularly a focus on supporting to build **large applications**. [52]

Although being developed by the large company Google, the switch from AngularJS to Angular made an indent into the user base with the introduction of many **new concepts**. Google is using its own technology on many different sites, as well as the websites PayPal, Forbes and Upwork. Angular is told to boost user and developer experience. [3, 82]

- Short Recap of Angular -

- Angular is a framework built on TypeScript.
- It is component-based with well-integrated libraries and its own CLI.
- It offers special features such as templates, directives and services.
- It uses the compilation and rendering pipeline Angular Ivy with an incremental DOM approach.

New Developments: Web Assembly

WebAssembly (abbr. Wasm) is a new type of code that can be run in modern web browsers. Introduced as a low-level assembly-like language with a compact binary instruction format, it runs with near-native performance. It has a stack-based virtual machine and provides languages like C/C++, C# and Rust with a compilation target. In other worlds, this means another language can be used for programming applications, adding further functionality and making it available to the JavaScript runtime. This opens the possibility of running other programming languages performant on the web with the security of a virtual machine as a safe environment to execute code. The ‘sandboxed’ execution environment can even be implemented inside virtual machines. [17, 89, 90]

Figure 2.46: WebAssembly Logo [91]

Wasm is also designed to run **alongside JavaScript**, allowing both to work together. Since its introduction in 2019, the developers using web assembly have doubled in size with 15.6 % in 2021, still growing. Its popularity seems to have made an evolution for browser plugins, being a specification instead of a plugin runtime. [17, 89, 90]

The programme code in Wasm is encoded into a a compact **binary instruction format**, which makes the programme size- and load-efficient. No unnecessary code is loaded and time-critical applications are suited to the processors configurations. Because the compile time is limited, Wasm offers a faster access to a website with a short loading time. WebAssembly is used within Angular to improve speed in particular. [89]

2.2.3.3 Vue.js

The Progressive JavaScript Framework.
An approachable, performant and versatile framework for building web user interfaces. [50]

Vue (pronounced like 'view') is JavaScript framework which was first released in February 2014. It is **declarative and component-based**, similar to React and Angular. Originally developed for rapid-prototyping by *Evan You* Vue states to efficiently help develop user interfaces. Vue is written in TypeScript and provides first-class TypeScript support. [93, 94]

Figure 2.47: VueJS Logo [92]

The framework is declared as progressive. It is possible to start with a core library and progressively add utilities. Due to its reactive, compiler-optimised rendering system it requires no manual code optimisation. This makes Vue projects **performant** as they scale and the code rendering fast. The Vue ecosystem scales between a library and a full-featured framework. Designed with **versatility** in mind, a developer might choose between an extensive collection of plugins. It is lightweight, which refers to few dependencies and being easy to add to existing software. [94, 95]

The essentials behind Vue
A core feature of Vue is the concept of **Declarative Rendering**. Just like Angular, Vue uses a template syntax, that allows the developer to extend basic HTML. Thereby the HTML output is declaratively described based on the JS state. Using the component-based approach, the main component style Vue features is the **Single-File Component** (SFC or *.vue file). It encapsulates the logic (JS), the template (HTML) and styles (CSS) into a single file (e.g. 2.48). [94]

```
1  <script>
2      export default {
3        data() {
4          return {
5            name: "John"   // Property which will be exposed on `.this'
6          }
7        }
8      }
9  </script>
10
11 <template>                // HTML template
12     <h1>My name is {{ name }}</h1>
13 </template>
14
15 <style scoped>  // scoped applies css to elements of the current component
16     h1 {
17         font-weight: bold;
18     }
19 </style>
```

Figure 2.48: Vue Single File Component (SFC) Example

SFCs can be authored through two different API styles to define logic. The **Options API** let the components logic be defined by an object of options, such as data, methods and mounted. Properties inside options are exposed with the keyword this, which is a pointer to the instance of the component. The **Composition API** defines logic through imported API functions with a <script> tag. The attribute setup makes Vue perform compile-time transforms that allow to use Composition API between templates and components. It therefore is more flexible to reuse. [94]

The functions data, methods and mounted used in Options API are **Lifecycle Hooks**. They can be called when the stage of the component's lifecycle is happening. An exemplary lifecycle of a component is setting up data observation, compiling the template, mounting the instance to the DOM and update the DOM when data changes. These functions give developers the opportunity to add own code at specific stages. [94]

The Options API is implemented on top of the Composition API and both share the concepts of Vue. To differentiate, the Options API is centered around a **components instance** (this). It aligns with object-oriented programming, which is useful for developers with that background. Furthermore, it is more beginner-friendly because some reactivity details are abstracted away. The Composition API on the other hand is centered around declaring **reactive state** variables in a function scope, and composing **states** from multiple functions together. The developer has free decisions to handle the complexity but requires knowledge of how reactivity in Vue works. The flexibility makes it more powerful for organising and reusing logic. [94]

Both API styles can be used. When beginning to learn Vue, Vue **recommends** to take the style which looks easier to the developer. For production and no build tools in use with low-complexity applications, they recommend Options API. For building full applications, a developer should rather got with the Composition API and SFCs. [94]

To add additional functionality such as reactive behaviour to the SFC and the rendered DOM, **directives** are used. They are indicated with the prefix v-. An example of a directive is v-on, which **handles events**, which can be shorted with a @ symbol. v-on:click="handler" is equal to @click="handler". The handler takes either inline JavaScript like the native JavaScript onclick handler or via a method handler. The method gets declared on the component. They get bound as event listeners in the templates. [94]

Another core feature is **Reactivity**. Vue automatically updates state changes and efficiently updates the DOM when it happens. It thereby also uses a **Virtual DOM** (cf. 2.23). Just like React it updates changes of the DOM at a faster rate. The templates get compiled to highly-optimised code. Vue is then intelligently deciding the minimal number of components to re-render. It then applies the minimal amount of DOM manipulations when the state changes. Vue also uses the concept of tree shaking, making its code fast. [94]

Using Vue

Vue also offers its own CLI like Angular. It enables the developer to run Vue commands in the terminal, for instance scaffolding a new Vue project, updating the version of Vue and configuring specific files. Also plugins can be added with a line of command. To install the Vue CLI the command npm install @vue/cli needs to be run in the terminal. It then can be accessed by entering vue. In comparison to the Angular CLI, Vue offers only three basic commands: for creating a new project, installing a plugin and print debugging information. [96]

To be able to use Vue, a build step is not necessary. It can even be used via plain JavaScript without a build step. But using a build step allows the developer to use SFCs. The official build tool for Vue is **Vite**, a lightweight and extremely fast frontend tool (cf. 3.3.3). To create a Vue project, Vue offers the npm init vue@latest command. This installs and executes create-vue, which is the official Vue project **scaffolding tool**. [97, 98]

The quick start tool lets the user choose options on how the project needs to be set up directly in the command line (shown in 2.49). The user can simply enter 'y' for yes and 'n' for no or enter a value (such as the name) that is appropriate for the project. In these options included are the project name, whether TypeScript should be used or if JSX support is needed. Furthermore it can automatically add some frameworks like the Vue Router for Single Page Application, Pinia for state management, Vitest and Cypress for testing purposes, ESLint for code quality and prettier for code formatting. [97]

```
nadine@nadinedevice:~\mydirectory  vue init vue@latest

Vue.js - The Progressive JavaScript Framework

✓ Project name: … button-vue
✓ Add TypeScript? … No / Yes
✓ Add JSX Support? … No / Yes
✓ Add Vue Router for Single Page Application development? … No / Yes
✓ Add Pinia for state management? … No / Yes
✓ Add Vitest for Unit Testing? … No / Yes
✓ Add Cypress for both Unit and End-to-End testing? … No / Yes
✓ Add ESLint for code quality? … No / Yes
? Add Prettier for code formatting? › No / Yes
```

Figure 2.49: Vue Quick Start Setup Options

With these options, in just a few minutes, a whole project is created. Especially code quality and formatting are wanted features, that many developers add by choice already. After the scaffolding, the project can be started with the command npm run dev executing the script. [97]

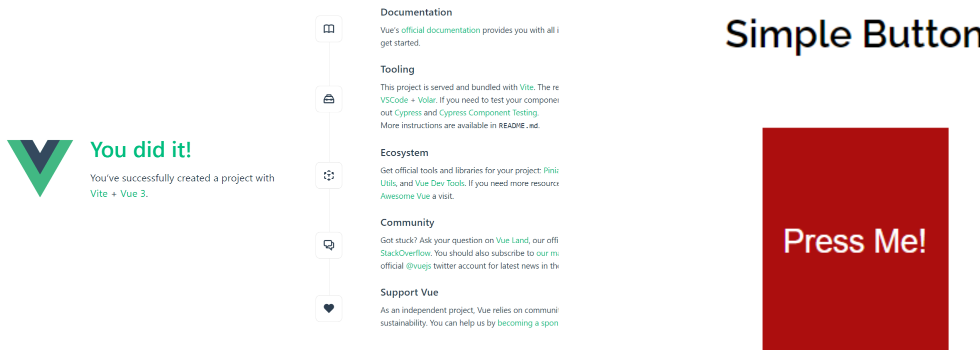


Figure 2.50: Vue Quick Start page before any modifications

Figure 2.51: Simple Button in Vue

Figure 2.52: Vue Application UI

Button Example

To start with the Vue project, the command npm init vue@latest was executed. This creates a new Vue project, custom to the users needs. In the case of this simple project, simple JavaScript and the Options API was chosen. The name of the component is returned by the script when being build.

```
<script>
    export default {
        data() {
            return {
                name: 'CustomButton.js'
            }
        },
        methods: {
            hello() {
                alert('Hello!')
            }
        }
    }
</script>

<template>
    <h1>Simple Button</h1>
    <button id="my_button" @click="hello()">Press Me!</button>
</template>

<style scoped>
    /* Button CSS */
</style>
```

Figure 2.53: Button Single File Component (CostumButton.vue)

The Button component gets imported into the root .vue file and added to the template with the specified name in angular brackets <CostumButton /> as a single html tag. It can be reused in other components as well.

```
<script setup>
  import CostumButton from './components/CostumButton.vue'
</script>

<template>
  <CostumButton />
</template>

<style scoped>
    /* App CSS */
</style>
```

Figure 2.54: Button included as Component (App.vue)

Featured Scripts and Frameworks

Components can be embedded in any HTML page. It is also possible to use Vue as a standalone script file where no build step is required. This is especially useful for apps where the frontend isn't complex enough to require much logic. Vue thereby shows similarity to jQuery. An alternative and rather new distribution is **petite-vue**, that is optimised for progressively enhancing existing HTML. Unsing only small amout of interactions, it has less features than Vue and is extremely lightweight. [99]

Another version of Vue is **Vitepress**. They are extremely popular. For higher-level full-stack development, the framework **Nuxt** is useful. It automatically generates the router configuration and code-splitting for all the routes. This eases the development process [17, 56, 97]

Vue combines different approaches to a very progressive and flexible framework. [3, 82]

- Short Recap of Vue -

- Vue is a progressive framework built on TypeScript.
- It uses a component-based approach with its own CLI.
- It is declarative.
- It uses a Virtual DOM.

Mentionable: jQuery

> *A fast, small, and feature-rich JavaScript library.* [101]

As seen in the statistics and usage section (2.2.2) there are many other JavaScript frameworks. A very important and still very known framework is jQuery. JQuery was introduced in 2005 and was the first JavaScript library of many web developers, beginning to be used around 2006. It helps manipulate the DOM, event handling, animation and makes AJAX much simpler. It revolutionised web development through having **query selectors** for HTML components. Since then all browsers adapted this feature. [3, 100]

Figure 2.55: jQuery Logo [100]

It self-claims to have changed the way millions of people write JavaScript but is now known as becoming obsolete, opening other technologies – such as React, Vue or Angular – the door to becoming the new standards. [3, 100]

2.2.4 Backend

> *JavaScript on the server*
> – StateOfJS 2021 [17]

The backend refers to the server-side of the application. It includes everything a user cannot see directly on the user interface and takes care of the functionality of a website. [62]

The distinction between frontend and backend frameworks is not always easy as they are fluent and depend strongly on the type of technology stack (cf. 2.3). The base of most JavaScript Backend Frameworks is Node.js. Therefore is why this chapter starts with this fundamental technology and ends with a look into Express.js and the database MongoDB, which are commonly used in JavaScript technology stacks. [17]

At the time of this thesis' publication, the backend frameworks and the database MongoDB are available in the following versions:

Backend Framework	Version	Last Release	Initial Release	LTS
Node.js [39]	18.4.0	June 2022	Mai 2009	16.16.0
Express.js [54]	4.18.0	April 2022	November 2010	–
MongoDB [102]	5.0.9	May 2022	February 2009	–

Table 2.3: Versions of Backend Frameworks

2.2.4.1 Node.js

> *Node.js is a JavaScript runtime built on Chrome's V8 JavaScript engine.* [39]

Figure 2.56: NodeJS Logo [103]

The possibilities of working with JavaScript amplified when Node.js was introduced in 2009 as a registered trademark by the **OpenJS Foundation** [104]. The OpenJS Foundation is made up of 32 open source JavaScript projects, including Electron [80], ESLint [105], jQuery [101], Node.js and Webpack [106]. Their mission is to support the growth of JavaScript and web technologies by providing a neutral organisation to host and sustain projects, as well as collaboratively fund activities that benefit the ecosystem as a whole. Amongst the foundation members and supporters are companies like GitHub, Google, IBM, Microsoft, Netflix and Uber. The OpenJS Foundation creates a great sense of community with many forums and possibilities to interact with web developers around the world. [104]

Node.js (or simply Node) is a cross-platform **runtime environment**. Within a runtime environment a programme is executed. This is needed by JavaScript. In the case of Google Chrome and Node.js the engine beyond the runtime is **V8** [107], which in turn is based on the ECMA-262 language specification. V8 is Google's open source high-performance JavaScript and WebAssembly engine, written in C++. It provides JavaScript APIs for different applications. [90, 107]

A simple JavaScript runtime environment in comparison to Node is the browser on the client-side. The **JavaScript API** provided by browsers, focuses on *DOM interaction and data management*, while the runtime in Node.js focuses on providing APIs for *interacting with the file system, the network or the database of the operating system*. The language JavaScript in all cases stays the same, there are just different APIs beneath its processing. [90, 107, 108]

With this functionality, Node.js opened the **opportunity** of running JavaScript code on the server and has since been widely adopted for server and embedded applications. An example of application is to build a simple web server with JavaScript or to use Node for an Internet of Things (IoT) application. The StateOfJS 2021 shows that it is the most regular used runtime for JavaScript with only 71.5 % usage – even before the browser with 68.4 % usage [17].The achievement of executing JavaScript without a browser thus enabled to create **Full-Stack Applications**. According to Stack Overflows Developer Survey 2022, Node is the most common web technology, alongside React with 47.12 % usage. It is used by professional developers as well as those learning to code. [108]

Node.js is currently on version 18.4.0 with its last release having been in April 2022. The recommended **Long-Term Support** (LTS) version is at 16.5.1 and was lastly updated July 14, 2022. With node.js the LTS release status guarantees that critical bugs will be fixed for a total of 30 months. Major versions are released every six months. [109]

Some of the **characteristics** of Node are that it is asynchronous, non-blocking and event-driven. Many connections thereby can be handled concurrently, although JavaScript is executed single threaded. This is because the JavaScript callback-functions can be executed by the event loop *after* completing other work as **non-blocking**, also called asynchronous operations. Node.js states that it is free from dead-locking a process (plainly said the waiting time between processes), because there are no locks which could block a process (non-blocking). On that account Node is a great system to build high-scalable network applications with high speed and performance. [39, 71]

The characteristic **event-driven** means that if an Input / Output (I/O) operation is happening, the single event-driven thread continues executing the code managed in a queue-like mechanism (see 2.57). The callback for the I/O operations are then fired when the responses come back. As it is designed for streaming and low-latency in mind, it is great for **real-time applications** that need constant updates when information changes and quick execution. Many frameworks are built to be used with or on top Node.js. For instance Express.js, which is explained in the following section. [39, 71]

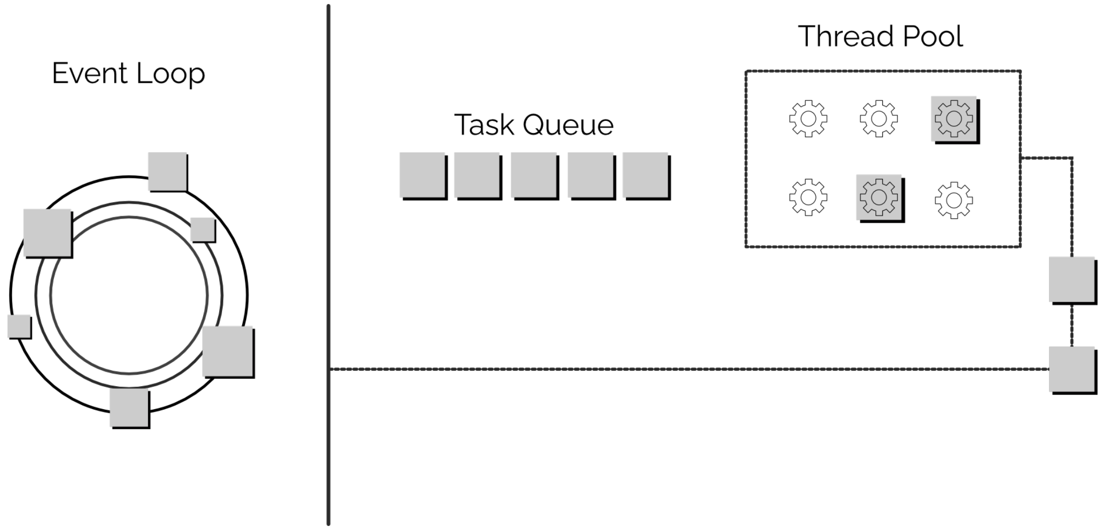


Figure 2.57: Node.js Event Loop and Task Queue, adapted by [3]

Being single-threaded is a disturbing factor for highly computational processes. This is because blocking the single thread happens when a lot of actions are being taken. Node therefore added support for the possibility of multi-threading (version 10 and higher). This could be interesting for machine learning operations like neural networks. [3]

Node.js can be used as a command-line tool when entering the command node. The Node.js **console** comes with a virtual, interactive environment called REPL. REPL stands for **Read-Evaluate-Print-Loop** and is indicated in with >_ in console. The principle is: The console waits for the command (read), executes it (evaluate), prints feedback (print) and repeats its behaviour (loop). [110]

As Node has offered many possibilities, more and more software **companies** are integrating Node in their projects. Especially, big players such as LinkedIn, Microsoft, Netflix or PayPal which require high performance have it in use. [71]

Side comment: Another popular runtime for JavaScript is **Deno** [111]. The inventor Ryan Dahl – the project leader of Node.js – specifically wanted to avoid the design mistakes of Node that had been made. The design of deno additionally concentrated on being a runtime for the flavour TypeScript in particular. As seen in 2.1.2 this is a big advantage Node might need adapt to. [111]

npm – Node Package Manager

The Package Manager npm was founded by the company npm, Inc. in 2014 and was acquired by GitHub in 2020. It is the package manager for Node.js helping JavaScript Developers share modules, reuse own modules and improve their development experience [112]. The registry is a **collection of public packages** of open-source code for Node.js applications, self-claiming to be the center of JavaScript code sharing with more than one million packages. npm has been declared as a fundamental tool for professional developers by Stack Overflow with 65.17 % usage of all respondents [38].

It adds extensive modularity to all Node projects. Although most core modules are stable and well-tested, most smaller modules in the registry don't have good structure and are badly documented. This needs to be kept in mind when adding modules to a project. [112, 113]

Besides being the public collection of packages, npm is the **command line client** that allows developers to install and publish packages. It is used in many projects, and is included in the Node.js installation as a starting point (see Figure 2.58). [112]

npm commands & package.json

Important npm commands are npm install <module> (for installing a npm module), npm ls (to list what modules are installed in a project), npm update <module> (to update a module) and npm uninstall <module> (to unistall packages). [71]

An important file is the **package.json** file in the root directory of a project. It is like a manifest, containing metadata information to npm about the project and what it needs to do. This is the key to giving another developer the project, because it will tell npm what modules to install in which version. The file can be created manually or automatically by npm via the command npm init. When executing npm install, it will cause npm to read the package.json file and automatically install all the dependencies. This file should not be deleted, as npm needs to know how to do its job. [71]

Lastly, npm start the package.json calls the script which is under the attribute 'start'. This line could look like "start": "node server.js" to open the script which is in the file 'server.js'. [71]

Useful npm modules

With npm you can install all JavaScript Flavours, some frameworks or a module developed by another developer. Here are some useful npm modules, which might be helpful for a project using JavaScript and many modules. [112]

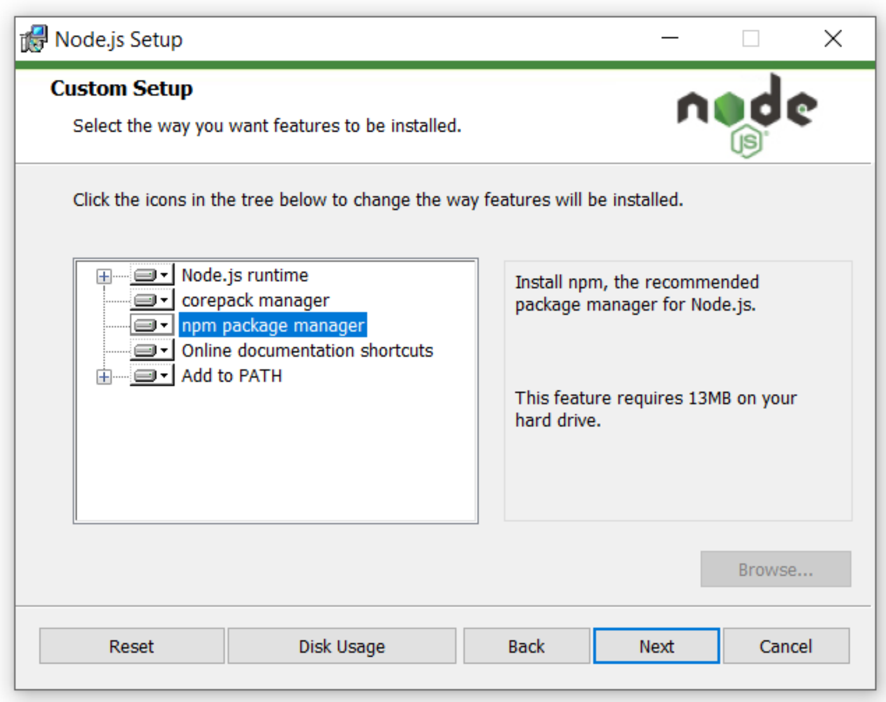


Figure 2.58: Node.js Installation Process including the npm package manager

Node is only capable of executing JavaScript. With using TypeScript in the Node console it is easier to handle larger scripts. For that reason **ts-node** was written. It is a **TypeScript** execution engine and REPL (Read-Evaluate-Print-Loop) for Node.js. It transforms TypeScript into JavaScript, enabling you to directly execute TypeScript on Node.js without precompiling. To enter the TypeScript REPL mode, the command npx ts-node must be executed. Typechecking is also possible via the flag ts-node --typechecking. [114]

Because node version management and dependencies are rather complicated and updates are easily forgotten, the interactively managing tool **n** can be used [115]. Offering an overview of the versions already downloaded or simply changing the version which is currently in use – these are only some of the use cases. The managing tool is listed in the recommended tool list of the State-OfJS 2021 [17].

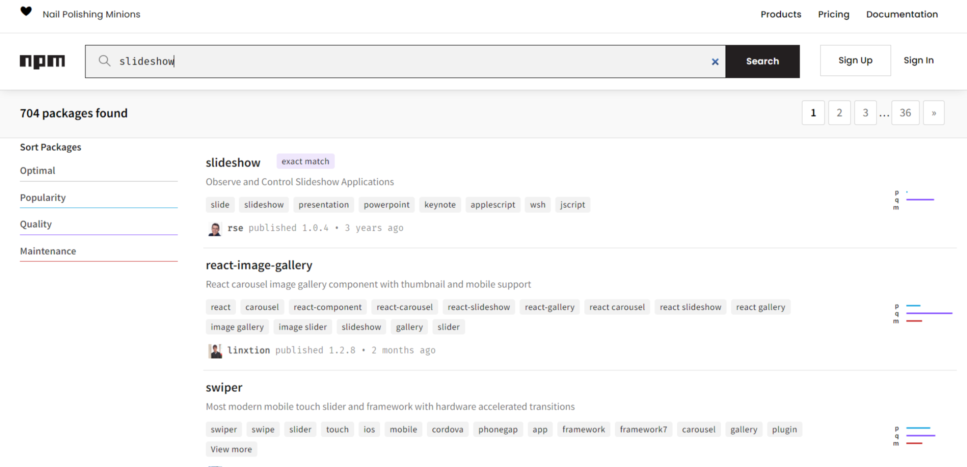


Figure 2.59: npm search of the keyword *slideshow* [116]

2.2.4.2 Express.js

> *Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications.* [54]

Figure 2.60: Express Logo [54]

Express.js is a backend web framework, that sits on top of Node. Being a **minimal and flexible** application, it is a combined set of tools managing the interaction between the client side and the data model. That makes its overall goal improving the data transfer. While providing a set of features for web apps, it is used to create robust APIs, manage HTTP requests and render routing. [3, 54]

As the frontend frameworks can't use the backend directly, they have to use Application Programming Interfaces (APIs). **APIs** provide functionality to exchange data via a machine-readable interfaces. They are easiliy integrated into an application and provide different functions to different websites. APIs furthermore ensure a secure connection for exchanging data and interact with components from frontend to backend. The frontend retrieves data from the web server using the API. They therefore simplify communication and exchange of data. [71]

As Node.js doesn't offer the functionality to speak via an API, Express.js needs to be used. It facilitates the data control by handling the data like **JSON objects** (cf. 2.64). Because it is a minimal framework, it delivers excellent performance on top of everything else. As it is un-opinionated, it doesn't constrain the user and offers full freedom of functionalities. It is especially useful for using REST (Representational State Transfer) APIs. 87.5 % of users are satisfied with the usage of Express. Together with React.js it is a technology that is safe to adopt, according to StateOfJS. [17, 54, 71]

REST stands for **Representational State Transfer** and is a paradigm that defines how information is exchanged over HTTP. Requests are made to invoke a specific functionality that is then applied on the data. The functionality is specified within the request. The most common requests are GET to retrieve, POST to create, PUT to edit or update and DELETE to delete data. [71, 117]

In order to handle requests from the client (i.e. a browser) a **web server** needs to be configured. The web server takes the HTTP requests from the client and passes it to the database, where the response is handled. With Express, a web server and their requests can be configured easily through the process of quickly creating efficient and robust APIs. It provides the functionality to access the database MongoDB through Node. [43, 118]

Express can be **installed** via the command 'npm install express'. The dependency is automatically saved in "package.json". The server should be running at all times in order to create a stable connection, where the user can send requests to and receives responses. The Express server is started by executing the command node server.js. It then is available on "http://localhost:3000/". [54]

```
1  const express = require("express")       // Requiring the usage of Express
2  const app = express()                     // Creating an instance of Express
3
4  app.get('/', (req, res) => {     // GET-Request on the root route (/)
5    res.send('Hello World!')       // Responding with "Hello World!"
6  })
7
8  app.listen(port)            // Server listens onPort3000 for connections
```

Figure 2.61: Creating an Express Server (server.js) [54]

The way an application sends responses to requests from a client at a particular endpoint is called **Routing**. An endpoint generally is a URI or path and a specific HTTP request method such as GET or POST. For instance, any time a state needs an update, the HTTP method PUT needs to be used. The route definition has the structure “app.*method*(*path, handler*)”. As seen in 2.61, app is an instance of express, METHOD is the HTTP request method. PATH is the path on the server and HANDLER the function that is executed when the route is matched. A route can have one or more **handler or callback functions** as arguments. In the head of the function (line 4), req (request) and res (response) are objects provided by Node. They can be invoked even when Express isn’t running. When multiple callbacks functions are meant to be invoked, the keyword next needs to be provided to the handler, as well as the method call “next( )” within the body of the function. It passes control to the next handler. [119]

An application listens to requests that match the specified routes and methods. When it detects a match, it calls the specified callback function. For instance the route in example 2.61 shows the application responds with ”Hello World!” for GET-requests for the root URL (/) or route. For every other path, it will respond with ”404 Not Found”. [54, 119]

Here is an overview of the basic HTTP request methods and an example with route parameters (#5) [119]:

```
1   const express = require("express");     // Requiring the usage of Express
2   const app = express();                  // Creating an Express instance
3
4   // URL Route Examples
5   app.get('/', (req, res) => {         // #1: GET-Request on root route
6     res.send('Welcome!')               // Responds "Welcome!" on homepage
7   })
8
9   app.post('/', (req, res) => {        // #2: POST-Request on root route
10    res.send('Got a POST request')     // Responds "Got a POST request!"
11  })
12
13  app.put('/user', (req, res) => {     // #3: PUT-Request on /user route
14    res.send('Got a PUT request at /user')
15  })
16
17  app.delete('/user', (req, res) => { // #4: DELETE-Request on /user route
18    res.send('Got a DELETE request at /user')
19  })
20
21  app.get("/user/:name", function(req, res) { // #5: GET-Request of a Param
22      res.send(req.params)             // Responds with names from users
23      // the req.params object captures values from the route parameter
           indicated with semicolon (:name)
24  });
25
26  app.listen(3000);          // Server listens on Port 3000 for connections
```

Figure 2.62: URL Route Example with Express (server.js)

Express also features a **generator**, which is a scaffolding tool to create a full app with numerous JavaScript files. To sum it up, Express offers many different plugins to help defining REST APIs, interact with databases, and handle a wide variety of services. This makes it a **robust solution** for creating a web server with Node. Many frontend frameworks feature their own router, to facilitate the integration of the connection to the routes. [119]

2.2.4.3 MongoDB

The platform for application data [102]

Figure 2.63: MongoDB Logo [120]

MongoDB is a non-relational database management system (DBMS). It is used for many different applications in web and mobile development, IoT, analytics, content management, gaming and serverless development.
A **database management system** is a software system that enables users to define, create, maintain and control access to a database. Databases are needed to store all sorts of information, whether there are from simple or complex applications including various data. [42]

MongoDB vs. MySQL
The most popular databases are MongoDB and MySQL [43]. **SQL** stands for "Structured Query Language" and is a query language for relational databases. **NoSQL** stands for "Not Only SQL" and is used in non-relational databases. SQL databases such as MySQL save data in rows and columns while NoSQL databases such as MongoDB save data in documents or collections in JSON format. A **collection** is similar to a table. It can contain numbers, strings or even nested objects. Documents in MongoDB map directly to objects in programming languages. This makes an object-oriented approach of programming very feasible. One of the main concepts of MongoDB is "If data is accessed at the same time, it should be stored close to each other". This is to be kept in mind when designing the database entries. [102, 121]

MySQL (SQL)	MongoDB (NoSQL)
Table (records of the same type)	Collection
Row	Document
Column	Field
Joins	Embedded documents, linking

Table 2.4: Overview of Terminologies in SQL and NoSQL Databases

In NoSQL databases every database entry consists of key-value pairs. These pairs are written in **JavaScript Object Notation** (JSON). JSON was defined as part of the JavaScript language by Douglas Crockford and is the popular standard for data interchange on the web. It is easy readable for both human and machine. An alternative to JSON is XML, which is more difficult for the human to read and doesn't fit object-oriented approaches. JSON objects are close to containers, where a key – mostly some kind of string – is mapped to a value, for instance a number, string, function or any other object. This form makes it very flexible and allows users to create database entries very easily. [122]

```
1 {
2     "_id": 1,
3     "colours" : { "red", "blue", "green", "yellow"},
4     "seq" : [ "r", "g", "b", "y" ],
5     "score": { 0 };
6     "highscore" : { 15 },
7 }
```

Figure 2.64: JSON Example of Database Entries

Although it is easy readable, JSON has some disadvantages. As a text-based format, parsing is very slow. Because of many spaces and line breaks in the file, it is not space-efficient. And it only supports a limited number of basic data types. To make it performant and general-purpose, BSON was invented. BSON is based on JSON and is the abbreviation for **Binary JSON**. It stores data in a binary representation of the JSON format. This makes parsing quicker, improves speed and support of file types like dates and binary objects. Alongside these advantages, it allows the data to be easily compared and directly calculated. [122]

MongoDB is designed to store BSON data natively and also passes BSON over to the network. It is built with JSON and JavaScript, which makes it the perfect candidate for the JavaScript Full-Stack. Specific language drivers feature detailed information on how the BSON data can be accessed. The **document-oriented** representation of data is highly flexible and allows to handle large data. This ensures great speed and performance. [3, 122, 123]

```
1  {"hello": "world"} ->
2      \x16\x00\x00\x00             // total document size
3      \x02                         // 0x02 = type String
4      hello\x00                    // field name
5      \x06\x00\x00\x00world\x00    // field value
6      \x00                         // 0x00 = type EOO ('end of object')
```

Figure 2.65: JSON to BSON Data Structure

Having a NoSQL database brings certain **advantages**. They allow the storage of unstructured data such as images, videos or pdf files. They aren't limited to the table, which means they can scale horizontally by adding more servers. Offering many drivers, MongoDB is compatible with many data models and environments. An additional advantage is, that JSON allows JS objects to be represented in a form that makes it easier to check values individually and write automatic tests. The **disadvantages** include extensive memory consumption while remembering key-value-pair names and data insecurity through fast operations. The security can be enhanced through user authentication, but this is not enabled by default. [122, 124]

According to the Developer Survey by Stack Overflow 2022, MongoDB is used by both professional developers and those learning to code. It is the second "*most popular*" database for those learning to code (behind MySQL). And with 60.51 % it is the third "*most loved*" database. [38]
The survey specifically mentions that those learning to code while currently using MySQL are more likely to use MongoDB over any other database. One reason for its **popularity** is that it supports a large number of languages and application development platforms. [38]

MongoShell
When using MongoDB, the Mongo Shell mongosh is a fully functional JavaScript and Node REPL, making interactions with MongoDB possible. It can be used to test queries and operations directly in the database. mongosh is available for download or through npm and is mainly used for the configuration of the database. Further operations are executed smoothly with a graphic user interface (GUI), where all APIs from the Mongo Shell can be used through the interface. It is more user-friendly than working with the console. That way documents can be inspected in detail through a few clicks. MongoDB shell clients are for instance Robo 3T or the official MongoDB Compass, which are both free to download and use. [125, 126]

The **requests** the database can receive contain a simple query, to update the records or to delete records. To interact with the database, default operations such as create, read, update and delete need to be executed. They are known under the acronym CRUD operations. To be able to connect to MongoDB from other environments such as a Node.js web server, a mongodb **driver** is needed. It can be installed via npm install mongodb. This enables the Node applications to connect to the database. [43, 102, 123]

MongoDB Atlas

MongoDB Atlas combines the power of the document database model with the leverage of the cloud and the power of a full data platform. It is a fully **cloud-based** database service and simplifies the deployment and managing of the database. Moving to the cloud is most effective, when scalability is hard to achieve for an application. MongoDB thereby takes care of infrastructure operations, for the developer to focus on the development. When an application suddenly grows and gets lots of requests – for instance a startup website – MongoDB scales the app to have the best performance. The scaling options by MongoDB can help reduce deployment and maintenance costs. This makes it a great choice for data-intensive applications, who vary in scale. Having the database in the cloud means, that the application can be run anywhere in the world. Through a unified query API, the data stored in Atlas in instantly available with the rest of the platforms services. [43, 127, 128]

Atlas supports **multi-cloud data distribution** across services like Amazon Web Services (AWS), Microsoft Azure or Google Cloud in over seventy cloud servers in different regions around the world. Further services by Atlas are full-text search, visualising data for real-time insights and workflow integrations. It already integrates important drivers for certain languages in order for fast development. A command line interface (CLI) is also available for creating, managing and automating Atlas resources from the command line. [127]

To **build and connect** to MongoDB Atlas, four main steps are involved:
The first step is to deploy the database. Next, the database is secured by managing database users and IP addresses. Thirdly, the connection to the database is created. Finally, by creating custom alerts and suggestions, optimisations are conducted to the database. An exemplary setup is described in 4.5 when using MongoDB in an application. [127, 128]

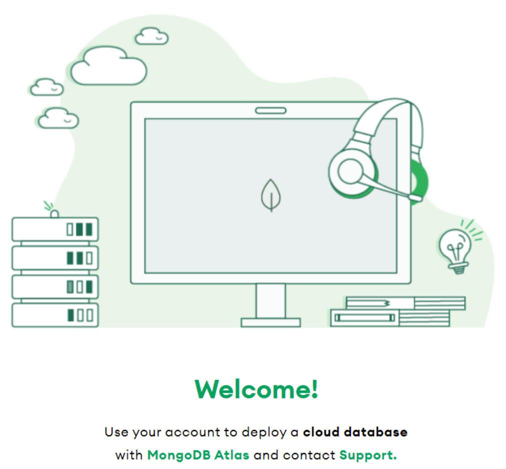


Figure 2.66: MongoDB Atlas Start Page

A **MongoDB Atlas** cluster is the term for a replica set or sharded cluster, which is the replication of a group of MongoDB servers that hold copies of the same data. To set up a Cluster, a number of configurations need to be made through the Atlas UI. Three different models are available: A *serverless database*, which automatically scales the application. A *dedicated database* for production and a *shared database*, which is a free tier. It allows to test MongoDB directly in the cloud. Only a registration is needed. Free clusters come with 512 MB of storage. Testing sample data and getting used to the platform is easy, especially for learning opportunities. [127, 128]

2.2.5 JavaScript Alternative: PyScript

> *PyScript is just HTML, only a bit (okay, maybe a lot) more powerful, thanks to the rich and accessible ecosystem of Python libraries.*
> – PyScript.net [129]

Python is the second most widely adopted language after JavaScript, with the gap between the two largest language communities gradually closing. It counts 15.7 million users, with 3.3 million new developers in Q1 2022 in comparison to DeveloperNation's last edition in Q3 2021. The **increase** than can be seen is due to the growing popularity of using Python in fields of data science and machine learning with around 70 % compared to other languages. [33, 129]

The Developer Survey by Stack Overflow shows a similar trend with Python being the "*most wanted*" language and TypeScript as close second with around 17 %. Additionally, they showed that 27,29 % of respondents – who worked with JavaScript – want to work with Python. It also has gotten more common to start learning to code with this language. As many beginners and more experienced developers use it, there is an indication that using Python in Web Development might be very wanted in the future. [38]

Developments in 2022 show a new framework which will allow Python programmers to run python code in their browser, named **PyScript**. Similar to JSX making it possible to run JavaScript in HTML, PyScript enables to run Python in HTML. [129]

The PyScript Developers are stating that PyScript is very new and under heavy development. There are many known issues from usability to loading times, and developers should expect things to change often. With the release of this version they rely on the open source community, encouraging people to play and explore with PyScript. Although not ready for production, it is an interesting concept for all machine learning and data science experts who are working with Python. If furthermore opens the opportunity for projects which are computation-heavy. [33, 129]

2.2.6 Characteristics

A framework eases the handling of JavaScript and provides different kinds of support via functions, modules or strictness of coding. They help abstracting and increase the speed of development. Additionally, they can fix some pain points of JavaScript through new functionality. To comprise the different elements of the invested JavaScript Frameworks, let's take a look at recapping the characteristics of each framework before diving into tech stacks. [90]

These are the characteristics in an overview:

- The Vanilla JS implementation is very simple. Nowadays it might seem as an obsolete way to create a web application, but it is still viable and can be used in frameworks. Because unopinionated decisions were made, the developer has full control over the code.
- React uses a component-based approach with a Virtual DOM. The creation of a class component was easy and simple to integrate into the app.
- Angular uses a component-based approach, an incremental DOM and directives. Through the template syntax, it was a more difficult to create the Button than withReact.
- Vue uses a component-based approach, a Virtual DOM and is declarative. It seemed like a mixture of React and Angular, featuring only the benefits for a fast creation.

After seeing these technologies in an overview the conclusion can be made that there are many ways to express one simple application: In a simple file or a complex combination of approaches.

2.3 Technology Stacks

As the name *Full-Stack Web Development* suggests a *stack* is part of the Web Development process. In this section there will be a focus on the combination of JavaScript frameworks, as so called *technology stacks*. Before diving deeper into the details of the most popular stack types, the fundamentals of stacks need to be addressed.

2.3.1 Fundamentals

The word *stack* is a common term used in **computer science**. It is a type of data structure often described as a pile of plates, where you only have access to the top of the "stack". The principle is called "Last-in-First-out-Structure" or abbreviated **LIFO**. Possible behaviours to change the state of a stack are to either add more elements to the stack with a push( )-method or remove elements from the stack with a pop( )-method. This means that the most recently added element is the first one to be chosen. Stacks can be compared to physical items such as mailboxes, disks or books stacked on top of each other. [130, 131]

A stack is often used in programming languages like Python, Java or C++. The JavaScript language as well as other languages themselves do not feature this **data structure** by default but there is a work-around by using a modified array as a stack. [131]

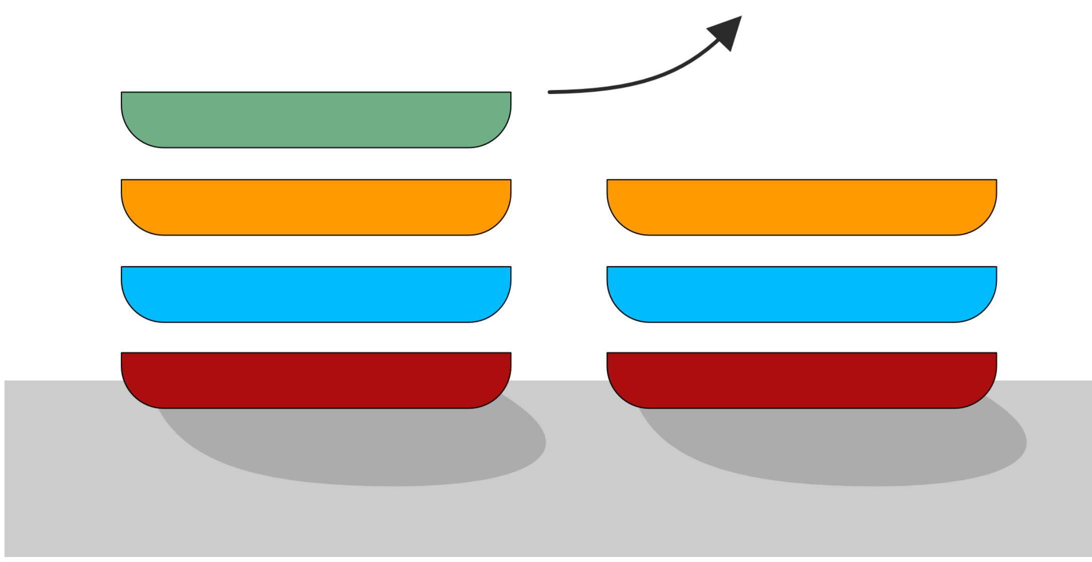


Figure 2.67: Stack Data Structure with Last-in/First-out (LIFO) manner

Leaving the programmers definition of the data structure, we will stick to the idea that different elements are stacked on top of each other like **layers**. This is essentially how a technology stack works: different frameworks and technologies layered, working together to form an application. [44]

A **Technology Stack** (abbr.: tech stack) is a combination of programming languages, frameworks, databases, and other tools or software that is used to build any web or mobile application. A **Web Technology Stack** or **Web Development Stack** (abbr.: web dev stack) is a combination of specific *stacked up* components that make the website fully functional. Once built up, a tech stack is specifically designed for a particular use case, type of company, the knowledge a Full-Stack Developer has and can be used repeatedly (cf. 2.4.1). [3, 42, 44]

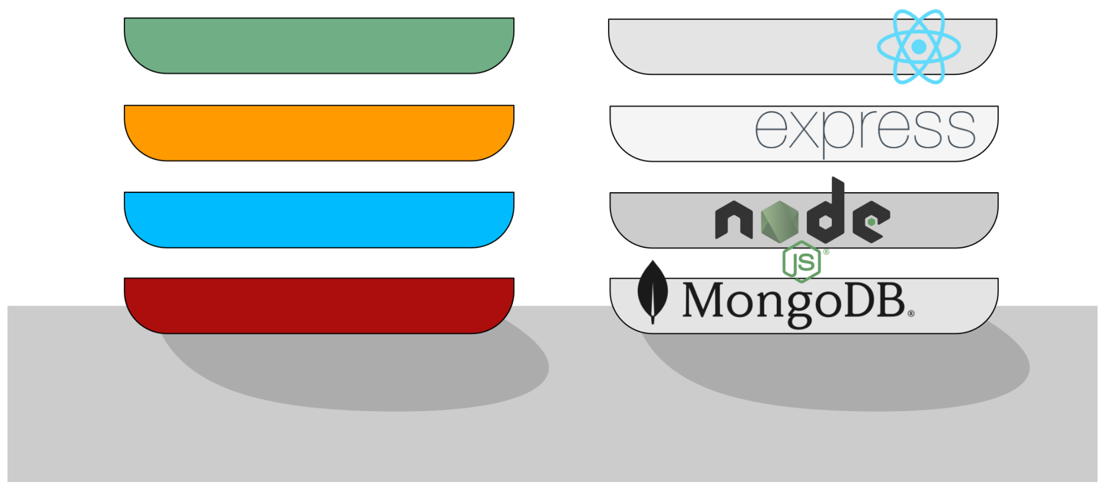


Figure 2.68: MERN Stack consisting of MongoDB, Express.js, React.js and Node.js in comparison to the stack data structure

A technology stack has become essential for building easy-to-maintain, scalable web applications [43]. As described in 2.2.1, most web developers specialise in either frontend or backend. The Stacks that Frontend-Developers have in use may include frameworks, libraries, package managers, build systems, testing tools, version control systems, caching tools or deployment software. Backend-Developers tend to use containerisation tools, APIs, databases, search engines, caching mechanisms and DevOps tools (see 2.4.2). With **Full-Stack Development** all aspects of Development are covered within one technology stack. A person that uses JavaScript as an universal language in both front- and backend is referred to as a JavaScript Full-Stack Developer. [42]

Figure 2.69: Overview of Technologies used in Full-Stack Development

2.3.2 Tech Stack Variants (MEAN / MERN / MEVN)

Firstly, a differentiation between the terms **tech stack and stack architecture** needs to be made. The Web Technology Stack is a list of technologies used to build a website and describes a project integrally. It is a list showing what is beyond the user interface. An architecture is an interface description. It shows the connection and activities in-between the components of the tech stack. Common architectures are monoliths and microservices, which will be presented shortly in 2.4.2. The terms *Stack Variant* and *Stack Architecture* cannot be used interchangeably. [132]

To sum it up, a **Web Development Full-Stack JavaScript Technology Stack** (Web Dev Full-Stack JS Tech Stack) is a combination of a frontend framework, a backend framework, a runtime environment (that makes it possible to use JavaScript in frontend *and* backend), and a database. Those are the elements needed to build a website. With Node (*Section 2.2.4.1*) developers were able to use the same language and knowledge on both client- and server-side. Before, this was only possible with Java and a selection of Microsoft technologies. The introduction of JavaScript on the server-side opened up the opportunity for Full-Stack JavaScript Development. [71]

As programming languages and frameworks tend to have some differences in their core and performance, tech stacks offer insights on the **strengths and weaknesses** of the entire application. An important performance marker thereby is the programming language itself. For instance in comparison to JavaScript-based stacks, PHP-based stacks tend to be more extensive and harder to debug due to PHP itself being a rather inefficient language. Knowing a tech stack helps the developer to decide upon which language to learn in order to implement changes. [133]

How to choose and build a Tech Stack

Tech stacks are often particular to a company or use case. There is no rigid definition on which tech stack is the best. A possible procedure in choosing a technology stack is shown in *Figure 2.70*. The process starts with the important part of choosing a **core language**. In this instance the selected language is JavaScript. Afterwards a **frontend framework** must be chosen. This depends on what the applications requires and what business purpose it pursuits. If it is a simple application, a rather easy framework might be the preferred choice. The framework needs to be in a state of optimal efficiency so that the design team can build up the application. As shown in *Section 2.2.3* there are many possibilities of frontend frameworks with different approaches. [43, 133]

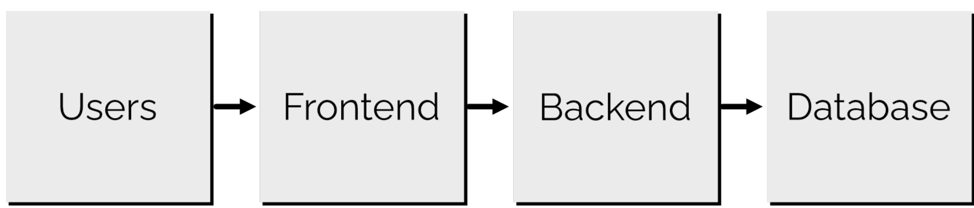


Figure 2.70: Building a Tech Stack, adapted by [133]

After choosing the frontend, the **backend** needs to be chosen in alignment with the frontend, building up the user interface and the connections to the database. The resulting stack of components is the tech stack. Once built up, it can be altered and suited to more specific use cases when gaining more experience. It needs to be kept in mind, that an alteration of the underlying technologies might influence all layers on top. The final tech stack then is a **consistent part** of the company. Knowing a specific tech stack is often a criteria in job interviews. [43, 133]

In a **modern tech stack** the base components can be complemented by additional ones which support the increase of numerous challenges such as big data or the number of devices from which users can access the application from. These modern tech stack components are for instance development operation (DevOps) tools like version control and microservices, performance monitoring and cloud services. Also business intelligence, event processing and analytics might be useful. Further explanations of DevOps are found in *Section 2.4.2*. The components of the stack truly depend on the type of application and if complex operations needed to be handled. [43]

Before deciding on the final tech stack, **tests** can be run to find the stack with the perfect fit concerning the knowledge and needs of the developers and type of company. It should be kept in mind to leave a certain **scalability** to the stack. If an application grows in popularity, it should scale without losing any performance. Update cycles and maintenance of the stack thereby also play an important role. With no experience, mature technologies and good community support are a good start. Full-Stack Developers might be the correct choice for a startup because of their diverse knowledge concerning Web Development. Knowing the details of the tech stack facilitates the communication of functions of the application. Last but not least, the budget should be checked before starting to implement the application. Planning the tech stack can save both costs and time throughout the entire project. [43, 133]

Another important aspect is to keep the **purpose of the application** in mind. For example whether it is used through a browser or an app, or if frequent database interaction is needed or which content gets displayed. All these factors decide on which frameworks should be used. Therefore the comparison of frameworks will not lead to a definition of the perfect technology stack. The comparison (in *Section 3.4*) will give hints on when its best to use which tech stack. [43, 133]

MEAN vs. MERN vs. MEVN

Technology stacks that cover the whole developing pipeline with JavaScript are MEAN, MERN, MEVN and MEEN. The most popular are MEAN and MERN. They all feature the database MongoDB managing the data, the backend framework Express.js, the runtime environment Node.js and a particular frontend framework providing the user interface. Instead of Express.js the technologies Meteor or Sails can be used. But Node.js and ExpressJS scored above all in the category popularity in the StateOfJS [17]. Therefore it is worth taking a look at the stack variants including ExpressJS. The dis- or advantages between MEAN, MERN, MEVN and MEEN consequently depend upon the **differences between** the used **frontend frameworks**. As seen in 2.2.3, the stacks with Angular, React.js and Vue.js will be investigated in detail. EmberJS will be excluded from the comparison. [3, 43, 62]

Full-Stacks with JavaScript	Full-Stacks without JavaScript
MERN (MongoDB, ExpressJS, ReactJS, NodeJS)	LAMP (Linux, Apache, MySQL, PHP)
MEAN (MongoDB, ExpressJS, AngularJS, NodeJS)	LEMP (Linux, Nginx, MySQL, PHP)
MEVN (MongoDB, ExpressJS, VueJS, NodeJS)	Django stack (Python, Django, MySQL)
MEEN (MongoDB, ExpressJS, EmberJS, NodeJS)	Ruby on Rails (Ruby, SQLite, Rails)

Table 2.5: Overview of popular Technology Stacks

Usage Overview: The MEAN stack is used by big companies like Google and PayPal. MERN is also used by big companies like Meta (Facebook, Instagram) and Netflix. Lastly, MEVN is used by smaller companies like GitLab, Grammarly and Xiaomi. MEEN is not commonly used. The **usage** can be explained through their backgrounds and the popularity. [3]

Angular, React and Vue are the client-side frameworks in the technology stack, while Express, Node and MongoDB are used on the server-side. Using these frameworks allows the development of web applications using **only JavaScript**. This means the developer learning those frameworks can focus on the foundational concepts. [134]

To explore what each letter in the MERN stack means, the connections in-between the technologies will now be explained. Node (**N**) is the runtime environment, that allows JS to be executed on the server side. Node and Express (**E**) build up the backend APIs, so that frontend and backend are able to communicate. The middleware Express furthermore is needed, so that sensitive user data is not exposed directly to users of the application. MongoDB (**M**) is the database. By clicking on a button on the UI, the frontend sends a requests through the Express/Node REST API to the MongoDB Server. Then MongoDB sends its response to the frontend as a JSON response. Using MongoDB and Express is great, as they both use JSON and BSON. The user interface is built with the frontend framework, in this instance React (**R**). It handles the responses and updates the changes for the user.

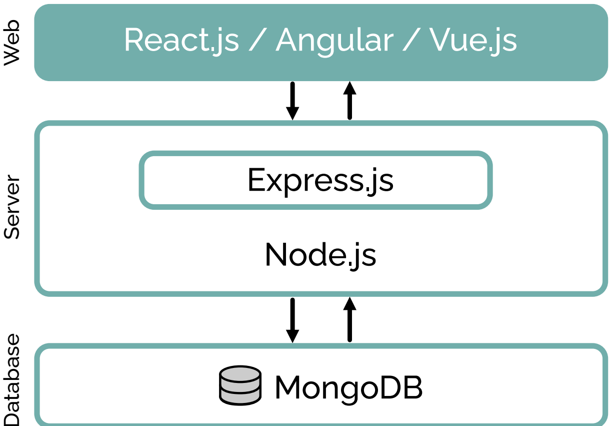


Figure 2.71: MERN / MEAN / MEVN Technology Stack Architecture, adapted by [135]

Other **popular tech stacks** – which are not based on JavaScript – are LAMP (Linux, Apache, MySQL, PHP) and LEMP (Linux, Nginx, MySQL, PHP) with PHP in the backend and Linux as the dominant web development environment. More commonly known names of tech stacks are Django (Python, Django, MySQL) and Ruby on Rails (Ruby, SQLite, Rails), featuring the most important technology of the stack in the name itself. The major alternatives of using Node as the runtime environment for JavaScript are Django based on Python, Symfony based on PHP and Ruby on Rails based on Ruby. Most stacks feature MySQL or SQLite as the database. The Django Stack might be interesting for Machine Learning Developers due to the popularity of Python. [3, 62]

The **LAMP Stack** for instance features the Linux operating system, the Apache HTTP server, relational DBMS MySQL and the programming language PHP. It is one of the most classic tech stack types and was one of the first open-source software stacks used worldwide, resulting in its popularity due to its simplicity and stability for developing new applications. Platforms such as WordPress (a popular content management system) also use the LAMP stack, because it handles dynamic pages well. An advantage of using this stack is as it runs on Linux, it can be easily modified. [62]

In comparison to JavaScript-based stacks, non-JavaScript-based stacks tend to be more tedious to learn, because of the differing languages on the frontend and backend. As a consequence no code can be reused, and additional learning of another language is needed. There are many other stack variants, but the final choice is up to the developer. [62]

2.4 Full-Stack Web Development

When thinking of the Web Development process, there are many things to consider: for instance code structure, interface design, and databases. As the complexity grows and a more robust and secure flow of data is required, experts who can manage this are needed. This section focuses on the role and skill set of a Full-Stack Developer, the concept of DevOps (Development Operations) and a concluding discussion about the advantages and disadvantages of being a Full-Stack Developer.

2.4.1 Full-Stack Developer Role and Skill Set

> *Being considered unicorns due to their rare expertise and versatility, they are in high demand for startups and large companies alike, from Facebook and eBay to Munchery, or Tinder.*
> – AltexSoft [3]

A lot goes into being a Full-Stack Developer. They usually can be found in either Web or Software Development. Web Developers generally specialise in either Front- or Backend-Development. Both parts are equally demanding and contain different **challenges**. That being the case, the combination offers a broad spectrum of challenges. [3, 44]

When working in Full-Stack Development, the **language** is the main skill and should be internalised. In this instance, this would be JavaScript. Taking the popularity of the language JavaScript into account, such developers are some of the most sought-after specialists in web development currently [3]. Having JavaScript as the core language of the technology stack therefore helps having a market full of opportunities. In contrast to JavaScript, being in a niche with a very particular programming language helps getting top-positions, but it becomes harder to find a job. [133]

As a web application is composed of different parts, the expertise of the Full-Stack Developer should consist of **specific frameworks and databases**. These are the building blocks of the tech stack and lead – when applied professionally – to a successful career. They can be seen as the mandatory skill set of the developer. Furthermore, there are tools accompanying the **modern tech stack**, that a developer should know. This includes performance monitoring, cloud services, business intelligence, testing skills and DevOps (explained in the next section, 2.4.2). [3]

Some basic skills complementing the workflow are the usage of the command line, the knowledge of working with graphic tools and knowledge of different operating systems. These may vary from different stacks or orientations of the developer. On the side there are **soft-skills** which complement the technologies and the work beyond. Knowledge of algorithms and data structures is recommended, as a good programmer has a certain amount of problem solving skills. This helps approaching problems logically, overall dealing with technologies, dealing with coding problems and the code itself being well-written. [121]

Soft-Skills such as resilience, patience and having fun in learning technologies, are highly recommended. As the tasks of the Full-Stack Developer vary from coding to designing, an eye for design (mobile vs desktop), a feel for positioning and creativity in every aspect of the work is handy. Being on the design side, the important part of **User Experience UX Design** and **Usability** cannot be underestimated, as they provide a certain look and feel to the application influencing the user. Furthermore, as a possible leader position or just being a team member in a project, communication skills as well as leadership skills are in demand. Being *conscientious* of decisions in every part of development is crucial. [121]

JavaScript may move fast, but it seems like JavaScript developers move even faster, as many relatively new features already show high adoption levels.
– State Of JS 2021 [17]

For understanding the skills and knowledge of each member, Full-Stack-Developers are a useful asset to a developer team. They offer knowledge in each stage of software development including mobile development, server operations and security, front-end and back-end development, and design.

Being **up-to-date** in the development community is important, especially when working with JavaScript. With the rising number of frameworks since the introduction of Node.js, there are many possibilities on how to build upon a web application. [17]

The knowledge of **current technologies** is very important and in high demand for large companies. They strive to boost new ideas, aiming to be the best at their jobs and deliver good products. Startups are interested in Full-Stack Developers, because they combine the expertise of a range of disciplines. This makes them perfect project leaders and good starting points of a web development journey that needs to be launched. [3]

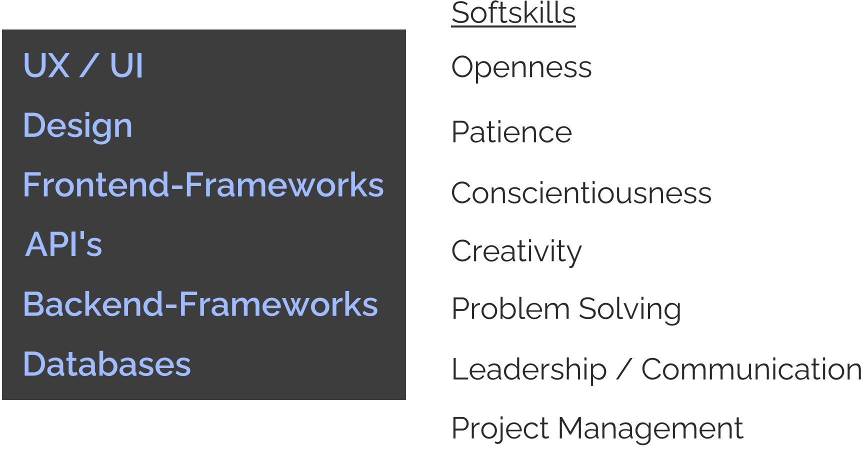


Figure 2.72: Skill Set of a Full-Stack Developer

Knowing different frameworks and technologies on a high level is very difficult and takes time to learn. A Full-Stack Developer for that reason often **specialises on a specific stack** or several similar stacks. As a potential worker of a specific company, knowing the web dev stack is an advantage. Job Offers often feature the specific tech stack which is used. Hence, a basis of applicant elimination is the level of experience regarding the specific stack or willingness to learn new stack variants. The *openness* to learn new technologies and keep the mind open of learning therefore is an important soft-skill. This keeps the job alive and fun when loving to learn new technologies. [42, 133]

2.4.2 Development Operations (DevOps)

> *A great developer experience is on everyone's mind. At work developers, managers, and organisations want tools and processes to be fast, delightful, and easy.*
> – State of the Octoverse, GitHub [32]

The term DevOps is composed of the parts „Dev" (Development) and „Ops" (Operations). The goal is to unite people, processes and technologies, for costumers to have continuous and high-quality products [136]. The Idea of DevOps can be seen as an automation or process of ensuring a faster development. While previously a web team had separate roles like development, quality checks and security checks, DevOps combines these roles via the concepts of **coordination and cooperation**. The Development and Operational IT-team are no longer separated through a barrier. Instead, they use a common toolset and methods to work closely together, supporting the productivity of the developer and the reliability of the process. When switching to this approach, it is nowadays known as introducing a 'DevOps culture'. [136–138]

DevOps Culture and Methods

DevOps tries to ensure not only faster development, but also more reliable and improved software products, which are more **efficient and effective** in practice. Thus, many companies try to implement a DevOps culture with methods and tools to create more trust in their products and react faster to demands of the costumers. This additionally increases customer satisfaction. Most companies are starting to switch to a cloud infrastructure with the cloud service Azure [136] and / or services by Amazon Web Services (AWS) [139]. As seen in relevant job portals like StepStone [140] or LinkedIn [141], the demand for DevOps Developers is increasing. [136, 137]

Traditionally, people using DevOps are **programmers and software developers**. According to Slash Data's DeveloperNation (cf. Figure 2.73) the percentage of classical programmers is 49.3 %. It is followed by 15.1 % computer or data science students and 10.1 % tech / engineering team leaders. Architects, data scientists and machine learning developers use DevOps with under ten percent. Web Development falling under the category of Software Developers, hence belongs to the field which is strongly increasing their focus on DevOps. [33]

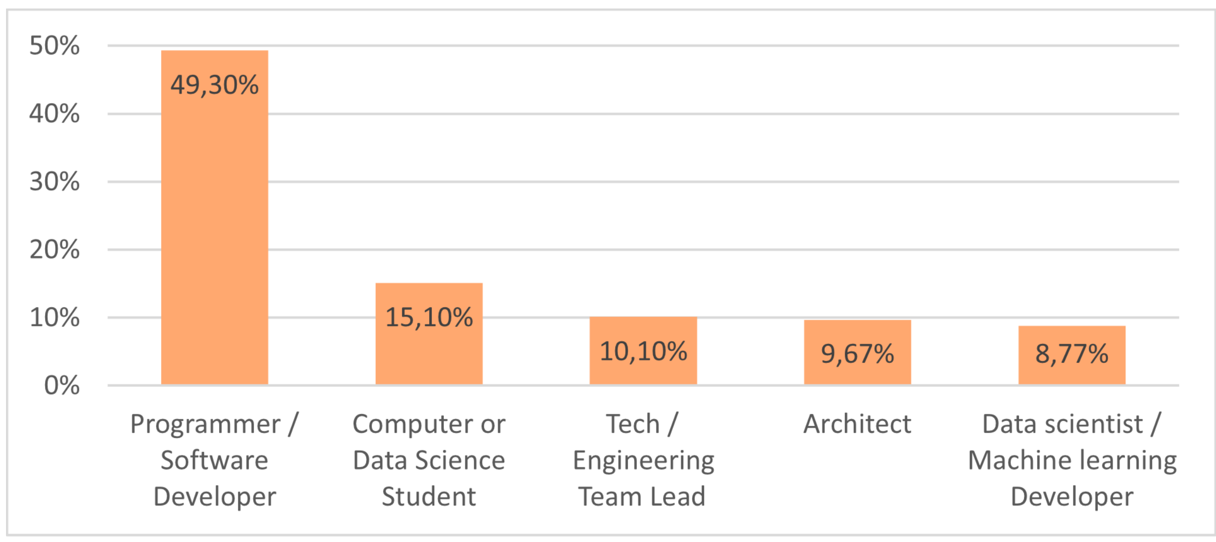


Figure 2.73: Involvement in DevOps by Company Role, adapted from SlashData's Developer Nation Q1 2022 [33]

DevOps consists of different methods. These methods can accelerate or optimise processes or try to help the connection of processes [136, 142]:

- **Continuous Integration** (CI): Continuous Integration of code changes in one place, where they automatically are built and tested.
- **Continuous Delivery** (CD): Continuous Deployment of all code changes to an environment after the build (from build to stage to production).
- **Version Control**: Maintenance of Code versions.
- **Microservices**: Building and deploying a microservice architecture using containers.
- **Monitoring and Logging**: Record logs and monitor application and infrastructure performance in near real-time (Version Control).
- **Infrastructure-as-a-Service** (IaaS): Monitoring and enforcing infrastructure compliance with version control.
- **Platform-as-a-Service** (PaaS): Deploy web applications without needing to provision and manage the infrastructure and application stack.
- **Software-as-a-Service** (SaaS): Hosting Application, which can be connected to and used by clients.

Using the **Cloud** for DevOps makes it more flexible. Major cloud providers like Amazon Web Services (AWS) and Microsoft Azure thereby especially provide the services 'Infrastructure-as-a-Service' (IaaS), 'Platform-as-a-Service' (PaaS) and 'Software-as-a-Service' (SaaS). They are used to implement DevOps methods. For instance, a CI/CD pipeline can be implemented through an IaaS instance. [136]

Taking a closer look into the **usage**: 47.3 % of developers utilise DevOps for continuous integration (CI) and 35.9 % to monitor software and infrastructure performance. As developing is a continuous process of improving and rewriting code, it is useful to approve code deployments before they are headed to the production website. 34.3 % of developers use it for that reason (see Figure 2.74). Another reason that was mentioned by SlashData are the programmatically provision and management of IT infrastructures with 27.3 % usage. Also 23.7 % of respondents create automated validation checks like Prettify or unit tests. [34]

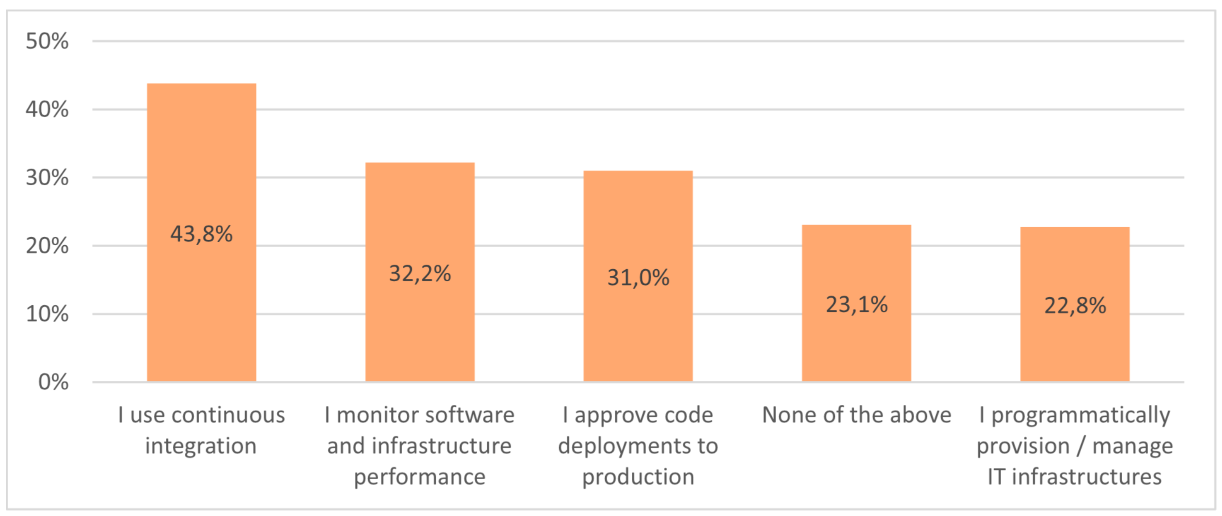


Figure 2.74: Type of involvement in DevOps, adapted from SlashData's Developer Nation Q1 2022 [33]

Continuous Integration / Continuous Delivery

CI/CD is the paradigm for automation in building, testing and deploying application development. It is used by gathering all code changes in a central repository. These changes are then automatically tested and deployed, depending on the number of steps in the so-called **CI/CD-Pipeline**. A pipeline includes a number of steps which need to be executed before the change can be deployed. This shortens waiting time and makes the deployment of version changes more smooth. This is also the reason why companies using DevOps often deploy more updates than companies using common software development methods. [138]

Many processes – especially error correction – can be eased using CI/CD. When integrating code formatters, automated testing and deployment, many mistakes can be avoided. This prevents an upload of a website with issues, which in return would mean downtime because of error research and failed components. This provides a more **robust and safe flow of programming**. [137]

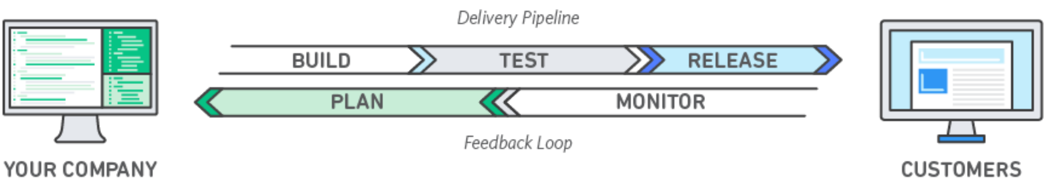


Figure 2.75: DevOps Delivery Pipeline and Feedback Loop [139]

The DevOps cycle is built upon different **steps in the pipeline**. The pipeline generally consists of testing, building and publishing. Automatised Software-Tests, for instance Unit Tests usually show mistakes of the software and keep the functionality within the estimated project scope. This means the tests are continuously integrated, leading to the name **Continuous Integration**. Therefore, mistakes are detected faster and less time is spend on correcting errors. The pipeline can be initialised and automated in Version Control Systems. [137, 138]

As part of the continuous integration pipeline, **code formatters** like **ESLint** [105] can additionally be run. This makes the code more consistent and avoids bugs caused by the syntax. Debates on different coding styles from different programmers are stopped and emotional negotiations of reviewing the code are ended [143]. ESLint is a tool for identifying and reporting patterns found in ECMAScript / JavaScript code. It statically analyses the code, being a linting utility for JavaScript and JSX. Many problems found can be fixed automatically, and it is possible to integrate customisation for the project. [105]

Another code formatting tool is **Prettier** [143]. Prettier supports many tools, frameworks and languages. It is *opinionated*, which means that it only allows few customisation's. They are supported in important editors like Visual Studio [144], VS Code [145], WebStorm [146], Vim [147] or Atom [148]. Prettier enforces a consistent code style across the entire codebase. [143]

> *It reminds me of how Steve Jobs used to wear the same clothes every day because he has a million decisions to make and he didn't want to be bothered to make trivial ones like picking out clothes. I think Prettier is like that.*
> – Prettier.io [143]

In the StateOfJS 2021, 70.3 % of survey respondents regularly use ESLint and 64 % use Prettier. Also, Babel also is included in the workflow with 48.5 % of usage. Additional important tools are the **node version manager** (nvm) for OSX and Linux or nodist for Windows Operating System (OS). [17]

Package & Version Managers

Package and Version managers are tools to manage dependencies. Packages are mostly reusable code-modules or plugins by third-party-suppliers which add functionality to the code. Some packages are dependent on others. This makes dependencies a very critical point of development, which can often cause errors. JavaScript has multiple possible package managers, including npm [149], yarn [150] and bower [151]. [11]

The most popular package manager is the node package manager, which was already presented in detail in 2.2.4.1. npm offers a good starting point because it is heavily used and offers good functionality. It helps keep track of what node version is used in a project and makes updating or switching between the versions easier. [17, 152]

Besides the useful npm module n mentioned in 2.2.4.1, the **Node Version Manager** (nvm) for Linux helps install and manage multiple version of Node on a more global scale. It is a standalone package which is commonly used for the easy switching to a different version of Node. A simple use case of the node version manager is also to upgrade to the LTS version of node through nvm shown in *Figure* 2.76. It only needs the command nvm install --lts, resulting in the automatic installation of the version. It then changes the usage to the just installed version. [152]

```
nadine@nadinedevice: ~\mydirectory nvm install --lts
Installing latest LTS version.
Downloading and installing node v16.16.0...
Downloading https://nodejs.org/dist/v16.16.0/node-v16.16.0-linux-x64.tar.xz...
################################################################################
Computing checksum with sha256sum
Checksums matched!
Now using node v16.16.0 (npm v8.11.0)
```

Figure 2.76: nvm Update of Node.js to the Long-Term Support Version

Microservices

Microservices describe an architecture of decoupling large, complex systems into simple, isolated projects. An application therefore is disassembled in components – called **services** – each fulfilling a specific function. These services are independent from each other. Splitting an application into services makes coordinating updates of a whole application easier, so that fixes on old services as well as adding new services are very simple. They are supervised by smaller teams, who have the responsibility for one service. This makes a project very flexible, easy to update and modules can be replaced easily. As Node.js is highly-scalable it is a great choice for creating a microservice via node modules and functions. [139, 153]

The architecture decouples front- and backend through various services in a backend that is connected to the frontend via API. The services need to fulfil the **criteria** of having a single responsibility, as they need to be able to be encapsulated to other services. The user interface thereby connects to the independent service, when it is needed. This leaves opportunities to reuse other modules, making apps more scalable. [153]

A prominent way to create microservices is to use **Docker or Kubernetes**, that each have 37.08 % and 23.82 % of usage. This makes them the most wanted DevOps tools in the Stack Overflow Developer Survey in 2022. They are also the most loved and wanted tools with 76.92 % for Docker and 74.75 % for Kubernetes. [38]

Docker containers help to isolate microservices. Building **containers** is providing independence across platforms and operating systems, which leads to the smooth execution of containers in any environment. A cluster of containers can be managed through Kubernetes. The containers therefore provide higher flexibility, consistent operations and less overhead of system resources. Microservices enable DevOps cultures to do fast, small updates where one element can be changed without influencing the whole application.

Digression: Microservice vs. Monolith architecture

Another software architecture common in web dev is the **monolith**. It is traditionally a self-contained application, also called *single-tier* application. The architecture is built up in three components, the UI, the business logic and the data interface that communicates database requests. They are managed in a large repository, which is connected to the database. The “all-in-one” solution has low costs, is rather easy to design and test, and eliminates cross-application difficulties such as monitoring. The **disadvantages** are that any change will influence the whole code base, any error will effect the entire application while simultaneously being harder to maintain. It also is bound to a single technology stack. Although being simple to develop at first, the monolith architecture tends to become complex to update and difficult to manage. [153]

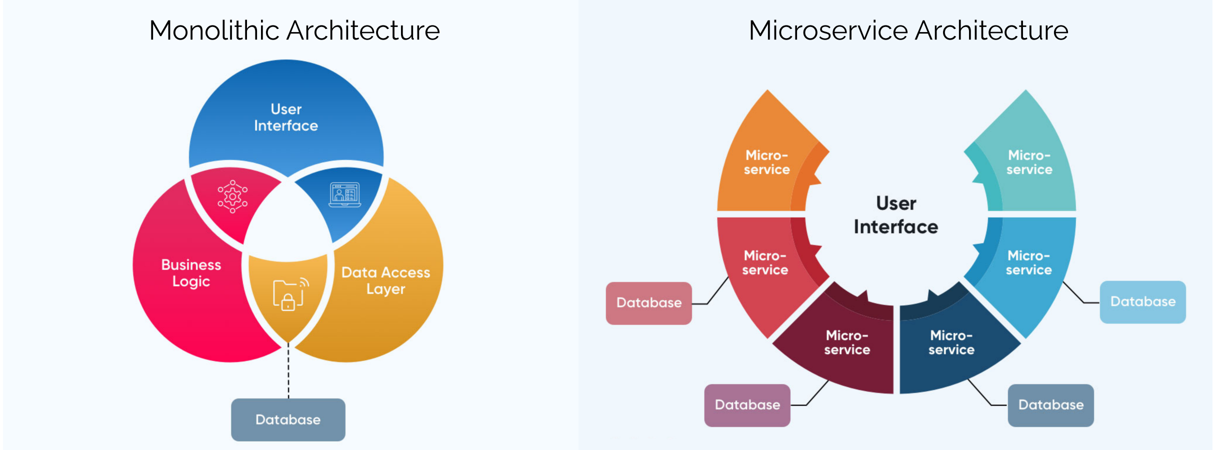


Figure 2.77: Scheme of Monolithic vs. Microservice Architecture, adapted from [153]

In contrast to the monolith, the Microservice architecture is much more easy to maintain, small changes don’t affect the whole application and processes can be accelerated because of the reusability of code. Furthermore, each microservice can use the tech stacks independently. Although the architecture brings a lot of advantages, this comes with a price. The **development** is more difficult, especially when different frameworks and technologies are in use. They need more logging and monitoring as well as keeping the general overview. While some services are easy to define, it can be difficult to design the microservices themselves. [153]

In the end, it depends on the development team and type of application to decide on the architecture. Both approaches have advantages and disadvantages, which need to be outweigh. The microservice approach is gaining popularity, especially in combination with cloud-services. [17]

Version Control

Version control is the **maintenance of code versions**. As a team of developers is constantly changing different pieces and revisions of code, the course of changes can be monitored and reviewed. This system shows clear processes and code changes, which makes code conflicts and rollbacks to previous code versions easy. Many developers use version control in their projects to make them more manageable. There are different **version control platforms**. According to the Stack Overflows Developers Survey 2022 compared to SVN and Mercurial, **Git** is by far the most widely used technology with 93.87 % óf usage, used by professional developers and learners equally. Two very popular version control platforms of Git are GitLab and GitHub. [38, 136]

In version control systems, developers upload their code in shared code **repositories**. A repository can use different tools such as kanban boards for project management, issue management with the possibility to assign different developers and creating different branches of code to test the functionality before applying it to the main application. A new feature of GitLab are **GitLab Pages**, where GitLab hosts an instance of the project directly on their server, showcasing the application. CI/CD pipelines also can be integrated to the code base, making monitoring easy and the deployed application protected from errors leading to downtime. [154]

In the beginning stage of developing, using Git seems frightening and it takes some time to fully understand its functionality. But the more complex a project gets, the **advantages** of code control, having an overview and the project management possibility of the platforms outweigh. Because Version Control is commonly used amongst web developers, it can also be seen as a fundamental tool of the technology stack that a developer must learn. [38, 154]

Takeaways about DevOps

DevOps brings a lot of **advantages** such as improved pace and reliability of operations, as they no longer rely on manual operations. The developers not only specialise in one process, they rather try to maintain the entire application life cycle from developing, to testing, to deployment and operation. This encourages to learn different skills, that can be used customisable to each project. Furthermore, engineers can manage code deployment by themselves without supervision. This adds an **interchangeability** of tasks to each team, with code still being ensured to be correct. [139]

As an application grows and more updates need to be done, the deployment gets more difficult. Especially when introducing the DevOps culture to a company, there are many challenges ahead. At this point Continuous Integration and Delivery (CI/CD) are very helpful. DevOps has the possibility of adding **quality insurance and security measures** like configuration administration or correct management to each step, adding additional security and compliance to the project. When using DevOps, a company has a shorter **time-to-market** with insurance of stability and reliability. It adapts faster to the market and competition, as well as going back to a **working solution** is faster. Planing and risk management thereby get eased, because of the incremental processes of continuous integration. An important aspect to keep in mind is the **reliability of the cloud**. Once linked to a cloud, a downtime has an influences on the application. [136]

There are some challenges in setting up a proper DevOps culture. As a web developer knowing these issues is important. In the digital world, that is constantly changing and growing in complexity, the concept of DevOps is very interesting. Designing and Implementing a DevOps Culture is a **process**. It takes an introduction to the methods and teamwork. Once implemented, it can lead to a more efficient work environment. [136, 139]

2.4.3 Advantages and Disadvantages of Full-Stack Development

> *Too bad! Same old story! Once you've finished building your house you notice you've accidentally learned something that you really should have known – before you started.*
> – Friedrich Nietzsche, Beyond Good and Evil

To conclude this chapter, the spotlights are on the advantages and disadvantages of Full-Stack Development. Beginning with the **disadvantages**, there is much a Full-Stack Developer has to learn. In contrast to mainstream Frontend- or Backend-Developers, one needs to acquire knowledge in both areas. This makes a difference in the amount of expertise. Specialising in both areas is very hard. One cannot know everything to one hundred percent. Furthermore, the time needed to execute a project might be longer because one person cannot alone work as fast as ten (provided that the organisation in the work group isn't bad). The expertise a single-sided or 'half-stack' developer has, is very dependent on his experience and specific use-cases. [121]

A Full-Stack JavaScript Developer has to learn a variety of skills. As Nietzsche said, only in the end, you realise what you should have learned before starting your project. Therefore the development projects a junior Full-Stack Developer has are smaller scope comparing to a senior position. With every small project the Full-Stack Developer acquires **valuable experience**. One might start with just Frontend-Development and then gets into Backend, only becoming a Full-Stack Developer in a larger time span. As the skill set is very demanding, one might even argue that a complete Full-Stack Developer doesn't exist. It takes several years to become a proficient Full-Stack Developer, as for Frontend or Backend Development the learning time might be shorter. Everyone has their strengths and weaknesses, so a Full-Stack Developer might be rather frontend or backend oriented. But the knowledge itself is the important part of the definition. Being a Full-Stack Developer simply shows endless opportunities of learning about technologies. Openness to experiences, agreeableness and conscientiousness are valuable character traits for this position. [33]

Continuing with the **advantages** it is easily said, that a Full-Stack Developer has the skills of building a web application all by himself. He not only can create the application, but also maintain and improve it at any point in the development workflow. Furthermore, the knowledge of the Full-Stack Developer contains every part of development. Between databases, git and frameworks, the usage of only one language – JavaScript – furthermore helps keep frontend and backend in sync. This makes is slightly more easy to keep track of the different sides. [3, 121]

The knowledge of the variety of things makes a full-stack developer extremely **versatile** and encourages looking at projects in a vast perspective. Thereby the competence of having lots of knowledge in different fields it the key to success. [3, 121]

Due to its expertise on many levels, a Full-Stack Developer has **excellent chances** on the job market, especially for project leading roles when being a senior developer. They are needed to keep on track with the users and developers needs. [121]

Figure 2.78: Full Stack Developer [155]

Chapter 3

Comparison

With the fundamentals of *Full-Stack JavaScript Development* shown in *Chapter 2*, the foundation of the comparison of *JavaScript Technologies* is laid. To gain useful insights, the scope conditions and appropriate measure criteria needed to be determined. This was done through the extensive research in *Chapter 2*, the evaluation of a questionnaire with students from the university course "*Interactive Distributed Systems*" (*Appendix A*) and the definition of the comparison criteria. The chapter concludes with the comparison of the frameworks in an overview.

3.1 Scope Conditions

Before diving into details of the questionnaire and the comparison criteria, it is important to keep the scope conditions in mind. The goal was to be able to recommend frameworks for specific use-cases with the input of future developers who need them.

When looking at the **target group** a distinction between the group this thesis aims to help and the group that is investigated needs to be done:

1. *Group this thesis aims to help*:
   The target group for this thesis are all people interested in web development. In particular university students, which are at the beginning stages of Full-Stack Development.
2. *Group that is being investigated*:
   The content of this thesis was taken as a basis for the course *Interactive Distributed Systems* at the University of Applied Science Offenburg. The course is for students from the media department, consisting of different study fields from computer science to media design over to media technology. Therefore the interests and viewpoints of the students of this particular course and the media department are very broad. This opened the opportunity to invest in a general overview of the students and the technologies in web development they are interested in. Through this thesis, it was possible to gain insights in what they look for in a framework and to examine those criteria in particular.

3.2 Questionnaire

Thinking of appropriate applications and materials for students to acquire knowledge of Full-Stack Development, the factor of already **existing knowledge** is of relevance. To be able to evaluate the students needs and interests, an empirical study was developed and executed. The form of a questionnaire was chosen and implemented with Google Forms [1].

The questionnaire contained 27 questions of which 19 were mandatory. For a logical **structure** individual headings for clarification of different topics were chosen. This showed a clear classification of questions, both for the later evaluation and for the students taking part. While developing the questionnaire the **orientation** of students with some, little or no knowledge of Full-Stack Development with JavaScript was kept in mind. Additional research was put into the current job market, to receive insights into what might be worth looking into for future graduates.

The entire questionnaire with all questions and results can be viewed in Appendix A. Not mentioned statistics will be examined in the criteria section (see 3.3).

3.2.1 Evaluation

Starting with basic questions, the evaluation dives deeper into existing knowledge and interests in Web Development, in particular Full-Stack Web Development, as well as impressions of the course '*Interactive Distributed Systems*'. The course comprises concepts of Web Development, the basic building blocks of the internet and offers practical insights through different laboratories, as for instance a lab experiment about the Internet of Things (IoT).

Overview

The majority of responding students (70 %) were studying the **bachelor degree** media and information (MI). This degree consists of a combination between computer science, management, design and media technology. They are great candidates for becoming Full-Stack Developers, because they have such a broad range of experiences. In addition, 15.2 % were studying enterprise and IT security (UNITS), which also has a potential focus on web security topics. 6.5 % were students of media technology and economy (MWP) with the option of extending to a teaching position. 5.4 % were **master students**, adding a deeper background to this questionnaire and lastly 2.2 % studying media design and production (MGP) (cf. 3.1).

All study courses have a study time of seven **semesters** (3 1/2 years). In the German university system a semester consists of half a year. Most students (37.0 %) were in their last semester of studies, 29.3 % were in their third year, 17.4 % in year two and 14.3 % in year one, including master students (cf. A.2). The questionnaire was advertised specifically for students interested in web development. Due to the global pandemic of Covid-19, many students took longer study times. This could have affected the results in general.

The average **age group** with 50 % of students belonging within the group was between 21 and 23 years. This corresponds to the number of students being in their last year of their study programmes. Closely following with 23.9 % was the group between 24 and 26 years. The age groups 18 to 20 years and 27 to 29 years both made up 10.9 % of respondents. Being 30 years or older, the smallest group was 3,3 %. None were younger or equal to 17 years old (cf. A.3).

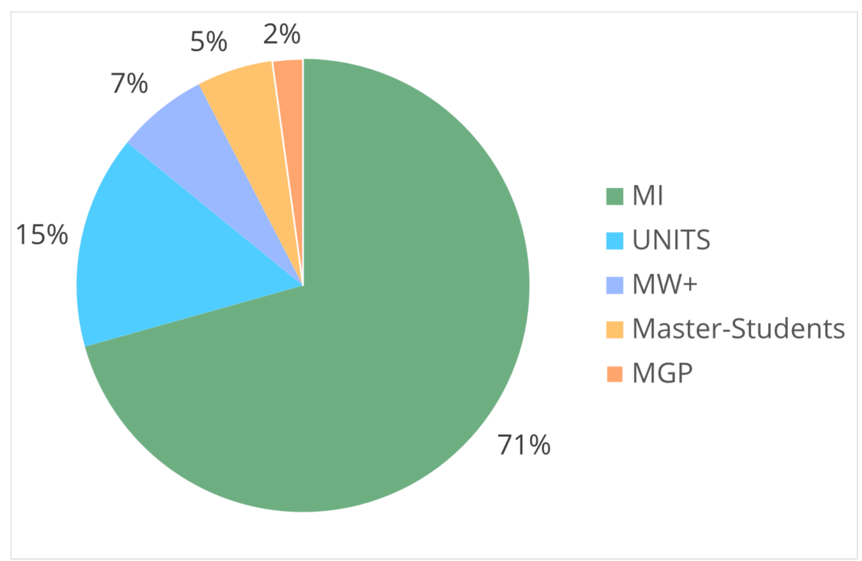


Figure 3.1: Study Programmes at the University of Applied Sciences Offenburg

As for the **Experience with Web Development** (cf. A.4), most of the students (28.3 %) had none. Students that had experience mostly acquired it during their practical semester (27.2 %), which is a mandatory semester during the study programme. Others worked in web development as a working student (25 %) or in an internship (7,6 %), are self-employed alongside their studies (18.5 %) or learned some in school. Also very common was following the Web Development path as a hobby (25.0 %), making web development a popular topic in the spare time of the students.

When asked about the overall **Development Fields**, the students were particularly interested in Web and Software Development with each 39.1 %. User Experience Design only was named specifically by 2.2 %, being included in Web Development itself. Following the diverse spectrum of study courses, many students did like the field of project management (30.4 %). Moreover Media Design / Media Management (19.6 %) and IT Security (18.5 %) are very popular as job perspectives. Some smaller percentages feature Mobile Development like Android (12.0 %) or iOS Development (8.7 %), making React Native a very interesting framework. Other development fields were Game Development (10.9 %), Search Engine Optimisation (SEO) and Search Engine Optimisation (cf. A.5).

To get a general overview on how many students are interested in which **Type of Web Development**, the tendency of interest was asked (cf. A.6). 33.7 % each were interested in Frontend and Full-Stack Development. 14.1 % chose Backend Development and 18.5 % were not interested in Web Development at all. The focus of this thesis therefore went to rather frontend-sided aspects such as Frontend-Frameworks, which – as already determined – have the most influence on the tech stacks featuring MongoDB, Express.js and Node.js in *Chapter 2.3*.

And lastly for the overview questions, the majority of students (specifically 52.2 %) attended the course **Interactive Distributed Systems** (cf. A.7). This leaves 47.8 % of students un-opinionated to the course contents, leaving space for neutral statements on their expectations to the course and for this thesis.

Programming and Development

As the knowledge which already exists is influencing the perception of learning, the general **Experience with JavaScript** was of interest (cf. A.8). The experience was divided into *at* and *outside of* university. At university, 76.1 % of students have gained experience with JavaScript. Because the course '*Interactive Distributed Systems*' is only mandatory for specific study programmes and can otherwise be chosen optionally, this is a rather high number. The experience thereby also could have been acquired through project works or other courses. The experience *outside of* the university context is lower but still high with 65.5 %.

There are many languages with different approaches. While JavaScript already showed popularity in other statistics, the overall **language experience** was explored to find out which concepts might be useful for students as analogies. The students had the most experience with HTML (95.7 %), CSS (90.2 %) and JavaScript (75.0 %). Followed by Java (80.4 %), C (75.0 %), PHP (50.0 %) and Python (46.7 %), all major languages specifically taught in these study courses are represented. Also interesting to highlight is that 20.7 % of students have used TypeScript before, supporting its popularity. Rather unusual language experiences are C# (30.4 %), C++ (30.4 %) and Go (6.5 %), resulting in a broad spectrum of diverse approaches like object-oriented programming and languages close to hardware level. JavaScript overall acquired high numbers in the questionnaire, which shows that there is a demand for it. Also, up-comping languages like TypeScript and Python might improve with their numbers in future courses (cf. A.9).

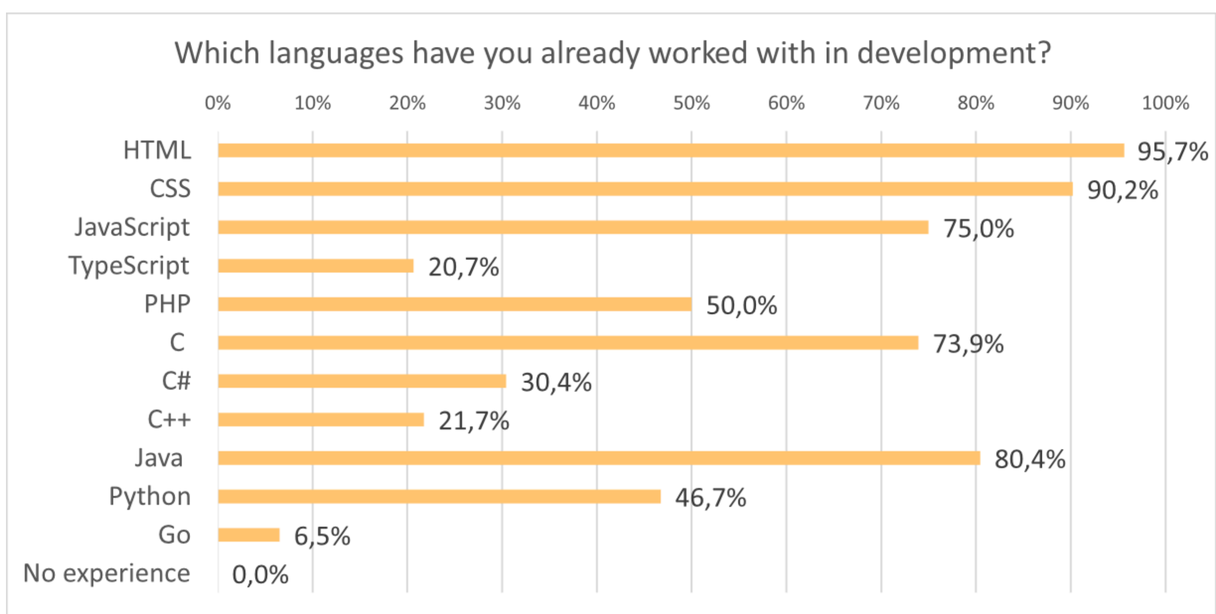


Figure 3.2: Language Experience of Students at the University of Applied Sciences Offenburg

Full-Stack JavaScript Development

Because the topic of Full-Stack Development is essential to this thesis, a number of questions were asked to explore the knowledge of the students. The results showed that 64.1 % of students are familiar with the **terminology** of 'Full-Stack Development', portraying that the trending topic is in the minds of the students (cf. A.10). As of the interest, Full-Stack Development tends to be a rather interesting topic with 50 % of students finding it interesting with a number of four and higher (cf. A.11). The scale was from one to five, were five was very interesting and one was not interesting.

With the high number of frameworks the opinion on using **JavaScript Frameworks** was important. 54.3 % of students had no particular opinion on the usage, having no experience or interest in frameworks. 23.3 % like using JS frameworks, whereas 7.6 % *only* use frameworks. In contrast to frameworks, only 2.2 % of students prefer sticking to Vanilla JavaScript (cf. A.12).

To specify the **experience with Full-Stack Web Dev**, 45.7 % of students responded to have no knowledge about it and 25.0 % had at least heard of it before. The rather low number of 21.7 % of students have gained some experience. Even lower was with 7.6 % the number students who work in the field professionally or regularly in projects (cf. A.13). This supports the scope conditions of Full-Stack Developers being at the beginning stage of their developer careers.

The determined frameworks from the statistics reflect a global scale of experiences. It was intriguing to see whether the knowledge of students corresponded to this experiences. Out of four given **Frontend Frameworks** – React.js, Angular.js, Vue.js and jQuery – 44.6 % have never worked with any of those frameworks. The highest number of knowledge was achieved by jQuery with 38.0 %, closely followed by React with 31.5 %. Angular is known by 15.2 % of students and Vue by 13.0 % (cf. A.14). Therefore it does not directly correspond to the statistics from the StateOfJS or DeveloperNation, but still shows that the main frameworks were of interest.

On the backend side, out of the four **backend technologies** Node.js, Express.js, Next.js and Gatsby.js, 47.3 % of students have never worked with any of those frameworks (cf. A.15). The runtime environment Deno specifically was mentioned by a few students, showing legitimate interest. This applies to predictions of for instance the StateOfJS, that Deno – being TypeScript supportive – is very interesting for the future and might even surpass Node.js as leading technology [17].

With specific **Knowledge of Tech Stacks** the questionnaire concluded the topic of Full-Stack JS Dev. 25.5 % of students have heard nothing about the specified tech stacks (seen in *Figure* 3.3). 59.6 % have heard or worked with the MERN stack, 44.7 % of MEAN, 21.3 % of MEVN and lastly 14.9 % of MEEN (cf. A.16).

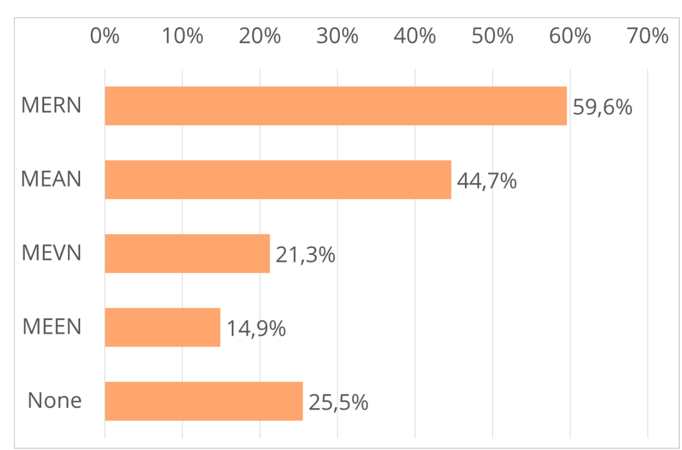


Figure 3.3: Knowledge of Stack Types of Students at the University of Applied Sciences Offenburg

The Course Interactive Distributed Systems

Asking students directly what they would like to learn often is a very effective way of getting meaningful results. Although being asked for the course of *Interactive Distributed Systems*, the following results are applicable for every learning institution or space, where people are beginning to learn about Full-Stack Web Development.

Possible **topics** which the students would like to have featured in a course were with 69.6 % JavaScript Frameworks in an overview (frontend as well as backend). The concept of Full-Stack Web Development was equally popular with 56.5 % of students wanting to learn it via a lecture or laboratory experiment. 34.8 % wanted to learn about different software architectures (e.g. monolithic vs. microservice) diving deeper into the programmers side. An lastly 33.7 % would like to know about the different tech stack variants (e.g. MERN or MEAN) (cf. A.17).

Specific **frameworks** that are interesting for students to try out in a laboratory experiment are React.js (61.4 %), Angular (30.1 %) and Vue (20.5 %) on the frontend side with React being the 'winner'. The popularity definitely is portrayed in these results. Node.js (53.0 %) won on the backend side with Express.js (15.7 %) and Deno (1.1 %) as other contestants (cf. A.18).

When the students were asked to add further comments on the **content of the course**, either the responses were specific or generic due to their knowledge about the course and web development overall. The conclusion of the evaluation of students will be done in the recommendations (cf. 3.2.2). The students seemed to be very interested in current trending technologies. This makes sense regarding the topicality and requests to be in touch with requirements of the job market. Also, the connections of technologies, an overview on how to behave with web technologies or how to group them were mentioned to be interesting to learn. More specifically, connections between individual areas were named as very important (cf. A.19).

Specifically mentioned was the **interest** in special APIs like the REST- and Graph-API, the concept of Web Assembly (Wasm), rendering methods, and the frameworks Swift, as well as the syntax extension JSX when using React. API design thereby seemed to be of importance to show how extensive a web application could be (cf. A.19).

Concerning **frameworks**, the students seemed to agree on that they tend to be opinionated and that learning the fundamentals is more important. React.js seemed to be of interest in particular. Some students would also like to learn about security aspects and overall development environments, in how to use them correctly and what there structure is like. Also, connecting to a database as part of the full-stack development process is important. Furthermore the process of how to build up a full-stack web application is of interest (cf. A.19).

Behaviour during Web Development

Looking at the interest in trends by students, 59.8 % of students would like to have a focus on trends, being up-to-date with the latest technologies. 27 % are neutral to trends and 13.0 % tend to stick to familiar technologies, which are tried and tested (cf. A.22).

Disclaimer: Other statistics of the Behaviour during Web Development as well as Learning Behaviour will be examined further in the following Section (3.3).

Learning behaviour

Getting to know **new technologies in the field of Web Development** happened for the students mainly by internet research (79.3 %) or YouTube Videos (79.3 %). This shows a dominant presence of online research. 58.7 % got to know frameworks in Forums such as Stack Overflow, 52.2 % in Lectures and 32.6 % by speaking to fellow students and coworkers. Additionally 21.7 % learned about technologies in Books and 18.5 % on Social Media Platforms such as Instagram. Learning Platforms like Udemy (2.2 %) and Learning by Doing (1.1 %) were not of interest concerning ways of learning new technologies (cf. A.23).

When **starting to use a framework** 18.5 % of students learn by starting to use the framework and experimenting on the go. 30.4 % of students showed the tendency to start directly while 26.1 % remained neutral. 10.9 % had the tendency to learn the frameworks features before applying them, and 10.9 % read the documentation before starting the project at all. The results show a range of opinions on how to get started with a framework, where a mixture of both is a good way to start (cf. A.24).

3.2.2 Recommendations

Thinking of materials and applications which could help improve the learning's of students at university levels, there were some trends shown in the evaluations of the questionnaire. The general **overview of possibilities** in web development and **illustrative examples** is very important. Besides Full-Stack Development with JavaScript, working with Content Management Systems seemed of interest for the students to get an impression on possible ways to develop an application. The concept of how to host a website was mentioned by students as well, to get a full overview on how to build a Full-Stack Website from the **beginning to deployment** (cf. A.19)

Understanding the **concepts of the frameworks** and actually being able to work with them seems like a big challenge to the students. The take away from this is to introduce an overview, like a "Stack 101: What is it, what is it used for, how is it applied." As most frameworks are opinionated and require knowledge of the fundamentals, it is crucial to learn the basics like JavaScript before actually going into the depths of a framework. Still an example of application of a widely used framework seems to be very interesting. The balance of useful details and too many insights needs to be found (cf. A.19).

Learning full-stack development in **learning institutions** such as at university is not that common yet. The focus on either frontend *or* backend is shown, resulting in learning only a "half-stack". As web technologies are changing very fast, being proficient in both front- and backend development needs to be addressed. Latest trends seem to influence the curriculum, where the unified approach to web development is the base and where academia needs to be paying attention to. [134]

Because JavaScript knowledge enables the possibility to become proficient in Full-Stack Development, it is fundamental to learn the the **JavaScript language** in particular. Furthermore it is a relief for students to learn only one language but to still have the freedom and knowledge of how to choose a technology stack. Web Development and the topicality of trends is important, and the students seem to want to learn more about it (cf. A.26).

3.3 Criteria

> *Developers are naturally curious and interested in new technologies.*
> – Stack Overflow Developer Survey 2022 [38]

Due to the variety of new technologies in web development, it is hard to tell which is the perfect technology stack to recommend for developers to use. There are **many factors** influencing the decision for a web technology [156]. To offer a common ground for comparison, appropriate criteria need to be determined.

The **StateOfJS 2021** listed an overview of the criteria important to their statistics. Amongst them were *documentation*, *developer experience*, *user base size*, *community*, *creators*, and *hype & momentum* [17]. These are very good indicators for how popular a web technology is. So far the focus was laid upon the beginner developer **developers perspective** with state of the art and relevant concepts having been presented. Now the criteria will be examined through the lens of a developer learning frameworks.

For every developer trying to stay up-to-date with JavaScript technologies, it might be overwhelming to switch approaches for each technology. Although natural curiosity is very good and can lead to improvements and a great community, joining with the **momentum** and following every trending web framework might not always be the best solution. npm sums up the discussion about why popularity, hype or momentum don't matter with this statement:

> "It's not our intention to suggest that everyone should always use the most popular framework. There are many metrics to help you decide which one to use, and frameworks vary enormously in scope and application. A framework that closely matches your specific application may not even be mentioned in this analysis but will still be the perfect framework for you." [157]

After careful consideration, empiric research through statistics and the questionnaire, the comparison criteria have been found:

- **Learning Curve**: The ability of how easy it is to learn a framework. The learning curve contains the documentation, tutorials and support of the community.
- **Maintainability**: The frequency of how often an application or tech stack in particular needs to be updated. Also dependent on the Long-Term Support and End of Life.
- **Modularity**: The interchangeability and reusability of code between projects.
- **Media Integration**: The support of integrating media. Dependent on bundler type, DOM type and existing or additional functionality.

These criteria were chosen as the base for the following comparison. Explanations for each criteria will now be given.

3.3.1 Learning Curve

Learning how to code is a very unique experience, with people using a variety of tools and resources to build their skills.
– Stack Overflow Developer Survey 2022 [38]

Learning a new technology is often difficult and takes some time. There are different **approaches of learning**, which also depend on the learning type of a person. A point certainly influencing the ability to learn a framework is its documentation. They are the main background of the comparison of all three frameworks. Many developers educate themselves through online material such as videos, blogs, forums and other online resources, instead of books and institutions. The need for a good support in learning, a good technical documentation and a sense of community is essential. This is why the first comparison criteria is the learning curve. [50, 61]

A learning curve is the **graphical representation** of the relationship between proficiency of a user and the amount of time they spent acquiring the experience. It shows the progress of learning certain information or acquiring certain skills, during a specific amount of time. To learn something, it is vital to understand that you have to get over the learning curve. It shows what phases the learning curve has and what is needed to accomplish the 'climbing' of the curve to get proficient. [158]

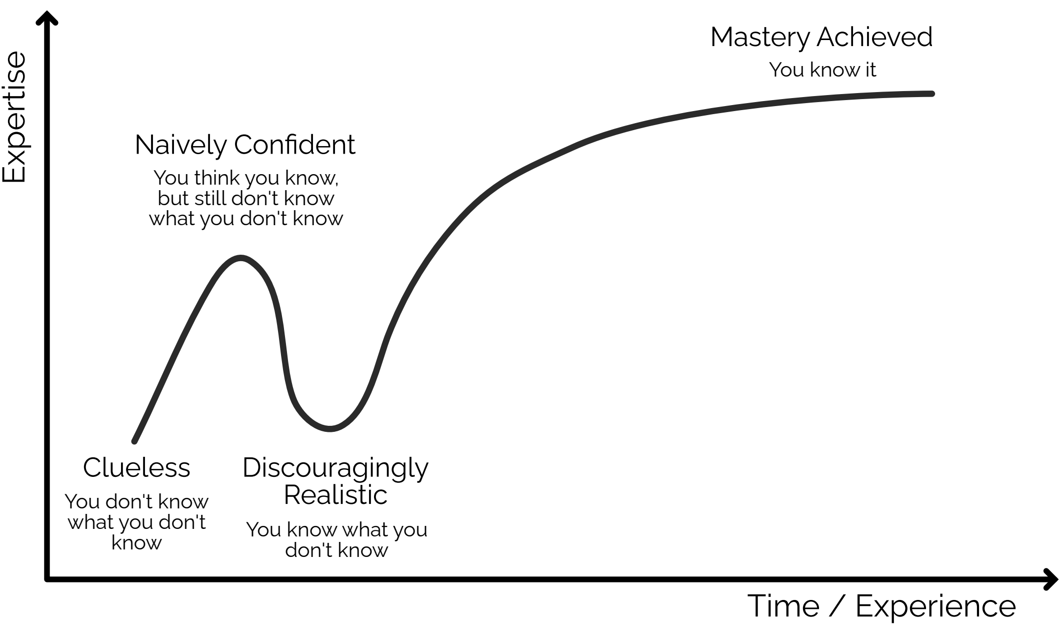


Figure 3.4: Example of a Learning Curve, adapted from [158]

As illustrated in Figure 3.4, a learning curve may include specific stages. On the horizontal axis is the **time or experience** acquainting oneself with a technology. On the vertical axis the **expertise**. The goal is to increase the expertise to its maximum, therefore being proficient in a technology. In this example the end level is called 'Mastery Achieved'. [158]

Common **relations** are that expertise increases with increased experience over time. The steeper a learning curve is, the faster a technology can be learned. Vice versa, a flat curve means that it takes longer time to learn. A common misleading interpretation is that a steep learning curve is equal to 'hard to climb' and therefore difficult to learn, although steep means a quick progress. [159]

As Figure 3.5 shows, every developer has a different way of learning. Being at university, students belong to the 41.1 % of undergraduates **learning to code** with the help of courses and teachers. Still 63.8 % of developers teach themselves how to programme, which is a high number by comparison but may be explained through the reading of documentation or their interest in the field of computer science itself. 30.1 % of developers learn on-the-job and 36.5 % take online courses such as Coursera [160], LinkedIn Learning (formerly Lynda.com) [161] or Udemy [162]. [34]

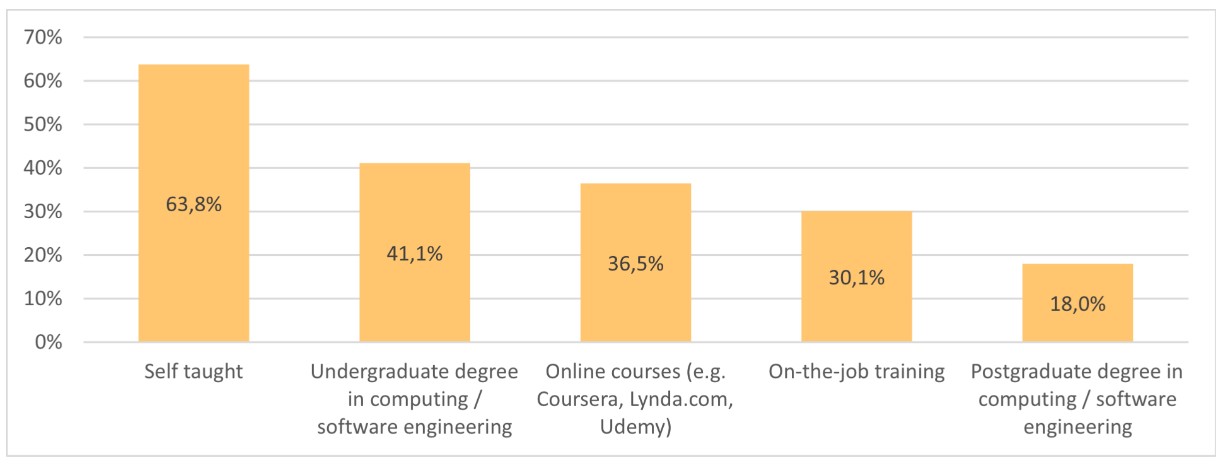


Figure 3.5: Top 5 ways in which developers learn to code, adapted by Slash Data's Developer Nation Q1 2022 [34]

The Developer Survey by Stack Overflow 2022 furthermore illustrates that respondents older than 45 are most likely to have learned from books, while younger ones primarily learn online. Especially the **age group** under 18 uses online sources and are most likely to have learned a technology from online courses or certifications. Amongst **online resources**, most people learn with the help of technical documentation (88.13 %), with blogs (75.35 %) and tutorial videos (59.92 %). Although some learning techniques vary in different age groups, consistently on top of the list is the possibility of learning how to code through school (i.e., University, College, etc.). This is a sign that it is very important to acquire a good base of techniques at learning institutions. [38]

The evaluation of the **questionnaire** (in 3.2) showed a similar impression. 70.7 % of students learn best through Online YouTube Tutorials, 43.5 % learn by implementing projects, closely followed by 42.4 % of students reading the official documentation. 33.7 % take Online Courses like LinkedIn Learning or Coursera), 31.5 % learn with the help of friends while 25.0 % rely on person to person courses such as at university. Only 13 % of students prefer to learn by books. Additionally to the quality of learning materials, factors such as interests and existing knowledge are of relevance and certainly influence the learning process (cf. A.24).

A last addition to the learning curve is the **community support**. Thanks to the interconnected nature of the JavaScript ecosystem, the scope covers a broad range of projects that represent a massive and diverse developer and end user community. When selecting a framework, the more innovative and more up-to-date one probably will be chosen. The community contributes to innovation and improves the experience of everyone using the framework – especially when being an open-source project. Furthermore, they answer with solutions in forums, write Blog articles about special features and contribute a stable environment of learning. [82, 104]

In conclusion, the criteria of a learning curve is influenced by the official documentation of frameworks, the variety of different learning formats officially offered, the usability of the given content and the overall community support.

3.3.2 Maintainability

> Simplicity is the ultimate sophistication.
> –- *Leonardo da Vinci*

Maintaining a website is important and takes up a different amount of time depending on the website and developer maintaining the website. The **maintenance** includes looking for updates, upgrading to recent versions, checking the dependencies and keeping the content up-to-date. [64]

As a website has different content and different modules in use, depending on how big the application is, the effort of maintenance can vary. Simply updating a project through a **package manager** hence seems like a very good option, especially for large projects. With many updates, every part of the website needs to be checked again and that takes additional time. Having this in mind, it is best to have a stable version and good community support if challenges on upgrades occur. [113]

When choosing to follow every **trend** in web development, testing out different versions and upgrading a website every few months, the demand of high-maintenance exists. Because the topicality is important in competition on the market, the students of the questionnaire (in 3.2) were asked if they are interested in following trends or prefer sticking to familiar technologies. 59.8 % of students would like to focus on trends. This is viable because they are looking for what the job market is looking for. Taking the risks of following a trend might be worth it for some, because the new technologies feature big advantages to makes the development process easier. Taking the risk into account and not following every trend might be more useful, to have stability (cf. A.22).

Every tech stack includes numerous technologies that need to be up-to-date for securing security issues. When a module or framework is not supported any longer, **security vulnerabilities** might occur. This leads to damage of the website and website owners and needs to be prevented. Prevention can happen through the consistent maintenance. [17]

The students were asked how often they thought one would have to maintain a website, or how often they would prefer to maintain a website. Most students (48.9 %) voted for weekly updates, followed by 32.6 % of students for monthly, 15.2 % for every two to three days, 2.2 % for yearly and for 1.1 % daily. A comment addressed in the answers of the questionnaire was that this particular question could only be answered with further information regarding type and content of the website as well as the **update cycle** of the technologies used in the tech stack. These are valid comments and also need to be considered. The overall impression of students maintaining a website weekly is therefore reasonable (cf. A.21, A.26).

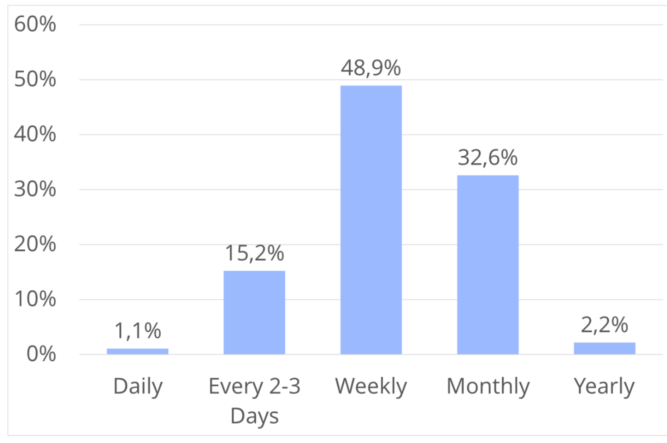


Figure 3.6: Expected frequency of maintaining a website by Students of the University of Applied Sciences Offenburg

As already mentioned by the students, the maintenance really depends on the updates of the technologies used in the tech stack and therefore cannot be generalised. If a technology has updates often or offers a **long-term support** version has an influence on the maintainability (cf. A.26).

In conclusion, the factors of package manager support, trends, version management, and LTS versions influence the maintainability.

3.3.3 Modularity

> *Keeping programmes under control is the main problem of programming. When a programme works, it is beautiful. The art of programming is the skill of controlling complexity. The great programme is subdued—made simple in its complexity.*
> – Haverbeke [6]

Solutions of code problems often find their way in registries such as npm or end up as an answer to questions in forums like Stack Overflow. It is very useful to be able to reuse already written code solutions from others or being able to use own code pieces as modules. The **concept of modularity** plays an important role in web development.

The questionnaire in 3.2 showed that the students agree on the **importance** of modularity with 50 % interested in using a high amount of modularity (number 4-5). Only 1.1 % of students are not interested in using modularity at all (number 1). The scale was from one to five, were five was 'very much modularity' and one was 'no modularity'.

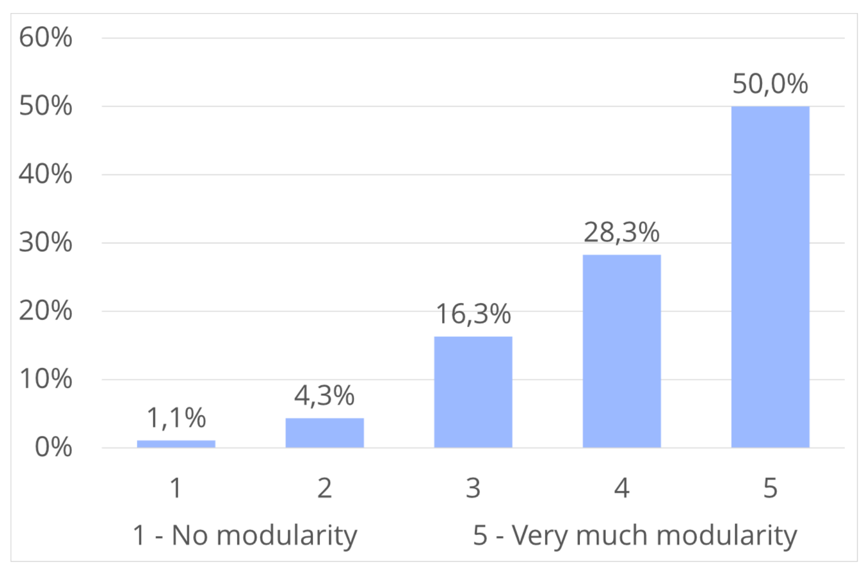


Figure 3.7: Interest in the concept of Modularity by Students at the University of Applied Sciences Offenburg

In the State of the Octoverse, they highlight the importance of modularity. Project internals like entitlement procedures, access restrictions, or information fragmentation might frighten developers to use modularity. But it generally **increases performance** at work up to 87% when reusing others code. They also state that friction-less reuse of code is becoming more powerful for open source projects, as they become to times more performant than projects with more friction. The friction is included through slow processes or multiple approval layer. [32]

Modularity is an important concept of programming and gets more and more attention through Node and npm. Using npm enables to add packages by other developers. Reacting to the popularity that modules are gaining, they are furthermore becoming an official feature in the EcmaScript standard. [163]

Webpack vs. Vite
Containing different modules and packages might take some time to load for an application. To make it more manageable and faster in the browser, **webpack** can be used. Webpack is an open-source **static module bundler** for modern JavaScript applications. It can be executed in computers and servers outside a browser environment, because it runs on Node. It can transform resources, bundle and pack them together. Thereby, it creates a dependency graph with the component of a project (also called entrypoints) and then combines them into one or more bundles. These bundles are static assets to service the content and can be configured in all its detail with webpack. This makes frontend-elements very modular and media integration easily manageable. React and Angular both use webpack. [106]

The official build tool for Vue is **Vite**, a lightweight and extremely fast frontend tool (cf. 2.2.3.3). It bundles the code just like webpack and outputs highly optimised static assets for production. Through pre-bundling it decreases the performance bottleneck of high amounts of JavaScript modules and files. Vite is also available for React. It improves the developer experience by providing ES modules in the browser and compiler-to-native bundler tools like esbuild. Futhermore, it offers support for syntax like JSX and TypeScript. The StateOfJS declared in its survey, that 2021 has been 'the year of Vite'. It had 98 % satisfaction and 30 % of usage in its first year, which is an amazing result compared to other technologies' first year. [17, 97]

In conclusion, the modularity is influenced by existing code bases in npm, the building tool and the overall capability of the technology to enhance modular programming. Another aspect is how little or much of the technology can be used.

3.3.4 Media Integration

When opening a website URL, the **page speed** and **page load time** are important criteria that influence our perception of the user experience. Page speed refers to the measurement of time it takes for the content on a page to load. It can be measured in "page load time" or "time to first byte" [164]. Looking at E-Commerce purposes the speed can also effect the bounce rate and rankings of a website. That it is a crucial point for attracting and retaining costumers. Ideally, a website should have loaded in less than two to three seconds. [164, 165]

A common demand for websites is that they are fully interactive, have images, videos, sound, animations and graphics, to create a full multimedia experience. But the more media assets included, the more difficult it is to maintain a good page speed. Another "must have" of website is the **Responsive Design**. It describes the layout and content adapting to the size of the screen. For example a button on a mobile device can't simply scale down from desktop to mobile, it needs to be big enough so that the user can tap it easily. [164, 165]

To influence the performance, different methods such as the **minification** of files and many reusable components can be used. Additionally, the **DOM type** (virtual vs. incremental) is influencing the storage usage, hence also the loading of the media. As already highlighted in the previous section about modularity (cf. 3.3.3), assets will be transformed into into a bundled form for the browser. Through bundling an application supports many different assets including fonts and stylesheets as well. [165]

In conclusion, the media integration is influenced by the type of bundler, the type of DOM, the difficulty of style integration, existing npm packages and customisability.

3.4 Comparison

The comparison was done by looking at the comparison criteria for each framework. The results are shown in a summary at the end of this chapter. Most textbooks for academia cannot keep up with the rapidly changing technologies in web development. Therefore, online resources such as tutorials, official documentation and online courses determined the following results. [134]

3.4.1 Detailed Comparison

Each criteria will now be examined in the frameworks React.js, Angular and Vue.js. Consequently comparing the web technologies to each other. The frameworks will be investigated in their order in the list of contents. In Table 3.1 an overview of the characteristics and basic concepts of frontend technologies used in the JavaScript-based stacks (cf. 2.3.2) is shown. They help evaluate the criteria:

	React.js	**Angular**	**Vue.js**
Creators	Facebook	Google	Evan You
Type	Library	Framework	Progressive Framework
Source Lang	JS	TS	JS / TS
TS Support	Yes	Yes	Yes
Components	Functional and Class Components	Component (@Component decorator with JS, html template and styles)	Single-File Component (SFC with JS, html template and styles)
DOM	Virtual	Incremental	Virtual
Own CLI	No	Yes	Yes
Features	JSX, props, stateful components, Hooks	Directives, Dependency Injection	Different APIs to define logic

Table 3.1: Comparison of Frontend Frameworks

Learning Curve

React.js

React.js has a **docs and tutorial** page directly on the official website (reactjs.org). The landing page shows many impressions and explanations of what is beyond this technology (e.g. a simple components, stateful components), making a good introduction of the concepts to the user. It furthermore animates to get immediately started or to take a tutorial. The docs page shows the basics on how to get started, main concepts and advanced guides. As concepts such as JSX, the virtual DOM and the component-based approach are a lot to take in, being introduced to them by a **step-by-step guide** is ideal. [46, 69]

Having the rather fast **pace** of their own official documentation in mind, React featured a specific link to an overview of React by Tania Rascia [77]. There the most important concepts are explained in a beginner-friendly way. This is recommended to begin with before continuing with the main documentation, making it very **accessible** for different learning stages of a developer. Other resources are occasionally linked to in the text. [46, 77]

A page with resources dedicated to designers exists and free / paid courses are linked to on the community page. The React **community** is very big an supportive, being represented in Stack Overflow, discussion forms, and being on Twitter. This is very helpful when questions occur. Small example projects, podcasts and videos are also referred to on the community page. [46, 166]

In addition to a very good documentation supplied on the official React website, React offers a **learning by doing** approach. In a tutorial the interactive game tic-tac-toe can be build. The guide is complemented by explanations to understand React and its syntax. To get a first impression of their product, they additionally offer simple first examples through a live editor. [46, 69]

Online Playgrounds such as Sandboxes enable quick tests and convey important impressions. Sandboxes let the user directly try out to programmes before installing the frameworks itself locally. React is represented in the main online sandboxes and therefore shows great presence. The Quick start option of React via the command "create new react app" makes starting with an own project possible within a minute. React also has **recommended toolchains** for larger applications, which is very good for beginner developers who want to advance their applications. [46, 69]

For all above mentioned reasons, but especially for the main concepts, the **learning curve** of React seems to be rather **flat** - especially in the beginning. Once a basic understanding of concepts like hooks, props, states etc. is granted, the learning curve steepens.

Quick Start Options

Before continuing with Angular, the **Quick Start** options will be investigated. The possibility of a quick start project setup is attractive to beginner developers as it is very fast to start beginning to develop. Having a look at the the frameworks, here is an overview of the basic setup when starting with the Quick Start projects.

	React.js	Angular	Vue.js
Command	npx create-react-app test	ng new test	npm init vue@latest...
Default Port	http://localhost:3000/	http://localhost:4200/	http://localhost:5173/
Customisation	No	Yes	Yes
Start Page	Simple, one link	Extensive, many links	Sophisticated, many links
Run App	npm run start	ng serve	npm run dev

Table 3.2: Comparison of Quick Start Options

These quick start options make starting with a project easy and fast, but is abstracted from the user with configurations that might not have to be added for some projects. It is a great way to work with a proper setup and use it to advance projects as they grow. Vue and Angular feature different options of **customisation** when starting a project (cf. row three). React just installs it. Angular offers customisability. And Vue additionally to customisability has added a lot of comments in where to put own code. This makes it more feasible for the user to comprehend the programme.

The projects are rendered automatically by the build engine on the **localhost**, which is he loopback-address. It addresses the own computer and is mostly used for testing before building it in a production environment. Every system thereby has a default Port number as indicated in the Table 3.2, which can be changed. Commands like npm run start can be altered in the config file changing the script. The different setups definitely show a similarity. Vue and Angular thereby score points in customisation.

Angular

Angular has a **features and docs** page included on the official website (angular.io). The landing page shows the main advantages of Angular: being "developed across all platforms", "speed and performance", "incredible tooling" and "being loved by millions" of developers. In addition to the features and docs page, an independent page with external **resources** exists. The resources are divided in three parts: Development, Education and Community. This enables the user to choose his own interests and level of learning and networking willingness in forums. The user gets a simple overview and is animated to use it. [52]

A **tutorial** on a sample project of creating a store with products can be followed, introducing the concepts of components, templates, directives and dependency injection (cf. 2.2.3.2). Angular also gets provided in **sandboxes** and features a page on **best practices** to use in projects. The documentation is very vast and advanced. It offers different approaches to learn the main concepts such as Components, Templates, Directives and Dependency Injection (cf. 2.2.3.3). [52]

The complete rewrite of the technology contained a **new syntax and many new concepts** compared to its predecessor AngularJS. The developer community wasn't and still isn't completely convinced about this framework. The developer Dave Gavigan states in a blog entry [167]:

> "Teams are googling Angular and seeing these breakup letters and getting scared to head in another direction. Its unfortunate they don't take time to evaluate the project for themselves and see all the progress the team has made of recent." [167]

Although the start of the launch was rough, the Angular **community** has established itself through high contribution in blogs and forum where they contribute their free time to help improve the Angular experience. There is a community on Stack Overflow, Discord and many events or meetups are hosted. Google as a big company also portrays a certain integrity, being maintained by experts of one of the biggest tech companies. This influences the trust of JavaScript developers in the overall concept of the framework. Because it has this integrity, it is very attractive to developers. It will continue to be updated and relevant. [82]

Due to many new concepts like directives and dependency injection, the start of developing in Angular takes longer time compared to React. Although documentation is well-written and showcases a lot of examples, it can't keep up with the **learning curve**. The learning curve therefore is more flat, because it takes longer time to learn Angular than React.

Vue.js

Vue features a **docs** page with a guide, a tutorial, examples, a quick start page and documentation on the current version of Vue on the official website (vuejs.org). They present the essentials when starting to develop with Vue, such as creating an application, template syntax and reactive state. Its clean and fresh design is appealing to the user, for instance with the toggle button to dark mode directly on the page and an introductory video. [50]

As one of the concepts Vue uses different **API styles** like the Options API and composition API. They also feature an own page on the website. Developers might have preferences in learning to code. A developer with an object-oriented background might prefer the Compositions API. To be able to **switch** between them easily, Vue has both coding styles provided in the documentation with code examples. This offers a clean code overview and encourages to learn about the different styles. This makes it very flexible. [50, 94]

A collection of official Vue online courses can be found on **Vue Mastery**. The website offers weekly vue.js tutorials for becoming a professional Vue developer. The learner thereby is teached by professional developers, such as Evan You itself, the creator of Vue. The collective database with video tutorials, online classes and cheatsheets has three different price models: The free account comes with lessons and cheatsheets, while the priced accounts have premium courses, new videos every week and exclusive content. The open-source accessibility of the platform itself is given through the official documentation. Offers such as Vue Mastery are very intriguing for beginner developers, as they have a playful approach and good edutainment. Although being a smaller player compared to Google and Facebook, the Vue **community** is very strong. Vue self-claims to be approachable due to their world-class documentation. The popularity is enhanced by many forums and interesting models such as Vue Mastery. [50, 168]

For the factors mentioned above, Vue tends to be rather easy to learn having multiple approaches of learning and a very strong, dedicated community. The **learning curve** is rather steep, being able to build up frontend very quickly. That being the case, Vue is perfect for small projects where a frontend needs to be build fast. Progressively enhancing the logic beyond is also possible.

Final Learning Curves

All frameworks show a lot of opportunities to learn the concepts behind them. They all offer great documentation and a **variety** of different learning formats. But there are some differences in the learning curves as seen in 3.8. The flat curve from React is showing, that is is easy to learn the basics, hard to learn the main concepts and very difficult to gain proficiency. Angular is even harder to learn, therefore the curve is more flat than React. The best result scores Vue with a rather steep learning curve compared to the other two technologies.

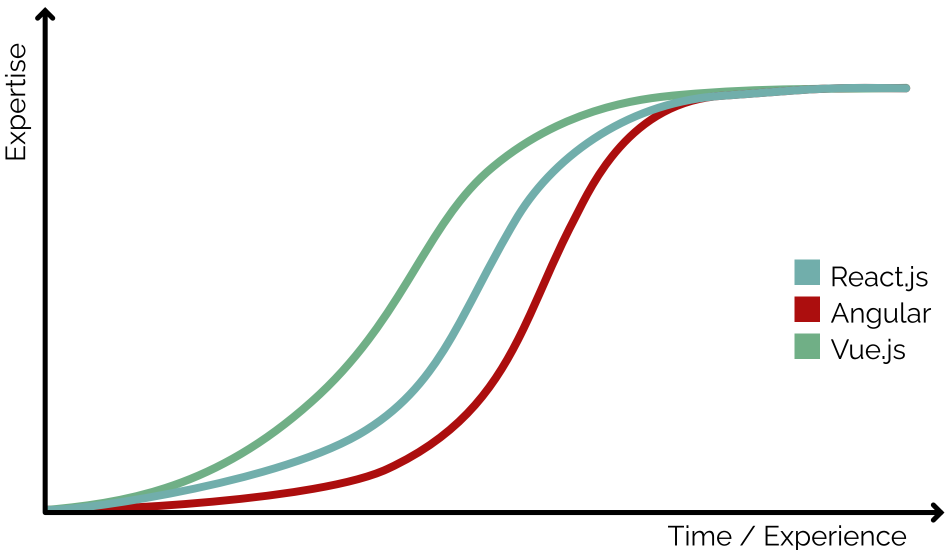


Figure 3.8: Learning Curves from React.js, Angular and Vue.js

Although many learning possibilities exist, the **concepts** of all frameworks are very complex at first. If there is no previous knowledge of vanilla JavaScript, it is strongly advised to learn JavaScript first. This qualifies for all JavaScript frameworks.

Note to Reader: This is a subjective opinion, which can be different from developer to developer. As this thesis had the beginner perspective in mind, the curves are aligned with this observation.

Maintainability

Package Manager Support

Keeping the different JavaScript technologies up-to-date is a challenge, which gets eased by npm. Looking directly at the **package security**, npm has constructed a useful command for maintaining the dependencies. Executing the command npm audit will scan the "package.json" file or global packages (with '-g' flag) and return a report with any known vulnerabilities in addition to how to remediate those vulnerabilities. The quick answer is executing the command npm audit fix, which will cause npm to update vulnerable packages to the newest version of the package with no reported vulnerability in it. This keeps track of severe vulnerabilities in installed modules. It can be used React, Angular and Vue, because all can be accessed through the npm package manager. No particular differences are existing. [71]

Long-Term Support

As seen in the overview of frontend framework versions (Table 2.2), the long-term support versions of Angular and Vue are set for usually a period around 18 months. React has no official **LTS versioning**. On that case, it is being rather unpredictable for development arrangements. With that being said usually versions get supported for six to nine months across all technologies, therefore having no differences for this part of maintainability. An **EOL** of a framework is a sign that developers should focus on a different framework. But for now, none of the frameworks are announced having an EOL. [63, 84, 169]

React.js

React is following **semantic versioning**, having dedicated supported releases channels on their website. New features are released in minor versions, patches are made if there is any security vulnerability. Any release thereby can include bug fixes. Since the introduction of React.js in 2013 it has continuously grown and kept its place in front of other frameworks for over five years now. Because it is heavily used, many custom solutions are deployed in npm, making it very easy and flexible to maintain. Because React is a library it is dependent on npm, but possibilities of support exist. The developer needs to individually keep track of new module version releases. [157]

There are also lots of possibilities to **contribute** in the maintenance of changes. React has an own blog as official source for updates from the React team. Hence, release notes and deprecation notices are always up-to-date. For the social media-savy, they have a Twitter account. For the rest of changes, they have a detailed changelog for every release. If older versions of the library are being used, the documentation of older versions can be found on a separate page. The official documentation contains the latest stable version of React. [69]

Angular

Angular is designed to make updating as straightforward as possible, compared to its predecessor AngularJS. The developers can for that reason take advantages of the latest developments with a minimum of effort. Angular major releases are supported for 18 months, splitting up in 6 months of active support and 12 months of LTS support. It follows **semantic versioning** and has an update guide available. [84, 169]

Like React, Angular shows different possibilities of development. As upgrading versions always is a bit risky and time-intensive, Angular 9 was the first release of the framework prioritising an **easy upgrade path** between major versions. This sets a new perspective on backward compatibility. [82]

In case of the switch from **AngularJS to Angular**, many websites (even pages from Google itself) still use the older and riskier version AngularJS because it is seems to be too much effort to transfer the system and content. A good transfer takes time and cannot be hurried, because then mistakes happen and security risks are available again. Because of the groundbreaking changes is Angular, many developers started to **switch to other stable technologies** like React or Vue for having a dependable framework. This enhances the importance of good communication and guidance from the official websites of the frameworks. Angular might not recover from the decrease in numbers. As seen in 2.18 by StateOfJS, they are now under the category of technologies that are harder to recommend. Given the time of transition, Angular has a tough time to go through [3, 82, 167]

Because Angular is an **opinionated** framework that already includes lots of functionalities (e.g. Angular Router), the maintenance of those official functionality is led by the Angular team. This means stable versions are in use and the developers can rely on the official support team. [52]

Vue.js

In contrast to the other technologies, Vue releases new patches when a release is needed. It follows **semantic versioning** as well, but has **no fixed release cycle** and consequently is a bit unpredictable. The developers accordingly need to check for update news themselves. But the Vue developers react to needs of the costumers very quickly, so that the update cycle also is having minor version released every six months. Major versions are expected every year and the LTS then automatically becomes 18 months changing the version to 'maintenance' mode. [169]

Jumping on the train and following the popularity of the **trending framework** might lead to a lot of maintenance. The transfer of technologies isn't straightforward, due to the difference in concepts. Having a lot of community support compensates rather unusual update cycles. Much like Angular, already included functionality in the **progressive** framework is officially supported and therefore not needed to be individually checked. Because the frameworks is very intuitive and has a great design, it fits the needs of the JavaScript developers very well. [3]

Conclusion on Maintainability

All frameworks have semantic versioning, have no EOL and get supported by npm for updating modules. React has limited proficiency in maintenance when using modules, as they individually need to be checked. Angular provides support for included functionality, as well as Vue which also features many included functionality. The more external modules and functionality are used, the more updates are needed. The best framework to use in terms of maintainability is the framework that doens't need to include external modules for the application. Angular and Vue therefore have a slight advantage due to the inluded functionality.

Modularity

The role of Node.js

When using Node, the modularity is already very good. It offers the opportunity to install npm packages and modules to every project. The **registry** therefore provides functionality and modularity to every framework. Consequently the differences in modularity lay in the communities and vendors maintaining the modules and the overall structure of the frameworks (e.g. type of components). [3]

React.js

React is a library with many options to choose from. The developer can choose between adding little or much of the **library** to the project, on that account being very flexible and un-opinionated. A drawback is, that features like Routing needed to be added externally as **tools**, because they don't come build-in with the library. Rebuilding the setup of tools in every application might result in consistency issues across applications. By using npm it is very easy to add modules to a React project. To create consistency, there needs to be an agreement on modules. There are many node modules included in the npm registry. [3]

Through the component-based design, React furthermore allows to **interface other libraries and frameworks** within the React application. This makes it very modular and highly wanted by experts who are already using very specific libraries. In-between functional and class components, the only restriction of modularity is that a return( ) statement can only return one JSX element. This means that mostly of the times, an unnecessary "<div>" is around the important code. React as well is great for microservices due to its **highly-scalable** design, being of interest for DevOps. With the adoption of **React Native**, Android Development is becoming very interesting as well, because modules simply can be re-used. [3, 46, 68]

Angular and Vue are TypeScript based. React is using JSX. The syntax of declarations is different, an the **components** feature different approaches (e.g. class and functional components in React vs. Component Templates in Angular vs. Single-File Components with different APIs by Vue). The web bundler **Vite** enhances the performance of Vue compared to React and Angular immensely through being highly optimised for static assets. The added TypeScript and JSX support thereby makes it very approachable and could be used in every framework. [95]

Angular

Angular has own toolset included in the rich framework. It is **opinionated** and constricts some dependencies, therefore having no options to use less of the framework than it offers. Having the functionalities at hand, it is very easy to use them. Through the extensive collection, no further libraries are needed. Features like routing, client-server communication and forms management are already included. [84]

Vue.js

As a progressive framework, Vue starts with a core framework of functions and allows adding utilities such as testing and routing with an **extensive library**. It needs additional tools, but has a rich ecosystem of libraries supported by the community. The Vue creators state: *We believe there is no "one size fits all" story for the web. This is why Vue is designed to be flexible and incrementally adoptable* [94]. Official Libraries such as the Vue Router ease the progressive enhancement of the application. Vue can be used to find an optimal balance between stack complexity, developer experience and performance. [94]

Final Modularity Evaluation

With all technologies using a component-based approach, the overall modularity is given for each. The differences lay in the differences of modules available. React has many modules available, less included because it is "just" a library and therefore high flexibility. Angular has already included many modules, there are less available by npm. This constraints its modularity. Lastly, Vue offers many official libraries. Due to the possibility of progressively adding functionality and enlarging the application, the modularity is highly flexible.

Media Integration

Bundler Type

React, Angular and Vue all use a bundler for the media assets which get integrated. React and Angular are using webpack, whilst Vue is using Vite. As examined in 3.3.3, the media assets are getting bundled through them. Through these bundles the page load time is improved. Vite has a slight advantage due to its high optimisation for production. Therefore Vue tends to load assets slightly faster. Page Speed further can be improved by minification of files. [98, 106, 165]

Differences in DOM

React and Vue use a **virtual dom** (based on efficient diffing algorithm), while Angular uses an **incremental dom** (based in instructions). Both approaches only re-render the components that are changing. The virtual DOM requires an interpreter, the compile time tends to be longer. Both can be used outside of the technologies as well. In the incremental DOM the advantages are that it is tree shake-able and has a low memory footprint, due to the usage of a set of instructions. This means it is **optimised for storage**, which is especially helpful for mobile devices. Both approaches have the advantages of very fast whilst computing the DOM changes. For large projects, where many different components need to be rendered, the incremental DOM approach looses against the virtual DOM being more-time consuming to calculate the instruction set compared to the diffing algorithm. [69, 88]

Frameworks

React has lots of modules available. For instance many sound modules can be found in the npm registry. Angular has an own toolset included in the rich framework, making it easy to integrate different media. Due to the massive community support, many modules with media integration are available for Vue. They are supported and really up the game for an easy and fast creation of an application. [116]

Style Integration: Fonts

Remembering the Button Example, which was examined through the different frameworks in the frontend section 2.2.3. It had been some style changes. They differed between simple **style changes** of the heading, the button itself (colour, margin, padding) and a box-shadow hover-effect. Also a new font-family called "Raleway" was integrated. Between the different frameworks, the ways the style could be applied mainly varied in syntax.

On a simple HTML page, adding fonts locally is rather easy. The downloaded font file, usually the TrueType (.ttf), gets added to the assets folder. Then the @font-face gets set to the default style for all (*) in the style.css. As a last step the style of the element e.g. <h1> need to be set with the CSS-attribute font-family: "Raleway". With React and Vue this was no difficulty. It pretty much worked like the vanilla JavaScript version. Angular however was slightly different, because the css selector changed to :host. Vue really showed an advantage with the "scoped" css attribute, which can be set in the style of the component. Applied in the style of a SFC the styles only influences the component an no other elements. [95]

Conclusion on Media Integration

Different prerequisites such as type of bundler and DOM influence the smooth integration of media assets. Vue has shown the best result due to the usage of Vite as and a Virtual DOM. It thereby offers the scoped attribute to only style a particular component. React and Angular also offer a great integration. All three frameworks have the ability to be personalised and adapted to the interactiveness of the application.

3.4.2 Overview of Comparison

Recommending a framework to learn or use in general is difficult. Through the detailed comparison many aspects, especially important to students, were revealed. The table 3.3 shows an overview of the comparison. Recommending a framework depends on **different factors**. Decisions like whether to build a small or big application or the use-cases of the application are need to be kept in mind (cf. 2.70). The decision also depends on criteria such as how the developers enjoy creating and what the industry needs. React and Angular are developed by the two industry giants Facebook and Google. Vue is developed by Evan You and an Online-Community. They all seem to have different documentations and different approaches, but still similarities like the virtual DOM or components were visible.

	MERN (React.js)	**MEAN (Angular)**	**MEVN (Vue.js)**
Learning Curve	Flat, hard to learn	Flat, difficult to learn	Steep, easier to learn
Maintainability	Fixed update cycle, flexible modules	Fixed update cycle, included functionality	Updates when needed, many features included
Modularity	Many modules, less included, high flexibility	Lesser modules, included functionality, constraint flexibility	Many libraries, progressive approach, high flexibility
Media Integration	Good, many modules	Very good, included rich functions	Good, many modules

Table 3.3: Comparison of JavaScript Full-Stacks (MERN vs. MEAN vs. MEVN)
(Technology Stacks dependent on Frontend Frameworks)

The documentations all were very well-written and offered support for different learning types. Although good usability of content was given, the **learning curve** was heavily influenced by the amount of new concepts a developer had to learn. Every developer has preferences and understands concepts faster or slower than others. It takes time to understand, and even after profession the learning doesn't stop. Vue was best in learning, before React and Angular at last place.

The **maintainability** and popularity of frameworks is heavily influenced by version and community support. All three technologies are supported and will be supported for the next couple of years. No EOL is in sight, letting the developers choose freely which technology they would like to explore. Angular still is recovering from the EOL of AngularJS. Many developers hence seem to be more interested in the popular and well-know approaches of React and Vue. The popularity of Vue in comparison to React and Angular showed what the requirements for frameworks (such as design, online courses etc.) for the JavaScript community are. It was found out, that the frameworks Angular and Vue, which already have included functionality, are easier to maintain. The reason was that React as a library needs to include special functionality through npm packages, which are not always maintained or up-to-date.

With the different component-approaches, every framework showed **modularity** in its code. React is very modular, due to the possibility of using many npm modules. Angular was proven to be an ideal candidate for large applications, having build-in support for important functionality. Vue with many official libraries, is very modular as well. The buid tool Vite again was proven to be very useful.

The criteria of **media integration** showed the least differences because solutions exist for almost every framework. The syntax varies in-between the frameworks, and they all support media integration to be able to built an interactive application. Due to the high number of Vue modules, it was shown that the progressive approach really is defining.

Reviewing Full-Stack Frameworks

React, Angular and Vue all offer a lot of possibilities to personalise and add functionality to an application. Although Vue and Angular are **opinionated frameworks**, they opinions are chosen for a reason. They provide consistency and a great base to start a project from. [90]

It was found out that MERN is great for fast implementations, MEAN is great for bigger implementations that benefit from the correctness of data structures and MEVN is great stack for frontend developers interested in an easy frontend creation.

The key to a successful implementation of any JavaScript framework is to get to know **JavaScript** first. It eases the process of learning other frameworks and technologies. Once learned, it is easier to switch from Frontend to Backend Development with Node. Furthermore, the deep understanding of JavaScript helps finding errors ans suitable solutions. Because there are so many possibilities of frameworks, a developer can come to an individual decision of which framework to use. [3, 45, 170]

In conclusion it might be said that it doesn't matter which of the mentioned three frameworks you use. Knowing one of them offers a **good position** to start from. React provides a lot of opportunities, a great documentation, a huge community and robustness. Therefore it is used for the implementation in *Chapter 4* as part of the MERN Stack.

Chapter 4

RemArrow – Can you remember?

After the comparison of JavaScript Full-Stack Technologies (in *Chapter 3*) an application needed to be found in which all relevant parts can be put to the test. As real-time interactions and media integration play important roles in games, an implementation of a game was a good way to test these features. Games are a fun alternative to very strict prototypes. While playing a computer game, you often experience the important features that a website should have and the unwanted problems that follow first-hand [171]. This chapter contains the planning, conception and implementation of an exemplary Full-Stack MERN Implementation of the game *RemArrow*.

> *" Much of my initial fascination with computers, like that of many nerdy kids, had to do with computer games. I was drawn into the tiny simulated worlds that I could manipulate and in which stories (sort of) unfolded – more, I suppose, because of the way I projected my imagination into them than because of the possibilities they actually offered. "*
> – Haverbeke [6]

As one of his arguments Haverbeke puts nicely that a game doesn't need to be complicated in order for it to be fascinating and fun to play. With careful consideration and having a look into games of all sorts, a **game design** similar to the 1979s game *Simon* was decided upon. *Simon* – or in the German language *Senso* – was on the shortlist for 'Game of the Year' 1979 in Germany (see Figure 4.1). [172]

Figure 4.1: Game Cover of the Electronics Game Senso (German Version) [172]

Simon in its classic version is an **electronics game** that can be played by one up to four players simultaneously. It is constructed out of **four coloured keys** – yellow, blue, red and green – each representing a different music note (see Figure 4.1). When pressing down the keys or starting the game itself they each are highlighted and begin making a sound with a specific **matching note**, which doesn't change throughout the game. [172]

4.1 Requirements Analysis

As already described in the introduction (*Chapter 1*), the final goal of this thesis is to examine a JavaScript Full-Stack Technology through a prototypical implementation of the game *RemArrow*.

Application Levels

To be able to judge the progress of the application, different functionality and features were defined and categorised in three different priority levels.

Prio 1 – Full-Stack Functionality
Essential game elements in order to create a Full-Stack MERN Application and basic game play through speaking from frontend to backend using all MERN technologies.

- User Interface Design (Basic Layout of the Application using Components)
- Implementation of Arrows (Game Zone)
- Responsive Design (Mobile, Tablet, Desktop)
- Real-life Button-Effect (hover, active)
- Score Counter (onClick, setState)
- Highscore saved in and retrieved from Database

Prio 2 – Game Play
Adding Game Logic to the user interface.

- Game play through Keys (in particular Arrow Keys)
- Sound & KeyPress Effects (onClick, onKeyPress)
- Game Logic & Animation (Timed Output and Input of Key Sequence)

Prio 3 – Possible Additions
Optional additions for future improvements.

- Dark Mode Design
- Navigation Bar with additional Pages
- Highscore List (containing the names of the people setting up the Highscore)
- Multiplayer Mode (Multiple devices logging in on one site)
- Background Music

Fun Fact: Programming games is rather uncommon as a way of learning when using online resources with (13.32 %). Even online books (43.87 %) and certification videos (14.88 %) are ranked above. But for the real-time interaction the concept of a game is very interesting. [38]

4.2 Game Logic

The classic game (in its hardware form) has three different game modes with essentially the same goal: **Remembering** and **playing** the longest sound combination in the correct order. The game computer plays one note at a time, each iteration adding a note to the sound melody and getting faster. The game stops when the wrong key has been pressed or the user took too much time remembering and pressing the correct keys.

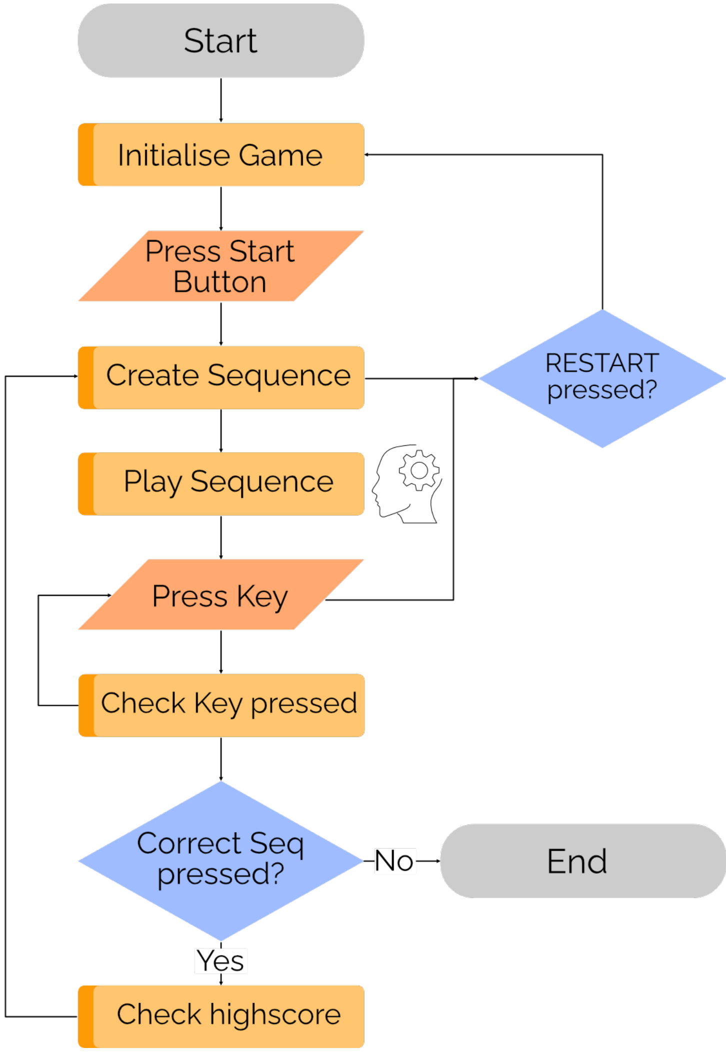


Figure 4.2: Flowchart of Game Logic (Basic structure)

Flowchart Shape	Operation / Meaning
Oval (grey)	Start / End of programme
Rectangular (orange)	Sequence
Parallelogram (coral)	Input
Diamond (blue)	Next sequence dependent on decision
Arrows	Direction of flow

Table 4.1: Flowchart Legend

The game starts by pressing the START Button and a random coloured field playing its complementary sound. After the first key finishes playing the sound and the light goes out, the user is required to take action in the form of pressing the same button, resulting in playing the same sound. When the correct button has been pressed, the game starts to add another key, adding a note to the sound melody. The game ends when the first error has been done. A restart is possible, resulting in a new game with a different combination of sounds. The flow of data and logic behind the application is portrayed in the following flowcharts. [172]

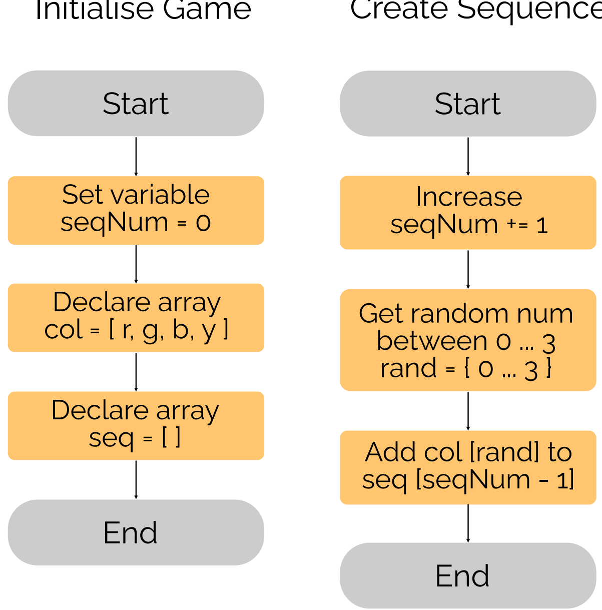


Figure 4.3: Flowchart of Game Logic (Initialising the Game and Creating a Sequence of Keys)

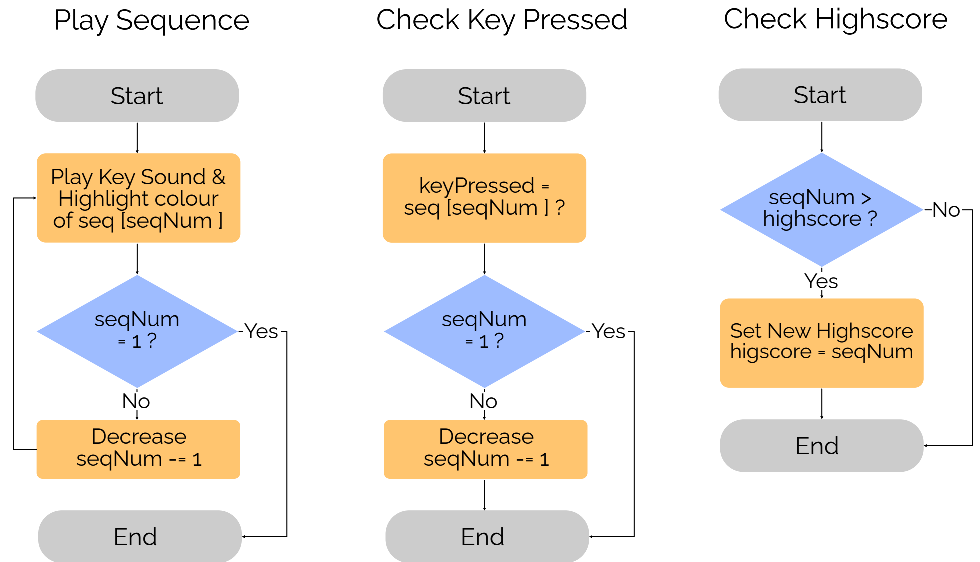


Figure 4.4: Flowchart of Game Logic (Playing the Sequence, Checking whether the correct Key was pressed and Setting up the Highscore)

It's a real-time game, supported by the application to make it reactive and interactive. The game is perfect to test **Media Integration**. Furthermore interactivity and processing in real time can be easily examined through game play.

4.3 Design

To add a uniqueness to the game a name and design scheme was invented. Differentiating it to its original version but still making it memorable and fun to play was important. During the process of finding design elements a brainstorming and scribbles of the user interface were executed.

The game consists of different **elements**, giving it its typical structure and game flow. The following different elements characterise it:

- The number 4 repeats itself several times: 4 Keys – 4 Sounds – 4 Colours.
- Different effects (hovers and highlights) to make interactivity perceivable.
- Randomness of the Key Sequence.
- Timed output and input of key sequence.
- Usage of Arrow Keys as a game controller.

Using keys as **game controls** adds another level of interactivity – the motivity – to the game. It makes the game play easier and faster than the navigation with the cursor on screen. Just like other famous computer games, this applications takes the arrow keys as functional components and additionally as a design element.

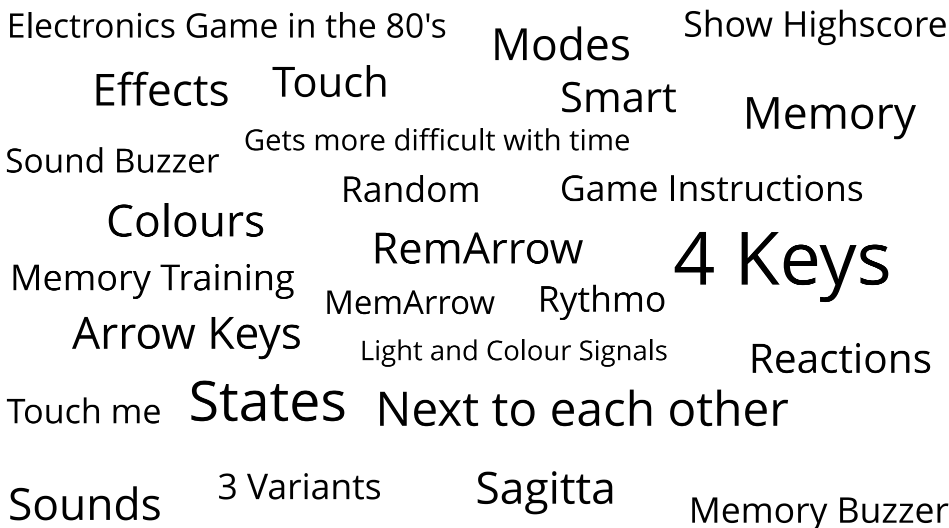


Figure 4.5: Name Scribbles & Game Components

A **name** engages the user to establish a connection and to make the game memorable. Since the elements of memory, buzzing, sound and light pay an important role, they were used to create a name for the implementation (cf. 4.5). Mentionable names are Memory Buzzer, Sound buzz or Sound Memory. As they didn't prove to be exciting enough, the game controller was addressed with the element of key usage – especially the arrow keys. Because the influence of Arrow Keys is a style element changing a simple button shape to arrows, the name got a new interpretation: RemArrow – Can you remember!

As colours play the role of differentiating the buttons and key sounds, they needed to be in good contrast to each other. Sticking to the colour circle, the basic colours red (r), green (g), blue (b) and yellow (y) were chosen as the **colour scheme**. A clean, minimalistic look was created.

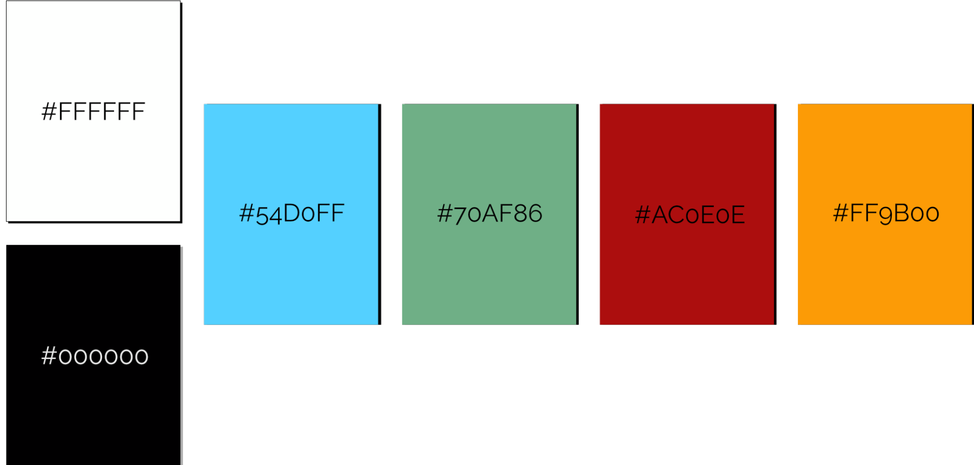


Figure 4.6: RemArrow Colour Scheme

Because the arrow keys are generic elements of the user interface, they are the main part of the **logo**. They additionally were the image-mark used for the favicon. The name *RemArrow* itself is expressive enough, so that it can be used as an word-mark.

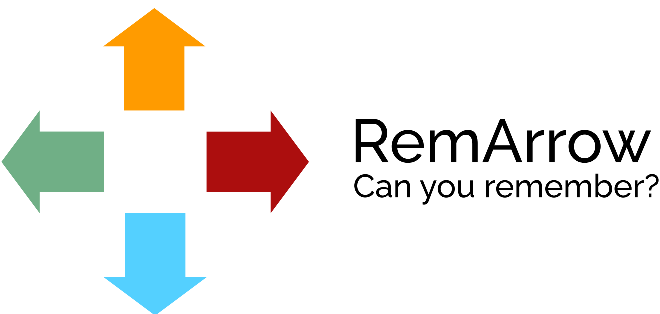


Figure 4.7: RemArrow Picture- and Word-Mark

4.4 User Interface Design

The UI should be appealing to the users, in order to be interesting and catch the user to stay on the website. Users access a website through different devices. For instance via mobile phones or computers. These require different webdesigns. As *RemArrow* strongly circles around the element of arrow key functions, the desktop version was prioritised and the UI developed **desktop-first**. It is a single-page application.

With the game zone containing the arrows, only the word-mark of *RemArrow* was integrated in the UIs top right corner of the **desktop version** (as seen in Figure 4.8). Score and Highscore are on the top left corner to create a balance between the UI elements. When the desktop gets wider (large monitors), the elements stay centered to the game zone, so that the user still has an overview. The main focus is on the game zone with the arrow keys, which can be clicked or pressed through the arrow keys.

A **hover effect** was created to give users feedback when playing with the mouse before clicking on the arrow. The design of the arrow buttons was chosen to be real-life like to help the user understand that it is also possible to press the buttons on the keyboard in a similar matter. They appear to have a shadow because of the "height" of the button. When the key is pressed, the element are shifted downwards to cover up the shadow – which gets smaller when the height of the button is decreased. Initiating effects like **hover or active states** suggest interactivity and motion to the user, introducing a potential excitement for the player of the game.

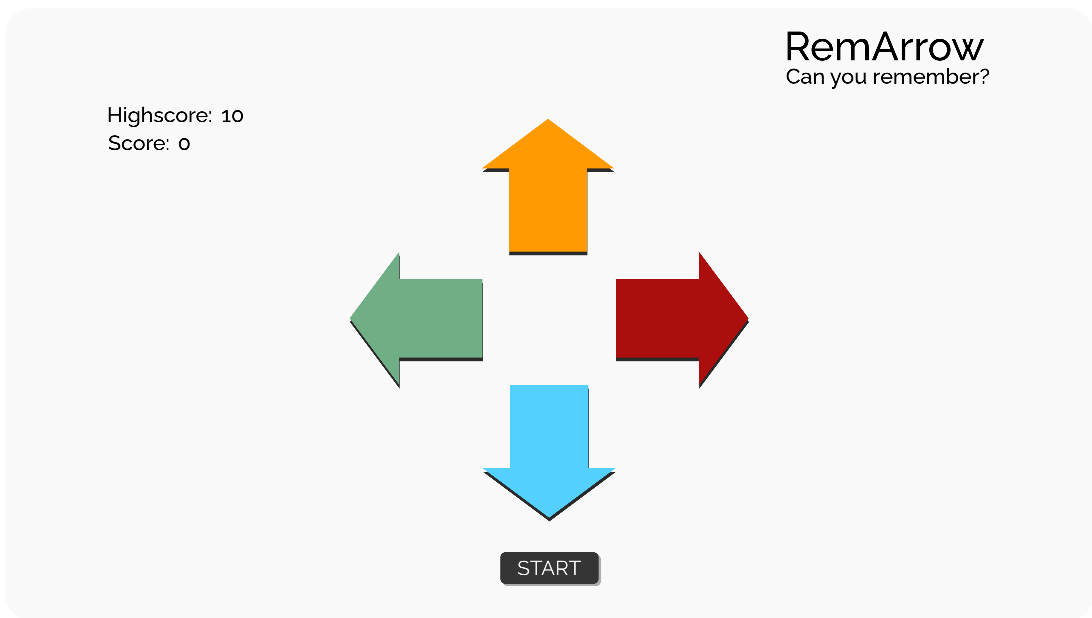


Figure 4.8: RemArrow UI Design Desktop

Continuing with the **mobile version**, the game is supposed to be played in portrait format to ensure the best partition of the arrows. The Titles and Scores are shifted to the center of the application.

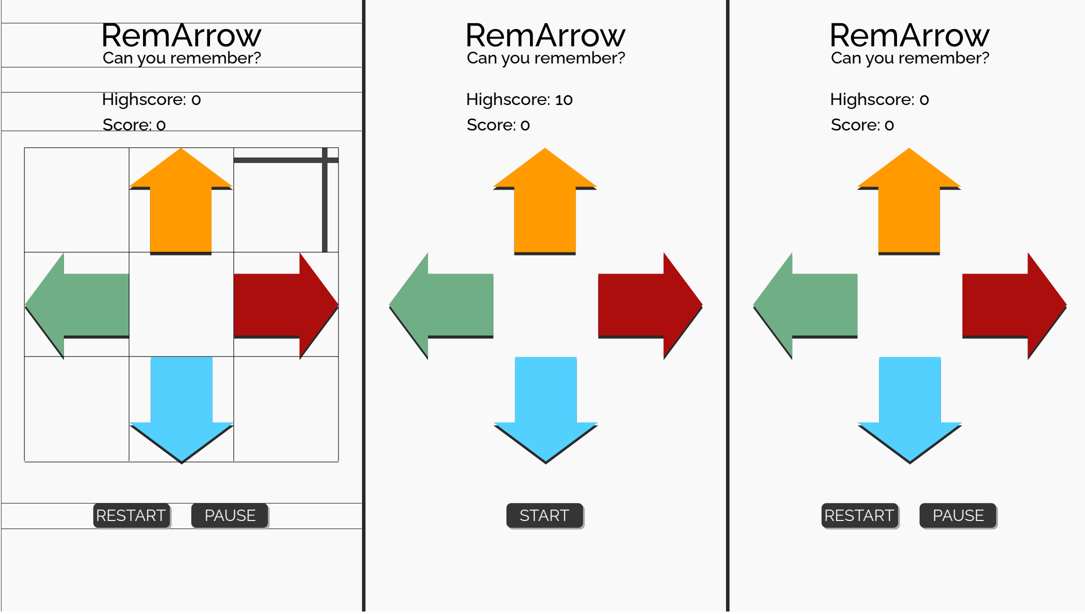


Figure 4.9: RemArrow UI Design Mobile

4.5 A Full-Stack MERN Implementation

In the following pages the development process for the implementation *RemArrow* is described. Starting with the setup of the development environment, the installation of MERN technologies, the connections between them and ending with the game presentation.

To build the application, the guidelines "How to Use MERN Stack: A Complete Guide" by MongoDB [135] and the "MERN Stack Tutorial" by NetNinja [173] were followed.

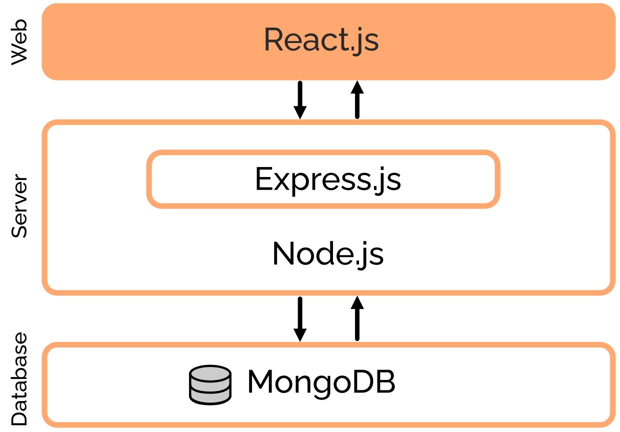


Figure 4.10: Structure of the MERN Stack, adapted by [135]

0. Setup

Firstly the development environment needed to be set up. The application was developed with the operating system Windows. As the most preferred **integrated development environment** (IDE) with 74.48 % usage and most loved editor with 81.00 % usage in the Stack Overflow Developer Survey, **Visual Studio Code** (VS Code) was chosen. It offers many great features such as linting (highlighting code features and errors) and Add-Ons for Git Integration to make working with version control more intuitive having a user interface. [38]

To avoid issues for build scripts between Linux and Windows, it was useful to use a Linux command line. Node.js is easier to combine with a Linux distribution, as well as important tools like nvm are only available for Linux. The **Windows Subsystem for Linux** (WSL2) offers an easy way to use Linux under Windows, enabling the user to install a preferred Linux distribution. This way no build scripts need to be executed with Windows, creating consistency in the development and production environment. The default Linux distribution – also used for this project – was **Ubuntu**. [38, 174]

The project folder "remarrow-game" was created and accessed through VS Code, having a connection to the WSL and an empty folder to start with.

1. Installing Node

The first step was to install **Node**. This was possible from either the official website "http://nodejs.org", or using a package manager. The Long-Term Support Version from the official website was chosen, which is recommended for production as it is more stable. Node provides a Windows Installer, with which it was possible to smoothly install. Included in the installation was the **npm package manager**. In the official installation guide, Microsoft also recommends using a version manager such as nvm, as versions change very quickly. Before continuing with the development, existing versions were upgraded (as seen in 2.76). [71, 174]

To simply **test** if Node works, the console was opened and the command "node" entered. With this the Node REPL (Read-Evaluate-Print-Loop) (cf. 2.2.4.1) was opened, and the command with the console.log( )-function could be typed and entered: console.log("Hello from Node!"). When executing the command in the (CLI), the output "Hello from Node!" was given and the functionality of Node was confirmed. [173]

2. Using GitLab

Connection to GitLab Repository

As seen in section 2.4.2 Version Control is a very useful feature for development. Therefore a remote GitLab **repository** was added to the project prior to anything being installed. VS Code has integrated source control management (SCM) support for Git, offering a source control tab that shows all the code changes. [175, 176]

To create a project, the website "https://gitlab.com/projects/new" was accessed. Being logged in with a user account, a **blank project** was chosen to start with. The project was named "RemArrow" with the project slug "https://gitlab.com/user-name/remarrow-game/". It then was initialised with a README file and the repository was created (see 4.11). To use the remote repository locally on the developing machine, VS code was opened. Through the WSL the repository could easily be cloned and connected to, after logging in with the correct credentials. Under the Source Control icon in the Activity Bar, an overview of changes could now be seen. For safer development, a branch of the project was created and changed to. This way the main branch contains stable versions and development is done on another branch. [175]

The ".gitignore" file automatically gets created during the installation process. This files lists all node modules. As they don't change between different commits (except for when updating a module), they will be ignored in the upload to the remote repository. Hence improving performance, because the size of the "node_modules" directory is typically rather large. [71, 176]

DevOps with GitLab

To explore the DevOps feature of **Pipeline** in GitLab, a **GitLab Page** is created. GitLab Page are a new feature of GitLab where GitLab hosts an instance of the project directly on their server, showcasing the application. With this feature, the concept of hosting a website is explored as well. The pipeline can be seen as a montage cycle. Before the application is deployed to the user, every function needs to be tested first. When the code is pushed into the repository (uploaded), the pipeline automatically is triggered. For instance this means the project is built, tests are executed, and then the code is made more legible and deployed.

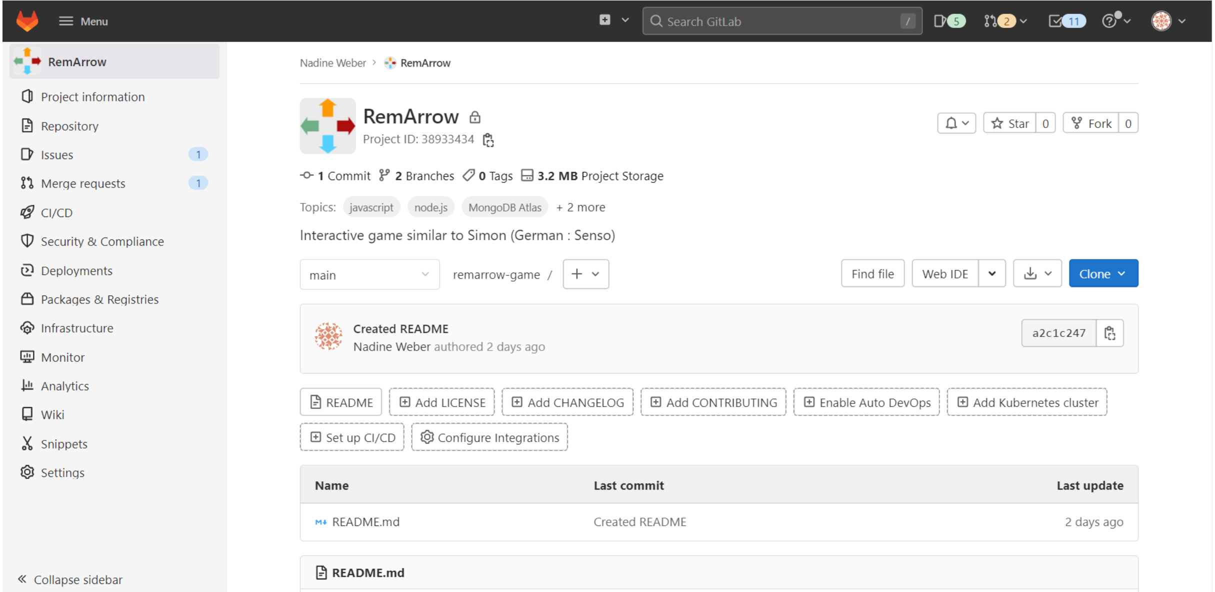


Figure 4.11: GitLab Repository of RemArrow

In case of the GitLab Pages for *RemArrow* shown in *Figure* 4.12, the pipeline consists of two steps: build and deploy. If there is any problem in the building process, the build fails and the developer is notified. The pipeline is repeated every time new code is uploaded, ensuring that the application is working. The automation removes the process of deployment and improves performance. [32, 138, 154]

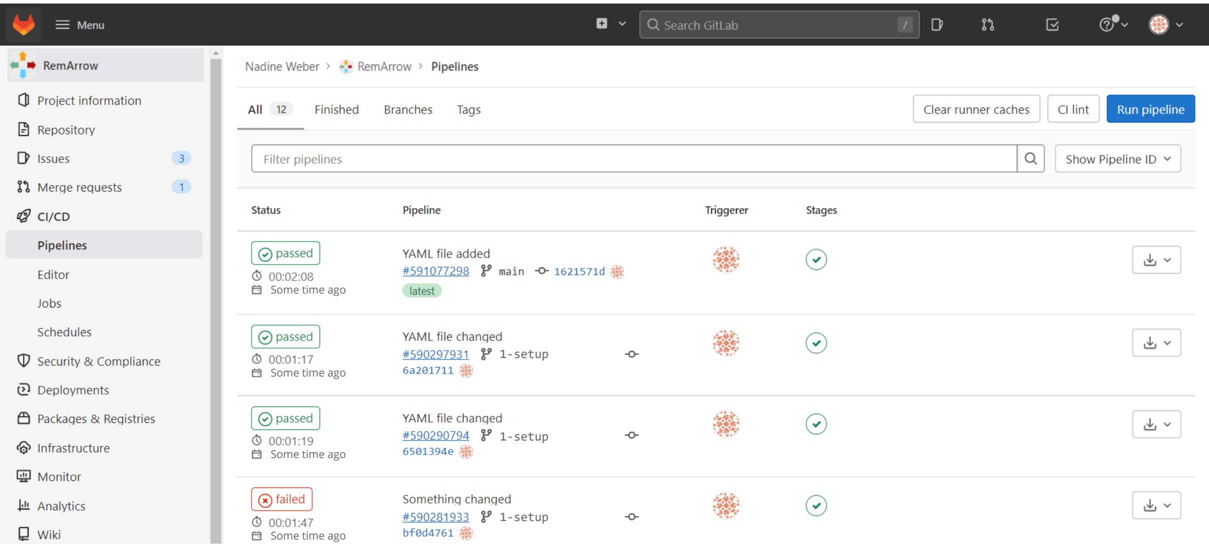


Figure 4.12: GitLab Page Pipeline

3. Installing Express

Building a Server

A folder for the frontend (frontend) and backend (server) was created. To keep up with the code and technology changes, the "package.json" file was created in the server folder. Every dependency and installed module is listed in the "package.json" file. For the creation, "npm init -y" was executed. npm automatically takes care of fetching those dependencies and writes them down in the "package-lock.json" file (cf. 2.2.4.1). Most versions of frameworks are indicated with an asterisk, for npm to grab the newest version available. The development environment was now prepared for the installation of technologies. [71]

Inside the server folder, Express was installed by using the npm command "npm install -g express". The 'g' means that the modules are installed in the global cache, this is where every other project also has access to by specifically adding the project to the global module. Without the 'g', Express would be saved in the local or project cache, which is the current directory. As different projects might not use Express in the same version, it's best to add Express locally as well. [71]

The "server.js" file (see 4.13) contains the code to register Express and MongoDB. To use the Express package, it needs to be required (line 1) and an instance created (line 2). To listen for requests, a Port number is indicated from where the server listens to. Through executing the command "node server.js" the server was started. The server then listened to the Port 4000. [43, 173]

```
1  const express = require("express")      // Requiring the usage of Express
2  const app = express()                   // Creating an instance of Express
3
4  app.get('/', (req, res) => {     // GET-Request to the root (/)
5    res.send('Hello World!')       // Responding with "Hello World!"
6  })
7
8  app.listen(4000);          // Server listens to Port 4000 for connections
```

Figure 4.13: Creating an Express Server (server.js)

Further Configurations

For easier development, the Node package **nodemon** was installed by running "npm install nodemon". It **automatically updates** the Node application by taking the code changes in the server file. Therefore no refreshing of the page is necessary. To ease the command, a new "dev" script was written in the "package.json" file (see 4.14). The server with nodemon could then be started via "npm run dev". [173, 177]

```
1  ...
2      "scripts": {
3          "start": "node server.js",
4          "dev": "nodemon sever.js"
5      },
6  ...
```

Figure 4.14: Defining a Custom Command for Development (package.json)

Hard-coded Port numbers, variables and user credentials are critical when uploading them in a public repository. Especially sensitive information consequently should be hidden by using **environmental variables**. This can easily be done through another Node module called **dotenv**. This package loads environment variables from a ".env" file into process.env file. This lets the developer separate configuration files from the code. The ".env" file is integrated into the ".gitignore" file, leaving all environmental information outside of the remote repository. In this instance the Port number and hard-coded information in "server.js" file then needs to be exchanged with the environmental variables. They are indicated in upper-case letters. The full access is done through the "process" object, which is globally available (see 4.15, l.6). By requiring the "dotenv" package the variables could easily be integrated. [43, 173, 178]

```
1  require('dotenv').config()  // Required dotenv package
2  const express = require('express')
3  const app = express()
4
5  // Listening for Requests
6  app.listen(process.env.PORT, () => {
7      console.log('listening on port', process.env.PORT)
8  })
```

Figure 4.15: Listening for Requests with an Environmental Port Number (server.js)

To specify some terminology: Middleware is everything that lies between executing requests and responses, from frontend to backend. The handlers in the REST-APIs are **middleware functions**. A global middleware can be implemented to get notifications on every requests that comes in. In addition to the "req" and "res" object, the function "next( )" needs to be provided and invoked, so that the next piece of middleware – aka the next handler – can be executed. [43, 173]

Creating APIs

Exemplary APIs for a simplified version of *RemArrow* are shown for the implemented features "score" and "highscore". These are the different APIs, which needed to be set up:

HTTP Request	Used for...
GET /api/scores/highscore	Getting the current highscore
POST /api/scores	Creating a new score

Table 4.2: Overview of Score APIs in RemArrow

The score needs to be created as a new database entry using POST. The highscore then is accessed by filtering through the database entries looking for its highest value (GET). More about fetching data will be explained in Section 4.5 (Connecting Frontend to Backend).

To keep the "server.js" file clean, a new folder named "routes" was created. Additionally, a JS file with the name of the APIs which are to be specified (here: "scores.js") was created. To add access to the app instance from the "scores.js" file, the Express **Router** was used (see 4.16, line 2). A handler like ".get" or ".post" can then be added to the router. By exporting the router and requiring it in the "server.js" file, the routes are accessible to the Express server. The routes with correctly specified method then invokes a function by the **Score Controller**, which contains all functions controlling the **Score Model** through requests. The Score Model is a schema that defines the structure of a document inside the database (here: score is of type number). It controls the values, so that they are correct (see model in B.3). [173]

```
const express = require('express')  // Using Express
const router = express.Router()     // Instance of the Express Router

// GET the highscore
router.get('/highscore', getHighscore)

// POST new score
router.post('/', createScore)

module.exports = router             // Exporting the Router
```

Figure 4.16: Score APIs (scores.js)

In the server file the router needs to be included and loaded (see 4.17, line 3). While loading, the route is set to where the "scores.js" file sits in the folder. By adding the route "/api/scores" to the route handler (line 8), the paths become relative. This sets the API before the slash of the router as seen in the instance "scoreRoutes". In the score.js file, this means that all routes already have "/scores" in the front of the API. Instead of ".get" the handle ".use" gets used. [173]

```
require('dotenv').config()
const express = require("express")
const scoresRoutes = require('./routes/scores') // Importing Score Routes
const app = express()

// Middlware
app.use(express.json())              // Attaches data to req body

// Importing Routes
app.use('/api/scores', scoresRoutes) // Attaching Score Routes to the app
    // if a request to the /scores route is send, the scoresRoutes
        instance gets used
```

Figure 4.17: Adding the Routes to the Server file (server.js)

Data is stored inside the request (req) object. To be able to access that data in the routes, the middleware "express.json" is required. By adding the statement "app.use(express.json( )" the app searches for data and adds it to the request body. This way the data can be accessed in the score routes. [173]

To keep the overview of all routes used in the project, the software **Postman** was used. It is a platform that keeps track of APIs. Postman simulates the routes that were created for the API and shows the responses in JSON format. This is why it is a great choice for development. The requests could be saved in the database of Postman, to be able to try them out later on. In this instance the collection is called "RemArrow". [173, 179]

The routes can be **controlled** through the Postman interface. This helps to test early on, whether the routes give the correct responses. Having them integrated in Postman additionally helps when the actual functionality is used. An exemplary overview is given in 4.18.

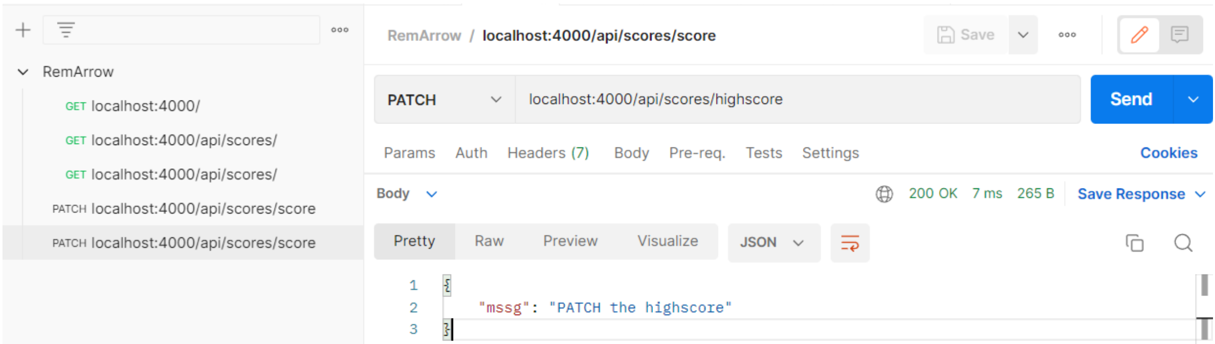


Figure 4.18: Testing API Responses in Postman

4. Using MongoDB Atlas

MongoDB Altas Setup

To retrieve data from a database, the free cloud-service MongoDB Atlas was used. As described in 2.2.4.3, the setup consists of three main steps which will now be examined. The fourth step of customising alerts was not used. Before setting up a cluster, the user needed to be signed in with a MongoDB account. [128]

Step 1: Deploying the database

First the database was created, by selecting the cluster type "shared" (see 4.19). The free model lets the user choose between different cluster tiers. The MO Sandbox is limited to 512 MB storage, but it can be accessed unlimited and is ideal for experimenting. A production cluster can be upgraded to at anytime, providing more storage. After selecting the cloud provider (AWS) and region (Germany, Frankfurt), the cluster was named "RemArrow" and thus created. [128]

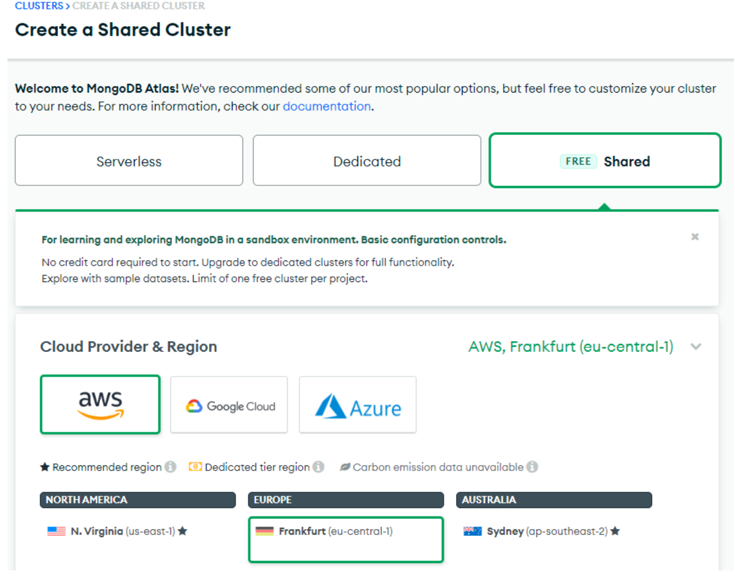


Figure 4.19: Creation of a free shared cluster in MongoDB Atlas

Step 2: Managing Users

To access data stored in Atlas, a user needs to be created and the network security controls need to be set up. The page where this is configured was called "Security Quickstart" (see 4.20). To authenticate a user, either a username and password or a certificate can be provided. Continuing with the configuration, a specific user needed to be created that has permission to read and write any data in the project. If a user existed before, he or she could be simply assigned to the new project. By entering the username and auto-generating a secure password the base for the connection is laid. [128]

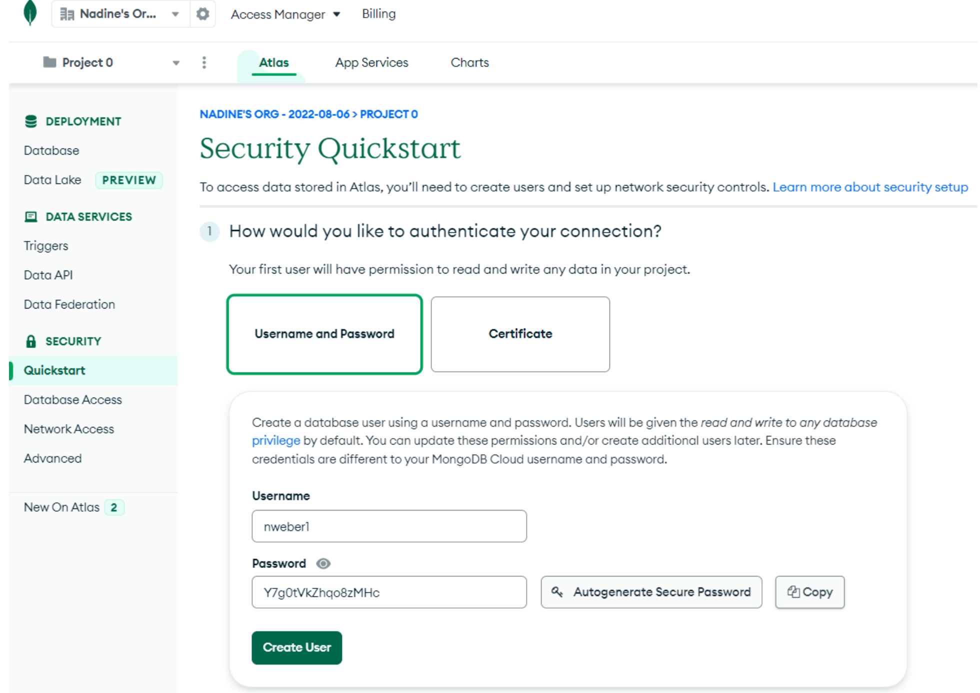


Figure 4.20: Creation of a User and Setting a Password for the Connection String

In the tab **network access** (see 4.21), an address needs to be added to the white list. The user can choose between adding a current address (needs updates) or allow access from anywhere (security risk). This ensures a safe connection. [128]

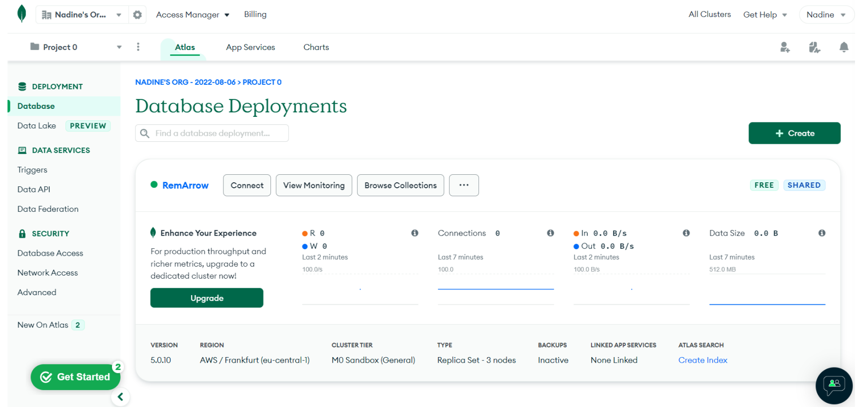


Figure 4.21: Network Access for all users

Step 3: Connection to the Cluster

To connect the application to the database cluster, the method “Connect your application” was chosen. It connects the cluster using MongoDB’s native driver for Node.js called “mongodb”. The driver was installed using ”npm install mongodb” [128]. Through the web interface, a connection string code was available to copy and paste into the application. As the string contained sensitive information with the password, it also was included into the “.env” file under the variable “MONGO_URI”. [128, 173]

Before the actual connection the Node package **mongoose** needed to be installed. Mongoose is the standard object model used for the data access layer. In other words it is the tool that helps to interact with MongoDB and also helps to connect to the database. It lets the developer start requests from the command line. Installation was done by ”npm install mongoose”. To be able to use mongoose, it needed to be required just like Express (see 4.22, line 3). [43, 134, 173]

```
1   require('dotenv').config()
2   const express = require('express')
3   const mongoose = require('mongoose')
4   const app = express()
5
6   // Connection to MongoDB Atlas
7   mongoose.connect(process.env.MONGO_URI)
8       .then(()=> {
9           // Listen for Requests
10          app.listen(process.env.PORT, () => {
11              console.log('Connected to MongoDB & listening on Port',
                    process.env.PORT)
12          })
13      })
14      .catch((error) => {
15          console.log(error)
16      })
```

Figure 4.22: Connecting MongoDB Atlas to Express using mongoose (server.js)

With the mongoose object, the .connect method was accessible. Inside the method, the “MONGO_URI” needed to be added. This activates the connection to the database. Databases need to be ready before any URL is loaded, that’s why an async function is needed. Through “.then” a function is fired when the database is loaded. Additionally, a catch function was added (line 15). If an error occurs, it is catched and logged to the console. As it is important that listening for requests happens after the database is loaded, the code for listening was added inside the “.then” function.

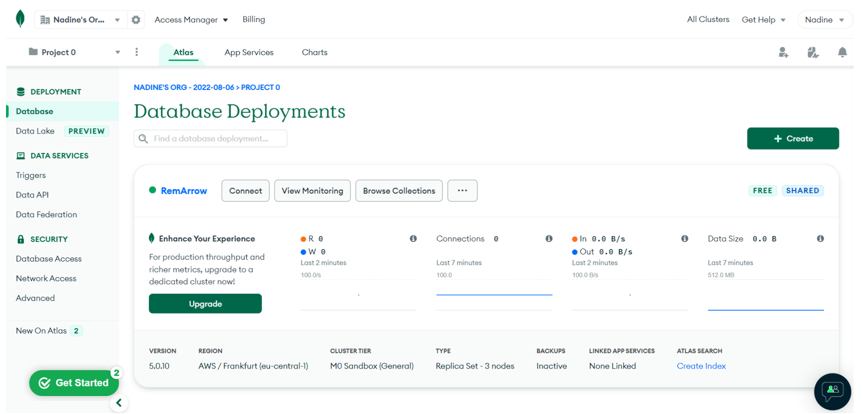


Figure 4.23: MongoDB Atlas Database RemArrow

With these configurations and file changes, the collection RemArrow was created, the connection implemented and the MongoDB Atlas database ready to be used.

5. Building the frontend with React

A new **React App** was created using the command "npx create-react-app frontend". The installation created the new React App in the folder "frontend" outside of the server folder, by installing different packages (cf. 2.2.3.1).

Next, the "App.js" file was opened and all preset code was removed. To start the local server and to run the app, command "npm run start" was used. Prior to running the app for the first time, "npm install" was entered to install all packages and dependencies. The application is available by default on localhost Port 3000 with the URL "http://localhost:3000" (cf. 3.4.1).

User Interface Design - React Components

As described in 4.4 the application is on a single page. The user interface consists of different areas. To use the **component-based approach**, the different areas needed to be determined and build through scripting. The designed user interface features four main regions which were implemented in a "containers" folder in the public directory of the frontend: Titles, Scores, Arrows and Button-Group. They are the building blocks of the UI and are the basic layout of the page. To keep further interface options, such as a navigation bar and different pages, a possibility, the "Game Zone" was built on a separate page named "<Remarrow />" as Class Component (4.24).

```
1  export default class RemArrow extends Component {
2      render() {
3          return (
4              <div id="GameZone">
5                  <Titles />
6                  <Scores />
7                  <Arrows />
8                  <ButtonGroup />
9              </div>
10         )
11     }
12 }
```

Figure 4.24: RemArrow Class Component (remarrow.js)

To add the Remarrow Component to the root page, it needed to be imported and added.

```
1  import Remarrow from './pages/remarrow';
2
3  export default function App() {
4    return (
5      <div className="App">
6          <Remarrow />
7      </div>
8    );
9  }
```

Figure 4.25: RemArrow Root File (App.js)

The components were build up being small and reusable, following the recommendation by React to make the App as modular as possible. For instance the "<Scores />" Component contained two "<Score />" Component with different attributes. Similar style applies to "<Titles />" and "<ButtonGroup />" The names and classNames were prepared to be as responsive and reusable as possible.

Implementation of Arrows

The arrow structure in the game zone first was designed with a graphical editor as a Scalable Vector Graphic (SVG). The arrows were arranged in a 3x3 grid, equal in width an height to be placed nicely on the UI. To create a consistent scaling of the arrows, the interface was implemented by adding them as images (in particular".svg" files) to an arrow component. They were delivered as props for the "imgURL" to the simple '<Arrow />" component. For that reason the vector graphics were exported individually and added to the "assets" folder. To be able to use them as props, they were imported and used as JSX variables.

```
1  // Import of Images
2  import arrowBlue from '../assets/img/arrow_blue.svg';
3  import arrowYellow from '../assets/img/arrow_yellow.svg';
4  import arrowGreen from '../assets/img/arrow_green.svg';
5  import arrowRed from '../assets/img/arrow_red.svg';
6
7  export default class Arrows extends Component {
8      render() {
9          return (
10             <div className="ArrowContainer">
11                 <Arrow imgURL={arrowYellow} id="arrowYellow" alt="Yellow"/>
12                 <Arrow imgURL={arrowRed}    id="arrowRed" alt="Red"/>
13                 <Arrow imgURL={arrowBlue}   id="arrowBlue" alt="Blue"/>
14                 <Arrow imgURL={arrowGreen}  id="arrowGreen" alt="Green"/>
15             </div>
16         )
17     }
18 }
```

Figure 4.26: Creating Arrow Components by passing through .svg files as imgURLs (arrows.js)

The Arrow Component then only needed to receive the props by calling "this.props.imgURL". This is also were the onClick handler gets accessed - by clicking on an image of the arrow. To place the arrows in the correct column and row, the "grid-area" CSS attribute were assigned.

```
1  import { increaseScore } from '../container/scores';
2
3  export default class Arrow extends Component {
4      render() {
5          return (
6              <img
7                  className="arrow"
8                  src={this.props.imgURL}
9                  id={this.props.id}
10                 alt={this.props.alt}
11                 onClick={() => increaseScore(1)}
12             />
13         )
14     }
15 }
```

Figure 4.27: Creating Arrow Components by passing through .svg files (arrows.js)

Due to asynchronous processing, React might update multiple state changes at one time. To prevent undesired results, an ES **Arrow Function** (line 13) is used.

Responsive Design

Through the assigned grid-areas, the grid-template was determining the size of the arrows. To suit the design to the size of the screen, **media queries** were implemented and the sizes of the '.ArrowContainer" adjusted. All style changes can be seen in B.6.

```
/* Small devices (phones and portrait tablets, up to 768px) */
@media screen and (max-width: 768px) {
  .ArrowContainer{
    grid-template-columns: 90px 90px 90px;
    grid-template-rows: 90px 90px 90px;
  }  }
}
/* Medium devices (tablets in landscape, 768px height and up) */
@media only screen and (min-height: 768px) and (min-width: 480px){
  .ArrowContainer{
    grid-template-columns: 200px 200px 200px;
    grid-template-rows: 200px 200px 200px;
  }
}
/* Extra large devices (large laptops and desktops, 2000px width and up) */
@media only screen and (min-width: 2000px) {
  .ArrowContainer{
    grid-template-columns: 250px 250px 250px;
    grid-template-rows: 250px 250px 250px;
  }
}
```

Figure 4.28: Media Queries for Arrow Container Areas (App.css)

Real-life Button-Effect

The arrow keys in the game design are meant to either be clicked on through a click motion by the cursor or by pressing the keys on the keyboard. To **showcase interactivity** and prepare for the game play through keys, the arrow as well as simple button components were designed to look like real-life buttons. In this instance, the design was chosen to have a shadow falling onto them.

When **hovering** over with the cursor, the buttons are signalling that they can be pressed through a simple effect. When actively pressing the key, the shadow gets smaller and the arrow gets transformed downwards on the y-scale. A shadow for SVGs can be implemented by a "drop-shadow". This can be seen in 4.29. The style can be applied when triggered by the "onKeyPress( )" function as well. Additionally, the style was applied to the Button Component to create consistent design. The "drop-shadow" of the svg files thereby only needed to be changed to a "box-shadow".

```
.arrow {
  transition: filter .1s ease-out;        /* Animation of drop-shadow */
  cursor: pointer;                        /* Changing the curser to pointer
      */
  filter: drop-shadow(0px 6px 0px #444);    /* SVG Shadow */
}
.arrow:hover {
  transition: 0.2s ease-out;
  filter: drop-shadow(3px 6px 3px #000);    /* Bigger Shadow to indicate
      interactiveness */
}
.arrow:active {
  filter: drop-shadow(0px 3px 0px #222);    /* Smaller Shadow because of
      "less light" */
  transform: translateY(4px);             /* Transforming arrow downwards on
      y-scale */
}
```

Figure 4.29: Arrow Styles (Hover and Active-Effect for Arrow Components) (App.css)

Score Counter

Props are objects which hold information, but as already explored in 2.2.3.1 they cannot be changed. States are generally updated by **event handlers**. The state of the score changes every few seconds through clicks on an arrow component. That being the case a state needed to be introduced into the class component "scores".

```
export default class Scores extends Component {
    constructor(props){
      super(props);
      this.state = {
          counter: 0,
          highscore: 0,
      };
    }
    render() {
        return (
            <div className="Scores">
                <Score scoreName="Highscore"
                    scoreNum={this.state.highscore}/>
                <Score scoreName="Score" scoreNum={this.state.counter} />
            </div>
        )
    }
}
```

Figure 4.30: Score and Highscore State (scores.js)

6. Connecting Frontend to Backend

After building the backend with database and the user interface in the frontend, the connection for retrieving and sending data needed to be configured. Exemplary, the first functionality of the Score and Highschore was added.

The simplified logic was this: When the user interfaces loads, it automatically loads the highscore from the database using the 'GET Highscore" API. The buttons from the button group originally being 'RESTART" and 'PAUSE" were exchanged to 'RESTART" (resetting the score) and 'SET SCORE" (sending a new database entry of the score with 'SET SCORE" API. The Highscore got updated when first loading or clicking on of the provided buttons. The score was updated by clicking on an '<Arrow />" Component.

To get the data from the database, React contains the 'fetch" method. It take an URL and fetches the data that it can retrieve. By getting the API response, the score can be updated with the fetched data. As soon as the database is loaded, the API gets called and the state reset. The State changes throughout the lifecycle of the component. That's why the state first needs to be defined in the constructor of the component's class, being the default values 0 for both scores. The states needs to be changed with the method "setState( )". Otherwise no changes get detected by React and the UI doesn't get repainted.

```
1  export function getHighscore () {
2    fetch("http://localhost:4000/api/scores/highscore")
3        .then((res) => res.json())
4        .then((data) => {
5          console.log(data)
6          this.setState({
7            highscore: data.score,
8          })
9        } )
10 }
11 export function resetScore () {
12   console.log("reset score")
13   this.setState({counter: 0})
14   getHighscore()
15 }
16 export async function setScore () {
17   console.log("set score", this.state.counter)
18   await fetch('http://localhost:4000/api/scores', {
19     method: 'post',
20     headers: {'Content-Type':'application/json'},
21     body: JSON.stringify({
22      "score": this.state.counter
23     })
24    })
25    getHighscore()
26 }
27 export function increaseScore (value) {
28   // console.log(value)
29   this.setState({counter: this.state.counter + value})
30 }
```

Figure 4.31: State / API Update Functions (scores.js)

To update the state Score out of the component Arrow, both components being siblings, needed to have access to the function. The interested class (in this case Score) needed to bind the method in the constructor. The score functionality was implemented.

```
export default class Scores extends Component {
    constructor(props){
      super(props);
      this.state = {
          counter: 0,
          highscore: 0,
      };
    }
    componentDidMount(){
      console.log("Mounted")
      getHighscore = getHighscore.bind(this)
      increaseScore = increaseScore.bind(this)
      resetScore = resetScore.bind(this)
      setScore = setScore.bind(this)
      getHighscore()
    }
    render() {
        return (
            <div className="Scores">
                <Score scoreName="Highscore"
                    scoreNum={this.state.highscore}/>
                <Score scoreName="Score" scoreNum={this.state.counter} />
            </div>
        )
    }
}
```

Figure 4.32: ComponentDidMount (scores.js)

To prevent problems with access in the browser, cors needed to be kept in mind. CORS stands for Cross-Origin Resource Sharing. It is the way of the server accepting requests, when they are coming from a different origin [180]. To be able to cooperate with the server, the browser needs an agreement with Express. For this reason the package "cors" was installed and integrated before any routes were in use. The final version of the code can be found in Appendix B. The reset and setting of the score were implemented in a similar style such as "increaseScore()". They can also be found in the Appendix. With all tech stack components connected and additional packages installed, the first version of the Prototype of *RemArrow* was running.

7. Recap

The implementation of the game *RemArrow* contained different challenges. Not only the technologies used in the MERN Stack needed to be known, it was important to know the connections between them. Additionally the project setup needed to be made thoroughly by adding additional functionality through Node packages (nodemon and mongoose) or best practices for development like environmental variables and the usage of Postman to test the APIs.

Being fluent in both frontend and backend technologies is a major challenge. The MERN stack showed its strength due to the usage of JavaScript and the connections between the technologies. Building stack the correct way in order for the connections to work correctly was difficult and needed knowledge of every part in the building process. As the Prio 1 Full-Stack Functionality of RemArrow now are is built up, the Prio 2 and Prio 3 functionalities can be implemented. The application is ready for further developments.

Chapter 5

Conclusion & Outlook

Throughout this thesis different technologies were presented and their backgrounds were explored. In this final chapter, the research will be concluded and evaluated. Additionally, an outlook into future developments will be given.

5.1 Conclusion

The state of the art of JavaScript technologies was investigated in Chapter 2 and showed a great range. To gain a basic understanding, in Chapter 2.1 insights into the fundamental history of JavaScript, important JavaScript flavours like TypeScript and the overall popularity of the programming language were given. A simple button example was presented, which was continued to be examined throughout the presentation of the frontend frameworks ReactJS, Angular and VueJS.

Concepts of the used Frontend- and Backend-Technologies were explored in Chapter 2.2. The frameworks which were compared were chosen by their popularity, based on different statistics. Through the invention of Node.js, the possibility for JavaScript Full-Stacks was opened. The Full-Stacks were presented and described how to be built in 2.3. Node.js for this reason was marked as the most influential JavaScript Technology.

In Chapter 2.4, Full-Stack Development was recognised as a diverse way of working in both frontend and backend. Aspects such as the role of a Full-Stack Developer (2.4.1) and DevOps (2.4.2) in the modern technology stack were analysed, showcasing an overview of what lies beyond the trend. Furthermore, the choosing of a technology stack seems to depend upon on many different factors such as purpose and size of the application. A concluding discussion on the advantages and disadvantages of this trend (2.4.3) was made.

Chapter 3 was all about the evaluation and comparison of Full-Stacks. Recommendations on which concepts of Full-Stack Development are important for a university-level course were evaluated in 3.2. The base for these recommendations primarily was taken from the questionnaire conducted with university students. The following comparison of the most common JavaScript-based Full-Stacks (MERN, MEAN and MEVN) in Chapter 3.4 was determined by different criteria. It was found out that MERN is great for fast implementation, MEAN is great for bigger implementations that benefit from the correctness of data structures. In addition to that MEVN is a great stack for frontend developers interested in an easy frontend creation. No particular Stack could be named as the best to use, as it was revealed they strongly depend on the type of application.

For testing purposes, the Full-Stack MERN application *RemArrow* was implemented in Chapter 4. Through the basic Full-Stack functionality, the connections between the tech stack components were examined and impressions of the development process were given. In retrospective, having followed only a frontend development path before, the connection between the different frameworks was difficult to understand. Full-Stack Development has proven to be an important field with lots of possibilities of specialisation.

5.2 Evaluation and Outlook

The results of this thesis point to the fact that Full-Stack JavaScript Development is far more than just an interesting trend. It influences the view on Web Development and the teaching of Web Technologies. Having many possibilities to choose from, Full-Stack Development was proven to be very diverse and full of challenges. Simple to complex applications can be built, deciding on the Full-Stack by determining the important factors for the project. Thus trying out different technologies in order to find the best solution is a good approach.

The prototype of the game *RemArrow* in Chapter 4 can be further modified corresponding to the ideas in the analysis of the requirements. This features the animation of the key sequences, a highscore list and a multiplayer mode.

Arising technologies such as the JavaScript runtime Deno and the build tool Vite combine approaches for frameworks and tooling, which might be interesting for future developments. The JavaScript Alternative PyScript also was proven to be of interest for machine learning developers.

The results of the questionnaire additionally enable the conception and integration of new curricula for university-level learning courses. For future comparisons of Full-Stack JavaScript technologies, the backend technologies could be investigated further.

Bibliography

[1] Google Forms. *Google Formulare: Kostenlos Umfragen erstellen und analysieren*. 2022. url: https://www.google.de/intl/de/forms/about/.

[2] PYPL. *PYPL PopularitY of Programming Language index*. 1.04.2022. url: https://pypl.github.io/PYPL.html.

[3] AltexSoft. *MEAN and MERN Stacks: Full Stack JavaScript Development Explained*. 2021. url: https://www.altexsoft.com/blog/engineering/mean-mern-javascript-full-stack/.

[4] Wikimedia Commons. *File:Unofficial JavaScript logo.svg*. 2022. url: https://commons.wikimedia.org/wiki/File:Unofficial_JavaScript_logo_2.svg.

[5] w3schools.com. *JavaScript Tutorial*. 31.03.2022. url: https://www.w3schools.com/js/default.asp.

[6] Marijn Haverbeke. *Eloquent JavaScript: A Modern Introduction to Programming: 3rd Edition*. 2018. url: https://eloquentjavascript.net/.

[7] Ecma International. *ECMAScript® 2023 Language Specification*. 26.05.2022. url: https://tc39.es/ecma262/.

[8] w3schools.com. *JavaScript Versions*. 2022. url: https://www.w3schools.com/js/js_versions.asp.

[9] Sean Maxwell. *10 things to learn for becoming a solid full-stack JavaScript developer*. 19.11.2018. url: https://levelup.gitconnected.com/10-things-to-learn-for-becoming-a-solid-full-stack-javascript-developer-8b76467711ac.

[10] Ecma International. *Home | Ecma International*. 20.08.2020. url: https://www.ecma-international.org/.

[11] Claire Brotherton. *PHP vs. JavaScript: Ein detaillierter Vergleich der beiden Skriptsprachen*. 13.01.2021. url: https://kinsta.com/de/blog/php-vs-javascript/#serverseitiges-vs-clientseitiges-skripting.

[12] liveScript.net. *LiveScript - a language which compiles to JavaScript*. 2022. url: http://livescript.net/#overview.

[13] coffeescript.org. *CoffeeScript*. 2022. url: https://coffeescript.org/#introduction.

[14] Wikipedia, ed. *LiveScript*. 2022. url: https://en.wikipedia.org/w/index.php?title=LiveScript&oldid=1078258091.

[15] typescriptlang.org. *TypeScript: Branding*. 1.06.2022. url: https://www.typescriptlang.org/branding/.

[16] typescriptlang.org. *TypeScript: JavaScript With Syntax For Types*. 2022. url: https://www.typescriptlang.org/.

[17] stateofjs.com. *The State of JS 2021*. Ed. by Sacha Greif. 2022. url: https://2021.stateofjs.com/en-US/.

[18] typescriptlang.org. *Documentation - TypeScript for the New Programmer*. 2022. url: https://www.typescriptlang.org/docs/handbook/typescript-from-scratch.html.

[19] elm-lang.org. *Elm - delightful language for reliable web applications*. 2022. url: https://elm-lang.org/.

[20] purescript.org. *PureScript*. 2022. url: https://www.purescript.org/.

[21] reasonml.github.io. *Reason · Reason lets you write simple, fast and quality type safe code while leveraging both the JavaScript & OCaml ecosystems*. 2022. url: https://reasonml.github.io/.

[22] clojurescript.org. *ClojureScript*. 2022. url: https://clojurescript.org/.

[23] Wikipedia. *Datei:Ecma International Logo.svg – Wikipedia*. 2022. url: https://de.m.wikipedia.org/wiki/Datei:Ecma_International_Logo.svg.

[24] Brendan Eich. *ECMAScript Harmony*. 2008. url: https://mail.mozilla.org/pipermail/es-discuss/2008-August/006837.html.

[25] www.javatpoint.com. *ES5 vs ES6 - javatpoint*. 2022. url: https://www.javatpoint.com/es5-vs-es6.

[26] Nishi Mahto. *Compiling vs. Transpiling | StackOverFlow*. 2021. url: https://stackoverflow.com/questions/44931479/compiling-vs-transpiling.

[27] babeljs.io. *Babel · The compiler for next generation JavaScript*. 2022. url: https://babeljs.io/.

[28] npmjs.com. *Bublé | npm: The blazing fast, batteries-included ES2015 compiler*. 2022. url: https://www.npmjs.com/package/buble.

[29] GitHub. *Traceur Compiler | GitHub: A JavaScript.next-to-JavaScript-of-today compiler*. 8.07.2022. url: https://github.com/google/traceur-compiler.

[30] Jamie Builds. *The super tiny compiler*. Ed. by GitHub. 2020. url: https://github.com/jamiebuilds/the-super-tiny-compiler.

[31] Stuart Langridge. *DHTML utopia: Modern web design using JavaSript & DOM*. Richmond, Australia: sitePoint Pty, 2005. isbn: 0957921896.

[32] GitHub Octoverse. *The State of the Octoverse*. 2022. url: https://octoverse.github.com/#top-languages-over-the-years.

[33] SlashData. *State of Developer Nation | 22th Edition*. 2022.

[34] SlashData. *Developer Nation Pulse Report | 22th edition*. 2022. url: https://www.developernation.net/resources/graphs.

[35] SlashData. *Methodology | SlashData*. 2022. url: https://www.slashdata.co/methodology/.

[36] TIOBE. *TIOBE Index - TIOBE*. 3.06.2022. url: https://www.tiobe.com/tiobe-index/.

[37] RedMonk. *RedMonk Statistics*. 2022. url: https://redmonk.com/data/.

[38] Stack Overflow. *Stack Overflow Developer Survey 2022 | Stack Overflow*. 2022. url: https://survey.stackoverflow.co/2022/.

[39] Node.js. *About | Node.js*. 6.04.2022. url: https://nodejs.org/en/about/.

[40] Jonathan Robie and Texcel Research. *What is the Document Object Model?* 1998. url: https://www.w3.org/TR/REC-DOM-Level-1/introduction.html.

[41] selfhtml.org, ed. *JavaScript/DOM*. 2022. url: https://wiki.selfhtml.org/wiki/JavaScript/DOM.

[42] Adrian Senecki. *Web development stacks – which stacks (should) we use in 2022?* Ed. by The Software House. 7.09.2020. url: https://tsh.io/blog/web-development-stacks/.

[43] MongoDB. *What Is A Technology Stack? Tech Stacks Explained | MongoDB*. 2022. url: https://www.mongodb.com/basics/technology-stack.

[44] Tapan Patel. *Guide to choose the web technology Stack*. 2022. url: https://www.thirdrocktechkno.com/blog/how-to-choose-a-technology-stack-for-web-application-development/.

[45] SitePoint Forums, ed. *JavaScript Flavors?* 2015. url: https://www.sitepoint.com/community/t/javascript-flavors/195567/4.

[46] Reactjs.org. *React – A JavaScript library for building user interfaces*. 2022. url: https://reactjs.org/.

[47] Sacha Greif. *What's New in the 2021 State of JavaScript Survey*. 16.02.2022. url: https://dev.to/sachagreif/whats-new-in-the-2021-state-of-javascript-survey-4eej.

[48] stateofjs.com. *The State of JS 2021: About*. Ed. by Sacha Greif. 2022. url: https://2021.stateofjs.com/en-US/about.

[49] svelte.dev. *Home | Svelte: Cybernetically enhanced web apps*. 2022. url: https://svelte.dev/.

[50] Vue.js. *Home | Vue.js: The Progressive JavaScript Framework*. 2022. url: https://vuejs.org/.

[51] preactjs.com. *Home | Preact: Fast 3kB alternative to React with the same modern API*. 2022. url: https://preactjs.com/.

[52] angular.io. *Angular | Home*. 2022. url: https://angular.io/.

[53] nextjs.org. *Introduction | Learn Next.js*. 2022. url: https://nextjs.org/learn/foundations/about-nextjs.

[54] expressjs.com. *Express - Node.js web application framework*. 2022. url: https://expressjs.com/.

[55] nestjs.com. *NestJS - A progressive Node.js framework*. 2022. url: https://nestjs.com/.

[56] nuxtjs.org. *Home | Nuxt: The Intuitive Vue Framework*. 2022. url: https://nuxtjs.org/.

[57] strapi.io. *Home | Strapi: Open source Node.js Headless CMS*. 2022. url: https://strapi.io/.

[58] gatsbyjs.com. *The Fastest Frontend for the Headless Web*. 2022. url: https://www.gatsbyjs.com/.

[59] solidjs.com. *Home | SolidJS*. 2022. url: https://www.solidjs.com/.

[60] Stack Overflow. *Home | StackOverFlow*. 2022. url: https://stackoverflow.com/.

[61] Stack Overflow. *Stack Overflow Developer Survey 2021 | Stack OverFlow*. 2021. url: https://insights.stackoverflow.com/survey/2021.

[62] Chika Ibeneme. *5 Best Tech Stacks You Need to Consider for 2022*. 2022. url: https://www.liquidweb.com/blog/tech-stack/.

[63] Reactjs.org. *Versions | React*. 2022. url: https://reactjs.org/versions/.

[64] Mark Thompson. *Discontinued Long Term Support for AngularJS*. Ed. by Angular Blog. 2022. url: https://blog.angular.io/discontinued-long-term-support-for-angularjs-cc066b82e65a.

[65] facebook. *React GitHub Repository*. 2022. url: https://github.com/facebook/react.

[66] meta. *Facebook*. 2022. url: https://www.facebook.com/.

[67] meta. *Instagram*. 2022. url: https://www.instagram.com/.

[68] reactnative.dev. *Home | React Native: Learn once, write anywhere*. 2022. url: https://reactnative.dev/.

[69] Reactjs.org. *Getting Started – React*. 2022. url: https://reactjs.org/docs/getting-started.html.

[70] Reactjs.org. *Tutorial: Intro to React | React*. 2022. url: https://reactjs.org/tutorial/tutorial.html#before-we-start-the-tutorial.

[71] Frank Zammetti. *Modern Full-Stack Development*. Ed. by SpringerLink. 2020. url: https://link.springer.com/book/10.1007/978-1-4842-5738-8?noAccess=true.

[72] Michael Hoffmann. *Why React (Re-)Renders a Component*. Ed. by Mokkapps. 2022. url: https://mokkapps.de/blog/debug-why-react-re-renders-a-component/.

[73] freeCodeCamp.org. *Interpreted vs Compiled Programming Languages: What's the Difference?* 1.10.2020. url: https://www.freecodecamp.org/news/compiled-versus-interpreted-languages/.

[74] w3schools.com. *React JSX*. 2022. url: https://www.w3schools.com/react/react_jsx.asp.

[75] Reactjs.org. *Create a New React App*. 2022. url: https://reactjs.org/docs/create-a-new-react-app.html#recommended-toolchains.

[76] JS Mastery. “Backend-Roadmap”. In: (2022).
[77] Tania Rascia. *React Tutorial: An Overview and Walkthrough*. 2022. url: https://www.taniarascia.com/getting-started-with-react/.
[78] meta. *React Developer Tools | Chrome Web Store*. 2022. url: https://chrome.google.com/webstore/detail/react-developer-tools/fmkadmapgofadopljbjfkapdkoienihi.
[79] reactnative.dev. *Learn the Basics | React Native*. 2022. url: https://reactnative.dev/docs/tutorial.
[80] electronjs.org. *Home | Electron: Plattformübergreifende Desktop-Anwendungen mit JavaScript, HTML und CSS entwickeln*. 2022. url: https://www.electronjs.org/.
[81] Angular.org. *AngularJS - Superheroic JavaScript MVW Framework*. 2022. url: https://angularjs.org/.
[82] agiledrop. *Angular or AngularJS?* 2022. url: https://www.agiledrop.com/blog/angular-or-angularjs.
[83] Mark Thompson. *Finding a Path Forward with AngularJS*. Ed. by Angular Blog. 2021. url: https://blog.angular.io/finding-a-path-forward-with-angularjs-7e186fdd4429.
[84] angular.io. *Angular | Docs*. 2022. url: https://angular.io/docs.
[85] PayPal, ed. *PayPal Community*. 2022. url: https://www.paypal-community.com/t5/PayPal-Community/ct-p/en.
[86] Made by Angular. *Made by Angular*. 2022. url: https://www.madewithangular.com/categories/angularjs/.
[87] Victor Savkin. *Understanding Angular Ivy: Incremental DOM and Virtual DOM*. 14.01.2019. url: https://blog.nrwl.io/understanding-angular-ivy-incremental-dom-and-virtual-dom-243be844bf36.
[88] AnAr Corporate. *Understanding Angular Ivy: Incremental DOM and Virtual DOM*. 2019. url: https://anarsolutions.com/understanding-angular-ivy/.
[89] webassemlby.org. *WebAssembly*. 2022. url: https://webassembly.org/.
[90] Corey Butler. “The Many Flavors of JavaScript”. In: *Medium* (30.03.2018). url: https://goldglovecb.medium.com/the-many-flavors-of-javascript-ba4a076ada29.
[91] Wikimedia Commons. *File:WebAssembly Logo.svg*. 2022. url: https://commons.wikimedia.org/wiki/File:WebAssembly_Logo.svg.
[92] Vue.js. *Logo | Vue.js*. 2022. url: https://v2.vuejs.org/.
[93] Vue.js. *Using Vue with TypeScript | Vue.js*. 2022. url: https://vuejs.org/guide/typescript/overview.html.
[94] Vue.js. *Introduction | Vue.js*. 2022. url: https://vuejs.org/guide/introduction.html.
[95] Vue.js. *FAQ | Vue.js*. 2022. url: https://vuejs.org/about/faq.html.
[96] Nwose Lotanna Victor. *All about the Vue CLI*. 2021. url: https://blog.openreplay.com/all-about-the-vue-cli.
[97] Vue.js. *Quick Start | Vue.js*. 2022. url: https://vuejs.org/guide/quick-start.html.
[98] vite. *Vite*. 2022. url: https://vitejs.dev/guide/.
[99] GitHub. *petite-vue | GItHub*. 2022. url: https://github.com/vuejs/petite-vue.
[100] jquery.org. *jQuery Brand Guidelines | Logos*. 2022. url: https://brand.jquery.org/logos/.
[101] js.foundation. *jQuery*. 2022. url: https://jquery.com/.
[102] MongoDB. *MongoDB: Die Plattform Für Anwendungsdaten*. 30.03.2022. url: https://www.mongodb.com/de-de.
[103] Node.js. *Logos and Graphics | Node.js*. 6.04.2022. url: https://nodejs.org/en/about/resources/.
[104] OpenJS Foundation. *OpenJS Foundation: About*. 2022. url: https://openjsf.org/.

[105] eslint.org. *Home | ESLint - Pluggable JavaScript Linter: Find and fix problems in your JavaScript code*. 2022. url: https://eslint.org/.

[106] webpack.js.org. *Home | webpack*. 2022. url: https://webpack.js.org/.

[107] v8.dev. *V8 JavaScript engine*. 2022. url: https://v8.dev/.

[108] codecademy.com. *Introduction to JavaScript Runtime Environments*. 2022. url: https://www.codecademy.com/article/introduction-to-javascript-runtime-environments.

[109] Node.js. *Introduction to Node.js | Node.js*. 21.04.2022. url: https://nodejs.dev/learn.

[110] tutorialsteacher.com. *Node.js Console / REPL*. 2022. url: https://www.tutorialsteacher.com/nodejs/nodejs-console-repl.

[111] DenoLand. *Home | Deno: A modern runtime for JavaScript and TypeScript*. 2022. url: https://deno.land/.

[112] npmjs.com. *About | npm*. 3.06.2022. url: https://www.npmjs.com/about.

[113] npmjs.com. *npm Docs | npm*. 22.04.2022. url: https://docs.npmjs.com/about-npm.

[114] npmjs.com. *ts-node | npm*. 2022. url: https://www.npmjs.com/package/ts-node#overview.

[115] npmjs.com. *n | npm*. 2022. url: https://www.npmjs.com/package/n.

[116] npmjs.com. *slideshow | npm search*. Ed. by npmjs.com. 2022. url: https://www.npmjs.com/search?q=slideshow.

[117] AltexSoft. *What is API: Definition, Types, Specifications, Documentation*. 2022. url: https://www.altexsoft.com/blog/engineering/what-is-api-definition-types-specifications-documentation/.

[118] GeeksforGeeks. *MERN Stack | GeeksforGeeks*. 2019. url: https://www.geeksforgeeks.org/mern-stack/.

[119] expressjs.com. *Basic routing with Express*. 2022. url: https://expressjs.com/en/starter/basic-routing.html.

[120] MongoDB. *MongoDB Brand Resources*. 2022. url: https://www.mongodb.com/brand-resources.

[121] michster. *Was ist ein Full Stack Webdeveloper*. 2019. url: https://michster.de/was-macht-ein-fullstack-webdeveloper/.

[122] MongoDB. *JSON And BSON*. 2022. url: https://www.mongodb.com/json-and-bson.

[123] Edward Kring. *Role of MongoDB In MERN/MEAN Stack*. 2021. url: https://dzone.com/articles/role-of-mongodb-in-mernmean-stack.

[124] AltexSoft. *Comparing Database Management Systems: MySQL, PostgreSQL, MSSQL Server, MongoDB, Elasticsearch, and others*. 2022. url: https://www.altexsoft.com/blog/business/comparing-database-management-systems-mysql-postgresql-mssql-server-mongodb-elasticsearch-and-others/.

[125] MongoDB. *MongoDB Shell (mongosh)*. 2022. url: https://www.mongodb.com/docs/mongodb-shell/.

[126] robomongo.org. *Robo 3T | Free, open-source MongoDB GUI*. 2022. url: https://robomongo.org/.

[127] MongoDB. *What is MongoDB Atlas? | MongoDB Atlas*. 2022. url: https://www.mongodb.com/docs/atlas/.

[128] MongoDB. *Start with ATLAS Guides*. 2022. url: https://www.mongodb.com/docs/guides/atlas/account/.

[129] Anaconda Inc. *Pyscript.net*. 2022. url: https://pyscript.net/.

[130] Jim Anderson. *How to Implement a Python Stack*. 2019. url: https://realpython.com/how-to-implement-python-stack/.

[131] JavaScript Tutorial. *Implementing a Javascript Stack*. 2022. url: https://www.javascripttutorial.net/javascript-stack/.

[132] zero added sugar. *Architecture vs. tech stack*. 2021. url: https://rishat.us/architecture-vs-tech-stack/.

[133] Mixpanel. *Was ist ein Technologie-Stack?* 2021. url: https://mixpanel.com/de/blog/was-ist-ein-technologie-stack/.

[134] Gomes, Luis Mendes, Francisco Martins, and Hélia Guerra. "Teaching Web Programming Using the MEAN Stack". In: *The Impact of the 4th Industrial Revolution on Engineering Education*. Ed. by Michael E. Auer, Hanno Hortsch, and Panarit Sethakul. Advances in Intelligent Systems and Computing. Cham: Springer International Publishing, 2020, pp. 275–262. isbn: 10.1007/978-3-030-40271-6. url: https://doi.org/10.1007/978-3-030-40271-6_26.

[135] MongoDB. *How To Use MERN Stack: A Complete Guide*. 22.08.2022. url: https://www.mongodb.com/languages/mern-stack-tutorial.

[136] Microsoft Azure. *Was ist DevOps? Eine Erläuterung*. 2022. url: https://azure.microsoft.com/de-de/overview/what-is-devops/.

[137] Wikipedia, ed. *DevOps*. 2022. doi: Page. url: https://de.wikipedia.org/w/index.php?title=DevOps&oldid=224050814.

[138] Milad Safar. *Was verbirgt sich hinter dem Begriff DevOps?* 2020. url: https://weissenberg-group.de/was-ist-devops/.

[139] Amazon Web Services, Inc. *Was ist DevOps? - Amazon Web Services (AWS)*. 2022. url: https://aws.amazon.com/de/devops/what-is-devops/#Die_DevOps-Kultur.

[140] stepstone.de. *Jobs sind unser Job | StepStone*. 3.07.2022. url: https://www.stepstone.de/.

[141] LinkedIn. *LinkedIn*. 2022. url: https://www.linkedin.com/.

[142] Amazon Web Services, Inc. *DevOps - Amazon Web Services (AWS)*. 2022. url: https://aws.amazon.com/devops/.

[143] prettier.io. *Prettier · Opinionated Code Formatter*. 30.03.2022. url: https://prettier.io/.

[144] Visual Studio. *Visual Studio: IDE und Code-Editor für Softwareentwickler und -teams*. 2022. url: https://visualstudio.microsoft.com/de/.

[145] code.visualstudio.com. *Visual Studio Code: Code Editing. Redefined*. 2022. url: https://code.visualstudio.com/.

[146] Jetbrains. *WebStorm: Die intelligenteste JavaScript-IDE, von JetBrains*. 2022. url: https://www.jetbrains.com/de-de/webstorm/.

[147] vim.org. *vim online*. 2022. url: https://www.vim.org/.

[148] atom.io. *Atom: A hackable text editor for the 21st Century*. 2022. url: https://atom.io/.

[149] npmjs.com. *Home | npm*. 30.03.2022. url: https://www.npmjs.com/.

[150] yarnpkg.com. *Home | Yarn*. 2022. url: https://yarnpkg.com/.

[151] bower.io. *Bower*. 2022. url: https://bower.io/.

[152] nvm.github.io. *Using a Node Version Manager | Introduction*. 2022. url: https://npm.github.io/installation-setup-docs/installing/using-a-node-version-manager.html.

[153] Amit Manchanda. *Monolithic vs Microservices Architecture: Which Option is Right for Your Enterprise?* 2021. url: https://www.netsolutions.com/insights/monolithic-vs-microservices/.

[154] GitLab.org. *GitLab | GitLab Pages*. 2022. url: https://docs.gitlab.com/ee/user/project/pages/.

[155] depositphotos. *Pictures by depositphotos*. 2022. url: https://de.depositphotos.com/vector-images/full-stack-developer.html.

[156] Anuj Adhikari. "Full Stack JavaScript: Web Application Development with MEAN". PhD thesis. Metropolia Ammattikorkeakoulu, 2016. url: https://www.theseus.fi/handle/10024/116597.

[157] npmjs.com. *npm Blog Archive: JavaScript Usage by Industry*. Ed. by npmjs.com. 13.01.2021. url: https://blog.npmjs.org/post/175311966445/javascript-usage-by-industry.

[158] Vin Clancy. *The learning curve is a kille - be aware of it | Traffic and Copy*. 2017. url: https://medium.com/trafficandcopy/the-learning-curve-is-killer-be-aware-of-it-31404ce0d398.

[159] collinsdictionairy.com. *Learning Curve | Collins Dictionairy*. 2022. url: https://www.collinsdictionary.com/de/worterbuch/englisch/learning-curve.

[160] Coursera. *Coursera | Degrees, Certificates, & Free Online Courses*. 2022. url: https://www.coursera.org/.

[161] LinkedIn. *LinkedIn Learning | Online Courses for Creative, Technology, Business Skills*. 2022. url: https://www.linkedin.com/learning/.

[162] Udemy. *Udemy | Online Courses - Learn Anything, On Your Schedule*. 2022. url: https://www.udemy.com/.

[163] vuejs.de. *Vite 2 – das noch bessere Development und Build Tool | Vuejs.de*. 26.08.2022. url: https://vuejs.de/artikel/vite-das-noch-bessere-development-und-build-tool/.

[164] WebsiteSetup. *30+ Website Load Time Statistics*. 2021. url: https://websitesetup.org/news/website-load-time-statistics/.

[165] Backlinko. *We Analyzed 5.2 Million Webpages. Here's What We Learned About PageSpeed*. 2019. url: https://backlinko.com/page-speed-stats.

[166] reactfordesigners.com. *React for Designers*. 2022. url: https://reactfordesigners.com/.

[167] Dave Gavigan. "The History of Angular - The Startup Lab - Medium". In: *The Startup Lab* (4.03.2018). url: https://medium.com/the-startup-lab-blog/the-history-of-angular-3e36f7e828c7.

[168] Vue Mastery. *Vue Mastery | The Ultimate Learning Resource for Vue.js Developers*. 2022. url: https://www.vuemastery.com/.

[169] endoflife.date. *End of Life Date*. 22.08.2022. url: https://endoflife.date.

[170] Francois-xavier P. Darveau. "You SHOULD Learn Vanilla JavaScript Before JS Frameworks". In: (24.06.2021). url: https://snipcart.com/blog/learn-vanilla-javascript-before-using-js-frameworks.

[171] Mike Lopez. *What do living room board games and mobile games have in common? Real-time interactions*. Ed. by pocketgamer.biz. 2022. url: https://www.pocketgamer.biz/interview/74581/what-do-living-room-board-games-and-mobile-games-have-in-common-real-time-interactions/.

[172] Spiel des Jahres. *Senso*. 2022. url: https://www.spiel-des-jahres.de/spiele/senso/.

[173] Netninja.dev. *MERN Stack Tutorial*. 2022. url: https://netninja.dev/p/mern-stack-tutorial.

[174] Microsoft. *Set up Node.js on WSL 2*. 2022. url: https://docs.microsoft.com/en-us/windows/dev-environment/javascript/nodejs-on-wsl.

[175] Visual Studio. *Version Control in Visual Studio Code*. 2022. url: https://code.visualstudio.com/docs/editor/versioncontrol.

[176] GitLab.org. *GitLab.org | Home*. 2022. url: https://gitlab.com/gitlab-org/gitlab.

[177] npmjs.com. *nodemon | npm*. 2022. url: https://www.npmjs.com/package/nodemon.

[178] npmjs.com. *dotenv | npm*. 2022. url: https://www.npmjs.com/package/dotenv.

[179] Postman. *Postman API Platform*. 2022. url: https://www.postman.com/.

[180] npmjs.com. *cors | npm*. 2022. url: https://www.npmjs.com/package/cors.

Glossary and Acronyms

AJAX (Asychronous JavaScript and XML). 23, 38

API (Application Programming Interface) Standardised interface to connect two computers or computer programs together that are otherwise incompatible. III, V, 14, 15, 27, 34, 35, 37, 39, 43, 44, 46, 47, 50, 53, 59, 68, 76, 78, 82, 98–100, 107, 108, 134

BSON (Binary JSON) Similar to JSON. Standard for data interchange on the Web but with the ability to store binary data as well [**MongoDB.json.bson.24.06.2022**]. II, 46, 53

CD (Continuous Delivery) Automation in deploying an application. 57

CI (Continuous Integration) Automation in building, testing and deploying application development. 57

CI/CD (Continuous Integration / Continuous Delivery) Paradigm for automation in building, testing and deploying application development. 57, 58, 61

CLI (Command Line Interface) Processes commands to a computer program in the form of lines of text. 31–33, 35, 38, 47, 76, 95

CRUD (Create, Read, Update, Delete) Acronym for the default database operations. 47

CSR (Client-Side Rendering) JavaScript gets loaded in browser. 15

CSS (Cascading Style Sheets) Syntax to style a Web Page layout defined with HTML. 3, 4, 14, 29, 32, 34, 66, 83, 104, 134

DBMS (Database Management System) Software system that enables users to define, create, maintain and control access to a database. 15, 45, 53

DevOps (Development Operations) Automation or process of ensuring a faster Web Development. II, 10, 50, 52, 54, 56–58, 60, 61, 82, 95, 109

DHTML (Dynamic HTML) Set of Web Development Technologies with JavaScript for providing interactivity. 4

DOM (Document Object Model) The visible tree structure of HTML elements in a Web Browser. I, II, 3, 4, 13, 14, 22–24, 27, 29, 30, 33, 35, 38, 39, 48, 70, 75, 76, 83, 84

Ecma (European Computer Manufacturers Association) Industry association dedicated to the standardisation of information and communication systems [10]. 4, 6, 7, 74, 120

EOL (End of Life) End of support of a software. After the EOL there are no fixes, security releases or enhancements. 28, 80, 81, 84

ES (ECMAScript) JavaScript standard meant to ensure the interoperability of Web Pages across different Web Browsers; standardised by Ecma International [7]. 4, 6–8, 23, 24, 75, 104

GUI (Graphical User Interface) Type of user interface through which a user gets visual feedback about internal processes. 46

HTML (Hypertext Markup Language) Syntax similar to XML to layout Web Pages, usually transmitted via HTTP. I, 3, 4, 13, 14, 23, 25, 26, 29–32, 34, 38, 48, 66, 83

HTTP (Hypertext Transfer Protocol) Internet protocol to transfer files over the internet, usually used to transmit HTML to Web Browsers. 43, 44, 53

IaaS (Infrastructure-as-a-Service) Monitoring and enforcing infrastructure compliance with version control. 57

IDE (Integrated Development Environment). 28, 94

JS (JavaScript) Universal Language of the Web. 3–5, 12, 14, 16, 20, 21, 23, 26, 28, 34, 46, 48, 51, 53, 67, 76, 98, 134

JSON (JavaScript Object Notation) Standard for data interchange on the Web. II, 7, 43, 45, 46, 53, 99

JSX (JavaScript + XML) Syntax extension to JavaScript. Allows the use of HTML elements in JavaScript [69]. 5, 8, 21, 23–27, 29, 36, 48, 68, 75, 76, 82, 104

LTS (Long Term Support) Version of Software, which gets supported for a longer period of time. Guarantees that critical bugs will be fixed.. 21, 39, 40, 59, 73, 80, 81

MEAN (MongoDB, ExpressJS, AngularJS and NodeJS) Four technologies which are used in this specific Stack; Full-Stack Variant based on JavaScript-Frameworks. 1, 52, 67, 84, 85, 109

MEEN (MongoDB, ExpressJS, EmberJS und NodeJS) Four technologies which are used in this specific Stack; Full-Stack Variant based on JavaScript-Frameworks. 52, 67

MERN (MongoDB, ReactJS, ExpressJS und NodeJS) Four technologies which are used in this specific Stack; Full-Stack Variant based on JavaScript-Frameworks. 1, 2, 25, 52, 53, 67, 84, 85, 87, 88, 94, 108–110

MEVN (MongoDB, ReactJS, VueJS und NodeJS) Four technologies which are used in this specific Stack; Full-Stack Variant based on JavaScript-Frameworks. 1, 52, 67, 84, 85, 109

npm (Node Package Manager) Recommended package manager for Node.js. II, 25, 27, 31, 41, 42, 59, 70, 74, 75, 80–85, 95, 97

nvm (Node Version Manager) Recommended version manager for Node.js. 58, 59, 94, 95

OS (Operating System). 58, 121

PaaS (Platform-as-a-Service) Deploy web applications without needing to provision and manage the infrastructure and application stack. 57

PYPL (PopularitY of Programming Language Index) Statistic of Programming Languages searched for in the search engine google. I, 9, 11

REPL (Read-Evaluate-Print-Loop) Virtual interactive environment (indicated with >_ in console). 41, 42, 46, 95

REST (Representational State Transfer) Paradigm that defines how information is exchanged over HTTP(S). 43, 44, 53

SaaS (Software-as-a-Service) Hosting Application, which can be connected to and used by client. 57

SEO (Search Engine Optimisation). 65

SFC (Single-File Component) Important component style in the Vue.js framework, a *.vue file [94]. 34, 35, 76, 83

SSR (Server-Side Rendering) JavaScript gets loaded in server. 15

TS (TypeScript) JavaScript with syntax for types. 5, 21, 76

UI (User Interface). I, II, IV, 15, 21, 24–26, 30–33, 36, 47, 53, 60, 92, 103, 104, 107, 149

URL (Uniform Resource Locator). 44, 75, 102, 103, 107

UX (User Experience). 54

Wasm (Web Assembly) Portable compilation target for programming languages, enabling deployment on the web for client and server applications [89]. 33, 68, 134

WSL (Windows Subsystem for Linux) Enables the usage of a preferred Linux distribution in Windows OS. 94, 95

WWW (World Wide Web). 4

XML (Extensible Markup Language). 5, 14, 23, 45

YAML (Yet Another Markup Language) Human-readable data-serialisation language. Mostly used in configuration files. 5

Appendix A

Questionnaire

The following pages show the results of the questionnaire that was conducted (for the thesis *Evaluation and Comparison of Full-Stack JavaScript Technologies* by Nadine Weber) with students of the University of Applied Sciences Offenburg. It was originally provided in the German language, due to the students background of studying at a German university. To fit the language in this thesis it was translated to English. The questionnaire acquired 92 results. The setup and results were discussed in *Chapter 3.2*.

After a general overview of the questions the results will be individually shown with a diagram.

Overview of Questions

The questionnaire contained 27 questions of which 19 were mandatory. The mandatory questions are indicated with an asterisk. Multiple Choice questions are marked with '(MC)'. For a better structure individual headings of each topic were created .

General questions

1. Study Course *
2. Semester *
3. Age Group *
4. Have you ever worked in the field of Web Development? * (MC)
5. In which field do you work or would you like to work in the future? (MC)
6. What kind of Web Development are you interested in (the most)? *
7. Did you attend the course Interactive Distributed Systems by Prof. Dr. Rüdebusch during your studies at the University of Applied Sciences Offenburg? *

Programming and Development

8. Have you ever developed in JavaScript? *
9. Have you ever developed in JavaScript outside of the course Interactive Distributed Systems (IVS)? *
10. Which languages have you already worked with in development? *

Full-Stack JavaScript Development

11. Have you ever heard of the term Full-Stack Web Development (before this survey)? *
12. Are you interested in Full-Stack Web Development with JavaScript? *
13. What is your opinion on JavaScript frameworks? *
14. How well do you know Full-Stack JavaScript Web Development? *
15. Which JavaScript Frontend frameworks / libraries have you worked with before?
16. Which JavaScript Backend frameworks have you worked with or at least heard of by name?
17. Which Stack Types do you know by name or have you already used in parts?

Questions about the course Interactive Distributed Systems

18. The topic of Full-Stack JavaScript Web Development is quickly becoming much to learn. What would you find interesting to learn within the lecture or the laboratory of the course Interactive Distributed Systems? *
19. Which frameworks would you like to learn within a laboratory experiment in Interactive Distributed Systems?
20. What other content would you like to learn within the lecture or laboratory of Interactive Distributed Systems related to web development?

Behaviour during Web Development

21. How much would you like to use the concept of modularity? (e.g. write functions in such a way that you can reuse them frequently – also in other projects) *
22. How often would you maintain a web application? (e.g. make updates, add new content) *
23. Are you interested in trends in Web Development or do you like to stick with the familiar and established technologies? *

Learning behaviour

24. How do you get to know new technologies in the field of Web Development? *
25. How do you learn new frameworks? *
26. Are you put off by having to learn a lot at the beginning to be able to use a technology or a framework? *

Comments

27. If you would like to describe the questions or answers in more detail, you have the chance to do so here.

Results

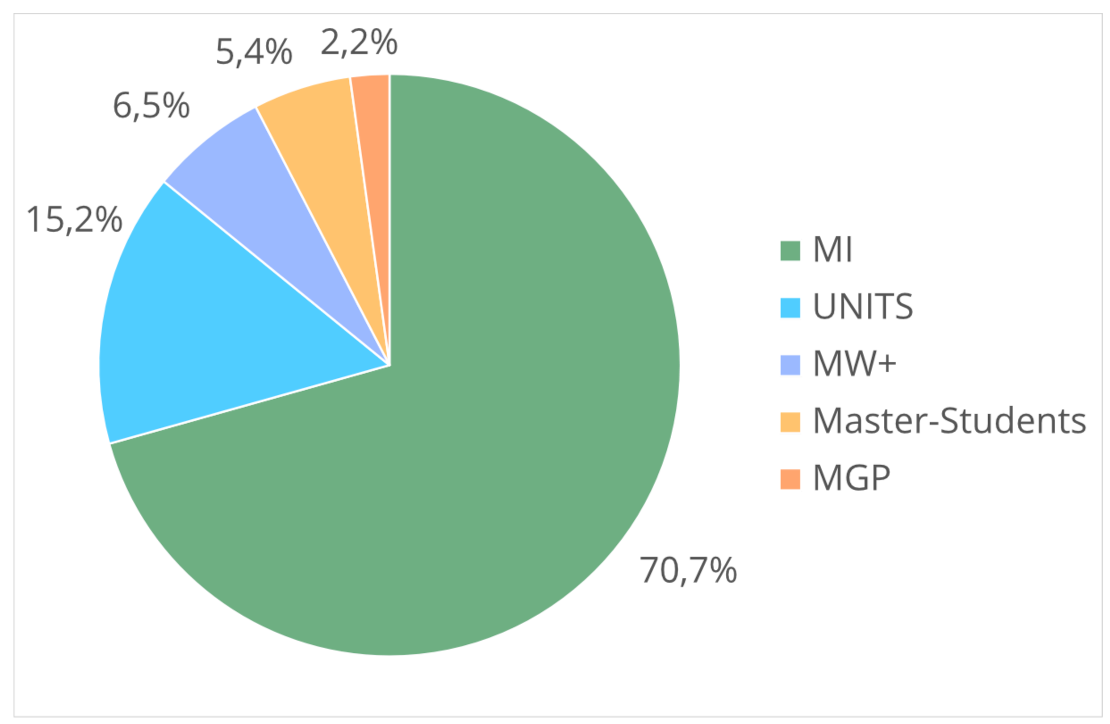


Figure A.1: Study Programmes

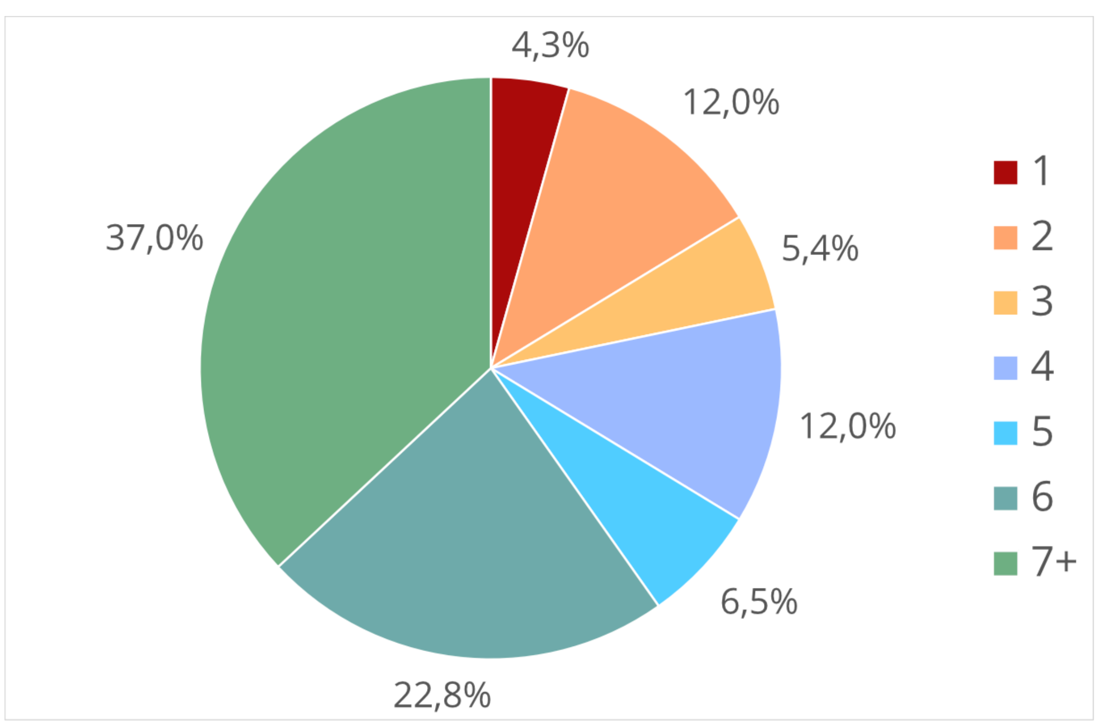


Figure A.2: Number of Semesters

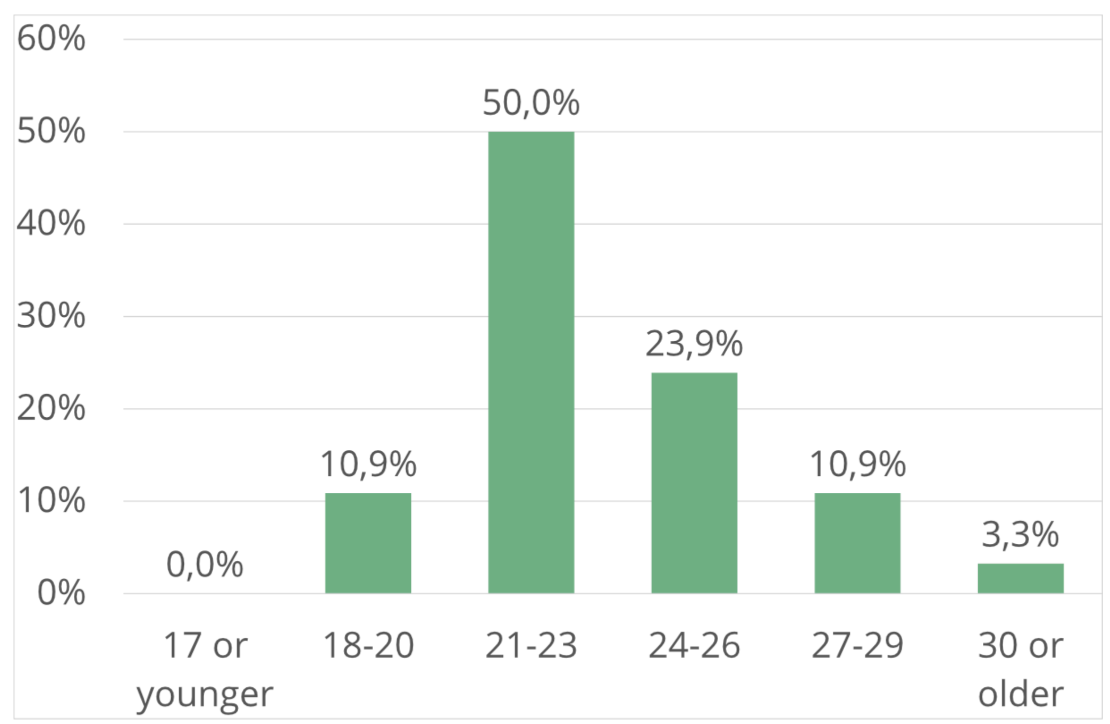


Figure A.3: Age Groups

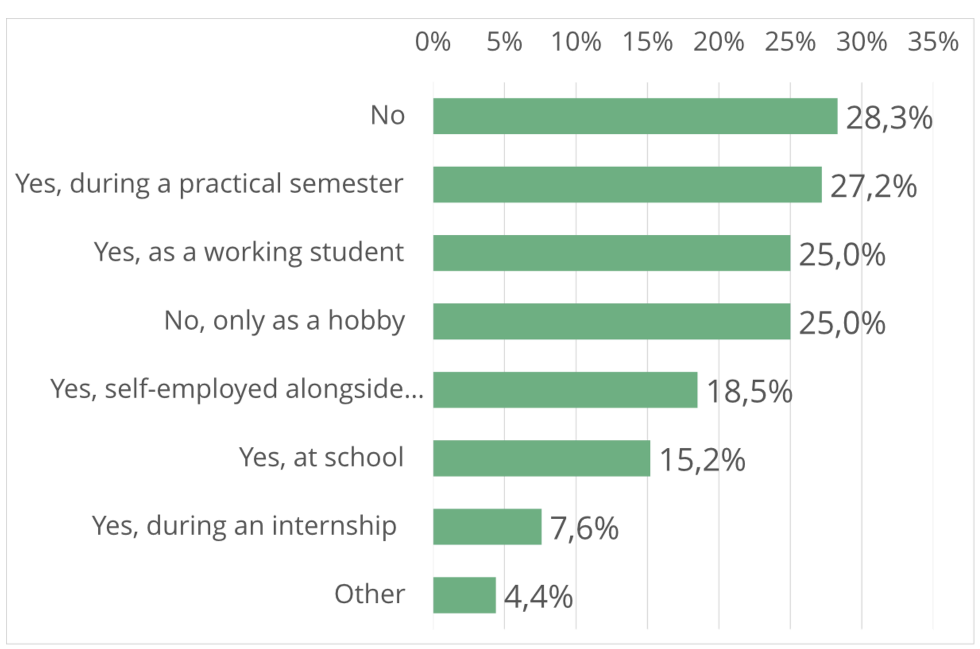


Figure A.4: Experience with Web Development

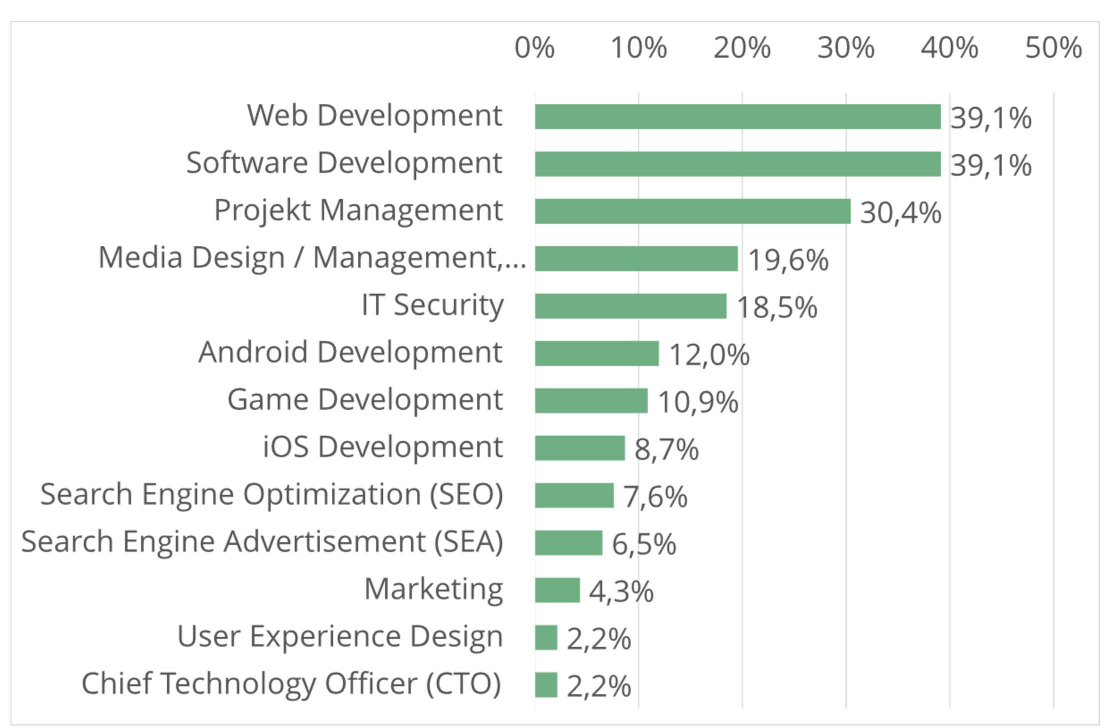


Figure A.5: Web Development Fields

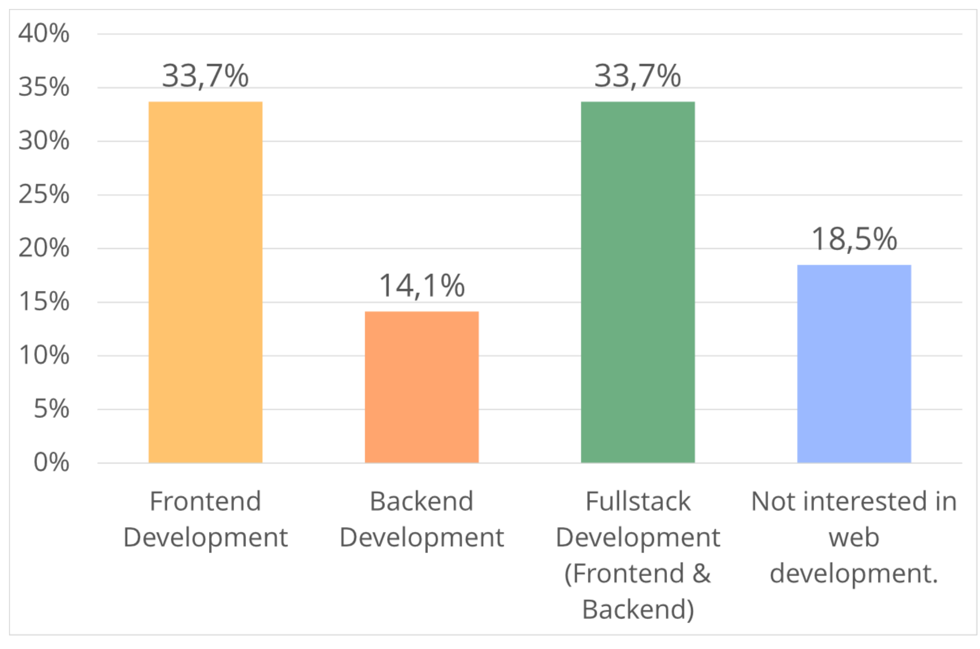


Figure A.6: Interest in Web Development Types

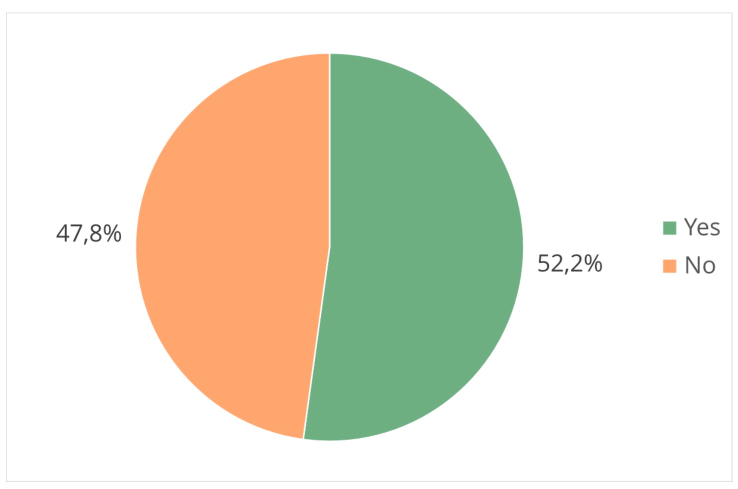
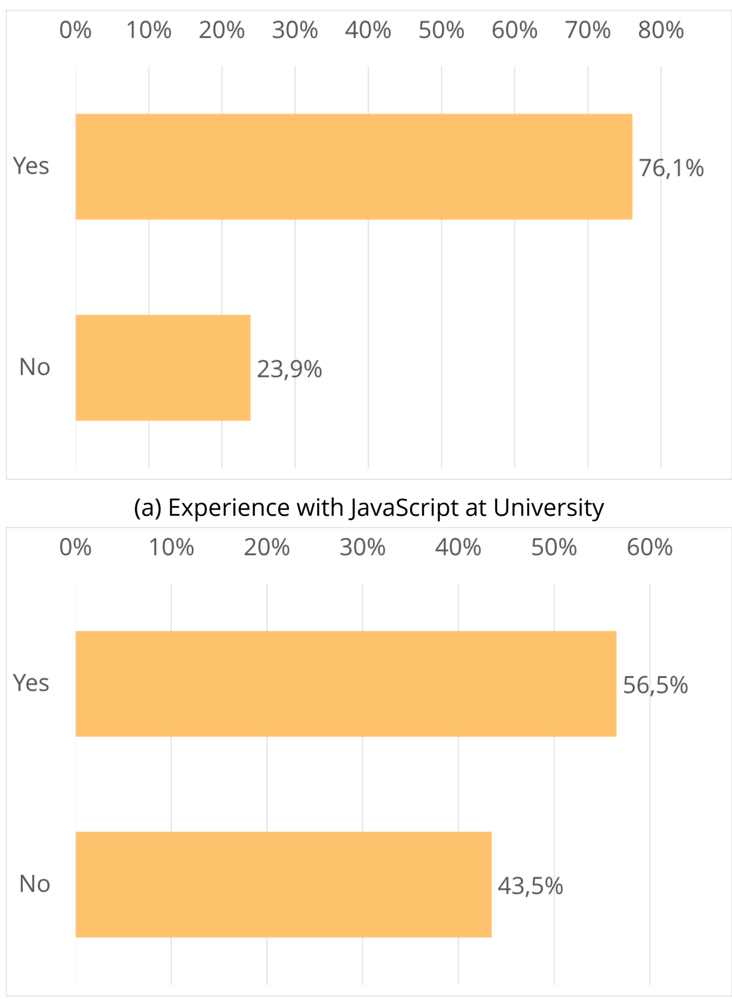


Figure A.7: Attendance of the course “Interactive Distributed Systems”



(a) Experience with JavaScript at University

(b) Experience with JavaScript outside of University

Figure A.8: Experience with JavaScript

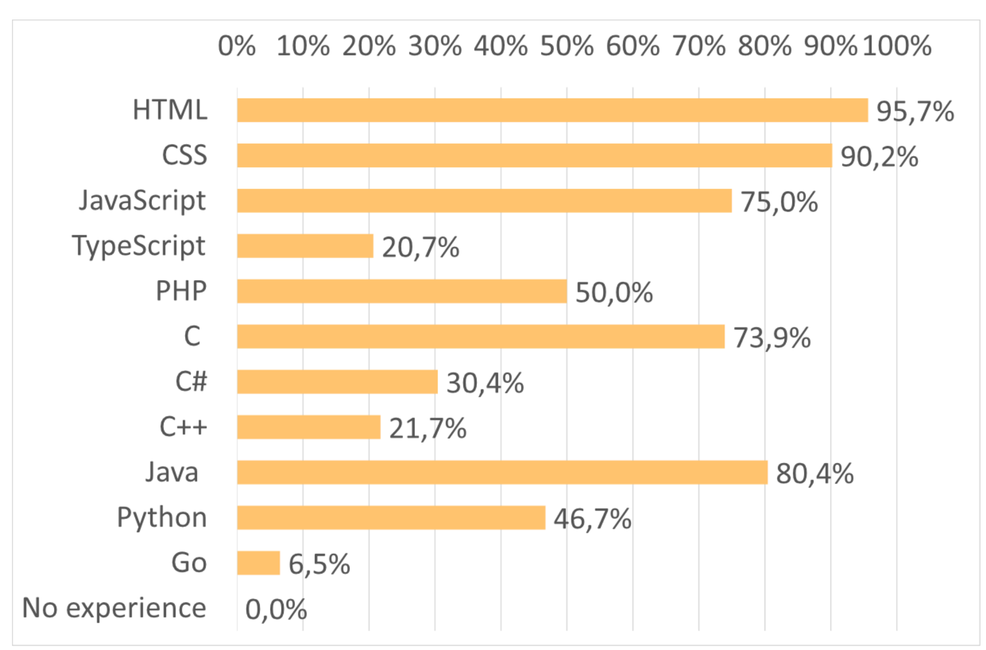


Figure A.9: Language Experience

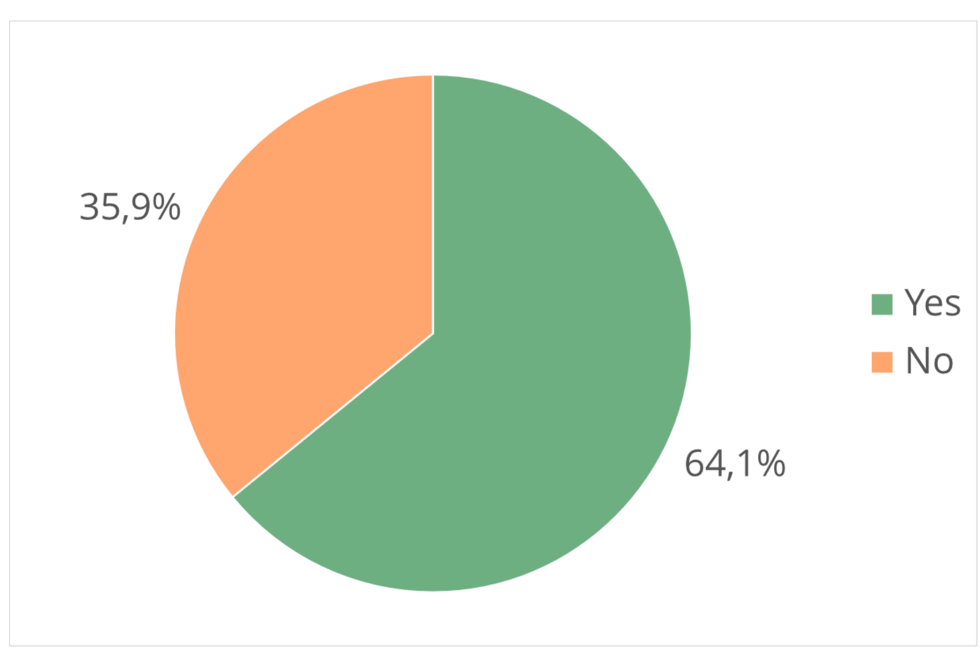


Figure A.10: Familiarity with the term Full-Stack Web Development

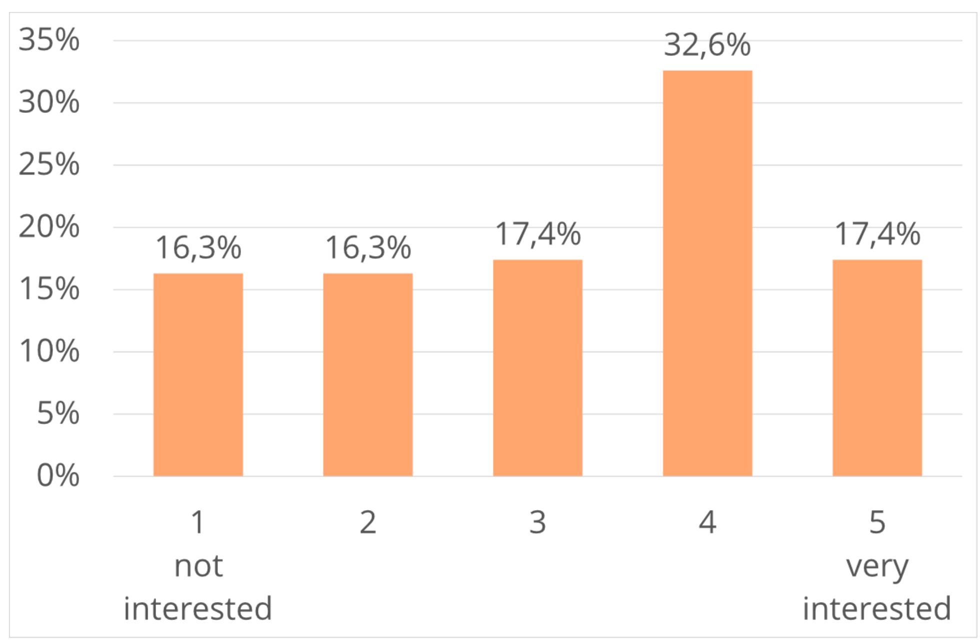


Figure A.11: Interest in Full-Stack Web Development

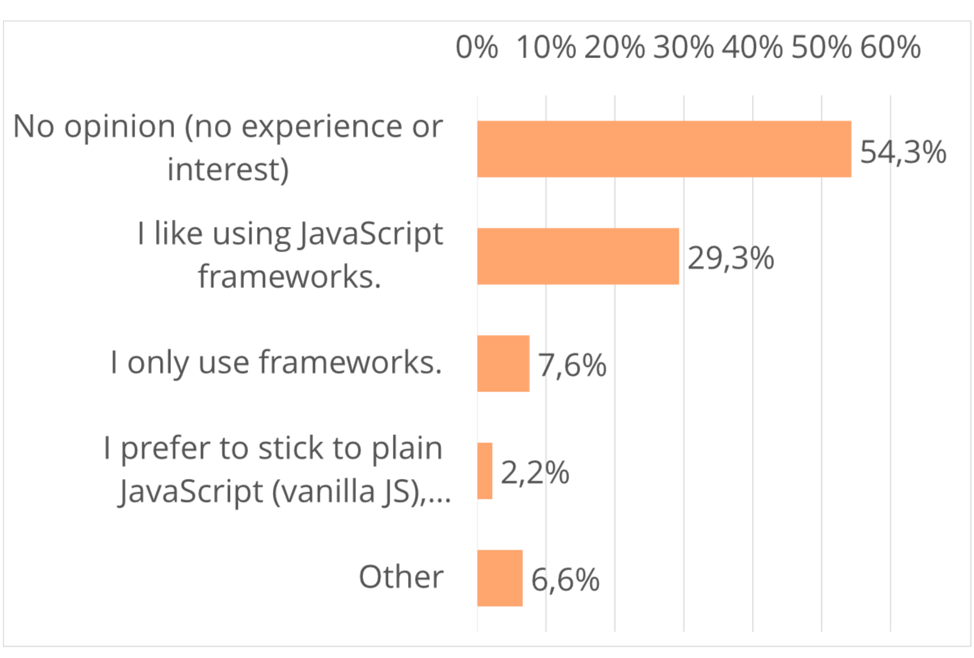


Figure A.12: Opinion on JavaScript Frameworks

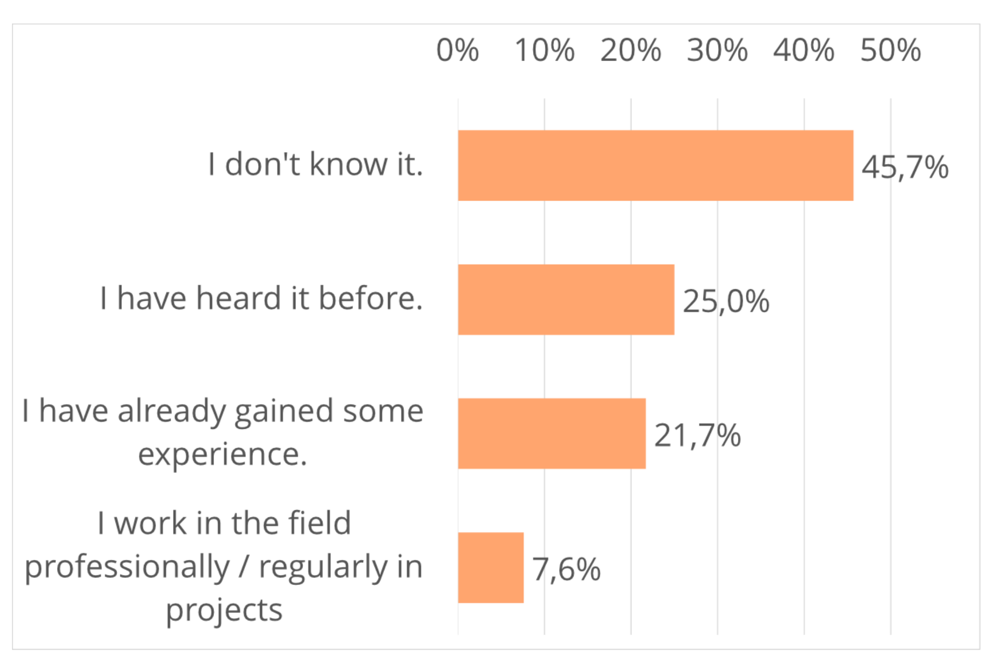


Figure A.13: Familiarity with Full-Stack Web Development

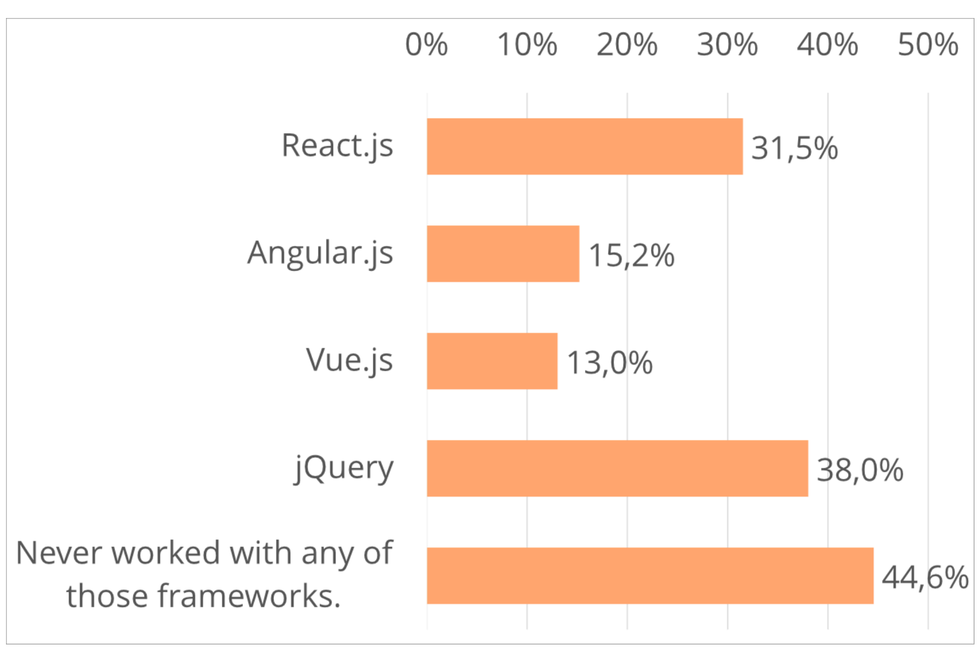


Figure A.14: Knowledge of Frontend Frameworks

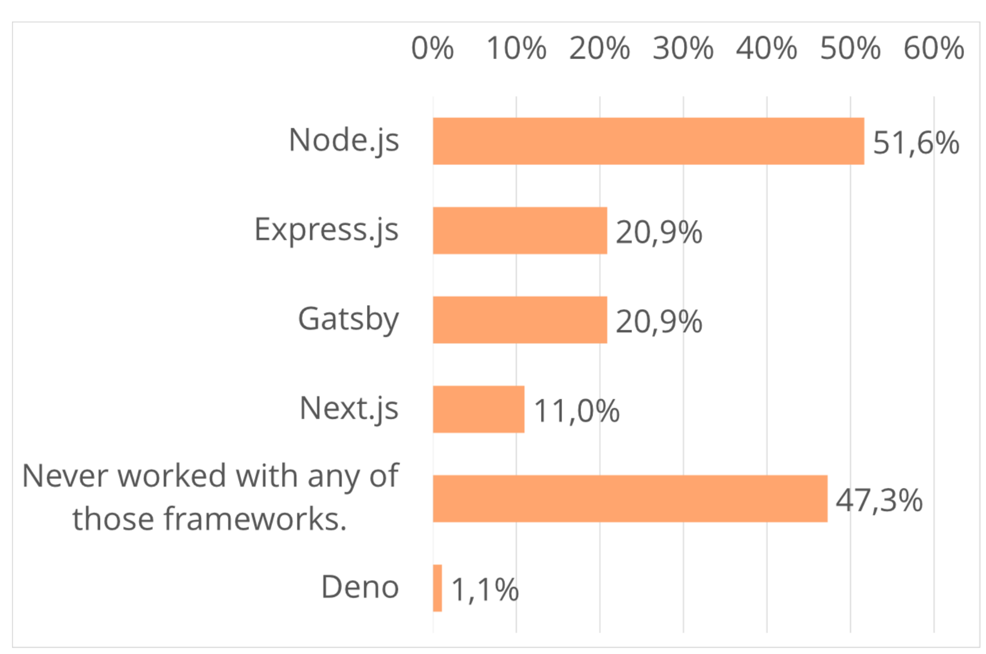


Figure A.15: Knowledge of Backend Frameworks

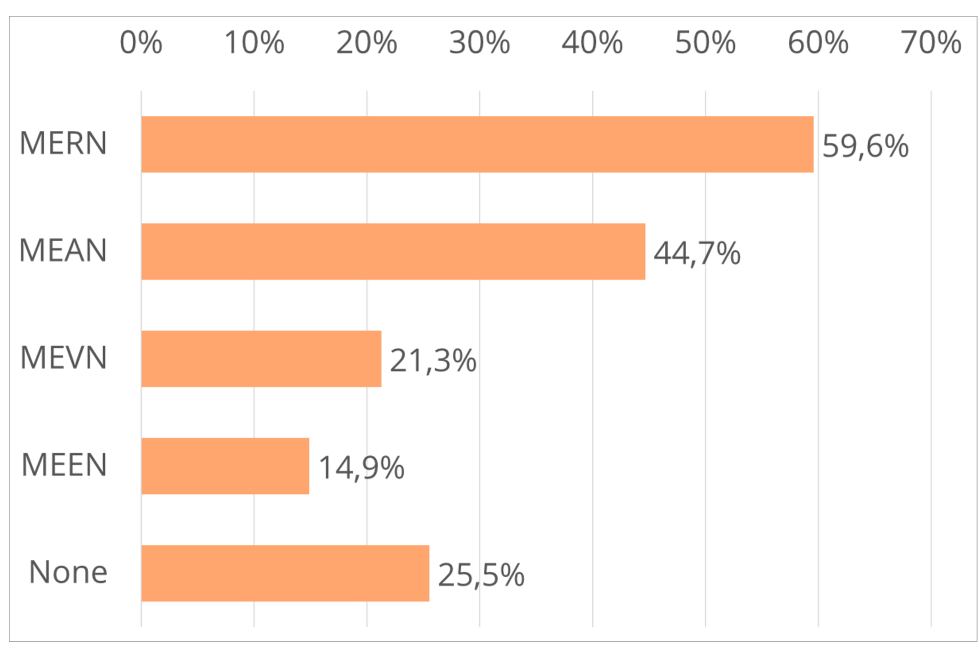


Figure A.16: Knowledge of Stack Types

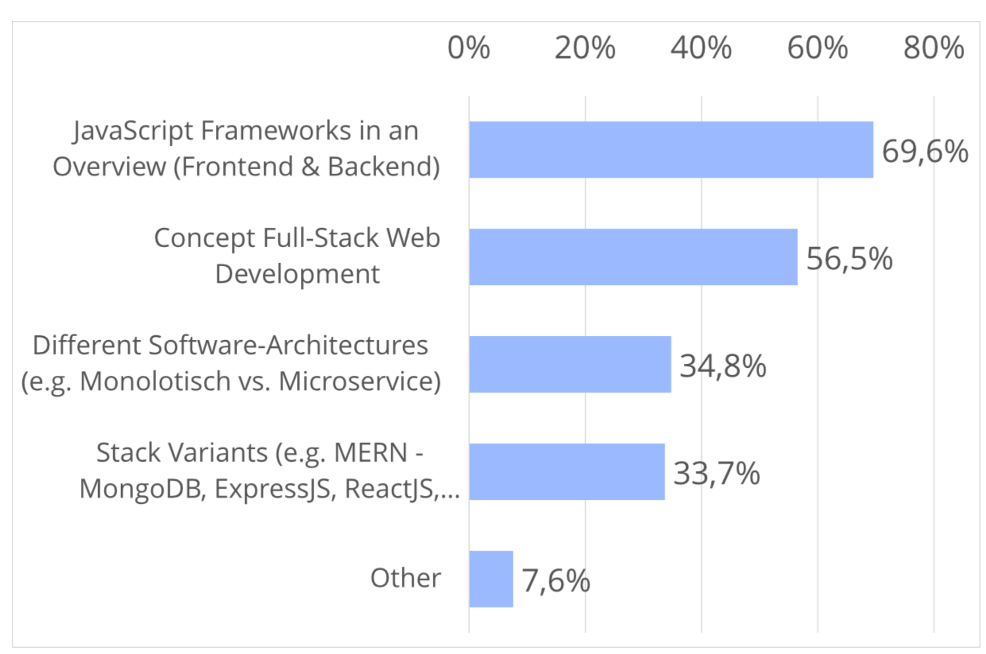


Figure A.17: Possible topics featured in the course Interactive Distributed Systems

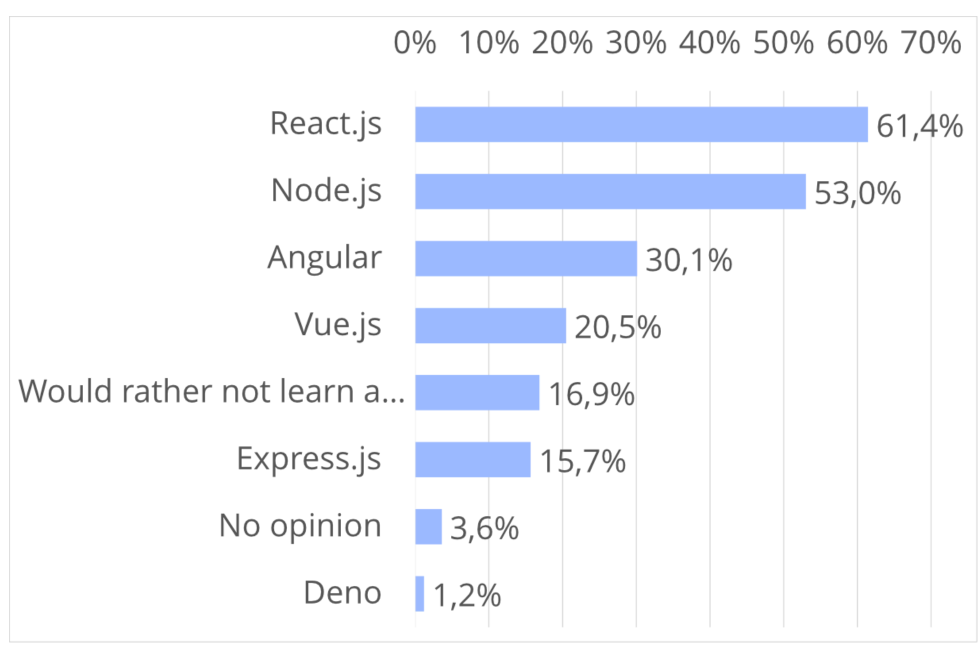


Figure A.18: Frameworks that should be used in a laboratory experiment

(Question 20) *What other content would you like to learn within the lecture or laboratory of Interactive Distributed Systems related to web development?*

- I have no opinion on this, as I did not attend the lecture.
- Everything that is currently trendy in the job market is interesting, so that I don't just have outdated technology skills after my graduation.
- The concrete connection between individual areas, e.g. front-end and back-end, back-end and database.
- You could possibly deal with frameworks like Bootstrap (briefly), since these also concern JS and CSS and this is frequently used in practice.
- REST- and GraphQL-APIs, Wasm
- Show overview of web development possibilities and illustrative examples.
- More details, it's all so superficial that you end up not being able to work with it after all. WordPress on the developer side would definitely be interesting too.
- How to host websites on a server
- I would love to go into more detail regarding the technical workings (i.e. where exactly what code runs when doing Full-Stack JavaScript Development and what security risks might come with that). It would also be cool to go into a bit more detail about the browser APIs, to show how extensive a web application can be. Especially the workflow of working with APIs (defining the target, reading the documentation, setting up the test environment, etc.) would be useful, because you're always confronted with this kind of thing and usually don't know where to start.
- I wouldn't go into details about the frameworks, because in the end it's all about Javascript and the frameworks are also very opinionated.
- React JSX
- I was very satisfied with the course. Even as a beginner/non-expert in this area, I was still able to take away and understand a lot from this lecture/lab. Much more would possibly make it too demanding.
- Development environments and the correct use and structure
- Structure of a stack, or a "stack-101": What it is, what it is used for, how it is applied.
- Realising database access via a web interface
- A good front-end is possible with React. On the other hand, there are also content management systems. I would look at both sides and then teach the predominant technology, React.
- Rendering methods
- I don't attend the lecture, so I don't know what is currently being taught, but I think it would be important to get a basic introduction to the current frameworks and possibly also to build a full-stack web application.
- Modern topics
- Swift
- Web hosting
- HTML, CSS and JavaScript exercises with more topicality. The examples from the lecture seemed outdated.

Figure A.19: Opinions on topics taught in the course Interactive Distributed Systems

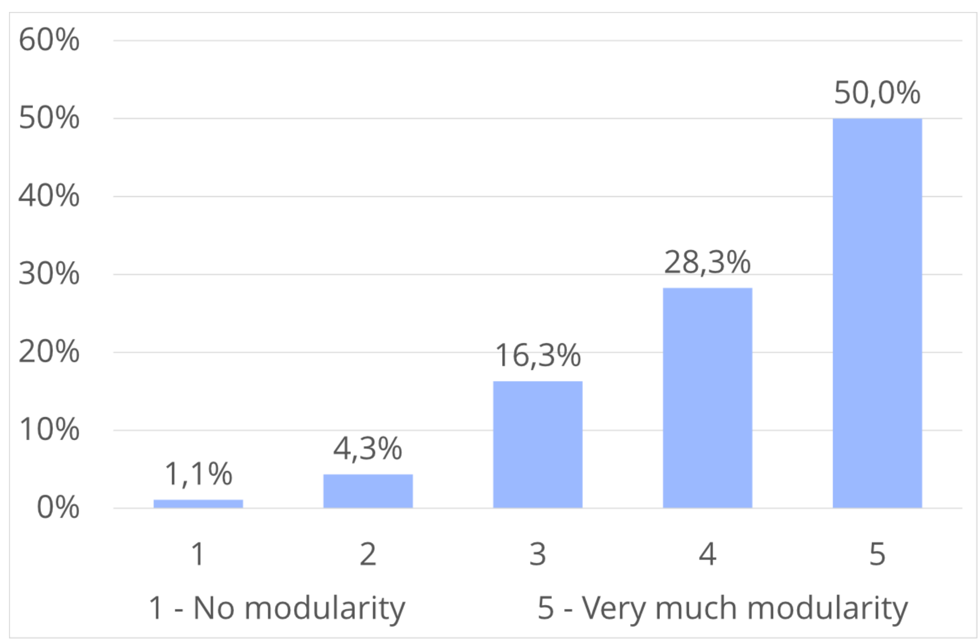


Figure A.20: Interest in the concept of Modularity

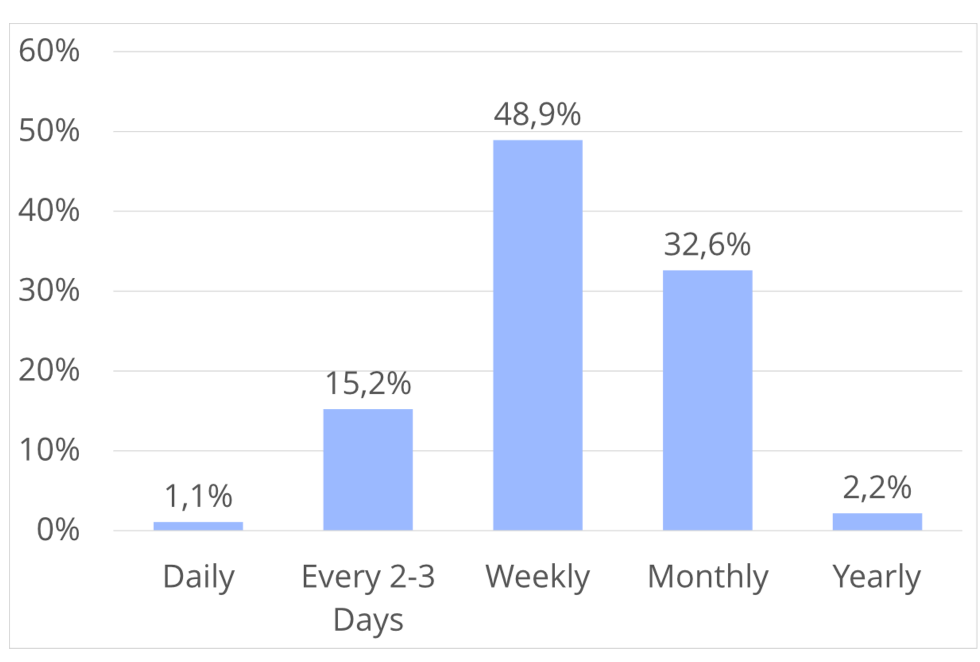


Figure A.21: Expected frequency of maintaining a website

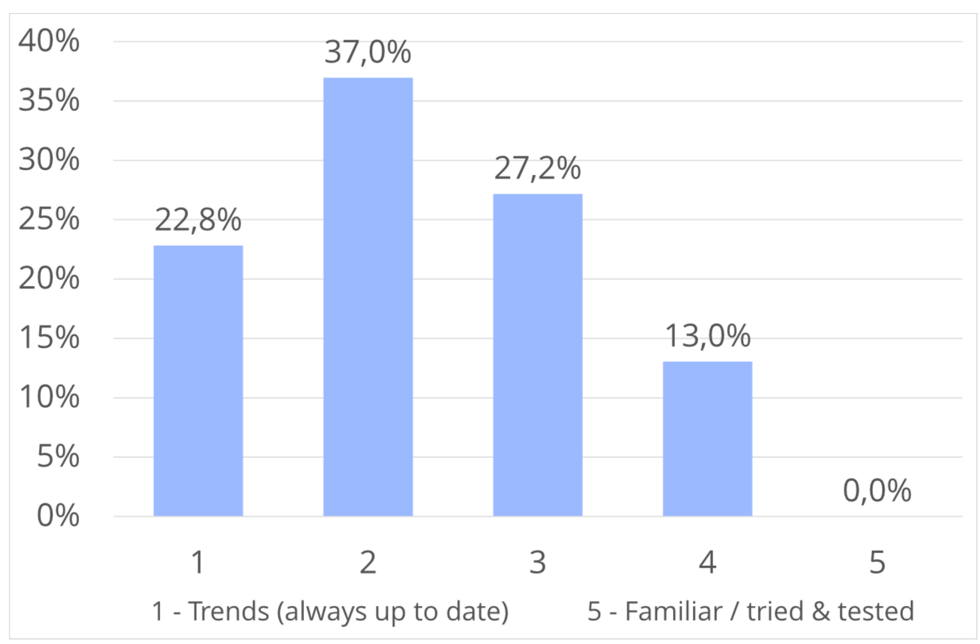


Figure A.22: Interest in trends

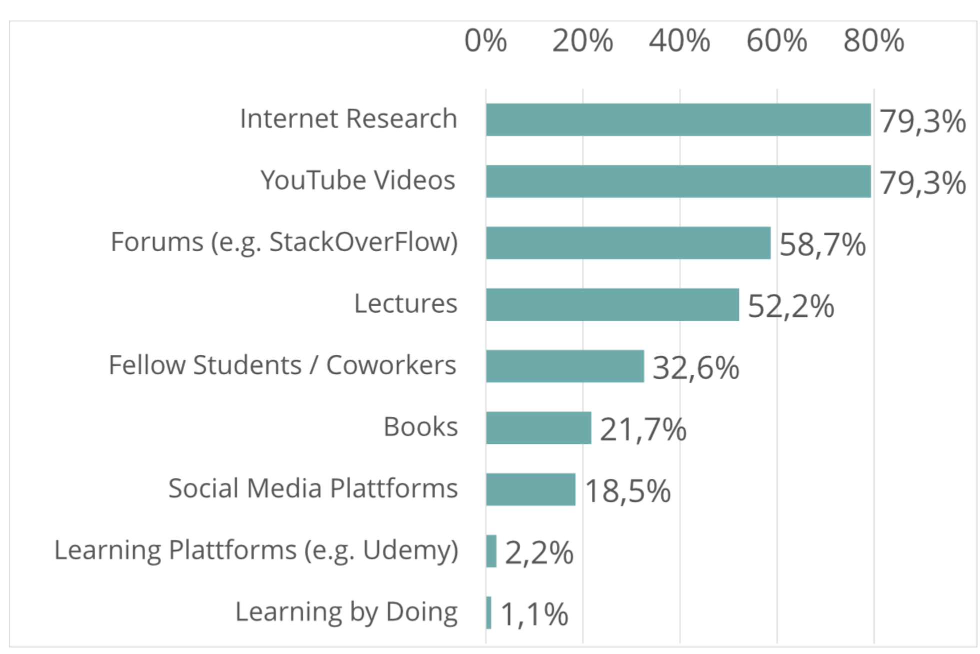


Figure A.23: Students way of getting to know new technologies in the field of Web Development

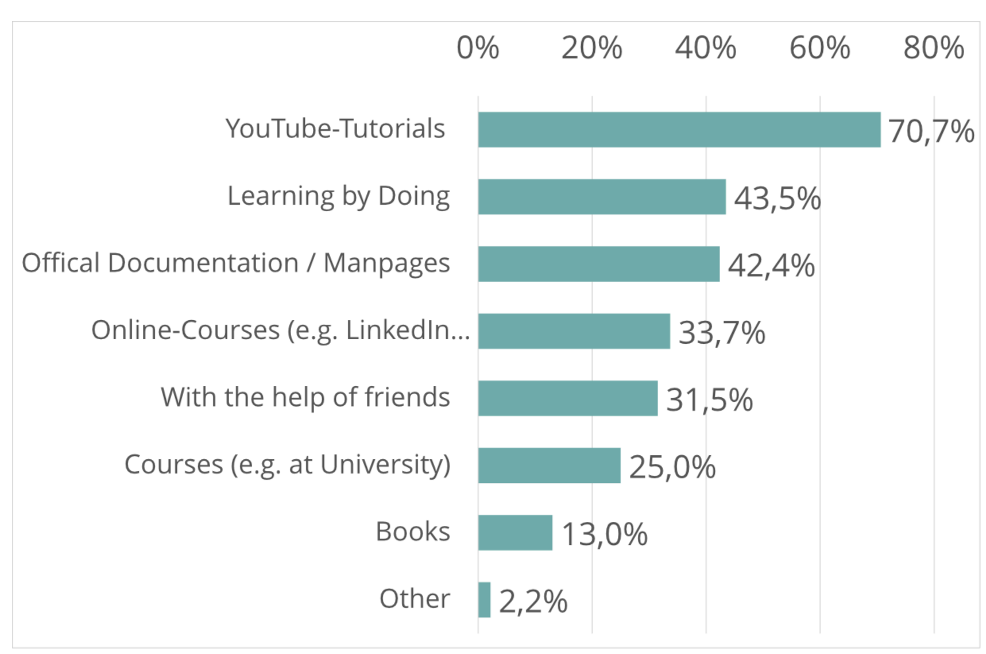


Figure A.24: Students way of learning new technologies

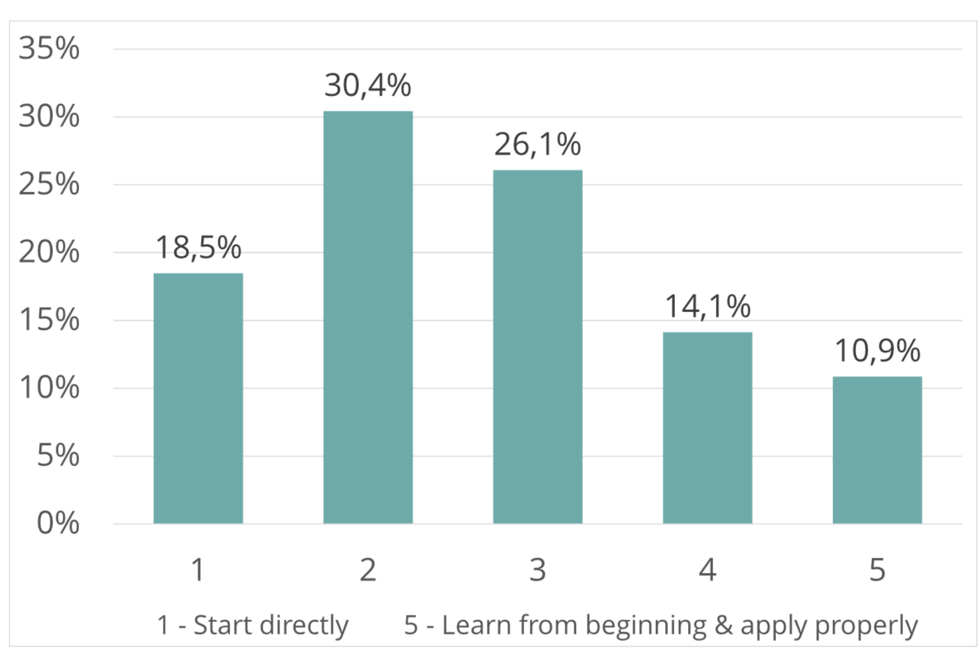


Figure A.25: Way of Learning starting to use a framework

(Question 27) *If you would like to describe the questions or answers in more detail, you have the chance to do so here:*

- Web development should be much more integrated into the study programme in general.
- In the IVS lecture, I would like to see the topicality given high priority, as development in the web area is extremely high.
- I learn best when I have a real, meaningful projects and specifically solve the problems that arise. Since I started studying in the first place because I want to start my career in the field of web dev professionally, anything that has to do with this subject is very important and highly interesting for me.
- The problem with Full-Stack Web Development is that you quickly create a dependency between the two sites. It is then simply no longer possible to support different frontends. One can therefore rather offer Full-Stack Development as a kind of possibility to focus fully on one frontend, e.g. build a whole stack only for a shop, game, app or similar and then consider it part of a microservice, which then accesses other services in the backend and then consider it as a single ”project” for one frontend.
- In the IVS lecture, one should therefore rather deal with basic concepts that are on the market, e.g.:
  - REST API (Lab would be great),
  - GraphQL,
  - and then what microservices and monoliths are.
- You could also show a widely used framework in the lab afterwards, so that people know how to get started with it.
- How often would you maintain a website: Depends on the update cycle of the technologies and cannot simply be generallised.

Figure A.26: Comments

Appendix B

Source Code Examples

As part of this thesis, code was written in various frameworks and forms. This is an overview of the produced source code for *RemArrow* which wasn't fully featured in the text.

Code Overview

a. Backend
   - B.1 server.js
   - B.2 scores.js
   - B.3 scoreModel.js
   - B.4 scoreController.js

b. Frontend
   - B.5 App.js
   - B.6 Frontend Styles - All, Scores (App.css)
   - B.7 Frontend Styles - Arrows, Buttons (App.css)
   - B.8 Frontend Styles - Media Queries (App.css)
   - B.9 Button Group Component (buttonGroup.js)
   - B.10 Titles Container (tiles.js)

```
// Packages for development
require('dotenv').config()

// Packages in the Node server
const express = require("express")    // Requiring the usage of Express
const mongoose = require('mongoose')
const cors = require("cors")

// Importing Score Routes
const scoresRoutes = require('./routes/scores')

// Express Application
const app = express()                  // Creating an instance of Express
app.use(cors());

// Middlware
app.use(express.json())

// Routes
app.use('/api/scores', scoresRoutes)  // Attaching Score Routes to the app

app.get('/', (req, res) => {    // GET-Request to the root (/)
  res.send({mssg: 'Hello World!'})
})

// Connection to MongoDB Atlas
mongoose.connect(process.env.MONGO_URI)
    .then(()=> {
        // Listen for Requests
        app.listen(process.env.PORT, () => {
            console.log('Connected to MongoDB & listening on Port',
                process.env.PORT)
        })
    })
    .catch((error) => {
        console.log(error)
    })
```

Figure B.1: Server - server.js

```
1  import React, {Component} from 'react'
2  import Score from '../components/score'
3
4  export default class Scores extends Component {
5      constructor(props){
6        super(props);
7        this.state = {
8            counter: 0,
9            highscore: 0,
10       };
11     }
12
13     componentDidMount(){
14       console.log("Mounted")
15       getHighscore = getHighscore.bind(this)
16       increaseScore = increaseScore.bind(this)
17       resetScore = resetScore.bind(this)
18       setScore = setScore.bind(this)
19       getHighscore()
20     }
21
22     render() {
23         return (
24             <div className="Scores">
25                 <Score scoreName="Highscore"
                       scoreNum={this.state.highscore}/>
26                 <Score scoreName="Score" scoreNum={this.state.counter} />
27             </div>
28         )
29     }
30 }
31
32 /* API Functions */
33 export function getHighscore () {
34   fetch("http://localhost:4000/api/scores/highscore")
35       .then((res) => res.json())
36       .then((data) => {
37         this.setState({
38           highscore: data.score,
39         })
40       } )
41 }
42
43 export function resetScore () {
44   this.setState({counter: 0})
45   getHighscore()
46 }
47
48 export async function setScore () {
49   await fetch('http://localhost:4000/api/scores', {
50     method: 'post',
51     headers: {'Content-Type':'application/json'},
52     body: JSON.stringify({
53      "score": this.state.counter
54     })
55    })
56    getHighscore()
57 }
58
59 export function increaseScore (value) {
60   this.setState({counter: this.state.counter + value})
61 }
```

Figure B.2: Server - scores.js

```
const mongoose = require('mongoose')

const Schema = mongoose.Schema

// Schema for the Score, so that no false value types can be send
const scoreSchema = new Schema ({
    score : {type: Number},
})

module.exports = mongoose.model('Score', scoreSchema)
```

Figure B.3: Server - scoreModel.js

```
// scoreController: contains the functions which control the models and
    requests
const Score = require("../models/scoreModel")   // Importing the scoreModel
const mongoose = require("mongoose")            // Requiring mongoose

// GET all scores
const getScores = async (req, res) => {
    const scores = await Score.find()
    res.status(200).json(scores)
}

// GET higscore
const getHighscore = async(req, res) => {
    const highscore = await Score.find().sort({score:"descending"})
    res.status(200).json(highscore[0])
}

// POST create new score
const createScore = async (req, res) => {
    const {score} = req.body
    const newScore = await Score.create({score})
    res.status(200).json(newScore)
}

// Exporting the functions
module.exports = {
    getScores,
    getHighscore,
    createScore,
    getScore,
    updateScore,
}
```

Figure B.4: Server - scoreController.js

```
import './App.css';
import { BrowserRouter, Routes, Route } from 'react-router-dom'

// Pages and Components
import Remarrow from './pages/remarrow';
import NotFound from './pages/notFound';

/* BrowserRouter to keep in sync with the URL */
export default function App() {
  return (
    <div className="App">
      { /* Routes are ordered from most specific to default */}
        <BrowserRouter>
          <div className="pages">
            <Routes>
                {/* Default View */}
                <Route path='/' element={<Remarrow />}/>
                {/* 404 page */}
                <Route path='/' element={<NotFound/>}/>
            </Routes>
          </div>
        </BrowserRouter>
    </div>
  );
}
```

Figure B.5: Frontend - App.js

```
/****************** Styling for all *****************/
.App {
  text-align: center;
  font-family: Raleway;
  user-select: none;      /* Text cannot be selected & copied */
  overflow: hidden;       /* Website non-scrollable */
}
/****************** Titles ******************/
.Titles {
  display: flex;
  flex-direction: column;
  margin: 10px 10vw;
  align-content: center;
}
.HeadingTitle, .HeadingSubtitle {
  margin: 0px 5px 5px auto;
  font-weight: bold;
}
.HeadingTitle {
  font-size: 3em;
}
.HeadingSubtitle {
  font-size: 1.5em;
}
/****************** Scores ******************/
.Scores {
  display: flex;
  flex-direction: column;
  margin: 5px 10vw;
}
#ScoreDiv {
  display: flex;
  flex-direction: row;
  justify-content: flex-start;
  align-items: center;
  margin: 2px;
  font-size: 1.5em;
  font-weight: bold;
}
.Score, .ScoreName, .ScoreNum {
  margin: 0px;
  padding: 5px;
}
#ScoreNameComponent {
  margin-right: 5px;
}
```

Figure B.6: Frontend Styles - All, Scores (App.css)

```
/****************** Arrows *****************/
.arrow {
  cursor: pointer;
  filter: drop-shadow(0px 6px 0px #444);
}
.arrow:hover {
  transition: 0.2s ease-out;
  filter: drop-shadow(3px 6px 3px #000);
}
.arrow:active {
  filter: drop-shadow(0px 3px 0px #222);
  transform: translateY(4px);
}
.highlight:hover {
  filter: drop-shadow(0px 5px 0px #fff);
}
.ArrowContainer {
  display: grid;
  margin-bottom: 20px;
  margin-top: -100px;
  justify-items: center;
  justify-content: center;
  grid-template-columns: 170px 170px 170px;
    grid-template-rows: 170px 170px 170px;
  grid-template-areas:
    ".          arrowTop          ."
    "arrowLeft    .       arrowRight"
    ".        arrowBottom       .";
}
#arrowYellow { grid-area: arrowTop;    }
#arrowRed    { grid-area: arrowRight; }
#arrowBlue   { grid-area: arrowBottom; }
#arrowGreen  { grid-area: arrowLeft;   }

/****************** Button *****************/
.buttonGroup {
  display: flex;
  justify-content: center;
}
.button{
  background-color: rgb(71, 71, 71);
  font-size: 24px;
  font-family: Raleway;
  color: white;
  padding: 4px 10px;
  border: none;
  border-radius: 4px;
  margin: 10px;
  box-shadow: 4px 4px 0px 0px rgba(175, 175, 175, 0.75);
}
.button:hover {
  transition: 0.2 ease-out;
  box-shadow: 4px 4px 4px 1px rgba(175, 175, 175, 0.75);
  transition-duration: 0.1s;
}
.button:active {
  box-shadow: 0px 0px 0px 0px rgba(0,0,0,0.75);
  transform: translateY(3px);
}
```

Figure B.7: Frontend Styles - Arrows, Buttons (App.css)

```
/* +++++++++++++ MEDIA QUERIES +++++++++++++ */
/* Small devices (phones and portrait tablets, up to 768px) */
@media screen and (max-width: 768px) {
  .ArrowContainer{
    grid-template-columns: 90px 90px 90px;
    grid-template-rows: 90px 90px 90px;
    margin-top: 10px;
  }
  .Titles{
    display: flex;
    flex-direction: column;
    margin: 5px 10vw;
    justify-content: center;
    align-items: center;
    margin: 15px 5px 5px 0px;
  }
  .HeadingTitle, .HeadingSubtitle{
    margin: 0px 5px 5px 0px;
  }
  .Scores{
    margin: 35px 20vw 0px 20vw;
  }
  #ScoreDiv {
    font-size: 1.2em;
  }
  .HeadingTitle {
    font-size: 2.5em;
  }
  .HeadingSubtitle, .startButton, .ScoreNameComponent, .ScoreNumComponent {
    font-size: 1.3em;
  }
  .button{
    font-size: 18px;
  }
}

/* Medium devices (tablets in landscape, 768px height and up) */
@media only screen and (min-height: 768px) and (min-width: 480px){
  .ArrowContainer{
    grid-template-columns: 200px 200px 200px;
    grid-template-rows: 200px 200px 200px;
    margin-top: -40px;
  }
}

/* Extra large devices (large laptops and desktops, 2000px width and up) */
@media only screen and (min-width: 2000px) {
  .ArrowContainer{
    grid-template-columns: 250px 250px 250px;
    grid-template-rows: 250px 250px 250px;
    /* margin-top: -45px; */
  }
  .Scores, .Titles {
    margin: 10px 25vw;
  }
}
```

Figure B.8: Frontend Styles - Media Queries (App.css)

```
1   import { resetScore, setScore } from './scores'
2
3   export default class ButtonGroup extends Component {
4       render() {
5           return (
6               <div id="buttonGroup">
7                   <Button id="startButton" text="RESTART" onClick={
                        resetScore} />
8                   <Button id="setScoreButton" text="SET SCORE"
                        onClick={setScore} />
9               </div>
10          )
11      }
12  }
```

Figure B.9: Button Group Component (ButtonGroup.js)

```
1   import Title from '../components/title.js';
2
3   export default class Titles extends Component {
4       render() {
5           return (
6               <div className="Titles">
7                   <Title className="HeadingTitle" text="RemArrow"/>
8                   <Title className="HeadingSubtitle" text="Can you
                        remember?"/>
9               </div>
10          )
11      }
12  }
```

Figure B.10: Titles Container (titles.js)

Appendix C

Screenshots of Prototypical Application

Design is a process. Here are some screenshots of the *RemArrow* prototype from the beginning to the final version.

Figure C.1: Graphic of a possible UI Design for Desktop

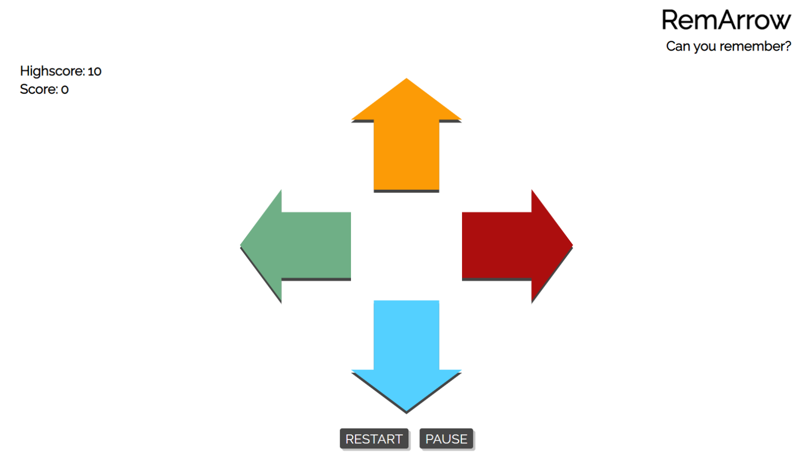


Figure C.2: RemArrow Desktop Size

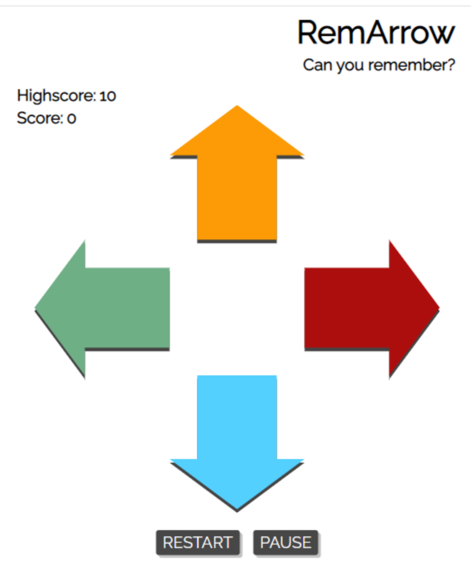


Figure C.3: RemArrow Tablet Size

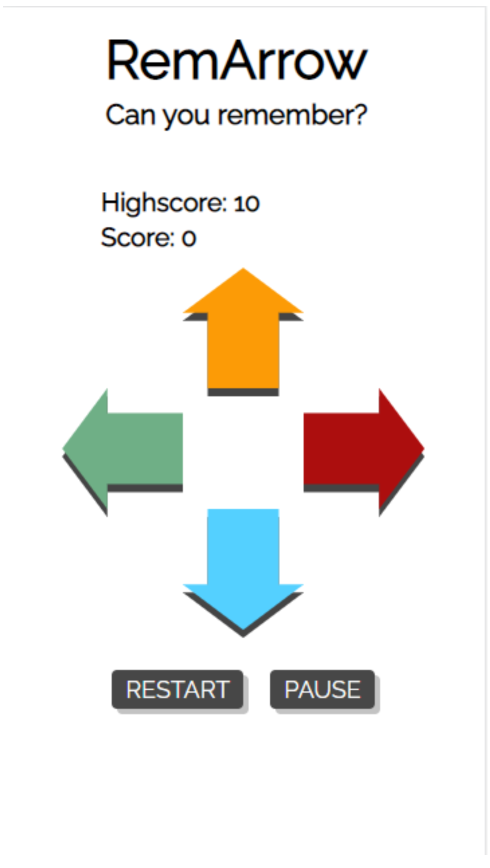


Figure C.4: RemArrow Smartphone Size

Appendix D

Poster

The following poster was created for the colloquium exam of this thesis at the University of Applied Science Offenburg at the *Werkschau 2022*.

Evaluation and Comparison of Full-Stack JavaScript Technologies

Bachelor Thesis by Nadine Weber

About JavaScript

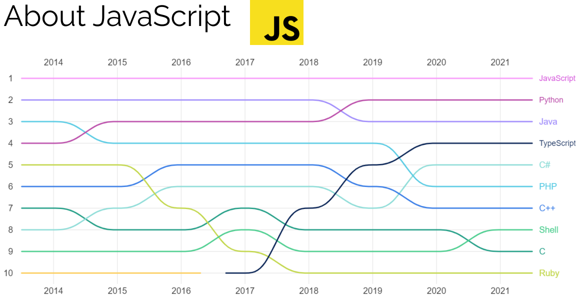
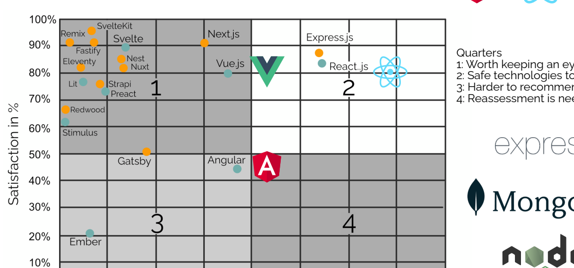


Fig. 1: Ranking of Programming Languages used in GitHub Projects [1]

Frameworks & Comparison

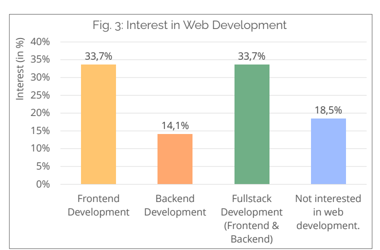


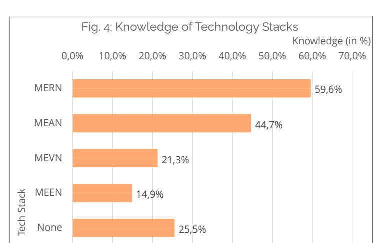
Quarters
1: Worth keeping an eye on
2: Safe technologies to adopt
3: Harder to recommend currently
4: Reassessment is needed

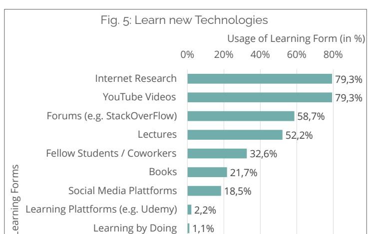
Fig. 2: Satisfaction to User Count Ratio by StateOfJS 2021, adapted from [2]

Questionnaire

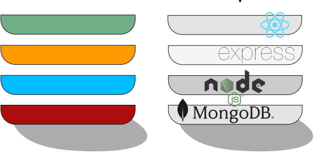
Target Group: Students from the University of Applied Sciences Offenburg
23 Questions about Development Experience, Full-Stack Development, the course „Interactive Distributed Systems" and Learning Behaviour.







Full-Stack Development



Working in both Frontend and Backend with JavaScript Technologies.

Fig. 6: Technology Stack MERN (MongoDB, Express.js, React.js and Node.js)

Comparison Criteria

- **Learning Curve:** Is it easy to learn the framework?
- **Maintainability:** How often do I need to update my application?
- **Modularity:** Can I reuse components?
- **Media Integration:** Does the framework support media assets?

	Developed by	Learning Curve	Maintainabiltiy	Modularity	Media Integration
Angular	Google	Steep curve	Easy managable	Yes, easily	Yes, good support
React.js	Facebook	Steep curve	Easy managable	Yes, very easy	Yes, many npm modules
Vue.js	Evan You	Flat curve	Easy managable	Not yet that easy	Yes, although still new

Tab. 1: Comparison of Frontend Frameworks used in popular technology stacks

RemArrow - Full-Stack Implementation

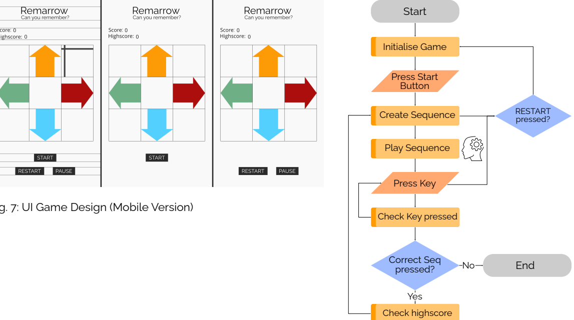


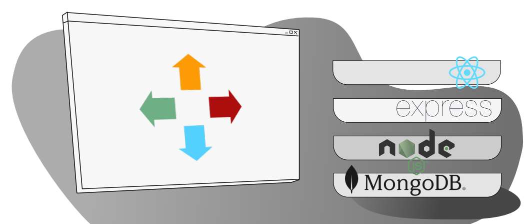
Fig. 7: UI Game Design (Mobile Version)

Fig. 8: Game Logic Flow Diagram

Results

- Large JavaScript community with a variety of technologies
- Dis-/Advantages between MEAN, MERN and MEVN depend upon the differences between frontend frameworks used in stacks
- Results of Comparison:
  - MERN - easy to learn and for fast implementations
  - MEAN - for bigger implementations which benefit from the correctness of data structures (e.g. Typescript usage)
  - MEVN - for developers interested in an easy frontend creation
- Python might be very interesting for all ML, KI developers

[1] GitHub Octoverse, The State of the Octoverse. [Online]. Available: https://octoverse.github.com/#top-languages-over-the-years (accessed: Jul. 7 2022).
[2] Stateofjs.com, The State of JS 2021. [Online]. Available: https://2021.stateofjs.com/en-US/ (accessed: Jul. 7 2022).
[3] AltexSoft, MEAN and MERN Stacks: Full Stack JavaScript Development Explained. [Online]. Available: https://www.altexsoft.com/blog/engineering/mean-mern-javascript-full-stack/ (accessed: Jul. 7 2022).



Media and Information Werkschau 21. Juli 2022
Prof. Dr. Tom Rüdebusch, Prof. Dr. Volker Sänger